Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Review article

Machine learning algorithms for FPGA Implementation in biomedical engineering applications: A review

Morteza Babaee Altman [a], Wenbin Wan [b], Amineh Sadat Hosseini [c], Saber Arabi Nowdeh [d,*], Masoumeh Alizadeh [e]

[a] *Department of Energy Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran 1591634311, Iran*
[b] *Department of Mechanical Engineering, University of New Mexico, MSC01 1150, Albuquerque, NM 87131, USA*
[c] *Department of Electrical and Biomedical Engineering, Islamic Azad University, Golestan, Iran*
[d] *Golestan Technical and Vocational Training Center, Gorgan, Iran*
[e] *Department of Electrical Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, 1591634311,Iran*

A R T I C L E I N F O

Keywords:
Biomedical applications
Embedded System
FPGA
Hardware acceleration
Machine learning algorithms

A B S T R A C T

Field Programmable Gate Arrays (FPGAs) are integrated circuits that can be configured by the user after manufacturing, making them suitable for customized hardware prototypes, a feature not available in general-purpose processors in Application Specific Integrated Circuits (ASIC). In this paper, we review the vast Machine Learning (ML) algorithms implemented on FPGAs to increase performance and capabilities in healthcare technology over 2001–2023. In particular, we focus on real-time ML algorithms targeted to FPGAs and hybrid System-on-a-chip (SoC) FPGA architectures for biomedical applications. We discuss how previous works have customized and optimized their ML algorithm and FPGA designs to address the putative embedded systems challenges of limited memory, hardware, and power resources while maintaining scalability to accommodate different network sizes and topologies. We provide a synthesis of articles implementing classifiers and regression algorithms, as they are significant algorithms that cover a wide range of ML algorithms used for biomedical applications. This article is written to inform the biomedical engineering and FPGA design communities to advance knowledge of FPGA-enabled ML accelerators for biomedical applications.

1. Introduction

The field of medicine has witnessed the application of Artificial Intelligence (AI) in two primary domains: physical and virtual [1]. The first aspect refers to physical objects, medical devices, care bots [2], assistant surgeon robots [3], and humanoid robots, which can help monitor the effectiveness of treatment in rehabilitative care [4]. On the other hand, the virtual component encompasses Machine Learning (ML) techniques, which utilize mathematical algorithms to develop learning through experience [1]. ML has been used in diverse biomedical areas such as genomics and proteomics [5], medical imaging [6], bio-imaging [7], Brain-Computer Interface (BCI) [8], and Public and medical health management (PmHM) (Healthcare big data analysis) [9]. The combination of large datasets, high-performance computational capacities, and improving algorithms has facilitated many successful biomedical applications that were previously challenging to implement [10]. However, implementing these ML algorithms on high-performance, real-time,

* Corresponding author.
E-mail address: saber.arabi17@gmail.com (S. Arabi Nowdeh).

https://doi.org/10.1016/j.heliyon.2024.e26652
Received 25 July 2023; Received in revised form 9 February 2024; Accepted 16 February 2024
Available online 18 February 2024
2405-8440/Â© 2024 Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

low-power mobile hardware, which suits biomedical applications, still needs to be completed.

Field Programmable Gate Arrays (FPGAs) emerge as a promising hardware candidate to meet the requirements of ML at the processing edge. For instance, FPGAs are highly suitable for latency-sensitive inference applications such as real-time image processing [11]. FPGAs are one of the most desirable architecture options for ML algorithms when any of the following are required [12].

- High performance and throughput
- Parallel processing
- Real-Time processing
- Reconfigurability

FPGAs have been shown to be a remarkable hardware tool for ML biomedical applications. Therefore, in this article, we review previous work on FPGA-based ML inference accelerators for various biomedical applications. In particular, we review and discuss nine different classifiers and regression algorithms implemented on FPGAs during 2001–2023. Additionally, we present an extensive analysis and comprehensive comparison of previous works while discussing their challenges and shortcomings. Moreover, we propose crucial future research trends. This paper's motivation is that ML is a prominent topic in biomedical engineering, and its efficient hardware implementation is paramount. To address this, we aim to answer the following research questions.

- How can the best hardware architecture be found for different classifiers and regression algorithms to embedded systems constraints, especially for biomedical applications?
- How do existing FPGA-based implementations of ML algorithms compare in terms of efficiency and accuracy?
- What trade-offs are desirable to achieve higher classification accuracy while meeting the design constraints of embedded systems?
- What are the advantages and limitations of FPGA-based implementations for each of the surveyed machine learning algorithms?
- What are the current trends and advancements in FPGA architectures, tools, and design methodologies that impact the implementation of machine learning algorithms?
- In terms of performance, energy consumption, and resource utilization, how do FPGA-based implementations compare with conventional software implementations?
- What are the future directions and potential research opportunities for FPGA-based implementation of machine learning algorithms in biomedical applications?

While several previous review articles [13,14] have discussed one or more specific ML algorithms, they have yet to cover and compile all ML algorithms used in biomedical applications. This article is the first to provide a comprehensive overview of nine distinct ML algorithms, excluding deep learning algorithms implemented on FPGAs for various biomedical uses. To indicate the contribution of this article in the area of FPGA-based hardware acceleration of ML algorithms, Table 1 compares the discussed ML algorithms in our work and previous survey papers. The table clearly shows that our work is the most comprehensive survey. The main contributions of the research are defined below.

2. Research methodology

This review started with a comprehensive search that yielded hundreds of publications within the initial scope of ML algorithm implementation on FPGA, concentrating on biomedical applications. During the search procedure, six distinct scientific databases were considered: *IEEE Xplore*, *Scopus*, *Google Scholar*, the *ACM Digital Library*, *PubMed*, and *ScienceDirect*. In addition, machine learning algorithms, FPGA implementation, FPGA hardware architecture, embedded systems, and biomedical applications were used as search terms. The resulting compilation of papers contains only conference and journal articles published after the year 2000. Given the significant advancements in FPGA technology to the present day, this period selection guarantees a comprehensive view of technology. This study's primary objective was to examine numerous facets of FPGA-based hardware systems.

This section provides a comprehensive overview of the research methodology employed in this review, including literature searches, data collection, validity threats, and data curation.

Table 1
Comparing the contribution of this article as a survey paper in the field of FPGA-based implementation of ML algorithms with other previously published literature.

Discussed Topics	Afifi et al. [13]	Skoda et al. [14]	This work
Regression Algorithms	×	×	✓
Decision Tree	×	✓	✓
K-Nearest Neighbor	×	×	✓
Support Vector Machine	✓	✓	✓
Clustering Algorithms	×	×	✓
K-Means Algorithms	×	✓	✓
Random Forest	×	×	✓
Mixture Models	×	×	✓
Genetic Algorithm	×	×	✓

2.1. Literature search

To ensure a thorough review, we conducted an extensive literature search. To ensure a comprehensive review, we conducted a comprehensive literature search. The process involved several key steps, as described as follows.

2.1.1. Selection of databases

This review started with a comprehensive search that yielded publications within the initial scope of ML algorithm implementation on FPGA, with a concentration on biomedical applications, by exploring various scientific databases renowned for their coverage of relevant academic publications. Specifically, we utilized six distinct databases: IEEE Xplore, Scopus, Google Scholar, the ACM Digital Library, PubMed, and ScienceDirect. These databases were selected due to their extensive coverage of research articles in machine learning, FPGA-based hardware acceleration, and biomedical applications. Using multiple databases can ensure consideration of all the relevant articles.

2.1.2. Search terms

We employed a carefully crafted set of keywords to retrieve articles of interest. These keywords encompassed relevant terms such as "machine learning algorithms," "FPGA implementation," "FPGA hardware architecture," "embedded systems," "FPGA Hardware Acceleration," and "biomedical applications." When searching for a specific algorithm, e.g., "Support Vector Machine," we used the particular name of the algorithm along with these keywords. Using these keywords allowed us to focus our search on articles directly related to our research objectives. We created comprehensive search criteria that include relevant keywords, synonyms, and Boolean operators (AND, OR) to ensure we capture a wide range of relevant articles.

2.1.3. Publication date filter

To ensure the inclusion of the most recent and comprehensive research, our search was limited to articles published after 2000. This restriction was prompted by the rapid evolution of FPGA technology over the past few years, guaranteeing that our review includes the most recent developments in the field.

2.1.4. Inclusion criteria

We evaluated each article's relevance to our review's scope during the initial search. We included conference and journal articles that addressed the implementation of machine learning algorithms on FPGA platforms, specifically within the context of biomedical engineering applications. Using the search terms explained in Subsection 2.1.2, we filtered the Selection of papers, discerning those that bore direct or indirect relevance to the subject matter. The word "direct" pertains to instances where the article explicitly references the application of the implemented algorithm on FPGA within its title or contextual content. On the other hand, the term "indirect" pertains to cases where the discussed architectures possess a broader scope but may also find applications within biomedicine.

2.1.5. Processing tools

To collect, organize, and cite research materials, including articles and web pages, we used Zotero reference management software and Microsoft Excel. For visualization and diagramming, we employed Inkscape. The project management tool that we used for organizing tasks and workflows was Trello.

2.1.6. Relevant articles

A systematic and organized approach is used to ensure we consider all the relevant articles. Clearly defined the scope and objectives of your review paper, including the specific research questions we wanted to address; comprehensive search criteria that include relevant keywords, synonyms, and Boolean operators (AND, OR) is created to ensure we capture a wide range of relevant articles, multiple databases, including IEEE Xplore, Scopus, Google Scholar, the ACM Digital Library, PubMed, and ScienceDirect are used, search filters provided by databases to narrow down results are applied based on publication date, article type, and relevant keywords, and the search queries, date of search, and the number of results retrieved is documented to conduct a systematic search. In the next step, the search results are reviewed, and the titles and abstracts of articles are checked to determine their relevance to your research questions. We created a systematic way to organize the articles we find using reference management software (Zotero) to keep track of articles, their titles, authors, publication dates, and relevance and avoid duplicated articles. We also examined the reference lists of the articles we found (especially in seminal papers) to identify additional relevant sources. Continuously, our search is updated as we progress with our review paper as new research might have emerged during our writing process, and before finalizing the review paper, seeking input from colleagues and experts in this field is considered to ensure that we have not missed any critical articles.

2.2. Data collection

To maintain transparency and facilitate reproducibility, we meticulously documented the details of our data collection process for each algorithm discussed in the paper. In the initial phase, we initiated our inquiry by employing the search queries mentioned in Section 2.1.2. These search terms were accompanied by the specific designation of the algorithm under investigation, such as KNN or SVM. In the subsequent step, we assessed the relevance of the identified sources to biomedical applications, considering both direct and indirect connections. We also checked specific biomedical applications as a keyword, e.g., "image processing," at the next step. This search yielded a total of 316 conference and journal articles. Subsequently, a meticulous examination of the abstracts and applying the criteria outlined in Table 2 reduced the number of papers to 143.

Furthermore, Table 2 presents a comprehensive breakdown of the individual counts of papers scrutinized for each algorithm. In Table 2, some algorithms were directly relevant to a biomedical application, mentioned explicitly in the article. This inclusion criterion is demonstrated by the biomedical relevance (BR) tag. However, in some cases, e.g., when investigating the Ridge Regression algorithm, the application is mentioned as "real-time optical flow," which is the basis for applications in biomedicine. In this case, we used the FPGA hardware architecture for medical applications (FHA) tag as the inclusion criterion.

2.3. *Validity threats and data curation*

To mitigate validity threats, we acknowledge potential sources of bias or limitations in our review process. These include language bias, as our search was conducted in English. The data Curation process involved careful screening of articles to ensure they met our inclusion criteria. We employed a systematic approach, with two independent reviewers assessing each article's relevance. Any discrepancies were resolved through discussion and consensus.

3. Groundwork

We introduce the fundamental concepts behind the hardware acceleration of ML algorithms by explaining the basics of FPGA technology and its advantages for implementing these algorithms. This section will also describe FPGA implementation techniques. We then introduce concepts surrounding performance and energy analyses, followed by a subsection on codesigning ML algorithms and architectures. These background concepts are provided to assist our discussions in the overview of ML algorithm implementations on FPGAs.

3.1. *Fundamentals of FPGA technology and its benefits for ML algorithm implementation*

Due to their distinctive characteristics and capabilities, FPGAs have become a robust hardware platform for implementing ML algorithms. FPGAs offer a hardware architecture that is highly flexible and configurable and is capable of being programmed and customized to perform particular tasks efficiently. FPGAs facilitate the creation of application-specific hardware accelerators, making them well-suited for accelerating ML algorithms in various applications, including biomedical domains. FPGAs are comprised of a set of programmable logic units linked by programmable routing channels [15]. These logic blocks, which are shown in Fig. 1, can be configured to perform specific functions, such as arithmetic operations, memory access, and control logic. In addition, FPGAs provide dedicated resources, such as Digital Signal Processing (DSP) blocks, on-chip memory, and high-speed interfaces [16], which can be utilized to optimize the performance and efficacy of ML algorithms.

One of the most significant benefits of FPGA-based designs is the ability to parallelize computations. FPGAs can use fine-grained parallelism by executing multiple tasks or data operations concurrently, resulting in significantly increased processing rates. This parallelism is especially advantageous for ML algorithms involving computationally intensive operations such as matrix multiplication, convolution, and traversal of decision trees. Using the parallel nature of FPGAs, ML algorithms can be executed in real-time, enabling applications with stringent latency requirements [17,18]. Moreover, FPGAs provide the versatility of reconfigurability. FPGAs can be reprogrammed and reconfigured to accommodate changing conditions or algorithm optimizations, unlike application-specific integrated circuits (ASICs), which are designed for specific tasks and are challenging to modify. This flexibility permits researchers and developers to iteratively refine and improve FPGA-based implementations of ML algorithms, thereby enhancing performance and resource utilization. For example, they can experiment with various architectural configurations, memory organizations, and algorithmic optimizations to achieve the desired balance between precision and efficiency. FPGAs have the potential for low-power implementations [19], an additional benefit. By meticulously designing the hardware architecture and utilizing

Table 2
The number of reviewed articles regarding the algorithms implemented on FPGA during 2000–2023 and the filter criteria of the articles by type of algorithm.

Algorithm		Number of reviewed papers	Number of papers included	Inclusion Criteria
Regression	Linear	18	9	FHA
	Ridge	11	5	FHA
	Logistic	13	6	FHA
Classifiers	KNN	35	11	FHA - BR
	DT/RF	32	16	BR
	SVM	86	40	FHA - BR
Clustering	Mean-Shift	20	9	FHA
	K-means	47	23	FHA- BR
mixture models	Gaussian	28	12	BR
Genetic		26	12	FHA
Total		316	143	–

BR: Biomedical relevance, FHA: FPGA hardware architecture for medical applications.

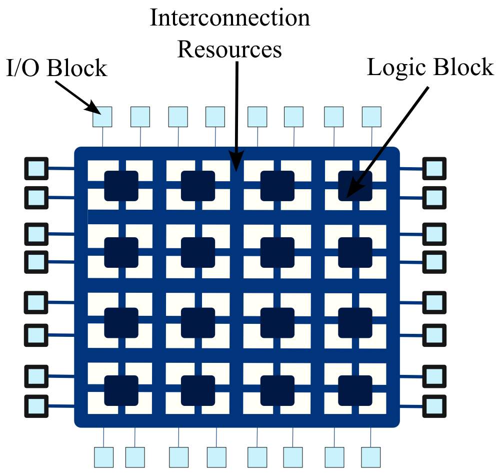


Fig. 1. A typical FPGA architecture, which includes logic blocks and I/O blocks (IOBs).

the parallelism inherent in ML algorithms, it is possible to decrease power consumption. FPGAs also provide the ability to activate hardware components only when necessary selectively, facilitating dynamic power management and reducing energy waste [20]. This is especially significant in energy-constrained applications, such as wearable devices, implantable medical devices, and remote sensing systems, where power efficiency is essential for extending battery life or reducing energy consumption.

Additionally, FPGAs offer a perfect platform for hardware/software co-design because they enable the implementation of ML algorithms' algorithmic components in hardware while leaving control and management tasks to embedded processors or co-processors. This combination of hardware acceleration and software control permits efficient resource utilization and the execution of ML algorithms [21]. This implementation technique will be discussed in the following section. The unique properties of FPGAs, such as their configurability, parallelism, reconfigurability, and potential for low-power implementations, make them a compelling choice for accelerating ML algorithms. By leveraging the capabilities of FPGAs, researchers and engineers can create high-performance, energy-efficient solutions for various biomedical applications, facilitating real-time data analysis, decision-making, and individualized healthcare interventions.

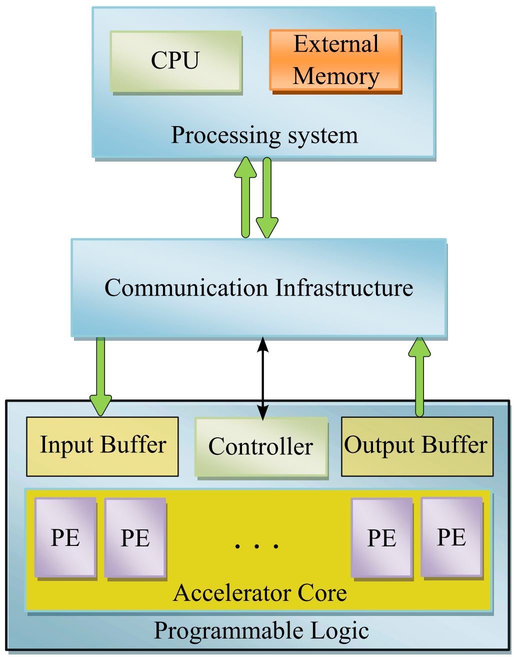


Fig. 2. The standard framework of FPGA-based hardware accelerator.

3.2. FPGA Implementation techniques

FPGAs provide a flexible substrate for implementing machine learning algorithms, and a number of techniques and methodologies can be employed to optimize their performance and resource utilization. This section discusses the most essential aspects of FPGA implementation techniques.

3.2.1. Hardware/software Co-design

Hardware/Software co-design (HW/SW Co-design) is an approach to implementing complicated systems, such as ML algorithms, on FPGA platforms that incorporate hardware and software components. By employing a hardware/software co-design strategy, FPGA-based ML implementations can benefit from the inherent parallelism and reconfigurability of FPGAs while preserving the flexibility and control of software components [22]. This method facilitates the efficient use of system resources, improved performance, and the ability to meet the specific design constraints of biomedical applications. Identifying the algorithmic components that can be effectively implemented in hardware and those that are better adapted for software execution is the first step in hardware/software co-design. In general, computationally intensive duties or data-parallel operations are excellent candidates for hardware acceleration, whereas software typically implements control functions and sequential procedures [23]. Fig. 2 depicts a generic FPGA-based HW accelerator architecture designed in Ref. [22]. This optimization method improves system performance by employing the loop unrolling or multiple processing elements (PEs) technique. FPGAs offer parallelism and high-performance capabilities that can substantially accelerate the execution of ML algorithms by implementing specific components in hardware. The algorithmic details implemented in hardware can exploit the parallelism inherent to the FPGA architecture, allowing for the simultaneous and efficient processing of multiple data samples or iterations. While hardware components handle computationally intensive duties, embedded processors, or coprocessors, typically execute control and management functions. These software components perform data preprocessing, algorithm configuration, decision-making, and device or network interfacing. To ensure efficient operation and data flow, hardware and software members communicate and synchronize with one another. This involves establishing interfaces and protocols for exchanging data between the software and hardware domains, managing synchronization points, and coordinating the system's overall behavior. One of the critical merits of hardware/software co-design is its inherent flexibility [24]. FPGAs can be reprogrammed to conform to changing algorithm requirements, allowing ML models to be refined and optimized iteratively. This flexibility permits the investigation of various hardware/software partitioning strategies and the ability to fine-tune the system based on performance, resource utilization, and energy efficiency objectives.

3.2.2. Algorithm mapping

The process of transforming an ML algorithm into a hardware design that can be effectively implemented on an FPGA is known as algorithm mapping. It involves decomposing algorithmic operations into computational assignments that can be executed efficiently on an FPGA architecture. This may entail identifying critical algorithmic functions, such as matrix multiplications, convolutions, or evaluations of decision trees, and determining how these operations can be distributed efficiently across the FPGA's configurable logic resources. FPGAs excel at parallel computation, and mapping algorithmic duties to similar hardware resources can substantially boost performance [25]. Loop unrolling [26], in which iterations of loops are processed in parallel, and pipeline parallelism [27], in which multiple computation phases are overlapped, are utilized to increase throughput.

The mapping procedure is affected by data dependencies within the algorithm. Techniques such as scheduling and data buffering [28] are used to manage dependencies and ensure efficient data flow between computational duties. Utilizing the FPGA's routing resources can facilitate data transfer and reduce delays caused by dependencies. In addition, FPGAs provide various memory options, such as on-chip block RAM and external memory interfaces, and selecting the appropriate memory type and access mechanism can substantially affect performance. Techniques such as data streaming [29], where data is transmitted directly from memory to computation units, and data partitioning [30] are utilized to optimize memory access patterns. Alternatively, task partitioning techniques enable the subdivision of the algorithm into smaller, more manageable tasks that can be efficiently assigned to FPGA resources. Methods such as spatial and temporal partitioning are applied in which duties are time-multiplexed on shared resources [31].

3.2.3. Trade-offs and design constraints

In implementing ML algorithms on FPGAs, tradeoffs and design constraints play a crucial role. It is essential to consider a range of factors and make well-informed decisions based on the specific needs of biomedical applications. Following is a discussion of the most crucial elements of design constraints.

3.2.4. Resource utilization vs. algorithm accuracy

Achieving the optimal balance between algorithm accuracy and resource utilization involves a trade-off. Each algorithm component implemented on an FPGA consumes a certain quantity of resources (logic elements, memory blocks, DSP blocks, and other specialized components). As the algorithm's complexity and accuracy requirements increase, resource consumption rises. ML algorithms, on the other hand, strive for high accuracy in their predictions or classifications. However, greater precision frequently necessitates additional computational resources. Precision requirements, such as those for fixed-point or floating-point arithmetic, impact the use of resources. Designers may investigate techniques such as dynamic precision scaling [32], in which precision can be dynamically adjusted based on the computational requirements of individual algorithm components. In addition, algorithmic optimizations such as quantization-aware training [33] and reduced precision quantization can mitigate the impact of lower precision on accuracy while minimizing resource consumption. Reducing the precision of calculations can substantially reduce resource requirements, albeit at the expense of some precision. It is possible to use approximate computing techniques to sacrifice accuracy for reduced resource consumption. Also, algorithmic optimizations, such as removing redundant connections or quantizing weights [34], can reduce computational complexity and, consequently, resource consumption.

3.2.5. Power management and energy efficiency

Power management and energy efficiency are crucial considerations in FPGA-based implementations, including biomedical applications. Efficient power management strategies can aid in reducing power consumption while preserving performance levels. Dynamic power gating is a technique that entails selectively shutting off power to inactive or idle FPGA design components. Dynamic power gating reduces energy consumption by shutting off power to unused modules and features [35]. This method can be beneficial when dealing with FPGA designs with dynamic duties or variable computational demands. It permits power allocation on demand, reducing power waste and enhancing energy efficiency. Clock gating is an additional power management method. Clock gating selectively turns clock signals on or off for particular FPGA circuit elements. Hence, it reduces unnecessary clock switching and associated power consumption by turning off the clock on inactive or dormant components [36]. This technique helps conserve energy by lowering switching activity and can be advantageous in designs with lengthy periods of inactivity or minimal training.

Voltage scaling is an additional power management technique utilized in FPGA-based implementations. It involves dynamically adjusting the operational voltage of FPGA components [37]. By decreasing the voltage supplied to particular regions or modules within the FPGA, power consumption can be drastically reduced. Utilizing the fact that power consumption is proportional to the square of the voltage, this method enables significant power savings without sacrificing performance. However, voltage scaling must be carefully managed to avoid degrading performance or introducing functional errors. Suitable control mechanisms must accompany power management techniques to ensure a seamless transition between power-saving states and active operation, thereby preserving system dependability and responsiveness. Reconfigurable architectures, such as FPGAs and Coarse-grained Reconfigurable Arrays (CGRAs), are gaining popularity in biomedical applications [38–41]. They have the potential to improve efficiency and performance. Nevertheless, programming models for such architectures frequently have limitations. Traditional Hardware Description Languages (HDLs) lack the abstractions necessary for productivity, whereas using higher-level languages can add complexity. To overcome these limitations, Koeplinger et al. [42] developed Spatial, a novel domain-specific language and compiler for describing application accelerators at a high level. They outlined the compiler stages required to support hardware-centric abstractions such as scheduling the pipeline, automatic memory banking, and active ML-driven automated design tuning. In addition, applications written in Spatial were 42% shorter on average. They achieved a mean speedup of 2.9 × over Xilinx SDAccel HLS when targeting a Xilinx UltraScale with VU9PFPGA on an Amazon EC F1 case.

4. An overview of the implementation of ML algorithms on FPGA

This section will review some of the recent studies implementing the ML algorithms presented in Table 1.

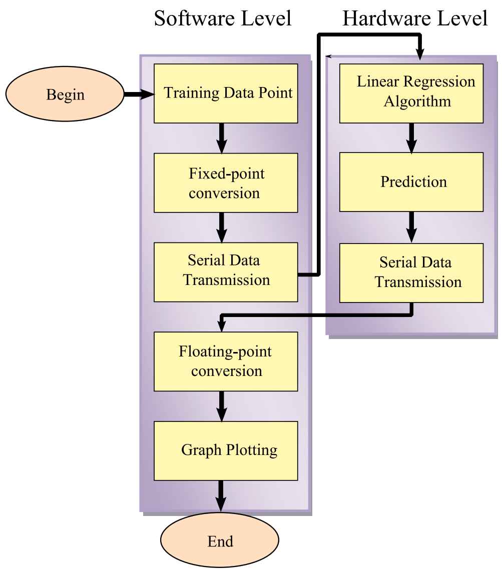


Fig. 3. Flowchart of simple linear regression. Adapted from Ref. [44].

4.1. Linear regression

Regression algorithms are essential in biomedical applications because they facilitate the analysis and prediction of continuous variables based on input features. There are numerous advantages to implementing regression algorithms on FPGAs, but unique challenges must be addressed. This section will examine implementing regression algorithms on FPGAs, highlighting their benefits, difficulties, and performance evaluations in biomedical applications.

4.1.1. Hardware design of linear regression algorithms

Based on continuous variables, linear regression is utilized to approximate actual values. The relationship between independent and dependent variables is determined by fitting the finest line. This line, known as the regression line, is characterized by the linear equation $y=\alpha+\beta x$ where y is the Dependent Variable, β represents the Slope, x is the Independent Variable, and α represents the Intercept. The α and β coefficients are identified by minimizing the distance between each data point and the regression line. To fit this model, the optimal parameters and the ordinary least squares approach are required [43]. In Ref. [44], a basic linear regression model was developed, whereby the function predicting y takes a single x variable as its input, as illustrated by equation (1):

$$y = \varphi_1 x + \varphi_0 \tag{1}$$

Using matrix algebra operations and the least-squares estimation method, it is possible to modify φ values using (2) [44]:

$$\varphi = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} = \left(X^T X\right)^{-1} X^T y \tag{2}$$

Where, X and y are the training data points. Fig. 3 shows the flowchart of simple linear regression and its relevant algorithm via matrix algebra is represented in algorithm 1.

Algorithm 1. Matrix algebra as a linear regression algorithm

```
Input: Training data point ($X = [x_0 \ldots x_{n-1}]$ and $y = [z_0 \ldots z_{n-1}]$) and start signal
Output: $\varphi = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}$
1: Insert n (the amount of training examples)
2: if rising edge (start) then
3:   Compute $(X^T X) = \begin{bmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{bmatrix}$
4:   Compute $(X^T X)^{-1}$
5:   Compute $(X^T X)^{-1} X^T$
6:   Compute $(X^T X)^{-1} X^T y = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix}$
7: end if
8: return φ
```

The linear regression algorithm was implemented using the Very High-Speed Integrated Circuit Hardware Description Language (VHDL) on the Altera DE2-115 development board [44]. Fixed-point arithmetic was utilized by Ferreira et al. to optimize the execution time. Signed integer numbers were modeled as decimal values using 32-bit arrays. To address (2), the linear algebra operations were divided into sequential operations and encoded within a *PROCESS VHDL* structure. The sensitivity list included the start signal. The obtained modeling generated a unique intellectual property (IP) entity. With eight training data points, the embedded FPGA system consumed approximately 136.82 mW of energy. In Ref. [45], the authors discuss the computational challenges associated with microarray analysis, specifically the computation of linear regressions in a high-dimensional parameter space. They effectively demonstrate the suitability of FPGAs by achieving a remarkable 1600-fold increase in speed over serial implementations. This study highlights the efficacy of FPGA coprocessors in resolving comparable problems in functional genomics.

Different hardware architectures were proposed to implement the linear regression method on FPGAs, focusing on restrictive area systems [46–48] because linear regression finds wide application in digital signal processing. These architectures conserved space by restricting input signal lengths to predetermined values. An Automatic Modulation Classifier (AMC) scheme employing statistical characteristics of phase and instantaneous frequency for real-time signal processing was proposed in Ref. [44]. Fixed-point arithmetic and the Altera Quartus II 12.0 Web Edition were utilized to implement the system. Executing in a single clock cycle, the FPGA design supported a custom linear regression and prediction entity with eight training data points. The architecture could be expanded to accommodate larger training data sets and more complicated ML hardware implementations, such as Deep Learning (DL) algorithms. To implement the linear regression algorithm on FPGAs, Royer et al. [46] offered a number of hardware topologies, focusing on space-constrained devices. To achieve its space-saving goals, the suggested system would limit the duration of the input signals to predetermined values. Similar to Ref. [44], they implemented their proposed scheme using AMC, meeting the problematic real-time limitations of this system. To find the linear function that best matches a particular set of samples, the coefficients α and β should minimize the Mean Squared Error (MSE) [46,49].

Fig. 4 depicts the proposed implementation plan for minimizing the MSE. Constants can be stored in read-only memory (ROM) using this scheme. This linear regression model can be utilized for an AMC, specifically for the computation of the critical statistics for detecting phase modulations applicable to bio-signal processing. Unknown in length, the input data is a two-sample per clock cycle signal. The signal is divided into 256 sample segments that are fed to the linear regression module for processing. The scheme proposed in Fig. 4a primarily employs accumulators (ACCs) and multiply accumulators (MACCs), making the Xilinx-embedded DSP48E a suitable choice for its implementation. According to the Xilinx DSP48E architecture, a 25 × 18 signed multiplier with ports A and B is followed by a 3-input 48-bit arithmetic logic unit (ALU), which can also operate as an accumulator [50]. The DSP48E can be configured as a 2-input accumulator (Fig. 4b).

Similarly, in implementing a real-time AMC on a Xilinx Virtex 4 LX100 FPGA, built-in multipliers and the CORDIC algorithm led to efficient resource utilization [47]. With sampling rates exceeding 1 Gsample/s, the classifier operated in real-time. The linear regression blocks occupied most of the area (71%), whereas the CORDIC submodule occupied a minor portion (6.5%) but provided the optimal area and performance. The FPGA utilization attained 16%, which was lower than the target of 25%, and the system achieved a fivefold increase in speed relative to the required input rate of 20 Msamples/s. The implementation operated at 100 MHz and satisfied real-time requirements. Recent literature [51] presents an FPGA-based system that uses a visible light optical spectrometer and a regression predictor to measure neurotransmitter concentration. The design makes efficient use of hardware resources and predicts molecule concentrations from measured data with exceptional accuracy. This study emphasizes the potential of FPGAs to enable accurate and real-time measurements of neurotransmitter concentration, thereby enhancing biomedical researchers' ability to comprehend brain behavior.

4.2. *Ridge regression*

Tikhonov Regularization, usually known as ridge regression, is the most commonly used regression algorithm to approximate an equation with no unique solution. The ridge regression model may be used imaginatively to analyze medical data rather than abandon the usual regression approach [52]. For instance, ridge regression can be applied as a classifier in biomedical image processing [53]. While optical flow algorithms are primarily used in image processing applications, they can also have biomedical applications. It has been shown to be efficient in real-time optical flow sensors [54]. Motion estimation and video compression have developed as significant aspects of optical flow research [55]. A ridge regression-based optical flow algorithm was developed to overcome the collinear problem in conventional least-squares approaches. Implemented on a BYU Helios FPGA board [56], the algorithm incorporated spatial and temporal smoothing operations to enhance accuracy. Traditional computationally expensive algorithms are not suitable for many real-time applications because of the computation power constraint. Hence, Wei et al. [56] devised and implemented a real-time optical flow algorithm on a low-power, compact FPGA device. As advancement, a new hardware architecture for calculating the optical flux of real-time video streams is introduced in Ref. [57]. Similarly, in Ref. [58], a pipelined hardware architecture using a full-search block matching (FSBM) algorithm is proposed for real-time optical flow estimation. The system was implemented on a Xilinx Virtex-5 FPGA XC5VLX330-10 device, achieving a processing speed of 400 frames per second (fps) for standard VGA images at the maximum frequency of 160.7 MHz.

4.3. *Logistic regression*

Advances in biosensors and wearable technology enable the collection of routine health data. Real-time detection of health events like epileptic convulsions using electroencephalography (EEG) is possible with suitable models [59]. Page et al. [60] discussed using logistic regression (LR) for a low-power, multi-channel EEG feature extractor and classifier to predict seizures from scalp EEG data. The proposed scheme allows sequential processing of each channel's extraction and classification within a half-second computation time [61]. A sequential implementation of classifier algorithms was achieved on the Virtex-5 FPGA platform. A 22-channel seizure detection hardware system was developed, with the LR classifier performing best regarding FPGA resource utilization, power consumption, and computation latency for 256 samples per window. Fig. 5 illustrates the Block diagrams of Logistic Regression's Feature Extraction

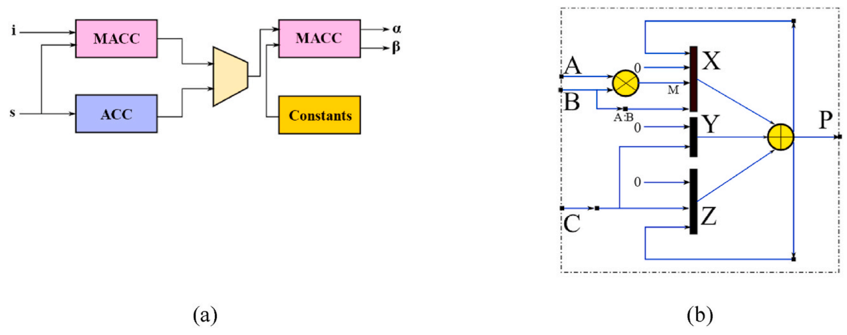
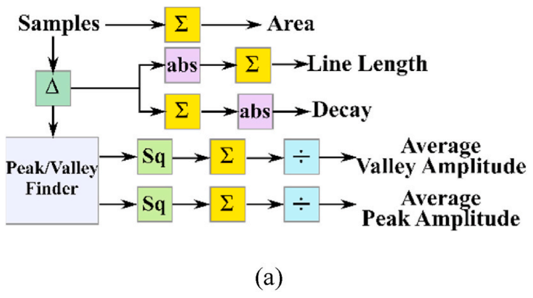


Fig. 4. (a) The scheme proposed in Ref. [46] for computing α and β coefficients, and (b) DSP48E implementation of a 2-input accumulator, connected to terminals A, B, and C, the two inputs load the X and Y multiplexers. P is connected to Z, which is a multiplexer [46].



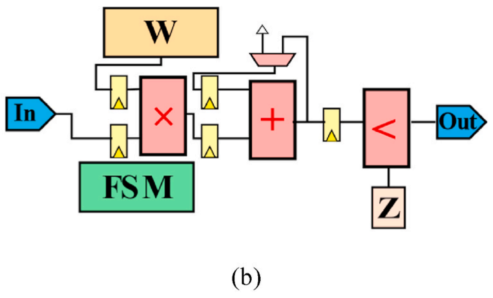


Fig. 5. The figure illustrates (a) the block diagrams of the feature extraction process and (b) reconfigurable classifier employing Logistic Regression. The diagrams are adapted from Ref. [61].

(Fig. 5a) and reconfigurable classifier (Fig. 5b) [61].

Logistic regression implemented in PLINK has proven to be a robust framework for evaluating gene-gene interactions, particularly for complex chronic diseases like ankylosing spondylitis [62]. However, computing regression models for each pair of markers in a genome-wide dataset is computationally demanding. To address this [63], introduced a combination of a Xilinx UltraScale FPGA and an Nvidia Tesla GPU, achieving significantly faster runtimes (minutes) for genome-wide logistic regression tests. This approach resulted in potential speedups of 1000 to 1600 compared to multi-threaded PLINK on a server-grade computing platform. The heterogeneous FPGA-GPU computing architecture, initially proposed in Ref. [64] and further improved in Ref. [63], demonstrates significant performance gains by utilizing high-end off-the-shelf components. The system consists of a powerful mainboard with dual Intel XeonE5-2667v4 8-core CPUs, 256 GB of RAM, an NVIDIA Tesla P100 GPU with 16 GB VRAM, and an Alpha Data ADM-PCIE-8K5 FPGA accelerator card. The FPGA accelerator features a modern Xilinx Kintex UltraScale KU115 FPGA with 8 GB SODIMM memory modules. As depicted in Fig. 6 of [63], the architecture exhibits a notable acceleration, exceeding 10 to 15 times for the given example datasets, even when conducting double-precision calculations. This performance improvement is primarily attributable to the synergistic effect of the FPGA and GPU accelerators, whose combined processing capacity enhances the overall processing capabilities.

Reference [65] presents a novel method for producing real-time vibrotactile feedback signals in a robotic hand. By combining FPGA technology with Multinomial Logistic Regression, this approach shows promise for classifying vibrotactile feedback in real time. The experiment involved a prosthetic hand equipped with sensors, with the sensor data transmitted to the FPGA card. While Multinomial Logistic Regression performed well in identifying object types, improvements are needed for movement and combined types. Exploring alternative machine learning algorithms is recommended to enhance classification accuracy. Overall, this approach offers the potential for real-time vibrotactile feedback in robotic hands.

4.4. *Classifiers*

For classification purposes, numerous ML techniques are applicable. Usually, the backpropagation (BP) and iterative characteristics of these methods are not appropriate for high-performance applications with large-scale data since the ML algorithms' parameters require to be gradually tuned in a learning process. In the upcoming sections, we will explore the utilization of various classification algorithms on FPGAs specifically for biomedical applications.

4.4.1. *K-nearest neighbors algorithm*

The K-nearest neighbor (KNN) classifier is one of the most prevalent ML classification algorithms. Typically, this algorithm is utilized for pattern recognition [66], text categorization [67], data mining [68], image classification [69], and predictive analysis [70]. In biomedicine, KNN finds extensive application in the pharmaceutical industry, particularly in identifying the proliferation of oncogenic (cancerous cells) [71]. Other applications consist of predicting solvent accessibility in protein molecules [72], data analysis

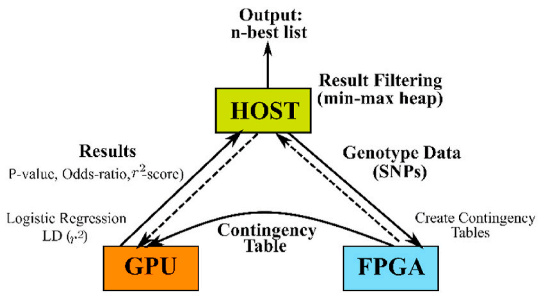


Fig. 6. Workflow of a heterogeneous FPGA-GPU system for the compute-intensive biomedical task of assessing gene-gene interaction for epistasis detection [63].

of microarray gene expression [73], Studying Heart Rate Signals [74], and medical image segmentation for brain Magnetic Resonance Imaging (MRI) region classification [75]. KNN, as a nonparametric classification method, is widely used in three critical applications, including (i) entropy estimation (tomography, motion estimation, and object tracking), (ii) classification and clustering (pattern recognition systems), and (iii) content-based image retrieval. Here, we focus only on the second application, as it has been implemented on FPGAs in recent literature and used in medical engineering.

Several research works have been achieved in recent years, primarily focused on enhancing the performance and efficiency of the KNN algorithm through hardware implementations, specifically using FPGAs. Table 3 presents a summary of these studies. These works explore various techniques, platforms, and architectures to accelerate the KNN algorithm and improve its execution time compared to general-purpose processors (GPPs) (such as works [76,77]) or other hardware devices. The authors employ high-level synthesis (HLS), parallelization, pipelining, and heterogeneous computing systems to optimize the KNN algorithm's execution on FPGAs. Some related efforts leverage OpenCL and Dynamic Partial Reconfiguration (DPR) to improve FPGA-based implementation performance and flexibility. Speedup ratios, energy efficiency, classification accuracy, and dataset dimensions are analyzed in most related papers.

Reference [78] presents an early study on FPGA-based implementation of KNN algorithms. The proposed approach utilizes Partial Distance Search (PDS) in the wavelet domain to determine the k nearest vectors in the design set for each input vector. By employing techniques such as subspace search, bit-plane reduction, and multiple-coefficient accumulation, the design achieves a cost-effective FPGA implementation with high throughput and low area cost, suitable for image processing and pattern recognition applications. Reference [79] offers a high-performance, HLS-based KNN method. Thus, operating at higher levels of abstraction and verifying scheme functioning at the C-Level made HLS more efficient than traditional HDLs and increased hardware designer productivity. They optimized their plan and pipelined the design using C code and Vivado HLS (Fig. 7a). In this figure, the DC block is made up of several calculation units to compute the distances between the query vector and all the training vectors, and the KF block is comprised of K comparators, used to determine the K nearest neighbors and store their correspondent distances and class labels. The CLV block is a majority voter that involves C counters and a comparator. The authors evaluated their IP core on the Xilinx ZC706 FPGA board with Vivado 2015.2 at 100 MHz. They tested their design with 1024 training vectors, 2D, 2 classes, and K = 7. The hardware architecture they designed exhibits a speed improvement of 35.1 times compared to a GPP-based implementation. However, two-dimensional testing does not demonstrate the design's capabilities. Pu et al. introduce in Ref. [80] a highly efficient parallel implementation of the KNN algorithm. This implementation leverages the architecture of FPGA-based heterogeneous computing systems and utilizes the Open Computing Language (OpenCL) framework (Fig. 7b). The GPU implementation is 410 times faster than the CPU.

In contrast, the FPGA implementation is 148 times faster. The FPGA also demonstrates higher energy efficiency compared to the GPU. However, the classification accuracy is not mentioned in the results.

In [81,82], the designers introduced a systematic design for two linear array IP cores for a KNN classifier (Fig. 8). The architecture proposed in Ref. [81] incorporated M + k+1 PEs, while [82] proposed an architecture with 2 N + 1 processors, where N and M represent the number of vectors and features, respectively. To reduce classification time, the data paths of the two designs have been pipelined. The design has been parameterized so it can be utilized for different M, N, and K values. However, the number of processors varied depending on M and K in Ref. [81] and N in Ref. [82]. The design has been parameterized so it can be utilized for different M, N, and K values. However, the number of processors varied depending on M and K in Ref. [81] and N in Ref. [82]. Therefore, the two distinct architectures in Refs. [81,82] result from the fact that the first architecture does not fit in the FPGA device when N is

Table 3
The key features of the literature discussed in Section 4.4.1.

Reference	Hardware Platform	Speedup	Features
[76]	Xilinx ML 403 platform board with XilinxXC4VFX12FF668-10 FPGA	FPGA: 92 × faster than GPP	Implementation of a multi-core ensemble K-NN classifier/About 5 × faster reconfiguration time
[77]	Xilinx ML 403 platform board with XilinxXC4VFX12FF668-10 FPGA	FPGA: 76 × faster than GPP	FPGA implementation of KNN classifier with fixed K value
[78]	Altera Stratix EP1S40	Not mentioned	a cost-effective FPGA-based implementation of KNN classification embedded in a softcore CPU
[79]	Xilinx ZC706 FPGA board	FPGA: 35.1 × faster than GPP	HLS-based KNN algorithm solution, C-Level verification/ parallelism, and design pipelining
[80]	Stratix IV 4SGX530	GPU: 410 × faster than CPU, FPGA: 148 × faster than CPU	Parallel implementation using FPGA-based heterogeneous computing systems architecture with OpenCL/Comparison with CPU and GPU implementations.
[81]	Virtex 2 Pro XC2VP30-6 FPGA board	FPGA:1.5 × to 2.96 × faster than GPU	Systematic design of IP cores with linear arrays for KNN classifier. Comparison of performance with CUDA-based GPU implementation.
[82]	Virtex 5 XC5VLX110T-3 FPGA	FPGA: Slower than GPU in most cases	Similar to [81]
[83]	Virtex-6 FPGA	FPGA: 127 × speedup compared to software	Novel hardware architecture for KNN classifier on mobile devices. Mixed VHDL and Verilog design.
[88]	Xilinx XC4VSX35 FPGA	about 8 × speed up in reconfiguration time	Application of FPGA-KNN to the analysis of biomedical data, such as Microarray data, with decreased power consumption.
[89]	Not mentioned	Not mentioned	a breathing disorder recognition system with a high accuracy
[90]	ARM processor FPGA	Not mentioned	Remote cardiac monitoring: ECG signal processing and KNN-based abnormality detection

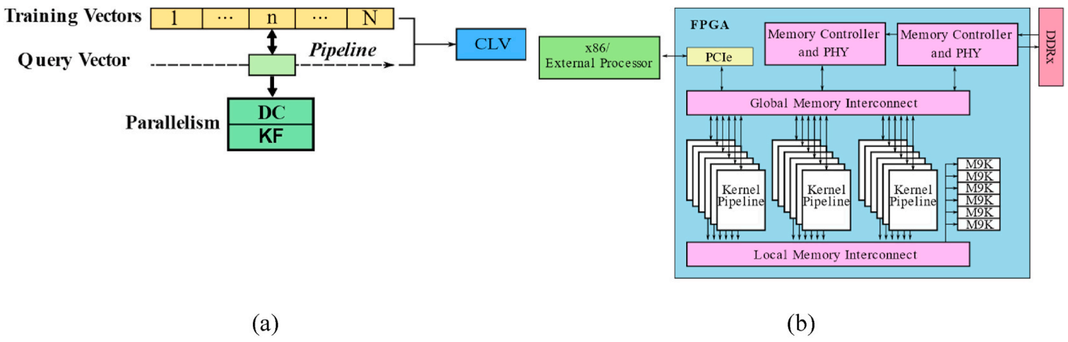


Fig. 7. (a) The K-NN hardware architecture proposed in Ref. [79], and (b) The OpenCL platform in Ref. [80].

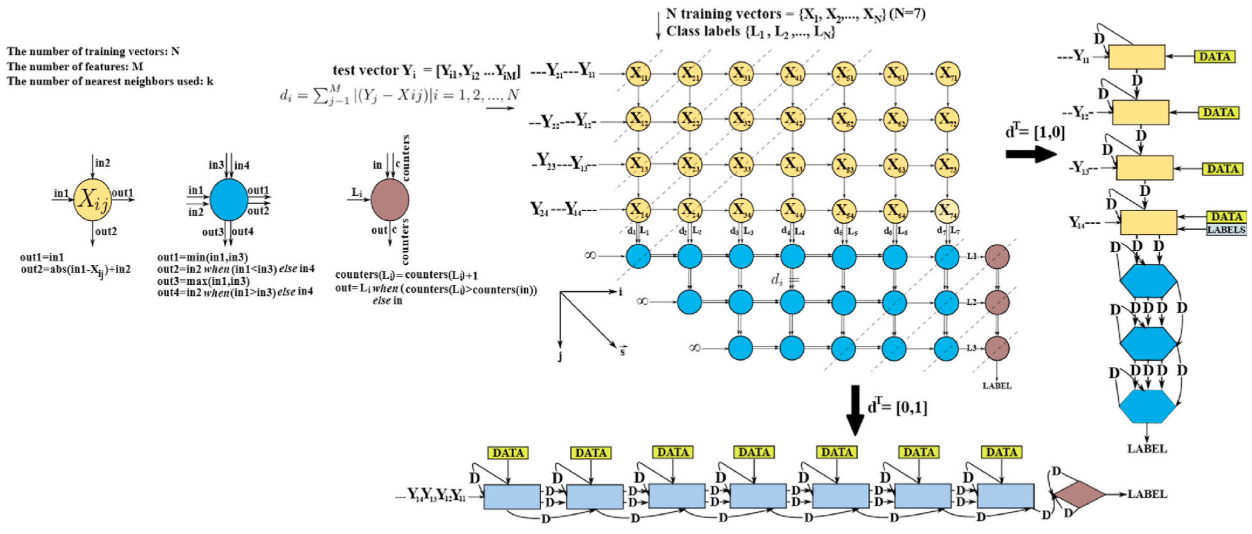

The number of training vectors: N
The number of features: M
The number of nearest neighbors used: k
N training vectors = {X1, X2,..., XN} (N=7)
Class labels {L1, L2,..., LN}
test vector Yi = [Yi1,Yi2 ...YiM]
d^T= [1,0]
d^T= [0,1]
DATA
LABELS
LABEL

Fig. 8. The Dependences Graph (DG) of the KNN algorithm (N = 7, M = 4 and k = 3) in Refs. [81,82], and Signal Flow Graph (SFG) array architectures of a horizontal and a vertical DG projection [82].

significantly smaller than M, and the second architecture does not fit in the FPGA device when N is substantially larger than M. In Fig. 8, the KNN algorithm is divided into two phases. Several exemplar vectors (N) with M dimensions (features) are kept in memory along with their class labels during the first or training phase. A test vector of an unknown class is provided in the subsequent, or classification, phase, and its distances from each training vector are computed. The k-lowest distances between the training vector and

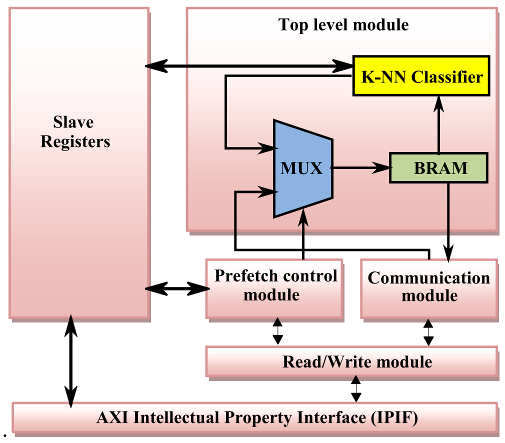


Fig. 9. High-Level Architecture of User-Designed Hardware Module proposed in [83].

the test vector are determined. In Ref. [81], the FPGA architecture achieved a speed of 100 MHz on a Virtex 2 Pro XC2VP30-6 FPGA board, while in Ref. [82], the same architecture ran at 100 MHz on a Virtex 5 XC5VLX110T-3 FPGA. A comparison was made with the results from Ref. [82], where a CUDA API-based implementation on a Pentium 4 PC with an NVIDIA GeForce 8800GTX GPU was used. The results showed that in Ref. [81], the FPGA was generally 1.5 to 2.96 times faster than the GPU for various K and N values. However, in Ref. [82], the FPGA was slower than the GPU in most cases, with the best case showing equivalent speeds between the FPGA and GPU.

In [83], a novel hardware design for accelerating the KNN classifier on mobile devices was presented, addressing complexity and processing capacity limitations. The hardware architecture was 127 times faster than its software counterpart while maintaining 100 percent classification accuracy. The design was implemented and verified on a 100 MHz Virtex-6 FPGA using VHDL/Verilog hardware modules, as shown in Fig. 9 [83]. Verification was performed using Xilinx ISE, XPS, and iSim, while C-written software modules were optimized and tested on a Micro Blaze FPGA. The architecture consisted of four stages: min-max normalization, Euclidean distance measurement, distance value sorting, and data classification.

In [84], the authors introduced a perfecting technique to mitigate high memory access latency. The classification accuracy depended on different datasets and test set percentages. For the Iris dataset, a test set percentage of 10% resulted in 100% classification accuracy across different K values. The heart dataset achieved the % classification accuracy of 83.6% when the test set percentage was 70% and K was 9. It’s worth noting that this scheme did not employ parallel processing techniques. Supervised learning methods, including Support Vector Machines (SVM), KNNs, and neural networks, are commonly used to analyze biomedical data, such as Microarray data. Data mining techniques applied to these datasets have led to the discovery of disease-associated genes, particularly in cancer research [85–87]. These findings have been utilized to improve predictive models, leading to the development of commercial diagnostic kits like Mamma Print and Symphony by Agendia. The hardware architecture described in Ref. [88] offers the potential to build ensemble classifiers for Microarray and other biomedical data, aiming to enhance the accuracy of prediction models.

Reviewing existing research and implementations shows that FPGA-based KNN designs can achieve substantial speedup and power efficiency improvements compared to traditional computing platforms. FPGAs’ parallelism and reconfigurability enable concurrent processing of multiple data points, resulting in faster classification or regression predictions. These advancements have shown promising results in addressing the challenges of large-scale datasets and real-time requirements. However, FPGA-based KNN implementations still face memory constraints [84], design complexity, and algorithm scalability. There is a need for further investigation into innovative approaches that resolve these obstacles and enhance the precision and efficiency of FPGA-based KNN systems.

4.4.2. Decision tree and random forest classifiers

Implementing decision trees and Random Forest algorithms on FPGAs has attracted considerable interest in biomedical applications. FPGAs provide parallel processing capabilities and hardware-level customization, making them ideal for real-time and resource-constrained applications. Decision tree and Random Forest algorithms are extensively utilized in biomedical domains for disease diagnosis, patient monitoring, and healthcare decision-making. This paper examines the benefits, challenges, and advancements of utilizing FPGA technology to accelerate decision tree and Random Forest algorithms for biomedical applications. By exploiting the parallelism and reconfigurability of FPGAs, these implementations are anticipated to improve the efficacy, precision, and speed of biomedical data analysis, leading to enhanced healthcare outcomes and individualized treatment strategies. In this section, the existing literature for hardware implementation of the decision tree/random forest algorithm on FPGA is reviewed, and several publications are identified.

Struharik [91] presented four distinct architectures, including a single module per level (SMPL) and SMPL-P architectures, as well as a universal node (UN) and UN-P architectures with varying hardware complexity, throughput, classification speed, and speedup. Using an FPGA, Narayanan et al. [92] proposed a decision tree classification that was 5.58 times faster than a software implementation. They focused on optimizing the compute-intensive kernel, Gini score computation, by employing a bitmapped data structure and rearranging calculations to reduce bandwidth requirements. Saqib et al. [93] developed a pipelined architecture for parallel decision tree binary classification that outperformed existing hardware implementations with a speed improvement of 3.5 times. The architecture included a high-speed communication unit to enhance data processing and transmission.

Meanwhile, Kulaga et al. [94] implemented decision trees and their ensembles using Vivado High-Level Synthesis, optimizing classification accuracy, throughput, and resource utilization. Their approach of employing fixed-point arithmetic and processing significant sample inputs resulted in the highest classification throughput and resource utilization, as indicated by the results. Barbareschi et al. [95] proposed a Von Neumann architecture dubbed tree visiting processing unit (TVPU) and an FPGA accelerator architecture, attaining remarkable efficiency and speed. In addition, they offered an FPGA accelerator architecture based on majority voting citeBarbareschi2015 and demonstrated a 114-fold increase in performance and a mere 0.03% increase in energy consumption compared to software approaches. The authors introduced an FPGA accelerator architecture design for the decision tree classifier and its ensemble using majority voting [96]. This architecture is based on a pipelined technique and follows the decision tree design presented in Ref. [97]. Compared to the software approach, the proposed architecture offers a 114-fold increase in performance with minimal energy usage. Tong et al. demonstrated the advantages of FPGA implementations for decision tree algorithms in biomedical applications by showing increased speed, enhanced throughput, and effective resource utilization.

In a different paper, Barbareschi [98] proposed hardware implementation of a decision tree using Xilinx Virtex-5 XC5VLX110T FPGA device to determine the scalability of required resources. In Ref. [99], an FPGA-based decision tree classifier for intrusion detection systems is proposed. The architecture includes three modules: transformation and integration, semantic classifier, and post-reasoner. The design achieves excellent performance with high decision box rates and low delay times using pipelined decision boxes and a Boolean Net. The Hybrid DT (HDT) [100] offers simplified training and obtains an 8 $\times$ speedup compared to conventional decision trees. Its hardware implementation permits the FPGA to operate at a maximal frequency of 125 MHz, resulting in a 1000 × acceleration over software-based performances and a 60 × acceleration over existing FPGA training accelerators. In addition, a wearable intelligent Electromyography (sEMG) recorder with incorporated gradient boosting decision tree (GBDT)-based hand gesture recognition is proposed in Ref. [101], using a low-latency parallel implementation of GBDT. The neural signal processing unit (NSPU) based on GBDT is implemented on an FPGA adjacent to the analog front-end (AFE) microprocessor. Overall, 91% of the hand gestures that were tested were correctly identified. Recent research has proposed an FPGA-based Automatic Pill Dispenser with a DT Classifier [102]. The authors suggested that the DT algorithm could classify features with high accuracy among all ML algorithms and manage complex datasets. In this study, the Decision Tree algorithm is developed for a sample time-based tablet dispensing dataset. Using a real-time Xilinx Artix-7 FPGA device, the developed decision tree-based classification is encoded in HDL and validated. Overall, the studies that were looked at show that using FPGAs to implement decision tree algorithms in biomedical applications has benefits, such as increased speed, throughput, and use of resources [[[91–99]]]. These results contribute to the expanding corpus of knowledge on leveraging FPGA technology for efficient and high-performance implementations of decision trees. Table 4 provides information regarding the throughput and clock rate of the architectures above.

4.4.2.1. Works related to accelerating RF/DT training on FPGAs. A few studies on training RF/DT on FPGA platforms have been proposed. FPGA is utilized in Ref. [92] to speed up the search for the optimal split point (weak learner) while the host computer conducts the sorting operation. Vertical data parallelism is the foundation of the parallel scheme. Chrysos et al. presented HC-CART [103], a heterogeneous GPP-FPGA system to address DT training issues. The system is equipped with an FPGA-based coprocessor designed to expedite the partitioning of categorical attribute lists. On a Convey HC-1 server, the coprocessor consists of multiple FPGAs and a scalar processor. Utilizing a large memory bandwidth and scalable shared memory, this architecture optimizes parallel operator networks across FPGAs. Like [92], the system parallelizes processing using vertical data parallelism. In Ref. [104], a proposed FPGA architecture speeds up the training of DT ensembles using a bagging technique [105]. The architecture incorporates HereBoy [106], an evolutionary algorithm that induces oblique DTs. While the hardware implementation does not utilize data parallelism, a parallel processing module computes the oblique weak learners involved in the training algorithm. Table 5, which was introduced previously, compares multiple FPGA-based works.

4.4.3. *Support Vector Machines*

Support vector machines (SVM) are a supervised algorithm for object labeling [107], including classifying microarray gene expression profiles and cancer. SVMs also find applications in classifying protein and DNA sequences, microarray expression profiles, and mass spectra [108]. The computational complexity of SVM algorithms poses challenges for embedded systems, leading to efforts in optimizing performance and cost through hardware implementations, particularly on FPGAs. FPGA hardware architectures have emerged as a promising solution for overcoming the constraints of embedded systems. This section offers an overview of FPGA-based architectures for implementing SVMs, addressing the need for efficient and effective SVM implementations in various biomedical and real-time applications. Due to their computational complexity, software implementations of SVM that attain high accuracy could be better for embedded applications. Reconfigurable computing provides a solution for attaining high-performance computing with minimal expenses and power usage [109]. The hardware implementation of SVM classifiers can be divided into six distinct categories based on their architecture (Fig. 10). Some studies have investigated multiple architectures and, as a result, belong to various categories. Reconfigurable computing enables the efficient and economical implementation of SVM algorithms, making them appropriate for embedded applications.

4.4.3.1. Architectures using parallel pipelined structure. Several pipelined architectures for accelerating SVM classification using FPGAs for effective parallel processing have been proposed. One study [110] used an FPGA Xilinx Virtex-6 to construct a fully pipelined architecture. To implement an SVM classifier with 760 support vectors, 768 DSP48E1 slices and 800 Block RAMs (BRAMs) were used. At a frequency of 370.096 MHz, the architecture obtained a throughput of 2.89106 classifications per second. Another pipelined architecture that aims to achieve a flexible SVM with adaptable input data, kernel selection, dimensions, and support vectors [111]. This architecture supported the runtime selection of linear, polynomial, and Radial Basis Function (RBF) kernels. For exponential function computation, it utilized embedded DSP-based Multiply-Accumulate (MAC) units, a Look-Up Table (LUT)-based adder tree,

Table 4
A comparison among different decision tree implementations mentioned in section 5.2.

Reference	Maximum clock rate	Average Throughput
[91]	100 MHz	SMPL-P > SMPL = UN-P < UN
[92]	100 MHz	3.24 Gbps
[93]	100 MHz	96.26%
[94]	100 MHz	1.3508 MSa/s
[95]	360 MHz	26.91 Gbps
[96]	(Not specified)	112.8796 MS/s
[97]	180 MHz/334 MHz	460 Gbps/6 Gbps
[98]	450 MHz	–
[99]	(Not specified)	562113546.9 (float/s)
[100]	125 MHz	–

Table 5
The difference among various FPGA-based implementations of RF/DT.

Reference	Part mapped on FPGA	Supported attribute weak learner used class in hardware
[92]	Searching of optimal split	numerical axes-parallel
[103]	Searching of optimal split	categorical axes-parallel
[104]	Complete training	numerical oblique/non-linear

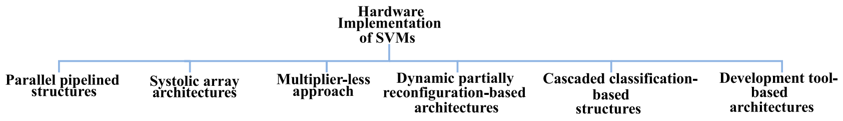

Hardware Implementation of SVMs
Parallel pipelined structures
Systolic array architectures
Multiplier-less approach
Dynamic partially reconfiguration-based architectures
Cascaded classification-based structures
Development tool-based architectures

Fig. 10. Different types of hardware implementing for SVM classifiers.

and the Xilinx Coordinate Rotation Digital Computer (CORDIC) [112] IP core. The RBF kernel attained a speed of 50 MHz by employing fixed-point arithmetic for dot-product calculations and single-precision floating-point format for all other calculations. A pipelined architecture for a universal coarse-grained reconfigurable architecture capable of implementing multiple ML tools, including SVM, was proposed in Ref. [113]. The SVM implementation employed reconfigurable blocks, adders, multipliers, and subtractors for partial sums governed by an FSM model. Compared to software implementations, this implementation exhibited significant acceleration while effectively utilizing hardware resources. Another hardware architecture demonstrated in Ref. [114] used FPGAs' parallelism and pipelining properties. It utilized a standard single-precision floating-point format and addressed generation counters to access data stored in BRAMs to perform required calculations. The architecture's maximal clock frequency was 200 MHz, and its simulation accuracy was 97.87%. The study [115] introduces a parallel architecture consisting of a two-stage pipeline that employs resource sharing to facilitate both linear and nonlinear SVM classification. The circuit, which included shared adders and multipliers for performing inner product computations, achieved a frame rate of 33.8 fps when operating at a maximum frequency of 152 MHz for 640×480 images. Other architectures for three-class SVM identification systems have been proposed [116,117]. These architectures used fixed-point arithmetic with variable bit lengths to balance identification precision and hardware area.

Additionally, a combination of two-class classifiers was implemented, resulting in an 18% increase in processing performance for inner-product computations. An improved two-pipelined-stage architecture was proposed in Ref. [118], which obtained a system throughput greater than 21.2 frames per second at 100 MHz with an accuracy greater than 90%. Reference [119] describes a pipelined FPGA scheme that implements a simplified SVM algorithm based on posterior probability. The structure utilized LUTs for calculating sigmoid functions and added, multiplied, and divided for other computations. Compared to the original floating-point algorithm, experimental results revealed a marginal decrease in recognition rate and a reduction in computation complexity, making this algorithm plausible for real-time constraints. A comparison was conducted between FPGA and GPU implementations of a human epidermis classifier [120,121]. The FPGA implementation used entirely pipelined architectures, whereas the GPU implementation was extremely power-hungry. The findings indicated that the FPGA implementation outperformed the GPU implementation when handling a small number of image pixels. However, the GPU exhibited superior performance for larger pixel counts, albeit at the cost of higher power consumption, making it less suitable for embedded applications.

4.4.3.2. Systolic array architectures in SVMs. The systolic array architecture combines parallelism and pipelining techniques to enhance computation speed. It employs an array of simple processors (PEs) for efficient data transmission and memory management [13]. In the context of matrix multiplication-based applications, the systolic array architecture is implemented using parallel FPGAs. Kyrkou and Theocharides introduced the Systolic Chain of Processing Elements (SCoPE) to realize a generic systolic array for SVM object classification in embedded image and video applications [122]. Building upon this framework, a scalable, flexible, and adaptive parallel architecture was proposed in Ref. [123]. The hardware implementations achieved high frame rates of 40, 46, and 122 fps for the evaluated applications, without compromising detection accuracy.

4.4.3.3. The multiplier-less approach. The multiplier-less approach to hardware design seeks to reduce complexity by omitting computationally intensive multipliers. Introduced in Ref. [124], a hardware-friendly kernel function provides acceptable classification performance without using traditional Gaussian kernels. In Ref. [125], a straightforward hardware architecture employing the hardware-friendly kernel for classification and regression tasks is presented. The system replaces multipliers with shifts and adds using parallel processing and the CORDIC iterative algorithm. The hardware simulation obtained a 4 % error rate for the 4-class classifier while operating at 30 MHz and utilizing 75 % of the device logic. Floating-point arithmetic and up to thirty to forty support vectors per binary classifier resulted in an error rate of zero percent. In Ref. [126], another hardware architecture employing the multiplier-less kernel is proposed, similar to the previous hardware-friendly kernel in Ref. [124]. The design employs straightforward shifters for multiplication and the CORDIC algorithm for the exponential function. Although no FPGA implementation is provided in channel equalization experiments, the proposed kernel obtains classification accuracy similar to the original radial kernel. In Ref. [127], a hardware implementation for rapid SVM based on an FPGA is proposed. The design consists of three subcircuits for performing kernel calculations using an iterative algorithm with adders and shifters. The implemented scheme performs computations six times quicker than previous CORDIC circuit implementations [128] with minimal hardware resources. In Ref. [129], an embedded hardware SVM implementation utilizing the hardware-friendly kernel is presented. The system comprises six primary components, one of which is a Sum of Absolute Differences (SAD) Calculator. The implementation exhibits minimal resource utilization, occupying approximately 167 slices in the target FPGA devices, and is, therefore, suitable for onboard image analysis and compression. Similarly, in Ref. [130], a SAD-based tree formation is utilized for 1-norm computation between vectors, thereby accelerating processing time. This research investigates the precision of support vector classification (SVC) input parameters using fixed-point arithmetic, obtaining acceptable regression performance with a mean square error of 0.004 or less. Using a parallel pipelined systolic array architecture [131], suggests a multiplier-free kernel implementation. Simple shift and add operations reduce complexity and energy consumption in hardware. Additionally, the study employs three distinct classifiers that consume less energy than the vector product kernel. These studies demonstrate the efficacy of multiplier-less approaches and hardware-friendly seeds in implementing SVM algorithms, resulting in enhanced performance, reduced complexity, and increased resource utilization in FPGA-based implementations.

4.4.3.4. SVM architectures based on DPR technology. DPR is a feature offered by modern FPGA, resulting in more efficient hardware resource utilization [132]. DPR is used to design dynamically adaptable computing systems that can spontaneously reconfigure dynamically selected areas on the FPGA while other components continue to operate [13]. DPR allows the time-sharing of physical resources to execute multiple design modules called run-time reconfiguration (RTR) [133]. Fig. 11 illustrates the fundamental premise of partial reconfiguration (PR).

The work [134] presents an SVM architecture comprised of four blocks, Memory, Kernel Computation, Accumulation, and Decision Making, in a modular structure employing DPR and a systolic array architecture. On the Xilinx ML 403 FPGA platform board, the implementation obtains a maximum speedup of 85 $\times$ over the equivalent GPP. This demonstrates the suitability of FPGAs for bioinformatics applications employing SVM-based analysis. A similar work explores partial reconfiguration schemes for Xilinx to optimize power consumption [135]. The systolic array architecture efficiently manages memory, reduces complexity, and facilitates data transfer. After reconfiguration, the total power consumption is reduced by 3–5%, decreasing from 2.042 W to 2.021 W. Hussain et al. [136] continue their prior efforts [134] by designing two systolic array-based structures for various dataset sizes. In addition, they present two DPR architectures for the SVM classifier that achieve approximately 61 $\times$ and 49 $\times$ speedups over the corresponding GPP. DPR implementations reduce reconfiguration time considerably compared to non-DPR performances. In their extended work, Hussain et al. introduce a DPR implementation of an adaptive architecture for a multiclassifier based on SVM/KNN [88]. The design permits the dynamic swapping of SVM and KNN classifiers with distinct parameters. Compared to a non-DPR implementation, the multiclassifier DPR implementation reduces reconfiguration time by at least eightfold while requiring fewer hardware resources. As shown by these studies, extensive speedups, decreased power consumption, efficient memory management, and adaptable classifier configurations are just some of the benefits that DPR and systolic array architectures bring to FPGA-based SVM implementations.

4.4.3.5. Structures based on a cascaded classification System. The cascaded classification architecture comprises multiple classification phases organized in a cascading structure, aiming to accelerate the classification process (Fig. 12). In Ref. [137], a hardware architecture for SVM cascade processing and a hardware reduction technique was proposed and implemented using a Virtex 5 FPGA. This method employs the multiplier-less approach, substituting shift operations for multiplication operations. The hybrid cascaded architecture combines parallel and sequential processing, employing pipelined PEs for the primary early stages and an SVM classifier with a higher level of complexity. The cascade architecture is five times faster than a single parallel SVM implementation, processing 640 $\times$ 480 images at an average rate of 70 fps. The suggested method for hardware reduction decreases custom logic resources by 43% and energy consumption by 20%, with a minimal loss of 0.7% in classification accuracy of 84%.

In [138], a cascaded FPGA-based SVM classifier with a parallel structure of multipliers and a pipelined adder tree for kernel computations was proposed. The data path was divided into fixed-point and single-floating-point precision. A novel cascade classifier system was also created, leveraging the heterogeneous architecture characteristics and adopting a hardware-friendly design approach. The architecture featured two interconnected classifiers by integrating a primary, low-precision classifier with a higher-precision classifier that incurs a more significant area cost. This heterogeneous design exhibited a performance advantage of over seven times compared to CPU systems, surpassing the capabilities of other FPGA and GPU systems. In Ref. [139], the work was expanded to

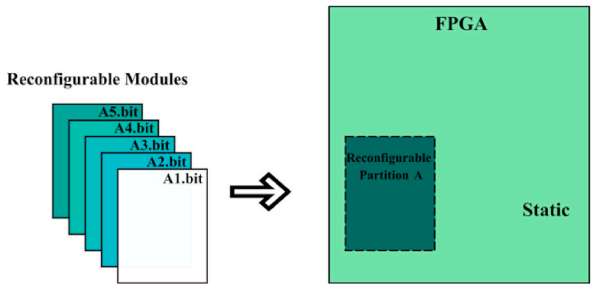


Fig. 11. Fundamental concept of Partial Reconfiguration [13].

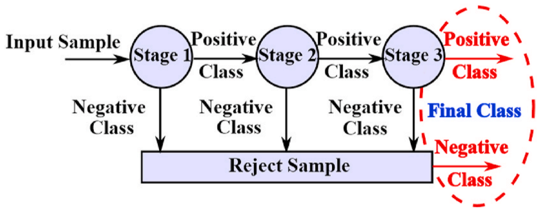


Fig. 12. Cascaded classifier scheme from [137].

include FPGA reconfigurability, allowing the cascade to transition between low- and high-precision classifiers. This enhancement helped performance even further. The proposed heterogeneous FPGA classifier optimized word lengths to accommodate the dynamic ranges of the dataset, thereby maximizing parallelization potential and facilitating custom circuitry. In Refs. [140,141], the previous work [137] is extended by incorporating a feature extraction mechanism to improve the accuracy of detection in the proposed architecture. Focus is placed on the efficient hardware implementation of cascaded SVMs for intelligent embedded systems, specifically for real-time classification applications. Classifiers based on Neural Networks are incorporated into the response evaluation procedure, thereby reducing the number of samples evaluated before the final, highly complex classification stage and enhancing classification speed. Tested on a Spartan 6 FPGA platform, the system obtains an 80% detection rate while processing high-resolution images in real-time, with a 5 × speedup over a single parallel SVM classifier. The proposed method for reducing hardware requirements in FPGA effectively decreases custom-logic resources by 25% and peak power consumption by 20% compared to a standard implementation. This reduction is achieved with minimal impact on classification accuracy, which only decreases by a negligible 0.7%. In Ref. [142], a cascade SVM classifier for melanoma clinical image classification using a scalable multicore architecture on a hybrid Zynq SoC platform is presented. Compared to a single SVM implementation, the design utilizes 34% of the resources and consumes 2 W of power, resulting in lower resource utilization and energy consumption. The classifier's high classification accuracy (97.9%) and rapid speed (1.8μs) make it suitable for integrating into a real-time, low-cost handheld medical scanning device to detect skin cancer.

4.4.3.6. Developing SVMs utilizing design tools. Significantly, few researchers have utilized the most advanced software design tools, which facilitate the complex hardware design process and reduce development time in lieu of the conventional HDL approach. The most up-to-date software design and development tools that have been utilized in the reviewed papers in this section are Xilinx System Generator, Synopsys Signal Processing Workbench (SPW), and High-level Synthesis (HLS). In Ref. [143], the authors employ Xilinx System Generator, a utility provided by Xilinx, to create a parallel hardware architecture for Xilinx FPGAs. This tool enables efficient system modeling and automated code generation using MATLAB/Simulink, resulting in high-performance designs. Quantization is performed using serial and parallel schemes with fixed-point numbers. For linear classification, simulation results indicate a maximum frequency of 202.840 MHz, while nonlinear classification achieves an error rate of 1.33% and an accuracy of 98.67%. The processing time for linear SVM simulation is 1.4 ms and 0.27 ms for nonlinear SVM simulation. The FPGA-based system exhibits reduced computation time for both classifiers compared to the MATLAB implementation.

A DSP slice-based processor is designed for ultrasound applications using the Xilinx System Generator, according to Ref. [143]. In comparison to software implementation, the results demonstrate high accuracy (93.6%) and a significant reduction in processing time. The results also showed reasonable hardware resource utilization (69.09% of DSP Slices, 26.37% of LUTs, and 75.71% of BRAMs were utilized). Table 6 demonstrates that the System Generator is also used in Ref. [144], where it is used to design a hardware architecture of linear SVM for three-class facial expression classification, with a confirmed accuracy of 97.87% at a frequency of 132.7 MHz (using a combination of parallel and series systems). Using Synopsys Signal Processing Workbench (SPW), as described in Ref. [145], is another method. This software utilizes a model-based design methodology and permits the description of highly parallel systems. The processing speed of the hardware implementation was roughly 2.53 × that of the software implementation, despite a minor loss of accuracy. In Ref. [142], a comparable approach for melanoma detection was presented, which involved the use of BRAM interfaces for data transfer instead of streaming data via the stream interface/bus with the Direct Memory Access (DMA) IP, as described in Ref. [146]. Although additional resources were utilized in comparison to Ref. [146] and the amount of power consumed increased slightly, the area and power values decreased in comparison to other applicable implementations. The most noteworthy aspect of this work is that a significant acceleration factor of 32 is achieved in comparison to related software implementations, with a 97.92% accuracy and a processing time of 11.46 μs (2865 clock cycles) at 250 MHz frequency, while the proposed system demonstrated efficient utilization of hardware resources, and consumed a total power of only 2 W in the Zynq implementation.

On the other hand, HLS allows engineers to develop and implement FPGA applications using C, C++, SystemC, and MATLAB without prior hardware architecture expertise [147]. A systematic two-level strategy and prototype architecture are used to implement an HLS-based SVM IP for electrocardiogram (ECG) analysis and arrhythmia diagnosis [148]. The suggested method optimizes the code structure utilizing data and instruction-level parallelism then uses HLS directives to investigate various architectural choices. The findings showed a maximum 98.78% execution latency gain over the default Vivado-HLS optimization of the original SVM algorithm. According to the authors, the optimized SVM accelerator approach implemented on a Zynq programmable SoC outperforms pure software implementations on different single- or dual-core target platforms. The proposed method achieves speedups ranging from 10 × to 78 × , as claimed by the authors. The Xilinx Vivado HLS tool implements a low-cost SVM IP [142,146,149][–][152] on a state-of-the-art FPGA platform, "Zynq SoC," with a hybrid architecture integrating a processor and FPGA in a single device (see

Table 6
A summary of different architectures for SVMs.

Reference	Hardware implementation	SVM type	Kernel	FPGA platform	Application
[110]	Parallel pipelined	Binary	RBF Polynomial Sigmoid	Xilinx Virtex-6	–
[111]	Parallel pipelined	Binary	Linear RBF Polynomial	Xilinx ML505	Pedestrian detection
[113]	Parallel pipelined	Binary	RBF Polynomial	Xilinx Virtex-7	UCI datasets
[114]	Parallel pipelined	Multiclass	Linear	Xilinx ML510 (Virtex-5 FXT)	Facial expression classification
[115]	Parallel pipelined	Binary	RBF	–	–
[116,117]	Parallel pipelined	Multiclass	Linear	Altera Stratix-IV	Colorectal cancer detection
[119]	Parallel pipelined	Multiclass	Linear Sigmoid	Xilinx Virtex-5	Language Recognition
[120,121]	Parallel pipelined	Binary	Gaussian	Xilinx Virtex-5/Xilinx Spartan 6	Skin classification
[123]	Systolic array	Binary	RBF Polynomial	Xilinx ML505	Object detection
[125]	Multiplier-less	Multiclass	Hardware-friendly	Cyclone II (EP2C20)	Image recognition
[126]	Multiplier-less	Multiclass	Digital kernel	–	UCI datasets
[127]	Multiplier-less	Binary	Hardware-friendly	–	–
[130]	Multiplier-less	Binary	Hardware-friendly	Xilinx Virtex-5/Spartan-3E	Satellite onboard
[131]	Multiplier-less/Systolic array	Binary/Multiclass	Linear polynomial	Xilinx Virtex-7	Fisher's iris dataset
[88,134,136]	DPR	Binary	Linear	Xilinx ML403	biomedical data classification
[137,155]	Cascaded classification	Binary	Linear polynomial	Xilinx ML505	Face detection
[138,139]	Cascaded classification	Binary	Gaussian polynomial sigmoid	Altera's Stratix III	MNIST dataset
[140]	Cascaded classification	Binary	Linear polynomial	Xilinx Spartan-6	Face detection
[141]	Cascaded classification	Binary	Linear polynomial	Xilinx Spartan-6	Face/pedestrian detection
[142,152]	Cascaded classification/Development tool-based	Binary	Linear	Xilinx Zynq-7	Melanoma detection
[143]	Development tool-based	Multiclass	Linear Gaussian	Xilinx Virtex-4	Persian handwritten digits dataset
[156]	Development tool-based	Binary	RBF	Xilinx Zynq-7000	Ultrasonic flaw detection
[144]	Development tool-based	Multiclass	Linear	Xilinx Virtex-5	Female facial expression classification
[145]	Development tool-based	Multiclass	RBF	Xilinx Virtex-II	Multispeaker phoneme recognition
[146,149–151]	Development tool-based	Binary	Linear	Xilinx Zynq-7ZC702	Melanoma detection
[148]	Development tool-based	Binary	RBF	Zedboard Zynq	ECG-based arrhythmia detection
[153]	Development tool-based	Binary	RBF	Zynq-7 ZC706	ECG-based arrhythmia detection
[154]	Development tool-based	Multiclass	Linear	Xilinx ML605	MNIST and CIFAR-10 datasets

Table 6). The experimental results met embedded system limits while retaining classification accuracy rates. In another work, Vivado HLS [153] uses an optimized approximation SVM accelerator to identify arrhythmia from ECG signals, like [148]. The suggested SVM coordinates two approximation computing systems with HLS tool tweaking knobs to boost efficiency. The approximation HLS-SVM classifier detected arrhythmia from ECG signals with 15 × acceleration and 96.7% accuracy. HLS is used to design an energy-efficient embedded binarized SVM (eBSVM) architecture with binarized inputs and weights [154]. The results revealed reduced power consumption (3.6× greater energy efficiency) and execution time (45 × speed improvement) than CPU and GPU implementations, with a 1.6 × accuracy loss. A summary of all discussed SVM architectures is presented in Table 6 to have a comprehensive comparison.

4.4.4. Clustering algorithms

Using clustering algorithms, unlabeled data can be classified into multiple categories in polynomial time. Typical clustering algorithm implementations in biomedical research [157] include gene expression data analysis, genomic sequence analysis, MRI image analysis, and document mining. FPGAs enable parallel processing, which permits the acceleration of clustering computations and the reduction of overall execution time. This parallelism is especially advantageous when working with large datasets or real-time clustering applications. However, implementing clustering algorithms on FPGAs presents its own set of obstacles. Designing effective and scalable hardware architectures for sophisticated clustering algorithms requires both clustering algorithm and FPGA design expertise. In addition, it is essential to optimize hardware resources and memory usage while maintaining a great degree of accuracy. Processing of high-resolution medical images requires a great deal of memory and computation. FPGA parallel arrays can speed up image processing. Thus, references [158,159] introduce a high-throughput data clustering algorithm with a two-dimensional organization, providing fast performance for general-purpose clustering tasks. The algorithm presented in Ref. [158] achieves linear processing time based on the number of pixels, ensuring image occupancy and size stability. The implementation on xc5vlx300 FPGAs showed scalability of hardware size and clock speed with the grid size. The program achieved a processing time of 6 ns and utilized 1% logic for an 8 × 8 grid and 20 ns and 21% logic for a 120 × 32 grid. The clock speed reached 550 MHz. A comparable approach to Ref. [158] is given for FPGA-implemented pixel detectors in Ref. [159]. The program clusters data and analyzes each cluster to extract information. One xc5vlx155 FPGA implements the algorithm. 60% of its logic processes one S-Link input. An xc5vlx330 uses 15% area, while an xc5vlx155 uses 30%. Using Xilinx Spartan 6 LX150T, a multi-core 2D-clustering technique for real-time image processing is reported in Ref. [160]. The algorithm reduces FPGA resource utilization by using a moving window strategy. It works with the ATLAS Fast Tracker Data Formatter in various clustering applications. Only 2% of FPGA Slice LUTs are used to reach over 80 MHz.

The results indicated that the implementation would occupy approximately 20% of FPGA resources. In Ref. [161], a model-based clustering algorithm analysis and FPGA implementation of neural electrical signals are presented to address the problem of classifying neuronal action potentials. However, there are still numerous flaws due to interference and noise, among other constraints. For instance, the Wave-clus algorithm's classification action potential increases the error source in the first phase. Although a novel FPGA implementation technique for the clustering algorithm has been proposed, neither parallelism nor the greedy method has been employed in this study. Incremental Clustering of Evolving Data (ICED) is an unsupervised clustering method [162] used for sensor data that changes over time. This technique runs 39 times faster on a Xilinx Virtex 5 than on a 3 GHz Intel Xeon processor in C. Table 7 summarizes the implementations reviewed in this section.

4.4.5. Mean-shift clustering

Introduced by Fukunaga and Hostetler [163], the Mean-Shift Clustering algorithm provides a valuable method for clustering data based on their natural distribution, making it a significant unsupervised learning algorithm. Mean-shift clustering, unlike other methods, does not presume a particular shape for the issuance or limit the number of clusters, allowing for a flexible and adaptable approach. Due to its robustness and minimal parameterization requirements, this algorithm has found widespread use in real-time image processing applications. With the advent of FPGAs, it is possible to investigate hardware implementations of the Mean-Shift Clustering algorithm, utilizing the parallel processing capabilities of FPGAs to accomplish accelerated and efficient clustering tasks. This integration of Mean-Shift Clustering and FPGA technology can improve real-time performance and enable applications in domains such as object tracking, image segmentation, and anomaly detection. This section aims to review the implementation of the Mean-Shift Clustering algorithms on FPGAs and the possibility of leveraging hardware acceleration for faster and more efficient clustering in real-time image processing systems. The paper [164] proposes a multiprocessor architecture utilizing a parallel processing model of the mean shift clustering algorithm for real-time applications on an FPGA (Spartan 6 XC6SLX75T). Multiple homogeneous PEs are

Table 7
+++ A summary of different architectures for Clustering Algorithms.

Clustering Algorithm/Architecture	Reference	Proposed FPGA	Application
2-D fast general-purpose algorithm	[158]	Xilinx xc5vlx300/xv5vlx330	Real-Time Image Processing
2-D fast general-purpose algorithm	[159]	Xilinx xc5vlx155/xv5vlx330	Pixel detectors
multi-core 2D-clustering	[160]	Xilinx Spartan 6 LX150T	real-time image processing
model-based clustering algorithm	[161]	–	neural electrical signal classification
clustering algorithm (Incremental Clustering of Evolving Data)	[162]	RTAX2000 (Xilinx Virtex 5)	Pulse Deinterleaving

interconnected via a pipeline in this design. With 10.31% of total device slice registers and 33% of total slice LUTs utilized, the evaluation of the algorithm on 99 images revealed that only a tiny portion of FPGA resources were employed. With and without fractional bits, clustering rates ranged from 351 to 8772 ps in the experimental data. The proposed architecture obtained impressive speedup values, ranging from a minimum of 128,735 with seven fractional bits and a single PE to a maximum of 2,615,611 with no fractional bits and 16 PEs. The parallel architecture's scalability permits efficient use of available resources and accommodates more PEs. This architecture outperformed other mean shift clustering implementations regarding area and resource consumption, optimum speedup, and throughput (in fps) [165,166]. In Ref. [165], a scalable architecture is assessed with 32-bit fixed-point pipelines on a PROCStar III board containing four Stratix III E260 FPGAs. The system requires additional FPGAs to accelerate the processing of a 464 by 332 8-bit grayscale image. As an improvement, reference [166] implements the mean shift filter on an FPGA for color image segmentation. This system ran at 100 fps for 768 × 512 pixels photos with a 15 × 15 filter and 50 for 31 × 31. The Xilinx XC4VLX160 on the RC20004 FPGA board implemented the design. The 31 × 31 filter consumed 376 BRAMs in the suggested implementation. Similar to Ref. [166], the paper [167] describes a color image segmentation technique on an FPGA using the mean shift algorithm.

The paper highlights the feasibility of achieving real-time processing by adopting a relaxed data management approach and enabling pipeline data processing. The implementation requires approximately 400 BRAMs, and the efficacy of 768 × 512 pixel images was sufficient for real-time applications. Despite the satisfactory results of the mean-shift method in image segmentation, its computational challenges have hindered its widespread adoption in image processing. The method's long runtimes have limited its applicability to real-time applications, particularly for high-resolution images exceeding VGA (640 × 480). Hence, Craciun et al. [168] introduce a scalable architecture that accelerates the Gaussian mean-shift algorithm by clustering multiple pixels in parallel using dedicated hardware pipelines. This system is ideal for embedded applications due to its accelerated performance and low power consumption.

In comparison to the most recent GPU implementations, this scheme achieved comparable performance while consuming less energy. The segmentation results were compared to a floating-point C code baseline to ensure that the fixed-point approximations did not result in any errors. In Ref. [169], a generalized Laplacian of Gaussian (LoG) filter is utilized for real-time histopathology applications, specifically for detecting cell nuclei. The approach involves using local maxima as seed points for cell centers and employing a mean-shift clustering algorithm to combine multiple detections. The design is implemented on an FPGA development board, resulting in a modest acceleration compared to a software implementation on a high-end computer. The FPGA design achieves a 34.5% performance boost over a standard CPU, mainly due to the lower clock rate of the FPGA board. Further improvements can be achieved by leveraging data and task parallelism in each sub-module. By employing parallelized pipeline design, reference [170] proposes a novel method that is more productive and less time-consuming. The proposed architecture includes eight pipelines and scales up to 128 on a single FPGA. One FPGA can accommodate a maximum of 128 pipelines, allowing the probability density function (PDF) gradient to be computed simultaneously at 128 separate locations. The performance of this fixed-point architecture is segmented into two grayscale images with varying resolutions and complexities, and the results are contrasted with the implementation of floating-point data. The misclassification errors for two images demonstrate that up to 10% of pixels in hardware converge to a distinct cluster. Xilinx is used to implement the hardware, and the FPGA provides satisfactory results compared to the previous results.

A parallelized pipeline design is proposed in Ref. [170], which offers increased productivity and reduced processing time. The architecture contains multiple pipelines, which can scale to 128 pipelines on a single FPGA. This allows for the simultaneous computation of the probability density function (PDF) the gradient at multiple locations. The efficacy of the fixed-point architecture is analyzed using grayscale images of varying resolutions and complexities. Compared to implementations utilizing floating-point numbers, the results demonstrate that the hardware obtains satisfactory performance with up to 10% of pixels converging into distinct clusters. The commission is carried out with Xilinx tools, and the FPGA-based approach produces favorable results compared to earlier findings. A new mean-shift architecture for scalable multiprocessor implementation is proposed in Ref. [171]. The proposed model employs multiple processors operating in parallel on independent data to accelerate algorithm convergence. The design can be implemented with adequate error margins using fixed-point arithmetic.

4.4.6. *K-means clustering*

The simplicity and effectiveness of the K-means algorithm make it a popular choice in biomedical data analysis. Several techniques have been employed to improve its execution performance or reduce the computation required. However, the oldest FPGA implementations of K-means date back about two decades [172,173]. Since then, FPGAs have become increasingly capable, datasets have significantly improved, and unsupervised and supervised ML algorithms have become a popular domain to accelerate. In this section, we explore previous works achieved from 2000 to 2023 regarding the FPGA implementation of the K-means clustering algorithm, focusing on the parallel implementation of FPGAs. Estlick et al. [173] investigated two modifications to the mean-shift clustering algorithm for enhanced parallelism: alternative distance metrics and reduced input vector word breadth. Regarding the complexity of implementation, the experimental analysis revealed that the Manhattan distance is the optimal metric. It was discovered that the word breadth of input data could be reduced without affecting the quality of clustering when employing the Manhattan distance metric without word width reduction. The implementation utilized a pipeline architecture for distance computation, minimum distance determination, and cluster association. The FPGA-based performance deployed on the Xilinx Virtex platform with a clock rate of 50 MHz was 200 times faster than software execution on a 500 MHz Pentium III computer.

In [174], a k-means clustering system is presented for hyperspectral image segmentation implemented on the Altera Excalibur platform, comprised of an Altera Apex FPGA and an ARM CPU. The system was comprised of two implementations, one with an ARM CPU and the other with an NIOS soft-core CPU. The final implementation featured 32 PEs and direct memory access (DMA) for efficient data transmission in a hardware unit for distance computation. Compared to a Pentium III computer with a clock speed of 1 GHz, the system operating at 33 MHz was 11.8 × faster. A real-time k-means clustering system for large-size color images and many clusters is presented in Ref. [175]. The system reduces computational overhead by utilizing dynamically generated kd-trees [176] on an FPGA. The system is structured as a three-stage pipeline with phases for kd-tree construction, square-of-distance computation, and cluster center determination. The system achieves real-time performance of over 30 fps for 512 × 512 and 640x480 images and 20 to 30 fps for 768 × 512 images when implemented on the Xilinx Virtex-II platform with a clock rate of 66 MHz. Covington et al. [177] implemented k-means clustering using the Cosine Theta Distance Metric for document clustering. The system's three modules are Distance computation, comparison, and cluster center computation. On the Xilinx Virtex-E platform, the system operates at a clock rate of 80 MHz, obtaining a 26 × speed increase compared to a 3.6 GHz Xeon CPU. The Xilinx Virtex-4 platform enables the K-means algorithm to run at 250 MHz, achieving a 328 × speed increase. Cluster centers are stored in external memory connected to the FPGA. The system's performance is evaluated by assigning input vectors to the closest cluster centers.

The K-means clustering algorithm for image processing was implemented by Wang and Leeser [178] by adding a hardware floating-point division unit to the hardware implementation. Based on [173], they developed a cluster association structure, accumulators for pixel summation, and counters for monitoring cluster sizes. Calculating the cluster centers by dividing the accumulator values by the corresponding counters The system was deployed on the Xilinx Virtex-II platform, and three implementations were evaluated: software-only, FPGA with CPU-based division, and FPGA with FPGA-based division. For 50 and 1000 iterations of the algorithm, respectively, speedups of 11 × and 174 × were attained compared to the fully software implementation. There were no substantial differences in execution time between FPGA implementations that utilized CPU or FPGA division.

Some recent attempts at K-means implementation using FPGAs are discussed as follows:

Hussain et al. [179] introduced an FPGA implementation of parameterized K-means clustering for microarray data analysis. They parallelized the accumulation kernel based on the number of clusters and dimensions and contrasted the performance of FPGA, CPU, and GPU implementations. The entire algorithm was implemented in hardware using data-range-specific fixed-point arithmetic. The system was implemented on the Xilinx Virtex-4 platform, obtaining speedups of 10.33 × and 51.73 × for single-core and five-core implementations, respectively, when compared to a Core 2 Duo computer operating at 3 GHz. In addition, they designed a highly parameterized FPGA core for K-means clustering [180], which outperformed GPP and GPU implementations in terms of energy efficiency and processing performance. Their implementation was 615 × and 31 × more energy efficient than GPP and GPU,

Table 8
A summary of FPGA-based implementation of the k-means clustering algorithm for biomedical applications in recent years.

Reference	Architecture	Maximum speed-up compared to software implementation	Target platform	Application
[173]	Parallel pipelined	200 ×	Xilinx Virtex	Processing multi-spectral images
[174]	PEs in parallel	11.8 ×	Altera Excalibur	Segmenting hyperspectral images
[175]	Parallel pipelined using kd-trees	(Not specified)	Xilinx Virtex-II	color image processing
[177]	Modular structure	26 × 328 ×	Xilinx Virtex-E Xilinx Virtex-4	document clustering
[178]	Floating-Point Divide	174 ×	Xilinx Virtex-II	Multispectral image processing
[179]	Multi-core/parallel	51.7 ×	Xilinx Virtex-4	Bioinformatics/Microarray data
[180]	Highly parameterized FPGA	8.4 ×	Xilinx ML403	Bioinformatics/Microarray data
[181]	Multi-core/Parallel pipelined	3.2 ×	Xilinx Virtex-4	(not specified)
[182]	Parallel implementation	9 ×	Xilinx Virtex-II	Data Clustering
[183]	Parallel pipelined	16.32 ×	Altera Stratix III E110	Object detection
[184]	Parallel pipelined	(Not specified)	Xilinx Virtex 6	real-time data mining
[185]	Parallel pipelined using kd-trees	(Not specified)	Xilinx Virtex 7	Image processing
[186]	Parallel using HW/SW co-design	118 ×	Altera stratix III E110	Multimedia applications
[187]	Parallel pipelined	20.5 ×	Xilinx ZedBoard	(not specified)
[188]	Heterogenous/pipelined	3.8 ×	Intel QuickAssist	(not specified)
[189]	Parallel using Altera SDK for OpenCL	21 ×	Stratix V A7	Multiple applications
[190]	Multi-core/Parallel	10 ×	Xilinx Zynq-7000	Big data analysis
[191]	PEs in parallel	368 ×	(not specified)	Big data analysis
[192]	Parametrizable/parallel	(Not specified)	Xilinx Zynq7020	Data mining
[193]	Heterogeneous/parallel	4 ×	Xilinx VC707 evaluation boards (EVBs)	Big data analysis
[194]	Parametrizable/parallel	(Not specified)	Xilinx Artix7	Data mining
[195]	Parallel pipelined/ Heterogeneous	12.7 × /2.4 ×	Intel Xeon + Arria 10	Data mining
[196]	Parallel implementation	(Not specified)	Virtex-6	Big data analysis

respectively. However, their work requires further optimization to utilize FPGA resources more efficiently. Nagarajan et al. [181] designed hardware for PDF approximation but modified it to implement correlation computation and k-means clustering. The k-means clustering hardware consisted of three stages.

- Calculation of the square of the distance
- Calculation of the minimal distance
- Update of the cluster center

The second and third stages resemble Saegusa and Maruyama's [175] implementation. The design was implemented on the Xilinx Virtex-4 platform, which resulted in a 3.2 × performance increase over a 3.2 GHz Xeon computer. The designers attributed the minor acceleration to the CPU's and FPGA's limited communication bandwidth. Singaraju and Chandy [182] designed an FPGA-based network switch architecture for k-means clustering implementation. Their system processed data as it traversed the network as Ethernet packets. All aspects of the algorithm were implemented on the FPGA using floating-point arithmetic for computations. They parallelized distance computation and minimum distance retrieval to enable cluster parallelism. Furthermore, the system supported data parallelism across multiple FPGAs, with each FPGA managing distinct data points. On the Xilinx Virtex-II platform, a single FPGA outperformed an Opteron 1.8 GHz computer by a maximum of 10 × . Comparing multiple FPGAs to a parallel software implementation with an equivalent number of nodes, the speedups reached up to 9 × . Several works have also been accomplished to implement k-means algorithms in the last decade, a summary of which is presented in Table 8. This section's remainder provides a summary of these publications.

In [183], a hardware solution for a K-means-based multi-prototype learning system's high computational demand is proposed. Compared to software, GPU, previous FPGA, and analog circuit implementations, the system's software-hardware collaboration results in a significantly quicker recognition rate. The co-processor design requires a portion of the FPGA's resources to achieve a 97.91% accuracy rate with rapid learning and classification speeds. On an Altera Stratix III E110 FPGA device, 12.9% of logic elements and 58% of block memory bits are utilized. Reference [184] describes a configurable FPGA architecture for high-speed K-means clustering algorithm implementation in real-time applications. The architecture uses pipelining and parallelism to achieve a maximal clock frequency of 400 MHz and a continuous throughput of 1 sample data per clock cycle, regardless of user inputs. Using tree-based data structures, reference [185] presents an FPGA implementation of a computationally efficient filtering algorithm for K-means clustering. The algorithm employs pipelining and parallelism with on-chip memory banks, boosting throughput, decreased resource consumption, and a 200 MHz clock frequency. In Ref. [186], a hardware/software co-designed method for K-means clustering that emphasizes nearest-neighbor searching is presented. The hardware architecture is optimized for low-dimensional feature vectors, increasing performance and adaptability. The implementation achieved 118 × speedup in comparison to software implementation. Work [192] proposes a highly parameterizable architecture for the Lloyds K-Means clustering algorithm, analyzing the effects of different parameters on cost, frequency, and performance. Similar research is conducted in Ref. [194], despite both papers presenting only simulated results without experimental validation. In Ref. [193], the K-means clustering algorithm is described for storing big datasets on a Hadoop cluster using FPGA-based hardware accelerators. The experimental results show that compared to a Hadoop cluster without FPGA-based hardware accelerators, this system can achieve a speedup of 4 × while clustering 125 million three-dimensional input datasets.

A highly efficient implementation of the k-means algorithm using a multi-core HW/SW co-design architecture is presented in Ref. [195]. A two-layer filtering algorithm enables parallel processing of binary kd-tree structures on each processing core of the ZYNQ, which is used to reduce performance time. Using this multi-core FPGA-based architecture on a ZCU102 evaluation board outfitted with a Zynq-7000, experimental results demonstrate a significant speedup of 330 × in comparison to a conventional software-only solution. KPynq was recently introduced in Ref. [197] as an FPGA-based architecture of K-means clustering for managing large-size, high-dimensional datasets. Compared to CPU-based implementations, it demonstrates superior performance, attaining speedups of up to 4.2 × and energy efficiency gains of up to 218 × . Another work [198] proposes an FPGA-based method for hardware acceleration of image processing with an 8 × speedup over software implementation. On the Intel Xeon + FPGA platform, software hardware acceleration of Kmeans clustering is attained [199], demonstrating significant performance enhancements over CPU threads or FPGA alone. In addition, a parallel implementation of the K-means algorithm on an FPGA is presented [196], demonstrating high throughput and achieving a speedup of about 15573 × compared to a partial serial implementation. These studies demonstrate the efficacy of FPGA-based solutions for accelerating K-means clustering and image processing operations. In Ref. [200], an FPGA-based High-Level Synthesis method is proposed for classifying cutaneous tumors using hyperspectral images (HSI). The algorithm includes preprocessing, K-means clustering, and SVM classification. Results show that the Virtex UltraScale + FPGA outperforms GPU models regarding power consumption and energy efficiency. While the FPGA architecture is not yet real-time, it demonstrates rapid pixel classification with lower power consumption than GPU-based solutions. This research highlights the potential of FPGAs for efficient and low-power classification of hyperspectral images in skin tumor analysis [201].

The following characteristics are shared by most implementations and can be highlighted in this section: The design process for all implementations begins with algorithm analysis. The clock rate of an FPGA is typically one to two orders of magnitude slower than that of a CPU, and this difference is compensated for by utilizing available parallelism. FPGAs can simultaneously perform several thousand operations depending on the type of operation and its implementation. Typically, pipelined implementations result in a quicker clock rate and execution time. Fixed-point arithmetic is often used because it is less complicated and uses fewer FPGA resources than floating-point arithmetic. Only some implementations execute the entire algorithm on hardware.

In general, implementations are divided into software and hardware components, and systems fully implemented in FPGA often utilize a softcore CPU to run the main program. Earlier performances of FPGA rarely used multiplication due to high resource costs, whereas recent FPGA families utilized by DSP applications contain highspeed hardware multipliers.

4.5. Mixture models

The Gaussian mixture model, the multivariate Gaussian mixture model, and the categorical mixture model are all subsets of the broader category of mixture models. Among these algorithms, the Gaussian mixture model (GMM) is widely utilized for biomedical ends such as pattern recognition (for example, to classify various limb motions) [202] and fuzzy image segmentation [203]. Implementing Gaussian Mixture Models on FPGA has the potential for high performance, real-time processing, and customizability. Utilizing parallelism and hardware acceleration can improve computational efficiency. However, implementing GMM on FPGAs presents additional obstacles. GMM algorithms involve complex computations, such as matrix operations and iterative optimization, necessitating cautious resource management and design on FPGA platforms. Balancing the utilization of FPGA resources, including memory and computational units, is crucial to ensuring efficient and effective GMM implementation.

In the following, this section overviews the related literature regarding FPGA-based hardware implementation of GMM algorithms.

Reference [204] describes a gas identification method using class conditional density estimation via GMMs that can rapidly identify gases with an accuracy of greater than 96%. In another study, a hardware implementation of a pattern recognition system based on a GMM classifier is presented in Ref. [205]. To reduce hardware complexity, the implementation employs Distributed Arithmetic (DA) and a novel exponential calculation circuit. With acceptable complexity, the GMM classifier shows superior classification performance compared to the KNN and Multilayer Perceptron (MLP) algorithms. Using DA, the hardware implementation obtained remarkable speeds of 5.402 × 106 classifications/second at a maximum frequency of 27.010 MHz and 5.318 × 106 classifications/second at a maximum frequency of 26.590 MHz. In contrast, the software implementation on a 2.6 GHz Pentium computer achieved a maximal speed of 0.067 × 106 classifications per second. The findings demonstrate the significant speed improvements of the hardware-based GMM implementation, which outperforms software-based methods by roughly two orders of magnitude.

In many applications, it is essential to estimate the parameters of a generalized linear model (GMM) from data before employing it. This computationally intensive operation for Gaussian Mixture Models can be accomplished by the Expectation-Maximization algorithm (EM-GMM). In Ref. [206], a pipeline-friendly Expectation-Maximization algorithm for Gaussian Mixture Models (EM-GMM) is proposed, designed for conversion into a fully-pipelined hardware architecture. Guo et al. implemented a fixed-point arithmetic evaluation unit for the Gaussian probability density function on an FPGA, outperforming CPU and GPU-based solutions. The FPGA implementation achieved a 517 × speedup over the CPU-based solution and a 28 × speedup over the GPU-based system, with a steady throughput of around 1.5 × 108 instances per second.

Similarly, in Ref. [207], a new design for the EM-GMM algorithm is introduced, tailored explicitly for reconfigurable platforms. The authors developed a fully pipelined EM-GMM algorithm and a customized Gaussian PDF evaluation unit. The FPGA-based solution demonstrated significant speedups, with a maximum 207 × speedup over a CPU solution and a 96 × speedup over a GPU-based solution. The energy efficiency of their answers on the Virtex-6 and StratixV FPGA platforms surpassed the CPU-based platform by 301 and 635 times, respectively.

Reference [208] introduced a hardware/software scheme for face recognition and verification using GMM-parts-based topology modeling. Their FPGA-based system outperformed a software implementation on a 3.3 GHz Core i3 processor by a speedup of approximately 5.1 × . Using an LX15 Virtex-5 FPGA operating at a maximum frequency of 142.65 MHz, the system obtained an identification accuracy of approximately 96.8%. Incorporating an exponential calculation circuit design, the proposed method reduced complexity and provided a tradeoff between effectiveness and complexity. reference [209] provides a hardware implementation of the GMM algorithm for real-time video segmentation compatible with OpenCV. The circuit reaches processing speeds of at least 45 fps in 1080p format on Virtex 6 and Stratix IV FPGAs by employing optimized equation formulation and ROM compression methods. The implemented circuit operates between 35.30 MHz and 98.12 MHz and utilizes less than 5% of FPGA resources while processing over 43 HD fps.

Similarly, Genovese and Napoli [210] created a hardware implementation of the GMM algorithm for real-time background detection in HD video sequences using OpenCV. Utilizing an innovative formulation of GMM equations, their circuit accomplished a processing speed of 91 HD frames per second on a Virtex 6 vlx75t FPGA while using only 3% of the FPGA's logic resources. The implementation exhibited increased speed and logic utilization compared to previous circuits, and it demonstrated the potential for low-power implementation through voltage scaling techniques. Experimental demonstrations on online video systems utilizing the Cyclone II EP2C35 FPGA validated the efficacy of the proposed background identification circuit. The GMM-based algorithm for gamma/neutron pulse shape discrimination (PSD) was successfully implemented on a quad-core Mac Pro workstation and a Zynq 7010 SoC in Ref. [211]. The performance of the algorithm on these platforms was compared to other standard techniques in PSD. Despite approximations and the use of FPGA parallelism, the implemented system did not experience a significant loss of accuracy. The design latency was 1.9 μs, which was faster than a CPU-based implementation. The system utilized 72% of DSP, 13% of FFs, 11% of BRAM, and 63% of LUTs. Recently, a GMM-MRCoHOG algorithm implementation utilizing an FPGA for image processing applications was proposed [212]. The computation of angles in 36 directions and a mixed two-dimensional Gaussian distribution slowed processing speeds and presented difficulties for hardware implementation. However, the authors effectively optimized the algorithm, resulting in a 0.122-ms processing time reduction compared to the conventional approach. Compared to the conventional software algorithm, the optimized FPGA implementation obtained a processing time reduction of 0.148 ms and In summaryedup.

To sum up, employing the GMM algorithm on an FPGA provides numerous benefits for a variety of applications. Implementations of GMM based on FPGA have demonstrated significant enhancements in computational speed, energy efficiency, and hardware resource utilization over software-based approaches. FPGA implementations have achieved considerable speedups and reduced processing times through optimizations such as pipeline-friendly designs, distributed arithmetic, and customized circuits, making them suitable for real-time and high-performance computing tasks. Overall, FPGA-based implementations of the GMM algorithm offer a promising solution for improving computational efficiency and enabling advanced applications in disciplines including image processing.

4.6. Genetic algorithm

A genetic algorithm (GA) is a search heuristic inspired by Charles Darwin's theory of natural evolution that belongs to the evolutionary algorithm (EA) class. The algorithm resembles the process of natural selection, in which the most fit organisms are chosen for reproduction to produce the next generation. GAs are used to address numerous optimization problems in science and engineering, employing a heuristic approach that relies primarily on random numbers to determine an approximation of the optimal solution. Countless medical specialties stand to benefit from the application of the GA; a review of them is discussed in Ref. [213]. This section reviews the previous FPGA-based GA and PGA (parallel genetic algorithm) systems. It then analyses their limitations: low-level programmability and lack of architectural flexibility.

Increasing demand for high-performance gas and high flexibility to adjust parameters caused researchers to investigate using hardware platforms such as microprocessors, FPGAs, and GPUs to accelerate the EAs. Microprocessors can provide a flexible platform for GA implementation. Goldberg introduced a Simple GA (SGA) system as a basic template for further researchers [214]. Multiple GA tools exist based on SGA, including GALOPPS, GALib [215], GA toolbox in Matlab, and Genetic Algorithm Utility Library (GAUL). On the other hand, FPGAs are one of the most promising platforms for enhancing the efficacy of GAs. Existing FPGA-based GA systems are efficient for a variety of applications [216–221]. In 1995, the first Genetic Algorithm (HGA) based on hardware was introduced. Scott et al. [220] designed the system based on a typical software GA using the VHDL programming language, and they separated the software GA's process into several hardware modules. The proposed HGA system reads the parameters, including population size and operator rates, from a shared memory module through a memory interface.

Furthermore, the procedure is controlled by the status data kept within the memory module. As these modules are not fully pipelined, they only read new inputs after the prior inputs' operations have been completed. Only selection modules are parallel. However, this system reduced the execution time by 96% compared to a software-based GA before the advent of enhanced HGA with parallelism and pipeline approach [219,222]. Hardware-based Parallel GA Systems: There are parallel GA systems besides a significant number of conventional FPGA-based GA systems. FPGA-based master-slave GAs and distributed genetic algorithms (DGAs) are introduced in Refs. [[[222–226]]], while FPGA-based compact genetic algorithms (CGAs) are proposed in Refs. [227,228]. There seem to be more Master-slave GAs [224] and DGAs as they are more accessible for FPGA place and route due to fewer connections between their GA kernels. The popular topology architectures in hardware-based parallel GA include chain, ring, grid, and island, as illustrated in Fig. 13 [222–228]. Also, Table 9 represents some examples of different topology architectures of the genetic algorithm. As depicted in Fig. 13, there is no connection between parallel GAs in the Island architecture, and GA instances connect to their neighbors in the Chain structure. Chain connection added to the head GA instance and connected to the tail GA instance forms Ring architecture. A GA instance combines to its four neighbors to create a Grid topology.

Ultimately, GA instances connect in a cube to form a Cube architecture. The specific choice of topology in FPGA-based hardware

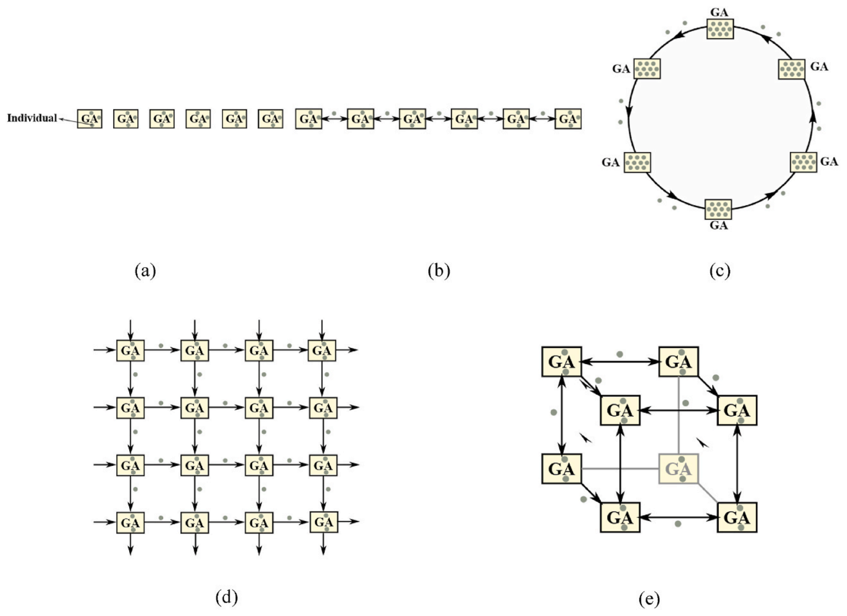


Fig. 13. Popular topology architectures in hardware-based parallel GA [222–228]: (a) Island, (b) Chain, (c) Ring, (d) Grid, and (e) Cube.

Table 9
A number of literature examples of various topology architectures of the genetic algorithm.

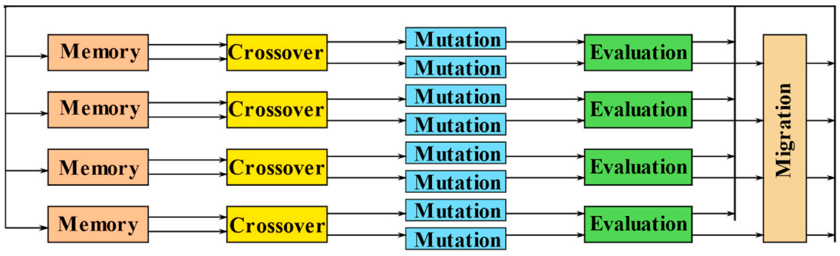
Reference	Type of GA system	GA Architecture
[222,223,226]	DGA	Ring
[225]	DGA	Chain
[224]	DGA	Grid
[228]	CGA	

acceleration depends on problem characteristics, communication requirements, and resource constraints. FPGA's parallel processing capabilities and capacity to manage large-scale data make it well-suited for implementing these similar GA topologies, enabling quicker and more efficient evolutionary optimization.

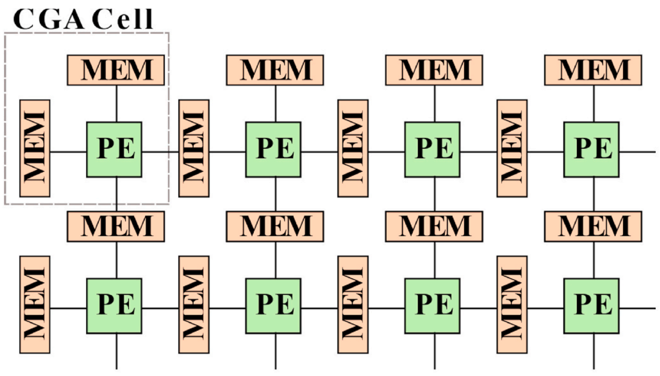
Below, we will summarize the current tasks completed on implementing GA on FPGAs.

Kamimura and Kanasugi [226] designed a DGA processor using VHDL. Fig. 14a depicts the system's architecture, which consists of a single migration unit and four GA instances. Their scheme is implemented as three numeric functions with binary chromosomes. However, their paper did not present performance evaluation results; it merely provided simulation error rate results. A CGA hardware system with a scalable framework is proposed in Ref. [228]. As illustrated in Fig. 14b, the GA kernels contact other kernels through the shared memory. Every PE supports a tiny population with only four individuals, while two of them are shared with other PEs. The system divides the large memory into small pieces, resulting in a speedup of 22 × over a CPU and 2200 × faster than a software NIOS processor in FPGAs. Due to the limited resources of a single FPGA, the scope and degree of parallelism of the problems solved are insufficient. Hence, researchers adapt Parallel GA (PGAs) to multiple FPGAs [229]. Work [229] achieved a two-time speedup using five chips compared to one chip, although the system did not present an uncomplicated way to modify migration policy.

The Random Number Generator (RNG) is vital for population generation and genetic operators for GA operations. FPGA-based methods utilize pre-generated random numbers from the CPU or hardware-based RNGs [220]. Other relevant sources include [[[230–235]]]. In Ref. [230], the hardware realization of FPGA-based GA is described, the outcomes of which can be used to implement evolvable hardware (EHW). reference [231] presents a parameterized GA IP core implemented at 92 MHz on FPGA. It is adaptable to various problem specifications, operates at a clock frequency five times higher than other GA implementations, and supports multiple customizable crossovers and mutation strategies. Using software simulations, Vizitiu et al. [232] developed a genetic approach implemented for optimal center selection and training of radial basis function (RBF) neural networks (GARBF system) as a pattern recognition system. Classification rates greater than 95% are guaranteed by the planned GARBF system. Yan-Cong et al. [234] proposed an FPGA-implemented Travel Salesman Problem (TSP) solution using GA. The findings from the experiments demonstrated that this method enhanced the running speed by two to three orders of magnitude and reduced resource consumption, allowing for the



(a)



-(b)

Fig. 14. (a) A DGA System adapted from Ref. [226], and (b) A CGA System adapted from Ref. [228].

algorithm's practical application. The techniques discussed, such as pipelining, parallel mechanisms, and storage space optimization, can potentially be applied to various domains, including biomedical applications, to improve running speed and resource utilization.

Similarly, the parallel implementation of GA on an FPGA demonstrated in Ref. [235] aims to reduce the system's processing time. This paper investigates the impacts of various population sizes on the processing time and area utilization of FPGAs. The onboard GA occupies less than 20% of the testing-utilized Virtex 7 logic cells. For the hardware implementation, studies were also conducted on the accuracy of the GA response for the optimization of two variable functions. Reference [236] proposes an FPGA-based ECG signal classification system using a parallel genetic algorithm and block-based neural network. The system offers online learning and portability for long-term patient monitoring. A similar genetic algorithm, EC-CGA, optimizes the BBNN and achieves faster processing than software-based approaches. Despite some limitations, the proposed method enables complete hardware implementation of ECG signal classification, particularly when combined with wavelet transform-based feature selection. This work shows promise for personalized ECG signal classification with FPGAs. Some GA implementations offer high flexibility in addition to their high efficacy [237].

Nonetheless, these existing FPGA-based systems have at least one limitation. Low-level programmability restricts a user who needs a thorough understanding of hardware architectures. In contrast, most systems are not general-purpose and only support one type of chromosome. Consequently, the absence of architectural flexibility renders various GA systems inappropriate for multiple applications. Changing GA parameters is a time-consuming procedure requiring recompilation and verification, which can take several hours. Moreover, the pipeline and parallelism techniques are only partially used in the systems, resulting in low performance. Many modules in the designs need to be fully pipelined or parallel.

5. Conclusion

This paper examined methods for designing FPGA-based accelerators for various machine learning (ML) algorithms. Several criteria are used to classify the works to emphasize their similarities and differences. We highlighted the primary research directions and summarized the key concepts of various works. This paper concentrated on a literature review concerning implementing ML algorithms on FPGAs for biomedical applications. Researchers have also investigated modern algorithms that efficiently satisfy the stringent requirements of real-time applications or applications that utilize large datasets in recent years. This article attempts to provide an exhaustive survey of hardware architectures used for ML algorithms on FPGA from 2000 to 2023, including classifiers and regression algorithms. This paper also compares existing works about the limitations of various implementations.

We can draw the following conclusions from all the papers examined in this review.

– Different linear regression-based approaches were discussed, highlighting their advantages and applications. Using VHDL and fixed-point arithmetic optimized the execution time and energy consumption. The suitability of FPGAs for computational challenges in microarray analysis and functional genomics was demonstrated, showing significant speed improvements compared to serial implementations.
– The accuracy and real-time performance of ridge regression-based optical flow algorithms implemented on FPGAs were enhanced. Logistic regression has been demonstrated to be effective in seizure detection and gene-gene interaction evaluation, with FPGA implementations offering low power consumption and fast computation. The combination of FPGA and GPU technologies significantly increased genome-wide logistic regression analyses' speed.
– Recent research has concentrated on FPGA-based implementations of KNN to improve performance and efficiency. To optimize execution time, these implementations have employed techniques such as HLS, parallelization, pipelining, and heterogeneous computing systems. Compared to GPPs, studies have revealed significantly higher acceleration ratios and energy efficiency. Memory constraints, design complexity, and algorithm scalability are still obstacles for precise and efficient FPGA-based KNN systems, necessitating additional research.
– In biomedical applications, FPGA-based decision tree and Random Forest algorithm implementations provide enhanced speed, throughput, and resource utilization. Numerous architectures and techniques have been proposed to optimize these implementations, including pipelining, high-level synthesis, and parallel processing. Compared to software-based methods, the reviewed studies demonstrate significant improvements in performance and efficiency. FPGA-based training of decision tree ensembles using parallel processing and evolutionary algorithms has also been investigated.
– FPGA-based implementations of SVMs for biomedical applications offer efficient solutions. Different architectures and techniques have been proposed, such as parallel pipelined structures, systolic arrays, multiplier-less approaches, DPR-based architectures, and cascaded classification systems. These FPGA implementations improve classification speed, resource utilization, power consumption, and accuracy compared to software-based approaches. The choice of architecture depends on specific application requirements. Overall, FPGA-based SVM implementations offer high-performance solutions for real-time and embedded biomedical applications.
– FPGA implementations offer parallel processing capabilities that enable faster and more efficient clustering computations, making them suitable for real-time and large-scale data processing. For Mean-Shift Clustering, architectures utilizing parallel processing, such as multiprocessor designs and similar pipelined structures, have demonstrated significant speedups compared to software implementations. Using kd-trees and pipelining techniques further improved the performance and throughput of the algorithms. Various FPGA architectures have been proposed, including parallel pipelined structures, multi-core designs, and parallel processing models. These implementations have shown significant speedups compared to software-based execution, demonstrating the potential of FPGAs for accelerating K-means clustering. Techniques such as fixed-point arithmetic, pipelining, and parallelization

have optimized the FPGA designs. The comparison of different implementations reveals that FPGA-based approaches consistently outperform software implementations in terms of processing speed. FPGA architectures exhibit scalability, allowing for the efficient utilization of available resources and accommodating a more significant number of processing elements. The FPGA implementations also often consume less power than alternative solutions like GPUs. On the other hand, optimization of hardware resources and memory usage while maintaining accuracy is essential. Recent works have explored using HLS and hardware/software co-design methodologies to facilitate the development and optimization of FPGA-based clustering algorithms.

– FPGA-based implementations of GMMs offer significant improvements in computational speed and energy efficiency compared to software-based approaches. Optimizations such as pipeline-friendly designs and customized circuits have resulted in substantial speedups and reduced processing times. FPGA-based GMM implementations have found applications in real-time and high-performance computing. Careful resource management and design considerations are necessary due to the complexity of GMM algorithms.
– Implementations of GAs based on FPGAs offer high performance and adaptability for optimization problems. While microprocessors provide flexibility, FPGAs have the potential to improve GA performance. Existing FPGA-based GA systems are restricted regarding programmability, architectural flexibility, and pipeline and parallelism techniques. To maximize the potential of FPGA-based GA systems, additional enhancements are required.
– While migrating designs to multi-FPGA systems, it is necessary to conduct additional research to comprehend the potential advancements that can be made and the constraints that may need to be addressed. Future work directions include investigating and developing design patterns for solving other problem sets with algorithmic similarities. This research indicates numerous architectures for distinct classifications of ML algorithms.

CRediT authorship contribution statement

Morteza Babaee Altman: Writing – original draft, Visualization, Resources, Methodology, Investigation, Conceptualization. **Wenbin Wan:** Supervision, Investigation. **Amineh Sadat Hosseini:** Writing – review & editing, Methodology, Investigation. **Saber Arabi Nowdeh:** Visualization, Resources, Methodology, Conceptualization. **Masoumeh Alizadeh:** Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. Hamet, J. Tremblay, Artificial intelligence in medicine, Metabolism 69 (2017) S36–S40.
[2] G. Cornet, Chapter 4. robot companions and ethics: a pragmatic approach of ethical design, Int. J. Bioeth. 24 (2013) 49–58.
[3] Gurusamy K.S., Samraj K., Davidson B.R., Robot assistant for laparoscopic cholecystectomy, Cochrane Database Syst. Rev. 1 (2009) Art. 1 – 2. No.: CD006578.
[4] M. Simonov, G. Delconte, Humanoid assessing rehabilitative exercises, Methods Inf. Med. 54 (2015) 114–121.
[5] R. Li, L. Li, Y. Xu, J. Yang, Machine learning meets omics: applications and perspectives, Briefings Bioinf. 11 (2021) bbab460.
[6] J. Latif, C. Xiao, A. Imran, S. Tu, Medical imaging using machine learning and deep learning algorithms: a review, in: 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), IEEE, 2019, pp. 1–5.
[7] A. Kan, Machine learning applications in cell image analysis, Immunol. Cell Biol. 95 (6) (2017) 525–530.
[8] S. Lu, S. Liu, P. Hou, B. Yang, M. Liu, L. Yin, W. Zheng, Soft tissue feature tracking based on DeepMatching network, CMES-Computer Modeling in Engineering & Sciences 136 (1) (2023).
[9] A.L. Beam, I.S. Kohane, Big data and machine learning in health care, JAMA 319 (13) (2018) 1317–1318.
[10] M.R. Azghadi, C. Lammie, J.K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, G. Indiveri, Hardware implementation of deep network accelerators towards healthcare and biomedical applications, IEEE Transactions on Biomedical Circuits and Systems 14 (6) (2020) 1138–1159.
[11] Y. Zhuang, N. Jiang, Y. Xu, Progressive distributed and parallel similarity retrieval of large CT image sequences in mobile telemedicine networks, Wireless Commun. Mobile Comput. 2022 (2022) 1–13.
[12] J.C. Porcello, Designing and implementing machine learning algorithms for advanced communications using fpgas, in: 2017 IEEE Aerospace Conference, IEEE, 2017, pp. 1–10.
[13] S. Afifi, H. GholamHosseini, R. Sinha, Fpga implementations of svm classifiers: a review, SN Computer Science 1 (2020) 133.
[14] P. Skoda, B.M. Rogina, V. Sruk, Fpga implementations of data mining algorithms, in: 2012 Proceedings of the 35th International Convention MIPRO, May 2012, pp. 362–367, 2012 Proceedings of the 35th International Convention MIPRO.
[15] J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, Architecture of field-programmable gate arrays, Proc. IEEE 81 (7) (1993) 1013–1029.
[16] Y. Tang, S. Liu, Y. Deng, Y. Zhang, L. Yin, W. Zheng, An improved method for soft tissue modeling, Biomed. Signal Process Control 65 (2021) 102367.
[17] J.M. Nadales, J.M. Manzano, A. Barriga, D. Limon, Efficient fpga parallelization of lipschitz interpolation for real-time decisionmaking, IEEE Trans. Control Syst. Technol. 30 (5) (2022) 2163–2175.
[18] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii, A. Acquaviva, A convolutional neural network fully implemented on fpga for embedded platforms, in: 2017 New Generation of CAS (NGCAS), 2017, pp. 49–52.
[19] D.R. Menaka, D.R. Janarthanan, D.K. Deeba, Fpga implementation of low power and high speed image edge detection algorithm, Microprocess. Microsyst. 75 (2020) 103053.
[20] M. Skuta, D. Macko, K. Jelemensk a, Automation of dynamic` power management in fpga-based energy-constrained systems, IEEE Access 8 (2020) 165894–165903.
[21] C. Pham-Quoc, X.-Q. Nguyen, T.N. Thinh, Towards an fpgatargeted hardware/software co-design framework for cnn-based edge computing, Mobile Network. Appl. 27 (5) (2022) 2024–2035.
[22] C. Pham-Quoc, X.-Q. Nguyen, T.N. Thinh, Hardware/software co-design for convolutional neural networks acceleration: a survey and open issues, in: Context-Aware Systems and Applications: 10th EAI International Conference, ICCASA 2021, Virtual Event, October 28– 29, 2021, Proceedings, vol. 10, Springer, 2021, pp. 164–178.
[23] G. De Michell, R.K. Gupta, Hardware/software co-design, Proc. IEEE 85 (3) (1997) 349–365.

[24] S.-A. Li, C.-C. Hsu, C.-C. Wong, C.-J. Yu, Hardware/software co-design for particle swarm optimization algorithm, Inf. Sci. 181 (20) (2011) 4582–4596.
[25] C. Johnston, K. Gribbon, D. Bailey, Implementing image processing algorithms on fpgas, in: Proceedings of the Eleventh Electronics New Zealand Conference, ENZCon'04, 2004, pp. 118–123.
[26] J.M. Cardoso, P.C. Diniz, Modeling loop unrolling: approaches and open issues, in: Proceedings, vol. 3, Springer, 2004, pp. 224–233. Computer Systems: Architectures, Modeling, and Simulation: Third and Fourth International Workshops, SAMOS 2004, Samos, Greece, July 21-23, 2004 and July 19-21, 2004.
[27] J. Drozd, O. Drozd, S. Antoshchuk, A. Kushnerov, V. Nikul, Effectiveness of matrix and pipeline fpga-based arithmetic components of safety-related systems, in: 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), vol. 2, 2015, pp. 785–789.
[28] X. Liang, J. Jean, K. Tomko, Data buffering and allocation in mapping generalized template matching on reconfigurable systems, J. Supercomput. 19 (2001) 77–91.
[29] K. Katahira, K. Sano, S. Yamamoto, Fpga-based lossless compressors of floating-point data streams to enhance memory bandwidth, in: ASAP 2010-21st IEEE International Conference on Applicationspecific Systems, Architectures and Processors, IEEE, 2010, pp. 246–253.
[30] K. Kara, J. Giceva, G. Alonso, Fpga-based data partitioning, in: Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 433–445.
[31] I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, R. Vemuri, An integrated partitioning and synthesis system for dynamically reconfigurable multi-fpga architectures, in: Parallel and Distributed Processing: 10 IPPS/SPDP'98 Workshops Held in Conjunction with the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing Orlando, vol. 12, Springer, Florida, USA, 1998, pp. 31–36. March 30–April 3, 1998 Proceedings.
[32] Z. Zhang, D. Guo, S. Zhou, J. Zhang, Y. Lin, Flight trajectory prediction enabled by time-frequency wavelet transform, Nat. Commun. 14 (1) (2023) 5258.
[33] M. Kirtas, A. Oikonomou, N. Passalis, G. Mourgias-Alexandris, M. Moralis-Pegios, N. Pleros, A. Tefas, Quantization-aware training for low precision photonic neural networks, Neural Network. 155 (2022) 561–573.
[34] C. Ding, S. Wang, N. Liu, K. Xu, Y. Wang, Y. Liang, Req-yolo: a resource-aware, efficient quantization framework for object detection on fpgas, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 33–42.
[35] S. Ishihara, M. Hariyama, M. Kameyama, A low-power fpga based on autonomous fine-grain power gating, IEEE Trans. Very Large Scale Integr. Syst. 19 (8) (2010) 1394–1406.
[36] T.J. Swamy, P. Kumar, Novel, clock gating broadcasting applications for low-power fpga architectures, in: 2023 International Conference on Computer Communication and Informatics (ICCCI, 2023, pp. 1–5.
[37] C.T. Chow, L.S.M. Tsui, P.H.W. Leong, W. Luk, S.J. Wilton, Dynamic voltage scaling for commercial fpgas, in: Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, IEEE, 2005, pp. 173–180, 2005.
[38] M. Alareqi, R. Elgouri, M. Tarhda, K. Mateur, A. Zemmouri, A. Mezouari, L. Hlou, Design and fpga implementation of realtime hardware co-simulation for image enhancement in biomedical applications,, in: 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), 2017, pp. 1–6.
[39] M.R. Azghadi, C. Lammie, J.K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, G. Indiveri, Hardware implementation of deep network accelerators towards healthcare and biomedical applications, IEEE Transactions on Biomedical Circuits and Systems 14 (6) (2020) 1138–1159.
[40] A. Kulkarni, A. Page, N. Attaran, A. Jafari, M. Malik, H. Homayoun, T. Mohsenin, An energy-efficient programmable manycore accelerator for personalized biomedical applications, IEEE Trans. Very Large Scale Integr. Syst. 26 (1) (2018) 96–109.
[41] F. Karatas¸, I. Koyuncu, M. Tuna, M. Alc¸ın, E. Avcioglu, A. Akgul, Design and implementation of arrhythmic ecg signals for biomedical engineering applications on fpga, Eur. Phys. J. Spec. Top. 231 (Jun 2022) 869–884.
[42] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis, K. Olukotun, Spatial: a language and compiler for application accelerators, SIGPLAN Not 53 (2018) 296–311.
[43] B. Moons, D. Bankman, M. Verhelst, Embedded Deep Learning: Algorithms, Architectures and Circuits for Always-On Neural Network Processing, Springer International Publishing, 2019.
[44] S. Lu, B. Yang, Y. Xiao, S. Liu, M. Liu, L. Yin, W. Zheng, Iterative reconstruction of low-dose CT based on differential sparse, Biomed. Signal Process Control 79 (2023) 104204.
[45] T. Van Court, M.C. Herbordt, R.J. Barton, Case study of a functional genomics application for an fpga-based coprocessor, Lect. Notes Comput. Sci. (2003) 365–374.
[46] P. Royer del Barrio, M.A. S′ anchez Marcos, M.′ L′ opez Vallejo, C.A. Lopez Barrio, Area-efficient linear regression architecture' for real-time signal processing on fpgas, in: Proceedings of 26th Conference on Design of Circuits and Integrated Systems, 2011.
[47] J. Grajal, O. Yeste-Ojeda, M.A. Sanchez, M. Garrido, M. LopezVallejo, Real time fpga implementation of an automatic modulation classifier for electronic warfare applications,, in: 2011 19th European Signal Processing Conference, 2011, pp. 1514–1518, 2011 19th European Signal Processing Conference.
[48] D. Yang, H. Li, G.D. Peterson, A. Fathy, Compressed sensing based uwb receiver: hardware compressing and fpga reconstruction, in: 2009 43rd Annual Conference on Information Sciences and Systems, 2009, pp. 198–201, 2009 43rd Annual Conference on Information Sciences and Systems.
[49] Y. Zhu, R. Huang, Z. Wu, S. Song, L. Cheng, R. Zhu, Deep learning-based predictive identification of neural stem cell differentiation, Nat. Commun. 12 (1) (2021) 2614.
[50] X. Yi, X. Guan, C. Chen, Y. Zhang, Z. Zhang, M. Li, C. Zee, Adrenal incidentaloma: machine learning-based quantitative texture analysis of unenhanced CT can effectively differentiate sPHEO from lipid-poor adrenal adenoma, J. Cancer 9 (19) (2018) 3577.
[51] S. Bellemare-Rousseau, G. Lachance, S.-D. Niyonambaza, E. Boisselier, M. Bouakadoum, A. Miled, Fpga-based prediction system for neurotransmitter concentration measurement from spectrophotometry data, in: 2020 18th IEEE International New Circuits and Systems Conference (NEWCAS), IEEE, 2020, pp. 267–270.
[52] R.K. Jain, Ridge regression and its application to medical data, Comput. Biomed. Res. 18 (1985) 363–368.
[53] Y. Zhuang, S. Chen, N. Jiang, H. Hu, An effective WSSENet-based similarity retrieval method of large Lung CT image databases, KSII Transactions on Internet & Information Systems 16 (7) (2022).
[54] K. Seyid, A. Richaud, R. Capoccia, Y. Leblebici, Fpga-based hardware implementation of real-time optical flow calculation, IEEE Trans. Circ. Syst. Video Technol. 28 (2018) 206–216.
[55] G. Lu, X. Zhang, W. Ouyang, L. Chen, Z. Gao, D. Xu, An Endto-End Learning Framework for Video Compression, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, p. 1, 1.
[56] Z. Wei, D. Lee, B.E. Nelson, J.K. Archibald, Hardware-friendly vision algorithms for embedded obstacle detection applications, IEEE Trans. Circ. Syst. Video Technol. 20 (2010) 1577–1589.
[57] T. Simons, D.J. Lee, A new hardware architecture for the ridge regression optical flow algorithm, in: 2018 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI), 2018, pp. 125–128.
[58] S. Jin, D. Kim, D.D. Nguyen, J.W. Jeon, Pipelined hardware architecture for high-speed optical flow estimation using fpga, in: 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010, pp. 33–36, 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines.
[59] A. Kulkarni, A. Jafari, C. Sagedy, T. Mohsenin, Sketchingbased high-performance biomedical big data processing accelerator, in: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 1138–1141.
[60] A. Page, S.P.T. Oates, T. Mohsenin, An ultra low power feature extraction and classification system for wearable seizure detection, in: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015, pp. 7111–7114.

[61] A. Page, C. Sagedy, E. Smith, N. Attaran, T. Oates, T. Mohsenin, A flexible multichannel eeg feature extractor and classifier for seizure detection, IEEE Transactions on Circuits and Systems II: Express Briefs 62 (2015) 109–113.

[62] A.-A.-A. S. C. (TASC), W. T. C. C. C. . (WTCCC2), D.M. Evans, C.C. Spencer, J.J. Pointon, Z. Su, D. Harvey, G. Kochan, U. Oppermann, A. Dilthey, et al., Interaction between erap1 and hla-b27 in ankylosing spondylitis implicates peptide handling in the mechanism for hla-b27 in disease susceptibility, Nat. Genet. 43 (8) (2011) 761–767.

[63] L. Wienbrandt, J.C. Kassens, M. Hübenthal, D. Ellinghaus, 1000× faster than plink: combined fpga and gpu accelerators for logistic regression-based detection of epistasis, Journal of Computational Science 30 (2019) 183–193.

[64] L. Wienbrandt, J.C. Kassens, M. Hubenthal, D. Ellinghaus, Fast genome-wide third-order snp interaction tests with information gain on a low-cost heterogeneous parallel fpga-gpu computing architecture, in: International Conference on Computational Science, ICCS 2017, 12-14 June 2017 vol. 108, Procedia Computer Science, Zurich, Switzerland, 2017, pp. 596–605.

[65] I. Erbas¸, D.A. Vargas, B. Guc¸LÜ, Fpga implementation of multinomial logistic regression for vibrotactile feedback in a robotic hand, in: 2020 International Conference on E-Health and Bioengineering (EHB), IEEE, 2020, pp. 1–4.

[66] J.M. Keller, M.R. Gray, J.A. Givens, A fuzzy k-nearest neighbor algorithm, IEEE Transactions on Systems, Man, and Cybernetics 15 (1985) 580–585. SMC-.

[67] P. Soucy, G.W. Mineau, A simple knn algorithm for text categorization, in: Proceedings 2001 IEEE International Conference on Data Mining, IEEE International Conference on Data Mining, 2001, pp. 647–648. Proceedings 2001.

[68] D.A. Adeniyi, Z. Wei, Y. Yongquan, Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method, Appl. Comput. Inform. 12 (2016) 90–108.

[69] G. Amato, F. Falchi, Knn based image classification relying on local feature similarity, in: Proceedings of the Third International Conference on SImilarity Search and APplications, SISAP '10, 2010, pp. 101–108. Proceedings of the Third International Conference on SImilarity Search and APplications, Association for Computing Machinery.

[70] R.M. Parry, W. Jones, T.H. Stokes, J.H. Phan, R.A. Moffitt, H. Fang, L. Shi, A. Oberthuer, M. Fischer, W. Tong, M.D. Wang, Knearest neighbor models for microarray gene expression analysis and clinical outcome prediction, Pharmacogenomics J. 10 (2010) 292–309.

[71] B.V. Ramana, M.S.P. Babu, N.B. Venkateswarlu, et al., A critical study of selected classification algorithms for liver disease diagnosis, Int. J. Database Manag. Syst. 3 (2011) 101–114.

[72] J. Sim, S.-Y. Kim, J. Lee, Prediction of protein solvent accessibility using fuzzy k-nearest neighbor method, Bioinformatics 21 (Apr. 2005) 2844–2849.

[73] L. Li, T.A. Darden, C.R. Weingberg, A.J. Levine, L.G. Pedersen, Gene assessment and sample classification for gene expression data using a genetic algorithm/k-nearest neighbor method, Comb. Chem. High Throughput Screen. 4 (2001) 727–739.

[74] A. Goshvarpour, Radial basis function and k-nearest neighbor classifiers for studying heart rate signals during meditation, Int. J. Mod. Educ. Comput. Sci. 4 (2012).

[75] A. Ahirwar, Study of Techniques Used for Medical Image Segmentation and Computation of Statistical Test for Region Classification of Brain Mri, vol. 5, IJ Information Technology and Computer Science, 2013, pp. 44–53.

[76] H. Hussain, K. Benkrid, C. Hong, H. Seker, "An adaptive fpga implementation of multi-core k-nearest neighbour ensemble classifier using dynamic partial reconfiguration," in 22nd International Conference on Field Programmable Logic and Applications (FPL), in: 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012, pp. 627–630.

[77] H.M. Hussain, K. Benkrid, H. Seker, An adaptive implementation of a dynamically reconfigurable k-nearest neighbour classifier on fpga, in: 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2012, pp. 205–212.

[78] Y.-J. Yeh, H.-Y. Li, W.-J. Hwang, C.-Y. Fang, Fpga implementation of knn classifier based on wavelet transform and partial distance search, in: B.K. Ersbøll, K. S. Pedersen (Eds.), Image Analysis, 512–521, Image Analysis, Springer Berlin Heidelberg, 2007.

[79] Z.-H. Li, J.-F. Jin, X.-G. Zhou, Z.-H. Feng, K-nearest neighbor algorithm implementation on fpga using high level synthesis, in: 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), IEEE, 2016, pp. 600–602.

[80] Y. Pu, J. Peng, L. Huang, J. Chen, An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl, in: 2015 IEEE 23rd Annual International Symposium on FieldProgrammable Custom Computing Machines, 2015, pp. 167–170, 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines.

[81] E.S. Manolakos, I. Stamoulias, Ip-cores design for the knn classifier, in: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 4133–4136.

[82] I. Stamoulias, E.S. Manolakos, Parallel architectures for the knn classifier – design of soft ip cores and fpga implementations, ACM Trans. Embed. Comput. Syst. 13 (2013).

[83] M.A. Mohsin, D.G. Perera, An fpga-based hardware accelerator for k-nearest neighbor classification for machine learning on mobile devices, in: Proceedings of the 9th International Symposium on HighlyEfficient Accelerators and Reconfigurable Technologies, HEART 2018, Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, Association for Computing Machinery, 2018.

[84] D.G. Perera, K.F. Li, FPGA-based reconfigurable hardware for compute intensive data mining applications, in: 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, IEEE, 2011, October, pp. 100–108.

[85] J. Shi, Z. Li, J. Jia, Z. Li, C. Shen, J. Zhang, N. Chi, Waveform-to-Waveform end-to-end learning framework in a seamless Fiber-Terahertz integrated communication system, J. Lightwave Technol. 41 (8) (2023) 2381–2392.

[86] J. Zhang, Q. Shen, Y. Ma, L. Liu, W. Jia, L. Chen, J. Xie, Calcium Homeostasis in Parkinson's disease: from Pathology to treatment, Neurosci. Bull. 38 (10) (2022) 1267–1270.

[87] J. Li, J. Li, C. Wang, F.J. Verbeek, T. Schultz, H. Liu, Outlier detection using iterative adaptive mini-minimum spanning tree generation with applications on medical data, Front. Physiol. 14 (2023) 1233341.

[88] D.J. Brennan, S.L. O'Brien, A. Fagan, A.C. Culhane, D.G. Higgins, M.J. Duffy, W.M. Gallagher, Application of dna microarray technology in determining breast cancer prognosis and therapeutic response, null 5 (Aug. 2005) 1069–1083.

[89] H.M. Hussain, K. Benkrid, H. Seker, Dynamic partial reconfiguration implementation of the svm/knn multi-classifier on fpga for bioinformatics application, in: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015, pp. 7667–7670, 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).

[90] C. Feng, H. Zhao, Q. Liu, H. Hong, C. Gu, X. Zhu, Implementation of radar-based breathing disorder recognition using fpga, in: 2019 IEEE MTT-S International Microwave Biomedical Conference (IMBioC), vol. 1, 2019, pp. 1–3.

[91] B. Venkataramanaiah, J. Kamala, Ecg signal processing and knn classifier-based abnormality detection by vh-doctor for remote cardiac healthcare monitoring, Soft Comput. 24 (22) (2020) 17457–17466.

[92] J.R. Struharik, Implementing decision trees in hardware, in: 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics, 2011, pp. 41–46, 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics.

[93] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, J. Zambreno, An fpga implementation of decision tree classification, in: 2007 Design, Automation & Test in Europe Conference & Exhibition, 2007, pp. 1–6.

[94] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, P.S. Marios, Pipelined decision tree classification accelerator implementation in fpga (dtcaif), IEEE Trans. Comput. 64 (2013) 280–285.

[95] R. Kulaga, M. Gorgon, Fpga implementation of decision trees and tree ensembles for character recognition in vivado hls, Image Processing & Communications 19 (2014) 71.

[96] M. Barbareschi, E. Battista, N. Mazzocca, S. Venkatesan, A hardware accelerator for data classification within the sensing infrastructure, in: Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), P. 400–405, Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), 2014.

[97] M. Barbareschi, S.D. Prete, F. Gargiulo, A. Mazzeo, C. Sansone, Decision tree-based multiple classifier systems: an fpga perspective, in: F. Schwenker, F. Roli, J. Kittler (Eds.), Multiple Classifier Systems, 194–205, Multiple Classifier Systems, Springer International Publishing, 2015.
[98] D. Tong, L. Sun, K. Matam, V. Prasanna, High throughput and programmable online trafficclassifier on fpga, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, P. 255–264, Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Association for Computing Machinery, 2013.
[99] M. Barbareschi, Implementing hardware decision tree prediction: a scalable approach, in: 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2016, pp. 87–92, 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA).
[100] F. Amato, M. Barbareschi, V. Casola, A. Mazzeo, An fpgabased smart classifier for decision support systems, in: F. Zavoral, J.J. Jung, C. Badica (Eds.), Intelligent Distributed Computing VII, Intelligent Distributed Computing VII, Springer International Publishing, 2014, pp. 289–299.
[101] R. Choudhury, S.R. Ahamed, P. Guha, Fpga implementation of low complexity hybrid decision tree training accelerator, in: 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2021, pp. 511–514.
[102] W. Song, Q. Han, Z. Lin, N. Yan, D. Luo, Y. Liao, M. Zhang, Z. Wang, X. Xie, A. Wang, et al., Design of a flexible wearable smart semg recorder integrated gradient boosting decision tree based hand gesture recognition, IEEE transactions on biomedical circuits and systems 13 (6) (2019) 1563–1574.
[103] M. Shruthi, J.A. Prathap, R. Manaswini, et al., Fpga-based automatic pill dispenser using decision tree classifier, Journal of Population Therapeutics and Clinical Pharmacology 30 (11) (2023) 143–152.
[104] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, A. Dollas, Hc-cart: a parallel system implementation of data mining classification and regression tree (cart) algorithm on a multi-fpga system, ACM Trans. Archit. Code Optim. 9 (Jan) (2013).
[105] Struharik, R.J. and Novak, L.A., Evolving decision trees in hardware, J CIRCUIT SYST COMP 18 (Oct. 2009) 1033–1060.
[106] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.
[107] D. Levi, Hereboy: a fast evolutionary algorithm, in: Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware, Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware, IEEE, 2000, pp. 17–24.
[108] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92, P. 144–152, Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Association for Computing Machinery, 1992.
[109] B. Scholkopf, K. Tsuda, J.P. Vert, Support Vector Machine Applications in Computational Biology, 2004.
[110] D.B.A.M. Khosrow-Pour, M.P. Vestias, ′ High-Performance Reconfigurable Computing Granularity, IGI Global, 2015.
[111] Y. Ago, K. Nakano, Y. Ito, A classification processor for a support vector machine with embedded dsp slices and block rams in the fpga, in: 2013 IEEE 7th International Symposium on Embedded Multicore Socs, 2013, pp. 91–96, 2013 IEEE 7th International Symposium on Embedded Multicore Socs.
[112] M. Berberich, K. Doll, in: Highly Flexible Fpga-Architecture of a Support Vector Machine, 2014, pp. 25–32.
[113] R. Andraka, A survey of cordic algorithms for fpga based computers, in: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, Association for Computing Machinery, 1998, pp. 191–200. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays.
[114] V.S. Vranjkovic, R.J.R. Struharik, L.A. Novak, Reconfigurable' hardware for machine learning applications, J CIRCUIT SYST COMP 24 (Feb. 2015) 1550064.
[115] S. Saurav, S. Singh, R. Saini, A.K. Saini, Hardware accelerator for facial expression classification using linear svm, in: S.M. Thampi, S. Bandyopadhyay, S. Krishnan, K.-C. Li, S. Mosin, M. Ma (Eds.), Advances in Signal Processing and Intelligent Recognition Systems, Advances in Signal Processing and Intelligent Recognition Systems, Springer International Publishing, 2016, pp. 39–50.
[116] S.-J. Kim, S.-Y. Lee, K.-S. Cho, Design of high-performance unified circuit for linear and non-linear svm classifications, JSTS: Journal of Semiconductor Technology and Science 12 (2012) 162–167.
[117] T. Koide, A.T. Hoang, T. Okamoto, S. Shigemi, T. Mishima, T. Tamaki, B. Raytchev, K. Kaneda, Y. Kominami, R. Miyaki, T. Matsuo, S. Yoshida, S. Tanaka, Fpga implementation of type identifier for colorectal endoscopie images with nbi magnification, in: 2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2014, pp. 651–654, 2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS).
[118] S. Shigemi, T. Mishima, A.-T. Hoang, T. Koide, T. Tamaki, B. Raytchev, K. Kaneda, Y. Kominami, R. Miyaki, T. Matsuo, in: Customizable Hardware Architecture of Support Vector Machine in Cad System for Colorectal Endoscopic Images with Nbi Magnification, 2013, pp. 298–303.
[119] S. Shigemi, An fpga implementation of support vector machine identifier for colorectal endoscopic images with nbi magnification, in: Proc. Of the 28th International Conference on Circuits/Systems, Computers and Communications (ITC-Cscc2013), 2013, pp. 571–572. Proc. of the 28th International Conference on Circuits/Systems, Computers and Communications (ITC-CSCC2013).
[120] Z. Nie, X. Zhang, Z. Yang, An Fpga Implementation of Multi-Class Support Vector Machine Classifier Based on Posterior Probability, 2012.
[121] M. Pietron, M. Wielgosz, D. Zurek, E. Jamro, K. Wiatr, Comparison of gpu and fpga implementation of svm algorithm for fast image segmentation, in: H. Kubatov′ a, C. Hochberger, M. Dan′ ek, B. Sick (Eds.), Architecture of Computing Systems – ARCS 2013, Springer Berlin Heidelberg, 2013. ˘ pp. 292–302, Architecture of Computing Systems – ARCS 2013.
[122] M. Wielgosz, E. Jamro, D. Zurek, K. Wiatr, ˙ FPGA Implementation of the Selected Parts of the Fast Image Segmentation, Springer Berlin Heidelberg, 2012.
[123] C. Kyrkou, T. Theocharides, Scope: towards a systolic array for svm object detection, IEEE Embedded Systems Letters 1 (2009) 46–49.
[124] C. Kyrkou, T. Theocharides, A parallel hardware architecture for real-time object detection with support vector machines, IEEE Trans. Comput. 61 (2012) 831–842.
[125] D. Anguita, S. Pischiutta, S. Ridella, D. Sterpi, Feed-forward support vector machine without multipliers, IEEE Trans. Neural Network. 17 (2006) 1328–1331.
[126] M. Ruiz-Llata, G. Guarnizo, M. Yebenes-Calvino, Fpga im-′ plementation of a support vector machine for classification and regression, in: The 2010 International Joint Conference on Neural Networks (IJCNN), 2010, pp. 1–5.
[127] V. Vranjkovic, R. Struharik, New architecture for svm clas-′ sifier and its application to telecommunication problems, in: 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers, 2011, pp. 1543–1545, 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers.
[128] J.G. Sarciada, H.L. Rivera, M. Jimenez, Cordic Algorithms For' Svm Fpga Implementation, vol. 7703, Apr. 2010.
[129] H. Lamela, J. Gimeno, M. Jimenez, M. Ruiz, Performance' Evaluation of a Fpga Implementation of a Digital Rotation Support Vector Machine, vol. 6979, Apr, 2008.
[130] A.M. Jallad, L.B. Mohammed, "Hardware support vector machine (svm) for satellite on-board applications," in 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), in: 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2014, pp. 256–261.
[131] X. Pan, H. Yang, L. Li, Z. Liu, L. Hou, Fpga Implementation of Svm Decision Function Based on Hardware-Friendly Kernel, 2013, pp. 133–136, 2013 International Conference on Computational and Information Sciences.
[132] B. Mandal, M.P. Sarma, K.K. Sarma, N. Mastorakis, Implementation of systolic array based svm classifier using multiplierless kernel, in: 2014 International Conference on Signal Processing and Integrated Networks (SPIN), 2014, pp. 35–39.
[133] L. Pezzarossa, A.T. Kristensen, M. Schoeberl, J. Sparsø, Using dynamic partial reconfiguration of fpgas in real-time systems, Microprocess. Microsyst. 61 (2018) 198–206.
[134] T. N. Sasamal and R. Prasad, "Module Based and Difference Based Implementation of Partial Reconfiguration on Fpga: A Review,".
[135] H.M. Hussain, K. Benkrid, H. Seker, Reconfiguration-based implementation of svm classifier on fpga for classifying microarray data, in: 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2013, pp. 3058–3061, 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).
[136] R.A. Patil, G. Gupta, V. Sahula, A.S. Mandal, Power aware hardware prototyping of multiclass svm classifier through reconfiguration, in: 2012 25th International Conference on VLSI Design, 2012, pp. 62–67, 2012 25th International Conference on VLSI Design.
[137] H. Hussain, K. Benkrid, H. S¸Eker, Novel dynamic partial reconfiguration implementations of the support vector machine classifier on fpga, Turk. J. Electr. Eng. Comput. Sci. 24 (5) (2016) 3371–3387.

[138] C. Kyrkou, T. Theocharides, C. Bouganis, An embedded hardware-efficient architecture for real-time cascade support vector machine classification, in: 2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), P. 129–136, 2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2013.

[139] M. Papadonikolakis, C. Bouganis, A novel fpga-based svm classifier, in: 2010 International Conference on Field-Programmable Technology, 2010, pp. 283–286, 2010 International Conference on FieldProgrammable Technology.

[140] M. Papadonikolakis, C. Bouganis, Novel cascade fpga accelerator for support vector machines classification, IEEE Transact. Neural Networks Learn. Syst. 23 (2012) 1040–1052.

[141] C. Kyrkou, C. Bouganis, T. Theocharides, M.M. Polycarpou, Embedded hardware-efficient real-time classification with cascade support vector machines, IEEE Transact. Neural Networks Learn. Syst. 27 (2016) 99–112.

[142] C. Kyrkou, T. Theocharides, C.-S. Bouganis, M. Polycarpou, Boosting the hardware-efficiency of cascade support vector machines for embedded classification applications, Int. J. Parallel Program. 46 (2018) 1220–1246.

[143] S. Afifi, H. GholamHosseini, R. Sinha, Svm classifier on chip for melanoma detection, in: 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2017, pp. 270–274, 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).

[144] D. Mahmoodi, A. Soleimani, H. Khosravi, M. Taghizadeh, Fpga simulation of linear and nonlinear support vector machine, J. Software Eng. Appl. (2011) 9, vol. Vol.04No.05.

[145] R. Saini, S. Saurav, D.C. Gupta, N. Sheoran, Hardware implementation of svm using system generator, in: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), P. 2129–2132, 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2017.

[146] M. Cutajar, E. Gatt, I. Grech, O. Casha, J. Micallef, Hardwarebased support vector machine for phoneme classification, in: Eurocon 2013, Eurocon, 2013, pp. 1701–1708, 2013.

[147] S. Afifi, H. GholamHosseini, R. Sinha, A low-cost fpga-based svm classifier for melanoma detection, in: 2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES), 2016, pp. 631–636, 2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES).

[148] R. Akeela, B. Dezfouli, Software-defined radios: architecture, state-of-the-art, and challenges, Comput. Commun. 128 (2018) 106–125.

[149] V. Tsoutsouras, K. Koliogeorgi, S. Xydis, D. Soudris, An exploration framework for efficient high-level synthesis of support vector machines: case study on ecg arrhythmia detection for xilinx zynq soc, Journal of Signal Processing Systems 88 (2017) 127–147.

[150] S. Afifi, H. GholamHosseini, R. Sinha, Hardware acceleration of svm-based classifier for melanoma images, in: F. Huang, A. Sugimoto (Eds.), Image and Video Technology - PSIVT 2015 Workshops, Image and Video Technology - PSIVT 2015 Workshops, Springer International Publishing, 2016, pp. 235–245.

[151] S. Afifi, H. GholamHosseini, R. Sinha, A system on chip for melanoma detection using fpga-based svm classifier, Microprocess. Microsyst. 65 (2019) 57–68.

[152] S. Afifi, H. Gholamhosseini, R. Sinha, M. Linden, in: A Novel Medical' Device for Early Detection of Melanoma, 2019, pp. 122–127.

[153] S. Afifi, H. GholamHosseini, R. Sinha, Dynamic hardware system for cascade svm classification of melanoma, Neural Comput. Appl. 32 (2020) 1777–1788.

[154] K. Koliogeorgi, G. Zervakis, D. Anagnostos, N. Zompakis, K. Siozios, Optimizing svm classifier through approximate and high level synthesis techniques, in: 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2019, pp. 1–4, 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST).

[155] O. Elgawi, A.M. Mutawa, A. Ahmad, Energy-efficient embedded inference of svms on fpga, in: 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), P. 164–168, 2019, 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).

[156] C. Kyrkou, T. Theocharides, C.S. Bouganis, A hardware-efficient architecture for embedded real-time cascaded support vector machines classification, in: Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI, GLSVLSI '13, P. 341–342, Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI, Association for Computing Machinery, 2013.

[157] Y. Jiang, K. Virupakshappa, E. Oruklu, Fpga implementation of a support vector machine classifier for ultrasonic flaw detection, in: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017, pp. 180–183.

[158] R. Xu, D.C. Wunsch, Clustering algorithms in biomedical research: a review, IEEE Reviews in Biomedical Engineering 3 (2010) 120–154.

[159] A. Annovi, M. Berretta, F. Crescioli, M. Dell'Orso, P. Giannetti, P. Laurelli, G. Maccarrone, A. Sansoni, L. Sartori, G. Volpi, A fast fpga-based clustering algorithm for real time image processing, in: 2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC), P. 4138–4141, 2009, 2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC).

[160] A. Annovi, M. Beretta, A fast general-purpose clustering algorithm based on fpgas for high-throughput data processing, in: 11th Pisa Meeting on Advanced Detectors vol. 617, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2010, pp. 254–257.

[161] C. Sotiropoulou, A. Annovi, M. Beretta, P. Luciano, S. Nikolaidis, G. Volpi, A multi-core fpga-based clustering algorithm for realtime image processing, in: 2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC), 2013, pp. 1–5, 2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC).

[162] H. Hou, W. Sun, L. Wang, Y. Qin, B. You, Clustering algorithm analysis and fpga implementation of neural electrical signal, in: 2017 Chinese Automation Congress (CAC), 2017 Chinese Automation Congress (CAC), 2017, pp. 5433–5436.

[163] S. Bailie, M. Leeser, Incremental clustering applied to radar deinterleaving: a parameterized fpga implementation, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '12, 2012, pp. 25–28. Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Association for Computing Machinery.

[164] K. Fukunaga, L. Hostetler, The estimation of the gradient of a density function, with applications in pattern recognition, IEEE Trans. Inf. Theor. 21 (1975) 32–40.

[165] A. Tehreem, S.G. Khawaja, A.M. Khan, M.U. Akram, S.A. Khan, Multiprocessor architecture for real-time applications using mean shift clustering, Journal of Real-Time Image Processing 16 (2019) 2233–2246.

[166] S. Craciun, G. Wang, A.D. George, H. Lam, J.C. Principe, A scalable rc architecture for mean-shift clustering, in: 2013 IEEE 24th International Conference on Application-specific Systems, Architectures and Processors, 2013, pp. 370–374, 2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors.

[167] D.B.K. Trieu, T. Maruyama, An implementation of the mean shift filter on fpga, in: 2011 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 219–224, 2011 21st International Conference on Field Programmable Logic and Applications.

[168] D.B.K. Trieu, T. Maruyama, Real-time color image segmentation based on mean shift algorithm using an fpga, Journal of Real-Time Image Processing 10 (2015) 345–356.

[169] S. Craciun, R. Kirchgessner, A.D. George, H. Lam, J.C. Principe, A real-time, power-efficient architecture for mean-shift image segmentation, Journal of Real-Time Image Processing 14 (2018) 379–394.

[170] H. Zhou, R. Machupalli, M. Mandal, Efficient Fpga Implementation of Automatic Nuclei Detection in Histopathology Images, 2019.

[171] B.V. Kiran, K.H.S. Kumar, Hardware efficient mean shift clustering algorithm implementation on fpga, International Journal of Application or Innovation in Engineering & Management (IJAIEM) 3 (2014) 460–464.

[172] A. Tehreem, S.G. Khawaja, M.U. Akram, S.A. Khan, A novel mean-shift architecture for scalable multiprocessor implementation, in: 2016 Future Technologies Conference (FTC), 2016, pp. 1107–1111, 2016 Future Technologies Conference (FTC).

[173] M. Leeser, J. Theiler, M. Estlick, J.J. Szymanski, Design tradeoffs in a hardware implementation of the k-means clustering algorithm, in: Proceedings of the 2000 IEEE Sensor Array and Multichannel Signal Processing Workshop. SAM 2000 (Cat. No.00EX410), 2000, pp. 520–524. Proceedings of the 2000 IEEE Sensor Array and Multichannel Signal Processing Workshop. SAM 2000 (Cat. No.00EX410).

[174] M. Estlick, M. Leeser, J. Theiler, J.J. Szymanski, Algorithmic transformations in the implementation of k- means clustering on reconfigurable hardware, in: Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, FPGA '01, 2001, pp. 103–110. Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, Association for Computing Machinery.

[175] M. Gokhale, J. Frigo, K. Mccabe, J. Theiler, C. Wolinski, D. Lavenier, Experience with a hybrid processor: K-means clustering, J. Supercomput. 26 (2003) 131–148.

[176] T. Saegusa, T. Maruyama, An fpga implementation of k-means clustering for color images based on kd-tree, in: 2006 International Conference on Field Programmable Logic and Applications, 2006, pp. 1–6, 2006 International Conference on Field Programmable Logic and Applications.

[177] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Trans. Pattern Anal. Mach. Intell. 24 (2002) 881–892.

[178] G.A. Covington, C.L.G. Comstock, A.A. Levine, J.W. Lockwood, Y.H. Cho, High Speed Document Clustering in Reconfigurable Hardware, IEEE, 2006, pp. 1–7.

[179] X. Wang, M. Leeser, K-means clustering for multispectral images using floating-point divide, in: 15th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines (FCCM 2007), Apr. 2007, pp. 151–162, 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007).

[180] H.M. Hussain, K. Benkrid, H. Seker, A.T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: an accelerated approach to clustering microarray data," in 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), in: 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), June 2011, pp. 248–255.

[181] H.M. Hussain, K. Benkrid, A.T. Erdogan, H. Seker, Highly parameterized k-means clustering on fpgas: Comparative results with gpps and gpus, in: 2011 International Conference on Reconfigurable Computing and FPGAs, Nov. 2011, pp. 475–480, 2011 International Conference on Reconfigurable Computing and FPGAs.

[182] K. Nagarajan, B. Holland, A.D. George, K.C. Slatton, H. Lam, Accelerating machine-learning algorithms on fpgas using patternbased decomposition, Journal of Signal Processing Systems 62 (2011) 43–63.

[183] J. Singaraju, J.A. Chandy, Active storage networks for accelerating k-means data clustering, in: A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, T. El-Ghazawi (Eds.), Reconfigurable Computing: Architectures, Tools and Applications, 102–109, Reconfigurable Computing: Architectures, Tools and Applications, Springer Berlin Heidelberg, 2011.

[184] F. An, T. Koide, H.J. Mattausch, A k-means-based multi-prototype high-speed learning system with fpgaimplemented coprocessor for 1-nn searching, IEICE Transactions on Information and Systems E95.D (2012) 2327–2338.

[185] J.S.S. Kutty, F. Boussaid, A. Amira, A high speed configurable fpga architecture for k-mean clustering, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS), 2013 IEEE International Symposium on Circuits and Systems (ISCAS), May 2013, pp. 1801–1804.

[186] F. Winterstein, S. Bayliss, G.A. Constantinides, Fpga-based kmeans clustering using tree-based data structures, in: 2013 23rd International Conference on Field Programmable Logic and Applications, 2013, pp. 1–6, 2013 23rd International Conference on Field programmable Logic and Applications.

[187] F. An, H.J. Mattausch, K-means clustering algorithm for multimedia applications with flexible hw/sw co-design, J. Syst. Architect. 59 (2013) 155–164.

[188] F. Jia, C. Wang, X. Li, X. Zhou, Sakma: specialized fpgabased accelerator architecture for data-intensive k-means algorithms, in: G. Wang, A. Zomaya, G. Martinez, K. Li (Eds.), Algorithms and Architectures for Parallel Processing, 106–119, Algorithms and Architectures for Parallel Processing, Springer International Publishing, 2015.

[189] T.S. Abdelrahman, Accelerating k-means clustering on a tightlycoupled processor-fpga heterogeneous system, in: 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), July 2016, pp. 176–181, 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP).

[190] Q.Y. Tang, M.A.S. Khalid, Acceleration of k-means algorithm using altera sdk for opencl, ACM Trans. Reconfigurable Technol. Syst. (TRETS) 10 (Sept) (2016).

[191] J. Canilho, M. Vestias, H. Neto, Multi-core for k-means clus-′ tering on fpga, in: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Aug. 2016, pp. 1–4, 2016 26th International Conference on Field Programmable Logic and Applications (FPL).

[192] R. Raghavan, D.G. Perera, A fast and scalable fpga-based parallel processing architecture for k-means clustering for big data analysis, in: 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Pp. 1–8, 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Aug. 2017.

[193] A. Americai, Design trade-offs in configurable fpga architectures for k-means clustering, Stud. Inf. Control 26 (2017) 43–48.

[194] C. Chung, Y. Wang, Hadoop cluster with fpga-based hardware accelerators for k-means clustering algorithm, in: 2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCETW), June 2017, pp. 143–144, 2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW).

[195] F.W. Wibowo, M. Sulistiyono, et al., Hardware platform design analysis of k-means clustering algorithm implementation, Int. J. Eng. Technol. 7 (2018) 90–93.

[196] H.M. Kamali, Using Multi-Core Hw/sw Co-design Architecture for Accelerating K-Means Clustering Algorithm, 2018 arXiv preprint arXiv:1807.09250.

[197] L.A. Dias, J.C. Ferreira, M.A.C. Fernandes, Parallel implementation of k-means algorithm on fpga, IEEE Access 8 (2020) 41071–41084.

[198] Y. Wang, Z. Zeng, B. Feng, L. Deng, Y. Ding, Kpynq: a workefficient triangle-inequality based k-means on fpga, in: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Apr. 2019, p. 320, 320, 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).

[199] F. Siddiqui, S. Amiri, U.I. Minhas, T. Deng, R. Woods, K. Rafferty, D. Crookes, Fpga-based processor acceleration for image processing applications, Journal of Imaging 5 (2019).

[200] T.S. Abdelrahman, Cooperative software-hardware acceleration of kmeans on a tightly coupled cpu-fpga system, ACM Trans. Archit. Code Optim. 17 (Aug) (2020).

[201] E. Marenzi, E. Torti, G. Danese, F. Leporati, Fpga high level synthesis for the classification of skin tumors with hyperspectral images, in: 2022 11th Mediterranean Conference on Embedded Computing (MECO), IEEE, 2022, pp. 1–4.

[202] R.A. Baxter, Mixture Model, Springer US, 2010.

[203] Y. Huang, K.B. Englehart, B. Hudgins, A.D.C. Chan, Optimized Gaussian mixture models for upper limb motion classification, in: The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, vol. 1, 2004, pp. 72–75. The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.

[204] J.J. Shen, A stochastic-variational model for soft mumford-shah segmentation, Int. J. Biomed. Imag. 2006 (2006) 092329.

[205] S. Brahim-Belhouari, A. Bermak, M. Shi, P.C.H. Chan, Fast and robust gas identification system using an integrated gas sensor technology and Gaussian mixture models, IEEE Sensor. J. 5 (2005) 1433–1444.

[206] M. Shi, A. Bermak, S. Chandrasekaran, A. Amira, An efficient fpga implementation of Gaussian mixture models-based classifier using distributed arithmetic, in: 2006 13th IEEE International Conference on Electronics, Circuits and Systems, 2006, pp. 1276–1279, 2006 13th IEEE International Conference on Electronics, Circuits and Systems.

[207] C. Guo, H. Fu, W. Luk, A fully-pipelined expectationmaximization engine for Gaussian mixture models, in: 2012 International Conference on Field-Programmable Technology, 2012, pp. 182–189, 2012 International Conference on Field-Programmable Technology.

[208] C. He, H. Fu, C. Guo, W. Luk, G. Yang, A fully-pipelined hardware design for Gaussian mixture models, IEEE Trans. Comput. 66 (2017) 1837–1850.

[209] M. Neggazi, M. Bengherabi, Z. Boulkenafet, A. Amira, An efficient fpga implementation of Gaussian mixture models based classifier: application to face recognition, in: 2013 8th International Workshop on Systems, Signal Processing and Their Applications (WoSSPA), 2013, pp. 367–371, 2013 8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA).

[210] A. Ashrafi, M. Genovese, E. Napoli, D.D. Caro, N. Petra, A.G.M. Strollo, Fpga implementation of Gaussian mixture model algorithm for 47 fps segmentation of 1080p video, Journal of Electrical and Computer Engineering 2013 (2013) 129589.

[211] M. Genovese, E. Napoli, Asic and fpga implementation of the Gaussian mixture model algorithm for real-time segmentation of high definition video, IEEE Trans. Very Large Scale Integr. Syst. 22 (2014) 537–547.

[212] L.M. Simms, B. Blair, J. Ruz, R. Wurtz, A.D. Kaplan, A. Glenn, Pulse discrimination with a Gaussian mixture model on an fpga, Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip. 900 (2018) 1–7.
[213] Y. Nagamine, K. Yoshihiro, M. Shibata, H. Yamada, S. Enokida, H. Tamukoh, A hardware-oriented algorithm of gmm-mrcohog for high-performance human detection by an fpga, in: International Workshop on Advanced Imaging Technology (IWAIT), vol. 11766, SPIE, 2021, pp. 62–67, 2021.
[214] A. Ghaheri, S. Shoar, M. Naderan, S.S. Hoseini, The applications of genetic algorithms in medicine, Oman Med. J. 30 (Nov. 2015) 406–416.
[215] D.E. Goldberg, "Genetic Algorithms in Search," Optimization, and MachineLearning, 1989.
[216] M. Wall, Galib: A C++ Library of Genetic Algorithm Components, vol. 87, Mechanical Engineering Department, Massachusetts Institute of Technology, 1996, p. 54.
[217] B. Shackleford, G. Snider, R.J. Carter, E. Okushi, M. Yasuda, K. Seo, H. Yasuura, A high-performance, pipelined, fpga-based genetic algorithm machine, Genet. Program. Evolvable Mach. 2 (2001) 33–60.
[218] C. Aporntewan, P. Chongstitvatana, A hardware implementation of the compact genetic algorithm, in: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), vol. 1, 2001, pp. 624–629, 1, Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546).
[219] M. Vavouras, K. Papadimitriou, I. Papaefstathiou, High-speed fpga-based implementations of a genetic algorithm, in: 2009 International Symposium on Systems, Architectures, Modeling, and Simulation, 2009, pp. 9–16.
[220] P.R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, A. Stoica, Customizable fpga ip core implementation of a general-purpose genetic algorithm engine, IEEE Trans. Evol. Comput. 14 (2010) 133–149.
[221] S.D. Scott, A. Samal, S. Seth, Hga: a hardware-based genetic algorithm, in: Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays, 1995, pp. 53–59. Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays, Association for Computing Machinery.
[222] W. Tang, L. Yip, "Hardware Implementation of Genetic Algorithms Using Fpga," the 2004 47th Midwest Symposium on Circuits and Systems, MWSCAS '04, 2004, 2004.
[223] N. Yoshida, T. Yasuoka, Multi-gap: parallel and distributed genetic algorithms in vlsi, in: IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), vol. 5, 1999, pp. 571–576, vol.5, IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028).
[224] Y.-H. Choi, D. jin Chung, Vlsi processor of parallel genetic algorithm, in: Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No.00EX434), 2000, pp. 143–146. Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No.00EX434).
[225] M.S. Jelodar, M. Kamal, S.M. Fakhraie, M.N. Ahmadabadi, Sopc-based parallel genetic algorithm, in: 2006 IEEE International Conference on Evolutionary Computation, 2006, pp. 2800–2806, 2006 IEEE International Conference on Evolutionary Computation.
[226] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, M. Ito, General architecture for hardware implementation of genetic algorithm, in: 2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, pp. 291–292, 2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.
[227] T. Kamimura, A. Kanasugi, A parallel processor for distributed genetic algorithm with redundant binary number, in: 2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012), 2012, pp. 125–128, 2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012).
[228] Y. Jewajinda, P. Chongstitvatana, Fpga implementation of a cellular compact genetic algorithm, in: 2008 NASA/ESA Conference on Adaptive Hardware and Systems, 2008, pp. 385–390, 2008 NASA/ESA Conference on Adaptive Hardware and Systems.
[229] P.V. dos Santos, J.C. Alves, J.C. Ferreira, A framework for hardware cellular genetic algorithms: an application to spectrum allocation in cognitive radio, in: 2013 23rd International Conference on Field Programmable Logic and Applications, 2013, pp. 1–4, 2013 23rd International Conference on Field programmable Logic and Applications.
[230] J. Newborough, S. Stepney, A generic framework for populationbased algorithms, implemented on multiple fpgas, in: C. Jacob, M.L. Pilat, P.J. Bentley, J. I. Timmis (Eds.), Artificial Immune Systems, Artificial Immune Systems, Springer Berlin Heidelberg, 2005, pp. 43–55.
[231] T. Lei, Z. Ming-cheng, W. Jing-xia, The hardware implementation of a genetic algorithm model with fpga, in: 2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings, 2002, pp. 374–377.
[232] K.M. Deliparaschos, G.C. Doyamis, S.G. Tzafestas, A parameterised genetic algorithm ip core: fpga design, implementation and performance evaluation, Int. J. Electron. 95 (2008) 1149–1166.
[233] I.C. Vizitiu, I.C. Rˆıncu, A. Radu, I. Nicolaescu, F. Popescu, Optimal fpga implementation of garbf systems, in: 2010 12th International Conference on Optimization of Electrical and Electronic Equipment, 2010, pp. 774–779, 2010 12th International Conference on Optimization of Electrical and Electronic Equipment.
[234] Design and implementation of pid controller based on fpga and genetic algorithm, in: Proceedings of 2011 International Conference on Electronics and Optoelectronics, vol. 4, July 2011. V4–308–V4–311, Proceedings of 2011 International Conference on Electronics and Optoelectronics.
[235] Z. Yan-cong, G. Jun-hua, D. Yong-feng, H. Huan-ping, Implementation of genetic algorithm for tsp based on fpga, in: 2011 Chinese Control and Decision Conference (CCDC), 2011, pp. 2226–2231.
[236] M.F. Torquato, M.A.C. Fernandes, High-performance parallel implementation of genetic algorithm on fpga, Circ. Syst. Signal Process. 38 (2019) 4014–4039.
[237] Y. Jewajinda, P. Chongstitvatana, Fpga-based online-learning using parallel genetic algorithm and neural network for ecg signal classification, in: ECTI-CON2010: the 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2010, pp. 1050–1054.