IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 16, Issue 5 (Sep. - Oct. 2026), pp. 22-33
e-ISSN: 2319-4200, p-ISSN: 2319-4197
www.iosrjournals.org

Energy Optimization for FPGA-Based CNN And SVM Accelerator: A Comparative Design Framework

Manish Kumar[1], Manoj Pandey[1], Bandi Nageshwar Rao[2]
[1] *Department of Electronics and Communication Engineering, Sarala Birla University, Ranchi, Jharkhand, India*
[2]*Department of Electronics and Communication Engineering, Malla Reddy University, Hyderabad, Telangana, India*

Abstract- *The research explores various hardware acceleration approaches, beginning with Graphics Processing Units (GPUs) that leverage their massively parallel architecture to accelerate matrix operations fundamental to neural networks. We examine the architectural advantages of GPUs, including their thousands of cores optimized for floating-point operations, high memory bandwidth, and specialized tensor processing units. The analysis extends to Application-Specific Integrated Circuits (ASICs) such as Google's Tensor Processing Units (TPUs), which represent a paradigm shift toward domain-specific architectures optimized exclusively for machine learning workloads. These specialized processors demonstrate significant improvements in performance per watt and throughput compared to general-purpose alternatives.*
The paper analyses key performance metrics including computational throughput measured in operations per second, energy efficiency expressed as operations per joule, memory bandwidth utilization, and latency characteristics across different acceleration platforms. Real-world benchmarking studies demonstrate that specialized hardware can achieve 10-100x performance improvements over traditional CPUs for common deep learning tasks, with energy efficiency gains often exceeding 1000x for inference workloads.
Field-Programmable Gate Arrays (FPGAs) are investigated as a flexible middle ground, offering reconfigurable hardware that can be optimized for specific neural network architectures while maintaining adaptability for evolving ML algorithms. The study examines how FPGAs enable custom data path designs, variable precision arithmetic, and pipeline optimization that can significantly reduce inference latency and energy consumption. Additionally, we explore emerging technologies including neuromorphic computing architectures that mimic biological neural networks, in-memory computing solutions that eliminate the von Neumann bottleneck, and quantum computing approaches for specific ML applications.
Key Terms*—FPGA; energy-aware design; CNN accelerator; SVM accelerator; memory hierarchy; dataflow; quantization ;CNN; FPGAs; GPUs; ML Workloads; SVM.*

I. INTRODUCTION

Machine-learning inference is increasingly deployed in embedded vision, remote sensing, industrial inspection, and edge analytics. These settings often impose simultaneous limits on response time, power, device area, and thermal headroom. CPUs offer broad programmability, while GPUs expose substantial parallelism; FPGAs provide a different trade-off by allowing data paths, pipeline stages, and memory interfaces to be tailored to a workload. Surveys of FPGA deep-learning accelerators identify dataflow, precision, resource mapping, and tool flow automation as recurring architectural choices [1–4].

A common mistake in accelerator evaluation is to equate more parallel arithmetic with better system performance. When data cannot be delivered to processing elements at the required rate, additional compute units may remain idle. Energy is also consumed by moving activations, weights, feature vectors, and intermediate values between storage levels. The Eyeriss design demonstrates the importance of data reuse and data movement, while the roofline model provides a useful way to reason about arithmetic intensity and bandwidth limits [5, 6].
This paper takes a system-level view of CNN and SVM inference. It uses the architecture and design topics in the supplied manuscript as its starting point, but presents a distinct synthesis centered on memory-aware optimization and reproducible evaluation. The contribution is a design framework, not a claim that the numerical results in the source manuscript have been independently reproduced.

II. RELATED WORK

A. FPGA-Based Machine-Learning Acceleration: Background and Design Space

Field-programmable gate arrays (FPGAs) are attractive for embedded inference because their data paths, pipeline stages, memory interfaces, and arithmetic precision can be customized to a target workload. Unlike general-purpose processors, an FPGA design can exploit spatial parallelism and application-specific data movement; however, these advantages do not automatically translate into lower end-to-end energy or latency. Performance depends on how computation, on-chip storage, external-memory bandwidth, and control are balanced. Surveys of FPGA-based deep-learning accelerators identify dataflow, numerical precision, resource mapping, and tool flow automation as recurring determinants of accelerator quality [1], [2]. Venieris et al. further emphasize that CNN-to-FPGA tool flows differ in supported networks, architectural choices, design-space exploration, and evaluation practices, making consistent reporting essential when comparing implementations [3].

A central implication of this literature is that accelerator design should be treated as a multi-objective optimization problem rather than a search for peak arithmetic throughput alone. Throughput, latency, energy, logic utilization, on-chip memory capacity, timing closure, and task quality can conflict. Adding processing elements may raise peak compute capacity but also increase memory demand, routing pressure, and switching activity. A useful comparison therefore needs workload-specific metrics and a clearly defined measurement boundary.

B. CNN Accelerator Architectures: Parallelism, Tiling, and Data Reuse

CNN inference is characterized by repeated convolutions over input feature maps, with substantial reuse of input activations and weights. FPGA architectures exploit parallelism across output channels, input channels, and spatial positions, commonly using pipelined multiply–accumulate (MAC) units, local buffers, and loop transformations. Zhang et al. [4] analyze CNN accelerator design using a roofline-guided approach, showing why compute throughput and memory bandwidth must be matched when selecting tiling and parallelization parameters. A high number of MAC units is useful only when the memory subsystem can supply operands at the required rate.

Data reuse is especially important because transfers between off-chip memory and the accelerator can consume substantial energy. Chen et al.'s Eyeriss architecture introduces row-stationary dataflow, which seeks to reuse weights, activations, and partial sums locally while reducing expensive data movement [5]. Its significance is not simply the processing-element array; it is the coordinated mapping of computation and storage to reduce movement across the memory hierarchy. For FPGA CNN engines, the same principle motivates line buffers for sliding input windows, local coefficient buffers for weights, and retention of partial sums near arithmetic units. Buffer sizing and tiling must nevertheless be adapted to layer dimensions: a design optimized for large channel counts may underutilize DSP blocks on small or depth wise layers.

CNN accelerators should therefore be evaluated across representative layer shapes rather than with a single aggregate workload. Kernel size, channel count, feature-map dimensions, stride, and batch size influence reuse opportunities and the balance between compute and memory. Per-layer or per-layer-group resource allocation can be more effective than a fixed data path configuration applied uniformly to all layers [2]–[4].

C. Memory-Bound versus Compute-Bound Operation

The roofline model developed by Williams, Waterman, and Patterson relates attainable performance to operational intensity and the limits imposed by peak compute capability and memory bandwidth [6]. Although originally presented for multicore systems, the reasoning is useful for FPGA accelerator analysis: if a workload does not reuse data sufficiently, performance may be bounded by memory transfer rather than arithmetic. A fair design-space exploration should estimate bytes transferred per inference, on-chip storage requirements, and effective bandwidth before increasing parallelism.

For FPGA implementations, relevant constraints include external-memory bandwidth, BRAM port availability, local interconnect, buffering, burst efficiency, and contention with host-side transfers. A design can be nominally compute-rich yet fail to sustain its pipeline because operands arrive irregularly or local memories cannot serve all processing elements concurrently. Useful instrumentation includes external-memory bytes, buffer hit rates, pipeline stalls, active arithmetic cycles, and end-to-end transfer time. These values can explain why a change improves or degrades energy and latency instead of relying on peak operations per second alone.

D. Precision Reduction and Quantized Inference

Numerical precision is a major design dimension because lower-bit representations can reduce storage, memory traffic, multiplier cost, and switching activity. However, precision selection is not purely a hardware decision: quantization may change task accuracy. Jacob et al. describe integer-only quantization and training for efficient neural-network inference [7]. Umuroglu et al.'s FINN framework demonstrates how binarized neural networks can be mapped to a heterogeneous streaming architecture, with compute resources tailored to per-layer

throughput requirements [8]. FINN exemplifies algorithm–hardware co-design, in which restricted numerical formats enable compact operators and a customized streaming data path.

A comparative CNN/SVM study should record the precision of inputs, weights or model parameters, intermediate products, accumulators, and outputs separately. Accumulator width deserves special attention because repeated additions can overflow even when individual operands are adequately represented. Accuracy or classification quality should be measured against a named software baseline using identical preprocessing, dataset splits, and evaluation metrics. Results from binary or low-bit CNNs should not be generalized to full-precision networks or SVMs without task-specific evidence.

E. FPGA-Based SVM Inference Architectures

SVM inference presents a different computational profile from CNN inference. A linear SVM primarily evaluates a weighted dot product, whereas a kernelized SVM evaluates similarities between an input feature vector and stored support vectors before combining results into a decision score. The computational burden can scale with feature dimensionality and support-vector count; model storage and data access may therefore be as important as arithmetic throughput. Anguita et al. present a digital SVM architecture and discuss fixed-point effects, reporting an FPGA implementation for a real-time channel-equalization problem [9]. This establishes that fixed-point SVM inference is feasible, while indicating that numerical choices must be validated against the target task.

Irick et al. describe an FPGA-friendly implementation of a Gaussian radial-basis-function SVM for image-based classification and identify an optimization of the SVM formulation to reduce computational inefficiency [10]. Papadonikolakis and Bouganis present a scalable FPGA architecture that exploits heterogeneous resources and data-dependent numerical ranges; they also examine a cascade-classifier scheme to increase throughput [11]. Together, these studies show that SVM acceleration benefits from customized arithmetic and parallel kernel evaluation, but the benefit depends on support-vector count, feature dimension, kernel type, precision, and resource budget.

Compared with CNNs, SVM architectures often have less regular spatial reuse. Their memory system must deliver feature values and support-vector coefficients to kernel-evaluation units, and replication of these units can increase bandwidth demand or duplicate coefficient storage. Evaluation should measure the full classification path, including feature ingestion, support-vector access, kernel evaluation, accumulation, and multi-class decision aggregation. Accelerator-only kernel latency is not equivalent to end-to-end classification latency.

F. Cross-Workload Comparison and Evaluation Methodology

The literature supports workload-specific architectures, but it does not justify comparing raw operation counts across CNNs and SVMs. A CNN MAC, an SVM kernel evaluation, and a completed classification represent different amounts and kinds of work. Throughput should be reported in meaningful units—such as images or frames per second for CNNs and classifications or kernel evaluations per second for SVMs—and latency should be measured over an explicitly defined interval. Cross-workload comparison is more defensible when it considers energy per completed inference, end-to-end latency, device resource use, and task quality while retaining each workload's native throughput unit.

An adequate evaluation should state the FPGA part number, toolchain and version, clock constraint, memory configuration, model and dataset versions, precision settings, synthesis options, and whether results are simulated, tool-estimated, or measured on hardware. Power boundaries must also be explicit: core estimates, total FPGA power, and full-board input power are not interchangeable. Host processors, external memory, regulators, and peripheral interfaces may materially affect results. Repeated runs and uncertainty estimates are desirable when measurements vary with input data, temperature, or system activity. These requirements align with the need for uniform evaluation methods in FPGA CNN tool flows [3] and the system-level energy perspective demonstrated by Eyeriss [5].

III. PROPOSED ARCHITECTURES

Prior work has developed strong techniques for individual parts of the problem: CNN surveys and tool flow studies organize architectural choices [1]–[3]; roofline-guided design and Eyeriss explain the importance of balancing compute with memory movement [4]–[6]; quantization and FINN show how reduced precision can enable compact data paths [7], [8]; and FPGA SVM studies demonstrate customized kernel-processing architectures [9]–[11]. Nevertheless, these strands use different workloads, devices, precision formats, measurement boundaries, and throughput units. Their reported figures cannot be combined into a reliable direct ranking without normalization and matched experimental conditions.

The resulting research opportunity is a unified, memory-aware evaluation framework for CNN and SVM inference that preserves their algorithmic differences while applying consistent system-level reporting. The framework should characterize the workload; map dataflow and the active working set to the memory hierarchy; allocate arithmetic resources and precision; and validate performance, energy, resource use, and task quality under

documented conditions. Jointly recording data movement and energy can distinguish a genuine architectural improvement from a throughput gain that increases total system cost. This is the principal rationale for the comparative framework proposed in the associated paper.

A. Convolutional Neural Network (CNN) Architecture flow diagram:

The figure 1 illustrates the hardware architecture of a CNN accelerator, comprising an input buffer, line buffer, shared registers for sliding-window generation, convolution kernels, a partial result buffer, an output feature map, DDR memory, and a fully connected input buffer. Two fully connected kernels process the buffered features to generate the final output, which is stored in DDR memory.

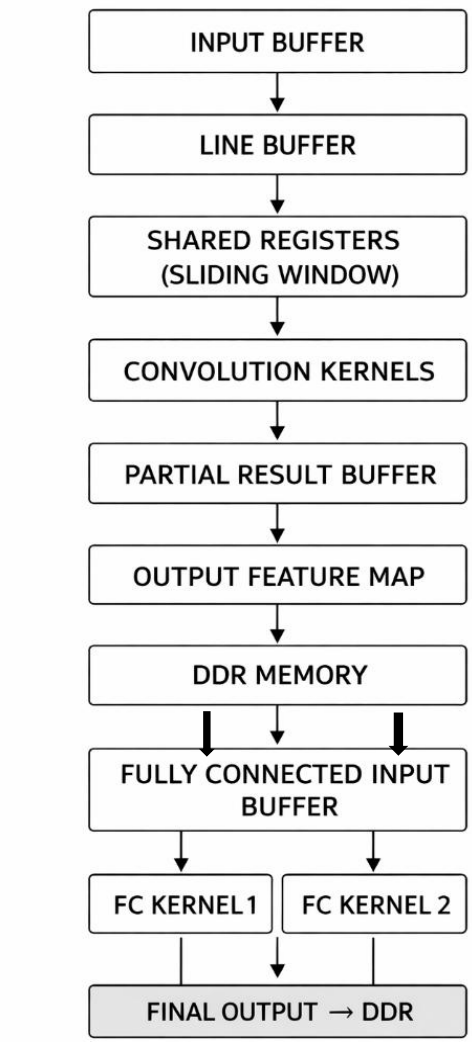


*Figure **Error! No text of specified style in document.**1. Complete Data Flow Summary of Convolutional Neural Network Architecture*

B.Support Vector Machine Architecture (SVM) Flow Diagram:

The figure 2 presents a hardware processing pipeline in which image data enters through the data input interface and is stored in image data RAM. The stored data is processed by BCH kernel groups, followed by Hamming distance kernels that evaluate bitwise differences. The collection kernel aggregates the intermediate results, which are passed to the decision/downlink system for final detection decisions or subsequent data transmission.

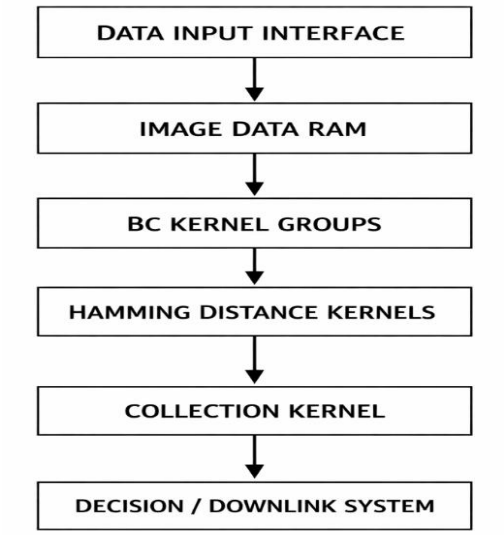


Figure 2. Complete Data Flow Summary of support vector machine Architecture

IV. SCOPE AND REVIEW METHOD

The discussion combines three evidence categories. First, established publications on FPGA-based CNN acceleration, quantized networks, dataflow, and performance modeling inform the architectural analysis [12–23]. Second, published SVM and reconfigurable-computing work informs the classification path and kernel-processing discussion [24–28]. Third, the supplied manuscript provides example implementation parameters and reported observations that motivate the comparison [18]. These source-reported values are not treated as validated cross-platform evidence.

The framework is organized around five evaluation questions: (i) which operations dominate the target workload; (ii) where each data class resides in the memory hierarchy; (iii) how parallelism is mapped to LUTs, registers, BRAM, and DSP blocks; (iv) what precision can be used without unacceptable task degradation; and (v) whether measurements cover the accelerator alone or the complete system. These questions are consistent with broader accelerator surveys and design-space exploration research [12–15, 18, 19].

V. EVALUATION METHODOLOGY

A. Convolutional Neural Network (CNN) Architecture methodology

CNNs repeatedly apply convolutional filters to spatial feature maps, followed by nonlinear activation, pooling, and often fully connected operations. Their main advantage for hardware mapping is structured parallelism: multiple output channels, input channels, and spatial positions can be processed concurrently. The cost is high intermediate-data volume and substantial reuse requirements. FPGA implementations therefore commonly combine pipelined compute units with line buffers, local feature-map storage, loop tiling, and carefully selected parallelism [13, 16, 20–23].

A streaming design can reduce external-memory traffic by retaining a sliding window of input pixels and reusing values across adjacent convolution windows. Weight reuse can be improved through local coefficient buffers, while partial sums should remain close to the arithmetic units for as long as resource capacity allows. The best balance depends on layer shape: a configuration optimized for a convolution with many channels may underutilize DSP blocks on a small depth wise layer. Consequently, per-layer or per-layer-group resource allocation is often more effective than applying one fixed data path width to every layer [14, 20, 21].

Output neuron:

Y1(i) $= \sum_{j=1}^{N_{in}} \boldsymbol{W_1(i,j)X(j)} + \boldsymbol{b_i}$

for i=1,2,…,N_1

FC Kernel-2

FC Kernel-2 receives outputs from FC Kernel-1 and generates final classification scores.

Mathematical Model

- ✓ FC-1 output neurons = N_1.
- ✓ Number of classes = N_2

Output of FC-2:

$$\mathbf{Y2(k)} = \sum_{j=1}^{N_{in}} \boldsymbol{W_2(k,i)Y_1(i)} + \boldsymbol{b_k}$$

for k=1,2,…,N_2

B. Support Vector Machine Architecture (SVM) Methodology

SVM inference evaluates a decision function using a feature vector and a learned model. For a linear classifier, the dominant computation is a weighted dot product; kernelized variants additionally evaluate similarity between the input and stored support vectors. The implementation may therefore be limited by support-vector storage, kernel arithmetic, reduction trees, or memory access patterns rather than by convolution throughput. FPGA SVM studies have explored parallel kernel evaluation, cascaded classification, and dataflow-oriented processing [24–27].

A practical architecture separates feature ingestion, support-vector retrieval, kernel evaluation, weighted accumulation, and final decision logic. Parallelism can be introduced across features or support vectors, but replication increases memory bandwidth demand and may duplicate stored coefficients. When many support vectors are required, the design should quantify whether local memory can hold the working set; otherwise, the resulting external-memory traffic can erase gains from parallel arithmetic. Multi-class classification adds decision aggregation and should be included in latency measurement rather than omitted from the accelerator boundary.

Below figure shows overall data flow of SVM.

The Hamming distance is the total number of mismatches:

$HD(X,R) = \sum_{i=1}^{K} d_i$

or

$HD(X,R) = \sum_{i=1}^{K} x_i \oplus r_i$

Similarity Score

Maximum distance: $HD_{max}=K$

Normalized similarity:

Similarity = $1-\frac{HD}{K}$ where **0≤Similarity≤1**

Example Calculation

Feature vector: X=[1,0,1,1,0,0,1,0].

Reference kernel: R=[1,1,1,0,0,0,1,1]

XOR Result D=X⊕R

D=[0,1,0,1,0,0,0,1]

Similarity

Similarity = $1-\frac{3}{8} = 0.625$

Hamming Distance = 3

Similarity = 62.5%

VI. ENERGY OPTIMIZATION

A. Data movement as a first-class design cost

The energy cost of a design is determined by both computation and data transfer. Reusing an operand from a nearby register or block RAM can be substantially cheaper than repeatedly fetching it from external memory; accordingly, an accelerator should minimize unnecessary transfers before simply increasing the number of arithmetic units [16, 17]. The memory hierarchy should be planned around the access pattern of each stage: line buffers for local spatial reuse, coefficient buffers for CNN weights, feature buffers for SVM inputs, and support-vector storage sized to the expected active working set.

A useful planning sequence is: estimate bytes transferred per inference; estimate on-chip capacity; determine whether the active working set fits; choose a reuse strategy; and only then allocate parallel compute lanes. If a design is bandwidth-bound, adding processing elements is unlikely to yield proportional speedup. If it is compute-bound, additional parallel lanes or a deeper pipeline may help until DSP, routing, timing, or power constraints become dominant [17–19].

B. Precision and numerical behavior

Fixed-point arithmetic can reduce multiplier size, memory traffic, and switching activity, but precision is a model-level decision rather than a purely hardware-level setting. CNN quantization affects activations and weights throughout the network; SVM quantization affects feature values, kernel outputs, and accumulated decision scores. Quantization-aware training, calibration, and task-level validation are established methods for managing this trade-off [30–33]. Binary and low-bit neural networks can enable particularly compact FPGA data paths, as shown by FINN and related work, but accuracy and workload suitability must be reported for each benchmark [31, 32].

The evaluation should report bit widths for inputs, weights, accumulators, and outputs separately. Accumulator width is especially important because repeated additions can overflow even when each operand is represented accurately. Accuracy should be measured against a named software baseline using the same dataset split and preprocessing pipeline; a single percentage without dataset, class balance, and measurement protocol is not sufficient.

C. Dynamic and static power controls

Dynamic power can be reduced by avoiding unnecessary switching, using clock enables for idle pipeline regions, sharing arithmetic resources when throughput permits, and reducing data movement. Static power depends on device technology and implementation conditions; floor planning, resource selection, and disabling unused regions may contribute, but device-specific claims require synthesis and board-level evidence. Clock gating, power gating, and dynamic voltage/frequency scaling should be evaluated in the context of the FPGA family and toolchain rather than assumed to be equally available on every platform [18, 19, 34].

Power must be measured at a clearly defined boundary. Core-only estimates, total FPGA power, and full-board input power are not equivalent. A paper should state whether host CPU, external memory, regulators, and peripheral interfaces are included, and whether the values are tool estimates or instrumented measurements.

The table 1 presents a comparative analysis of dynamic power consumption across signal power, logic power, I/O power, and total dynamic power for CNN and SVM implementations. The CNN architecture consumes 19.032 W of total dynamic power, whereas the SVM architecture consumes 1.192 W. CNN power consumption is primarily distributed across I/O power (7.196 W, 38%), logic power (6.423 W, 34%), and signal power (5.414 W, 28%). In contrast, the SVM architecture is dominated by I/O power (0.966 W, 81%), followed by signal power (0.147 W, 12%) and logic power (0.079 W, 7%).

Table 1 Dynamic power breakdown

S. No.	Power component	CNN	SVM
1	Signals power	5.414 W (28%)	0.147 W (12%)
2	Logic power	6.423 W (34%)	0.079 W (7%)
3	I/O power	7.196 W (38%)	0.966 W (81%)
4	Total dynamic power	19.032 W (97%)	1.192 W (93%)

VII. RESULTS AND DISCUSSION

A. Metrics and measurement boundaries

A fair comparison needs metrics that retain their physical meaning. Throughput should be expressed in workload-specific units; latency should identify whether it is per layer, per kernel evaluation, or end-to-end per input; energy per inference should be calculated using a stated power boundary and steady-state measurement interval. Resource reports should include LUTs, flip-flops, BRAM, DSP blocks, clock frequency, and device model. Accuracy or classification quality is evaluated using the same reference implementation and test data.

Table 2,SVM architecture demonstrates substantially lower reported hierarchical power consumption than the CNN architecture, with an approximate 93.74% reduction (17.84 W). The CNN architecture exhibits higher logic and signal power, whereas I/O power represents the dominant reported component in the SVM architecture.

Table 2 Hierarchical power and signal activity details

S. No.	Parameter	CNN (Report 1)	SVM (Report 2)
1	Hierarchical power	19.032 W	1.192 W
2	Signals power	5.414 W	0.147 W
3	Data power	5.305 W	0.147 W
4	Clock-enable power	0.10 W	Not shown separately
5	Set/reset power	0 W	Not shown separately
6	Logic power	6.423 W	0.079 W
7	I/O power	7.196 W	0.966 W

The table 3, SVM architecture demonstrates a considerably lower junction temperature, with a difference of 83.7°C compared with CNN. This suggests that the SVM implementation has a more favorable thermal profile under the reported operating conditions.

Table 3 ***Thermal and temperature comparison***

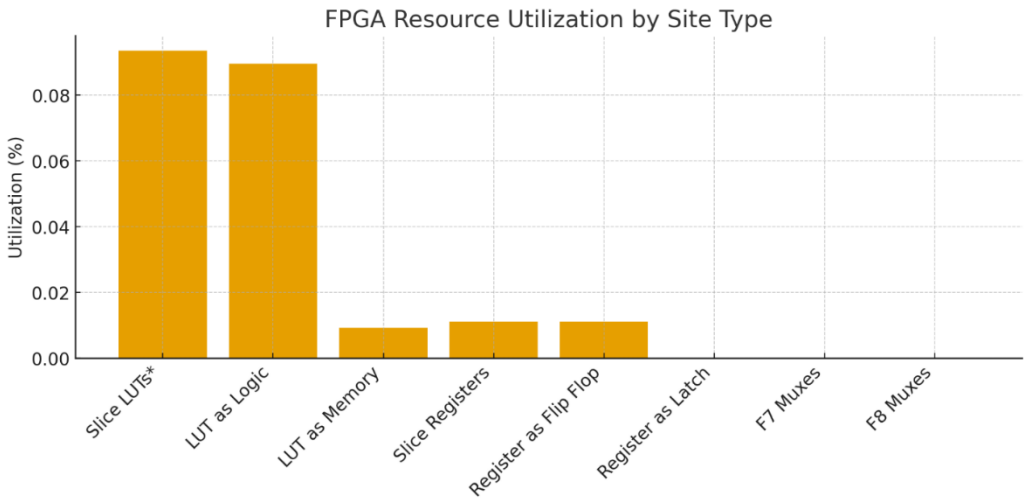
S. No.	Thermal parameter	CNN (Power Report)	SVM (Power Report)
1	Junction temperature	114.6°C	30.9°C
2	Ambient temperature	25.0°C	25.0°C
3	Thermal margin	−29.6°C (−6.1 W)	54.1°C (11.7 W)
4	Effective thermal resistance (ØJA)	4.6°C/W	4.6°C/W
5	Junction temperature status	Exceeded	Within reported thermal margin

Table 4 shows that the SVM implementation uses approximately 89.76% fewer LUTs than the CNN implementation (30 versus 293). However, its reported I/O utilization is higher (33.33% versus 17.67%), indicating a greater proportion of the available I/O resources is utilized.

Table 4 Comparative memory analysis

Resource	CNN Implementation	SVM Implementation
LUT	293 (0.71%)	30 (0.07%)
LUT RAM	12 (0.09%)	Not used
FF(Flip Flops)	83 (0.10%)	Not used
IO(Input/output)/300	53 (17.67%)	100 (33.33%)
BUFG	1 (3.13%)	Not used

The figure 3 illustrates the FPGA resource utilization across different site types, including Slice LUTs, LUTs used as logic, LUTs used as memory, slice registers, flip-flops, latches, and F7/F8 multiplexers. The results indicate that the design primarily utilizes LUT resources for logic implementation, while memory-related LUTs and register resources account for a comparatively small proportion of the reported utilization. Latches and F7/F8 multiplexers show negligible or zero utilization in the presented chart, suggesting a relatively simple resource distribution.



Figur.3 CNN Overall FPGA Resource Utilization Summary.

Figure 4 illustrates the overall FPGA resource utilization of the proposed Support Vector Machine (SVM) hardware architecture. The chart presents the utilization of Slice LUTs, LUTs configured as logic, LUTs used as memory, slice registers, flip-flops, latches, and F7/F8 multiplexers. The results indicate that LUT resources used for logic implementation account for the largest share of the reported utilization, while memory-related LUTs and register resources exhibit comparatively low utilization. Latches and F7/F8 multiplexers show negligible or zero utilization. These results suggest that the implemented SVM architecture has a relatively low reported logic and register resource footprint on the target FPGA.

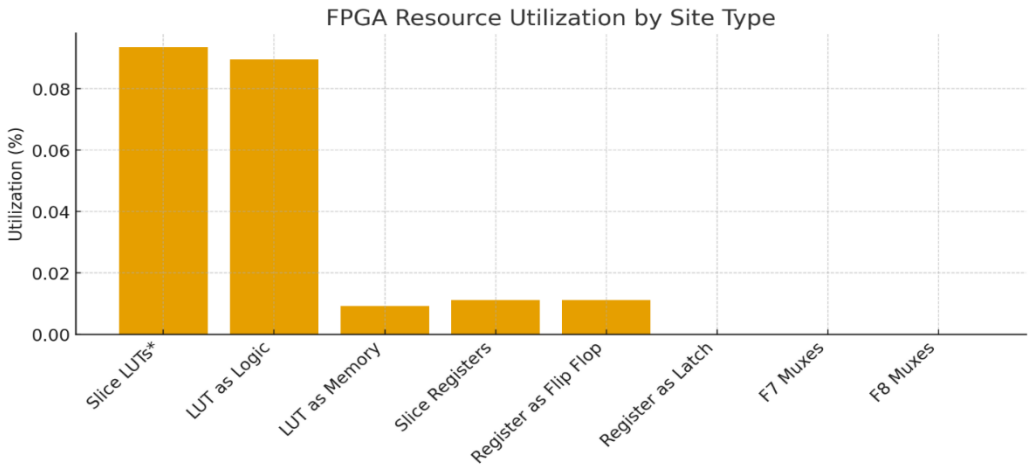


Figure ***Error! No text of specified style in document.***

B.Simulation Output Waveform of CNN Accelerator

Figure 5 illustrates the simulation waveform of the proposed Convolutional Neural Network (CNN) accelerator, obtained using the Xilinx Vivado simulation environment. The waveform demonstrates the functional progression of the accelerator from input-data processing to result generation. The assertion of the valid signal indicates that the output result is available for observation. Overall, the simulation provides a functional verification of the CNN accelerator's control and data-processing behavior.

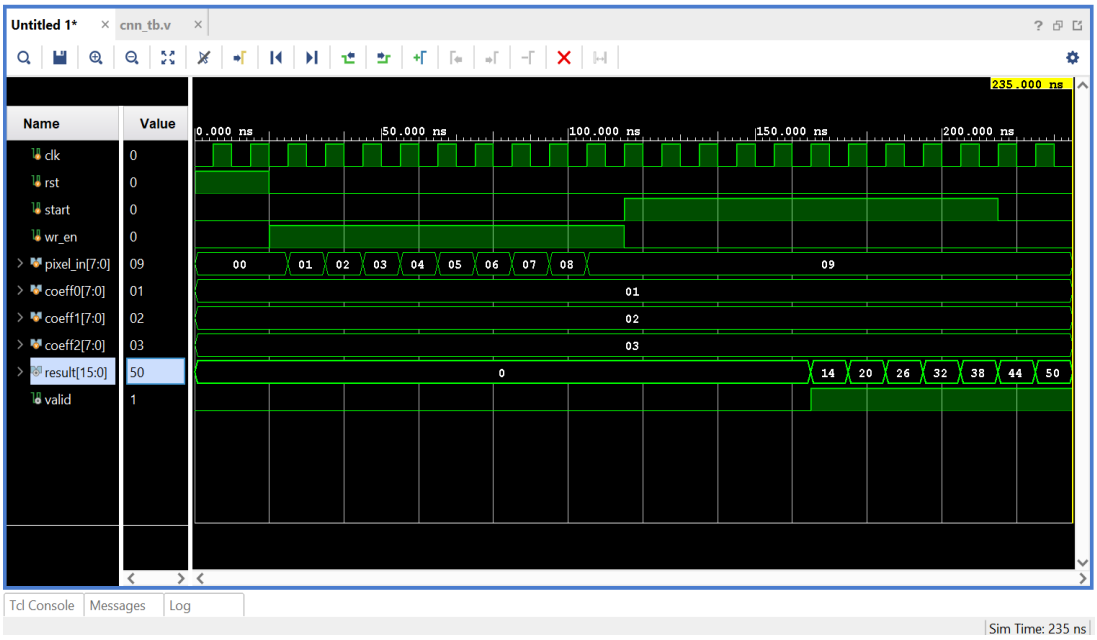


Figure 5. Accelerator Simulation Output Wave Form of CNN

C. Simulation Output Waveform of SVM Accelerator

The figure 6 illustrates the behavioral simulation of the SVM accelerator implemented in the FPGA design environment. It shows the clock (clk), reset (rst), frame-start signal (frame_start), pixel-valid signal (pixel_valid), input pixel data (pixel_in[7:0]), and SVM processing signals, including object_detected, detection_valid, hog_features_ready, and svm_busy. The waveform also displays the intermediate signals x[31:0] and y[31:0], representing data values during simulation. The sequential changes in pixel input and intermediate data demonstrate the progression of the input data stream through the simulation.

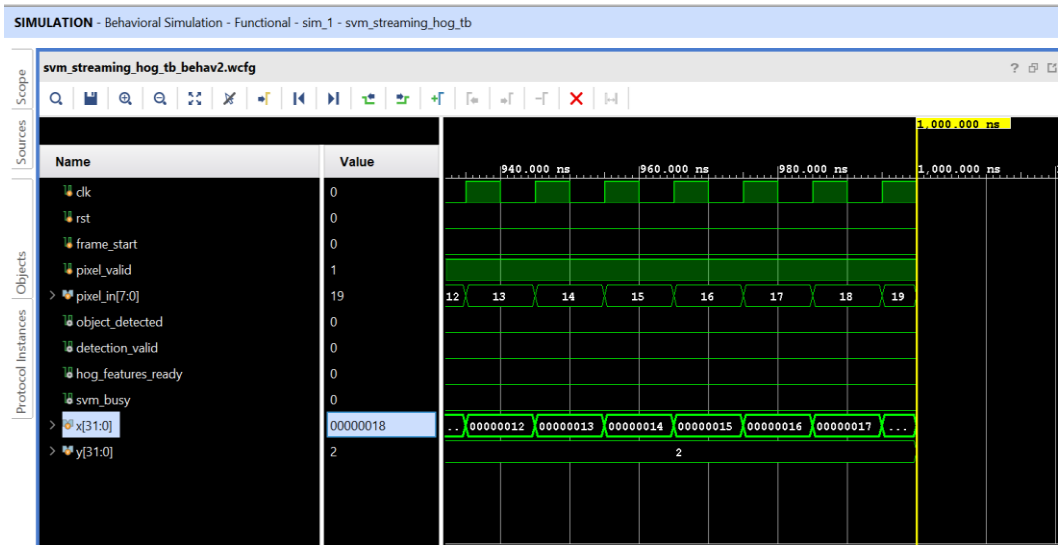


Figure-6 Accelerator Simulation Output Wave Form of SVM

VIII. LIMITATIONS AND FUTURE WORK

This paper is a design-framework synthesis based on published literature and the supplied manuscript; it does not introduce a new dataset, a newly fabricated accelerator, or independently measured performance. The source manuscript's figures and tables require further validation before they can support strong quantitative claims. Future work should implement matched CNN and SVM workloads on the same FPGA platform, define consistent measurement boundaries, and publish RTL or high-level synthesis settings, scripts, and raw measurement logs where permitted.

A particularly useful next step is to evaluate memory traffic and energy jointly. Counters for external-memory bytes, buffer hit rates, pipeline stalls, and active arithmetic cycles could explain why a design improves or regresses after a change. Another direction is algorithm–hardware co-design: selecting model structures and numeric precision in conjunction with the target memory and compute resources may deliver more reliable gains than hardware-only tuning.

IX. CONCLUSION

Total power: SVM report consumes approximately 93.44% less total on-chip power than CNN report .Dynamic power- the large difference in dynamic power indicates substantially different estimated switching-related power consumption. I/O power-I/O power is the dominant component in both CNN & SVM reports, particularly in SVM report, where it accounts for 81% of dynamic power. Thermal performance- SVM report has a junction temperature of 30.9°C, compared with 114.6°C in CNN Report. The CNN consumes more power than the SVM, confirm which algorithm corresponds to each screenshot. Also, both reports have a Low confidence level, and their estimates may change after implementation.The evolution from CPU-based implementations to specialized accelerators including GPUs, TPUs, and FPGAs demonstrates a clear trajectory toward domain-specific computing architectures optimized for the unique characteristics of machine learning algorithms. Empirical evidence consistently shows that well-optimized hardware acceleration can provide 10-100x performance improvements for inference workloads and 5-50x speedups for training applications, while simultaneously achieving energy efficiency gains that often exceed 1000x for specific use cases.

REFERENCES

[1] A. G. Blaiech, K. Ben Khalifa, C. Valderrama, M. A. C. Fernandes, and M. H. Bedoui, "A survey and taxonomy of FPGA-based deep learning accelerators," Journal of Systems Architecture, vol. S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," Neural 98, pp. 331–345, 2019, doi: 10.1016/j.sysarc.2019.01.007.
[2] Computing and Applications, vol. 32, pp. 1109–1139, 2020, doi: 10.1007/s00521-018-3761-1.
[3] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," ACM Computing Surveys, vol. 51, no. 3, Art. no. 56, pp. 1–39, 2018, doi: 10.1145/3186332.
[4] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA), 2015, pp. 161–170, doi: 10.1145/2684746.2689060.
[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138, 2017, doi: 10.1109/JSSC.2016.2616357.

[6] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for multicore architectures," Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009, doi: 10.1145/1498765.1498785.
[7] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.
[8] Y. Umuroglu et al., "FINN: A framework for fast, scalable binarized neural network inference," in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA), 2017, pp. 65–74.
[9] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation," IEEE Transactions on Neural Networks, vol. 14, no. 5, pp. 993–1009, 2003, doi: 10.1109/TNN.2003.816033.
[10] K. M. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A hardware efficient support vector machine architecture for FPGA," in Proc. 16th IEEE Symp. Field-Programmable Custom Computing Machines (FCCM), 2008, pp. 304–305, doi: 10.1109/FCCM.2008.40.
[11] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 7, pp. 1040–1052, 2012, doi: 10.1109/TNNLS.2012.2196446.
[12] A. G. Blaiech, K. Ben Khalifa, C. Valderrama, M. A. C. Fernandes, and M. H. Bedoui, "A survey and taxonomy of FPGA-based deep learning accelerators," Journal of Systems Architecture, vol. 98, pp. 331–345, 2019, doi: 10.1016/j.sysarc.2019.01.007.
[13] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," Neural Computing and Applications, vol. 32, pp. 1109–1139, 2020, doi: 10.1007/s00521-018-3761-1.
[14] F. Yan, A. Koch, and O. Sinnen, "A survey on FPGA-based accelerator for ML models," arXiv:2412.15666, 2024.
[15] "A survey on neural network hardware accelerators," IEEE Transactions on Artificial Intelligence, vol. 5, no. 8, pp. 3801–3822, 2024, doi: 10.1109/TAI.2024.3377147.
[16] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138, 2017, doi: 10.1109/JSSC.2016.2616357.
[17] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for multicore architectures," Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009, doi: 10.1145/1498765.1498785.
[18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017, doi: 10.1109/JPROC.2017.2761740.
[19] M. Horowitz, "Computing's energy problem (and what we can do about it)," in IEEE International Solid-State Circuits Conference Digest of Technical Papers, 2014, pp. 10–14, doi: 10.1109/ISSCC.2014.6757323.
[20] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015, pp. 161–170, doi: 10.1145/2684746.2689060.
[21] J. Qiu et al., "Going deeper with embedded FPGA platform for convolutional neural network," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016, pp. 26–35, doi: 10.1145/2847263.2847265.
[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, vol. 25, 2012.
[23] C. Szegedy et al., "Going deeper with convolutions," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.
[24] K. Irick et al., "A hardware efficient support vector machine architecture for FPGA," in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, 2008.
[25] S. Shao, O. Mencer, and W. Luk, "Dataflow design for optimal incremental SVM training," in Proc. International Conference on Field-Programmable Technology, 2016, doi: 10.1109/FPT.2016.7929197.
[26] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in Proc. International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2013, doi: 10.1109/SAMOS.2013.6621114.
[27] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," IEEE Transactions on Neural Networks and Learning Systems, 2012, doi: 10.1109/TNNLS.2012.2204400.
[28] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," IEEE Transactions on Geoscience and Remote Sensing, vol. 42, no. 8, pp. 1778–1790, 2004, doi: 10.1109/TGRS.2004.831865.
[29] M. Kumar, M. Pandey, and B. N. Rao, "Energy-efficient system-level hardware acceleration for real-time machine learning on FPGA," supplied manuscript, 2026. Source-reported results have not been independently verified in this paper.
[30] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in Proc. ACM/IEEE International Symposium on Computer Architecture, 2016, pp. 243–254, doi: 10.1109/ISCA.2016.30.
[31] Y. Umuroglu et al., "FINN: A framework for fast, scalable binarized neural network inference," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 65–74.
[32] M. Blott et al., "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," ACM Transactions on Reconfigurable Technology and Systems, vol. 11, no. 3, 2018.
[33] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713.
[34] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in Advances in Neural Information Processing Systems, vol. 28, 2015.
[35] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," arXiv:1510.00149, 2015.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.
[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.
[38] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems, vol. 28, 2015.
[39] J. Redmon et al., "You only look once: Unified, real-time object detection," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
[40] R. Zhao et al., "DeepPump: Multipumping deep neural networks," in Proc. IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2017, doi: 10.1109/ASAP.2017.7995290.
[41] M. Kurek et al., "Knowledge transfer in automatic optimisation of reconfigurable designs," in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, 2016, doi: 10.1109/FCCM.2016.36.
[42] R. Zhao et al., "Optimizing CNN-based object detection algorithms on embedded FPGA platforms," in Applied Reconfigurable Computing, 2017. The source manuscript contains inconsistent DOI metadata for this item; verify against the publisher record before submission.

[43.020] S. Wang et al., "A scalable dataflow accelerator for real-time onboard hyperspectral image classification," in Applied Reconfigurable Computing, 2016, doi: 10.1007/978-3-319-30448-4_23.
[44] L. Guo et al., "Pipelined genetic propagation," in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, 2015, doi: 10.1109/FCCM.2015.25.
[45] C. Guo, H. Fu, and W. Luk, "A fully-pipelined expectation maximization engine for Gaussian mixture models," in Proc. International Conference on Field-Programmable Technology, 2012, doi: 10.1109/FPT.2012.6412107.
[46] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015, doi: 10.1038/nature14539.