A Streaming Dataflow Accelerator for Sparse SVM Kernel Computation in Hyperspectral Image Classification

B. B. SHABARINATH [1,2], Pullakandam MURALIDHAR [1]

[1] Dept. of ECE, National Institute of Technology Warangal, Warangal, 506004 Warangal, India
[2] Dept. of ECE, VNR Vignana Jyothi Institute of Engineering and Technology, 500090 Hyderabad, India

bbshabarinath@student.nitw.ac.in, pmurali@nitw.ac.in

Submitted April 9, 2025 / Accepted June 18, 2025 / Online first July 21, 2025

Abstract. *Hyperspectral images (HSI) provide extensive spectral information but their high dimensionality and redundancy create substantial challenges for computation and storage while increasing energy demands. The proposed solution combines sparse dictionary learning with Field Programmable gate Array (FPGA)-accelerated Sparse matrix vector multiplication (SpMV) operations and Support Vector Machine (SVM) training to tackle these issues. Spatial patches and spectral blocks partition HSI to enable the extraction of compact discriminative sparse features through the use of a learned sub-dictionary. In contrast to deep learning frameworks which demand large training datasets and generate significant computational overhead, the SVM-based approach achieves efficient real-time training and adaptation. The FPGA accelerator executes intensive SpMV operations through dynamic load balancing. We tested our approach with four varied HSI datasets gathered from aerial and UAV systems as well as terrestrial platforms on the PYNQ-Z2 board. Our design reaches classification accuracies between 98.65% and 99.95% across datasets including Indian Pines,AVRIS-NG, Cubert-UAV, Cubert-Terrestrial with per-pixel classification times below 7 μs and inference times up to 36× faster than optimized software baselines which remain under typical sensor acquisition times. The strategy requires less than 0.24 W of on-chip power at maximum load which makes it ideal for deployment on satellites or UAVs. The proposed method outperforms existing FPGA-based SVM architectures in classification accuracy and throughput while enabling on-device incremental learning which makes it ideal for analyzing hyperspectral images in real-time.*

Keywords

Hyperspectral image classification, support vector machine, sparse dictionary learning, sparse matrix-vector multiplication, load balancing

1. Introduction

Hyperspectral imaging technology shows great potential for use in remote sensing as well as precision agriculture, environmental monitoring, and defense because it collects detailed spectral data across multiple narrow bands [1]. The detailed spectral resolution of the imagery provides precise identification of materials and deeper understanding of scenes. While HSI's capabilities make it a powerful tool, these same features lead to substantial computational and practical challenges which need to be addressed. The high-dimensional structure of HSI data generates massive information volumes which create severe memory restrictions and demand higher computational power during processing [2].

High redundancy in spectral and spatial dimensions makes conventional algorithms inefficient and time-consuming. Current HSI classification approaches mostly emphasize inference processes instead of training stages [3–5]. Deep learning-based frameworks drive many modern solutions which achieve quick classification following pre-training of their models. Inference-focused systems achieve high accuracy results but fail to provide the necessary flexibility for real-time system adaptation. Dynamic environments face significant limitations when models cannot be updated quickly enough to adapt to changing spectral signatures or newly emerging classes. Traditional GPU-based methods have high energy demands which prevent their use in resource-limited environments like edge devices in mobile or remote applications [3], [4]. Support Vector Machines (SVMs) offer a solution that combines reduced complexity with enhanced flexibility [6], [7]. The need for less training data combined with efficient optimization and hardware acceleration capabilities allows SVMs to support quick retraining while minimizing power usage. Through Field Programmable Gate Array (FPGA)-based acceleration SVMs achieve competitive classification performance and higher adaptability and efficiency which makes them optimal for real-time HSI processing in changing scenarios [8]. The fundamental complexity of SVM training creates essential difficulties for HSI classification. The iterative SVM optimization process demands significant computational re-

DOI: 10.13164/re.2025.0482

sources because it requires calculating kernel matrix entries for numerous high-dimensional feature vectors. Feature extraction in many existing systems operates through static dictionaries that simplify data representation but often fail to identify the most distinctive data features. Sparse representation techniques have become a focus in research because they enable better use of HSI data redundancy to achieve both more accurate and computationally efficient results.

The current difficulties necessitate an integrated solution that allows for both precise inference and rapid adaptive training. The system needs to process high-dimensional data while updating in response to new datasets in real time and maintain energy efficiency for embedded devices. Our research introduces an innovative framework that integrates sparse dictionary learning for effective HSI feature extraction with an SVM training process accelerated by FPGA hardware. Our approach takes advantage of FPGA hardware's reconfigurable properties to handle computationally demanding sparse matrix-vector multiplication (SpMV) operations instead of using traditional GPU or CPU-based solutions. The system maintains ideal resource usage through dynamic load balancing in its FPGA architecture which handles the changing processing demands of HSI applications.

Our integrated system surpasses current methods by allowing embedded devices to perform complete training operations so they can adapt models continuously and make real-time updates which inference-only systems currently lack. The developed framework boosts processing speed and energy efficiency while enabling stronger and more adaptable hyperspectral image classification under practical constraints. Overall, this study primarily aims to provide the following contributions:

- A sparse representation method for HSI is developed that exploits both spectral and spatial redundancies through partitioning HSI into spectral blocks and spatial groups which transforms high-dimensional optimization into smaller solvable sub-problems to both lower computational complexity and improve classification accuracy.

- A parallel architecture for SpMV on FPGA platforms features dynamic load balancing to improve resource utilization efficiency and reduce overall energy consumption.

- A comprehensive SVM training capability for embedded systems has been established by combining dictionary learning with sparse coding and FPGA-powered SpMV to support adaptive re-training and real-time model updates beyond standard inference functions. The integration of AXI DMA into hardware/software co-design demonstrates superior throughput performance and latency reduction for SVM training compared to direct DDR access.

The structure of this paper continues with the following sections. The second section discusses related research in sparse representation techniques and FPGA-accelerated computing approaches. Section 3 explains the proposed dataflow-based methodology. The accelerator design, hardware/software partitioning and system integration process is detailed in Sec. 4. Section 5 showcases experimental outcomes. Section 6 ends with a summary of findings and explores future research directions.

2. Related Works

HSI classification continues to draw significant research attention because of the abundant spectral data present in these images but faces major hurdles in acquisition and processing because of their high dimensional nature and inherent data redundancy. HSIs suffer from limited spatial resolution which leads to spectral mixing that is typically represented through the Linear Mixture Model [9]. Traditional techniques mainly used neural networks for pixel-level spectral classification while utilizing dimension reduction or band selection approaches to handle the curse of dimensionality [10, 11]. The separate processing of individual pixels in these methods ignores essential spatial correlations which reduces their overall classification accuracy. Recent research efforts have implemented spatial feature extraction methods including Local Binary Patterns [12], wavelet transforms [13], and morphological profiles [14] to improve the capture of contextual information. Deep learning systems [15], [16] demonstrate effective performance through simultaneous modeling of spectral and spatial features. This comes at the cost of requiring extensive computational resources and large training datasets which limits their practicality for real-time applications and environments with restricted resources.

Sparse Representation (SR) serves as a powerful framework for HSI processing by overcoming the difficulties presented by extensive spectral data dimensions and data redundancies [17]. The learning process creates a dictionary using information from hyperspectral data. This dictionary functions as a matrix which uses its columns as atoms. Atoms function as vectors that represent distinct spectral features found in the data. The SR algorithm represents each pixel spectrum as a linear combination of a subset of dictionary atoms. Multiple SR techniques in hyperspectral image processing operate with predefined dictionaries that originate from spectral libraries or training data [18], [19]. Peng et al. [20] created a dictionary together with their spatial neighbors then improved it into a locally adaptive dictionary by choosing atoms that showed the highest correlation for each test set. Fixed dictionaries require minimal computational resources to operate but their performance depends heavily on training data selection and external spectral libraries may need further calibration. Researchers now focus on deriving optimal dictionaries straight from data sets [21–23] which leads to higher computational ex-

penses [24]. Former approaches emphasized spectral information but overlooked spatial correlations between neighboring pixels [25], [26]. Various research has started overcoming this issue through spatial pixel grouping with techniques like non-overlapping square patches [21], Laplacian constraints [27], and sliding window methods [28] to create a uniform sparsity pattern. Techniques include more dynamic methods like super-pixel segmentation to identify homogeneous regions as shown in research [29], [30]. Although progress has been made advancements in utilizing both spectral and spatial redundancies still pose challenges together while the large dimensionality of HSIs maintains the computational difficulty for spectral reconstruction optimization.

The combination of sparse dictionary representations and SVM classifiers improves accuracy by offering discriminative feature vectors well-suited to SVM decision boundaries. Nevertheless, the computational constraint persists more or less in the SpMV operations intrinsic in the SVM classification step, particularly during computation on sparse coefficient vectors in real-time. FPGAs, due to their capability for parallel computation, have a very viable solution to overcome computational constraints in sparse data operations. In particular, FPGA architectures are significantly well-suited to speed up SpMV computations, facilitate parallel processing with low latency, and significantly improve power efficiency. However, hardware-based-only designs may fail to leverage the flexibility and programmability of current-day embedded processing systems to the fullest extent, instigating hardware-software co-design as a very appealing and feasible solution. The hyperspectral classification FPGA-based hardware accelerator [3] is plagued with issues such as high DSP block dependence and assumptions of ideal data conditions that cannot accommodate real latency and memory conditions. Although the composite kernel [4] enhances the classification accuracy, it raises the computational complexity, and the latter might require approximations to accommodate real-time requirements on FPGA. The real-time accelerator [5] is bounded by its limitations that are mainly caused by over-reliance on DSP resources, scalability problems when classes or data are intricate in nature, accuracy trade-offs with fixed kernel choices, inflexible adaptation to spectral bands, and ideal conditions' assumptions that overlook actual data transfer delays.

The proposed approach introduces a new hardware-software co-design strategy for hyperspectral image classification using sparse dictionary learning and FPGA-hardware accelerated SpMV within an SVM classification pipeline.The integration allows for the creation of compact discriminative features that take advantage of both spectral and spatial redundancies. By splitting computation tasks smartly performing computationally heavy SpMV tasks in hardware specialized FPGA chips with dynamic load balancing and the rest of the classification tasks in software the new strategy takes the advantages of both hardware acceleration and software programmability.

3. Proposed Methodology

The novel approach employs sparse dictionary learning, dynamic load balancing, and FPGA-accelerated SpMV to tackle the computationally demanding task of HSI classification as depicted in Algorithm 1. In our system, an HSI cube $\mathbf{H} \in \mathbb{R}^{H\times W\times S}$ of height H, width W, and S spectral bands is initially split into spatial patches and spectral blocks. The spatial patches, referred to as $SG_i \in \mathbb{R}^{m\times m\times S}$ for $i = 1, 2, \ldots, G$, are formed by dividing the image into non-overlapping $m \times m$-sized regions. Each patch is subsequently split along the spectral dimension into B equal blocks, with each spectral block $SG_{ij} \in \mathbb{R}^{m\times m\times \frac{S}{B}}$ covering a different subset of the spectral bands. This division decreases the data's dimensionality, making dictionary learning more manageable. The second operation is to build a training matrix $\mathbf{Y}_\mathrm{s} \in \mathbb{R}^{\frac{S}{B}\times(B\cdot N)}$ by vectorizing all the blocks of spectra extracted from the patches, where N is the number of pixels (or patches) involved. The objective is to learn a sub-dictionary $\mathbf{D} \in \mathbb{R}^{\frac{S}{B}\times k}$ such that the spectral blocks are represented sparsely. This is formulated as an optimization problem:

$$\min_{D,X_\mathrm{s}} \frac{1}{2} \|\mathbf{Y}_\mathrm{s} - \mathbf{D}X_\mathrm{s}\|_F^2 + \lambda R(X_\mathrm{s}) \tag{1}$$

therein, X_s represents the sparse coefficients, λ is the sparsity regularization parameter, and $R(\cdot)$ is a sparsity-inducing regularization function typically being the $l1$-norm. The following dictionary $\mathbf{D}$ compresses the most discriminative features of spectra simultaneously and suppresses redundancy. In order to further improve discrimination, each spectral block j is examined by calculating its variance σ_j^2. A block is labeled as "active" when its variance is larger than a given threshold T; otherwise, it is labeled as inactive. This selection is indicated in a diagonal selection matrix $\mathbf{W} \in \{0, 1\}^{B\times B}$ so that $W_{jj} = 1$ for active blocks and $W_{jj} = 0$ otherwise. The selective method guarantees that only the most discriminating spectral blocks are used for the sparse representation.

We determine the threshold T for active spectral blocks by performing cross-validation on validation subsets from each dataset. High spectral variance blocks need to be maintained because they hold the majority of discriminative features essential for classification. When T is set to a low value the system processes additional spectral blocks which enhances classification accuracy but demands more computational resources. With a high T value the system actively removes less informative bands which simplifies the process but risks a decrease in accuracy. Our experiments determined the optimal T value by finding a balance between performance and resource efficiency which varied slightly across datasets due to their unique spectral characteristics.

Sparse coding is applied to each active spectral block for each spatial patch SG_i. The problem solved for each block is expressed by

$$X_{ij} = \arg\min_{X} \frac{1}{2} \left\| SG_{ij} - \mathbf{D}X \right\|_F^2 + \lambda R(X) \tag{2}$$

and the resultant sparse coefficient vectors X_{ij} are concatenated to create the global feature vector X_i of the patch.

The final training matrix is formed by merging all feature vectors from the sparse modeling process in a row-wise manner. The matrix undergoes conversion to Compressed Sparse Row (CSR) format which enables efficient storage and processing by removing zero entries as shown in Fig. 1.

The transformation of a training matrix into CSR format captures only the non-zero elements of the sparse matrix to decrease memory requirements and enhance computational performance.The CSR format excels with sparse matrices by minimizing storage needs and maximizing memory access efficiency but remains unsuitable for non-sparse matrices. Dense feature representations might suffer more from CSR indexing overhead than they gain in benefits. Our system switches back to a basic dense matrix format and avoids CSR encoding when required to keep throughput stable. The proposed architecture maintains its efficiency across various data densities due to its adaptive capabilities.

The process requires three arrays: val contains non-zero matrix values while col_idx holds the indices of corresponding columns and row_ptr indicates the start of each row in val. The CSR format avoids storing zero entries to allow quick matrix-vector computations. The CSR representation of sparse coefficients offers substantial resource savings in hyperspectral image classification which makes it ideal for real-time embedded system applications including FPGAs and low-power processors.

The sparse matrix form a compact and discriminative feature representation of the hyperspectral data, which are then employed as inputs to the SVM classifier. SVM training is sped up with FPGA-based SpMV. Labeled feature pairs $\{X_i, y_i\}$ (with y_i as class labels) are input to an SVM solver [31]. The kernel computations $K(X_i, X_j)$ used in SVM training are recast as SpMV operations and transferred to an FPGA. The FPGA implementation uses a parallel architecture with dynamic load-balancing that keeps execution queues and redistributes tasks across available processing units to achieve high hardware utilization and low latency.

After the SVM has converged, the model parameters (α^*, b^*) are then retrieved. For classification, test patches are processed similarly X_i sparse coefficients are retrieved, and the decision function $f(X_i) = \sum_j \alpha_j K(X_i, X_j) + b^*$ is calculated using FPGA-accelerated SpMV, with final class labels produced. This whole solution not only facilitates efficient training on embedded hardware but also dynamically adjusts to new data, allowing real-time re-training capabilities not typically found in the current inference-only-based systems.

Algorithm 1. FPGA-accelerated HSI classification.

Input: Hyperspectral cube $\mathbf{H} \in \mathbb{R}^{h \times W \times S}$, label map L, parameters: m, B, k, λ, T

Output: Trained SVM model (α^*, b^*), class labels $\{\hat{y}_i\}$

Partition $\mathbf{H}$ into G spatial patches $SG_i \in \mathbb{R}^{m \times m \times S}$
for $i = 1$ to G **do**
  Divide SG_i into B spectral blocks $SG_{ij} \in \mathbb{R}^{m \times m \times \frac{S}{B}}$
end for
Form training matrix $\mathbf{Y}_s$ by vectorizing all spectral blocks
Train dictionary $\mathbf{D} \in \mathbb{R}^{\frac{S}{B} \times k}$:
  $\min_{\mathbf{D}, X_s} \frac{1}{2} \|Y_s - \mathbf{D}X_s\|_F^2 + \lambda R(X_s)$
for each spectral block j **do**
  Compute variance σ_j^2
  if $\sigma_j^2 < T$ **then**
    Mark block j as inactive
  end if
end for
Construct diagonal selection matrix $\mathbf{W} \in \{0, 1\}^{B \times B}$
for each patch SG_i **do**
  for each active block j **do**
    Solve sparse coding:
      $X_{ij} = \arg\min_X \frac{1}{2} \|SG_{ij} - \mathbf{D}X\|_F^2 + \lambda R(X)$
  end for
  Concatenate X_{ij} to form feature vector X_i
end for
Collect labeled training pairs $\{X_i, y_i\}$
Initialize SVM solver with kernel $K(\cdot, \cdot)$
while not converged **do**
  Identify kernel evaluations $K(X_i, X_j)$
  Map kernel evaluations to SpMV form
  Offload SpMV tasks to FPGA
  Perform dynamic load-balancing:
    Monitor queues, redistribute tasks to idle threads
  Retrieve results and update α, gradients
end while
for all test patches SG_i **do**
  Compute sparse feature X_i as above
  Evaluate decision function:
    $f(X_i) = \sum_j \alpha_j K(X_i, X_j) + b^*$
  Offload kernel computation to FPGA
  Assign label: $\hat{y}_i = \text{sign}(f(X_i))$
end for
return (α^*, b^*) and predicted labels $\{\hat{y}_i\}$

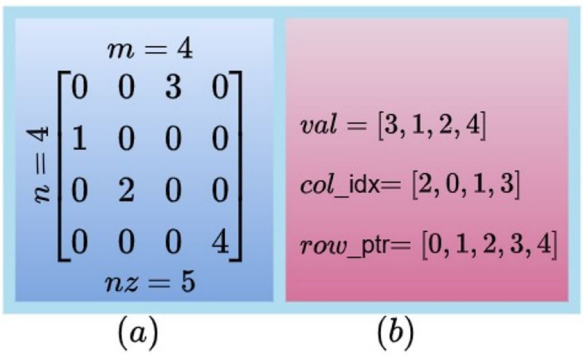

Fig. 1. (a) Example training matrix; (b) Sparse representation using Compressed Sparse Row (CSR) format.

4. SpMV Accelerator

4.1 Architecture Overview

Figure 2 displays the architecture design and dataflow of the proposed SpMV accelerator architecture. The system has multiple processing elements (PEs) functioning concurrently with a shared task dispatcher for dynamic load balancing along with on-chip vector caches and dedicated memory interfaces for matrix values and indices.

The accelerator uses the CSR data format and includes specialized hardware blocks for fast data retrieval and parallel processing. The design architecture diagram illustrates multiple parallel PEs each equipped to execute multiply-accumulate operations specifically for SpMV. Key components include:

- *Off-chip Memory Interfaces:* The non-zero data (*nz*) is delivered to all PEs at the same time through independent DDR memory ports assigned for matrix values and column indices. Every port maintains a width of 64 to 128 bits to support high-bandwidth burst transmissions. The dual-port system enables simultaneous streaming of CSR values and col_idx arrays to the PEs. The row pointer array (*row_ptr*) occupies less space and can thus be kept on-chip or loaded during burst operations.

- *BRAM Cache:* Storing input dense vector **x** in on-chip BRAM buffers reduces the random access frequency to DRAM. The complete vector or its portions are transferred to a multi-bank BRAM cache during the pre-computation phase. The system enables multiple PEs to access **x** elements quickly in parallel which prevents the need for repeated off-chip memory reads. Multiple banks partition the cache to enable parallel reads by each PE from different vector elements without causing contention. When the vector size exceeds the on-chip capacity the design loads segments of **x** into the chip and reuses them across multiple blocks of rows.

- *Processing Elements (PEs):* A pipelined unit called each PE processes individual rows one at a time. The PE accesses an uninterrupted section of values and *col_idx* arrays spanning *row_ptr*[*r*] to *row_ptr*[*r*+1]–1 to multiply each value by the matching **x** element. The products undergo accumulation into a partial sum dedicated to their row. The dedicated adder and multiplier units (single or double precision floating-point) with local registers for accumulation are available to every PE. Partitioning the matrix by rows allows each processing element to manage a dynamic subset of rows while utilizing parallelism on independent rows.

- *Task Dispatcher (Dynamic Load Balancer):* This new dynamic load balancing unit distributes rows to processing elements during operation to balance the workload. The dispatcher keeps a global task queue of row indices instead of statically assigning specific rows for computation to each PE. Once a PE completes its current row and becomes available, the dispatcher assigns it the next row index from the queue. The system guarantees that all PEs have tasks to process even when some rows contain significantly more non-zero elements than others.

- *Output Accumulation:* Every PE stores its finished calculation (one row's dot-product) into the output vector **y**. The absence of cross-PE reduction results from each PE being responsible for a complete row. The system can store results in an on-chip buffer or send them directly to DDR memory. The results are first stored in an output buffer within BRAM before being moved to DDR memory in burst transfers. Our design prevents random writes and enables coalesced memory access patterns for the result vector. Data is written back either as each row finishes or in bulk when all PEs have completed their tasks. The CSR generates one value for each row which means that write activity remains low when compared to read activity.

The accelerator functions as a streaming dataflow engine where the row dispatcher sends row indices to PEs and each PE fetches matrix values and indices for its row through burst reads from DDR and then sends out its results. The matrix data's memory access pattern remains highly sequential since it reads contiguous values and *col_index* subarrays which enables optimal DDR burst efficiency. On-chip dual-ported memory allows vector **x** to service access requests in a single cycle when it is accessed in quasi-random order based on *col_index*. This design strategy maintains computational unit activity by executing memory operations alongside computation through pipelined streaming techniques. The system mitigates SpMV's memory-bound limitations through multiple memory ports and on-chip caching that effectively reach theoretical bandwidth thresholds.

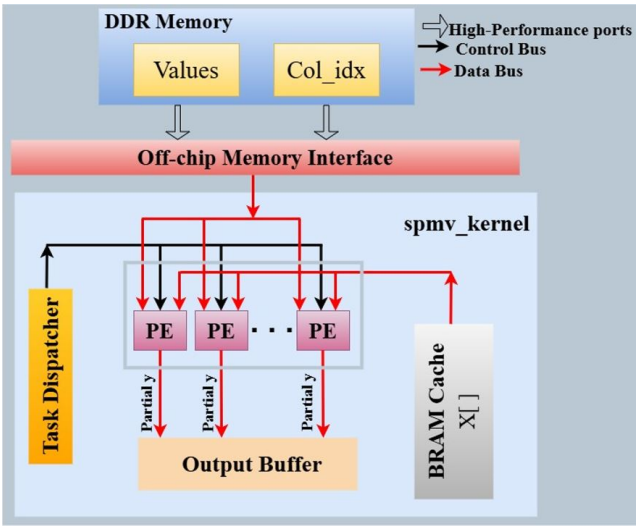

Fig. 2. Proposed SpMV accelerator architecture.

4.2 HLS Kernel Implementation

The HLS C++ kernel pseudocode for the SpMV accelerator is shown in Algorithm 2 as a simple representation. The code requires that the CSR arrays (*row_ptr, col_idx, values*) along with the input/output vectors connect through AXI-stream or memory-mapped interfaces. Multiple PE consumer processes fetch row indices from a task queue to achieve load balancing.

The HLS stream taskQueue stores row indices while the input vector is preloaded into the BRAM-stored local array *x_local*. The producer loop populates the queue with all row indices from 0 through *nrows*–1 before appending –1 tokens to indicate completion to each PE. We use loop unrolling to generate *PE_COUNT* parallel consumer tasks which instantiate multiple copies of the original computation loop. The processing engine (PE) enters a while-loop to retrieve the next row index from taskQueue. If the index is –1 it breaks out (all tasks done); otherwise it processes that row: The process begins by reading the pointers that mark the beginning and end of a row followed by iterating through each nonzero element performing multiplication and accumulation. The inner *row_loop* executes one multiply-add operation per clock cycle after the pipeline becomes full due to full pipelining. Each processing element achieves a computation rate of one nonzero operation per clock cycle after initial overhead for loop setup. The sum result is then stored in **y**[*row*]. PEs work independently on unique rows without requiring explicit synchronization because the hardware stream ensures each row assignment happens just once.

Algorithm 2. Parallel SpMV kernel (PE-based).

Inputs: $nrows, ncols, nz, row\_ptr[\,], col\_idx[\,], values[\,], x[\,]$
Output: **y**[ ] -resulting output vector after SpMV

Step 1: Prefetch Vector x into On-Chip Memory
for $j \leftarrow 0$ **to** $ncols - 1$ **do**
  $x\_local[j] \leftarrow x[j]$
end for
Step 2: Populate Task Queue with Row Indices
for $r \leftarrow 0$ **to** $nrows - 1$ **do**
  $taskQueue.enqueue(r)$
end for
Step 3: Add Termination Tokens for Each PE
for $p \leftarrow 0$ **to** $PE\_COUNT - 1$ **do**
  $taskQueue.enqueue(-1)$
end for
Step 4: Parallel Processing by PEs
for all processing elements p **in parallel do**
  while TRUE **do**
    $row \leftarrow taskQueue.dequeue()$
    if $row == -1$ **then**
      break
    end if
    $sum \leftarrow 0$
    $start \leftarrow row\_ptr[row]$
    $end \leftarrow row\_ptr[row+1]$
    for $j \leftarrow start$ **to** $end - 1$ **do**
      $col \leftarrow col\_idx[j]$
      $val \leftarrow values[j]$
      $sum \leftarrow sum + val \times x\_local[col]$
    end for
    $y[row] \leftarrow sum$
  end while
end for

We enhance SpMV performance on Artix-7 FPGAs using techniques such as loop pipelining with II=1 and loop unrolling along with dataflow streaming and burst memory transfers combined with memory partitioning double buffering and custom precision. The process of pipelining innermost loops reaches full capacity of the multiplier-adder pipeline with every cycle. PEs achieve thread-level parallelism along with dataflow techniques that synchronize producer-consumer loops for uninterrupted streaming. AXI4 bursts handle row-wise contiguous data efficiently. BRAM partitioning enables multiple PEs to read vector x data at the same time. The use of double buffering effectively conceals transfer delay when working with large vectors. Floating-point designs tailored for DSP applications meet necessary constraints by operating at feasible frequencies from 100 MHz to 150 MHz. These optimizations work cumulatively to maintain PE activity and boost system efficiency during memory-bound operations.

4.3 Dynamic Load Balancing

The dynamic load balancing Algorithm 3 proposed for FPGA-based SpMV manages computational workload by adaptively distributing tasks across parallel PEs.

The initial distribution of matrix rows to PEs utilizes non-zero counts to establish an approximate balance in workload distribution. The algorithm assesses each PEs workload during execution by tracking processed non-zeros or execution time and calculates the workload average for all PEs.

The algorithm redistributes the lightest row(s) from the overloaded PE to the least-loaded PE when the workload of any PE surpasses the average load by a predefined threshold chosen by the user. The workload balancing process repeats until every workload meets acceptable performance standards. The method achieves low overhead

Algorithm 3. Dynamic load balancing for SpMV.

Inputs: N rows, P processing elements (PE), $Row\_nz[N]$, $Threshold(Th), Interval$

1. Initialize: Evenly distribute rows to PEs based on Row_nnz
2. Perform SpMV computation in parallel across PEs
3. For every $Interval$ **iterations:**
  a. Monitor workload per PE
  b. Compute $AvgLoad \leftarrow \sum workloads/P$
  c. For each PE:
    i. If $workload(PE) > AvgLoad \times (1+Th)$:
      While $workload(PE) > AvgLoad \times (1+Th)$:
        Identify smallest row in overloaded PE
        Identify least-loaded PE
        Move smallest row to least-loaded PE
        Update workloads accordingly
4. Repeat steps 2-3 until computation is complete

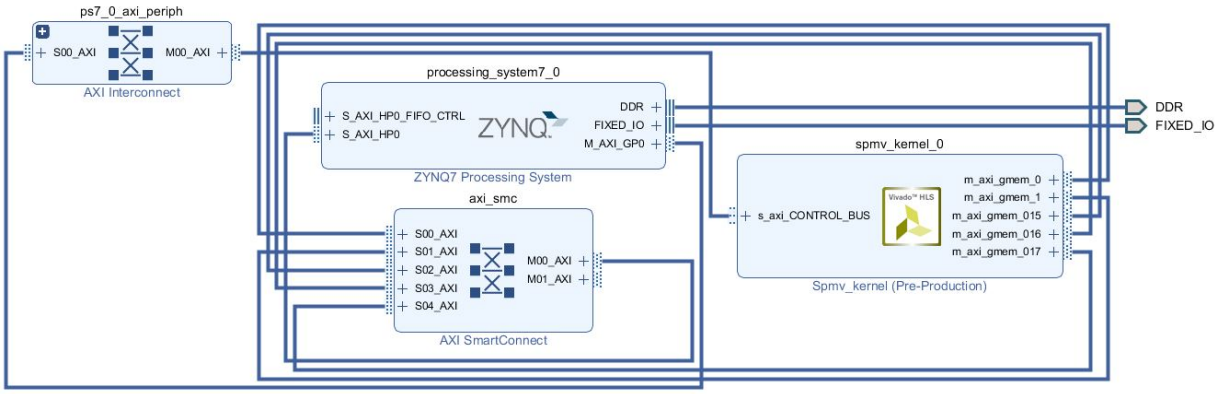

Fig. 3. Hardware/software co-design with direct DDR access.

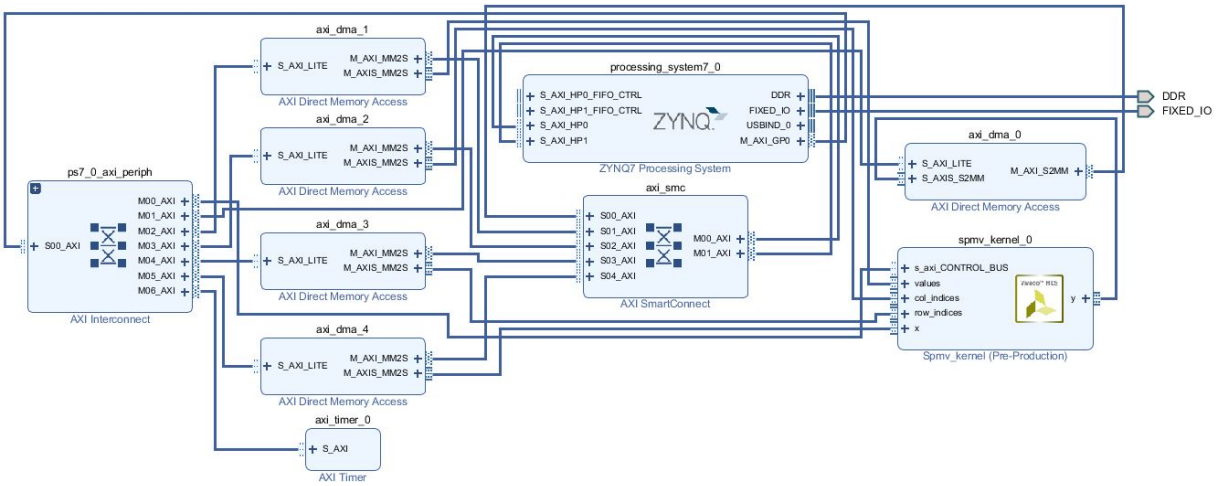

Fig. 4. Hardware/software co-design with AXI DMA integration.

by initiating re-balancing when required and concentrating on inexpensive row transfers which leads to better hardware utilization and fewer idle cycles. For matrices with highly irregular sparsity patterns this method demonstrates superior results compared to static partitioning by delivering higher performance and improved energy efficiency.

4.4 Hardware/Software Co-Design

The design of the hardware/software codesign system uses a Zynq-7000 Processing System (PS) which connects to a custom spmv_kernel Intellectual Property (IP) core as shown in Fig. 3.

The spmv_kernel uses gmem0 and gmem1 AXI master interfaces to connect to external DDR memory through an AXI SmartConnect for accelerating sparse matrix-vector multiplication. The system uses specialized interfaces for fast data movement between large matrix/vector datasets which reduces processing demands on the PS. The kernel uses an AXI-Lite slave implementation for its ap_ctrl interface which connects to one of the PS general-purpose master ports. Through this connection the PS can configure and control the kernel by writing to control registers and starting the processing tasks. An AXI Interconnect block facilitates correct routing and address mapping between multiple AXI interfaces and keeps the design synchronized by correctly applying clock and reset signals. The system architecture efficiently separates control and data paths to deliver high performance throughput and optimized hardware-software integration for intense computational tasks.

The modified block design combines AXI DMA engines with the Zynq PS and custom spmv_kernel IP to deliver improved data transfer performance over the previous design is shown in Fig. 4. This architecture uses AXI DMA blocks to move data in bulk between DDR memory and the spmv_kernel which results in diminished processor load while boosting concurrency levels. The DMA engines operate through direct memory access which enables them to handle large data buffers in DDR memory while simultaneously processing incoming and outgoing streams by the spmv_kernel. The AXI SmartConnect creates connections between multiple master and slave interfaces while maintaining proper address mapping and optimizing data flow. The architecture includes an AXI Timer module that measures performance metrics for DMA transfers and kernel execution with high precision.

The updated design shifts data transfer responsibilities to specialized hardware engines which lets the processing system handle additional tasks unlike the previous version where the spmv_kernel directly accessed DDR using its own AXI master ports. This architectural change results in superior system throughput and responsiveness especially for compute-demanding tasks such as sparse matrix-vector multiplication. AXI DMA module integration serves as an essential strategy for optimizing hardware-software collaboration and gaining high-performance acceleration.

5. Experimental Results

Our extensive experimental assessment of the SVM learning strategy was performed on the PYNQ-Z2 platform as shown in Fig. 5 that integrates the ZYNQ-7000 System on Chip with an ARM Cortex-A9 processor alongside an Artix-7 FPGA and 512 MB of DDR3 memory. Users can control hardware accelerators with Python and C++ through Jupyter notebooks because this platform enables hybrid CPU-FPGA execution. We selected the best hyperparameters for the MC-SVM model by using grid search and implemented fixed-point arithmetic to achieve efficient FPGA performance.

5.1 HSI Datasets

The proposed SVM learning strategy was tested on the ZYNQ SoC with four benchmark hyperspectral image datasets acquired through aerial imaging, UAV operations and terrestrial acquisitions as shown in Tab. 1. The datasets used in our study consist of subsets from Indian Pines [32] and AVIRIS-NG [33], which were captured by AVIRIS sensors and two additional datasets [34], [35] captured using Cubert UHD 185 sensors with one mounted on a UAV and another on a tripod. The datasets exhibit unique spectral ranges and class distributions which enable extensive evaluation of classification performance under different scenarios. All datasets used a consistent 30%–70% train-test split for conducting experiments.

5.2 Sparse Representation

Figure 6 visualizes dictionary atoms found through a dataset patch from Indian Pines showing spectral signatures created by our dictionary learning technique. Every line represents an individual learned atom which displays its

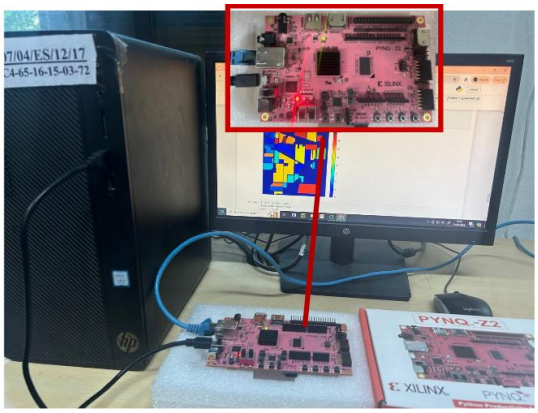

Fig. 5. Implementing SVM learning strategy on PYNQ-Z2.

Dataset	Platform	Classes	Spectral bands	Dimension
Indian Pines	Ariel	5	220	145×145
AVIRIS-NG	Ariel	5	351	400×565
Cubert-UAV	UAV	3	139	1000×1000
Cubert-Terrestrial	Terrestrial	3	139	1000×1000

Tab. 1. HSI datasets.

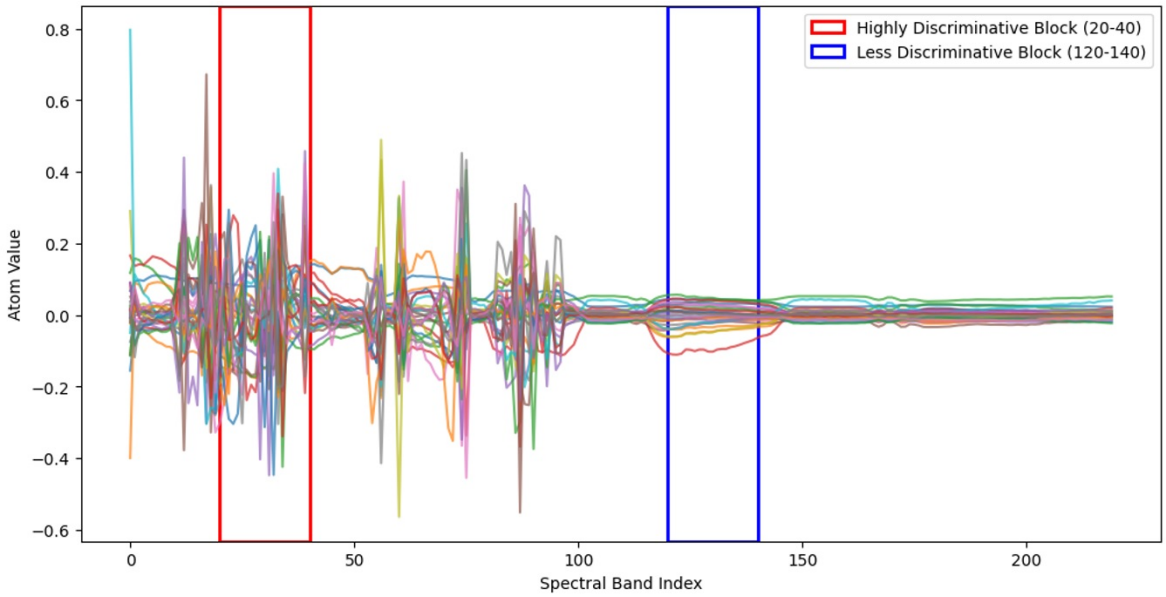

Fig. 6. Learned dictionary atoms for Indian Pines dataset.

distinct spectral signature. Spectral band indices are plotted on the x-axis and atom values are shown on the y-axis. The red shading between bands 20 and 40 marks an area with high discriminative power whereas the blue shading from bands 120 to 140 shows a region of lower discriminative strength. The analysis of these intervals demonstrates that specific spectral bands hold more valuable data for classification while showcasing the advantage of pinpointing and concentrating on discriminative spectral blocks. Thus, classification efficiency is enhanced.

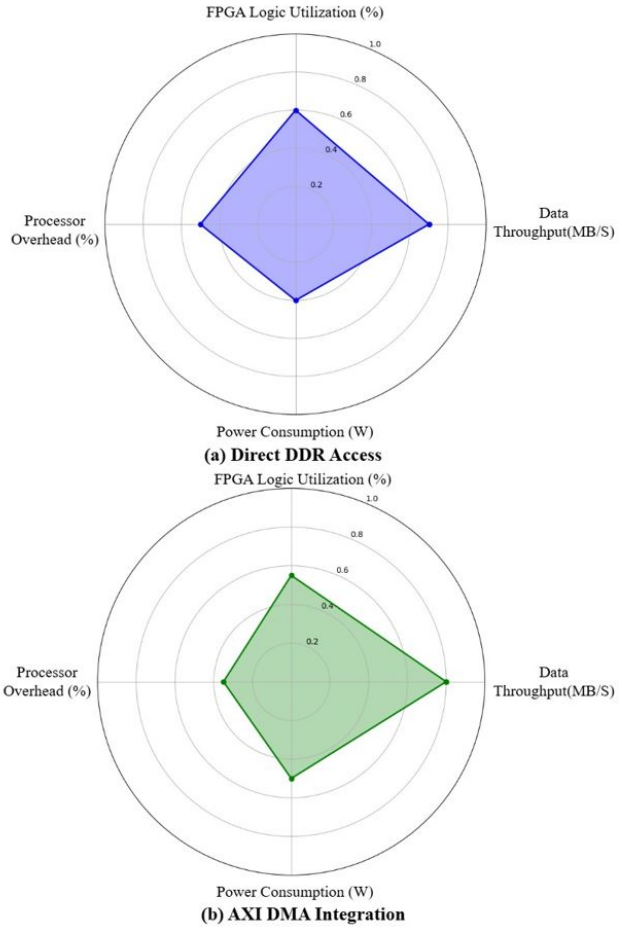

Fig. 7. Evaluating hardware/software co-design strategies.

5.3 Performance of Co-design Strategies

The study evaluated two distinct designs for accelerating SPMV computations during SVM training for hyperspectral image classification on a Zynq SoC platform. A comparative radar chart analysis in Fig. 7 shows that AXI DMA Integration design surpasses Direct DDR Access design across crucial performance indicators. Data throughput improved from 100 MB/s to 160 MB/s while processor overhead dropped significantly from 30% to 15%. Despite a slight increase in FPGA logic use from 40% to 45% and a small increase in power consumption from 0.028 W to 0.24 W the AXI DMA design shows better overall performance improvements in data handling and processing efficiency.

The processor overhead (%) measurement involves calculating the portion of processor cycles used by control operations compared to total cycles and power consumption (W) measurement relies on Xilinx Vivado Power Analyzer tool which used simulation-based techniques to capture realistic switching activity.

5.4 Inference Accuracy Assessment

Four HSI datasets served as evaluation benchmarks for testing the proposed SVM learning strategy on the PYNQ-Z2 platform. We evaluated inference and per-class accuracies through the ratio of correctly classified pixels. Table 2 demonstrates that the system obtained excellent classification accuracy for each dataset. The demonstrated results confirm that the hardware-accelerated SVM classifier functions reliably and is appropriate for real-time HSI applications.

5.5 Real-Time Performance Assessment

We implemented the new SVM learning strategy on the ZYNQ-7000 SoC where the Artix-7 FPGA served to speed up demanding SVM operations. The spmv_kernel IP block underwent HLS optimization to match dataset requirements based on pixel dimensionality and class count. The increase in resource consumption directly correlated with both feature dimensionality and the number of classes as depicted in Tab. 3. The operating frequencies mentioned here represent post place-and-route (implementation) values derived from Vivado tools. The design used multiple optimization

Indian Pines		**AVIRIS-NG**		**Cubert-UAV**		**Cubert-Terrestrial**	
Classes	**Per-class accuracy** [%]	**Classes**	**Per-class accuracy** [%]	**Classes**	**Per-class accuracy** [%]	**Classes**	**Per-class accuracy** [%]
Alfalfa	99.89	Water	99.65	Tomato	99.98	Pipe	95.78
Corn-notill	97.65	Soil	99.78	Eggplant	99.92	Cabbage	99.85
Soybean-mintill	93.56	Building	95.32	Cabbage	99.94	Soil	99.68
Wheat	98.79	Road	95.45	NA	NA	NA	NA
Woods	97.98	Vegetation	99.86	NA	NA	NA	NA
Inference accuracy [%]	**98.65**	**Inference accuracy** [%]	**98.92**	**Inference accuracy** [%]	**99.95**	**Inference accuracy** [%]	**99.84**

Tab. 2. Per-class accuracy and inference accuracy obtained using proposed learning strategy on ZYNQ SoC for HSI classification.

Resources	Available	Indian Pines utilization	AVIRIS-NG utilization	Cubert-UAV utilization	Cubert-Terrestrial utilization
LUTs	53200	34780 (65.38%)	34780 (65.38%)	43456 (81.68%)	43456 (81.68%)
Flip-Flops	106400	34613 (32.53%)	34613 (32.53%)	48562 (45.64%)	48562 (45.64%)
BRAMs	140	136 (96.79%)	136 (96.79%)	131 (93.57%)	131 (93.57%)
DSPs	220	124 (56.36%)	124 (56.36%)	29 (13.18%)	29 (13.18%)
Maximum operating frequency [MHz]	–	177.36	177.36	198.46	198.46

Tab. 3. Resource utilization for SVM learning strategy on ZYNQ SoC for HSI classification.

HSI datasets	Latency [µs]		Throughput		Inference period			Power [Watts]
	Learning	Classification	Pixel vectors per second	Data rate [MB/s]	Software [s]	Accelerator [s]	Speedup	
Indian Pines	5.54	1.26	147059	129	1.144	0.026	44	0.172
AVIRIS-NG	6.76	2.14	112359	158	12.98	0.483	26.87	0.238
Cubert-UAV	4.69	0.75	183823	102	27.36	0.75	36.48	0.028
Cubert-Terrestrial	4.69	0.75	183823	102	27.36	0.75	36.48	0.028

Tab. 4. Performance metrics of the proposed incremental strategy on the ZYNQ SoC for HSI classification.

Method	FPGA Device	Accuracy	Throughput	Latency	Power	Training ability
[3]	Artix-7	~73.8%	~9k pixels/s (single) ~54k pixels/s (6-core)	0.66 ms/pixel (single) 0.11 ms/pixel (6-core)	-	No
[4]	Kintex-7	~98.0%	1.3 Mpixels/s	-	-	No
[5]	Artix-7	~98.9%	~100k pixels/s	3.45 µs/pixel	~0.3 W	No
Proposed	Artix-7	~98.65%	~147k pixels/s	5.54 µs (learning) 1.26 µs (classification)	0.172 W	**Yes**

Tab. 5. Comparison of FPGA-based hyperspectral image classification methods for Indian Pines dataset.

strategies to obtain timing closure and control routing congestion during periods of extensive resource utilization. The introduction of numerous pipeline stages throughout PEs successfully minimized the length of critical paths. The implementation of streaming interfaces to organize data minimized internal congestion by improving data handling efficiency. Classification latency reached $r + 10$ clock cycles and learning latency reached $r + 792$ clock cycles which resulted in 6.8 µs and 8.9 µs respectively as shown in Tab. 4. These latencies were significantly below prior work sensor dwell times of 15.6–31.5 µs [36], [37].

The system achieved up to 183,823 pixel vectors per second and data rates above 150 MBPS which varied based on the dataset complexity. Our hardware-accelerated MC-SVM employs parallel PEs to concurrently run compute intensive SpMV unlike software implementations where SVM classifiers work sequentially. Parallel processing elements enabled the system to achieve an average speed improvement of 36 times compared to the standard software baseline on an ARM Cortex-A9 (650 MHz) processor. The AVIRIS-NG image with dimensions 400×565 achieved classification within 0.483 seconds which falls significantly below the sensor's 3.5-second acquisition time thereby demonstrating real-time performance capability. Power consumption stayed below 0.24 W for all configurations due to the number of processing elements used, pipeline depth, and kernel complexity. Power consumption measurements were collected with Xilinx Vivado Power Analyzer tool which used simulation-based techniques to capture realistic switching activity. The system implementation utilized optimized fixed-point arithmetic to effectively decrease DSP and logic usage overhead. Dual-port BRAM caches minimized off-chip memory accesses which led to a substantial reduction in dynamic power consumption. The efficient allocation of resources resulted in fewer idle cycles and less total energy usage. The experimental findings support the deployment of the design on power-constrained and latency-critical platforms such as UAVs, satellites and terrestrial imaging systems.

5.6 Comparison with State-of-the-Art

The classification of the Indian Pines dataset evaluates four FPGA-based methodologies in Tab. 5. Tajiri and Maruyama [4] achieved impressive throughput with their Kintex-7 device reaching 1.3 Mpixels/s but their work lacks detailed latency information. Gyaneshwar and Nidamanuri [5] provide superior accuracy levels (98.9%) together with reasonable throughput (100k pixels/s) while avoiding any on-chip training processes. Martins et al. Martins et al. [3] demonstrate a scalable FPGA design through Zynq-7000 devices yet attain an accuracy level of approximately 73.8%. The proposed solution achieves around 98.65% accuracy with a throughput of up to 180k pixels/s while keeping the on-chip power consumption below 0.24 W. The system uses on-chip learning to allow real-time model adaptation

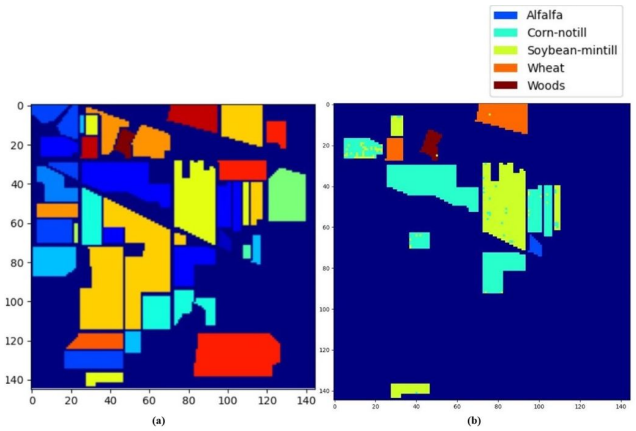

Fig. 8. Indian Pines: (a) Reference sample map; (b) Classification map.

without needing external retraining which makes it perfect for dynamic environments. The system minimizes data transfer overhead while improving classification accuracy and increases autonomous functioning in edge computing deployments with strict power and latency needs like UAVs and satellites.

Spatial distribution patterns of Alfalfa, Corn-notill, Soybean-mintill, Wheat, and Woods classes throughout the Indian Pines dataset are shown in the Fig. 8. The classification process targets only specific pixels associated with selected classes while masking all other areas to background. The use of different color segments demonstrates precise predictions that match ground truth boundaries for each specific region. The selective classification method maintains spatial context and minimizes computational requirements. Through visualization analysis the proposed model achieves high classification accuracy for specific classes and demonstrates strong spatial domain generalization which makes it ideal for real-time hyperspectral applications in resource-restricted environments.

6. Conclusions

The paper presents a new hyperspectral image classification framework which combines sparse dictionary learning with SVM training alongside FPGA-accelerated SpMV operations. Our method divides images into spatial patches and spectral blocks to learn compact sub-dictionaries that enable discriminative sparse feature extraction. The dynamic FPGA architecture with load-balancing capabilities speeds up SVM kernel evaluations which allows complete SVM training on embedded devices. Experimental results indicate that our system delivers better classification accuracy together with faster processing times and improved energy efficiency than existing systems. Upcoming research directions will target the development of advanced compression methods while working on fixed-point arithmetic and adaptive block formation technology and exploring expansion possibilities to multi-FPGA platforms. The framework shows outstanding capabilities in scalability along with robustness and efficiency performance.

References

[1] PATHAK, D. K., KALITA, S. K., BHATTACHARYA, D. K. et al. Hyperspectral image classification using support vector machine: a spectral spatial feature based approach. *Evolutionary Intelligence*, 2022, vol. 15, p. 1809–1823. DOI: 10.1007/s12065-021-00591-0

[2] MACIAS, R., BERNABE, S., BASCONES, D. et al. FPGA implementation of a hardware optimized automatic target detection and classification algorithm for hyperspectral image analysis. *IEEE Geoscience and Remote Sensing Letters*, 2022, vol. 19, p. 1–5. DOI: 10.1109/LGRS.2022.3189109

[3] MARTINS, L. A., VIEL, F., SEMAN, L. O. et al. A real-time SVM-based hardware accelerator for hyperspectral images classification in FPGA. *Microprocessors and Microsystems*, 2024, vol. 104, p. 1–13. DOI: 10.1016/j.micpro.2023.104998

[4] TAJIRI, K., MARUYAMA, T. FPGA acceleration of a composite kernel SVM for hyperspectral image classification. *IEEE Access*, 2023, vol. 11, p. 214–226. DOI: 10.1109/ACCESS.2022.3230066

[5] GYANESHWAR, D., NIDAMANURI, R. R. A real-time FPGA accelerated stream processing for hyperspectral image classification. *Geocarto International*, 2022, vol. 37, no. 1, p. 52–69. DOI: 10.1080/10106049.2020.1713231

[6] SHENMING, Q., XIANG, L., ZHIHUA, G. et al. A new hyperspectral image classification method based on spatial-spectral features. *Scientific Reports*, 2022, vol. 12, p. 1–15. DOI: 10.1038/s41598-022-05422-5

[7] KAUL, A., RAINA, S. Support vector machine versus convolutional neural network for hyperspectral image classification: a systematic review. *Concurrency and Computation: Practice and Experience*, 2022, vol. 34, p. 1–35. DOI: 10.1002/cpe.6945

[8] ZHENG, Z., ZHONG, Y., MA, A. et al. FPGA: Fast patch-free global learning framework for fully end-to-end hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2020, vol. 58, no. 8, p. 5612–5626. DOI: 10.1109/TGRS.2020.2967821

[9] NASCIMENTO, J., MARTIN, G. Nonlinear spectral unmixing. Chapter in: AMIGO, J. M. (ed.). *Data Handling in Science and Technology*, Elsevier, 2019, vol. 32, p. 151–166. ISBN: 9780444639776. DOI: 10.1016/B978-0-444-63977-6.00008-0

[10] RATLE, F., CAMPS-VALLS, G., WESTON, J. et al. Semisupervised neural networks for efficient hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2010, vol. 48, no. 5, p. 2271–2282. DOI: 10.1109/TGRS.2009.2037898

[11] FU, H., et al. A novel band selection and spatial noise reduction method for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2022, vol. 60, p. 1–13. DOI: 10.1109/TGRS.2022.3189015

[12] LI, F., WANG, J., LAN, R., et al. Hyperspectral image classification using multi-feature fusion. *Optics & Laser Technology*, 2019, vol. 110, p. 176–183. DOI: 10.1016/j.optlastec.2018.08.044

[13] FENG, S., ITOH, Y., PARENTE, M., et al. Hyperspectral band selection from statistical wavelet models. *IEEE Transactions on Geoscience and Remote Sensing*, 2017, vol. 55, no. 4, p. 2111–2123. DOI: 10.1109/TGRS.2016.2636850

[14] KUMAR, B., DIKSHIT, O. Hyperspectral image classification based on morphological profiles and decision fusion. *International Journal of Remote Sensing*, 2017, vol. 38, no. 20, p. 5830–5854. DOI: 10.1080/01431161.2017.1348636

[15] BHATTI, U. A., YU, Z., CHANUSSOT, J., et al. Local similarity-based spatial–spectral fusion hyperspectral image classification with deep CNN and Gabor filtering. *IEEE Transactions on Geoscience and Remote Sensing*, 2022, vol. 60, p. 1–15. DOI: 10.1109/TGRS.2021.3090410

[16] MEI, S., LI, X., LIU, X., et al. Hyperspectral image classification using attention-based bidirectional long short-term memory network. *IEEE Transactions on Geoscience and Remote Sensing*, 2022, vol. 60, p. 1–12. DOI: 10.1109/TGRS.2021.3102034

[17] WANG, H., CELIK, T. Sparse representation-based hyperspectral image classification. *Signal, Image and Video Processing*, 2018, vol. 12, p. 1009–1017. DOI: 10.1007/s11760-018-1249-1

[18] IORDACHE, M.-D., BIOUCAS-DIAS, J. M., PLAZA, A. Sparse unmixing of hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 2011, vol. 49, no. 6, p. 2014–2039. DOI: 10.1109/TGRS.2010.2098413

[19] CHEN, Y., NASRABADI, N. M., TRAN, T. D. Hyperspectral image classification using dictionary-based sparse representation. *IEEE Transactions on Geoscience and Remote Sensing*, 2011, vol. 49, no. 10, p. 3973–3985. DOI: 10.1109/TGRS.2011.2129595

[20] PENG, J., JIANG, X., CHEN, N., et al. Local adaptive joint sparse representation for hyperspectral image classification. *Neurocomputing*, 2019, vol. 334, p. 239–248. DOI: 10.1016/j.neucom.2019.01.034

[21] SOLTANI-FARANI, A., RABIEE, H. R., HOSSEINI, S. A. Spatial-aware dictionary learning for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2015, vol. 53, no. 1, p. 527–541. DOI: 10.1109/TGRS.2014.2325067

[22] FU, W., LI, S., FANG, L., et al. Contextual online dictionary learning for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2018, vol. 56, no. 3, p. 1336–1347. DOI: 10.1109/TGRS.2017.2761893

[23] XIE, M., JI, Z., ZHANG, G., et al. Mutually exclusive-KSVD: Learning a discriminative dictionary for hyperspectral image classification. *Neurocomputing*, 2018, vol. 315, p. 177–189. DOI: 10.1016/j.neucom.2018.07.015

[24] CASTRODAD, A., VASILESCU, M., SAPIRO, G., et al. Learning discriminative sparse representations for modeling, classification, and reconstruction of hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 2011, vol. 49, no. 11, p. 4263–4281. DOI: 10.1109/TGRS.2011.2159265

[25] CAMPS-VALLS, G., BRUZZONE, L. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2005, vol. 43, no. 6, p. 1351–1362. DOI: 10.1109/TGRS.2005.846154

[26] MELGANI, F., BRUZZONE, L. Classification of hyperspectral remote sensing images with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, 2004, vol. 42, no. 8, p. 1778–1790. DOI: 10.1109/TGRS.2004.831865

[27] GAO, S., TSANG, I. W.-H., CHIA, L.-T. Laplacian sparse coding, hypergraph Laplacian sparse coding, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, vol. 35, no. 1, p. 92–104. DOI: 10.1109/TPAMI.2012.63

[28] HUANG, J., HUANG, T., DENG, L., et al. Joint-sparse-blocks and low-rank representation for hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 2019, vol. 57, no. 4, p. 2419–2438. DOI: 10.1109/TGRS.2018.2873326

[29] FU, W., LI, S., FANG, L., et al. Hyperspectral image classification via shape-adaptive joint sparse representation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2015, vol. 9, no. 2, p. 556–567. DOI: 10.1109/JSTARS.2015.2477364

[30] HE, Z., LIU, L., DENG, R., et al. Low-rank group inspired dictionary learning for hyperspectral image classification. *Signal Processing*, 2016, vol. 120, p. 209–221. DOI: 10.1016/j.sigpro.2015.09.004

[31] CHANG, C.-C., LIN, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011, vol. 2, no. 3, p. 1–27. DOI: 10.1145/1961189.1961199

[32] BAUMGARDNER, M. F., BIEHL, L. L., LANDGREBE, D. A. 220 *Band AVIRIS Hyperspectral Image Data Set: June 12, 1992 Indian Pine Test Site 3*. Purdue University Research Repository, 2015. DOI: 10.4231/R7RX991C

[33] BHATTACHARYA, B. K., JAIN, S., PARIHAR, J. S., et al. An overview of AVIRIS-NG airborne hyperspectral science campaign over India. *Current Science*, 2019, vol. 116, no. 7, p. 1082–1088. DOI: 10.18520/CS/V116/I7/1082-1088

[34] ASTOR, T., DAYANANDA, S., NAUTIYAL, S., et al. Vegetable crop biomass estimation using hyperspectral and RGB 3D UAV data. *Agronomy*, 2020, vol. 10, no. 10, p. 1–19. DOI: 10.3390/agronomy10101600

[35] SARMA, A. S., NIDAMANURI, R. R. Active learning-enhanced plant-level crop mapping with drone hyperspectral imaging and evolutionary computing. In *Proceedings of the Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. Athens (Greece), 2023, p. 1–5. DOI: 10.1109/WHISPERS61460.2023.10430799

[36] WU, Z., LIU, J., PLAZA, A., et al. GPU implementation of composite kernels for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 2015, vol. 12, no. 9, p. 1973–1977. DOI: 10.1109/LGRS.2015.2441631

[37] THORPE, A. K., ROBERTS, D. A., FRANKLIN, J., et al. Mapping methane concentrations from a controlled release experiment using the next generation airborne visible/infrared imaging spectrometer (AVIRIS-NG). *Remote Sensing of Environment*, 2016, vol. 179, p. 104–115. DOI: 10.1016/j.rse.2016.03.032

About the Authors . . .

B. B. SHABARINATH graduated from SASTRA University, Thanjavur, India, with a B. Tech in Electronics and Communication Engineering in 2009, and from AMITY University, Noida, India, with an M. Tech in Embedded Systems Technology in 2013. Presently engaged in the pursuit of a Ph.D. at the National Institute of Technology Warangal, he also holds the position of Assistant Professor at the VNR Vignana Jyothi Institute of Engineering and Technology in Hyderabad. His research interests include developing hardware accelerators for Machine Learning Algorithms.

Pullakandam MURALIDHAR received B.Tech in Electronics and Communication Engineering and M.Tech degree in Electronic Instrumentation from National Institute of Technology, Warangal, India, in 1993 and 2004 respectively. Then he has received his Ph.D. degree from NIT Warangal. He joined Apollo Computing Labs Hyderabad in 1994 where he was engaged in design and development of high-speed digital circuits. He joined NIT Warangal in 1997. Since then he has been working in the ECE Department, NIT Warangal. His research interests include design of Embedded Systems and VLSI Architectures for Machine Learning and Video Processing Systems.