Spatial–Spectral Feature Extraction With Local Covariance Matrix From Hyperspectral Images Through Hybrid Parallelization

Emanuele Torti, Elisa Marenzi, Giovanni Danese, Antonio J. Plaza, *Fellow, IEEE*, and Francesco Leporati

***Abstract*—This article presents the optimization and hybrid parallelization of a spatial–spectral feature extraction (FE) method from hyperspectral images (HSIs) using local covariance matrix (CM) representation, exploiting hybrid parallelism through multicore and manycore processors. The aim is to evaluate the performance of parallel versions of this innovative algorithm that characterizes spatial–spectral information prior to classification when conducting FE. The HSI is first projected into a subspace, using the maximum noise fraction method. Then, for each test pixel, its most similar neighbors are clustered using the cosine distance measurement. The result is used to calculate a local CM with each nondiagonal entry characterizing the correlation between different spectral bands. Such matrices represent the spatial–spectral features and are fed to a support vector machine for classification. To optimize the successive parallelization process, a new version of the original MATLAB code has been first developed using C language. This serial version serves as baseline for hybrid parallelization in OpenMP and CUDA. Performance analysis has been conducted using publicly available HSI datasets, confirming that our parallel implementation ensures the quality of the classification while significantly reducing the involved processing times.**

***Index Terms*—Covariance matrix (CM), feature extraction (FE), graphic processing unit (GPU), hyperspectral imaging (HSI), parallelization.**

I. Introduction

HYPERSPECTRAL image (HSI) classification plays a crucial role in various application fields. However, when the training samples available a priori are limited, classification accuracy can significantly degrade due to multiple causes. To address this problem, feature extraction (FE) methods, which include supervised and unsupervised approaches, have been proposed. As a consequence, characterization of the spatial–spectral information in HSI prior to classification is a fundamental task in terms of efficiency [1]. In the case of limited availability of training samples, the challenge is to find new methods that can outperform state-of-the-art techniques.

HSIs are made of hundreds of spectral bands [1], where each pixel is a high-dimensional vector (also defined as spectral signature) providing spectral information that can be used to distinguish between different materials [2]. Single pixels carry information from across the electromagnetic spectrum of an image; the associated spectral signature is formed by many contiguous wavelengths, which generally cover the visible and infrared parts of such spectrum [3]. Their main advantage is the ability to collect these unique “fingerprints” that can be later used for classification purposes. Specifically, HSIs enable the identification of the materials that compose a specific object, providing a better understanding of the scene under analysis. For this reason, they have been widely used in many applications, such as target and change detection [4], [5], image enhancement [6], and classification [7].

Nevertheless, the huge amount of data collected needs to be processed and analyzed, typically using complex algorithms with a heavy computational burden, reducing the amount of time required as much as possible. This fact often prevents the use of HSI-based systems in applications with real-time or near real-time constraints. Parallel processing using parallel hardware devices, such as field programmable gate arrays (FPGAs) and graphic processing units (GPUs), offers a possible solution for this problem. For example, many works describe the implementation of the entire unmixing chain into GPUs [8], [9], [10], [11], [12], providing the fastest results with independence of the size and content of the HSI and the characteristics of the used GPU architecture [12].

This work presents the optimization and hybrid parallelization of an innovative method to simultaneously exploit the correlation among different spectral bands and the spatial information in the HSI, allowing for a more discriminative FE process, through the concept of local covariance matrix representation (LCMR).

This article is organized as follows. Section II provides a review of relevant related works. Section III presents the algorithm for spatial–spectral FE with LCMR and support vector machine (SVM) classification, which we have considered for parallelization purposes. Section IV presents the code optimization from the original MATLAB algorithm to the serial C language version, together with parallelization techniques for this method using OpenMP paradigm and CUDA language. Section V discusses the conducted performance analysis, which

Manuscript received 6 April 2023; revised 14 July 2023; accepted 30 July 2023. Date of publication 3 August 2023; date of current version 15 August 2023. *(Corresponding author: Elisa Marenzi.)*

Emanuele Torti, Elisa Marenzi, Giovanni Danese, and Francesco Leporati are with the Department of Electrical, Computer and Biomedical Engineering, University of Pavia, 27100 Pavia, Italy (e-mail: emanuele.torti@unipv.it; elisa.marenzi@unipv.it; giovanni.danese@unipv.it; francesco.leporati@unipv.it).

Antonio J. Plaza is with the Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, E-10071 Cáceres, Spain (e-mail: aplaza@unex.es).

Digital Object Identifier 10.1109/JSTARS.2023.3301721

compares serial and parallelized versions with the same publicly available HSI images (standard test datasets used in the remote-sensing field). Finally, Section VI concludes this article.

II. Related Work

A. Parallelization Challenges

HSIs are widely used in a variety of fields, due to their richness of information, such as image processing [5], [6], [7], [8], [9], [10], [11], [12], [13], precision agriculture [14], urban mapping [15], and environmental monitoring [16].

Hyperspectral sensors acquire scenes at different wavelengths of the electromagnetic spectrum and they produce a cube [17]. Contiguous wavelengths are grouped into bands, and each pixel is represented by three components (x, y, z), where x and y denote the two-dimensional spatial position, whereas z identifies the band. Due to its high memory size (that can range from hundreds of megabytes up to many gigabytes) and the complexity of the algorithms used, HSI analysis is characterized by high computational requirements. Therefore, different approaches have been designed to reduce information redundancy, caused by the high correlation between adjacent bands of the HSI [18]. A common strategy is dimensionality reduction (DR), which produces a new compact dataset by applying a transformation based on a suitable criterion. Another approach is the band selection (BS), which can be divided into supervised and unsupervised BS [19], [20], [21]. Newer methods are based on the distance covariance descriptor for exploring spectral–spatial relationships, especially linear and nonlinear interdependence in the spectral domain, or on the principal component analysis (PCA) combined with the 2-D singular spectral analysis [22], [23].

An important aspect in HSI processing is that pixel resolution is generally not fine enough to separate different endmembers [4]. As a consequence, there is the need to develop real-time implementations of spectral unmixing techniques [24].

A possible downside is the processing of huge amounts of data, typically using complex algorithms with a considerable computational burden. This can limit their use in applications under real-time or near real-time constraints. Parallel processing using appropriate hardware devices offers a possible solution [3].

B. Available Parallelization Techniques

To accelerate processing speed and reduce storage for the classification, a series of parallel and distributed-based approaches have been proposed [25], [26].

Anomaly detection is another field where recent advances require parallel and distributed systems for efficient implementations [27].

This poses issues for their applications in real-time contexts without the aid of proper high-performance computing (HPC) techniques and technologies [28]. More specifically, the latter have been used in many works for various stages of the unmixing process because of their speed [29], [30], whereas GPUs are able to implement the entire unmixing chain with optimal results, independent to the specific HSIs and the device model involved [8], [9], [10], [11], [12]. Other approaches make use of OpenCL (Open Computing Language), a high-level design language that can efficiently parallelize code on multiple types of devices [12].

Moreover, in recent years, cloud computing has emerged as the new paradigm for many remote-sensing parallelization and optimization applications. In fact, a cloud-based technique has been developed for spectral analysis and compression of HSIs of the autoencoder deep neural network for nonlinear data compression [31]. In addition, cloud computing is used for deep learning inference at FE, metric mapping, and semantic segmentation [32].

In specific remote-sensing contexts, especially in onboard scenarios where high computational complexity algorithms are extremely useful though demanding, low power consumption HPC architectures are promising solutions [33].

1) Hybrid Parallelization: Hybrid parallelism is a form of parallelization technique that makes use of different languages or methodologies, which are combined together in a specific application or approach.

Recently, a combination of model and data parallelization methods has been proposed to minimize communication overhead in multidevice parallel training of DNN models. Moreover, a hybrid CPU-GPU real-time implementation of the unmixing chain has been proposed that makes use of GPU's CUDA CuBLAS library for linear algebra routines, together with OpenMP and BLAS for multicore parallelization [10]. In fact, performance of existing parallelization methods can still be improved by optimally allocating model computations and data partitions to the devices for better model training performance [34].

In this work, a hybrid parallelization approach has been implemented combining two parallel programming languages: OpenMP and CUDA for GPU.

2) OpenMP: OpenMP is a standard application programming interface for shared memory programming [35]. It comprises compiler directives, library routines, and environment variables that make this an easy parallel and portable programming model [36].

3) CUDA: NVIDIA's Compute Unified Device Architecture (CUDA) is a coevolved hardware–software architecture that enables the development of massively parallel programs with GPUs providing a development environment using C programming language [37], [38]. Threads run on streaming processors, each executing the same portion of code in single-instruction-multiple-thread fashion [39].

GPUs can be understood as an array of independent processors. Each of these processors corresponds with an independent thread of execution. The parallelism that can be inferred in this architecture totally differs from the FPGA architecture parallelism. Hence, the parallelism inferred in the GPU can be efficiently exploited when there is a situation in which a set of operations is to be independently performed to many different elements of a dataset.

The GPU models used in this work are the Tesla K40c active and the GeForce RTX 2080: this kind of accelerators can process datasets that have twice the dimensions of previous models, since they have an integrated memory of 12 GB of RAM. This family of devices can have up to ten times the performance of CPUs thanks to the GPUBoost functionality, by converting available power in performance increase, which is directly managed by the user. For the Tesla model, the main characteristics are a memory bandwidth of 288 GB/s with 2880 CUDA cores and a maximum power consumption of 235 W; the system interface is a PCI Express 3.0 × 16 with a clock frequency of 875 MHz.

In the case of the RTX device, the bandwidth is 448 GB/s with 2944 CUDA cores with a clock frequency of 1.8 GHz and a maximum power consumption of 225 W; the system interface is a PCI Express 3.0 × 16.

C. Classification Methods

HSI classification is one of the main challenges regarding their processing. Various classifiers have been adopted, including SVMs [40], neural networks [41], random forests [42], and sparse representation techniques [43]. However, when a priori training samples are limited, the accuracy [40], [41], [42] can significantly degrade due to the Hughes effect. To find a subspace in which the separability among the transformed samples can be enlarged, many FE methods have been developed, which can be both supervised (e.g., linear discriminant analysis [44]) and unsupervised. Regarding the latter, typical approaches are PCA [45], independent component analysis [46], and maximum noise fraction (MNF) [47].

More specifically, SVMs represent an important classifier in this context. Many works have benefited from this approach since the SVMs can deal with small-sized training datasets and provide higher classification accuracy than other traditional methods. Besides, SVMs have high memory efficiency and strong generalization [48].

A convolutional neural network (CNN) based on one-dimensional SVM convolution operations has been proposed to perform pixel-based classification with one-dimensional HSI signatures, analyzing each pixel spectrum independently from the pixel spatial neighborhood [49].

Another application concerned a novel PCA and segmented-PCA (SPCA) based multiscale 2-D-singular spectrum analysis (2-D-SSA) fusion method for joint spectral–spatial HSI FE and classification [50]. PCA and SPCA are used for spectral dimension reduction, whereas multiscale 2-D-SSA extracts abundant spatial features at different scales and is followed by PCA for DR. The obtained multiscale spatial features are then fused to form multiscale spectral–spatial features, whose performance is evaluated using the SVM classifier.

D. Spatial–Spectral FE and Covariance Matrix (CM) Approaches

Spatial and spectral information are used together to achieve superior classification performance. A recent example implemented Gabor filtering for spatial FE in conjunction with sparse random projections for spectral FE and DR. Finally, a supervised classification with a 3-D CNN performed volumetric data analysis [51].

Many spatial–spectral FE techniques have been proposed in recent years, such as the extended morphological profile, which extracts spatial information based on the structure of HSIs using morphological operations [52]. Based on the assumption that there is a strong correlation among neighboring pixels, various works use edge-preserving filtering to combine spatial and spectral information [53], [54].

In addition, deep learning algorithms have been used, such as CNNs, due to their success in the computer vision community [55], [56], [57], [58]. In all these cases, usually a DR method is first applied. The resulting bands are therefore processed separately by the filters or feature descriptors. However, the main drawback of deep models is the need of large labeled datasets to be successfully trained.

Another approach when using a limited number of training samples is based on the SuperPixel-based multiple statistical FE [59]. For each dimension-reduced pixel obtained by MNF, the most similar superpixel-based neighbors of different sizes are highlighted based on the spatial structures of the HSIs to exploit contextual spatial information. Then, the mean, covariance descriptor, and a Gaussian feature are extracted to explore spatial geometry information, correlations between spectral bands, and spatial–spectral variations. Next, multiple kernels map these features in the Euclidean and Riemannian manifold spaces (MSs) to a uniform Hilbert space and embed them into a multitask kernelized sparse representation classification model, effectively fusing the features for classification performance and robustness.

LCMR is an effective feature representation method that can make use of the correlation between different features and is applied for fusion of spatial and spectral data [41], [42]. Each nondiagonal element represents the covariance between two features. The aim is to obtain discriminative features of different pixels by computing CMR from the local neighborhood determined by a sliding window.

III. Algorithm Considered for Parallelization

In this article, we present a parallelization of a spatial–spectral FE method for HSIs using local CM representation, to highlight correlation between spectral bands and their spatial-contextual information.

To exploit this method, a novel LCMR approach is proposed, enabling a more discriminative FE process.

Here, the first step is the application of the MNF-based DR to the input HSI, useful to reduce the computational complexity and to remove noise. Its aim is to identify a linear transformation matrix to maximize the SNR.

After that, for each test pixel of the image, the $K-1$ most similar neighboring pixels are measured using cosine distance. A window with size $T \times T$ is used to extract the neighbors of a central pixel; then, the cosine-distance-based K-NN filter is computed on these pixels. Such local neighbors are both spatially and spectrally close to the reference pixel. This approach is used because of its simplicity and effectiveness, especially for classification purposes [60], [61]. In addition, it does not need careful parameter setting or heavy computational burden [43], [62]. The next step is the CM calculation for the K pixels, the test pixel, and its neighboring ones. Each nondiagonal entry of the CM represents the correlation among two spectral bands. Finally, such covariance matrices are given as input to an SVM as spatial–spectral features, for the final classification and label assignment with the Log-Euclidean-based kernel, since the covariance matrices lie on the Riemannian MS and not on the Euclidean space. In fact, a regularization is necessary to serve as

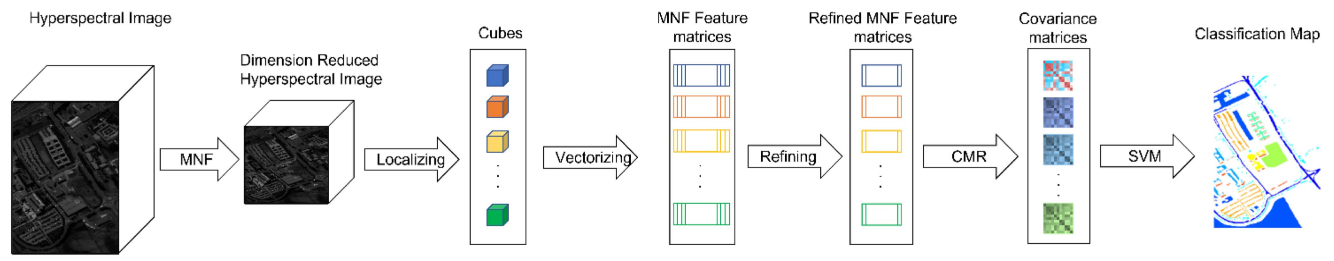


Fig. 1. Flowchart of the LCMR approach, showing all the main steps [1].

Algorithm 1: Original Classification Algorithm.

```
 1: Load HSI
 2: Define and initialize parameters: labels, number of
    classes, window_size, K, number of training iterations
 3: MNF for dimensionality reduction
 4: if FE not done:
 5:     LCMR
 6: else
 7:     Load the FE HSI
 8: end
 9: for i from 0 to n_training_iterations:
10:     Generate the sample image
11:     Set variables for training and test sets
12:     Calculate covariance matrix
13:     Apply SVM for classification
14:     Calculate error matrix
15: end
16: Build the classification matrix and print as output
```

input for learning algorithms. Fig. 1 shows the following main consecutive steps of the considered classification method: 1) MNF-based DR; 2) cubes localization; 3) cubes vectorization resulting in MNF feature matrices; 4) KNN-based neighboring pixel refining; 5) local CM calculation; and 6) kernel-based SVM for label assignment.

The SVM is executed using the LIBSVM library [63] adopting a Gaussian kernel. In this case, the number of MNF PCs is the same as in the case of LCMR.

The pseudocode in Algorithm 1 presents the structure of the algorithm: after loading the HSI to be processed, all parameters are defined and initialized. This is followed by the MNF and by the LSMR if the FE has not been done already, otherwise the processed image is loaded for the successive steps.

During training, the sample image is generated together with setting all variables for test and training sets; hence, the CM is calculated, followed by the SVM; finally, the error matrix is obtained.

The output is represented by the classification matrix.

IV. Code Optimization and Parallelization

A. Optimization in C Language

The original algorithm was developed in MATLAB and evaluated using a set of synthetic images that represent standard HSI testbed. It defines 9 classes as final output, 25 as the window size, 102 as the K-NN parameter, and 36 samples used for training. Usually, a good practice is to have a number of samples that is approximately 5 times the number of labeled pixels in the classes. In this case, an appropriate compromise is to have 4 samples per class, as presented also in [1].

The first step is to perform DR by applying the MNF algorithm to the image.

Since it is done multiple times, the next step is to check whether FE has already been applied to the image: if so, the processed image is loaded, otherwise the LCMR method is performed taking into account both previously defined parameters and the result of the MNF. Successively, the sample image is generated to train the algorithm. The outputs, which are the test and training ids, labels, covariance, and K-NN matrices, are then used as input for the classification with SVM. To highlight the classified pixels, different colors have been used in order to have a clearer representation of the classification map.

Since the aim of this work is to evaluate the performance of a parallelization using multicore and manycore approaches, the first step of the optimization is to convert the original MATLAB code into a serial C algorithm. This is very useful because both OpenMP and CUDA languages are written on a C code base.

Therefore, attention has been paid to maintain all the elements of the original approach: DR, covariance calculus, training and test of SVM, and errors processing. The algorithm initializes all parameters and calculates MNF and LCMR; then, memory is allocated for all variables and parameters in order to apply SVM.

B. Proposed Hybrid Parallelization

The presented approach is suitable for the use of the *libsvm* library; therefore, this has been integrated into the algorithm as the classification step. It is preceded by DR with the MNF approach, followed by the LCMR. This choice has been made because this function is already optimized in terms of performance and accuracy of results. After that, profiling is performed.

This has also influenced the hybrid parallelization process, combining parts with OpenMP and other portions of code using CUDA language on GPU.

In particular, the SVM classification has not been subject of parallelization, since this is not the most burdensome part of the computation, whereas the best results can be obtained by parallelizing all the matrices calculations, which make up about 80% of the whole methodology. Moreover, the training of an SVM model is not a task that can be efficiently implemented on GPUs, since the obtained acceleration is very low [64]. The sample generation has been parallelized using OpenMP, together with the error calculus after applying SVM: in fact, the entire function *errorMatrixGeneration*, which is the first to be applied, is composed of a *for* loop with many operations to be done and this can be efficiently parallelized. Following this part, accuracy estimation has been kept in the serial C version. On the contrary, the training part of the algorithm has been optimized using CUDA on GPU. *cuBLAS* library has been adopted, since it allows faster linear algebra computations, whereas the rest of the training phase has been parallelized with OpenMP.

The pseudocode in Algorithm 2 shows the hybrid parallelization approach: after all parameters and variables initialization, the HSI is loaded and memory is allocated for all matrices involved in the algorithm.

If the FE has not been done previously, the minimum noise fraction, followed by the LCMR, is performed; otherwise, the already processed image is loaded.

The successive step is to first allocate memory in the host CPU for all data involved in the learning and classification phases and then allocate memory in the device GPU for data involved in the training phase, since this is the part that is parallelized with CUDA. Hence, data are copied from host to device memory and grid and blocks are initialized for the execution of the GPU kernel for the training. The sample image is generated in parallel, by creating a matrix containing samples for each class. The threads copy from the input images a subset of the spectral pixels ensuring that the classes are balanced. Here, the *cuBLAS* library has been adopted because it is a standard library for arithmetic operations. To use this library, a special variable called *handle* is declared and initialized. This variable stores all information need by the library to perform the operation and is given as first input parameter to all the *cuBLAS* functions. In particular, the *cublasDgemm* routine has been adopted. It computes the standard matrix–matrix product, which can be used to estimate the CM. This routine takes as input two matrices already stored in the GPU memory and performs the multiplication, saving the resulting matrix in the GPU memory. It also allows to automatically transpose the input matrices, if needed by the computation. Moreover, it adopts the column-major matrix storage format. This is not the standard data layout for the C language, which adopts the row-major format. This issue is solved by exploiting the property of the matrix–matrix multiplication and transposing the first input matrix.

The entire training is performed in CUDA, generating the sample image, applying covariance, followed by copying the output data back from device GPU to host CPU, since the *cublasDgemm* library stores the results only in the GPU memory space.

OpenMP parallelization is used to calculate probability values for the classification, which is not parallelized on the GPU (since it is not the most demanding computational phase in this specific case), together with the SVM prediction step. In particular, the #pragma omp parallel for directive has been exploited. It distributes the different *for* loop iterations among the parallel threads. Moreover, it accepts clauses to manage data partitions and load balancing. The shared and private clauses indicate which data are private to a single thread and the information shared among all threads, respectively. In the proposed parallel implementation, the loop variables, such as the indices, are kept private to the single thread while the covariance matrices are shared among the threads. Concerning the load balancing, it is managed by the schedule clause. OpenMP gives five possible scheduling modes, namely static, dynamic, guided, auto, and runtime. In the static mode, the iterations are equally distributed among the threads. On the other hands, the dynamic scheduler iterations are organized in chunks and each chunk is assigned to a thread. The chunks are assigned to the threads following the "first-come first-do" approach. The guided mode is a special case of the dynamic scheduler in which the chunk size is not fixed but it is exponentially decreasing. The auto mode and the runtime mode demand to the operating system the choice of the scheduling approach. In this case, after several experiments, we choose the dynamic scheduling, since it is the one with the lowest processing times. Fig. 2 summarizes the hybrid parallelization approach, highlighting the data transfers between host and device. The blue dashed arrow denotes the data transfer from host to device, whereas the red dashed arrow indicates the copy from device to host.

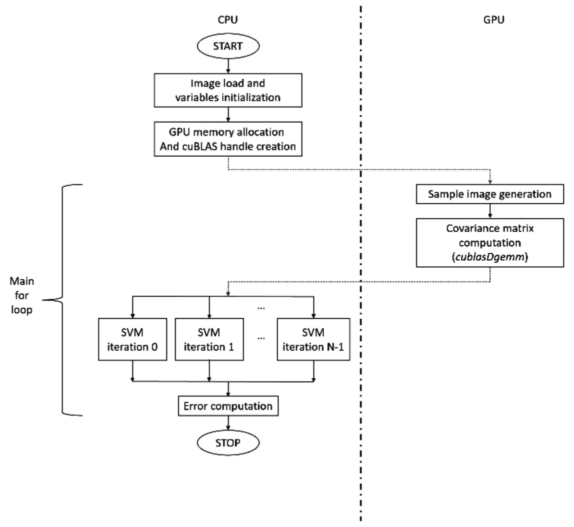

CPU
GPU
START
Image load and variables initialization
GPU memory allocation And cuBLAS handle creation
Sample image generation
Covariance matrix computation (cublasDgemm)
Main for loop
SVM iteration 0
SVM iteration 1
...
SVM iteration N-1
Error computation
STOP

Fig. 2. Summary of the proposed hybrid parallelization strategy. The vertical black dashed line represents the separation of the address spaces of CPU and GPU. The blue dashed arrow represents the memory copy from host to device, whereas the red dashed arrow denotes the data copy from device to host.

The error matrix is calculated without parallelization and, at last, all memory allocations in the CPU are freed.

The sample generation code has been parallelized using OpenMP: for each class, regarding the test set, the id, label, and class index initializations have been parallelized. After that, the training data are set and the sample is generated, as shown in Algorithm 3.

Also, the error matrix calculation has been performed with OpenMP (see Algorithm 4); after loading memory, parameters, and variables, the main cycle for generating the matrix is parallelized. Finally, the accuracy is evaluated.

V. Discussion

To analyze the performance of the parallelized algorithm and evaluate the differences with the serial version, a set of real HSIs, which are widely used in the remote-sensing field, have been adopted.

Algorithm 2: Hybrid Parallelization Approach.

```
 1:  Parameters and variables initialization
 2:  Read HSI
 3:  Host memory allocation for all matrices involved
 4:  if FE not done:
 5:      Apply MNF
 6:      Apply LCMR
 7:  else
 8:  Read the already processed image
 9:  end
10:  Host memory allocation for all data involved in
     learning and classification
11:  Device memory allocation for data involved in training
12:  Copy data: host -> GPU device
13:  Grid and blocks initialization for GPU kernel
     execution
14:  cublasCreate(&handle);
15:  cudaStreamCreate(&stream1);
16:  for i from 0 to n_training_iterations:
17:      Generate the sample image
18:      loadTrainData<<<dimGrid,dimBlock>>>…
19:      cublasDgemm(…); for covariance calculation
20:      cudaDeviceSynchronize();
21:  end
22:  Copy data: GPU device -> host
23:  #pragma omp parallel for private(jj,j)
     schedule(dynamic)
24:  {
25:      OpenMP parallelization of the probability values
         for each pixel, for the classification step
26:  }
27:  Apply SVM training function
28:  #pragma omp parallel for private(j) schedule(dynamic)
29:  {
30:      OpenMP parallelization of SVM prediction step
31:  }
32:  Error matrix calculation
33:  Free memory allocations
```

Computation has been performed on an Intel Processor i7-3770 working at 4.08 GHz with ten iterations for the training part.

The first validation has been performed between the serial C version and the original MATLAB code [1] to ensure that the results were the same. Then, the hybrid parallel version results have been compared against the C serial version. In this case, the classification accuracy differed for only about 10^{-3} between the two implementations. Therefore, the error introduced by the parallel version is negligible considering that accuracy is often expressed as a percentage.

More specifically, the same datasets involved in the analysis of the serial implementation have been adopted, for comparison purposes [1], which are as follows.

Algorithm 3: Sample Generation.

```
 1:  for each class:
34:      #pragma omp parallel for private(i)
35:      {
36:              OpenMP parallelization of:
37:              - id and label initialization for the test set
38:              - Initialization of the class_index
 2:      }
 3:  Set the training data
 4:  Construct the sample
```

Algorithm 4: Error matrix calculation.

```
 1:  Load parameters and initialize variables
 2:  Load memory
 3:  Apply error matrix generation function:
39:      #pragma omp parallel for private(variables)
         schedule(dynamic)
40:      {
41:          OpenMP parallelization of the main for cycle that
             represents the generation
42:      }
43:   Kclass accuracy calculation
44:   Overall accuracy calculation
```

1) *University of Pavia:* Acquired by the ROSIS-03 sensor over the campus at the University of Pavia, Pavia, Italy. This dataset contains 103 spectral bands after the noise-corrupted bands are discarded; each band is of size 610 $\times$ 340. The spatial resolution is 1.3 m, and the spectral coverage ranges from 0.43 to 0.86 μm.
2) *Pavia town center:* Acquired by the same ROSIS-03 sensor over the town center of Pavia, Italy. This dataset contains 102 spectral bands after the noise-corrupted bands are discarded; each band is of size 1096 $\times$ 1096. The spatial resolution is 1.3 m, and the spectral coverage ranges from 0.43 to 0.86 μm.
3) *Indian Pines:* It covers the agricultural Indian Pines test site in Northwestern Indiana and was collected by the AVIRIS sensor. The dataset is of size 145 $\times$ 145 $\times$ 220, with a spatial resolution of 20 m and a spectral range from 0.2 to 2.4 μm. Before the classification, 20 spectral bands (i.e., 104th–108th, 150th–163rd, and 220th) are discarded due to low SNR. The HSI image contains 16 classes.
4) *Salinas:* This image was also collected by the AVIRIS sensor over the Salinas Valley, CA, USA. The dataset is of size 512 $\times$ 217 $\times$ 224, with a spatial resolution of 3.7 m/pixel. Before classification, 20 water absorption bands were removed (namely: 108th–112th, 154th–167th, and 224th).

Different parameters have been evaluated, which are as follows.

1) *Processing time and speedup:* The main loop of the algorithm is subject to parallelization with both OpenMP and CUDA, hence the analysis focused on the comparison between this version with the MATLAB and the C codes, together with the speedup parameter. It can be observed

TABLE I
PERFORMANCE ANALYSIS IN TERMS OF PROCESSING TIMES FOR THE MAIN FOR LOOP

Times on the main *for* loop (OpenMP + CUDA)					
Image	MATLAB (s)	Serial C (s)	CUDA + OpenMP (s)	Speedup C/CUDA	Speedup ML/CUDA
Indian Pines	1.33	6.25	0.51	12.25	2.61
Salinas	6.22	32.58	3.08	10.58	2.02
Uni	6.17	33.55	4.43	7.57	1.39
Pavia Town Center	23.49	139.68	26.3	5.31	0.89

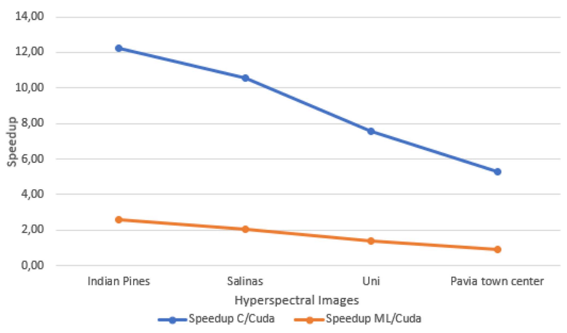


Fig. 3. Speedup of the C language implementation regarding the MATLAB original version: this parameter progressively decreases with the increase of the HSI dimensions, even though it is almost always over 1. This means that, apart from the Pavia town center image (for which the MATLAB algorithm remains the fastest to complete), in all other cases, the combination of OpenMP and GPU is the best one in term of elaboration times. In particular, the best results are obtained in the comparison with the C code.

from Table I that, among the two serial versions, MATLAB is always faster than C language. In fact, the four images under evaluation have been tested applying ten iterations for the training, showing the main *for* loop, where the parallelization is implemented. The serial C version is slower than MATLAB code, whereas the parallelized algorithm greatly decreases computation times by a factor between 5,31 and 12,25 in the case of C version and in the range [0,89-2,61] for the original code. However, the parallelization significantly reduces elaboration times, making this approach the fastest one, as can be highlighted also by the speedup. In the case of the smaller images, it can reach a value higher than 10 for the C code and it is usually twice as fast as the MATLAB implementation. In the case of the speedup, this progressively reduces with the increase of the dimension of the image to be processed (see Fig. 3).

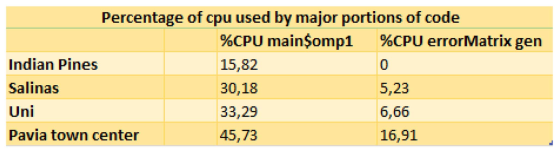
2) *Processing time and speedup for the generation of covariance matrices and noise reduction parts:* Due to the nature of the algorithm, these portions of code were parallelized using only OpenMP and not through hybrid parallelization (see Table II). Concerning the speedup for the C implementation, the parallelization shows better results for all the images, whereas the MATLAB version remains faster than the other two approaches. This is due to the HSI images used (details of each image are reported in Table III) and, more importantly, due to the nature of the algorithm, which does not allow to parallelize many parts, since it has a series of sequential steps.

TABLE II
PERFORMANCE ANALYSIS IN TERMS OF PROCESSING TIMES FOR GENERATION OF THE COVARIANCE MATRICES AND NOISE REDUCTION

Times on the generation of the covariance matrices and noise reduction (pre processing with OpenMP)					
Image	MATLAB (s)	Serial C (s)	OpenMP (s)	Speedup C/OpenMP	Speedup ML/OpenMP
Indian Pines	6.76	28.16	20.86	135	0.32
Salinas	36.72	151.54	112.02	1.35	0.33
Uni	59.86	224.98	191.40	1.18	0.31
Pavia Town Center	182.49	861.21	720.07	1.20	0.25

TABLE III
CHARACTERISTICS OF THE PROCESSED HSIS

Image	Height	Width	Bands	Classes
Indian Pines	145	145	220	16
Salinas	512	217	224	16
Uni	610	340	103	9
Pavia Town Center	1096	715	102	9



Percentage of cpu used by major portions of code		
	%CPU main$omp1	%CPU errorMatrix gen
Indian Pines	15,82	0
Salinas	30,18	5,23
Uni	33,29	6,66
Pavia town center	45,73	16,91

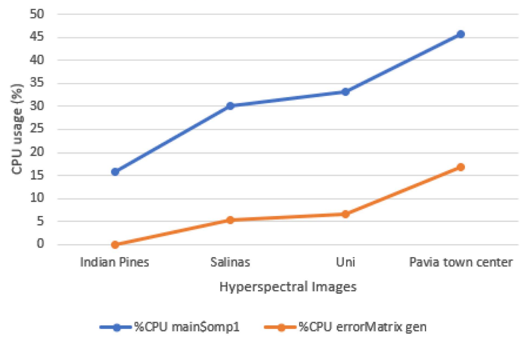


Fig. 4. CPU usage for different parts of the algorithm.

3) Another important parameter is the percentage of CPU used in different portions of the code. Fig. 4 shows the amount of work done for two specific parts of the algorithm: The main loop and the error matrix generation.

The Tesla K40 device is a mature technology that has been developed mainly for HPC purposes, whereas the RTX 2080 model is an evolution of the Tesla family of GPUs and it has been conceived not only for the HPC field, but also for graphics applications. Its clock is higher than the first GPU, therefore in terms of throughput, the RTX 2080 is faster.

As a consequence, tests performed with the RTX model aimed at minimizing possible outliers due to the different configurations of this device. Hence, 400 iterations have been done for each of the four datasets available. As shown in Table IV, execution times are comparable among the two GPUs for the smaller datasets, due to their reduced dimensions that do not allow to fully exploit GPU advantages. However, in the University of Pavia dataset, even though both GPUs have similar execution times, it is possible to note a better speedup in the case of the RTX 2080 device. Moreover, the Pavia town center

TABLE IV
PERFORMANCE COMPARISON WITH TWO GPU MODELS

Execution times for all datasets with serial and parallelized code using different GPUs								
Image	MATLAB (s)	Serial C (s)	CUDA + OpenMP (s) on Tesla K40	CUDA + OpenMP (s) on RTX 2080	Speedup ML/Tesla K40	Speedup ML/RTX 2080	Speedup C/Tesla K40	Speedup C/RTX 2080
Indian Pines	8.09	34.41	0.51	0.73	15.86	11.08	67.47	47.14
Salinas	42.94	184.12	3.08	3.85	13.94	11.15	59.78	47.82
Uni	66.03	258.53	4.43	4.01	14.91	16.47	58.36	64.47
Pavia Town Center	205.98	1000.89	26.3	15.8	7.83	13.04	38.06	63.35

dataset shows a reduction in the RTX 2080 model, which can be observed also considering the speedups.

The reason for the different performance of the two boards is the architectural organization of the CUDA cores. The Tesla K40c GPU features 15 SMX and can then process 15 threads blocks in parallel. On the other hand, the RTX 2080 board is equipped with 128 SMX, allowing to process more blocks in parallel than the Tesla K40c board. This is a critical issue when the data dimensionality increases. In other words, more SMX allow to process more blocks in parallel and to elaborate more data at the same time. When the data dimensionality is low, the number of blocks processed in parallel is no more a critical issue, whereas having more cores inside the same SMX allows to obtain a better performance since more cores can process data while the others are accessing the memory. These considerations are supported by the processing times of Table IV, where the Tesla K40c GPU achieves the best performance with the two smallest image, whereas the RTX 2080 board outperforms the first device with the other images.

VI. CONCLUSION

This article presents the optimization and hybrid parallelization of an FE methodology applied to HSIs, using the LCMR. Our aim is to accelerate the execution of the original MATLAB code and optimize portions of the algorithm, which are suitable for OpenMP (multicores) and CUDA (manycores). The hybrid parallelization approach, in contrast with solely OpenMP or CUDA, has been evaluated as a better option, since it allows to optimize every eligible portion with the most appropriate parallelization technique. For this reason, a first optimization has been performed by obtaining a new serial version of this algorithm in C language, since this is the backbone of both OpenMP and CUDA parallelization. The algorithm applies, at first, a DR with the maximum noise reduction operator, then the LCMR is calculated for FE purposes. After that, the SVM is used for the classification and, at last, the error matrix calculation is performed and the classification matrix is produced as output. Specific parts of the procedure have been written with the addition of OpenMP directives, such as the generation of the sample and the error matrix calculation. Concerning GPU implementation, CM calculation has been performed with this approach. This version has been tested using four test sets of real HSIs, which are usually applied in the remote-sensing context. The parallelized solution demonstrates optimal results compared to the serial C code while performance remains faster in the case of the original MATLAB implementation. This is due to the dimensions of such images and also the nature of the algorithm, which does not allow complete parallelization.

REFERENCES

[1] L. Fang, N. He, S. Li, A. J. Plaza, and J. Plaza, "A new spatial–spectral feature extraction method for hyperspectral images using local covariance matrix representation," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 6, pp. 3534–3546, Jun. 2018, doi: 10.1109/TGRS.2018.2801387.

[2] P. Ghamisi, J. Plaza, Y. Chen, J. Li, and A. J. Plaza, "Advanced spectral classifiers for hyperspectral images: A review," *IEEE Geosci. Remote Sens. Mag.*, vol. 5, no. 1, pp. 8–32, Mar. 2017, doi: 10.1109/MGRS.2016.2616418.

[3] R. Guerra, E. Martel, J. Khan, S. López, P. Athanas, and R. Sarmiento, "On the evaluation of different high-performance computing platforms for hyperspectral imaging: An OpenCL-based approach," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 11, pp. 4879–4897, Nov. 2017, doi: 10.1109/JSTARS.2017.2737958.

[4] X. Kang, X. Zhang, S. Li, K. Li, J. Li, and J. A. Benediktsson, "Hyperspectral anomaly detection with attribute and edge-preserving filters," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 10, pp. 5600–5611, Oct. 2017, doi: 10.1109/TGRS.2017.2710145.

[5] E. Marenzi, E. Torti, F. Leporati, E. Quevedo, and G. M. Callicò, "Block matching super-resolution parallel GPU implementation for computational imaging," *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 368–376, Nov. 2017, doi: 10.1109/TCE.2017.015077.

[6] C. Wu, L. Zhang, and B. Du, "Kernel slow feature analysis for scene change detection," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 4, pp. 2367–2384, Apr. 2017, doi: 10.1109/TGRS.2016.2642125.

[7] Y. Liu, J. Li, P. Du, A. Plaza, X. Jia, and X. Zhang, "Class-oriented spectral partitioning for remotely sensed hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 691–711, Feb. 2017, doi: 10.1109/JSTARS.2016.2588980.

[8] S. Bernabé, S. Sánchez, A. Plaza, S. López, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Jun. 2013, doi: 10.1109/JSTARS.2013.2254470.

[9] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs," *J. Real Time Image Process.*, vol. 10, no. 3, pp. 469–483, 2015, doi: 10.1007/s11554-012-0269-2.

[10] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid CPU–GPU real-time hyperspectral unmixing chain," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 945–951, Feb. 2016, doi: 10.1109/JSTARS.2015.2485399.

[11] L. I. Jiménez and A. Plaza, "HyperMix: An open-source tool for fast spectral unmixing on graphics processing units," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 9, pp. 1883–1887, Sep. 2015, doi: 10.1109/LGRS.2015.2435001.

[12] E. Martel, R. Guerra, S. López, and R. Sarmiento, "A GPU-based processing chain for linearly unmixing hyperspectral images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 3, pp. 818–834, Mar. 2017, doi: 10.1109/JSTARS.2016.2614842.

[13] E. Marenzi, A. Carrus, G. Danese, F. Leporati, and G. M. Callico, "Efficient parallelization of motion estimation for super-resolution," in *Proc. 25th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2017, , pp. 274–277, doi: 10.1109/PDP.2017.64.

[14] Y. Guan, S. Guo, Y. Xue, J. Liu, and X. Zhang, "Application of airborne hyperspectral data for precise agriculture," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2004, vol. 6, pp. 4195–4198, doi: 10.1109/IGARSS.2004.1370060.

[15] J. Rosentreter, R. Hagensieker, A. Okujeni, R. Roscher, P. D. Wagner, and B. Waske, "Subpixel mapping of urban areas using EnMAP data and multioutput support vector regression," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 5, pp. 1938–1948, May 2017, doi: 10.1109/JSTARS.2017.2652726.

[16] H.-C. Chang and A. Burke, "Integration of hyperspectral and polarimetric radar remote sensing techniques for monitoring invasive weeds," in *Proc. IEEE Geosci. Remote Sens. Symp.*, 2014, pp. 2950–2952, doi: 10.1109/IGARSS.2014.6947095.

[17] A. Fontanella, E. Marenzi, E. Torti, G. Danese, A. Plaza, and F. Leporati, "A suite of parallel algorithms for efficient band selection from hyperspectral images," *J. Real Time Image Process.*, vol. 15, pp. 537–553, 2018, doi: 10.1007/s11554-018-0765-0.

[18] C.-I. Chang and S. Wang, "Constrained band selection for hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 44, no. 6, pp. 1575–1585, Jun. 2006, doi: 10.1109/TGRS.2006.864389.

[19] M. Gong, M. Zhang, and Y. Yuan, "Unsupervised band selection based on evolutionary multiobjective optimization for hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 1, pp. 544–557, Jan. 2016, doi: 10.1109/TGRS.2015.2461653.

[20] K. Sun, X. Geng, L. Ji, and Y. Lu, "A new band selection method for hyperspectral image based on data quality," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2697–2703, Jun. 2014, doi: 10.1109/JSTARS.2014.2320299.

[21] S. Jia, G. Tang, J. Zhu, and Q. Li, "A novel ranking-based clustering approach for hyperspectral band selection," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 1, pp. 88–102, Jan. 2016, doi: 10.1109/TGRS.2015.2450759.

[22] M. Li, W. Li, Y. Liu, Y. Huang, and G. Yang, "Adaptive mask sampling and manifold to Euclidean subspace learning with distance covariance representation for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5508518, doi: 10.1109/TGRS.2023.3265388.

[23] Y. Yan, J. Ren, Q. Liu, H. Zhao, H. Sun, and J. Zabalza, "PCA-domain fused singular spectral analysis for fast and noise-robust spectral–spatial feature mining in hyperspectral classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 20, 2023, Art. no. 5505405, doi: 10.1109/LGRS.2021.3121565.

[24] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid CPU-GPU real-time hyperspectral unmixing chain," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 945–951, Feb. 2016, doi: 10.1109/JSTARS.2015.2485399.

[25] Z. Wu et al., "GPU parallel implementation of spatially adaptive hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 4, pp. 1131–1143, Apr. 2018, doi: 10.1109/JSTARS.2017.2755639.

[26] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 6, pp. 2270–2278, Jun. 2016, doi: 10.1109/JSTARS.2016.2542193.

[27] Q. Du, B. Tang, W. Xie, and W. Li, "Parallel and distributed computing for anomaly detection from hyperspectral remote sensing imagery," *Proc. IEEE*, vol. 109, no. 8, pp. 1306–1319, Aug. 2021, doi: 10.1109/JPROC.2021.3076455.

[28] E. Marenzi, E. Torti, G. Danese, and F. Leporati, "FPGA high level synthesis for the classification of skin tumors with hyperspectral images," in *Proc. 11th Mediterranean Conf. Embedded Comput.*, 2022, pp. 1–4, doi: 10.1109/MECO55406.2022.9797211.

[29] C. González, D. Mozos, S. Lopez, and R. Sarmiento, "FPGA implementation to estimate the number of endmembers in hyperspectral images," in *Proc. 25th Int. Conf. Field Programmable Log. Appl.*, 2015, pp. 1–8, doi: 10.1109/FPL.2015.7293936.

[30] H. Du and H. Qi, "A reconfigurable FPGA system for parallel independent component analysis," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, 2006, Art. no. 023025, doi: 10.1155/ES/2006/23025.

[31] J. M. Haut et al., "Cloud deep networks for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 12, pp. 9832–9848, Dec. 2019, doi: 10.1109/TGRS.2019.2929731.

[32] J. M. Haut, M. E. Paoletti, S. Moreno-Álvarez, J. Plaza, J.-A. Rico-Gallego, and A. Plaza, "Distributed deep learning for remote sensing data interpretation," *Proc. IEEE*, vol. 109, no. 8, pp. 1320–1349, Aug. 2021, doi: 10.1109/JPROC.2021.3063258.

[33] G. M. Callicó, S. Lopez, B. Aguilar, J. F. López, and R. Sarmiento, "Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using OpenCL," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 8, pp. 3650–3659, Aug. 2014, doi: 10.1109/JSTARS.2014.2340579.

[34] S. Akintoye, P. L. Han, D. X. Zhang, H. Chen, and P. D. Zhang, "A hybrid parallelization approach for distributed and scalable deep learning," *SSRN Electron. J.*, 2022, doi: 10.2139/ssrn.4043672.

[35] M. Klemm, B. R. de Supinski, "*OpenMP Application Program Interface Version 5.0*." Independently published, 2019.

[36] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan.–Mar. 1998, doi: 10.1109/99.660313.

[37] "NVIDIA CUDA programming guide 3.1," May 2010.

[38] D. Luebke, "CUDA: Scalable parallel programming for high-performance scientific computing," in *Proc. 5th IEEE Int. Symp. Biomed. Imag.: From Nano Macro*, 2008, pp. 836–838, doi: 10.1109/ISBI.2008.4541126.

[39] D. Mustafa, "A survey of performance tuning techniques and tools for parallel applications," *IEEE Access*, vol. 10, pp. 15036–15055, 2022, doi: 10.1109/ACCESS.2022.3147846.

[40] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 8, pp. 1778–1790, Aug. 2004, doi: 10.1109/TGRS.2004.831865.

[41] F. Ratle, G. Camps-Valls, and J. Weston, "Semisupervised neural networks for efficient hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 5, pp. 2271–2282, May 2010, doi: 10.1109/TGRS.2009.2037898.

[42] J. Ham, Y. Chen, M. M. Crawford, and J. Ghosh, "Investigation of the random forest framework for classification of hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 492–501, Mar. 2005, doi: 10.1109/TGRS.2004.842481.

[43] L. Fang, C. Wang, S. Li, and J. A. Benediktsson, "Hyperspectral image classification via multiple-feature-based adaptive sparse representation," *IEEE Trans. Instrum. Meas.*, vol. 66, no. 7, pp. 1646–1657, Jul. 2017, doi: 10.1109/TIM.2017.2664480.

[44] W. Li, S. Prasad, J. E. Fowler, and L. M. Bruce, "Locality-preserving dimensionality reduction and classification for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 4, pp. 1185–1198, Apr. 2012, doi: 10.1109/TGRS.2011.2165957.

[45] S. Prasad and L. M. Bruce, "Limitations of principal components analysis for hyperspectral target recognition," *IEEE Geosci. Remote Sens. Lett.*, vol. 5, no. 4, pp. 625–629, Oct. 2008, doi: 10.1109/LGRS.2008.2001282.

[46] A. Villa, J. A. Benediktsson, J. Chanussot, and C. Jutten, "Independent component discriminant analysis for hyperspectral image classification," in *Proc. 2nd Workshop Hyperspectral Image Signal Process.: Evol. Remote Sens.*, 2010, pp. 1–4, doi: 10.1109/WHISPERS.2010.5594853.

[47] A. A. Green, M. Berman, P. Switzer, and M. D. Craig, "A transformation for ordering multispectral data in terms of image quality with implications for noise removal," *IEEE Trans. Geosci. Remote Sens.*, vol. 26, no. 1, pp. 65–74, Jan. 1988, doi: 10.1109/36.3001.

[48] H. Li, "An overview on remote sensing image classification methods with a focus on support vector machine," in *Proc. Int. Conf. Signal Process. Mach. Learn.*, 2021, pp. 50–56, doi: 10.1109/CONF-SPML54095.2021.00019.

[49] M. A. Shafaey et al., "Pixel-wise classification of hyperspectral images with 1D convolutional SVM networks," *IEEE Access*, vol. 10, pp. 133174–133185, 2022, doi: 10.1109/ACCESS.2022.3231579.

[50] H. Fu, G. Sun, J. Ren, A. Zhang, and X. Jia, "Fusion of PCA and segmented-PCA domain multiscale 2-D-SSA for effective spectral-spatial feature extraction and data classification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5500214, doi: 10.1109/TGRS.2020.3034656.

[51] B. Praveen and V. Menon, "Study of spatial-spectral feature extraction frameworks with 3-D convolutional neural network for robust hyperspectral imagery classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 14, pp. 1717–1727, 2021, doi: 10.1109/JSTARS.2020.3046414.

[52] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson, "Classification of hyperspectral data from urban areas based on extended morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 480–491, Mar. 2005, doi: 10.1109/TGRS.2004.842478.

[53] X. Kang, S. Li, and J. A. Benediktsson, "Feature extraction of hyperspectral images with image fusion and recursive filtering," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 6, pp. 3742–3752, Jun. 2014, doi: 10.1109/TGRS.2013.2275613.

[54] X. Kang, S. Li, and J. A. Benediktsson, "Spectral–spatial hyperspectral image classification with edge-preserving filtering," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 5, pp. 2666–2677, May 2014, doi: 10.1109/TGRS.2013.2264508.

[55] L. Mou, P. Ghamisi, and X. X. Zhu, "Deep recurrent neural networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 7, pp. 3639–3655, Jul. 2017, doi: 10.1109/TGRS.2016.2636241.

[56] W. Zhao and S. Du, "Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 8, pp. 4544–4554, Aug. 2016, doi: 10.1109/TGRS.2016.2543748.
[57] W. Li, G. Wu, F. Zhang, and Q. Du, "Hyperspectral image classification using deep pixel-pair features," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 2, pp. 844–853, Feb. 2017, doi: 10.1109/TGRS.2016.2616355.
[58] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 10, pp. 6232–6251, Oct. 2016, doi: 10.1109/TGRS.2016.2584107.
[59] D. Li, F. Kong, J. Liu, and Q. Wang, "Superpixel-based multiple statistical feature extraction method for classification of hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 59, no. 10, pp. 8738–8753, Oct. 2021, doi: 10.1109/TGRS.2021.3056722.
[60] T. Lu, S. Li, L. Fang, L. Bruzzone, and J. A. Benediktsson, "Set-to-set distance-based spectral–spatial classification of hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 12, pp. 7122–7134, Dec. 2016, doi: 10.1109/TGRS.2016.2596260.
[61] C.-I. Chang, "An information-theoretic approach to spectral variability, similarity, and discrimination for hyperspectral image analysis," *IEEE Trans. Inf. Theory*, vol. 46, no. 5, pp. 1927–1932, Aug. 2000, doi: 10.1109/18.857802.
[62] T. Lu, S. Li, L. Fang, Y. Ma, and J. A. Benediktsson, "Spectral–spatial adaptive sparse representation for hyperspectral image denoising," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 1, pp. 373–385, Jan. 2016, doi: 10.1109/TGRS.2015.2457614.
[63] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, May 2011, Art. no. 27, doi: 10.1145/1961189.1961199.
[64] E. Torti et al., "Acceleration of brain cancer detection algorithms during surgery procedures using GPUs," *Microprocessors Microsyst.*, vol. 61, pp. 171–178, 2018, doi: 10.1016/j.micpro.2018.06.005.

Emanuele Torti received the bachelor's degree in electronic engineering, the master's degree in computer science engineering (*cum laude*), and the Ph.D. degree in electronics and computer science engineering from the University of Pavia, Pavia, Italy, in 2009, 2011, and 2014, respectively.

He is currently an Assistant Professor with the Engineering Faculty, University of Pavia. His research interest includes high-performance architectures for real-time image processing and signal elaboration.

Elisa Marenzi received the B.S. and M.S. degrees in biomedical engineering and the Ph.D. degree in bioengineering and bioinformatics from the University of Pavia, Pavia, Italy, in 2007, 2010, and 2014, respectively.

She is currently an Assistant Professor with the Custom Computing and Programmable Systems Laboratory, University of Pavia. Her research interests include GPU parallel computing and the design and development of electronic and embedded systems.

Dr. Marenzi obtained the second place in the Best Student Paper contest organized by the IEEE Sensors Applications Symposium in February 2012. In June 2012, she obtained third place in an academic competition promoted by the local industrial association and won a national contest for people under 30 on social innovation and technology, the Lifebility Award 2012, promoted by Milan's Lion Club. In 2014, she was awarded for her Ph.D. thesis from Rotary and the National Association for Automatic Computing.

Giovanni Danese received the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy, in 1987.

Since 1998, he has been the Leader of the Microcomputer Laboratory, Computer Engineering and Systems Science Department, University of Pavia. Since 2021, he has also been a Full Professor with the Computer Programming and Computer Architecture, Engineering Faculty, University of Pavia. His research interests include parallel computing, special-purpose computers, and multiprocessor devices for artificial neural network implementations.

Antonio J. Plaza (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the University of Extremadura, Badajoz, Spain, in 1999 and 2002, respectively.

He is currently the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 600 publications, including 400 journal papers, 25 book chapters, and 330 peer-reviewed conference proceeding papers. His research interests include hyperspectral data processing and parallel computing of remote-sensing data.

Dr. Plaza is a Fellow of the IEEE for contributions to hyperspectral data processing and parallel computing of earth observation data. He was a member of the Editorial Board for the *IEEE Geoscience and Remote Sensing Newsletter* from 2011 to 2012 and the *IEEE Geoscience and Remote Sensing Magazine* in 2013. He was also a member of the steering committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He was a recipient of the recognition of best reviewers of the *IEEE Geoscience and Remote Sensing Letters* in 2009 and the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, for which he served as an Associate Editor from 2007 to 2012. He was a recipient of the Best Column Award of the *IEEE Signal Processing Magazine* in 2015, the 2013 Best Paper Award of the JSTARS journal, and the most highly cited paper in 2005–2010 in the *Journal of Parallel and Distributed Computing*. He was also the recipient of the best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He is also an Associate Editor for the IEEE ACCESS. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) from 2011 to 2012, and as a President of the Spanish Chapter of the IEEE GRSS from 2012 to 2016. He has reviewed more than 500 manuscripts for more than 50 different journals. He served as the Editor-in-Chief for the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING journal. He has guest edited 17 special issues on hyperspectral remote sensing for different journals. He has been included in the Highly Cited Researchers list in 2018–2022.

Francesco Leporati received the Laurea degree in electronics engineering and the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy, in 1988 and 1993, respectively.

He is currently an Associate Professor with the University of Pavia, where he teaches mechatronics, industrial informatics and embedded systems, and digital systems design. His research interests include high-performance computing and embedded systems in particular for bioengineering.

Dr. Leporati is a member of the IEEE Computer Society and Euromicro Society (the Director of Italian correspondents and General Secretary).

Open Access funding provided by 'Università degli Studi di Pavia' within the CRUI CARE Agreement