Extended Anisotropic Diffusion Profiles in GPU for Hyperspectral Imagery

Álvaro Acción, Francisco Argüello and Dora B. Heras

Version: accepted article

How to cite:

Álvaro Acción, Francisco Argüello and Dora B. Heras (2019) Extended Anisotropic Diffusion Profiles in GPU for Hyperspectral Imagery. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 12 (12), 4964 - 4976.

Doi: 10.1109/JSTARS.2019.2939857

Extended Anisotropic Diffusion Profiles in GPU for Hyperspectral Imagery

Álvaro Acción, Francisco Argüello, and Dora B. Heras

***Abstract*—Morphological profiles are a common approach for extracting spatial information from remote sensing hyperspectral images by extracting structural features. Other profiles can be built based on different approaches such as, for example, differential morphological profiles, or attribute profiles. Another technique used for characterizing spatial information on the images at different scales is based on computing profiles based on edge-preserving filters such as anisotropic diffusion filters. Their main advantage is that they preserve the distinctive morphological features of the images at the cost of an iterative calculation. In this paper, the high computational cost associated to the construction of Anisotropic Diffusion Profiles (ADPs) is highly reduced. In particular, we propose a low cost computational approach for computing ADPs on Nvidia GPUs as well as a detailed characterization of the method, comparing it in terms of accuracy and structural similarity to other existing alternatives.**

***Index Terms*—Hyperspectral, anisotropic diffusion profile, nonlinear diffusion, remote sensing, CUDA.**

I. Introduction

HYPERSPECTRAL imagery for remote sensing is showing widespread adoption in fields such as agriculture [1] or ecological science to study, for example, biodiversity [2] or land cover changes [3]. Advances in sensor technology and lower costs for image acquisition are yielding an ever increasing range of applications, higher resolution (e.g. spectral, spatial and temporal) images and more diverse acquisition conditions [4]. Hyperspectral images are computationally expensive to process due to the high volume of data they contain and the complex processing tasks that they involve, becoming good candidates for the leveraging of parallel computing platforms, GPUs in particular [5, 6].

Mathematical morphology [7] is one of the most common techniques used to extract spatial-spectral information from hyperspectral images such as, for example, the size, orientation, and contrast of the spatial structures present in the image. A Morphological Profile (MP) [7] is built by sequentially applying opening and closing transformations with a Structuring Element (SE) of increasing size over the bands of the image [8]. The combination of several MPs [7] in a single dataset for subsequent analysis steps is called Extended Morphological Profile (EMP) [9]. Other different methods for extracting spatial information were subsequently derived from the basic MP. For example, Differential Morphological Profiles (DMPs) [7] are an extension of MPs constructed by computing the difference between two MPs for contiguous sizes of the SE, i.e., for different scales. Attribute Profiles (APs) [10] are also a generalization of MPs [7] and overcome some of their shortcomings. The introduction of multiple different attributes via Attribute Filters (AFs) for the characterization of the image allows for a more accurate modeling of the scene compared to traditional MPs [7]. Extinction Profiles (EPs) [11] make use of Extinction Filters (EFs) [11] to create a fully automatic method using regional extrema of attributes. EPs [11] were shown to provide better classification results than APs [10].

As the number of components obtained by the application of profiles is high, they are applied, in the case of hyperspectral images, not to the original image but to a dimensionally reduced image obtained after the application of Feature Extraction (FE) techniques. These techniques usually exploit the spectral information of the image while maintaining or even increasing the separability of the different classes in the images. This is the case of Principal Component Analysis (PCA) [12], Independent Component Analysis (ICA) [13] or Singular Spectrum Analysis (SSA) [14].

The diffusion equation is one of the possibilities to build a linear scale space [15] from an image. A scale-space is an image representation at a continuum of scales, embedding gradually simplified versions of the image, provided that it fulfills certain requirements [16]. Multiple examples of the application of scale space regularization have been applied to remote sensing [17]. In the case of nonlinear diffusion, good results have been obtained for the classification of multivalued and hyperspectral images [18], [19].

Anisotropic diffusion can also be used to enhance multispectral images increasing their signal-to-noise ratio [20] and, in particular, it can be used for the generation of extended profiles for classification [21]. The application of nonlinear diffusion filtering has shown great promise [22] compared to traditional linear filtering due to its ability to preserve the distinctive morphological features of the images such as edges or even enhance them.

Until recent times, the schemes used to implement nonlinear diffusion were characterized by their high computational cost. Weickert *et al.* [23], [24] introduced efficient schemes for nonlinear diffusion filtering using Additive Operator Splitting (AOS), that allows the application of nonlinear filtering in a very efficient and reasonably accurate way. Later on, the schemes were further improved with the addition of Fast Explicit Diffusion (FED) schemes [25]. These are computationally more efficient, while providing higher precision than the existing AOS schemes.

Graphics Processing Units (GPUs) are high performance

The authors are with the Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Spain. (E-mail: alvaro.accion.montes@usc.es, dora.blanco@usc.es, francisco.arguello@usc.es)

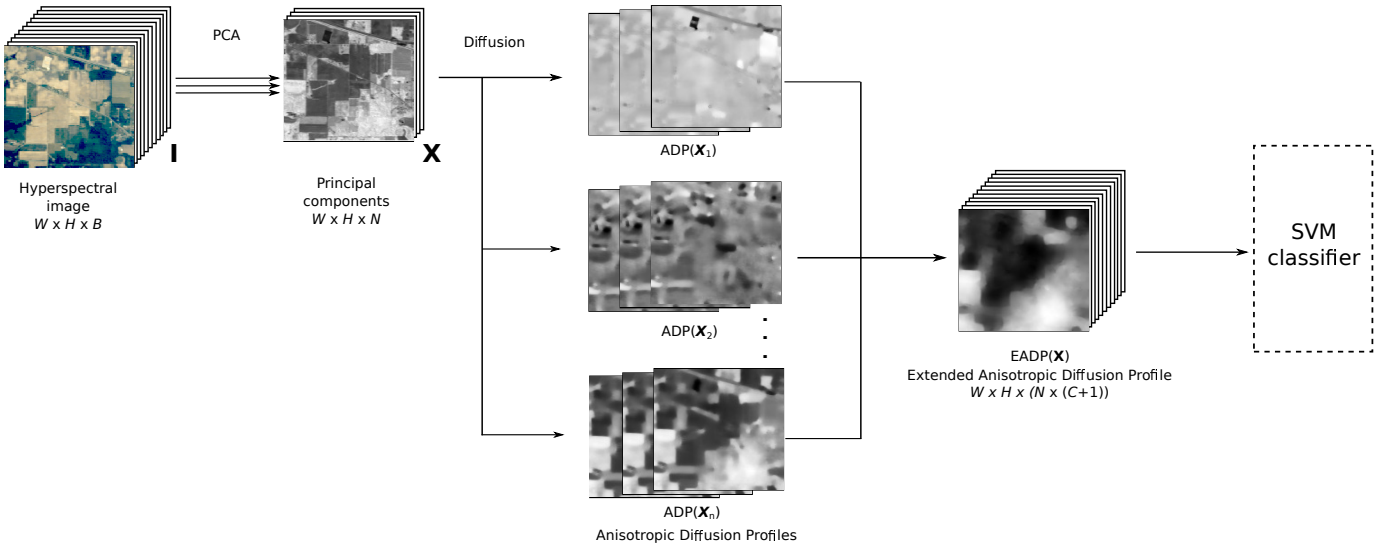


Fig. 1: Proposed scheme for the classification of hyperspectral images on GPU based on the extraction of spatial information using diffusion profiles.

computing platforms that can be used to efficiently compute multiple algorithms for hyperspectral image processing in real or near real time [26], [27], [28], [29]. The availability of commodity GPUs capable of performing compute heavy tasks allows for previously server-grade performance to be achieved in an inexpensive manner.

In this paper, we propose a new approach for the efficient extraction of spectral-spatial information on GPU for hyperspectral imagery by computing Extended Anisotropic Diffusion Profiles (EADPs). It is based on the application of nonlinear diffusion filtering and built by the concatenation of different Anisotropic Diffusion Profiles (ADPs). In turn, each ADP is generated by applying several instances of nonlinear diffusion filtering to a band of the image. The main contributions are:

1) A method for the construction of an Extended Anisotropic Profile (EADP) is proposed. It is computed by using non-linear diffusion based on FED, due to its low computational cost and suitability for parallel computing.
2) The low number of components of the profile required by the method allows for a fast and efficient subsequent classification process.
3) The efficient GPU implementation of the proposed method exploits the features of the CUDA 6.1 architecture, and it is suitable for execution on commodity GPUs.
4) A detailed characterization of the method based on exhaustively varying the algorithm parameters is presented, as well as a comparison to different profiles for spectral-spatial information extraction proposed in the literature.

The paper is organized into six sections. Section II presents a brief introduction to the concepts of nonlinear diffusion and the FED scheme used to implement it. Section III introduces the steps involved in the computation of the ADPs. Section IV discusses the characteristics of the CUDA implementation. The experimental results for the evaluation of the classification performance are presented in Section V. Finally, Section VI summarizes the main conclusions.

II. Nonlinear Diffusion Filtering

The physical concept of diffusion can be thought of as the process that reduces the differences in concentration of a substance L without changing its mass. The classical formulation for the diffusion equation is as follows:

$$\frac{\delta L}{\delta t} = \mathrm{div}(c(x,y,t) \cdot \nabla L). \tag{1}$$

In the equation above, div represents the divergence operator; ∇, the gradient operator and c, called the conductivity function, is a function that controls the diffusion and adapts it to the image structure. Perona and Malik [30] proposed a variable and adaptive c that reduces smoothing across boundaries and is chosen as a function of the gradient magnitude,

$$c(x,y,t) = g(|\nabla L_\sigma(x,y,t)|), \tag{2}$$

where L_σ represents the original image after being smoothed by the Gaussian kernel of mean 0 and variance σ^2. The t variable represents time, but it can also be thought of as a scale value that controls the level of detail of the resulting image, with larger values leading to simpler representations.

Nonlinear diffusion filtering, also called non-homogeneous diffusion, is a process that applies nonlinear diffusion equations to the pixel values in an image. Applied to the field of image processing, it tries to balance the concentration of gray value across the elements of an image.

There are several approaches to the calculation of the diffusion coefficients. Perona and Malik [30] described two different formulations for the conductivity function, shown in Eqs. (3) and (4), while Weickert [31] proposed a conductivity

for which intraregional smoothing is used instead of interregional blurring, shown in Eq. (5):

$$g_1 = \exp\left(-\frac{|\nabla L_\sigma|^2}{k^2}\right), \tag{3}$$

$$g_2 = \frac{1}{1+\frac{-|\nabla L_\sigma|^2}{k^2}}, \tag{4}$$

$$g_3 = \begin{cases} 1 & \text{if } |\nabla L_\sigma|^2 = 0 \\ 1-\exp\left(\frac{0.331488}{(|\nabla L_\sigma|/k)^8}\right) & \text{if } |\nabla L_\sigma|^2 > 0. \end{cases} \tag{5}$$

The parameter k, also called contrast parameter serves as a threshold enabling backwards diffusion. Higher k values generally imply low contrast edges will be smoothed out.

A. Fast Explicit Diffusion

Fast Explicit Diffusion or FED is an efficient numerical scheme used to solve parabolic and elliptic partial differential equations.

FED exploits the fact that the Gaussian kernel can be approximated by any symmetric 1-D kernel with the weights of the coefficients w_k fulfilling the conditions $w_k = w_{-k}, \forall k \in \{1,\ldots,n\}$ and $\sum_{k=0}^{n} w_k = 1$.

In matrix notation, the FED cycle can be defined as:

$$L^{j+1} = \left(I + \tau_j A\left(L\right)\right) L^j, \quad j = 0,\ldots,n-1, \tag{6}$$

where L^{j+1} is the solution to the diffusion problem defined by Eq. (2) at a given step and A is the conductivity matrix computed from c [15].

The FED parameters can be expressed as,

$$\tau_j = \frac{\tau_{max}}{2\cos^2(\pi \cdot \frac{2j+1}{4n+2})}, \tag{7}$$

$$n = \left\lceil -\frac{1}{2} + \frac{1}{2}\sqrt{1+\frac{12T}{M\tau_{\max}}} \right\rceil, \tag{8}$$

where τ_j is the step size of the j-th iteration resulting from the factorization of the box filter, τ_{max} is the maximal step size that does not violate the stability condition of the explicit scheme and T is the global process time. Since τ_{max} is known, it is possible to obtain the number of steps n using Eq. (8). For large values of n, the resulting τ_j can be significantly larger than the stability condition [32].

FED works by performing M cycles composed of n explicit diffusion steps. In each step, a varying τ_j is used [32]. M controls the quality of the approximation, with higher values leading to approximations with a lower error at the expense of higher complexity due to the limitation imposed in the growth of τ_j.

III. Extended Diffusion Profile

In this section we describe the algorithm for the generation of the EADP and determine its computational complexity. Due to the large number of highly correlated bands in hyperspectral images, FE methods are required to reduce computational requirements and obtain a lower number of highly significant feature points. In this case FE by PCA is performed.

A. Diffusion Profiles

As shown in Fig. 1, the ADPs are generated after applying several different instances of nonlinear diffusion filtering to every principal component extracted from the original image. Each ADP will have $C+1$ components where C is the number of instances of diffusion applied, producing as a result images with decreasing levels of detail. The resulting EADP will have $N \times (C+1)$ components, where N is the number of principal components retained. An example of the graphical result obtained after calculating the EADP for the IndianP image for seven principal components can be seen in Fig. 2. Note that, as the number of instances of diffusion increases, the degree of smoothness in each area of the image also increases. As a consequence the noise is also reduced and the homogeneity of pixels in the area is increased. In more detail, the classification steps are the following:

1) PCA computation: PCA was used over the original hyperspectral cube in order to extract the most relevant principal components. The steps of the PCA computation are described in Algorithm 1.

Starting with the original hyperspectral image (**I**), the data are centered band by band. A PCA is later applied pixel wise for the pixel values in all bands of the feature-scaled hyperspectral cube. The covariance matrix is then obtained by multiplying the original matrix by its transpose, resulting in a matrix of dimensions $B \times B$ (line 3). The matrix is then fed to a SVD function in order to obtain the eigenvectors of the image (line 4), that will be finally used to calculate the principal components (line 5). The output of this step is a reduced matrix **X**, which will then be used for the generation of the EADP.

2) EADP computation: Once the principal components of the image are obtained, C instances of the nonlinear diffusion algorithm will be applied to each principal component $\boldsymbol{X}_i$ using different process times T_i. This strategy will generate components with decaying levels of detail and thus containing different spatial information. The first component of the profile is always the principal component that was used to generate it. Denoting the diffusion process as $\mathrm{Diff}(\boldsymbol{X}_i, T_i)$, an ADP for the principal component $\boldsymbol{X}_i$ is defined as:

$$\mathrm{ADP}(\boldsymbol{X}_i) = \{\boldsymbol{X}_i, \mathrm{Diff}(\boldsymbol{X}_i, T_1), \ldots, \mathrm{Diff}(\boldsymbol{X}_i, T_c)\} \tag{9}$$

The algorithm in this paper uses a FED scheme in order to obtain the highest possible efficiency in the resolution of the difussion equations, as it is described in Algorithm 2. Two stages are required: the computation of the diffusivity matrix and the computation of the FED process.

The first step in the computation of the diffusivity matrix begins with the application of Gaussian filtering of variance σ^2 to a principal component $\boldsymbol{X}_i$ (line 4). This smooths the original image prior to the computation of the derivatives. The Scharr function is then applied to the resulting image both vertically and horizontally to obtain the derivatives (line 5). The derivatives are then used to compute the contrast parameter if none has been specified (line 6). In this case, the selected value was 0.7, as described in [33]. Once the previous

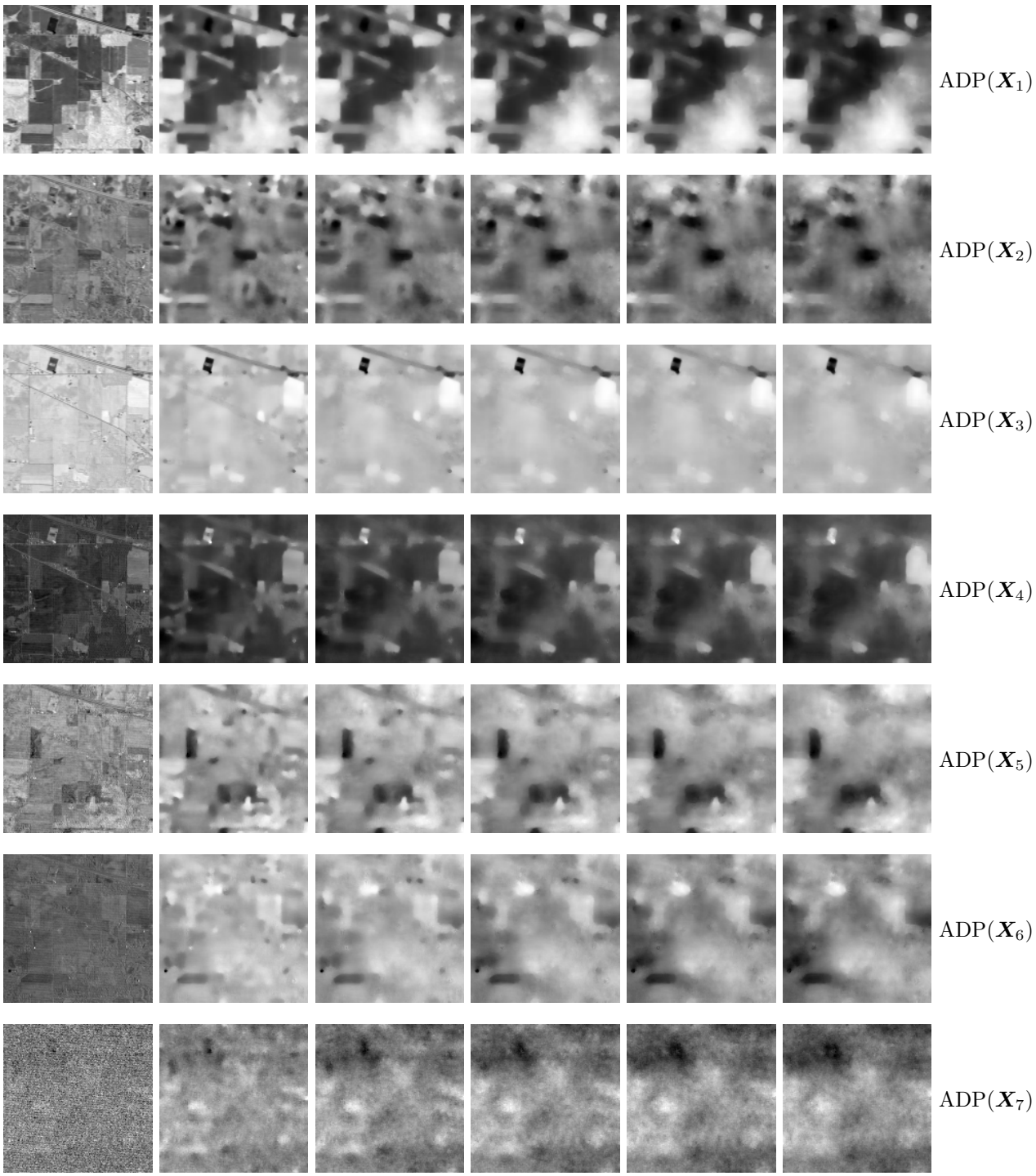


Fig. 2: EADP for IndianP. The images represent, for each $\mathrm{ADP}(\boldsymbol{X}_i)$ with $C = 5$ and from left to right, the i-th principal component followed by the five components resulting from diffusion filtering.

steps are complete, the diffusivity matrix can be obtained by applying Eq. (4) (line 7).

The second stage, that corresponds to the FED computation, starts with the initialization of the required internal parameters based on process time T and the number of cycles n provided; the number of inner steps is derived from the process time and number of cycles (line 10). The time steps τ_j are then computed (line 11). After this, the state of the FED algorithm is iteratively updated (lines 13-15) and the cycle for the current state is then applied. After all the cycles of the FED process described in Eq. (6) finish, the resulting image will be a component for one of the ADPs being computed.

The EADP is the set of all the ADPs resulting from the application of the diffusion process:

$$\mathrm{EADP}(\mathbf{X}) = \{\mathrm{ADP}(\boldsymbol{X}_1), ..., \mathrm{ADP}(\boldsymbol{X}_N)\} \qquad (10)$$

Algorithm 1 Pseudocode for the PCA computation

Input:
$\mathbf{I}$: Hypespectral image.
Output:
$\mathbf{X}$: PCA-reduced image (of N components)

Image centering phase
Obtain the sum of all pixels in each band of $\mathbf{I}$
Center pixels of $\mathbf{I}$ obtaining $\mathbf{I}_{\mathrm{cen}}$

PCA phase
Calculate the covariance matrix $\mathbf{I}_{\mathrm{cen}} \cdot \mathbf{I}_{\mathrm{cen}}^T$ of centered image
Calculate the matrix of eigenvectors $\mathbf{V}$
$\mathbf{X} \leftarrow \mathbf{I}_{\mathrm{cen}} \cdot \mathbf{V}$

Algorithm 2 Pseudocode for the EADP computation

Input:
$\mathbf{X}$: PCA-reduced image (of N components)
σ^2: Variance of the Gaussian filter.
$\{T_1, \ldots, T_c\}$: list of process times.
Output:
EADP($\mathbf{X}$): EADP of image $\mathbf{X}$.

$\mathrm{EADP} \leftarrow \emptyset$
for $i = 1 \rightarrow N$ **do**
  $\mathrm{ADP}_i \leftarrow \emptyset$

  Computation of diffusivity matrix
  Apply Gaussian filtering to $\boldsymbol{X}_i$, obtaining $\boldsymbol{X}_{i,\sigma}$
  Apply Scharr to $\boldsymbol{X}_{i,\sigma}$, obtaining $\nabla \boldsymbol{X}_{i,\sigma}$
  Obtain the contrast parameter k
  Compute the diffusivity matrix $\boldsymbol{A}(\nabla \boldsymbol{X}_{i,\sigma})$

  Computation of FED process
  $\boldsymbol{X}_i^{0,0} \leftarrow \boldsymbol{X}_i$
  for $c = 1 \rightarrow C$ **do**
    Compute the number of FED inner steps n using T_c
    Compute the step sizes τ_j
    Apply FED cycles and obtain $\boldsymbol{X}_i^{c,n}$
    $\mathrm{ADP}_{\mathrm{cen}} \leftarrow \boldsymbol{X}^{c,n} \cup \mathrm{ADP}_i$
  end for
  $\mathrm{EADP} \leftarrow \mathrm{ADP}_{\mathrm{cen}} \cup \mathrm{EADP}$
end for

B. Analysis of computational complexity

The generation of the ADPs requires the application of the diffusion algorithm to the input image resulting from the FE by PCA carried out in a previous step. Most of the complexity, thus, comes from the diffusion process itself, and is heavily dependent on the selected approach.

PCA is applied on the previously centered input data, using the SVD approach. The complexity for the application of the PCA to the input image is the combination of the complexities for each one of the steps: $O(WHB)$ for data centering and $O(WHB^2)$ for the computation of the covariance matrix. SVD requires $O(B^3)$ operations, and the multiplication to get the components is again $O(WHB^2)$. So, the final complexity is $O(B^3 + WHB^2)$.

For the FED scheme, the computational complexity is related to the cycle time for the approximation of a Gaussian filter using box filters, namely, to the selected process time T and the dimensions of the image. The complexity is $O(WHNC \log T)$.

The memory requirements of the algorithm depend only on the dimensions of the hyperspectral image being processed and the number of components after the feature reduction step. The memory usage remains constant during the complete execution and requires 4 buffers. The amount of memory required is approximately $4WHN$ words.

IV. GPU IMPLEMENTATION

This section discusses the specific details of the proposed CUDA GPU implementation for the creation of the EADP, along with the optimization techniques applied. The pseudocode for some of the more relevant functions and kernels is included for reference. Next to each line, a comment with one of the following: REG, exclusively register operations; SM, shared memory operations or GM, global memory operations, will indicate what resources the computation relies on.

CUDA is a parallel computing architecture and programming model developed by NVIDIA that allows the execution of relatively simple functions, called kernels, inside a GPU. The following is a list of the optimization techniques used to improve the performance of the CUDA code:

1) Minimization of memory allocations and deallocations.
2) Minimization of memory usage by performing some computations in place.
3) Data packing in order to reuse memory elements over multiple operations.
4) Vectorial instructions to maximize instruction parallelism and optimize memory accesses.
5) Reduction operations using warp-level primitives to avoid shared memory latency.
6) Use of shared memory atomics vs global memory ones due to specific hardware support since Maxwell.
7) Use of pinned memory for data transfers between CPU and GPU [34].
8) Use of high performance cuBLAS, cuSOLVER and cuSPARSE libraries wherever possible.

All the kernels were profiled using the NVIDIA Visual Profiler in order to analyze the performance and identify potential bottlenecks. The tool can also help to detect potential optimization spots based on their impact on the computation times. The optimization strategies were geared towards the high impact kernels.

A. PCA computation

Algorithm 3 Pseudocode for the PCA computation in CUDA

Input:
$\mathbf{I}$: Hyperspectral image stored in global memory.
B: Number of bands in hyperspectral image $\mathbf{I}$.
Output:
$\mathbf{X}$: PCA-reduced image (of N components) stored in global memory

Image centering phase
$Sums_b \leftarrow <\mathrm{reduce\_sum}> (\mathbf{I}, B)$ ▷ REG + GM
$\mathrm{I}_{\mathrm{cen}} \leftarrow <\mathrm{center}> (\mathbf{I}, Sums_b)$ ▷ SM + GM

PCA phase
$\mathbf{S} \leftarrow <\mathrm{gemm}> (\mathbf{I}_{\mathrm{cen}}, \mathbf{I}_{\mathrm{cen}}^T)$ ▷ SM + GM
$\mathbf{V} \leftarrow <\mathrm{gesvd}> (\mathbf{S})$ ▷ SM + GM
$\mathbf{X} \leftarrow <\mathrm{gemm}> (\mathbf{I}_{\mathrm{cen}}, \mathbf{V})$ ▷ SM + GM

The PCA computation process using CUDA is described in Algorithm 3. The implementation takes advantage of optimization 8 described in the previous subsection wherever possible in order to speed up the algorithm. The image pre-processing phase is implemented using a combination of two different CUDA kernels, one to compute the sums of all pixels in each

band and then a second one to use the previous result in order to center them.

The initial loading of data in memory is performed asynchronously and in parallel to the sum operation on each band and to the centering. In order to accomplish this, the image is divided into groups of consecutive bands that will be processed (loaded and then centered) using a CUDA stream. The $<$ reduce_sum $>$ kernel (line 1 of the pseudocode) computes the reduction operation. CUDA optimizations 4 and 5 are applied to this kernel. The centering kernel (line 2) will then center the elements in the bands.

The covariance matrix S is obtained by a matrix multiplication using the cuBLAS *gemm* function (line 3). The SVD decomposition is performed using the cuSOLVER *gesvd* function in order to obtain V (line 4). Lastly, we use *gemm* again to calculate $\mathbf{X}$ (line 5).

B. Diffusion computation

Algorithm 4 Pseudocode for EADP computation in CUDA

Input:
$\boldsymbol{X}_i$: PCA component, $i = 1..N$.
σ^2: Variance of the Gaussian filter.
$\{T_1, \ldots, T_c\}$: list of process times.
Output:
EADP($\mathbf{X}$): EADP of image $\mathbf{X}$.

1: EADP $\leftarrow \emptyset$
2: **for** i=1 $\rightarrow N$ **do**
3: ADP$_i \leftarrow \emptyset$

Computation of diffusivity matrix
4: $\boldsymbol{X}_{i,\sigma} \leftarrow <$ apply_row_conv $>(\boldsymbol{X}_i, G_\sigma(\boldsymbol{X}_i))$ ▷ SM + GM
5: $\boldsymbol{X}_{i,\sigma} \leftarrow <$ apply_col_conv $>(\boldsymbol{X}_{i,\sigma}, G_\sigma(\boldsymbol{X}_i))$ ▷ SM + GM
6: $\nabla\boldsymbol{X}_{i,\sigma} \leftarrow <$ scharr $>(\boldsymbol{X}_{i,\sigma})$ ▷ SM + GM
7: $max \leftarrow <$ reduce_max $>(\boldsymbol{X}_{i,\sigma})$ ▷ REG + GM
8: $histo \leftarrow <$ create_histogram $>(\nabla\boldsymbol{X}_{i,\sigma}, max)$ ▷ SM + GM
9: $k \leftarrow$ get_bin$(histo, 0.7)$
10: $\boldsymbol{A}(\nabla\boldsymbol{X}_{i,\sigma}) \leftarrow <$ pm2_coefficients $>(\nabla\boldsymbol{X}_{i,\sigma}, k)$ ▷ GM

Computation of FED process
11: $\boldsymbol{X}_i^0 \leftarrow \boldsymbol{X}_i$
12: **for** $c = 1 \rightarrow C$ **do**
13: $\tau_j \rightarrow$ fed_tau_by_process_time(T_c)
14: $\boldsymbol{X}_i^{c,0} \leftarrow \boldsymbol{X}_i^{c-1}$
15: **for** $j = 1 \rightarrow n$ **do**
16: $\Delta\boldsymbol{X}_i^{c,j} \leftarrow <$ fed_nld_step $>(\boldsymbol{X}_i^{c,j-1}, \boldsymbol{A}(\nabla\boldsymbol{X}_{i,\sigma}))$ ▷ SM + GM
17: $\boldsymbol{X}_i^{c+1,j+1} \leftarrow <$ fed_nld_update $>(\boldsymbol{X}_i^{c,j}, \Delta\boldsymbol{X}_i^{c,j})$ ▷ GM
18: **end for**
19: ADP$_c \leftarrow X^{c,n} \cup$ ADP$_c$
20: **end for**
21: EADP $\leftarrow$ ADP$_c \cup$ EADP
22: **end for**

The diffusion computation follows the pseudocode shown in Algorithm 2. The computation of the diffusivity matrix begins with the initial load of a principal component $\boldsymbol{X}_i$ into GPU memory. After the component is loaded, two convolutions with a Gaussian kernel are performed in order to smooth the original image (lines 4-5 in the pseudocode). Each convolution is performed by a kernel specifically designed to exploit the memory locality depending on the direction of the operation. Afterwards, the kernel computing the derivatives of the image obtains $\nabla\boldsymbol{X}_{i,\sigma}$ (line 6). The previous three kernels benefit from optimization strategy number 3. After that, the maximum value of the smoothed image, $G_\sigma(\boldsymbol{X}_i)$, is obtained with a reduction kernel (line 7), which applies optimizations 4 and 5. The maximum is used in the creation of an histogram. A new kernel will take $\nabla\boldsymbol{X}_{i,\sigma}$ and compute a histogram with the distances between the vertical and horizontal derivatives (line 8), atomically storing them in shared memory and then atomically adding them to the histogram in global memory (optimization strategy 6). The contrast parameter is calculated using CPU code (line 9), due to the low computational complexity of the task. Lastly, the diffusivity matrix is computed applying Eq. (4) to every pixel of the image (line 10).

The initial calculation of the number of steps and τ_j are executed on the CPU (line 13). The diffusion process in the GPU implementation performs the update of $\boldsymbol{X}_i^{c+1,j+1}$ iteratively within a kernel, using optimization number 3 (lines 16-17).

Algorithm 5 shows the kernel used for the computation of the Scharr derivatives in CUDA. The first step in the kernel is to compute the indices for the elements of the image (lines 1-2). Additionally, in order to be able to apply the kernel to the full image, the size of the image is increased by repeating two pixels in each dimension (lines 3-4). The required pixels are copied from global memory into shared memory for latter use (line 5). The values of **b** used in the Scharr convolution are copied to variables ensuring the use of registers (lines 8-11). Lastly, the horizontal and vertical derivatives are computed and stored in global memory (lines 12-13).

Algorithm 5 Pseudocode for the scharr kernel in CUDA

Input:
I: Image in global memory
Output:
$\boldsymbol{scharr}(\boldsymbol{I}_x)$: Derivative of **I** in the x direction
$\boldsymbol{scharr}(\boldsymbol{I}_y)$: Derivative of **I** in the y direction

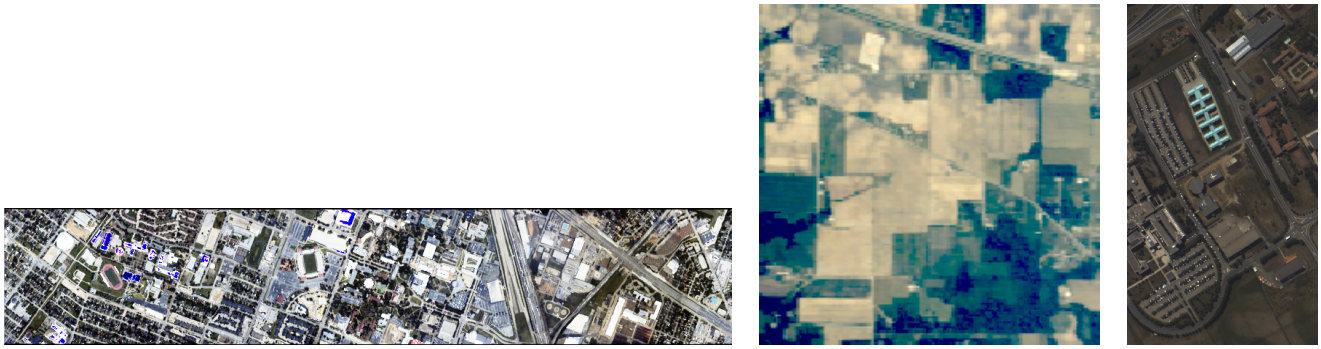
1: $idx \leftarrow bid.x * bdim.x + tid.x$ ▷ REG
2: $idy \leftarrow bid.y * bdim.y + tid.y$ ▷ REG
3: $xp \leftarrow (idx == 0?0 : (idx >= \mathbf{I}.width?\mathbf{I}.width - 1 : idx - 1))$ ▷ REG
4: $yp \leftarrow (idy == 0?0 : (idy >= \mathbf{I}.height?\mathbf{I}.height - 1 : idy - 1))$ ▷ REG
5: $b[tidx.y * (bdimx.x + 2) + tidx.x] \leftarrow \mathbf{I}[yp * \mathbf{I}.pitch + xp]$ ▷ SM
6: __syncthreads()
7: $bidx \leftarrow (tidx.y + 1) * (bdim.x + 2) + (tidx.x + 1)$ ▷ REG
8: $ul = b[bidx - bdim.x - 1];$ ▷ REG + SM
9: $ur = b[bidx - bdim.x + 1];$ ▷ REG + SM
10: $ll = b[bidx + bdim.x - 1];$ ▷ REG + SM
11: $lr = b[bidx + bdim.x + 1];$ ▷ REG + SM
12: $\boldsymbol{scharr}(\boldsymbol{I}_x)[idy*\mathbf{I}.pitch+idx] \leftarrow 3*(lr-ll+ur-ul)+10*(b[bidx+1] - b[bidx - 1])$ ▷ SM + GM
13: $\boldsymbol{scharr}(\boldsymbol{I}_y)[idy*\mathbf{I}.pitch+idx] \leftarrow 3*(lr+ll-ur-ul)+10*(b[bidx+BW] - b[bidx - BW])$ ▷ SM + GM

V. Performance evaluation

This section describes the experimental setup and the results. First of all, the CUDA implementation is evaluated in terms of execution time comparing it to a OpenMP implementation. To illustrate the effectiveness of the proposed approach, we include a brief characterization of the EADPs in terms of classification accuracy, structural similarity and image simplification. Finally, a comparison to other common techniques based on spatial-spectral information extraction from the literature is performed.

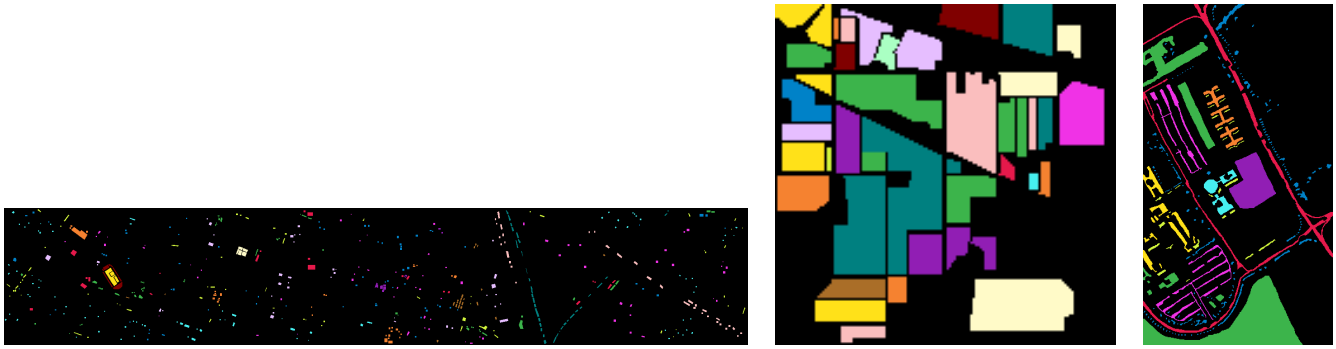
Three hyperspectral datasets were used to evaluate the algorithm proposed in this paper:

1) Houston University (Houston): The aerial view of Houston University was obtained by the CASI sensor. The



(a) (b) (c)

Fig. 3: Hyperspectral images commonly used for testing in remote sensing: (a) Houston, (b) IndianP, (c) PaviaU.



(a) (b) (c)

Fig. 4: Reference data for the test images: (a) Houston, (b) IndianP, (c) PaviaU.

image has a spatial resolution of 2.5 meters/pixel and it covers a spectral range from 380 to 1050 nm. The size of the image is 1905 x 349 pixels and 144 spectral bands. The reference information contains fifteen classes.

2) Indian Pines (IndianP): The mixed vegetation area of IndianP was obtained by NASA's AVIRIS sensor over the Indian Pines test site in North-western Indiana. The spatial resolution is 20 meters/pixel and it covers a spectral range from 400 to 2500 nm. IndianP dataset consists of 145 x 145 pixels and 220 spectral bands. The ground truth available is divided into sixteen classes.
3) Pavia University (PaviaU): acquired by the ROSIS-03 sensor over the city of Pavia, Italy. Its spatial resolution is 2.6 meters/pixel and covers the spectral range from 430 to 860 nm. PaviaU consists of 610 x 340 pixels and 103 spectral bands. The ground truth contains nine classes.

Figs. 3 and 4 show the false color composite images and reference data images corresponding to each of the datasets. For the reference data, pixels of the same class are depicted with the same color, as shown in Table I where the number of non-overlapping train and test samples used for classification are shown [3].

The experiments were carried out on a PC with a 6-core Intel i5 8400 CPU at 2.80 GHz and 32 GB of RAM, and a NVIDIA GeForce GTX 1060 with 6 GB. All experiments ran under Ubuntu Linux 16.04 64-bits and were compiled with GCC version 7.4.0, OpenMP 4.0 and CUDA toolkit 10.0. The code was built with the *-O3* flag and the additional CUDA flags *-arch=sm_61 –cudart=static –use_fast_math –expt-relaxed-constexpr* were used when applicable. Table II shows the technical specifications for the CPU and GPU used in the experiments.

A. Parameter selection

The classification process was performed using a SVM classifier and, more precisely, the LIBSVM [35] implementation. SVM classifiers have been found to provide similar results to other commonly used, non-parametric classifiers such as RF and can handle scenarios with a low number of training samples [36]. SVM is also presented as a standard non-contextual classifier for remote sensing classification [37]. The hyperparameters γ and C were selected by a 5-fold cross-validation [38] for each dataset via accuracy maximization with values in the range $C = [2^5, 2^3, \ldots, 2^{15}], \gamma = [2^3, 2^1, \ldots, 2^{15}]$.

The classification accuracy was evaluated in terms of the the standard measures: Overall Accuracy (OA), Average Accuracy (AA) and Kappa coefficient (κ). All the accuracy results are obtained as the average of 100 experiments.

In the experiments performed over the three datasets, special attention was paid to the optimal selection of parameters depending on the image. The number of principal components for each image was fixed to seven considering the trade-off between computational cost and classification accuracy obtained.

TABLE I: Houston, IndianP, and PaviaU datasets. Color codes for the classes and numbers of train and test sets [3].

	Houston			**IndianP**			**PaviaU**		
#	Classes	Train	Test	Classes	Train	Test	Classes	Train	Test
1.	Grass Healthy	198	1053	Alfalfa	15	31	Asphalt	548	6083
2.	Grass Stressed	190	1064	Corn-notill	50	1365	Meadows	540	18109
3.	Grass Synthetic	192	505	Corn-mintill	50	767	Gravel	392	1707
4.	Tree	188	1056	Corn	50	177	Trees	524	2540
5.	Soil	186	1056	Grass/pasture	50	426	Metal	265	1080
6.	Water	182	143	Grass-trees	50	673	Bare soil	532	4497
7.	Residential	196	1072	Grass-pasture-mowed	15	11	Bitumen	375	955
8.	Commercial	191	1053	Hay-windrowed	50	430	Bricks	514	3168
9.	Road	193	1059	Oats	15	5	Shadows	231	716
10.	Highway	191	1036	Soybean-notill	50	915			
11.	Railway	181	1054	Soybean-mintill	50	2386			
12.	Parking Lot 1	192	1041	Soybean-clean	50	525			
13.	Parking Lot 2	184	285	Wheat	50	158			
14.	Tennis Court	181	247	Woods	50	1209			
15.	Running Track	187	473	Bld-Grass-Trees	50	321			
16.				Stone-Steel	50	41			

TABLE II: Hardware specifications of the test system

Hardware	Intel Core i5 8400	GeForce GTX 1060
Compute Units	6 cores	10 MPs
Compute Width	256-bit (AVX2)	32 work-items
Concurrency	1 thread/core	4 warps/MP
Core clock (MHz)	2800	1506
RAM (GB)	32	6
L1 (KB)	384	48
L2 (KB)	1536	1536
L3 (KB)	9216	-
GFLOPS (peak)	268.8	4375
Bandwidth (GB/s)	41.6	192.2

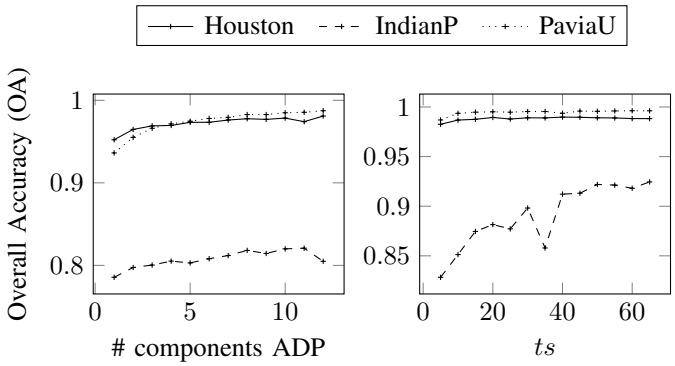


Fig. 5: OA values varying C, with ts=2 (left) and varying ts, with a fixed C= 8 (right).

The adequate number of components in an ADP depends on the process time values applied to the diffusion process. In order to simplify this parameter selection, the different process times used in each component were obtained by summing a fixed value or time step ts to the process time of the previous component. Process times were thus obtained according to $T_i = ts \times i$ with $i = 0, \ldots, C$.

The left plot in Fig. 5 shows the observed OA after modifying the number of components for a fixed process time step for the three images. As it can be seen, the accuracy keeps increasing as the number of components does. The greater C, the higher the expected accuracy due to the addition of new components providing more granulometry levels. The higher number of components comes at a cost, however, causing the size of the feature vector to grow substantially and leading to a much more resource intensive classification process.

The right plot in Fig. 5 shows how the accuracy evolves as the time step ts value increases (and thus the process time T) for a given ADP_i, while the other parameters remain constant. In general, the observed OA results show that the time step ts value is positively correlated with the accuracy resulting from the classification process. In contrast to the increase in the number of components, the increase in the step size provides additional accuracy without modifying the size of the profile, and therefore, without additional cost in the classification step.

The final results indicate that optimal or nearly optimal parameters can be selected by default and applied to all the datasets. In the generation of the profiles, the parameters $N = 7$, $C = 8$, $\sigma = 1$ and $ts = 65$ were used as a compromise between accuracy and number of components of the extended profile. The total ADP size is thus, 9 and the total EADP size is 63. These are the default values for all the experiments performed, unless otherwise stated.

B. CUDA performance comparison

In order to compare the performance of the GPU implementation to a high performance baseline, an OpenMP implementation of the algorithm aimed at scaling efficiently in modern multicore systems was developed. This subsection briefly describes the stages of the proposed method and provides time measurements from both the OpenMP CPU and CUDA GPU implementations for each one of them. The times shown correspond to the average of 20 repetitions per experiment.

Fig. 6 shows the scaling of the OpenMP implementation as the number of cores in the test system being used for the execution is increased. The obtained results are not linear due to some of the stages of the algorithm being memory bound or becoming memory bound as the amount of cores increases.

Table III displays a performance comparison between the CPU and the GPU implementations for all the datasets. The diffusion process is broken down into stages and the total

TABLE III: Average OpenMP CPU and CUDA GPU computation times (in milliseconds) broken down by stage (see Algorithm 4 for the lines corresponding to each stage that are between parenthesis in the table) for each scene during the generation of an EADP of 63 components.

Stage (lines)	Houston			IndianP			PaviaU		
	CPU	GPU	Speedup	CPU	GPU	Speedup	CPU	GPU	Speedup
Setup	0.003	0.42	0.006×	0.001	0.022	0.0257×	0.001	0.141	0.007×
Gaussian (4-5)	3.46	0.23	14.79×	0.54	0.05	11.82×	0.98	0.10	9.56×
Scharr (6)	5.01	0.16	31.30×	0.13	0.02	6.93×	1.59	0.06	26.16×
Contrast (7-9)	61.45	0.34	178.30×	1.59	0.07	22.40×	17.29	0.17	103.53×
Diffusivity (10)	0.65	0.14	4.58×	0.01	0.02	0.34×	0.25	0.05	4.72×
FED τ (13)	0.006	0.002	0.34×	0.003	0.01	0.53×	0.006	0.01	0.61×
FED (15-18)	69.32	16.35	8.51×	7.76	0.71	10.86×	29.12	5.33	5.46×
Cleanup	0.11	0.41	0.22×	0.001	0.02	0.06×	0.002	0.142	0.01×
Total	208.69	17.99	11.60×	10.02	0.91	10.99×	49.24	6.01	8.19×

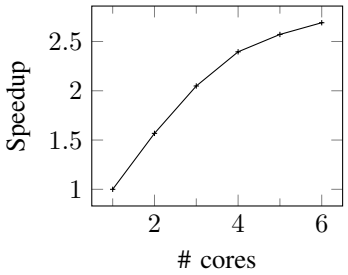


Fig. 6: Speedup for the OpenMP implementation for the Houston dataset image as the number of cores used in the execution increases.

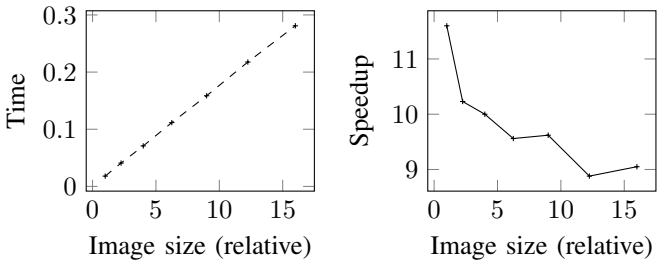


Fig. 7: Computation time and speedup of the proposed CUDA implementation for the Houston dataset as image size increases. The base size in the horizontal axis corresponds to the original size of the Houston dataset.

aggregated time (in milliseconds) for all iterations in each stage is displayed, as well as the speedup of the CUDA code over the OpenMP one. Results are shown for the case of extracting 7 principal components and generating an EADP of 63 components.

The Setup and Cleanup stages in Table III correspond to the initial loading of the principal component into device memory and the transfer of the diffused image to host memory respectively. The other stages reference the corresponding lines in Algorithm 4 next to the name.

The highest speedups can be observed in the Scharr and Contrast stages with up to $31.30\times$ and $178.30\times$ the performance of the OpenMP implementation. The massive parallelism of the GPUs together with the use of memory atomic operations provide high speedups during the execution of the Contrast stage. The more flexible memory access patterns in GPU aided by the use of shared memory, allow achieving a very significant performance gain for the execution of the $< scharr >$ kernel (shown in Algorithm 5).

It is worth noting that most of the computation time of the GPU implementation is spent during the Setup and Cleanup steps, where the data is being moved between the device and the host. The performance of the implementation is, thus, heavily penalized by transfers between host and device memories.

The CUDA implementation outperforms the CPU one by nearly an order of magnitude. Fig. 7 shows, on the left, the scaling of the computation time for the CUDA implementation being almost perfectly linear with regards to the image size; on the right, the speedup factor of the CUDA code over the OpenMP implementation displaying a small negative correlation with image size. The negative correlation is mostly related to the FED stage, with the OpenMP implementation showing better scalability than the CUDA one. The speedups achieved for all the datasets are comparable, with PaviaU lagging slightly behind. The differences in performance are attributed to the aspect ratio of the images, among other factors.

It is worth noting that, due to the requirements of the FED algorithm, the operations have to be performed in double precision arithmetic, which is severely limited in consumer grade NVIDIA GPUs: double precision performance is 1/32th of its single precision counterpart whereas in professional grade GPUs the performance is the expected 1/2.

Table IV shows the GPU occupancies broken down by kernel, with both the theoretical and the observed values. Occupancies were obtained using the NVIDIA Profiling Tool and represent the weighted average of the executions for each kernel. We can see that all the occupancy values for the kernels involved in the diffusion process stay at reasonable levels: two kernels achieve occupancies of only ≈ 0.7. In the case of $< create\_histogram >$, the reason is, partially, due to the high number of collisions in the atomic functions that update the counter for the histogram bins, yielding a very low efficiency in memory operations and also the high number of registers in use per execution of the kernel. There are two other kernels, $< apply\_row\_conv >$ and $< scharr >$, with occupancy limitations due to the number of blocks that can be concurrently executed in the GPU. Most of the kernels have no

TABLE IV: Occupancies broken down by kernel for the Houston scene for the execution of the diffusion algorithm during the generation of an EADP of 63 components.

Kernel (lines)	#	Theoretical Occupancy	Achieved Occupancy	Threads per block	Registers per thread	SM per Block (KiB)	Limiter
apply_row_conv (4)	56	0.94	0.76	164	32	1.28	Max Warps or Max Blocks per Multiprocessor
apply_col_conv (5)	56	1	0.93	256	32	11	None
scharr (6)	56	0.94	0.78	612	22	9.56	Max Warps or Max Blocks per Multiprocessor
reduce_max (7)	56	1	0.84	1024	32	0	None
create_histogram (8)	56	0.75	0.74	512	35	2	Registers per Multiprocessor
pm2_coefficients (10)	56	1	0.83	256	24	0	None
fed_nld_step (16)	3178	1	0.83	512	25	7.97	None
fed_nld_update (17)	3178	1	0.80	512	10	0	None

TABLE V: Efficiency metrics broken down by kernel for the Houston scene for the execution of the diffusion algorithm during the generation of an EADP of 63 components.

Kernel (lines)	#	Glob. load Efficiency	Glob. store Efficiency	Shared Efficiency	Warp Exec. Efficiency	Non-predicated Warp Exec. Efficiency
apply_row_conv (4)	56	0.64	1	0.95	0.93	0.82
apply_col_conv (5)	56	0.72	1	0.97	0.97	0.86
scharr (6)	56	0.78	0.94	0.92	0.94	0.92
reduce_max (7)	56	0.88	n/a	n/a	0.74	0.72
create_histogram (8)	56	1	n/a	0.09	0.97	0.94
pm2_coefficients (10)	56	1	1	n/a	1	0.97
fed_nld_step (16)	3178	0.78	0.95	0.92	0.95	0.94
fed_nld_update (17)	3178	1	1	n/a	0.98	0.95

TABLE VI: FLOPS and bandwidth usage of CUDA implementation.

Kernel (lines)	GFLOPS DP	Relative to peak	Bandwidth (GBps)	Relative to peak
apply_row_conv (4)	20.94	0.15	81.25	0.42
apply_col_conv (5)	6.95	0.05	96.42	0.50
scharr (6)	59.98	0.44	82.23	0.43
reduce_max (7)	0	0	46.61	0.24
create_histogram (8)	24.43	0.18	81.23	0.42
pm2_coefficients (10)	66.06	0.48	114.04	0.59
fed_nld_step (16)	85.24	0.62	65.37	0.34
fed_nld_update (17)	6.37	0.05	142.35	0.74

limiting factors and achieve successful performances of over 0.8. Even though high occupancy does not guarantee a better performance [34], a low occupancy level can show potential bottlenecks that may be limiting the performance of a kernel.

Table V shows the execution efficiency metrics broken down by kernel, as reported by NVIDIA's nvprof tool. It can be observed that all the metrics for the kernels remain at good or reasonable values in most cases. The efficiency in the access to shared memory (Shared Efficiency in the table) for the $< create\_histogram >$ kernel is the lowest. The reason is that many writes that are not split evenly across the shared memory banks are issued, producing a high number of conflicts in the bank accesses.

Table VI shows the number of GFLOPS, GFLOPS relative to the theoretical peak, combined read/write bandwidth usage and combined read/write bandwidth usage relative to the theoretical peak. We would like to note that the kernels involved in the computation of the FED process, even when achieving high memory efficiency and utilization, may yield significantly lower performance than the theoretical peak. The reason behind this behavior lies in the existence of other integer and single precision arithmetic operations, as well as flow control instructions that are part of the logic of these kernels (the so called instruction mix). The kernels that exhibit performance closer to the theoretical peaks are $< fed\_nld\_step >$ and $< fed\_nld\_update >$ for the GFLOPS and bandwidth metrics, respectively. In the former case, the kernel achieves the highest GFLOPS value due to the high ratio of double precision arithmetic operations to other executed instructions. In the latter, the bandwidth measured is the highest as a result of the simplicity of the arithmetic involved in the kernel execution compared to the amount of memory accesses performed.

C. Image simplification and structural Similarity

In this section we carry out a brief study of the image simplification and the structural similarity between the original feature reduced image and the resulting profile after applying the proposed algorithm. The metrics employed for this purpose are briefly described in the next lines:

1) Simplification Ratio (SR): SR is a metric computed as the ratio between the number of flat zones of a transformed image (a component of the profile, for example) and the number of flat zones of the original image from which it was computed. The number of flat zones corresponds to the number of connected components in

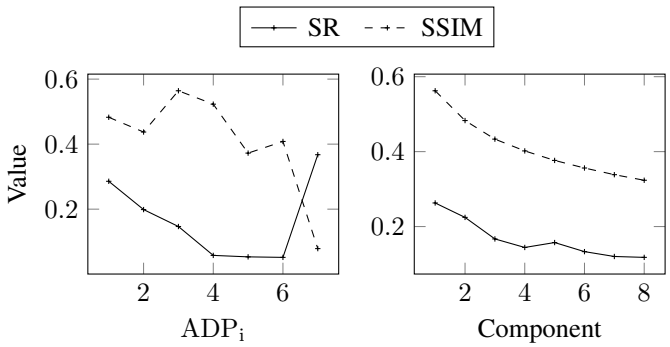


Fig. 8: Average SR and SSIM values per ADP_i (left) and per component (right) for IndianP after the generation of an EADP with ts=5.

an image obtained after applying a binary thresholding using Otsu's method [39]. SR ranges from 0 to 1 and higher values imply a higher similarity between the images.

2) Structural Similarity Index Metric (SSIM): SSIM is a metric that aims to detect the changes in spatial structures between two images [40]. SSIM is calculated based on the pixel by pixel difference between both images and provides a decimal value in the range from -1 to 1, where 1 means two identical images or sets of data.

Both, SR and SSIM, are calculated using the following formula:

$$\hat{r}_i = \frac{1}{C}\sum_{j=1}^{C} r_{i,j} \tag{11}$$

if we compute the average per ADP or

$$\hat{r}_j = \frac{1}{N}\sum_{i=1}^{N} r_{i,j} \tag{12}$$

if we compute the average value per component. The value $r_{i,j}$ is defined as:

$$r_{i,j} = \mathrm{reg}(\mathrm{ADP}_{i,j})/\mathrm{reg}(\mathrm{PCA}(\boldsymbol{X}_i)) \tag{13}$$

for the SR denoting $\mathrm{reg}(\boldsymbol{X}_i)$ the number of flat regions for the image $\boldsymbol{X}_i$ and

$$r_{i,j} = \mathrm{SSIM}(\mathrm{ADP}_{i,j}, \mathrm{PCA}(\boldsymbol{X}_i)) \tag{14}$$

for SSIM, respectively.

The experimental results obtained for the IndianP dataset are shown in Fig. 8. The graph on the left shows the values of the indexes calculated between each component of an ADP an the principal component from with the ADP is built, i.e., is calculated over a row of Figure 2 for each ADP and averaged as shown in Equation 11. We can observe that for all the ADP_i profiles a high rate of simplification (less than 50% of the initial number of regions remain) and a relatively low similarity are obtained. The graph on the right side of the figure shows the averaged values per column of Figure 2, as defined in Equation 12. The main observation is that both, SR and SSIM values, decrease as we advance through the components of the profile, which is the expected behavior as T increases in each diffusion step. Since the diffusion process alters the concentration of grey values in the pixels, the morphological details are progressively removed as the well-defined contours present in the original scene are smoothed. This causes a higher number of connected components in the image, thus reducing the number of flat zones.

SSIM is calculated between images at the pixel level, so the sensitivity towards changes is high. Whereas a diffusion process may not affect the number of flat areas in an image if, for example, the original image has a low number of flat zones, the only way for a SSIM index to be equal for both images is that both images are the same. In general, any diffusion process will cause the SSIM index value to monotonically decrease.

TABLE VII: Comparison of the proposed EADP algorithm to other approaches based on morphological profiles in terms of classification accuracy [3].

Dataset	Method	**OA**	**AA**	$\boldsymbol{\kappa}$
Houston	EADP	98.82	98.82	98.72
	EMP	80.01	82.78	78.34
	$\mathrm{EAP_a}$	79.50	82.47	77.70
	$\mathrm{EEP_a}$	80.32	83.36	78.66
	EMAP	78.92	82.23	77.21
	EMEP	80.83	83.64	79.20
IndianP	EADP	92.45	95.27	91.34
	EMP	91.99	95.04	90.85
	$\mathrm{EAP_a}$	91.38	93.54	90.15
	$\mathrm{EEP_a}$	92.99	95.58	91.99
	EMAP	91.65	95.15	90.46
	EMEP	93.70	96.00	92.79
PaviaU	EADP	99.61	99.66	99.47
	EMP	91.82	93.54	89.12
	$\mathrm{EAP_a}$	90.33	93.47	87.71
	$\mathrm{EEP_a}$	94.82	96.17	93.32
	EMAP	93.52	94.82	91.65
	EMEP	95.46	96.57	94.07

D. Classification accuracy

To evaluate the quality of EADPs, a comparison to five different algorithms that are widely used in the field for spatial-spectral information extraction was performed.

Table VII compares the classification accuracy in terms of OA, AA and $\boldsymbol{\kappa}$ to five different approaches in the literature. The results shown in the table are extracted from [3] except for EADP, the approach presented in this paper. The methods considered calculate different kinds of profiles. In the case of EMP [41], the morphological features of the image are extracted by performing morphological transformations (opening and closings) using SEs of different sizes. In the case of $\mathrm{EAP_a}$ [42] the profile is built by applying different AFs over a single attribute, in this case area. For the case of $\mathrm{EEP_a}$ [11] AFs are replaced by EFs. The case of EAPs but considering multiples attributes to build the same profile is called EMAP [42] in the table. Finally, EMEP [43] applies several EFs over multiple attributes. The results show that the proposed profile, denoted as EADP in the figure, improves the classification accuracy results for all the images except for the case of IndianP, for which the result is similar to the one obtained by the other methods. IndianP is the image with the most regular spatial

structures in the image, that are easily extracted by performing morphological transformations.

VI. Conclusion

In this paper, a computationally efficient GPU algorithm for the generation of Anisotropic Diffusion Profiles (ADPs) based on Fast Explicit Diffusion (FED) is proposed. The first step is a feature extraction process by computing Principal Component Analysis (PCA). The profiles are then computed by applying multiple instances of nonlinear diffusion to the bands of a hyperspectral image, which are subsequently stacked to generate an Extended Anisotropic Diffusion Profile (EADP). The characterization of these profiles based on analyzing the parameters involved in the diffusion filtering is also described. The proposed CUDA GPU implementation achieves a performance up to $11.60\times$ higher than the OpenMP reference implementation for the Houston dataset considering an EADP of 63 components. The classification accuracy obtained achieves values higher than the standard techniques for spatial information extraction in the literature, with values of up to 99.61% for the Pavia University dataset.

Several future research lines that would benefit from the current proposal have also been considered. A multi-GPU implementation of the EADPs that will further decrease the computational time is planned. The GPU algorithm proposed in this paper is also applicable as part of the HSI-KAZE [44] registration algorithm for hyperspectral images. In HSI-KAZE, nonlinear diffusion filtering replaces Gaussian filters in the creation of the scale space. The computational efficiency of the resulting registration algorithm will also benefit from the aforementioned multi-GPU implementation.

Supplemental data

The underlying research materials for this paper can be accessed at https://wiki.citius.usc.es/hiperespectral:eadp_gpu.

Acknowledgment

This work was supported in part by the Consellería de Educación, Universidade e Formación Profesional [grant numbers GRC2014/008, ED431C 2018/19, and ED431G/08] and Ministerio de Economía y Empresa, Government of Spain [grant number TIN2016-76373-P]. All are co–funded by the European Regional Development Fund (ERDF).

Bibliography

[1] Y. Lanthier, A. Bannari, D. Haboudane, J. R. Miller, and N. Tremblay, "Hyperspectral data segmentation and classification in precision agriculture: A multi-scale analysis," in *IGARSS 2008-2008 IEEE International Geoscience and Remote Sensing Symposium*, vol. 2. IEEE, 2008, pp. II–585.

[2] F. Wagner, A. Sanchez, Y. Tarabalka, R. Lotte, M. Ferreira, M. Aidar, M. Gloor, O. Phillips, and L. Aragão, "Using convolutional network to identify tree species related to forest disturbance in a neotropical forest with very high resolution multispectral images," in *AGU Fall Meeting Abstracts*, 2018.

[3] P. Ghamisi, E. Maggiori, S. Li, R. Souza, Y. Tarabalka, G. Moser, A. De Giorgi, L. Fang, Y. Chen, M. Chi *et al.*, "Frontiers in spectral-spatial classification of hyperspectral images," *IEEE Geoscience and Remote Sensing Magazine*, 2018.

[4] G. A. Shaw, "Spectral imaging for remote sensing," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 3–28, 2003.

[5] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 528–544, 2011.

[6] Y. Ma, L. Chen, P. Liu, and K. Lu, "Parallel programing templates for remote sensing image processing on GPU architectures: design and implementation," *Computing*, vol. 98, no. 1-2, pp. 7–33, 2016.

[7] M. Pesaresi and J. A. Benediktsson, "A new approach for the morphological segmentation of high-resolution satellite imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 2, pp. 309–320, 2001.

[8] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 11, pp. 3804–3814, 2008.

[9] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson, "Classification of hyperspectral data from urban areas based on extended morphological profiles," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 3, pp. 480–491, 2005.

[10] M. Dalla Mura, J. A. Benediktsson, B. Waske, and L. Bruzzone, "Morphological attribute profiles for the analysis of very high resolution images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 10, pp. 3747–3762, 2010.

[11] P. Ghamisi, R. Souza, J. A. Benediktsson, X. X. Zhu, L. Rittner, and R. A. Lotufo, "Extinction profiles for the classification of remote sensing data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 5631–5645, 2016.

[12] G. Licciardi, P. R. Marpu, J. Chanussot, and J. A. Benediktsson, "Linear versus nonlinear PCA for the classification of hyperspectral data based on the extended morphological profiles," *IEEE Geoscience and Remote Sensing Letters*, vol. 9, no. 3, pp. 447–451, 2012.

[13] A. Villa, J. A. Benediktsson, J. Chanussot, and C. Jutten, "Hyperspectral image classification with independent component discriminant analysis," *IEEE transactions on Geoscience and remote sensing*, vol. 49, no. 12, pp. 4865–4876, 2011.

[14] J. Zabalza, J. Ren, Z. Wang, S. Marshall, and J. Wang, "Singular spectrum analysis for effective feature extraction in hyperspectral imaging," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 11, pp. 1886–1890, 2014.

[15] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "KAZE features," in *European Conference on Computer Vision*.

Springer, 2012, pp. 214–227.

[16] J. Weickert, *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998, vol. 1.

[17] L. M. Bruce and J. Li, "Wavelets for computationally efficient hyperspectral derivative analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 7, pp. 1540–1546, 2001.

[18] J. M. Duarte-Carvajalino, P. E. Castillo, and M. Velez-Reyes, "Comparative study of semi-implicit schemes for nonlinear diffusion in hyperspectral imagery," *IEEE Transactions on Image Processing*, vol. 16, no. 5, pp. 1303–1314, 2007.

[19] S. Velasco-Forero and V. Manian, "Improving hyperspectral image classification using spatial preprocessing," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 2, pp. 297–301, 2009.

[20] S. Acton and J. Landis, "Multi-spectral anisotropic diffusion," *International Journal of Remote Sensing*, vol. 18, no. 13, pp. 2877–2886, 1997.

[21] F. Mirzapour and H. Ghassemian, "Hyperspectral image classification using profiles based on partial differential equations," in *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*. IEEE, 2015, pp. 288–292.

[22] Y. Wang, R. Niu, and X. Yu, "Anisotropic diffusion for hyperspectral imagery enhancement," *IEEE Sensors Journal*, vol. 10, no. 3, pp. 469–477, 2010.

[23] J. Weickert, "Theoretical foundations of anisotropic diffusion in image processing," in *Theoretical Foundations of Computer Vision*. Springer, 1996, pp. 221–236.

[24] J. Weickert, B. T. H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 398–410, 1998.

[25] S. Grewenig, J. Weickert, and A. Bruhn, "From box filtering to fast explicit diffusion," in *Joint Pattern Recognition Symposium*. Springer, 2010, pp. 533–542.

[26] Á. Ordóñez, F. Argüello, and D. B. Heras, "GPU accelerated FFT-based registration of hyperspectral scenes," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 4869–4878, 2017.

[27] S. Bernabe, S. Sanchez, A. Plaza, S. López, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 3, pp. 1386–1398, 2013.

[28] A. S. Garea, D. B. Heras, and F. Argüello, "GPU classification of remote-sensing images using kernel elm and extended morphological profiles," *International journal of remote sensing*, vol. 37, no. 24, pp. 5918–5935, 2016.

[29] E. Martel, R. Lazcano, J. López, D. Madroñal, R. Salvador, S. López, E. Juarez, R. Guerra, C. Sanz, and R. Sarmiento, "Implementation of the principal component analysis onto high-performance computer facilities for hyperspectral dimensionality reduction: Results and comparisons," *Remote Sensing*, vol. 10, no. 6, p. 864, 2018.

[30] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[31] J. Weickert, "Efficient image segmentation using partial differential equations and morphology," *Pattern Recognition*, vol. 34, no. 9, pp. 1813–1824, 2001.

[32] P. Gwosdek, S. Grewenig, A. Bruhn, and J. Weickert, "Theoretical foundations of gaussian convolution by extended box filtering," in *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer, 2011, pp. 447–458.

[33] J. M. Duarte-Carvajalino, G. Sapiro, M. Vélez-Reyes, and P. E. Castillo, "Multiscale representation and segmentation of hyperspectral imagery using geometric partial differential equations and algebraic multigrid methods," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 8, pp. 2418–2434, 2008.

[34] N. Corporation. (2018) CUDA C best practices guide. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf

[35] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[36] P. Ghamisi, J. Plaza, Y. Chen, J. Li, and A. J. Plaza, "Advanced spectral classifiers for hyperspectral images: A review," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 1, pp. 8–32, 2017.

[37] P. Ghamisi, E. Maggiori, S. Li, R. Souza, Y. Tarablaka, G. Moser, A. De Giorgi, L. Fang, Y. Chen, M. Chi *et al.*, "New frontiers in spectral-spatial hyperspectral image classification: The latest advances based on mathematical morphology, markov random fields, segmentation, sparse representation, and deep learning," *IEEE Geoscience and Remote Sensing Magazine*, vol. 6, no. 3, pp. 10–43, 2018.

[38] H. Trevor, T. Robert, and F. JH, "The elements of statistical learning: data mining, inference, and prediction," 2009.

[39] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[40] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

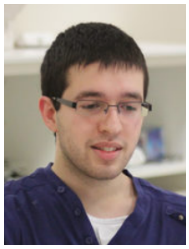
[41] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Advances in spectral-spatial classification of hyperspectral images," *Proceedings of the IEEE*, vol. 101, no. 3, pp. 652–675, 2013.

[42] P. Ghamisi, M. Dalla Mura, and J. A. Benediktsson, "A survey on spectral–spatial classification techniques based on attribute profiles," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2335–2353, 2015.

[43] P. Ghamisi, R. Souza, J. A. Benediktsson, L. Rittner, R. Lotufo, and X. X. Zhu, "Hyperspectral data classification using extended extinction profiles," *IEEE Geo-*

science and Remote Sensing Letters, vol. 13, no. 11, pp. 1641–1645, 2016.

[44] Á. Ordóñez, F. Argüello, and D. B. Heras, "Alignment of hyperspectral images using KAZE features," *Remote Sensing*, vol. 10, no. 5, p. 756, 2018.



Álvaro Acción received the B.S. in Computer Science and the M.S. in Big Data Technologies from the University of Santiago de Compostela, Santiago de Compostela, Spain, in 2014 and 2017, respectively, where he is currently working towards the Ph.D. degree as an assistant researcher of Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS). His main research interests include image analysis and processing.

Dora B. Heras (M'17) received the M.S. degree in physics in 1994 and the Ph.D. degree in 2000 from the University of Santiago, Santiago, Spain. She is currently an Associate Professor with the Department of Electronics and Computer Engineering, University of Santiago. Her research interests include parallel and distributed computing, software optimization techniques for emerging architectures and image processing especially focused on remote sensing images and she has published extensively on these topics. Dr. Heras has been a member of several relevant international conference committees and she is currently a member of the Steering Committee of the Euro-Par Conference in charge of Workshops as well as a Referee for several journals on image processing and remote sensing.

Francisco Argüello received the B.S. and Ph.D. degrees in physics from the University of Santiago, Santiago, Spain, in 1988 and 1992, respectively. He is currently an Associate Professor with the Department of Electronic and Computer Engineering, University of Santiago. His current research interests include signal and image processing, computer graphics, parallel and distributed computing, and quantum computing