International Journal of Remote Sensing

ISSN: 0143-1161 (Print) 1366-5901 (Online) Journal homepage: http://www.tandfonline.com/loi/tres20

GPU classification of remote-sensing images using kernel ELM and extended morphological profiles

Alberto S. Garea, Dora B. Heras & Francisco Argüello

To cite this article: Alberto S. Garea, Dora B. Heras & Francisco Argüello (2016) GPU classification of remote-sensing images using kernel ELM and extended morphological profiles, International Journal of Remote Sensing, 37:24, 5918-5935, DOI: 10.1080/01431161.2016.1251629

To link to this article: http://dx.doi.org/10.1080/01431161.2016.1251629

Published online: 08 Nov 2016.

INTERNATIONAL JOURNAL OF REMOTE SENSING, 2016
VOL. 37, NO. 24, 5918–5935
http://dx.doi.org/10.1080/01431161.2016.1251629

GPU classification of remote-sensing images using kernel ELM and extended morphological profiles

Alberto S. Garea[a], Dora B. Heras[a] and Francisco Argüello[b]

[a]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain; [b]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, Spain

ABSTRACT
Nowadays, the use of hyperspectral sensors has been extended to a variety of applications such as the classification of remote-sensing images. Recently, a spectral–spatial classification scheme (ELM-EMP) based on Extreme Learning Machine (ELM) and Extended Morphological Profiles (EMPs) computed using Principal Component Analysis (PCA) and morphological operations has been introduced. In this work, an efficient implementation of this scheme over commodity Graphics Processing Units (GPUs) is shown. Additionally, several techniques and optimizations are introduced to improve the accuracy of the classification. In particular, a scheme using an ELM classifier based on kernels (KELM) and EMP is presented (KELM-EMP). Similar schemes adding a spatial regularization process (KELM-EMP-S and ELM-EMP-S) are also proposed. Moreover, two PCA algorithms have been compared in both accuracy and speed terms. Regarding the GPU projection, different techniques and optimizations have been applied such as the use of optimized Compute Unified Device Architecture (CUDA) libraries or a block-asynchronous execution technique. As a result, the accuracy obtained by the two proposed schemes (ELM-EMP-S and KELM-EMP-S) is better than for the original scheme ELM-EMP and the execution time has been significantly reduced.

ARTICLE HISTORY
Received 13 January 2016
Accepted 12 October 2016

1. Introduction

The advances on image sensor technology have made it possible to extend the range of electromagnetic spectrum that can be captured, going from a few bands in multispectral images to hundreds of bands in hyperspectral images (Landgrebe 2002). This wide range provides more information that can be used to improve the recognition and classification of materials. Owing to the high dimensionality of hyperspectral images, specific techniques are needed to take advantage of all this information (Fauvel et al. 2013).

In the field of neural networks, Extreme Learning Machine (ELM) is a term used to describe a class of Single-hidden Layer Feedforward Neural Networks (SLFNs) with random weights (Huang 2014). It has been shown (Tamura and Tateishi 1997; Huang

CONTACT Alberto S. Garea jorge.suarez.garea@usc.es Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago, Spain

2003) that SLFNs (with *N* hidden nodes) with randomly chosen input weights and hidden layer biases can exactly learn *N* distinct observations (Huang, Zhu, and Siew 2006). The same authors also proposed the use of a pseudo-inverse in the ELM algorithm to avoid the low inference of the back-propagation training algorithms. In the ELM, the assignment of random weights between the input layer and the hidden layer produces a wide variation in the classification accuracy in different trials using the same number of hidden nodes. To avoid this, Huang et al. (2012) proposed the use of a kernel function for the ELM instead of the hidden layer.

One additional advantage of ELM is that it is an extremely suitable algorithm to be implemented on commodity Graphical Processing Units (GPUs) and other parallel architectures because the required operations are mostly matrix operations that can be computed in independent blocks, i.e. without data dependencies among them. Van Heeswijk et al. (2011) developed a GPU implementation of some parts of a classification scheme based on executing hundreds of ELM ensembles. Later, a complete and efficient GPU implementation of ELM to classify hyperspectral images was published (López-Fandiño et al. 2015). Nevertheless, for the kernel version of ELM, no GPU implementations have been published.

The accuracy of the results provided by a pixel-wise classification can be improved by adding spatial information to the classifier (Plaza et al. 2009; Fauvel et al. 2013; Dalla Mura et al. 2011; Palmason et al. 2005). This means that the decision to assign a pixel to a specific class is based both on the spectral feature, which is the pixel value, and on certain information extracted from the pixel's neighbourhood that can be considered spatial information. The spatial methods include those based on segmentation as the watershed transform (Zhang, Feng, and Le 2008; Tarabalka, Chanussot, and Benediktsson 2010; López-Fandiño et al. 2015) and Evolutionary Cellular Automata based Segmentation (ECAS-II) (Priego, Bellas, and Duro 2015), based on partitional clustering techniques (Tarabalka, Benediktsson, and Chanussot 2009), based on minimum spanning forest (Fauvel et al. 2013), or based on local filtering (Kang, Li, and Benediktsson 2014).

The spatial information can also be extracted from hyperspectral data using the tools of mathematical morphology. Two widely used morphological operators are opening and closing, which are based on the fundamental operations of erosion and dilation. From these basic operations, the so-called Morphological Profile (MP) can be constructed (Quesada-Barriuso, Argüello, and Heras 2014; Pesaresi and Benediktsson 2001; Soille and Pesaresi 2002; Palmason et al. 2005). A MP contains information of the structures of the image at different resolution sizes. In remote sensing, MPs are usually computed from hyperspectral data using Principal Component Analysis (PCA) (Benediktsson, Pesaresi, and Amason 2003; Fauvel et al. 2013; Marpu et al. 2012). If several components are retained, the MPs obtained for each Principal Component (PC) can be used all together in one Extended Morphological Profile (EMP) (Benediktsson, Palmason, and Sveinsson 2005; Licciardi et al. 2011). This can be further generalized in order to model the spatial information more accurately. For example, a morphological Attribute Profile (AP) is created in the same way as the MP but considering the attribute operators (Dalla Mura et al. 2010). In Dalla Mura et al. (2011), an Extended Morphological Attribute Profile (EMAP) is created using morphological attribute filters. In Quesada-Barriuso, Argüello, and Heras (2014) and Benediktsson, Pesaresi, and Amason (2003), spectral–spatial classification schemes based on SVM as the classifier and introducing the information provided by EMPs for hyperspectral remote-sensing images are presented. In the case of Chen et al. (2014) instead of EMP, Gabor features are extracted from the components generated by PCA and directly concatenated to the original image, and the classification is performed by kernel ELM (KELM). This last article also presents a multi-hypothesis prediction-based KELM classifier. In Argüello and Heras (2015), a spatial–spectral ELM-based classification scheme for hyperspectral remote-sensing images is presented that integrates the information provided by an EMP. The profile is created from components generated using PCA. The proposed spectral–spatial classifier allows different weights for both spatial and spectral features.

In this article, three main tasks have been addressed. The first one has been the improvement of the scheme presented in Argüello and Heras (2015) in terms of execution time in the Central Processing Unit (CPU). Thus, two PCA methods have been compared. The second task has been the development of a similar scheme based on ELM with kernel functions. These kernel functions replace the generation of random weights in the original ELM-based scheme. The third task has been the efficient implementation on GPU of the previously described schemes to reduce the execution times. The rest of this article is organized as follows. Section 2 describes the algorithms required. In Section 3 we present the implementation of the classification scheme in GPU. The evaluation is performed in Section 4. And, finally, Section 5 presents the conclusions.

2. Spectral–spatial ELM-based classification

In this section, the different steps involved in the classification scheme under study that we call ELM-EMP, as in Argüello and Heras (2015) (see Figure 1), are explained. This process starts with the extraction of the spatial information. The first step is to reduce the dimensionality of the hyperspectral image using PCA. The second step is to construct the EMP by morphological operations. Both spectral and spatial data contributions are adjusted using a composite feature mapping method, called weighted concatenation. It allows one to assign different weights to the different sources of data, k_w and k_s

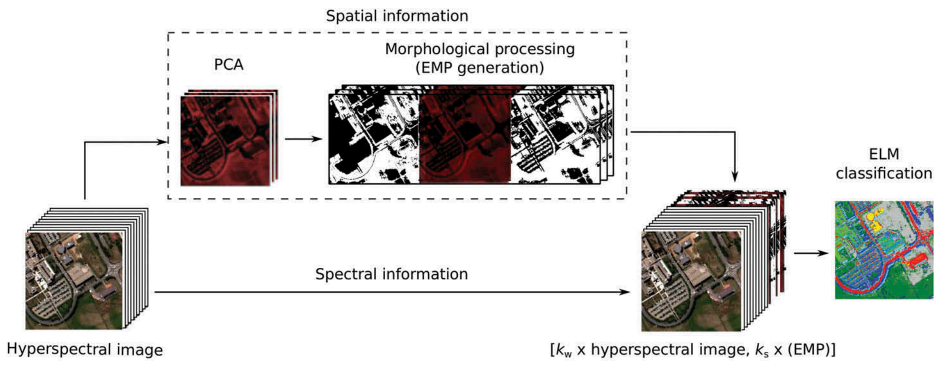
Spatial information
PCA
Morphological processing (EMP generation)
ELM classification
Spectral information
Hyperspectral image
[k_w x hyperspectral image, k_s x (EMP)]

Figure 1. Spectral–spatial classification scheme based on ELM and composite feature mappings (ELM-EMP).

for the spectral and the spatial sources, respectively. Finally, the classification by ELM is performed.

2.1. *Principal component analysis*

The main idea of PCA is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much of the variation present in the data set as possible.

Mathematically, there are several methods to compute the PCA. It can be performed by Single Value Decomposition (SVD) of the data set or by Eigenvalue Decomposition (EVD) of the covariance matrix of the data set, which is the method used in this article. Both methods need a centred data set (Abdi and Williams 2010).

Let us assume that data matrix $\mathbf{X}$ is centred, i.e. column arithmetic means have been subtracted from each column of $\mathbf{X}$ and are now equal to zero. Then the covariance matrix $\mathbf{A} = \mathbf{X}^{\mathbf{T}}\mathbf{X}/n$ is computed, where n is the number of features. It can be diagonalized as follows:

$$\mathbf{A} = \mathbf{USV}^{\mathbf{T}}, \tag{1}$$

where the diagonal elements of $\mathbf{S}$ are the eigenvalues of $\mathbf{A}$ and the first columns of $\mathbf{U}$ and $\mathbf{V}$ are the left and right eigenvectors of $\mathbf{A}$. Projections of the data on the principal axes are called PCs, also known as scores.

An alternative method to calculate PCA is to use an iterative algorithm. The classical Gram–Schmidt (CGS) algorithm recursively constructs a set of orthogonal basis vectors for the subspace spanned by a given set of linearly independent normalized vectors (Golub and Van Loan 1996). The CGS can be formulated using matrix-vector operations and hence it is suitable for parallel computation (Lingen 2000). Owing to its rounding error, the CGS algorithm is numerically unstable, but the stability can be achieved applying it iteratively (Lingen 2000).

In this article, the comparison between a PCA method using EVD and an iterative PCA method based on the Gram–Schmidt (GS) re-orthogonalization process called GS-PCA (Andrecut 2009) is shown.

2.2. *Extended morphological profile*

Morphological transformations have been proposed to use spatial information in remote-sensing classification (Pesaresi and Kanellopoulos 1999; Benediktsson, Pesaresi, and Amason 2003; Benediktsson, Palmason, and Sveinsson 2005). In Benediktsson, Pesaresi, and Amason (2003), the MP was introduced and it was extended to multi-dimensional images in Benediktsson, Palmason, and Sveinsson (2005) in order to extract information about the contrast and the size of the structures present in the image. The MP of order n from the image I can be expressed as

$$\mathrm{MP}^{(n)}(I) = \{\gamma_r^{(n)}(I), \ldots, \gamma_r^{(1)}(I), I, \phi_r^{(1)}(I), \ldots, \phi_r^{(n)}(I)\}, \tag{2}$$

where $\gamma_r^{(i)}$ and $\phi_r^{(i)}$are the opening and closing by reconstruction operators, respectively, with a structuring element i from 1 to n, whose size increases normally in steps of 1 or 2.

When the MP approach is applied to hyperspectral data, the m most significant PCs are used as base images. The result is an EMP:

$$\mathrm{EMP}_m^{(n)}(I) = \{(\mathrm{MP})_1^{(n)}(I), \ldots, (\mathrm{MP})_m^{(n)}(I)\}, \tag{3}$$

that contains $m(2n+1)$ components.

Once the spatial information has been extracted from the image, it must be integrated into the classifier.

2.3. *KELM-based classification*

ELM was originally developed as a training technique for a class of SLFNs with random weights (Huang, Wang, and Lan 2011; Huang, Zhu, and Siew 2006). Huang et al. (2012) proposed the use of a kernel function in the hidden layer instead of the random weights. The objective is to avoid the wide variation in classification accuracy in different trials produced by the random weights. Furthermore, Huang et al. (2012) suggested adding a positive value $\frac{1}{C}$ (where C is defined by the user) to calculate the output weights β such that

$$\boldsymbol{\beta} = \mathbf{H}^{\mathrm{T}}\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^{\mathrm{T}}\right)^{-1}\mathbf{M}, \tag{4}$$

where $\mathbf{H}$ is the hidden-layer output matrix and $\mathbf{M}$ is the target matrix for the training data set. This positive value tends to obtain a stable solution and a better performance. Then, for $\boldsymbol{x} \in \mathbf{R}^d$ $|\boldsymbol{x}$ is a sample vector of size d from the input data set, the output function of ELM is

$$f(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})\boldsymbol{\beta} = \boldsymbol{h}(\boldsymbol{x})\mathbf{H}^{\mathrm{T}}\left(\frac{\mathbf{I}}{\mathbf{C}} + \mathbf{H}\mathbf{H}^{\mathrm{T}}\right)^{-\mathbf{1}}\mathbf{M}, \tag{5}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is a feature mapping. A kernel matrix $\boldsymbol{\Omega}$ for ELM can be represented as

$$\boldsymbol{\Omega}_{\mathrm{ELM}} = \mathbf{H}\mathbf{H}^{\mathrm{T}} : \boldsymbol{\Omega}_{\mathrm{ELM}\,i,j} = [\boldsymbol{h}(\boldsymbol{x}_i) \times \boldsymbol{h}(\boldsymbol{x}_j)] = [\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)], \tag{6}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are training samples, $i = 1, \ldots, N$, $j = 1, \ldots, N$, and $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a kernel function. Finally, the output function can be written as

$$f(\boldsymbol{x}) = \underbrace{\begin{bmatrix} K(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ K(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{N}}) \end{bmatrix}^{\mathrm{T}}}_{\text{Kernel\_1}} \left(\frac{\mathbf{I}}{C} + \underbrace{\boldsymbol{\Omega}_{\mathrm{ELM}}}_{\text{Kernel\_2}}\right)^{-1}\mathbf{M}, \tag{7}$$

where the functions Kernel_1 and Kernel_2 will be used in the Compute Unified Device Architecture (CUDA) algorithm. Thus, ELM with kernels can be summarized as follows.

Algorithm ELM with kernels. Given a training set $\mathbf{J} = \{\boldsymbol{x}_i | \boldsymbol{x}_i \in R^d, i = 1, \ldots, N\}$, target training matrix $\mathbf{M}$, kernel function $K(\boldsymbol{u}, \boldsymbol{v}), \boldsymbol{u}, \boldsymbol{v} \in R^d$ (e.g. $K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\lambda ||\boldsymbol{u} - \boldsymbol{v}||^2)$) and user-defined parameters C and λ,

(1) Calculate $\left(\frac{\mathbf{I}}{C} + \mathbf{\Omega}_{\mathrm{ELM}}\right)^{-1}\mathbf{M}$, with $\mathbf{\Omega}_{\mathrm{ELM}} = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbf{J}$ and K***(u,v)*** $= \exp(-\lambda||\boldsymbol{u} - \boldsymbol{v}||^2)$.

(2) Compute $\begin{bmatrix} K(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ K(\boldsymbol{x}, \boldsymbol{x}_N) \end{bmatrix}^T$, where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbf{J}$ and $\boldsymbol{x}$ refers to all the points of test data.

(3) Calculate the output function of ELM as Equation (7).

In our hyperspectral image case, each training sample represents a random selected pixel on the image and each component of the sample represents a weighted spatial and spectral component of the pixel. The spatial components provided by the EMP are multiplied by a factor k_s whereas the spectral components of the original image are multiplied by a constant k_w. The output obtained after the classification phase is the predicted class for each pixel of the image.

3. Spectral–spatial hyperspectral image classification in GPU

In this section, we introduce some CUDA programming fundamentals as well as the implementation in GPU of the algorithms proposed in Sect. II. The kernels executed in GPU are placed between <> symbols. The pseudocodes also include the GM and SM acronyms to indicate kernels executed in global memory and in shared memory, respectively.

3.1. *CUDA GPU programming fundamentals*

CUDA is a hardware/software platform combination that enables NVIDIA GPUs to execute programs invoking parallel functions called kernels (Nvidia 2012). The kernels are executed by threads that are organized into blocks. The blocks are arranged in a grid that is mapped to a hierarchy of CUDA cores in the GPU. Threads can access data from multiple memory spaces. First, each thread has a private local memory and registers. Each block of thread has a shared memory that is visible exclusively to the threads within this block and whose lifetime is equal to the block lifetime. Finally, all threads access the same global memory space.

Shared memory lifetime makes it difficult to share data among thread blocks, and thus it implies the use of global memory whose access is slower than shared memory access. The Kepler architecture (Nvidia 2012) includes a two-level cache hierarchy. Different performance optimization strategies have been applied in this work.

(1) **Maximize parallel execution**. To organize the algorithm in computational blocks that can be executed independently.
(2) **Improve the efficiency in the use of the memory hierarchy**. To perform the maximum number of computations on the data already stored in shared memory.
(3) **Reduce the number of global synchronizations by computing asynchronous blocks.** In the EMP computation each block is updated a number of times through local synchronization before a global synchronization takes place.

(4) **Add a border to the data regions**. Since in the EMP stage processing each pixel requires data from its neighbours, each data region is extended with a border in order to minimize dependencies among blocks.
(5) **Overlap CPU and GPU operations**. Sometimes it is possible to execute several independent operations in CPU and GPU at the same time, reducing the execution time. In particular, in ELM computation, the generation of random weights for the input data and the hidden neurons is overlapped with the generation of the matrix $\mathbf{X}_{test}$.
(6) **Exploit the available optimized libraries**. Owing to the fact that most of the operations of the algorithms used in this work are matrix operations, different CUDA libraries for linear algebra and image processing have been used. In particular, Matrix Algebra on GPU and Multicore Architectures (MAGMA) (MAGMA 2015), which is a dense linear algebra library for heterogeneous/hybrid architectures used to obtain the matrix $\mathbf{X}_{\mathrm{test}}$ and the pseudo-inverse of matrix $\mathbf{H}$, and CUDA Linear Algebra (CULA) tools (Nvidia 2015b), a set of GPU-accelerated linear algebra libraries, have been used to calculate the EVD in the PCA algorithm. Finally, CUDA Basic Linear Algebra Subroutine (CUBLAS) library (Nvidia 2015a), a GPU version of the standard Basic Linear Algebra Subprograms (BLAS) library, was used to obtain the matrix correlation in the PCA calculation and to obtain the weight matrix in the training step of the ELM and KELM algorithms.

3.2. *PCA on GPU*

The PCA algorithm using EVD (EVD-PCA) explained in Section 2.1 is applied to reduce the dimensionality of the data set.

Algorithm 1 includes the pseudocode of the EVD-PCA algorithm in GPU. Suppose that matrix $\mathbf{X}$ contains the data set and it is stored in the global memory on the GPU. It is necessary to preprocess the data set before the analysis. The data set must be centred by subtracting from each pixel the mean of the coefficients of that band.

The centred stage starts by creating a vector $\boldsymbol{y}$ of ones of a size equal to the number of pixels (line 1 in Algorithm 1), which is passed together with the matrix $\mathbf{X}$ to the *cublasSgemv* CUBLAS function to perform the matrix-vector product $z = \mathbf{X}^{\mathrm{T}} \times \mathbf{y}$ (line 2). Each value of the vector contains the sum of all pixels of one band of the data set $\mathbf{X}$. The centring stage finishes the centring matrix $\mathbf{X}$ using the values of vector $\boldsymbol{z}$ (line 3). This step is performed using a CUDA kernel, where each thread operates over all bands of each pixel of $\mathbf{X}$. To improve the efficiency, it is necessary to store $\boldsymbol{z}$ in the shared memory.

Once the matrix $\mathbf{X}$ contains the centred data, the PCA stage commences. First, the correlation matrix of matrix $\mathbf{X}$ is calculated using the *cublasSsyrk* CUBLAS function (line 4). This function performs the $\mathbf{XX}^{\mathrm{T}}$operation, but owing to the fact that the result is symmetric, only the upper triangular matrix is computed. The next step is to complete the correlation matrix $\mathbf{XX}^{\mathrm{T}}$ (line 5).

The PCA stage continues with the computation of EVD using the *culaDeviceSgesvd* CULA function. It performs the operation of Equation (1) (line 6). The last step is to compute the PC using the *cublasSgemm* CUBLAS function, which calculates the product between $\mathbf{X}$ and $\mathbf{V}$.

The second algorithm to reduce the dimensionality used in this article is an iterative PCA algorithm (GS-PCA) (Andrecut 2009) based on the GS re-orthogonalization process (see Section 2.1). The GPU implementation is illustrated in Algorithm 2. This algorithm iteratively extracts one PC at a time. Similar to the EVD-PCA algorithm, it is necessary to centre the data set before the analysis (lines 1–3 in the pseudocode). Then, an iterative process starts. It is repeated as many times as the number of PCs required (line 4). Inside it, another iterative process performs the operations to obtain both eigenvectors and eigenvalues, which will be used to calculate the *k*-th PC (lines 6–9). This second iterative process ends when either the limit of iterations is reached or the residual error is achieved. All the operations of the GS-PCA algorithm have been performed using the CUBLAS library.

Algorithm 1 GPU EVD-PCA algorithm

```
Require: The dataset $\mathbf{X}$ is initially stored in the Global Memory.
                                                              ▷ Dataset centering phase
 1: <Create a vector of 1's>                                                      ▷ GM
 2: Obtain the sum of all pixels in each band of dataset                     ▷ SM + GM
 3: <Dataset centering>                                                           ▷ SM
                                                                           ▷ PCA phase
 4. Calculate the covariance matrix $\mathbf{XX}^T$ of centered dataset          ▷ SM + GM
 5: <Complete the lower triangular values of $\mathbf{XX}^T$>                        ▷ GM
 6: Calculate the matrix of eigenvectors $\mathbf{V}$                              ▷ SM + GM
 7: Obtain the PCs by $\mathbf{XV}$                                                ▷ SM + GM
```

Algorithm 2 GPU GS-PCA algorithm

```
Require: The dataset $\mathbf{X}$ is initially stored in the Global Memory.
                                                              ▷ Dataset centering phase
 1: <Create a vector of 1's>                                                      ▷ GM
 2: <Obtain the sum of all pixels in each band of dataset>                        ▷ GM
 3: <Dataset centering>                                                           ▷ SM
                                                                           ▷ PCA phase
 4: for each PC do
 5:     Copy the data from the dataset                                       ▷ SM + GM
 6:     repeat
 7:         Calculate the eigenvalues and the eigenvectors                   ▷ SM + GM
 8:         Calculate the PC                                                 ▷ SM + GM
 9:     Until either the limit of iterations is reached or the residual error is achieved
10. end for
```

Algorithm 3 GPU KELM algorithm

```
Require: Hyperspectral dataset $\mathbf{X}$, target set $\mathbf{M}$ and user-defined parameters $C$ and $\lambda$
                                                                 ▷ Preprocesing phase
 1: <Normalize hyperspectral dataset>                                        ▷ SM + GM
 2: Randomly choose the training points ($\mathbf{X}_{train}$)
 3: <Process training target matrix $\mathbf{M}$>                                   ▷ GM
                                                         ▷ Kernel function computation
 4: <Calculate Kernel_2 (Equation 7)>                                             ▷ GM
 5: <Calculate $\alpha = (\mathbf{I}/C + \text{Kernel\_2})^{-1}\mathbf{M}$>                        ▷ GM
 6: <Take test points> ($\mathbf{X}_{test}$)                                        ▷ GM
 7: <Calculate Kernel_1 (Equation 7)>                                             ▷ GM
                                                                      ▷ Classification
 8: <Calculate $\alpha$Kernel_1>                                                  ▷ GM
```

3.3. *EMP on GPU*

The EMP is the set of MPs created through opening and closing by reconstruction for each coefficient-band resulting from the PCA step.

The implementation selected is based on a block-asynchronous propagation (Quesada-Barriuso, Heras, and Argüello 2013) where multiple scans are performed in both directions at the same time. The main advantage of this GPU asynchronous implementation (Quesada-Barriuso et al. 2015) for the morphological reconstruction (consisting of intra- and inter-block updates) is that as many updates as possible are performed in shared memory with the available data before performing a global synchronization among the thread blocks. Data updated within a block in shared memory can be reused, which is much faster than the global memory updates (Kirk and Wen-Mei 2010). In this asynchronous proposal, the number of global synchronization is reduced compared with the synchronous version of the algorithm (Quesada-Barriuso, Heras, and Argüello 2013).

3.4. *KELM classification on GPU*

The KELM algorithm has three main phases: preprocesing, kernel function computation, and classification. The pseudocode in Algorithm 3 shows the implementation on GPU as explained in Section 2.3.

In our case the KELM is part of the KELM-EMP algorithm, so its input is the result of joining and normalizing the hyperspectral data set and the EMP. In the joining phase, the profile data obtained from the EMP algorithm are multiplied by the spatial weight. Then, in the normalization step (line 1 in the pseudocode), the minimum value of the hyperspectral data set is subtracted from all its pixels. For the EMP data, the normalization consists of subtracting the minimum value of each band from all pixels of the band. The kernel used to calculate the minimum value takes advantage of the shared memory and avoids bank conflicts that reduce the execution time.

Given that ELM and KELM are supervised learning algorithms, the pixels for the training data set are randomly selected from the reference data, scaled in the range [0:1] and stored in the matrix $\mathbf{X}_{\text{train}}$ (line 2 in the pseudocode). Finally, the training target matrix is processed so that each row represents a sample and each column represents a class, where a value of 1 indicates membership to a class (line 3 in the pseudocode).

The second phase (computation of kernel functions) starts computing the second kernel function, Kernel_2, where $K(\boldsymbol{u},\boldsymbol{v}) = \exp(-\lambda||\boldsymbol{u}-\boldsymbol{v}||^{\mathbf{2}})$ and ***u,v*** belong to the training data set. First, we calculate the matrix $\mathbf{X}_{\text{train}}(\mathbf{X}_{\text{train}})^{\text{T}}$ containing all possible products between two elements of the training data set. Then we use this matrix, its diagonal, and the user-defined parameter λ to compute Kernel_2 matrix. The next step is generating a modified identity matrix dividing all the elements of its diagonal by the user-defined parameter C. Then, we add the modified identity matrix to the previously computed Kernel_2 matrix. Finally, we use a *magma_dgesv_gpu* MAGMA function to solve a system of linear equations and obtain $\boldsymbol{\alpha}$, where

$$\boldsymbol{\alpha} = (\mathbf{I}/C + \text{Kernel\_2})^{-1}\mathbf{M}.$$

This second phase continues computing the first kernel function, Kernel_1, as the previous one, but this time $\boldsymbol{u}$ belongs to the test data set and $\boldsymbol{v}$ belongs to the training data set. First, the test data set is scaled in the range [0:1] and stored in the matrix $\mathbf{X}_{test}$. Then, to compute the kernel function, we create one matrix to store $\mathbf{X}_{test}\mathbf{X}_{train}$, one vector to store the diagonal of, $\mathbf{X}_{test}\mathbf{X}_{test}$ and another one for the diagonal of $\mathbf{X}_{train}\mathbf{X}_{train}$. Finally, the kernel function for all the elements of $\mathbf{X}_{test}$ and all the elements of $\mathbf{X}_{train}$ is computed.

The last phase of the KELM algorithm only needs to compute the product of Kernel_1 by $\boldsymbol{\alpha}$ to obtain the final classification.

3.5. *KELM-EMP classification on GPU*

KELM-EMP is an alternative algorithm to the ELM-EMP one explained in Sect. 2.3 but replacing ELM by KELM. The GPU implementation of the different stages of the algorithm has been explained in the previous sections.

Variants of the ELM-EMP (ELM-EMP-S) and KELM-EMP (KELM-EMP-S) schemes have been developed, including a post-processing step after the ELM classification consisting of a spatial regularization (Heras, Argüello, and Quesada-Barriuso 2014). This regularization is an iterative process performed over the classification map obtained by the ELM or KELM. Each pixel checks the class label of its neighbours and all the pixels do it simultaneously. If more than half of the neighbours share the same label, and this label is different from that of the pixel, the pixel updates its own class label. This is computed by a single kernel that is iteratively executed as many times as it takes to reach stability (no changes between two consecutive iterations in any of the pixels).

4. Results

This section shows the experimental results obtained by the classifiers. The proposed algorithms have been evaluated on a PC with a quad-core Intel Xeon E5-2609v2 at 2.5 GHz and 15 GB of RAM. The code has been compiled using the gcc 4.8.4 version with OpenMP (OMP) 3.0 support under Linux using four threads. The OPENBLAS library (OpenBLAS 2015) has been used to accelerate the algebra operations included in the algorithms. Regarding the GPU implementation, CUDA codes run on an NVIDIA GeForce GTX Titan with 14 Streaming Multiprocesors (SMXs) and 192 CUDA cores each. The CUDA 7.5 of the toolkit under Linux has been used.

The accuracy results are expressed in percentage in terms of overall accuracy (OA), which is the percentage of correctly classified pixels; average accuracy (AA), which is computed as the mean of the class accuracies; and kappa coefficient (Richards 1999), which is the percentage of agreement corrected by the amount of agreement that could be expected due to chance alone.

The performance results are expressed in terms of run times (in seconds) and speedups. The results are the average of 100 runs. The run times for the GPU codes do not include CPU–GPU data transfers, only computation times.

The algorithms have been used over three remote-sensing images: a 103-band ROSIS image of the University of Pavia (Pavia Univ.) with a spatial dimension of 610 $\times$ 340 pixels, a 220-band AVIRIS image of 145 $\times$ 145 pixels taken over Northwest Indiana

(Indian Pines), and a 204-band AVIRIS image 512 × 217 pixels from Salinas Valley, California (Salinas).

The number of training samples is the same as in the reference works in the literature (Argüello and Heras 2015) in order to perform a reliable comparison. Table 1 presents some information on the remote-sensing images including the dimensions of the images and the number and percentage of training samples. The training set is randomly chosen and these samples are excluded when the accuracy evaluation is performed. The number of hidden-layer neurons employed for the ELM are 1000 for Pavia Univ., 300 for Indian Pines, and 350 for Salinas in all the cases (Argüello and Heras 2015).

In the case of spectral–spatial classifiers, the EMP was computed as a preprocessing step. The best results were obtained for an EMP with seven PCs and seven openings and closings by reconstruction, using disks of increasing radius (the structuring element sizes are 3, 5, 9, 13, 17, 21, and 25, giving a total of 105 components), and a weighted concatenation-type composite feature mapping. The weight of the spectral feature, k_w, was set to 1, whereas the weight of the spatial feature, k_s, was adjusted for each case. The best results obtained were for k_s values of 1, 5, and 3 for the Pavia University, Indian Pines, and Salinas data sets, respectively. As mentioned in Section 2.3, in order to maximize the discriminative capacity of the classifier, each data set χ was shifted to the $[0, \max(\chi) - \min(\chi)]$ range. This process was performed independently for the hyperspectral data and for each one of the individual components of the EMP. Finally, after concatenating both features, the entire data set was scaled in the range of [0,1].

The GPU implementations of the two versions of PCA explained in Section 2.1, Gram–Schmidt (GS-PCA) and Eigenvalue Decomposition (EVD-PCA), have been computed over the three hyperspectral images. The GS-PCA algorithm is up to 19 times slower than the EVD-PCA one (See Table 2). Therefore, the EVD-PCA algorithm will be used in the final versions of the classifiers.

Regarding the comparison between the original scheme based on ELM (ELM-EMP) and the version including a spatial post-regularization (ELM-EMP-S), eight neighbours are considered for each pixel. Table 3 shows that the most time-consuming stages of the ELM-EMP algorithm in CPU are the profile (EMP) and ELM calculations. Comparing these results for Pavia Univ. to those presented in Table 4 for the same image, we see that the total time in GPU for ELM-EMP is 2.29 s with an OA value of 99.64%, whereas for the ELM-EMP-S the time is 2.30 s and the OA value is improved to 99.82%. Therefore, ELM-EMP-S

Table 1. Information for the test remote-sensing images.

Data set	Sensor	No. of classes	Dimensions	No. of samples	No. of training samples
Pavia Univ.	ROSIS	9	610 × 340 × 103	42776	3921 (9.17%)
Indian Pines	AVIRIS	16	145 × 145 × 220	10249	625 (6.10%)
Salinas	AVIRIS	16	512 × 217 × 204	54129	1076 (1.99%)

Table 2. Execution time results for the PCA algorithms in GPU.

	Execution time (s)		
PCA Algorithm	Pavia Univ.	Salinas	Indian Pines
GS-PCA	0.96	0.39	0.12
EVD-PCA	0.05	0.07	0.06

Table 3. Execution times and classification accuracies of the ELM-EMP scheme. Results for the Pavia Univ.

	Execution time (s)			
Platform	PCA	EMP	ELM	Total
CPU	0.43	14.93	14.54	29.90
OMP	0.28	4.35	6.41	11.04
GPU	0.05	1.08	1.16	2.29
OA: 99.64		AA: 99.60		Kappa: 99.51

Table 4. Execution times and classification accuracies of the ELM-EMP-S scheme.

		Execution time (s)			
Data set	Platform	PCA	EMP	ELM	Total
Pavia Univ.	CPU	0.43	14.93	14.97	30.33
	OMP	0.28	4.35	6.70	11.33
	GPU	0.05	1.08	1.17	2.30
	OA: 99.82	AA: 99.76		Kappa: 99.75	
Indian Pines	CPU	0.13	1.52	0.82	2.47
	OMP	0.07	0.44	0.4	0.91
	GPU	0.06	0.58	0.07	0.71
	OA: 95.05	AA: 96.44		Kappa: 94.32	
Salinas	CPU	0.60	8.00	3.96	12.56
	OMP	0.31	2.30	2.06	4.67
	GPU	0.07	0.70	0.22	0.99
	OA: 99.21	AA: 99.09		Kappa: 99.12	

will be used in the final versions of the classifier. Table 4 shows also the time and accuracy results for the other two hyperspectral images. The accuracy results are always equal to or better than those previously published for the ELM-EMP method (Argüello and Heras 2015). The speedups achieved for the ELM-EMP-S scheme are shown in Table 5. If the GPU versus OMP (with four threads) speedups are observed, values of 4.92× are achieved for the Pavia Univ. data set. The lowest value is obtained for the Indian Pines data set (1.28×). The reason is that this image is 4.5 times smaller than the other two used in this article. Therefore, the number of threads required is smaller and the cost of memory transfers is not fully hidden.

In Section 3.5 we also presented a version of the classification scheme based on KELM. In Table 6 results of the K-ELM-EMP-S scheme (including spatial post-regularization) are shown. The values of the user-defined parameters C and λ (see Section 3.4) for the three hyperspectral images are: $C = 10^8$ and $\lambda = 10$ for Pavia Univ.; $C = 10^6$ and $\lambda = 10$ for Indian Pines; and $C = 10^8$ and $\lambda = 12$ for Salinas. The accuracies achieved for the first two images (Pavia Univ. and Indian Pines) with this scheme (Table 6) are better than those obtained for the ELM-EMP-S scheme (Table 4). However, the K-ELM-EMP-S scheme requires more time than the ELM-EMP-S to achieve the final result. Table 7 shows the speedups achieved for the K-ELM-EMP-S scheme. If the GPU versus OMP speedups are

Table 5. Speed increase (multiple of original speed) for the ELM-EMP-S scheme.

Data set	OMP vs. CPU	GPU vs. OMP	GPU vs. CPU
Pavia Univ.	2.68	4.92	13.19
Indian Pines	2.71	1.28	3.48
Salinas	2.69	4.71	12.68

Table 6. Execution times and classification accuracies of the KELM-EMP-S scheme.

		Execution time (s)			
Data set	Platform	PCA	EMP	KELM	Total
Pavia Univ.	CPU	0.43	14.93	51.61	66.97
	OMP	0.28	4.35	14.92	19.55
	GPU	0.05	1.08	2.80	3.93
	OA: 99.83		AA: 99.79	Kappa: 99.77	
Indian Pines	CPU	0.13	1.52	1.22	2.87
	OMP	0.07	0.44	0.56	1.07
	GPU	0.06	0.58	0.15	0.79
	OA: 95.39		AA: 96.82	Kappa: 94.72	
Salinas	CPU	0.60	8.00	8.92	17.52
	OMP	0.31	2.30	2.81	5.42
	GPU	0.07	0.70	0.62	1.39
	OA: 99.16		AA: 99.05	Kappa: 99.06	

Table 7. Speed increase (multiple of original speed) for the KELM-EMP-S scheme.

Data set	OMP vs. CPU	GPU vs. OMP	GPU vs. CPU
Pavia Univ.	3.42	4.97	17.04
Indian Pines	2.68	1.35	3.63
Salinas	3.23	3.90	12.60

observed, values of 4.97× are achieved for the Pavia Univ. data set. Once again, the lowest value is obtained for the Indian Pines data set due to its reduced size.

In Table 8 the ELM-EMP-S and K-ELM-EMP-S proposed schemes are compared to other schemes projected to GPU and available in the literature in terms of classification accuracy and execution times in GPU. These schemes use ELM as a classifier such as ELM+wat (López-Fandiño et al. 2015), and SVM as a classifier such as SVM+wat (Tarabalka, Chanussot, and Benediktsson 2010). – Both schemes use watershed (wat) to extract spatial information. Another example that uses SVM is WT-EMP (Quesada-Barriuso, Argüello, and Heras 2014) that utilizes Wavelet Transforms (WT) and EMP to extract spatial information. All the experiments in the table were performed in the same experimental conditions as described at the beginning of this section. In the table, the best results are in bold. The ELM implemented in the López-Fandiño et al. (2015) scheme performs a regularization after the classification as the proposed ELM-EMP-S and K-ELM-EMP-S schemes. The ELM +wat (López-Fandiño et al. 2015) scheme combines the information provided by the ELM scheme with a spatial information obtained through a watershed algorithm. The SVM (Tarabalka, Chanussot, and Benediktsson 2010) and the SVM+wat (Tarabalka, Chanussot, and Benediktsson 2010) schemes use SVM, but the second one adds spatial information using a watershed algorithm. Finally, in the WT-EMP (Quesada-Barriuso, Argüello, and Heras 2014) scheme, an EMP is created from the features extracted by wavelets and combined with the denoised image in a stacked vector used for the SVM classification.

In Table 8 we can see that the proposed schemes (ELM-EMP-S and K-ELM-EMP-S) obtain very close results in terms of accuracies. If the execution time is a priority, then ELM-EMP-S must be chosen (see Tables 4 and 6).

Based on the detailed classification results obtained by each classification scheme for each pixel, we also perform the standard McNemar test (Foody 2004; Chen et al. 2014). It is employed to verify the statistical significance of accuracy differences among pairs of schemes. Table 9 presents the results of the test where each *Z* value compares the

Table 8. Accuracies and execution times in GPU for the different images.

Scheme	Pavia Univ.				Indian Pines				Salinas			
	OA (%)	AA (%)	Kappa (%)	t (s)	OA (%)	AA (%)	Kappa (%)	t (s)	OA (%)	AA (%)	Kappa (%)	t (s)
KELM-EMP-S	**99.83**	**99.79**	**99.77**	3.93	**95.39**	**96.82**	**94.72**	0.79	99.16	99.05	99.06	1.39
	± **0.07**	± **0.04**	± **0.09**	± 0.13	± **0.80**	± **0.63**	± **0.92**	± 0.15	± 0.18	± 0.20	± 0.20	± 0.16
ELM-EMP-S	99.82	99.76	99.75	2.30	95.05	96.44	94.32	0.71	**99.21**	**99.09**	**99.12**	0.99
	± 0.09	± 0.05	± 0.13	± 0.11	± 0.74	± 0.56	± 0.85	± 0.13	± **0.25**	± **0.18**	± **0.28**	± 0.14
ELM	96.94	96.13	95.83	0.99	82.25	89.85	79.93	0.06	93.63	96.38	92.89	0.19
	± 0.31	± 0.34	± 0.43	± 0.01	± 1.57	± 0.98	± 1.73	± 0.01	± 0.29	± 0.25	± 0.32	± 0.01
ELM+wat	97.20	96.42	96.30	1.16	80.64	79.90	78.40	0.10	93.60	96.41	92.86	0.39
	± 0.33	± 0.43	± 0.44	± 0.01	± 2.46	± 3.04	± 2.73	± 0.01	± 0.32	± 0.27	± 0.36	± 0.01
SVM	79.43	86.62	76.70	3.74	79.43	86.62	76.70	1.50	88.45	92.37	87.08	2.49
	± 0.96	± 0.86	± 1.05	± 0.05	± 0.96	± 0.86	± 1.05	± 0.03	± 0.47	± 0.50	± 0.54	± 0.07
SVM+wat	96.18	96.58	94.81	3.91	87.65	91.41	85.96	1.54	–	–	–	-
	± 0.52	± 0.25	± 0.70	± 0.05	± 1.26	± 1.78	± 1.42	± 0.03	–	–	–	-
WT-EMP	98.70	98.72	98.22	3.16	89.73	94.12	88.28	1.66	–	–	–	-
	± 0.18	± 0.13	± 0.24	± 0.05	± 1.14	± 0.67	± 1.28	± 0.07	–	–	–	-

The results shown are averages of 100 runs. After ± symbol standard deviations are shown. The best accuracies are indicated in bold.

Table 9. Statistical significance of differences in classification accuracies expressed as Z values among KELM-EMP-S and the schemes proposed in the table.

Data set	ELM-EMP-S	ELM	ELM+wat	SVM	SVM+wat	WT-EMP
Pavia Univ.	0.62	33.94	32.31	64.30	38.25	19.72
Indian Pines	1.55	31.27	26.86	89.76	89.52	88.76
Salinas	−1.48	50.99	74.32	51.09	-	-

KELM-EMP-S classifier to each one of the methods shown in the table. The difference in accuracy between each pair of classifiers is considered to be significantly different at 95% confidence level if $|Z| > 1.96$, and at 99% confidence level if $|Z| > 2.58$. A positive sign of Z indicates that the first classifier outperforms the second one ($Z > 0$). We can observe that only in one case, the comparison between KELM-EMP-S and ELM-EMP-S, the small differences in accuracy are not statistically relevant, indicating that both classification schemes are similar in accuracy terms.

5. Conclusion

In this article, a GPU implementation of a spectral–spatial ELM-based classification scheme for hyperspectral images (ELM-EMP) that integrates spatial information by a composite feature mapping method is presented. Several optimizations are introduced to improve the accuracy of this scheme. In particular, an ELM-EMP algorithm based on kernels (KELM-EMP) has been proposed. Moreover, a spatial regularization process is applied to the previous schemes (ELM-EMP-S and KELM-EMP-S) outperforming the first ones. These schemes require an initial reduction of dimensionality of the image by PCA, followed by the computation of an EMP. A later weighted concatenation combines the profile with the initial hyperspectral image. In order to improve the execution time, two different PCA algorithms have been compared, an EVD method and an iterative GS algorithm. The first one was selected because of the lower execution times.

Both schemes (ELM-EMP-S and K-ELM-EMP-S) obtain high accuracies, achieving 99.83% accuracy for the Pavia Univ. image. The K-ELM-EMP-S algorithm in GPU achieves a speedup over the sequential implementation of 17.04×.

Acknowledgements

We thank our colleague Chen Chen at the University of Texas for his collaboration sharing his MATLAB codes with us.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the Ministry of Science and Innovation, Government of Spain, cofounded by the FEDER funds of European Union [2013-41129-P], and by Xunta de Galicia, Program for Consolidation of Competitive Research Groups ref. 2014/008.

ORCID

Alberto S. Garea http://orcid.org/0000-0001-9394-0330
Dora B. Heras http://orcid.org/0000-0002-5304-1426
Francisco Argüello http://orcid.org/0000-0001-9279-5426

References

Abdi, H., and L. J. Williams. 2010. "Principal Component Analysis." *Wiley Interdisciplinary Reviews: Computational Statistics* 2 (4): 433–459. doi:10.1002/wics.101.

Andrecut, M. 2009. "Parallel GPU Implementation of Iterative PCA Algorithms." *Journal of Computational Biology* 16 (11): 1593–1599. doi:10.1089/cmb.2008.0221.

Argüello, F., and D. B. Heras. 2015. "ELM-Based Spectral-Spatial Classification of Hyperspectral Images Using Extended Morphological Profiles and Composite Feature Mappings." *International Journal of Remote Sensing* 36 (2): 645–664. doi:10.1080/01431161.2014.999882.

Benediktsson, J. A., J. A. Palmason, and J. R. Sveinsson. 2005. "Classification of Hyperspectral Data from Urban Areas Based on Extended Morphological Profiles." *IEEE Transactions on Geoscience and Remote Sensing* 43 (3): 480–491. doi:10.1109/TGRS.2004.842478.

Benediktsson, J. A., M. Pesaresi, and K. Amason. 2003. "Classification and Feature Extraction for Remote Sensing Images from Urban Areas Based on Morphological Transformations." *IEEE Transactions on Geoscience and Remote Sensing* 41 (9): 1940–1949. doi:10.1109/TGRS.2003.814625.

Chen, C., W. Li, H. Su, and K. Liu. 2014. "Spectral-Spatial Classification of Hyperspectral Image Based on Kernel Extreme Learning Machine." *Remote Sensing* 6 (6): 5795–5814. doi:10.3390/rs6065795.

Dalla Mura, M., J. A. Benediktsson, B. Waske, and L. Bruzzone. 2010. "Morphological Attribute Profiles for the Analysis of Very High Resolution Images." *IEEE Transactions on Geoscience and Remote Sensing* 48 (10): 3747–3762. doi:10.1109/TGRS.2010.2048116.

Dalla Mura, M., A. Villa, J. A. Benediktsson, J. Chanussot, and L. Bruzzone. 2011. "Classification of Hyperspectral Images by Using Extended Morphological Attribute Profiles and Independent Component Analysis." *IEEE Geoscience and Remote Sensing Letters* 8 (3): 542–546. doi:10.1109/LGRS.2010.2091253.

Fauvel, M., Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton. 2013. "Advances in Spectral–Spatial Classification of Hyperspectral Images." *Proceedings of the IEEE* 101 (3): 652–675. doi:10.1109/JPROC.2012.2197589.

Foody, G. M. 2004. "Thematic Map Comparison." *Photogrammetric Engineering & Remote Sensing* 70 (5): 627–633. doi:10.14358/PERS.70.5.627.

Golub, G. H., and C. F. Van Loan. 1996. *Matrix Computations. 1996*, 374–426. Baltimore, MD: Johns Hopkins University Press.

Heras, D. B., F. Argüello, and P. Quesada-Barriuso. 2014. "Exploring ELM–Based Spatial–Spectral Classification of Hyperspectral Images." *International Journal of Remote Sensing* 35 (2): 401–423. doi:10.1080/01431161.2013.869633.

Huang, G.-B. 2003. "Learning Capability and Storage Capacity of Two–Hidden–Layer Feedforward Networks." *IEEE Transactions on Neural Networks* 14 (2): 274–281. doi:10.1109/TNN.2003.809401.

Huang, G.-B. 2014. "An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels." *Cognitive Computation* 6 (3): 376–390. doi:10.1007/s12559-014-9255-2.

Huang, G.-B., D. H. Wang, and Y. Lan. 2011. "Extreme Learning Machines: A Survey." *International Journal of Machine Learning and Cybernetics* 2 (2): 107–122. doi:10.1007/s13042-011-0019-y.

Huang, G.-B., H. Zhou, X. Ding, and R. Zhang. 2012. "Extreme Learning Machine for Regression and Multiclass Classification." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42 (2): 513–529. doi:10.1109/TSMCB.2011.2168604.

Huang, G.-B., Q.-Y. Zhu, and C.-K. Siew. 2006. “Extreme Learning Machine: Theory and Applications.” *Neurocomputing* 70 (1–3): 489–501. doi:10.1016/j.neucom.2005.12.126.

Kang, X., S. Li, and J. A. Benediktsson. 2014. “Spectral–Spatial Hyperspectral Image Classification with Edge-Preserving Filtering.” *IEEE Transactions on Geoscience and Remote Sensing* 52 (5): 2666–2677. doi:10.1109/TGRS.2013.2264508.

Kirk, D., and W. H. Wen-Mei. 2010. *Programming Massively Parallel Processors: A Hands–On Approach*. Vol. 1. Elsevier Science, Applications of GPU Computing Series. Waltham, MA: Elsevier.

Landgrebe, D. 2002. “Hyperspectral Image Data Analysis.” *IEEE Signal Processing Magazine* 19 (1): 17–28. doi:10.1109/79.974718.

Licciardi, G., P. R. Marpu, J. A. Benediktsson, and J. Chanussot. 2011. “Extended Morphological Profiles Using Auto–Associative Neural Networks for Hyperspectral Data Classification.” In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2011 3rd Workshop on*, Portugal, June 6–9, edited by A. Plaza and B. Dias, 1–4. IEEE.

Lingen, F. J. 2000. “Efficient Gram-Schmidt Orthonormalisation on Parallel Computers.” *Communications in Numerical Methods in Engineering* 16 (1): 57–66. doi:10.1002/(ISSN)1099-0887.

López-Fandiño, J., P. Quesada-Barriuso, D. B. Heras, and F. Argüello. 2015. “Efficient ELM-Based Techniques for the Classification of Hyperspectral Remote Sensing Images on Commodity GPUs.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8 (6): 2884–2893. doi:10.1109/JSTARS.2014.2384133.

MAGMA. 2015. “Matrix Algebra on GPU and Multicore Architectures.” Accessed 25 February 2015. http://icl.cs.utk.edu/projectsfiles/magma/doxygen/.

Marpu, P. R., M. Pedergnana, M. D. Mura, S. Peeters, J. A. Benediktsson, and L. Bruzzone. 2012. “Classification of Hyperspectral Data Using Extended Attribute Profiles Based on Supervised and Unsupervised Feature Extraction Techniques.” *International Journal of Image and Data Fusion* 3 (3): 269–298. doi:10.1080/19479832.2012.702687.

Nvidia. 2012. “Nvidia Kepler GK110 Architecture.” Accessed 15 February 2015. https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

Nvidia. 2015a. “CUDA Toolkit Documentation: CUBLAS.” Accessed March 5, 2015. http://docs.nvidia.com/cuda/cublas/index.html.

Nvidia. 2015b. “CULA Tools.” Accessed 13 April 2015. http://www.culatools.com/.

OpenBLAS. 2015. “An Optimized BLAS Library.” Accessed 3 April 2015. http://www.openblas.net/.

Palmason, J. A., J. A. Benediktsson, J. R. Sveinsson, and J. Chanussot. 2005. “Classification of Hyperspectral Data from Urban Areas Using Morphological Preprocessing and Independent Component Analysis.” In *International Geoscience and Remote Sensing Symposium* Seoul, July 25–29, edited by P. Moon, K. Lee, and C. Emery, Vols. 1, 176–179.

Pesaresi, M., and J. A. Benediktsson. 2001. “A New Approach for the Morphological Segmentation of High–Resolution Satellite Imagery.” *IEEE Transactions on Geoscience and Remote Sensing* 39 (2): 309–320. doi:10.1109/36.905239.

Pesaresi, M., and I. Kanellopoulos. 1999. “Detection of Urban Features Using Morphological Based Segmentation and Very High Resolution Remotely Sensed Data.” In *Machine Vision and Advanced Image Processing in Remote Sensing*, edited by K. Ioannis, W. G. Graeme, and M. Theo, 271–284. Berlin: Springer.

Plaza, A., J. A. Benediktsson, J. W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, et al. 2009. “Recent Advances in Techniques for Hyperspectral Image Processing.” *Remote Sensing of Environment* 113: S110–S122. doi:10.1016/j.rse.2007.07.028.

Priego, B., F. Bellas, and R. J. Duro. 2015. “ECAS-II: A Hybrid Algorithm for the Construction of Multidimensional Image Segmenters.” In *Neural Networks (IJCNN), 2015 International Joint Conference on*, Killarney, July 12–17, edited by D.-S. Huang and Y. Choe,, 1–8. IEEE.

Quesada-Barriuso, P., F. Argüello, and D. B. Heras. 2014. “Spectral–Spatial Classification of Hyperspectral Images Using Wavelets and Extended Morphological Profiles.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7 (4): 1177–1185. doi:10.1109/JSTARS.2014.2308425.

Quesada-Barriuso, P., F. Argüello, D. B. Heras, and J. A. Benediktsson. 2015. “Wavelet-Based Classification of Hyperspectral Images Using Extended Morphological Profiles on Graphics Processing Units.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8 (6): 2962–2970. doi:10.1109/JSTARS.2015.2394778.

Quesada-Barriuso, P., D. B. Heras, and F. Argüello. 2013. “Efficient 2D and 3D Watershed on Graphics Processing Unit: Block-Asynchronous Approaches Based on Cellular Automata.” *Computers & Electrical Engineering* 39 (8): 2638–2655. doi:10.1016/j.compeleceng.2013.04.020.

Richards, J. A. 1999. *Remote Sensing Digital Image Analysis*. Vol. 3. Berlin: Springer.

Soille, P., and M. Pesaresi. 2002. “Advances in Mathematical Morphology Applied to Geoscience and Remote Sensing.” *IEEE Transactions on Geoscience and Remote Sensing* 40 (9): 2042–2055. doi:10.1109/TGRS.2002.804618.

Tamura, S., and M. Tateishi. 1997. “Capabilities of a Four–Layered Feedforward Neural Network: Four Layers versus Three.” *IEEE Transactions on Neural Networks* 8 (2): 251–255. doi:10.1109/72.557662.

Tarabalka, Y., J. A. Benediktsson, and J. Chanussot. 2009. “Spectral–Spatial Classification of Hyperspectral Imagery Based on Partitional Clustering Techniques.” *IEEE Transactions on Geoscience and Remote Sensing* 47 (8): 2973–2987. doi:10.1109/TGRS.2009.2016214.

Tarabalka, Y., J. Chanussot, and J. A. Benediktsson. 2010. “Segmentation and Classification of Hyperspectral Images Using Watershed Transformation.” *Pattern Recognition* 43 (7): 2367–2379. doi:10.1016/j.patcog.2010.01.016.

Van Heeswijk, M., Y. Miche, E. Oja, and A. Lendasse. 2011. “GPU-Accelerated and Parallelized ELM Ensembles for Large-Scale Regression.” *Neurocomputing* 74 (16): 2430–2437. doi:10.1016/j.neucom.2010.11.034.

Zhang, Y., X. Feng, and X. Le. 2008. “Segmentation on Multispectral Remote Sensing Image Using Watershed Transformation.” In *Image and Signal Processing, 2008. CISP’08. Congress on*, Sanya, May 27–30, edited by D. Li and G. Deng, 4773–4777. IEEE.