# Cloud Deep Networks for Hyperspectral Image Analysis

Juan Mario Haut, *Student Member, IEEE*, Jose Antonio Gallardo, Mercedes E. Paoletti, *Student Member, IEEE*, Gabriele Cavallaro, *Member, IEEE*, Javier Plaza, *Senior Member, IEEE*, Antonio Plaza, *Fellow, IEEE*, and Morris Riedel, *Member, IEEE*

***Abstract*— Advances in remote sensing hardware have led to a significantly increased capability for high-quality data acquisition, which allows the collection of remotely sensed images with very high spatial, spectral, and radiometric resolution. This trend calls for the development of new techniques to enhance the way that such unprecedented volumes of data are stored, processed, and analyzed. An important approach to deal with massive volumes of information is data compression, related to how data are compressed before their storage or transmission. For instance, hyperspectral images (HSIs) are characterized by hundreds of spectral bands. In this sense, high-performance computing (HPC) and high-throughput computing (HTC) offer interesting alternatives. Particularly, distributed solutions based on cloud computing can manage and store huge amounts of data in fault-tolerant environments, by interconnecting distributed computing nodes so that no specialized hardware is needed. This strategy greatly reduces the processing costs, making the processing of high volumes of remotely sensed data a natural and even cheap solution. In this paper, we present a new cloud-based technique for spectral analysis and compression of HSIs. Specifically, we develop a cloud implementation of a popular deep neural network for non-linear data compression, known as autoencoder (AE). Apache Spark serves as the backbone of our cloud computing environment by connecting the available processing nodes using a master–slave architecture. Our newly developed approach has been tested using two widely available HSI data sets. Experimental results indicate that cloud computing architectures offer an adequate solution for managing big remotely sensed data sets.**

***Index Terms*— Autoencoder (AE), cloud computing, dimensionality reduction (DR), high-performance computing (HPC), high-throughput computing (HTC), hyperspectral images (HSIs), speedup.**

## I. Introduction

EARTH observation (EO) has evolved dramatically in the last decades due to the technological advances incorporated into remote sensing instruments in the optical and microwave domains [1]. With their hundreds of contiguous and narrow channels within the visible, near-infrared, and shortwave infrared spectral ranges, hyperspectral images (HSIs) have been used for the retrieval of bio-chemical, geo-chemical, and physical parameters that characterize the surface of the earth. These data are now used in a wide range of applications, aimed at monitoring and implementing new policies in the domain of agriculture, geology, assessment of environmental resources, urban planning, military/defense, disaster management, and so on [2]–[4].

Most of the developments carried out over the last decades in the field of imaging spectroscopy have been achieved via spectrometers onboard airborne platforms. For instance, the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) [5] has been dedicated to remote sensing of the earth in a large number of experiments and field campaigns since the late 1980s. Other examples of airborne missions include the European Space Agency (ESA)'s Airborne Prism Experiment (APEX) (2011–2016) [6] or the Compact Airborne Spectrographic Imager (CASI) [7] (since 1989), among many others.

The vast amount of data collected by airborne platforms has paved the way for EO satellite hyperspectral missions. The Hyperion instrument onboard National Aeronautics and Space Administration (NASA)'s Earth Observing One (EO-1) spacecraft (2000–2017) [8] and the Compact High Resolution Imaging Spectrometer (CHRIS) on ESA's Proba-1 microsatellite [9] (since 2001) have to be the main sources of space-based HSI data in the last decades. Currently, there are several HSI missions under development, including the Environmental Mapping and Analysis Program (EnMAP) [10] and the

Manuscript received January 15, 2019; revised May 4, 2019 and July 3, 2019; accepted July 16, 2019. Date of publication August 14, 2019; date of current version November 25, 2019. This work was supported in part by the Ministerio de Educación (Resolución de 26 de diciembre de 2014 y de 19 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas para la formación de profesorado universitario, de los subprogramas de Formación y de Movilidad incluidos en el Programa Estatal de Promoción del Talento y su Empleabilidad, and en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016), in part by the Junta de Extremadura (decreto 14/2018, ayudas para la realización de actividades de investigación y desarrollo tecnológico, and de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura) under Grant GR18060, in part by the MINECO Project under Grant TIN2015-63646-C5-5-R, in part by the National Science Foundation under Grant ACI-1548562, and in part by the European Union under Grant 734541-EXPOSURE. *(Corresponding author: Juan M. Haut)*

J. M. Haut, J. A. Gallardo, M. E. Paoletti, J. Plaza, and A. Plaza are with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain (e-mail: juanmariohaut@unex.es; mpaoletti@unex.es; jplaza@unex.es; aplaza@unex.es).

G. Cavallaro is with the Jülich Supercomputing Center, 52428 Jülich, Germany (e-mail: g.cavallaro@fz-juelich.de).

M. Riedel is with the Jülich Supercomputing Center, 52428 Jülich, Germany, and also with the Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, 107 Reykjavik, Iceland (e-mail: m.riedel@fz-juelich.de)

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TGRS.2019.2929731

Prototype Research Instruments and Space Mission technology Advancement (PRISMA) [11], among others.

The adoption of an open and free data policy by the NASA [12] and, more recently, by ESA's Copernicus initiative (the largest single EO program) [13] is now producing an unprecedented amount of data to the research community. Even though the Copernicus space component (i.e., the Sentinels) has not included a hyperpectral instrument yet (Sentinel-10 is an HSI mission expected to be operational around 2025–2030), it has been shown that the vast amount of open data currently available calls for re-definition of the challenges within the entire HSI life cycle (i.e., data acquisition, processing, and application phases). It is not by coincidence that remote sensing data are now described under the big data terminology, with characteristics such as volume (increasing scale of acquired/archived data), velocity (rapidly growing data generation rate and real-time processing needs), variety (data acquired from multiple sources), veracity (data uncertainty/accuracy), and value (extracted information) [14], [15].

In this context, traditional processing methods, such as desktop approaches (i.e., MATLAB, R, SAS, and ENVI), offer limited capabilities when dealing with such large amounts of data, especially regarding the velocity component (i.e., the demand for real-time applications). Despite modern desktop computers and laptops are becoming increasingly more powerful, with multi-core and many-core capabilities, including graphics processing units (GPUs), the limitations in terms of memory and core availability currently limit the processing of large HSI data archives. Therefore, the use of highly scalable parallel/distributed architectures (such as GPUs, clusters [16], grids [17], or clouds [18], [19]) is a mandatory solution to improve the access to and the analysis of such great amount of complex data, in order to provide decision-makers with clear, timely, and useful information [20], [21]. In this context, parallel and distributed computing approaches can be categorized into high-performance computing (HPC) or high-throughput computing (HTC) solutions. Contrary to an HPC system [22] (generally, a supercomputer that includes a massive number of processors connected through a fast dedicated network, i.e., a cluster), an HTC system (for instance, a grid) is more focused on the execution of independent and sequential jobs that can be individually scheduled on many different computing resources, regardless of how fast an individual job can be completed. Cloud computing is the natural evolution of grid computing, adopting its backbone and infrastructure [19] but delivering computing resources as a service over the network connection [23]. In other words, the cloud moves desktop and laptop computing (via the Internet) to a service-oriented platform using large remote server clusters and massive storage to data centers. In this scenario, computing relies on sharing a pool of physical and/or virtual resources rather than on deploying local or personal hardware and software. The process of virtualization has enabled the cost-effectiveness and simplicity of cloud computing solutions [24] (i.e., it exempts users from the need to purchase and maintain complex computing hardware), such as infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS). Several cloud computing resources are currently available commercially on a *pay as you go* model from providers, such as Amazon Web Services (AWS) [25], Microsoft Azure [26], and Google's Compute Engine [27].

Cloud computing infrastructures can rely on several computing frameworks that support the processing of large data sets in a distributed environment. For example, the MapReduce model [28] is the basis of a large number of open-source implementations. The most popular ones are Apache Hadoop [29] and its variant, Apache Spark [30] (an in-memory computing framework). Despite the recent advances in cloud computing technology, not enough efforts have been devoted to exploiting cloud computing infrastructures for the processing of HSI data. However, cloud computing offers a natural solution for the processing of large HSI databases, as well as an evolution of the previously developed techniques for other kinds of computing platforms, mainly due to the capacity of cloud computing to provide Internet-scale, service-oriented computing [31]–[33].

In this paper, we focus on the problem of how to develop scalable data analysis and compression techniques [4], [34]–[36] with the goal of facilitating the management of remotely sensed HSI data. In this sense, deep learning (DL) methods based on neural network architectures have gained significant interest in HSI image analysis and processing [37] due to the flexibility of their architectures, their learning models, and the amount of tasks that they can perform [38]. As a result, dimensionality reduction (DR) of HSIs is a fundamental pre-processing step that is applied before many data transfer, store, and processing operations. On the one hand, when HSI data are efficiently compressed, they can be handled more efficiently onboard satellite platforms with limited storage and downlink bandwidth. On the other hand, since HSI data live primarily in a subspace [39], a few informative features can be extracted from the hundreds of highly correlated spectral bands that comprise HSI data [40] without significantly affecting the data quality (lossy compression of HSIs can still retain informative data for the subsequent processing steps).

Specifically, this paper develops a new cloud implementation of HSI data compression based on neural networks. As in [41], we adopt the Apache Spark framework as well as a map-reduce methodology [24] to carry out our implementation. In addition, we address the DR problem using a non-linear deep autoencoder (AE) neural network instead of the standard linear principal component analysis (PCA) algorithm. In fact, we implement a new scalable cloud implementation of the neural model proposed in [42], which is characterized by its flexibility to perform different tasks beyond DR, such as classification [42] or spectral unmixing [43], and therefore, the proposed methodology can be easily adapted to perform different tasks. Focusing on DR, the performance of our newly proposed cloud-based AE is validated using two widely available and known HSI data sets. Our experimental results show that the proposed implementation can effectively exploit cloud computing technology to efficiently perform non-linear compression of large HSI data sets while accelerating significantly the processing time in a distributed environment.

The remainder of this paper is organized as follows. Section II provides an overview of the theoretical and operational details of the considered AE neural network for HSI data compression and the considered optimization method. Section III presents our cloud-distributed AE network for HSI data compression, describing the details of the network configuration and the distributed implementation. Section IV evaluates the performance of the proposed approach using two widely available HSI data sets, considering the quality of the compression and signal reconstruction and also the computational efficiency of the implementation in a real cloud environment. Finally, Section V concludes this paper, summarizing the obtained results and suggesting some future research directions.

## II. Background

HSI data are characterized by their intrinsically complex spectral characteristics, where samples of the same class exhibit high variability due to data acquisition factors or atmospheric and lighting interferers. DR and feature extraction (FE) methods are fundamental tools for the extraction of discriminative features that reduce the intra-class variability and inter-class similarity [44] present in the HSI data sets. Furthermore, by reducing the high spectral dimensionality of HSIs, these methods are able to alleviate the curse of dimensionality [45], which makes HSI data difficult to interpret by supervised classifiers due to the Hughes phenomenon [46].

Several methods have been developed to perform DR and FE from HSIs, For instance, Kang *et al.* [47] proposed a decolorization method to reduce the spectral dimensionality of HSI scenes, preserving most of the information contained in them. In [48], they adopted DR methods with data filtering, implementing image fusion and recursive filtering (IFRF) to preserve the physical meaning of the spectral pixels. In [49], they implemented morphological attribute thinning and thickening (attribute filtering) to perform advanced FE for HSI interpretation. Other interesting DR and FE methods are the independent component analysis (ICA) [50], [51] or the maximum noise fraction (MNF) [52], [53], being PCA [54]–[56] one of the most widely used methods for FE purposes. This unsupervised, linear algorithm reduces the original high-dimensional and correlated feature space to a lower dimensional space of uncorrelated factors [also called principal components (PCs)] by applying an orthogonal transformation through a projection matrix, which makes it a simple yet efficient algorithm. However, PCA is restricted to a linear map projection and is not able to learn non-linear transformations. In this context, auto-associative[1] neural networks, such as AEs [57], offer a more flexible architecture for FE and DR purposes, managing the non-linearities of the data through an architecture made up of stacked layers and non-linear activation functions [called stacked AE (SAE)] that can provide more detailed data representations from the original input image (one per layer), which can be easily reused by other HSI processing methods [42], [43].

[1]Auto-associative networks are those whose inputs can be inferred from the outputs. The proposed implementation of a non-linear AE belongs to this kind of networks.

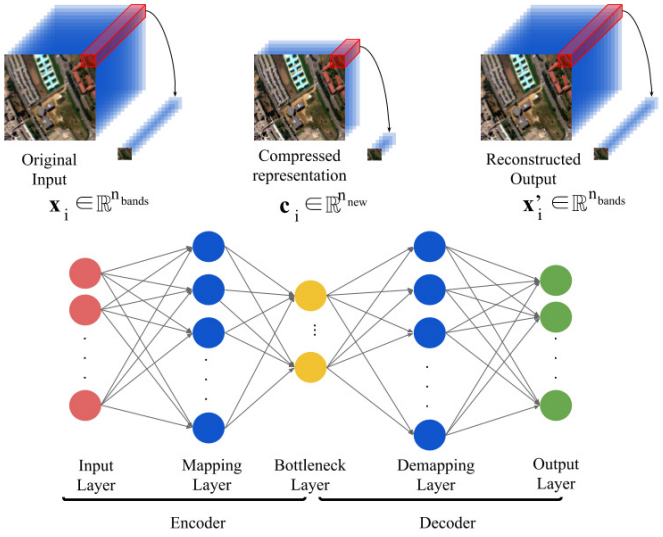



Fig. 1. Graphic representation of a traditional AE for spectral compression and restoration of HSIs.

### A. Autoencoder Neural Network

Let us consider an HSI data cube $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times n_{\text{bands}}}$, where $n_1 \times n_2$ are the spatial dimensions and $n_{\text{bands}}$ is the number of spectral bands. $\mathbf{X}$ is traditionally observed by pixel-based algorithms as a collection of $n_1 \times n_2$ spectral samples, where each $\mathbf{x}_i \in \mathbb{R}^{n_{\text{bands}}} = [x_{i,1}, x_{i,2}, \ldots, x_{i,n_{\text{bands}}}]$ contains the spectral signature of the observed surface material. In this sense, the goal of DR methods is to obtain, for each $\mathbf{x}_i$, a vector $\mathbf{c}_i \in \mathbb{R}^{n_{new}}$ that captures the most representative information of $\mathbf{x}_i$ in a lower feature space, being $n_{new} << n_{\text{bands}}$. To achieve this goal, the SAE applies an unsupervised symmetrical deep neural network to encode the data in a lower dimensional latent space, performing a traditional embedding and then decoding it to the original space through a reconstruction stage. In fact, the SAE can be interpreted as a mirrored net, where three main parts can be identified, as shown in Fig. 1: 1) the encoder or mapping layers; 2) the middle or bottleneck layer; and 3) the decoder or demapping layers. Based on the traditional multilayer perceptron (MLP), the $l$th layer defined in the SAE performs an affine transformation between the input data $\mathbf{x}_i^{(l)}$ and its set of weights $\mathbf{W}^{(l)}$ and biases $b^{(l)}$, as shown in the following equation:

$$\mathbf{x}_i^{(l+1)} = \mathcal{H}(\mathbf{x}_i^{(l)} \cdot \mathbf{W}^{(l)} + b^{(l)}) \tag{1}$$

where $\mathbf{x}_i^{(l+1)} \in \mathbb{R}^{n^{(l)}}$ is an abstract representation (or feature representation) of the original input data $\mathbf{x}_i$ in the feature space obtained by the $n^{(l)}$ neurons that compose the $l$th layer, where the output of the $k$th neuron is obtained as the dot product between the $n^{(l-1)}$ outputs of the previous layer and its weights, passed through an activation function that is usually implemented by the rectified linear unit (ReLU) [58], i.e., $\mathcal{H}(x) = \max(0, x)$. Finally, the $k$th feature in $\mathbf{x}_i^{(l+1)}$ can be obtained as

$$x_{i,k}^{(l+1)} = \mathcal{H}\left(\sum_{j=1}^{n^{(l-1)}} (x_{i,j}^{(l)} \cdot w_{k,j}^{(l)}) + b^{(l)}\right). \tag{2}$$

With this in mind, the SAE applies two main processing steps to each input sample $\mathbf{x}_i$. The first one, known as coding stage, performs the embedding of the data, mapping it from $\mathbb{R}^{n_{\text{bands}}}$ space to $\mathbb{R}^{n_{new}}$ latent space, that is, the $n_{encoder}$ layers of the encoder map their input data to a projected representation following (1) and (2) until reaching the bottleneck layer. As a result, the bottleneck layer contains the projection of each $\mathbf{x}_i \in \mathbb{R}^{n_{\text{bands}}}$ in its latent space, defined by its $n_{new}$ neurons, $\mathbf{c}_i \in \mathbb{R}^{n_{new}}$. As a result, the SAE allows to generate compressed ($n_{\text{new}} < n_{\text{bands}}$), extended ($n_{new} > n_{\text{bands}}$), or even equally ($n_{new} = n_{\text{bands}}$) dimensional representations, depending on the final dimension of the code vector $\mathbf{c}_i$.

The second stage performs the opposite operation, i.e., the decoding, where the network tries to recover the original information, obtaining an approximate reconstruction of the original input vector [59]. In this case, the $n_{decoder}$ layers of the decoder demap the code vector $\mathbf{c}_i$ until reaching the output layer, where a reconstructed sample $\mathbf{x}'_i$ is obtained. Equation (3) shows an overview of the encoding–decoding process followed by the SAE:

$$\begin{aligned}\mathbf{c}_i &\leftarrow \text{For } l \text{ in } n_{\text{encoder}}\text{: } \mathbf{x}_i^{(l+1)} = \mathcal{H}\left(\mathbf{x}_i^{(l)} \cdot \mathbf{W}^{(l)} + b^{(l)}\right)\\ \mathbf{x}'_i &\leftarrow \text{For } ll \text{ in } n_{\text{decoder}}\text{: } \mathbf{c}_i^{(ll+1)} = \mathcal{H}\left(\mathbf{c}_i^{(ll)} \cdot \mathbf{W}^{(ll)} + b^{(ll)}\right).\end{aligned} \quad (3)$$

In order to obtain a lower dimensional (but more discriminative) representation of the input data, the network parameters are iteratively adjusted in an unsupervised fashion, where the optimizer minimizes the reconstruction error between the input data at the encoding stage, $\mathbf{x}_i$, and its reconstruction at the end of the decoding stage, $\mathbf{x}'_i$. This error function, given by (4), is usually implemented in the form of a mean squared error (MSE)

$$E(\mathbf{X}) = \min \parallel \mathbf{X} - \mathbf{X}' \parallel_2 = \min \sum_{i=1}^{n_1 \cdot n_2} \parallel \mathbf{x}_i - \mathbf{x}'_i \parallel_2 . \quad (4)$$

### *B. Broyden–Fletcher–Goldfarb–Shanno Algorithm*

After describing the operational procedure of SAEs, it is now important to observe the network optimization process. As any artificial neural network with backpropagation, the optimizer tries to find the set of parameters (synaptic weights and biases) that, for a given network architecture, minimize the error function $E(\mathbf{X})$ defined by (4). This function evaluates how well the neural network fits the data set $\mathbf{X}$ and depends on the adaptative and learnable parameters of the network, which can be denoted as $\mathcal{W}$, so $E(\mathbf{X}, \mathcal{W})$. As $E(\mathbf{X}, \mathcal{W})$ is non-linear, its optimization must be carried out iteratively, reducing its value until an adequate stopping criterion is reached. In this sense, standard optimizers back-propagate the error signal through the network architecture calculating, for each learnable parameter, the gradient of the error, i.e., the direction and displacement that the parameter must undergo in order to minimize the final error (also interpreted as the importance of that parameter when obtaining the final error). Mathematically, the updating of $\mathcal{W}$ in the $t$th epoch can be calculated by

$$\begin{aligned}\mathcal{W}_{t+1} &= \mathcal{W}_t + \Delta\mathcal{W}\\ \text{being } \Delta\mathcal{W} &= \mu_t \cdot \mathbf{p}_t\end{aligned} \quad (5)$$

where $\mu_t$ and $\mathbf{p}_t$ are the learning rate (a positive scalar) and the descent search direction, respectively [60]. The main goal of any optimizer is to obtain the correct $\mathbf{p}_t$ in order to descend properly in the error function until the minimum is reached.

As opposed to standard optimizers, traditional Newton-based methods determine the descent direction $\mathbf{p}_t$ using the second derivative information contained into the Hessian matrix, rather than just the gradient information, thus stabilizing the process

$$\begin{aligned}\mathbf{H}_t \cdot \mathbf{p}_t &= -\nabla E(\mathbf{X}, \mathcal{W}_t)\\ \mathbf{p}_t &= -\mathbf{H}_t^{-1} \cdot \nabla E(\mathbf{X}, \mathcal{W}_t)\\ \mathcal{W}_{t+1} &= \mathcal{W}_t - \mu_t \cdot \mathbf{H}_t^{-1} \cdot \nabla E(\mathbf{X}, \mathcal{W}_t)\end{aligned} \quad (6)$$

where $\nabla E(\mathbf{X}, \mathcal{W}_t)$ is the gradient of the error function evaluated with the network's parameters at the $t$th epoch, $\mathcal{W}_t$, and $\mathbf{H}_t$ and $\mathbf{H}_t^{-1}$ are, respectively, the Hessian matrix and its inverse obtained at the $t$th epoch. However, these methods obtain the Hessian matrix and its inverse at each epoch, which is quite expensive to compute, requiring a large amount of memory. Instead of that, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [61] performs an estimation of how the Hessian matrix has changed in each epoch, obtaining an approximation (instead of the full matrix) that is improved for every epoch. In fact, as any algorithm of the family of multivariate minimization quasi-Newton methods, the BFGS algorithm modifies the last expression of (6) as follows:

$$\mathcal{W}_{t+1} = \mathcal{W}_t - \mu_t \cdot \mathbf{G}_t \cdot \nabla E(\mathbf{X}, \mathcal{W}_t) \quad (7)$$

where $\mathbf{G}_t$ is the inverse Hessian approximation matrix (usually, when $t = 0$ the initial approximation matrix is the identity matrix, $\mathbf{G}_0 = \mathbf{I}$). This $\mathbf{G}_t$ is updated at each epoch by means of an update matrix

$$\mathbf{G}_{t+1} = \mathbf{G}_t + \mathbf{U}_t. \quad (8)$$

However, such an update needs to comply with the quasi-Newton condition, which is described next. Assuming that $E(\mathbf{X}, \mathcal{W})$ is continuous for $\mathcal{W}_t$ and $\mathcal{W}_{t+1}$ (with gradients $\mathbf{g}_t = \nabla E(\mathbf{X}, \mathcal{W}_t)$ and $\mathbf{g}_{t+1} = \nabla E(\mathbf{X}, \mathcal{W}_{t+1})$, respectively) and the Hessian $\mathbf{H}$ is constant, then (9) is satisfied

$$\begin{aligned}&\mathbf{q}_t \equiv \mathbf{g}_{t+1} - \mathbf{g}_t \text{ and } \mathbf{p}_t \equiv \mathcal{W}_{t+1} - \mathcal{W}_t\\ &\text{Secant condition on the Hessian: } \mathbf{q}_t = \mathbf{H} \cdot \mathbf{p}_t\\ &\text{Secant condition on the inverse: } \mathbf{H}^{-1} \cdot \mathbf{q}_t = \mathbf{p}_t.\end{aligned} \quad (9)$$

Since $\mathbf{G} = \mathbf{H}^{-1}$, the last expression in (9) can be modified to $\mathbf{G} \cdot \mathbf{q}_t = \mathbf{p}_t$, so the approximation matrix $\mathbf{G}$ can be obtained (at each epoch $t$) as a combination of the linearly independent directions and their respective gradients. Following the Davidon–Fletcher–Powell (DFP) rank 2 formula [62], $\mathbf{G}$ can be updated using

$$\mathbf{G}_{t+1} = \mathbf{G}_t + \frac{\mathbf{p}_t \cdot \mathbf{p}_t^\top}{\mathbf{p}_t^\top \cdot \mathbf{q}_t} - \mathbf{G}_t \cdot \frac{\mathbf{q}_t \cdot \mathbf{q}_t^\top}{\mathbf{q}_t^\top \cdot \mathbf{G}_t \cdot \mathbf{q}_t} \cdot \mathbf{G}_t. \quad (10)$$

Finally, the BFGS method updates its approximation matrix by computing the complementary formula of the DFP method,

**Algorithm 1** BFGS Algorithm

1: **procedure** BFGS($\mathcal{W}_t$: current parameters of the neural network, $E(\mathbf{X}, \mathcal{W})$: Error function, $\mathbf{G}_t$: current approximation to the Hessian)
2: $\mathbf{g}_t = \nabla E(\mathbf{X}, \mathcal{W}_t)$
3: $\mathbf{p}_t = -\mathbf{G}_t \cdot \mathbf{g}_t$
4: $\mathcal{W}_{t+1} = \mathcal{W}_t + \mu_t \cdot \mathbf{p}_t$ ▷ $\mu_t$ by linear search
5: $\mathbf{g}_{t+1} = \nabla E(\mathbf{X}, \mathcal{W}_{t+1})$
6: $\mathbf{q}_t = \mathbf{g}_{t+1} - \mathbf{g}_t$
7: $\mathbf{p}_t = \mathcal{W}_{t+1} - \mathcal{W}_t$
8: $\mathbf{A} = \left(1 + \frac{\mathbf{q}_t^\top \cdot \mathbf{G}_t \cdot \mathbf{q}_t}{\mathbf{q}_t^\top \cdot \mathbf{p}_t}\right) \cdot \left(\frac{\mathbf{p}_t \cdot \mathbf{p}_t^\top}{\mathbf{p}_t^\top \cdot \mathbf{q}_t}\right)$
9: $\mathbf{B} = \frac{\mathbf{p}_t \cdot \mathbf{q}_t^\top \cdot \mathbf{G}_t + \mathbf{G}_t \cdot \mathbf{q}_t \cdot \mathbf{p}_t^\top}{\mathbf{q}_t^\top \cdot \mathbf{p}_t}$
10: $\mathbf{G}_{t+1} = \mathbf{G}_t + \mathbf{A} - \mathbf{B}$
**return** $\mathcal{W}_{t+1}$, $\mathbf{G}_{t+1}$
11: **end procedure**

changing $\mathbf{G}$ by $\mathbf{H}$ and $\mathbf{p}_t$ by $\mathbf{q}_t$, and therefore, (10) is finally modified as follows:

$$\mathbf{H}_{t+1} = \mathbf{H}_t + \frac{\mathbf{q}_t \cdot \mathbf{q}_t^\top}{\mathbf{q}_t^\top \cdot \mathbf{p}_t} - \mathbf{H}_t \cdot \frac{\mathbf{p}_t \cdot \mathbf{p}_t^\top}{\mathbf{p}_t^\top \cdot \mathbf{H}_t \cdot \mathbf{p}_t} \cdot \mathbf{H}_t. \quad (11)$$

As the BFGS method intends to compute the inverse of $\mathbf{H}$ and $\mathbf{G} = \mathbf{H}^{-1}$, it inverts (11) to analytically obtain the final update of the approximation matrix

$$\mathbf{G}_{t+1} = \mathbf{G}_t + \left(1 + \frac{\mathbf{q}_t^\top \cdot \mathbf{G}_t \cdot \mathbf{q}_t}{\mathbf{q}_t^\top \cdot \mathbf{p}_t}\right) \cdot \left(\frac{\mathbf{p}_t \cdot \mathbf{p}_t^\top}{\mathbf{p}_t^\top \cdot \mathbf{q}_t}\right) - \frac{\mathbf{p}_t \cdot \mathbf{q}_t^\top \cdot \mathbf{G}_t + \mathbf{G}_t \cdot \mathbf{q}_t \cdot \mathbf{p}_t^\top}{\mathbf{q}_t^\top \cdot \mathbf{p}_t}. \quad (12)$$

Algorithm 1 provides a general overview of how the BFGS method works in one epoch. As shown in line 4 of Algorithm 1, as opposed to most optimization algorithms, the learning rate is inferred by performing a linear search instead of using an input parameter. A weakness of BFGS is that it requires the computation of the gradient on the full data set, consuming a large amount of memory to properly run the optimization. Considering the dimensionality of HSIs, we can conclude that this method is not able to scale with the number of samples [63]. In order to overcome this limitation, and with the aim of speeding up the computation of both the forward (affine transformations) and backward (optimizer) steps of the AE for DR of HSIs, in Section III, we develop a distributed solution for cloud computing environments.

## III. Proposed Implementation

### A. Distributed Framework

We have developed a completely new distributed AE for HSI data analysis[2] that follows the diagram shown in Fig. 2, where it is shown that the proposed cloud-based neural model has been implemented as a master–slave, multi-node environment. In this context, two problems have been specifically addressed in this paper: 1) the computing engine and 2) the distributed programming model over the cloud architecture.

[2]Code available on: https://github.com/jgallardst/cloud-nn-hsi

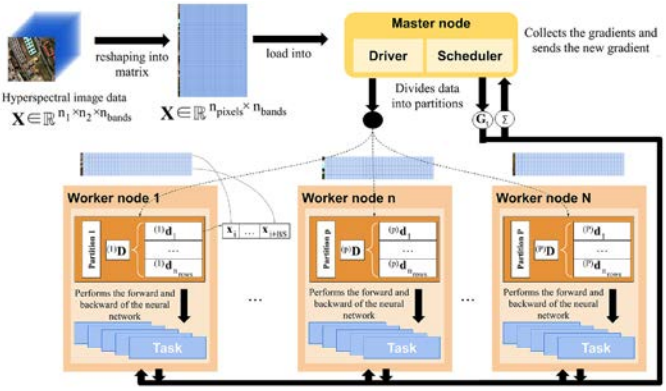



Fig. 2. Diagram summarizing the overall framework of the proposed cloud implementation, where the master node takes the HSI data and partitions it between the different workers' nodes, which apply the tasks extracted from the graph of transformations, executing the forward and backward steps over their data. The obtained gradients are collected and pooled by the master node, which transmits the final gradient to the neural models stored by each worker node.

Regarding the first problem, our distributed implementation of the network model is run on top of a standalone Spark cluster, due to its capacity to provide fast processing of large data volumes on distributed platforms, in addition to fault tolerance. Furthermore, the Spark cluster is characterized by a master–slave architecture, which makes it very flexible. Specifically, a Spark cluster is formed by a master node, which manages how the resources are used and distributed in the cluster by hosting a Java virtual machine (JVM) driver, and the scheduler, which distributes the tasks between the execution nodes and $N$ worker nodes (which can be more than one per node) that execute the program tasks by creating a Java distributed agent, called executor (where tasks are computed), and store the data partitions (see Fig. 3).

In relation to the second point, the adopted programming model to perform the implementation of the distributed AE is based on organizing the original HSI data in tuples or key/value pairs, in order to apply the *MapReduce* model [41], which divides the data processing task into two distributed operations: 1) mapping, which processes a set of data tuples, generating intermediate key–value pairs and 2) reduction, which gathers all the intermediate pairs obtained by the mapping to generate the final result. In order to achieve this behavior, data information in Spark is abstracted and encapsulated into a fault-tolerant data structure called Resilient Distributed data set (RDD). These RDDs are organized as distributed collections of data, which are scattered by Spark across the worker nodes when they are needed on the successive computations, being persisted in the memory of the nodes or on the disk. This architecture allows for the parallelization of the executions, achieved by performing MapReduce tasks over the RDDs on the nodes. Moreover, two basic operations can be performed on an RDD: 1) the so-called transformations that are based on applying an operation to every row on a partition, resulting in another RDD and 2) actions that retrieve a value or a set of values that can be both RDD data or the result of an operation where some RDD data are involved. Operations are queued until an action is called; the needed

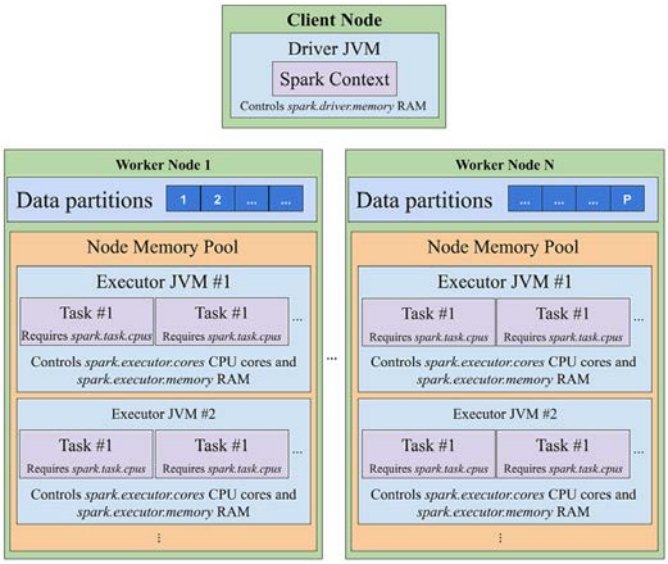



Fig. 3. Graphic representation of a generic Spark cluster, which is composed by one client node and $N$ worker nodes, where in each node, several executor JVMs are running in parallel over several data partitions.

transformations are placed into a dependence graph, where each node is a job stage, following a lazy execution paradigm. This means that operations are not performed until they are really needed, avoiding the repetition of a single operation more than once.

In order to enable a simple and easy mechanism for managing large data sets, the Spark environment provides another level of abstraction that uses the concept of Dataframe. These Dataframes allow data to be organized on named columns, being easier to manipulate (as in relational tables, columns can be accessed by the column name instead of by the index). With this in mind, the Spark standalone cluster functionality can be summarized as follows.

1) The master node (also called driver node) creates and manages the Spark driver (see Fig. 3), a Java process that contains the SparkContext of the application.
2) The driver context performs the data partitioning and parallelization between the worker nodes, assigning to each one a number of partitions, which depends on two main aspects: the block size and the way that the data are stacked. Also, the driver creates the executors on the worker nodes, which store data partitions on the worker node and perform tasks on them.
3) When an action is called, a job is launched and the master coordinates how the associated tasks are distributed into the different executors. In order to reduce the data exchanging time, the Spark driver attempts to perform "smart" task allocations so that there are more possibilities to assign a task to the executor, located in the worker where the data partition used by the task to perform the operation has been allocated.
4) When all the tasks on a given stage are finished, the Scheduler allocates another stage of the job (if it was a transformation) or retrieves the final output (if it was an action).

**Algorithm 2** Iterative Process

```
procedure SPARK FLOW
    PartitionedData ← Spark.parallelizeData()
    t ← 0
    while t < n_iterations do
        broadcastOutputData().
        foreach partition ∈ PartitionedData do
            PartitionedData.applyTask().
        end for
        retrieveOutputData().
        t ← t + 1
    end while
end procedure
```

Algorithm 2 shows a general overview of how our algorithm is pipelined in the considered Spark cluster.

### *B. Cloud Autoencoder Pipeline Implementation: Training and Optimization Process*

This section describes in detail the full distributed training process, from the parallelization of HSI data across nodes to the intrinsic logic of each training step, explaining the benefits of our distributed training algorithm. Fig. 4 shows a general overview of the full data pipeline developed in this paper.

In the beginning, the original 3-D HSI data cube $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times n_{\text{bands}}}$, where $n_1 \times n_2$ are the spatial dimensions (height and width) and $n_{\text{bands}}$ is the spectral dimension given by the number of spectral channels, is reshaped into an HSI matrix $\mathbf{X} \in \mathbb{R}^{n_{\text{pixels}} \times n_{\text{bands}}}$, where $n_{\text{pixels}} = n_1 \times n_2$, i.e., each row collects a full spectral pixel, being each column the corresponding value in the spectral band. This matrix $\mathbf{X}$ is read by the Spark Driver, which collects the original HSI data and partitions it into $P$ smaller subsets that are delivered to the worker nodes in parallel. These workers store the obtained partitions on their local disks. In this sense, each data partition composes an RDD.

It must be noted that complex neural network topologies derive on greedy RAM memory usage on the driver node. Since Spark transformations apply an operation to every row of the RDD, the fewer the number of rows, the fewer the number of operations that must be carried out. In order to improve the computation of the distributed model, a blocksize (BS) hyperparameter is provided, with the aim of indicating how many pixels should be stacked into a single row in order to compute them together. With this observation in mind, the $p$th data partition (with $p = 1, \ldots, P$) can be seen as a 2-D matrix ${}^{(p)}\mathbf{D} \in \mathbb{R}^{n_{\text{rows}} \times (\text{BS} \cdot n_{\text{bands}})}$ composed by $n_{\text{rows}}$ rows, where each one stores BS concatenated spectral pixels, i.e., ${}^{(p)}\mathbf{d}_j \in \mathbb{R}^{(\text{BS} \cdot n_{\text{bands}})} = [\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+\text{BS}}]$. In the end, each data partition ${}^{(p)}\mathbf{D}$ stores $\text{BS} \cdot n_{\text{rows}}$ pixels. The resulting partitions are then distributed across the worker nodes. Such distribution allows the executors, located in each worker node, to apply the subsequent tasks to those partitions that each worker receives.

After distributing the data into RDDs, a distributed data analysis process begins prior to the application of neural

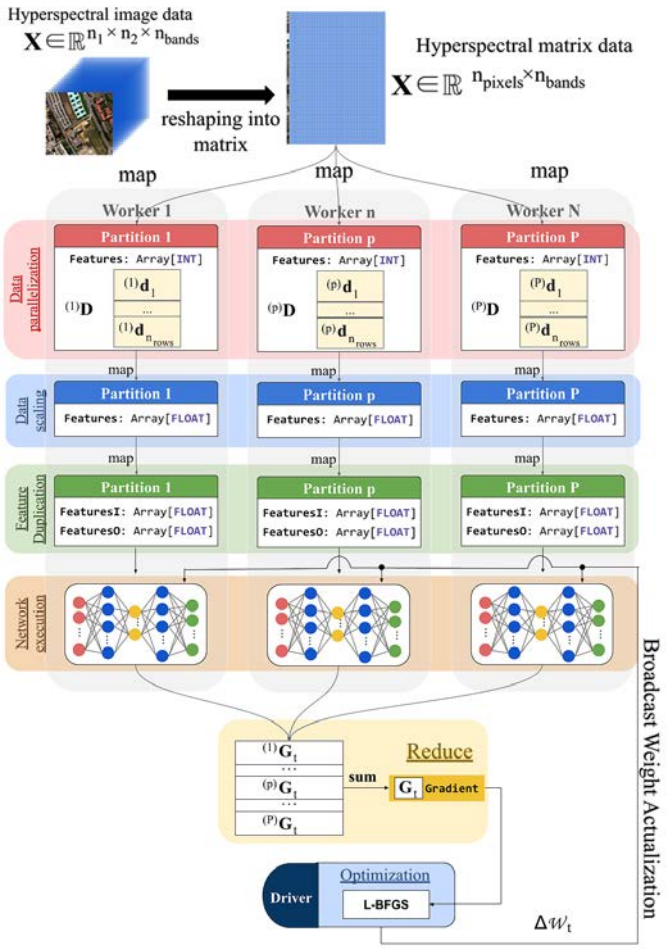



Fig. 4. Data pipeline of our distributed AE, where the input HSI cube is first reshaped into a matrix and then split into several partitions allocated into the Spark worker nodes, composed by several rows where each one contains BS spectral pixels. These data partitions are then duplicated in order to obtain the input network data and the corresponding output network data. The AE is then executed and, for each iteration $t$, the gradients are collected by the Spark driver, which calculates the final gradient and performs the optimization with the L-BFGS algorithm. The updated weights are finally broadcasted to each neural model contained in the cluster.

network-based processing. In the first step, the data contained in each partition ${}^{(p)}\mathbf{D}$ are scaled in a distributed way, taking advantage of the cloud architecture and the available parallelization of resources. In this sense, each partition's row ${}^{(p)}\mathbf{d}_j$ (and, internally, each pixel contained within) is transformed based on the global maximum and minimum features ($\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$) of the whole image $\mathbf{X}$, and the column local maximum and minimum features [${}^{(p)}\mathbf{d}_{\max}$ and ${}^{(p)}\mathbf{d}_{\min}$] of the $p$th partition where the data are allocated

$$
\begin{aligned}
{}^{(p)}\hat{\mathbf{d}}_j &= \frac{{}^{(p)}\mathbf{d}_j - {}^{(p)}\mathbf{d}_{\min}}{{}^{(p)}\mathbf{d}_{\max} - {}^{(p)}\mathbf{d}_{\min}} \\
{}^{(p)}\mathbf{d}_j &= {}^{(p)}\hat{\mathbf{d}}_j \cdot (\mathbf{x}_{\max} - \mathbf{x}_{\min}) + \mathbf{x}_{\min}.
\end{aligned} \tag{13}
$$

Once the HSI data have been split into partitions and scaled, the next step consists of the application of the AE model. The proposed AE is composed of five layers, as summarized in Table I. These layers are: $l^{(1)}$, the input layer that receives the spectral signature contained in each pixel $\mathbf{x}_i$ of $\mathbf{X}$ (i.e., the rows of the distributed partitions), composed by as many

TABLE I
TOPOLOGY OF THE PROPOSED AE NEURAL NETWORK FOR HSI ANALYSIS

| **Layer ID** | $l^{(1)}$ | $l^{(2)}$ | $l^{(3)}$ | $l^{(4)}$ | $l^{(5)}$ |
|---|---|---|---|---|---|
| **Neurons per** $l^{(i)}$ | $n_{bands}$ | 140 | 60 | 140 | $n_{bands}$ |

neurons as spectral bands; $l^{(2)}$, $l^{(3)}$, and $l^{(4)}$, the hidden AE layers, and $l^{(5)}$, the output layer that obtains the reconstructed signature $\mathbf{x}'_i$, composed also by as many neurons as spectral bands.

With the topology described in Table I in mind, the encoder part is composed by $l^{(1)}$, $l^{(2)}$, and $l^{(3)}$, which performs the mapping from the original spectral space to the latent space of the bottleneck layer $l^{(3)}$. In addition, the decoder part is composed by $l^{(3)}$, $l^{(4)}$, and $l^{(5)}$, which performs the demapping from the latent space of $l^{(3)}$ to the original spectral space.

At this point, it is interesting to briefly comment the performance of the AE network. In order to correctly propagate the data through the network, from each partition ${}^{(p)}\mathbf{D} \in \mathbb{R}^{n_{\text{rows}} \times (\text{BS} \cdot n_{\text{bands}})}$, a matrix of unstacked pixels ${}^{(p)}\mathbf{X} \in \mathbb{R}^{(\text{BS} \cdot n_{\text{rows}}) \times n_{\text{bands}}}$ is extracted, i.e., the BS spectral pixels that are contained in each ${}^{(p)}\mathbf{d}_j = [\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+\text{BS}}]$ (with $j = 1, \ldots, n_{\text{rows}}$ and $i = 1, \ldots, n_{\text{pixels}}$) are each extracted to create, one by one, the rows of ${}^{(p)}\mathbf{X}$, denoted as ${}^{(p)}\mathbf{x}_k$ [with $k = 1, \ldots, (\text{BS} \cdot n_{\text{rows}})$] in order to determine the level at which the AE is working.

Every training iteration $t$ is performed using the traditional neural network forward–backward procedure, in addition to a tree-aggregate operation that computes and sums the executors' gradients and losses to return a single loss value and a matrix of gradients. Each executor computes its loss by forwarding the input network data ${}^{(p)}\mathbf{X}$ through the AE layers and comparing the $l^{(5)}$ layer's output vector with the vector of input features, following (4) and obtaining (at each $t$) the corresponding MSE of the partition: ${}^{(p)}\text{MSE}_t = E({}^{(p)}\mathbf{X}, \mathcal{W}_t)$. Gradients are then computed by back-propagating the error signal through the AE, obtaining for each partition the ${}^{(p)}\mathbf{G}_t$ matrix at iteration $t$. Each gradient matrix is reduced in the Driver, which runs the optimizer in order to obtain the final matrix $\Delta\mathcal{W}_t$. This matrix indicates how much each neuron weight should be modified before finishing the $t$th training iteration, based on how that neuron impacts the output. Fig. 5 shows a graphical overview of the adopted training procedure.

If we denote by $P$ the number of total partitions and by ${}^{(p)}\mathbf{X} \in \mathbb{R}^{(\text{BS} \cdot n_{\text{rows}}) \times n_{\text{bands}}}$ the $p$th unstacked partition data, composed by (BS $\times$ $n_{\text{rows}}$) normalized rows/feature vectors of $n_{\text{bands}}$ spectral features, i.e., ${}^{(p)}\mathbf{x}_k \in \mathbb{R}^{n_{\text{bands}}} = [{}^{(p)}x_{k,1}, \ldots, {}^{(p)}x_{k,n_{\text{bands}}}]$, and considering the $l$th layer of the AE model, composed by $n^{(l)}_{\text{neurons}}$, its output is denoted by ${}^{(p)}\mathbf{X}^{(l+1)}$ and it is computed by adapting (1) into (14) as the matrix multiplication

$$
{}^{(p)}\mathbf{X}^{(l+1)} = \mathcal{H}({}^{(p)}\mathbf{X}^{(l)} \cdot \mathbf{W}^{(l)} + b^{(l)}) \tag{14}
$$

where the meaning of each term is given in the following.

1) ${}^{(p)}\mathbf{X}^{(l+1)} \in \mathbb{R}^{(\text{BS} \times n_{\text{rows}}) \times n^{(l)}_{\text{neurons}}}$ is the matrix that represents the output of the neurons in layer $l$ with size $(\text{BS} \cdot n_{\text{rows}}) \times n^{(l)}_{\text{neurons}}$, where $n^{(l)}_{\text{neurons}}$ is the number of

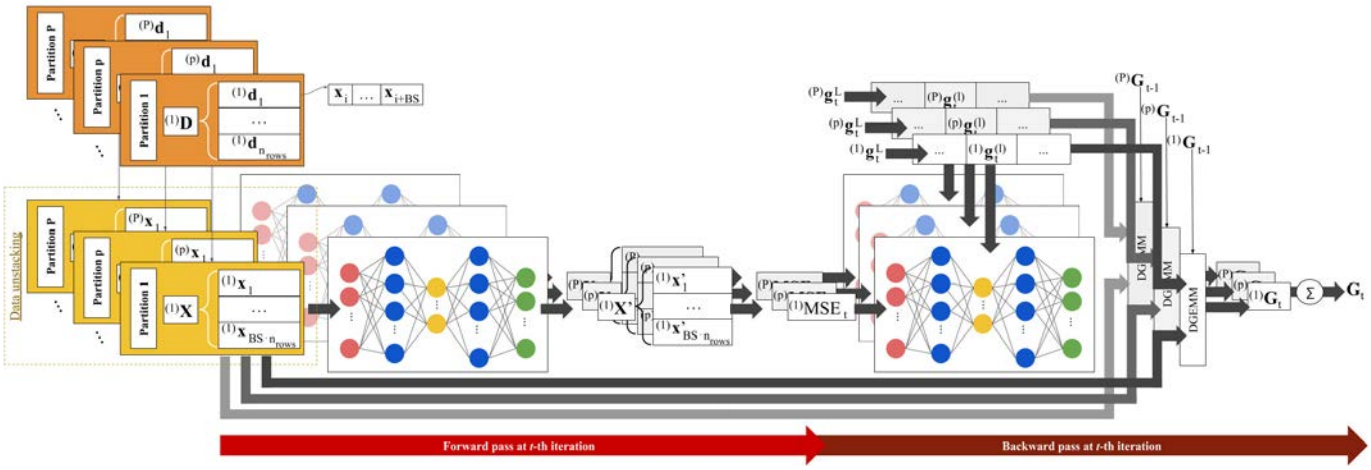



Fig. 5. Distributed forward and backward pipelines of the training stage (at iteration $t$) after unstacking the hyperspectral pixels in each distributed data partition (each one allocated to a different worker node).

neurons of the $l$th layer (in the case that $l = 5$ and $n^{(5)}_{\text{neurons}} = n_{\text{bands}}$).

2) ${}^{(p)}\mathbf{X}^{(l)} \in \mathbb{R}^{(\text{BS}\times n_{\text{rows}})\times n^{(l-1)}_{\text{neurons}}}$ is the matrix that serves as the input to the $l$th layer, which contains the $(\text{BS}\cdot n_{\text{rows}})$ pixel vectors represented in the feature space of the previous layer, defined by $n^{(l-1)}_{\text{neurons}}$ neurons.
3) $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l-1)}_{\text{neurons}}\times n^{(l)}_{\text{neurons}}}$ is the matrix of weights, which connects the current $n^{(l)}_{\text{neurons}}$ neurons with the $n^{(l-1)}_{\text{neurons}}$ neurons of the previous layer, and $b^{(l)}$ is the bias of the current layer.
4) $\mathcal{H}$ is the ReLU activation function, which gives the following non-linear output: $\text{ReLU}(x) = \max(0, x)$.

After data forwarding, the reconstructed data ${}^{(p)}\mathbf{X}'$ in the $p$th partition at the $t$th iteration are compared to the original input ${}^{(p)}\mathbf{X}$ by applying the MSE function defined by (4) on each executor. Executors then retrieve the error computed by their carried data, obtaining a value ${}^{(p)}\text{MSE}_t$ per partition. Then, the final error is obtained as the mean of all executor errors, as shown in the following equation:

$$
\begin{aligned}
{}^{(p)}\text{MSE}_t &= \frac{1}{(\text{BS}\times n_{\text{rows}})}\sum_{k=1}^{(\text{BS}\times n_{\text{rows}})} \|{}^{(p)}\mathbf{x}_k - {}^{(p)}\mathbf{x}'_k\|_2 \\
\text{MSE}_t &= \frac{1}{P}\sum_{p=1}^{P} {}^{(p)}\text{MSE}_t
\end{aligned}
\tag{15}
$$

where $(\text{BS}\times n_{\text{rows}})$ is the number of pixels that compose the $p$th data partition, whereas ${}^{(p)}\mathbf{x}_k \in {}^{(p)}\mathbf{X}$ and ${}^{(p)}\mathbf{x}'_k \in {}^{(p)}\mathbf{X}'$ are the original input sample and output reconstructed sample in the $p$th data partition, respectively. Those partition errors are then back-propagated to compute the gradient ${}^{(p)}\mathbf{G}_t$ matrix of each partition at iteration $t$. In this sense, for each layer in the neural model (using the resulting outputs), the impact that each neuron has on the final error is obtained as the result of the ReLU's derivative of every output, which is defined as follows:

$$
\mathcal{H}'(x) = \begin{cases} 0, & \text{if } x \le 0 \\ 1, & \text{if } x > 0. \end{cases}
\tag{16}
$$

Such impact can be denoted as ${}^{(p)}g^L_t = [{}^{(p)}g^{(1)}_t, \ldots, {}^{(p)}g^{(5)}_t]$, where the $l$th element ${}^{(p)}g^{(l)}_t$ stores the impact of the $n^{(l)}_{\text{neurons}}$ allocated into the $l$th layer of the network.

The gradient of each partition, ${}^{(p)}\mathbf{G}_t$, is then computed by applying the double-precision general matrix to matrix multiplication (DGEMM) operation where, given three input matrices ($\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$) and two constants ($\alpha$ and $\beta$), the obtained results are calculated by (17) and stored in $\mathbf{C}$

$$
\mathbf{C} = \alpha * \mathbf{A} * \mathbf{B} + \beta * \mathbf{C}. \tag{17}
$$

DGEMM is performed to compute the entire gradient matrix in parallel, instead of computing each layer gradient vectors separately. This allows us to make neural computations faster and efficient in terms of reducing power consumption. In this sense, each item of (17) has been replaced by the following parameters.

1) $\alpha = (1/n_{\text{bands}})$ is a parameter regularizer.
2) $\mathbf{A} = {}^{(p)}\mathbf{X}$ is the input data partition matrix.
3) $\mathbf{B} = {}^{(p)}g^L_t$ is the matrix representing the impact of each neuron on every layer of the neural network.
4) $\beta = 1$ is also a parameter regularizer. As $\mathbf{C}$ should be unchanged, it has been set to 1.
5) $\mathbf{C} = {}^{(p)}G_{t-1}$ is initially the older gradient matrix of the $p$th partition. After the updates resulting from the DGEMM operation, the current gradient ${}^{(p)}G_t$ is stored on $\mathbf{C}$.

Finally, the gradient matrix $G_t$ of the whole network is computed as the average of the sum of all partition's gradients: ${}^{(p)}G_t$. The entire training process on each data partition is graphically shown in Fig. 5.

The final optimization step is performed locally on the master node using a variant of the BFGS algorithm, called limited BFGS (L-BFGS). Since BFGS needs a huge amount of memory for the computation of the matrices, L-BFGS limits the memory usage, so it fits better into our implementation. The optimizer uses the computed gradients and a step-size procedure to get closer to a minimum of (4). The procedure is repeated until a desired number of iterations, $n_{\text{iterations}}$, is reached.

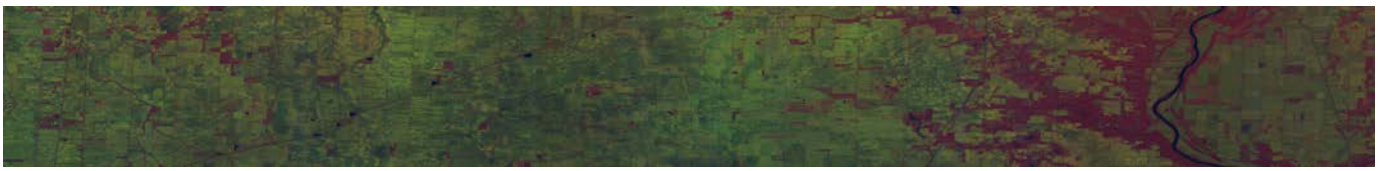


Fig. 6. False RGB color map of the BIP scene, represented using the visualization method in [47].

## IV. Experimental Evaluation

### A. Configuration of the Environment

In order to test our newly developed implementation, a dedicated hardware and software environment based on a high-end cloud computing paradigm has been adopted. The virtual resources have been provided by the JetStream Cloud Services3[3] at the Indiana University Pervasive Technology Institute (PTI) through XSEDE allocation TG-ASC180012 [64]–[66]. Its user interface is based on Atmosphere computing platform4[4] and uses Openstack5[5] as the operational software environment.

The hardware environment consists of a collection of cloud computing nodes. In particular, the cluster consists of one master node and eight slave nodes that are hosted in virtual machines with six virtual cores at 2.5 GHz each. Every node has 16 GB of RAM and 60 GB of storage via a Block Storage File System. As mentioned earlier, Spark performs as the backbone for node interconnection; meanwhile, data transfers are supported by a local $4 \times 40$ Gb/s dedicated network.

Each virtual machine runs Ubuntu 16.04 as operating system, with Spark 2.1.1 and Java 1.8 serving as running platforms. The Spark framework provides a distributed machine learning library known as Spark Machine Learning Library (MLLib),[6] which is used as support for the implementation of our distributed AE network for remotely sensed HSI data analysis. Moreover, the proposed implementation has been coded in Scala 2.11, compiled into Java bytecode and interpreted in JVMs. Finally, mathematical operations from MLLib are handled by Breeze (the numerical processing library for Scala), in its 0.12 version, and by Netlib 1.1.2. In this sense, Netlib wraps JVM calls into low-level Basic Linear Algebra Subprograms (BLAS) calls, and therefore, those calls are executed faster than the traditional executions.

### B. Hyperspectral Data Sets

With the aim of testing the performance of our newly developed cloud-based and distributed AE network model, two different HSI data sets have been considered in our experiments. These data sets correspond to the full version of the AVIRIS Indian Pines scene, referred hereinafter as the big Indian Pines (BIP) scene, and a set of images corresponding to six different zones captured by the Hyperion spectrometer [67] onboard NASA's EO-1 Satellite, which we have designated as the Hyperion data set (HYPERION). Both data sets are characterized by their huge size, which makes them ideal to be processed in a cloud-distributed environment. In the following, we provide a description of the aforementioned data sets.

1) The BIP scene scene (see Fig. 6) was collected by AVIRIS in 1992 [5] over agricultural fields in northwestern Indiana. The image comprises a full flightline with a total of $2678 \times 614$ pixels (with 20 m/pixel spatial resolution), covering 220 spectral bands from 400 to 2500 nm.
2) The Hyperion data set (HYPERION) is composed by six full flightlines (see Fig. 7) collected in 2016 by the Hyperion spectrometer mounted on NASA's EO-1 satellite, which collects spectral signatures using 220 spectral channels ranging from 357 to 2576 nm with a 10-nm bandwidth. The captured scenes have a spatial resolution of 30 m/pixel. The standard scene width and the length are 7.7 and 42 km, respectively, with an optional increased scene length of 185 km. In particular, the considered images have been stacked and treated together as a single image comprising $20\,401 \times 256$ pixels with the spectral range mentioned earlier. These images have been provided by the Earth Resources Observation and Science (EROS) Center in GEOTIFF format.[7] Also, each scene is accompanied by one identifier in the format *YDDDXXXML*, which indicates the day of acquisition (*DDD*), and the sensor that recorded the image (*XXX*, denoting Hyperion, ALI, or AC with $0 =$ OFF and $1 =$ ON), the pointing mode (*M*, which can be *N* for nadir, *P* for pointed within path/row or *K* for pointed outside path/row) and the scene length (*L*, which can be *F* for full scene, *P* for partial scene, *Q* for second partial scene, and *S* for swath). Also, other letters can be used to create distinct entity IDs, for example, to indicate the ground/receiving station (GGG) or the version number (VV). In this case, the identifiers of the six considered images are: 065110KU, 035110KU, 212110KR, 247110KW, 261110KR, and 321110KR.

### C. Experiments and Discussion

Four different experiments have been conducted in order to validate the performance of our cloud-distributed AE for HSI data compression:

1) The first experiment analyzes the scalability of our cloud-distributed AE using a medium-sized data set. For this purpose, the BIP data set has been processed with a fixed number of training samples in the cloud

[3]https://jetstream-cloud.org/

[4]https://www.atmosphereiot.com/platform.html

[5]https://www.openstack.org/

[6]https://spark.apache.org/mllib

[7]These scenes are available online from the Earth Explorer site, https://earthexplorer.usgs.gov

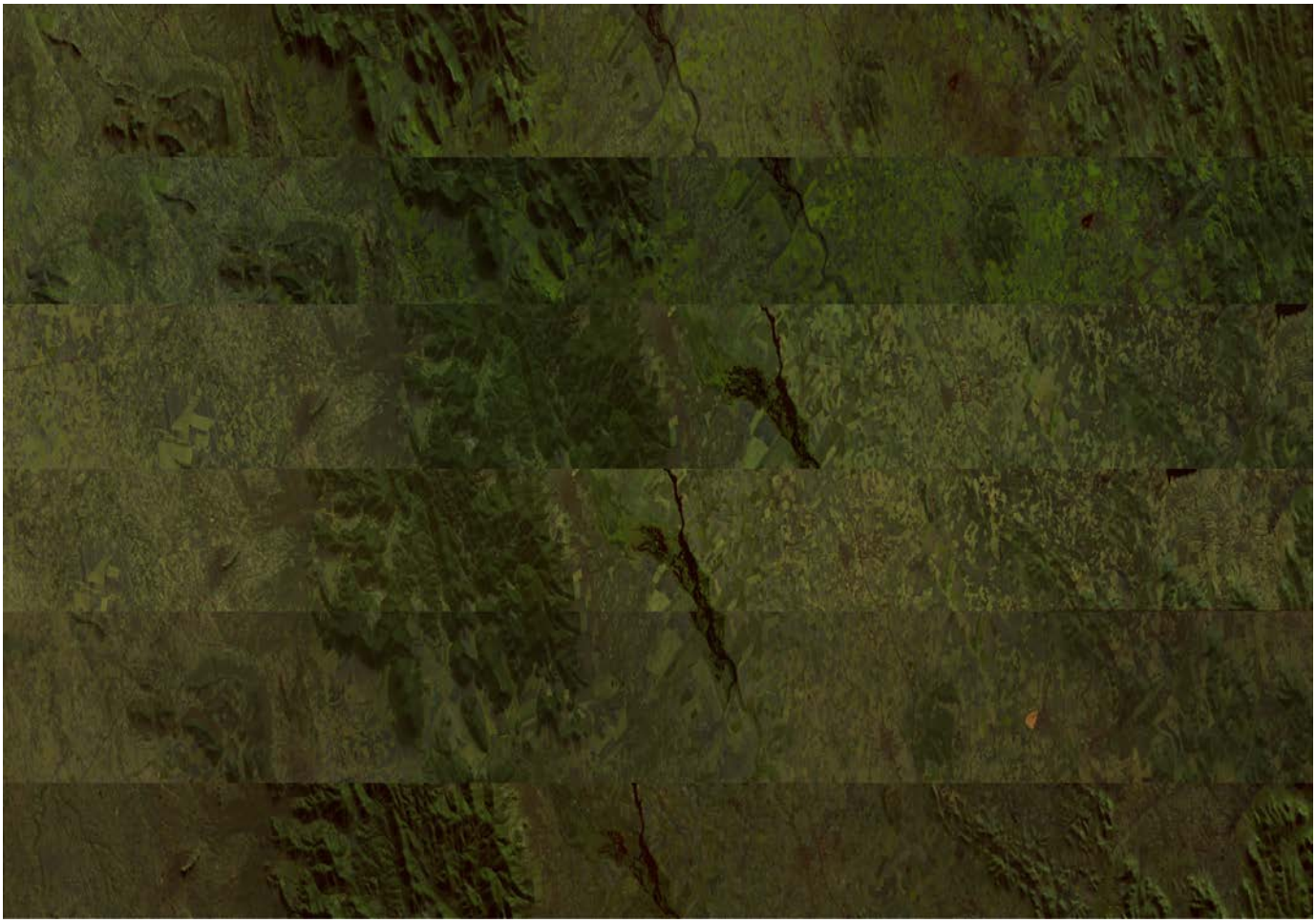


Fig. 7. False RGB color map of the Hyperion data set (HYPERION), represented using the visualization method in [47].

environment described earlier, using one master and different numbers of worker nodes. Here, we have reduced the dimensionality of the BIP data set using the implemented cloud AE model to obtain a reduced data representation with 60 spectral channels.

2) The second experiment illustrates the internal parallelization (at the core level) of the worker nodes. Our main goal is to show that using a fixed amount of workers and increasing only the data volumes, speedups grow linearly as well the computing times. For this purpose, the HYPERION data set has been processed using four different percentages of training data and eight worker nodes in the considered cluster, each with six virtual cores. As in the previous experiment, we reduced the dimensionality of the HYPERION data set using the implemented cloud AE network, extracting 60 spectral channels.
3) The third experiment tests the performance of our cloud-distributed AE using different numbers of training samples and worker nodes over a large data set to illustrate how efficiency grows with the number of workers. This experiment allows us to understand the internal operation of data partitions. In this sense, the HYPERION data set used in the previous experiment has been considered again using four different training percentages and six different numbers of worker nodes.
4) The fourth experiment compares the compression performance of our considered AE implementations (using different activation functions: linear and ReLU) versus the compression performance of a state-of-the-art method, such as PCA. In this experiment, we use a subset of the BIP data that have been widely used in the hyperspectral imaging community, with $145 \times 145$ pixels and 200 spectral bands.

At this point, it is important to emphasize that the earlier experiments have been performed by running 400 training iterations, with the BS set to 256. Also, in order to evaluate the performance of the proposed cloud method, the MSE, the mean absolute error (MAE), and the spectral angle distance (SAD) metrics have been considered.

*1) Experiment 1:* Our first experiment evaluates the performance of the distributed implementation of the proposed AE, using the BIP scene, reduced to 60 spectral channels. In this case, the network employs 80% of randomly selected samples to create the training set and the remaining 20% of the samples to create the test set. In order to demonstrate the scalability of our cloud-distributed AE, the cloud environment has been configured with one master node and different numbers of

TABLE II

AVERAGE RUNTIME AND SPEEDUP WITH THE PROCESSING TIMES AND SPEEDUPS OBTAINED FOR DIFFERENT NUMBERS OF WORKERS WHEN PROCESSING THE BIP IMAGE

| | | | Number of workers | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Training percentage (size) | 1 | 2 | 4 | 8 | 12 | 16 |
| BIP | Time (s) | 80 (2760 MB) | 17398.74 | 8991.12 | 4518.39 | 2354.91 | 1803.27 | 1288.69 |
| | Speedup | 80 (2760 MB) | 1 | 1.9308 | 3.8506 | 7.3882 | 9.6484 | 13.5011 |

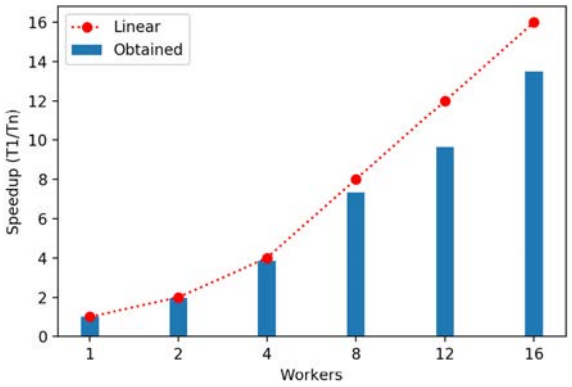



Fig. 8. Scalability of the proposed cloud-distributed network when processing the BIP data set with 1, 2, 4, 8, 12, and 16 worker nodes and 1 master node. Red line: theoretical speedup value. Blue bars: actual values reached.

TABLE III

PERFORMANCE EVALUATION USING THE BIG AND HYPERION DATA SETS (THE FIRST ONE WITH A FIXED NUMBER OF TRAINING SAMPLES AND DIFFERENT NUMBERS OF WORKER NODES, AND THE SECOND ONE WITH DIFFERENT NUMBERS OF TRAINING SAMPLES AND WORKER NODES)

| | | | Implementations | |
|---|---|---|---|---|
| | | Training percentage (size) | Parallel version | Cloud version |
| BIP | MSE | 80 (2760 MB) | 1.461e-4 | 1.590e-4 |
| | MAE | | 7.812e-3 | 8.373e-4 |
| | SAD | | 5.489e-2 | 5.816e-4 |
| | | | Implementations | |
| | | Training percentage (size) | Parallel version | Cloud version |
| HYPERION | MSE | 20 (1838 MB) | 1.05e-4 | 1.23e-4 |
| | | 40 (3676 MB) | 8.01e-5 | 1.24e-4 |
| | | 60 (5515 MB) | 6.79e-5 | 1.26e-4 |
| | | 80 (7353 MB) | N/A | 1.27e-4 |
| | MAE | 20 (1838 MB) | 5.58e-3 | 6.18e-3 |
| | | 40 (3676 MB) | 4.73e-3 | 6.21e-3 |
| | | 60 (5515 MB) | 3.60e-3 | 6.07e-3 |
| | | 80 (7353 MB) | N/A | 6.36e-3 |
| | SAD | 20 (1838 MB) | 11.97e-2 | 11.8e-2 |
| | | 40 (3676 MB) | 11.63e-2 | 11.7e-2 |
| | | 60 (5515 MB) | 12.01e-2 | 11.9e-2 |
| | | 80 (7353 MB) | N/A | 12.0e-2 |

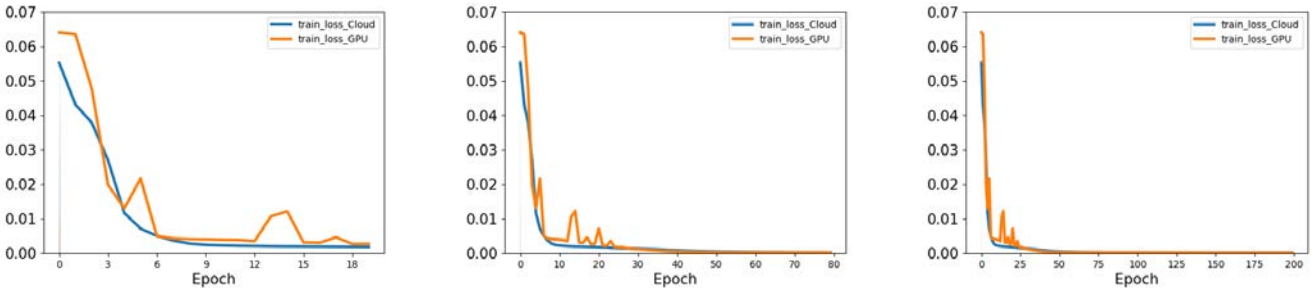
worker nodes, specifically 1, 2, 4, 8, 12, and 16 workers. In order to show the robustness of our model, five Monte Carlo runs have been executed, obtaining as a result the average and the standard deviation of those executions.

Fig. 8 shows the obtained speedup in a graphical way. Such speedup has been calculated as $T_1/T_n$, where $T_1$ is the execution time of the slowest execution with one worker node and $T_n$ is the average time of the executions with $n$ worker nodes. Comparing the theoretical and real speedup values obtained, it can be observed that the model is able to scale very well, reaching a speedup value that is very close to the theoretical one with two, four, and eight workers. However, for 12 workers and beyond, we can see that the communication times between the nodes hamper the speedup due to the insufficient amount of data, a fairly common behavior in cloud environments, in which the main bottleneck occurs in the communication between the nodes. As a result, it is important to make sure that there exists an adequate balance between the total amount of data to be processed and the number of processing nodes. Tables II and III tabulate the performance data collected in this experiment, coupled with the reconstruction errors, computation times, and speedups. In particular, Table III compares the obtained results with a parallel version of the same AE architecture using a standard DL-framework (Torch). As we can observe, the proposed cloud methodology is able to obtain a similar MSE than a traditional DL-framework; however, both the MAE and the SAD measurements are significantly smaller (in particular, the SAD), demonstrating that the proposed implementation is able to improve the performance of a current DL-framework. Also, Fig. 9 shows the evolution of the training loss of the proposed method compared with the one exhibited by the Torch implementation. As we can see in Fig. 9(a), the proposed implementation is more stable than the parallel version, being able to reduce the loss faster than the Torch implementation, until both lines converge on Fig. 9(c), being the train loss of the proposed method slightly lower.

These very low errors are finally reflected in Fig. 10, which shows three reconstructed signatures of different materials in the BIP scene. As it can be seen in Fig. 10, the reconstructed signatures are extremely similar to the original ones, a fact that allows for their exploitation in advanced processing tasks, such as classification or spectral unmixing.

*2) Experiment 2:* Our second experiment explores the internal parallelization of each worker node (at the core level) in order to illustrate that computation and communication times grow in a similar (linear) way. For this purpose, the cloud-distributed AE has been tested on the HYPERION data set, again reducing the spectral dimensionality to 60 spectral bands and randomly collecting 20%, 40%, 60%, and 80% of training samples to create the training set and the remaining 80%, 60%, 40%, and 20% to create the test set. Moreover, one master node and eight worker nodes (each one with six virtual cores) have been considered to implement the cloud environment.

Fig. 11(a) shows the results obtained in this experiment. If we compare the theoretical speedup values and the real ones obtained, it can be seen that our implementation is able to reach a speedup that is almost identical to the theoretical one. This is quite important, as the obtained results indicate that




(a) (b) (c)

Fig. 9. Training loss evolution of the proposed cloud AE compared with a parallel version implemented with the Torch DL-framework on the BIG data set during (a) first 20 epochs, (b) first 80 epochs, and (c) first 200 epochs.

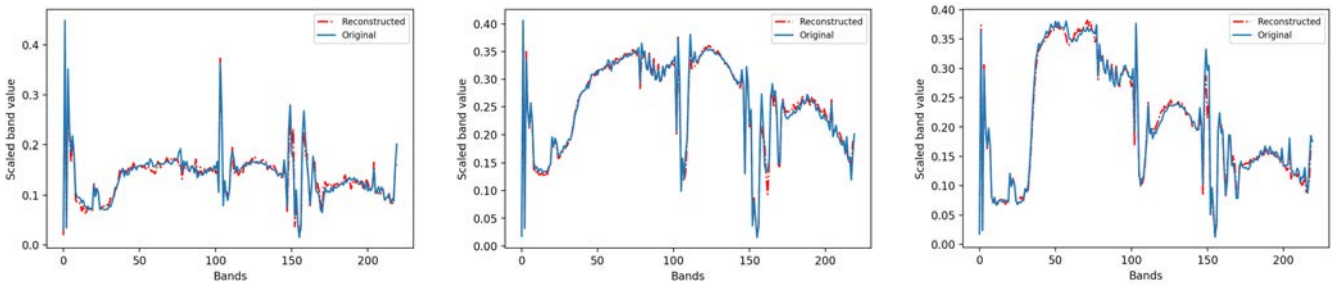
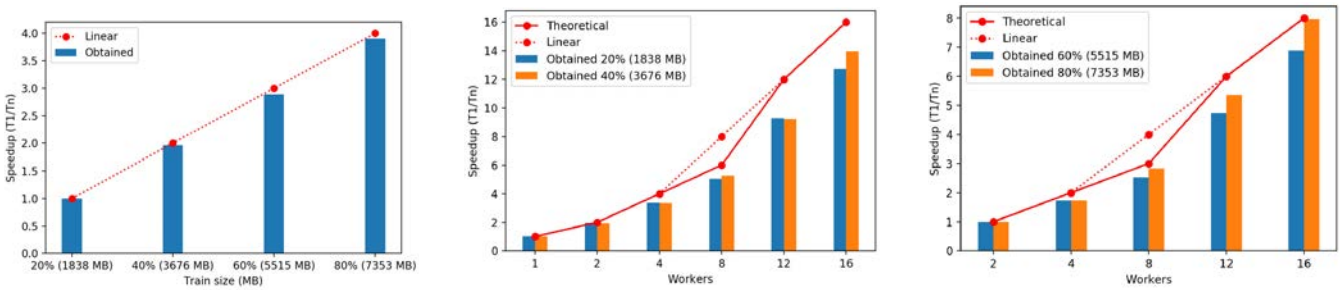



Fig. 10. Comparison between the original (blue line) and reconstructed (red dotted line) spectral signatures extracted from the BIP scene by the proposed cloud AE implementation using eight workers.




(a) (b) (c)

Fig. 11. Scalability of the proposed cloud-distributed network when processing the HYPERION data set in experiments 2 and 3 with (a) 8 worker nodes and 1 master node, considering 20%, 40%, 60%, and 80% of training data (experiment 2), (b) 1, 2, 4, 8, 12, and 16 worker nodes and 1 master node, considering 20% and 40% of training data (experiment 3, first part), and (c) 2, 4, 8, 12, and 16 worker nodes and 1 master node, considering 60% and 80% of training data (experiment 3, second part). The numbers in the parentheses indicate the total amount of data used in MB. Red lines: theoretical speedup value (red continuous line) and linear speedup value (red dotted line). Blue and orange bars: actual values reached.

the scalability achieved in each node (in terms of computing time) is almost linear with regard to the size of the HSI scenes considered in each node, thanks to the cores available in each node. In this way, the proposed cloud-distributed AE implementation takes full advantage of all the available resources, both in parallel (multi-core) and distributed fashion.

*3) Experiment 3:* Our third experiment evaluates how the performance of the proposed cloud AE for HSI data compression is affected by the number of workers available in the cloud environment and the amount of training data using a very large-sized data set. The HYPERION images have been considered for this purpose. Due to the great amount of data, this experiment has been split into two parts. The first part performs a comparison over a cloud environment composed by 1, 2, 4, 8, 12, and 16 worker nodes, and 1 master node, employing the 20% and 40% of the samples to create the training set, and the remaining 80% and 60% of data to create the test set. However, due to the memory limitations of the workers, the second part performs a comparison over a cloud environment composed by 2, 4, 8, 12, and 16 worker nodes, and 1 master node, employing the 60% and 80% of the samples to create the training set, and the remaining 40% and 20% of data to create the test set. In this context, it must be noted that while in the first part the speedup is obtained based on the implementation with one worker node, in the second one, the speedup is obtained based on the implementation with two worker nodes.

Fig. 11(b) and (c) shows the results obtained by the two parts of this experiment in a graphical way. In this case, it is interesting to observe that the theoretical speedup and the

TABLE IV
AVERAGE RUNTIME AND SPEEDUP WITH THE PROCESSING TIMES AND SPEEDUPS OBTAINED FOR DIFFERENT NUMBERS OF WORKERS AND DIFFERENT AMOUNTS OF TRAINING DATA WHEN PROCESSING THE HYPERION DATA SET

| | | | Number of workers | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Training percentage (size) | 1 | 2 | 4 | 8 | 12 | 16 |
| HYPERION | Time (s) | 20 (1838 MB) | 14919.73 | 7632.64 | 4433.11 | 2952.27 | 1606.94 | 1171.87 |
| | | 40 (3676 MB) | 30526.79 | 15709.24 | 9087.66 | 5721.97 | 3311.36 | 2182.60 |
| | | 60 (5515 MB) | N/A | 21505.27 | 12458.54 | 8456.84 | 4536.73 | 3122.92 |
| | | 80 (7353 MB) | N/A | 32645.44 | 18900.49 | 11633.75 | 6084.69 | 4103.02 |
| | Speedup | 20 (1838 MB) | 1 | 1.9547 | 3.3655 | 5.0536 | 9.2845 | 12.7315 |
| | | 40 (3676 MB) | 1 | 1.9432 | 3.3591 | 5.2796 | 9.2187 | 13.9864 |
| | | 60 (5515 MB) | N/A | 1 | 1.7247 | 2.5191 | 4.7402 | 6.8862 |
| | | 80 (7353 MB) | N/A | 1 | 1.7268 | 2.8304 | 5.3651 | 7.9564 |

linear speedup values do not coincide. When we talk about linear speedup, we normally refer to the expected speedup when linear partitioning is performed in the cluster. However, in a real environment, the partitioning is not always linear. In fact, we can observe a performance gap in the eight-node configuration. This can be explained by the relationship between the total number of cores in the cluster, $C$ (obtained as the number of cores per node multiplied by the number of nodes), and the number of existing data partitions, $P$, given by

$$(\lambda - 1) \cdot C < P < \lambda \cdot C \tag{18}$$

where $\lambda$ is a scalar. For instance, when using eight-node configuration, its value is set to $\lambda = 2$. Taking (18) into consideration and assuming that the cluster cores execute tasks when they are free, the noncompliance of (18) leads to the fact that some cores remain idle after finishing their first allocated tasks, so the fine-grained parallelism is not fully exploited in this case.

In the considered cluster, since each node has six cores, a total of $C = 6 \times N$ working cores can be exploited. Furthermore, these $C$ working cores allow for the processing of the data partitions in batches of $C$ tasks at most. For instance, when a configuration of eight nodes is used, the cluster environment is made up of a total of $C = 6 \times 8 = 48$ working cores. This indicates that, at most, in one processing batch, Spark will launch 48 tasks. As Spark splits the HSD data into 58 data partitions, 58 tasks must be executed over each partition. However, in each batch, only 48 tasks can be performed. This means that two batches must be run: the first one with 48 tasks and the second one with only 10. As a result, the second batch cannot fully exploit fine-grained parallelism as only 10 cores are being used, with 38 idle nodes. This results in an unnecessary waste of computing resources.

However, when the idle cores from the second batch are used, the performance improves. This is the case of the 12-node configuration ($C = 72$), where the partitioning becomes more efficient, complying with (18). Linear speedup based on workers needs to be added to this core-level speedup, leading to a new speedup that is calculated as the multiplication of those speedups, as indicated by

$$\frac{T_1^w}{T_n^w} \cdot \frac{T_1^p}{T_n^p}, \tag{19}$$

where $T_n^w$ is the processing time at the worker level and $T_n^p$ the processing time at the core level.

With the aforementioned observations in mind, and focusing on the results given by the first part of the experiment and reported in Fig. 11(b), we can observe that for each configuration and training with 20% and 40% of the available samples, the proposed AE exhibits quite similar speedups, with slight variations due to the distribution of data and the role of idle nodes. It is interesting to observe with 1–8 nodes how the speedup is quite similar to the theoretical one, while with 12–16 nodes, the differences between the obtained and theoretical speedup values are higher, indicating that the proposed AE with only 20% and 40% of training samples does not take full advantage of the cloud environment's potential.

On the other part, Fig. 11(c) shows the obtained results of the second part of this experiment. In this case, the base implementation of the AE is conducted on a cloud environment with two worker nodes, employing 60% and 80% of training data. With 2 and 4 worker nodes, the obtained speedup values are very similar, employing 60% and 80% of the available samples, while with 8, 12, and 16 nodes, it is clear how the version with more training data is able to achieve a superior speedup, reaching a value very similar to the theoretical one with 16 nodes. This indicates that the amount of data handled in this case is more convenient to take full advantage of the way that Spark organizes data partitions and tasks in batches, achieving better parallelization at the core level (fine-grained parallelism) and also better distribution at the worker level (coarse-grained parallelism). These conclusions are supported by the data tabulated in Table IV where the speedup employing 20% and 40% of training data has been obtained taking as base times the cloud environment with one node, while for 60% and 80% of training data, the speedup is obtained comparing with the environment composed by two worker nodes due to the exhausting use of memory. Regarding the compression performance, Table III compares the MSE, MAE, and SAD obtained by the proposed version and a standard implementation using the DL framework Torch. In this case, the MSE is slightly worse than the parallel version, while the MAE and SAD are quite similar. It is interesting to observe that the considered measures keep constant with different amounts of training data, which indicates that the network is

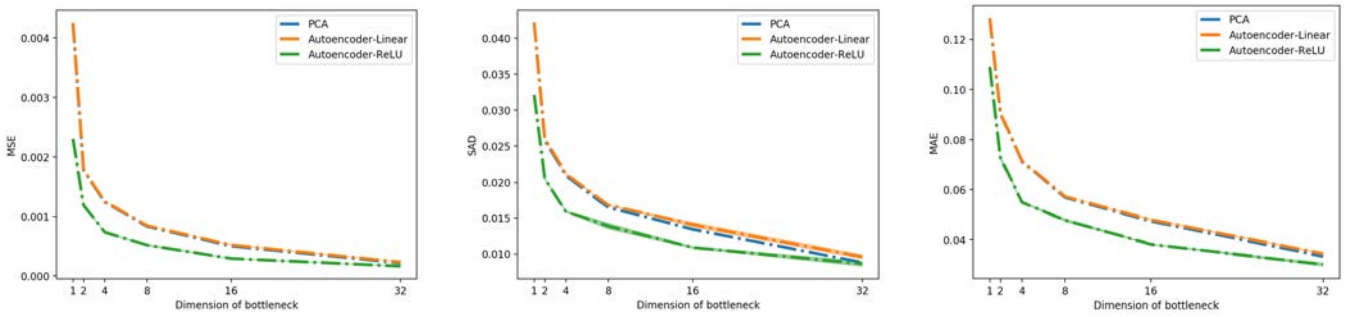



Fig. 12. Obtained compression metrics. (a) MSE, (b) SAD, and (c) MAE for different bottleneck sizes, after using PCA and the proposed AE implementation with different activation functions: linear and non-linear (ReLU). In all cases, we train the models using 75% of the AVIRIS Indian Pines image, which is actually a subset of the BIP image (composed by 145 × 145 pixels with 200 bands).

able to optimize very well, without overfitting the parameters when 60% or 80% of the available training samples are used, but also avoiding underfitting when a few samples are used for training purposes.

*4) Experiment 4:* In order to evaluate the compression performance against another state-of-the-art method, a comparison between the proposed AE implementation and the well-known PCA is given in Fig. 12. Specifically, this experiment reports three different compression metrics (MSE, MAE, and SAD) obtained (for different bottleneck sizes) using the PCA algorithm, the linear AE, and the non-linear AE (ReLU). This experiment has been carried out by training the considered methods with 75% of the small AVIRIS Indian Pines data set (a subset of the BIP image with a size of 145 × 145 pixels and 200 spectral bands). In this sense, with the aim of providing a fair and in-depth comparison between the PCA and the proposed neural-inspired models, we have scaled the input data, introducing batch normalization to the models' layers in order to center the data, as the considered implementation of PCA standardizes the data before the singular vector decomposition (SVD; employing LAPACK solver). Moreover, the considered bottleneck sizes are in the range from 1 to 32, with the steps of $2^n$ and $n = \{0, 1, 2, 3, 4, 5\}$. As we can clearly see in Fig. 12, when reducing the number of dimensions, our proposed implementation (with a linear activation function) performs as well as PCA. However, our proposed implementation (with non-linear activation function) clearly outperforms PCA. This is due to the fact that the ReLU-based AE is able to find non-linear relationships in the data, which assists in the task of reducing the dimensionality of such data and provides a more effective dimensionality reduction strategy than linear methods such as PCA, whose performance clearly decays as we attempt to reduce the data to a small number of dimensions. In this case, the PCA (as a linear method) is not able to find such non-linear correlations between the variables. It is also remarkable that as the target dimensionality grows, the difference between both methods tends to be smaller. From this experiment, we can conclude that the small error obtained with the non-linear AE when reducing the data into a small number of dimensions makes it a highly desirable alternative against linear solutions such as PCA, as it is able to represent the input data better than PCA in a reduced dimension space.

## V. Conclusion

This paper presents a new cloud-based AE neural network for remotely sensed HSI data analysis in a distributed fashion. This kind of artificial neural network finds non-linear solutions when compressing the data, as opposed to traditional techniques such as PCA. In this sense, the proposed approach is more suitable for complex data sets, such as HSIs. The proposed AE implementation over a Spark cluster exhibits a great performance, not only in terms of data compression and error reconstruction but also in terms of scalability when processing huge data volumes, which cannot be achieved by traditional (sequential) AE implementations. Those sequential algorithms may be a valid option when the data to be managed and analyzed can be stored in a single machine with limited processing and memory resources. However, for large amounts of HSI data, sequential implementations can easily run out of memory or require a vast amount of computing time, which cannot be assumed when reliable processing is needed in a reasonable time. In this regard, both HPC and HTC alternatives have provided new paths to solve those problems, including parallelization on GPUs and distribution/parallelization on clusters with cloud computing-based solutions. The experiments carried out in this paper demonstrate that cloud versions of HSI data processing methods provide efficient and effective HPC-HTC alternatives that successfully solve the inherent problems of sequential versions by increasing hardware capabilities in a cheaper way compared with other solutions, such as grid computing. Also, the obtained results reveal that the computation performance of cloud-based solutions easily increases with larger data sets, taking advantage of the computational load distribution when there is a good balance between the amount of data and the cluster complexity. Encouraged by the good results obtained in this paper, in the future, we will develop other implementation of HSI processing techniques in cloud computing environments. Further work will also explore the design of more sophisticated scheduling algorithms in order to circumvent the negative impact introduced by idle processing cores in our current implementation.

## ACKNOWLEDGMENT

The authors would like to thank D. Simmel for his assistance with the cloud environment, which was made possible through the XSEDE Extended Collaborative Support Service (ECSS) Program [65], [66]. They would also like to thank the editors and the anonymous reviewers for their outstanding comments and suggestions, which greatly helped to improve the technical quality and presentation of this paper.

## REFERENCES

[1] W. Emery and A. Camps, *Basic Electromagnetic Concepts and Applications to Optical Sensors*. Amsterdam, The Netherlands: Elsevier, 2017, pp. 43–85.

[2] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for earth remote sensing," *Science*, vol. 228, no. 4704, pp. 1147–1153, 1985.

[3] M. Teke, H. S. Deveci, O. Haliloğlu, S. Zübeyde Gürbüz, and U. Sakarya, "A short survey of hyperspectral remote sensing applications in agriculture," in *Proc. 6th Int. Conf. Recent Adv. Space Technol. (RAST)*, Jun. 2013, pp. 171–176.

[4] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel hyperspectral image and signal processing [applications corner]," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 119–126, May 2011.

[5] R. O. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, Sep. 1998.

[6] K. I. Iteen and *et al.*, "APEX - the hyperspectral ESA airborne prism experiment," *Sensors*, vol. 8, no. 10, pp. 6235–6259, 2008.

[7] C. D. A. K. Stephen Babey, "Compact airborne spectrographic imager (CASI): A progress review," *Proc. SPIE*, vol. 1937, Jul. 1993, Art. no. 193712. doi: 10.1117/12.157052.

[8] S. G. Ungar, J. S. Pearlman, J. A. Mendenhall, and D. Reuter, "Overview of the earth observing one (EO-1) mission," *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 6, pp. 1149–1159, Jun. 2003.

[9] M. J. Barnsley, J. J. Settle, M. A. Cutter, D. R. Lobb, and F. Teston, "The PROBA/CHRIS mission: A low-cost smallsat for hyperspectral multiangle observations of the Earth surface and atmosphere," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 7, pp. 1512–1520, Jul. 2004.

[10] L. Guanter and *et al.*, "The EnMAP spaceborne imaging spectroscopy mission for earth observation," *Senser*, vol. 7, no. 7, pp. 8830–8857, Jul. 2015.

[11] C. Galeazzi, A. Sacchetti, A. Cisbani, and G. Babini, "The PRISMA program," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, vol. 4, Jul. 2008, pp. IV-105–IV-108.

[12] M. A. Wulder, J. G. Masek, W. B. Cohen, T. R. Loveland, and C. E. Woodcock, "Opening the archive: How free data has enabled the science and monitoring promise of Landsat," *Remote Sens. Environ.*, vol. 122, pp. 2–10, Jul. 2012.

[13] J. Aschbacher, *ESA's Earth Observation Strategy Copernicus*. New York, NY, USA: Singapore, 2017, pp. 81–86. doi: 10.1007/978-981-10-3713-9_5.

[14] Y. Ma *et al.*, "Remote sensing big data computing: Challenges and opportunities," *Future Generat. Comput. Syst.*, vol. 51, pp. 47–60, Oct. 2015.

[15] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu, "Big data for remote sensing: Challenges and opportunities," *Proc. IEEE*, vol. 104, no. 11, pp. 2207–2219, Nov. 2016.

[16] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *J. Parallel Distrib. Comput.*, vol. 66, pp. 345–358, Mar. 2006.

[17] D. Gorgan, V. Bacu, T. Stefanut, D. Rodila, and D. Mihon, "Grid based satellite image processing platform for earth observation application development," in *Proc. IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst. Technol. Appl.*, Sep. 2009, pp. 247–252.

[18] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Comput. Environ. Work. (GCE)*, Nov. 2008, pp. 1–10.

[19] Z. Chen, N. Chen, C. Yang, and L. Di, "Cloud computing enabled Web processing service for earth observation data processing," *IEEE J. Sel. Topics Appl. Earth Observat. Remote Sens.*, vol. 5, no. 6, pp. 1637–1649, Dec. 2012.

[20] A. Plaza, J. Plaza, and D. Valencia, "Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data," *J. Supercomput.*, vol. 40, no. 1, pp. 81–107, Apr. 2007.

[21] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, no. 1, pp. S110–S122, Sep. 2009.

[22] G. Hager and G. Wellein, *Introduction to High Performens Computing for Scientists Engineers*. Boca Raton, FL, USA: CRC Press, 2010.

[23] K. Stanoevska-Slabeva, T. Wozniak, and S. Ristol, *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. New York, NY, USA: Springer, 2010.

[24] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesus, J. M. Benítez, and F. Herrera, "Big data with cloud computing: An insight on the computing environment, MapReduce, and programming frameworks," *Wiley Interdiscipl. Rev. Data Mining Knowl. Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[25] A. W. Services, "Overview of Amazon Web services," *Amazon Web Services*, vol. 1, pp. 1–22, Jan. 2017.

[26] *Microsoft Azure Cloud Computing Platform; Services*, Microsoft, Redmond, WA, USA, 2017.

[27] C. Severance, *Using Google App Engine: Start Building Running Web Apps Google's Infrastructure*. Newton, MA, USA: O'Reilly, 2009.

[28] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Design Implement. (OSDI)*, San Francisco, CA, USA, 2004, pp. 137–150.

[29] D. Borthakur, "The Hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.

[30] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[31] J. Haut, M. Paoletti, J. Plaza, and A. Plaza, "Cloud implementation of the K-means algorithm for hyperspectral image analysis," *J. Supercomput.*, vol. 73, no. 1, pp. 514–529, 2017.

[32] V. A. A. Quirita *et al.*, "A new cloud computing architecture for the classification of remote sensing data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 409–416, Feb. 2017.

[33] Y. Zhang *et al.*, "A distributed parallel algorithm based on low-rank and sparse representation for anomaly detection in hyperspectral images," *Sensors*, vol. 18, no. 11, p. 3627, Oct. 2018.

[34] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for parallel on-board hyperspectral image processing," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 4, pp. 424–437, 2008.

[35] A. J. Plaza and C. I. Chang, *High Performance Computing in Remote Sensing*. Boca Raton, FL, USA: CRC Press, 2008.

[36] C. González, S. Sánchez, A. Paz, J. Resano, D. Mozos, and A. Plaza, "Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing," *Integration*, vol. 46, no. 2, pp. 89–103, Mar. 2013.

[37] J. E. Ball, D. T. Anderson, and C. S. Chan, "Comprehensive survey of deep learning in remote sensing: Theories, tools, and challenges for the community," *J. Appl. Remote Sens.*, vol. 11, no. 4, p. 042609, 2017.

[38] S. Kuching, "The performance of maximum likelihood, spectral angle mapper, neural network and decision tree classifiers in hyperspectral image analysis," *J. Comput. Sci.*, vol. 3, no. 6, pp. 419–423, Jul. 2007.

[39] A. Plaza, P. Martínez, J. Plaza, and R. Pérez, "Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations," in *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 466–479, Mar. 2005.

[40] D. Tuia, S. Lopez, M. Schaepman, and J. Chanussot, "Foreword to the special issue on hyperspectral image and signal processing," *IEEE J. Sel. Topics Appl. Earth Observat. Remote Sens.*, vol. 5, no. 2, pp. 347–353, Apr. 2015.

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

[42] F. Del Frate, G. Licciardi, and R. Duca, "Autoassociative neural networks for features reduction of hyperspectral data," in *Proc. 1st Workshop Hyperspectral Image Signal Process. Evol. Remote Sens.*, Aug. 2009, pp. 1–4.

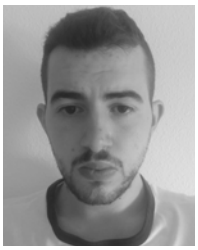
[43] G. A. Licciardi and F. D. Frate, "Pixel unmixing in hyperspectral data by means of neural networks," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 11, pp. 4163–4172, Nov. 2011.

[44] X. Jia, B.-K. Kuo, and M. M. Crawford, "Feature mining for hyperspectral image classification," *Proc. IEEE*, vol. 101, no. 3, pp. 676–697, Mar. 2013.
[45] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton, NJ, USA: Princeton Univ. Press, 2015.
[46] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Trans. Inf. Theory*, vol. 14, no. 1, pp. 55–63, Jan. 1968.
[47] X. Kang, P. Duan, S. Li, and J. A. Benediktsson, "Decolorization-based hyperspectral image visualization," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 8, pp. 4346–4360, Aug. 2018.
[48] X. Kang, P. Duan, X. Xiang, S. Li, and J. A. Benediktsson, "Detection and correction of mislabeled training samples for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 10, pp. 5673–5686, Oct. 2018.
[49] X. Kang, X. Zhang, S. Li, K. Li, J. Li, and J. A. Benediktsson, "Hyperspectral anomaly detection with attribute and edge-preserving filters," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 10, pp. 5600–5611, Oct. 2017.
[50] T.-W. Lee, "Independent component analysis," in *Independent Component Analysis*. New York, NY, USA: Springer, 1998, pp. 27–66.
[51] J. Wang and C.-I. Chang, "Independent component analysis-based dimensionality reduction with applications in hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 44, no. 6, pp. 1586–1600, Jun. 2006.
[52] A. A. Green, M. Berman, P. Switzer, and M. D. Craig, "A transformation for ordering multispectral data in terms of image quality with implications for noise removal," *IEEE Trans. Geosci. Remote Sens.*, vol. 26, no. 1, pp. 65–74, Jan. 1988.
[53] N. He *et al.*, "Feature extraction with multiscale covariance maps for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 2, pp. 755–769, Feb. 2018.
[54] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics Intell. Lab. Syst.*, vol. 2, nos. 1–3, pp. 37–52, 1987.
[55] D. Fernandez, C. Gonzalez, D. Mozos, and S. Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *J. Real-Time Image Process.*, vol. 1, pp. 1–12, Nov. 2016.
[56] E. Martel *et al.*, "Implementation of the principal component analysis onto high-performance computer facilities for hyperspectral dimensionality reduction: Results and comparisons," *Remote Sens.*, vol. 10, no. 6, p. 864, 2018.
[57] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, Jun. 2014.
[58] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[59] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and Helmholtz free energy," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst.*, 1993, pp. 3–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2987189.2987190
[60] M. S. Apostolopoulou, D. G. Sotiropoulos, I. E. Livieris, and P. Pintelas, "A memoryless BFGS neural network training algorithm," in *Proc. 7th IEEE Int. Conf. Ind. Inform.*, Jun. 2009, pp. 216–221.
[61] N. M. Nawi, M. R. Ransing, and R. S. Ransing, "An improved learning algorithm based on the Broyden-fletcher-goldfarb-shanno (BFGS) method for back propagation neural networks," in *Proc. 6th Int. Conf. Intell. Syst. Design Appl.*, vol. 1, Oct. 2006, pp. 152–157.
[62] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Comput. J.*, vol. 6, no. 2, pp. 163–168, Aug. 1963.
[63] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proc. 28th Int. Conf. Int. Conf. Mach. Learn.*, 2011, pp. 265–272.
[64] N. Wilkins-Diehr *et al.*, "An overview of the XSEDE extended collaborative support program," in *Proc. Int. Conf. Supercomput. Mex.*, vol. 595, May 2016, pp. 3–13.
[65] J. Towns *et al.*, "XSEDE: Accelerating scientific discovery," *Comput. Sci. Eng.*, vol. 16, no. 5, pp. 62–74, Sep./Oct. 2014.
[66] D. S. G. Cavallaro, and J. M. Haut, "XSEDE project TG-asc180012: Cloud-based distributed autoencoder for hyperspectral images compression," XSEDE, Tech. Rep., 2019.
[67] J. Pearlman, S. Carman, C. Segal, P. Jarecke, P. Clancy, and W. Browne, "Overview of the hyperion imaging spectrometer for the NASA EO-1 mission," in *Proc. IGARSS*, vol. 7, Jul. 2001, pp. 3036–3038 vol.7.

**Juan Mario Haut** (S'17) received the B.Sc. and M.Sc. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2011 and 2014, respectively, and the Ph.D. degree in information technology from the University of Extremadura through a University Teacher Training Programme from the Spanish Ministry of Education in 2019.

He is currently a member of the Hyperspectral Computing Laboratory, Department of Computer and Communication, University of Extremadura. His research interests include remote sensing and the analysis of very high-spectral resolution with the current focus on machine (deep) learning and cloud computing.

Dr. Haut was a recipient of the recognition of Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2019. He has been a manuscript reviewer of IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, and IEEE GEOSCIENCE AND REMOTE SENSING LETTERS.



**Jose Antonio Gallardo** received the B.Sc. degree in computer engineering from the University of Extremadura, Cáceres, Spain, in 2019, where he has been developing his final degree work at the Hyperspectral Computing Laboratory, Department of Computers and Communications.

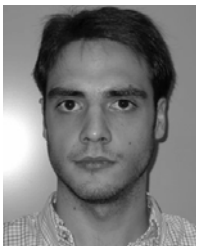
**Mercedes E. Paoletti** (S'17) received the B.Sc. and M.Sc. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2014 and 2016, respectively, where she is currently pursuing the Ph.D. degree through a University Teacher Training Programme from the Spanish Ministry of Education.

She is currently a member of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. Her research interests include remote sensing and analysis of very high spectral resolution with the current focus on deep learning and high-performance computing.

**Gabriele Cavallaro** (S'13–M'17) received the B.S. and M.S. degrees in telecommunications engineering from the University of Trento, Trento, Italy, in 2011 and 2013, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Iceland, Reykjavík, Iceland, in 2016.

He is currently a Post-Doctoral Research Assistant with the Juelich Supercomputing Centre, Jülich, Germany, where he is also a part of a scientific research group focused on high-productivity data processing within the Federated Systems and Data Division. His research interests include remote sensing image processing with parallel machine learning algorithms that scale on high performance and distributed systems.

Dr. Cavallaro was a recipient of the IEEE GRSS Third Prize in the Student Paper Competition of the 2015 IEEE International Geoscience and Remote Sensing Symposium, Milan, Italy, in July 2015.



**Javier Plaza** (M'09–SM'15) received the M.Sc. and Ph.D. degrees in computer engineering from the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain, in 2004 and 2008, respectively.

He is currently a member of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 150 publications, including over 50 JCR journal papers, ten book chapters, and 90 peer-reviewed conference proceeding papers. His main research interests include hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza was a recipient of the Outstanding Ph.D. Dissertation Award at the University of Extremadura in 2008. He was also a recipient of the Best Column Award of the *IEEE Signal Processing Magazine* in 2015 and the Most Highly Cited Paper (2005–2010) in the *Journal of Parallel and Distributed Computing*. He received the best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He has guest edited four special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and IEEE Remote Sensing Code Library. Additional information: http://www.umbc.edu/rssipl/people/jplaza

**Antonio Plaza** (M'05–SM'07–F'15) received the M.Sc. degree and the Ph.D. degree in computer engineering from the Hyperspectral Computing Laboratory, the Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain, in 1999 and 2002, respectively.

He is currently the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 600 publications, including over 200 JCR journal papers (over 160 in IEEE journals), 23 book chapters, and around 300 peer-reviewed conference proceeding papers. His research interests include hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza was a member of the Editorial Board of the IEEE Geoscience and Remote Sensing Newsletter from 2011 to 2012 and the IEEE GEOSCIENCE AND REMOTE SENSING MAGAZINE in 2013. He was also a member of the Steering Committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He is also a fellow of IEEE for contributions to hyperspectral data processing and parallel computing of earth observation data. He was a recipient of the recognition of Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2009 and the Best Reviewer of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, for which he has served as an Associate Editor from 2007 to 2012. He was also a recipient of the Best Column Award of the *IEEE Signal Processing Magazine* in 2015, the 2013 Best Paper Award of the JSTARS journal, and the Most Highly Cited Paper (2005–2010) in the *Journal of Parallel and Distributed Computing*. He received best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He has served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) from 2011 to 2012 and as the President of the Spanish Chapter of IEEE GRSS from 2012 to 2016. He has reviewed more than 500 manuscripts for over 50 different journals. He has served as the Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING from 2013 to 2017. He has guest edited ten special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor of the IEEE ACCESS (receiving the recognition as an Outstanding Associate Editor of the journal in 2017). Additional information: http://www.umbc.edu/rssipl/people/aplaza

**Morris Riedel** (M'10) received the Ph.D. degree from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

He started the work in parallel and distributed systems in the field of scientific visualization and computational steering of e-Science applications on large-scale HPC resources. He held various positions at the Jülich Supercomputing Centre, Jülich, Germany. At this institute, he is also the Head of a specific scientific research group focused on high productivity data processing as a part of the Federated Systems and Data Division. He is currently an Adjunct Associate Professor with the Department of Mechanical Engineering and Computer Science, University of Iceland, Reykjavík, Iceland.