A New Deep Convolutional Neural Network for Fast Hyperspectral Image Classification

M.E. Paoletti, J.M. Haut, J. Plaza, A. Plaza

Hyperspectral Computing Laboratory (HyperComp)
Department of Computer Technology and Communications
Escuela Politecnica de Caceres, University of Extremadura
Avenida de la Universidad s/n, E-10002 Caceres, SPAIN

Abstract

Artificial neural networks (ANNs) have been widely used for the analysis of remotely sensed imagery. In particular, convolutional neural networks (CNNs) are gaining more and more attention in this field. CNNs have proved to be very effective in areas such as image recognition and classification, especially for the classification of large sets composed by two-dimensional images. However, their application to multispectral and hyperspectral images faces some challenges, especially related to the processing of the high-dimensional information contained in multidimensional data cubes. This results in a significant increase in computation time. In this paper, we present a new CNN architecture for the classification of hyperspectral images. The proposed CNN is a 3-D network that uses both spectral and spatial information. It also implements a border mirroring strategy to effectively process border areas in the image, and has been efficiently implemented using graphics processing units (GPUs). Our experimental results indicate that the proposed network performs accurately and efficiently, achieving a reduction of the computation time and increasing the accuracy in the classification of hyperspectral images when compared to other traditional ANN techniques.

Keywords: Hyperspectral imaging, deep learning, convolutional neural networks (CNNs), classification, graphics processing units (GPUs).

1. INTRODUCTION

Remote sensing image acquisition and processing has become very important in recent times in Earth observation problems, exhibiting many practical

applications such as monitoring and management of the environment, agriculture or security and defense/intelligence issues. Of particular importance is the computationally efficient processing of images formed by multiple spectral bands, called multispectral and hyperspectral images. These kinds of images collect information corresponding to large observation areas on the surface of the Earth, using dozens/hundreds of contiguous spectral bands [Chang (2003)], thus creating a three-dimensional data cube with size significantly larger than traditional remotely sensed images. As a result, multispectral and hyperspectral images require particular computational improvements, especially for their storage and advanced processing.

Several methods have been developed for fast processing and classification of multispectral and hyperspectral images [Cheng et al. (2017); Yuan et al. (2015)], from those that only use spatial or spectral information to those that combine both kinds of data. This includes unsupervised techniques such as clustering [Haut et al. (2017a); Tarabalka et al. (2009); Paoletti et al. (2017)]. However, supervised classifiers are often preferred, due to their capacity to provide high classification accuracies, although these methods may be affected by the limited availability of training samples as they generally need a large number of samples in order to obtain those good results. In particular, supervised methods face challenges in the classification of hyperspectral images due to the unbalance between the high dimensionality of the data and the limited number of training samples available in practice (*Hughes phenomenon*) [Khodadadzadeh et al. (2014)]. In this sense, support vector machines (SVMs) [Scholkopf and Smola (2001)] and multinomial logistic regression (MLR) [Böhning (1992)] have been proved to be very useful for the supervised classification of hyperspectral images due to their ability to deal with large input spaces [Melgani and Bruzzone (2004); Fauvel et al. (2008); Plaza et al. (2009); Camps-Valls and Bruzzone (2005); Wu et al. (2015); Haut et al. (2017b)]. Also, some sampling query strategies have been proposed to address the limited availability of training samples, such as semi-supervised and active learning methods [Li et al. (2010, 2011); Rajan et al. (2008)].

At this point, we can highlight several spatial-spectral classification methods that combine the strengths of semi-supervised methods and active learning techniques, for example those based on morphological component analysis (MCA) [Starck et al. (2005)], a method that decomposes images into texture and cartoon (piecewise smooth) parts. In [Zhou and Prasad (2017)], authors presented a new framework that combines active and semi-supervised learning with MCA for hyperspectral image classification. Also, in [Xu et al.

(2016b)] authors presented a new classification framework for the fusion of hyperspectral and light detection and ranging (LiDAR) data, combining MCA for textural feature extraction and MLR for classification purposes. The multiple MCA (MMCA) [Xu et al. (2016a)] is an extension of the MCA that uses both spatial and spectral features. Its goal is to separate an image into two components: a smoothness component and a texture component.

On the other hand, due to their success in the field of pattern recognition [Bishop (1995); Atkinson and Tatnall (1997)] and the availability of multiple training techniques (including machine learning, deep learning and active learning techniques, as well as supervised, unsupervised and semi-supervised approaches) to deal with linearly non-separable data [Benediktsson and Swain (1990)], artificial neural networks (ANNs) have attracted the attention of a large number of researchers in the area of hyperspectral image classification and analysis [Benediktsson et al. (1993); Yang (1999)] as compared to probabilistic methods. In particular, we highlight the use of convolutional neural networks (CNNs) [LeCun et al. (1998a)] as a powerful deep learning model for image classification, which can effectively combine the spatial and spectral information.

Deep learning and CNNs: a review

For years, building a machine learning system required a great deal of effort in designing a feature extractor that would transform raw data (i.e. pixel values from an image) into a feature vector from which the learning subsystem could detect/classify patterns [LeCun et al. (2015)]. Deep learning (or deep structured learning) emerged in 2006 with deep belief networks (DBNs)[1] [Hinton et al. (2006); Hinton and Salakhutdinov (2006)] as a part of a machine learning system that exploits many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and also for pattern analysis and classification [Bengio (2009)].

After DBNs, two new unsupervised deep models were developed: 1) a method for learning sparse, overcomplete features that uses a linear encoder-decoder preceded by a sparsifying non-linearity that turns a code vector into

[1]A DBN is composed by a stack of restricted Boltzmann machines [Smolensky (1986); Larochelle and Bengio (2008)] (RBMs). The DBN core is a greedy learning algorithm that optimizes the network weights layer by layer. Its complexity grows linearly with the size and depth of the network.

a quasi-binary sparse code vector [Ranzato et al. (2006)] and 2) a variant of autoencoder with greedy layer-wise training [Bengio et al. (2007)].

With the advancement of technology (both hardware and software) and the development of new optimization algorithms[2] [LeCun et al. (1998b)] new milestones were achieved in deep learning, giving as result three types of deep methods:

- Unsupervised deep networks (generative learning): these methods work without labeled classes, looking for patterns between pixels through capturing high-order correlation of data (e.g. autoencoder-based methods[3] [Licciardi and Del Frate (2011); Chen et al. (2014)], RBMs [Midhun et al. (2014)], DBNs [Li et al. (2014b)], and deep Boltzmann machines or DBMs [Salakhutdinov and Hinton (2009); Wu et al. (2016)]).

- Supervised deep networks (discriminative deep networks): these models work with labeled information and their goal is to categorize the input data in these labels. They represent the most common form of machine learning, deep or not [LeCun et al. (2015)], and these kinds of models are usually more efficient to train and test, more flexible to construct, and more suitable for end-to-end learning of complex systems [Deng and Yu (2014)]. We can distinguish between linear supervised deep method[4] (e.g. deep neural networks or DNNs[5] with linear activation functions) and non-linear supervised methods (e.g. deep stacking networks or DSNs [He et al. (2016)], recurrent neural networks or RNNs[6]

[2]For example, new variants of gradient descent optimizer were developed, including batch and mini-batch gradient descent, the stochastic gradient descent (SGD) or the included momentum in SGD [Qian (1999)]. New optimizers also appear as Adagrad optimizer [Duchi et al. (2011)] and its extension Adadelta [Zeiler (2012)] or the Adam optimizer [Kingma and Ba (2014)]

[3]An autoencoder [Cho (2014); Karhunen et al. (2015)] is a neural network (or mapping method) where the desired output is equal to the input data vector. We can distinguish between linear autoencoders, with only one hidden layer that works like a principal component analysis (PCA) if its weights between the encoder and decoder are tied, and non-linear or deep autoencoders, which have more modeling power by employing multiple nonlinear intermediate layers symmetrically in the encoder and decoder [Cho (2014)].

[4]Linear classifiers have an important limitation: these methods can only carve their input space into very simple regions (half-spaces) separated by a hyperplane [Chien (1974)].

[5]A DNN is a multi-layer perceptron (MLP) with many hidden and fully connected layers.

[6]RNNs can also be used as generative learning models if the output is not a label

and Convolutional neural networks or CNNs [LeCun et al. (2015)])[7].

- Hybrid deep networks (semisupervised methods): these methods make use of both generative and discriminative model components, i.e., they work with and without labeled data (e.g. generative adversarial networks or GANs [Goodfellow et al. (2014)]). Semi-supervised learning is very useful in hyperspectral image classification in order to deal with the limited training samples problem [Ma et al. (2016)].

Focusing on CNNs, these supervised non-linear models are a special type of deep learning model that is inspired by neuroscience [Ghamisi et al. (2017)] and are designed to process data that come in the form of multiple arrays. The literature on CNN applied to remote sensing classification shows different points of view in the way these models are used. Basically there are three ways to apply CNNs:

1. Extracting only spectral information: spectral-based classification approaches are conceptually simple and easy to be implemented, but they neglect the spatial components [Li et al. (2014a)]. Normally, these methods assume that each pixel is pure and typically labeled as a single land use and land cover type [Fisher (1997)]. For spectral feature classification with CNNs, the spectral feature of the original image data is directly deployed as the input vector [Zhang et al. (2016a)], so we obtain a 1-D CNN architecture that receives $N \times 1$ input vectors, where N is the number of spectral bands [Ghamisi et al. (2017); Hu et al. (2015b); Chen et al. (2016)].
2. Extracting only spatial information: these models consider the neighboring pixels of a certain pixel in the original remote sensing image in order to extract the spatial feature representation [Zhang et al. (2016a)]. As a result, 2-D CNN architectures are adopted, where the input data is a patch of $P \times P$ neighboring pixels [Vetrivel et al. (2017);

sequence associated with the input data sequence. Long short-term memory networks (LSTMs) are a kind of RNN [Hochreiter and Schmidhuber (1997)].

[7]CNNs can also work in unsupervised and semi-supervised mode [Dosovitskiy et al. (2014); Romero et al. (2016); Liu et al. (2017)]. On the other hand, recent efforts have resulted in deconvolutional neural networks (DCNN) [Zeiler et al. (2010)]. In [Lu et al. (2017)] authors combined the spatial pyramid model (SPM) with a shallow weighted DCNN to learn a set of feature maps and filters by minimizing the reconstruction error between the input image and the convolution result.

Chen et al. (2016); Hu et al. (2015a)]. In this sense, several methods have been implemented in order to extract high-level spatial features, as multi-scale image information [Liu et al. (2016); Zhao and Du (2016a); Zhang et al. (2016b); Yu et al. (2017)]. For hyperspectral image analysis normally it is necessary a pre-processing of the spectral information, with reduction of the number of bands for example (using, e.g., PCAs or autoencoders).

3. Extracting spectral-spatial information: the use of spatial features with spectral information in combined fashion can significantly improve the classification accuracy. When we talk about extracting spectral-spatial information with CNNs, two types of models can be highlighted:
   - Those models that mix various techniques in addition to CNNs to extract spectral-spatial information separately and then combine them, for example using a 1-D CNN and 2-D CNN [Zhang et al. (2017); Yang et al. (2016)] or combining different spectral feature extractors with spatial CNNs [Zhao and Du (2016b)]. These methods do not take full advantage of the joint spatial/spectral correlation information.
   - 3-D CNN architectures [Chen et al. (2016); Li et al. (2017)] that compute each pixel in association with a $P \times P$ spatial neighborhood and B spectral bands ($P \times P \times B$). These models apply 3-D kernels in order to learn the local signal changes in both the spatial and the spectral domain of the hyperspectral data cubes, exploiting important discriminative information for classification and taking full advantage of the structural characteristics of the 3-D remote sensing data in general and hyperspectral images in particular [Li et al. (2017)].

In this work, we propose an improved 3-D deep CNN model composed by 5 layers which uses all the spatial-spectral information of the hyperspectral image. We also include a specific strategy for management of the borders of the image and further develop an efficient implementation in graphics processing units (GPUs) to significantly speed up the computational performance. Deep Learning techniques, and in particular CNNs, involve a huge amount of matrix and vector operations. Most of these operations can be easily and massively parallelized using GPUs, due to their inherent design with hundreds/thousands of cores that can compute one or several matrix

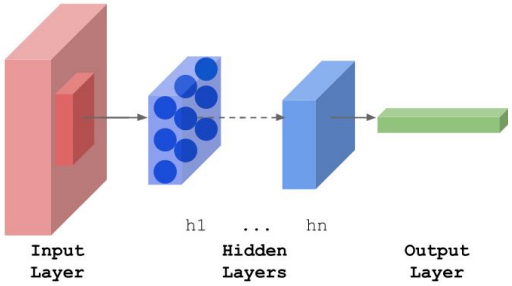


Figure 1: CNN architecture. Each block or layer of a CNN transforms the input volume to an output volume of neuron activations. Neurons in layer l are connected to a small region of layer $l-1$.

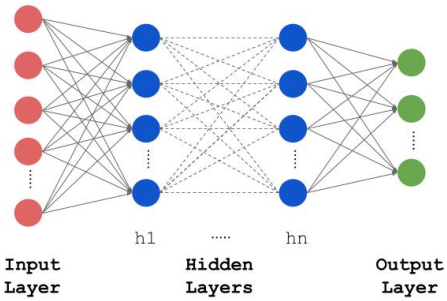


Figure 2: MLP architecture. All nodes in one layer are fully connected with the nodes of the previous layer.

operations in parallel which, compared to a CPU with a few cores, results in a important decrease in computation time.

As a result, the main contributions of our work can be highlighted as follows: 1) the development of a new CNN architecture that considers the spatial and spectral information contained in hyperspectral images in simultaneous fashion, and 2) the development of an efficient implementation of the newly proposed architecture on GPUs that allows for efficient exploitation of the proposed methodology in real applications.

The remainder of the paper is organized as follows. Section 2 provides some general aspects about CNNs. Section 3 describes the proposed CNN. Section 4 validates the proposed approach by comparing it with classic ANNs such as the MLP and other CNN implementations in the literature, in order to illustrate the advantages of the proposed implementation in terms of both computational efficiency and classification accuracy. Section 5 concludes the paper with some remarks and hints at plausible future research lines.

2. CNNs

CNNs are composed by a set of blocks that can be applied both across space and across time (images, audio and video signals). Each block transforms the input volume to an output volume of neuron activations which will serve as input to the next block. In contrast to conventional ANNs, the blocks of neurons in CNNs operate like kernels which are connected and

applied over one patch of the input volume (see Figure 1), that is, the neurons of a block are not fully-connected to all neurons of the previous layer as in the standard MLP (see Figure 2). These blocks actually compose feature extraction stages, which specifically consist of three layers [Zhang et al. (2016a)] that are the key parts of almost all CNN models:

1. Convolution layer: a 3-D layer where each neuron computes the dot product between its weights and a small region of the input volume, i.e. a rectangular section of the previous layer, to which it is connected. Its goal is to identify certain features from the previous layer and mapping their appearance to a feature map [LeCun et al. (2015)]. We can see this layer as set of k filters of size $l \times l \times q$ (filter bank) where the neurons share the same weights and bias and connect the input volume to the output volume [Zhang et al. (2016a)]. Each filter detects a particular feature at every location on the input. The resulting output volume of the layer l is a feature map of size $d^l \times d^l \times k^l$ that stores the information where the feature occurs in the original input volume and is calculated as $z_i^l = B^l + \sum_{j=1}^{k^{l-1}} W_{i,j}^l * z_j^{l-1}$, where $i \in [1, k^l]$, B^l is the bias matrix of layer l and $W_{i,j}^l$ is the weight matrix or filter (also known as kernel or feature detector) [8] that connects the j^{th} feature map in layer $l-1$ (z_j^{l-1}) with the i^{th} feature map in layer l.
2. Nonlinearity layer: this layer embeds a nonlinear function (as the rectified linear unit or ReLU [Jarrett et al. (2009); Nair and Hinton (2010); Glorot et al. (2011)] that is applied to each feature map's component in order to learn nonlinear representations: $a^l = f(z^l)$.
3. Pooling layer: this layer is used to make the features invariant from the location and to summarize the output of multiple neurons in convolution layers through a pooling function. In our case, this layer executes a max operation within a small spatial region R over the resulting feature map after the nonlinearity layer: $p^l = \max_{i \in R} a_i^l$.

Sparse connectivity and shared weights make CNNs ideal for processing and classifying images, reducing the number of parameters to be learned

[8] The concept of weight matrix can be understood as a feature detector or filter which can beused to search for an specific spatial characteristic of the input data. The weight matrix will assign a greater weight to the pixels that collect this characteristic penalizing the pixels that do not exhibit this spatial behaviour.

by the network and ensuring some degree of shift, scale, and distortion invariance.

3. Proposed CNN

The structure of our new CNN is shown in Figure 3. As we can see, our CNN consists of an input layer, three convolution layers with ReLU as nonlinear activation function, two maxpool layers, and four fully-connected layers. The last one is the output layer which obtains the desired label for the input data. In the following, we first provide details about our preprocessing strategy for hyperspectral data, particularly to account for spatial information, prior to feeding this information to the CNN. Then, we provide a detailed explanation about the considered architecture.

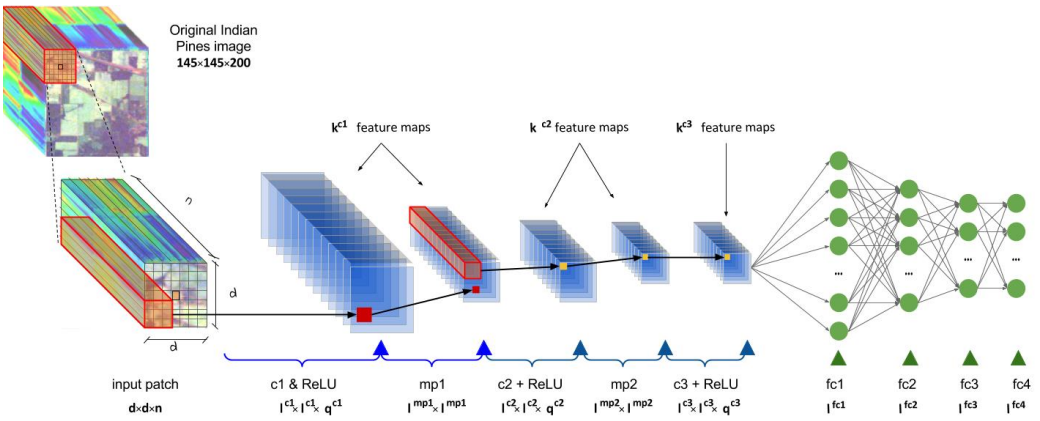


Figure 3: Proposed CNN architecture.

3.1. Data preprocessing

Normally, CNNs receive a complete normalized image prior to classification. However, in hyperspectral images the classes are typically mixed within the image, so we feed the pixel (vectors) one by one to the network. This allows exploiting the rich spectral information contained in the hyperspectral data, but we need an additional mechanism in order to include also the spatial information.

To achieve this and take advantage of both the spatial and the spectral information simultaneously, we have implemented a 3-D approach in which we feed the network with a neighborhood window centered around each pixel

vector. In this way, the input layer accepts volumes of size $d \times d \times n$, where d is the width and height of the input volume and n is the total number of bands of the original hyperspectral image. This requires a pre-processing stage to divide the hyperspectral image into patches of size $d \times d \times n$. The desired label to be reached by the network will be the one that owns the central pixel of the patch $[d/2 + 1, d/2 + 1, n]$.

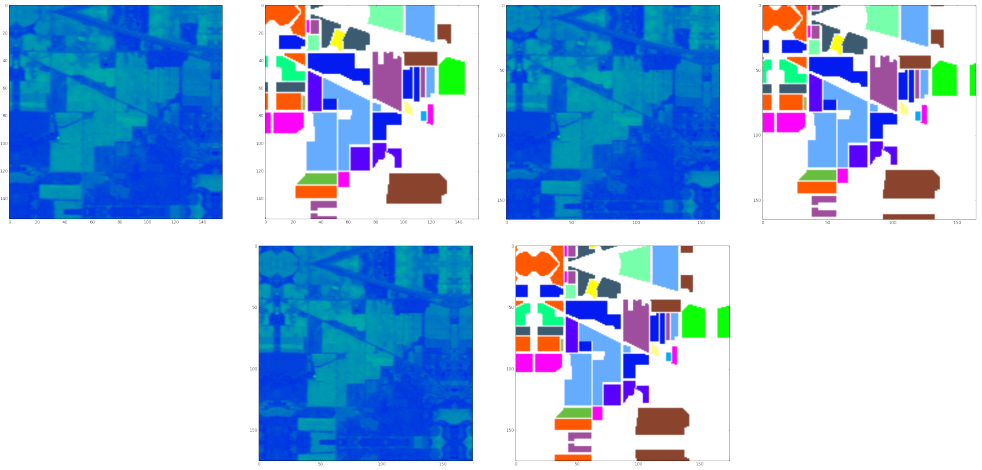


Figure 4: Graphical illustration of our border mirroring strategy (with $d = 9$, $d = 19$ and $d = 29$) using the well-known AVIRIS Indian Pines hyperspectral image (and its associated ground-truth) as an example

However, this preprocessing strategy faces a problem: for the pixels belonging to the borders of the image a $d \times d$ surrounding neighborhood cannot be defined. As we increase d, we cannot properly account for the border information around some pixels. Some approaches simply disregard those border pixels for which they do not have spatial neighbours. This supposes a significant loss of samples that, together with the scarcity of the samples, can make the network result in overfitting. In order to avoid this problem, we have implemented a simple algorithm to replicate borders that allows us to feed the net with all the border pixels and to use them as any other pixel in the image. In this way, we can classify the complete hyperspectral image by replicating the pixels near the border, i.e. mirroring the $d/2$ pixels of border outwards, in order to create the corresponding patches or windows of the original border pixels, as illustrated graphically in Figure 4 using the well-known AVIRIS Indian Pines hyperspectral image (described in detail in

section 4)

3.2. CNN architecture details

Once the edges are replicated and the hyperspectral image is split into 3-D patches, these are grouped in batches of size b and sent to the CNN. Then, the $d \times d \times n$ patches are sent as input volume to the first convolution layer ($c1$), composed by k^{c1} filters of $l^{c1} \times l^{c1} \times q^{c1}$, where $q^{c1} = n$, the stride is fixed to 1, and there is no padding.

After applying the ReLU function, the k^{c1} feature maps generated by $c1$ are sent to the first MaxPool layer ($mp1$), with a $l^{mp1} \times l^{mp1}$ kernel, stride of 2, and padding. The resulting output volume $p^{mp1} = d^{mp1} \times d^{mp1} \times k^{c1}$ is sent to the second convolution layer ($c2$) with k^{c2} filters of size $l^{c2} \times l^{c2} \times q^{c2}$, where $q^{c2} = k^{c1}$, with the same stride as in the first convolution and no padding either.

Again, after applying the ReLU function, the k^{c2} feature maps generated by $c2$ are sent to the second MaxPool layer ($mp2$), with a $l^{mp2} \times l^{mp2}$ kernel, stride of 2 and padding. The resulting output volume $p^{mp2} = d^{mp2} \times d^{mp2} \times k^{c2}$ is sent to the last convolution layer ($c3$), that has k^{c3} filters of size $l^{c3} \times l^{c3} \times q^{c3}$. This layer has the purpose of further refining the feature maps by processing each element one by one, so $k^{c3} = q^{c3} = k^{c2}$ and $l^{c3} = 1$. There is no third maxpool layer, so the output volume is reshaped in order to send it to fully-connected layers.

Four fully-connected layers were implemented ($fc1$, $fc2$, $fc3$ and $fc4$) with l^{fc1}, l^{fc2}, l^{fc3} and l^{fc4} nodes, respectively. The first three fully-connected layers compute their output as $y^{fc} = f(w^{fc} y^{fc-1} + b^{fc})$, where w^{fc} are their weight matrices, b^{fc} are their bias vectors, y^{fc-1} is the output of the previous layer (in the first case, $y^{fc-1} = p^{C3}$, i.e. the output of $C3$ layer) and the activation function $f(\cdot)$ is ReLU. Finally, the last resulting matrix y^{fc3} is sent to $fc4$, which computes the outputs of the network with a softmax function as $y^{fc4} = w^{fc4} y^{fc3} + b^{fc4}$, where y^{fc4} contains the desired labels for the original $d \times d \times n$ input data.

In Algorithm 1 we can see a scheme of the operation of the proposed method. As we can notice, the method uses *cross-entropy* in order to determine the *loss* of the CNN model. It is defined as $H_{y'}(y) = \sum_i y'_i log(y_i)$, where y is our predicted probability distribution and y' is the true distribution, so the cross-entropy is a measure of how inefficiently predictions are calculated for describing the truth.

Algorithm 1 Proposed CNN method

procedure CNN_METHOD($Y \rightarrow$ original hyperspectral image)
  $max\_epochs \rightarrow$ Set number of epochs value
  $max\_iters \rightarrow$ Set number of iterations value
  $d \rightarrow$ Set patch size value
  $n = Y.bands$
  $Y_{norm} = band\_mean\_normalize(Y)$
  $Y' = border\_mirroring(Y_{norm}, d) \rightarrow$ mirroring of d border pixels
  $P = patches\_creation(Y', d, n) \rightarrow$ splitting Y' into patches of $d \times d \times n$
  $T_a, T_b = training\_test\_sets(P) \rightarrow$ training T_a and testing T_b sets
  $G = batches\_creation(T_a, b) \rightarrow$ grouping patches in batches of size b
  for $e < max\_epochs$ **do**
    for $it < max\_iters$ **do**
      $G' = get\_next\_batch(G)$
      $labels\_G'\_predicted = forward\_pass(G')$
      $error = cross\_entropy(labels\_G', labels\_G'\_predicted)$
      $W, B = optimizer(error)$
    end for
  end for
end procedure

As an interesting point we note that, for the initialization of all weights and bias of the network, we have used the so-called Xavier initializer [Glorot and Bengio (2010)], that allows the network to achieve greater stability, and the Adagrad optimizer [Duchi et al. (2011)], as a simple method for learning rate adaptation. All the network characteristics (weights and bias initialization, optimizer, learning rate, steps, kernels size, use of padding and size of strides) are configurable through a JavaScript Object Notation (JSON)[9] file, as well as parameters d and b, which makes the implementation of the network flexible and easy to modify. Such implementation has been performed in Python[10]. In this way, the CNN takes advantage of the information of the central pixel neighbors as well as all the available spectral information, being able to adapt its structure easily and quickly.

As we anticipated in subsection 3.1, this kind of networks may suffer from

[9] http://www.json.org/
[10] https://www.python.org/

an overfitting problem because of the complexity of the model derived from the large number of parameters that must be learned, and a lack of training samples (that is quite common in remote sensing applications). This problem may result in poor predictive performance in the testing phase, despite a high accuracy can be obtained in the training phase. To avoid such overfitting problems, we have allowed an optional and configurable dropout mechanism in the first and second convolution layers. The dropout method sets the output of some randomly selected hidden neurons to zero, so that the dropped neurons do not contribute in the forward pass and they are not used in the back-propagation stage [Hinton et al. (2012)].

To conclude this section, it is important to note that the proposed CNN has been implemented using the TensorFlow open source library for machine intelligence[11] including its GPU functionalities, that allow for fast performance even when dealing with very large hyperspectral image volumes. In the following section, we evaluate the proposed CNN from the viewpoint of both computational performance and classification accuracy.

4. Experiments and results

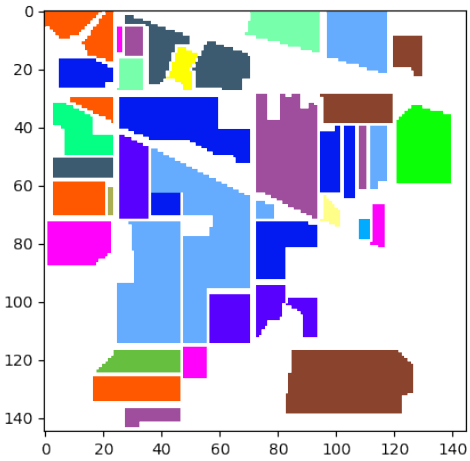
4.1. Experimental Configuration

In order to evaluate the performance of our newly presented CNN architecture, we use a hardware environment composed by a 6th Generation Intel® Core™i7-6700K processor with 8M of Cache and up to 4.20GHz (4 cores/8 way multitask processing), 40GB of DDR4 RAM with a serial speed of 2400MHz, a GPU NVIDIA GeForce GTX 1080 with 8GB GDDR5X of video memory and 10Gbps of memory frequency, a Toshiba DT01ACA HDD with 7200RPM and 2TB of capacity, and an ASUS Z170 pro-gaming motherboard.

4.2. Hyperspectral Image Data

In our experiments, we have used two well-known hyperspectral image data sets. The first one is the Indian Pines image, gathered in 1992 by the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) sensor [Green et al. (1998)] over a set of agricultural fields with regular geometry and with a multiple crops and irregular patches of forest in Northwestern Indiana.

[11] https://www.tensorflow.org

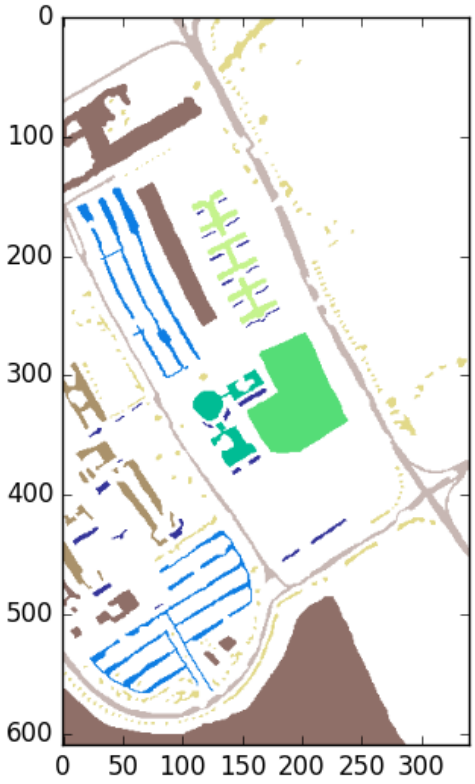


Color	Land cover type	Samples
	Background	10776
	Alfalfa	46
	Corn-notill	1428
	Corn-min	830
	Corn	237
	Grass/Pasture	483
	Grass/Trees	730
	Grass/Pasture-mowed	28
	Hay-windrowed	478
	Oats	20
	Soybeans-notill	972
	Soybeans-min	2455
	Soybeans-clean	693
	Wheat	205
	Woods	1265
	Bldg-Grass-Tree-Drives	386
	Stone-steel towers	93
Total samples		21025

Figure 5: Original ground-truth image of the AVIRIS Indian Pines scene and number of samples per class.

The AVIRIS Indian Pines scene has 145x145 pixels with 224 spectral bands in the range from 400 to 2500nm, with 10nm of spectral resolution, 20m moderate spatial resolution, and 16 bits radiometric resolution. After an initial analysis, 4 zero bands and another 20 bands with lower signal-to-noise ratio (SNR) have been removed because of atmospheric absorption phenomena in those bands, retaining only 200 spectral channels. Moreover, about half of the pixels in the hyperspectral image (10249 of 21025, i.e. 48.74%) contain ground-truth information, which comes in the form of a single label assignment for each pixel with a total of 16 ground-truth classes (see Figure 5).

The second data set used in experiments was collected by the Reflective Optics System Imaging Spectrometer (ROSIS) sensor [Kunkel et al. (1988)] during a flight campaign over the city of Pavia, in northern Italy. The dataset covers an urban environment, with various solid structures (asphalt, gravel, metal sheets, bitumen, bricks), natural objects (trees, meadows, soil), and shadows (9 classes in total). Other objects whose compositions differ from the labeled ones are considered as clutter. The scene was collected over an university area. It contains 103 spectral bands with 610×340 pixels in the spectral range from 0.43 to 0.86μm, and spatial resolution of 1.3m/pixel. About 20.62% of the pixels in the hyperspectral image (42776 of 207400)



Color	Land cover type	Samples
	Background	164624
	Asphalt	6631
	Meadows	18649
	Gravel	2099
	Trees	3064
	Painted metal sheets	1345
	Bare Soil	5029
	Bitumen	1330
	Self-Blocking Bricks	3682
	Shadows	947
	Total samples	207400

Figure 6: Original ground-truth image of the ROSIS University of Pavia scene and number of samples per class.

contain ground-truth information (see Figure 6).

4.2.1. Data preprocessing: division of training and testing sets

When our method divides the hyperspectral images into training and testing sets, it follows a method of preprocessing with class balancing that must be conveniently explained at this point. In addition to their huge dimensionality and the existing correlation between the spectral features collected [Melgani and Bruzzone (2004)], hyperspectral images present another complication: the class imbalance problem [He and Garcia (2009)]. This problem appears in a dataset when some of the classes are heavily under-represented (in terms of their labeled samples) as compared to other classes [García et al. (2011)], leading to poor classification performance in many real-world applications, especially for the minority classes. In order to deal with this problem, and taking into account that we could not identify a common pattern about

sample selection strategies in the literature, we have tried to keep a stratified sampling strategy in our experiments. As a result, we tried to balance the number of samples selected in accordance with the number of available samples per class.

Indian Pines

clase	pixels	200 samples per class	100 samples per class	50 samples per class	Chen et al. (2016)
Alfalfa	46	33	33	33	30
Corn-notill	1428	200	100	50	150
Corn-min	830	200	100	50	150
Corn	237	181	100	50	100
Grass/pasture	483	200	100	50	150
Grass/trees	730	200	100	50	150
Grass/pasture-mowed	28	20	20	20	20
Hay-windrowed	478	200	100	50	150
Oats	20	14	14	14	15
Soybeans-notill	972	200	100	50	150
Soybeans-min	2455	200	100	50	150
Soybeans-clean	593	200	100	50	150
Wheat	205	143	100	50	150
Woods	1265	200	100	50	150
Bldg-grass-tree-drives	386	200	100	50	50
Stone-steel towers	93	75	75	50	50
Total	10249	2466	1342	717	1765

Table 1: Indian Pines: number of samples in the training set used by the proposed method and by the method in [Chen et al. (2016)].

The first step is to divide randomly the original dataset in two subsets: the first one (training set) with 75%[12] of the samples and the second one (testing set) with the remaining 25%. If we keep that 75% of samples for the training set, we will once again experience the class imbalance problem. For instance, in the Indian Pines image the Alfalfa class has only 46 samples as opposed to Corn-notill, which has 1428. 75% of each is $34,5 \approx 36$ for Alfalfa and 1071 for Corn-notill, a completely unbalanced result. The solution adopted in this case is to simply reduce the number samples until a balanced result is achieved. After we get 75% sampling of each class, we set a maximum number of samples per class (as a threshold), for example 50, 100 or 200 samples per class. For those classes with many samples, we simply cut the samples until reaching the threshold. However, for those classes that have very few samples and do not reach the threshold, only those available pixels are taken. In Table 1 we can observe the real number of samples that we

[12]For each class we make sure that about 75% of each class is taken, until we have a number of samples close to 75% of the complete dataset, so we make sure that all classes are represented in the subset.

are using in each experiment when we refer to "50 samples per class", "100 samples per class" or "200 samples per class". Except for those classes that do not reach the proposed threshold (and that use 75% complete), the rest of classes work with 15-25% of their samples. In Table 2 we can see that the same solution is adopted for the University of Pavia data. In both cases, the proposed method uses less samples than [Chen et al. (2016)] with the exception of Indian Pines with 200 samples per class.

Pavia University

class	pixels	200 samples per class	100 samples per class	50 samples per class	Chen et al. (2016)
Asphalt	6631	200	100	50	548
Meadows	18649	200	100	50	540
Gravel	2099	200	100	50	392
Trees	3064	200	100	50	542
Painted metal sheets	1345	200	100	50	256
Bare soil	5029	200	100	50	532
Bitumen	1330	200	100	50	375
Self-blocking bricks	3682	200	100	50	514
Shadows	947	200	100	50	231
Total	42776	1800	900	450	3930

Table 2: University of Pavia: number of samples in the training set used by the proposed method and by the method in [Chen et al. (2016)].

4.3. Hyperparameter tuning

The first step to carry out the experiments has been to adjust the configuration parameters of the convolutional network to get the best possible classification accuracy in the considered hyperspectral datasets, through cross-validation.

For the Indian Pines dataset, the CNN configuration parameters have been adjusted according to Table 3. As we can see in the convolution layers, the noise of some of the Indian Pines image bands is mitigated by a first expansion of depth, and the overfitting problem is solved by adding dropout in the first and second convolution layers and in the first fully connected layer[13]. The third convolution layer refines the $c2$'s feature maps with the objective of obtaining a better classification.

For the University of Pavia dataset, the CNN configuration parameters have been adjusted according to Table 3. Also, we improved the quality of

[13]After the first experiment, we realized that these values were insufficient and the network was fine-tuned again. A 10% dropout was added to the third convolution layer and a 30% dropout was added to the first fully-connected layer to avoid overfitting.

the spectral information by extending the depth of the feature maps in the first convolution. In this case, the overfitting problem is worse than in the Indian Pines scene (mainly due to the greater number of parameters to be learned), so we increased the value of the dropout in the first and second convolution layers. Again, the third convolution layer refines the $c2$'s feature maps, with the objective of obtaining a better classification.

CNN proposed topologies

Hyperspectral datasets	**convolution layers**				**Fully Connected Layers**		
	Kernel size $k^c \times l^c \times l^c \times q^c$	*ReLU*	*Pooling* $l^{mp} \times l^{mp}$	*Dropout*	*Nº neurons* l^{fc}	*Function*	*Dropout*
Indian Pines	$600 \times 5 \times 5 \times 200$	Yes	2×2	Yes (10%)	1024	ReLU	Yes(10%)
	$200 \times 3 \times 3 \times 600$	Yes	2×2	Yes (10%)	1024	ReLU	No
	$200 \times 1 \times 1 \times 200$	Yes	No	No	512	ReLU	No
					16	Softmax	No
Pavia University	$380 \times 7 \times 7 \times 103$	Yes	2×2	Yes(20%)	2048	ReLU	No
	$350 \times 5 \times 5 \times 380$	Yes	2×2	Yes(20%)	2048	ReLU	No
	$350 \times 1 \times 1 \times 350$	Yes	No	No	1024	ReLU	No
					9	Softmax	No

Common parameters

Bach size b	**Steps (iterations)**	**Epochs**	**Learning rate**	**Optimizer**
100	1500	20	0.01	AdagradOptimizer

Table 3: Configuration of the CNN architecture for the Indian Pines and University of Pavia datasets. The kernel size (like the number and type of layers, strides and padding) is one of the design choices of the proposed CNN architecture. Large kernels allow our CNN to learn more complex features, although with larger kernels the computational time of the training/testing phase is also greater.

4.4. Performance evaluation

To test the proposed CNN for hyperspectral image classification with the configurations described in section 4.3, several experiments have been conducted, first with the Indian Pines hyperspectral dataset and, second, with the University of Pavia hyperspectral dataset. At this point, we emphasize that data pre-processing plays a very important role in this kind of deep learning algorithms. In practice, many classification methods work better after a data normalization procedure. In this case hyperspectral datasets have been scaled between in the range $[-0.5, 0.5]$ and a band-mean normalized procedure has been performed. This means that each of the spectral channels in the image have been normalized by subtracting the mean.

Testing parameter d: We tested different sizes of parameter d, using a fixed number of 100 samples per class. For the Indian Pines data, we have considered three patch sizes: $d = 9$, $d = 19$ and $d = 29$. In Figure 7 we illustrate the effect of using different patch sizes over a random pixel in the

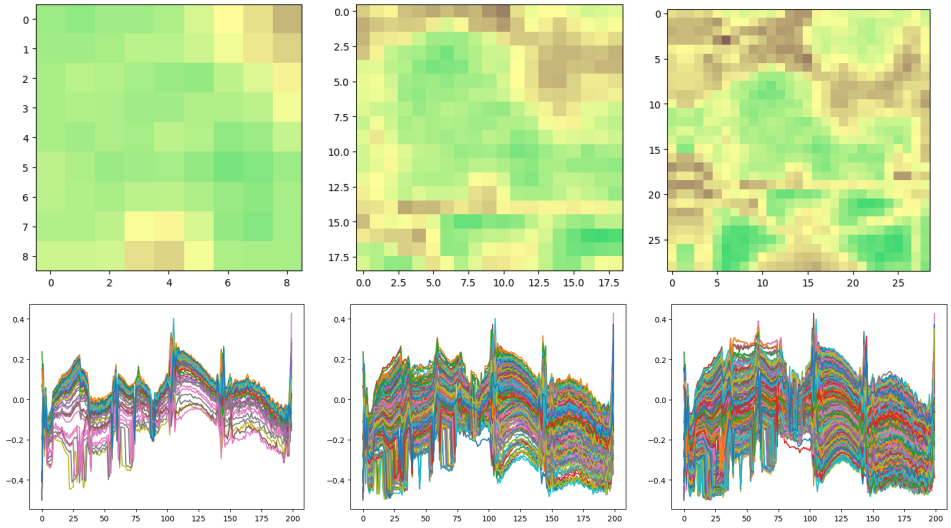


Figure 7: Illustration of the effect of using different patch sizes for the Indian Pines scene. The upper row shows the size of the neighborhood around the central pixel when we use $d = 9$, $d = 19$ and $d = 29$, as we can see on the axes from left to right (color map of band 140). In the lower row we can see the spectral signature of the pixels that form each neighborhood, respectively.

Indian Pines dataset. As we add pixels to the neighborhood, the spatial information around the considered pixel is more clear, especially when it comes to edge pixels. However, a patch too large can detract from the target pixel. As for the spectral signature, adding more neighbors to the target pixel also makes the signature as a whole more defined.

For the Pavia University data, we have tested other patch sizes: $d = 15$, $d = 21$ and $d = 27$. The difference is motivated by our pre-assessment of the size of relevant features in the image. We also provide in Figure 8 an illustrative example of how the size of patches impacts the overall performance in the University of Pavia data set. In this case, the scene presents many object borders in the leftmost part. By adding more spatial information we can better identify the pixels belonging to such edges. However, given the reduced number of classes, if we add too many spectral signatures we can make the patches slightly homogeneous.

For each d, we divided the original hyperspectral image into pieces (mirroring the borders of the image, if necessary) and grouping the patches into training samples and test samples. To split the patches, we followed the

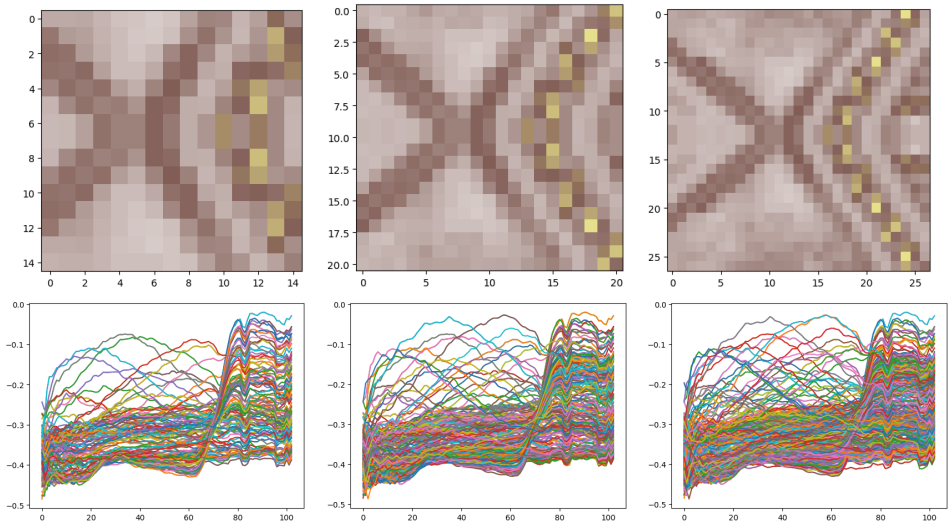


Figure 8: Illustration of the effect of using different patch sizes for the University of Pavia scene. The upper row shows the size of the neighborhood around the central pixel when we use $d = 15$, $d = 21$ and $d = 27$, as we can see on the axes from left to right (color map of band 10). In the lower row we can see the spectral signature of the pixels that form each neighborhood, respectively.

steps described in section 4.2.1. Each execution of this experiment has been repeated 5 times.

Dataset	**Patch size**	**Time per step**		**Total time**		**Accuracy**	
		Avg.	Std. dev.	Avg.	Std. dev.	Avg.	Std. dev.
Indian Pines	$d = 9$	0.02	0.01	29.22	1.05	78.46	4.45
	$d = 19$	0.08	0.02	116.30	3.22	91.34	1.59
	$d = 29$	0.16	0.03	248.29	7.91	95.53	0.48
University of Pavia	$d = 15$	0.02	0.01	34.78	1.13	94.02	0.62
	$d = 21$	0.05	0.02	74.66	2.22	95.08	1.41
	$d = 27$	0.09	0.02	131.66	2.88	94.13	0.66

Table 4: Execution times (in seconds) and accuracies measured with patches of size $d = 9$, $d = 19$ and $d = 29$ for the Indian Pines dataset and $d = 15$, $d = 21$ and $d = 27$ for the University of Pavia dataset. The CNN configuration is the one indicated in Table 3, with 1500 iterations, 100 samples per class and 5 executions.

Table 4 reports the obtained results. We can observe that, for the Indian Pines image, $d = 29$ achieves the best result, reaching an overall accuracy

of 95.53% with a smaller error in fewer iterations (with only 400 iterations, $d = 29$ has already reached a minimum error, while for $d = 19$ it needs about 1000 iterations to reach the same error and $d = 9$ is not able to reduce its error in 1500 iterations, see Figure 9). However, the time required for each step when $d = 29$ is adopted is greater than with $d = 19$ or $d = 9$: each step of $d = 29$ is 0.08 seconds slower than the steps of $d = 19$ and 0.14 seconds slower than the steps of $d = 9$ (see Figure 10). In terms of the accuracy/time ratio, the best option is $d = 19$, although the best accuracy is achieved by using $d = 29$ (but its execution time is larger).

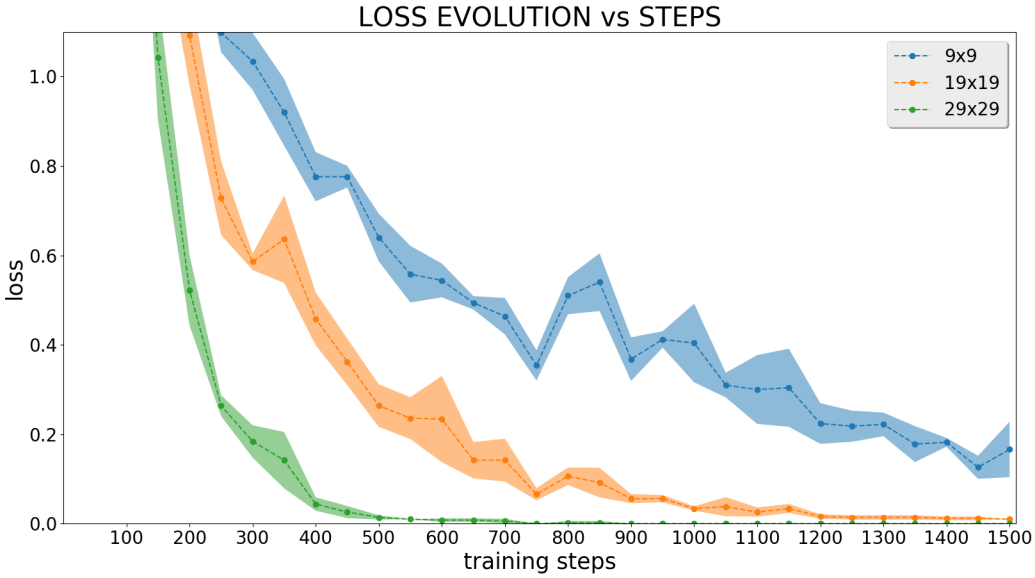


Figure 9: Indian Pines: Evolution of the validation loss in terms of steps (iterations) with $d = 9$, $d = 19$ and $d = 29$. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 1500 iterations.

On the other hand, the results obtained for the University of Pavia dataset are also shown in Table 4. In this case, the patch with size $d = 21$ achieves the best accuracy results: 95.08% in 74.66 seconds, reaching 1.06 percentage more than $d = 15$ and 0.95 percentage more than $d = 27$. As for the number of iterations, in Figure 11 we can see that $d = 27$ needs less iterations to reach an acceptable error (between 700-800 iterations) while $d = 21$ needs around 1000-1100 iterations, which is very similar to $d = 15$ that also needs around 1000-1100 iterations to reach a low error. On the other hand, the

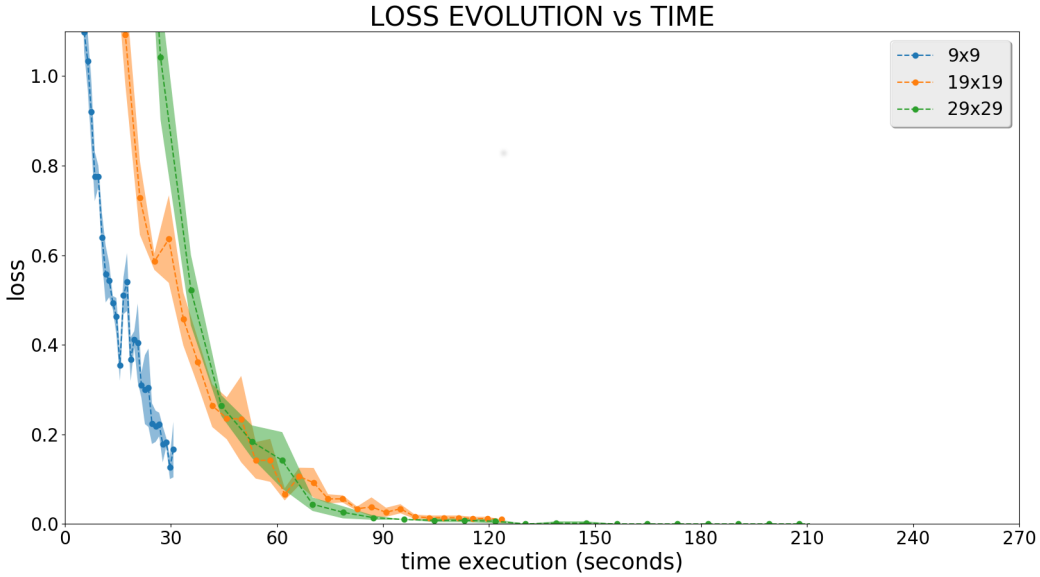


Figure 10: Indian Pines: Evolution of the validation loss in terms of time (seconds) with $d = 9$, $d = 19$ and $d = 29$. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 1500 iterations.

time per iteration for each patch size is different, being the fastest $d = 15$ (2.15 times faster than $d = 21$ and 3.79 times faster than $d = 27$) and the slowest $d = 27$ (around 1.76 times slower than $d = 21$), see Figure 12. With this information at hand, we can conclude that the best patch size is $d = 21$, as it reaches the best result in a fairly reasonable time.

Testing the number of samples per class: At this point, we tested the accuracy achieved for different patch sizes ($d = 9$, $d = 19$ and $d = 29$ for Indian Pines and $d = 15$, $d = 21$ and $d = 27$ for Pavia) with different amounts of training data, in particular with 50, 100 and 200 samples per class.

The results obtained for the Indian Pines dataset are shown in Table 5 (with standard deviation). In this case, for each experiment the parameters of the CNN have been fine tuned, in order to achieve the best possible accuracy. As we can observe in Figures 9 and 10, the configuration of the CNN in Table 3 for Indian Pines presents a marked overfitting problem (that the standard deviation also seems to indicate). As a result, the dropout percentages have been modified for the Indian Pines experiment, specifically we

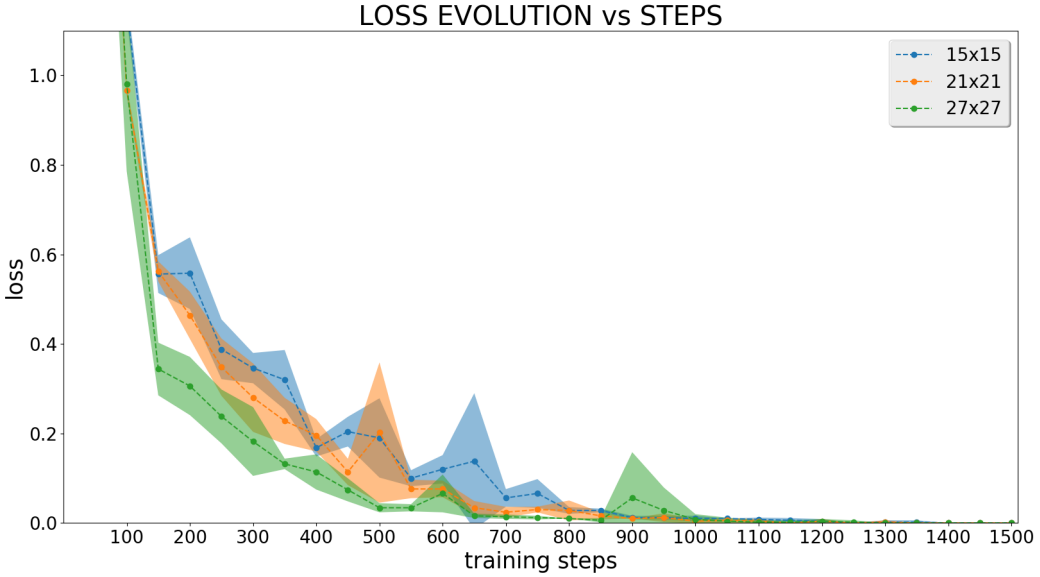


Figure 11: Pavia University: Evolution of the validation loss in terms of steps (iterations) with $d = 15$, $d = 21$ and $d = 27$. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 1500 iterations.

Neural networks	CNN $d = 9$			CNN $d = 19$			CNN $d = 29$		
Samples per class	50	100	200	50	100	200	50	100	200
Alfalfa	98.70 (1.06)	99.13 (1.06)	99.13 (1.06)	99.57 (0.87)	100.00 (0.00)	99.57 (0.87)	99.13 (1.74)	99.57 (0.87)	99.13 (1.06)
Corn-notill	70.76 (3.42)	76.93 (2.50)	80.48 (9.37)	76.74 (3.89)	85.84 (1.92)	94.47 (2.46)	82.10 (3.94)	91.32 (0.46)	98.17 (0.67)
Corn-min	78.92 (5.37)	90.55 (1.97)	96.65 (1.20)	82.77 (2.61)	93.23 (2.09)	98.22 (0.87)	86.41 (4.29)	94.84 (0.47)	98.92 (0.68)
Corn	96.54 (1.89)	99.75 (0.21)	99.66 (0.17)	99.41 (0.83)	99.66 (0.32)	100.00 (0.00)	97.81 (2.92)	100.00 (0.00)	100.00 (0.00)
Grass/Pasture	89.86 (4.56)	97.81 (0.81)	99.46 (0.52)	95.20 (1.54)	98.18 (0.73)	99.75 (0.20)	96.15 (3.75)	98.34 (1.23)	99.71 (0.21)
Grass/Trees	97.40 (0.94)	98.11 (0.88)	99.53 (0.32)	92.96 (2.64)	97.92 (1.31)	98.90 (0.43)	96.47 (2.43)	98.66 (0.94)	99.40 (0.52)
Grass/pasture-mowed	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Hay-windrowed	99.29 (0.34)	99.62 (0.08)	99.67 (0.21)	99.12 (1.06)	99.08 (0.45)	99.62 (0.47)	99.62 (0.50)	99.96 (0.08)	100.00 (0.00)
Oats	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Soybeans-notill	80.39 (3.52)	87.86 (1.95)	92.43 (2.03)	88.35 (4.26)	94.81 (1.94)	98.00 (0.78)	88.66 (1.62)	95.21 (1.26)	98.62 (1.39)
Soybeans-min	65.82 (5.83)	79.45 (1.74)	76.42 (4.61)	69.52 (3.63)	85.85 (1.35)	94.32 (2.02)	79.40 (1.33)	90.52 (1.16)	96.15 (0.57)
Soybean-clean	80.84 (4.29)	90.96 (3.58)	97.74 (0.86)	84.65 (3.19)	96.90 (1.61)	99.09 (0.45)	88.67 (1.80)	97.17 (1.64)	99.33 (0.18)
Wheat	99.61 (0.78)	99.80 (0.39)	99.71 (0.24)	99.90 (0.20)	99.90 (0.20)	100.00 (0.00)	99.32 (0.39)	100.00 (0.00)	99.90 (0.20)
Woods	91.21 (1.17)	94.80 (1.43)	97.71 (0.74)	88.35 (3.57)	95.54 (1.43)	98.85 (0.94)	96.74 (1.22)	98.12 (0.54)	98.96 (0.46)
Bldg-Grass-Tree-Drives	91.14 (2.07)	98.50 (0.81)	99.27 (0.60)	96.94 (3.11)	98.81 (1.42)	99.90 (0.13)	99.07 (0.26)	99.69 (0.25)	100.00 (0.00)
Stone-steel towers	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	99.57 (0.86)	100.00 (0.00)	100.00 (0.00)	99.78 (0.43)	98.92 (0.96)	100.00 (0.00)
Overall Accuracy	80.85 (1.58)	88.46 (0.32)	90.11 (0.67)	83.73 (1.30)	92.54 (0.16)	97.23 (0.30)	88.78 (0.78)	95.05 (0.28)	98.37 (0.17)
Average Accuracy	90.03 (0.98)	94.58 (0.22)	96.12 (0.28)	92.07 (0.66)	96.61 (0.24)	98.79 (0.10)	94.33 (0.35)	97.64 (0.13)	99.27 (0.11)
Kappa	78.44 (1.74)	86.92 (0.36)	88.81 (0.76)	81.68 (1.44)	91.54 (0.18)	96.85 (0.34)	87.31 (0.88)	94.38 (0.31)	98.15 (0.19)
Run time	47.97 (0.01)	48.50 (0.01)	48.21 (0.01)	193.40 (0.01)	192.85 (0.01)	197.65 (0.01)	421.07 (0.02)	404.15 (0.02)	405.46 (0.02)

Table 5: Classification accuracies obtained by our CNN (with patch sizes of $d = 9$, $d = 19$ and $d = 29$) for the Indian Pines hyperspectral dataset. We used 2500 iterations and repeated the experiment 5 times.

add a 10% dropout in the third convolution layer and raise the dropout from 10% to 30% on the first fully-connected layer. Also, for this experiment we

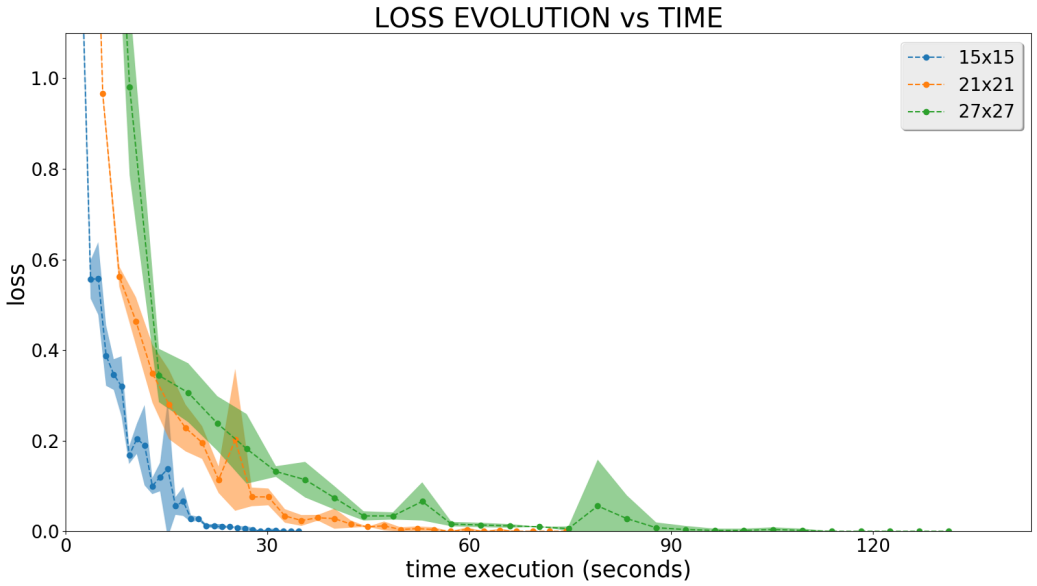


Figure 12: Pavia University: Evolution of the validation loss in terms of time (seconds) with $d = 15$, $d = 21$ and $d = 27$. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 1500 iterations.

have increased the number of iterations from 1500 to 2500. Thanks to the overfitting reduction, the network is able to converge much faster, drastically reducing the initial execution times (i.e., being 1.32 times faster with $d = 9$, 1.57 times faster with $d = 19$ and 1.66 times faster with $d = 29$ with 100 samples per class). In addition, stability has been improved by reducing the standard deviation of each run to 0.01. In Table 5 we can observe that the results with $d = 9$, $d = 19$ and $d = 29$ increase (in terms of accuracy) as more training samples per class are included, reaching the maximum values with 200 samples per class. Again, $d = 29$ reaches the best accuracy results with 200 samples per class, although the results obtained with a patch size of $d = 19$ are quite similar to those found using $d = 29$, just one or two percentage points below.

In Figures 13, 14 and 15 we report the classification maps obtained in each experiment, without the mirroring of the borders. First, in Figure 13 we can see the classified image without background (the first three images) and with background (the last three), obtained with a patch size of $d = 9$ and 50, 100 and 200 samples per class. When the background is removed,

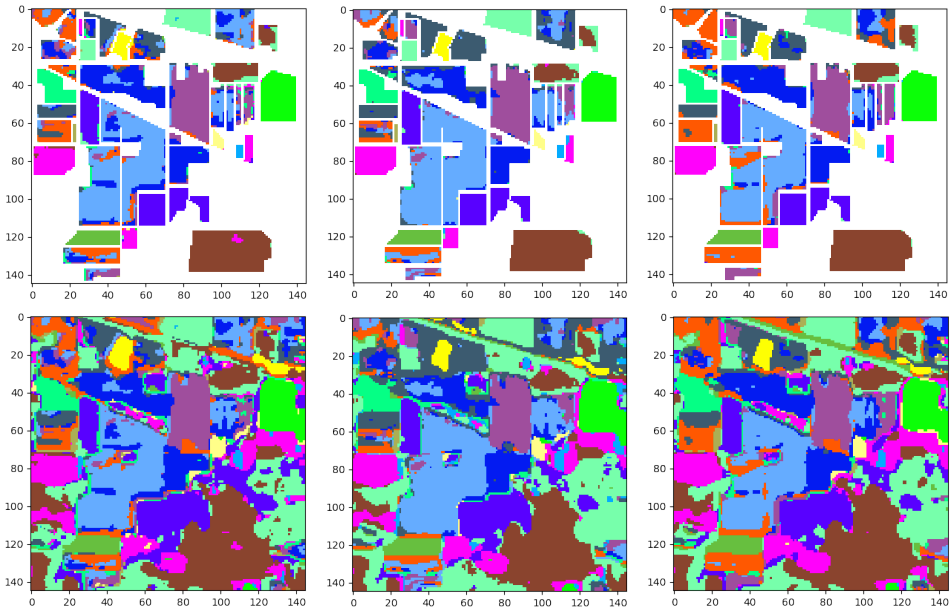


Figure 13: Classification results for Indian Pines image with $d = 9$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

Neural networks	CNN $d = 15$			CNN $d = 21$			CNN $d = 27$		
Samples per class	50	100	200	50	100	200	50	100	200
Asphalt	79.86 (4.46)	90.58 (0.74)	92.81 (0.72)	86.41 (0.47)	91.30 (1.81)	95.31 (0.97)	82.66 (1.18)	92.07 (1.05)	96.31 (0.19)
Meadows	88.97 (1.65)	94.20 (1.94)	97.20 (0.95)	89.44 (4.85)	93.39 (1.52)	98.16 (0.12)	90.39 (3.03)	93.56 (1.76)	97.54 (0.39)
Gravel	83.52 (1.92)	92.28 (2.12)	96.97 (0.90)	85.31 (4.51)	92.01 (3.55)	97.92 (0.59)	88.74 (0.90)	93.66 (1.16)	96.84 (0.29)
Trees	96.31 (1.02)	97.45 (1.34)	98.62 (0.43)	94.36 (0.49)	96.87 (0.74)	98.74 (0.18)	90.88 (1.35)	94.51 (2.12)	97.58 (0.41)
Painted metal sheets	99.83 (0.09)	99.93 (0.11)	100.00 (0.00)	99.38 (0.13)	99.88 (0.09)	100.00 (0.00)	99.31 (0.21)	99.53 (0.37)	99.65 (0.15)
Bare Soil	90.72 (1.47)	95.18 (0.93)	98.57 (0.74)	93.54 (2.45)	98.50 (1.20)	99.57 (0.31)	88.73 (2.00)	97.67 (0.88)	99.33 (0.25)
Bitumen	91.88 (1.61)	92.38 (1.22)	97.27 (0.96)	91.00 (0.57)	96.19 (1.50)	99.75 (0.09)	92.73 (1.25)	95.16 (1.29)	98.90 (1.14)
Self-Blocking Bricks	82.91 (5.26)	92.07 (1.49)	96.17 (1.70)	89.77 (2.62)	94.41 (0.93)	98.20 (0.21)	91.74 (1.49)	94.88 (1.43)	98.89 (0.47)
Shadows	99.65 (0.22)	99.54 (0.51)	99.86 (0.13)	99.65 (0.10)	99.75 (0.13)	99.82 (0.18)	97.40 (1.87)	98.91 (0.88)	99.58 (0.09)
Overall Accuracy	88.17 (0.31)	93.95 (0.74)	96.83 (0.19)	90.22 (1.78)	94.37 (1.10)	98.06 (0.13)	89.58 (1.95)	94.35 (1.05)	97.80 (0.22)
Average Accuracy	90.40 (0.38)	94.85 (0.41)	97.50 (0.14)	92.10 (0.63)	95.81 (0.82)	98.61 (0.09)	91.40 (1.26)	95.55 (0.68)	98.29 (0.25)
Kappa	84.63 (0.36)	92.07 (0.94)	95.83 (0.24)	87.28 (2.21)	92.63 (1.42)	97.44 (0.18)	86.37 (2.50)	92.61 (1.35)	97.09 (0.29)
Run time	43.02 (0.01)	42.78 (0.01)	42.83 (0.01)	74.30 (0.01)	74.17 (0.01)	74.09 (0.01)	132.77 (0.02)	132.48 (0.02)	132.68 (0.02)

Table 6: Obtained classification accuracies (with patch sizes of $d = 15$, $d = 21$ and $d = 27$) for the University of Pavia hyperspectral dataset. We used 1500 iterations and repeated the experiment 5 times.

we can see how the pixels of each class are mixed, in particular near the edges. Also, with background the results are poorly defined, although these are improved by adding more training samples in each class.

Secondly, in Figure 14 we can see the classified images (with and without

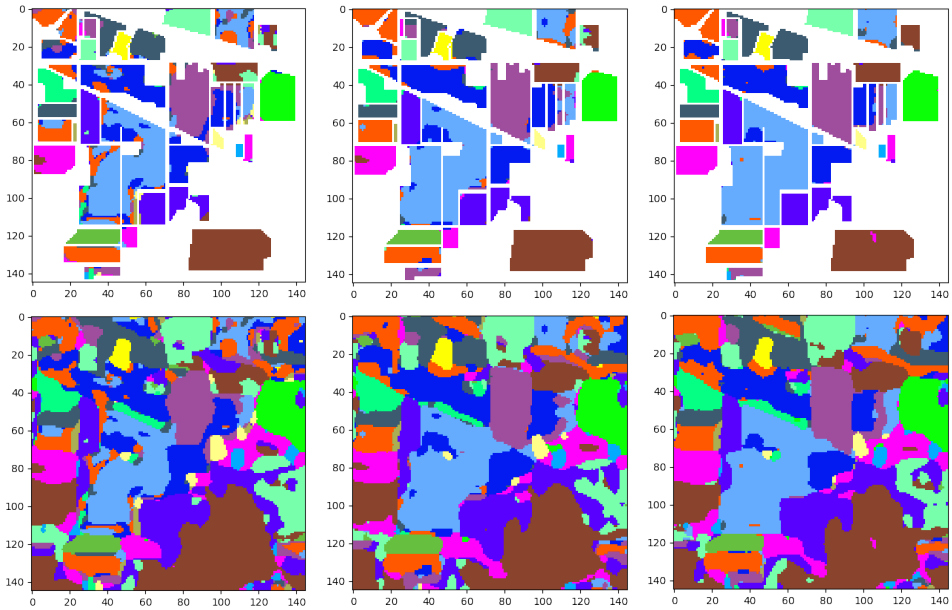


Figure 14: Classification results for Indian Pines image with $d = 19$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

background) obtained with a patch size of $d = 19$ and 50, 100 and 200 samples per class. With better results than with $d = 9$, the borders between the classes are better defined in this case, also in the classification with background. The best result is achieved with 200 samples per class.

Finally, in Figure 15 we can observe the obtained classified images with a patch size of $d = 29$. First we report the classification images without background and below them, we show the corresponding ones with background. With only ground-truth pixels, the first three images show better defined classes than the previously displayed ones with $d = 9$ and $d = 19$, being the classification map obtained with 200 samples per class the most similar to the original ground-truth image. Even in the classification with background, classes appear better defined since the borders between them are more regular.

On the other hand, the results for the University of Pavia dataset are shown in Table 6. For each patch size, we can observe that the accuracy improves as the number of samples per class increases, reaching the best ac-

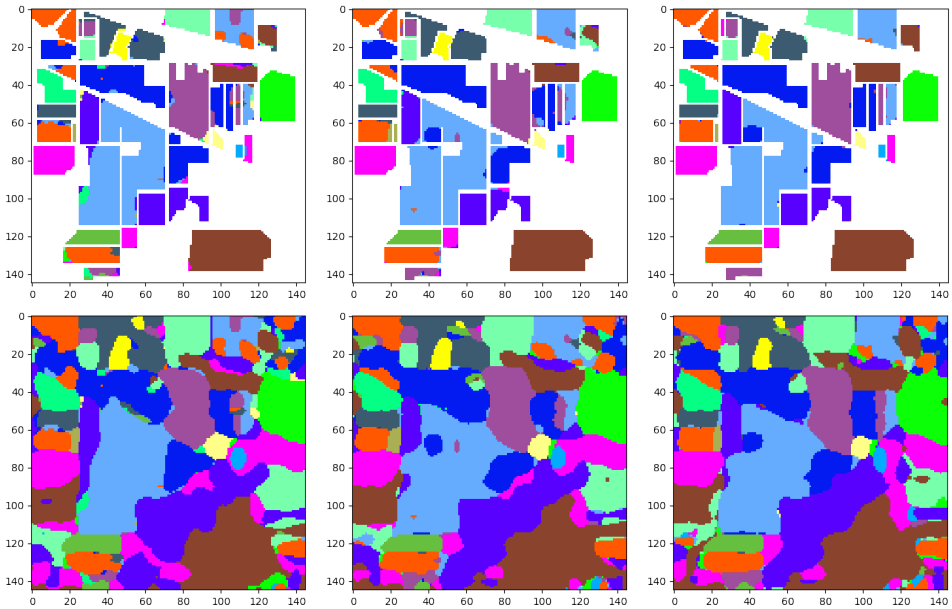


Figure 15: Classification results for Indian Pines image with $d = 29$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

curacy results with 200 samples per class. If we look for the most suitable patch size, we can say that $d = 21$ with 200 samples per class can be considered the best, with similar and balanced accuracy data. This is because, after adding more neighbors to the pixel, the data become more homogeneous.

In Figures 16, 17 and 18 we report the classification maps obtained for the University of Pavia dataset obtained in each experiment. The first one, Figure 16 shows the Pavia classification results with a patch size of $d = 15$. Here, the accuracy becomes better when the number of samples per class is increased, although there are many mixed pixels in the three cases, especially when the background is added to the classification. We can observe a poor result on top of the meadows class, and the ones adjacent to the light green bare soil at the lowest part of the image.

Figure 16 shows the Pavia classification results with a patch size of $d = 21$, which results in better results than $d = 15$. Pixels appear better defined, also with background. As expected, the best classification map is obtained with the maximum number of samples per class, i.e. 200, where we can see how the worst ranked pixels in the experiment with $d = 15$ are now better classified, e.g. the pixels at the top of the meadows class and the bare soil pixels.

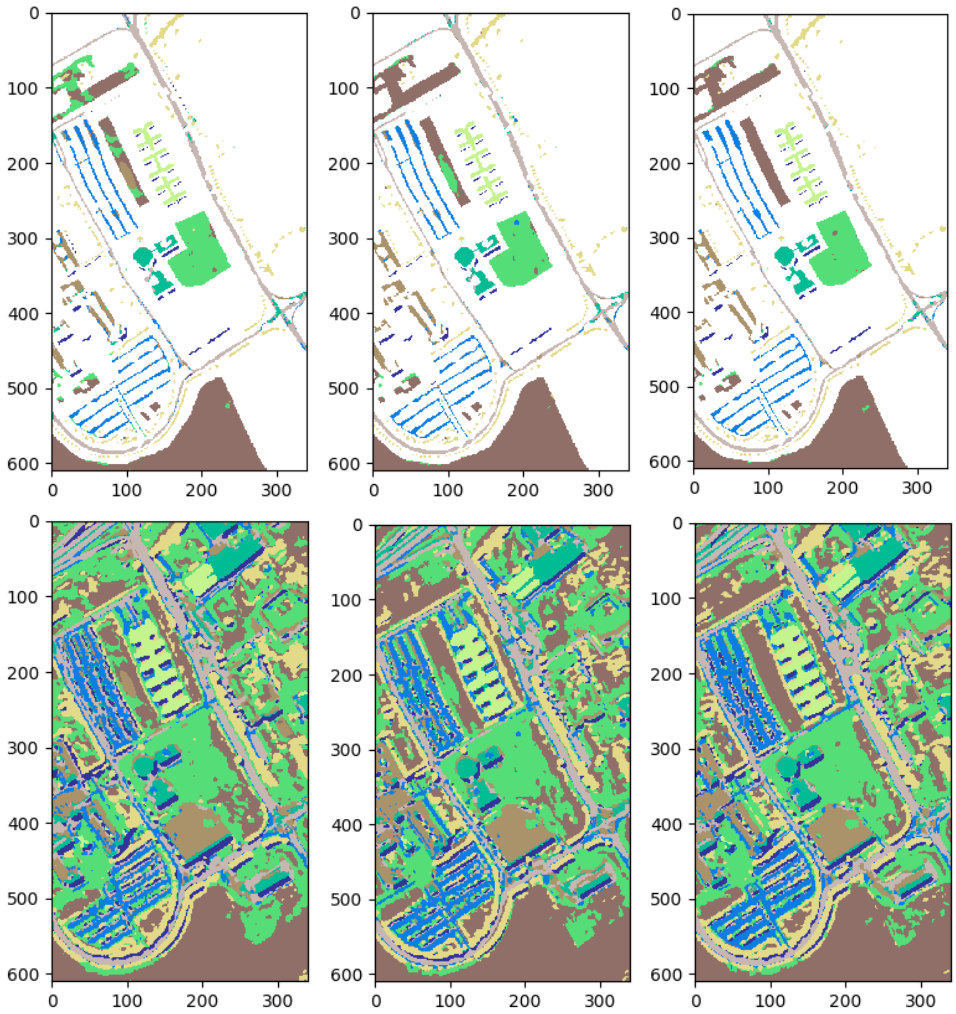


Figure 16: Classification results for the University of Pavia data set with $d = 15$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

Finally, in Figure 18 the classification results with patch size $d = 27$ are shown. As in the previous cases, the result improves as more pixels are added to the training set (with 200 samples per class resulting in the best accuracy results). Although the pixels at the top of the image are better classified, the CNN finds it more difficult to classify the pixels of the meadow class in the center of the image, and the final result is slightly worse than the one

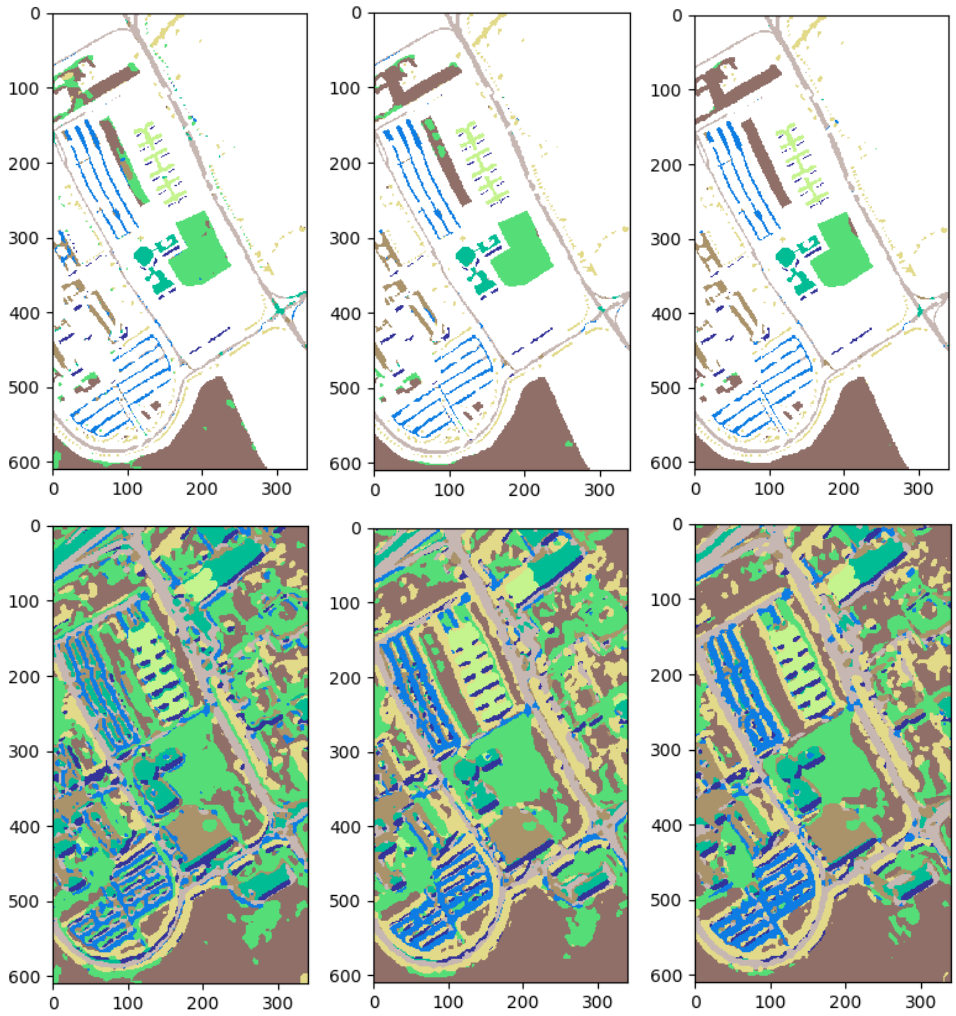


Figure 17: Classification results for the University of Pavia data set with $d = 21$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

obtained with patch size of $d = 21$

4.5. Comparison with other algorithms

In this section we show comparisons of our proposed method with other existing methods, including a standard MLP and the 1-D, 2-D and 3-D CNNs in [Chen et al. (2016)]. This represents an exhaustive and complete valida-

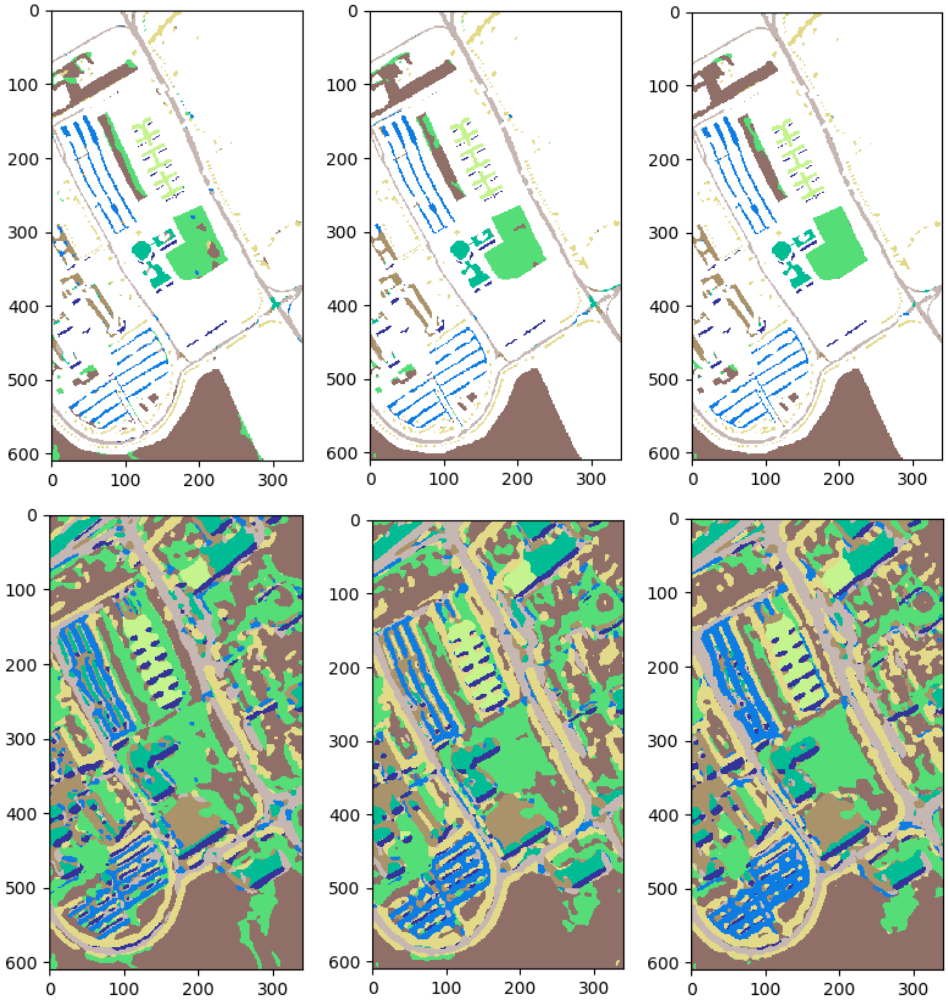


Figure 18: Classification results for the University of Pavia data set with $d = 27$ and 50 (left), 100 (center) and 200 (right) samples per class. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

tion of our method with state-of-the-art CNNs in the hyperspectral imaging literature. Tables 7 and 8 respectively show the configurations (for both scenes) of the MLP and CNNs used in experiments for comparison purposes.

Comparison with MLP: For the MLP, the chosen topology is a *single layer feedforward network* (SLFN) with three layers: an input layer which receives a pixel in all its bands, a hidden layer whose number of nodes is

MLP Topologies

Hyperspectral datasets	Layers		
	Type	Nº neurons	Activation Function
Indian Pines	Input	200 *(nº bands)*	-
	Hidden	144 *(optimal nº)*	ReLU
	Output	16 *(nº classes)*	Softmax
University of Pavia	Input	103 *(nº bands)*	-
	Hidden	75 *(optimal nº)*	ReLU
	Output	9 *(nº classes)*	Softmax

Common parameters

Batch size	Iterations	Learning rate	Optimizer
100	5000	0.045	AdamOptimizer

Table 7: Configuration of the MLP used in experiments for comparative purposes.

1-D CNN Topologies [Chen et al. (2016)]

Hyperspectral datasets	Conv. Layers	ReLU	Pooling
Indian Pines	1×5	Yes	1×2
	1×5	Yes	1×2
	1×4	Yes	1×2
	1×5	Yes	1×2
	1×4	Yes	1×2
University of Pavia	1×8	Yes	1×2
	1×7	Yes	1×2
	1×8	Yes	1×2

2-D CNN Topologies [Chen et al. (2016)]

Hyperspectral datasets	Conv. Layers	ReLU	Pooling	Dropout
Indian Pines	$32 \times 4 \times 4$	Yes	2×2	No
	$64 \times 5 \times 5$	Yes	2×2	50%
	$128 \times 4 \times 4$	Yes	No	50%
University of Pavia	$32 \times 4 \times 4$	Yes	2×2	No
	$64 \times 5 \times 5$	Yes	2×2	50%
	$128 \times 4 \times 4$	Yes	No	50%

3-D CNN Topologies [Chen et al. (2016)]

Hyperspectral datasets	Conv. Layers	ReLU	Pooling	Dropout
Indian Pines	$128 \times 4 \times 4 \times 32$	Yes	2×2	No
	$192 \times 5 \times 5 \times 32$	Yes	2×2	50%
	$256 \times 4 \times 4 \times 32$	Yes	No	50%
University of Pavia	$32 \times 4 \times 4 \times 32$	Yes	2×2	No
	$64 \times 5 \times 5 \times 32$	Yes	2×2	50%
	$128 \times 4 \times 4 \times 32$	Yes	No	50%

Table 8: Configuration of the CNNs used in experiments for comparative purposes. These are the 1-D, 2-D and 3-D CNNs configurations described in [Chen et al. (2016)]

calculated by $(n\_bands + n\_classes) \cdot \frac{2}{3}$, with a ReLU as activation function, and an output layer with the number of nodes equals as the number of classes and a *softmax* function. So, the final MLP topology for the Indian Pines hyperspectral data set is $200 - 144 - 16$ and for the University of Pavia data set is $103 - 75 - 9$, both with Adam optimizer [Kingma and Ba (2014)] and 0.0045 of learning rate. The proposed topologies have been tested with three different numbers of samples per class: 50, 100 and 200, repeating each one five times. Results are shown in Table 9, where we show the average time and accuracy of each execution with 5000 iterations and repeated five times. As we can see, as we add training data the accuracy increases, however the execution time remains fairly stable.

Datasets	**Samples**	**Time per step**		**Total time**		**Accuracy**	
		average	Std. deviation	average	Std. deviation	average	Std. deviation
Indian Pines	50	0.0036	0.0002	0.1791	0.0148	74.60	1.60
	100	0.0035	0.0001	0.1757	0.0149	79.29	1.35
	200	0.0036	0.0002	0.1800	0.0148	82.56	1.23
University of Pavia	50	0.0033	0.0001	0.1672	0.0147	82.79	1.92
	100	0.0031	0.0001	0.1550	0.0149	87.16	0.93
	200	0.0031	0.0001	0.1530	0.0149	87.76	1.75

Table 9: Execution times and accuracies obtained by the MLP (with configuration: $200 - 144 - 16$) for the Indian Pines scene and by the MLP (with configuration: $103 - 75 - 9$) for the University of Pavia scene, using 50, 100 and 200 samples per class. This experiment has been executed using 5000 iterations.

In Table 10 we can observe in detail the results obtained for each class of the Indian Pines dataset. After 5000 iterations, we can see that the best overall accuracy result (84.60%) is obtained with the maximum number of samples per class, i.e. 200, without an overwhelming time difference. However, even with 5000 iterations, the MLP is still not able to reach 90% overall accuracy (although it reaches a 91.66% of average accuracy). If we pay attention to Figure 19, we can see how the MLP needs fewer iterations to reach a low error as the number of samples per class increases in training, with a very little difference in execution times: each iteration of MLP with 200 samples per class is only 1.01 times slower than with 50 samples and 1.02 times slower than with 100 samples per class, as we can see in Figure 20. Note how the optimizer in the first second quickly evolves from a very high initial error to a more reasonable value, given the cost function under which it is iterating.

Neural network	MLP		
Samples per class	50	100	200
Alfalfa	98.26 (2.54)	98.70 (1.06)	97.39 (0.87)
Corn-notill	63.40 (4.60)	71.11 (3.74)	78.36 (4.46)
Corn-min	66.00 (3.58)	82.53 (2.47)	86.17 (3.31)
Corn	86.16 (2.77)	91.31 (2.08)	92.41 (2.28)
Grass/Pasture	90.52 (2.67)	91.88 (2.30)	96.31 (1.27)
Grass/Trees	93.45 (1.35)	95.64 (1.45)	97.73 (0.94)
Grass/pasture-mowed	97.14 (2.67)	97.86 (2.86)	97.86 (2.86)
Hay-windrowed	96.32 (1.47)	97.45 (0.80)	98.62 (0.54)
Oats	99.00 (2.00)	98.00 (4.00)	100.00 (0.00)
Soybeans-notill	75.12 (4.54)	83.87 (2.11)	87.00 (2.33)
Soybeans-min	62.81 (2.86)	62.42 (5.28)	68.99 (3.70)
Soybean-clean	79.39 (1.54)	84.69 (1.13)	87.86 (1.84)
Wheat	98.54 (0.31)	99.22 (0.59)	99.41 (0.37)
Woods	83.65 (4.78)	91.07 (1.55)	94.15 (2.29)
Bldg-Grass-Tree-Drives	74.46 (1.94)	82.75 (2.30)	85.03 (4.89)
Stone-steel towers	98.49 (1.10)	99.35 (0.53)	99.35 (0.53)
Overall Accuracy	75.24 (1.51)	80.34 (1.10)	84.60 (0.71)
Average Accuracy	85.17 (1.21)	89.24 (0.41)	91.66 (0.29)
Kappa	72.18 (1.51)	77.93 (1.10)	82.65 (0.71)
Runtime (sec.)	0.1791	0.1757	0.1800

Table 10: Classification accuracies (and standard deviation) obtained by MLP with 50, 100 and 200 samples per class for the Indian Pines hyperspectral dataset. This experiment has been executed using 5000 iterations.

Now we can compare the MLP classifier with our proposed CNN for the Indian Pines scene. We have considered two experiments: in the first one, we compared the MLP with 100 samples per class with the proposed CNN with also 100 samples per class and patch sizes of $d = 9$, $d = 19$ and $d = 29$. Each classifier has been executed five times with 1500 iterations. In Figure 21 we can observe the evolution of the error in terms of the number of iterations of the MLP and the CNN. We can conclude that our CNN needs significantly less iterations to reach a low error when it uses patches of size $d = 29$ and $d = 19$ (specifically, the CNN reaches an error below 0.1 with only 300 iterations when $d = 29$ and around 700-800 iterations when $d = 19$) while the MLP barely drops from 0.2 in 1500 iterations. However, in Figure 22 (in which we show the error evolution in terms of time in seconds) we can observe that one iteration of the CNN (with any patch size) is always slower than one iteration of the MLP.

The second comparison between the MLP and our CNN is reported in Table 12. In this case we used 200 samples per class for the MLP and CNN, and patches of size $d = 9$, $d = 19$ and $d = 29$. Table 12 shows that the MLP is the fastest classification method (all its executions take around 0.17-0.18 seconds), reaching its better average and overall accuracy values (91.66% and

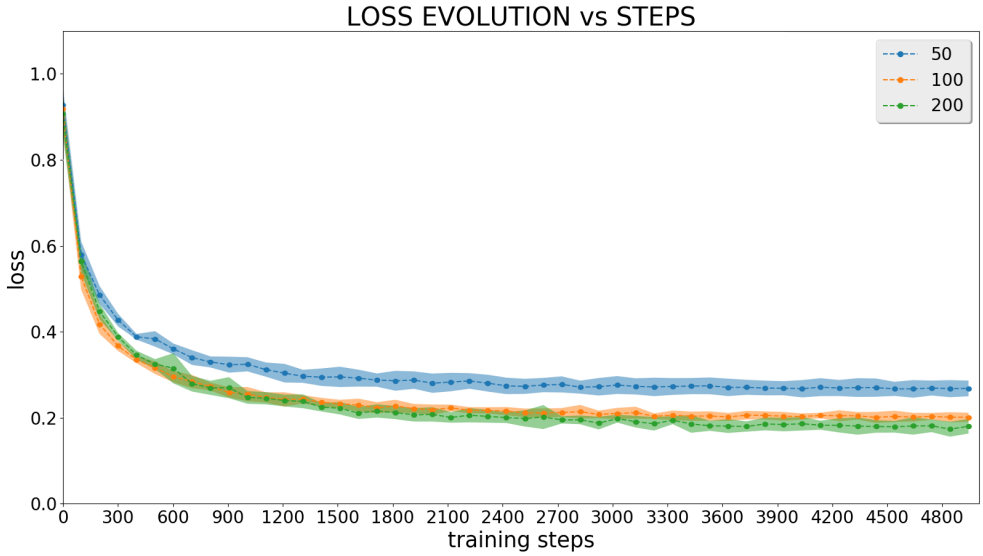


Figure 19: Evolution of the validation error in terms of steps (iterations) with 50, 100 and 200 samples per class for the MLP with the Indian Pines scene. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 5000 iterations.

84.60%, respectively) with 200 samples per class. However, these results are several points lower than the accuracies reached by the proposed CNN with patches of size $d = 19$ and $d = 29$. Specifically, the MLP reaches an overall accuracy around 14 points lower than the CNN, and an average accuracy around 8 points lower than the one achieved by the CNN for $d = 29$. With $d = 19$, the MLP reaches an overall accuracy around 11 lower than that achieved by the CNN and an average accuracy around 7 points lower than that achieved by the CNN. Only if we compare the MLP with the CNN and $d = 9$ the MLP reaches better overall and average accuracies: around 3 points better than the overall accuracy achieved by the CNN and around 1.5 points better than the average accuracy achieved by the CNN.

The resulting classification maps obtained by the MLP are shown in Figure 23, where the classification results without background (top row) and with background (lower row) are reported. In both cases, unlike the CNN, the MLP results are not well defined, with many pixels of different classes appearing mixed in the final classification. An increase in the number of samples per class slightly improves the classification results, without reaching the quality of the classification maps provided by the CNN in Figures 13,

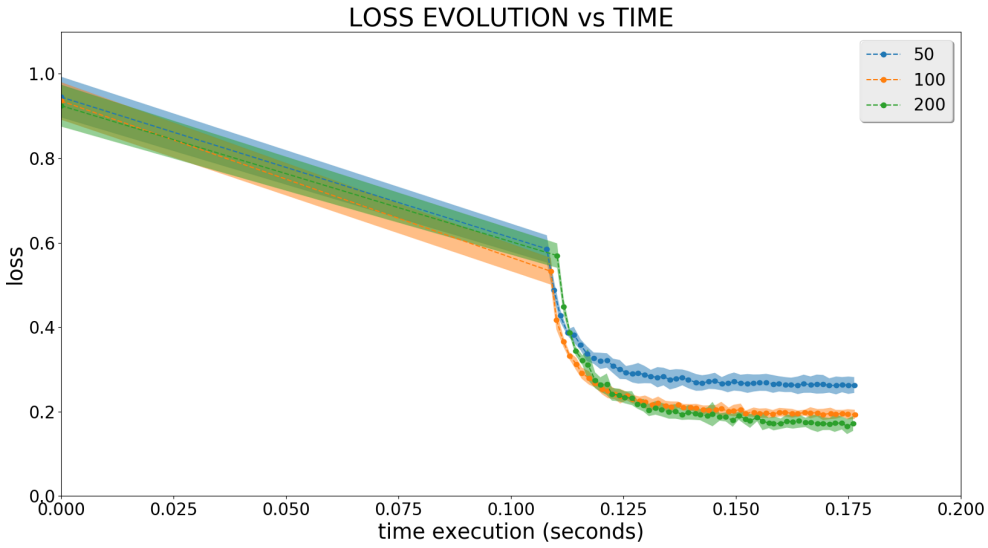


Figure 20: Evolution of the validation error in terms of time (seconds) with 50, 100 and 200 samples per class for the MLP with the Indian Pines scene. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$). This experiment has been executed using 5000 iterations.

14 and 15.

Neural network	MLP		
Samples per class	50	100	200
Asphalt	81.29 (1.15)	83.34 (1.18)	84.92 (0.87)
Meadows	82.83 (4.46)	85.44 (2.06)	89.56 (3.75)
Gravel	84.71 (1.79)	85.98 (4.89)	94.68 (1.41)
Trees	91.64 (2.03)	94.77 (0.50)	96.48 (1.54)
Painted metal sheets	99.18 (0.16)	99.38 (0.36)	99.43 (0.30)
Bare Soil	84.87 (3.31)	88.94 (2.89)	90.75 (2.85)
Bitumen	89.35 (3.10)	92.63 (0.18)	93.76 (0.90)
Self-Blocking Bricks	79.57 (2.35)	79.50 (6.26)	63.94 (3.50)
Shadows	99.79 (0.09)	99.54 (0.36)	99.96 (0.05)
Overall Accuracy	84.37 (2.30)	86.68 (0.67)	88.20 (1.50)
Average Accuracy	88.14 (1.05)	89.95 (0.38)	90.39 (0.30)
Kappa	79.87 (2.30)	82.79 (0.67)	84.67 (1.50)
Runtime (sec.)	0.1672	0.1550	0.1530

Table 11: Classification accuracies obtained by MLP with 50, 100 and 200 samples per class for the University of Pavia dataset.

Also, Table 9 summarizes the experiments conducted using the MLP classifier with the University of Pavia dataset for 50, 100 and 200 samples per class (all repeated five times, with 5000 iterations per execution). We can

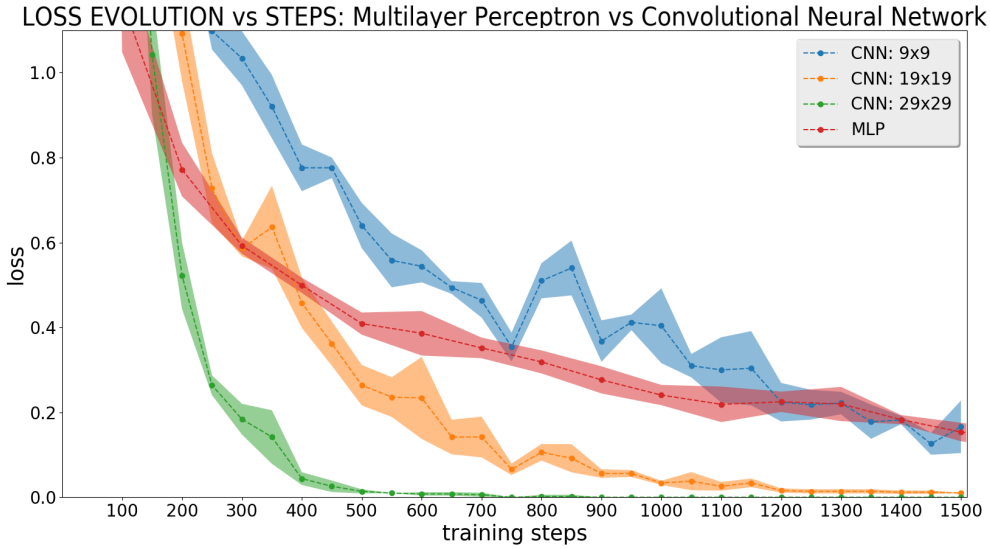


Figure 21: Evolution of the validation error for the MLP and CNN (with $d = 9$, $d = 19$ and $d = 29$) in terms of steps for the Indian Pines image. The shadow shows the standard deviation of the loss for each network repeated five times. Zoom in $(0, 1)$. This experiment has been executed using 1500 iterations.

observe that the best accuracy result (87.76%) is obtained with the maximum number of samples per class, which is 200, as it was already the case with the Indian Pines image. The execution times for 50, 100 and 200 samples per class are very similar too. However, with 5000 iterations the MLP is also unable to reach 90% accuracy. In Figure 24 we can see how the error descends as the MLP iterates, needing less iterations as the number of samples in the training increases, with a very little difference in execution times as we can see in Figure 25. Also, in Table 11 we can observe the accuracy results for each class obtained after executing five times the MLP with the Pavia dataset.

Now, we can compare the results obtained by the MLP over the University of Pavia data set with the results obtained by the considered CNN architectures. In order to do so, we have performed two experiments: in the first one we execute the MLP with 100 samples per class and we choose a patch size of $d = 15$, $d = 21$ and $d = 27$ for the CNN, and further execute it with 100 samples per class too. Each experiment has been run using 1500 iterations. In Figure 26 we can see that the MLP needs more iterations than the CNN to reduce its associated error, without reaching in any case the er-

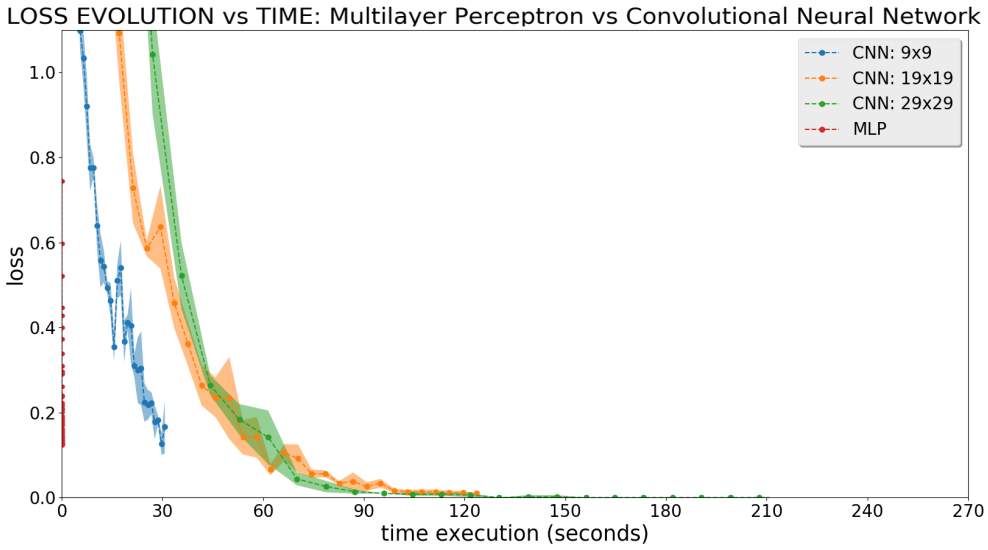


Figure 22: Evolution of the validation error for the MLP and CNN (with $d = 9$, $d = 19$ and $d = 29$) in terms of time (seconds) for the Indian Pines image. The shadow shows the standard deviation of the loss for each network repeated five times. Zoom in $(0, 1)$. This experiment has been executed using 1500 iterations.

ror achieved by the CNN, although its iterations are faster than those of the CNN, as we can observe in Figure 27, where it is shown that the execution tine of the MLP takes less than one second.

The second experiment reports a comparison between the MLP and the proposed CNN with 200 samples per class for both classifiers, and patch sizes of $d = 15$, $d = 21$ and $d = 27$ for the CNN. Table 13 shows that the MLP is the fastest classification method, even faster than the MLP with Indian Pines (all its executions take around 0.15-0.16 seconds), reaching its best average ad overall values (91% and 89% respectively) with 200 samples per class. But, again, the CNN reaches better accuracy values, with an average accuracy of 98% and overall accuracy of 97%, i.e. around 8-9% points better than MLP due to the inability of the latter architecture to improve its outcome.

In Figure 28 we can observe the MLP classification maps for the University of Pavia. The top images show the classification with ground-truth pixels, whose result improves as more samples are added in the training. However the classification maps are very mixed as compared to those obtained by the CNN in Figures 16, 17 and 18. On the other hand, the bottom images show the classification with the whole background. In this case, the

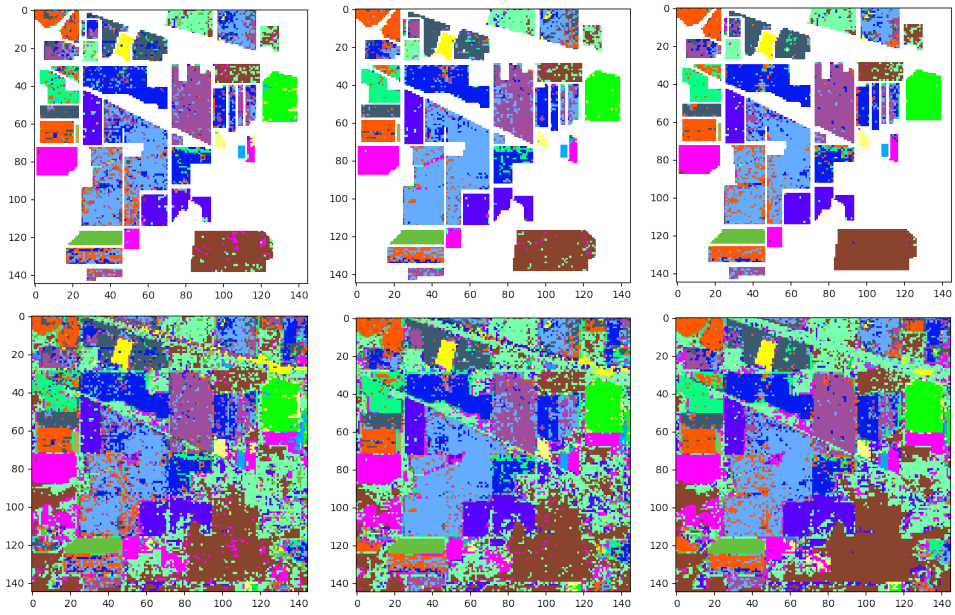


Figure 23: Indian Pines classification results achieved by the MLP with 50 (left), 100 (center) and 200 (right) samples per class with 1500 iterations. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

areas of the image are relatively well distinguished (even better when more samples per class are added), although the number of mixed pixels is greater than in the CNN experiments with $d = 27$, $d = 21$ and $d = 15$.

Comparison with other convolutional networks: Now we compare our proposed CNN with other deep architectures, in particular with the 1-D, 2-D and 3-D CNNs described in [Chen et al. (2016)]. In this work the authors studied the application of supervised CNNs in hyperspectral imaging feature extraction. Three deep feature extraction architectures based on the CNN were proposed to extract the spectral, spatial, and spectral-spatial features of hyperspectral imaging, respectively. To address the overfitting problem caused by the limited number of training samples, the authors implemented some regularization strategies, including L2 regularization and dropout in the training process. Also, they proposed a virtual sample enhanced method to create training samples. The main differences between our method and the one described in [Chen et al. (2016)] can be summarized in the following points:

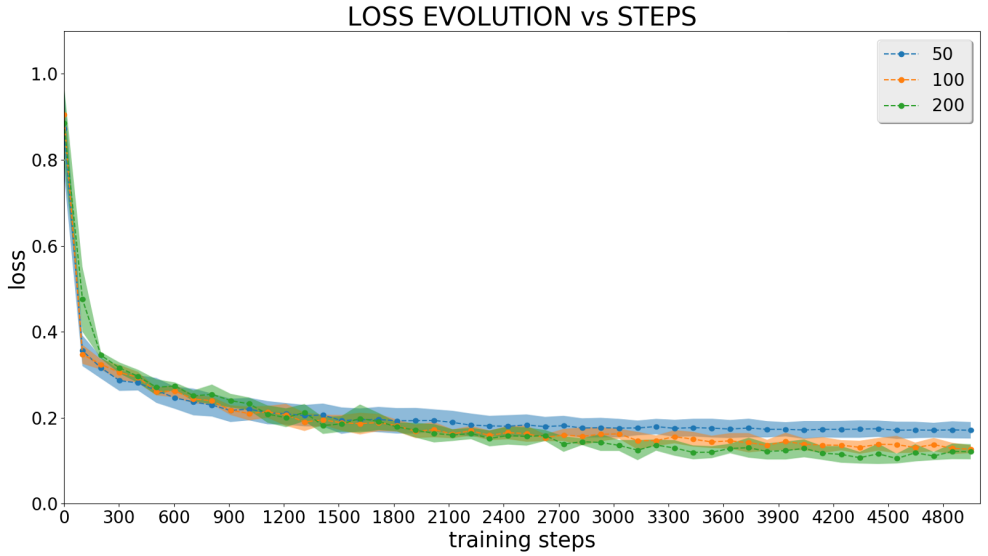


Figure 24: Error evolution for the MLP in terms of steps (iterations) with 50, 100 and 200 samples per class for the University of Pavia data set. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$).

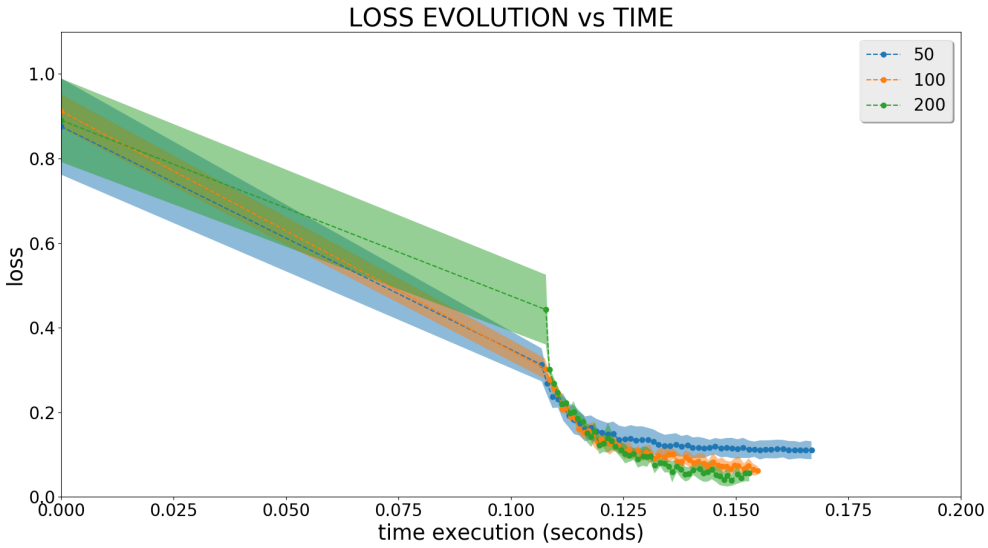


Figure 25: Error evolution for the MLP in terms of time (seconds) with 50, 100 and 200 samples per class for the University of Pavia data set. The shadow shows the standard deviation of the loss for the five executions of each patch (zoom in error interval $[0, 1]$).

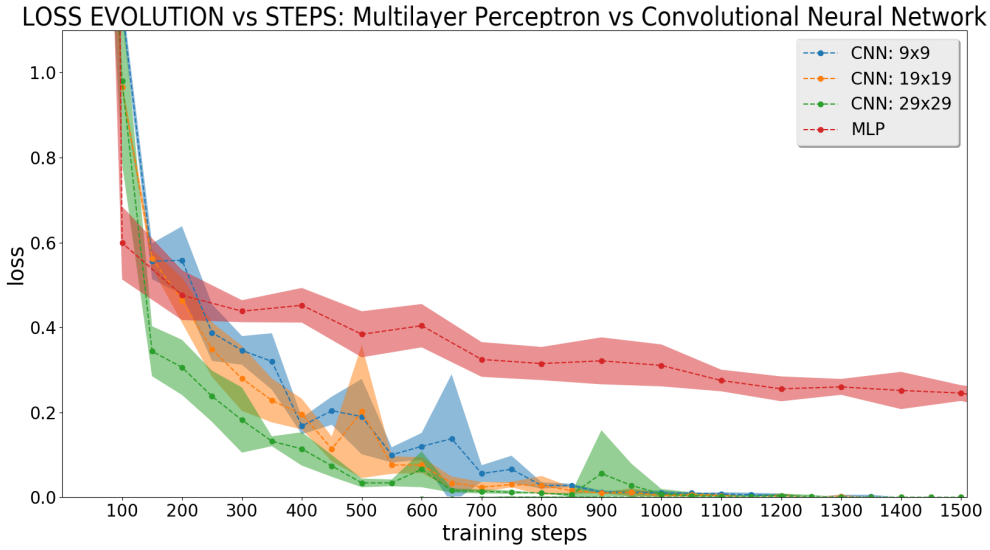


Figure 26: Error evolution for the MLP and CNN (with $d = 15$, $d = 21$ and $d = 27$) in terms of steps for the University of Pavia data set. The shadow shows the standard deviation of the loss for each network repeated five times. Zoom in $(0, 1)$

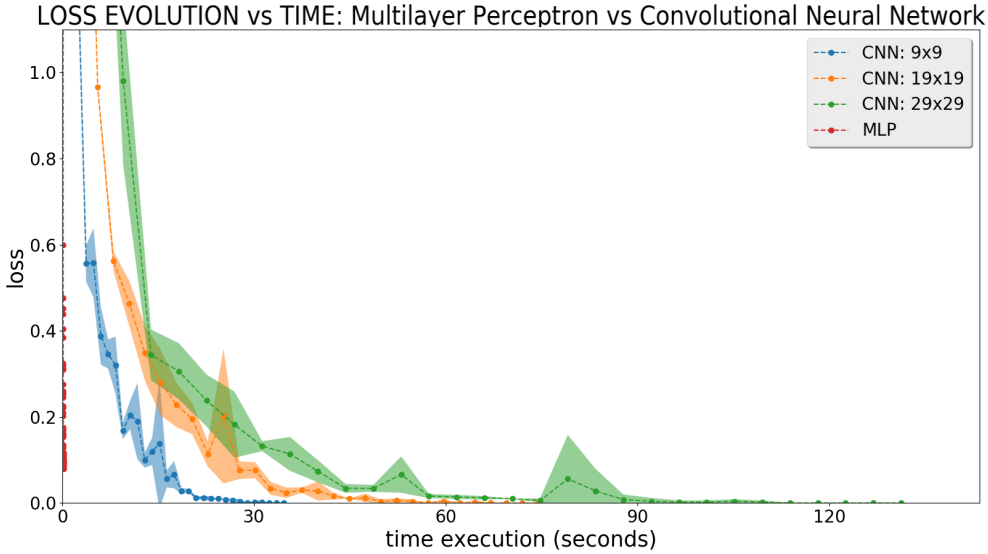


Figure 27: Error evolution for the MLP and CNN (with $d = 15$, $d = 21$ and $d = 27$) in terms of time (seconds) for the University of Pavia data set. The shadow shows the standard deviation of the loss for each network repeated five times. Zoom in $(0, 1)$

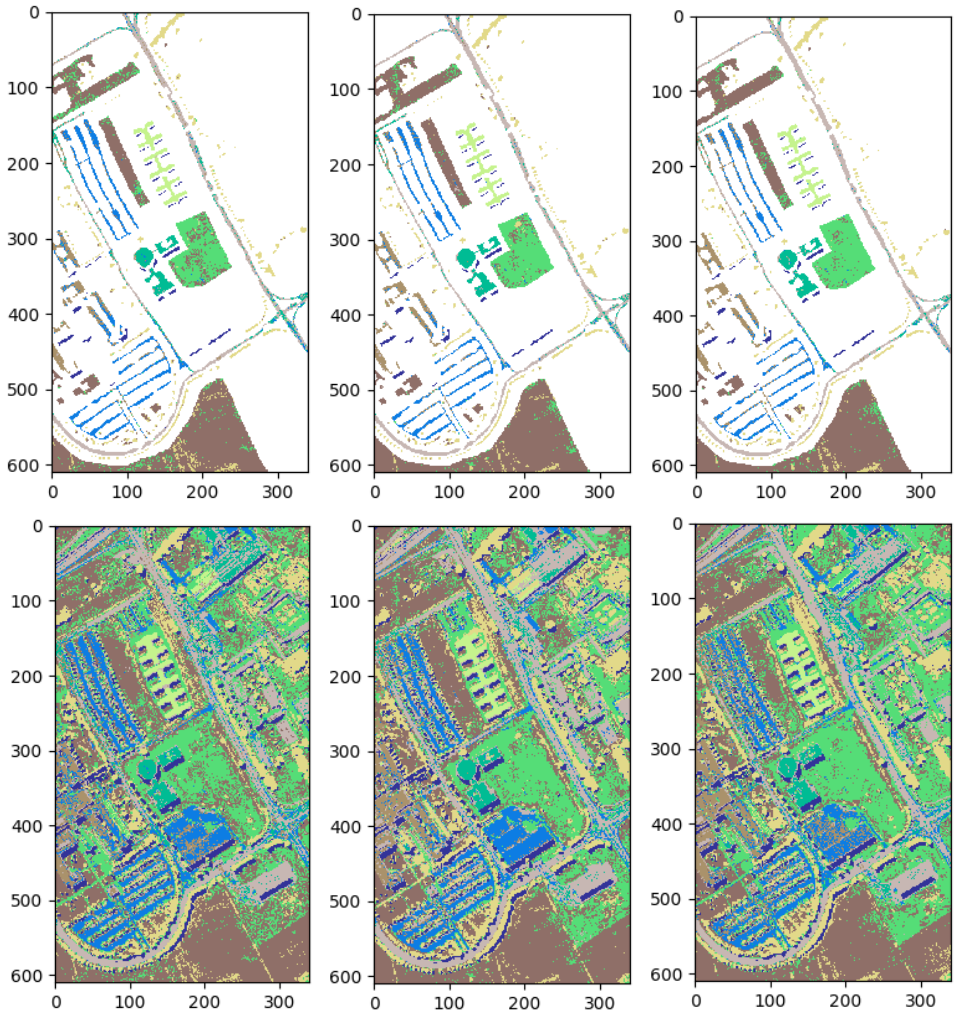


Figure 28: Classification results achieved by the MLP for the University of Pavia data set with 50, 100 and 200 samples per class with 1500 iterations. The upper row displays the classification result without the background, and the lower row displays the classification result with the background.

- Regarding the configuration of the 1-D CNN, we provide a detailed description in Table 8. For the Indian Pines dataset, the learning rate of the 1-D CNN is fixed to 0.005 with 700 training epochs, while for University of Pavia dataset, the learning rate is fixed to 0.001 with 600 epochs. The datasets are divided into training and testing sets. For the Indian Pines dataset, 1765 labeled pixels are chosen to create the

training set, while for the University of Pavia dataset, the authors use 3930. This spectral-CNN receives a normalized pixel vector (1×200 if it is an Indian Pines pixel and 1×103 if it is an University of Pavia pixel) in the range $[-1, 1]$. The data suffers a L2 regularization along the CNN and, at the end of the CNN procedure, the input pixel vector is converted into a feature vector that is fed to *Logistic Regression* (LR) for classification. The authors selected a mini-batch update strategy, and the cost function is calculated on a mini-batch of inputs as $c_o = -\frac{1}{m}\sum_{i=1}^{m}[x_i \log(z_i) + (1 - x_i)\log(1 - z_i)]$, using mini-batch stochastic gradient descent as optimizer of the 1-D CNN.

- Regarding the configuration of the 2-D CNN, we provide a detailed description in Table 8. This spatial-CNN receives, through a preprocessin with PCA, patches of size 27×27 normalized in the range $[-0.5, 0.5]$ and grouped in batches of 100. The output of the CNN is a feature vector of 1×128 that is sent to LR for classification. As in the previous 1-D CNN, the input image is represented by some feature vectors, which capture the spatial information contained in the neighborhood region of the input pixel. Then, the learned features are fed to the LR for classification.

- Finally, the configuration of the 3-D CNN is shown in Table 8. This spatial-spectral CNN receives patches of size $27 \times 27 \times n_{bands}$ normalized in range $[-0.5, 0.5]$ and grouped in batches of 100. In this case, n_{bands} is fixed to 32. The learning rate is fixed to 0.003 and the training epochs is set to 400. After convolutional and pooling layers, the input data is transformed and fed to LR for classification.

In Table 12 we can see a detailed comparison between the different tested neural networks using Indian Pines dataset. The first column reports the results obtained by the MLP (trained with 200 samples per class). The second column provides a comparison between the CNNs in Chen et al. (2016) and our CNN (using the same number of samples per class as the methods in Chen et al. (2016)), with patch sizes of $d = 9$, $d = 19$ and $d = 29$. Finally, in the third column we also report the results obtained by our proposed CNN (trained with 200 samples per class) for comparison.

If we focus on analyzing the results provided by the different implementations of Chen et al. (2016) in the second column, we can see that the one with spatial information (2-D CNN) achieves better results than the one with

Accuracy table

Neural networks	MLP		CNNs in [Chen et al. (2016) versus the proposed CNN							Proposed CNN			
	SLFN	Samples	1-D	2-D	3-D	$d = 9$	$d = 19$	$d = 29$	Samples	$d = 9$	$d = 19$	$d = 29$	Samples
Alfalfa	97.39	33	89.58	99.65	100.00	100.00	100.00	100.00	30	99.13	99.57	99.13	33
Corn-notill	78.36	200	85.68	90.64	96.34	90.57	94.06	97.17	150	80.48	94.47	98.17	200
Corn-min	86.17	200	87.36	99.11	99.49	97.69	96.43	98.17	150	96.65	98.22	98.92	200
Corn	92.41	181	93.33	100.0	100.00	99.92	100.00	100.00	100	99.66	100.00	100.00	181
Grass/Pasture	96.31	200	96.88	98.48	99.91	98.10	98.72	98.76	150	99.46	99.75	99.71	200
Grass/Trees	97.73	200	98.99	97.95	99.75	99.34	99.67	100.00	150	99.53	98.90	99.40	200
Grass/pasture-mowed	97.86	20	91.67	100.00	100.00	100.00	100.00	100.00	20	100.00	100.00	100.00	20
Hay-windrowed	98.62	200	99.49	100.00	100.00	99.58	99.92	100.00	150	99.67	99.62	100.00	200
Oats	100.00	14	100.00	100.00	100.00	100.00	100.00	100.00	15	100.00	100.00	100.00	14
Soybeans-notill	87.00	200	90.35	95.33	98.72	94.28	97.63	99.14	150	92.43	98.00	98.62	200
Soybeans-min	68.99	200	77.90	78.21	95.52	87.75	92.93	94.59	150	76.42	94.32	96.15	200
Soybean-clean	87.86	200	95.82	99.39	99.47	94.81	97.17	99.06	150	97.74	99.09	99.33	200
Wheat	99.41	143	98.59	100.00	100.00	100.00	100.00	100.00	150	99.71	100.00	99.90	143
Woods	94.15	200	98.55	97.71	99.55	98.09	97.88	99.76	150	97.71	98.85	98.96	200
Bldg-Grass-Tree-Drives	85.03	200	87.41	99.31	99.54	89.79	95.80	98.39	50	99.27	99.90	100.00	200
Stone-steel towers	99.35	75	98.06	99.22	99.34	100.00	99.57	98.92	50	100.00	100.00	100.00	75
Overall Accuracy (OA)	84.60		87.81	89.99	97.56	93.94	96.29	97.87		90.11	97.23	98.37	
Average Accuracy (AA)	91.66		93.12	97.19	99.23	96.87	98.11	99.00		96.12	98.79	99.27	
Kappa	82.65		85.30	87.95	97.02	93.12	95.78	97.57		88.81	96.85	98.15	
Runtime (sec.)	0.1800		457.8	357.0	1675.2	74.47	189.51	158.42		48.21	197.65	405.46	
Total samples		2466							1765				2466

Table 12: Classification accuracies obtained by different neural networks tested using the Indian Pines dataset: 1) first column: results obtained by the MLP (trained with 200 samples per class); 2) second column: comparison between the results obtained by the 1-D CNN, 2-D CNN and 3-D CNN in [Chen et al. (2016)] and the results obtained by our CNN (trained with the same number of samples per class as the CNNs in [Chen et al. (2016)]), using different values of parameter d); 3) third column: results obtained by our CNN (trained with 200 samples per class, using different values of parameter d).

spectral information (1-D CNN). Also, we can see that the inclusion of the two sources of information (3-D CNN) leads to an overall improvement of the accuracy. In the same column we can observe that, when using our CNN (with the same number of samples per class as the methods in Chen et al. (2016)), the increase in the value of parameter d leads to an improvement in the obtained classification result. Also, our method is faster than all the methods reported in Chen et al. (2016) and comparable in terms of overall accuracy to the best methods reported in that work. Specifically, our proposed CNN implemented with $d = 29$ is 2.89 times faster than the 1-D CNN, with an overall accuracy that is 10.06 percentage points better; 2.25 times faster than the 2-D CNN, with an overall accuracy 7.88 percentage points better; and 10.57 times faster than the 3-D CNN, with very similar overall accuracy. Finally, a comparison between the results in the first and third columns of Table 12 indicate that our proposed CNN can achieve better results in terms of overall accuracy than the MLP, but the MLP is faster.

Also in Figure 29 we provide a graphical comparison of the overall accu-

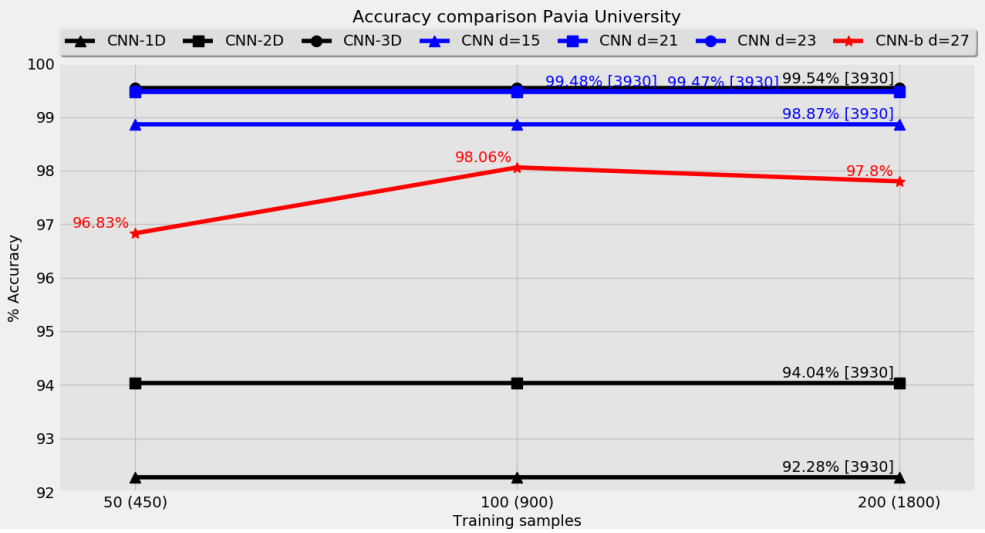


Figure 29: Comparison of the overall accuracy achieved by CNN classifiers with the Indian Pines scene. The horizontal black lines show the overall accuracy results reached by the 1-D, 2-D and 3-D CNNs in Chen et al. (2016), and the horizontal blue lines show the overall accuracy results obtained by our proposed CNN, implemented with different values of d and trained with the same number of samples than the CNNs in Chen et al. (2016). The red line (marked as CNN-b in the figure) corresponds to our CNN, implemented with $d = 29$ but trained with 50, 100 and 200 samples per class. Above each black and blue line, we report the number of used samples and the overall accuracy reached (in square brackets). For the red line, we only report the overall accuracy reached (the number of used samples for this line is defined by the x-axis). On the other hand, the y-axis shows the overall accuracies obtained in the experiments.

racy results obtained by the proposed CNN the CNNs implemented by Chen et al. (2016). The horizontal blue lines in Figure 29 show the overall accuracy results obtained by our proposed CNN with different values of d and trained using the same number of samples than the CNNs in Chen et al. (2016). The red line corresponds to our CNN, implemented with $d = 29$ but trained with 50, 100 and 200 samples per class. As we can see, the proposed CNN can reach better overall accuracies than the compared 1-D and 2-D CNNs. For the 3-D CNN, the results can be comparable in terms of overall accuracy. However, since our architecture is optimized and executed on a GPU, we can get better results from the viewpoint of processing time.

In Table 13 we report a comparison between the different tested neural networks using Pavia University data set. The first column reports the results obtained by the MLP (trained with 200 samples per class). The second

Accuracy table

Neural networks	MLP		CNNs in [Chen et al. (2016) versus the proposed CNN							Proposed CNN			
	SLFN	Samples	1-D	2-D	3-D	$d = 15$	$d = 21$	$d = 23$	Samples	$d = 15$	$d = 21$	$d = 27$	Samples
Asphalt	84.92	200	92.06	97.11	99.36	97.53	98.80	98.59	548	92.81	95.31	96.31	200
Meadows	89.56	200	92.80	87.66	99.36	98.98	99.46	99.60	540	97.20	98.16	97.54	200
Gravel	94.68	200	83.67	99.69	99.69	98.96	99.59	99.45	392	96.97	97.92	96.84	200
Trees	96.48	200	93.85	98.49	99.63	99.75	99.68	99.57	542	98.62	98.74	97.58	200
Painted metal sheets	99.43	200	98.91	100.00	99.95	99.93	99.78	99.61	256	100.00	100.00	99.65	200
Bare Soil	90.75	200	94.17	98.00	99.96	99.42	99.93	99.84	532	98.57	99.57	99.33	200
Bitumen	93.76	200	92.68	99.89	100.00	98.71	99.88	100.00	375	97.27	99.75	98.90	200
Self-Blocking Bricks	63.94	200	89.09	99.70	99.65	98.58	99.53	99.67	514	96.17	98.20	98.89	200
Shadows	99.96	200	97.84	97.11	99.38	99.87	99.79	99.83	231	99.86	99.82	99.58	200
Overall Accuracy (OA)	88.20		92.28	94.04	99.54	98.87	99.47	99.48		96.83	98.06	97.80	
Average Accuracy (AA)	90.39		92.55	97.52	99.66	99.08	99.60	99.57		97.50	98.61	98.29	
Kappa	84.67		90.37	92.43	99.41	98.51	99.30	99.32		95.83	97.44	97.09	
Runtime (sec.	0.15		994.80	607.19	2769.00	43.16	94.57	107.56		42.83	74.09	132.68	
Total samples		1800							3930				1800

Table 13: Classification accuracies obtained by different neural networks tested using the University of Pavia dataset: 1) first column: results obtained by the MLP (trained with 200 samples per class); 2) second column: comparison between the results obtained by the 1-D CNN, 2-D CNN and 3-D CNN in [Chen et al. (2016)] and the results obtained by our CNN (trained with the same number of samples per class as the CNNs in [Chen et al. (2016)]), using different values of parameter d); 3) third column: results obtained by our CNN (trained with 200 samples per class, using different values of parameter d).

column provides a comparison between the CNNs in Chen et al. (2016) and our CNN (using the same number of samples per class as the methods in Chen et al. (2016)), with patch sizes of $d = 15$, $d = 21$ and $d = 23$ (in this case, due to the number of samples per class used by Chen et al. (2016) there is not enough memory to run the CNN with $d = 27$). Finally, in the third column we also report the results obtained by our proposed CNN (trained with 200 samples per class and with patch sizes of $d = 15$, $d = 21$ and $d = 23$) for comparison.

Again, we can see in the second column that our method is faster than all the methods reported in Chen et al. (2016) and comparable in terms of overall accuracy to the best methods reported in that work. Specifically, our proposed CNN implemented with $d = 23$ is is 9.25 times faster than the 1-D CNN with an overall accuracy that is 7.20 percentage points better; 5.65 times faster than the 2-D CNN with an overall accuracy that is 5.44 percentage points better; and 25.74 times faster than the 3-D CNN, with very similar overall accuracy. A comparison between the results in the first and third columns of Table 13 again reveals that our proposed CNN can achieve better results in terms of overall accuracy than the MLP, but the MLP is faster.

In Figure 30 we can graphically compare the overall accuracies obtained

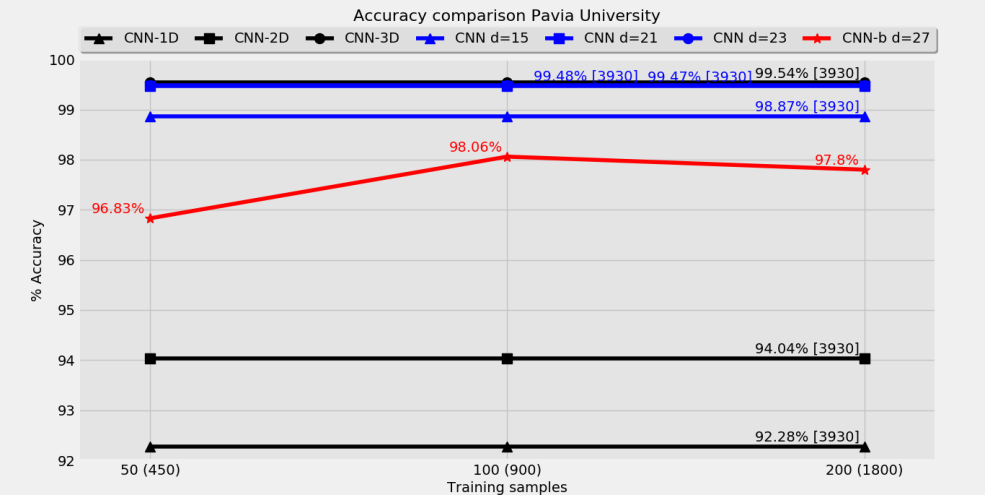


Figure 30: Comparison of the overall accuracy achieved by CNN classifiers with the University of Pavia scene. The horizontal black lines show the overall accuracy results reached by the 1-D, 2-D and 3-D CNNs in Chen et al. (2016), and the horizontal blue lines show the overall accuracy results obtained by our proposed CNN, implemented with different values of d and trained with the same number of samples than the CNNs in Chen et al. (2016). The red line (marked as CNN-b in the figure) corresponds to our CNN, implemented with $d = 29$ but trained with 50, 100 and 200 samples per class. Above each black and blue line, we report the number of used samples and the overall accuracy reached (in square brackets). For the red line, we only report the overall accuracy reached (the number of used samples for this line is defined by the x-axis). On the other hand, the y-axis shows the overall accuracies obtained in the experiments.

by our proposed CNN with those obtained by the 1-D, 2-D and 3-D CNNs reported in Chen et al. (2016). Again, the horizontal blue lines in Figure 30 show the overall accuracy results obtained by our proposed CNN, implemented with different values of d and trained using the same number of samples than the methods in Chen et al. (2016). The red line corresponds to our CNN implemented with $d = 27$ but trained with 50, 100 and 200 samples per class. As we can see, the proposed CNN can reach overall accuracies that are better than those achieved by the 1-D and 2-D CNNs, and comparable to those achieved by the 3-D CNN in Chen et al. (2016). However, the runtime of our implementation is considerably smaller, thanks to our GPU implementation of the network.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have developed a new deep 3-D CNN architecture for spatial-spectral classification of hyperspectral data. The joint consideration of spectral information together with spatial information provides better classification results than those reached by traditional neural networks that only include spectral information. With a proper topology selection and a good election of parameters, we can obtain high classification accuracies in acceptable processing times, enforced by the fact that our CNN has been implemented efficiently using GPUs. Our detailed comparison with other 1-D, 2-D and 3-D CNNs in Chen et al. (2016) (that also include spatial and spectral information simultaneously) reveals a good compromise between the classification results obtained by our newly proposed CNN architecture and the time needed to obtain these results in the considered computing environments, which is important for practical exploitation of the proposed methodology in real applications. Our experiments specifically suggest that, with a proper and simple adaptation, the use of GPUs allows us to realize the full potential of deep learning techniques for remotely sensed hyperspectral image classification by naturally and efficiently combining the spatial and the spectral information contained in these images. This has also been verified with a classic MLP model used for comparative purposes in this work. As future work, we will conduct additional experiments with other hyperspectral scenes and also test other high performance computing architectures for efficient implementation.

Acknowledgement

This work has been supported by 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, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016). This work has also been supported by Junta de Extremadura (decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR15005). Last but not least, the authors

would like to take this opportunity to gratefully thank the Editors and the Anonymous Reviewers for their careful assessment of our manuscript and for their outstanding comments and suggestions, which greatly helped us to improve the technical quality and presentation of our work.

References

Atkinson, P. M., Tatnall, A. R. L., 1997. Introduction Neural networks in remote sensing. International Journal of Remote Sensing 18 (4), 699–709. URL http://dx.doi.org/10.1080/014311697218700

Benediktsson, J. A., Swain, P. H., 1990. Statistical Methods and Neural Network Approaches for Classification of Data from Multiple Sources. Ph.D. thesis, PhD thesis, Purdue Univ., School of Elect. Eng. West Lafayette, IN.

Benediktsson, J. A., Swain, P. H., Ersoy, O. K., 1993. Conjugate gradient neural networks in classification of very high dimensional remote sensing data. International Journal of Remote Sensing 14 (15), 2883–2903.

Bengio, Y., 2009. Learning Deep Architectures for AI. Machine Learning 2 (1), 1–127.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., 2007. Greedy Layer-Wise Training of Deep Networks. In: Schölkopf, B., Platt, J., Hoffman, T. (Eds.), Advances in Neural Information Processing Systems 19 (NIPS'06). MIT Press, pp. 153–160.

Bishop, C. M., 1995. Neural Networks for Pattern Recognition. Clarendon Press.
URL https://books.google.es/books?id=-aAwQO_-rXwC

Böhning, D., 1992. Multinomial logistic regression algorithm. Annals of the Institute of Statistical Mathematics 44 (1), 197–200.

Camps-Valls, G., Bruzzone, L., 2005. Kernel-based methods for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing 43 (6), 1351–1362.

Chang, C.-I., 2003. Hyperspectral Imaging: Techniques for Spectral Detection and Classification. Springer US.

Chen, Y., Jiang, H., Li, C., Jia, X., Ghamisi, P., 2016. Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. IEEE Transactions on Geoscience and Remote Sensing 54 (10), 6232–6251.
URL http://ieeexplore.ieee.org/document/7514991/

Chen, Y., Lin, Z., Zhao, X., Wang, G., Gu, Y., 2014. Deep Learning-Based Classification of Hyperspectral Data. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 7 (6), 2094–2107.

Cheng, G., Han, J., Lu, X., 2017. Remote sensing image scene classification: Benchmark and state of the art. Proceedings of the IEEE PP (99), 1–19.

Chien, Y., 1974. Pattern Classification and Scene Analysis. IEEE Transactions on Automatic Control 19 (4), 462–463.

Cho, K., 2014. PhD thesis: Foundations and advances in deep learning. Ph.D. thesis, Aalto University.

Deng, L., Yu, D., 2014. Deep Learning: Methods and Applications. Foundations and Trends® in Signal Processing 7 (3–4), 197–387.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., Brox, T., 2014. Discriminative Unsupervised Feature Learning with Convolutional Neural Networks. In: Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger (Ed.), Advances in Neural Information Processing Systems 27. Curran Associates, Inc., pp. 766–774.

Duchi, J., Edu, J. B., Hazan, E., Singer, Y., 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization *. Journal of Machine Learning Research 12, 2121–2159.

Fauvel, M., Benediktsson, J. A., Chanussot, J., Sveinsson, J. R., 2008. Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles. IEEE Transactions on Geoscience and Remote Sensing 46 (11), 3804–3814.

Fisher, P., 1997. The pixel: a snare and a delusion. int . j . remote sensing 18 (3), 679–685.

García, V., Sánchez, J. S., Mollineda, R. A., 2011. Classification of high dimensional and imbalanced hyperspectral imagery data. In: Pattern Recognition and Image Analysis: 5th Iberian Conference, IbPRIA 2011, Las Palmas de Gran Canaria, Spain, June 8-10, 2011. Proceedings. Springer Berlin Heidelberg, pp. 644–651.

Ghamisi, P., Plaza, J., Chen, Y., Li, J., Plaza, A., 2017. Advanced Supervised Spectral Classifiers for Hyperspectral Images: A Review. IEEE Geoscience and Remote Sensing Magazine 5 (1), 8–32.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics. pp. 249–256.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep Sparse Rectifier Neural Networks. In: Geoffrey J. Gordon and David B. Dunson (Ed.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11). Journal of Machine Learning Research - Workshop and Conference Proceedings, pp. 315–323.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative Adversarial Nets. In: Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger (Ed.), Advances in Neural Information Processing Systems. Vol. 27. Curran Associates, Inc., pp. 2672–2680.

Green, R. O., Eastwood, M. L., Sarture, C. M., Chrien, T. G., Aronsson, M., Chippendale, B. J., Faust, J. A., Pavri, B. E., Chovit, C. J., Solis, M., Olah, M. R., Williams, O., 1998. Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). Remote Sensing of Environment 65 (3), 227–248.

Haut, J. M., Paoletti, M., Plaza, J., Plaza, A., Jan 2017a. Cloud implementation of the k-means algorithm for hyperspectral image analysis. The Journal of Supercomputing 73 (1), 514–529.
URL https://doi.org/10.1007/s11227-016-1896-3

Haut, J. M., Paoletti, M. E., Paz-Gallardo, A., Plaza, J., Plaza, A., 2017b. Cloud implementation of logistic regression for hyperspectral image classification. In: J. Vigo-Aguiar (Ed.), Proceedings of the 17th International

Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2017. pp. 1063–2321.

He, H., Garcia, E. A., 2009. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering 21 (9), 1263 – 1284.

He, M., Li, X., Zhang, Y., Zhang, J., Wang, W., 2016. Hyperspectral image classification based on deep stacking network. In: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). pp. 3286–3289.

Hinton, G., Salakhutdinov, R., 2006. Reducing the dimensionality of data with neural networks. Science 313, 504–507.

Hinton, G. E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural Comput. 18 (7), 1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. CoRR.
URL `http://arxiv.org/abs/1207.0580`

Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Comput. 9 (8), 1735–1780.

Hu, F., Xia, G.-S., Hu, J., Zhang, L., Foody, G. M., Wang, L., Thenkabail, P. S., 2015a. Transferring Deep Convolutional Neural Networks for the Scene Classification of High-Resolution Remote Sensing Imagery in Surveying, Mapping and Remote Sensing. Remote Sens 7 (11), 14680–14707.
URL `www.mdpi.com/journal/remotesensing`

Hu, W., Huang, Y., Wei, L., Zhang, F., Li, H., 2015b. Deep convolutional neural networks for hyperspectral image classification. Journal of Sensors.

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., Lecun, Y., 2009. What is the Best Multi-Stage Architecture for Object Recognition? In: ICCV. IEEE, pp. 2146–2153.

Karhunen, J., Raiko, T., Cho, K., 2015. Unsupervised Deep Learning: A Short Review.

Khodadadzadeh, M., Li, J., Plaza, A., Ghassemian, H., Bioucas-Dias, J. M., Li, X., 2014. Spectral–Spatial Classification of Hyperspectral Data Using Local and Global Probabilities for Mixed Pixel Characterization. IEEE Transactions on Geoscience and Remote Sensing 52 (10), 6298–6314.

Kingma, D. P., Ba, J. L., 2014. ADAM: {A} method for stochastic optimization. CoRR.
URL http://arxiv.org/abs/1412.6980

Kunkel, B., Blechinger, F., Lutz, R., Doerffer, R., van der Piepen, H., 1988. ROSIS (Reflective Optics System Imaging Spectrometer) - A candidate instrument for polar platform missions. In: Seeley, J., Bowyer, S. (Eds.), Optoelectronic technologies for remote sensing from space. pp. 134–141.

Larochelle, H., Bengio, Y., 2008. Classification using Discriminative Restricted Boltzmann Machines. In: Proceedings of the 25th international conference on Machine learning - ICML '08. p. 536.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep Learning. Nature 521, 436–444.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998a. Gradient-based learning applied to document recognition. Proceedings of the IEEE.

LeCun, Y. A., Bottou, L., Orr, G. B., Müller, K. R., 1998b. Efficient backprop. In: Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop. Springer-Verlag, pp. 9–50.

Li, J., Bioucas-Dias, J. M., Plaza, A., 2010. Semisupervised Hyperspectral Image Segmentation Using Multinomial Logistic Regression With Active Learning. IEEE Transactions on Geoscience and Remote Sensing 48 (11), 4085–4098.

Li, J., Bioucas-Dias, J. M., Plaza, A., 2011. Hyperspectral Image Segmentation Using a New Bayesian Approach With Active Learning. IEEE Transactions on Geoscience and Remote Sensing 49 (10), 3947–3960.

Li, M., Zang, S., Zhang, B., Li, S., Wu, C., 2014a. A review of remote sensing image classification techniques: The role of Spatio-contextual information. European Journal of Remote Sensing 47, 389–411.

Li, T., Zhang, J., Zhang, Y., 2014b. Classification of hyperspectral image based on deep belief networks. In: Image Processing (ICIP), 2014 IEEE International Conference on. pp. 5132–5136.

Li, Y., Zhang, H., Shen, Q., 2017. Spectral–Spatial Classification of Hyperspectral Imagery with 3D Convolutional Neural Network. Remote Sensing 9 (1), 67.

Licciardi, G. A., Del Frate, F., 2011. Pixel unmixing in hyperspectral data by means of neural networks. IEEE Transactions on Geoscience and Remote Sensing 49 (11), 4163–4172.

Liu, B., Yu, X., Zhang, P., Tan, X., Yu, A., Xue, Z., 2017. A semi-supervised convolutional neural network for hyperspectral image classification. Remote Sensing Letters 8 (9), 839–848.
URL http://dx.doi.org/10.1080/2150704X.2017.1331053

Liu, Q., Hang, R., Song, H., Zhu, F., Plaza, J., Plaza, A., 2016. Adaptive Deep Pyramid Matching for Remote Sensing Scene Classification. CoRR abs/1611.03589.
URL http://arxiv.org/abs/1611.03589

Lu, X., Zheng, X., Yuan, Y., 2017. Remote sensing scene classification by unsupervised representation learning. IEEE Transactions on Geoscience and Remote Sensing PP (99), 1–10.

Ma, X., Wang, H., Wang, J., 2016. Semisupervised classification for hyperspectral image based on multi-decision labeling and deep feature learning. ISPRS Journal of Photogrammetry and Remote Sensing 120, 99 – 107.

Melgani, F., Bruzzone, L., 2004. Classification of Hyperspectral Remote Sensing Images With Support Vector Machines. IEEE Transactions on Geoscience and Remote Sensing 42 (8).

Midhun, E. M., Nair, S. R., Prabhakar, N., Kumar, S., 2014. Deep Model for Classification of Hyperspectral image using Restricted Boltzmann Machine. In: Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing. ACM, New York, NY, USA, pp. 35:1–35:7.

Nair, V., Hinton, G. E., 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In: Johannes Fürnkranz and Thorsten Joachims (Ed.), Proceedings of the 27th International Conference on Machine Learning (ICML-10). Omnipress, pp. 807–814.

Paoletti, M. E., Haut, J. M., Plaza, J., Plaza, A., 2017. Yinyang K-means clustering for hyperspectral image analysis. In: J. Vigo-Aguiar (Ed.), Proceedings of the 17th International Conference on Computational and Mathematical Methods in Science and Engineering. Rota, pp. 1625–1636.

Plaza, A., Benediktsson, J. A., Boardman, J. W., Brazile, J., Bruzzone, L., Camps-Valls, G., Chanussot, J., Fauvel, M., Gamba, P., Gualtieri, A., Marconcini, M., Tilton, J. C., Trianni, G., 2009. Recent advances in techniques for hyperspectral image processing. Remote Sensing of Environment 113 (1), S110–S122.

Qian, N., 1999. On the momentum term in gradient descent learning algorithms. Neural Networks 12 (1), 145 – 151.

Rajan, S., Ghosh, J., Crawford, M. M., 2008. An Active Learning Approach to Hyperspectral Data Classification. IEEE Transactions on Geoscience and Remote Sensing 46 (4), 1231–1242.

Ranzato, M. A., Poultney, C., Chopra, S., Lecun, Y., 2006. Efficient Learning of Sparse Representations with an Energy-Based Model. In: B. Schölkopf and J. Platt and T. Hoffman (Ed.), Advances in Neural Information Processing Systems 19. MIT Press, pp. 1137–1144.

Romero, A., Gatta, C., Camps-Valls, G., March 2016. Unsupervised deep feature extraction for remote sensing image classification. IEEE Transactions on Geoscience and Remote Sensing 54 (3), 1349–1362.

Salakhutdinov, R., Hinton, G., 2009. Deep Boltzmann Machines. In: 12th International Conference on Artificial Intelligence and Statistics. p. 3.

Scholkopf, B., Smola, A. J., 2001. Learning with Kernels:Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA.

Smolensky, P., 1986. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: Rumelhart, David E.; McLelland, J. L.

(Ed.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press, Ch. 6, p. 194–281.

Starck, J.-L., Elad, M., Donoho, D. L., 2005. Image Decomposition via the Combination of Sparse Representations and a Variational Approach. IEEE Transactions on Image Processing 14 (10), 1570–1582.

Tarabalka, Y., Benediktsson, J. A., Chanussot, J., 2009. Spectral–Spatial Classification of Hyperspectral Imagery Based on Partitional Clustering Techniques. IEEE Transactions on Geoscience and Remote Sensing 47 (8), 2973–2987.

Vetrivel, A., Gerke, M., Kerle, N., Nex, F., Vosselman, G., 2017. Disaster damage detection through synergistic use of deep learning and 3D point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning. ISPRS Journal of Photogrammetry and Remote Sensing.

Wu, Q., Diao, W., Dou, F., Sun, X., Zheng, X., Fu, K., Zhao, F., 2016. Shape-based object extraction in high-resolution remote-sensing images using deep Boltzmann machine. International Journal of Remote Sensing 37 (24), 6012–6022.

Wu, Z., Wang, Q., Plaza, A., Li, J., Wei, Z., 2015. Real-Time Implementation of the Sparse Multinomial Logistic Regression for Hyperspectral Image Classification on GPUs. IEEE Geoscience and Remote Sensing Letters 12 (7), 1456–1460.

Xu, X., Li, J., Huang, X., Dalla Mura, M., Plaza, A., 2016a. Multiple morphological component analysis based decomposition for remote sensing image classification. IEEE Transactions on Geoscience and Remote Sensing 54 (5), 3083–3102.

Xu, X., Lil, f., Plaza, A., 2016b. Fusion of hyperspectral and LiDAR data using morphological component analysis. In: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). pp. 3575–3578.

Yang, H., 1999. A back-propagation neural network for mineralogical mapping from AVIRIS data. International Journal of Remote Sensing 20 (1), 97–110.
URL http://dx.doi.org/10.1080/014311699213622

Yang, J., Zhao, Y., Chan, J. C. W., Yi, C., 2016. Hyperspectral image classification using two-channel deep convolutional neural network. In: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). pp. 5079–5082.

Yu, S., Jia, S., Xu, C., 1 2017. Convolutional neural networks for hyperspectral image classification. Neurocomputing 219, 88–98.

Yuan, Y., Mou, L., Lu, X., 2015. Scene recognition by manifold regularized deep learning architecture. IEEE Transactions on Neural Networks and Learning Systems 26 (10), 2222–2233.

Zeiler, M. D., 2012. ADADELTA: An Adaptive Learning Rate Method. CoRR abs/1212.5701.

Zeiler, M. D., Krishnan, D., Taylor, G. W., Fergus, R., June 2010. Deconvolutional networks. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 2528–2535.

Zhang, H., Li, Y., Zhang, Y., Shen, Q., 2017. Spectral-spatial classification of hyperspectral imagery using a dual-channel convolutional neural network. Remote Sensing Letters 8 (5), 438–447.

Zhang, L., Zhang, L., Du, B., 2016a. Deep Learning for Remote Sensing Data Advances in Machine Learning for Remote Sensing and Geosciences. IEEE Geoscience and Remote Sensing Magazine 4 (2), 22–40.
URL http://ieeexplore.ieee.org/document/7486259/

Zhang, P., Gong, M., Su, L., Liu, J., Li, Z., 2016b. Change detection based on deep feature representation and mapping transformation for multi-spatial-resolution remote sensing images. ISPRS Journal of Photogrammetry and Remote Sensing 116, 24–41.

Zhao, W., Du, S., 2016a. Learning multiscale and deep representations for classifying remotely sensed imagery. ISPRS Journal of Photogrammetry and Remote Sensing 128, 223 – 239.

Zhao, W., Du, S., 2016b. Spectral-Spatial Feature Extraction for Hyperspectral Image Classification: A Dimension Reduction and Deep Learning Approach. IEEE Transactions on Geoscience and Remote Sensing 54 (8), 4544–4554.

Zhou, X., Prasad, S., 2017. Active and Semisupervised Learning With Morphological Component Analysis for Hyperspectral Image Classification. IEEE Geoscience and Remote Sensing Letters 14 (8), 1348–1352.