*Proceedings of the 17th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2017
4–8 July, 2017.*

GPU Classification for Hyperspectral Images based on Convolutional Neural Networks

Alberto S. Garea[1], Dora B. Heras[1] and Francisco Argüello[2]

[1] *Centro singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela*

[2] *Departamento de Electrónica y Computación, Universidade de Santiago de Compostela*

emails: `jorge.suarez.garea@usc.es`, `dora.blanco@usc.es`, `francisco.arguello@usc.es`

Abstract

Recently, deep learning techniques based on Convolutional Neural Networks (CNN) have started to be used for the classification of hyperspectral images. These techniques present high computational cost when preprocessing stages are applied. In this paper, a GPU (Graphical Processor Unit) implementation of a spatial-spectral supervised classification scheme based on CNNs and applied to remote sensing datasets is presented. The scheme comprises convolution filters for processing the spectral information and a patch around each pixel to take the spatial information into account. To reduce the size of the filters, the dimensionality of the dataset is previously reduced using Principal Component Analysis (PCA). In order to achieve an efficient GPU projection, different techniques and optimizations have been applied such as the use of the deep learning framework Caffe. Speedups of up to 38.66× over the Pavia University dataset are obtained together with competitive classification accuracies.

Key words: Hyperspectral, Classification, Convolutional neural network, Deep learning, Caffe, GPU.

1 Introduction

Hyperspectral images contain a large amount of information that can be exploited during the processing. This information is not only spectral but there is also a lot of spatial information in the neighborhood of each pixel. Hyperspectral techniques that can exploit

both types of information are known by the name of spectral-spatial techniques [1]. When these techniques are introduced in the classification of hyperspectral images, experimental results show great improvements in the accuracy results.

Recently, deep-learning techniques have started to be introduced in the field of classification of hyperspectral datasets [2, 3, 4, 5, 6]. These classifiers consist of several layers with nonlinear processing units to extract and transform different features. Each layer uses the output of the previous layer as input and the network can be trained in a supervised or unsupervised manner. Applications include pattern recognition and statistical classification. The proposed methods extract spatial information using structures such as Multilayer Perceptrons (MLP) or Convolutional Neural Networks (CNN). Usually before the extraction of spatial information, a dimensionality reduction is performed using techniques such as Principal Component Analysis (PCA), Independent Component Analysis (ICA) or wavelets in order to obtain moderately small vectors.

A CNN contains convolutional layers that can be used to perform spatial convolutions on the hyperspectral image bands. Usually pooling layers are also included in order to apply some kind of decimation and reduce the number of coefficients. A CNN may have one or more convolutional layers, but the final classification is performed using one or more fully-connected layers. Activation functions to introduce non-linearity, usually of sigmoid type, can be included in convolutional layers. Such functions are similar to those used in MLPs. Usually the backpropagation algorithm is used to set the coefficients of both, neurons of fully-connected layers and convolution filters.

Some published deep-learning schemes applied to hyperspectral images use only the spectral information. Thus, Hu *et. al* [4] propose a scheme based on CNNs, which does not consider spatial information since each input is a single pixel-vector. Other schemes incorporate the spectral and spatial information separately to the classifier, often constructing a stack-vector for input to the neural network and using PCA [2, 3, 6, 5].

Remote sensing hyperspectral applications are computationally demanding and, therefore, good candidates to be projected in high performance computing infrastructures such as clusters or specialized hardware devices [7]. GPUs provide a cost-efficient solution to carry out onboard real-time processing of remote sensing hyperspectral data for performing hyperspectral unmixing, classification or change detection, among others [8]. In the case of deep learning techniques, different high-level frameworks optimized for GPU computing are available, such as Theano, Caffe [9], TensorFlow or Torch. The implementations of deep learning methods for hyperspectral images are in some cases presented in terms of execution times but without an analysis of the computational cost [10]. In other cases the use of an optimized framework such as Caffe [11] is mentioned but without including execution times or a detailed analysis of the implementations.

In this paper we propose a CUDA GPU spectral-spatial classification scheme for hyperspectral images based on CNNs and implemented mainly by using the Caffe analyzing

 ISBN: 978-84-617-8694-7

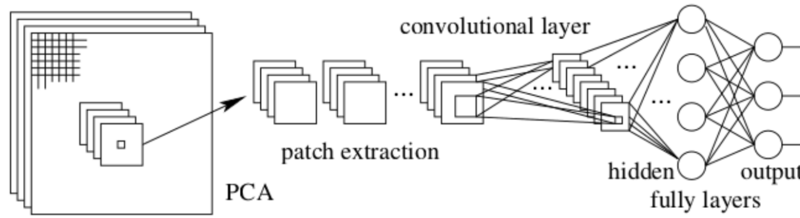


Figure 1: HYCNN scheme for the classification of hyperspectral images.

the details of the implementation. In order to reduce the size of the convolution filters, the image dimensionality is previously reduced using PCA, as it will be explained in the next section.

The paper is organized as follows: section 2 presents the proposed spectral-spatial classification scheme in CPU, section 3 presents the GPU code. The evaluation is performed in section 4, and, finally, section 5 presents the conclusions.

2 Spectral-Spatial CNN-Based Classification

In this section we present a scheme for the classification of hyperspectral images based on PCA, patch extraction, and CNNs, that we called HYCNN. Fig. 1 shows the operations performed and the network structure. These are described in more detail in the pseudocode of Algorithm 1, which also indicates the adjustable parameters in the scheme.

Algorithm 1 Steps of the HYCNN scheme

Input: Hyperspectral image
Output: Classification map
Parameters:
N_1: number of principal components
$H \times V$: patch size
N_2: number of convolution filters
$F_1 \times F_2$: spatial size of filters
$D_1 \times D_2$: decimation factor
N_3: number of neurons in hidden layer
η: learning parameter

1. Preprocessing
1.1 PCA on the image
2. Patch extraction
2.1 Patch around each pixel
3. Convolutional layer
3.1 Convolution filtering
3.2 Pooling (average)
3.3 Activation function (sigmoid)
4. Fully-connected layers
4.1 Hidden layer (with sigmoid)
4.2 Output layer (with sigmoid)

As a first step, HYCNN performs a reduction of image dimensionality using PCA. It extracts the most significant information from the hyperspectral image in the spectral dimension and reduces the number of components, which progress to the next step of the algorithm.

 ISBN: 978-84-617-8694-7

A patch is then extracted around each pixel to be classified. This step aims at getting the spatial information in the neighborhood of a pixel in addition to the spectral information. Accordingly, the patch has the same number of components as those retained from the PCA, and comprises a window around the pixel. The window size is an adjustable parameter of the classification. Each patch is considered a sample and used as the unit of information during the training and classification phases by the CNN.

The next step is the processing of each patch by the CNN. This consists of three parts: convolutional filters, pooling layer and activation function. A convolutional layer is a locally connected structure which is convolved with the image to produce several feature maps, one for each filter. Each filter consists of a rectangular grid of neurons. Unlike a fully-connected layer, the filter coefficients used in all the nodes are the same.

The convolutional layer of our scheme processes several components (spectral bands). The inputs to the filters are the patches, which we assume to have in this sequential algorithm a size of $H \times V \times N_1$, being H and V the size of the spatial dimensions, and N_1 the number of bands. In order to extract multiple features, the convolutional layer comprises N_2 filters, so we will have this same number of maps (planes) at the output. Regarding the size of the filters, if $F_1 \times F_2$ is the size of the spatial grid, each filter will have $F_1 \times F_2 \times N_1$ coefficients.

The pooling layer takes small rectangular blocks from the convolutional layer and subsamples them to produce a single output from each block. For the pooling layers each map is subsampled with mean pooling over blocks of size $D_1 \times D_2$. After the subsampling, a sigmoidal nonlinearity is applied to each feature map.

The last part of the scheme consists of fully-connected layers, which perform the high-level reasoning of the CNN. A fully connected layer takes all the outputs in the previous layer and connects then to every single neuron it has. This type of layer is arranged in one dimension, so they are not spatially located operations anymore. In this paper we use the typical two-layer MLP, with hidden and output layers. The number of neurons in the hidden layer is the adjustable parameter N_3, while the output layer has a number of neurons equal to the number of classes in the hyperspectral image. The activation function in both, convolutional and fully-connected layers, is of sigmoid type.

The learning of all the layers of the CNN in this scheme is conducted using a backpropagation algorithm. The error is computed at the output of the network using the training samples and comparing the results to the reference data. Then, the error is propagated backwards through the network. The backpropagation is used in conjunction with an optimization method, in this case a gradient descent. It calculates the gradient of a cost function with respect to all the weights of the network, and then updates the weights in an attempt to minimize the cost function. The learning parameter, usually denoted as η, indicates how much the weights are adjusted at each update.

3 Spectral-Spatial CNN-Based Classification in GPU

In this section we introduce some Compute Unified Device Architecture (CUDA) programming fundamentals as well as the CUDA GPU implementation of the scheme proposed in Sect. 2.

3.1 CUDA GPU programming fundamentals

CUDA is a parallel computing platform and programming model that enables NVIDIA GPUs to execute programs invoking parallel functions called kernels [12]. Each kernel launches a user-defined number of threads that are organized into blocks. The blocks are arranged in a grid that is mapped to a hierarchy of CUDA cores in the GPU. Threads can access data from multiple memory spaces. Each block has a shared memory that is visible exclusively to the threads within this block and whose lifetime is equal to the block lifetime. The shared memory lifetime makes it difficult to share data among thread blocks. This implies the use of global memory whose access is slower than shared memory access. The new Pascal architecture has introduced changes regarding the memory hierarchy [13].

Different performance optimization strategies have been applied in this work. The most important is to reduce the data transfers between the CPU and the GPU memories. Another key is to improve the efficiency in the use of the memory hierarchy by performing the maximum number of computations on the data already stored in shared memory. The search for the best kernel configurations is also fundamental. To get the highest possible occupancy is the only way to hide latencies and keep the hardware busy. To achieve this, the maximum block size for each kernel is selected with the requirement that the number of registers and the shared memory usage do not act as occupancy limiters. Finally, the existing CUDA optimized libraries must be used. CULA [14], MAGMA [15], and CUBLAS [16] are used for algebra operations. For the deep learning calculations the Caffe framework is used. It performs calls to CuDNN [17], CUBLAS and MAGMA. CuDNN is a GPU-accelerated library for deep neural networks.

3.2 CUDA implementation

In this section the GPU implementation of the HYCNN algorithm described in section 2 is detailed. The pseudocode in Algorithm 2 shows a detailed description of the classification scheme. The kernels executed in GPU are placed between <> symbols. The pseudocodes also include the GM and SM acronyms to indicate kernels executed only in global memory and kernels that only use shared memory, respectively. The whole forward-backward process for the training phase of the algorithm is detailed. The CNN is implemented using Caffe. Since the calls to Caffe functions produce a high number of calls to libraries, these are grouped in the pseudocode by steps of the scheme and only the most repeated kernels are

 ISBN: 978-84-617-8694-7

included pointing out the call sequence.

Algorithm 2 HYCNN classifier for hyperspectral images (GPU) → Training step

Input: Hyperspectral image

GPU EVD-PCA algorithm

1: **for** each epoch **do**
<u>**Forward**</u>
Convolution filtering
2: **for** each training sample **do**
3: im2col_gpu() → <im2col_gpu> ▷ GM
4: caffe_gpu_gemm() → cublasSgemm() → <gemmSN_NN>, <gemmK1> ▷ SM + GM
5: **end for**
Average Pooling
6: PoolingLayer::Forward_gpu() → <AvePoolForward> ▷ GM
Convolution Activation
7: CuDNNSigmoidLayer::Forward_gpu() → cudnnActivationForward() → <activation_fw_4d> ▷ GM
First Inner
8: InnerProductLayer::Forward_gpu() → caffe_gpu_gemm() → <sgemm_largeK>, <gemmk1> ▷ SM + GM
First Inner activation
9: CuDNNSigmoidLayer::Forward_gpu() → cudnnActivationForward() → <activation_fw_4d> ▷ GM
Second Inner
10: InnerProductLayer::Forward_gpu() → caffe_gpu_gemm() → <sgemm>, <gemmk1> ▷ SM + GM
Second Inner Activation
11: CuDNNSigmoidLayer::Forward_gpu() → cudnnActivationForward() → <activation_fw_4d> ▷ GM
SoftMax with Loss
12: CuDNNSoftmaxLayer::Forward_gpu() → cudnnSoftmaxForward() → <softmax_fw> ▷ SM + GM
13: SoftmaxLossForwardGPU() → <SoftmaxLossForwardGPU>, <cublasSasum> ▷ GM

<u>**Backward**</u>
14: SoftmaxLossBackwardGPU() → <SoftmaxLossBackwardGPU>, <cublasSscal> ▷ GM
Second Inner Activation
15: CuDNNSigmoidLayer::Backward_gpu() → cudnnActivationBackward() → <activation_bw_4d> ▷ GM
Second Inner
16: InnerProductLayer::Backward_gpu() → <sgemmNT2>, <gemmv2N>, <sgemm_128x64> ▷ SM + GM
First Inner Activation
17: CuDNNSigmoidLayer::Backward_gpu() → cudnnActivationBackward() → <activation_bw_4d> ▷ GM
First Inner
18: InnerProductLayer::Backward_gpu() → <sgemm_128x64>, <gemmv2N>, <sgemm_128x64> ▷ SM + GM
Convolution Activation
19: CuDNNSigmoidLayer::Backward_gpu() → cudnnActivationBackward() → <activation_bw_4d> ▷ GM
Pooling
20: PoolingLayer::Backward_gpu() → <AvePoolBackward> ▷ GM
Convolution filtering
21: **for** each training sample **do**
22: ConvolutionLayer::Backward_gpu():
23: backward_gpu_bias() → <gemv2T> ▷ SM + GM
24: weight_gpu_gemm() → <im2col_gpu> ▷ GM
25: backward_gpu_gemm() → <gemmSN_TN> ▷ SM + GM
26: **end for**
Weights update
27: caffe::SGDSolver() → <SGDUpdate> ▷ GM
28: **end for**

 ISBN: 978-84-617-8694-7

As a first step, the PCA algorithm using EVD (EVD-PCA) is applied to reduce the dimensionality of the dataset. For details of the GPU implementation see [18].

A patch is then extracted around each pixel and stored into a two different Lightning Memory-Mapped Databases (LMDBs) to be accessed from the Caffe framework. The first database stores the training patches whereas the second one stores the test patches. As shown in the pseudocode, both, the CNN steps and the fully-connected layers steps are applied to each patch N times (epochs).

The training phase is divided into two main steps: forward and backward. The forward step computes all the training patches through the full network to obtain a classification result and the backward step updates the network weights to adjust the obtained classification result.

The forward step starts applying the convolution filters to each training patch. Unlike in the CPU version where the $H \times V$ pixels of the patch are computed through the convolution filters sequentially, in the GPU version the patch is converted first into a matrix using the im2col_gpu function (line 3 in the pseudocode 2). Then, it is multiplied by a matrix containing the convolutional values using the cublasSgemm function (line 4).

Next, a pooling substep is performed using a Caffe kernel called AvePoolForward. This function computes the pooling over all the training patches at the same time. The last substep of the CNN is the activation. The CuDNNSigmoidLayer::Forward_gpu() calls the CuDNN funtion to perform the sigmoid activation (line 7).

Once the CNN has finished, two fully-connected layers perform the classification. First, an inner product function (line 8) that uses CuBLAS multiplies the CNN output matrix by a matrix of learned weights. Next, a sigmoid activation function (line 9) is applied over the previous result using a CuDNN function. The previous two operations are repeated over the last fully-connected layer (lines 10 and 11).

At this point, the output of the full network contains the classification of each training patch. Then, a softmax function (line 12) is applied to get a probability distribution over classes. This function takes a vector of arbitrary real-valued scores and converts it to a vector of values between zero and one that sum one. The last substep of the forward is to compute the loss of the network using the function SoftmaxLossForwardGPU (line 13).

Regarding the backward step, it includes all the substeps of the forward step but applied in reverse order (lines 14-26). This allows to update the values of all the neurons in the full network based on the results of the loss function computed in the forward step. At the end of the loop, the update of all the weights of the full network is performed (line 27). This task is carried out by a Caffe function.

 ISBN: 978-84-617-8694-7

4 Results

This section shows the experimental results obtained for the GPU HYCNN scheme comparing to the CPU scheme in terms of computation time and classification accuracy.

The proposed algorithms have been evaluated on a PC with a quad-core Intel i5-6600 at 3.3GHz and 32 GB of RAM. The codes have been compiled using the gcc 4.8.4 version with OpenMP (OMP) 3.0 support under Linux using four threads. The OPENBLAS library has been used to accelerate the algebra operations included in the algorithms. Regarding the GPU implementation, CUDA codes run on an Pascal NVIDIA GeForce GTX 1070 with 15 Streaming Multiprocesors (SMs) and 128 CUDA cores each. The CUDA codes have been compiled under Linux using the nvcc version 8.0.26 of the toolkit. As usual in remote sensing [19], measures of classification accuracy are given in terms of overall accuracy (OA), which is the percentage of correctly classified pixels comparing to the reference data information available. The computational performance results are expressed in terms of execution times and speedups. The results are the average of 10 independent executions.

The algorithms have been used over two remote sensing datasets: a 103-band ROSIS image of the University of Pavia (Pavia Univ.) and a 220-band AVIRIS image taken over Northwest Indiana (Indian Pines). The images and the corresponding reference data are shown in Fig. 2.

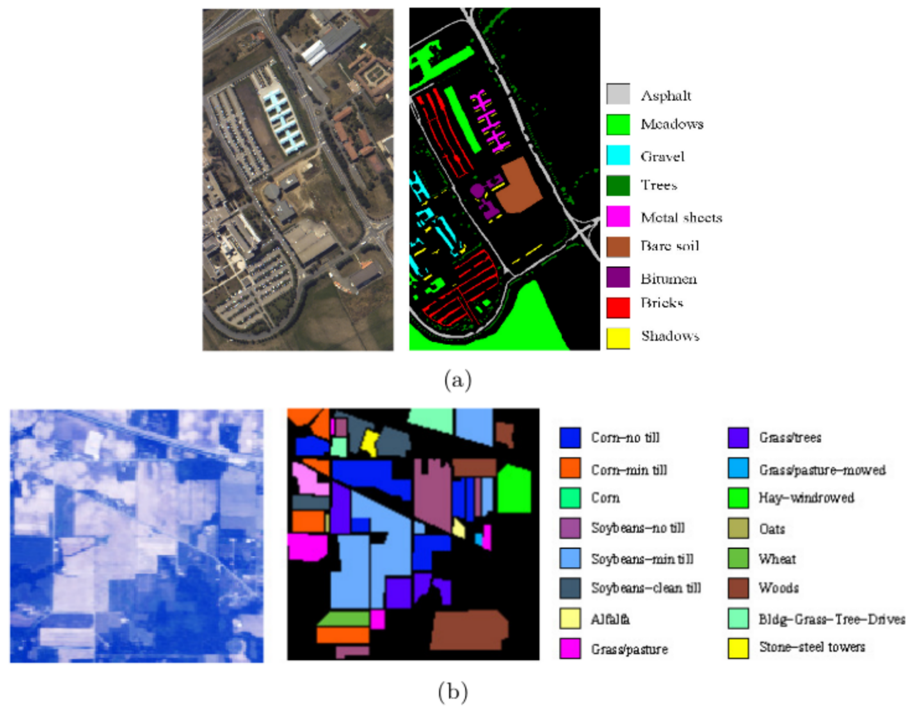
For each dataset the samples are randomly distributed between the training [18] and testing sets. During the testing stage all pixels of the image are classified, but the samples used in the training stage are excluded for the calculation of the accuracy results (see Table 1).

Datasets	Sensor	classes	Dimensions	samples	training samples
Pavia Univ.	ROSIS	9	610×340×103	42776	3921 (9.17%)
Indian Pines	AVIRIS	16	145×145×220	10249	695 (6.78%)

Table 1: Information for the test remote sensing datasets.

The configuration parameters were determined by performing experiments varying the number of principal components, the batch size and the filter size for the code executed in CPU. These parameters are also used for the GPU implementation when both codes are compared from the computational point of view. The base parameters considered for the comparison are $H = V = 28$ (patch size), $N_1 = 4$ (number of principal components), $N_2 = 16$ (number of filters), $N_3 = 100$ (neurons in the hidden layer), $F_1 = F_2 = 5$ (filter size), and $D_1 = D_2 = 2$ (decimation factor). The backpropagation algorithm was performed with learning parameter $\eta = 0.2$, batch size equal to the number of training samples, and a total of 200 epochs. The Caffe framework using a block size of 512 is used to execute the GPU version of the HYCNN scheme except for the initial PCA algorithm.

 ISBN: 978-84-617-8694-7



(a)

(b)

Figure 2: Hyperspectral datasets: (a) *Pavia Univ.*, (b) *Indian Pines*.

Table 2 shows the comparison between the CPU and the GPU implementations of the scheme when it is applied to the Pavia Univ. image. For a better understanding of the results, different parts of the code have been grouped into higher abstraction level functions. The times are split following the functions of the pseudocode in Algorithm 2 but aggregating the results for the backward step. The speedups are calculated as the number of times that the GPU code is faster than the CPU code. The biggest speedup is observed for the First Inner function that comprises the update of the hidden layer neurons in the fully-connected network. The speedup for the Second Inner is lower because the size of the matrix by matrix multiplication is smaller as it corresponds to layers with fewer neurons. The most time consuming function is the Convolution in the Forward step. Its speedup is only $47.41\times$ because this function includes a group of kernels with low occupancy.

Table 3 shows the execution times and speedups for the whole classification scheme for the two test datasets (including the training and testing steps and also the PCA step) as well as the classification accuracies for both implementations. It is important to enhance that the same configuration parameters were used for both implementations in order to compare the computational time in the same conditions. Nevertheless, for the GPU accuracies the

 ISBN: 978-84-617-8694-7

Step	Lines	CPU	GPU	Speedup
Forward step				
Convolution	2-5	2.06537s	0.04356s	47.41×
Average Pooling	6	0.03557s	0.00556s	6.40×
Convolution Act.	7	0.26165s	0.01389s	18.83×
First Inner	8	1.92316s	0.00225s	854.74×
First Inner Act.	9	0.01281s	0.00063s	20.33×
Second Inner	10	0.00621s	0.00006s	103.50×
Second Inner Act.	11	0.00126s	0.00005s	25.20×
Loss	12-13	0.00023s	0.00015s	1.53×
Backward step				
Second Inner	14-16	0.00350s	0.00007s	50.00×
First Inner	17-18	0.95480s	0.00537s	177.80×
Convolution	19-26	1.61309s	0.04792s	33.66×
Total		6.87742s	0.11939s	57.60×

Table 2: CPU and GPU execution times and speedups for the training step of the HYCNN scheme for Pavia Univ. dataset. The column Lines shows the lines in Algorithm 2

	CPU		GPU		Speedup
Dataset	Time	Accuracy (%)	Time	Accuracy (%)	
Pavia Univ.	1404.26s	98.50	36.32s	97.15	38.66×
Indian Pines	252.33s	97.14	7.60s	84.84	33.20×

Table 3: Execution times,speedups and classification accuracies for the HYCNN scheme.

parameters were optimized for the GPU code separately. The patch size was reduced to 256. In addition, the number of epochs was set to 1300 and 3683 for the Pavia Univ. and the Indian Pines images respectively. The differences in classification accuracy among the CPU and the GPU schemes are produced by the weights update during the backpropagation. For the CPU case the update is carried out for each sample separately, on the contrary for the GPU case the updates are performed by blocks of samples.

5 Conclusions

In this paper we propose a spectral-spatial scheme in GPU based on convolutional neural networks for the classification of hyperspectral images and evaluate its results on several public datasets used in remote sensing for land-cover applications. The scheme consists of principal component analysis, patch extraction, convolution filters and fully-connected layers. The learning is performed using the standard backpropagation algorithm.

The CUDA GPU implementation is based on the use of the Caffe optimized framework for deep learning and other optimization strategies including calls to optimized libraries such as CULA, CUBLAS and MAGMA. Details on the Caffe implementation are given. The experiments obtain speedups of up to 38.66% for the Pavia Univ. dataset with accuracies of up to 97.15%.

Acknowledgments

This work was supported in part by the Consellería de Cultura, Educación e Ordenación Universitaria [grant numbers GRC2014/008 and ED431G/08] and Ministry of Education, Culture and Sport, Government of Spain [grant numbers TIN2013-41129-P and TIN2016-76373-P]. These grants are co-funded by the European Regional Development Fund (ERDF).

References

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

[2] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu (2014), *Deep learning-based classification of hyperspectral data*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 7(6), 2094-2107.

[3] J. Yue, W. Zhao, S. Mao, and H. Liu (2015), *Spectral-spatial classification of hyperspectral images using deep convolutional neural networks* Remote Sensing Letters, 6(6), 468-477.

[4] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li (2015), *Deep Convolutional Neural Networks for Hyperspectral Image Classification*, Journal of Sensors, 2015, 258619.

[5] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis (2015), *Deep supervised learning for hyperspectral data classification through convolutional neural networks*, Proc. IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS), 4959-4962.

[6] W. Zhao, Z. Guo, J. Yue, X. Zhang, and L. Luo (2015), On combining multiscale deep learning features for the classification of hyperspectral remote sensing imagery, International Journal of Remote Sensing, 36(13), 3368-3379.

[7] E. Christophe, J. Michel, and J. Inglada, *Remote sensing processing: From multicore to GPU*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 4, no. 3, pp. 643-652, 2011

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

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, *Caffe: Convolutional Architecture for Fast Feature Embedding*, arXiv preprint arXiv:1408.5093, 2014

[10] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, *Deep feature extraction and classification of hyperspectral images based on convolutional neural networks*, IEEE Transactions on Geoscience and Remote Sensing, vol. 54, no. 10, pp. 6232-6251, 2016

[11] E. Aptoula, M.C. Ozdemir, and B. Yanikoglu, *Deep Learning With Attribute Profiles for Hyperspectral Image Classification*, IEEE Geoscience and Remote Sensing Letters, vol. 13, no. 12, pp. 1970-1974, 2016

[12] David B. Kirk and Wen-mei W. Hwu, *Programming Massively Parallel Processors A Hands-on Approach*, Morgan Kaufmann, 2016.

[13] Nvidia, *Whitepaper: NVIDIA Tesla P100 (2017)*, Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, accessed: December 2, 2016

[14] Nvidia, *CULA Tools (2015)*, Available: http://www.culatools.com/, accessed: January 13, 2015

[15] MAGMA, *Matrix Algebra on GPU and Multicore Architectures (2015)*, Available: http://icl.cs.utk.edu/projectsfiles/magma/doxygen/, accessed: January 13, 2017

[16] Nvidia, *CUDA Toolkit Documentation: CUBLAS (2015)*, Available: http://docs.nvidia.com/cuda/cublas/index.html, accessed: January 11, 2017

[17] Nvidia, *CuDNN*, Available: https://developer.nvidia.com/cudnn, accessed: March 22, 2017

[18] A. S. Garea, D. B. Heras, and F. Argüello, *GPU classification of remote sensing images using kernel ELM and extended morphological profiles*, International Journal of Remote Sensing, vol. 37, no. 24, pp. 5918-5935, 2016

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

 ISBN: 978-84-617-8694-7