MULTICORE IMPLEMENTATION OF THE MULTI-SCALE ADAPTIVE DEEP PYRAMID MATCHING MODEL FOR REMOTELY SENSED IMAGE CLASSIFICATION

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

Hyperspectral Computing Laboratory
Department of Technology of Computers
and Communications. University of Extremadura,
Caceres, Spain

Q. Liu, R. Hang †

Jiangsu Key Laboratory of
Big Data Analysis Technology
Nanjing University of
Information Science and Technology

ABSTRACT

Artificial neural networks (ANNs) have been widely used in the analysis of remotely sensed imagery. In particular, convolutional neural networks (CNNs) are gaining more and more attention. Unlike traditional CNNs methods, where the relevant information to classify the elements of a remotely sensed image is extracted only from the last fully-connected layer, the new adaptive deep pyramid matching (ADPM) model [1] takes advantage of the features from all of the convolutional layers. This model allows the optimal fusing weights for different convolutional layers be learned from the data itself. In addition, the combination of CNNs with spatial pyramid pooling (SPP-net) to create the basic deep network allows the use of images with multiple scales, which results in better learning process thanks to the complementary information. The original ADPM method is divided in two parts: the multi-scale deep feature extraction and the ADPM core. In this paper we present a computational improvement of the ADPM core, coding a parallel-multicore version. This strategy is shown to significantly enhance performance in the analysis of remotely sensed data.

Index Terms— Convolutional neural network (CNN), adaptive deep pyramid matching (ADPM), convolutional features, multi-core implementation, image classification.

*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).

†

1. INTRODUCTION

Convolutional neural networks (CNNs) have proved to be very useful in the processing of sensed images. However, there still exist some issues to be solved in CNN-based remote sensing scene classification. First of all, most of methods only consider the information of the last fully-connected layers as features for subsequent processing and classification. This way, the information that convolutional layers offer is being disregarded. Information is usually more generic, more suitable for transfer learning and contains more spatial information (that improve the classification) than fully-connected features [2, 3, 4, 5, 6].

Another point is that the objects of study normally have different scales in different remotely sensed images, and even a single scene may contain objects with different sizes while CNNs usually require a fixed input image size. The fastest solution is to resize the image to the appropriate size, but this inevitably causes the loss of effective discriminant information. The multi-scale Adaptive Deep Pyramid Matching (ADPM) model [1] is able to solve these two problems.

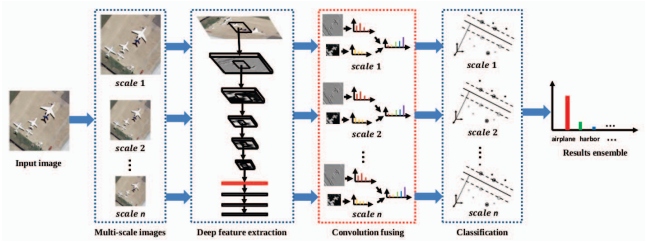

Fig. 1: Flowchart of the proposed method.

As can be seen in Fig. 1, ADPM procedure consists of four main steps: 1) An input image is deformed into different scales; 2) the resulting multi-scale images are fed into the SPP-net to extract multiscale deep features; 3) for each scale, ADPM model is employed to fuse the extracted representations from all of the convolutional layers, and 4)

the learned representation is fed into a support vector machine (SVM) to obtain a final classification result. To integrate the multiple results of all scales a majority voting strategy is used. We can divide the architecture in two parts:

- The multi-scale deep feature extraction, composed of a CNN that contains five successive convolutional layers, an SPP layer, two fully-connected layers, and a softmax layer.
- The ADPM core, which is responsible for fusing optimally each feature of the five convolutional layers. This work focuses on the computational improvement of this part of the architecture, through the elaboration of a multicore parallel version.

2. OPERATION OF THE ADPM CORE

For an input image I_1, each convolutional layer ($l = 1, 2, 3, 4, 5$) calculates its output or feature map, as a three-dimensional matrix, $F_{1,l} \in R^{n_l \times n_l \times p_l}$. Each location (i, j) of $F_{1,l}$, denoted as $f_{1,l}^{(i,j)}$, with $1 \leq i \leq n_l$ and $1 \leq j \leq n_l$, defines a p_l-dimensional representation for a local patch of the image I_1.

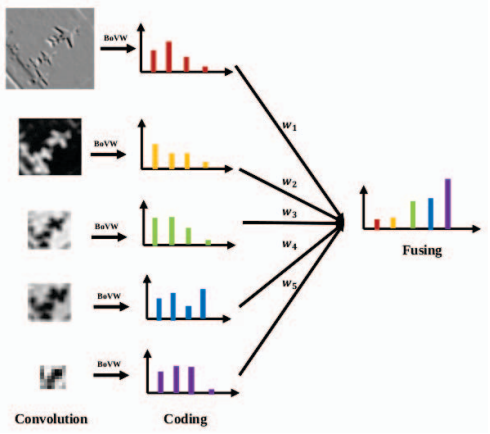

Fig. 2: Flowchart of ADPM core.

As we can see in Fig. 2, the ADPM uses the K-means method to cluster the $n_l \times n_l$ features into $D = 300$ centers $c_l^1, ..., c_l^D$ in order to form the visual codebook with 300 code words similar to BoVW [7, 8, 9]. Each $f_{1,l}^{(i,j)}$ is assigned to its nearest visual word c_l^d, with $1 \leq d \leq D$, allowing to generate a histogram from $F_{1,l}$:

$$H_{1,l} = \left[\sum_{i,j} \delta \left(f_{1,l}^{(i,j)}, c_l^1 \right), ..., \sum_{i,j} \delta \left(f_{1,l}^{(i,j)}, c_l^D \right) \right] \quad (1)$$

Where $\delta \left(f_{1,l}^{(i,j)}, c_l^d \right)$ is a function that indicates that c_l^d is the nearest visual word to $f_{1,l}^{(i,j)}$ when its value is 1 or the most distant one when its value is 0. The next step of the ADPM is to calculate the kernel matrix K of training data as:

$$K(I_1, I_2) = \sum_{l=1}^{5} \omega_l K_l(I_1, I_2)$$

where I_1 and I_2 are two images, ω_l is the weights of each convolutional layer and K_l is the histogram intersection:

$$K_l(I_1, I_2) = \sum_{d}^{D} min \left(\sum_{i,j} \delta \left(f_{1,l}^{(i,j)}, c_l^d \right), \sum_{i,j} \delta \left(f_{2,l}^{(i,j)}, c_l^D \right) \right)$$

Next to the kernel K, the ideal kernel matrix, Y, is calculated using the label information for training images, such that the objective function is:

$$\min_{w} \parallel K - Y \parallel_F^2 + \lambda \parallel w \parallel_2^2 \quad (2)$$

where $\parallel K - Y \parallel_F^2$ represents the element-wise sum between matrices K and Y and w is the optimal weight. The parameter λ is set to 0.5 empirically. Finally, the kernel K is fed to an SVM for classification, whose results derived from all scales are integrated via the majority voting method to achieve the final results.

3. PARALLEL MULTICORE VERSION OF ADPM

In order to reducee the computational cost of the ADPM method, we have introduce several changes in the original algorithm. First, the original Matlab code has been fully translated into Python, which has allowed an improvement of the execution time thanks to the use of scientific computing libraries such as *Numpy* and *Scipy* [10].

The original method of K-means has been replaced by the version implemented in the Python machine learning library *Scikit-learn* [11], allowing two modes of execution: the normal K-means with the initialization of K-means++ [12], and the *Mini-batch K-means* [13]. This last is a variant of the K-means algorithm which uses mini-batches[1] thus reducing the computation time. Mini-batch K-means converges faster than K-means, but the quality of the results is slightly worse than the standard algorithm. In addition, K-means++ initialization is offered on the Mini-batch K-means, in order to improve the process convergence.

On the other hand, the calculation of the histograms has been parallelized using the threads of *PyQt*, the Python bindings for the Qt application development framework.

For the calculation of the optimal weights, w, the Python package for convex optimization, *cvxopt*, has been used. The main purpose of using this package is the easy and simple

[1] Each mini-batch is a subset of the input data, randomly sampled in each training iteration

resolution of quadratic problems like:

$$\min_x x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub. \end{cases} \quad (3)$$

Equations (2) and (3) are similar since Equation (2) can be transcribed as $min_w w^T(A + \lambda I)w - 2b^T w$ such that:

$$\begin{cases} \omega_l \geq 0, \\ l = 1, 2, 3, 4, 5 \\ \sum_{l=1}^{5} \omega_l = 1. \end{cases} \text{ where each } A_{i,j} \text{ contains } tr(K_i^T K_j),$$

b is a vector where each b_i is $tr(Y^T K_j)$ and I is the matrix identity.

Finally, the SVM has been coded using the *LIBSVM* [14] for Python. The training SVM (in C-support vector classication mode [15, 16]) is configured by the optimized parameters c (the cost, that is explored between 10^{-1} and 10^8), t (kernel type, in this case SVM uses the precomputed training kernel), v (is the n-fold cross validation mode, fixed to n=5). In the training phase, SVM receives the training labels and the training kernel, while in the predict phase SVM uses the test labels and the test kernel, responsible for obtaining the final result.

The aforementioned process is repeated ten times in order to reduce the effect of random selection of test and training sets, and we report the mean values and standard deviations of the obtained accuracies. The program flow is described in Figure 3, we emphasize that the parallelization is in the calculation of K-means and in the formation of histograms, the ADPM initialization has not been modified.

4. EXPERIMENTAL RESULTS

4.1. Experimental Configuration

Our environment is composed of an Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz (8 cores), 16 GB RAM, HHD storage 320 GB (5400 rpm). In our experiments, we used Ubuntu 15.10 x64 as operating system, Python 2.7.10 and the libraries Numpy 1.10.1, Scikit Learn 0.18.1, Scipy 0.16.1, Libsvm 3.22 and PyQt4. The original Matlab code was executed in Matlab 13 environment.

4.2. Performance evaluation

In this subsection, our goal is to highlight that the parallel multicore implementation provides comparable clustering results with regards to the serial algorithm. For the execution of the tests we use a database composed of 50 images of 600 $\times$ 600 pixels for each of 19 classes (airport, beach, bridge, commercial area, desert, farmland, football field, forest, industrial area, meadow, mountain, park, parking, pond, port, railway station, residential area, river and viaduct) extracted from Google Earth [17, 18]. The dataset is randomly divided into two sets: the training set (of 95 images) and the testing set (of 855 images).

We compared the original version in Matlab with the multicore version in Python. In order to do so, we calculated the times of each iteration, as well as the computational time of the parallelized operations (K-means, with random initialization, and Histograms). The obtained results are shown in Table 1.

	Original		Multicore	
Iteration	**Accuracy**	**Time**	**Accuracy**	**Time**
1	88.30	266.65	88.42	86.32
2	84.68	264.76	83.86	88.74
3	83.04	253.42	82.46	91.03
4	84.44	265.74	82.69	91.93
5	81.05	249.03	81.40	88.60
6	82.34	290.40	82.81	93.79
7	82.10	304.25	82.46	96.71
8	83.63	312.11	84.56	97.14
9	83.74	285.64	84.09	91.21
10	85.50	277.05	86.20	92.18
Average (Std.)	83.88 (1.94)	276.91 (19.88)	83.90 (1.98)	91.77 (3.28)
Total time		2769.05		917.65

Table 1: Classification accuracies and execution times (in seconds) obtained using the original version and the multicore implementation of ADPM.

As can be seen from Table 1, the accuracy results are very similar in both cases but the computational cost of the multicore version is considerably lower than the computational cost of the original version, reducing the execution time in 185.14 seconds on average per iteration. This reduction represents a speedup of 3.02 while preserving the accuracy of the algorithm, which does not suffer major changes, remaining at an average of 84%. These results clearly demonstrate the improvements that can be gained by our newly proposed parallel implementation.

5. CONCLUSIONS AND FUTURE LINES

In this paper, we have proposed a new parallel implementation of the ADPM method. Our experiments show that the proposed parallel scheme is effective, not only in terms of parallel performance, but also when considering classification accuracy. As future work, we will explore the suitability of other specialiced hardware architectures such as the Intel Xeon Phi or graphical processing units (GPUs) to further improve the parallel performance of the analyzed algorithm.

6. REFERENCES

[1] Qingshan Liu, Renlong Hang, Huihui Song, Fuping Zhu, Javier Plaza, and Antonio Plaza, "Adaptive deep pyramid matching for remote sensing scene classification," *CoRR*, vol. abs/1611.03589, 2016.

[2] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, "How transferable are features in deep neural networks?," *CoRR*, vol. abs/1411.1792, 2014.

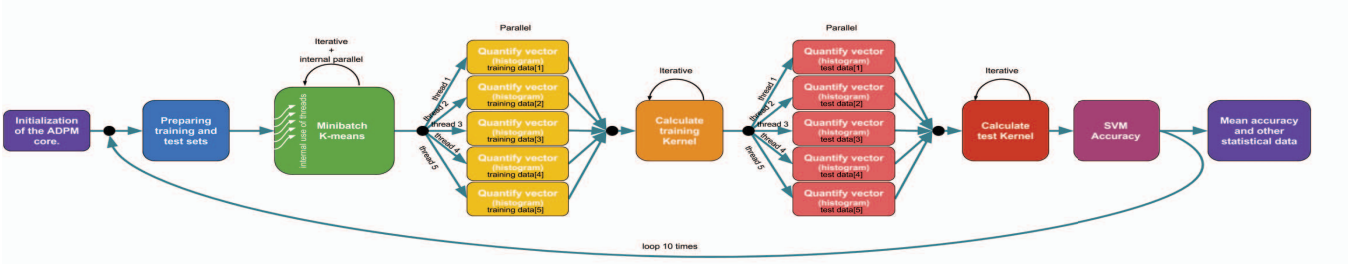

Fig. 3: ADPM multicore parallelization flow.

[3] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, Washington, DC, USA, 2006, CVPR '06, pp. 2169–2178, IEEE Computer Society.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014.

[5] X. Chen, S. Xiang, C. L. Liu, and C. H. Pan, "Vehicle detection in satellite images by hybrid deep convolutional neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 10, pp. 1797–1801, Oct 2014.

[6] Fan Hu, Gui-Song Xia, Jingwen Hu, and Liangpei Zhang, "Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery," *Remote Sensing*, vol. 7, no. 11, pp. 14680, 2015.

[7] J. Sivic and A. Zisserman, "Video google: a text retrieval approach to object matching in videos," in *Proceedings Ninth IEEE International Conference on Computer Vision*, Oct 2003, pp. 1470–1477 vol.2.

[8] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cdric Bray, "Visual categorization with bags of keypoints," in *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004, pp. 1–22.

[9] Fei-Fei Li and Pietro Perona, "A bayesian hierarchical model for learning natural scene categories," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, Washington, DC, USA, 2005, CVPR '05, pp. 524–531, IEEE Computer Society.

[10] Eric Jones, Travis Oliphant, Pearu Peterson, et al., "SciPy: Open source scientific tools for Python," 2001–.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] David Arthur and Sergei Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA, USA, 2007, SODA '07, pp. 1027–1035, Society for Industrial and Applied Mathematics.

[13] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, New York, NY, USA, 2010, WWW '10, pp. 1177–1178, ACM.

[14] Chih-Chung Chang and Chih-Jen Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.

[15] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, 1992, COLT '92, pp. 144–152, ACM.

[16] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] Gui song Xia, Wen Yang, Julie Delon, Yann Gousseau, Hong Sun, and Henri Matre, "Structural high-resolution satellite image indexing," 2010.

[18] D. Dai and W. Yang, "Satellite image classification via two-layer sparse coding with biased image representation," *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 1, pp. 173–176, Jan 2011.