End-to-end Learning of Deep Spatio-temporal Representations for Satellite Image Time Series Classification

Nicola Di Mauro[1], Antonio Vergari[1], Teresa M.A. Basile[2,3], Fabrizio G. Ventola[1], and Floriana Esposito[1]

[1] Department of Computer Science, University of Bari "Aldo Moro", Bari, Italy
[2] Department of Physics, University of Bari "Aldo Moro", Bari, Italy
[3] National Institute for Nuclear Physics (INFN), Bari Division, Bari, Italy

Abstract. In this paper we describe our first-place solution to the discovery challenge on time series land cover classification (TiSeLaC), organized in conjunction of ECML PKDD 2017. The challenge consists in predicting the Land Cover class of a set of pixels given their image time series data acquired by the satellites. We propose an end-to-end learning approach employing both temporal and spatial information and requiring very little data preprocessing and feature engineering. In this report we detail the architecture that ranked first—out of 21 teams—comprising modules using dense multi-layer perceptrons and one-dimensional convolutional neural networks. We discuss this architecture properties in detail as well as several possible enhancements.

Keywords: Satellite image time series classification; deep learning; convolutional neural networks.

1 Introduction

The time series land cover classification challenge (TiSeLaC)[4] was organized by the 2017 European Conference on Machine Learning & Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2017). It consists in a multi-class single label classification problem where the examples to classify are pixels described by the time series of satellite images and the prediction is related to the land cover class associated to each pixel. In particular, the goal is to predict the Land Cover class of a set of pixels given their time series acquired by the satellite images time series. Both training and test data come from the same time series of satellite images, i.e., they span over the same time period and are generated by the same distribution.

We are moved by many recent advancements and successes of deep neural networks for multivariate time series classification of hyperspectral and multispectral images, as reported in [2,8,4,11]. Our approach is fully automated,

[4] `https://sites.google.com/site/dinoienco/tiselc`

requires very little feature engineering and integrates into a single deep architecture both Multi Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) to leverage the spatio-temporal nature of the data. Inspired by [3], we successfully exploited both landsat and spatial information of each pixel.

2 Data

The data provided for the challenge comprises 23 high resolution[5] images gathered from an annual time series by Landsat 8 and depicting the Reunion Island provided at level 2A in 2014. Both training set (consisting of 81714 instances) and testing set (consisting of 17973 instances) pixels are sampled from the same 23 Landsat 8 images. Data have been processed to fill cloudy observations via pixel-wise multi-temporal linear interpolation on each multi-spectral band (OLI) independently. For each pixel, and at each timestamp, a total of 10 landsat features is provided. They include 7 surface reflectances—Ultra Blue, Blue, Green, Red, NIR, SWIR1 and SWIR2—with 3 additional complementary radiometric indices—NDVI, NDWI and BI.

Therefore, three kinds of information are provided and are exploitable for the challenge: i) *band information* comprising the 10 features describing each pixel; ii) *temporal information* as represented by the 23 time points in which band features have been recorded, and lastly iii) *spatial information* in the form of the coordinates associated to each pixel. Our winning solution leveraged all of three into a single model.

Details about class distribution in the training and test data[6] are reported in Table 1. Note how class proportions are consistent across train and test splits and are quite imbalanced w.r.t. minority classes like Other crops and Water. Nevertheless, this has not been an issue for our one-model winning solution, since no additional care for balancing classes has been taken.

Class ID	Class Name	Train	Test
1	Urban areas	16000	4000
2	Other built-up surfaces	3236	647
3	Forests	16000	4000
4	Sparse vegetation	16000	3398
5	Rocks and bare soil	12942	2588
6	Grassland	5681	1136
7	Sugarcane crops	7656	1531
8	Other crops	1600	154
9	Water	2599	519

Table 1. Training and testing class distribution.

[5] 2866 X 2633 pixels at ∼30m spatial resolution.
[6] Test class information has been provided at the end of the competition.

Let scalars be denoted by lower-case letters (e.g. x, y) and vectors by bold lower-case ones (e.g. $\mathbf{x}, \mathbf{y}$). We employ upper-case bold letters for matrices, e.g. $\mathbf{X}, \mathbf{Y}$, while tensors are denoted as sans-serif bold upper-case letters, e.g. X, Y. For a tensor X, $\mathsf{X}_{i..}$ denote the i-th slice matrix along the first axis. Similar notation carries over for slicing on other dimensions.

Let $\mathcal{D} = \{\mathbf{X}_i, y_i\}_{i=1}^N$ be the set of pixels, where each $\mathbf{X}_i \in \mathbb{R}^{23\times 10}$ is a time-series of 23 feature vectors $\mathbf{x}_{i,j} \in \mathbb{R}^{10}$, $1 \leq j \leq 23$: $\mathbf{X}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,23}\}$. Each $\mathbf{x}_{i,j}$ is the 10-dimensional band representation vector. Each i pixel is labeled with a value y_i according to an unknown distribution assigning a class among the $k = 9$ available. All the landsat feature information in $\mathcal{D}$ can be jointly represented as a tensor $\mathsf{X} \in \mathbb{R}^{N\times t\times d}$, where N is the number of pixels (i.e. instances), $t = 23$ the length of the time series (i.e. the number of observations) and $d = 10$ is the number of bands—the landsat features associated to a pixel.

3 The Winning Solution

Two of the three modules of the proposed deep architecture work directly on raw pixel time series data, i.e., X and $\mathbf{y}$, while the third one is fed a preprocessed version of X and $\mathbf{y}$ accounting for spatial autocorrelation [3]. We now describe how to reproduce it.

We exploit the spatial information related to each pixel i by looking at its *spatial neighborhood* $\mathcal{N}_i(r)$—the set of pixels surrounding it with a radius r in the grid [7,3]. Once the spatial neighborhood has been located, we model it by aggregating some features of the pixels in it, e.g., the mean or standard deviation per band [7]. In particular, for each pixel i we compute a d-dimensional spatial feature descriptor $\mathbf{x}_i^s \in \mathbb{R}^d$ as follows. Fixed a radius r, we can efficiently compute the pixel spatial neighborhood $\mathcal{N}_i(r)$ by employing a nearest neighbor balltree based algorithm[7]. Then, we first employ a frequency based operator [3,1] to compute a set of spatial features $\mathbf{c}_i^r \in \mathbb{R}^9$ in which each feature $c_k \in \mathbf{c}_i^r$ computes $\frac{1}{|\mathcal{N}_i(r)|}\sum_{j=1}^{|\mathcal{N}_i(r)|} \mathbb{1}\{y_j = k\}$ for one class label k, i.e., the k-th class label relative frequency in $\mathcal{N}_i(r)$. A second set of spatial features $\mathbf{b}_i^r \in \mathbb{R}^6$ is obtained by considering the moments of a subset of the bands. Specifically, we compute the mean and standard deviation of the three radiometric indexes—the last three bands NDVI, NDWI and BI—for each pixel in the spatial neighborhood $\mathcal{N}_i(r)$, since we discovered the time progression of these three indexes to be somehow correlated to the class labels.

To build the complete spatial feature descriptor $\mathbf{x}_i^s$, we repeat the aforementioned construction process by letting the neighborhood radius r vary in $R = \{1, 3, 5, 7, 9, 11, 13, 15, 17\}$. We determined R by validating it on our held-out set. In the end, we obtain the spatial description $\mathbf{x}_i^s = [\mathbf{c}_i^1\ \mathbf{b}_i^1\ \mathbf{c}_i^3\ \mathbf{b}_i^3 \cdots \mathbf{c}_i^{17}\ \mathbf{b}_i^{17}, \mathbf{p}_i] \in \mathbb{R}^{N=137}$ by concatenating both sets of spatial features for each radius and the pixel coordinates $\mathbf{p}_i$. $\mathbf{X}^s \in \mathbb{R}^{N\times 137}$ indicates the spatial data matrix of all pixels.

All the feature values in both X and $\mathbf{X}^s$ are standardized by removing the mean and scaling to unit variance—for X this has been made *per* band.

[7] We used the BallTree implementation in the Python `scikit-learn` library.

3.1 Model architecture

Our architecture comprises three main components, whose parameters are trained jointly. The basic blocks along all three components are dense layers a-là Multi-Layer Perceptron (MLP) layers and 1-dimensional convolutional layers as in Convolutional Neural Networks (CNNs). The former is a layer in which each hidden neuron is connected to all neurons from the previous layer. The latter, on the other hand, is well suited for sequential data as it exploits local and sparse connections with weight sharing, convolving a vector mask—a kernel—along one input dimension. Additionally, we employ parameter-free layers like max pooling, i.e. subsampling activations; dropout, to regularize training, and layers flattening or concatenating representations.

The first component in our model takes care of learning representations leveraging independencies and dependencies among single bands. The second component, models time series data by jointly learning representations for all bands with fully-convolutional layers. Lastly, the third one, applies a MLP to learn spatial representations over the neighborhood extracted features (see Section 3).

The complete deep model we used is reported in Figure 2. In the following, we describe each component topology in detail.

3.2 Independent band modeling

The first model we employ is inspired by the work in [13]. There, the authors proposed a multi-channel CNN (MC-DCNN) model in which filters are applied on each single channel—here band—and then flattened across channels to be the input of a dense layer. The component we propose is named IBM—standing for Interdependent Band Modeling—and leverages both representations learned on each band independently as in [13] as well as correlations among bands at two different levels of abstraction. Each band $\mathbf{X}_{..i}$ is fed into a CNN architecture with three layers as reported in Figure 1. We firstly apply 1d convolutions of eight filters[8] to $\mathbf{X}_{..i}$ capturing time interactions per band at a first level of abstraction. Higher level time correlations are captured by other four 1d convolutional filters of the same size. After each convolution block, a max pooling layer [9], with a window size equal to 2, reduces their dimensionality. Finally, the features outputting from the last pooling layer are flattened into vectors.

Differently from [13], we also model time cross-correlations by learning *inter*-bands representations. The features computed at the first level by the first ten convolution layers are then concatenated and fed into eight 1d filters with size equal to 10, providing an additional flattened representation. The same process applies to the features computed by the other ten convolution layers at the higher abstraction level, but using four filters only instead. Again, in this way we are modeling time cross-correlation among higher level representations. The non-linearities in all convolution layers are fulfilled by rectified linear units (ReLUs) [6], simply computing $\max(0, x)$.

[8] With the length of all convolutional kernels being equal to 3.

3.3 Fully-convolutional band modeling

We found beneficial to model band interactions considering all time points jointly. The idea behind this is to differentiate learned representations from the one extracted by our IBM component: while IBM features are constrained to learn per band interactions at first, now we are able to capture interactions among time and bands from the start. Therefore, as a second component along IBM we learn additional representations by feeding the whole tensor $\mathbf{X}$ to a fully-convolutional CNN architecture as classical image segmentation architectures [10]. For this role we also experimented with deep recurrent neural networks in the form of deep LSTM, however with no great success in training them on the "few" pixels present in $\mathbf{X}$. Figure 2 depicts this additional component–named FCA as for Fully-Convolutional Architecture–in the combined architecture. For FCA, we employ two sets of 32 1d convolutional filters with size 3 each. We experimented with deeper CNNs for this component, however with very marginal F1 score gains on a validation set, compared to the longer learning times. The output of the last convolution of FCA, flattened into a vector, concurs into the whole representation learned by our architecture.

3.4 Dense spatial representation modeling

Lastly, we model spatial information with our last component, named SDA—standing for Spatial Dense Architecture. Since much information is already contained in the pre-processed $\mathbf{X}^s$ spatial feature representation, we limit our component topology to an MLP model comprising two dense layers. As reported in Figure 2, a first dense layer comprises 128 hidden units and is followed by another one with 64 hidden units. The simple addition of these two layers enhanced the capacity of the model to memorize the training set. To contrast overfitting, after each dense layer a dropout layer [12] has been applied, setting the probability of zeroing a neuron output to 0.3. Finally, as shown in Figure 2, the representations outputted by the three components are concatenated into a single flattened array, then fed to a softmax layer with 9 output units.

4 Learning and Results

We implemented our complete model in Python by employing the Keras[9] library[10]. We trained our model variants by optimizing the categorical cross-entropy as the surrogated loss of the metric employed to rank submissions in TiSeLaC, the weighted F1 score. We initialized all our model weight parameters using the standard "Glorot uniform" scheme. During our experiments we reserved a stratified (according to the class distribution showed in Table 1) portion of the training set, consisting in 10% of the instances, as an held-out split for validation. We tuned our model hyperparameters on this validation set by

[9] `https://keras.io/`.

[10] Code available at `https://github.com/nicoladimauro/TiSeLaC-ECMLPKDD17`.

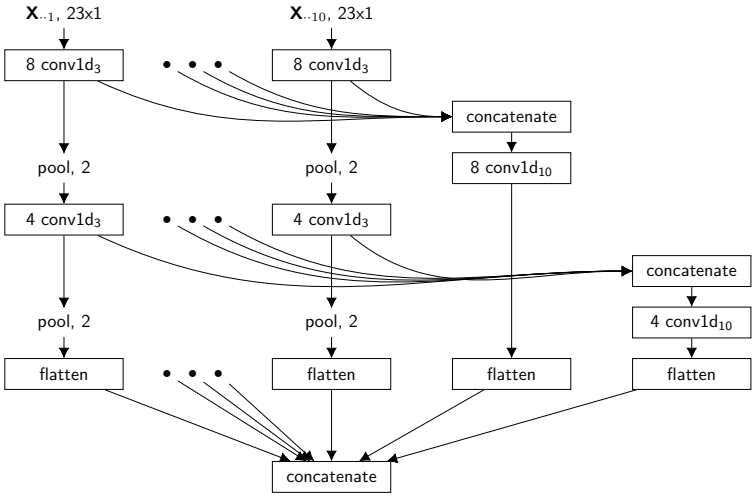

Fig. 1. The IBM independent band modeling module representation. The j-th band time series is fed into a series of two 1-d convolutions followed by max pooling. Inter-band correlations are modeled by concatenating the learned representations at the two different levels (after each single-band convolution) and fed into another convolution each. All learned representations are merged in a single embedding.

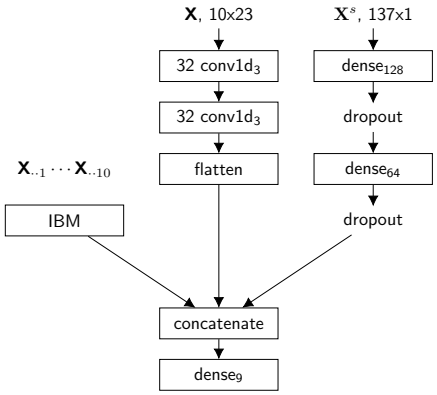

Fig. 2. The complete model representation featuring the independent band modeling module (IBM) on the left (see Figure 1), the fully-convolutional module (FCA) in the center and the dense spatial module (SDA) on the right.

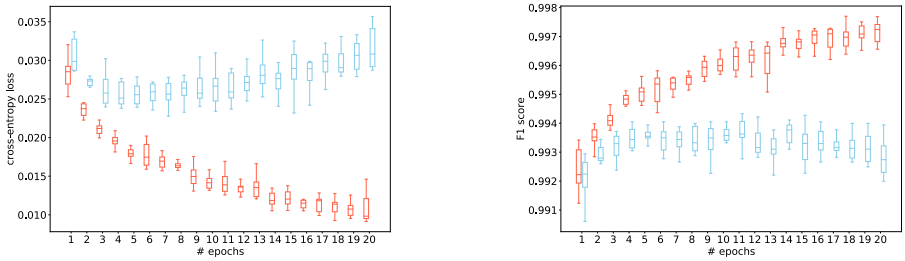

Fig. 3. The box and whiskers plots, on 100 different runs, showing the performance of the architecture on the training (red) and test (blue) set along the 20 epochs employed. Left: the cross-entropy loss; right: the F1 score.

looking at the categorical cross-entropy loss values scored by our model. Among all the hyperparameters considered for tuning there are the spatial neighborhood radius r, as well as the number of filters and the kernel size chosen for each convolutional layer. We choose Adam as the SGD optimizer [5] and we set the learning rate to 0.002, the β_1 and β_2 coefficients to 0.9 and 0.999 respectively and no learning rate decay. We employed mini batches of size 32. We empirically found out that 20 epochs were enough to achieve a low cross-entropy score on the validation set before overfitting.

4.1 Submission

Once the test data has been released, we retrained our model on the whole training set employing the same hyperparameter choices listed above and presented throughout the previous Sections. The single model test set prediction submitted to the competition scored 99.29 F1 score, and this was enough to rank first (2nd placed team scored 99.03). Once the test label have been released after the competition ended we got back to our proposed solution. We noted that both the cross-entropy loss and the F1 score were quite influenced by the initialized model weights. This lead to a range of achievable scores after 20 epochs that is reported in Figure 3 where we measure both metrics on the whole training and test set over 100 different random initializations. The mean test F1 score is reported to be 99.31, confirming the validity of the model effectiveness. As a side note, monitoring the true effect of both metrics on the labeled test data reveals that we should have stopped our training way earlier—even just after 7 epochs.

5 Discussion and conclusions

In contrast to many popular first place solution recipes for nowadays data science challanges—like boosting and stacking—our architecture is a parsimonious single model network. The simplicity of our model only apparently contrasts with its incredible effectiveness. We argue its ultimate success lies into the independent

exploitation of the three different kinds of information available, leading to a *joint spatio-temporal landsat representation* for each pixel in the last layer.

A very quick extension to an ensemble committee can be implemented by leveraging the high variability of results due to different initializations (see previous Section). By averaging the softmax predictions of the 100 models trained for Figure 3, we are able to obtain a more stable and stronger classifier that outputs 99.44 F1 score on the test data (and 99.87 on the training set). This suggests that this single architecture can be plugged into more sophisticated ensembling schemes, likely leading to even larger improvements.

References

1. Appice, A., Guccione, P., Malerba, D.: Transductive hyperspectral image classification: toward integrating spectral and relational features via an iterative ensemble system. Machine Learning 103(3), 343–375 (2016)
2. Chen, Y., Jiang, H., Li, C., Jia, X., Ghamisi, P.: Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. IEEE Trans. Geoscience and Remote Sensing 54(10), 6232–6251 (2016)
3. Guccione, P., Mascolo, L., Appice, A.: Iterative hyperspectral image classification using spectral-spatial relational features. IEEE Trans. Geoscience and Remote Sensing 53(7), 3615–3627 (2015)
4. Ienco, D., Gaetano, R., Dupaquier, C., Maurel, P.: Land cover classification via multi-temporal spatial data by recurrent neural networks. CoRR abs/1704.04055 (2017)
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014)
6. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML. pp. 807–814 (2010)
7. 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.: Recent advances in techniques for hyperspectral image processing. Remote Sensing of Environment 113, S110 – S122 (2009)
8. Romero, A., Gatta, C., Camps-Valls, G.: Unsupervised deep feature extraction for remote sensing image classification. IEEE Trans. Geoscience and Remote Sensing 54(3), 1349–1362 (2016)
9. Schmidhuber, J.: Multi-column deep neural networks for image classification. In: CVPR. pp. 3642–3649 (2012)
10. Shelhamer, E., Long, J., Darrell, T.: Fully convolutional networks for semantic segmentation. IEEE Trans. Pattern Anal. Mach. Intell. 39(4), 640–651 (2017)
11. Slavkovikj, V., Verstockt, S., Neve, W.D., Hoecke, S.V., de Walle, R.V.: Hyperspectral image classification with convolutional neural networks. In: Proceedings of the 23rd Annual ACM Conference on Multimedia Conference. pp. 1159–1162 (2015)
12. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15(1), 1929–1958 (2014)
13. Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L.: Time series classification using multi-channels deep convolutional neural networks. In: 15th International Conference on Web-Age Information Management. pp. 298–310 (2014)