



# A hybrid CUDA, OpenMP, and MPI parallel TCA-based domain adaptation for classification of very high-resolution remote sensing images

Alberto S. Garea<sup>1</sup> · Dora B. Heras<sup>1</sup> · Francisco Argüello<sup>2</sup> · Begüm Demir<sup>3</sup>

Accepted: 16 November 2022  
© The Author(s) 2022

## Abstract

Domain Adaptation (DA) is a technique that aims at extracting information from a labeled remote sensing image to allow classifying a different image obtained by the same sensor but at a different geographical location. This is a very complex problem from the computational point of view, specially due to the very high-resolution of multispectral images. TCANet is a deep learning neural network for DA classification problems that has been proven as very accurate for solving them. TCANet consists of several stages based on the application of convolutional filters obtained through Transfer Component Analysis (TCA) computed over the input images. It does not require backpropagation training, in contrast to the usual CNN-based networks, as the convolutional filters are directly computed based on the TCA transform applied over the training samples. In this paper, a hybrid parallel TCA-based domain adaptation technique for solving the classification of very high-resolution multispectral images is presented. It is designed for efficient execution on a multi-node computer by using Message Passing Interface (MPI), exploiting the available Graphical Processing Units (GPUs), and making efficient use of each multicore node by using Open Multi-Processing (OpenMP). As a result, an accurate DA technique from the point of view of classification and with high speedup values over the sequential version is obtained, increasing the applicability of the technique to real problems.

**Keywords** CUDA · OpenMP · MPI · GPU · Multicore · Domain adaptation · Feature extraction · Remote sensing · Multispectral

---

Alberto S. Garea, Dora B. Heras, Francisco Argüello and Begüm Demir have contributed equally to this work.

---

✉ Alberto S. Garea  
jorge.suarez.garea@usc.es

Extended author information available on the last page of the article

## 1 Introduction

The classification of images is one of the most common processes in the Remote Sensing (RS) field [1–3]. During this process, each of the pixels that comprises the image is classified, that is, it is assigned to one of the previously defined class labels. The solution to classification problems using supervised methods is strongly linked to the existence of reference field information. In many cases, building this information is costly in time and effort since it may require data from difficult access areas and a subsequent contrast of information provided by different sources. In addition, when working with high-resolution images, most of the common classification algorithms fail to distinguish the different elements present in the image [4].

As an alternative, Machine Learning (ML)-based classification algorithms play a very important role in the remote sensing literature for very high spatial resolution images [5, 6]. ML encompasses a set of algorithms that capture system dynamics without human intervention. This makes them more robust and less dependent on human experts. Deep Learning (DL) defines a subset of ML algorithms with high computational cost, which learn to represent the problem as a nested hierarchy of concepts where more general concepts are defined in relation to simpler concepts and more abstract representations are obtained based on less abstract ones. ML-based architectures such as Artificial Neural Networks (ANNs) [7, 8] or architectures based on DL such as Deep artificial Neural Networks (DNNs) [9] or Convolutional Neural Networks (CNNs) [10–12] have been successfully used for remote sensing image classification.

The lack of high-quality reference data [13, 14] poses a particular challenge in the classification of the images. The problem is more complex if images belonging to different spatial areas need to be classified or if they were taken by different sensors or at different times. In all these cases, the spectral shift between the different images, produced during in-flight data acquisitions due, for example, to instrumental pressure, temperature, or vibrations, could reduce the accuracy of a joint classifier [14, 15]. Furthermore, given the fact that the scarcity of available reference information affects the success of the classification task, and since the manual labeling process is very expensive [16], the need to find techniques that take advantage of all the available reference information gains particular relevance. Transfer Learning (TL) methods contribute to alleviate the problem.

TL encompasses a set of techniques that are responsible for applying the knowledge previously acquired for one or more tasks in a given source domain, to another task related to the initial one, in a target domain [17]. Domain Adaptation (DA) techniques [18–21], which are a subset of TL, use reference information from images belonging to a source domain to try to classify images belonging to a target domain, for which no reference information is available. Among the techniques applied to perform DA, we can highlight Feature Extraction (FE) techniques [22, 23], which try to reduce the displacement between the two domains by finding a function that can map the data to a new space that better defines them. Within these techniques, CORAL [23] is an example, based on the alignment of the covariance of both domains. Other more complex examples are based on neural networks such as techniques based on Stacked Denoising Auto-Encoders (SDAE) [22].

The use of DL-based architectures for DA, such as CNNs, has also been proven effective [24]. This type of network is basically made up of a set of nested convolutional filters. They require an iterative process to learn the weights of the network [25], which, depending on its complexity, could become very expensive. Therefore, different alternatives to this type of neural network have been proposed to reduce the computational cost. In [26], the authors proposed to replace the costly iterative process of training a CNN by computing the convolutional network filters using Principal Component Analysis (PCAs). For the case where the data are not linearly separable, [27] presented a scheme that uses kernel PCA (KPCA) to extract features.

Similarly, TCANet [28] was proposed by the authors as a DL network for DA applied to multi and hyperspectral remote sensing images. Similarly to [26], in this case, the convolutional filters that operate on patches of the multispectral image are obtained through Transfer Component Analysis (TCA). TCA [29] is a feature extraction technique specially designed for DA. TCANet has the main advantage that it does not require training based on backpropagation, since TCA is itself a learning method that obtains the filter coefficients directly from the input data. Even with this advantage, TCANet incurs in a high computational cost requiring parallel computation. The high computational cost of processing RS information makes the use of parallel computing resources imperative [30].

As in other research fields, the requirement for rapid and effective solutions for processing the massive amounts of data associated with RS has led to the extended use of different computing paradigms the last few years. These include supercomputing, cloud computing, specialized hardware computing, and quantum computing, among others [31, 32]. In particular, supercomputers have been widely used in RS applications to accelerate and scale the processes of image classification, target detection, clustering, registration, data fusion, compression or feature selection/extraction [33]. Many of the papers focus on exploiting the heterogeneous parallel architectures of the computing nodes by using a hybrid MPI, OpenMP and CUDA implementation [34]. For example, [35] presents a hybrid implementation of a target classification and recognition technique based on the Cross model Neural Cognitive Computing (CNCC) algorithm.

Regarding techniques based on DL, most of them are designed for being executed at least using GPU-based architectures. Nevertheless, new parallel and big data implementations are being provided by the authors trying to reduce the computational time of the applications. For example, [36] presented a cloud-based implementation of an autoencoder (AE), a Deep Neural Network (DNN) for nonlinear data compression. In [37] a hyperspectral pyramidal ResNet architecture model executed on a heterogeneous system was proposed. Ordoñez et al. [38] exploited the parallelism of a cluster in a multi-node, multi-GPU implementation for the effective co-registration of bands and multispectral images.

In this paper, a hybrid parallel supervised classification scheme based on DA by using TCANet and Support Vector Machines (SVM) as classifier is presented. It is applied to very high-resolution multispectral images corresponding to vegetation areas captured by a drone. An optimization of the technique based on reducing the computational cost by reducing the number of data blocks processed in the training stage of the scheme is firstly proposed. In addition, the code is modified to exploit a multi-node architecture by using Message Passing Interface (MPI), the multicore architecture of each node by using Open Multiprocessing (OpenMP), and the GPUs available in each node by using Compute Unified Device Architecture (CUDA). The resulting implementation considerably decreases the execution time of the classification scheme

while preserving the quality of the classification results. The codes and datasets used in the experiments are available at [https://gitlab.citius.usc.es/hiperespectral/tcanet\\_jos\\_2022](https://gitlab.citius.usc.es/hiperespectral/tcanet_jos_2022).

## 2 TCANet-based classification scheme

TCANet is, as explained in the previous section, based on TCA for the computation of the filters corresponding to each layer of the DA network. This section first describes TCA and then presents the proposed supervised classification scheme based on TCANet [28].

### 2.1 TCA for domain adaptation

TCA is a kernel-based feature extraction technique proposed by [29]. To achieve this objective, TCA tries to learn transfer components across domains in a Reproducing Kernel Hilbert Space (RKHS). To minimize the distance between the two distributions, TCA applies Maximum Mean Discrepancy (MMD) [39], a nonparametric and computationally simpler kernel-based measure defined as:

$$\text{dist}(\mathbf{X}_S, \mathbf{X}_T) = \left\| \frac{1}{N_S} \sum_{i=1}^{N_S} \phi(\mathbf{x}_{S_i}) - \frac{1}{N_T} \sum_{i=1}^{N_T} \phi(\mathbf{x}_{T_i}) \right\|_{\mathcal{H}}^2, \quad (1)$$

where  $\mathbf{X}_S$  corresponds to the data of the first distribution (source),  $\mathbf{X}_T$  to the data of the other one (target),  $N_S$  and  $N_T$  are the number of samples from the source and the target domains, respectively.  $\mathcal{H}$  is a universal RKHS [40], and  $\phi : \mathcal{X} \rightarrow \mathcal{H}$  is a nonlinear mapping function that can be found by minimizing the distance as defined by (1). Using the kernel trick (i.e.,  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)' \phi(\mathbf{x}_j)$ ), [41] proposed to convert this problem into a kernel learning problem, where (1) can be rewritten as:

$$\text{dist}(\mathbf{X}_S, \mathbf{X}_T) = \text{trace}(\mathbf{K}\mathbf{L}), \quad (2)$$

where

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{S,S} & \mathbf{K}_{S,T} \\ \mathbf{K}_{T,S} & \mathbf{K}_{T,T} \end{bmatrix} \in \mathbb{R}^{(N_S+N_T) \times (N_S+N_T)} \quad (3)$$

is a kernel matrix whose elements have been defined by  $K()$  on the source domain ( $\mathbf{K}_{S,S}$ ), on the target domain ( $\mathbf{K}_{T,T}$ ), and also on the cross-domain ( $\mathbf{K}_{S,T}$  and  $\mathbf{K}_{T,S}$ ), and  $\mathbf{L} = [L_{ij}] \geq 0$  with

$$L_{ij} = \begin{cases} \frac{1}{N_S^2} & \text{if } \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}_S, \\ \frac{1}{N_T^2} & \text{if } \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}_T, \\ -\frac{1}{N_S N_T} & \text{otherwise.} \end{cases} \quad (4)$$

Based on a unified kernel learning method proposed by [29] that uses an explicit low-rank representation, the kernel learning problem solved by TCA can be summarized as:

$$\begin{aligned} \min_w \quad & \text{trace}(\mathbf{W}^T \mathbf{W}) + \mu \text{trace}(\mathbf{W}^T \mathbf{K} \mathbf{L} \mathbf{K} \mathbf{W}) \\ \text{s.t.} \quad & \mathbf{W}^T \mathbf{K} \mathbf{H} \mathbf{K} \mathbf{W} = \mathbf{I}, \end{aligned} \tag{5}$$

where the second trace is the distance between mapped samples  $\text{dist}(\mathbf{X}'_S, \mathbf{X}'_T)$  such that  $\mathbf{X}'_S = \{\mathbf{x}'_{S_i}\} = \{\phi(\mathbf{x}_{S_i})\}$ ,  $\mathbf{X}'_T = \{\mathbf{x}'_{T_i}\} = \{\phi(\mathbf{x}_{T_i})\}$ ,  $\mu$  is a trade-off parameter,  $\mathbf{I}$  is an identity matrix of size  $m \times m$ , and  $\mathbf{H}$  is a centering matrix. The first trace in (5) is a regularization term needed to control the complexity of the projection matrix  $\mathbf{W} \in \mathbb{R}^{(N_S+N_T) \times m}$ ,  $m \ll N_S + N_T$ .  $\mathbf{W}$  is necessary to transform the corresponding feature vectors to the new  $m$ -dimensional space. To avoid the trivial solution ( $\mathbf{W} = 0$ ), the constraint  $\mathbf{W}^T \mathbf{K} \mathbf{H} \mathbf{K} \mathbf{W} = \mathbf{I}$  is added.

Equation (5) can be reformulated as a trace maximization problem where the solution of the projection matrix  $\mathbf{W}$  comes through the eigendecomposition of

$$\mathbf{E} = (\mathbf{I} + \mu \mathbf{K} \mathbf{L} \mathbf{K})^{-1} \mathbf{K} \mathbf{H} \mathbf{K}, \tag{6}$$

giving the  $m$  eigenvectors corresponding to the  $m$  principal eigenvalues of  $\mathbf{E}$ .

### 2.2 TCANet-based classification scheme

This section describes the TCANet-based classification scheme used in this paper and previously presented in [28]. Figure 1 shows the block diagram of the proposed classification scheme that includes a final supervised classification stage using SVM. The diagram details the steps followed for the training stage.

After a first step where  $N_S$  and  $N_T$  random samples are selected as a training set from the source and the target images, respectively, a patch extraction process is performed. This process extracts a patch of size  $D \times D \times B$  around each selected training sample being  $D$  the spatial width and height of the patch and  $B$  the number of bands. The training samples are  $\{\mathbf{s}_i\}_{i=1}^{N_S}$ ,  $\mathbf{s}_i \in \mathbb{R}^{D \times D \times B}$  for the source, and  $\{\mathbf{t}_i\}_{i=1}^{N_T}$ ,  $\mathbf{t}_i \in \mathbb{R}^{D \times D \times B}$  for the target. Then, each patch is reshaped from 3D ( $D \times D \times B$ ) to a 2D patch of size  $D^2 \times B$ . So, the input for the next step is built by stacking  $n_S$  2D patches from the source and  $n_T$  2D patches from the target  $\{\mathbf{x}_i\}_{i=1}^{n_1}$ ,  $\mathbf{x}_i \in \mathbb{R}^{D^2 \times B}$ ,  $n_1 = N_S + N_T$ .

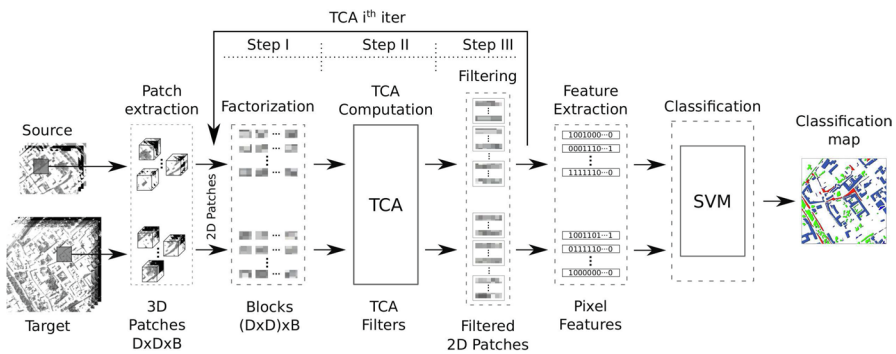


Fig. 1 Diagram of the classification scheme based on TCANet. Steps followed for training

After the patch extraction step, a sequence of TCA stages equivalent to the different layers of a CNN are computed. These are represented in Fig. 1 as TCA iterations. The output of each iteration is the input to the following one. Similar to a CNN where the number of outputs from a layer is proportional to the number of filters it applies, the number of outputs of each TCA iteration is proportional to the number of TCA filters computed in the iteration. It consists of three steps as indicated in the figure. First, in step I, a factorization of the 2D patches is computed. It consists of extracting from each 2D patch, blocks of data for the TCA computation of step II. TCA is used to perform the computation of the called TCA filters, which are used to compute a convolutional operation over all the previously extracted 2D patches. During step III a filtering process is performed by applying the computed filters of the iteration producing filtered 2D patches as output.

Once the last TCA iteration is computed, a feature extraction process is responsible for producing the new features generated by the network. A new representation for each of the  $N_S$  and  $N_T$  input samples is obtained based on these new features. The last step of the procedure is the classification process in which each of the initial input samples is assigned to a class among a set of predefined possible classes. Any supervised algorithm could be selected for classification. In this case, SVM, the same classifier as in [28] was selected. SVM classifiers have been found to provide similar results to other commonly used, nonparametric classifiers such as Random Forest (RF) and can handle scenarios with a low number of training samples [42]. SVM is also presented by some authors as a standard noncontextual classifier for remote sensing classification [43].

For the test stage, all the steps described above are applied, except for the computation of the TCA filters. These filters are computed in the training stage and only applied in the test stage. As can be seen, there is a high potential of parallelism in the scheme as the same operations are performed over a large number of data patches.

### 3 Hybrid parallel TCANet-based classification scheme

The classification process described in the previous section incurs in high computational costs for different reasons. The main one is the large size of the multispectral datasets as a consequence of the very high spatial resolution of the images. Another reason is that the domain adaptation network requires the extraction of a 3D patch for each pixel used for training and for testing. In addition, the number of sub-patches increases with each iteration of TCA as this number is multiplied by a factor equal to the number of filters for the iteration. As a consequence, the process is hardly computable without applying optimizations and exploiting the concurrency capabilities of the available hardware. Different optimization and parallelization techniques have been applied in order to design a technique that is computationally efficient. In particular, we propose a hybrid CUDA, OpenMP, and MPI parallel scheme for classification based on DA.

Algorithm 1 presents the pseudocode summarizing the parallel proposal for an optimized TCANet-based classification scheme. The full scheme requires executing the code, first for training and then with some changes for testing. The pseudocode corresponds only to the training stage. It consists, as explained in the previous section, of the following steps: patch extraction, iterative TCA computation made up of three steps (factorization, TCA calculation, and filtering), feature extraction, and, finally, classification training by using the SVM. Later on, we will explain the differences between the training and the test stages of the scheme.

**Algorithm 1** Pseudocode for the parallel TCANet-based classification scheme. Training stage.

**Input:** source and target images of B bands and their reference maps.

**Output:** SVM classification model.

**Parameters:**

$N_S$ : number of source samples for training.

$N_T$ : number of target samples for training.

$K$ : number of TCA iterations.

$F_i$ : number of filters for iteration  $i$ .

$D$ : width and height of each patch and block.

$\mathbf{V}_i$ : filter matrix for iteration  $i$ . Dimensions:  $(D^2 \times B) \times (F_i)$ .

$\mathbf{X}_i$ : matrix of extracted blocks. Dimensions:  $(D^2 \times B) \times ((N_s + N_t) \times F_{(i-1)})$ .

$\mathbf{H}$ : centering matrix. Dimensions:  $((N_s + N_t) \times F_{(i-1)}) \times ((N_s + N_t) \times F_{(i-1)})$ .

$\mathbf{L}$ : scale matrix. Dimensions:  $((N_s + N_t) \times F_{(i-1)}) \times ((N_s + N_t) \times F_{(i-1)})$ .

**Preprocessing.**

- 1: Read datasets and reference maps

**Patch extraction.**

Extraction of  $N_S + N_T$  patches

- 2: Select  $N_S$  and  $N_T$  training samples from source and target datasets, respectively
- 3: *#OpenMP parallel for with p threads*
- 4: **for** each training sample **do**
- 5:     Extract a  $D \times D \times B$  patch
- 6: **end for**

**TCA iteration  $i$ .**

Produces  $\mathbf{X}_i$  for the next TCA iteration

**Step I. Factorization.**

- 7: **if**  $i = 1$  **then**
- 8:     *#OpenMP parallel for with p threads*
- 9:     **for** each input training patch **do**
- 10:         Extract a  $D^2 \times B$  block
- 11:     **end for**
- 12: **else**
- 13:     *#OpenMP parallel for with p threads*
- 14:     **for** each patch obtained by the TCA iteration  $i - 1$  **do**
- 15:         Extract a  $D^2 \times B$  block
- 16:     **end for**
- 17: **end if**

**Step II. TCA Computation in GPU.**

- 18: *#Equation 6 computed in GPU using CUDA*
- 19:  $\langle \text{cublasXtDgemm} \rangle \rightarrow$  Generate  $\mathbf{H}$
- 20:  $\langle \text{cublasXtDgemm} \rangle \rightarrow$  Generate  $\mathbf{L}$
- 21:  $\langle \text{cublasXtDgemm} \rangle \rightarrow$  Compute  $\mathbf{X}_i \mathbf{L} \mathbf{X}_i^T$
- 22:  $\langle \text{cublasXtDgemm} \rangle \rightarrow$  Compute  $\mathbf{X}_i \mathbf{H} \mathbf{X}_i^T$
- 23:  $\langle \text{cusolverDnDsygv} \rangle \rightarrow$  Compute eigenvalues ( $\mathbf{V}_i$ )

**Step III. Filtering.**

- 24: **if**  $i = 1$  **then**
- 25:     *#OpenMP parallel for with p threads*
- 26:     **for** each input training patch **do**
- 27:         Extract overlapping blocks of size  $D^2 \times B$
- 28:         Apply the matrix filter  $\mathbf{V}_i$  to the blocks
- 29:     **end for**

---

```

30: else
31:   #OpenMP parallel for with p threads
32:   for each patch obtained by the TCA iteration  $i - 1$  do
33:     Extract overlapping blocks of size  $D^2 \times B$ 
34:     Apply the matrix filter  $\mathbf{V}_i$  to the blocks
35:   end for
36: end if

```

**Feature extraction.**

```

Calculate  $\prod F_i, i = 1, \dots, K - 1$  features
37: #OpenMP parallel for with p threads
38: for each patch in  $X_K$  obtained from the source dataset do
39:   Apply the Heaviside function
40: end for
41: #OpenMP parallel for with p threads
42: for each group of  $F_K$  patches do
43:   Reduction by summation
44:   Binarization by threshold obtaining sub-features
45:   Concatenation of sub-features
46: end for

```

**Classification. Model generation.**

```

47: Training SVM with the extracted features

```

---

The first optimization applied to the code with respect to the first version of TCANet presented in [28] consists of reducing the computational cost by minimizing the number of blocks that are extracted from each patch during the factorization step (step I shown in lines 7–17 of the pseudocode in Algorithm 1). The step has been modified for extracting only non-overlapping blocks, so that the spatial context information is still considered but the number of blocks is reduced.

The detailed operations associated to one iteration of TCA (lines 7–36 in the pseudocode) can be observed in Fig. 2. In particular, the factorization is shown in the bottom left part of the figure, denoted as Step I. All the extracted blocks are used in step II to compute the TCA filters (lines 18–23 in the pseudocode) as shown in Fig. 2. Note in the parameter definition of the pseudocode that, for each TCA iteration, the sizes of matrices  $\mathbf{X}_i$ ,  $\mathbf{H}$ , and  $\mathbf{L}$  are increased, in particular, they are multiplied by the number of filters computed in the iteration. The size of the matrix of filters for iteration  $i$  is  $\mathbf{V}_i$  and it is directly proportional to the number of filters for the iteration. The cost of the TCA computation is, therefore, increased for each new TCA iteration. The computed TCA filters are used, as in a CNN, to perform convolutional operations on input patches in step III (lines 24–36 of the pseudocode and also represented in Fig. 2). During Step III the computed filters (matrix  $\mathbf{V}_i$ ) are applied to each input patch producing the outputs of the iteration (see [28] for a more detailed explanation).

After the filtering process, features need to be extracted. A complete explanation of the steps followed for feature extraction (see lines 37–46 of the pseudocode) are described in [28]. This feature extraction consists basically on two loops. The first one performs a binarization of the filtered patches produced by the last TCA iteration by using the Heaviside function. The second one is a loop that in each iteration reduces the features obtained for each group of  $F_K$  binarized filtered patches being  $K$  the number of TCA iterations. The output of this process is a feature vector describing each input patch. It was proven that the accuracy of



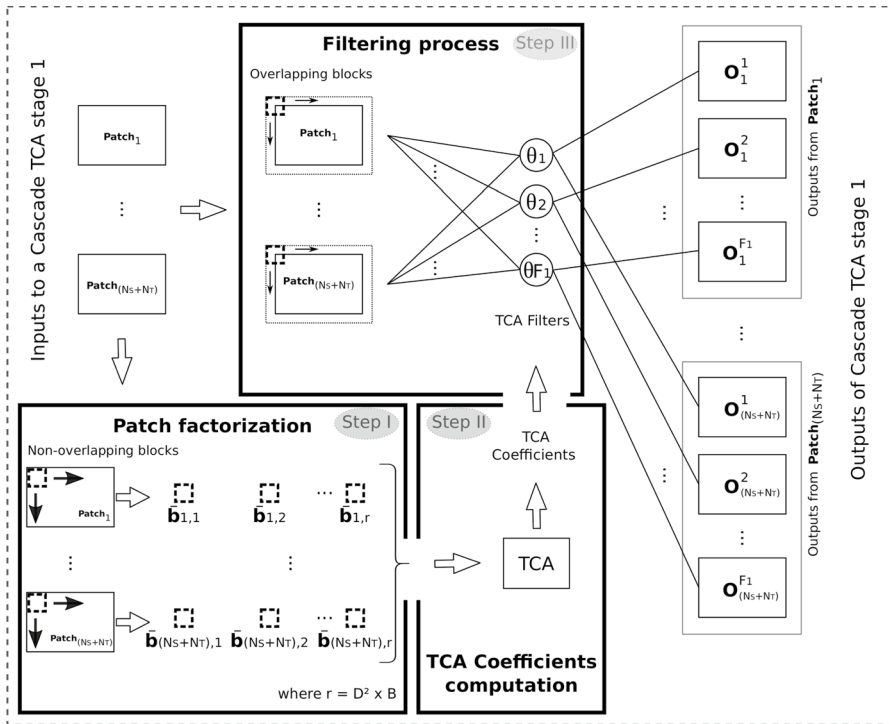


Fig. 2 Details of one TCA iteration [28]

the classification obtained by means of the parallel algorithm is the same as that obtained by the original TCANet.

The whole classification scheme was designed to exploit multiple computing nodes by using MPI, the multicore processors of each computing node by using OpenMP, and the multi-GPU architecture available in the computing nodes through the use of CUDA.

OpenMP is a standard and portable Application Programming Interface (API) for writing shared memory parallel programs [44]. It means that it is designed for systems where all their threads have direct access to all available memory. OpenMP provides the programmer with high-level tools (compiler directives and library routines) that facilitate the parallelization of serial programs in Fortran and C/C++.

In the OpenMP execution model, a thread called master is always defined, and it exists during the execution of the program. Only when a parallel region is encountered, additional threads are created to perform the parallel task. The master thread is responsible for creating and activating these threads, whose number can be defined by the user or by the system where the program is being executed. The synchronization of all the threads takes place at the end of the parallel region with an implicit barrier. Once the last thread has finished its work and, therefore, it has reached the barrier, the execution continues with the master thread until a new parallel region is encountered.

In particular, OpenMP was applied in Algorithm 1 to the steps called patch extraction, factorization, filtering, feature extraction, and classification. The main regions of the code affected by

the OpenMP parallelization are indicated (lines 3, 8, 13, 25, 31, 37, and 41 of the pseudocode). With this computing model, the work is executed by different threads assigned to different cores by means of OpenMP inside each computing node. As the tasks performed within the loops of the code may be different for each thread, a dynamic scheduling strategy was selected for automatically partitioning the iterations of those loops with OpenMP thus avoiding idle threads.

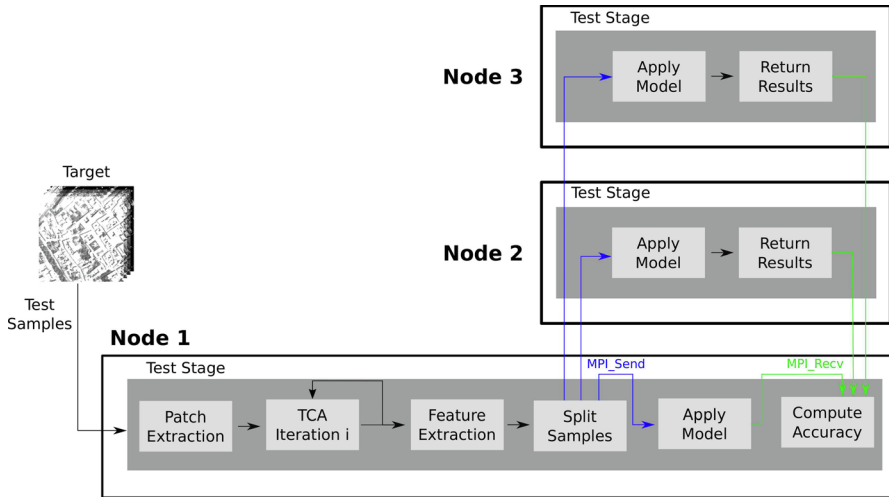
CUDA is a general-purpose parallel computing platform and programming model introduced by NVIDIA in 2006 [45]. The way NVIDIA GPUs execute programs using CUDA is by invoking parallel functions called kernels that run as a grid of blocks of threads. In our case, as shown in Algorithm 1 (lines 18–23), step II of each TCA iteration (lines 19–23) is computed using the GPUs available in the computing node. Different kernels are executed by the different GPUs by using CUDA over C++. For each kernel executed in a GPU, each block of threads corresponding to the kernel is assigned to any of the Streaming Multiprocessors available in the GPU, so not all blocks run concurrently.

After the training stage described in Algorithm 1, a similar test stage needs to be executed. The main differences between training and test are that the test does not include TCA computation and that it is executed for test samples, the pixels of the target not belonging to the training set. As the number of test pixels is much higher than the number of training pixels, the computational cost of the test stage is also higher than the cost of the training stage. Trying to mitigate this high cost, MPI is used for exploiting the different computing nodes available in the architectures. MPI is a standard for communication between processes running in a distributed memory system. Figure 3 shows the operation of the MPI code. First, the input patches are extracted and processed by a master computing node, using the pre-computed TCA filters, to obtain their new features. CUDA and OpenMP implementations are used for each step as it was indicated in Algorithm 1. Then, the *MPI\_Send* and *MPI\_Recv* functions are used to establish a point-to-point communication between the master node and the other two slave nodes to equally distribute these features among them as shown in Fig. 3. Each MPI node applies the SVM classification model to a group of samples. The results of the classifications performed in each slave node (i.e., classification maps that label each pixel indicating the class to which it belongs) are collected again by the master node. This scheme could be extended to a higher number of nodes if the number of test samples were high enough.

## 4 Experimental results

This section shows the experimental results obtained for the different implementations of the hybrid CUDA, OpenMP, and MPI parallel TCANet-based classification scheme. Several optimizations and parallelization techniques are compared in terms of computation time and speedups.

The different versions of the proposed implementation have been coded in C++. In addition, a version using MATLAB has been developed to compare runtime parallelization differences between C++ and MATLAB in those implementations without MPI. Regarding hardware, a PowerEdge R730 server has been used for running the MATLAB codes. This server includes two Intel Xeon E52623v4 processors with four cores (eight threads) each and 128 GB of DDR4 memory. For the C++ codes, the Finisterrae III supercomputer has been used. This supercomputer is located at the Galician Supercomputing Center (CESGA) and consists of 354 nodes. Each node includes two Intel Xeon Ice Lake 8352Y processors with



**Fig. 3** Diagram of the test stage using MPI on three nodes

32 cores each and a minimum of 256 GB of memory. For the experiments, three nodes were used. Although the code could be efficiently executed on a higher number of MPI nodes, datasets of a higher size would be necessary for compensating the overhead of distributing the data among those nodes. CUDA codes run on the Nvidia A100 GPUs available in each computing node of Finisterrae. Each node has two GPUs based on the NVIDIA Ampere architecture and is equipped with 108 multiprocessors and 64 cores per multiprocessor, resulting in 6912 cores each. The CUDA capability version is 8.0 and each card has 40 GB of memory available.

The codes have been compiled using the g++ 10.1.0 version with OpenMP 4.0 support under Linux using different configurations of threads and MATLAB 2015. The OpenBLAS library was used to accelerate algebraic operations. Regarding the GPU implementation, the CUDA codes have been compiled under Linux using the nvcc version 11.2 of the toolkit. The cublasXt API, included in the cuBLAS library, was used to perform multi-GPU computations. Version 4.1.4 of the OpenMPI library was used for the experiments.

The LIBSVM library [46] in its version 3.25 was used for the classification step. The type of SVM was C-Support Vector Classification (C-SVC) using the Radial Basis Function (RBF) kernel and parameters gamma and C with values  $2^{-10}$  and  $2^5$ , respectively. As usual in remote sensing [47], the classification accuracy results are presented in terms of overall accuracy (OA), which is the percentage of correctly classified pixels [48].

#### 4.1 Datasets

Three remote sensing datasets were constructed for the experiments based on the data acquired by the MicaSense RedEdge multispectral camera mounted on a custom Unmanned aerial vehicle (UAV). This sensor captures five different spectral bands: blue (475 nm), green (560 nm), red (668 nm), red-edge (717 nm), and NIR (840 nm). The three images were taken at a height of 120m, with a high spatial resolution of 10 cm/pixel, during the summer months

of 2018 in Galicia, Spain. Seven different classes corresponding to different materials were considered for the experiments: water, tiles, asphalt, bare soil, rock, concrete, and vegetation. For each of the three images, two disjoint regions were selected and labeled as source and target. The objective is the classification of the target region.

The first dataset, shown in Fig. 4, was built based on an image capturing the watershed of the river Oitavèn, Oitavèn from now on. An RGB composite color of the image and the corresponding reference data for classification are shown. This image has a spatial size of  $6689 \times 6722$  pixels. The regions selected as source and target are also depicted. Figure 5 shows the Eiras dam dataset, Eiras from now on, with a spatial size of  $5176 \times 18224$  pixels. Finally, the third dataset shown in Fig. 6 corresponds to creek Ermidas, Ermidas from now on, and its size is  $11924 \times 18972$  pixels.

Tables 1, 2, and 3 show the number of available samples of the source and the target regions for the Oitavèn, Eiras, and Ermidas datasets, respectively, as well as the dimensions in pixels of the regions. A maximum number of 2000 samples per class from both source and target were randomly selected for the training phase. In those classes in which the number of samples is less than 2000, all available samples were selected. The remainder of the pixels from the target were used for the test phase. Seven different classes were considered.

The parameter values of the experiments, as defined in Algorithm 1, for the three datasets are shown in Table 4. It can be seen that the values for all the parameters, independent from the size of the datasets, are the same for all the datasets for comparative purposes. In particular, the number of TCA iterations selected was 2, being 2 and 16 the sizes of the TCA filters  $f_1$  and  $f_2$  for the first and the second iteration, respectively.

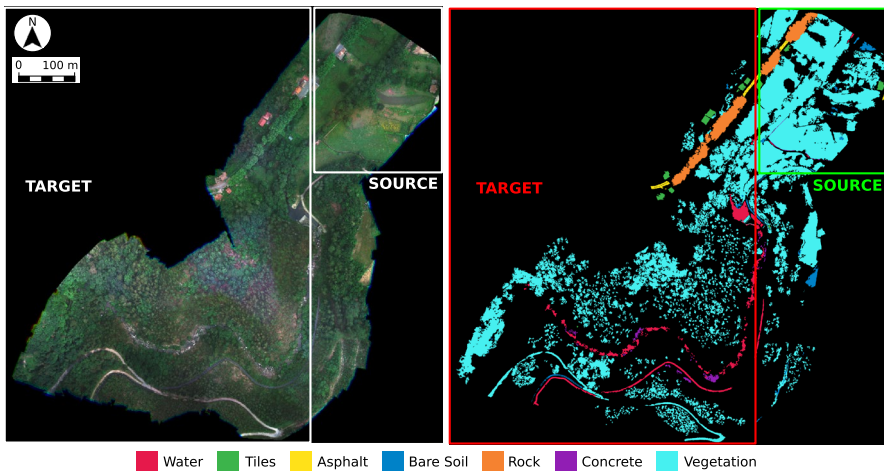


Fig. 4 Oitavèn. False color composite and reference data for classification

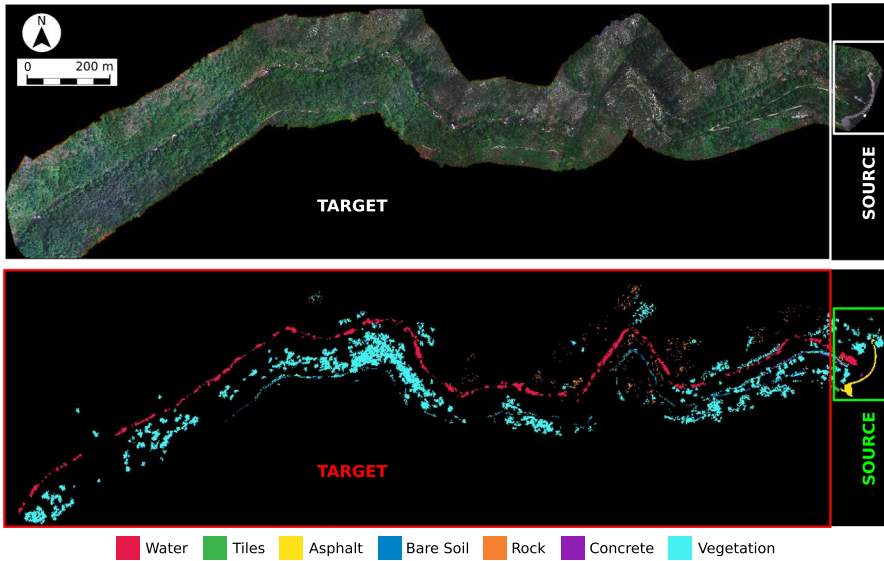


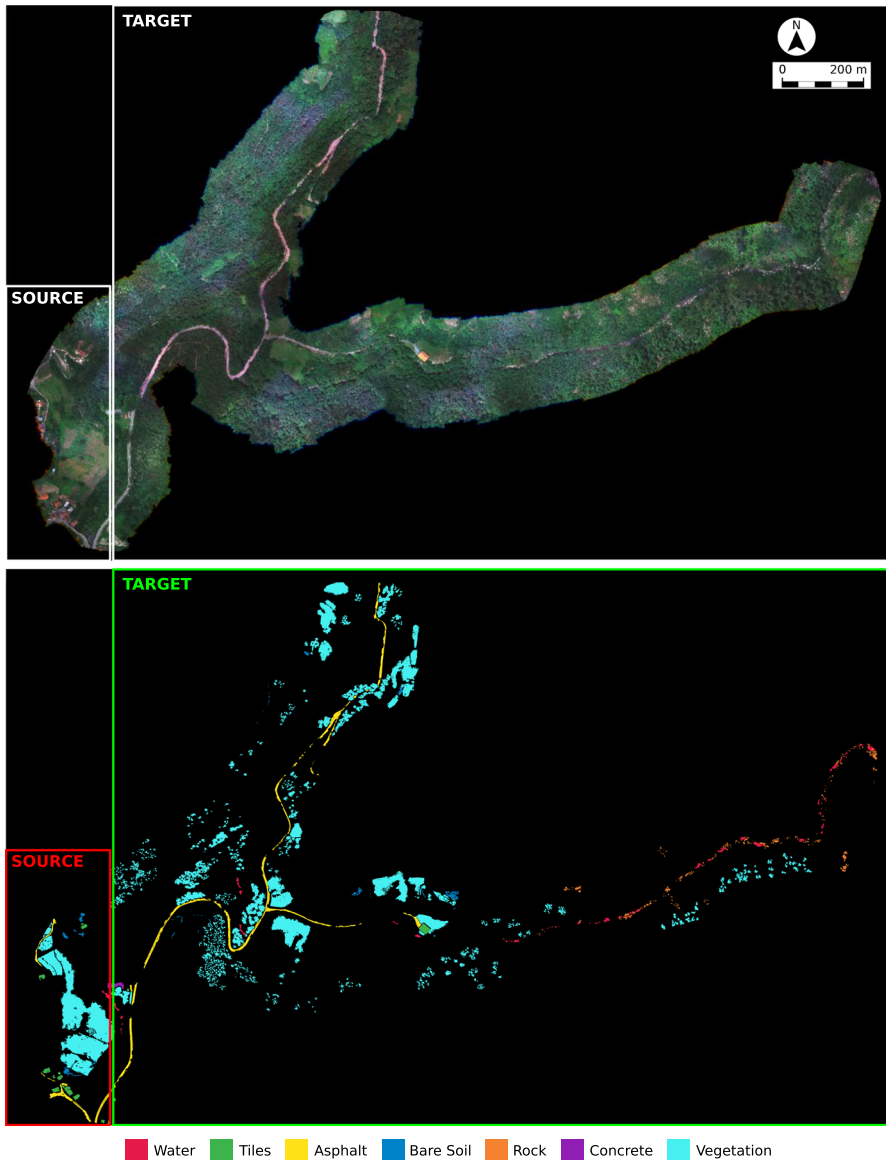
Fig. 5 Eiras. False color composite and reference data for classification

## 4.2 Results

This section presents the experimental results obtained by applying the hybrid CUDA, OpenMP, and MPI parallel TCANet-based classification scheme over the three very high-resolution multispectral datasets described in the previous section.

Table 5 shows a summary of results for the three datasets. All the results presented were obtained after averaging ten different executions. The setting of parameters was selected to be the same for the three datasets in order to have the same number of iterations and make the comparison of execution times for the different datasets of increasing sizes easier. Therefore, the parameter setting is not tuned for achieving the best classification accuracies for each image. The execution times for training and test are aggregated in the Table. The speedup of the OpenMP code with one thread is calculated with respect to the time of the MATLAB code. The speedups of the OpenMP version with 64 threads and of the final hybrid version using MPI, openMP, and CUDA are calculated over the OpenMP version executed with one thread. The lowest execution times are obtained for the version exploiting nodes, cores, and GPUs. It can be seen that the bigger the image the higher the speedups achieving values of up to 40.53 $\times$  over the OpenMP code with one thread.

The Ermidas image has been selected for a more detailed description of the times for all the steps involved in the computation of the proposed scheme, as it is the largest one, which implies the highest computational cost. Table 6 shows the execution times (in seconds) for both the training and the test phases of the scheme applied to Ermidas and the OA achieved by the classification scheme (88.88%). All the implementations presented offered the same OA value as the initial MATLAB version of the code. The execution time was split following the different algorithm stages as shown in Algorithm 1. For training, the steps are: patch extraction, TCA iterations (two iterations were selected in this



**Fig. 6** Ermidas. False color composite and reference data for classification

case which would be equivalent to a two-layer CNN), feature extraction, and classification training. For the test stage, the steps are the same but it is not necessary to compute the TCA filters (Step II) since this process is only performed in the training stage.

For the initial MATLAB code, 16 cores were used, obtaining the highest computation time. For the OpenMP code (denoted as OMP in the figure), different numbers of threads have been used: 1, 16, 32, and 64. It can be observed that the use of OpenMP reduces the

**Table 1** Oitavèn. Samples of the source and the target regions

Classes	Source		Target	
	2501 × 1898		6689 × 4701	
	Samples	%	Samples	%
Water	3944	0.16	283219	8.01
Tiles	16787	0.68	61093	1.73
Asphalt	11845	0.48	27413	0.76
Bare Soil	32417	1.32	29898	0.85
Rock	141373	5.74	233494	6.60
Concrete	5414	0.22	40767	1.15
Vegetation	2249483	91.40	2859168	80.88
Total	2461263		3535052	

**Table 2** Eiras. Samples of the source and the target regions

Classes	Source		Target	
	1851 × 971		5176 × 17250	
	Samples	%	Samples	%
Water	43480	15.52	691137	18.56
Tiles	1386	0.46	6846	1.18
Asphalt	84276	28.13	933	0.03
Bare Soil	675	0.23	96260	2.56
Rock	6601	2.20	138199	3.71
Concrete	5844	1.95	21217	0.57
Vegetation	157291	52.51	2769021	74.36
Total	299553		3723613	

**Table 3** Ermidas. Samples of the source and the target regions

Classes	Source		Target	
	5824 × 2701		11924 × 16271	
	Samples	%	Samples	%
Water	13291	0.72	150639	2.98
Tiles	100576	5.42	38102	0.75
Asphalt	139710	7.53	597699	11.80
Bare Soil	49238	2.66	74178	1.47
Rock	0	0	174088	3.43
Concrete	13145	0.71	19721	0.39
Vegetation	1538427	82.96	4009663	79.18
Total	1854387		5064090	

**Table 4** Values of the parameters defined in Algorithm 1 for the different datasets

Parameter	Oitavén	Eiras	Ermidas
D	3	3	3
B	5	5	5
$N_S$	14000	12061	12000
$N_T$	14000	12200	12000
$K$	2	2	2
$F_1$	2	2	2
$F_2$	16	16	16
$N_{\text{Test}}$	3521052	3693413	4878002

**Table 5** Execution times (in seconds) and speedups for the different datasets

Dataset	MATLAB	OMP		OMP		OMP	
		1 threads		64 threads		GPU	
						MPI	
Oitavén	64273.89	3290.54	<b>19.53×</b>	154.68	<b>21.27×</b>	91.98	<b>35.77×</b>
Eiras	64050.72	1965.72	<b>32.58×</b>	100.33	<b>19.59×</b>	56.56	<b>34.75×</b>
Ermidas	83036.34	2323.07	<b>35.74×</b>	108.18	<b>21.47×</b>	57.31	<b>40.53×</b>

execution time. The version of the code using three nodes and exploiting the GPUs of the nodes by using CUDA (denoted in the table as OMP GPU MPI) also uses 64 cores and achieves the lowest execution times. As it can be observed, the hybrid parallelization strategy mainly impacts the feature extraction and the classification steps in the test as they have the highest execution times. The reason is that they are executed over each single test sample being the test samples all the available samples of the target image excluding those used for training. A similar behavior was observed for all the images.

## 5 Conclusions

In this paper, a hybrid CUDA, OpenMP, and MPI parallel TCANet-based supervised classification scheme is presented. It is applied to very high-resolution remote sensing multispectral datasets. TCANet is a DA technique that allows extending knowledge from one source image to the classification of a different image, corresponding in our case to a different geographical location. It is designed as a DL network similar to a standard CNN. Several stages built based on convolutional filters operate on patches of the multispectral image. The filters are calculated by using the TCA feature extraction algorithm. The resulting DL network does not require backpropagation. Even with this feature, the computational cost of the classification scheme is high, as it operates over a large number of pixels. The application of the method requires a final step of supervised classification performed by SVM in the experiments. Different optimizations and a parallel implementation exploiting the different levels



**Table 6** Execution times in seconds (training and test stages) and classification accuracy for the Ermidas dataset

	MATLAB	OMP 1 thread	OMP 16 threads	OMP 32 threads	OMP 64 threads	OMP GPU MPI
<b>Training</b>						
Patch Extr.	8.07	0.02	0.01	0.01	0.01	0.01
TCA iteration 1						
· Steps I and II	7.58	8.28	2.99	2.78	2.50	2.46
· Step III	2.60	0.33	0.04	0.04	0.02	0.02
TCA iteration 2						
· Steps I and II	30.72	35.95	13.07	11.26	9.43	9.99
· Step III	419.25	0.42	0.05	0.05	0.04	0.04
Feature Extr.	412.54	0.04	0.01	0.01	0.01	0.01
Classification	24.70	15.24	16.54	17.01	16.58	16.76
<b>Test</b>						
Patch Extr.	728.98	1.87	0.59	0.55	0.57	0.65
TCA iteration 1						
· Steps I and III	3679.39	67.59	5.35	3.54	·	1.06
TCA iteration 2						
· Steps I and III	38302.26	238.45	40.80	28.21	18.47	5.96
Feature Extr.	36834.32	22.20	4.51	3.78	4.09	1.41
Classification	2585.93	1932.68	139.12	115.31	54.24	18.94
Total	83036.34	2323.07	223.08	182.55	108.18	57.31
Overall accuracy	88.88 %					

of hardware available in the heterogeneous nodes of a supercomputer are applied. In particular, a multi-node, multicore, and multi-GPU implementation is presented and tested over real datasets captured by the MicaSense RedEdge sensor onboard drones. The whole procedure requires two stages: training and testing. The testing is the stage mainly affected by the reduction in execution time provided by the parallel execution, in particular, the steps devoted to TCA filter application, feature extraction, and classification by SVM. Speedups up to 40.53× are achieved when the code is executed over nodes of the Finisterrae III supercomputer. The size of the matrices involved in the operations of TCANet increases with the number of TCA iterations (equivalent to the number of layers in a standard CNN). This fact limits the depth of TCANet when it is applied in a single processor and could limit the performance for some datasets. The presented parallel implementation could be scaled to larger images and for more complex DA cases, making the application to real scenarios possible.

**Acknowledgements** The authors want to acknowledge Centro de Supercomputación de Galicia (CESGA) for providing access to the supercomputer Finisterrae III, and to RSIM group from TU Berlin.

**Author contributions** Conceptualization and methodology FA and DBH; software and experimentation ASG; validation ASG, FA, DBH, and BD; writing ASG, DBH, and FA. All authors have read and agreed to the published version of the manuscript.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was supported in part by the Ministerio de Ciencia e Innovación, Government of Spain (grant numbers PID2019-104834GB-I00 and TED2021-130367B-I00), the Consellería de Educación, Universidade e Formación Profesional (grant number 2019–2022 ED431G-2019/04 and 2021–2024 ED431C 2022/16), and by the Junta de Castilla y León (project VA226P20 (PROPHET II Project)). All are co-funded by the European Regional Development Fund (ERDF). Alberto S. Garea acknowledges USC for its “Convocatoria de Recualificación do Sistema Universitario Español - Margarita Salas” postdoctoral fellowship under the “Plan de Recuperación y Transformación” program funded by the Spanish Ministry of Universities with the European Union’s NextGeneration funds.

**Availability of data and materials** Supplementary data is available at [https://gitlab.citius.usc.es/hiperespectral/tcanet\\_jos\\_2022](https://gitlab.citius.usc.es/hiperespectral/tcanet_jos_2022).

## Declarations

**Conflict of interests** The authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

**Ethical approval** Not applicable.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Mehmood M, Shahzad A, Zafar B, Shabbir A, Ali N (2022) Remote sensing image classification: a comprehensive review and applications. *Math Problems Eng*. <https://doi.org/10.1155/2022/5880959>
2. Cheng G, Han J, Lu X (2017) Remote sensing image scene classification: benchmark and state of the art. *Proc IEEE* 105(10):1865–1883
3. Chutia D, Bhattacharyya D, Sarma KK, Kalita R, Sudhakar S (2016) Hyperspectral remote sensing classifications: a perspective survey. *Trans GIS* 20(4):463–490
4. Benediktsson JA, Chanussot J, Moon WM (2012) Very high-resolution remote sensing: challenges and opportunities [point of view]. *Proc IEEE* 100(6):1907–1910
5. Maxwell AE, Warner TA, Fang F (2018) Implementation of machine-learning classification in remote sensing: an applied review. *Int J Remote Sens* 39(9):2784–2817
6. Tong X-Y, Xia G-S, Lu Q, Shen H, Li S, You S, Zhang L (2020) Land-cover classification with high-resolution remote sensing images using transferable deep models. *Remote Sens Environ* 237:111322
7. Jensen RR, Hardin PJ, Yu G (2009) Artificial neural networks and remote sensing. *Geogr Compass* 3(2):630–646
8. Abiodun OI, Jantan A, Omolara AE, Dada KV, Mohamed NA, Arshad H (2018) State-of-the-art in artificial neural network applications: a survey. *Heliyon* 4(11):00938
9. Cheng G, Xie X, Han J, Guo L, Xia G-S (2020) Remote sensing image scene classification meets deep learning: challenges, methods, benchmarks, and opportunities. *IEEE J Sel Topics Appl Earth Observ Remote Sens* 13:3735–3756. <https://doi.org/10.1109/JSTARS.2020.3005403>
10. Hu W, Huang Y, Wei L, Zhang F, Li H (2015) Deep convolutional neural networks for hyperspectral image classification. *J Sens*

11. Yue J, Zhao W, Mao S, Liu H (2015) Spectral–spatial classification of hyperspectral images using deep convolutional neural networks. *Remote Sens Lett* 6(6):468–477
12. 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 Trans Geosci Remote Sens* 54(10):6232–6251
13. Lu D, Weng Q (2007) A survey of image classification methods and techniques for improving classification performance. *Int J Remote Sens* 28(5):823–870
14. Quinonero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2008) *Dataset shift in machine learning*. The MIT Press, Cambridge
15. Jia G, Hueni A, Schaeppman ME, Zhao H (2017) Detection and correction of spectral shift effects for the airborne prism experiment. *IEEE Trans Geosci Remote Sens* 55(11):6666–6679
16. Congalton RG (1991) A review of assessing the accuracy of classifications of remotely sensed data. *Remote Sens Environ* 37(1):35–46
17. Zhuang F, Qi Z, Duan K, Xi D, Zhu Y, Zhu H, Xiong H, He Q (2020) A comprehensive survey on transfer learning. *Proc IEEE* 109(1):43–76
18. Tuia D, Persello C, Bruzzone L (2016) Domain adaptation for the classification of remote sensing data: an overview of recent advances. *IEEE Geosci Remote Sens Mag* 4(2):41–57
19. Weiss K, Khoshgoftar TM, Wang D (2016) A survey of transfer learning. *J Big Data* 3(1):9
20. Tan C, Sun F, Kong T, Zhang W, Yang C, Liu C (2018) A survey on deep transfer learning. In: *International conference on artificial neural networks*. Springer, Berlin, pp 270–227
21. Tuia D, Persello C, Bruzzone L (2021) Recent advances in domain adaptation for the classification of remote sensing data. *arXiv preprint arXiv:2104.07778*
22. Glorot X, Bordes A, Bengio Y (2011) Domain adaptation for large-scale sentiment classification: a deep learning approach. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp 513–520
23. Sun B, Feng J, Saenko K (2016) Return of frustratingly easy domain adaptation. In: *Proceedings of the AAAI conference on artificial intelligence*, vol 30
24. Song S, Yu H, Miao Z, Zhang Q, Lin Y, Wang S (2019) Domain adaptation for convolutional neural networks-based remote sensing scene classification. *IEEE Geosci Remote Sens Lett* 16(8):1324–1328
25. Hecht-Nielsen R (1992) Theory of the backpropagation neural network. In: *Neural networks for perception*. Elsevier, Amsterdam, pp 65–93
26. Chan T-H, Jia K, Gao S, Lu J, Zeng Z, Ma Y (2015) Pcanet: a simple deep learning baseline for image classification? *IEEE Trans Image Process* 24(12):5017–5032
27. Fauvel M, Chanussot J, Benediktsson JA (2009) Kernel principal component analysis for the classification of hyperspectral remote sensing data over urban areas. *EURASIP J Adv Signal Process* 2009:1–14
28. Garea AS, Heras DB, Argüello F (2019) TCANet for domain adaptation of hyperspectral images. *Remote Sens* 11(19):2289
29. Pan SJ, Tsang IW, Kwok JT, Yang Q (2010) Domain adaptation via transfer component analysis. *IEEE Trans Neural Netw* 22(2):199–210
30. Lee CA, Gasster SD, Plaza A, Chang C-I, Huang B (2011) Recent developments in high-performance computing for remote sensing: a review. *IEEE J Sel Topics Appl Earth Observ Remote Sens* 4(3):508–527
31. Riedel M, Sedona R, Barakat C, Einarsson P, Hassanian R, Cavallaro G, Book M, Neukirchen H, Lintermann A (2021) Practice and experience in using parallel and scalable machine learning with heterogeneous modular supercomputing architectures. In: *2021 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, pp 76–85
32. Cavallaro G, Heras DB, Wu Z, Maskey M, Lopez S, Gawron P, Coca M, Datcu M (2022) High-performance and disruptive computing in remote sensing: Hdcrs—a new working group of the GRSS earth science informatics technical committee. *IEEE Geosci Remote Sens Mag*
33. Plaza A, Du Q, Chang Y-L, King RL (2011) Foreword to the special issue on high performance computing in earth observation and remote sensing. *IEEE J Sel Topics Appl Earth Observ Remote Sens* 4(3):503–507
34. Ma Y, Wu H, Wang L, Huang B, Ranjan R, Zomaya A, Jie W (2015) Remote sensing big data computing: challenges and opportunities. *Futur Gener Comput Syst* 51:47–60
35. Liu Y, Xie Y, Yang W, Zuo X, Ge Q, Zhou B (2020) Target classification and recognition for high-resolution remote sensing images: using the parallel cross-model neural cognitive computing algorithm. *IEEE Geosci Remote Sens Mag* 8(3):50–62
36. Haut JM, Gallardo JA, Paoletti ME, Cavallaro G, Plaza J, Plaza A, Riedel M (2019) Cloud deep networks for hyperspectral image analysis. *IEEE Trans Geosci Remote Sens* 57(12):9832–9848
37. Paoletti ME, Haut JM, Fernandez-Beltran R, Plaza J, Plaza AJ, Pla F (2018) Deep pyramidal residual networks for spectral–spatial hyperspectral image classification. *IEEE Trans Geosci Remote Sens* 57(2):740–754

38. Ordóñez Á, Heras DB, Argüello, F (2022) Multi-GPU registration of high-resolution multispectral images using HSI-KAZE in a cluster system. In: IGARSS 2022—2022 IEEE international geoscience and remote sensing symposium, pp 5527–5530. <https://doi.org/10.1109/IGARSS46834.2022.9884717>
39. Borgwardt KM, Gretton A, Rasch MJ, Kriegel H-P, Schölkopf B, Smola AJ (2006) Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics* 22(14):49–57
40. Steinwart I (2001) On the influence of the kernel on the consistency of support vector machines. *J Mach Learn Res* 2:67–93
41. Pan SJ, Kwok JT, Yang Q (2008) Transfer learning via dimensionality reduction. *AAAI* 8:677–682
42. Ghamisi P, Plaza J, Chen Y, Li J, Plaza AJ (2017) Advanced spectral classifiers for hyperspectral images: a review. *IEEE Geosci Remote Sens Mag* 5(1):8–32
43. Ghamisi P, Maggiori E, Li S, Souza R, Tarablaka Y, Moser G, De Giorgi A, Fang L, Chen Y, Chi M (2018) New frontiers in spectral–spatial hyperspectral image classification: the latest advances based on mathematical morphology, Markov random fields, segmentation, sparse representation, and deep learning. *IEEE Geosci Remote Sens Mag* 6(3):10–43
44. OpenMP Architecture Review Board: OpenMP Website. <https://www.openmp.org/> (online). Accessed 8 Mar 2021
45. NVIDIA: CUDA toolkit Website. <https://developer.nvidia.com/cuda-toolkit> (online). Accessed 5 Jan 2021
46. Chang C-C, Lin C-J (2011) LIBSVM: a library for support vector machines. *ACM Trans Intell Syst Technol* 2:27–12727. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
47. Fauvel M, Tarabalka Y, Benediktsson JA, Chanussot J, Tilton JC (2012) Advances in spectral–spatial classification of hyperspectral images. *Proc IEEE* 101(3):652–675
48. Richards J, Jia X (1999) *Remote sensing digital image analysis: an introduction*. Springer, Berlin

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Authors and Affiliations

Alberto S. Garea<sup>1</sup> · Dora B. Heras<sup>1</sup> · Francisco Argüello<sup>2</sup> · Begüm Demir<sup>3</sup>

Dora B. Heras  
dora.heras@usc.es

Francisco Argüello  
francisco.arguello@usc.es

Begüm Demir  
demir@tu-berlin.de

<sup>1</sup> Centro Singular de Investigación en Tecnologías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

<sup>2</sup> Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, Spain

<sup>3</sup> Remote Sensing Image Analysis (RSiM) Group, Technische Universität Berlin, Berlin, Germany