Learning approaches for large-scale remote sensing image classification

Emmanuel Maggiori

UNIVERSITÉ CÔTE D'AZUR

informatiques mathématiques

École Doctoral STIC
Sciences et Technologies de l'Information
et de la Communication

Thèse

pour obtenir le titre de

Docteur en Automatique, Traitement du Signal et des Images

de l'Université Côte d'Azur

par

Emmanuel Maggiori

Learning Approaches for Large-Scale Remote Sensing Image Classification

Thèse dirigée par Pierre Alliez, Inria,
codirigée par Yuliya Tarabalka, Inria,
et co-encadrée par Guillaume Charpiat, Inria,
preparée à Inria Sophia Antipolis-Méditerranée,
et soutenue le 22 juin 2017.

Jury :

Pierre	Alliez	Inria	Directeur de thèse
Yuliya	Tarabalka	Inria	Codirectrice de thèse
Sébastien	Lefèvre	Université Bretagne Sud	Rapporteur
Devis	Tuia	Université de Zurich	Rapporteur
Frédéric	Precioso	Laboratoire I3S	Examinateur
Philippe	Salembier	Univ. polytech. de Catalogne	Examinateur
Guillaume	Charpiat	Inria	Invité

Abstract

The analysis of airborne and satellite images is one of the core subjects in remote sensing. In recent years, technological developments have facilitated the availability of large-scale sources of data, which cover significant extents of the earth's surface, often at impressive spatial resolutions. In addition to the evident computational complexity issues that arise, one of the current challenges is to handle the variability in the appearance of the objects across different geographic regions. For this, it is necessary to design classification methods that go beyond the analysis of individual pixel spectra, introducing higher-level contextual information in the process.

In this thesis, we first propose a method to perform classification with shape priors, based on the optimization of a hierarchical subdivision data structure. We then delve into the use of the increasingly popular convolutional neural networks (CNNs) to learn deep hierarchical contextual features. We investigate CNNs from multiple angles, in order to address the different points required to adapt them to our problem. Among other subjects, we propose different solutions to output high-resolution classification maps and we study the acquisition of training data. We also created a dataset of aerial images over dissimilar locations, and assess the generalization capabilities of CNNs.

Finally, we propose a technique to polygonize the output classification maps, so as to integrate them into operational geographic information systems, thus completing the typical processing pipeline observed in a wide number of applications. Throughout this thesis, we experiment on hyperspectral, satellite and aerial images, with scalability, generalization and applicability goals in mind.

Résumé

L'analyse des images satellite et aériennes figure parmi les sujets fondamentaux du domaine de la télédétection. Ces dernières années, les avancées technologiques ont permis d'augmenter la disponibilité à large échelle des images, en comprenant parfois de larges étendues de terre à haute résolution spatiale. En plus des questions évidentes de complexité calculatoire qui en surgissent, un de plus importants défis est l'énorme variabilité des objets dans les différentes régions de la terre. Pour aborder cela, il est nécessaire de concevoir des méthodes de classification qui dépassent l'analyse du spectre individuel de chaque pixel, en introduisant de l'information contextuelle de haut niveau.

Dans cette thèse, nous proposons d'abord une méthode pour la classification avec des contraintes de forme, basée sur l'optimisation d'une structure de subdivision hiérarchique des images. Nous explorons ensuite l'utilisation des réseaux de neurones convolutionnels (CNN), qui nous permettent d'apprendre des descripteurs hiérarchiques profonds. Nous étudions les CNN depuis de nombreux points de vue, ce qui nous permettra de les adapter à notre objectif. Parmi les sujets abordés, nous proposons différentes solutions pour générer des cartes de classification à haute résolution et nous étudions aussi la récolte des données d'entrainement. Nous avons également créé une base de données d'images aériennes sur des zones variées, pour évaluer la capacité de généralisation des CNN.

Finalement, nous proposons une méthode pour polygonaliser les cartes de classification issues des réseaux de neurones, afin de pouvoir les intégrer dans des systèmes d'information géographique. Au long de la thèse, nous conduisons des expériences sur des images hyperspectrales, satellites et aériennes, toujours avec l'intention de proposer des méthodes applicables, généralisables et qui passent à l'échelle.

Contents

Acknowledgements

I wish to express my sincere gratitude to my advisors Pierre, Yuliya and Guillaume for their continuous support and direction during my thesis. It has been deeply satisfying to work with you in this project over the past few years. I would like to stress in particular my gratitude to Yuliya for her invaluable companionship and advice on my career and life in general.

I extend my recognition to everyone else who has contributed in their own way to making this project possible.

To you all: Thank you! Merci ! ¡Muchas gracias!

Chapter 1

Introduction

Over the last few years, technological developments have significantly increased the resolution and availability of remotely sensed images. For example, the constellation of Pléiades satellites provide images with a spatial resolution of less than a meter, revisiting the entire earth every day. It has thus become extremely important to develop techniques to automatically analyze such data.

In this setting, one of the first challenges is to adapt to the characteristics of the new sensors, since traditional methods may not be directly applicable. For example, for many years significant research was focused on discovering the materials *mixed* together in a single pixel's measurement [75]. Now that pixels became small compared to the objects, research efforts have shifted to establishing a consistency in the analysis of the different pixels that belong to the same object. A second challenge is the scalability of the systems. Since there is more and more access to images covering a large geographic extent, there is a growing interest in developing *scalable* methods, even to process the entire earth at once [102]. While there is certainly a strong need for scalability in terms of computational efficiency [104, 80], it is also a challenge to design methods that consistently yield good results at dissimilar areas of the earth, given the variability of the appearance of landscapes across the planet. This variability is well appreciated in the examples of aerial imagery depicted in Fig. 1.1. The type of analysis required has also shifted from low-level physical materials (e.g., *metal sheets* [10]) to more abstract semantic classes (e.g., *cars* [155]).

The automatic interpretation of remotely sensed imagery is usually formulated as a problem of pixelwise *classification*, which consists in the assignment of a class (e.g., *building*, *bricks*, etc.) to each pixel in the image. Alternatively, one can also formulate the problem as the subdivision of the image into segments, and the assignment of a class

Figure 1.1: Examples of aerial images around the earth.

to every segment. The second formulation highlights the existence of semantic objects, which is particularly useful to conduct an *object-based* analysis [12] on the image. In this thesis we focus on *supervised* classification, in which the classifier is trained from labeled reference data.

Let us remark that outside remote sensing, the term *classification* usually denotes the assignment of a category to an entire image (e.g., *animal*, *vehicle*, etc.), which is similar to the land cover classification problem in remote sensing (e.g., *agricultural*, *urban*, etc.). To denote the classification of individual pixels, the terms *pixelwise classification*, *dense classification* or *scene parsing* are used instead in computer vision literature, or even *semantic segmentation* to formulate it from an object-based perspective. Moreover, the terms *classification* and *labeling* tend to be used interchangeably.

1.1 Objective

This thesis seeks to incorporate high-level cues into the remote sensing image classification process, such as shape descriptors and deep learned features, in order to facilitate the analysis of images at a large scale, and in complex and realistic datasets.

In the following sections we first introduce remote sensing imagery, and then summarize our contributions and publications.

1.2 Remote sensing images

Remote sensing is usually defined, and to the purpose of this thesis, as the measurement of object properties on the earth's surface using data acquired from aircraft and

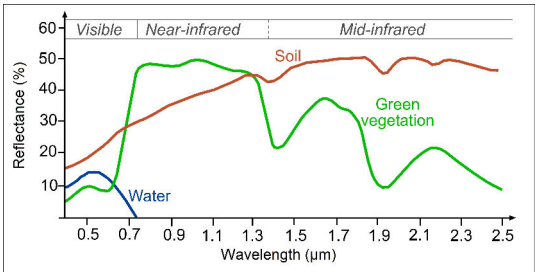


Figure 1.2: Spectral signatures of different materials.

satellites [129]. We often discuss *image resolution*, but let us remark that there are in fact three types of resolution: *spatial*, *spectral* and *temporal*. Spatial resolution is related to the pixel spacing on the earth's surface, or *ground sampling distance*. For example, while older satellites provided image products of several meters per pixel, we now find satellite imagery with pixels less than half a meter apart. Spectral resolution is related to the difference in wavelengths that can be distinguished. Notably, instead of the three traditional RGB channels of color images, some sensors provide hundreds of bands and span beyond the visible spectrum. Finally, temporal resolution relates to the time lapse between acquisitions over the same area, and is particularly relevant, for example, in the context of change detection [156].

We now introduce some of the most common types of remote sensing images.

1.2.1 Hyperspectral images

The different materials over the earth can be characterized by the way they reflect the sun's radiation. For a given wavelength of light incident to the surface, a percentage of that light is reflected and captured by the remote sensor. One can plot the reflectance values for different wavelengths, referred to as the *spectral signature*, and potentially distinguish the materials based on the curve. Fig. 1.2 illustrates the spectral signatures of different materials (source: SEOS Project).

Hyperspectral images are aerial images obtained from an aircraft equipped with a hyperspectral remote sensor, with the goal of estimating the spectral signature of the materials. For this, they allow the simultaneous acquisition of hundreds of spectral bands with narrow contiguous bandwidths for each image pixel. For example, the AVIRIS sensor provides images with 224 contiguous bands with a bandwidth of 10 nm each, and the ROSIS sensor provides 115 bands with a bandwidth of 4 nm each. In the spectral domain, pixels are represented as vectors for which each component is a measurement corresponding to specific wavelengths. The size of the vector equates the number of spec-

Table 1.1: Spectral bands of the sensor in Pléiades satellites 1A and 1B.

Band	Wavelength
Blue	430–550 nm
Green	490–610 nm
Red	600–720 nm
Near Infrared	750–950 nm
Panchromatic	480-830 nm

tral bands that the sensor collects. Over a hundred of bands are typically available. This detailed spectral information increases the possibility of more accurately discriminating materials of interest [26]. The capabilities of hyperspectral sensors go beyond the identification of land cover, facilitating also the characterization of minerals [29], soils [13] and biodiversity [54].

Due to their discriminating power, hyperspectral images have received a lot of attention from the remote sensing community and become the common framework to design and assess pixelwise classification methods. Let us remark, however, that due to their cost and complexity their availability is quite restricted. There are few publicly available datasets and they mostly cover a very restricted geographic extent, ranging from a university to a few neighboring agricultural fields.

1.2.2 Multi-spectral satellite images

A multi-spectral sensor acquires images in several wavelength "regions", for example, spanning both the visible spectrum and infrared wavelengths. Most satellite sensors capture this type of images. For example, the Operational Land Imager sensor on board Landsat 8 provides eleven bands, including RGB, an ultra blue band and seven bands that overlap the infrared wavelength area. Let us remark that, compared to hyperspectral images, the bands of multi-spectral images do not represent adjacent wavelengths. Instead, they are designed with specific goals in mind, often leaving gaps in the spectrum or overlapping each other. For example, one of the bands in Landsat 8 is specifically geared at spotting high-altitude clouds (cirrus clouds) which particularly degrade the observations from the other bands, hence the interest in locating them.

Notice that the characteristics of the *product* distributed by the image provider may differ from the sensor specifications. For example, a sensor may capture at a spatial resolution ranging from 0.2 m to 1 m depending on the angle of the earth's surface with respect to the satellite, while this may be resampled to yield a uniform 0.5 m in the final image product.

One particular characteristic of satellite images is that there is typically an additional

(a) Multi-spectral (RGB bands) (b) Panchromatic (c) Pansharpened

Figure 1.3: Close-up of Pléiades image.

panchromatic band that covers a wide spectrum at once (often including most of the wavelengths covered by the other bands), but at a higher spatial resolution. For example, a Pléiades satellite image product consists of four bands at 1 m spatial resolution (R, G, B and Near Infrared), and a panchromatic band at 0.5 m resolution. In Table 1.1 we summarize the spectral bands of Pléiades. In the literature, the panchromatic band is usually considered to be a separate additional band, and the term "multi-spectral" is reserved to denote all the rest of the bands. Figs. 1.3(a,b) illustrate examples of the RGB and panchromatic bands over the same geographical area. The process of combining both data sources is known as *pansharpening* [4], and could be seen as the procedure of "painting" the pansharpened image with the content of the other bands. The result of the pansharpening is shown on Fig. 1.3(c), which looks as if the spatial resolution of the RGB image had been increased. Effective pansharpening is a research domain in itself [3, 4].

During the development of this thesis we mostly dealt with the RGB bands of pansharpened satellite imagery (pansharpened by using Orfeo Toolbox[1]. This allows us to be more general and design methods that can be directly adapted to other types of images (e.g., color aerial images). Of course the specific properties of a certain type of satellite image product could be exploited to conceive more specific tools.

1.2.3 Aerial images

The interest in high-resolution aerial images has been growing in the past few years. While hyperspectral images are also captured from an aircraft, the term *aerial image* refers to simpler airborne RGB images, eventually RGB-Infrared. Their availability is significantly more important than that of hyperspectral images, but it still falls behind

[1] www.orfeo-toolbox.org

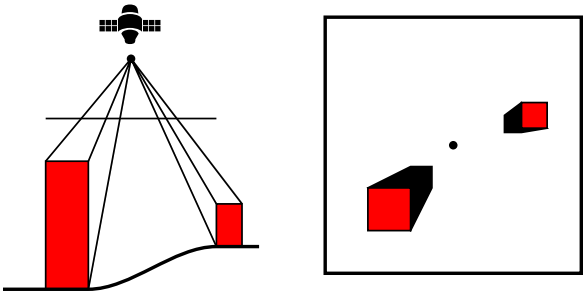


Figure 1.4: Perspective view from remote sensor.

satellite images, since dedicated flights are required every time an image over a region is requested.

There are important free access databases of aerial images. For example, the United States Geological Survey released 15–30 cm spatial resolution images over almost the entire country. Higher-resolutions images (less than 10 cm) but covering more restricted areas have also been released and quickly considered as benchmarks for the comparison of classification methods, including the Vaihingen and Potsdam datasets [70]. We previously showed some examples of aerial images in Fig. 1.1.

1.2.4 A note on perspective

The raw images captured either from satellites or from planes tend to be geographically inaccurate, and cannot be used to measure distances or be overlaid with a map. This is a result of a number of perspective distortions introduced in the capture process. One such case arises when the camera is tilted, i.e., not pointing in the *nadir* direction (right below). But even if the camera points in the nadir direction, it will, for example, capture the facades and not the rooftops of the buildings that are located far away from the nadir (see Fig. 1.4). The irregular elevation of the terrain introduces further distortions.

Both satellite and aerial images usually go through a process known as *orthorectification* [86], in which a terrain elevation model is used to align the pixels to their true geographic location. This is done at ground level, hence we still observe shifts in the top of man-made objects. The rarer *true orthophotos* [62] provide a perfect orthographic view from above, built by composing images taken from several angles, estimating the underlying 3-D structure and filling the parts of the objects that are occluded in each separate shot. Pansharpening and simple orthorectification are the typical corrections performed on satellite image products.

1.3 Contributions

The following paragraphs summarize our contributions and their organization in this thesis. Every chapter reviews relevant literature and explains the background concepts related to the development of our methods.

Chapter 2 It is known that shape descriptors (such as convexity and rectangularity) improve classification. We devise a method for shape-aware multi-object multi-class classification, based on a hierarchical data structure known as binary partition trees. For this, we propose an optimization algorithm that prunes and regrafts tree branches to incorporate shape information. We present a theoretical study about the optimization moves to reduce the search space and make optimization efficient.

Chapter 3 We study convolutional neural networks (CNNs) for classification, since they can learn complex contextual features from training data in a scalable way. We study the proper type of architecture for pixelwise classification, and discuss the sources of training data and how deal with its potential imperfections. We thus develop an operational end-to-end framework for the classification of remotely sensed images with CNNs.

Chapter 4 One of the concerns about using CNNs for pixelwise classification is the coarseness of the CNN's outputs in terms of spatial resolution. A number of post-processing iterative methods have been recently proposed to sharpen and correct the classification maps around image edges. Instead of designing such an algorithm by hand, we design a recurrent neural network that directly learns the iterative enhancement algorithm from training data.

Chapter 5 An alternative to post-processing is to design specific architectures to yield high-resolution outputs from a CNN. We thoroughly analyze the architectures recently presented in the literature and propose a novel CNN architecture that learns how to combine features at different resolutions. We also create a large-scale dataset of aerial images over significantly dissimilar urban landscapes, to assess the generalization power of CNNs.

Chapter 6 In many application domains, it is required to go beyond raster data and represent objects as polygons. To complete the processing pipeline, we thus propose a mesh approximation method to polygonize classification maps, so that they can be integrated into geographic information systems.

We make concluding remarks about our contributions and discuss possible future directions of work in Chapter 7.

1.4 Publications

Accepted

Book chapters

- E Maggiori, A Plaza, Y Tarabalka. "Models for Hyperspectral Image Analysis: from Unmixing to Object-Based Classification", *Mathematical Models for Remote Sensing Image Processing*. Springer, 2017.

Journal papers

- E. Maggiori, G. Charpiat, Y. Tarabalka, and P. Alliez. "Recurrent Neural Networks to Correct Satellite Image Classification Maps". IEEE Transactions on Geoscience and Remote Sensing. Accepted with minor revisions on March 2017.
- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification". IEEE Transactions on Geoscience and Remote Sensing. Volume 55(2), pages 645–657, February 2017.

International conference papers

- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "Polygonization of Remote Sensing Classification Maps by Mesh Approximation". 2017 IEEE International Conference on Image Processing (ICIP). September 2017.
- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Iamge Labeling Benchmark". 2017 IEEE Geoscience and Remote Sensing Symposium (IGARSS). July 2017.
- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "High-Resolution Image Classification with Convolutional Networks". 2017 IEEE Geoscience and Remote Sensing Symposium (IGARSS). July 2017.
- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "Fully Convolutional Neural Networks For Remote Sensing Image Classification". IEEE Geoscience and Remote Sensing Symposium (IGARSS). July 2016.

- E. Maggiori, Y. Tarabalka and G. Charpiat. "Optimizing Partition Trees for Multi-Object Segmentation with Shape Prior". 26th British Machine Vision Conference (BMVC). September 2015.

- E. Maggiori, Y. Tarabalka and G. Charpiat. "Improved partition trees for multi-class segmentation of remote sensing images". IEEE Geoscience and Remote Sensing Symposium (IGARSS). July 2015.

Under revision

Journal papers

- E. Maggiori, Y. Tarabalka, G. Charpiat and P. Alliez. "High-Resolution Aerial Image Labeling with Convolutional Neural Networks". IEEE Transactions on Geoscience and Remote Sensing. Accepted with minor revisions on May 2017.

Chapter 2

Classification with Binary Partition Trees

Significant efforts have been made in remote sensing classification under the assumption that an image's spectrum is very rich and discriminant. Such is the case of hyperspectral images, which seek to capture the entire spectral signature of materials. Problems such as the unmixing of a mixed spectrum into its underlying components [75] and incorporating spatial cues with respect to neighboring pixels [10, 43] have been core subjects in the community.

Over the past few years, instead of equipping satellites and airplanes with hyperspectral sensors, there has been a strong tendency toward producing images at a larger scale, matching the *big data* phenomenon observed in many other domains as well. For example, we have now free access to aerial imagery over the entire United States. However, the spectrum is not rich in these large-scale data sources, only color or color-infrared images being provided.

In this new context, we need higher-level information to properly classify images, such as the shape or texture of the objects. At the same time, there is a growing interest in extracting semantically significant objects (e.g., roads, cars) [102, 155]. Even if hyperspectral images were used, this would still require to introduce higher-level cues than just spectral signatures.

In this chapter we study the use of a shape prior to conduct classification, since it is known that the introduction of shape descriptors may significantly improve its quality [31]. While classification is usually formulated as the minimization of an objective function, or *energy*, it is difficult to optimize such an energy if it involves shape priors because of their non-local nature [85]. The state-of-the art methods require either the design

of an optimizer specific to the particular shape prior [60, 136], or a complex way of incorporating it, when possible, into energies minimizable by standard techniques [33, 150]. Moreover, in the context of remote sensing we require to treat classification as a multi-object multi-class problem, since we typically find multiple instances of the different classes in a single image.

We here propose a method for shape-aware classification, which takes into account the typical shape of object classes in terms of shape descriptors, such as rectangularity, compactness and solidity. For this, we use a hierarchical data structure, the binary partition tree (BPT) [126], which can be easily augmented to include shape information. However, their traditional greedy construction approach does not yield a good utilization of the shape constraints. We thus propose a method to iteratively optimize the structure of BPTs to produce better partitions with shape constraints. This enables us to perform multi-object multi-class classification with shape prior, and to enhance BPTs so that they better represent the underlying scenes.

2.1 Related work

Shape and optimization

In the literature regarding shape features for segmentation, we can distinguish contributions that focus on explicit shape models in a *template matching* manner and others geared at incorporating discriminative features (e.g., convexity, compactness).

In the context of *template matching*, a number of approaches have proposed to use the active contours framework [30, 87]. The evolution of the curves is usually slow and prone to get trapped in poor local minima. A second family of approaches finds global optimal solutions, but on high-dimensional specially constructed graphs. For example, to perform template matching, dynamic programming was used on a specially constructed graph that exploits the properties of a triangulation of the template [44]. Schoenemann and Cremers [128] proposed to look for minimal ratio cycles in the product graph of the input image and the shape template. These methods find a globally optimal segmentation in polynomial time on the high-dimensional graph. However, besides their complexity, these contributions are in general not suitable for fitting multiple templates in the same image [74]. Regarding graph-based optimization, a template fitting method to fit was proposed by Fredman and Zhang [48], which must be run repeatedly to account for non-rigid deformations. Other iterative models have been also proposed by coupling graphical models with shape cues [67, 77]. These approaches are computationally intensive, since

every loop contains expensive operations.

To include *discriminative features* (e.g., compactness, ellipticity), Slaubaugh and Unal [137] proposed to repeatedly fit ellipses on minimal cuts for an elliptical prior. More recent contributions have managed to express certain shape priors (star-shape [150], compactness [33, 49]) in energies minimizable by traditional s-t cuts. These approaches rely on the ability to express the shape prior in the pairwise interaction term of a Markov random field which might be, when feasible, algorithmically complex (e.g.,, analyzing intersections with a rotating discrete line around a used-defined point [150]). The trust regions framework [59] can minimize high-order functionals, and has been adapted for certain shape priors (volume, shape moments [59]). It requires to provide a linear approximation of the energy around the current solution. Gorelick et al. [60] expressed the convexity prior as the count of '1-0-1' configurations along a discrete set of lines. They proposed a linear approximation and a dynamic programming algorithm for its efficient computation. It is not clear however how to directly adapt the framework to further priors (e.g., rectangularity index) or to combine different priors under the same scheme. Moreover, most of these techniques do not contemplate the occurrence of multiple object instances. For example, multiple convex segments are indeed penalized in [60] when counting the '1-0-1' sequences in the scene.

Multi-object segmentation

To cope with the simultaneous detection of multiple objects, the marked point process framework has been used to fit an unknown number of parametric shape models. Rectangles [113] and ellipses [35] are some of the geometric elements that have been considered. The optimal solution is sought by stochastic optimization. Karantzalos and Paragios [74] have extended the active contours framework to fit multiple templates in a single image.

To our best knowledge, most recent contributions cited above (e.g., [48, 60, 150]) require to isolate every object (with prior knowledge on their location) and segment it individually.

Hierarchical trees

The notion of hierarchy has been particularly exploited in several image analysis applications, such as the detection of objects by its parts [45] and depth ordering [115].

A number of data structures have been designed to represent images as a hierarchy of regions, such as min/max-trees [127], α-trees [139, 84] and binary partition trees [126]. In this thesis we focus on binary partition trees (BPTs), which have been particularly

useful in various domains including remote sensing. Multi-label classifications can be extracted efficiently from a BPT by performing a *cut* on the tree that covers all pixels at arbitrary scales. Recent works have explored the use of shape descriptors during the cuts [149, 152].

BPTs are constructed by successively merging regions with similar colors. Even though BPTs can constitute a good hierarchical approximation to the underlying structure of an image, the bottom-up approach propagates and amplifies the errors produced at the lower scales. As a result, it is very likely that nodes in the BPT will not represent complete significant objects [94, 152]. Previous works mitigated this problem by adding penalties, such as the growth of perimeters [152], the elongation of the regions [78] or edge information [145]. The authors of [152] fitted templates on partially detected objects. These approaches can only alleviate the effect.

As another consequence of the bottom-up approach, shape information cannot be used during construction: the ultimate shape of an object in a branch cannot be predicted by a portion of it. As a result, the criteria used at construction and processing of the trees are different (as in [149, 152]), which limits the feasibility of the tool *as it is* to perform segmentation with shape priors.

Optimization of hierarchical trees

Optimization of hierarchical trees has been carried out in the area of computational phylogenetics, on the construction of *phylogenetic trees*. These trees represent the evolutionary relationships among species [88]. Several common optimization algorithms (hill climbing, simulated annealing, genetic algorithms) have been applied on the tree structures [55]. The standard moves (known as *branch-swapping* or *swappers*) are nearest neighbor interchange, subtree pruning and regrafting (SPR) and tree bisection and reconnection (TBR). SPR is the pruning/paste of a subtree into another location, while TBR rearranges the subtree before pasting. The most common objective is to maximize the *parsimony* of the tree, i.e., to explain the observed data with the least evolutionary change. We have incorporated the idea of regrafting tree branches. Our optimization objective is however different, and our context requires to define moves that preserve the parent, child and spatial adjacency relations of BPTs.

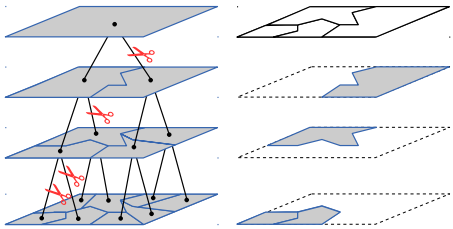


Figure 2.1: A binary partition tree (BPT) is a hierarchical subdivision of an image. An exhaustive partitioning can be extracted by "cutting" branches at different scales.

2.2 Background on binary partition trees

Binary partition trees (BPTs) were pioneered by Salembier and Garrido [126] as a means to represent a set of meaningful image regions in a compact and structured manner. The root node corresponds to the entire image, the following level represents the subdivision of the entire image into two disjoint regions, and so on. It represents then a hierarchical abstraction of an image, which can be navigated to extract meaningful regions at different scales. The typical workflow involves an initial tree construction stage, followed by a second stage of information extraction from the tree. For example, once a tree is constructed, an exhaustive segmentation of the image can be obtained by performing a horizontal "cut" on the structure (see Figure 2.1). In this procedure, commonly referred to as *pruning*, branches can be selected at different scales, an inherent advantage of such hierarchical structure. Visual browsing [126], object localization [152] and depth ordering [115] are some of the application domains where BPTs have been used, beyond remote sensing.

The construction of a BPT is performed in a bottom-up fashion, by iteratively clustering pairs of similar regions. The starting point is an initial subdivision of the image represented by a region adjacency graph (RAG), where every node conveys a region and the edges link spatial neighbors (i.e., candidates for merging). The typical initial RAG is the pixel grid, though nothing prevents the approach from being used with other inputs too (e.g., a RAG of small regions containing similar pixels, known as superpixel segmentation). Every edge in the RAG is labeled with a *dissimilarity* value that compares the two associated regions.

BPTs are constructed by following a global mutual best fitting region merging approach [80]: at each iteration, the two most similar regions in the current subdivision are merged together (i.e., the least weighted edge out of all edges in the RAG). When a merge occurs, a new region is added to the BPT, connected to its two corresponding

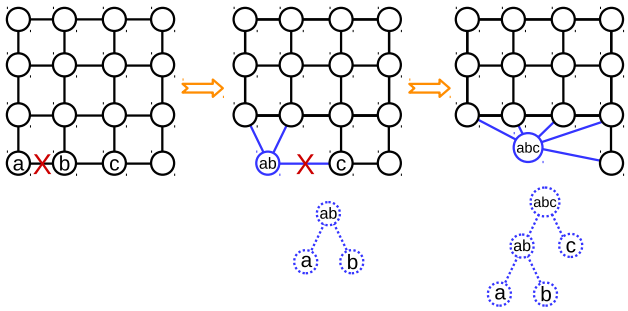


Figure 2.2: A BPT is constructed by iteratively removing edges in a region adjacency graph (RAG). The resulting BPT encodes the history of the merges.

children, as illustrated in Figure 2.2. The process finishes when there are no more edges left in the RAG. A BPT thus records the history of merges that occurred during the execution of a region merging algorithm.

The overall process can be implemented efficiently by using an updatable priority queue structure on top of the RAG edges to keep track of the highest priority element. Such a structure is first constructed in linear time and every subsequent update incurs in a logarithmic time cost. When two regions R_1 and R_2 are merged into a new region R_{12}, one must update the RAG (and the associated priority queue). The edge connecting R_1 and R_2 must be removed, but let us also remark that all the edges adjacent to R_1 and R_2 must also be eliminated from the RAG, since none of both regions exists anymore. We must then add the adjacency relations of the new region R_{12}. The computation is straightforward: the neighbors of the new region R_{12} are nothing but the union of the neighbors of the old R_1 and R_2 (with special care to remove any duplicates that may arise). The dissimilarity value associated to each of these edges must be computed and pushed to the priority queue. The complexity of the overall BPT construction process is $O(n \log(n) M)$, n being the initial number of nodes and M the maximum number of neighbors of a merged region during the construction. Given that typically $M \ll n$, the algorithm is quasilinear in practice. Recent advances have further accelerated the algorithm [1]. Among other improvements, the priority queue only stores the edge with the lowest dissimilarity value for each node. The other edges are brought back into consideration if the surrounding nodes are affected by a merge.

The final tree contains exactly $2n - 1$ nodes, which is a very space-efficient representation. Let us remark though that only a subset of all possible planar subdivisions is represented by the tree, which highlights the importance of research efforts to construct a good initial tree that conveys meaningful objects of the underlying image.

The key elements to define the behavior of a BPT are the *region model*, i.e. how regions are represented, and the *dissimilarity function*, i.e. the function to compare the region models, used to define the priority of the merges during tree construction. We review next the contributions related to these two elements.

Region model

The object-based nature of BPTs yields rich representations of the region that go beyond pixel spectra. Every BPT node can convey *regional* information, describing the region as a whole and not as a set of individual pixels. Examples of the regional data that can be associated to every node are the standard variation of the spectral signatures in the region, or shape features such as compactness.

To represent the spectrum of a region (and then compare it to the spectra of other regions) there are essentially two alternatives: parametric and non-parametric models. A parametric model makes assumptions about the homogeneity or Gaussian distribution inside the regions. A typical parametric model is to represent the spectrum of a region as the mean spectrum of its pixels. Non-parametric models, on the contrary, consist of per-band histograms of the pixel values, hence they represent the real observed distributions. In hyperspectral imagery, non-parametric models have a better performance since they can describe the internal variability of a region [148]. For example, a texture might correspond to several peaks in the histogram. When averaging spectra in regions with high variability, one might end up representing the region with a "false" spectrum that is not present in any of the individual pixels. In addition to spectral data, the model usually stores the area of the region, since it is commonly used in the dissimilarity function.

Dissimilarity function

To establish a priority for merging during BPT construction, it is required to provide a means to compare models of two regions. A dissimilarity function $O(R_1, R_2)$ typically used for this purpose comprises two factors as follows:

$$O(R_1, R_2) = \min(|R_1|, |R_2|)^{\beta} D(R_1, R_2), \tag{2.1}$$

where $|R_i|$ denotes the area of region R_i. The first part of (2.1), $\min(|R_1|, |R_2|)^{\beta}$, is the so-called *area-weighting* factor. This is an agglomerative force intended to cluster regions that are very small compared to the rest of the elements in the RAG. When no area-weighting is used (i.e., $\beta = 0$), the resulting BPT might isolate small noisy areas

and connect them to the rest only near the root of the tree. With moderate values of β, small regions are merged at some point, forcing the tree to better look like a hierarchical subdivision. When β is too large, the trees may become too biased toward being balanced, hampering their representation capabilities. Even though this parameter is barely discussed in the literature, being mostly set to $\beta = 0.5$ or $\beta = 1$, we must point out that it is an arbitrary parameter that has to be selected. In our experience, no area-weighting leads to poor representations (e.g., the root containing two children: one noisy pixel and all the rest of the image), while low values of β solve this issue without biasing the trees too much. Alternatively, Calderero and Marques [18] proposed to keep track of the out-of-scale regions and force their merging at some point, while Valero et al. [148] used a weighted sum of pixel values in a window to initialize the histograms, as a way of smoothing out outliers.

The second factor, $D(R_1, R_2)$, compares both regions based on their spectra. Kullback-Leiber divergence and Bhattacharyya distance are popular choices both in hyperspectral imagery and other types of images [18, 148]. Spectra are seen as probability distributions and compared using standard information theory concepts. Every bin of one histogram is compared against the corresponding bin of the other histogram. However, using cross-bin measures, which go beyond individual bins, has proven to be more robust [148]. The average of Earth Mover's Distances [122] among histograms of all bands has also been used as a robust and efficient cross-bin dissimilarity function [115]. Every distribution is seen as a pile of dirt, and the difference between two distributions is seen as the amount of work required to turn one pile into the other one.

2.3 Proposed method

In the following, we first pose our problem as the minimization of an energy, based on probability distributions of spectrum and shape features. We then describe how we build an initial BPT and process it to find the best partition represented by this tree. Finally, we describe an optimization algorithm to modify the initial BPT, in order to extract a classification that minimizes our energy.

2.3.1 Energy formulation

Let $I = (I_j)_{1 \leqslant j \leqslant n}$ be an input image containing n pixels. We assume we are given a set of possible object classes, as well as priors for each class. Multi-label segmentation consists in an exhaustive partitioning of the pixels into a non-overlapping set of regions

$\mathcal{R} = (R_i)$, together with associated class labels $\mathcal{L} = (L_i)$, where labels L_i belong to a set Ω of available labels. It can be stated as an optimization problem: minimize

$$E(\mathcal{R}, \mathcal{L}) = E_C(I, \mathcal{R}, \mathcal{L}) + \sum_{R_i \in \mathcal{R}} E_S(R_i, L_i), \tag{2.2}$$

where E_C expresses the color prior (quantifying how the segmentation fits the image spectrum), and E_S, the shape prior.

2.3.1.1 Color prior

For each object class, we suppose we are given training examples, from which the color distribution can be estimated and used as a prior. Given a candidate segmentation $(\mathcal{R}, \mathcal{L})$, the color prior is defined as follows:

$$E_C(I, \mathcal{R}, \mathcal{L}) = \sum_{R_i \in \mathcal{R}} \sum_{I_j \in R_i} -\log P(L_i|I_j). \tag{2.3}$$

One way of obtaining the posterior $P(L_i|I_j)$ is to train classifiers based on the samples' colors, using support vector machines (SVM), and to extend them to output probability estimates as usual in classification problems [162].

2.3.1.2 Low-complexity shape features in BPTs

Similarly, the shape prior term is defined as follows:

$$E_S(R_i, L_i) = -|R_i| \log P(L_i|\mathbf{S}_i), \tag{2.4}$$

$|R_i|$ being the area of region R_i, and $P(L_i|\mathbf{S}_i)$ being the probability of assigning the label L_i to the region R_i, given a vector $\mathbf{S}_i$ of shape features of that region. Common regularization (such as boundary length [111]) can be incorporated as part of this term. The weight on the area makes the per-pixel contribution of the color prior and the per-region contribution of the shape prior equally important.

We wish to enrich the nodes of BPTs by including shape information of the corresponding regions. Given that the optimization of the trees involves recomputing region descriptors, we must design a pool of features that can be computed efficiently from children nodes. In addition, the errors in estimating the shape descriptors in the finer levels should not be amplified in the upper ones.

Area can be efficiently computed by adding the areas of the children. **Rectangular-**

ity and **elongatedness** shape descriptors have been used in the context of hierarchical methods [78, 144, 149], though no details on how to efficiently implement these features are provided. An oriented rectangle B of height h and width w is said to be the *minimum area enclosing rectangle* (MAR) of a region R if it is the rectangle of minimal area that entirely contains R. Rectangularity measures the resemblance to a rectangle, and is computed as $|R|/(h \cdot w)$. Elongatedness measures the resemblance to a line, and is defined as w/h.

We propose to store the **convex hull** of the region at every node. When two regions are merged, the convex hull of the new region can be computed by merging the convex hulls of its children. This can be done in linear time in the size of the input polygons by using *rotating calipers* [146]. The MAR can be efficiently computed after it [146]. Rectangularity, elongatedness and other descriptors like **solidity** [165] (filled fraction of the convex hull) are then directly derived. In a balanced tree, which is enforced by our construction function, convex hulls incur in an $O(n \log(n))$ increase of the storage required. Their computation does not increase the complexity of tree construction (proofs in Appendix A.2).

In a discrete environment, the errors of computing the region areas converge as the areas grow, so do the convex hulls. As a consequence, the aforementioned features tend to be more precise in the upper levels of the trees.

Another useful shape descriptor is **compactness** [109], related to the resemblance to a circle. It is typically defined as $\delta R^2/(4\pi|R|)$, where δR is the perimeter of R. However, this formulation imposes difficulties in a discrete environment [109] due to the fact that the error in the estimation of δR does not converge. Li et al. [90] proved the robustness of computing compactness as $|R|^2/(2\pi I_g)$, I_g being the moment of inertia of the shape with respect to its centroid. Given that the centroid and moment of inertia of a region can be computed in constant time from the children, we propose this method to measure compactness in BPTs.

Let us assume that a probability density function $p(s|L)$ is available for every feature and class. These densities can be obtained by smoothing histograms of training samples [133]. Let us call $\mathbf{S} = s_1, ..., s_m$ a vector of shape features. Assuming features' conditional independence, we have:

$$P(L|\mathbf{S}). \tag{2.5}$$

Bayes' theorem and the density functions are used to compute the posterior probabilities

required in (2.5).

$$P(L|s_k) = \frac{p(s_k|L)P(L)}{\sum\limits_{L_j \in \mathcal{L}} p(s_k|L_j)P(L_j)}. \tag{2.6}$$

2.3.1.3 Total energy

Combining Eq. (2.2), (2.3) and (2.4), the energy criterion to minimize is formulated as:

$$E(\mathcal{R}, \mathcal{L}) = -\sum_{R_i \in \mathcal{R}}^{|\mathcal{R}|} \Big(\sum_{j \in R_i} \log P(L_i|I_j) + |R_i| \log P(L_i|\mathbf{S}_i) \Big) \tag{2.7}$$

2.3.2 Tree construction and processing

In this section we describe our proposed BPT construction approach, as well as the algorithm to extract the optimal classification from a BPT.

2.3.2.1 Supervised BPT construction

In Section 2.2, it was mentioned the region merging algorithm used to construct BPTs requires to define a *model* to represent the regions and a *dissimilarity* function to compare the regions. We choose a non-parametric model (i.e., spectral histograms) to represent the region, due to their well-known advantages (see Section 2.2). We propose, however, a modified dissimilarity function to construct the BPT.

In BPT-based classification methods (e.g., [148]), the dissimilarity measure used to construct the trees is purely based on the comparison of spectral histograms. In fact, commonly used dissimilarity functions (Eq. 2.1) always penalize the merging of dissimilar regions. Let us recall that non-parametric models were introduced to represent and compare inhomogeneous regions, useful for textures and light gradients. However, internal class variability (e.g., an object composed by areas of different spectra) is not at all considered. In an unsupervised context, where there is no notion of object class, there is little hope to deal with this, since there is no reason to cluster dissimilar regions. However, when class probabilities are available we propose to include an additional force that clusters regions belonging to the same class, despite being spectrally dissimilar. The new function is as follows:

$$O(R_1, R_2) = \min(|R_1|, |R_2|)^{\beta} \Big[(1-\alpha) D(R_1, R_2) - \alpha \log P(L_{R_1}{=}L_{R_2}) \Big]. \tag{2.8}$$

As in the original dissimilarity function (2.1), there is an area-weighting factor and an

unsupervised term $D(R_1, R_2)$, which is computed by comparing spectral histograms of regions without any preliminary training. We here use the Earth's Mover Distance, due to its robustness to changes in illumination and its efficient computation [122, 115]. Equation 2.8 adds a *supervised* term $P(L_{R_1}{=}L_{R_2}|R_1, R_2)$, the probability of assigning the same label to both regions. This way, while the unsupervised term penalizes spectral dissimilarity, the supervised term will encourage merging regions that are likely to belong to the same class. The trade-off between both terms is controlled by parameter α.

The term $P(L_{R_1}{=}L_{R_2}|R_1, R_2)$ is computed by marginalizing over the classes as follows:

$$P(L_{R_1} = L_{R_2}|R_1, R_2) = \sum_{j=1}^{K} P(L_j|R_1)P(L_j|R_2), \tag{2.9}$$

where K denotes the number of classes and $P(L_j|R_k)$, with $k \in \{1, 2\}$, represents the probability of assigning a certain label L_j to segment R_k. We must now define a way to compute $P(L_j|R_k)$ based on the posteriors of the individual pixels contained in the region. One way to do this is to compute the probability of assigning the label to all pixels, conditioned by the fact that all labels are known to be equal inside the region:

$$P(L_j|R_k) = \prod_{\mathbf{x}_i \in R_k} P(L_j|\mathbf{x}_i) \Big/ \left[\sum_{\omega_m \in \Omega} \prod_{\mathbf{x}_i \in R_k} P(\omega_m|\mathbf{x}_i) \right]. \tag{2.10}$$

Alternatively, one can estimate $P(L_j|R_k)$ by averaging the individual pixel probabilities:

$$P(L_j|R_k) = \frac{1}{|R_k|} \sum_{\mathbf{x}_i \in R_k} P(L_j|\mathbf{x}_i). \tag{2.11}$$

While the first expression is closer to a strict Bayesian interpretation, we found the second one to be a simple yet useful approximation.

By introducing (2.8) we expect to better cluster semantically significant objects, according to the classifier's output.

2.3.2.2 Best segmentation represented by a given tree

The common processing on BPTs consists in selecting the highest or lowest branches satisfying a given condition [94, 148]. However, some contributions have formulated the problem in terms of energy minimization [125, 126]. In particular, Salembier et al. [125] have interpreted segmentation as a horizontal s-t *cut* on the tree (see Fig. 2.1), i.e., with a source at every leaf and a sink at the root. Let us denote τ a tree and $C(\tau)$ the energy

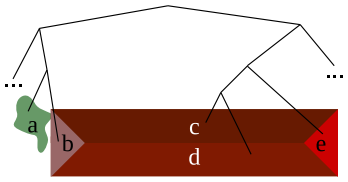


Figure 2.3: Faulty BPT: the object ($bcde$) is not represented in a single node, since a part of it (b) merged first to something else.

of the cut on τ with minimal (2.7) among all possible cuts. Our task is to find such a minimal cut.

Considering that the branches in the tree are independent, the globally optimal cut can be found by a dynamic programming algorithm. Let us denote by

$$\mathcal{E}(R) = \min_{L \in \Omega} E(\{R\}, \{L\})$$

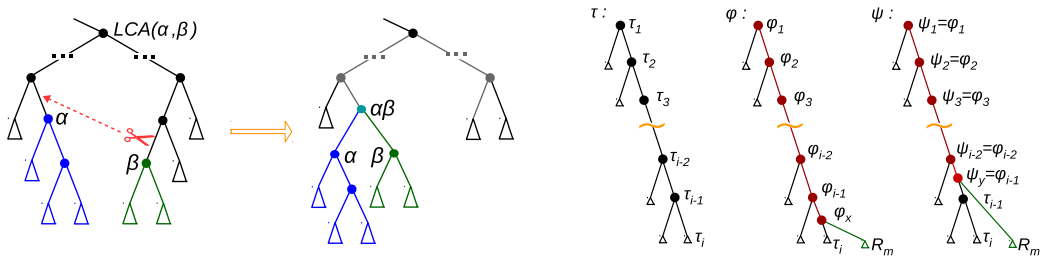
the lowest possible energy of a region R (by assigning the label that incurs the lowest cost). The tree is traversed in a bottom-up manner. Whenever a region R is visited, the following property is evaluated:

$$\mathcal{E}(R) \leqslant C(R_{left}) + C(R_{right}), \tag{2.12}$$

where R_{left} and R_{right} are the children of R. If the property does not stand, we set $C(R) = C(R_{left}) + C(R_{right})$ and keep the best cuts of both children. Otherwise, we set $C(R) = \mathcal{E}(R)$ and replace the cuts by R with label L. This process is executed recursively until reaching the root of the tree. The overall algorithm is linear in the image size, since only one BPT traversal is required, and it guarantees the optimal cut in the space of solutions represented by the BPT.

2.3.3 Optimizing the trees

Even though the globally optimal cut on a BPT can be found efficiently, the organization of the nodes in the tree structure restricts the possible cuts that can be done on them. In Fig. 2.3 a toy example illustrates this issue. Let us suppose that an aerial shot of a city captures a house with a non-uniform roof. During the construction of the tree, a and b are merged together because they feature the lower dissimilarity among every pair of regions. Even by using our improved construction function (2.8), we cannot expect much better results, unless the classifier properly labels all regions. If we wish to include shape priors to enhance the results, let us point out that at the moment a and b were



(a) *Prune-and-paste* move.

(b) Alternative paste places.

Figure 2.4: Family of moves to optimize BPTs.

merged, it was impossible to know that b would eventually form a more significant object under a different sequence of merges, with the typical shape of a building. In other words, following this greedy construction approach we cannot predict the final shape of the objects in the output classification map. The resulting tree does not allow to perform any cut that would include the whole building into the same object, not even using strong shape priors, given that it is split through different branches. This is why we now propose to *optimize the tree itself*.

Our method consists in constructing an initial BPT with the usual region merging greedy algorithm, and then optimize it to extract a classification map that minimizes Eq. 2.7, thus incorporating the shape prior. To optimize BPTs we follow a local search approach, in which a solution is iteratively modified by performing local transformations on the trees, named *moves*.

2.3.3.1 Moves and associated updates

We propose a *prune-and-paste* move that prunes a branch of the tree and inserts it into another part of the tree. The pruned node must be pasted in a spatially adjacent location. Fig. 2.4(a) illustrates such a move: α is the paste place and β is the pruning place. We denote by $\mathrm{LCA}(\alpha, \beta)$ their lowest common ancestor in the tree. The move creates a new node $\alpha\beta$ in the paste side that comprises α and β. In the pruned side, the tree is collapsed after β is removed. In a balanced tree, which is encouraged by setting $\alpha > 0$ in (2.8), the number of possible moves is bounded by $O(n \log(n))$ (see proof in Appendix A). The neighborhood system is much richer than, for instance, Markov random fields (MRFs) on the pixel grid, considering that it comprises pairs of adjacent regions at several scales.

We store at each node R the branch cost $C(R)$ of the best possible cut within its branch. When applying a move as depicted in Fig. 2.4(a), it is necessary to recompute the branch cost C till the ancestry of α and β only. The rest of the branches are unaffected,

as observed in (2.12). Among the ancestry, only the nodes below $\mathrm{LCA}(\alpha, \beta)$ require to recompute their models (shape and color features), given that further up the regions represented by the nodes do not change. Thus we recompute the features (and thus $\mathcal{E}(R)$ and $C(R)$) only in that part of the tree, and for the nodes in the tree above $\mathrm{LCA}(\alpha, \beta)$ we recompute only their branch cost $C(R)$, which simply involves reassessing (2.12) without reevaluating $\mathcal{E}(R)$ nor their features.

In a balanced tree there are at most $O(\log(n))$ ancestors of α and β, and, as stated before, for some of these ancestors the model of the regions must be updated. If we denote by K the complexity of updating a model (i.e., merging two children), the computation of the new costs C is $O(K \log(n))$. Usually $K \ll \log(n)$, therefore the time is $O(\log(n))$ in practice.

The adjacency relations in a BPT can be derived from the children by observing that the ancestors of two adjacent nodes are also adjacent, until the LCA. At the finest level the adjacency is known (e.g., a 4-connected grid).

2.3.3.2 Properties of the moves

We now explore some properties of the *prune-and-paste* move. In particular, we will show that finding all possible energy decreasing moves does not require to exhaustively evaluate the energy gain of every possible move. Proofs are available in Appendix A.

Let us consider the situation of Fig. 2.4(b). In the second tree, a node was pasted at the position τ_i. We wish to compare the effect of pasting higher instead (as in the third tree). Let us denote by $\tau_i < \tau_j$ the *is-a-descendant-of* relation.

Proposition 1. *Given a tree τ, suppose a node R_m is pasted at $\tau_i < \tau_1$ leading to a new tree φ. Let us consider an alternative move that pastes R_m at τ_j, with $\tau_i < \tau_j < \tau_1$, producing a tree ψ. In the cases where either $C(\varphi_1) - C(\tau_1) \leqslant 0$ or $C(R_m) \geqslant C(\varphi_1) - C(\tau_1)$, then $C(\psi_1) \geqslant C(\varphi_1)$.*

The proposition states that, if a move reduces the energy in the branch, a higher paste place will not do better. Under certain assumptions, if the move increases the energy, pasting higher will also increase it. Intuitively, pasting lower is more general.

Proposition 2. *Let us consider a case where Prop. 1 hypotheses do not apply. There might then exist a higher paste place τ_α so that $C(\psi_1) < C(\varphi_1)$. Let us suppose that instead of pasting at τ_α we paste at τ_β, with $\tau_\alpha < \tau_\beta < \tau_1$, leading to a tree ρ. Then $C(\rho_1)$ would monotonously decrease as the paste place τ_β is located higher.*

When the hypotheses of Prop. 1 do not apply, there might exist a favorable paste place higher in the branch. However, we know that the higher it is, the most beneficial it can be. As a result, we can just consider the highest possible paste place (right below the LCA). However, any paste place between the location of the original cut and the LCA would lead to the same energy. These cases happen when it is preferable to cut the pruned node apart. The higher we paste R_m, the less we condition the way the rest of the tree must be cut.

Following these properties, an exhaustive search of energy decreasing moves can be achieved as follows: for every possible pruning place we check the energy gain of the moves for only the lowest paste places. If the move is beneficial, we keep that move as a candidate. If the move is not beneficial we can discard it as well as all the paste places in the branch (but we do this only after performing one additional check with the paste place at some point between the cut and the LCA).

2.3.3.3 Optimization algorithm

We propose the following optimization scheme, which must be iterated:

1. Construct a heap of all moves according to the branch cost variation ΔC.
2. (a) Either apply the best move, or
   (b) apply the best k moves (when still appropriate).

The first step involves exploring the whole search space, featuring an $O(n \log^2(n))$ complexity. Moves are simulated to measure the energy gain but are not applied. Propositions (1-2) can be used to reduce the execution time of this step, and the energy gain can be evaluated in parallel.

In the second step, energy decreasing moves are applied. As soon as a move is applied, the tree is restructured and the effect of some other moves may be affected. New energy decreasing moves may also arise. 2a) just applies the best move and reiterates. As a shortcut, we can just update the entries in the heap of the moves that may have been affected. This option is still costly because ΔC must be recomputed for any move that could have been possibly affected, even though in practice this might be the case for just a few of them. This approach guarantees to apply the best move each time. Considering the fact that there might be many unrelated energy decreasing moves in the tree, 2b) proposes to apply a number k of best moves, but verifying for each move that ΔC did not increase as a result of the previous transformations done on the tree. This second

approach will apply a number of independent moves first, ignoring the fact that some new energy decreasing moves might arise, which will be dealt with in the next iteration. The loop stops when there are no more moves whose ΔC is negative.

The lowest scale both on the pruning and paste side can be set to the pixel level. A coarser scale on the pruning side can be used to adjust the precision of the moves. If a coarser scale is set on the paste side, we limit the minimum object size of the partitions, since we ignore moves that lower the cut below a certain level. This can be adjusted by observing the density functions of the area feature.

2.4 Experiments

We first validate our supervised BPT construction approach. We do this on hyperspectral images, since they contain a rich and discriminant spectrum, and evaluate whether our function (2.8) indeed clusters regions together based on the classifier's probabilities. However, such classification power based solely on pixel's values is notoriously degraded in other types of images. We therefore introduce shape features in order to enhance classification on different types of inputs, including urban aerial remote sensing images.

2.4.1 Supervised BPT construction

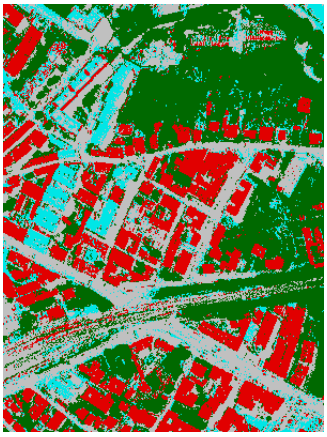
We perform experiments on the *Pavia Center* hyperspectral dataset, acquired with the Reflective Optics System Imaging Spectrometer (ROSIS-03). The image has spatial dimensions 400×300 and contains 102 bands, covering a range from 0.43 to 0.86 μm and 1.3 m spatial resolution. A color composition of the image is shown in Figure 2.5(a).

A reference image that labels entire objects was built, including four classes (see Figure 2.5-b). This reference was constructed by combining the labeling of isolated pixels provided with the original image, visual inspection and publicly available official Italian records of building boundaries. Since the boundaries of buildings are well defined, there is a particular interest in analyzing the performance of BPTs to extract buildings.

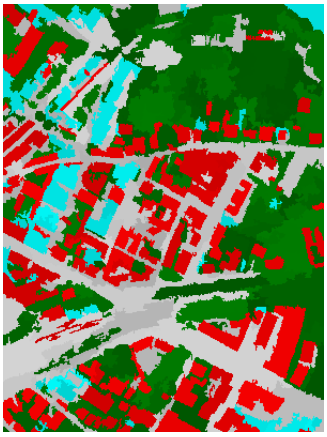
An SVM with a Gaussian radial basis function kernel is first trained on 100 randomly selected samples of each class. The SVM parameters are set by 5-fold cross validation ($C = 128$, $\gamma = 2^{-5}$). The SVM classification is shown in Figure 2.5(c). A BPT is then constructed on top of the SVM probabilities. We use a non-parametric model with 30 histogram bins per band and mild area-weighting ($\beta = 0.1$). Two variants were tested: **a)** totally unsupervised construction, i.e., setting $\alpha = 0$ in (2.8), which is equivalent to the previous function (2.1); **b)** supervised construction with equal contribution from both

(a) Color composition

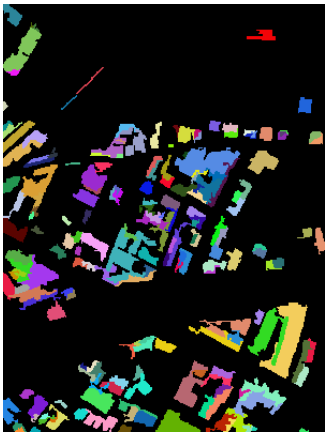
(b) Reference data: tiles, bitumen, roads , vegetation



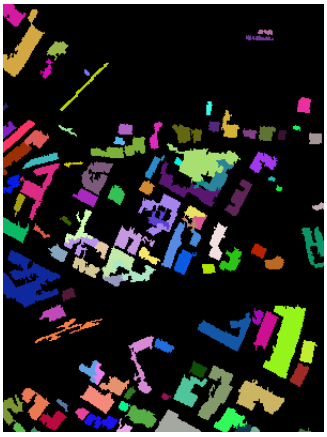
(c) SVM classification



(d) BPT classification ($\alpha = 0.5$)



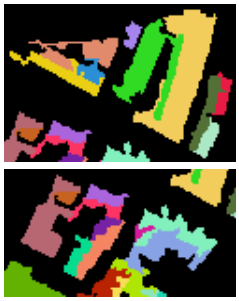
(e) Tile building detection (BPT with $\alpha = 0$)



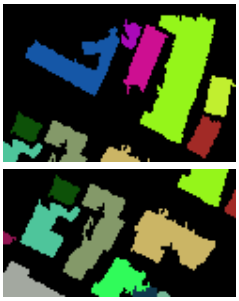
(f) Tile building detection (BPT with $\alpha = 0.5$)

Figure 2.5: Experiments on *Pavia Center* hyperspectral image.

(a) Color composition



(b) BPT with $\alpha = 0$



(c) BPT with $\alpha = 0.5$

Figure 2.6: Unsupervised (b) versus supervised (c) BPT construction. In the supervised case, regions are better clustered together to represent significant objects.

Table 2.1: Numerical evaluation on the *Pavia Center* dataset.

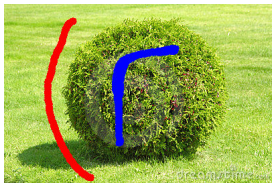
	SVM	Graph cut	$\text{BPT}_{\alpha=0}$	$\text{BPT}_{\alpha=0.5}$
Building overlap	0.51	0.51	0.54	**0.56**
Overall accuracy	0.88	**0.94**	0.91	**0.94**

terms in (2.8), i.e., $\alpha = 0.5$. Instead of using complex shape features, now we simply regularize the cut by the total number of regions:

$$E(\mathcal{R}, \mathcal{L}) = \lambda ||\mathcal{R}|| - \sum_{R_i \in \mathcal{R}} \sum_{I_j \in R_i} \log P(L_i | I_j), \tag{2.13}$$

where λ controls the coarseness of the output. We experimentally set $\lambda = 40$ to optimize the accuracy. The resulting classification map with supervised tree construction ($\alpha = 0.5$) is shown in Figure 2.5(d). We also executed a graph cut with α-expansion [17] on the probabilities derived from the SVM, which proved to be effective in the past for hyperspectral image classification [143]. Its regularity parameter was also set empirically to optimize the accuracy.

Figures 2.5(e-f) and the close-ups of Figure 2.6 compare the results obtained by applying the unsupervised and supervised approaches for BPT construction. These figures isolate the *tile* objects from the rest and assign a random color to every individual object. From these illustrations we can clearly appreciate that including class probabilities during BPT construction yields an improved tree that better clusters the objects together. To validate this numerically we compute the overlap between every building (belonging either to *tiles* or *bitumen* classes) in the reference data and the most overlapping building region in the BPT output. The overlap is measured with Dice's coefficient [36] defined as: $2|R_1 \cap R_2|/(|R_1| + |R_2|)$. The resulting overlap coefficients are averaged over all reference buildings to estimate how the BPT output matches the reference data from an object-based perspective. The numerical results, together with the overall accuracy, are summarized in Table 2.1, which also includes the values for SVM and graph cut. A first observation we can make is that $BPT_{\alpha=0.5}$ performs better than $BPT_{\alpha=0}$, corroborating the visual result from Figs. 2.5(e-f). Secondly, while graph cut is known to improve the SVM classification, we can verify that while this is true from a pixelwise perspective (in terms of overall accuracy), it is not from an object-based perspective (in terms of building overlap). Finally, the use of $BPT_{\alpha=0.5}$ outperforms the other methods in terms of object overlap. This validates the idea of including class probabilities during tree construction.



(a) Input (400 × 267)

(b) Gorelick et al.

(c) BPT Opt

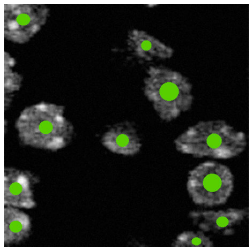
Figure 2.7: Convex object. Method [60] and optimized BPTs.

2.4.2 Classification with shape priors

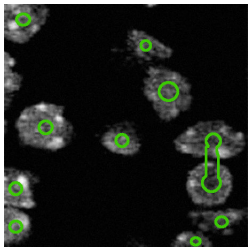
In a first series of experiments we constrain our method to binary segmentation with convexity shape prior, in order to compare it with a recent state-of-the-art technique designed for this purpose. In a second series of experiments we move to multi-object multi-class segmentation in remote sensing imagery and compare the behavior of our algorithm against the common approaches in the field.

An algorithm for a soft convexity shape prior in image segmentation was introduced by Gorelick et al. [60]. We will first show that we can perform similarly in a single convex object, while our technique is not specifically designed for this prior. A natural image extracted from [60] and the markers used are shown in Fig. 2.7(a). We set $\omega = 1$, a contrast sensitive Potts model (weight 10^{-4}) and an 8×8 orientation stencil, the parameters required by [60]. We found this parameter setting to yield the most visually pleasant result. We ran our BPT optimization approach with a data term learned from markers' histograms and with density functions favoring solidity and compactness for the foreground class, which convey the notion of convexity. For BPT construction we set $\alpha = 0.5$ throughout all the experiments, to make the terms in (2.8) equally relevant. We sample moves that involve regions of at least ten pixels. In this case we apply all good moves in the queue at every iteration, which are in practice less than 10. We did not impose hard constraints on the marker locations. The resulting segmentations are depicted in Figs. 2.7(b-c). Our BPT optimization approach achieves similar segmentation performance.

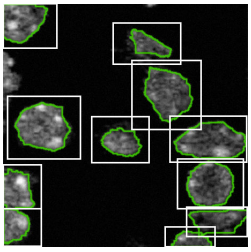
Fig. 2.8(a) shows a slice of a laser scanning microscopy image of brain tissue, where we wish to identify cell nuclei (green markers). If we apply the method by Gorelick et al. [60] to the whole image with all the markers, the result is inaccurate because the technique is not designed to segment more than one object at once (Fig. 2.8-b). In Fig. 2.8(c) we overlap the result of applying [60] to different fragments of the image that include each individual object (parameter ω in [60] is set to 0.01). The technique individually outlines



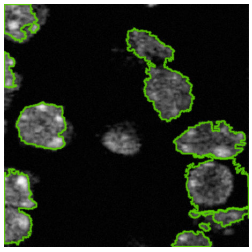
(a) Input (457×454)



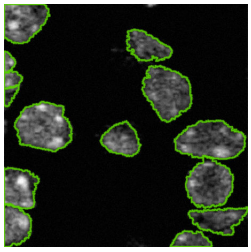
(b) Gorelick et al. [60]



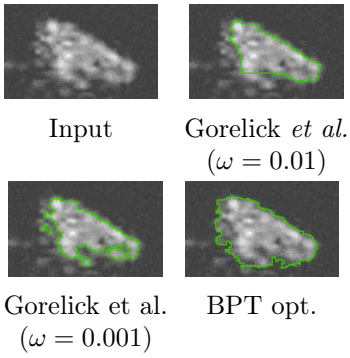
(c) Gorelick et al. [60], on user-supplied windows



(d) BPT



(e) Optimized BPT



(f) Amplified fragment

Figure 2.8: Multiple convex objects (cell nuclei). Gorelick et al. [60] and BPT optimization.

each of the nuclei in the absence of the rest, but prior knowledge about their location is required.

We constructed a BPT on this image (α in Eq. 2.8 is set to 0.5 in all the experiments). As expected, the best-cut segmentation without optimizing the tree (Fig. 2.8-d) is not competitive, because many tree nodes do not satisfactorily represent objects. We ran our BPT optimization approach with a data term learned from markers' histograms and with density functions favoring solidity and compactness for the foreground class, which convey the notion of convexity. We sample moves that involve regions of at least ten pixels. In this case we apply all energy decreasing moves in the queue at every iteration, which are in practice less than 10. We did not impose hard constraints on the marker locations. After optimizing the BPT with the method proposed in Sec. 2.3.3, each object is segmented individually and adjacent objects are delineated separately (Fig. 2.8-e). Our approach also produces accurate boundaries despite the low foreground/background contrast. The method by Gorelick et al. tends to either oversmooth the boundary to enforce convexity, or produce a very non-convex object, depending on the parameter ω (see the amplified nucleus in Fig. 2.8-f).

In the context of multi-object multi-class segmentation, we tested our method on images of urban scenes extracted from Google Maps screenshots. Figs. 2.9(a) and 2.10(a) show two color images acquired over New York City and the area of Brest, respectively. For both images, the manual segmentation was based on visual inspection combined with cadastral records available online. The list of the considered object classes is given in Fig. 2.9(b) (no instance of the *internal road* class is present in the Brest image). In the particular case of buildings, cadastral information was used to delineate every object independently even when they are spatially adjacent.

To evaluate the performance of the proposed method, we use two criteria:

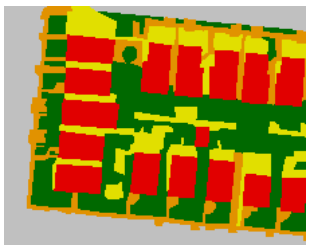
1) Overall accuracy $\mathcal{A}$, defined as the proportion of correctly classified pixels.

2) Building's overlap $\mathcal{D}$. For every building in the manually segmented image, we search for the most overlapping *building* region in the segmentation map in terms of Dice's coefficient [36]. Criterion $\mathcal{D}$ is estimated by averaging the computed coefficients.

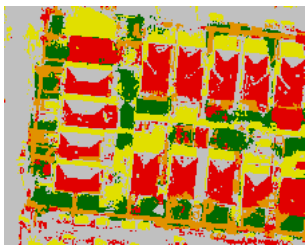
As in Section 2.4.1, an SVM with a Gaussian radial basis function kernel was used for the data term, tuned by tenfold cross-validation. The criterion (2.4) involved area, rectangularity and elongatedness shape descriptors. The distributions were trained by kernel density estimation on a set of sample objects from an adjacent image. In the area covered by these objects, 100 random pixels per class were selected to train the SVM.

We compared the performance of the proposed approach with the following methods: 1) SVM; 2) graph cut with α-expansion [17] (GC); 3) cut on the BPT, regularized by the

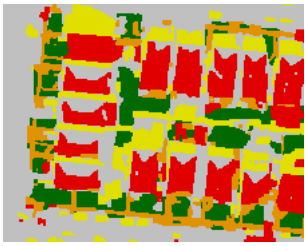
(a) Input (225×180)



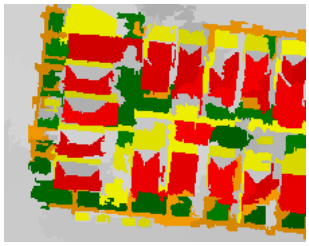
(b) Reference: tiles, roads, internal roads , veget., shadow



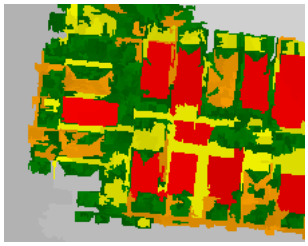
(c) SVM
($\mathcal{A} = 0.64$, $\mathcal{D} = 0.8$)



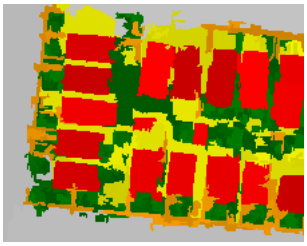
(d) GC
($\mathcal{A} = 0.68$, $\mathcal{D} = 0.81$)



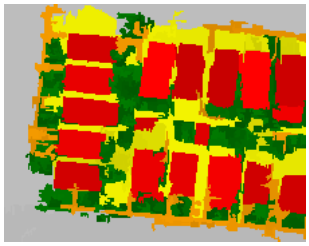
(e) TC
($\mathcal{A} = 0.71$, $\mathcal{D} = 0.71$)



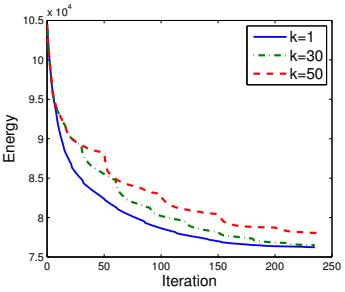
(f) TSC
($\mathcal{A} = 0.65$, $\mathcal{D} = 0.50$)



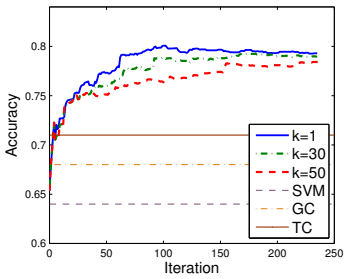
(g) BPT opt., $k = 1$
($\mathcal{A} = 0.79$, $\mathcal{D} = 0.89$)



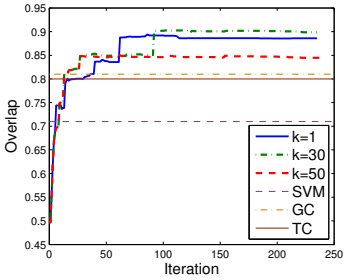
(h) BPT opt., $k = 30$
($\mathcal{A} = 0.79$, $\mathcal{D} = 0.90$)



(i) Evolution of the energy



(j) Evolution of overall accuracy



(k) Evolution of building's overlap

k	Time[s]	Time[s] (Props. 1-2)
1	707	334
30	23.08	9.02
50	12.47	4.69

(l) Execution time

Figure 2.9: Experimental results for the image over New York City.

number of regions without using shape priors (TC) [125]; 4) cut on the same BPT with our shape formulation (2.7), but without tree optimization (TSC). Figs. 2.9(c–f) illustrate the output of these techniques for the image of New York. Figs. 2.9(g–h) depict the results obtained by applying our optimization method with different values of k (see Sec. 2.3.3.3). The SVM classification exhibits many issues, notably the assignment of some roof parts to the wrong class. GC and TC smooth the results though do not correct the main mistakes in the classification. In the initial cut with shape priors on the unoptimized tree (TSC), some regions are enhanced but some others are significantly deteriorated with respect to the previous methods. This is due to the faulty tree construction that does not represent the entire objects in unique nodes. Figs. 2.9(g–h) show that the optimization of the tree copes with these issues, not only enhancing the initial cut on the tree but also outperforming the other techniques.

The evolution of the energy (2.7), the accuracy $\mathcal{A}$ and the building's overlap $\mathcal{D}$ with respect to the number of iterations are depicted in Figs. 2.9(i–k). As expected, the energy curve becomes less smooth as k increases. For small enough values of k, the segmentation maps are almost identical. This validates the fact that many branch moves are independent and can be applied prior to reconstructing the heap.

The BPT for this image is constructed in 1.25 seconds on an 8-CPU 2.7 GHz processor. The optimization time, summarized in Fig. 2.9(l), is considerably faster when Propos. 1-2 are used.

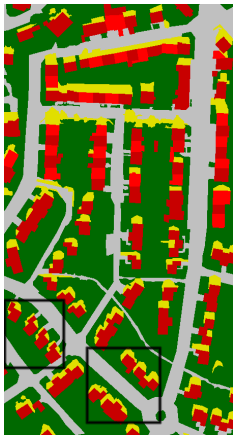
Fig. 2.10 illustrates experimental results for the image of Brest. Our method was executed with $k = 600$. Fig. 2.10(c) shows the obtained segmentation map. Two fragments of the map (boxed in Fig. 2.10(b)) are amplified for comparison in Fig. 2.10(d). These results validate the previous observations. The BPT is built in 13 seconds and the optimization takes 84 seconds using Propositions. 1-2 (see Section 2.3.3.2), against 214 s.

We highlight in another portion of image over Brest (Fig. 2.11) that our method is able to separate an entire building blob into rectangles. In addition, the use of shape features permits to "switch" a small building to the correct class, even though its color was assigned to *road* by the classifier.

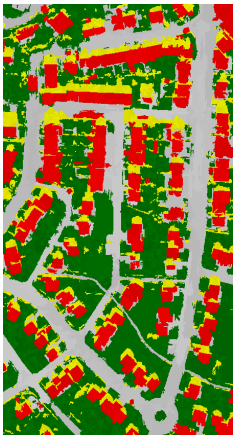
2.5 Concluding remarks

We have presented a BPT-based approach for the multi-class multi-object classification of images with shape priors. The problem is formulated as an energy minimization task, using the learned probability distributions of spectrum and shape features. We construct

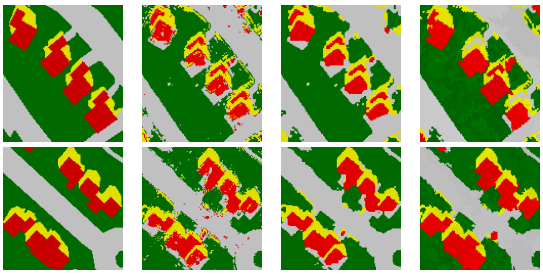
(a) Color image



(b) Manual classification



(c) BPT optimization



Manual SVM GC BPT opt.

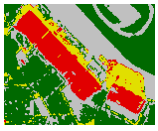
(d) Comparison on close-ups of the image

	$\mathcal{A}$	$\mathcal{D}$
SVM	0.764	0.50
GC	0.799	0.49
TC	0.804	0.47
BPT opt.	**0.813**	**0.69**

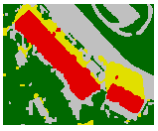
(e) Performance

Figure 2.10: Experimental results for the image over Brest.

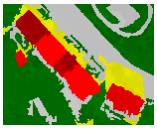
(a) Color image.



(b) SVM



(c) Graph cut



(d) BPT opt.

Figure 2.11: BPT optimization detects individual buildings.

a BPT of the image and search for the horizontal cut on the tree which minimizes the proposed energy function. One of the key innovations of the proposed method is that instead of using the standard BPTs, we optimize them by pruning and regrafting their branches at different scales in order to minimize the best classification that can be extracted from the tree. BPTs are a good departure point for optimization, since they allow to perform moves of variable region sizes. Conversely, in an MRF approach defined on the pixel grid, each time we perform a move we would have to search for a relevant region to perform a label switch (e.g., by using minimum spanning forests). A BPT can in fact be seen as a way of dynamically storing a set of candidate adjacent regions at different scales. Our theoretical study permits to reduce the space of branch moves. We also proposed an improved BPT construction function that adds a force to cluster together regions that are similar in terms of the classifier's output.

The experiments show that the method effectively incorporates shape information into classification, even when multiple classes and objects are present in the image, thus gaining a competitive edge with respect to recent literature in the field.

The main limitation of the method concerns its scalability. On the one hand, while the execution times are satisfactory compared to other shape-based classification techniques, efficient machine learning methods have been growing in the vision community, such as deep convolutional neural networks (CNNs). These are able to incorporate high-level features in terms of simple operations that are highly parallelized in GPUs. When we consider, e.g., the size of a Pléiades satellite image product (around $60,000 \times 75,000$ pixels), it is clear that highly parallel approaches will gain attention in the future. On the other hand, BPTs lack scalability in terms of the features themselves, i.e., one must select and design efficiently computable shape features. However, the new trend also introduced by CNNs is to automatically learn hundreds of task-specific features, which may allow us to easily scale the pool of contextual features considered.

Chapter 3

Large-Scale Classification with Convolutional Neural Networks

There is a growing interest in analyzing remote sensing images at a large scale [80, 102, 104]. One of the related challenges is the computational complexity of the algorithms. Another important challenge is the intra-class variability: due to the large spatial extent covered by the datasets, the object classes may have considerably different appearances. To conduct an accurate classification, we must then have a thorough understanding of the context such as, e.g., the shape of objects and their location with respect to other objects.

In the previous chapter, we proposed a technique to include shape features on top of a classifier, based on binary partition trees. There is however an incipient trend in the vision literature to design classifiers that automatically learn expressive high-level contextual features. This adds scalability in terms of the features themselves: we give room to learn more complex spatial features depending on the context, beyond e.g., rectangularity or convexity. This is the motivation behind convolutional neural networks (CNNs) [83], which have gained significant attention over the last few years in the vision community. CNNs consist of a stack of learned convolution filters, and are a popular form of *deep learning*. They have outperformed other approaches in various domains such as digit recognition [27] and natural image categorization [76]. Considering also their computational efficiency and scalability to handle large inputs, we here explore the application of CNNs to classify remote sensing images.

The goal of this chapter is to devise an end-to-end framework to classify satellite imagery with CNNs, i.e., where the image is directly inputted to the system, which gives a classification map as output. The context of large-scale satellite image classification

Figure 3.1: Fragments of the Forez training set (red: *building*), derived from OpenStreetMap.

introduces certain specific challenges that we must first address in order to turn common CNNs into a relevant classification tool. First of all, we must decide which is the architectural prototype from which to derive remote sensing classification networks, since most CNNs have been designed with a different goal in mind.

We must also discuss the training data to be used. The fact of automatically learning contextual features requires more training *per se*, as we must show to the classifier the many different contexts in which a pixel class can be embedded, and not just its spectral values. One large source of data is OpenStreetMap[1] (OSM), a collaborative online mapping platform. It is built by volunteers, often with the input of official data when this is possible. However, in many areas of the earth, the coverage of OSM is limited. In the examples of Fig. 3.1 we observe large areas with missing data and a general misregistration of the vectorial maps with respect to the actual structures. In addition, the misregistration is not uniform and neighboring buildings are often shifted in different directions. Note that in the samples of Fig. 3.1 the buildings have been delineated in OSM based on the official French cadaster. However, even the cadastral records are not always accurate up to the meter resolution, and are sometimes digitized from paper maps. Furthermore, satellite images undergo a series of corrections. For example, the use of inexact elevation models for orthorectification (see Chapter 1) might introduce misregistration throughout the images. As a result, the OSM raw data is imperfect and thus not fully reliable.

The reference data obtained from OSM, as shown in Fig. 3.1, provides a rough idea of the location of the buildings, but rarely outlines them. In such a setting, CNNs would

[1] www.openstreetmap.org

hardly learn that building boundaries are likely to fall on visible edges, since this is not what the reference data depicts. Under these circumstances, we expect the predictions not to be very confident, especially on the border of the objects. As we will illustrate in Section 3.4.2, this yields a "blobby" and overly fuzzy aspect on challenging datasets.

Another concern is the obtention of fine-grained output classification maps, because the powerful capability of CNNs for taking a large context into account to conduct predictions comes at the price of losing resolution in the output. This is because some degree of downsampling along the network is required in order to increase the amount of context without an excessive number of learnable parameters. Such coarse resolution amplifies the fuzzy aspect around output object edges and corners.

In this chapter we propose to use the so-called *fully convolutional networks* (FCNs) [93] as the elementary type of architecture for remote sensing classification. To deal with imperfect training data, we propose to train CNNs with large amounts of imperfect data and fine-tune them with a little piece of manual segmentation. We also present a multi-resolution CNN module to provide high-resolution outputs. This completes our end-to-end framework to classify satellite imagery.

3.1 Related work

Convolutional neural networks (CNNs) were introduced by LeCun et al. [83] for handwritten digit recognition. Compared to traditional multilayer perceptrons, CNNs restrict the number and type of connections between layers, yielding a smoother optimization surface and ease of training.

A decade later, in 2012, the AlexNet network [76] boosted the interest in deep learning and is probably one of the most influential papers to date, setting the foundations on the way CNNs are now used in computer vision. This network incorporated several recent developments, notably, the recent ReLU activation function [110, 57], which quickly became the most popular one [82]. It also incorporated a recent strategy to overcome overfitting, known as dropout [65, 141]. AlexNet significantly outperformed other methods on the ImageNet Large-Scale Visual Recognition Challenge, which drove attention from the community.

CNNs are commonly used for *image categorization*, i.e., the assignment of a class to the entire image (e.g., a digit [83] or an object category [76]). In remote sensing, the equivalent problem is to assign a category to an entire image patch, such as "residential" or "agricultural" area, a problem referred to as land use classification. CNN-based methods have been recently proposed for land use classification [95, 130], as well as to solve other

remote sensing problems such as object detection [22] and road tracking [157]. Our context differs in that we wish to conduct *dense* pixelwise labeling. We must thus design a CNN that outputs a per-pixel classification and not just a category for the entire input.

The problem of pixelwise classification has been an important research field for years in the context of hyperspectral classification. CNNs have also been recently used to classify hyperspectral images, by performing convolutions in the 1-D domain of the spectrum of each pixel [89, 105, 138]. (unlike the 2-D spatial domain of most common CNNs). Alternatively, a spectral-spatial approach has been taken by performing convolutions in the 1-D flattened spectrum vector of a group of adjacent pixels [25, 24]. Note however that these approaches do not learn spatial contextual features such as the typical shape of the objects of a class. Recent works have incorporated convolutions in the spatial domain after extracting the principal components of the hyperspectral image [100, 168, 170]. Outside hyperspectral images, Mnih et al. [107, 108] used CNNs to label buildings and roads, by modifying the last layer of a typical categorization CNN to output not just one value but an entire classification patch.

In computer vision, an influential work was published by Long et al. [93], in which they proposed to perform semantic pixelwise labeling of natural images by using the so-called fully convolutional networks. These networks contain only convolutional layers, unlike categorization CNNs that also include fully connected layers. This removes redundant computations compared to a patch-based approach, and allows to input images of different sizes to the network and directly output the corresponding classification map, without increasing the number of trainable parameters.

In this chapter we build upon the work by Mnih and modify it so that it becomes fully convolutional, pointing out the clear benefits of this type of architecture. Since our publications on this topic, other researchers validated our observations by comparing fully convolutional networks against patch-based approaches in remote sensing [73, 131]. Moreover, over the past two years fully convolutional networks became indeed the de facto network prototype for pixelwise classification in multiple domains [21, 155, 167].

There is a well-known trade-off between the receptive field (how much context is taken to conduct predictions) and the output resolution (how fine is the prediction) if we wish to keep a reasonable number of trainable parameters [21, 93]. The idea of providing high-resolution outputs has thus gained significant attention. Several methods have been specifically designed to generate higher-resolution outputs through improved network architectures [8, 112] or post-processing [21, 171]. Let us remark though that in the particular context of remote sensing imagery, the spatial precision of the classification maps is of paramount importance. Objects are small and a boundary misplaced by a few pix-

els significantly hampers the overall classification quality. In other application domains, such as semantic segmentation of natural scenes, while there have been recent efforts to better shape the output objects, a high-resolution output seems to be less of a priority. For example, in the popular Pascal VOC semantic segmentation dataset [41], there is a band of several unlabeled pixels around the objects, where accuracy is not computed to assess the performance of the methods. In the context of remote sensing, the high-resolution datasets associated to the ISPRS semantic segmentation challenge [70] recently became a popular framework to assess and compare pixelwise labeling methods. In subsequent chapters we delve into the ideas of post-processing to refine classification maps (Chapter 4) and we explore in depth the most recent contributions for high-resolution classification and experiment on the ISPRS datasets (Chapter 5).

3.2 Background on convolutional neural networks (CNNs)

An artificial neural network is a system of interconnected neurons that pass messages to each other. When the messages are passed from one neuron to the next one without ever going back (i.e., the graph of message passing is acyclic) they network is referred to as feed-forward [11], which is the most common type of network in image categorization. An individual neuron takes a vector of inputs $\mathbf{x} = x_1 \dots x_n$ and performs a simple operation to produce an output a. The most common neuron is defined as follows:

$$a = \sigma(\mathbf{w}\mathbf{x} + b), \tag{3.1}$$

where $\mathbf{w}$ denotes a weight vector, b a scalar known as *bias* and σ an activation function. Put simply, a neuron computes a weighted sum of its inputs and applies a possibly nonlinear scalar function to the result. The weights $\mathbf{w}$ and biases b are the parameters of the neurons that define the function. The goal of training is to find the optimal values for these parameters, so that the function computed by the neural network performs the best on the task assigned.

The most common activation functions σ are sigmoids, hyperbolic tangents and rectified linear units (ReLU). For image analysis, ReLUs have become the most popular choice due to some practical advantages at training time, but novel activation functions have been recently proposed as well [28].

Despite an apparent simplicity, neural networks are extremely expressive: by using at least one layer of nonlinear activation functions, a sufficiently large network can represent *any* function within a given bounded error [11].

Instead of directly connecting a huge set of neurons to the input, it is common to organize them in groups of stacked layers that transform the outputs of the previous layer and feed it to the next layer. This enforces the networks to learn hierarchical features, performing low-level reasoning in the first layers (such as edge detection) and higher-level tasks in the last layers (e.g. [45], assembling object parts). For this reason, the first and last layers are often referred to as lower and upper layers, respectively.

In an image categorization problem, the input of the network is an image (or a set of features derived from an image), and the goal is to predict the correct category of the entire image. We can view the pixelwise semantic labeling problem as taking an image patch and categorizing its central pixel. Finding the optimal neural network classifier reduces to finding the weights and biases that minimize a loss L between the predicted labels and the target labels in a training set. Let Ω be the set of possible semantic classes. Labels are typically encoded as a vector of length $|\Omega|$ with values '1' at the position of the correct label and '0' elsewhere. The network contains thus as many output neurons as possible labels. A softmax normalization is performed on top of the last layer to guarantee that the output is a probability distribution, i.e. the label values are between zero and one and sum to one. The multi-label problem is then seen as a regression on the desired output label vectors.

The loss function L quantifies the misclassification by comparing the target label vectors $\mathbf{y}^{(i)}$ and the predicted label vectors $\hat{\mathbf{y}}^{(i)}$, for p training samples $i = 1 \dots p$. In this work we use the common cross-entropy loss, defined as:

$$L = -\frac{1}{p} \sum_{i=1}^{p} \sum_{k=1}^{|\Omega|} y_k^{(i)} \log \hat{y}_k^{(i)}. \tag{3.2}$$

Training neural networks by optimizing this criterion converges faster, compared with, for instance, the Euclidean distance between $\mathbf{y}$ and $\hat{\mathbf{y}}$. In addition, it is numerically stable when coupled with softmax normalization [11].

Note that in the special case of binary labeling we can produce only one output (with targets '1' for positive and '0' for negative). In this case a sigmoid normalization and cross-entropy loss are analogously used, although a multi-class framework can also be used for two classes.

Once the loss function is defined, the parameters (weights and biases) that minimize the loss are found via gradient descent, by computing the derivative $\frac{\partial L}{\partial w_i}$ of the loss function with respect to every parameter w_i, and updating the parameters with a learning

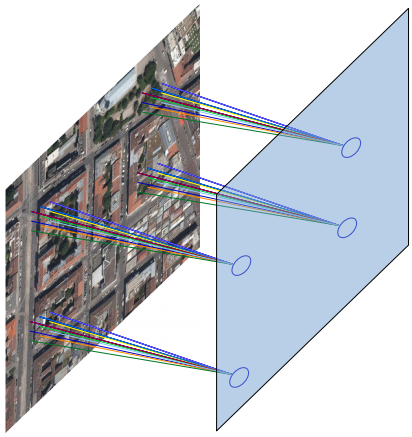


Figure 3.2: Convolutional layer. Connections are limited to a local spatial neighborhood and the weights are shared across the different neurons.

rate λ as follows:

$$w_i \leftarrow w_i - \lambda \frac{\partial L}{\partial w_i}. \tag{3.3}$$

The derivatives $\frac{\partial L}{\partial w_i}$ are obtained by *backpropagation*, which consists in explicitly computing the derivatives of the loss with respect to the last layer's parameters and using the chain rule to recursively compute the derivatives of each layer's outputs with respect to its weights and inputs (the inputs being the previous layer's outputs). In practice, instead of averaging over the full dataset, the loss (3.2) is estimated from a random small subset of the training set, referred to as *mini-batch*. This learning technique is known as *stochastic gradient descent*.

Convolutional neural networks (CNNs) are a particular type of neural network. The overall success of CNNs lies mostly in the fact the the networks are forced by construction to learn hierarchical contextual translation-invariant features, which is particularly useful in the context of image analysis. We will now describe the ingredients of a CNN.

Convolutional layers

Convolutional neural networks (CNNs) [83] contain so-called convolutional layers, a specific type of layer that imposes a number of restrictions compared to a more general fully connected layer, where every neuron is connected to all outputs of the previous layer. These restrictions (e.g., local connectivity) have been introduced for image categorization networks because they make sense in that particular context.

In CNNs, each neuron is associated to a spatial location (i, j) in the input image

(see Fig. 3.2). The output a_{ij} associated with location (i, j) in a convolutional layer is computed as:

$$a_{ij} = \sigma((\mathbf{W} * \mathbf{X})_{ij} + b), \tag{3.4}$$

where $\mathbf{W}$ denotes a kernel with learned weights, $\mathbf{X}$ the input to the layer and '$*$' the convolution operation. Notice that this is a special case of the neuron in Eq. 3.1 with the following constraints:

- The connections only extend to a limited spatial neighborhood determined by the kernel size;
- The same filter is applied to each location, guaranteeing translation invariance.

Multiple convolution kernels are usually learned in every layer, interpreted as a set of spatial feature detectors. The responses to every learned filter are thus referred to as *feature maps*. Note that the convolution kernels are actually three-dimensional: in addition to their spatial extent (2D), they span along all the feature maps in the previous layer (or eventually through all the bands in the input image). As this third dimension can be inferred from the previous layer it is rarely mentioned in the architecture descriptions.

Compared to the fully connected layer, a convolutional layer highly reduces the number of parameters by enforcing the aforementioned constraints. This results in an easier optimization problem, without losing much generality. This opened the door to using the image itself as an input without any feature design and selection process, as CNNs discover the relevant spatial features to conduct classification.

Downsampling and fully connected layers

In addition to convolutional layers, state-of-the-art networks such as Imagenet [76] involve some degree of downsampling, i.e., a reduction in the resolution of the feature maps. The goal of downsampling is to increase the so-called *receptive field* of the neurons, which is the part of the input image that neurons can "see". For the predictions to take into account a large spatial context, the upper layers should have a large receptive field. This is achieved either by increasing the convolution kernel sizes or by downsampling feature maps to a lower resolution. The first alternative increases the number of parameters and memory consumption, making the training and inference processes prohibitive. State-of-the-art CNNs thus tend to keep the kernels small and add some degree of downsampling instead. This can be accomplished either by including pooling layers (e.g., taking the average or maximum of adjacent locations) or by introducing a so-called *stride*, which amounts to skip some convolutions through, e.g., applying the filter once every four

locations. We include a detailed explanation of receptive fields and the different forms of downsampling in Section 5.1.

Classification networks typically contain a few fully connected layers on top of the sequence of convolution and pooling layers. The idea behind these layers is to classify the image based on the learned spatial features. The last of these layers is designed to have as many outputs as classes, so as to yield the final classification scores.

3.3 Proposed end-to-end classification framework

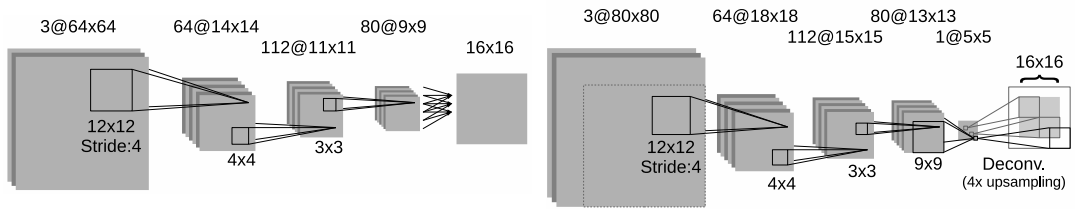
Our goal is to perform *dense* classification, i.e., not just the categorization of an entire image, but a full pixelwise labeling into the different classes. For this goal, in the next section we analyze an existing patch-based approach for aerial image labeling, point out some of its limitations and propose to transform it into a *fully convolutional* architecture that addresses these limitations.

We then address the issue of inaccurate labels for CNN training. For this we propose a two-step procedure: 1) the network is first trained on raw OpenStreetMap (OSM) data, 2) it is then fine-tuned on a tiny piece of manually labeled image. This method provides us with a means to deal with the inaccuracy of training data, by increasing the confidence and sharpness of the predictions.

However, we still cannot expect it to provide highly precise boundaries just by using a fully convolutional architecture. This is because such network involves some downsampling, required to capture the long-range spatial dependencies that help recognize the classes. However, downsampling makes the whole system lose spatial precision, which is not recovered by naively upsampling the outputs. We thus propose a new architecture that incorporates information at multiple scales in order to alleviate this trade-off. Our architecture combines low-resolution long-range features with high-resolution local features that conduct predictions with a higher level of detail. This architecture, when combined with our two-step training approach, provides a framework that can be used end-to-end to classify satellite imagery.

3.3.1 Fully convolutional network

To perform dense classification of aerial imagery, Mnih proposed a patch-based convolutional neural network [107]. Training and inference are performed patch-wise: the network takes as input a patch of an aerial image, and generates as output a classified patch. The output patch is smaller, and centered in the input patch, to take into ac-



(a) Patch-based

(b) Fully convolutional (16×16 output)

Figure 3.3: Convolutional neural network architectures (e.g., "64@14×14" means 64 feature maps of size 14×14).

count the surrounding context for more accurate predictions. The way to create dense predictions is to increase the number of outputs of the last fully connected classification layer, in order to match the size of the target patch.

Fig. 3.3(a) illustrates the patch-based architecture from [107]. The network takes 64×64 patches (on color images of 1 m^2 spatial resolution) and predicts 16×16 centered patches of the same resolution. Three convolutional layers learn 64, 112 and 80 convolution kernels, of 12×12, 4×4 and 3×3 spatial dimensions, respectively. The first convolution is strided (one convolution every four pixels), which implies a downsampling factor 4.

After the three convolutional layers, a fully connected layer transforms the high-level features of the last convolutional layer into a classification map of 256 elements, matching the required 16×16 output patch.

Training is performed by selecting random patches from the training set, and grouping them into mini-batches as required by the stochastic gradient descent algorithm.

3.3.1.1 Limitations of the patch-based scheme

We now point out some limitations of the patch-based approach discussed above, which motivate the design of an improved network architecture. Let us first analyze the role of the last fully connected layer that constructs the output patches. In the architecture of Fig. 3.3(a), the size of the feature maps in the last convolutional layer (before the last fully connected one) is 9×9. The resolution of these filters is $1/4$ of the resolution of the input image, due to the 4-stride in the first convolution. The output of the fully connected layer is, however, a full-resolution 16×16 classification map. This means that the fully connected layer does not only compute the classification scores, but also learns how to upsample them. Outputting a full-resolution patch is then the result of upsampling and not of an intrinsic high-resolution processing. We also observe that the

(a) Color input (b) Patch-based prediction

Figure 3.4: The patch-based predictions exhibit artifacts on the patch borders.

fully connected layer allows outputs at different locations to have different weights with respect to the previous layer. For example, the weights associated to an output pixel at the top-left corner of a patch can be different to those of a pixel at the bottom right. In other words, the network can learn priors on the position inside a patch. This makes sense in some specific contexts such as the labeling of outdoor scenes: the system could learn a prior for the sky to be at the top of the image. In our context, however, the partition of an image into patches is arbitrary, hence the "in-patch location" prior is irrelevant since allowing different weights at different patch locations may yield undesirable properties. For example, feeding two image patches that are identical but rotated by 90 degrees could produce different classification maps.

When training the network of Fig. 3.3(a) we expect that, after processing many training cases, the fully connected layer will end up learning a location-invariant function. Fig. 3.4 illustrates a fragment of an output score map by using such an architecture. Notice the discontinuities at the border of the patches, which reveal that the network did not succeed in learning to classify pixels independently of their location inside the patch. While this issue is partly addressed in [107] by smoothing the outputs with a conditional random field, we argue that avoiding such artifacts by construction would be beneficial. In addition, generating similar results regardless of image tiling is an important property for large-scale satellite image processing, and an active research topic [80, 104]. Another concern with the fully connected layer is that the receptive field of every patch output is not centered in itself. For example, a prediction near the center of the output patch can "see" about 32 pixels in every direction around it. However, the prediction at the top-left corner of the output patch considers a larger portion of the image to the bottom and to the right than to the top and to the left. Considering that the division into patches is arbitrary, this behavior is hard to justify.

A deeper understanding of the role played by every layer of the network, as described in this section, motivates the design of a more suitable architecture from a theoretical point of view, with the additional goal of boosting the overall performance of the approach.

3.3.1.2 Fully convolutional network

We propose a *fully convolutional* neural network architecture (FCN) to produce dense predictions. FCNs were first introduced in [93]. We explicitly restrict the process to be location-independent, enforcing the outputs to be the result of a series of convolutions only (see Fig. 3.3-b).

A classification network may be "convolutionalized" as follows. We first convert the fully connected layer that carries out the classification to a convolutional layer. The convolution kernel is chosen so that its dimensions coincide with the previous layer. Thus, its connections are equivalent to a fully connected layer. The difference is that if we enlarge the input image, the output size is also increased, but the number of parameters remains constant. This may be seen as convolving the whole original network around a larger image to evaluate the output at different locations.

To increase the resolution of the output map, we then add a so-called "deconvolutional" layer [93]. The goal of this layer is to upsample the feature maps from the previous layer, which is achieved by performing an interpolation from a set of nearby points. Such an interpolation is parametrized by a kernel that expresses the extent and amount of contribution from a pixel value to its neighboring positions, only based on their locations. For an effective interpolation, the kernels must be large enough to overlap in the output. The interpolation is then performed by multiplying the values of the kernel by every input and adding the overlapping responses in the output. This process is illustrated by Fig. 3.5 for a 2x upsampling. Notice that the scaling step is performed based on a constant 4×4 kernel. In our framework, and as in previous work [93], the interpolation kernel is another set of learnable parameters of the network instead of being determined a priori, e.g., by setting them to represent a bilinear interpolation. Note also that the upsampled feature map has a central part computed by adding the contribution of two neighboring kernels and an outer border obtained solely by the contribution of one kernel (the two leftmost and rightmost output columns in Fig. 3.5). The outer border can be seen as an extrapolation of the input while the inner part can be seen as an interpolation. The extrapolated border can be cropped from the output to avoid artifacts.

As compared to a patch-based approach, we can expect our fully convolutional network to exhibit the following advantages:

- Elimination of discontinuities due to patch borders, by construction;
- Improved accuracy due to a simplified learning process, with a smaller number of parameters;

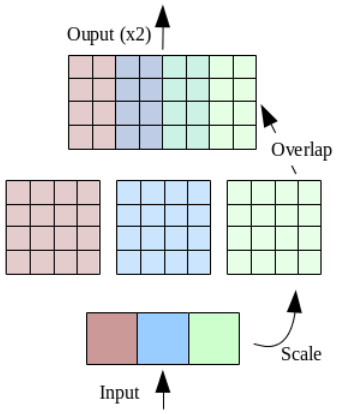


Figure 3.5: "Deconvolution" layer for upsampling.

- Lower execution time at inference, due to the reduced number of parameters and the fast GPU execution of convolution operations.

Our FCN network is constructed by convolutionalizing the existing patch-based network depicted by Fig. 3.3(a). We choose an existing framework to benefit from a mature architecture and to carry out a rigorous comparison. The architectural decisions (i.e., the choice of the number of layers and filter sizes) of the base network are described in [107].

Fig. 3.3(b) depicts the resulting FCN. First, we pretend that the output patch of the original network is only of size 1×1, thus just focusing on a single output centered in its receptive field. Second, we rewrite the fully connected layer as a convolutional layer with one feature map and the spatial dimensions of the previous layer (9×9). Third, we add a deconvolutional layer that upsamples its input by a factor of 4 (with a learnable kernel of size 8×8), in order to recover the input resolution. Notice that the tasks of classification and upsampling are now separated.

This new network can take input images of different sizes, with the output size varying accordingly. For example, during the training stage we wish to output patches of size 16×16 in order to emulate the learning process as was done in the patch-based network of Fig. 3.3(a). For this we require a patch input of size 80×80, as in the architecture of Fig. 3.3(b). Notice that the input is larger than the original 64×64 patches. This is not because we are taking more context to carry out the predictions, but instead because every output is now centered in its context. At inference time we can take inputs of arbitrary sizes and feed them to the network to construct the classification maps, and the number of network parameters does not vary.

3.3.2 Dealing with imperfect training data

Fine-tuning is a very common procedure in the neural network literature. The idea is to refine an existing pretrained model to a specific problem by executing a few training iterations on a new dataset. The notion of fine-tuning is based on the intuition that low-level information/features can be reused in different applications, without training from scratch. Even when the final classification objective is different, it is also a relevant approach for initializing the learnable parameters close to good local minima, instead of initializing with random weights. After proper fine-tuning, low-level features tend to be quite preserved from one dataset to another, while the higher layers' parameters are updated to adapt the network to the new problem [166].

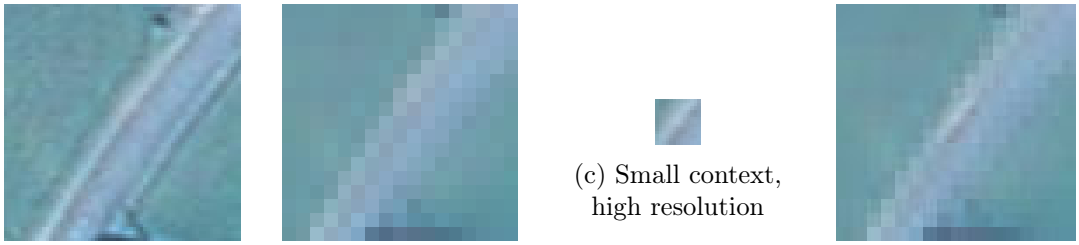
When fine-tuning, the training set is usually substantially smaller than the one used to train the original network. This is because one assumes that some generalities of both problems are well conveyed in the pretrained network (e.g., edge detectors in different directions) and the fine-tuning phase is just needed to adapt the parameters to the specific application. When the training set used for fine-tuning is very small, additional considerations to avoid overfitting are commonly taken, such as early stopping (executing just a few iterations on the new training dataset), fixing the weights at the lower layers or reducing the learning rate.

We now incorporate the idea of neural network fine-tuning, in order to perform training on imperfect data. Our approach proceeds in two steps. In step one large amounts of training data are used to train a fully convolutional neural network. This raw training data is extracted directly from OSM, without any hand correction. The goal of this step is to capture the generalities of the dataset such as, e.g., the representative shape of object classes.

In step two, we fine-tune the network by using a small portion of carefully labeled image. This phase is designed to compensate for the inaccuracy of labels obtained in step one, by fine-tuning the network on small yet consistent target outputs. Assuming that most of the generalities have been captured during the initial training step, the fine-tuning step should locally correct the network parameters to output more accurate classifications. The efforts of fine-tuning are thus limited to manually labeling a small dataset, while the large inaccurate dataset is automatically extracted from OSM.

3.3.3 Conducting fine predictions

The resolution at which the proposed fully convolutional networks operates yields probability maps that, once upsampled, are coarse in terms of spatial accuracy. A naive way



(a) Large context, high resolution

(b) Large context, low resolution

(c) Small context, high resolution

(d) Combined resolutions

Figure 3.6: Different types of context to predict the central pixel's class. A multi-resolution context such as in (d) alleviates the trade-off between classification accuracy and number of learnable parameters.

to increase the resolution of the network would be to use higher-resolution filters, which requires to increase their dimensions if we want to preserve the receptive field. For example, instead of applying a 5×5 filter at a fourth of the image resolution, one could use a 20×20 filter at full resolution, hence covering the same spatial extent. However, such an increase in filter sizes is prohibitive, hampering the spatial and temporal efficiency of the algorithm and producing less accurate results due to the difficulty of optimizing so many parameters.

Nevertheless, we observed that we do not need full-resolution filters to conduct accurate predictions. One requires a higher resolution only in the center of the convolution filters (assuming that the pixel we wish to predict is in the center of the context of interest). A large spatial extent is indeed required to capture contextual information, but it is not necessary to conduct this analysis at full resolution. For example, the presence of two parallel bands of grass can help identify a road (and distinguish it from, for instance, a building with a gray rooftop), but a precise localization of the grass is not necessary. On the contrary, at the center of the convolution filter, a higher-resolution analysis is required to specifically locate the boundary of the aforementioned road.

Fig. 3.6 illustrates this observation. In Fig. 3.6(a) we observe the area around a pixel whose class we wish to predict, at full resolution. A filter taking such an amount of context with that resolution would be prohibitive in the number of parameters, as well as unnecessary. Fig. 3.6(b) depicts the same context at a quarter of the resolution. Notice that it is still possible to visually infer that there is a road. However, identifying the precise location of the boundaries of the road becomes difficult. Alternatively, Fig. 3.6(c) depicts a small patch but at full resolution. We can now better locate the precise boundary of the object, but with so little context it is difficult to identify that the object is

indeed a road. Large filters at low resolution - see Fig. 3.6(b) or small filters at high resolution - see Fig. 3.6(c), which would both have a reasonable number of parameters, are bad alternatives: the first filter is too coarse and the second filter is using too little context.

We propose convolutional filters that combine multiple resolutions instead. In Fig. 3.6(d) the large-size low-resolution context of Fig. 3.6(b) is combined with the small high-resolution context of Fig. 3.6(d). This provides us with a means to simultaneously infer the class by observing the surroundings at a coarse resolution, and determine the precise boundary location by using a finer context. This way, the amount of parameters is kept small while the trade-off between recognition and localization is alleviated.

Les us denote by $\mathcal{S}$ a set of levels of detail expressed as a fraction of the original resolution. For example, $\mathcal{S} = \{1, 1/2\}$ is a set comprising two resolutions: full resolution and half of the full resolution. We denote by $\mathbf{x}_s$ a feature map $\mathbf{x}$ downsampled to a certain level $s \in \mathcal{S}$. For example, $\mathbf{x}_{1/2}$ is a feature map downsampled to half of the original resolution. Inspired by Equation 3.1, we design a special type of neuron that adds the responses to a set of filters applied at different scales of the feature maps in the previous layer:

$$a = \sigma\left(\sum_{s\in\mathcal{S}} \mathbf{w}_s \mathbf{x}_s + b\right). \tag{3.5}$$

Notice that individual filters $\mathbf{w}_s$ are learned for every scale s. Such a filter is easily implemented by using a combination of elementary convolutional, downsampling and upsampling layers. Fig. 3.7 illustrates this process in the case of a two-scale ($\mathcal{S} = \{1, 1/2\}$) module. In our implementation we average neighboring elements in a window for downsampling and perform bilinear interpolation for upsampling, but other approaches would also be valid. The kernel sizes of the convolutions at both scales are set to be equal (e.g., 3×3), yet the amount of context taken varies from one path to the other due to the different resolutions. The addition is an elementwise operation, followed by the nonlinear activation function.

3.4 Experiments

In a first series of experiments we compare the patch-based vs fully convolutional networks. We do this in the context of *building* vs *not building* classification problem, using the Massachusetts buildings dataset [107].

In a second series of experiments we test our overall classification framework on a challenging satellite image dataset over Forez, France, with poor quality OpenStreetMap

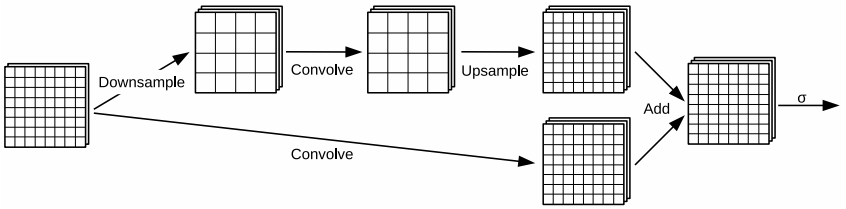
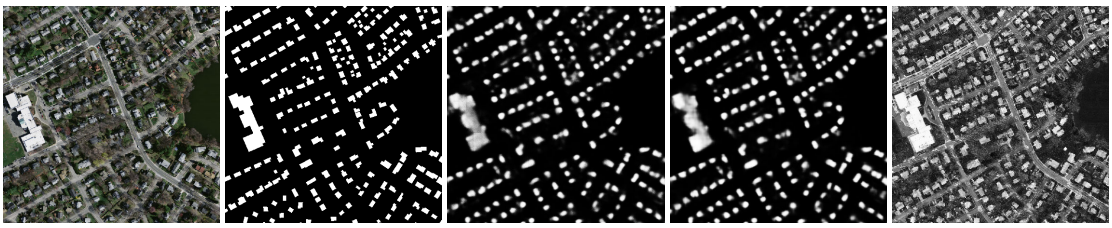


Figure 3.7: Two-resolution convolutional module that simultaneously combines coarse large-range and fine short-range reasoning.



(a) Color image (b) Reference data (c) Patch-based fuzzy map (d) FCN fuzzy map (e) SVM fuzzy map

Figure 3.8: Experimental results on a fragment of the Massachusetts dataset.

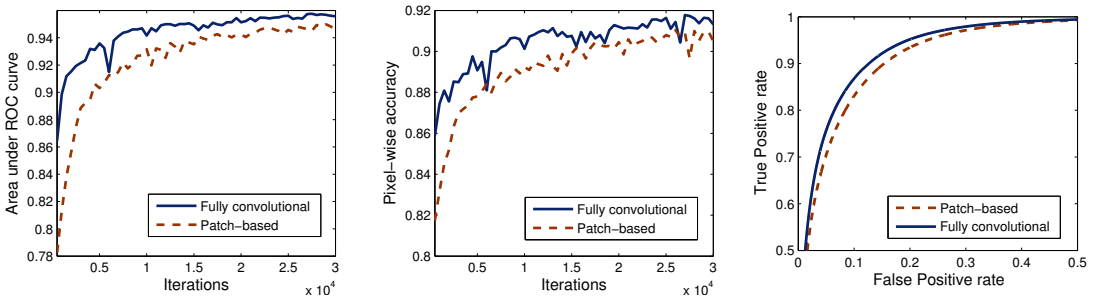
reference data.

3.4.1 Patch-based vs fully convolutional approaches

We implemented the patch-based and fully convolutional networks described in Section 3.3.1 using the *Caffe* [71] deep learning framework[2] and compare the approaches on the Massachusetts Buildings Dataset [107]. This dataset consists of color images over the area of Boston with 1 m^2 spatial resolution, covering an area of 340 km^2 for training, 9 km^2 for validation and 22.5 km^2 for testing. The images are labeled into two classes: *building* and *not building*. The labeling is quite precise, and was developed by manually correcting the OSM data. A portion of an image and its corresponding reference are depicted in Figs. 3.8(a-b).

We train the patch-based and fully convolutional networks (Figs. 3.3(a) and 3.3(b) respectively) for 30,000 stochastic gradient descent iterations, until we observe barely no further improvement on the validation set. The patches are sampled uniformly from the whole training set, with mini-batches of 64 patches each and a learning rate of 0.0001. A momentum and an L2 parameter penalty are introduced to regularize the learning process and avoid overfitting. Momentum adds a fraction of the previous gradient to

[2]Source code available at https://github.com/emaggiori/CaffeRemoteSensing



(a) Performance evolution (b) ROC curves

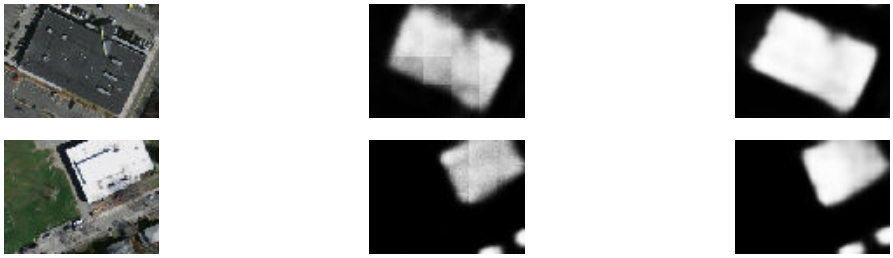
Figure 3.9: Evaluation of patch-based and fully convolutional neural networks on the Massachusetts test set.

the current one in order to smooth the descent, while an L2 penalty on the learned parameters discourages neurons to specialize too much on particular training cases [11]. The weights of these regularizers are set to 0.9 and 0.0002 respectively. Further details on these so-called hyperparameters and rationale for selecting them are provided by Mnih [107].

To evaluate the accuracy of the classification we use two different measures: pixelwise accuracy (proportion of correctly classified pixels, obtained by thresholding the output probabilities with threshold 0.5) and area under the receiver operating characteristics (ROC) curve [47]. The latter quantifies the relation between true and false positives at different thresholds, and is appropriate to evaluate the overall quality of the fuzzy maps.

Fig. 3.9(a) plots the evolution of the area under the ROC curve and pixelwise accuracy in the test set, through iterations. The FCN consistently outperforms the patch-based network. Fig. 3.9(b) shows ROC curves for the final networks after convergence, the FCN exhibiting the best relation between true and false positive rates. Fig. 3.8(c-d) depicts some visual results.

To further illustrate the benefits of neural networks over other learning approaches we train a support vector machine (SVM) with Gaussian kernel on 1,000 randomly selected pixels of each class. We train on the individual pixel spectra without any feature selection. The SVM parameters are selected by 5-fold cross-validation, as commonly performed in remote sensing image classification [143]. As shown by Fig. 3.8(e), the pixelwise SVM classification often confuses roads with buildings due to the fact that their colors are similar, while CNNs better infer and separate the classes by taking into account the geometry of the context. The accuracy of the SVM on the Massachusetts test dataset is 0.6229 and its area under ROC curve is 0.5193, i.e., significantly lower than with CNNs,



(a) Color (b) Patch-based (c) FCN

Figure 3.10: The patch-based predictions exhibit artifacts on the patch borders while the FCN prevents them by construction.

as shown in Fig. 3.9. If we wished to successfully use an SVM for this task, we should design and select spatial features (e.g., texture) and use them as the input to the classifier instead.

The amplified fragment in Fig. 3.10 shows that the border discontinuity artifacts present in the patch-based scheme are absent in our fully convolutional setting. This behaves as expected considering that the issues described in Section 3.3.1.1 are addressed by construction in the new architecture. This confirms that imposing sensible restrictions to the connections of a neural network has a positive impact in the performance.

In terms of efficiency the FCN also outperforms the patch-based CNN. At inference time, instead of carrying out the prediction in a small patch basis, the input of the FCN is simply increased to output larger predictions, better benefiting from the GPU parallelization of convolutions. The execution time required to classify the whole Boston 22.5 km^2 test set (performed on an Intel I7 CPU @ 2.7Ghz with a Quadro K3100M GPU) is 82.21 s with the patch-based CNN against 8.47 s with the FCN. The speedup is about 10x, a relevant improvement considering the large-scale processing capabilities required by new sensors.

3.4.2 End-to-end satellite image classification

We now conduct experiments on a Pléiades image over the area of Forez, France[3]. An orthorectified color pansharpened version of the image is used, at a spatial resolution of 0.5 m^2. Our training subset amounts to 22.5 km^2. The criterion to construct the training set was to choose ten 3000×3000 tiles with at least some OSM coverage. The shape files were rasterized with GDAL[4] to create the binary reference maps. We previously showed

[3]All Pléiades images are ©CNES (2012 and 2013), distribution Airbus DS / SpotImage.
[4]http://www.gdal.org

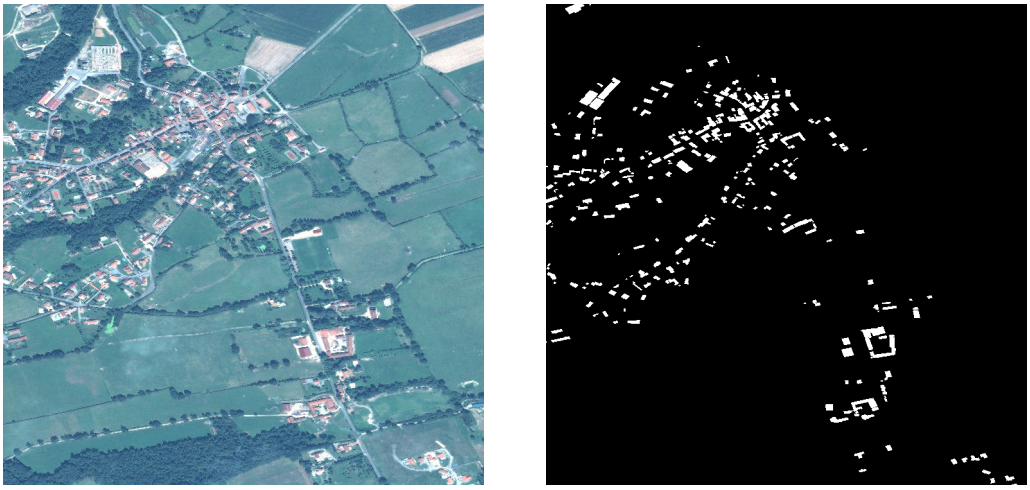


Figure 3.11: Manually labeled tile for fine-tuning (3000×3000). Left: color input, right: reference.

Figure 3.12: Close-up of the fine-tuning tile. Red borders enclose building areas.

some fragments of the reference data in Fig. 3.1, where inconsistent misregistrations and omissions are observed all over.

We manually labeled a 2.25 km^2 tile for FCN fine-tuning, and a different 2.25 km^2 tile for testing. The manual labeling takes about two hours for each of the tiles. The entire tuning tile is depicted by Fig. 3.11 and a close-up is shown in Fig. 3.12.

The fully convolutional network (FCN) described in Section 3.3.1.2, which was used for the Massachusetts dataset, is now trained with the Forez set, under a similar experimental setting. Note that this FCN was designed for images which have a 1 m^2 resolution, while Pléiades imagery features a 0.5 m^2 resolution. In order for the architectural decisions of FCN to be valid in our new dataset, one must preserve the receptive field size in terms of meters, not pixels. We thus downsample Pléiades images prior to entering the first layer of the FCN, and bilinearly upsample the output classification maps. Even

Method	Accuracy	AUC	IoU
FCN	0.99126	0.99166	0.48
FCN + Fine-tuning	0.99459	0.99699	0.66
Two-resolution FCN	0.99129	0.98154	0.47
Two-resolution FCN + Fine-tuning	0.99573	0.99836	0.72

Table 3.1: Performance evaluation on the Pléiades test set.

though a new network directly tailored to the Pléiades resolution could be designed, we favor this proven architecture to conduct our experiments. The aforementioned concepts are however general and can be used to design other networks.

After training the network on the raw OSM Forez dataset, we fine-tune the weights on the manually labeled tuning tile. The hyper-parameters of the learning algorithm are kept in the fine-tuning step, but an early stopping criterion interrupts it after 200 iterations.

To assess the performance of fine-tuning we use as criteria pixelwise accuracy and area under the ROC curve (AUC), as described in Section 3.4.1. Since there are many more non-building pixels than building pixels in this dataset, these accuracy measures might seem overly high, a well-known issue of pixelwise accuracy in imbalanced datsets [32]. We add then the intersection over union criterion (IoU), an object-based overlap measure typically used for imbalanced datasets [32]. In our case it is defined as the number of pixels labeled as buildings both in the classified image and in the ground truth, divided by the total amount of pixels labeled as such in either of them. These criteria are evaluated on the manually labeled test set, which is used neither for training nor for fine-tuning. The first two rows of Table 3.1 show that fine-tuning enhances the quality of the predictions in terms of accuracy, AUC and IoU. To confirm the significance of the accuracy, we use a McNemar's test [103], verifying that the improvement is not a result of mere luck with a probability greater than 0.99999. Besides, the IoU is improved by over a third with the fine-tuned network.

Fig. 3.13(a-d) shows the impact of fine-tuning on several amplified fragments of the test set. A greater confidence in the fine-tuned network predictions is observed. The objects exhibit better alignment to the objects of the image, even though the boundaries could better line up to the underlying edges.

Fig. 3.14 illustrates the first-layer convolutional filters learned by the initial and fine-tuned networks. We observe a combination of low- and high-frequency filters, a behavior typically observed in CNNs. We also observe edge and color blob detectors. These filters remain virtually unchanged after fine-tuning, even though no constraints are introduced to enforce this. Fine-tuning corrects the weights in the high-level layers, which suggests that the initial low-level features were useful indeed, but the inaccuracy in the labels was

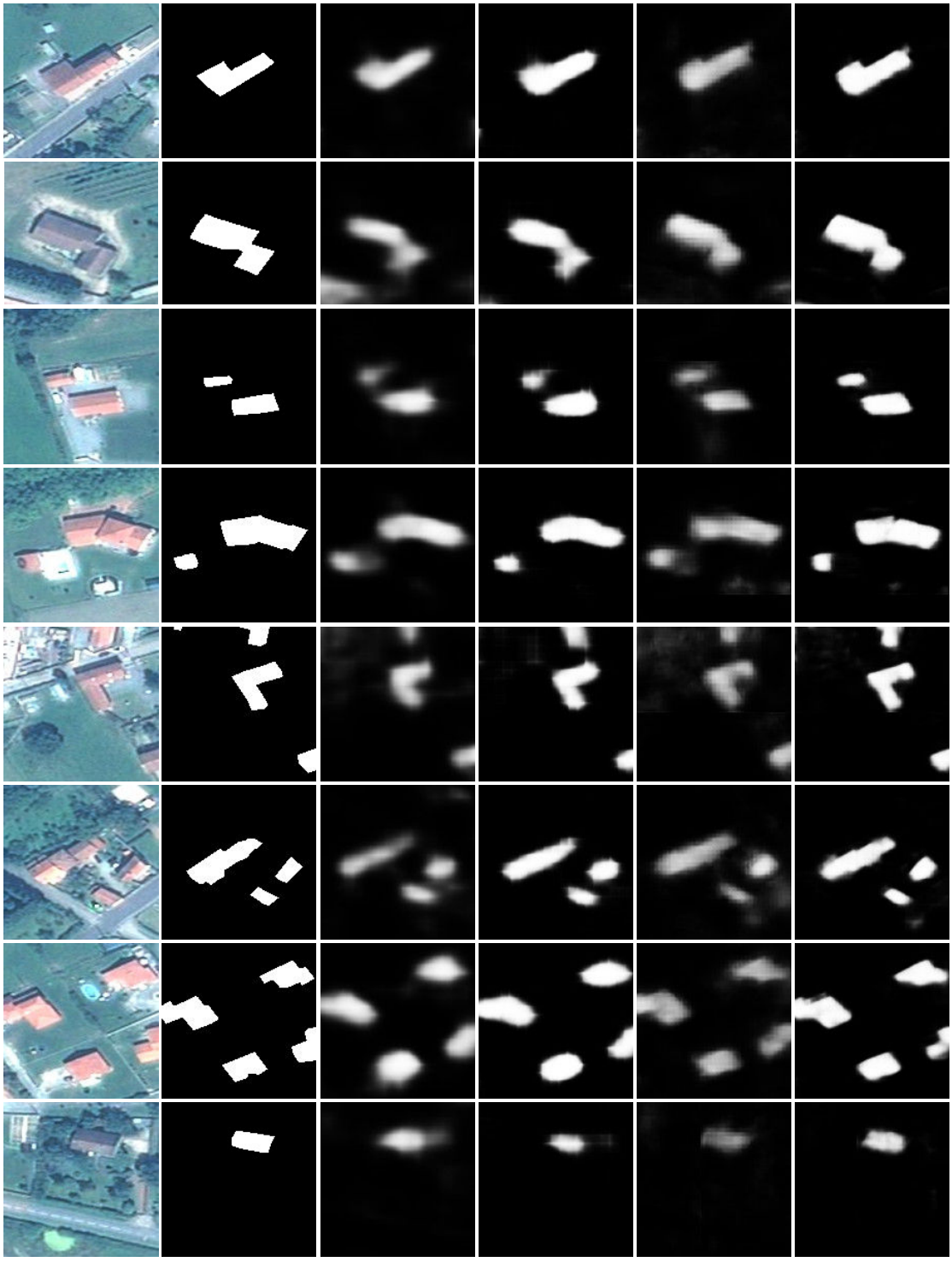
introducing fuzziness in the upper layers of the network.

We now evaluate the performance of a two-resolution network. The FCN architecture described in Section 3.3.1.2 is replaced by three two-resolution stacked modules, with resolutions $\mathcal{S} = \{1, 1/4\}$. We select $\mathcal{S} = 1/4$ as it corresponds to the degree of downsampling of the original FCN network, and $\mathcal{S} = 1$ is added to refine the predictions. The three modules learn 3×3 filters in both resolutions. The first two modules generate 64 feature maps and the last module generates a single map with the building/non-building prediction.

The two-resolution network is trained and fine-tuned in a similar setting as the FCN network. The results summarized in the last two rows of Table 3.1 show that fine-tuning significantly enhances the classification performance, and that the fine-tuned two-scale network outperforms the single scale network. Notably, IoU goes from 0.48 to 0.72, implying that objects overlap with the ground truth 50% better by adding a scale and performing fine-tuning. Note that if a scale is added but no fine-tuning is performed, there is actually a slight decrease in performance. A possible explanation for this is that including a finer scale adds even more confusion to the training algorithm if only noisy misregistered labels are provided.

Figs. 3.13(e-f) illustrate the results on visual fragments of the test set. The two-resolution network yields classification maps that better correspond to the actual image objects, and exhibit sharper angles and straighter lines. The entire classified test tile for the fine-tuned two-resolution network is depicted by Fig. 3.15c. The time required to generate this result corresponds to three hours for training on the OSM dataset, two hours to manually label an image tile and about a minute for fine-tuning. The prediction of the 3000×3000 test tile using the hardware described in Section 3.4.1 takes 3.2 seconds, and it grows linearly in the size of the image. As in Section 3.4.1, we ran an SVM on the individual pixel values (see the classification map in Fig. 3.15-b). Accuracy is 0.9487 and IoU 0.19, yielding poorer results than the presented CNN-based approaches.

As validated by the experiments, the absence of large amounts of high-quality reference data can be alleviated by providing the network with a small amount of accurate data in a fine-tuning step. Our multi-resolution neuronal modules combine reasoning at different levels of detail to effectively produce fine predictions, while keeping a reasonable number of parameters. Such a framework can be used end-to-end to perform the classification task directly from input imagery. More resolutions can be easily added and, besides the fact of being fully convolutional, there are little constraints on the architecture itself, admitting a different number of classes, input bands or number of feature maps.



(a) Color image (b) Reference (c) FCN (d) FCN + Fine-tuning (e) 2-resol. FCN (f) 2-resol. FCN + Fine-tuning

Figure 3.13: Classified fragments of the Pléiades test image. Fine-tuning increases the confidence of the predictions, and the two-scale network produces fine-grained classification maps.

Figure 3.14: First layer filters before and after fine-tuning.

(a) Color pansharpened input (b) SVM on individual pixels (c) FCN (two resolutions + fine-tuning)

Figure 3.15: Binary classification maps on the Forez test image.

3.5 Conclusion and discussion

Convolutional neural networks have become a popular classifier in the context of image analysis due to their potential to automatically learn relevant contextual features. Initially devised for the categorization of natural images, these networks must be revisited and adapted to tackle the problem of pixelwise labeling in remote sensing imagery.

We first proposed the fully convolutional architecture as the base prototype from which to derive networks for our problem. Despite their outstanding learning capability, the lack of accurate training data may limit the applicability of CNNs in realistic remote sensing datasets. We therefore proposed a two-step training approach combining the use of larger amounts of raw OpenStreetMap data and a small sample of manually labeled reference. Let us remark the increasing interest on the use of crowd-sourced data (such as OpenStreetMap) in the community, by combining it with other sources of data [72], using it as an additional input to CNNs [6] or even correcting it automatically [102]. The last ingredient required to yield a usable end-to-end framework for remote sensing image classification is to produce fine-grained classification maps, since typical CNNs tend to degrade the resolution of the output as a side effect of taking large amounts of context. We proposed a type of neuron module that simultaneously reasons at different scales.

Experiments showed that our fully convolutional network indeed outperforms previous models in multiple aspects: the accuracy of the results is improved, the visual artifacts are removed and the inference time is reduced by a factor of ten. Using such an architecture leads then to a win-win situation in which no aspect is compromised for the others. This was achieved by analyzing the role played by every layer in the network in order to propose a more appropriate architecture, showing that a deep understanding of how CNNs work is important for their success. Further experimentation showed that the two-step training approach effectively combines imperfect training data with manually labeled data to capture the dataset's generalities and its precise details. Moreover, the multi-resolution modules increase the level of detail of the classification without making the number of parameters explode, attenuating the trade-off between detection and localization. We further explore the problem of yielding high-resolution outputs in subsequent chapters.

Our overall framework shows that convolutional neural networks can be used end-to-end to process large amounts of satellite images and provide accurate pixelwise classifications.

Note that we may use the probabilities generated by a CNN in combination with other lower-level processing algorithms, such as superpixels [5] an the BPT approach

(a) Color (b) CNN (c) CNN+BPT

Figure 3.16: The CNN's output probabilities may be used in conjunction with the BPT method presented in Chapter 2.

presented in Chapter 2. It is in fact possible to construct a BPT by combining the CNN's probabilities with the image data, as in Eq. 2.8. We do not see the need of using BPT optimization with shape features now, since CNNs already identify pretty well the location of the objects. We indeed face less and less situations such as the one in Fig. 2.9(c), where entire parts of objects are assigned to the wrong class due to a poor classifier performance. The use of BPTs with CNN probabilities may help to better define the object's boundaries when these are well captured by the image. For example, Fig. 3.16(b) shows the CNN's output for the building/non-building classification problem on our Pléiades dataset. In Fig. 3.16(c) we show the result of extracting a classification from the same tree with a regularity penalty on the number of regions (regions belonging to the same class are illustrated with random shades of the same color). We observe that the building object better aligns to the real boundaries in the color image. Another advantage of this hybrid method is that we also provide a superpixel subdivision of the background class, and we can even navigate the tree to extract information at different levels of detail. However, we could not consistently find cases such as the one in Fig. 3.16 where the use of BPTs seems to significantly improve the objects. Therefore, we decided to continue exploring the CNN direction, with the goal of providing high-quality high-resolution classification maps.

Chapter 4

Learning iterative processes to enhance classification maps

Convolutional neural networks have become one of the most studied methods for pixelwise image classification, both in natural images and in remote sensing. One of the key concerns is the coarseness of the outputs, since elementary CNNs fail to yield fine-grained high-resolution classification maps. In addition to the eventual imperfection of the training data, this is due to a well-known trade-off between the recognition power of CNNs (to identify objects) and their localization power (to precisely outline them). One tendency to overcome this issue is to design new architectures specifically intended to provide high-resolution outputs. In Chapter 3 we proposed a step forward in this direction, through a network that combined two resolutions of reasoning in its layers. In Chapter 5 we also resume this direction of work. A second popular option is to add a dedicated post-processing module to correct the fuzzy output of the original CNN. The base CNN is used as a rough classifier of the objects' locations, and then the output classification maps are processed so that the objects better align to real image edges. This step requires to use the original image as guidance to improve the fuzzy maps.

In this chapter we explore this second option, i.e., incorporating image information a posteriori in an enhancement module that sharpens the coarse classification maps around objects. The pioneering approach in this direction [21] uses a fully connected conditional random field (CRF) to perform the enhancement. Since then, other methods have been presented [20, 171].

An algorithm to enhance coarse classification maps requires, on the one hand, to define the image features to which the objects must be attached. This is data-dependent, since not every image edge necessarily coincides with an object boundary. On the other

hand, we must also decide which enhancement algorithm to use, and tune it (e.g., the fully connected CRF in [21]). Besides the efforts that this requires, we could also imagine that the optimal approach would go beyond the algorithms presented in the literature. For example we could perform different types of corrections on the different classes, based on the type of errors that are often present in each of them.

Our goal in this chapter is to develop a system that learns the appropriate enhancement algorithm itself, instead of designing it by hand. This involves learning not only the relevant features but also the rationale behind the enhancement principle, thus intensively leveraging the power of machine learning.

To achieve this, we first formulate a generic partial differential equation governing a broad family of iterative enhancement algorithms. This generic equation conveys the idea of progressively refining a classification map based on local cues, yet it does not provide the specifics of the algorithm. We then observe that such an equation can be expressed as a combination of common neural network layers, whose learnable parameters define the specific behavior of the algorithm. We then implement the whole iterative enhancement process as a recurrent neural network (RNN).

As in Chapter 3, we use large amounts of possibly inaccurately labeled data to train the coarse CNN. We also assume we can afford to manually label small amounts of data. The RNN is provided with a piece of manually labeled image, and trained end to end to improve the otherwise coarse classification maps. It automatically discovers relevant data-dependent features to enhance the classification and the specifics of an iterative process to do so.

4.1 Related work

Enhancing classification

The idea of using an enhancement algorithm on the CNN's output has been a recent trend in the natural image semantic segmentation domain. Notably, the authors of the Deeplab network [21] added a fully connected conditional random field (CRF) on top of both the CNN and the input color image, in order to enhance the classification maps. Zheng et al. [171] recently reformulated the fully connected CRF of Deeplab as a recurrent neural network (RNN), and Chen et al. [20] designed an RNN that emulates the domain transform filter [51]. Such a filter is used to sharpen the classification maps around image edges, which are themselves detected with a CNN. In these methods the enhancement algorithm is designed beforehand and only few parameters that rule the algorithm are

learned as part of the network's parameters. The innovating aspect of these RNN-based approaches, compared to the original Deeplab, is that both steps (coarse classification and enhancement) can be seen as a single end-to-end network and optimized simultaneously.

The idea of enhancing classification with a CRF was also recently used in the context of aerial image labeling [114], as well as the fully connected CRF formulation [101, 131].

Learning iterative algorithms

To our knowledge, little work has explored the idea of learning an iterative algorithm. In the context of image restoration, the preliminary work by Liu et al. [91, 92] proposed to optimize the coefficients of a linear combination of predefined terms. Chen et al. [23] later modeled this problem as a diffusion process and used an RNN to learn the linear filters involved as well as the coefficients of a parametrized nonlinear function. Our problem is however different, in that we use the image as guidance to update a classification map, and not to restore the image itself. Besides, while we drew inspiration on diffusion processes, we are also interested in learning other iterative processes like active contours, thus we do not restrict our system to diffusions but consider all PDEs.

Recurrent neural networks

In a recurrent neural network (RNN), and contrary to feed-forward networks, the graph of message passing among neurons contains cycles [61]. They have been particularly popular in the domain of natural language processing, since its internal memory can be used to process text or speech sequences of arbitrary length [46, 106, 124], without the need of increasing the amount of trainable parameters.

RNNs can be trained by an algorithm known as *backpropagation through time* [159]. The idea is to "unroll" the cycle a finite number of times, thus constructing a feed-forward neural network. Such network is trained by back-propagation as usual, with the constraint that the weights should be the same across different cycles (a process often referred to as weight sharing). If independent weights are used instead, the idea of recurrency would be lost and the architecture would just represent a traditional feed-forward network. In back-propagation, weight sharing is enforced by averaging the gradient computed over all the shared parameters, and using that value in the gradient descent.

Deep RNNs are known to be difficult to train [116], due to the problem of vanishing and exploding gradients (which also occurs in deep networks in general [56]). The long-short term memory (LSTM) [66, 53] has become one of the most popular types of RNN to enhance the training process of deep RNNs.

(a) Color image

(b) CNN heat map

(c) Ground truth

Figure 4.1: Sample classification of buildings with a CNN.

4.2 Proposed method

Let us assume we are given a set of score (or "heat") maps u_k, one for each possible class $k \in \Omega$, in a pixelwise labeling problem. The score of a pixel reflects the likelihood of belonging to a class, according to the classifier's predictions. The final class assigned to every pixel is the one with maximal value u_k. Alternatively, a softmax function can be used to interpret the results as probability scores: $P(k) = e^{u_k} / \sum_{j \in \Omega} e^{u_j}$, as usually done in CNN literature. Fig. 4.1 shows a sample of the type of fuzzy heat map outputted by the CNN presented in Chapter 3, for the class 'building'. More examples can be found in Fig. 3.13(c).

Our goal is to combine the score maps u_k with information derived from the input image (e.g., edge features) to sharpen the scores near the real objects in order to enhance the classification.

One way to perform such a task is to progressively enhance the score maps by using partial differential equations (PDEs). In the following section, we first describe different types of PDEs we could certainly imagine to design in order to solve our problem. Instead of discussing which one is the best, we later propose a generic iterative process to enhance the classification maps without specific constraints on the algorithm rationale. Finally, we show how this equation can be expressed and trained as a recurrent neural network (RNN).

4.2.1 Partial differential equations (PDEs)

We can formulate a variety of diffusion processes applied to the maps u_k as partial differential equations. For example, the heat flow is described as:

$$\frac{\partial u_k(x)}{\partial t} = \operatorname{div}(\nabla u_k(x)), \tag{4.1}$$

where $\mathrm{div}(\cdot)$ denotes the divergence operator in the spatial domain of x. Applying such a diffusion process in our context would smooth out the heat maps. Instead, our goal is to design an image-dependent smoothing process that aligns the heat maps to the image features. A natural way of achieving this is to modulate the gradient in Eq. 4.1 by a scalar function $g(x, I)$ that depends on the input image I:

$$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(g(I, x)\nabla u_k(x)). \tag{4.2}$$

Eq. 4.2 is similar to the Perona-Malik diffusion [118] with the exception that Perona-Malik uses the smoothed function itself to guide the diffusion. $g(I, x)$ denotes an edge-stopping function that takes low values near borders of $I(x)$ in order to slow down the smoothing process there.

Another possibility would be to consider a more general variant in which $g(I, x)$ is replaced by a matrix $D(I, x)$, acting as a diffusion tensor that redirects the flow based on image properties instead of just slowing it down near edges:

$$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(D(I, x)\nabla u_k(x)). \tag{4.3}$$

This formulation is related to the so-called anisotropic diffusion process [158].

Alternatively, one can draw inspiration from the level set framework. For example, the geodesic active contours technique formulated as level sets translates into:

$$\frac{\partial u_k(x)}{\partial t} = |\nabla u_k(x)|\mathrm{div}\left(g(I, x)\frac{\nabla u_k(x)}{|\nabla u_k(x)|}\right). \tag{4.4}$$

Such a formulation favors the zero level set to align with minima of $g(I, x)$ [19]. Schemes based on Eq. 4.4 could then be used to improve heat maps u_k, provided they are scaled so that segmentation boundaries match zero levels.

As shown above, many different PDE approaches can be devised to enhance classification maps. However, several choices must be made to select the appropriate PDE and tailor it to our problem. For example, one must choose the edge-stopping function $g(I, x)$ in Eqs. 4.2, 4.4. Common choices are exponential or rational functions on the image gradient [118], which in turn requires to set an edge-sensitivity parameter. Extensions to the original Perona-Malik approach could also be considered, such as a popular regularized variant that computes the gradient on a Gaussian-smoothed version of the input image [158]. In the case of opting for anisotropic diffusion, one must design $D(I, x)$.

Instead of using trial and error to perform such design, our goal is to let a machine

learning system discover by itself a useful iterative process for our task.

4.2.2 A generic classification enhancement process

PDEs are usually discretized in space by using finite differences, which represent derivatives as discrete convolution filters. We build upon this scheme to write a generic discrete formulation of an iterative enhancement process.

Let us consider that we take as input a score map u_k (for class k) and, in the most general case, an arbitrary number of feature maps $\{g_1, ..., g_p\}$ derived from image I. In order to perform differential operations, of the type $\{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial^2}{\partial x \partial y}, \frac{\partial^2}{\partial x^2}, ...\}$, we consider convolution kernels $\{M_1, M_2, ...\}$ and $\{N_1^j, N_2^j, ...\}$ to be applied to the heat map u_k and to the features g_j derived from image I, respectively. While we could certainly directly provide a bank of filters M_i and N_i^j in the form of Sobel operators, Laplacian operators, etc., we may simply let the system learn the required filters. We group all the feature maps that result from applying these convolutions, in a single set:

$$\Phi(u_k, I) = \left\{ M_i * u_k, \;\; N_l^j * g_j(I) \; ; \;\; \forall i, j, l \right\}. \tag{4.5}$$

Let us now define a generic discretized scheme as:

$$\frac{\partial u_k(x)}{\partial t} = f_k\big(\; \Phi(u_k, I)(x) \; \big), \tag{4.6}$$

where f_k is a function that takes as input the values of all the features in $\Phi(u_k, I)$ at an image point x, and combines them. While convolutions M_i and N_i^j convey the "spatial" reasoning, e.g., gradients, f_k captures the combination of these elements, such as the products in Eqs. 4.2 and 4.4.

Instead of deriving an arbitrary number of possibly complex features $N_i^j * g_j(I)$ from image I, we can think of a simplified scheme in which we directly operate on I, by considering only convolutions: $N_i * I$. The list of functionals considered in Eq. 4.6 is then

$$\Phi(u_k, I) = \left\{ M_i * u_k, \;\; N_j * I \; ; \;\; \forall i, j \right\} \tag{4.7}$$

and consists only of convolutional kernels directly applied to the heat maps u_k and to the image I. From now on, we here stick to this simpler formulation, yet we acknowledge that it may be eventually useful to work on a higher-level representation rather than on the input image itself. Note that if one restricts functions f_k in Eq. 4.6 to be linear, we still obtain the set of all linear PDEs. We consider *any* function f_k, thus introducing

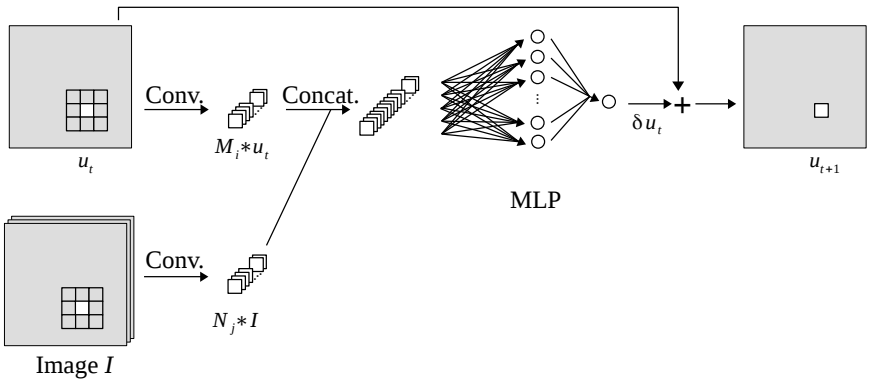


Figure 4.2: One enhancement iteration represented as common neural network layers.

non-linearities.

PDEs are usually discretized in time, taking the form:

$$u_{k,t+1}(x) = u_{k,t}(x) + \delta u_{k,t}(x), \tag{4.8}$$

where $\delta u_{k,t}$ denotes the overall update of $u_{k,t}$ at time t.

Note that the convolution filters in Eqs. 4.5 and 4.7 are class-agnostic: M_i, N_j and N_l^j do not depend on k, while f_k may be a different function for each class k. Function f_k thus determines the contribution of each feature to the equation, contemplating the case in which a different evolution might be optimal for each of the classes, even if just in terms of a time-step factor. In the next section we detail a way to learn the update functions $\delta u_{k,t}$ from training data.

4.2.3 Iterative processes as RNNs

We now show that the generic iterative process can be implemented as an RNN, and thus trained from labeled data. This stage requires to provide the system with a piece of accurately labeled ground truth (see e.g., Fig. 3.12).

Let us first show that one iteration, as defined in Eqs. 4.6-4.8, can be expressed in terms of common neural network layers. Let us focus on a single pixel for a specific class, simplifying the notation from $u_{k,t}(x)$ to u_t. Fig. 4.2 illustrates the proposed network architecture. Each iteration takes as input the image I and a given heat map u_t to enhance at time t. In the first iteration, u_t is the initial coarse heat map to be improved, outputted by another pre-trained neural network in our case. From the heat map u_t we derive a series of filter responses, which correspond to $M_i * u_t$ in Eq. 4.7. These

responses are found by computing the dot product between a set of filters M_i and the values of $u_{k,t}(\cdot)$ in a spatial neighborhood of a given point. Analogously, a set of filter responses are computed at the same spatial location on the input image, corresponding to the different $N_j * I$ of Eq. 4.7. These operations are convolutions when performed densely in space, $N_j * I$ and $M_i * u_t$ being feature maps of the filter responses.

These filters are then "concatenated", forming the pool of features Φ coming from both the input image and the heat map, as in Eq. 4.7, and inputted to f_k in Eq. 4.6. We must now learn the function δu_t that describes how the heat map u_t is updated at iteration t (see Eq. 4.8), based on these features.

Eq. 4.6 does not introduce specifics about function f_k. In (4.1)-(4.4), for example, it includes products between different terms, but we could certainly imagine other functions. We therefore model δu_t through a multilayer perceptron (MLP), because it can approximate any function within a bounded error [11]. We include one hidden layer with nonlinear activation functions followed by an output neuron with a linear activation (a typical configuration for regression problems), although other MLP architectures could be used. Applying this MLP densely is equivalent to performing convolutions with 1×1 kernels at every layer. The implementation to densely label entire images is then straightforward.

The value of δu_t is then added to u_t in order to generate the updated map u_{t+1} (see Fig. 4.2). This addition is performed pixel by pixel in the case of a dense input. Note that although we could have removed this addition and let the MLP directly output the updated map u_{t+1}, we opted for this architecture since it is more closely related to the equations and better conveys the intention of a progressive refinement of the classification map. Moreover, learning δu_t instead of u_{t+1} has a significant advantage at training time: a random initialization of the networks' parameters centered around zero means that the initial RNN represents an iterative process close to the identity (with some noise). Training uses the asymmetry induced by this noise to progressively move from the identity to a more useful iterative process.

The overall iterative process is implemented by unrolling a finite number of iterations, as illustrated in Fig. 4.3, under the constraint that the parameters are shared among all iterations. Such a sharing is enforced at training time by a simple modification to the back-propagation training algorithm where the derivatives of every instance of a weight at different iterations are averaged [159]. Note that the spatial features are shared across the classes, while a different MLP is learned for each of them, following Eq. 4.6. As depicted by Fig. 4.3 and conveyed in the equations, the features extracted from the input image are independent of the iteration.

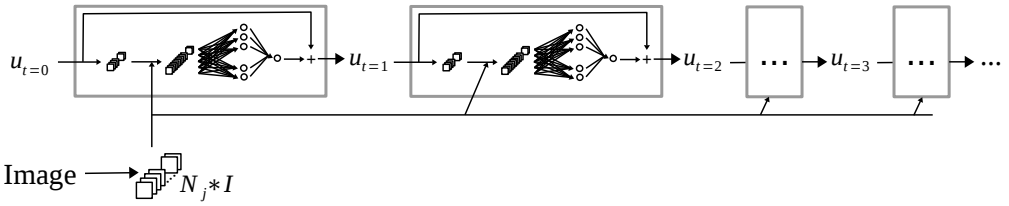


Figure 4.3: Modules of Fig. 4.2 are stacked (while sharing parameters) to implement an RNN.

The RNN of Fig. 4.3 represents a dynamical system that iteratively improves the class heat maps. Training such an RNN amounts to finding the optimal dynamical system for our enhancement task.

4.3 Experiments

4.3.1 Dataset and implementation details

We perform our experiments on the Pléiades satellite image used in Chapter 3, over the area of Forez, France. We use the same 22.5 km^2 training dataset, but this time we augment it to include three classes: *building*, *road* and *background*, obtained by rasterizing the raw OSM maps. In the case of roads, we rasterize the center line indicated in OSM with a fixed width of 7 meters (in accordance to [107]). Misregistrations and omissions are present all over the dataset. Buildings tend to be misaligned or omitted, while many roads in the ground truth are not visible in the image (or the other way around).

This dataset is used to train the initial coarse CNN. Such a CNN is based on the fully convolutional network presented in Chapter 3, but it outputs three classification heat maps to account for the newly added class. We also add extra feature maps along the layers, to account for the additional reasoning required to label the road class. The overall four-layer architecture is as follows: 64 conv. filters (12×12, stride 4) $\rightarrow$ 128 conv. filters (3×3) $\rightarrow$ 128 conv. filters (3×3) $\rightarrow$ 3 conv. filters (9×9). A deconvolutional layer is added on top to upsample the classification maps to the original resolution. As in the previous chapter, we also downsample the Pléiades images by a factor of two before feeding them to our networks, and bilinearly upsample the outputs.

We also use the same two manually labeled tiles of 2.25 km^2 as before, to train and test the RNN at enhancing the predictions of the coarse network. We manually labeled the additional class in these tiles. We denote them by *enhancement* and *test* sets, respectively. Note that our RNN system must discover an algorithm to refine an

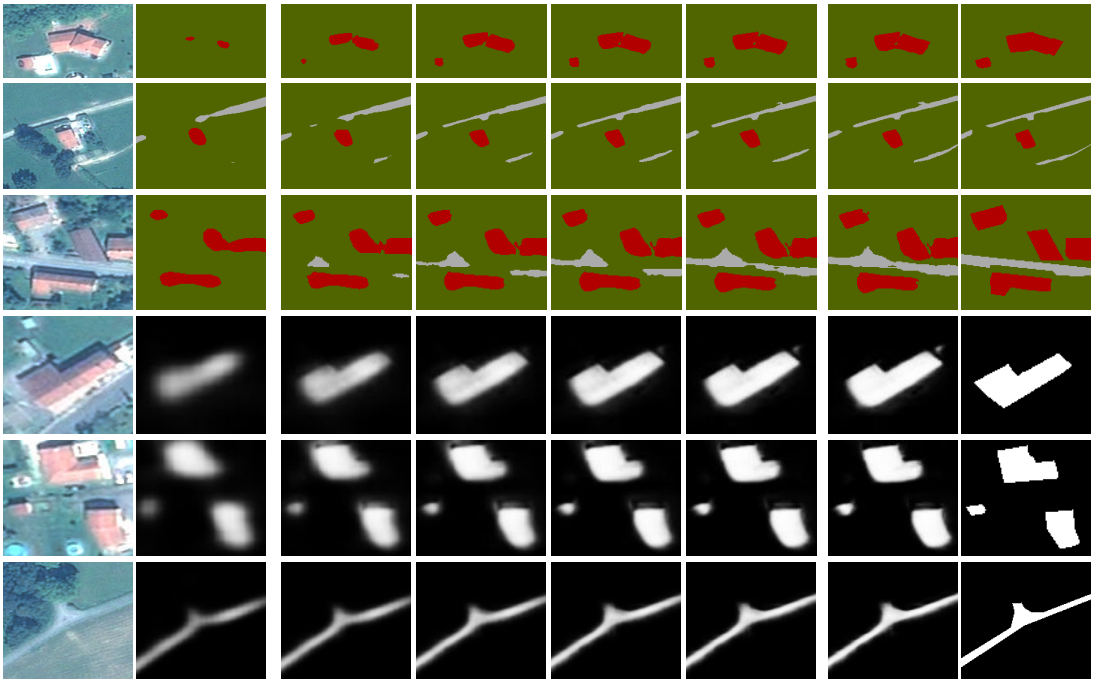


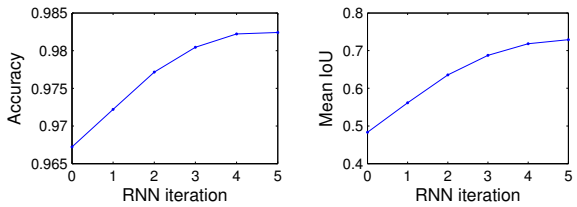
Figure 4.4: Evolution of fragments of classification maps (top rows) and single-class fuzzy scores (bottom rows) through RNN iterations.

existing classification map, and not to conduct the classification itself, hence a smaller training set should be sufficient for this stage.

Let us now detail the implementation of the RNN described in Sec. 4.2.3. We unroll five RNN iterations and learn 32 M_i and 32 N_j filters, both of spatial dimensions 5×5. As explained in Sec. 4.2.3, an independent MLP is learned for every class, using 32 hidden neurons each and with rectified linear activations. Training is performed on random patches and with the cross-entropy loss function, as done with the coarse CNN. The employed gradient descent algorithm is AdaGrad [39], which exhibits a faster convergence in our case, using a base learning rate of 0.01 (higher values make the loss diverge). All weights are initialized randomly by sampling from a distribution that depends on the number of neuron inputs, as described in [56]. We trained the RNN for 50,000 iterations, until observing convergence of the training loss, which took around four hours on a single GPU.

Method	Overall accuracy	Mean IoU	Class-specific IoU Build.	Road	Backg.
CNN	96.72	48.32	38.92	9.34	96.69
CNN+CRF	96.96	44.15	29.05	6.62	96.78
CNN+RNN$^{=}$	97.78	65.30	59.12	39.03	97.74
CNN+RNN	**98.24**	**72.90**	**69.16**	**51.32**	**98.20**

(a) Numerical comparison (in %)



(b) Evolution through RNN iterations

Figure 4.5: Quantitative evaluation on Pléaiades images test set over Forez, France.

4.3.2 Results

In the following, we report the results obtained by using the proposed method on the Pléiades dataset. Fig. 4.4 provides closeups of results on different fragments of the test dataset. The initial and final maps (before and after the RNN enhancement) are depicted, as well as the intermediate results through the RNN iterations. We show both a set of final classification maps and some single-class fuzzy probability maps. We can observe that as the RNN iterations advance, the classification maps are refined and the objects better align to image edges. The fuzzy probabilities become more confident, sharpening object boundaries. To quantitatively assess this improvement we compute two measures on the test set: the overall accuracy (proportion of correctly classified pixels) and the intersection over union (IoU) [93]. As summarized in the table of Fig. 4.5(a), the performance of the original coarse CNN (denoted by CNN) is significantly improved by attaching our RNN (CNN+RNN). Both measures increase monotonously along the intermediate RNN iterations, as depicted in Fig. 4.5(b).

The initial classification of roads has an overlap of less than 10% with the roads in the ground truth, as shown by its individual IoU. The RNN makes them emerge from the background class, now overlapping the ground truth roads by over 50%. Buildings also become better aligned to the real boundaries, going from less than 40% to over 70% overlap with the ground truth buildings. This represents a multiplication of the IoU by a factor of 5 for roads and 2 for buildings, which indicates a significant improvement at outlining and not just detecting objects.

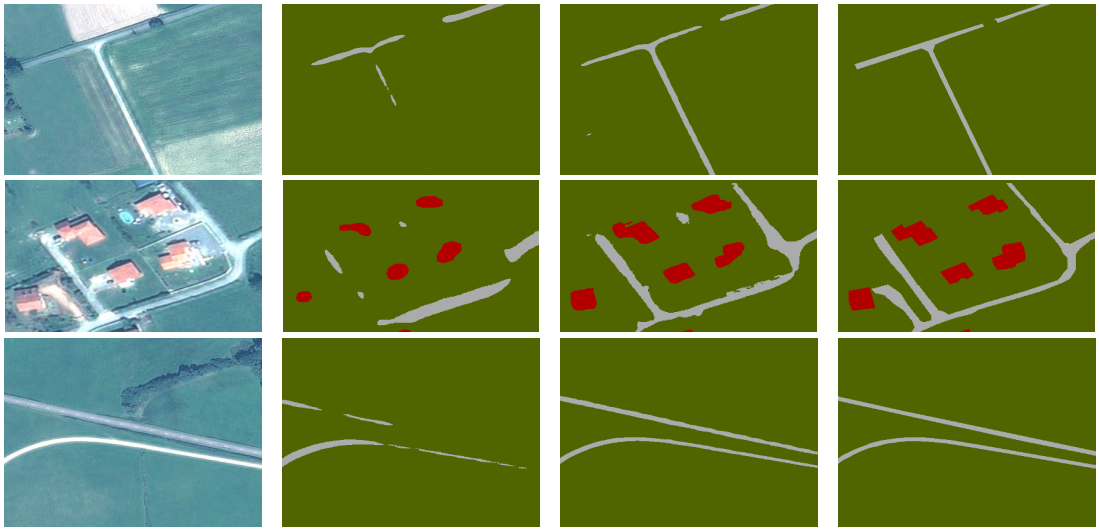


Figure 4.6: Initial coarse classifications and the enhanced maps by using RNNs.

Additional visual fragments before and after the RNN refinement are shown in Fig. 4.6. We can observe in the last row, for example, that the iterative process learned by the RNN both thickens and narrows the roads depending on the location.

We also compare our RNN to the approach in [21] (here denoted by CNN+CRF), where a fully connected CRF is coupled both to the input image and the coarse CNN output, in order to refine the predictions. This is the idea behind the Deeplab network, which constitutes one of the most important current baselines in the semantic segmentation community. While the CRF itself could also be implemented as an RNN [171], we here stick to the original formulation because the CRF as RNN idea is only interesting if we want to train the system end to end (i.e., together with the coarse prediction network). In our case we wish to leave the coarse network as is, otherwise we risk overfitting it to this much smaller set. We thus simply use the CRF as in [21] and tune the energy parameters by performing a grid search using the enhancement set as a reference. Five iterations of inference on the fully connected CRF were performed in every case.

To further analyze our method, we also consider an alternative enhancement RNN in which the weights of the MLP are shared across the different classes (which we denote by CNN+RNN$^{=}$). This enforces the system to learn the same function for updating all the classes, instead of a class-specific function.

Numerical results are included in the table of Fig. 4.5(a) and the classification maps are shown in Fig. 4.7. Close-ups of these maps are included in Fig. 4.8 to facilitate com-

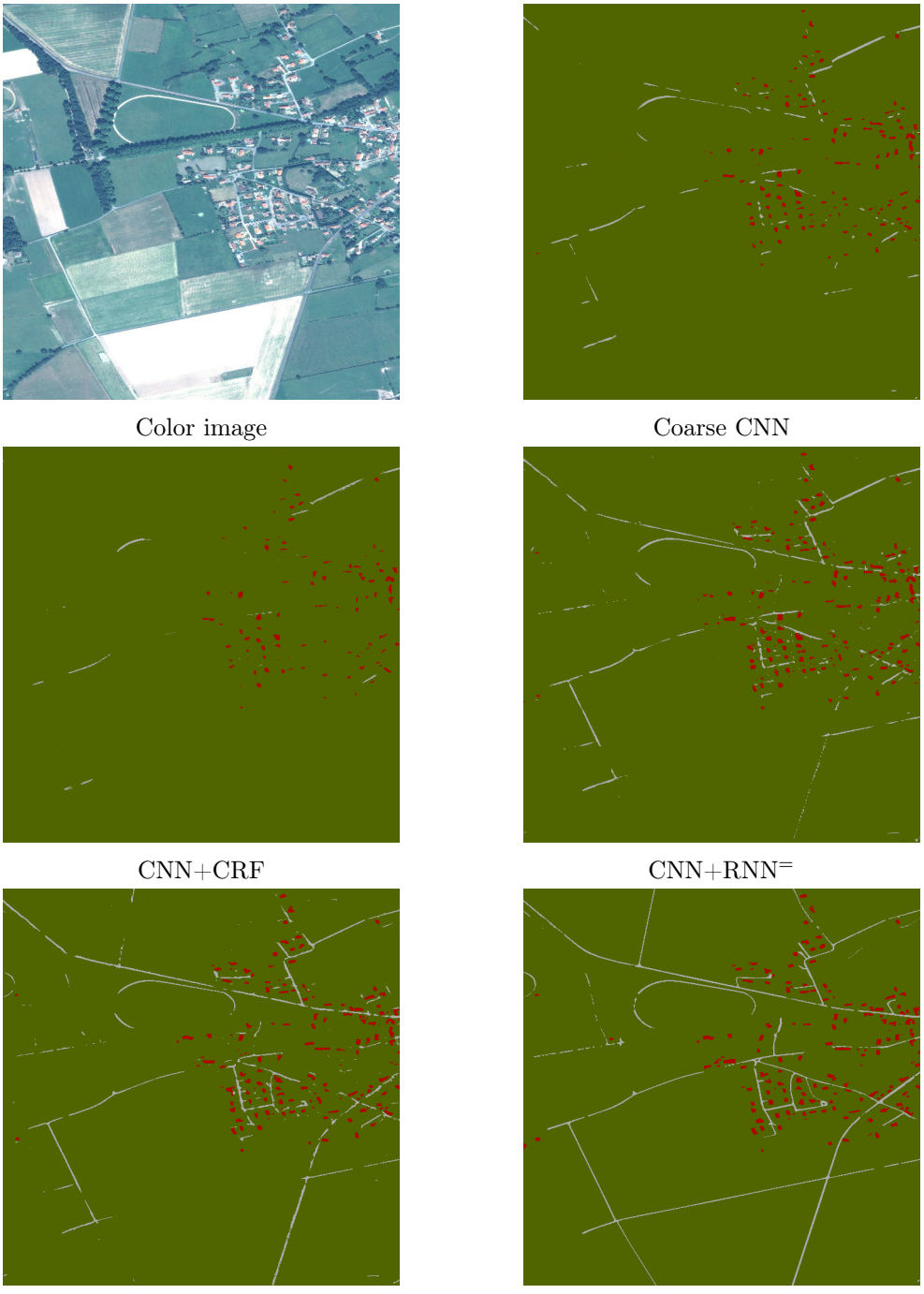


Figure 4.7: Visual comparison on a Pléiades satellite image tile of size 3000×3000 covering 2.25 km^2.

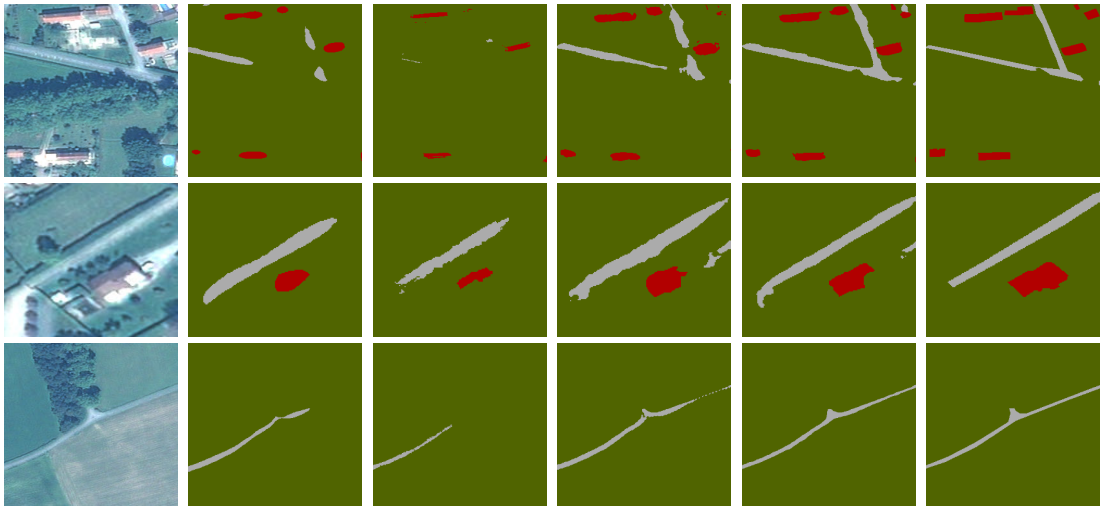


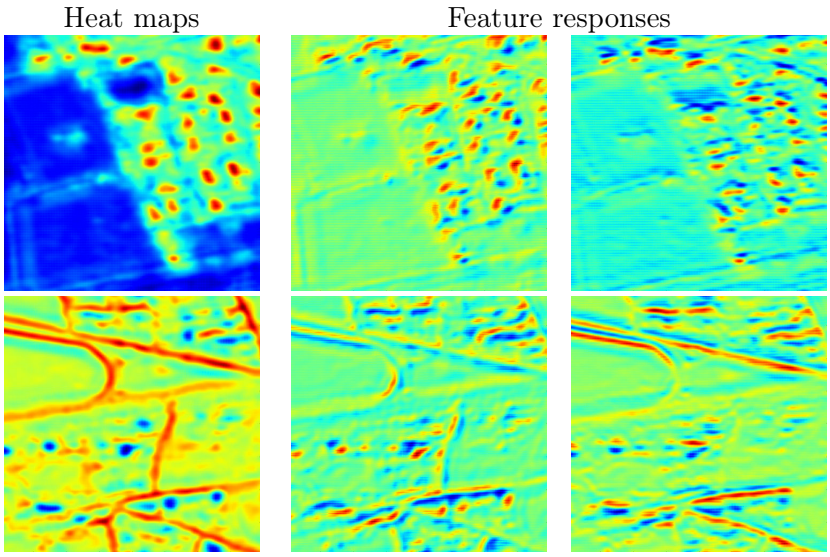
Figure 4.8: Visual comparison on closeups of the Pléiades dataset.

parison. The CNN+CRF approach does sharpen the maps but this often occurs around the wrong edges. It also makes small objects disappear in favor of larger objects (usually the background class) when edges are not well marked, which explains the mild increase in overall accuracy but the decrease in mean IoU. While the CNN+RNN$^=$ outperforms the CRF, both quantitative and visual results are beaten by the CNN+RNN, supporting the importance of learning a class-specific enhancement function.

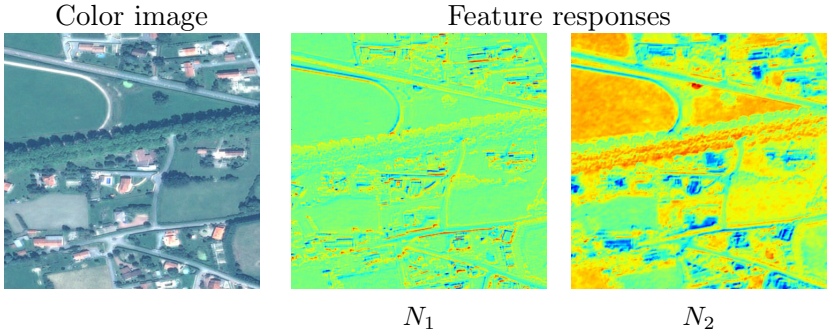
To validate the importance of using a recurrent architecture, and following Zheng et al. [171], we retrained our system considering every iteration of the RNN as an independent step with its own parameters. After training for the same number of iterations, it yields a lower performance on the test set compared to the RNN and a higher performance on the training set. If we keep on training, the non-recurrent network still enhances its training accuracy while performing poorly on the test set, implying a significant degree of overfitting with this variant of the architecture. This provides evidence that constraining our network to learn an iterative enhancement process is crucial for its success.

4.3.2.1 Feature visualization

Though it is difficult to interpret the overall function learned by the RNN, especially the part of the multi-layer perceptron, there are some things we expect to find if we analyze the spatial filters M_i and N_j learned by the RNN (see Eq. 4.7). Carrying out this analysis is a way of validating the behavior of the network.



(a) Filter M_1 acts like a gradient operator in the South-East direction and M_2 in the North direction (top: building, bottom: road).



(b) N_1 acts like a gradient operator in the North direction and N_2 highlights green vegetation.

Figure 4.9: Feature responses (red: high, blue: low) to selected M_i and N_j filters, applied to the heat maps and input image respectively (see Eq. 4.7).

The iterative process learned by the RNN should combine information from both the heat maps and the image at every iteration (since the heat maps constitute the prior on where the objects are located, and the image guides the enhancement of these heat maps). A logical way of enhancing the classification is to align the high-gradient areas of the heat maps with object boundaries. We expect then to find derivative operators among the filters N_j applied to the heat maps. Concerning the image filters N_j, we expect to find data-dependent filters (e.g., image edge detectors) that help identify the location of object boundaries.

To interpret the meaning of the filters learned by the RNN we plotted the map of responses of a sample input to the different filters. We here show some examples. Fig. 4.9(a) illustrates fragments of heat maps of the *building* and *road* classes, and the responses to two of the filters M_i learned by the RNN. When analyzing these responses we can observe that they act as gradients in different directions, confirming the expected behavior. Fig. 4.9(b) illustrates a fragment of the color image and its response to two filters N_j. One of them acts as a gradient operator and the other one highlights green vegetation, suggesting that this information is used to enhance the classification maps.

4.4 Concluding remarks

In this chapter we presented a recurrent neural network (RNN) that learns how to refine the coarse output of another neural network, in the context of pixelwise remote sensing image classification. The inputs are both the coarse classification maps to be corrected and the original color image. The output at every RNN iteration is an update to the classification map of the previous iteration, using the color image for guidance.

Little human intervention is required, since the specifics of the enhancement algorithm are not provided by the user but learned by the network itself. For this, we analyzed different iterative alternatives and devised a general formulation that can be interpreted as a stack of common neuron layers. At training time, the RNN discovers the relevant features to be taken both from the classification map and from the input image, as well as the function that combines them.

The experiments on satellite imagery show that the classification maps are improved significantly, increasing the overlap of the foreground classes with the ground truth, and outperforming other approaches by a large margin. The proposed method yields classification maps that not only detect but also outline the objects.

An important conclusion we can also draw from this chapter is that RNNs are capable of learning the specifics of an iterative process.

Chapter 5

High-resolution Classification with Convolutional Neural Networks

While neural networks have existed for a few decades, a combination of recent advances has facilitated their development as *deep learning* techniques. One example is the use of convolutional layers. While these layers impose significant restrictions to the neuronal connections compared to other approaches, the restrictions are well grounded in the domain of image analysis, reducing the optimization search space in a sensible way. This directs the network to learn a more appropriate function, yielding better results. The lesson learned is that finding the right type of architecture for a given problem often boosts the performance of neural networks. Moreover, fewer computational resources are required for training and classification.

Convolutional neural networks were originally devised for the image *categorization* problem, i.e., the assignment of one label to an entire image, and a sort of architectural prototype was incrementally developed in the community. It typically includes a combination of convolutional layers, pooling layers and rectifed linear units, followed by fully connected layers [76].

The goal of this chapter is to discuss the ideal prototype for dense pixelwise classification (i.e., assigning a class to every pixel), because the usual catogorization networks cannot be directly transferred. Indeed, while categorization networks are devised to lose spatial precision in order to identify objects that come in different appearances, dense classification networks should preserve the spatial resolution to correctly locate object boundaries. This is not straightforward to implement, because of a well-known trade-off between recognition and localization [21, 93], due to the need to keep the networks small (and thus more efficient and easier to train). Since both qualities are required in dense classification at the same time, it is important to design specific architectures for this

problem.

Through the development of this thesis, there were notable contributions in the vision community toward a finer pixelwise classifiction network (e.g., [8, 112]). Meanwhile, in remote sensing, the ISPRS Semantic Labeling Contest was launched [70], which became a benchmark to compare high-resolution classification methods. The ISPRS datasets consist of a series of 5–9 cm spatial resolution aerial images labeled into multiple classes. These datasets particularly highlight the specific challenges of aerial image labeling, where we require to outline small objects with a high spatial precision. While a multitude of such networks have been recently proposed, we argue that just by doing a proper analysis of the architecture required for the problem we may develop simpler, more efficient networks to achieve equivalent or even better results.

Our first contribution is a detailed review and analysis of the main families of CNN architectures proposed recently for the dense classification problem (Section 5.2). We group the different methods into three categories: *dilation* (e.g., [38, 131]), *deconvolution* (e.g., [8, 112, 117, 155]) and *skip* (e.g., [93, 101]) networks. These categories are different from each other in the way of addressing the aforementioned recognition/localization trade-off. For example, while the networks by Long et al. [93] and Marmanisa et al. [101] are substantially different in terms of structure and application domain, they are both *skip* networks in how they manage to provide a high-resolution output. After establishing the desired properties of a dense classification architecture, we position the different families of networks with respect to these properties. Let us recall that an alternative way of yielding higher-resolution outputs is to include post-processing modules, such as fully connected CRFs [21, 114, 131]. We also explored this direction of work in the previous chapter, with a recurrent neural network. However, we now focus on architectures that are specifically designed to provide a high-resolution output.

We then present a novel network architecture, referred to as MLP (after *multi-layer perceptron*), in Section 5.3. Derived from the notion of *skip* network, the MLP architecture yields high flexibility and expressiveness by extracting features at different resolutions (and thus at different levels of details), and *learning* how to combine them in order to generate fine-grained classification maps. We conduct experiments to compare the different approaches on the datasests proposed as part of the ISPRS Semantic Labeling Contest (Section 5.4).

Finally, we create a dataset for aerial image labeling that covers multiple dissimilar regions of the earth (Section 5.5). Most existing remote sensing classification datasets cover restricted regions and the training and test areas have a very similar appearance. We thus developed a new dataset to evaluate the capability of CNNs to generalize to

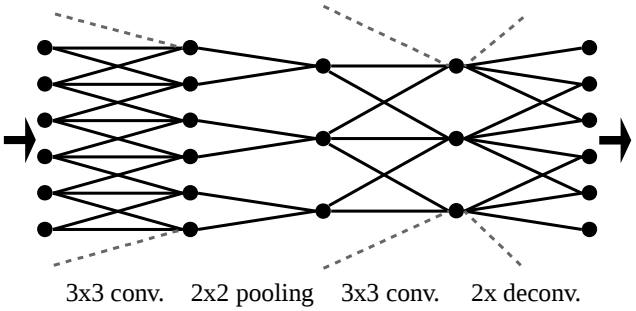


Figure 5.1: Lateral view of a fully convolutional network (dashed lines indicate inputs that have been padded in convolutional layers and cropped in the deconvolutional layer to preserve spatial dimensions).

different regions. The dataset covers a wide range of urban settlement appearances, from different geographic locations. Moreover, the cities included in the test set are different from those of the training set. The dataset was publicly released and we present our experiments using the CNNs developed in the first part of this chapter.

5.1 Background on receptive fields

Before reviewing the relevant literature, we here spend a few paragraphs to explain some additional concepts, useful for the development of this chapter. In particular, we comment on receptive fields and downsampling. For an introduction to convolutional neural networks (CNNs) in general, we refer the reader to Section 3.2.

In CNNs, the *receptive field* denotes the spatial extent of the input image connected to a certain neuron, possibly indirectly through other neurons in previous layers: it is the set of pixels on which a neuron depends. In other words, it quantifies how far a neuron can "see" in the image. In most applications, a large amount of spatial context must be taken into account in order to successfully label the images. For example, to deduce that a certain pixel belongs to a rooftop, it might not be enough to just consider its individual spectrum: we might need to observe a large patch around this pixel, taking into account geometry and structure of the objects, to infer its correct class.

Neural networks for image analysis should thus be designed to accumulate, through their layers, a large enough receptive field. While a straightforward way to do it is to use large convolution kernels, this is not a common practice mostly due to its computational complexity. Besides, this would aim at learning large filters all at once, with millions of parameters. It is preferable to learn a hierarchy of small filters instead, reducing the

number of parameters while remaining expressive, and thus making the optimization problem easier.

The most common approach to reduce the number of parameters for a given receptive field size is to downsample the feature maps throughout the network. This is commonly achieved progressively by interleaving downsampling layers with convolutional layers. This way, the resolution of the feature maps gets lower and lower as we traverse the layers from input to output. For example, neurons after a sequence of two 3×3 convolutions in successive layers would normally have a receptive field of 5×5 pixels. However, we can extend it to 12×12 pixels with an accumulated downsampling of factor 4.

To downsample the feature maps, the most popular approach is to use the so-called *max pooling* layer [16]. A max pooling layer takes a group of neighbors in the feature map and condenses them into a single output by computing the maximum of all incoming activations in the window. The pooling windows in general do not overlap, hence the output map is downsampled (see Fig. 5.1). For instance, if pooling is performed in a 2×2 window, the feature map is reduced to half of its resolution.

Computing the maximum value is inspired by the idea of detecting objects from their parts. For example, in a face detector it is important to identify the constituents of a face, such as *hair* or *nose*, while the exact locations of these components should not be such a determinant factor. The max pooling layer conveys then to which extent there is evidence of the *existence* of a feature in a vicinity. Other less popular forms of downsampling include average pooling and applying convolutions with a *stride*, i.e., "skipping" some of them (e.g., applying every other convolution).

Pooling operations (and downsampling in general) hard-code robustness to spatial deformations, a virtue that boosted the success of CNNs for image categorization. However, spatial precision is lost when downsampling. The increased receptive field (and thus recognition capability) comes at the price of losing localization capability. This well-reported trade-off [93, 21] is a major concern for dense labeling.

We could still imagine a downsampling network that preserves localization: it would learn features of the type "a corner at the center of the receptive field", "a corner one pixel left of the center of the receptive field", "a corner two pixels left of the center of the receptive field", and so on, multiplying the number of features to be learned. This would however discredit the use of downsampling to gain robustness to spatial variation in the first place. The recognition/localization trade-off must thus be properly addressed to design a high-resolution semantic labeling network.

(a) (b)

Figure 5.2: To classify the central gray pixel of this patch (and not to confuse it, e.g., with an asphalt road), we need to take into account a spatial context (a). However, we do not need a high resolution everywhere in the patch. It can be lower as we go away from the central pixel and still identify the class (b).

5.2 Review of high-resolution classification CNNs

Fully convolutional networks (FCNs), as described in Chapter 3, have become the standard in pixelwise classification. They contain only convolutional layers, i.e., no fully connected layers. Therefore, they can be applied to images with various sizes: inputting a larger image patch produces a larger output, the convolutions being performed on more locations. When an FCN has downsampling layers, the output contains fewer elements than the input, since the resolution has been decreased. This gave birth to the so-called deconvolutional (or upconvolutional) layer, which upsamples a feature map by interpolating neighboring elements (as the last layer in Fig. 5.1). However, simply adding a deconvolutional layer to upsample the output on top of a network provides dense outputs but imprecise labeling results, because the upsampling is performed in a naive way from the coarse classification. This is dissatisfying in many applications, such as high-resolution aerial image classification, where the goal is to precisely identify and outline tiny objects such as cars. The open question is then which type of FCN would be able to conduct fine predictions that provide detailed high-resolution outputs, while still taking large amounts of context into account and without exploding the number of trainable parameters.

We now describe what we consider to be the elementary principle from which to derive efficient dense classification architectures. Let us then first observe that while our goal is to take large amounts of context into account, we do not need this context at the same spatial resolution everywhere. This has already been discussed in Section 3.3.3, but let us here show another example, this time on an aerial image. Suppose we want

to classify the central pixel of the patch in Fig. 5.2(a). Such a gray pixel, taken out of context, could be easily confused with an asphalt road. Considering the whole patch at once helps to infer that the pixel belongs indeed to a gray rooftop. However, two significant issues arise if we take a full-resolution large patch for context: a) it requires many computational resources that are actually not needed for an effective labeling, and b) it does not provide robustness to spatial variation (we might actually not care about the *exact* location of certain features to determine the class). Conducting predictions from low-resolution patches instead is not a solution as it produces inaccurate coarse classification maps. Nevertheless, it is actually not necessary to observe all surrounding pixels at full resolution: the farther we go from the pixel we want to label, the lower the requirement to know the exact location of the objects. For example, in the patch of Fig. 5.2(b) it is still possible to classify the central pixel, despite the outer pixels being blurry. Therefore, we argue that a combination of reasoning at different resolutions is necessary to conduct fine labeling, if we wish to take a large context into account in an efficient manner.

In the following, we analyze the main families of high-resolution classification networks that have been proposed in the past two years. For each of them we discuss the following aspects:

- How a solution to the fine-grained labeling problem is provided;
- Where this solution stands with respect to the principle of Fig. 5.2;
- General strengths and weaknesses, and computational efficiency.

5.2.1 Dilation Networks

Dilation networks are based on the shift-and-stitch approach or *à trous* algorithm [93]. This consists in conducting a prediction at different offsets to produce multiple low-resolution outputs, which are then interleaved to compose the final high-resolution result. For example, if the downsampling factor of a network is S, one should produce S^2 classification maps by shifting the input horizontally and vertically. Such an interleaving can also be implemented directly in the architecture, by using "dilated" operations [167], i.e., performing them on non-contiguous elements of the previous feature maps. This principle is illustrated in Fig. 5.3.

Dilations have been used with two purposes:

1. As an alternative to upsampling for generating full-resolution outputs [38, 93].

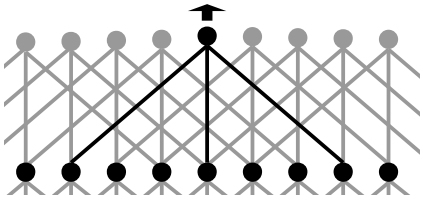


Figure 5.3: A dilated convolution (i.e., on non-adjacent inputs) with a dilation factor $S = 4$.

2. As a means to increase the receptive field [21, 167], by enlarging the area covered by a convolution kernel, without increasing the number of trainable parameters.

Regarding the first point, we must mention that there is no theoretical improvement compared to an FCN with naive upsampling, because the presence of pooling layers still reduces spatial precision. Executing the prediction multiple times at small offsets still keeps predictions spatially imprecise.

Regarding the second point, we must remark that while dilated convolutions increase the receptive field, this does not introduce robustness to spatial variation per se. For example, a network with only dilated convolution layers would have a large receptive field but would only be able to learn filters of the type "a building in the center, with a car *exactly* five pixels to the left". This robustness would have to be thus learned, hopefully, by using a larger number of filters.

The use of an interleaved architecture at training time, implemented with dilations, has been however reported to be beneficial. In the context of aerial image labeling, Sherrah [131] recently showed that it outperforms its FCN/upsampling counterpart[1]. The major improvement compared to the FCN/upsampling network was measured in the labeling capabilities of the *car* class, which is a minority class with tiny objects, difficult to recognize [155]. While the dilation strategy is not substantially different from an architectural point of view compared to naive upsampling, some advantages in training might explain the better results: In the upsampling case the network is encouraged to provide a coarse classification that, once upsampled, is close to the ground truth. In the dilation network, on the contrary, the interleaved outputs are directly compared to individual pixels in the ground truth, one by one. The latter seems to better avoid suboptimal solutions that absorb minority classes or tiny objects.

The computational time and memory required by dilation networks are significant, to the point that using GPUs might become impractical even with moderately large

[1]While such architecture is named a "no-downsampling" network in [131], a more appropriate name would probably be "no-upsampling", because there is indeed downsampling due to the max pooling layers.

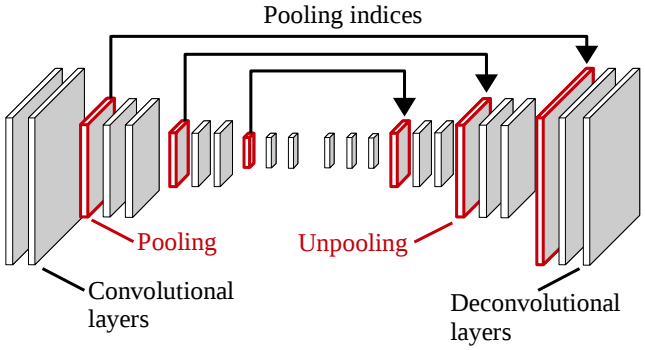


Figure 5.4: Deconvolution network. The CNN is "mirrored" to learn the deconvolution.

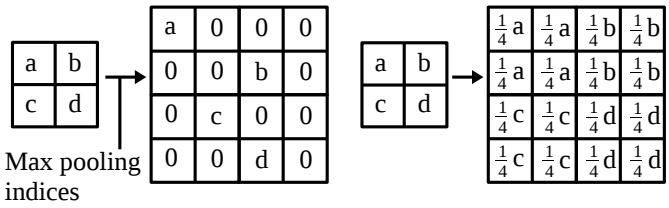


Figure 5.5: Max (left) and average (right) unpooling.

architectures. This is because the whole network rationale is applied to many contiguous locations.

Overall, while dilation networks have been reported to exhibit certain advantages, they are computationally demanding and do not particularly address the principle of Fig. 5.2.

5.2.2 Deconvolution Networks (unpooling)

Instead of naively upsampling the classification score maps with one deconvolutional layer, a more advanced approach is to attach a multi-layer network to learn a complex upsampling function. This idea was simultaneously presented by different research groups [8, 112] and later extended to different problems (e.g., [164]). A simple way to implement this idea is to "reflect" an existent FCN, with the same number of layers and kernel sizes, to perform the upsampling. The convolutional layers are reflected as deconvolutional layers, and the pooling layers as *unpooling* layers (see Fig. 5.4). While pooling condenses several activations into one representative value (typically, the maximum activation), unpooling layers must reconstruct the original size of activations. In the case of max unpooling, the location of the maximal activation is recalled from the corresponding pooling layer, and is used to place the activation back into its original pooled location.

The other elements in the unpooling window are set to zero, leading to sparse feature maps, as illustrated in Fig. 5.5. Unpooling was first introduced as part of a framework to analyze and visualize CNN features [169]. The arrows in Fig. 5.4 represent the communication of the pooling indices from the pooling layer to the unpooling layer. In the case of average pooling, the corresponding unpooling layer simply outputs at every location the input activation value divided by the number of elements in the target unpooling window(see Fig. 5.5). In this case, there is no need to transmit a location from pooling to unpooling.

This concept can be thought of as an "encoder–decoder", where the middle layer is seen as a common representation to images and classification maps, while the "encoder" and "decoder" ensure the translation between this representation and the two modalities. When converting an FCN to a deconvolution network, the final classification layer of the FCN is usually dropped before reflecting the architecture. This way the interface between the encoder and the decoder is a rich representation with multiple features. The first layer of the encoder takes as input as many channels as the ones in the input image, and the last layer of the decoder produces as many feature maps as the number of classes required. In [8, 9], alternatively, the network outputs a larger set of features that are then classified with additional layers.

While pooling is used to add robustness to spatial deformation, the fact of "remembering" the location of the max activation helps to precisely locate objects in the deconvolution steps. For example, the exact location of a road might be irrelevant to do any higher-level reasoning later on, but once the network decides to label the road as a semantic object we need to recover the location information to outline it with high precision. This illustrates how deconvolution networks balance the localization/recognition trade-off.

Note however that if one happens *not* to choose max pooling for downsampling, then the unpooling scheme is not able to recover *per se* the lost spatial resolution. There is no memory about the location of the higher-resolution feature. Even though max pooling is very common, it has been shown that average or other types of pooling might be more effective in certain applications [16]. In fact, recent results [140] suggested that max pooling can be emulated with a strided convolution and achieve similar performance. The deconvolution network idea is however leveraged when max pooling is chosen for downsampling.

This certainly does not mean that a deconvolutional network is incapable of learning without max pooling layers. Convolution/deconvolution architectures without max pooling have been successfully used in different domains [155, 134]. For example, a recent

submission to the ISPRS Semantic Labeling Challenge [155] is such type of network. The recognition/localization trade-off is not really alleviated in this case: the encoder should encode features of the type "an object boundary 5 (or 7, 10...) pixels away to the left", so that the decoder can really leverage this information and reconstruct a high-resolution classification map.

The depth of deconvolution networks is significantly larger, roughly twice the one of the associated FCN. This often implies a slower and more difficult optimization, due to the increase in the number of trainable parameters introduced by deconvolutional layers. While the decoding part of the network can be simplified [117], this adds arbitrariness to the design, since instead of just reflecting the encoder we must also design the decoder.

To conclude, the deconvolution scheme does address the recognition/localization trade-off, but only in the case where max pooling is used for downsampling. The increased network depth can be a concern for an effective training.

5.2.3 Skip Networks

In the original paper about fully convolutional networks, Long et al. [93] proposed the so-called "skip" architecture to generate high-resolution classification outputs. The idea is to build the final classification map by combining multiple classification maps, obtained from intermediate features of the network at different resolutions (and not just the last one).

The last layer of an FCN outputs as many feature maps as classes, which are interpreted as score or "heat" maps for every class. Intermediate layers, however, tend to have many more features than the number of classes. Therefore, skip networks add extra layers that convert the arbitrarily large number of features of intermediate layers into the desired number of heat maps. This approach allows us to extract multiple score maps for each class from a single network, at different resolutions. The lower-level score maps are fine but have a small receptive field, while the higher-level ones can see farther but with less detail. As a result, we have a pool of score maps.

The score maps are then combined pairwise, from the lower scales to the higher scales. At every step, the lower-resolution score maps are upsampled to match the higher-resolution ones. They are then added elementwise. This is repeated until all intermediate maps are processed. The overall combination of resolutions forms a directed acyclic graph, with links that "skip" ahead from lower to higher layers. A skip network is illustrated in Fig. 5.6.

Skip networks address the trade-off between localization and precision quite explicitly: the information at different resolutions is extracted and combined. The original paper

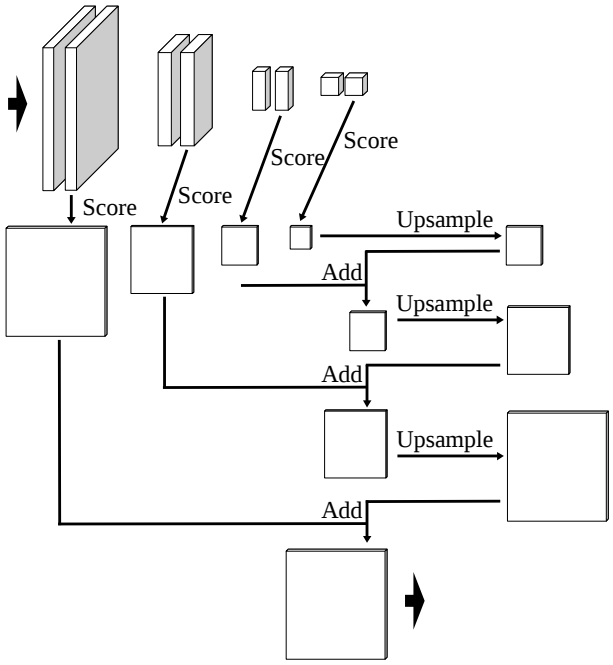


Figure 5.6: Skip network: multiple classification scores are obtained from intermediate CNN features at different resolutions, and are combined by element-wise adding and upsampling.

introduces this methodology as "combining what and where". This approach is closer to the principle described in Fig. 5.2 than the previous approaches reviewed above. The skip network mixes observations at different resolutions, without unnecessarily increasing the depth or width of the architecture (as in deconvolution and dilation networks respectively) and it does not impose a particular type of downsampling (as in deconvolution networks).

While the idea of extracting different resolutions is certainly very relevant, the skip model seems to be inflexible and arbitrary in how to combine them. First of all, it combines classification verdicts, instead of a rich set of features, coming from each of the resolutions. For example, it combines how a layer evaluates that an object is a building by using low-level information, with how another layer evaluates whether the same object is a building by using higher-level information. Let us recall that we use deep multi-layer schemes with downsampling because we actually consider that certain objects can only be detected at the upper layers of the network, when a large amount of context has been taken into account and at a high level of abstraction. It seems thus contradictory to try to refine the boundaries of an object detected at a high level, by using a classification conducted at a lower level, where the object might not be detected at all. Moreover, the element-wise addition restricts the combination of resolutions to be simply a linear combination. The skip links to combine resolutions are in fact parameterless (besides the addition of the scoring layers). We could certainly imagine classes that require a more complex nonlinear combination of high- and low-level information to be effectively classified.

It is worth noting that the creation of the intermediate score maps has also been referred to as a dimensionality reduction step [9]. It is however not by chance that the amount of reduced features coincides with the amount of classes: even though it is technically a dimensionality reduction, its spirit is to create a partial classification, not just to reduce the number of features. This is confirmed by the name of these layers in the public implementation by Long et al. [93]: "score" layers. Moreover, if this operation were indeed intended to be just a reduction of dimensionality, we could imagine outputting different amounts of feature maps from different resolutions. However, in that case there would be no way of adding them element by element as suggested. Hariharan et al. [63] also extract intermediate features ("skip" links) from a CNN. Additional convolutional layers are used to derive a fixed number of features from each of these intermediate layers, and then added to give the final output.

To conclude, the skip network architecture provides an efficient solution to address the localization/recognition trade-off, yet this could be done in a more flexible way that enables a more complex combination of the features.

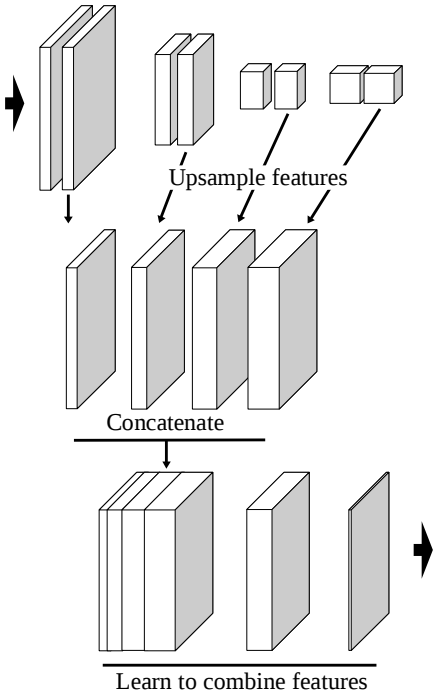


Figure 5.7: MLP network: intermediate CNN features are concatenated, to create a pool of features. Another network learns how to combine them to produce the final classification.

5.3 Proposed method: learning to combine resolutions

In this section we propose an alternative scheme for high-resolution labeling, derived as a natural consequence of our observations about the other families of methods. In particular, this architecture leverages the benefits of the skip network described in Section 5.2.3 while addressing its potential limitations.

Taking multiple intermediate features at different resolutions and combining them seems to be a sensible approach to specifically address the localization/recognition trade-off, as done with skip networks. In such a scheme, the high-resolution features have a small receptive field, while the low-resolution ones have a wider receptive field. Combining them constitutes indeed an efficient use of resources, since we do not actually *need* the high-resolution filters to have a wide receptive field, following the principle of Fig. 5.2.

The skip network combines *predictions* derived from the different resolutions, i.e., score maps for each of the classes. For example, we try to refine the "blobby" building outputted by the coarse classifier, via a higher-resolution classification. However, it is unclear how effectively the higher-resolution classifier detects such building, considering

its reduced receptive field and shallow reasoning.

We thus argue that a more relevant way of performing fine semantic labeling is to combine features, not classification maps. For example, to refine the boundaries of a coarse building, we would use high-resolution edge detectors and not high-resolution building detectors.

In our proposed scheme, intermediate features are extracted from the network and treated equally, creating a pool of features that emanate from different resolutions. A neural network then learns how to combine these features to give the final classification verdict. This adds flexibility to learn more complex relations between the different resolutions and generalizes the element-wise addition of the skip architecture.

The overall process is depicted in Fig. 5.7. First, a subset of intermediate features are extracted from the network. These are naively upsampled to match the resolution of the highest-resolution features. They are then concatenated to create the pool of features. Notice that while the spatial dimensions of the feature maps are all the same, they originally come from different resolutions. This way, the variation of the feature responses across space will be smoother in certain maps and sharper in others. Note that while it is practical to store in memory the upsampled responses, this is not intrinsically necessary. For example, we could imagine a system that answers to a high-resolution query by outputting the nearest neighbor in the coarser map or by interpolating neighboring values on the fly.

From the pool of features, a neural network predicts the final classification map (we could certainly use other classifiers, but this lets us train the system end to end). We assume that all the spatial reasoning has been conveyed in the features computed by the initial CNN. This is why we operate on a pixel-by-pixel basis to combine the features. Any need to look at neighbors should be expressed in the spatial filters of the CNN. This way we conceptually and architecturally separate the extraction of spatial features from their combination.

We can think of the multi-layer perceptron (MLP) with one hidden layer and a non-linear activation function as a minimal system to learn how to combine the pool of features. Such MLPs can learn to approximate any function and, since we do not have any particular constraints, it seems an appropriate choice. In practice, this is implemented as a succession of convolutional layers with 1×1 kernels, since we want the same MLP to be applied at every location. By introducing the MLP and executing it at a fine resolution, we must expect an overhead in processing time compared to the skip network.

The proposed technique is intended to learn how to combine information at different resolutions, not how to upsample a low-resolution classification. An example of the type

of relation conveyed by this scheme is as follows: "label a pixel as *building* if it is red and belongs to a larger red rectangular structure, which is surrounded by areas of green vegetation and near a road".

Finally, let us discuss the CNN from which features are extracted (the topmost part of Fig. 5.7). The different features are extracted from intermediate layers of a single CNN. This assumes that the higher-level features can be derived from the lower-level ones. It is basically a part-based model [45], where we consider that an object can be detected by its parts, and we are using those same parts as the higher-resolution features inputted to the MLP. This seems to be a sensible assumption, yet we must mention that we could eventually think of separate networks to detect features at different resolutions instead of extracting intermediate representations of a single network (as, e.g., in [42]). While we adopt the model of Fig. 5.7 in this work, the alternative could be also considered. It would be certainly interesting to study to which extent it is redundant to learn the features in separate networks and, conversely, how results could be eventually improved by doing it.

5.4 Experiments on Potsdam and Vaihingen datasets

We evaluate the aforementioned architectures on two benchmarks of aerial image labeling: Vaihingen and Potsdam, provided by Commission III of the ISPRS [70]. The Vaihingen dataset is composed of 33 image tiles (of average size 2494×2064), out of which 16 are fully annotated with class labels. The spatial resolution is 9 cm. Near infrared (NIR), red (R) and green (G) bands are provided, as well as a digital surface model (DSM), normalized and distributed by [52]. We select 5 images for validation (IDs: 11, 15, 28, 30, 34) and the remaining 11 images for training, following [131, 155, 114].

Potsdam dataset consists of 38 tiles of size 6000×6000 at a spatial resolution of 5 cm, out of which 24 are annotated. It provides an additional blue channel (which we here exclude for simplicity) and the normalized DSM. We select the same 7 validation tiles as in [131] (IDs: 2_11, 2_12, 4_10, 5_11, 6_7, 7_8 7_10) and the remaining 17 tiles for training. Both datasets are labeled into the following six classes: *impervious surface*, *building*, *low vegetation*, *tree*, *car* and *clutter/background*.

In order to account for labeling mistakes, another version of the ground truth with eroded boundaries is provided, on which accuracy is measured. To evaluate the overall performance, overall accuracy is used, i.e., the percentage of correctly classified pixels. To evaluate class-specific performance, the F1-score is used, computed as the harmonic mean between precision and recall [32]. We also include the mean F1 measure among

Table 5.1: Architecture of our base FCN.

Layer	Filter size	Number of filters	Stride	Padding
Conv-1_1	5	32	2	2
Conv-1_2	3	32	1	1
Pool-1	2		2	
Conv-2_1	3	64	1	1
Conv-2_2	3	64	1	1
Pool-2	2		2	
Conv-3_1	3	96	1	1
Conv-3_2	3	96	1	1
Pool-3	2		2	
Conv-4_1	3	128	1	1
Conv-4_2	3	128	1	1
Pool_4	2		2	
Conv-Score	1	5	1	

classes, since overall accuracy tends to be too insensitive to minority classes in imbalanced datasets.

5.4.1 Network architectures

To conduct our experiments we depart from a base fully convolutional network (FCN) and derive other architectures from it. Table 5.1 summarizes our base FCN for the Vaihingen dataset. The architecture is borrowed from [73], except for the fact that we increased the size of the filters from 3 to 5 in the first layer, since it is a common practice to use larger filters if there is a stride. Every convolutional layer (except the last one) is followed by a batch normalization layer [69] and a ReLU activation. We did not optimize the architecture of the base FCN. Padding refers to the amount of zero-valued pixels added around the input to a convolutional layer, so as to preserve the dimension of the feature maps (otherwise the convolution is not applied near the border and the maps become slightly smaller).

The total downsampling factor is 16, out of which 8 is the result of the max pooling layers and 2 of the stride in the first layer. The conversion of the last set of features to classification maps (the "score" layer) is performed by a 1×1 convolution. To produce a dense pixel labeling we must add a deconvolutional layer to upsample the predictions by a factor of 16, thus bringing them back to the original resolution.

To implement a *skip* network, we extract the features of layers *Conv-*_2*, i.e., produced by the last convolution in each resolution and before max pooling. Additional

scoring layers are added to produce classification maps from the intermediate features. The resulting score maps are then combined as explained in Section 5.2.3. Our MLP network was implemented by extracting the same set of features. As no intermediate scores are needed, we remove layer 'Conv-Score' from the base FCN. The features are combined as explained in Section 5.3. The added multi-layer perceptron contains one hidden layer with 256 neurons.

We also created a deconvolution network that exactly reflects the base FCN (as in [112]). This is straightforward, with deconvolutional and unpooling layers associated to every convolutional and pooling layer. The only particularity is that the last layer outputs as many maps as required classes and not as input channels. We here call it *unpooling* network, to differentiate it in the experiments from another method that uses a stack of deconvolutions but without unpooling [155], which we simply refer to as *deconvolution* network. To cover the last family of architectures of Sec. 5.2, the *dilation* network, we incorporate the results recently presented by Sherrah [131].

In both datasets we use the same four input channels: DSM, NIR, R and G. Notice that we simply add the DSM as an extra band. In the case of Vaihingen we predict five classes, ignoring the clutter class, due to the lack of training data for that class. In the case of Potsdam we predict all six classes.

Considering the difference in resolution in both datasets, in the case of Potsdam we downsample the input and linearly upsample the output by a factor of 2 (following [131]). We use the same architecture as for Vaihingen (besides the different number of output classes) between the downsampling and upsampling layers. This is to cover a roughly similar receptive field in terms of meters (and not pixels) for both datasets.

5.4.2 Training

The networks are trained by stochastic gradient descent [11]. In every iteration a group of patches is fed to the network for backpropagation. We sample random patches from the images, performing random flips (vertically, horizontally or both) and transpositions, augmenting the data 8 times. At every iteration we group five patches in the mini-batch, of size 256×256 for Vaihingen dataset and 512×512 for Potsdam (to roughly cover the same geographical area, considering the difference in resolution). In all cases, gradient descent is run with a momentum of 0.9, and an L2 penalty on the network's parameters of 0.0005. Weights are initialized following [64] and, since we use batch normalization layers before ReLUs, there is no need to normalize the input channels.

We start from a base learning rate of 0.1 and anneal it with an exponential decay.

Table 5.2: Numerical evaluation of architectures derived from our base FCN on the Vaihingen validation set.

	Imp. surf.	Building	Low veg.	Tree	Car	Mean F1	Overall acc.
Base FCN	91.46	94.88	79.19	87.89	72.25	85.14	88.61
Unpooling	91.17	95.16	79.06	87.78	69.49	84.54	88.55
Skip	91.66	95.02	79.13	88.11	77.96	86.38	88.80
MLP	**91.69**	**95.24**	**79.44**	**88.12**	**78.42**	**86.58**	**88.92**

Table 5.3: Numerical evaluation of architectures derived from our base FCN on the Potsdam validation set.

	Imp. surf.	Building	Low veg.	Tree	Car	Clutter	Mean F1	Acc.
Base FCN	88.33	93.97	84.11	80.30	86.13	75.35	84.70	86.20
Unpooling	87.00	92.86	82.93	78.04	84.85	72.47	83.03	84.67
Skip	89.27	94.21	84.73	**81.23**	93.47	75.18	86.35	86.89
MLP	**89.31**	**94.37**	**84.83**	81.10	**93.56**	**76.54**	**86.62**	**87.02**

The decay rate is set so that the learning rate is divided by ten every 10,000 iterations in the case of Vaihingen and every 20,000 iterations in Potsdam. We decrease the learning rate more slowly in the case of Potsdam because the total surface covered by the dataset is larger, thus we assume it must take longer to explore. Training is stopped after 45,000 iterations in the first dataset and 90,000 in the second one, when the error stagnates on the validation set.

To train the unpooling, skip and MLP networks we initialize the weights with the pretrained base FCN, and jointly retrain the entire architecture. We start this second training phase with a learning rate of 0.01, and stop after 30,000 and 65,000 iterations for Vaihingen and Potsdam datasets respectively. We verified that the initialization with the pretrained weights is indeed beneficial compared to training from scratch.

5.4.3 Numerical results

In this section we first present how our base FCN network compares to its derived architectures: unpooling, skip and MLP. We then position MLP with respect to other results reported in the literature, including a dilation network, thus completing the evaluation over all four families of techniques. We finally discuss our submission to the ISPRS contest.

Comparison of a base FCN to its derived unpooling, skip and MLP networks
The classification performances on the validation sets are included in Tables 5.2 and 5.3, for Vaihingen and Potsdam datasets, respectively. The MLP network exhibits the best

performance in almost every case. The skip network effectively enhances the results compared with the base network, yet it does not outperform MLP. Let us remark that the unpooling strategy does not necessarily improve the base FCN. This might be a result of the increased training difficulty due to the depth of the network and the sparsity of the unpooled maps. Our attempts to modify the training scheme did not conduce to improving its performance.

Overall, the numerical results show that the incorporation of lower-resolution features significantly improves the classification accuracy. MLP is the most competitive method, boosting the performance by learning how to combine these features.

Comparison with other methods Tables 5.4 and 5.5 (for Vaihingen and Potsdam datasets respectively) incorporate the numerical results reported by other authors using the same training and validation sets. Since not every method was applied to both datasets, the tables do not display exactly the same techniques. The MLP approach also outperforms the dilation strategy, in both datasets, thus positioning it as the most competitive category among those presented in Sections 5.2, 5.3 (dilation, unpooling, skip, MLP).

In the case of Vaihingen dataset, we also report the results of the *deconvolution* network [155], commented in Sec. 5.2.2, which performs upsampling by using a series of deconvolutional layers. Contrary to the *unpooling* network, the decoder does not exactly reflect the encoder and no unpooling operations are used. Additionally, we include the performance of other methods recently presented in the literature: the CNN+RF approach [114], which combines a CNN with a random forest classifier; the CNN+RF+CRF approch, which adds CRF post-processing to CNN+RF; and Dilation+CRF [131], which adds CRF post-processing to the dilation network. As depicted in the table, the MLP approach outperforms these other methods too.

For Potsdam dataset, Table 5.5 reports the performance of two other methods, presented in [131]. In both cases, a pretrained network based on VGG [135] is applied to the IR-R-G channels of the image, and another FCN is applied to the DSM, resulting in a huge hybrid architecture. An ordinary version (with upsampling at the end) and a *dilation* version are considered ('VGG pretr.' and 'VGG+Dilation' in Table 5.5, respectively). In the latter version, the dilation strategy could only be applied partially as it is too memory intensive. While MLP outperforms the non-pretrained simpler dilation network, the *VGG+Dilation* variants exhibits the best overall performance (though not on all of the individual classes). This suggests that the VGG component might be adding a competitive edge, though the authors stated that this is not the case on the Vaihingen

Table 5.4: Comparison of MLP with other methods on the Vaihingen validation set.

	Imp. surf.	Build.	Low veg.	Tree	Car	F1	Acc.
CNN+RF [114]	88.58	94.23	76.58	86.29	67.58	82.65	86.52
CNN+RF+CRF [114]	89.10	94.30	77.36	86.25	71.91	83.78	86.89
Deconvolution [155]						83.58	87.83
Dilation [131]	90.19	94.49	77.69	87.24	76.77	85.28	87.70
Dilation + CRF [131]	90.41	94.73	78.25	87.25	75.57	85.24	87.90
MLP	**91.69**	**95.24**	**79.44**	**88.12**	**78.42**	**86.58**	**88.92**

Table 5.5: Comparison of MLP with other methods on the Potsdam validation set.

	Imp. surf.	Build.	Low veg.	Tree	Car	Clutter	F1	Acc.
Dilation [131]	86.52	90.78	83.01	78.41	90.42	68.67	82.94	84.14
VGG pretr. [131]	89.84	93.80	85.43	83.61	88.00	74.48	85.86	87.42
VGG+Dilation [131]	**89.95**	93.73	**85.91**	**83.86**	**94.31**	74.62	**87.06**	**87.69**
MLP	89.31	**94.37**	84.83	81.10	93.56	**76.54**	86.62	87.02

dataset.

Overall, MLP provides better accuracies than most techniques presented in the literature, including dilation networks, ensemble approaches and CRF post-processing.

Submission to the ISPRS challenge We submitted the result of executing MLP on the Vaihingen test set to the ISPRS server (ID: 'INR'), which can be accessed online [70]. Our method scored second out of 29 methods, with an overall accuracy of 89.5%. Note that our MLP technique is very simple compared to other methods in the leaderboard, yet it scored better than them. For example, an ensemble of two *skip* CNNs was pretrained on large natural image databases [101], with over 20 convolutional layers and separate paths for the image and the DSM. Despite being simpler, our MLP network outperforms it in the benchmark.

5.4.4 Visual results

We include visual comparisons on closeups of classified images of both datasets in Fig. 5.8. As expected, the base FCN tends to output "blobby" objects, while the other methods provide sharper results. This is particularly noticeable for the cars of Rows 2, 5 and 6, and for the thin road at the lower left corner of Row 4. We also observe that the incorporation of reasoning at lower resolutions allows the derived networks to discover small objects that are otherwise lost. This is particularly noticeable in the 4th row, where there is a set of small round/cross-shaped objects of the *clutter* class (in red) that are omitted or grouped together by the base FCN.

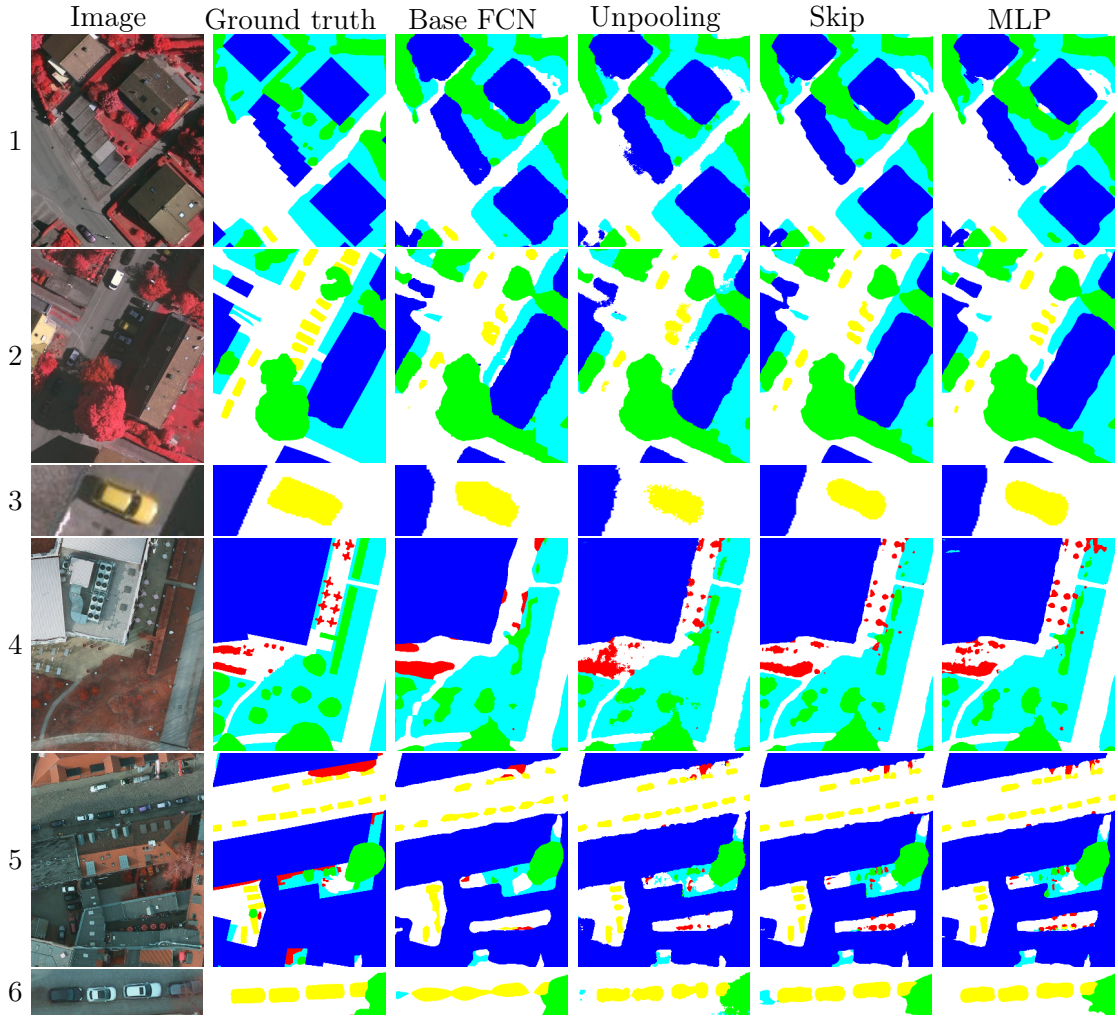


Figure 5.8: Classification of closeups of Vahingen (1–3) and Potsdam (4–6) validation sets. Classes: Impervious surface (white), Building (blue), Low veget. (cyan), Tree (green), Car (yellow), Clutter (red).

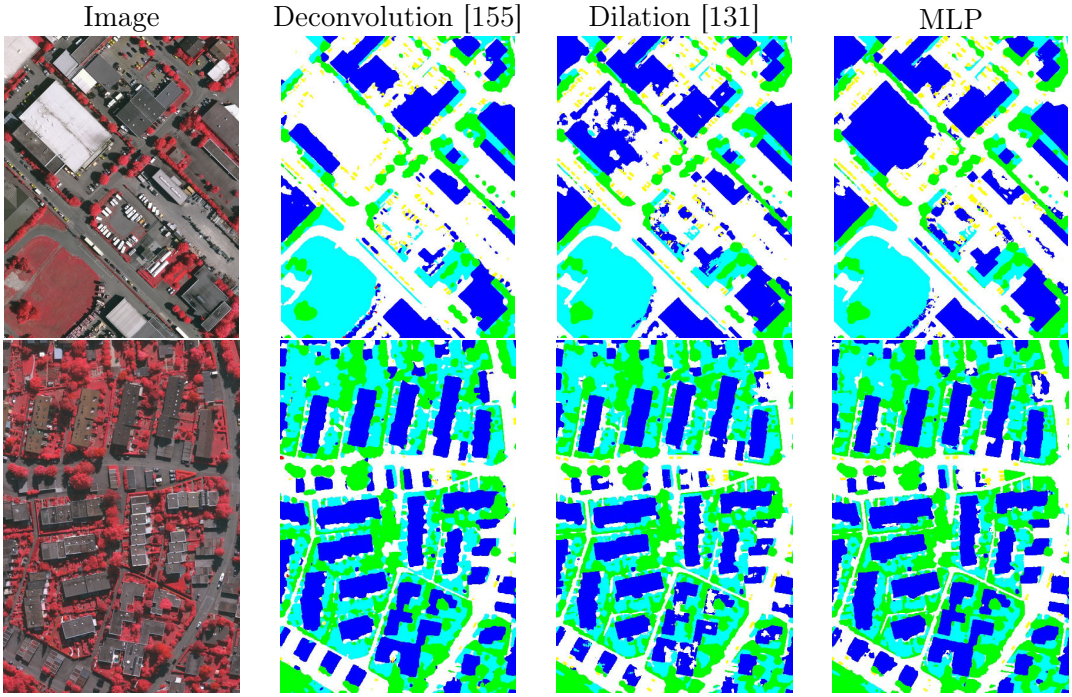


Figure 5.9: Classification of entire tiles of the Vaihingen test set.

The unpooling technique seems to be prone to outputting artifacts. These are often very small in size, even isolated pixels. This is well observed for example for the car of Row 3. This effect could be a natural consequence of the max unpooling mechanism, as depicted in Fig. 5.5, which upsamples into sparse matrices and delegates the task of reconstructing a smoother output to the deconvolutional layers.

At first sight it is more challenging to visually assess why MLP outperforms the skip network in almost every case in the numerical evaluation. Taking a closer look we can however observe that boundaries tend to be more accurate at a fine level in the case of MLP. For example, the "staircase" shape of one of the buildings in Row 1 is noticeably better outlined by the MLP network.

We can also observe that the ground truth itself is often not very precise. For example, the car in Row 3 does not seem to be labeled accurately, hence it is difficult to imagine that a network would learn to finely label that class. In Row 5, an entire lightwell between buildings has apparently been omitted in the ground truth (labeled as part of the building), yet recognized as an impervious surface by the CNNs.

The general recognition capabilities of CNNs can also be well appreciated in these fragments. For example, in Row 4, while there are tiny round objects both on the roof

Table 5.6: Execution times.

	Train [s]		Test [s/ha]	
	Vaih.	Pots.	Vaih.	Pots.
Base FCN	3.9	9.8	0.81	1.44
Unpooling	8.4	21.0	1.38	1.84
Skip	6.6	16.9	0.81	1.48
MLP	10.0	24.5	1.70	2.0
Dilation*	62	400	4.81	17.2

*As reported in [131] (see details in Sec. 5.4.5.)

of the building and outside the building, CNNs correctly label as *building* the ones on the roof and as *clutter* the other ones.

In Fig. 5.9 we show the classification of entire tiles of the Vaihingen set, obtained from the test set submissions. We include the *deconvolution* [155] and *dilation* [131] network results, together with our MLP. We can see, for instance, that a large white building in the first image is recognized by MLP but misclassified or only partially recovered by the other methods. In the second tile, the Dilation method outputs some holes in the buildings which are not present in the MLP results. A better combination of the information coming from different resolutions might explain why MLP successfully recognizes that these entire surfaces do belong to the same object.

5.4.5 Running times

Table 5.6 reports the running times for training and testing on both datasets. The training time of the architectures derived from the base FCN comprises the time to pretrain the base FCN first and the time to then train the whole system altogether (see details in Sec. 5.4.2). The architectures were implemented using Caffe [71] and run on an Intel I7 CPU @ 2.7Ghz with a Quadro K3100M GPU (4 GB RAM). We also add for comparison the results reported by the author of the Dilation network [131], run on a larger 12 GB RAM GPU. To classify large images we crop them into tiles with as much overlap as the amount of padding in the network, to avoid tile border effects.

As reported in the table, the unpooling, skip and MLP networks introduce an overhead to the base FCN. MLP is the slowest of the derived networks, followed by the unpooling and skip networks. MLP, which provides the highest accuracy, classifies the entire Vaihingen validation set in about 30 seconds and the Postdam validation set in 2 minutes. This is substantially faster than the dilation network. Incorporating the principle of Fig. 5.2 allows us to better allocate computational resources, not spending too much time and space in conducting a high-resolution analysis where it is not needed, boosting accuracy and performance.

5.5 Can classification methods generalize to any city?

Over the last few years, there has been a growing interest in processing remote sensing imagery at a large scale, often the entire earth at once [102]. New perspectives in remote sensing have particularly highlighted this interest, such as the use of aerial imagery for autonomous driving [102]. The improvements in the algorithms, and the use of clusters and GPUs have made the processing time less of a constraint. One of the current challenges is to design methods that generalize to different areas of the earth, considering the important intra-class variability encountered over large geographic extents.

The standard way of evaluating and comparing classification methods is to split the labeled data into two sets: one used for training and the other one for testing. For example, in the hyperspectral literature it is particularly common to randomly extract certain pixels from the labeled data and use them for training (ranging from as little as 50 pixels [43] to as much as 20% of all the labeled data [148]), while the rest is used for testing. The Pavia and Indian pines datasets [43] have become the standard benchmarks in the hyperspectral literature. They are mostly geared at distinguishing materials (e.g., *bitumen building* and *bricks*), thus leveraging the properties of hyperspectral imagery. However, those images cover limited geographic areas and the evaluation procedure does not assess how the methods generalize to different contexts or more abstract semantic classes.

With the goal of comparing classification methods over large areas, Mnih [107] created building and road classification datasets over Massachusetts, covering 340 km^2 and 2600 km^2 respectively. For testing, several randomly selected tiles were removed from the reference data. The training set thus covers a geographic surface with "holes", which are used for testing. This situation is analogous to the procedure used for the aforementioned hyperspectral datasets, though taken to a larger scale. While the Massachusetts datasets indeed cover a large surface with significant intra-class variability, the image tiles tend to be self-similar and with uniform color histograms. As shown in [107], a CNN trained on the Massachusetts dataset generalizes poorly to images over Buffalo, and a fine-tuning of the CNN to the new dataset is required.

In the context of high-resolution image classification, the Vaihingen and Potsdam datasets [155] have gained increasing attention over the last year, as discussed previously in this chapter. While they provide exhaustive reference data with multiple object classes, the area covered is limited (roughly 1.5 km^2 and 3.5 km^2 respectively). The Bavaria and Aerial KITTI datasets [102], used for road labeling, also cover small surfaces (5 km^2 and 6 km^2, respectively).

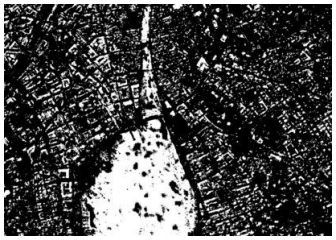


Figure 5.10: A CNN trained on a different dataset misclassifies most of Lake Zurich as a building.

In our experience, and in accordance to [107], training a classifier with images over a particular region and illumination conditions tends to generalize poorly to other images. For example, Fig. 5.10 depicts a classification map over Zurich into the *building/not building* classes, created by using one the CNNs presented in Chapter 3 and trained over Forez, France. We can observe Lake Zurich being mostly classified as *building*. Even though there were buildings and body waters in the French imagery, the CNN seems to have learned what a building looks like in that particular images and not simply what a building looks like.

Our goal is to provide a common framework to evaluate classification techniques and, in particular, their generalization capabilities. We created a benchmark database of labeled imagery that covers varied urban landscapes, ranging from highly dense metropolitan financial districts to alpine resorts. The data, referred to as the *Inria Aerial Image Labeling Dataset*[2], includes urban settlements over the United States and Austria, and is labeled into *building* and *not building* classes. Contrary to all previous datasets, the training and test sets are split by city instead of excluding random pixels or tiles. This way, a system trained, for example, on Chicago, is expected to classify imagery over San Francisco (with a significantly different appearance). The test set reference data is not publicly released, and a contest has been launched for researchers to submit their results.

In the following sections we first describe the dataset and then assess the performance of the networks presented previously in this chapter.

5.5.1 The dataset

One of the first key points to decide when creating the dataset was which geographic areas to include and which semantic classes to consider. The criteria were as follows:

- Recent orthorectified imagery available;

[2]project.inria.fr/aerialimagelabeling

- Recent official cadastral records available;
- Precise registration between the cadastral records and the orthorectified imagery;
- Open-access data, both for the images and the cadaster (free to access and distribute);
- Cover varied urban landscapes and illumination.

Let us first highlight the fact that we can only focus on regions where both the images and the reference data are available. In addition, we require the data to be open access in order to freely share our derived dataset with the community. After extensive research, we found that certain US and Austrian areas satisfy those requirements. In the case of the US, public domain orthoimages have been released by USGS through the National Map service (nationalmap.gov) in most urban areas of the country. Vectorial cadastral records have been released through certain local or statewide geographic information system (GIS) websites. We must focus on the zones where such reference data are available in addition to the images.

In the case of Austria, the different provinces have shared images through their respective GIS agencies. We focus, in particular, on Tyrol and Vienna provinces, since open vectorial cadastral data are also on hand. We obtained the images through the WMS services provided by the GIS departments[3] as well as the associated reference shapefiles.

The original US imagery is provided at either 15 or 30 cm resolution with three or four spectral bands (RGB/RGB-Infrared), depending on the area, and Vienna imagery contains three bands (RGB) at a resolution of 10 or 20 cm. We took out the common factor and built our dataset with 30 cm images (average resampling if needed) and using the three color bands.

We consider two semantic classes: *building* and *not building*. For this we must extract the so-called *building footprints* from the cadaster. While there are other classes present in some areas (e.g., trees and roads), the building class is the only one that is consistent across different areas. Roads, for example, are often represented with a line, but it is very often not located at the center of the road and its width is usually not specified. This makes it difficult to derive a pixelwise semantic labeling for roads, and is an active research problem itself [102].

Once we selected a number of candidate areas for the dataset, we visually inspected them to assess whether the cadaster is properly aligned with the images. In some regions, there are irregular shifts that led us to exclude them (e.g., Seattle and Spokane cities).

[3] https://gis.tirol.gv.at/arcgis/services/Service_Public/orthofoto/MapServer/WMSServer; http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml

Train	Tiles*	Total area	**Test**	Tiles*	Total area
Austin, TX	36	81 km^2	Bellingham, WA	36	81 km^2
Chicago, IL	36	81 km^2	San Francisco, CA	36	81 km^2
Kitsap County, WA	36	81 km^2	Bloomington, IN	36	81 km^2
Vienna, Austria	36	81 km^2	Innsbruck, Austria	36	81 km^2
West Tyrol, Austria	36	81 km^2	East Tyrol, Austria	36	81 km^2
Total	180	405 km^2	Total	180	405 km^2

Table 5.7: Dataset statistics. *Tile size: 1500^2 px. (0.3 m resolution).

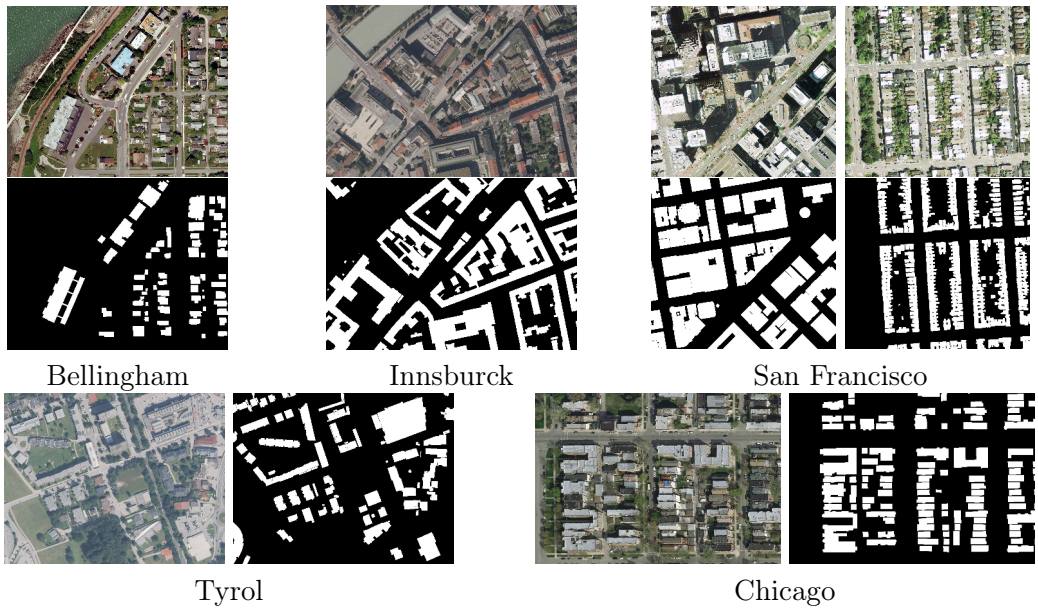


Figure 5.11: Close-ups of the dataset images and their corresponding reference data.

This may be the result of errors or imprecision in the terrain model used to orthorectify the images, or in the digitization of the cadaster. Note that we have only considered official image and cadaster data sources, ignoring, e.g., OpenStreetMap (OSM) data. Curiously enough, while we find the official San Francisco building footprints to be perfectly aligned with the USGS imagery, a team of OSM collaborators manually modified 150,000 buildings from these footprints prior to their inclusion in OSM[4], arguing that they were inaccurate. We now observe them to be misaligned with the imagery. A possible explanation for this is that, at the time of the edit, the Bing images used as the base layer of the OSM editor may have not been geographically precise.

The regions included in the dataset and their distribution into training and test subsets is depicted in Table 5.7. Note first that the amount of data in each of the

[4]https://www.mapbox.com/blog/status-san-francisco-complete/

Table 5.8: Numerical evaluation on small validation set.

		Austin	Chicago	Kitsap Co.	West Tyrol	Vienna	Overall
FCN	IoU	47.66	53.62	33.70	46.86	60.60	53.82
	Acc.	92.22	88.59	98.58	95.83	88.72	92.79
Skip	IoU	57.87	61.13	46.43	54.91	70.51	62.97
	Acc.	93.85	90.54	98.84	96.47	91.48	94.24
MLP	IoU	61.20	61.30	51.50	57.95	72.13	64.67
	Acc.	94.20	90.43	98.92	96.66	91.87	94.42

subsets is the same. This stresses our goal of properly assessing classification methods that generalize to different areas and images. The regions were split in such a way that each of the subsets contains both European and American landscapes, as well as high-density (e.g., Chicago/San Francisco and Vienna/Innsbruck) and low-density (e.g., Kistap/Bloomington, West/East Tyrol) urban settlements. While aerial images over Tyrol are present in both subsets, they have been obtained at different flights over the country, thus exhibiting different illumination characteristics. We have also selected dissimilar images inside some of the groups (e.g., Kitsap County contains tiles from two different flights with very dissimilar characteristics). The reference data was created by rasterizing the shapefiles with GDAL. Fig. 5.11 shows closeups of the images in the dataset.

We consider two evaluation measures to assess the performance of different methods on the dataset: the accuracy and the intersection over union (IoU) of the positive (*building*) class. We compute accuracy and IoU on the overall dataset and for every region independently (e.g., San Francisco).

5.5.2 Experiments

We experimented with convolutional neural networks on the dataset. We created a validation set by excluding the first five tiles of each area from the training set (e.g., Austin{1-5}). We first trained the base fully convolutional network (FCN) proposed from Potsdam dataset in Section 5.4, for 120,000 iterations on randomly sampled patches of our dataset (momentum is set to 0.9, the L2 penalty to 0.0005 and the learning rate to 0.001). To provide a finer classification, we derived an MLP network on top of the base FCN, as explained in Section 5.3 and illustrated in Fig. 5.7. The pretrained FCN was used to initialize the corresponding parameters in the MLP network, and then the overall system was trained for an extra 250,000 iterations, which took 50 hours on a single GPU. We started with a learning rate of 0.0001, multiplying it by 0.1 every 50k iterations.

The numerical results are summarized in Tables 5.8 and 5.9, for the validation and test

Table 5.9: Numerical evaluation on test set.

		Bellingham	Bloomington	Innsbruck	S. Francisco	East Tyrol	Overall
FCN	IoU	44.83	35.38	36.50	44.92	43.69	42.19
	Acc.	94.48	94.07	92.97	82.60	95.14	91.85
Skip	IoU	52.91	46.08	58.12	57.84	59.03	55.82
	Acc.	95.14	94.95	95.16	86.05	96.40	93.54
MLP	IoU	56.11	50.40	61.03	61.38	62.51	59.31
	Acc.	95.37	95.27	95.37	87.00	96.61	93.93

sets, respectively. We also include the performance of a *skip* network as an alternative way of combining features to refine the predictions of the coarse base FCN (see Section 5.2). Fig. 5.12 depicts close-ups of the classification on the test set, i.e., on regions never "seen" by the neural network at training time. While the FCN produces fuzzy results, it successfully identifies buildings in varied images. The MLP network provides finer outputs, as confirmed both numerically and visually.

The MLP network reaches about 60% IoU on the entire test set. This means that the output objects overlap the real ones by 60%, as assessed over a significant amount of test data. While there is certainly room for improvement, these values suggest that the current network does generalize well to different cities.

5.6 Concluding remarks

Convolutional neural networks (CNNs) are becoming the leading choice for high-resolution image classification. The major concern with this technique is the spatial coarseness of the outputs. Most of the works have moderately modified or post-processed well-known CNN architectures in order to counteract this issue. Often existing pretrained CNNs have been fine-tuned, but this still constrains us to adapt and readapt existing networks conceived for a different problem instead of creating new models. In this chapter we thus decided to rethink CNNs from a pixelwise classification perspective.

For this purpose, we first analyzed different families of dense classification CNN prototypes. This analysis bears some similarity with the reasoning that gave birth to CNNs themselves: we study which relevant constraints can be imposed in the architecture by construction, reducing the number of parameters and improving the optimization. We observed that existing networks often spend efforts in learning invariances that could be otherwise guaranteed, and reason at a high resolution even when it is not needed. While previous methods are already competitive, we can devise more optimal approaches.

We derived a model in which spatial features are learned at multiple resolutions

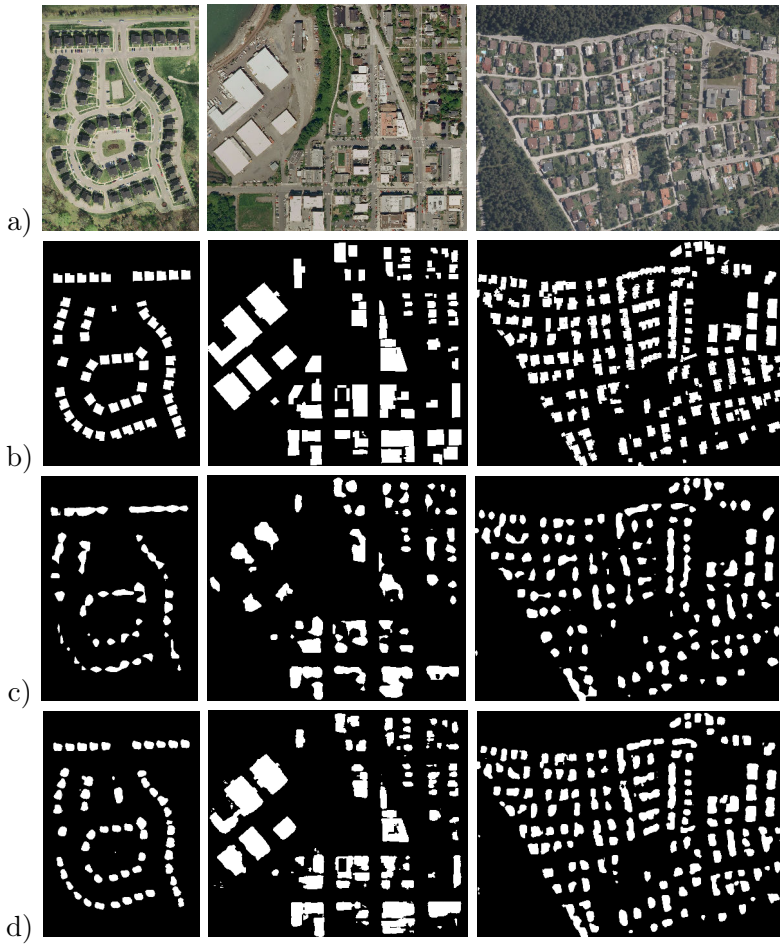


Figure 5.12: Visual close-ups on test set. (a) Color input. (b) Reference data. (c) FCN results. (d) MLP results.

(and thus different levels of detail) and a specific CNN module learns how to combine them. In our experiments on aerial imagery, such a model proved to be more effective than the other approaches to conduct high-resolution labeling. It provides a better accuracy with low computational requirements, leading to a win-win situation. Some of the outperformed methods are in fact significantly more complex than our approach, proving once again that striving for simplicity is often the way to go when using CNN architectures.

We also created a dataset for the classification of aerial images. This dataset highlights the need for methods that generalize to the dissimilar appearance of urban settlements around the earth. Contrary to previous work, the testing is not performed over excluded areas of the training surface, but over entirely different cities instead. We cover a wide range of urban densities, on both European and American cities.

Our experiments with deep neural networks show their satisfactory generalization capability. However, there is still a large room for improvement, as observed in the numerical results. We hope that this work will constitute a baseline for future research, and our dataset to be used as a benchmark for comparisons. The dataset has been made available online at mid December 2016, and has been downloaded over fifty times as of February 2017.

Let us finally remark that even when using networks specifically design to output high-resolution classification maps, we still often observe rounded corners and irregular boundaries. Let us recall that none of these regularities are enforced, only the pixel-wise classification cross-entropy error being used to train the CNNs. A possible future direction of work is to include research ways of including geometric regularities in CNNs.

Chapter 6

Polygonization of Classification Maps

One of the most important applications of remote sensing classification is to integrate the data into geographic information systems (GIS). This requires to represent the detected objects as polygons [132], which are a compact way of encoding geometric features, with the capability of editing the shape and changing the size of the objects without modifying the data file size. In addition, spatial databases such as PostGIS and SpatiaLite are optimized to query data in terms of geometry (e.g., 'how many buildings are in a certain area?').

Some object detection techniques in remote sensing directly produce polygonal data, e.g., by fitting rectangles to the image [7]. However, to account for more general shapes one must first classify every pixel and then polygonize the classification map. Moreover, with the advent of deep learning, the pixelwise classification of remote sensing imagery is becoming more and more effective [155]. The usual approach is to vectorize a raster object in a naive way, i.e, by creating a polygon whose points connect all pixels around the object boundary, and then to simplify this polygon. This is often referred to as polygon generalization [50].

We here propose a polygonization technique based on the approximation of the classification maps with a triangular mesh. A number of local operators on the mesh are simulated (but not applied) to measure their effect on the mesh based on an objective function. They are inserted into a modifiable priority queue, which allows to iteratively extract the most relevant operator and apply it, as well as to update in the queue the elements that may be potentially affected as a result of the operation. While previous methods simplify based on some measure of distance between the initial and approxi-

mated polygon boundaries, our method proceeds in an *integral* manner, i.e., considering the approximation error over the entire surface of the image. In addition, instead of just removing or flipping edges, our approach includes a rich set of operators which allows the relocation of vertices, thus adapting the mesh to the local geometry of objects.

6.1 Related work

Polygon generalization

The most common generalization algorithms can be classified into local and global processing routines. Local routines, such as the radial distance [132] and Reumann-Witkam [120] methods, compare subsequent points in a polygon to decide if one of them may be eliminated. For example, radial distance starts from a given point, and removes all points lying at a distance closer than a predefined threshold. Visvalingam-Whyatt [154] and Douglas-Peucker [37] are global greedy routines that measure the effect of including each point on the entire polygon and not only with respect to the nearest neighbors. For example, Douglas-Peucker starts from a coarse approximation (e.g. a triangle created by selecting only three points from the initial geometry). A new point is then added by bisecting each of the segments. For this, the algorithm chooses the point in the initial polygon that lies farther away from the segment. This is iterated recursively on the resulting segments until the required coarseness of the approximation is met, as determined by a user-defined parameter.

All these techniques are implemented in most GIS packages (e.g., GRASS, QGIS and ArcGIS), Douglas-Peucker being the most commonly used method by the community [132]. It has been extended to preserve topology and to generate non-self-intersecting polygons [123, 161].

Mesh approximation

A polygonal mesh is defined by a set of vertices, edges and faces, together with connectivity information, that define the shape of polyhedral objects. We here deal with the particular case in which meshes are seen as planar subdivisions [34], which divide the plane into a set of non-overlapping triangles.

The approximation of a mesh is a classical problem in geometry processing [15], where a set of operators are applied to the mesh. Half-edge collapses, edge flips and vertex removal are examples of discrete operators [15], while the relocation of a vertex to an arbitrary position is a continuous operator [2]. A flexible combination of different

operators in the same algorithm is the typical approach to successfully approximate a mesh [14]. The problem is usually formulated as the modification of a mesh to approximate another mesh or a point cloud. We here assign labels to every triangle and see the approximation error as the integral cost on the image of assigning the wrong labels.

One issue when using mesh operators is to verify that they are valid, for example, to avoid introducing overlapping or degenerate triangles. Topology preservation is also an important issue. We say that we preserve topology, when the the initial object represented by a mesh and the modified object after applying an operator are homeomorphic. This means that we can transform one object into the other one by a continuous deformation (i.e., without splitting or merging objects, introducing holes, etc.). One way of describing the class of topological space is by means of the Euler number [121], which gave birth itself to the study of topology. We here combine elements of mesh approximation and a strategy to preserve topology based on the Euler number to polygonize remote sensing classification maps.

Image vectorization

Our work can also be related to the domain of image vectorization, which seeks to approximate a color image with a set of geometric primitives. This problem gained attention with the popularization of vector graphic formats, such as SVG, which are particularly useful for Internet transmission [79]. Most approaches involve the use of triangulations or other types of meshes.

Notably, the authors of Ardeco [81] approximate the image with triangles which are used to partition it into a set of regions bounded by cubic splines. The inner colors of each region are approximated either with a constant color, or with a linear or circular gradient. Other types of meshes have been later introduced for the problem of vectorization. For example, Price and Barrett [119] use a Bézier grid to fit the image. To represent richer gradients without adding too many points, commercial graphic editors such as Adobe Illustrator and Corel CorelDraw have recently introduced gradient meshes, which consist in a grid of Ferguson patches. Such patches allow gradient control points inside the patch and not only at the vertices. Recent vectorization literature has thus focused on approximating images with gradient meshes [142, 163, 79]. In particular, Lai et al. [79] studied topology preservation on gradient meshes and proposed a fully automatic algorithm. Let us note that the goal of vectorization is significantly different than ours. Vectorization seeks to approximate an image with few geometric primitives while preserving fine details, such as high-order gradients. We wish instead to approximate classification maps

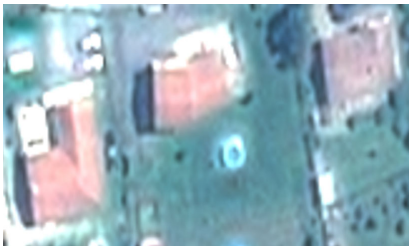
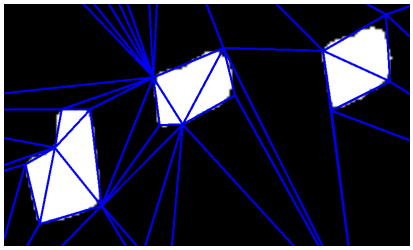
(a)  (b) 

Figure 6.1: Color input image (a) and corresponding classification map, approximated by a triangle mesh (b).

with a piecewise function. It is not our goal, for example, to approximate the gradient in a fuzzy probability map along the boundary between two objects.

6.2 Proposed method

Let us consider a triangular mesh T, consisting of a set of triangles $\{t_i\}$. This triangulation is overlaid on top of a classification map to approximate it. For example, Fig. 6.1 shows an image and its corresponding CNN classification, with a polygonal mesh on top of it (which has already been optimized by our algorithm). There is a set of class labels $\mathcal{L}$ and we define $C(l, x, y)$ to be the cost associated to assigning a certain label $l \in \mathcal{L}$ to a pixel (x, y) in the image. We assign a single label l_t to every triangle t. The cost of such an assignment is simply the cost of assigning the label uniformly to all the points inside the triangle. Throughout the algorithm, we always assign to each triangle the label with the lowest cost. For example, in Fig. 6.1 the triangles covering mostly white areas would bear the *building* label and the others the *not building* label. Our goal is to find the triangulation that minimizes:

$$E(T) = \sum_{t \in T} \left[\min_{l_t \in \mathcal{L}} \iint_{x,y \in t} C(l_t, x, y) dx dy + \lambda \right]. \tag{6.1}$$

For each triangle we sum the cost incurred by assigning the optimal label to it, plus an extra weight λ per triangle. Adding λ implies that the mere existence of a triangle has a cost, independently of its label, and is thus used as a regularization term to set the desired coarseness of the mesh.

We here consider that a classifier has been used to estimate $P(l, x, y)$, the probability $P(l, x, y)$ of assigning a certain label $l \in \mathcal{L}$ to a point (x, y) in the image (s.t. $\sum_{l \in \mathcal{L}} P(l, x, y) = 1$ and $P(l, x, y) \geq 0, \forall x, y, l$). The cost of assigning a particular label

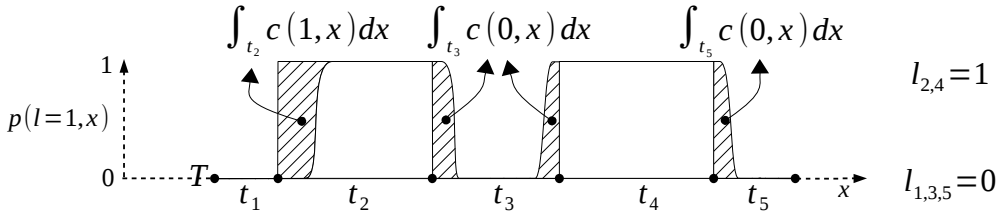


Figure 6.2: Illustration of the ℓ_1 cost on a 1-D triangulation.

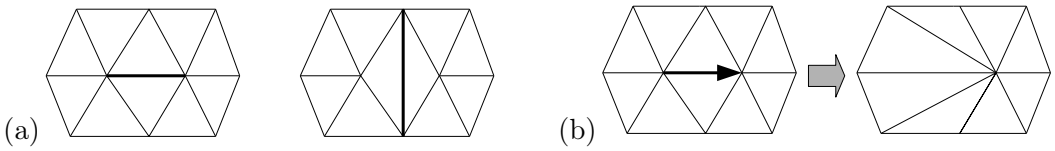


Figure 6.3: Operators: edge flip (a) and half-edge collapse (b).

to a pixel, as required by Eq. 6.1, is defined based on $P(l, x, y)$ as follows:

$$C(l, x, y) = ||1 - P(l, x, y)||_1, \tag{6.2}$$

which can be seen as the volume contained between the classifier's probability surface and the piecewise constant approximation implied by the labeled mesh. Fig. 6.2 illustrates an example of such a cost in 1D. Let us remark that in addition to using the fuzzy probabilities of a classifier, we can also use a hard classification map (i.e., a unique class assigned to each pixel) by supposing that the probability was set to 1 for the assigned class and 0 for the rest.

The triangle mesh is iteratively optimized by performing transformations, starting from an initial fine lattice mesh. We simulate a number of *changes* that transform the mesh, from T to T', and construct a modifiable priority queue on the energy variation $\Delta E = E(T') - E(T)$ associated to each change. We can restrict the calculation of ΔE to the triangles affected by the change, given the sum over independent triangles in (6.1). The highest-priority change is first popped out from the queue and applied to the mesh. Note that we must relabel the affected triangles and update those elements in the queue that may have been altered as a side effect. This is iterated until there are no changes left. We only consider changes that immediately improve T (i.e., $\Delta E < 0$) and that produce a valid triangulation (e.g., they do not lead to overlapping triangles).

The first two types of changes we consider are the *edge flip* and *half-edge collapse* [15, 151] operators. Edge flip considers the quadrangle formed by two adjacent triangles and flips the inner edge (see Fig. 6.3-a). Such a flip is only valid when the quadrangle is

strictly convex, in order to retain a valid triangulation. The half-edge collapse operator collapses an edge $\overline{\text{AB}}$ by moving vertex A toward vertex B and suppressing the two triangles adjacent to $\overline{\text{AB}}$ (see Fig 6.3-b). Since collapsing A to B and B to A are different operations, one can imagine that each edge is in fact composed of two directed "half"-edges (one from A to B and the other one from B to A) and that we collapse one of them. We must also verify that the resulting planar subdivision is a valid one.

We can transform a triangulation to become any possible simplified triangulation just by flips and collapses, as soon as the vertices are on fixed locations [15], making those two types of operators particularly appealing. However, we also add a *vertex relocation* operator that computes a new position for a certain vertex. This increases the expressiveness of the family of operators, compensating the limitation of flips/collapses that simply recombine predetermined vertices. For example, vertex relocation allows us to start the optimization from a relatively coarse mesh and yet achieve similar (or even better) results than if we departed from a fine mesh (e.g., a vertex on every pixel) and only used flips and collapses. Note that different types of operators can be mixed together in the queue, or be applied at will in subsequent stages of the algorithm.

Once the mesh has been optimized, the connected components of triangles with the same class label are outputted as polygonal objects of such class.

We explain next our vertex relocation operator, as well as a strategy to preserve topology.

6.2.1 Vertex relocation

In this section we explain our vertex relocation operator. We first see the object boundaries in the classification map as curves in the plane, an object being a connected component of pixels of the same class. We also see the boundaries of objects in the triangulation as curves in the plane. Denoting by M the "real" boundaries on the classification map and by $\hat{M}$ the ones implied by the labeled triangulation, our goal is to modify $\hat{M}$ so that it approaches M. We use the ℓ_1-norm and seek to minimize the area $A(M, \hat{M})$ contained between the two curves, referred to as area of symmetric differences. We adapt the algorithm from [2], where a volume criterion was used to simplify 3-D meshes.

Given a curve $\hat{M}$ parametrized by u, we define $\delta\hat{M}(u)$ as the displacement of a point that belongs to such curve (see Fig. 6.4), with the goal of approximating M. We assume that $\hat{M}$ can be locally approximated by its tangent, and define du to be a small displacement along the tangent. The variation of area $\delta A(M, \hat{M})_u$ incurred by performing the displacement $\delta\hat{M}(u)$ corresponds to the parallelogram generated by $\delta\hat{M}(u)$ and du.

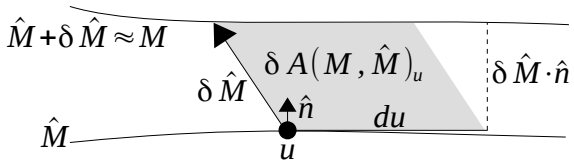


Figure 6.4: Elementary area variation δA generated by $\delta\hat{M}$.

The area of such a parallelogram is $|\delta\hat{M}(u)\cdot\hat{n}|\cdot du$, being $\hat{n}(u)$ a vector normal to $\hat{M}$. Since the variation of area $\delta A(M,\hat{M})_u$ may be positive or negative (because $\delta\hat{M}(u)$ may move $\hat{M}$ closer or farther away from M), we define:

$$\delta A(M,\hat{M})_u = \eta(u)(\delta\hat{M}(u)\cdot\hat{n})du, \tag{6.3}$$

where η returns -1 if if $\hat{n}$ points toward the area embedded between M and $\hat{M}$ and $+1$ otherwise, since it would be reducing and increasing the area, respectively (assuming $\hat{n}$ is oriented so that $\delta\hat{M}\cdot\hat{n}$ is positive). The total area variation upon applying all displacements is:

$$\delta A(M,\hat{M}) = \int_u \eta(u)(\delta\hat{M}(u)\cdot\hat{n})du. \tag{6.4}$$

However, while (6.4) applies to any curve, in our mesh when we move a vertex X_i the points along adjacent edges move accordingly. This is because we work in the space of piecewise linear meshes, and the degrees of freedom amounts to the number of control vertices in the mesh, not the total (infinite) amount of points in the mesh. In fact, the $\delta\hat{M}(u)$ of a point inside a segment $\overline{X_aX_b}$ is a linear combination of δX_a and δX_b:

$$\delta\hat{M}(u) = \lambda_a(u)\delta X_a + \lambda_b(u)\delta X_b, \tag{6.5}$$

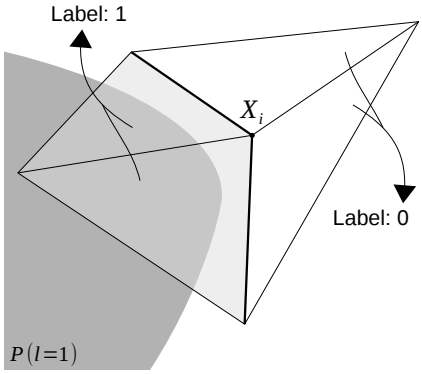
where $\lambda_{a,b}(u)$ are shape functions [68] such that λ_i is 1 on X_i and linearly decreases to zero until reaching the following and previous vertices in the curve. This interpolates X_a and X_b based on their relative distance to $\delta\hat{M}(u)$.

If we differentiate (6.4) with respect to a particular mesh vertex X_i and use (6.5), we get:

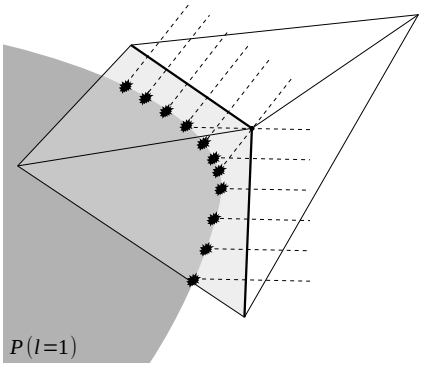
$$\frac{\partial A(M,\hat{M})}{\partial X_i} = \int_{u*} \eta(u)\lambda_i(u)\hat{n}(u)du. \tag{6.6}$$

We restrict the domain of integration to the points in edges adjacent to X_i (which we denote by $u*$) because only there $\lambda_i(u)$ is nonzero.

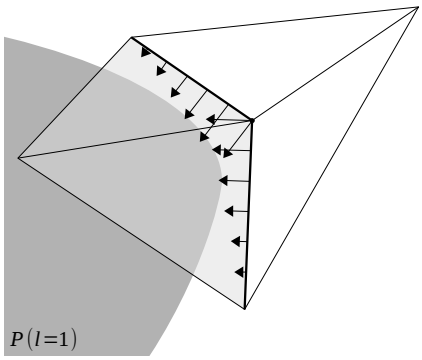
Note that the challenge of evaluating (6.6) reduces to the computation of $\eta(u)$, i.e., the relative orientation of the object we wish to approximate with respect to the mesh



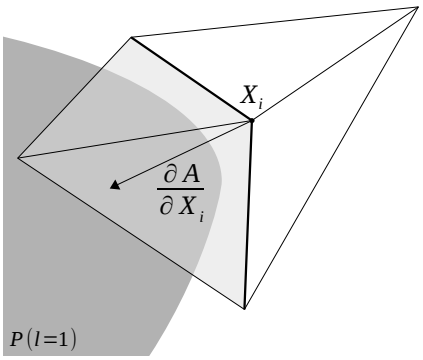
(a) Initial configuration around X_i.



(b) Ray shooting from various points u normal to the edges to estimate relative orientations $\eta(u)$.



(c) Resulting vectors $\eta(u)\lambda_i(u)\hat{n}(u)$ (scaled by the shape function λ_i, see Eq. 6.6).



(d) The summation of the vectors in (c) yields an estimation of the gradient.

Figure 6.5: Relocating vertex X_i. From the edges that are interfaces between labels 1 and 0 we shoot rays to estimate relative orientations. The gradient of the objective function with respect to the vertex location is then computed.

objects. We evaluate (6.6) as follows: first we take a discrete number of points along the edges adjacent to X_i. Not that we only consider the edges that are object boundaries (i.e., adjacent to triangles with different labels) and ignore the objects' inner edges. From each of these points along the edges we "shoot rays" normal to the edge to decide to which side lies the curve we want to approach. For example, in the 2-class case we just shoot rays in opposite directions and see which one crosses first the 0.5 probability level, where there is a change of classes and hence an object boundary. This way we know $\eta(u)$ to evaluate (6.6). The overall principle is illustrated in Fig. 6.5.

The iterative optimization of X_i is performed by gradient descent. Being k the

iteration number, we set:

$$X_i^{(k+1)} = X_i^{(k)} - \alpha^{(k)} \frac{\partial A^{(k)}(M, \hat{M})}{\partial X_i}, \tag{6.7}$$

where $\alpha^{(k)}$ is an adaptive step that starts at $\alpha^{(0)} = \alpha$ and is multiplied by a factor $\gamma < 1$ whenever an oscillation is detected ($\frac{\partial A^{(k)}}{\partial X_i} \cdot \frac{\partial A^{(0)}}{\partial X_i} < 0$).

As with the other operators, these moves are simulated and added to the priority queue based on the associated ΔE.

6.2.2 Topology preservation

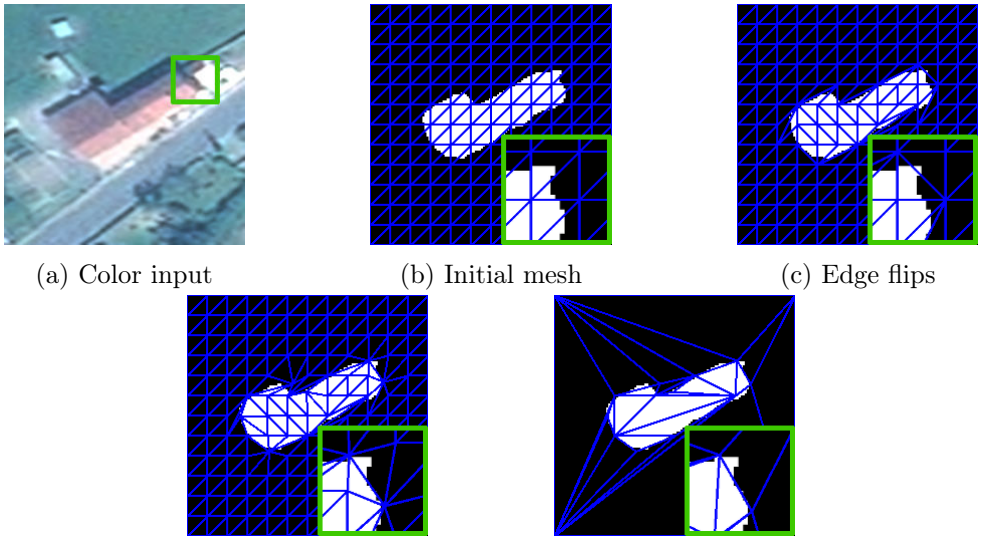
While some mesh approximations may correspond to a low objective function (6.1), they may be unpleasant from a qualitative point of view because they modify the topology of the objects. This frequently occurs when nearby buildings merge into one single object, or buildings that were not adjacent in the initial mesh are connected in the simplified mesh.

We deal with these cases by preventing changes that incur topological changes. To describe the topology of the objects of a certain class we use the Euler characteristic $\chi = V - E + F$, where V, E and F are the number of vertices, edges and faces in a mesh, respectively [121]. For example, when there is a single object of a class it is $\chi = 1$, for two separate objects $\chi = 2$, for one object with two holes $\chi = -1$. For each class we must verify that χ does not change as a result of applying an operator. We do this in practice by counting the changes ΔV, ΔE and ΔF incurred by the operator, only doing this locally on the elements that may have changed as a result of its application. We then verify that $\Delta\chi = \Delta V - \Delta E + \Delta F = 0$. We count those vertices, edges and faces that are adjacent to a triangle labeled with the class for which we are verifying the topological change.

Notice that we may adjust the tolerance to topological changes by the coarseness of the initial mesh. For example, if a small hole in the classification map is ignored in the initial labeling, it will remain like that. In other words, one reasonably expects that the initial mesh is finer than what is considered to be the minimum object size.

6.3 Experiments

We experiment on the dataset of Pléiades satellite imagery introduced in Section 3.4.2, and consider one of the tiles that has been manually labeled into two classes: building/not



(a) Color input (b) Initial mesh (c) Edge flips

(d) Vertex relocations (e) Edge collapses

Figure 6.6: Effect of applying the different mesh operators.

building. We use the classification maps obtained by the two-scale convolutional neural network presented in Section 3.3.3.

Let us first illustrate the effect of applying the different operators. Fig. 6.6(a) shows a piece of color image from the dataset. In Fig. 6.6(b) we show the corresponding classification map outputted by the CNN, overlayed with the initial mesh (a lattice with one vertex every 10 pixels). Through the samples we include a close-up that corresponds to the area indicated by the green square on Fig. 6.6(a). Upon filling the priority queue with flips and applying them till the queue is empty, we get the mesh on Fig. 6.6(c). We can observe that the edges get better aligned to the boundaries of the underlying object, yet the vertices are not located on the boundaries. On this mesh we perform vertex relocations in a similar fashion, obtaining the mesh in Fig. 6.6(d). We now see that the vertices have been relocated near the boundaries. Finally, collapses are done as shown in Fig. 6.6(e), simplifying the mesh as required.

We now compare our polygonization method with competing methods. Our algorithm is as follows: we first perform a stage involving only edge flips, followed by a stage of vertex relocations (as in Fig. 6.6). Then we perform edge collapses and, after each collapse we immediately simulate a vertex relocation on the collapsed node (because we assume that the collapsed node is not relocated to the optimal position right away). We can alternatively just mix all the operations together, but we found this way to be more efficient and elegant because the first two stages better align the initial mesh to

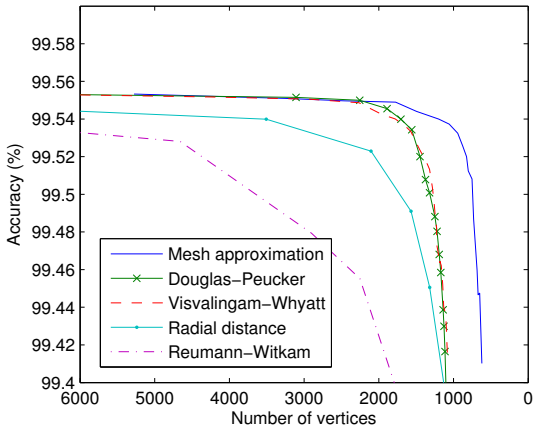


Figure 6.7: Performance comparison.

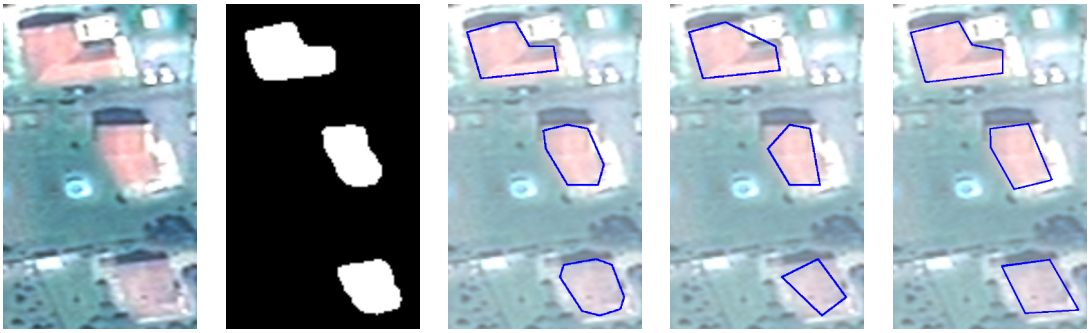
the classification map at very low cost, before starting to collapse edges. We start from a fine mesh (one vertex per pixel, in a band around object boundaries).

For vertex relocation we shoot five rays per segment (weighting them based on their distance du) and perform gradient descent with $\alpha = 0.1$, $\gamma = 0.1$, stopping when $\alpha^{(n)} < 0.0001$ or the displacement is below 0.01 pixels.

The initial steps of edge flips and vertex relocations take six seconds to execute on the 3000×3000 input (on a machine with an 8-CPU 2.76 GHz processor). The vertex relocation algorithm tends to converge in 5 to 9 gradient descent iterations. The subsequent edge collapse/vertex relocation stage takes six minutes. This is due to the fact that the elements in the queue that may be affected by the application of an operator must be updated and, as the algorithm progresses, the triangles become larger. Therefore, the time required to recompute the integral over the possibly affected triangles increases, and becomes the algorithm's bottleneck. Instead of performing a thorough update of the priority queue, we may consider in the future to follow a more relaxed approach, such as in the BPT optimization presented in Chapter 2, where k operators are applied at once without updating all the elements in the queue, and the entire process is iterated.

We compare our polygonization method with the standard techniques used in geographic information systems (GIS). We first vectorize the rasters with GDAL library's function *gdal_polygonize* and then use GRASS and QGIS implementations of the simplification methods mentioned in the related work section.

In Fig. 6.7 we plot the accuracy of the polygonal approximation as a function of the amount of vertices (e.g., for different values of λ in Eq. 6.1). This is done on the entire binary labeling test set presented in Section 3.4.2. Since we have ground truth data,



(a) Color input (b) Classif. map (c) DP, thr. 3 (d) DP, thr. 7 (e) Mesh approx.

Figure 6.8: Visual comparison ("DP, thr. 3": Douglas Peucker with threshold 3).

accuracy is measured as the percentage of correctly classified pixels when rasterizing the polygons. Our algorithm outperforms the other techniques, including the popular Douglas-Peucker and Vasvalingam-Whyatt which are the most accurate among them. The advantage is significant, for example, for a desired accuracy of 99.54%, Douglas-Peucker requires 1200 vertices while our technique achieves the same accuracy with only 700. This is also appreciated visually. Figs. 6.8(a-b) show a piece of color image and the corresponding classification map. Figs. 6.8(c) and (d) show the result of applying Douglas-Peucker with two different threshold parameters, and Figs. 6.8(d) illustrates our results. The parameters of Douglas-Peucker were selected so as to provide a similar accuracy to our method (in Fig. 6.8c) or a similar amount of vertices than our method (in Fig. 6.8-d). We observe that for a similar accuracy there are too many vertices compared to Fig. 6.8(d), while for the same number of vertices the polygons by Douglas-Peucker do not represent the underlying objects well, explaining the gap between the curves in Fig 6.7.

In Fig. 6.9 we illustrate the effect of removing the topological constraints described in Sec. 6.2.2, where we observe that our methodology effectively outputs polygons that better convey the topology of the underlying objects. If we observe the numerical results (see Fig. 6.10), we can appreciate that the topologically inconsistent algorithm yields a slightly better accuracy. This is because the mesh approximation algorithm indeed minimizes the energy, but such energy does not include the topological constraints. Penalizing topological changes in the energy would in fact require higher-level priors (such as the shape priors introduced in Chapter 2). We here effectively add topological consistency by invalidating operators that violate it. Finally, in Fig. 6.11 we show that the use of vertex relocation consistently outperforms an equivalent algorithm where vertex relocation is removed.

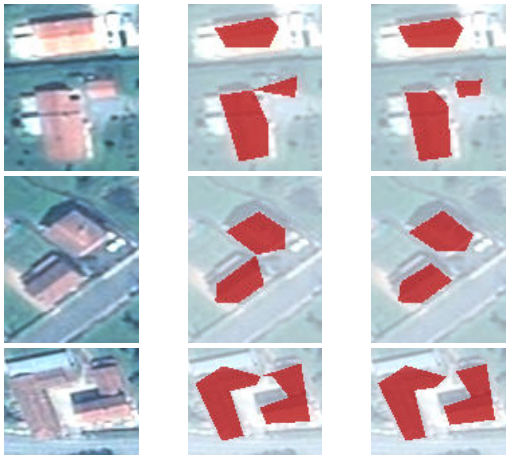


Figure 6.9: Examples of color inputs (left) and approximation without (center) and with (right) topological constraints.

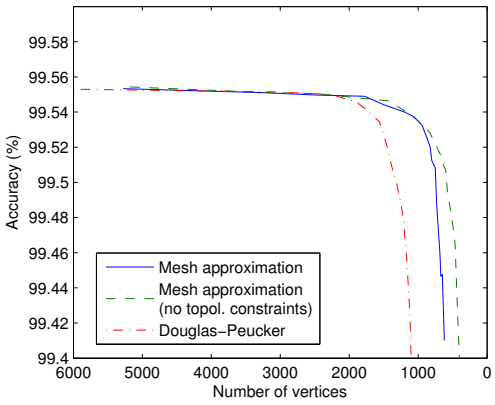


Figure 6.10: Approximation with and without topology preservation.

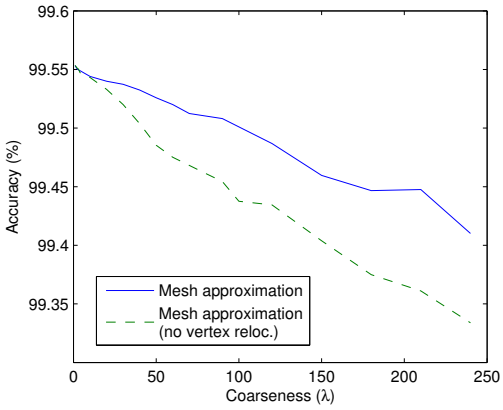


Figure 6.11: Approximation with and without vertex relocation.

6.4 Concluding remarks

We presented a mesh approximation algorithm to polygonize remote sensing classification maps. We depart from a fine lattice mesh over a classification map, and iteratively apply mesh operators that transform or simplify the triangulation. We include a vertex relocation operator that estimates the optimal vertex displacement direction, and relocates the vertices by gradient descent. We also include a mechanism to avoid changing the topology of the classified objects as a result of the application of the mesh operators.

By using the proposed method, the polygonal objects are more accurate than the current techniques used in the GIS community. We provide good approximations of the objects with a significantly lower number of vertices. This is partly because we allow vertices to be located anywhere and not just exactly on the boundary of the original raster objects. Another distinctive property of our technique is that we measure the approximation error in an *integral* manner over the entire surface of the image.

An interesting future direction of work is to reinforce regularity relationships (e.g., parallelism, right angles) in the framework, learned from training data, as well as to introduce machine learning in the decision of applying the different operators.

Chapter 7

Conclusions and Perspectives

The main objective of this thesis was to explore the problem of large-scale remote sensing image classification. To properly identify the different objects, it is required to add some degree of high-level semantic reasoning, since it insufficient to simply analyze the individual spectral values measured by the sensors. The current technological settings and application domains of remotely sensed images require the design of scalable methods, to process images covering large extents of the earth's surface and contemplating the potential landscape variability throughout the different geographic locations.

In a first approach (Chapter 2), we proposed a method for image classification with shape priors. Compared to sate-of-the-art techniques, we allow the occurrence of multiple objects of the same class in a single scene. In addition, we combine multiple different shape descriptors in a single energy, with both the spectral and shape feature's distributions being trained from labeled data. The proposed method builds upon binary partition trees (BPTs), a hierarchical data structure which we built in a greedy way and then optimize by pruning and regrafting tree branches.

In the following chapters of this thesis, we studied the use of convolutional neural networks (CNNs) for classification, since they are capable of learning complex shape features from training data. Compared to BPTs, this adds scalability in terms of the features that are used to perform the classification, as well as a remarkable computational performance, which is particularly important considering the current large-scale processing requirements. Being a recent tool, there is still significant work to do to understand and adapt CNNs for the problem of dense pixel classification. In Chapter 3, we proposed the so-called fully convolutional network as the architectural base from which to derive remote sensing classification networks. We also proposed a scheme to deal with the current imperfection in available labeled data, by training a CNN in two steps: one with massive but potentially imperfect data, followed by a fine-tuning step with a

small amount of accurate data. In addition, we presented an initial approach toward outputting fine-grained classification maps, a prevalent concern in CNNs.

In order to enhance a CNN's output and yield a fine-grained classification, it has become a common practice to add an iterative post-processing module that sharpens classification maps around object boundaries. In Chapter 4 we developed a recurrent neural network that directly learns an iterative process to perform such an enhancement, instead of defining the algorithmic details by hand. This represents a new way of using neural networks, and further highlights their potential.

While post-processing modules are certainly a popular workaround, we consider that we should still study the ways in which CNNs could yield finer outputs in the first place. For this purpose in Chapter 5 we reviewed and analyzed the recent literature on high-resolution labeling with CNNs, and proposed a network architecture well adapted to this problem. Notably, we experimented on the increasingly popular ISPRS datasets and submitted a contribution to the associated contest. Besides, we created an aerial image labeling dataset which covers dissimilar urban landscapes around the earth, to assess the different methods' generalization capabilities.

One of the main general conclusions we can draw from our work with CNNs is that the design of a proper architecture significantly improves the results and often the computational performance. This is achieved by imposing sensible restrictions to the neuronal connections. This is, in fact, what brought CNNs themselves to success. We have shown ourselves, for example, that fully convolutional networks (Chapter 3) and our MLP network (Chapter 5) improve performance by a thoughtful design of the architectures.

In a certain way, we may consider that we are redirecting some of the efforts previously spent on feature engineering (e.g., the design of shape descriptors in Chapter 2) toward the design of the CNN's architecture. Curiously enough, we have been asked on several occasions through the development of this thesis about why one would use a CNN if we *still* have to spend efforts on designing the architecture. It is however not our goal to do things effortlessly. In fact, one cannot pretend to design a method that solves any problem without any efforts or making certain assumptions, unless it is some sort of *magical* tool. Finding it would violate the principle of "no free lunches", which has been formalized in the context of machine learning [160]. What has driven so much interest to CNNs and deep learning may be the fact that the payoff of the efforts made is very good compared to other approaches. For example, often an initial sketch of a network, without a thorough analysis, yields acceptable solutions, as well as borrowing an existing architecture. This would of course be suboptimal compared to designing and tuning architectures specifically tailored for the problem. In other words, the abstract

work required to conceive an appropriate architecture seems to often have a practical competitive edge over the manual labor required to design all the features and fuzzy rules. CNNs certainly offer overall an interesting framework to discover and use high-level complex features for classification.

In the last section of this thesis (Chapter 6) we approached a different problem: the polygonization of classification maps, since some of the most common applications of remote sensing image analysis require to integrate the objects as polygons in a geographic information system. While CNN's outputs are unstructured, e.g., featuring rounded corners and irregular edges, we here introduce structure in terms of the underlying representation: a triangular mesh. We use an integral energy minimization approach, where we measure an approximation error over the entire image surface. This yields competitive results compared to the standard off-the-shelf tools used in the remote sensing community.

Perspectives

The acquisition of training data for remote sensing image classification opens interesting perspectives. The automatic correction of existing crowd-sourced databases, such as OpenStreetMap (OSM), is a possible research direction. Notably, the estimation of road width based on OSM data has been recently studied [102]. The recent use of recurrent neural networks to iteratively relocate face landmarks [147] may be a source of inspiration to learn to correct the vertices of OSM polygons. The fusion of different image sources, as well as the processing done on top of remote sensing images (e.g., orthorectification and pansharpening) could also be possibly improved by using machine learning mechanisms.

As mentioned before, one of the prevalent issues with CNN and non-CNN dense classification is the lack of geometric regularity in the outputs. One solution is to introduce geometry at the polygonization step (as done in Chapter 6) and another solution would be to, hopefully, introduce geometric constraints in the CNN classification. Regarding the first possibility, the introduction of more advanced regularities to the mesh approximation algorithm could be studied (e.g., right angles and parallelism). Concerning the second direction of work, it would certainly be interesting to design neural networks that output geometric objects instead of raster classification maps. For example, a recurrent neural network has been used to output the convex hull of a point cloud [153] and in [40] the coordinates of object bounding boxes are regressed by a CNN. Another possibility is to use the increasingly popular adversarial networks [58] to encourage a CNN to output such a realistic classification map that it can fool another classifier into believing that it is a real one.

Appendix A

Proofs related to BPT optimization

We first elaborate the proofs that supplement the paragraphs that describe the space of moves in the optimization approach. We then add the proofs of the space and computational complexities of including convex hulls in Binary Partition Trees (BPTs)

A.1 Properties of *prune-and-paste* moves

Proposition 1. *Given a tree τ, suppose a node R_m is pasted at $\tau_i < \tau_1$ leading to a new tree φ. Let us consider an alternative move that pastes R_m at τ_j, with $\tau_i < \tau_j < \tau_1$, producing a tree ψ. In the cases where either $C(\varphi_1) - C(\tau_1) \leqslant 0$ or $C(R_m) \geqslant C(\varphi_1) - C(\tau_1)$, then $C(\psi_1) \geqslant C(\varphi_1)$.*

Proof. Let us abbreviate $\mathcal{E}(\tau_i)$ as e_i^τ and $C(\tau_i)$ as c_i^τ. Following (2.12) in Chapter 2,

$$c_1^\tau = min(e_1^\tau, \overline{c_2^\tau} + min(e_2^\tau, \overline{c_3^\tau} + min(...min(e_{i-2}^\tau, \overline{c_{i-1}^\tau} + min(e_{i-1}^\tau, \overline{c_i^\tau} + c_i^\tau))...))), \quad \text{(A.1)}$$

where $\overline{c_i^\tau}$ denotes the sibling of c_i^τ (see Fig. 2.4-b). This implies:

$$\underbrace{\underset{k=1}{\overset{i-1}{\forall}} \left(e_k^\tau \geqslant c_1^\tau - \Sigma_{j=2}^k \overline{c_j^\tau} \right)}_{\alpha} \text{ and } \left(c_i^\tau \geqslant c_1^\tau - \Sigma_{j=2}^i \overline{c_j^\tau} \right). \quad \text{(A.2)}$$

Let us now suppose that a node R_m is pasted at τ_i. Then

$$c_1^\varphi = min(e_1^\varphi, \overline{c_2^\tau} + min(e_2^\varphi, \overline{c_3^\tau} + min(...min(e_{i-2}^\varphi, \overline{c_{i-1}^\tau} + min(e_{i-1}^\varphi, \overline{c_i^\tau} + min(e_x^\varphi, c_i^\tau + c_m^R)))...))), \quad \text{(A.3)}$$

which implies:

$$\underbrace{\underset{k=1}{\overset{i-1}{\forall}}\left(e_k^{\varphi} \geqslant c_1^{\varphi} - \Sigma_{j=2}^{k}\overline{c_j^{\tau}}\right)}_{\beta} \text{ and } \left(e_x^{\varphi} \geqslant c_1^{\varphi} - \Sigma_{j=2}^{i}\overline{c_j^{\tau}}\right) \text{ and } \underbrace{\left(c_m^{R} \geqslant c_1^{\varphi} - \Sigma_{j=2}^{i}\overline{c_j^{\tau}} - c_i^{\tau}\right)}_{\gamma}. \quad \text{(A.4)}$$

Let us now paste R_m one position upper than before. We wish to check if it is possible that this move will be better that the previous one ($c_1^{\psi} < c_1^{\varphi}$):

$$\begin{aligned} c_1^{\psi} =& min(e_1^{\varphi}, \overline{c_2^{\tau}} + min(e_2^{\varphi}, \overline{c_3^{\tau}} + min(...min(e_{i-2}^{\varphi}, \overline{c_{i-1}^{\tau}} + min(e_y^{\psi}, \\ & c_m^{R} + min(e_{i-1}^{\tau}, \overline{c_i^{\tau}} + c_i^{\tau})))...)) < c_1^{\varphi}. \end{aligned} \quad \text{(A.5)}$$

This can be true if and only if:

$$\begin{aligned} & \underbrace{\underset{k=1}{\overset{i-2}{\exists}}\left(e_k^{\varphi} < c_1^{\varphi} - \Sigma_{j=2}^{k}\overline{c_j^{\tau}}\right)}_{I} \text{ or } \underbrace{\left(e_y^{\psi} < c_1^{\varphi} - \Sigma_{j=2}^{i-1}\overline{c_j^{\tau}}\right)}_{II} \\ & \text{or } \underbrace{\left(e_{i-1}^{\tau} < c_1^{\varphi} - \Sigma_{j=2}^{i-1}\overline{c_j^{\tau}} - c_m^{R}\right)}_{III} \text{ or } \underbrace{\left(c_m^{\tau} < c_1^{\varphi} - \Sigma_{j=2}^{i}\overline{c_j^{\tau}} - c_i^{\tau}\right)}_{IV}. \end{aligned} \quad \text{(A.6)}$$

In this expression, I contradicts β. Considering that $e_y^{\psi} = e_{i-1}^{\varphi}$ (see Fig. 2.4-b), the term II also contradicts β. The term IV contradicts γ. We must now analyze III. By combining III and α:

$$c_1^{\tau} - \Sigma_{j=2}^{i-1}\overline{c_j^{\tau}} \leqslant e_{i-1}^{\tau} < c_1^{\varphi} - \Sigma_{j=2}^{i-1} - c_m^{R} \Rightarrow c_m^{R} < c_1^{\varphi} - c_1^{\tau}. \quad \text{(A.7)}$$

If $c_1^{\varphi} - c_1^{\tau} \leqslant 0$, then c_m^{R} must be non-positive, which contradicts our hypothesis. If $c_1^{\varphi} - c_1^{\tau} > 0$: then it must be $c_m^{R} < c_1^{\varphi} - c_1^{\tau}$. As a conclusion, if the first move decreases C, III is contradicted, hence it must be $c_1^{\psi} \geqslant c_1^{\varphi}$. For a positive gain, III is contradicted unless $c_m^{R} < c_1^{\varphi} - c_1^{\tau}$. □

Proposition 2. *Let us consider a case where Prop. 1 hypotheses do not apply. There might then exist a higher paste place τ_α so that $C(\psi_1) < C(\varphi_1)$. Let us suppose that instead of pasting at τ_α we paste at τ_β, with $\tau_\alpha < \tau_\beta < \tau_1$, leading to a tree ρ. Then $C(\rho_1)$ would monotonously decrease as the paste place τ_β is located higher.*

Proof. If Prop. 1 hypotheses do not apply, then the term III in its proof must be true. This term implies that when pasting at τ_j, the cut on the tree will be located at or below

R_m. The cost c_1^{ψ} associated with this move will then be

$$c_1^{\psi} = \Sigma_{j=2}^{i}\overline{c_j^{\tau}} + c_i^{\tau} + c_m^R, \tag{A.8}$$

considering the location of the new cut. Analogously, the cost when pasting R_m k units up of i is:

$$c_1^{\rho} = \Sigma_{j=2}^{i-k}\overline{c_j^{\tau}} + c_{i-k}^{\tau} + c_m^R. \tag{A.9}$$

Notice in the previous expression that we consider that the cut is still as low as R_m. The cut could not be higher, because if it were the case, then it would have already been cut there before.

Let us now see if the cost c_1^{ρ} could increase as k advances:

$$\begin{aligned} &\Sigma_{j=2}^{i-k}\overline{c_j^{\tau}} + c_{i-k}^{\tau} + c_m^R - (\Sigma_{j=2}^{i-k+1}\overline{c_j^{\tau}} + c_{i-k+1}^{\tau} + c_m^R) \\ =& -\overline{c_{i-k+1}^{\tau}} + c_{i-k}^{\tau} - c_{i-k+1}^{\tau} > 0 \\ \Leftrightarrow& c_{i-k}^{\tau} > c_{i-k+1}^{\tau} + \overline{c_{i-k+1}^{\tau}}, \end{aligned} \tag{A.10}$$

contradicting the algorithm to compute the cuts, hence the cost at R_1 must monotonously decrease. □

Proposition 3. *Let us suppose we paste R_m at or over the initial cut of tree τ, leading to tree φ. Let us consider we paste higher instead, producing tree ψ. It must then be $c_1^{\psi} \geqslant c_1^{\varphi}$.*

Proof. Let us resume the proof of Proposition 2. It was shown that $c_1^{\psi} < c_1^{\varphi}$ if and only if III was false. At that point we could not contradict III but show that under certain conditions it would be contradicted. Now we will show that the fact that we know the first cut was at or below τ_i will contradict III.

After III and γ we have:

$$\begin{aligned} &e_{i-1}^{\tau} + c_1^{\varphi} - \Sigma_{j=2}^{i}\overline{c_j^{\tau}} - c_i^{\tau} \leqslant e_{i-1}^{\tau} + c_m^R < c_1^{\varphi} - \Sigma_{j=2}^{i-1}\overline{c_j^{\tau}} \\ \Leftrightarrow& e_{i-1}\tau - \overline{c_i^{\tau}} - c_i^{\tau} \leqslant e_{i-1}^{\tau} + c_m^R < 0. \end{aligned} \tag{A.11}$$

If we now add the knowledge about the cut being below τ_{i-1}, it must be $e_{i-1}^{\tau} > c_i^{\tau} + \overline{c_i^{\tau}}$, then

$$c_i^{\tau} + \overline{c_i^{\tau}} - \overline{c_i^{\tau}} - c_i^{\tau} < e_{i-1}^{\tau} - \overline{c_i}^{\tau} - c_i^{\tau} \leqslant e_{i-1}^{\tau} + c_m^R < 0 \Leftrightarrow 0 < 0. \tag{A.12}$$

Therefore, III cannot be true, which proves the proposition.

□

Corollary: Combining Proposition 2, where C monotonously decreases ($\leqslant$) as the paste location gets higher, and Proposition 3, where C cannot decrease ($\geqslant$), it becomes evident that pasting a node anywhere between the initial cut and the lowest common ancestor produces the same effect on the energy.

Proposition 4. *The amount of spatially adjacent regions in a balanced BPT is bounded by* $O(n\log(n))$.

Proof. Let us call $\mathcal{N}_R$ the number of neighbors of the region R. At the lowest scale and in a discrete environment we can suppose that the number of neighbors is equal to its boundary length (δR). We are interested in knowing the number of neighbors at all scales. In a balanced tree it can be assumed that the number of neighbors at every scale is half the number at the following one. As a result:

$$\mathcal{N}_R = \delta R + \frac{1}{2}\delta R + \frac{1}{2^2}\delta R + \frac{1}{2^3}\delta R + ... < 2\delta R. \tag{A.13}$$

In a discrete implementation (assuming 4-connectivity):

$$\mathcal{N}_R < 2\delta R \leqslant 2 \cdot 4|R| = 8|R|. \tag{A.14}$$

The summation of the neighbors of *all* regions in a tree $\mathcal{T}$ is then

$$\sum_{R_i \in \mathcal{T}} \mathcal{N}_{R_i} < 8 \sum_{R_i \in \mathcal{T}} |R_i|. \tag{A.15}$$

Following (A.18), the total number of neighbors (the possible cut/paste moves) is a factor of $n\log(n)$. □

A.2 Complexity of incorporating convex hulls

Proposition 5. *The storage space required to add the convex hull to every node of a balanced BPT in a discrete environment is bounded by* $O(n\log(n))$.

Proof. Let us call $CH(R)$ the convex hull of a region R. In the extreme case (the most compact region), $CH(R)$ can be as large as the perimeter δR of R which, in a discrete implementation (assuming 4-connectivity) does not contain more points than four times the area of the region:

$$CH(R) \leqslant \delta R \leqslant 4|R|. \tag{A.16}$$

As a consequence, the points of the convex hull of *all* regions in a tree $\mathcal{T}$ must be

$$\sum_{R_i \in \mathcal{T}} |CH(R_i)| \leqslant 4 \sum_{R_i \in \mathcal{T}} |R_i|. \tag{A.17}$$

If we observe that in a balanced tree

$$\sum_{R_i \in \mathcal{T}} |R_i| = \sum_{l=1}^{\#levels} \sum_{R_j \in l \in \mathcal{T}} |R_j| = \sum_{l=1}^{\#levels} n = n \sum_{l=1}^{\#levels} 1 = n \cdot \#levels = n \log(n), \tag{A.18}$$

then (A.17) is bounded by a factor of $n \log(n)$.

□

Proposition 6. *The complexity of computing the convex hull of every region represented in a balanced BPT in a discrete environment, is bounded by* $O(n \log(n))$.

Proof. Let us call $CH(R)$ the convex hull of a region R. In the extreme case (the most compact region), $CH(R)$ can be as large as the perimeter δR of R which, in a discrete implementation (assuming 4-connectivity) does not contain more points than four times the area of the region:

$$CH(R) \leqslant \delta R \leqslant 4|R|. \tag{A.19}$$

The time to compute $CH(R_i)$ is linear on the number of points in the polygons of the children:

$$O(|\delta LeftChild(R_i)| + |\delta RightChild(R_i)|). \tag{A.20}$$

The time to compute the convex hull of every node in the tree is then bounded by a factor of:

$$\sum_{R_i \in \mathcal{T}} (|\delta LeftChild(R_i)| + |\delta RightChild(R_i)|)$$
$$\leqslant 4 \sum_{R_i \in \mathcal{T}} (|LeftChild(R_i)| + |RightChild(R_i)|) = 4 \sum_{R_i \in \mathcal{T}} |R_i|. \tag{A.21}$$

Following (A.18), the execution time is then a factor of $n \log(n)$. □

Résumé Étendu

R.1 Introduction

Ces dernières années, des avancées technologiques ont permis d'augmenter la résolution et disponibilité des images de télédétection. Par exemple, la constellation de satellites Pléiades fournit des images à une résolution spatiale de moins d'un mètre, en revisitant toute la terre chaque jour. Il est donc devenu extrêmement important de développer des méthodes pour l'analyse automatique de ces données.

Dans ces conditions, un des principaux défis est de s'adapter aux nouvelles caractéristiques des capteurs, étant donné que les anciennes méthodes ne sont plus forcément applicables. Par example, pendant des années, la recherche s'est concentrée sur la découverte des matériaux *mélangés* dans la mesure d'un pixel [75]. Maintenant, vu que les pixels sont petits par rapport à la taille des objets, les efforts de recherche se concentrent plutôt sur la consistance de l'analyse à travers les différents pixels appartenant à un même objet. Un deuxième défi est le passage à l'échelle des systèmes conçus. Étant donné qu'il y a de plus en plus d'accès à des images couvrant une large étendue de surface terrestre, il est notamment important de concevoir des approches « scalables » (passant à l'échelle), y compris pour traiter des images couvrant toute la terre [102]. En dehors de l'évidente nécessité d'efficacité en temps de calcul [104, 80], il est aussi important de concevoir des méthodes qui produisent de bons résultats de façon consistante sur des différentes zones de la planète, compte tenu de la variabilité de l'apparence des objets selon la région. Cette variabilité peut être appréciée sur les example de la Fig. R.1. De plus, l'analyse des matériaux physiques (ex. *tôle* [10]) a maintenant évolué plutôt vers des classes sémantiques, telles que *voiture* [155].

L'interprétation automatique des images de télédétection est souvent formulée comme un problème de *classification* pixel à pixel. On peut aussi la formuler comme la subdivision d'une image en segments, et l'attribution d'une classe à chaque segment, renforçant ainsi l'idée de l'existence d'objets sémantiques. Dans cette thèse on se focalise sur la

Figure R.1: Exemples d'images aériennes.

classification *supervisée*, où le classifieur est entraîné à partir de données labellisées.

Objectif

L'objectif de cette thèse est d'incorporer des descripteurs de haut niveau dans le processus de classification, tels que les descripteurs de forme ou ceux issus de l'apprentissage profond, pour analyser des images à large échelle et sur des jeux de données complexes et réalistes.

R.2 Contributions

Dans les paragraphes qui suivent, nous résumons nos contributions ainsi que leur organisation dans ce manuscrit.

Chapitre 2 Il est bien connu que l'ajout de descripteurs de forme (tels que la convexité ou la rectangularité) contribue à améliorer la classification [31]. Cependant, il est difficile d'optimiser des énergies qui comprennent des a priori de forme. Nous proposons une méthode pour la classification des images qui considère l'occurrence de multiples objets dans la même scène, ainsi que de multiples classes, en prenant compte aussi d'un ensemble de descripteurs de forme. Nous nous basons sur une structure de données hiérarchique appelée arbre binaire de partition [126]. L'usage typique de ces arbres ne permet pas d'inclure des contraintes de forme. Nous proposons donc un algorithme d'optimisation qui taille et greffe des branches de l'arbre pour incorporer l'information de la forme typique des objets. Les distributions du spectre et de forme sont apprises à partir de données d'entrainement.

Chapitre 3 Nous étudions l'usage des réseaux de neurones convolutionels (CNN) [76] pour la classification, étant donné qu'ils sont capables d'apprendre des descripteurs contextuels complexes à partir de données d'entrainement et, cela, avec une capacité remarquable de passer à l'échelle. Nous étudions d'abord le prototype d'architecture le plus approprié pour le problème de la classification pixel à pixel, et nous proposons l'utilisation des réseaux complètement convolutionnels [93] comme architecture de base. Nous discutons aussi l'obtention de données d'entrainement ainsi qu'une façon de gérer leurs imperfections : nous utilisons de larges bases de données imparfaites issues du *crowd-sourcing* pour entrainer un réseau, puis nous raffinons les paramètres dans une deuxième phase avec de faibles quantités de données labellisées à la main. Nous développons ainsi un premier cadre méthodologique pour la classification des images de télédétection avec des CNN.

Chapitre 4 Un des principaux problèmes auquel nous devons faire face pour classifier des pixels utilisant des CNN, c'est le niveau de détail de la sortie, qui semble souvent très floue. Plusieurs méthodes itératives de post-traitement ont été proposées pour rendre les cartes de classification plus nettes (ex. [21]). Au lieu de concevoir ce type d'algorithme à la main, nous étudions ici l'usage d'un réseau de neurones récurrent [61] pour directement apprendre un processus itératif qui fait cette amélioration. Le réseau apprend donc un algorithme itératif à partir des données d'entrainement.

Chapitre 5 Une alternative naturelle au post-traitement consiste à concevoir des architectures de réseau spécifiques qui fournissent en sortie des cartes de classification à haute résolution spatiale. Nous analysons d'abord en profondeur les architectures qui ont été récemment présentées dans la litérature (ex. [8, 155]). Nous proposons ensuite un réseau qui apprend à combiner des descripteurs provenant de différentes résolutions, permettant ainsi de générer des cartes de classification fines. En outre, nous créons une base de données à large échelle avec des images prises sur des régions très différentes, pour ainsi évaluer la capacité de généralisation des réseaux de neurones. Sur cette base de données, les réseaux doivent apprendre en se servant de données sur un certain nombres de villes, tant qu'on évalue la performance des méthodes sur des villes complètement différentes, méconnues du réseau.

Chapitre 6 Dans plusieurs domaines, il est nécessaire de polygonaliser les cartes de classification, en vue de les intégrer dans des systèmes d'information géographique. Pour ce faire, il est fréquent de polygonaliser les rasters de façon naïve, puis de

les simplifier avec des algorithmes tels que Douglas-Peucker [37]. Nous proposons une méthode de polygonalisation basée sur la simplification de maillages, qui nous permet de mieux représenter les objets et de façon efficace. La méthode consiste en une minimisation d'énergie, calculée de façon intégrale sur la surface de l'image.

R.3 Conclusions et perspectives

L'objectif premier de cette thèse a été d'explorer le problème de la classification des images de télédétection. Pour ce faire, il faut ajouter un raisonnement sémantique de haut niveau, car il n'est pas suffisant de juste analyser les valeurs individuelles du spectre mesurées par les capteurs. Les technologies actuelles ainsi que les applications qui utilisent la télédétection nécessitent des méthodes qui passent à l'échelle, pour traiter des images sur de larges étendues de terre, tout en étant robuste à la variabilité.

Dans une première approche, nous avons proposé une méthode pour la classification avec des contraintes de forme. À la différence d'autres méthodes, nous considérons l'occurrence de multiples objets d'une même classe dans la scène. La méthode proposée se base sur les arbres de partition, lesquels sont construits avec un algorithme glouton, suivi d'une étape d'optimisation.

Les chapitres suivants étudient les réseaux de neurones convolutionnels (CNN). Si on les compare avec les arbres de partition, on voit bien qu'on ajoute de la « scalabilité » en termes des descripteurs, ainsi qu'une haute performance en temps de calcul. Les CNN étant assez récents, il reste du travail à faire pour bien les adapter au problème de la classification pixel à pixel. Nous abordons plusieurs parmi ces sujets dans le Chapitre 3, qui dans l'ensemble propose un premier *framework* de classification avec des CNN.

Pour produire une carte de classification à haute résolution, plusieurs méthodes récentes proposent l'inclusion d'un module de post-traitement. Nous avons, dans le Chapitre 4, étudié l'usage d'un réseau récurrent pour découvrir cet algorithme à partir des données d'entrainement, au lieu de le faire à la main. Cela constitue une nouvelle façon d'utiliser les réseaux de neurones, soulignant encore leur potentiel.

Même si le post-traitement est une solution possible, nous considérons qu'il est nécessaire de proposer des réseaux de neurones qui, à la base, génèrent des cartes à haute résolution. Pour cela, nous avons d'abord analysé la bibliographie récente dans le sujet (Chapitre 5) et proposé un réseau adapté à ce problème. Nous avons fait des expériences sur les données d'ISPRS [70] et soumis une contribution à la compétition. En outre, nous avons créé un jeu de données qui montre que les réseaux peuvent généraliser à des images présentant des conditions assez différentes.

Une des principales conclusions que l'on peut faire à partir de notre travail avec des CNNs, c'est que la conception d'une bonne architecture peut améliorer les résultats de façon significative, ainsi que les temps de calcul. Ce résultat est obtenu en imposant des restrictions sensées aux connections neuronales. Ceci est notamment le cas des réseaux complètement convolutionnels (Chapitre 3) et du réseau MLP que nous avons proposé (Chapitre 5).

Dans la dernière partie de cette thèse (Chapitre 6), nous avons abordé un autre sujet : la polygonalisation des cartes de classification, étant donné que plusieurs applications de la télédétection nécessitent d'intégrer les objets dans des systèmes d'information géographique en tant que polygones. Les CNN produisant parfois des cartes qui ne sont pas forcément bien structurées (ex. des coins arrondis), nous ajoutons ici de la régularité géométrique à travers la représentation sous-jacente : un maillage triangulaire.

Perspectives

L'acquisition des données d'entrainement ouvre des perspectives intéressantes. La correction automatique des données existantes issues du crowd-sourcing, telle que OpenStreetMaps (OSM) est certainement une possible direction future de recherche. Notamment, l'estimation de la largeur des routes en utilisant des données OSM a été récemment étudiée [102]. L'usage récent de réseaux récurrents pour la localisation des point sur les visages [147] pourrait aussi être utile dans notre domaine, pour corriger les polygones. La fusion des différentes sources de données, ainsi que le post-traitement effectué sur les images satellites (ex. rectification et *pansharpening*) pourrait peut-être eux aussi être améliorés à travers l'apprentissage machine.

Une autre possible direction de recherche est l'inclusion de régularité géométrique dans les algorithmes de polygonalisation (ex. parallélisme et co-angularités). On pourrait aussi penser à générer des données géométriques directement des réseaux de neurones. Par exemple, un réseau récurrent a été récemment utilisé pour le problème du calcul de l'enveloppe convexe d'un nuage de points [153]. Comme alternative, on pourrait aussi penser à utiliser des réseaux « adversariaux » pour encourager un CNN à produire des cartes si vraisemblables qu'elles seront confondues par un autre classifieur.

Bibliography

[1] Abdullah Al-Dujaili, François Merciol, and Sébastien Lefèvre. Graphbpt: An efficient hierarchical data structure for image representation and probabilistic inference. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 301–312. Springer, 2015.

[2] Pierre Alliez, Nathalie Laurent, Henri Sanson, and Francis Schmitt. Mesh approximation using a volume-based metric. In *Proceedings of Seventh Pacific Conference on Computer Graphics and Applications*, pages 292–301. IEEE, 1999.

[3] Luciano Alparone, Lucien Wald, Jocelyn Chanussot, Claire Thomas, Paolo Gamba, and Lori Mann Bruce. Comparison of pansharpening algorithms: Outcome of the 2006 GRS-S data-fusion contest. *IEEE Transactions on Geoscience and Remote Sensing*, 45(10):3012–3021, 2007.

[4] Israa Amro, Javier Mateos, Miguel Vega, Rafael Molina, and Aggelos K Katsaggelos. A survey of classical methods and new trends in pansharpening of multispectral images. *EURASIP Journal on Advances in Signal Processing*, 2011(1):79, 2011.

[5] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefevre. How useful is region-based classification of remote sensing images in a deep learning framework? In *IEEE Geoscience and Remote Sensing Symposium (IGARSS)*, pages 5091–5094, 2016.

[6] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefèvre. Joint learning from earth observation and openstreetmap data to get faster better semantic maps. In *EarthVision - CVPR Workshop*, 2017.

[7] Mohammad Awrangjeb, Mehdi Ravanbakhsh, and Clive S Fraser. Automatic detection of residential buildings using lidar data and multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(5):457–467, 2010.

[8] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[9] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[10] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. M. Nasrabadi, and J. Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geosci. and Remote Sens. Mag.*, 1(2):6–36, 2013.

[11] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[12] Thomas Blaschke. Object based image analysis for remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 65(1):2–16, 2010.

[13] J. L. Boggs, T. D. Tsegaye, T. L. Coleman, K. C. Reddy, and A. Fahsi. Relationship between hyperspectral reflectance, soil nitrate-nitrogen, cotton leaf chlorophyll, and cotton yield: a step toward precision agriculture. *Journal of Sustainable Agriculture*, 22(3):5–16, 2003.

[14] Frank J Bossen and Paul S Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–74, 1996.

[15] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[16] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *ICML*, pages 111–118, 2010.

[17] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[18] Felipe Calderero and Ferran Marques. Region merging techniques using information theory statistical measures. *IEEE Trans. Image Process.*, 19(6):1567–1586, 2010.

[19] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

[20] Liang-Chieh Chen, Jonathan T Barron, George Papandreou, Kevin Murphy, and Alan Yuille. Semantic image segmentation with task-specific edge detection using CNNs and a discriminatively trained domain transform. *arXiv preprint arXiv:1511.03328*, 2015.

[21] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *ICLR*, May 2015.

[22] Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan. Vehicle detection in satellite images by hybrid deep convolutional neural networks. *IEEE Geosci. Remote Sens. Lett.*, 11(10):1797–1801, 2014.

[23] Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *IEEE CVPR*, pages 5261–5269, 2015.

[24] Yushi Chen, Zhouhan Lin, Xing Zhao, Gang Wang, and Yanfeng Gu. Deep learning-based classification of hyperspectral data. *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, 7(6):2094–2107, 2014.

[25] Yushi Chen, Xing Zhao, and Xiuping Jia. Spectral-spatial classification of hyperspectral data based on deep belief network. *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, 8(6):2381–2392, June 2015.

[26] D. Chutia, D.K. Bhattacharyya, K.K Sarma, R. Kalita, and S. Sudhakar. Hyperspectral remote sensing classifications: a perspective survey. *Transactions in GIS*, 2015.

[27] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE CVPR*, pages 3642–3649, 2012.

[28] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[29] Edward A. Cloutis. Hyperspectral geological remote sensing: evaluation of analytical techniques. *International Journal of Remote Sensing*, 17(12):2215–2242, 1996.

[30] Daniel Cremers, Stanley J Osher, and Stefano Soatto. Kernel density estimation and intrinsic alignment for shape priors in level set segmentation. *IJCV*, 69(3):335–351, 2006.

[31] Daniel Cremers, Florian Tischhäuser, Joachim Weickert, and Christoph Schnörr. Diffusion snakes: Introducing statistical shape knowledge into the Mumford-Shah functional. *IJCV*, 50(3):295–313, 2002.

[32] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, 2013.

[33] Piali Das, Olga Veksler, Vyacheslav Zavadsky, and Yuri Boykov. Semiautomatic segmentation with compact shape prior. *Image and Vision Computing*, 27(1):206–219, 2009.

[34] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[35] Stig Descamps, Xavier Descombes, Arnaud Béchet, and Josiane Zerubia. Automatic flamingo detection using a multiple birth and death process. In *IEEE ICASSP*, pages 1113–1116, 2008.

[36] Lee Dice. Measure of the amount of ecological association between species. *Ecology*, 26(3):297–302, 1945.

[37] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

[38] Anastasia Dubrovina, Pavel Kisilev, Boris Ginsburg, Sharbell Hashoul, and Ron Kimmel. Computational mammography using deep neural networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, pages 1–5, 2016.

[39] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011.

[40] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *IEEE CVPR*, pages 2147–2154, 2014.

[41] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[42] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1915–1929, 2013.

[43] Mathieu Fauvel, Yuliya Tarabalka, Jon Atli Benediktsson, Jocelyn Chanussot, and James C Tilton. Advances in spectral-spatial classification of hyperspectral images. *Proc. of the IEEE*, 101(3):652–675, 2013.

[44] Pedro F Felzenszwalb. Representation and detection of deformable shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(2):208–220, 2005.

[45] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[46] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *ICANN*, pages 220–229. Springer, 2007.

[47] Cèsar Ferri, José Hernández-Orallo, and Peter A Flach. A coherent interpretation of auc as a measure of aggregated classification performance. In *ICML*, pages 657–664, 2011.

[48] Daniel Freedman and Tao Zhang. Interactive graph cut based segmentation with shape priors. In *IEEE CVPR*, pages 755–762, 2005.

[49] Gareth Funka-Lea, Yuri Boykov, Charles Florin, M-P Jolly, Romain Moreau-Gobard, Rana Ramaraj, and Daniel Rinck. Automatic heart isolation for ct coronary visualization using graph-cuts. In *IEEE Int. Symp. Biomedical Imaging: Nano to Macro*, pages 614–617, 2006.

[50] Martin Galanda. *Automated polygon generalization in a multi agent system.* PhD thesis, Zurich: Zurich University, 2003.

[51] Eduardo S. L. Gastal and Manuel M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4):69:1–69:12, 2011.

[52] Markus Gerke. Use of the stair vision library within the ISPRS 2d semantic labeling benchmark (vaihingen). Technical report, University of Twente, 2015.

[53] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[54] A. Ghiyamat and H. Z. M. Shafri. A review on hyperspectral remote sensing for homogeneous and heterogeneous forest biodiversity assessment. *International Journal of Remote Sensing*, 31(7):1837–1856, 2010.

[55] Gonzalo Giribet. Efficient tree searches with available algorithms. *Evolutionary bioinformatics online*, 3:341, 2007.

[56] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[57] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *JMLR*, 15(106):275, 2011.

[58] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[59] Lena Gorelick, Frank R. Schmidt, and Yuri Boykov. Fast trust region for segmentation. In *IEEE CVPR*, pages 1714–1721, June 2013.

[60] Lena Gorelick, Olga Veksler, Yuri Boykov, and Claudia Nieuwenhuis. Convexity shape prior for segmentation. In *ECCV*, pages 675–690, 2014.

[61] Alex Graves. Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer, 2012.

[62] Arif Günay, Hossein Arefi, and Michael Hahn. True orthophoto production using lidar data. In *ISPRS Joint Workshop Visualization and Exploration of Geospatial Data*, volume 36, page 4, 2007.

[63] Bharath Hariharan, Pablo Arbelaez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE CVPR*, pages 1026–1034, 2015.

[65] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[66] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[67] Rui Huang, Vladimir Pavlovic, and Dimitris N Metaxas. A graphical model framework for coupling MRFs and deformable models. In *IEEE CVPR*, pages 739–746, 2004.

[68] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Dover Publications, 2012.

[69] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[70] ISPRS. 2D semantic labeling contest. http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html.

[71] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[72] Pascal Kaiser. Learning city structures from online maps. Master's thesis, ETH Zurich, 2016.

[73] Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *IEEE CVPR Workshops*, 2016.

[74] K. Karantzalos and N. Paragios. Recognition-driven two-dimensional competing priors toward automatic and accurate building detection. *IEEE TGRS*, 47(1):133–144, Jan 2009.

[75] Nirmal Keshava. A survey of spectral unmixing algorithms. *Lincoln Lab. J.*, 14:55–78, 2003.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[77] M Pawan Kumar, Philip H.S. Torr, and Andrew Zisserman. Obj cut. In *IEEE CVPR*, pages 18–25, 2005.

[78] Camille Kurtz, Nicolas Passat, Pierre Gancarski, and Anne Puissant. Extraction of complex patterns from multiresolution remote sensing images: A hierarchical top-down methodology. *Pattern Recognition*, 45(2):685–706, 2012.

[79] Yu-Kun Lai, Shi-Min Hu, and Ralph R Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. on Graphics*, 28(3):85, 2009.

[80] Pierre Lassalle, Jordi Inglada, Julien Michel, Manuel Grizonnet, and Julien Malik. A scalable tile-based framework for region-merging segmentation. *IEEE Tran. Geosci. Remote Sens.*, 53(10):5473–5485, 2015.

[81] Gregory Lecot and Bruno Levy. Ardeco: automatic region detection and conversion. In *17th Eurographics Symposium on Rendering*, pages 349–360, 2006.

[82] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[84] Sebastien Lefevre, Laetitia Chapel, and François Merciol. Hyperspectral image classification from multiscale description with constrained connectivity and metric learning. In *6th International Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2014.

[85] Victor Lempitsky, Andrew Blake, and Carsten Rother. Image segmentation by branch-and-mincut. In *ECCV*, pages 15–29, 2008.

[86] Sébastien Leprince, Sylvain Barbot, Franois Ayoub, and Jean-Philippe Avouac. Automatic and precise orthorectification, coregistration, and subpixel correlation of satellite images, application to ground deformation measurements. *IEEE Transactions on Geoscience and Remote Sensing*, 45(6):1529–1558, 2007.

[87] Michael E Leventon, W Eric L Grimson, and Olivier Faugeras. Statistical shape influence in geodesic active contours. In *IEEE CVPR*, pages 316–323, 2000.

[88] Fulu Li and Andrew Lippman. Random tree optimization for the construction of the most parsimonious phylogenetic trees. In *CISS*, pages 757–762, 2009.

[89] Tong Li, Junping Zhang, and Ye Zhang. Classification of hyperspectral image based on deep belief networks. In *IEEE ICIP*, pages 5132–5136, 2014.

[90] Wenwen Li, Michael F Goodchild, and Richard Church. An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems. *Int. Journal of Geographical Information Science*, 27(6):1227–1250, 2013.

[91] Risheng Liu, Zhouchen Lin, Wei Zhang, and Zhixun Su. Learning PDEs for image restoration via optimal control. In *ECCV*, pages 115–128. Springer, 2010.

[92] Risheng Liu, Zhouchen Lin, Wei Zhang, Kewei Tang, and Zhixun Su. Toward designing intelligent pdes for computer vision: An optimal control approach. *Image and vision computing*, 31(1):43–56, 2013.

[93] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE CVPR*, 2015.

[94] Huihai Lu, John C Woods, and Mohammed Ghanbari. Binary partition tree for semantic object extraction and image segmentation. *IEEE Trans. Circuits and Systems for Video Technology*, 17(3):378–383, 2007.

[95] Francois Luus, Brian Salmon, F Van Den Bergh, and BT Maharaj. Multiview deep learning for land-use classification. *IEEE Geosci. Remote Sens. Lett.*, 12(12):2448–2452, 2015.

[96] Emmanuel Maggiori, Yuliya Tarabalka, and Guillaume Charpiat. Improved partition trees for multi-class segmentation of remote sensing images. In *IEEE IGARSS*, pages 1016–1019, 2015.

[97] Emmanuel Maggiori, Yuliya Tarabalka, and Guillaume Charpiat. Optimizing partition trees for multi-object segmentation with shape prior. In *26th British Machine Vision Conference (BMVC)*, 2015.

[98] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Fully convolutional neural networks for remote sensing image classification. In *IEEE IGARSS*, pages 5071–5074, 2016.

[99] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Trans. Geosci. Remote Sens.*, 55(2):645–657, 2017.

[100] Konstantinos Makantasis, Konstantinos Karantzalos, Anastasios Doulamis, and Nikolaos Doulamis. Deep supervised learning for hyperspectral data classification through convolutional neural networks. In *IEEE IGARSS*, pages 4959–4962. IEEE, 2015.

[101] Dimitrios Marmanis, Jan D Wegner, Silvano Gallianib, Konrad Schindler, Mihai Datcu, and Uwe Stilla. Semantic segmentation of aerial images with an ensemble of cnns. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 473–480, 2016.

[102] Gellert Mattyus, Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Enhancing road maps by parsing aerial images around the world. In *IEEE ICCV*, 2015.

[103] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[104] Julien Michel, David Youssefi, and Manuel Grizonnet. Stable mean-shift algorithm and its application to the segmentation of arbitrarily large remote sensing images. *IEEE Tran. Geosci. Remote Sens.*, 53(2):952–964, 2015.

[105] ME Midhun, Sarath R Nair, VT Prabhakar, and S Sachin Kumar. Deep model for classification of hyperspectral image using restricted boltzmann machine. In *International Conference on Interdisciplinary Advances in Applied Computing*, page 35. ACM, 2014.

[106] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[107] Volodymyr Mnih. *Machine learning for aerial image labeling*. PhD thesis, University of Toronto, 2013.

[108] Volodymyr Mnih and Geoffrey E Hinton. Learning to detect roads in high-resolution aerial images. In *ECCV*, pages 210–223. Springer, 2010.

[109] Raul S Montero and Ernesto Bribiesca. State of the art of compactness and circularity measures. *Int. Mathematical Forum*, 4(27):1305–1335, 2009.

[110] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[111] Claudia Nieuwenhuis, Eno Töppe, and Daniel Cremers. A survey and comparison of discrete and continuous multi-label optimization approaches for the Potts model. *IJCV*, 104(3):223–240, 2013.

[112] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE CVPR*, pages 1520–1528, 2015.

[113] Mathias Ortner, Xavier Descombes, and Josiane Zerubia. Building outline extraction from digital elevation models using marked point processes. *IJCV*, 72(2):107–132, 2007.

[114] Sakrapee Paisitkriangkrai, Jamie Sherrah, Pranam Janney, Van-Den Hengel, et al. Effective semantic pixel labelling with convolutional networks and conditional random fields. In *IEEE CVPR Workshops*, 2015.

[115] Guillem Palou and Philippe Salembier. Occlusion-based depth ordering on monocular images with binary partition tree. In *IEEE ICASSP*, pages 1093–1096, 2011.

[116] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML*, 28:1310–1318, 2013.

[117] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[118] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.

[119] Brian Price and William Barrett. Object-based vectorization for interactive image editing. *The Visual Computer*, 22(9):661–670, 2006.

[120] K Reumann and APM Witkam. Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Symposium*, pages 467–472, 1974.

[121] David S Richeson. *Euler's gem: the polyhedron formula and the birth of topology*. Princeton University Press, 2012.

[122] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[123] Alan Saalfeld. Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999.

[124] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech*, pages 338–342, 2014.

[125] Philippe Salembier, Samuel Foucher, and Carlos López-Martínez. Low-level processing of PolSAR images with binary partition trees. In *IEEE IGARSS*, 2014.

[126] Philippe Salembier and Luis Garrido. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *IEEE Trans. Image Process.*, 9(4):561–576, 2000.

[127] Philippe Salembier, Albert Oliveras, and Luis Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, 1998.

[128] Thomas Schoenemann and Daniel Cremers. Globally optimal image segmentation with an elastic shape prior. In *ICCV*, pages 1–6, 2007.

[129] Robert A Schowengerdt. *Remote sensing: models and methods for image processing*. Academic press, 2006.

[130] Igor Ševo and Aleksej Avramović. Convolutional neural network based automatic object detection on aerial images. *IEEE Geosci. Remote Sens. Lett.*, 13(5):740–744, 2016.

[131] Jamie Sherrah. Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery. *arXiv preprint arXiv:1606.02585*, 2016.

[132] Wenzhong Shi and ChuiKwan Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44, 2006.

[133] Bernard W Silverman. *Density estimation for statistics and data analysis*. CRC press, 1986.

[134] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to simplify: fully convolutional networks for rough sketch cleanup. *ACM Trans. on Graphics*, 35(4):121, 2016.

[135] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014.

[136] Ali Kemal Sinop and Leo Grady. Uninitialized, globally optimal, graph-based rectilinear shape segmentation-the opposing metrics method. In *ICCV*, pages 1–8, 2007.

[137] Greg Slabaugh and Gozde Unal. Graph cuts segmentation using an elliptical shape prior. In *IEEE ICIP*, pages 1222–1225, 2005.

[138] Viktor Slavkovikj, Steven Verstockt, Wesley De Neve, Sofie Van Hoecke, and Rik Van de Walle. Hyperspectral image classification with convolutional neural networks. In *International Conference on Multimedia*, pages 1159–1162. ACM, 2015.

[139] Pierre Soille. Constrained connectivity for hierarchical image partitioning and simplification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1132–1145, 2008.

[140] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[141] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[142] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. on Graphics*, 26(3):11, 2007.

[143] Y. Tarabalka and A. Rana. Graph-cut-based model for spectral-spatial classification of hyperspectral images. In *IEEE IGARSS*, pages 3418–3421. IEEE, 2014.

[144] Yuliya Tarabalka and James C Tilton. Improved hierarchical optimization-based classification of hyperspectral images using shape analysis. In *IEEE IGARSS*, pages 1409–1412, 2012.

[145] James Tilton and Edoardo Pasolli. Incorporating edge information into best merge region-growing segmentation. In *IEEE IGARSS*, 2014.

[146] Godfried T Toussaint. The rotating calipers: An efficient, multipurpose, computational tool. In *ICCTIM*, pages 215–225, 2014.

[147] George Trigeorgis, Patrick Snape, Mihalis A Nicolaou, Epameinondas Antonakos, and Stefanos Zafeiriou. Mnemonic descent method: A recurrent process applied for end-to-end face alignment. In *IEEE CVPR*, pages 4177–4187, 2016.

[148] S. Valero, P. Salembier, and J. Chanussot. Hyperspectral image representation and processing with binary partition trees. *IEEE Trans. Image Process.*, 22(4):1430–1443, 2013.

[149] Silvia Valero, Philippe Salembier, and Jocelyn Chanussot. Object recognition in urban hyperspectral images using binary partition tree representation. In *IGARSS*, pages 4098–4101, 2013.

[150] Olga Veksler. Star shape prior for graph-cut image segmentation. In *ECCV*, pages 454–467, 2008.

[151] Antonio Wilson Vieira, Luiz Velho, Hélio Lopes, Geovan Tavares, and Thomas Lewiner. Fast stellar mesh simplification. In *SIBGRAPI*, pages 27–34. IEEE, 2003.

[152] Veronica Vilaplana, Ferran Marques, and Philippe Salembier. Binary partition trees for object detection. *IEEE Trans. Image Process.*, 17(11):2201–2216, 2008.

[153] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, pages 2692–2700, 2015.

[154] M Visvalingam and JD Whyatt. Line generalisation by repeated elimination of the smallest area. *The Cartographic Journal*, 30(1):46–51, 1992.

[155] Michele Volpi and Devis Tuia. Dense semantic labeling of sub-decimeter resolution images with convolutional neural networks. *IEEE Tran. Geosci. Remote Sens.*, 2016.

[156] Volker Walter. Object-based classification of remote sensing data for change detection. *ISPRS Journal of photogrammetry and remote sensing*, 58(3):225–238, 2004.

[157] Jun Wang, Jingwei Song, Mingquan Chen, and Zhi Yang. Road network extraction: a neural-dynamic framework based on deep learning and a finite state machine. *International Journal of Remote Sensing*, 36(12):3144–3169, 2015.

[158] Joachim Weickert. *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998.

[159] Paul Werbos. Backpropagation through time: what it does and how to do it. *Proc. of the IEEE*, 78(10):1550–1560, 1990.

[160] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[161] Shin-Ting Wu and Mercedes Rocío González Márquez. A non-self-intersection douglas-peucker algorithm. In *SIBGRAPI*, pages 60–66. IEEE, 2003.

[162] Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005, 2004.

[163] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. on Graphics*, 28(5):115, 2009.

[164] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, and Ming-Hsuan Yang. Object contour detection with a fully convolutional encoder-decoder network. *arXiv preprint arXiv:1603.04530*, 2016.

[165] Mingqiang Yang, Kidiyo Kpalma, Joseph Ronsin, et al. A survey of shape feature extraction techniques. *Pattern recognition*, pages 43–90, 2008.

[166] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

[167] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[168] Jun Yue, Wenzhi Zhao, Shanjun Mao, and Hui Liu. Spectral–spatial classification of hyperspectral images using deep convolutional neural networks. *Remote Sensing Letters*, 6(6):468–477, 2015.

[169] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*. Springer, 2014.

[170] Wenzhi Zhao and Shihong Du. Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach. *IEEE Tran. Geosci. Remote Sens.*, 54(8):4544–4554, 2016.

[171] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr. Conditional random fields as recurrent neural networks. In *IEEE CVPR*, pages 1529–1537, 2015.