Dense Semantic Labeling of Subdecimeter Resolution Images With Convolutional Neural Networks

Michele Volpi, *Member, IEEE*, and Devis Tuia, *Senior Member, IEEE*

***Abstract*— Semantic labeling (or pixel-level land-cover classification) in ultrahigh-resolution imagery (<10 cm) requires statistical models able to learn high-level concepts from spatial data, with large appearance variations. Convolutional neural networks (CNNs) achieve this goal by learning discriminatively a hierarchy of representations of increasing abstraction. In this paper, we present a CNN-based system relying on a downsample-then-upsample architecture. Specifically, it first learns a rough spatial map of high-level representations by means of convolutions and then learns to upsample them back to the original resolution by deconvolutions. By doing so, the CNN learns to densely label every pixel at the original resolution of the image. This results in many advantages, including: 1) the state-of-the-art numerical accuracy; 2) the improved geometric accuracy of predictions; and 3) high efficiency at inference time. We test the proposed system on the Vaihingen and Potsdam subdecimeter resolution data sets, involving the semantic labeling of aerial images of 9- and 5-cm resolution, respectively. These data sets are composed by many large and fully annotated tiles, allowing an unbiased evaluation of models making use of spatial information. We do so by comparing two standard CNN architectures with the proposed one: standard patch classification, prediction of local label patches by employing only convolutions, and full patch labeling by employing deconvolutions. All the systems compare favorably or outperform a state-of-the-art baseline relying on superpixels and powerful appearance descriptors. The proposed full patch labeling CNN outperforms these models by a large margin, also showing a very appealing inference time.**

***Index Terms*— Aerial images, classification, convolutional neural networks (CNNs), deconvolution networks, deep learning, semantic labeling, subdecimeter resolution.**

I. Introduction

SEMANTIC labeling is the task of assigning a semantic label (land-cover or land-use class) to every pixel of an image. When processing ultrahigh resolution data, most of the state-of-the-art methods rely on supervised classifiers trained on specifically hand-crafted feature sets (appearance descriptors), describing locally the image content. The extracted high-dimensional representation is assumed to contain enough information to cope with the ambiguities caused by the limited spectral information of the ultrahigh-resolution sensors.

In the pipeline described earlier, input images undergo a spatial feature extraction step implemented by specific operators on local portions of the image (patches, superpixels or regions, objects, and so on), so that particular *spatial* arrangements of colors are encoded into a high-dimensional representation. A supervised classifier is usually employed to learn a mapping from the appearance descriptors to the semantic label space, which in turn allows to assign a label to every region of a previously unseen test image. Common examples of spatial features are texture statistics, mathematical morphology, and oriented gradients [1]. Other common approach strategies rely on bag-of-visual-words [2]. This midlevel representation is based on a quantization of appearance descriptors, such as gradients, orientations, and texture (usually obtained with a clustering algorithm). This quantization is then pooled spatially into the histograms of spatial occurrences of cluster labels, or bag-of-visual-words. For instance, in [3], the bag-of-visual-words are used to classify image tiles and detect compound structures.

The drawback of these approaches is that the features depend on a specific (set of) feature extraction method, whose performance on the specific data is *a priori* unknown. Moreover, most appearance descriptors depend on a set of free parameters, which are commonly set by user experience via experimental trial-and-error or cross validation [1], [4]. Exhaustive and global optimization of such values is unfeasible in reasonable time, but the selection from random feature ensembles has shown to be an effective proxy [5]–[7]. In these cases, the filter families from which to choose from are predefined and the parameters of the system are selected heuristically by random search to minimize the error over the semantic labeling task. Although the selection of features is data-driven, the filters themselves are still not learned end-to-end from the data, thus potentially suboptimal.

Deep learning deals with the development of systems trainable in an end-to-end fashion. End-to-end usually means learning jointly a series of feature extraction from raw input data to a final, task-specific output. All deep learning systems implicitly learn representations optimizing the loss on the top of the network, driving the training of the model's weights. They usually minimize a task-specific differentiable

Manuscript received March 30, 2016; revised July 29, 2016 and September 7, 2016; accepted October 9, 2016. Date of publication October 28, 2016; date of current version December 29, 2016. This work was supported by the Swiss National Science Foundation through the Multimodal Machine Learning for Remote Sensing Information Fusion under Grant 150593.

The authors are with the MultiModal Remote Sensing Group, University of Zurich, 8057 Zürich, Switzerland (e-mail: michele.volpi@geo.uzh.ch; devis.tuia@geo.uzh.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TGRS.2016.2616585

loss function for classification, regression, semantic labeling, superresolution, or depth estimation, and the network learns representations, which minimize such loss. Most common deep learning algorithms are (stacked) autoencoders [8]–[10], restricted Boltzmann machines [11], [12], and deep belief networks [13], [14]. For a review of main approaches, we refer to [14] and [15]. In this paper, we focus on convolutional neural networks (CNNs) [16]. Differently from other approaches, the CNNs were specifically designed for image classification tasks, i.e., assigning a single class label to an entire image/scene. Representations are obtained by learning a hierarchy of convolution filters from the raw image. All the weights of the convolutions are learned end-to-end to minimize the classification error of the model.

The CNNs have become extremely successful in many modern, high-level, computer vision tasks, ranging from image classification to object detection, depth estimation, and semantic labeling. First examples of deep CNN architectures have been proposed for image classification problems. The most notable example is the Image large-scale visual recognition challenge (ILSVRC) in 2012,[1] where CNNs significantly outperformed the state-of-the-art systems based on hand-crafted appearance descriptors [17]. Notable extensions allowing to train deeper CNN (i.e., adding trainable layers, and thus increasing the capacity of the model) [18]–[20] were the introduction of drop-out [21], batch normalization [22], and other strategies allowing better propagation of gradients, such as rectified linear units (ReLUs) nonlinearities [20], [23]. Together with the (very) large annotated training data sets and powerful GPU making it possible to train such models, these intuitions made the CNNs the gold standard for image classification problems.

The CNNs have also been adapted for semantic labeling (semantic segmentation) problems [24]–[27]. These papers show two distinct approaches. In the first case, the model is trained to predict a single class label for each region (patch, superpixel, or object proposal). The output is usually a vector of scores or probabilities for each class, based on the appearance of an entire region. In the second case, the network is trained to predict *spatial arrangements* of labels at pixel level. These architectures are able to model local structures (e.g., spatial extent and class co-occurrences) across the input space. These *upsampling* steps are formulated by means of *deconvolutions* [26], [27]. In this paper, we adopt this approach and propose a strategy to learn locally dense semantic labeling of patches.

A. Deep Learning in Remote Sensing

Remote sensing image processing pipelines are beginning to exploit deep learning. Initially, such systems tackled the problem of image/tile classification: i.e., assigning to large patches (or small images) a single semantic label, such as "urban," "sparse urban," or "forest." This task is well represented by the UC Merced land-use classification data set.[2] Marmanis *et al.* [28] present a two-stages system, in which pretrained CNNs are combined with a second stage of supervised training. This strategy mitigates overfitting and shows excellent performances. Penatti *et al.* [29] show that models pretrained on general image classification tasks (specifically on ILSVRC) can be used as generic feature extractors also for remote sensing image (tile) classification, outperforming most of the state-of-the-art feature descriptors. Authors point out that such models were particularly well suited for high resolution aerial data. Finally, Castelluccio *et al.* [30] also explore pretrained architectures. They also study the impact of domain adaptation by fine tuning the model to the new task (from ILSVRC data to remote sensing images) or by training from scratch, always keeping exactly the same CNN architectures. They show that pretrained models fine-tuned on remote sensing data perform better than models with a same architecture but trained from scratch (with randomly initialized weights). This indicates that CNN architectures devoted to natural image classification tasks without specific adjustments tend to overfit remote sensing data. Typically, to model complex appearance variations, CNNs devoted for image classification contain many parameters, particularly in the fully connected layers, which could easily contain more than 90% (VGG network [18]) or 95% (AlexNet [17]) of the total number of learnable parameters. With the typical remote sensing data sets such large number of parameter would be hard to estimate, because of the limited amount of labeled samples they generally provide. For this reason, training networks for remote sensing problems requires adjustments.

Recently, semantic labeling tasks in remotely sensed data were also approached by means of deep learning. Chen *et al.* [31] rely on stacked autoencoders, trained to reconstruct PCA-compressed hyperspectral signals. The network is then fine-tuned by backpropagating errors from a softmax loss on top of the stacked autoencoders, which also provides the final classification of the pixels. Firat *et al.* [32] train a sparse convolutional autoencoder to perform object detection in remote sensing images. Although the model is only composed of a (wide) single layer and technically is not "deep," the idea of representation learning is tackled elegantly. Zhang *et al.* [33] propose a system based on stacked autoencoders: rather than training the model on random patches from the images, they follow a strategy sampling patches based on visual saliency, shown to improve over the model without region selection. Romero *et al.* [34] propose a deep convolutional sparse autoencoder relying on a specific sparsity criterion. Generic features are extracted for image patches and a separate classification is then performed to label each feature vector. These works have the attractivity of working in scarcely supervised settings, by employing unsupervised (pre)training schemes. However, these models can not learn discriminative representations, which is a task usually left to a classifier trained by using such generic representations. Paisitkriangkrai *et al.* [35] propose a system based on several CNNs trained on the Vaihingen challenge data set (see Section IV-A for the description of the data set) to perform semantic labeling. The potential of the CNNs is clearly shown by combining the features extracted from the CNN with

[1] ILSVRC, http://www.image-net.org/challenges/LSVRC/

[2] http://vision.ucmerced.edu/datasets/landuse.html

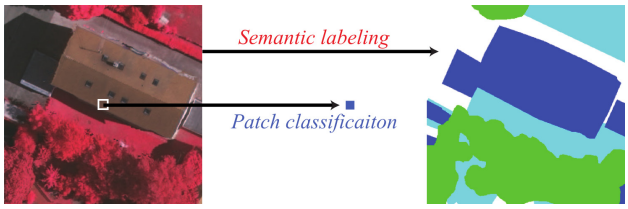

Fig. 1. Comparing patch classification and semantic labeling. The first learns a single label per patch (assumed to be the one of the central pixel), while the second learns to densely predict semantic labels for each location.

random forest classifiers, standard appearance descriptors, and conditional random fields, performing structured prediction on the probabilities given by the classifier. However, the CNN in [35] is not trained specifically for pixelwise semantic labeling tasks, but rather for patch classification: the network is designed to predict a single label from a patch (independently from the others, as in the "patch classification" part of Fig. 1). Sherrah [36] proposed a no-downsampling system relying on pretrained networks, together with a network designed to process digital surface model (DSM) data only, to predict class-conditional scores. Fused results are then postprocessed by a conditional random field. So far, these results are among the best on the benchmark data.

Training a network explicitly for semantic labeling or for classification problems is very different task, as shown in Fig. 1. In the former setting, we would like to train a model that is able to label each pixel present in the image. This should be achieved not only by learning the relationship between colors and labels, but mostly by learning and taking into account spatial relationships at different scales. In the second setting (patch classification), a network trained for classification predicts a single label per patch. This patch-level label is then assumed to be the label of the central pixel. In this setting, many queries are spatially concatenated to obtain the prediction map. In addition to being extremely inefficient, patch classification-based strategies are inappropriate, since they do not explicitly learn spatial configuration of labels and consequently oversmooth objects boundaries. To make an example, a CNN model aiming at classifying a patch centered on a car lying on a road could score high for both the classes "road" and "car": both the solutions are semantically correct, since both the classes are somehow lying in the center of the patch. Given the content and the context, it is hard to penalize one solution more than the other. In the semantic labeling case, we would like to predict the label of *each pixel* in the patch *jointly*, thus avoiding ambiguities given by the content of the patch and *de facto* performing structured prediction by learning class-specific structures contained in each patch.

In this paper, we explicitly tackle the semantic labeling problem and we show how a modern CNN-based system can be trained for dense semantic labeling tasks in a fully supervised fashion. We are particularly interested in the semantic labeling of ultrahigh spatial resolution images, where data contains a tremendous amount of geometrical information, but with a limited number of spectral channels. This is typically the case with off-the-shelf Unmanned aerial vehicle and most aerial imaging systems. We introduce a patch-based deconvolution network to first encode land-cover representations in a rough spatial map (that we name bottleneck) and then to upsample them back to the original input patch size. The modeling power of this downsampling-then-upsampling architecture relies on the fact that global spatial relationships can be modeled directly by learning locally, on a coarser spatial signal (downsampling part). The upsampling will then take into account local spatial structuring for each class while extracting nonlinear representations at the same time. Training the network patchwise allows us to deal with the images of any size, by decomposing the problem into subregions with representative spatial support. The structure of the network is given in Fig. 3. To train it, we employ standard stochastic gradient descent (SGD) with momentum, on the batches of training patches sampled from the training images. At inference time, we again take a step away from standard approaches that must crop large images into densely overlapping blocks (i.e., with a small stride) or rely on object proposals/regions) to maximally preserve the spatial resolution of predictions. In our system, the whole image can be directly fed to the trained network to obtain a posterior probability map of semantic labels *without loss in resolution*. Doing the same with standard patch-based CNNs would show a drastic loss in spatial resolution.

Differently from already published works in remote sensing, we train the network specifically for dense labeling, and not for classification. Compared with the no-downsampling extension of [36], we argue that a downsample-then-upsample architecture could make better use of contextual relationships, since without downsampling activations are location specific, and information is not explicitly shared across scales (layers) if not via learned filters. The results, however, suggest that both the strategies seem appropriate. The main difference between this paper and previous computer vision studies [26], [27] relies in the fact that our upsampling layers (deconvolution) learn spatial filter by an initial, coarser spatial map of activations (bottleneck layer). Instead, [27] performs upsampling from a single feature vector, by exactly mirroring the downsampling layers (thus not learning deconvolution filters with increasing dimensionality and consequent expressive power) and propagating pooling activations for unpooling signals. Moreover, to delineate ambiguous areas, they also make use of object proposals at test time. In [26], the activations from lower layers are combined with high-stride upsamplings, to cope oversmoothing. In our approach, the CNN directly delineates classes accurately without the need of relating different layer outputs.

Summing up, we propose to train a CNN for semantic labeling tasks by employing a dowsampling-then-upsampling architecture. We first review the main blocks of CNN architectures (summarized in Section II), which allows us to carefully review differences between the CNN we put forward and more standard strategies in Section III, where we also provide important details to set up such systems. We test our intuitions on two challenging aerial image data sets, the recently released Vaihingen and Potsdam semantic labeling challenges (presented in Section IV). Here, we also show how the proposed method compares with standard CNN approaches and we discuss its strengths with respect to the

state-of-the-art. In Section VI, we conclude this paper by summarizing main contributions.

II. Convolutional Neural Networks

The CNNs are composed by a sequential hierarchy of processing layers (Fig. 3). From the input to the final classification layer, data go through a series of trainable units. A general feed forward network can be seen as a concatenation of functions, starting from some input $\mathbf{x}$

$$g(\mathbf{x}) = g_L(g_{L-1}(g_l(g_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_l); \mathbf{w}_{L-1}); \mathbf{w}_L). \quad (1)$$

The functions g_l composing the L layers of the network g are usually linear functions, subsequently passed through nonlinearities, while weights $\mathbf{w}_l$ are learned from data. For instance, multilayer perceptrons model input–output relationships by a series of densely interconnected hidden layers composed by linear units and nonlinear activation functions. CNNs are structured in a similar way, but neurons are learnable convolutions shared at each image location.

In the following, we provide a comprehensive introduction to the structure of our CNN and present important strategies to reduce overfitting while training.

A. CNN Building Blocks

In this section, we detail five building blocks of the CNN architectures; this paper deals with convolutions, nonlinear activations, spatial pooling, deconvolutions, and loss function. In Fig. 3, the schematics of the CNNs are shown. Note that only the proposed CNN full patch labeling (CNN-FPL) makes use of the deconvolutions, while the CNN patch classification (CNN-PC) and CNN semipatch labeling (CNN-SPL) use standard blocks. In the following, we refer to the inputs and outputs to the lth layer as $\mathbf{x}^l$ and $\mathbf{x}'^l = \mathbf{x}^{l+1}$, respectively. We will simply refer to $\mathbf{x}$ and $\mathbf{x}'$, since the layer indexing is clear from the context.

a) Convolutions: The main building blocks of a CNN are convolutional layers. A convolutional layer is a set of K' filters (or neurons) with learnable parameters $\mathbf{w}$. In each neuron, parameters $\mathbf{w}$ are arranged in an array of size $M \times M \times K$ to process a K-dimensional input. For instance, in RGB data, $K = 3$ in the first convolutional layer. Therefore, K' 3-D $M \times M$ filters map to K'-dimensional activations.

We center neurons on i and j, denoting spatial coordinates relative to its input. Thus, the response for the k'th filter is

$$\mathbf{x}'_{ijk'} = \sum_{k=1}^{K} \sum_{q=1}^{M} \sum_{p=1}^{M} \mathbf{w}_{pqk} \cdot \mathbf{x}_{pqk} + b \quad (2)$$

where b is a learned bias term. Supposing that we have an input image of size $N \times L \times K$, the output of the convolutional layer is $(((N - M + 2z)/s) + 1) \times (((L - M + 2z)/s) + 1) \times K'$-dimensional, where s is the stride (the spatial interval between convolutions centers) and z is the number of zero-valued rows and columns added at the borders of the image, or *zero padding*. Zero padding the inputs is very important to control the size after convolution (e.g., to ensure an activation for each location with respect to the input).

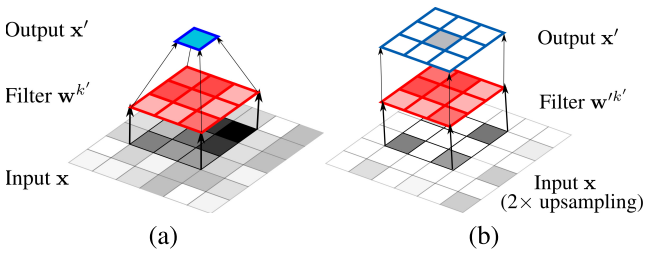

Fig. 2. Example of (a) convolution and (b) deconvolution.

Convolutional layers are not fully connected: neurons are *shared*, i.e., each filter is applied by sliding it over the whole input, without needing to learn a specific neuron per location. Response (activations) for each filter are then stacked and passed forward. An example is given in Fig. 2(a).

b) Nonlinear activations: The neural network community explored the use of many saturating nonlinearities. However, such nonlinearities can significantly slow down or even block weight convergence during training, since the gradients tend to zero when input magnitudes are large, making null or very small updates (a problem known as the *vanishing gradient* [20], [37]). For this reason, new (nonsaturating) nonlinearities have been proposed to improve gradient propagation and ultimately convergence and generalization accuracy. The most common activation employed is the ReLU, formulated as $\mathbf{x}'_{k'} = \max(0, \mathbf{x}_{k'})$ [11].

Being nonsaturated, the ReLU does not suffer from vanishing gradients. It can be computed very efficiently and it naturally sparsifies activations. However, sparsification can also be a drawback, since it can permanently "kill" a neuron. If an activation is below zero, it will never be reactivated, since the ReLU will not gate the gradient. To cope with this issue, [38] proposed a variant of the ReLU called "Leaky ReLu" (lReLU), allowing the propagation of gradients also for neurons that would have been deactivated

$$\mathbf{x}'_{k'} = [\![\mathbf{x}_{k'} \geq 0]\!] \cdot \mathbf{x}_{k'} + [\![\mathbf{x}_{k'} < 0]\!] \cdot \tau \, \mathbf{x}_{k'} \quad (3)$$

where $[\![\cdot]\!]$ is the Iverson brackets, returning 1 if the condition in the brackets is true, 0 otherwise. The scaling τ is usually a small number allowing small gradients to propagate and coping with dying neurons.

c) Spatial pooling: The spatial pooling layer has the function to summarize the signal spatially (downsamplings), preserving discriminant information. It additionally promotes translation invariance, by pooling over small windows (typically 2×2 or 3×3) into single values. Therefore, the pooling layers allow the model to recognize object instances independently on their location. In a classification setting, an image containing an airport will be labeled as "airport" independently on where the airport is spatially located within the image. The pooling layer operates on each dimension of the activations independently. Standard pooling strategies are average and the max-pooling. The former returns the average of a group of activations in the $P \times P$ window centered on ij, denoted as P_{ij} as $\mathbf{x}'_{ij} = (1/(|P_{ij}|)) \sum_{a \in P_{ij}} \mathbf{x}_a$, while the second returns the maximum value in P_{ij}, as $\mathbf{x}'_{ij} = \max_{a \in P_{ij}} \mathbf{x}_a$.

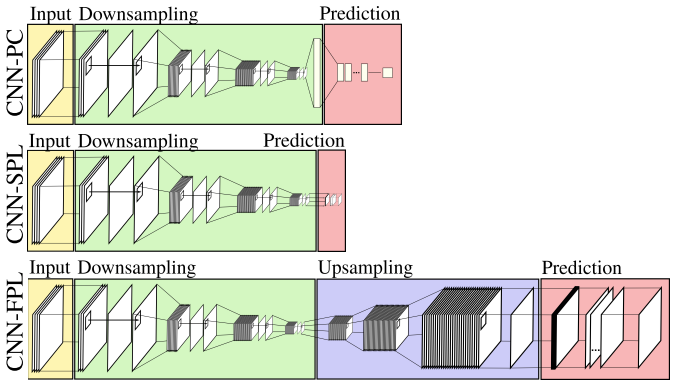

Fig. 3. Schematic of the networks used to perform semantic labeling.

It has been observed that the average pooling might not perform well, since small activations can cancel out larger ones. Max-pooling tends to perform much better, since it propagates only information pointing out the presence/absence of some particular feature. However, it might tend to overfit more easily training data. Very strong activations can control the learning of convolutions at lower layers, since backpropagation only makes the gradients flow through the maximum value occurring in each pooling window [39].

d) Deconvolutions: Deconvolution is the transposed convolution operator. A standard convolution outputs a single value per filter and location. This activation is a combination of the input values and the learned filters. The repeated application of convolutions performs a spatially weighted average of the original signals. By combining this with the downsampling nature of max-pooling or with the use of convolutions with a stride larger than one pixel, the output activations are downsampled with respect to the original resolution. Typically, downsamplings reduce the spatial activations by a factor of 2.

If one wants to perform an in-network upsampling, activations of the previous layer can be relocated in the upsampled grid and values interpolated by a *deconvolution* operator. This allows to upsample spatially the signals, while still learning channelwise filters increasing the expressive power of the model. An example of $2\times$ upsampling is shown in Fig. 2(b), employed three times in the upsampling block of our system [Fig. 3 (blue box)]. It is important to remark that the deconvolution operator simply corresponds to the backward-pass implementation of the standard convolution (by mapping max-pooling activations back, i.e., "unpooling"). Inversely, the backward-pass of the deconvolution corresponds to a convolution. Therefore, most implementations of the CNNs already include such operation. Learned deconvolutions can be followed by nonlinearity layers (e.g., ReLU) in order to learn nonlinear upsampling filters. In our implementation, we employ deconvolutions to compensate the max-poolings, thus mirroring the number of downsampling operations. Differently from other deconvolution layers in the CNNs (e.g., the fully convolutional network in [26] or the deconvolutional network in [27]), we only upsample with a $2\times$ factor at every layer, and all the deconvolution filters are learned (we do not fix or initialize any layer to bilinear upsamplings as in [26]). By such small upsamplings, we directly obtain geometrically accurate labelings, without the need for unpooling or combining with the activations of lower (higher resolution) layers.

e) Classification layer and loss: The topmost layer [Fig. 3 (red boxes)] is composed by a classifier, whose loss is differentiable. We use the commonly employed multinomial logistic regression, whose scores (class-conditional probabilities) are given by the *softmax* function

$$p(y_i|\mathbf{x}_i) = \frac{\exp(\mathbf{x}_i)}{\sum_{c=1}^{C}\exp(\mathbf{x}_{ic})} \tag{4}$$

for C classes. Inputs $\mathbf{x}_i$ are a C-dimensional vector representing unnormalized scores for the location i, as given by the penultimate layer (the one before the loss function). The filters of this penultimate layer can be interpreted as the weight vector of the classifier. The classification loss (cross entropy) is

$$L(y_i) = -\sum_{c=1}^{C}[\![y_{p(y_{ic})} = c]\!]\log(p(y_{ic}|\mathbf{x}_i)). \tag{5}$$

This loss function trains the network by forcing it to put all the mass on the correct labeling. In our dense labeling setting, the loss is not computed over a single prediction as for the CNN-PC, but over the grid of spatial predictions. During training, the actual value of $L(y)$ corresponds to the average per-patch loss, over the training batch. The loss for each patch is again the average per-location loss (at each predicted pixel). During inference, the predicted label for location i will be given by $y_i^* = \text{argmax}_c\ p(y_{ic}|\mathbf{x}_i)$.

B. Mitigating Overfitting

In this paper, we employ these five layers to build the CNNs as shown in Fig. 3. Although the data sets are large and the number of parameters is not prohibitive, training of such models might be difficult. The information content in such data is heavily redundant and most semantic classes are characterized by relatively uniform, while most of the variations in the data lie in the spatial arrangement of such classes. Predicting a single label per patch is suboptimal, since the model is not explicitly taking advantage of these regularities while learning, but it is only trained to predict their presence in the (center of) the patch. To learn such spatial arrangement, we might need models with larger capacity, since deconvolutions are needed after the bottleneck layer. However, this corresponds to optimizing over a larger number of parameters. In this section, we will review a series of strategies to cope with overfitting.

a) Dropout: This technique has been proposed to avoid co-adaptation of neurons during training [21]. Co-adaptation would result in filters in the same layer, which are interdependent one to each other. Therefore, such network would be harder to train and ultimately fitting too tightly training data, without any good generalization guarantee. Dropout draws from ensemble learning theory: randomly "turning OFF" a given percentage of neurons (dropout rate hyperparameter) and their connections corresponds to train a different, less correlated model at every epoch. At inference time, an approximate ensemble method is obtained, activating all the connections.

In practice, dropout could slow down training, but benefits largely surpass drawbacks, in particular when the number of parameters to learn is large (e.g., fully connected layers).

b) Batch normalization: Batch normalization [22] aims at speeding up training by allowing the use of larger learning rates and mini-batches. To do so, it learns the normalization for each batch so that the activations passed to the next layer follow a normal distribution with $\mathcal{N}(0, 1)$. This might seem trivial, but it avoids problems related to the drift of activation distributions during training and makes the whole system less sensible to layer initialization. Moreover, by keeping values normalized at each layer, difference in the randomly selected mini-batches and the activations they generate should have less influence on the weight updates across iterations. Each layer can now focus on the general improvement rather than learning to adapt to the previous updates. We included this layer right after every convolutional and deconvolutional layer.

c) Weight decay: Weight decay is an ℓ_2 regularizer adding a penalty term to weight updates during backpropagation. It is applied only to convolutional filter weights (and not to the biases) and favors smooth convolutional filters. The weight decay hyperparameter controls the penalization.

d) Data augmentation: A commonly used strategy to further increase the size of the training set is to perform *data augmentation*. It consists in creating new synthetic training examples from those already available, by applying label-preserving (random) transformations. This step ensures that the model sees different possible aspects of the data in different batches, improving generalization error by: 1) increasing the number of labeled samples to learn from; 2) regularizing the model [40]; and 3) reducing potential correlation between patches in the batch. To train the CNNs, we first create a *superbatch* by sampling a given number of training patches. From this superbatch, we then randomly sample the mini-batch used to effectively train the CNNs. We apply transformations at both the levels. We describe adopted augmentation strategies as follows.

1) *Random Sampling:* Semantic classes are unevenly distributed spatially and their frequency highly varying. For instance, the class "road" is ubiquitous, while the class "car" is localized (appearing mostly on roads) and rare (in the Vaihingen data set described in Section IV-A, 1.2% of the training ground truth represents "cars," while 27.94% "roads"). Consequently, we generally sample the training patches randomly in space (among the training images) and uniformly with respect to class frequencies. To control this process, we simply account for the class corresponding to the label of the central pixel of the patch.
2) *Random Transformations:* In order to encode invariances of interest and include stochasticity in the mini-batch generation, we randomly rotate and flip training patches. Random rotations are applied on each training image before sampling the superbatch (as explained later in Section III, the superbatch will be resampled at regular intervals). Random flippings are applied when selecting the mini-batch during training, independently to rotations.
3) *Noise Injection:* The last class-preserving random transformation consists in jittering, i.e., injecting small random additive noise to each patch in the mini-batch. Jittering is important, since it forces the model to learn spectrally smooth decision rules, by reducing correlation across similar patches in the mini-batch and making the input distribution denser.

The noise is sampled from the normal distribution $\mathcal{N}(0, 0.01)$. Note that the input data are scaled in $[0, 1]$.

III. CNN Architectures Considered in this Paper

A. Common Strategies

Besides architecture-specific training strategies (detailed in the following for each CNN), we define some general rules to train each architecture, in particular with respect to sampling training patches. To train each model, we first sample the superbatch set composed by N^{trp} training patches. We define every training *epoch* to be composed of 500 passes over the superbatch set. The superbatches (at both training and testing) are centered around the mean values of the training set. We set its size as $N^{\text{trp}} = N_b \cdot 500$, uniformly sampled across the training tiles. N_b denotes the mini-batch size, which depends on the architecture. Each patch is sampled randomly in the spatial domain. We resample the superbatch every 20 epochs for the Vaihingen and 5 for Potsdam. For all the models, input patches are 65×65 pixels in size.

For all the convolution/deconvolution filters described in Section III.B to Section III.D, weights are initialized from $((2)/(M^2 \cdot K'))^{1/2}\mathcal{N}(0, 1)$ (improved Xavier initialization) and are all applied with a stride $= 1$.

We employ backpropagation with SGD with momentum [17]. We fix the momentum multiplier to the standard value of 0.9. To monitor the validation error during training, we predict, at each epoch, $N_b \cdot 100$ validation patches from the validation images, and compute the error. We also sample validation patches uniformly across classes. Note that the validation patches are never used in the training process.

B. Architecture 1: Patch Classification (CNN-PC)

1) Structure: This architecture is a standard patch-classification system, i.e., a model that takes as input a patch and predict the label of the patch (the label of the central pixel of the patch). We employ 64 neurons at the first layer (7×7 filters), 64 at the second (5×5), 128 at the third, and finally (5×5) 256 in the fourth layer. The fourth layer includes max-pooling and ReLU, and then implements a fully connected layer to map activations to single class scores, per patch. A schematic is shown in Fig. 3. The lReLUs with a leak parameter of 0.1 are used right after the batch normalization layers, and right before 3×3, stride 2 max-pooling operations. Dropout with a 50% rate is applied at each layer. No batch normalization or dropout is applied before the softmax.

2) Training: Learning rates for the SGD start at 10^{-3} then reduced by 0.5 every 100 epochs, until 300 epochs are completed. Then, it is set to 10^{-5} for the last 100 epochs. The weight decay is fixed to 0.01. The network is trained for a total of 400 epochs with the mini-batches of the size of 128.

3) Inference: At inference time, we predict a single label per patch. As discussed in Section I, patch-based strategies, such as the CNN-PC, do not offer the ability of predicting the directly structured spatial arrangements of pixels, since inference is done by spatially independent predictions. To provide a dense pixel map, different strategies can be adopted. The simplest naïve approach is to decompose the image into a series of overlapping patches of size corresponding to the one of the CNN input and predict the class for each region independently. In this paper, we adopt this prediction strategy mainly to avoid any potential bias introduced by other techniques. However, as we will detail in Section V, we performed inference with a stride larger than 1, and then, we interpolated results back to the original resolution with a negligible loss in accuracy.

An alternative way to obtain dense predictions is to employ the dense neural pattern strategy, presented in [41] and used in [35]. A similar approach is presented in [42], where stacked activations are defined as "hypercolumns."

C. Architecture 2: Subpatch Labeling (CNN-SPL)

1) Structure: The second CNN we train for comparison predicts the labeling of a subpatch of size 9×9 around the 65×65 input patch center. This network corresponds exactly to the CNN-PC, but the last max-pooling is removed and the fully connected layer of the CNN-PC is replaced by 1×1 convolutions, so that the output is now a patch of labels of the same size as the bottleneck layer. lReLUs with a leak parameter of 0.1 are used right after the batch normalization layers, and right before 3×3, stride 2 max-pooling operations. Dropout with a 50% rate is applied at each layer. No batch normalization or dropout is applied after the 1×1 convolutions.

2) Training: The CNN-SPL is trained in the same way as the CNN-PC described in the following. We use the weights of the CNN-PC to warm start the training procedure. For this reason, learning rates for this network architecture are 10 times smaller than the ones employed for the CNN-PC (but the same number of epochs), as we basically only fine tune the system to the new output. Mini-batches are composed by 128 examples.

3) Inference: Because of the special structure in predicting the patches of labels, a feedforward pass of the whole image would produce a prediction with a resolution roughly 3 times higher compared with the CNN-PC. We upsample the outputs of the last layer (softmax scores) using bilinear interpolation to match the size of the original image tile.

D. Proposed Architecture: Full Patch Labeling by Learned Upsampling (CNN-FPL)

1) Structure: This architecture is composed by two main parts: a downsampling [Fig. 3 (green area)] and an upsampling block [Fig. 3 (blue area)]. The downsampling part corresponds exactly to the CNN-SPL, but the 1×1 convolutions are replaced by 3×3, 512-D deconvolution layers, doubling the size of the activation of the previous layer. We employ three deconvolution layers to compensate three levels of downsampling (max-pooling). The $65 \times 65 \times 512$ activations are mapped to class scores through $1 \times 1 \times 6$ convolutions, to obtain a $65 \times 65 \times 6$ score map. As for previous architectures, batch normalization is applied after every convolution/deconvolution, and the dropout is applied with a 50% rate. No dropout nor batch normalization is applied after the 1×1 convolutions. The lReLUs are employed with a leak factor of 0.1, and the max-pooling layers in the downsampling part are exactly as for the other architectures.

We argue that the bottleneck structure is beneficial, since it forces the network to learn a rough spatial representation for the classes present in the patch (after the downsampling block, the original 65×65 patch is reduced to a 9×9 map of activations), before learning upsampling of these representations back to the input resolution. Note that the embeddings of the CNN-SPL and CNN-FPL architectures at the 9×9 activation layer are completely different: in the former, the network learns to predict a small patch corresponding to the central area of the input region. Thus, the concepts embedded in the penultimate layer directly correspond to the arrangement of pixels in the original resolution image. In the proposed CNN-FPL network, the concepts in the bottleneck correspond to a degraded resolution of the entire input patch, pointing to main features in a coarser geometry. The upsamplings are learned so that the deconvolutions reconstruct the learned spatial and geometrical arrangements of activations at larger scale but acting locally (3×3). This strategy allows interpreting the output of the CNN-FPL as structured, since every predicted label is learned to be interdependent with its neighbors, conditioned on the receptive field of the previous layers.

2) Training: The CNN-FPL is trained in the same way as architectures described previously. However, since it involves the upsampling layers, the number of learnable parameters is significantly higher than for the two previous networks. We warm start the system by employing weights from the CNN-PC, for the common downsampling blocks. The learning rates are initialized at 10^{-3} and kept uniform for 100 epochs, then decreased to $5 \cdot 10^{-4}$ and 10^{-4} for 100 epochs each. The last 300 epochs are run with a learning rate of 10^{-5}. With respect to the Vaihingen experiment, this training procedure uses 100 epochs more since mini-batches are smaller. The weight decay is set as for the Vaihingen model.

3) Inference: At inference time, the proposed CNN-FPL approach produces a segmentation of the same size as the input patch by a single forward pass of the image. This way, the three main drawbacks of the aforementioned systems are circumvented: first, the whole image can be feedforwarded to the trained CNN, and we can directly obtain a dense labeling (stride is equal to 1 by construction) or a dense class-conditional score map; second, we do not need a second step of prediction upsampling; and third, the prediction performed this way is locally structured, as the CNN is able to exploit both color and semantic correlations represented in the input patch, without the need of a subsequent structured output postprocessing.

E. On the Choice of the Architecture and General Setup

We tested different architectural setups by evaluating the accuracy on the validation set. The final CNN-FPL network

architecture is driven by mainly three factors. First, the number of downsampling layers has been fixed to 3, after testing also two and four max-poolings layers (and consequently the total number to six plus one 1×1 score map layers). The former made the network too shallow, while the latter too deep to optimize. The solution in the middle offered the best tradeoff. Second, we tested patch sizes large enough to include some spatial context, in particular local co-occurrences and allowing filters to learn recurrent structures. We trained for a few iteration networks with varying input patch sizes in $\{25, 45, 55, 65, 85\}$ and observed that the size of 65 pixels side offered a good compromise between accuracy and memory requirements. The size of the input patch also directly influences the spatial extent at the bottleneck layer: 65×65 are reduced to 9×9 at the bottleneck, whose extent contains enough spatial information to allow direct and geometrically accurate upsamplings. Larger bottleneck provides similar accuracy, but requiring more memory during training. Third, the number of neurons has been chosen by training again for few iterations a series of networks with a varying number of neurons at the first layer, growing in the power of two (starting from 16). We follow roughly the rule that a layer has double the number of channels of its predecessor. Since the number of channels directly influences overfitting, it directly controls the number of learnable parameters, we did not selected more than 512 channels. The architectures of the competing the CNN models are all based on the CNN-FPL architecture, as described previously (by removing the deconvolutional part, we obtain the CNN-SPS network and by including a fully connected layer on top of it, we obtain the CNN-PC). This allows us to directly compare the effects of the architectures. Finally, we selected the batch size experimentally on the validation set and it has been fixed to 128 for the CNN-PC and the CNN-SPL, while to 32 for the CNN-FPL. The smaller batches cause slower convergence but ease the problem of overfitting.

All the presented results are computed on a desktop machine with an Intel Xeon E3-1200 (QuadCore), 32-Gb RAM and an Nvidia GeForce GTX Titan X (12-Gb RAM). We build the tested system using the DAGNN wrapper around CNN libraries provided by MatConvNet[3] version 1.0 beta 20.

IV. Data and Experimental Setup

A. Dataset Description

We evaluate the proposed system on the Vaihingen and Potsdam data sets provided in the framework of the 2-D semantic labeling contest organized by the International Society for Photogrammetry and Remote Sensing (ISPRS) Commission III.[4] The Vaihingen data are composed by a total of 33 image tiles (average size of 2494×2064), 16 of which are fully annotated. The remaining tiles compose the test set. Each image is composed by near infrared (NIR), red (R) and green (G) channels with a spatial resolution of 9 cm. We also dispose a DSM coregistered to the image data, which has been normalized (*n*DSM) and redistributed in [43]. In this paper, we jointly use spectral information (NIR-R-G) and the *n*DSM to train the network. An example of image tile is given in the first three panels of Fig. 5. We use 11 out of the 16 annotated images to train the networks and the remaining five to validate training and to evaluate the segmentation generalization accuracy (ID 11, 15, 28, 30, 34). The input of each network corresponds to stacked NIR-R-G and *n*DSM, for a total of four input dimensions. All the data available are rescaled into the $[0, 1]$ interval. Compared with state-of-the-art on this data set, we refer to the results published on the challenge Web site.[5]

The Potsdam 2-D semantic labeling challenge data set[6] features 38 tiles of size 6000×6000 pixels, with a spatial resolution of 5 cm. From the available patches, 24 are densely annotated, with the same classes as for the Vaihingen data set. This data set offers Near InfraRed, Red, Green and Blue channels together with the DSM and normalized DSM. We employ all spectral channels plus the DSM as input for our networks (five dimensions). In our setting, we use the tiles 02_12, 03_12, 04_12, 05_12, 06_12, and 07_12 for validation, and the remaining 18 for training. As for the Vaihingen case study, we employ as input all the spectra information and the *n*DSM, rescaled in $[0, 1]$.

B. Competing Method

Besides the CNN-PC and CNN-SPL architectures, acting as strong baselines, we also compare the CNN-FPL to a modern baseline implementing standard superpixel-based labeling with hand-crafted features. For each image, we stack raw color information, normalized difference vegetation index, normalized difference water index using G and NIR channels and normalized DSM. For each feature, we extract four mathematical morphology operators (opening, closing, and their reconstructions) as well as a texture statistic (entropy). Each filter is operated in three window sizes (7, 11, and 15 pixels). We then extract regions using the graph-based superpixelization of [44] on the channels of the raw data. We summarize the distribution of appearance descriptors in each superpixel by extracting minimum, maximum, average, and standard deviation of the values. The final feature set is composed of 384 features. We then train a random forest classifier on this set by training 500 trees with 100 random feature tests per split, and retaining the one with optimal Gini diversity criterion. Each tree is trained with the full training set composed of 5000 randomly selected example *per class*, over the 11 training images. A leaf in the tree is obtained when only two examples remain at the node. We refer to this approach as superpixels with multiscale features (SP-MSFs). We use the same strategy on both the data sets.

C. Evaluation Metrics

For both the data sets, we present figures of merit obtained on the corresponding validation sets. We employ four different

[3] http://www.vlfeat.org/matconvnet/

[4] http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html

[5] http://www2.isprs.org/vaihingen-2d-semantic-labeling-contest.html

[6] http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

TABLE I
NUMERICAL RESULTS FOR THE VAIHINGEN VALIDATION SET

	Model	OA	K	AA	F1
full	SP-MSF	79.79	73.28	65.80	66.20
	CNN-PC	82.62	76.97	62.10	64.19
	CNN-SPL	83.04	77.50	64.05	67.89
	CNN-FPL	**83.79**	**78.52**	**69.12**	**73.03**
no bk	SP-MSF	79.86	73.36	74.24	74.40
	CNN-PC	82.68	77.04	74.47	76.95
	CNN-SPL	83.08	77.55	70.76	72.89
	CNN-FPL	**83.83**	**78.57**	**76.45**	**78.76**
er full	SP-MSF	83.64	78.32	70.65	69.92
	CNN-PC	86.67	82.29	65.58	68.01
	CNN-SPL	87.24	83.03	69.17	73.47
	CNN-FPL	**87.80**	**83.79**	**74.94**	**78.60**
er no bk	SP-MSF	83.71	78.39	78.44	77.64
	CNN-PC	86.73	82.36	78.66	81.57
	CNN-SPL	87.28	83.07	75.03	77.78
	CNN-FPL	**87.83**	**83.83**	**81.35**	**83.58**

evaluation strategies. The first—**full**—is a standard full spatial evaluation with all classes present in the ground truth. The second strategy—**no bk**—excludes from the evaluation the class "background," but preserves the full spatial extent of the ground truth. The last two—**er full** and **er no bk**—estimate figures of merit as in the first two strategies, but after eroding the edges of each class in the ground truth with a 3-pixel circular structuring element, so that evaluation is tolerant to small errors on object edges. We report overall accuracy (OA), Kappa (K), average class accuracy (AA), and class-averaged F1 score (F1). The OA and K are the global measures of segmentation accuracy, the former providing information about the rate of correctly classified pixels and the second compensating for random chance in assignment. Both are biased toward large classes, meaning that the contributions of small classes are canceled out by those of larger classes. Both AA and F1 are class-specific and, therefore, independent of class-size. The former provides the average (per-class) ratio of correctly classified samples (it is therefore insensitive to the size of the ground truth classes), while the latter is the geometric mean between precision (user's accuracy) and recall (producer's accuracy). More details can be found in [45].

V. RESULTS

A. Vaihingen Dataset Results

1) Numerical Results: Table I presents results for all the tested CNN, and the competing method relying on the superpixels. In all the evaluation settings, the CNN-FPL is the most accurate under different accuracy metrics. By removing the "clutter" class, the standard CNN-PC baselines gains 11–12 points in AA and F1 scores, indicating that this class was mostly missed. The strategies trained on the patch of labels are more robust in this sense, gaining in the range of 4–5 points. By evaluating on eroded boundary ground truths, we observe a similar behavior, but with significantly higher accuracies. This indicates that in all situations the boundaries are often blurred within the 3-pixel erosion radius. The CNN-SPL shows performances in between the CNN-PC and the CNN-FPL. It is able to describe the class "background," but evaluations without this class bring it at performances lower than the ones of the CNN-PC. All the CNN-based models outperform the baseline, employing dense appearance descriptors and superpixel regions. The proposed CNN-FPL results in being the most accurate numerically. We argue that the CNN-FPL makes better use of the training data by learning class relationships and co-occurrences, naturally.

TABLE II
AVERAGE INFERENCE TIME PER IMAGE (ON VALIDATION SET) FOR THE CNN MODELS

	CNN-PC	CNN-SPL	CNN-FPL
time (s/image)	2768 (1360.6†)	1.8	6.2
† Inference time with stride 2			

The CNN-FPL is not only the most accurate method, but it is also extremely fast. As shown in Table II, it only takes 31 s (6.2 s/image) to perform inference on the five test images (average size of 2563 × 1810 pixels) on the GPU. The naïve inference used for the CNN-PC, i.e., predicting the label of the batches of patches independently then rearranging them spatially, requires 13841 s (2768 s/image) on the GPU. However, in the rest of the experiments, we performed inference with a stride of 2 and then upsampling the probabilities scores with bilinear interpolation. This way, the time can be roughly reduced by two (one prediction per 2 × 2 pixel grid, instead of 4) by losing very little accuracy (<1% OA). Note that there exist different strategies aiming at efficiently speeding up prediction time for the CNN. In this comparison, we mainly wanted to point out that learning full patch segmentations results *naturally* in a more accurate and dense prediction. The CNN-SPL is the fastest at predicting maps, since it only needs to evaluate few downsampling layers, only requiring 9 s to predict maps of the whole validation set (1.8 s/image).

2) Qualitative Results: In Fig. 4(1)–(4), we summarize some aspects of the predictions by clipping map portions from four out of the five validation images (the fifth image is shown in its entirety in Fig. 5). In all the clips, we can see how the different methods act on boundaries, in particular for the class "building." For buildings having high contrasted boundaries, the SP-MSF provides the best preservation of their geometry. However, there are many cases of "building" instances and particularly for other classes, in which boundaries are not sharp: in these cases, the SP-MSF is prone to fail. For single regions with ambiguous appearance, the predictions can be noisy and, since the context of each region is not taken into account, can result in wrong assignments. The CNN-based strategies result in more accurate and semantically coherent segmentations. The CNN-PC, because of its unstructured nature, does not preserve well object boundaries and tends to oversmooth classes with complex boundaries. As expected, the CNN-SPL offers a tradeoff between the CNN-PC and the CNN-FPL, which, in turn, offers best segmentations. The CNN-FPL makes better use of class co-occurrences, for instance by avoiding spurious prediction of small patches of the class building (e.g., in clip 1). The CNN-FPL deals better

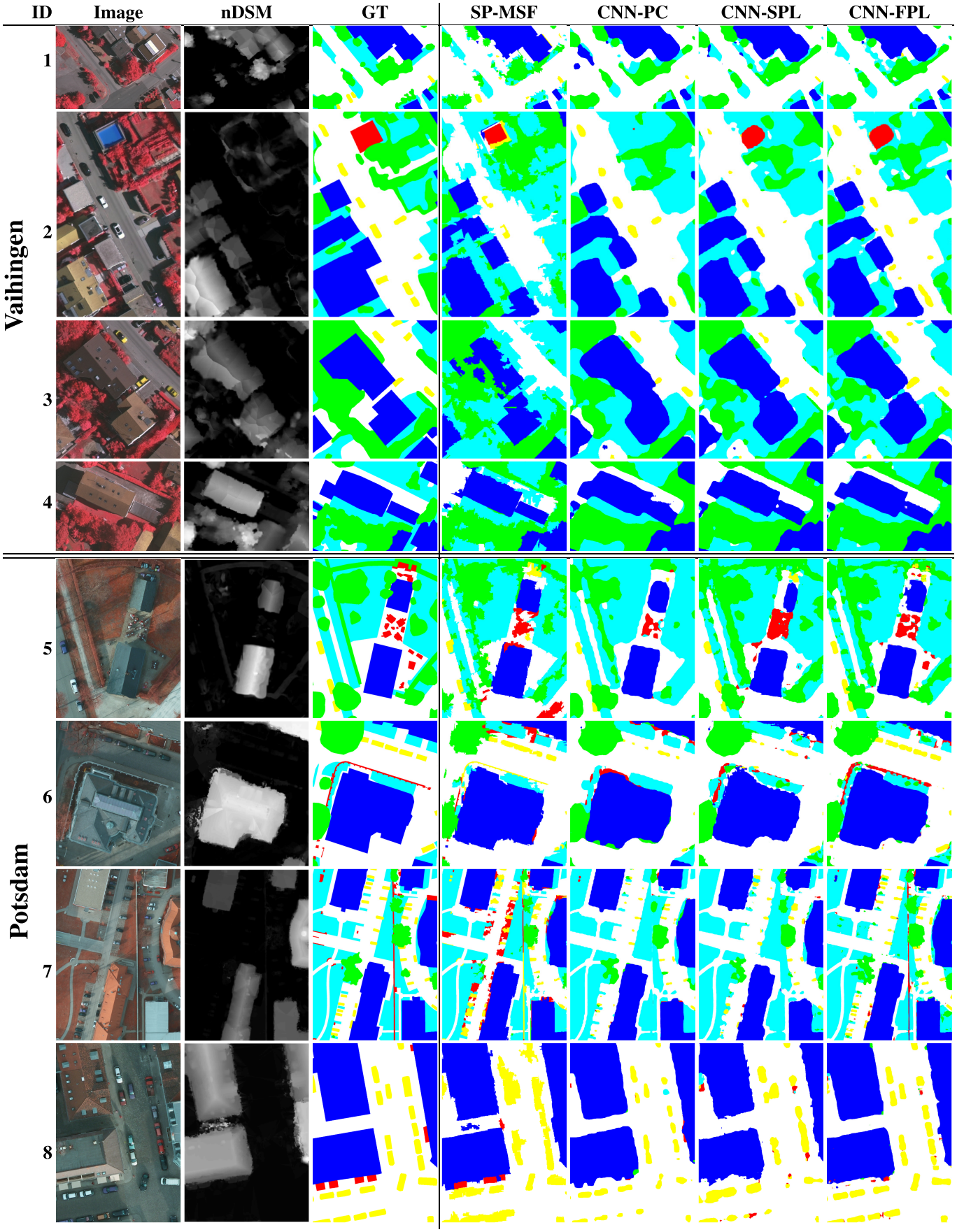

Fig. 4. Example predictions for the (1)–(4) Vaihingen and (5)–(8) Potsdam for the tested architectures. Legend—White: impervious surfaces. Blue: buildings. Cyan: low vegetation. Green: trees. Yellow: cars. Red: clutter, background. Best viewed in color pdf.

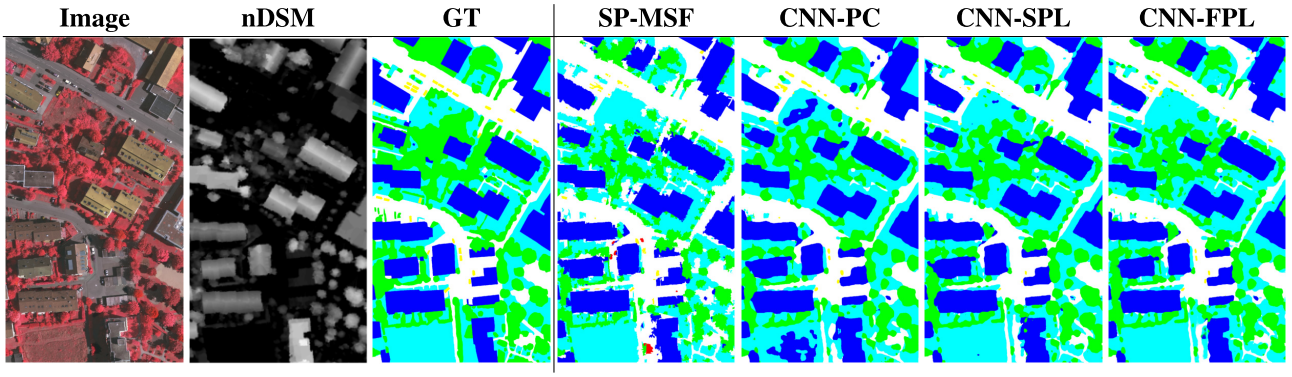

Fig. 5. Full prediction for tile ID 34. Legend—White: impervious surface. Blue: buildings. Cyan: low vegetation. Green: trees. Yellow: cars. Red: clutter.

with thin and elongated elements and boundaries in general, by preserving the shape of such structures (e.g., building shape in clip 4 and the gap between the buildings in clip 3) thanks to the learned upsamplings.

The class "car" is very difficult to correctly segment. The SP-MSF often misclassifies "cars" because superpixels do not always isolate class instances. Again, the CNNs are generally more accurate. The CNN-PC, although showing good segmentation for detected cars, misses most of them. Semantic labeling CNN is more accurate in detecting single cars, and, in particular CNN-FPL offers a good tradeoff between segmentation accuracy and detection (e.g., clips 2 and 3). The class "background" is detected by all methods with different success rates, mostly depending on its local appearance (recall that this class is not semantically nor visually coherent, since collecting different semantic classes).

Coupling elevation and spectral information eases the detection of buildings and trees (the elevated classes). Such information obviously helps in discriminating ambiguous occurrences of the class building, e.g., rooftops showing the same appearance as roads. Rooftops covered in dense vegetation are often misclassified as trees and never as grass, since being on two different elevation levels. In Fig. 5, the full segmentation of image tile 34 is given. What observed for the clips previously summarizes the classification of the entire tile.

3) Submission to Challenge: To compare with the state-of-the-art models, we submitted the maps obtained for the 17 unlabeled tiles, to obtain the test accuracy. With the independent Vaihingen evaluation criteria, we scored 87.3 points in OA. This sets the CNN-FPL as the fifth most accurate model,[7] on a tie with the ADL_2 entry [35]. The CNN employed by other participants ranged from fully convolutional networks [26] to ensemble of the (multiscale) patch-based CNN. However, it is worth pointing out that all the models showing higher accuracy on this test set combine either features learned by the CNN and handcrafted features, classified by an additional external nonlinear classifier (e.g., random forests) and/or postprocessed by a conditional random field of varying complexity. In our setting, we *only* employ classification scores as given by the CNN-FPL, with *no additional postprocessing techniques*. We could argue that by adding manually extracted features (to compensate for appearance variations not learned by the CNN) and/or adding a further smoothing by random fields models, we could gain a few additional accuracy points, as pointed out in [36]. But this would hinder the contribution of this paper.

[7]http://www2.isprs.org/vaihingen-2d-semantic-labeling-contest.html, submission results as March the 29th, 2016 (UZ_1 entry).

TABLE III
NUMERICAL RESULTS FOR THE POTSDAM VALIDATION SET

	Model	OA	K	AA	F1
full	SP-MSF	82.29	76.28	72.27	68.55
	CNN-PC	84.07	78.13	63.67	66.83
	CNN-SPS	83.00	76.93	66.91	67.90
	CNN-FPS	**85.85**	**80.88**	**74.31**	**73.81**
no bk	SP-MSF	85.73	80.55	83.24	80.16
	CNN-PC	85.83	80.36	74.27	77.33
	CNN-SPS	84.94	79.35	78.24	78.85
	CNN-FPS	**87.94**	**83.52**	**86.06**	**85.32**
er full	SP-MSF	84.40	78.94	74.89	70.30
	CNN-PC	86.58	81.47	66.35	69.78
	CNN-SPS	85.49	80.18	69.49	70.51
	CNN-FPS	**88.04**	**83.72**	**76.82**	**76.12**
er no bk	SP-MSF	87.56	82.94	85.75	82.16
	CNN-PC	88.05	83.36	77.26	80.52
	CNN-SPS	87.14	82.30	81.01	81.64
	CNN-FPS	**89.86**	**86.06**	**88.79**	**87.97**

B. Potsdam Dataset Results

For this data set, we trained the CNN models in two steps. In the first step, we created the superbatch by sampling all the 65×65 patches with an overlap of 33 pixels, for each image at a time. We iterated over all the entire training images for 200 epochs (i.e., no class-balanced random sampling). After that, we used the strategy employed for the Vaihingen data set, which samples patches approximately uniformly across classes. We employed this strategy to speed up training by first learning the most important recurring patterns/classes in the images. The second step aimed at fine tuning the network to learn specific, finer class-appearance representations.

1) Numerical Results: In Table III, we present results using the validation images shown in Section IV-A. For the CNN-PC approach, we performed inference with a stride of 5, since the size of the images is significantly larger (6000×6000 pixels)

and the spatial resolution of the images is roughly the half (5 cm). The loss in accuracy when using a stride of 5 is less than 0.5%, but the computational time is immensely reduced (the system 1 440 000 windows instead of 36 000 000, so 25× less). We use bilinear upsampling on the posterior probability maps to upsample results to the original image size, similar to the last layer of [26].

As observed for the Vaihingen data set, the superpixel baseline offers more balanced errors across classes compared with the CNN-PC, since the CNN-PC shows higher OA and K scores, while lower AA and F1. Both the approaches perform similarly on the "clutter" class, since balanced evaluation metrics significantly increase while removing such class. Directly predicting patches with the CNN-SPL strategy results in global metrics roughly on par to the baselines SP-MSF and CNN-PC, while slightly better than the CNN-PC on balanced metric. As for the Vaihingen data set, the improved modeling power of the CNN-FPL offers better accuracies for all the accuracy metrics considered.

2) Qualitative Results: In Fig. 4(6)–(9), we plot the examples of semantic labelings on the subsets of the validation images. Due to the higher spatial resolution (5 cm instead of 9 cm for the Vaihingen data set) the land-cover classes are represented by a more variate appearance, in both color and size. For instance, in clips 7 and 8, the thin fences and wall correspond to class "clutter" and only SP-MSF and CNN-FPS are able to detect it, while only the latter scoring the correct class most of the time. We acknowledge the good performance of the SP-MSF, but we also note that the ambiguous appearance of superpixels cannot be solved. Cars and buildings are generally segmented correctly by all the CNNs, but only the CNN-FPS is able to segment them in a geometrically accurate manner, e.g., single cars segmented as whole objects and not as multiple parts or undersegmented ones (CNN-PC and CNN-SPL). Again, this beneficial effect stems from learned deconvolutions, to upsample to full image resolution. We also note that for this data set, the class "clutter" and "trees" are hard to model. The former situation is mostly due to the very variate and mixed nature of it. For the latter, the fact that some trees do not have leaves makes the modeling of their actual extent hard, or even hard to detect when they stand on grass (see, e.g., clips 6 and 7). The CNN-based system is able to perform well, but again SP-MSF misses the difficult instances.

3) Submission to Challenge: To compare with the recent submissions of [36], we submitted the prediction for the 14 test images to the Potsdam 2-D semantic labeling challenge.[8] The predictions for the test images show an OA of 85.8%,roughly 5% less accurate than the approaches presented in [36]. In particular, the class "tree" is around 7%–8% worse than the aforementioned entry. However, we believe that the results are satisfactory, in particular since we are using significantly simpler network architectures if compared with the models presented in [18] (VGG16) and employing no postprocessing, which could potentially lead to better results.

[8]http://www2.isprs.org/potsdam-2d-semantic-labeling.html, submission results as August 2nd, 2016 (UZ_1 entry).

VI. Discussion and Conclusion

In this paper, we presented an approach to perform dense semantic labeling using the CNNs. The adopted architecture first encodes concepts in a rough spatial map represented by many channels, and then learns rules to upsample this spatially coarse features back to the original resolution, via learned deconvolutions. We proposed this system to cope with the high spatial and geometrical information contained in ultrahigh resolution images (<10 cm), usually coupled to little spectral information. However, the full patch segmentation network can also be applied to scenarios, where normalized DSMs are not available, spectral channels are numerous, and resolution is coarser. The main challenge to transfer such approach to the processing of satellite images would be the availability of densely annotated ground truth, to train discriminatively CNN models.

The numerical and qualitative results illustrated the advantages of learning CNN directly for segmentation tasks, as underlined in [36] as well. Predicting the segmentation for full patches is actually advantageous from both the efficiency and semantic/geometric accuracy perspectives. Practically, if compared with standard CNN performing patch classification (i.e., CNN-PC), we were not able only to leverage the power of semantic abstraction of standard CNN, but we could also learn nonlinear upsamplings, encoding class-relationships and co-occurrences at a higher semantic level. This is a direct consequence of interdepentent predictions, thanks to the hierarchical downsample-then-upsample architecture. One can interpret the learned upsamplings as activation specific interpolation filters, encoding the spatial dependence of locations. Advantages are clear compared with models predicting each location in isolation, based on the appearance of the patch.

We obtained results aligned with the state-of-the-art models on two extremely challenging data sets, without performing any postprocessing (e.g., CRF or MRF) and without recurring to strategies involving external classifiers and additional handcrafted features.

References

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

[2] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit.*, Oct. 2003, pp. 1470–1477.

[3] L. Gueguen, "Classifying compound structures in satellite images: A compressed representation for fast queries," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 4, pp. 1803–1818, Apr. 2015.

[4] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "*TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2006, pp. 1–15.

[5] D. Tuia, M. Volpi, M. D. Mura, A. Rakotomamonjy, and R. Flamary, "Automatic feature learning for spatio-spectral image classification with sparse SVM," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 10, pp. 6062–6074, Oct. 2014.

[6] P. Tokarczyk, J. D. Wegner, S. Walk, and K. Schindler, "Features, color spaces, and boosting: New insights on semantic classification of remote sensing images," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 1, pp. 280–295, Jan. 2015.

[7] D. Tuia, R. Flamary, and N. Courty, "Multiclass feature learning for hyperspectral image classification: Sparse and hierarchical solutions," *ISPRS J. Photogram. Remote Sens.*, vol. 105, pp. 272–285, Jul. 2015.

[8] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 1–8.
[9] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.
[10] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun, "Learning convolutional feature hierarchies for visual recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 1090–1098.
[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 1–8.
[12] V. Mnih and G. E. Hinton, "Learning to label aerial images from noisy data," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1–8.
[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[14] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[19] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.
[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit.*, Dec. 2015, pp. 1026–1034.
[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1–11.
[23] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. IEEE Int. Conf. Comput. Vis.*, Sep./Oct. 2009, pp. 2146–2153.
[24] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.
[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE/CVF Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[27] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1520–1528.
[28] D. Marmanis, M. Datcu, T. Esch, and U. Stilla, "Deep learning earth observation classification using ImageNet pretrained networks," *IEEE Geosci. Remote Sens. Lett.*, vol. 13, no. 1, pp. 105–109, Jan. 2015.
[29] O. A. B. Penatti, K. Nogueira, and J. A. dos Santos, "Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?" in *Proc. IEEE/CVF Comput. Vis. Pattern Recognit. Workshops*, Jun. 2015, pp. 44–51.
[30] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. (2015). "Land use classification in remote sensing images by convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1508.00092
[31] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, Jun. 2014.
[32] O. Firat, G. Can, and F. T. Y. Vural, "Representation learning for contextual object and region detection in remote sensing," in *Proc. Int. Conf. Pattern Recognit.*, Aug. 2014, pp. 3708–3713.
[33] F. Zhang, B. Du, and L. Zhang, "Saliency-guided unsupervised feature learning for scene classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 4, pp. 2175–2184, Apr. 2015.
[34] C. Romero, A. Gatta, and G. Camps-Valls, "Unsupervised deep feature extraction for remote sensing image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 3, pp. 1349–1362, Mar. 2016.
[35] S. Paisitkriangkrai, J. Sherrah, P. Janney, and A. Van-Den Hengel, "Effective semantic pixel labelling with convolutional networks and conditional random fields," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2015, pp. 36–43.
[36] J. Sherrah. (2016). "Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery." [Online]. Available: https://arxiv.org/abs/1606.02585
[37] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," in *Proc. Int. Conf. Mach. Learn. Workshops, Deep Learn.*, 2015, pp. 1–5.
[38] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectier nonlinearities improve neural network acoustic models," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1–6.
[39] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2013, pp. 1–9.
[40] C. M. Bishop, "Training with noise is equivalent to Tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, 1995.
[41] X. Wang, M. Yang, S. Zhu, and Y. Lin, "Regionlets for generic object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 2071–2084, Oct. 2015.
[42] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 447–456.
[43] M. Gerke, "Use of the stair vision library within the ISPRS 2D semantic labeling benchmark (Vaihingen)," ITC, Univ. Twente, Enschede, The Netherlands, Tech. Rep., 2015, doi: 10.13140/2.1.5015.9683.
[44] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, Sep. 2004.
[45] R. G. Congalton and K. Green, *Assessing the Accuracy of Remotely Sensed Data*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2008.

Michele Volpi (S'08–M'13) received the Ph.D. degree in environmental sciences from the University of Lausanne, Lausanne, Switzerland, in 2013.

Until 2016, he held a visiting post-doctoral position with the CALVIN Group, School of Informatics, Institute of Perception, Action, and Behaviour, University of Edinburgh, Edinburgh, U.K., where he is involved in the boundary of remote sensing and computer vision. Since 2016, he has been a Post-Doctoral Fellow with the MultiModal Remote Sensing Group, University of Zurich, Zürich, Switzerland. His current research interests include the interface of remote sensing, machine, and deep learning, computer vision, and pattern recognition for the extraction of information from aerial and satellite data.

Dr. Volpi has been a Co-Chair of the ISPRS Working Group II/6, Large Scale Machine Learning for Geospatial Data Analysis, since 2016.

Devis Tuia (S'07–M'09–SM'15) received the Ph.D. degree in environmental sciences from the University of Lausanne, Lausanne, Switzerland, in 2009.

He was a Post-Doctoral Researcher with the University of València, València, Spain, the University of Colorado, Boulder, CO, USA, and the École Polytechnique Fédérale de Lausanne, Lausanne. Since 2014, he has been an Assistant Professor with the University of Zurich, Zürich, Switzerland. His current research interests include the information extraction and data fusion of remote sensing images using machine learning algorithms.