Recurrent Convolutional Neural Networks for Scene Labeling

Pedro O. Pinheiro[1,2] PEDRO.PINHEIRO@IDIAP.CH
Ronan Collobert[2] RONAN@COLLOBERT.COM
[1]Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

[2]Idiap Research Institute, Martigny, Switzerland

Abstract

The goal of the scene labeling task is to assign a class label to each pixel in an image. To ensure a good visual coherence and a high class accuracy, it is essential for a model to capture long range (pixel) label dependencies in images. In a *feed-forward* architecture, this can be achieved simply by considering a sufficiently large input context patch, around each pixel to be labeled. We propose an approach that consists of a recurrent convolutional neural network which allows us to consider a large input context while limiting the capacity of the model. Contrary to most standard approaches, our method does not rely on any segmentation technique nor any task-specific features. The system is trained in an end-to-end manner over raw pixels, and models complex spatial dependencies with low inference cost. As the context size increases with the built-in recurrence, the system identifies and corrects its own errors. Our approach yields state-of-the-art performance on both the Stanford Background Dataset and the SIFT Flow Dataset, while remaining very fast at test time.

1. Introduction

In the computer vision field, *scene labeling* is the task of fully labeling an image pixel-by-pixel with the class of the object each pixel belongs to. This task is very challenging, as it implies solving jointly detection, segmentation and multi-label recognition problems.

The image labeling problem is most commonly addressed with some kind of *local* classifier constrained in its predictions with a graphical model (*e.g.* conditional random fields, markov random fields), in which *global* decisions are made. These approaches usually consist of segmenting the image into superpixels or segment regions to assure a visible consistency of the labeling and also to take into account similarities between neighbor segments, giving a high level understanding of the overall structure of the image. Each segment contains a series of input features describing it and contextual features describing spatial relation between the label of neighbor segments. These models are then trained to maximize the likelihood of correct classification given the features (Verbeek & Triggs, 2008; Gould et al., 2009; Liu et al., 2011; Kumar & Koller, 2010; Socher et al., 2011; Lempitsky et al., 2011; Tighe & Lazebnik, 2013). The main limitation of scene labeling approaches based on graphical models is the computational cost at test time, which limits the model to simple contextual features.

In this work, we consider a *feed-forward neural network* approach which can take into account long range label dependencies in the scenes while controlling the capacity of the network. We achieve state-of-the-art accuracy while keeping the computational cost low at test time, thanks to the complete feed-forward design. Our method relies on a recurrent architecture for convolutional neural networks: a sequential series of networks sharing the same set of parameters. Each instance takes as input both an RGB image and the classification predictions of the previous instance of the network. The network automatically learns to smooth its own predicted labels. As a result, the overall network performance is increased as the number of instances increases.

Compared to graphical model approaches relying on image segmentation, our system has several advantages: (i) it does not require any engineered features, since deep learning architectures train (hopefully) adequate discriminative filters in an end-to-end manner, (ii) the prediction phase does not rely on any label space searching, since it requires only the *forward evaluation of a function*.

The paper is organized as follows. Section 2 briefly

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

Table 1. Comparison between different methods for full scene labeling. The advantage of our proposed method is the simplicity of inference, not relying on any task-specific feature extraction nor segmentation method.

METHOD	TASK-SPECIFIC FEATURES
(Gould et al., 2009)	17-dimensional color and texture features, 9 grid locations around the pixel and the image row, region segmentation.
(Munoz et al., 2010)	Gist, pyramid histogram of oriented gradients, color Histogram CIELab, relative relocation, hierarchical region representation.
(Kumar & Koller, 2010)	Color, texture, shape, percentage pixels above horizontal, region-based segmentation.
(Socher et al., 2011)	Same as (Gould et al., 2009).
(Lempitsky et al., 2011)	Histogram of visual SIFT, histogram of RGB, histogram of locations, "contour shape" descriptor.
(Tighe & Lazebnik, 2013)	Global, shape, location, texture/SIFT, color, appearance, MRF.
(Farabet et al., 2013)	Laplacian pyramid, superpixels/CRF/tree segmentation, data augmentation.
Our Recurrent CNN	Raw pixels

presents related works. Section 3 describes the proposed strategy. Section 4 presents the results of our experiments in two standard datasets: the Stanford Background Dataset (8 classes) and the SIFT Flow Dataset (33 classes) and compare the performance with other systems. Finally, Section 5 provides a discussion followed by a conclusion.

2. Related Work

Recurrent Neural Networks (RNNs) date back from the late 80's. Already in (Jordan, 1986), the network was fed (in a time series framework) with the input of the current time step, plus the output of the previous one. Several variants have been later introduced, such as in (Elman, 1990). RNNs have been successfully applied to wide variety of tasks, including in natural language processing (Stoianov et al., 1997), speech processing (Robinson, 1994) and image processing (Graves & Schmidhuber, 2008). Our approach can be viewed as a particular instance of the Jordan's recurrent network adapted to image processing (we use a convolutional neural network instead). Providing feedback from the output into the input allows the network to model label dependencies, and correct its own previous predictions.

In a preliminary work, (Grangier et al., 2009) proposed an innovative approach to scene labeling without the use of any graphical model. The authors proposed a solution based on deep convolutional networks relying on a *supervised* greedy learning strategy. These network architectures when fed with raw pixels are able to capture texture, shape and contextual information.

(Socher et al., 2011) also considered the use of deep learning techniques to deal with scene labeling, where off-the-shelf features of segments are recursively merged to assign a semantic category label. In contrast, our approach uses the recurrent architecture to parse the scene with a smoother class annotation.

In (Socher et al., 2012), the authors proposed an approach which combines convolutional and recursive networks for classifying RGB-D images. The approach first extracts features using a convolutional network which is then fed to a standard recurrent net. In that respect, our approach is more end-to-end.

More recently, (Farabet et al., 2013) investigated the use of convolutional networks to extract features from a multiscale pyramid of images. This solution yields satisfactory results for the categorization of the pixels, but poor visual coherence. In order to improve visual coherence, three different over-segmentation approaches were proposed: (i) the scene is segmented in superpixels and a single class is assigned to each of the superpixels, (ii) a conditional random field is defined over a set of superpixels to model joint probabilities between them and correct aberrant pixel classification (such as "road" pixel surrounded by "sky"), and (iii) the selection of a subset of tree nodes that maximize the average "purity" of the class distribution, hence maximizing the overall likelihood that each segment will contain a single object. In contrast, our approach is simpler and completely feed-forward, as it does not require any image segmentation technique, nor the handling of a multiscale pyramid of input images.

Similar to (Farabet et al., 2013), (Schulz & Behnke, 2012) proposed a similar multiscale convolutional architecture. In their approach, the authors smooth out the predicted labels with pairwise class filters.

Compared to existing approaches, our method does not rely on any task-specific feature (see Table 1). Furthermore, our scene labeling system is able to extract relevant contextual information from raw pixels.

3. Systems Description

We formally introduce convolutional neural networks (CNNs) in Section 3.1 and we discuss how to capture long

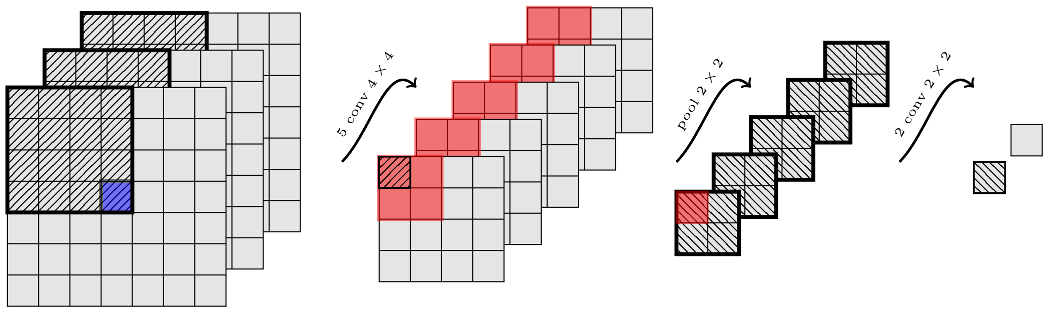

Figure 1. A simple convolutional network. Given an image patch providing a context around a pixel to classify (here blue), a series of convolutions and pooling operations (filters slid through input planes) are applied (here, five 4×4 convolutions, followed by one 2×2 pooling, followed by two 2×2 convolutions. Each 1×1 output plane is interpreted as a score for a given class.

range label dependencies with these types of models, while keeping a tight control over the capacity. Section 3.2 introduces our recurrent network approach for scene labeling. Finally, in Section 3.3, we show how to infer the full scene labeling in an efficient manner.

3.1. Convolutional Neural Networks for Scene Labeling

Convolutional neural networks (LeCun, 1989) are a natural extension of neural networks for treating images. Their architecture, somewhat inspired by the biological visual system, possesses two key properties that make them extremely useful for image applications: spatially shared weights and spatial pooling. These kind of networks learn features that are shift-invariant, *i.e.*, filters that are useful across the entire image (due to the fact that image statistics are stationary). The pooling layers are responsible for reducing the sensitivity of the output to slight input shift and distortions. This type of neural network has been shown to be very efficient in many vision applications, such as object recognition, segmentation and classification (LeCun et al., 1990; Jarrett et al., 2009; Turaga et al., 2010; Krizhevsky et al., 2012).

A typical convolutional network is composed of multiple stages, as shown in Figure 1. The output of each stage is made of a set of 2D arrays called feature maps. Each feature map is the outcome of one convolutional (or pooling) filter applied over the full image. A non-linear activation function (such as a hyperbolic tangent) always follows a pooling layer.

In the context of scene labeling, given an image I_k we are interested in finding the label of each pixel at location (i, j) in the image. More precisely, the network is fed with a squared *context patch* $I_{i,j,k}$ surrounding the pixel at location (i, j) in the k^{th} image. It can be shown (see Figure 1) that the output plane size sz_m of the m^{th} convolution or pooling layer is computed as:

$$sz_m = \frac{sz_{m-1} - kW_m}{dW_m} + 1\,, \tag{1}$$

where sz_0 is the input patch size, kW_m is the size of the convolution (or pooling) kernels in the m^{th} layer, and dW_m is the pixel step size used to slide the convolution (or pooling) kernels over the input planes.[1] Given a network architecture and an input image, one can compute the output image size by successively applying (1) on each layer of the network. During the training phase, the size of the input patch $I_{i,j,k}$ is chosen carefully such that the output layers produces 1×1 planes, which are then interpreted as scores for each class of interest.

Adopting the same notation as (Farabet et al., 2013), the output of a network f with M stages and trainable parameters $(\mathbf{W}, \mathbf{b})$, for a given input patch $I_{i,j,k}$ can be formally written as:

$$f(I_{i,j,k}; (\mathbf{W}, \mathbf{b})) = \mathbf{W}_M \mathbf{H}_{M-1}\,, \tag{2}$$

with the output of the m^{th} hidden layer computed as:

$$\mathbf{H}_m = \tanh(\text{pool}(\mathbf{W}_m \mathbf{H}_{m-1} + \mathbf{b}_m))\,, \tag{3}$$

for $m = \{1, ..., M\}$ and denoting $\mathbf{H}_0 = I_{i,j,k}$. $\mathbf{b}_m$ is the bias vector of layer m and $\mathbf{W}_m$ is the Toeplitz matrix of connection between layer $m-1$ and layer m. The pool$(\cdot)$ function is the max-pooling operator and $\tanh(\cdot)$ is the point-wise hyperbolic tangent function applied at each point of the feature map.

The network is trained by transforming the scores $f_c(I_{i,j,k}; (\mathbf{W}, \mathbf{b}))$ (for each class of interest $c \in \{1, ..., N\}$) into conditional probabilities, by applying a *softmax* function:

$$p(c|I_{i,j,k}; (\mathbf{W}, \mathbf{b})) = \frac{e^{f_c(I_{i,j,k};(\mathbf{W},\mathbf{b}))}}{\sum\limits_{d \in \{1,...,N\}} e^{f_d(I_{i,j,k};(\mathbf{W},\mathbf{b}))}}\,, \tag{4}$$

[1]Most people use $dW = 1$ for convolutional layers, and $dW = kW$ for pooling layers.

and maximizing the likelihood of the training data. More specifically, the parameters $(\mathbf{W}, \mathbf{b})$ of the network $f(\cdot)$ are learned in an end-to-end supervised way, by minimizing the negative log-likelihood over the training set:

$$L_f(\mathbf{W}, \mathbf{b}) = -\sum_{I_{(i,j,k)}} \ln p(l_{i,j,k} | I_{i,j,k}; (\mathbf{W}, \mathbf{b}))\,, \qquad (5)$$

where $l_{i,j,k}$ is the correct pixel label class at position (i, j) in image I_k. The minimization is achieved with the Stochastic Gradient Descent (SGD) algorithm with a fixed learning rate λ:

$$\mathbf{W} \longleftarrow \mathbf{W} - \lambda \frac{\partial L_f}{\partial \mathbf{W}} \;;\; \mathbf{b} \longleftarrow \mathbf{b} - \lambda \frac{\partial L_f}{\partial \mathbf{b}}\,. \qquad (6)$$

Scene labeling systems leverage long range label dependencies in some way. The most common approach is to add some kind of graphical model (*e.g.* a conditional random field) over local decisions, such that a certain global coherence is maintained. In the case of convolutional networks, an obvious way to efficiently capture long range dependencies would be to consider large input patches when labeling a pixel. However, this approach might face generalization issues, as considering larger context often implies considering larger models (*i.e.* higher capacity).

In Table 2, we review possible ways to control the capacity of a convolutional neural network by assuming a large input context. The easiest way is probably to increase the filter sizes in pooling layers, reducing the overall number of parameters in the network. However, performing large poolings decreases the network label output resolution (*e.g.*, if one performs a $1/8$ pooling, the label output plane size will be about $1/8^{th}$ of the input image size). As shown later in Section 3.3 this problem could be overcomed at the cost of a slow inference process.

Yet another approach would be the use of a *multiscale* convolutional network (Farabet et al., 2013). Large contexts are integrated into local decisions while making the model still manageable in terms of parameters/dimensionality. Label coherence can then be increased by leveraging, for instance, superpixels.

Another way to consider a large input context size while controlling the capacity of the model is to make the network recurrent. In this case, the architecture might be very deep (with many convolution layers), but parameters between several layers at various depths are *shared*. We will now detail our recurrent network approach.

3.2. Recurrent Network Approach

The recurrent architecture (see Figure 2) consists of the *composition* of P instances of the "plain" convolutional network $f(\cdot)$ introduced in Section 3.1. Each instance has

Table 2. Long range pixel label dependencies integration in CNN-based scene labeling systems. Methods to control capacity and speed of each architecture is reported.

MEANS	CAPACITY CONTROL	SPEED
GRAPHICAL MODEL	–	SLOW
MULTISCALE	SCALE DOWN INPUT IMAGE	FAST
LARGE INPUT PATCHES	INCREASE POOLING	SLOW
	RECURRENT ARCHITECTURE	FAST

identical (shared) trainable parameters $(\mathbf{W}, \mathbf{b})$. For clarity, we drop the $(\mathbf{W}, \mathbf{b})$ notation in subsequent paragraphs. The p^{th} instance of the network $(1 \leq p \leq P)$ is fed with an input "image" $\mathbf{F}^p$ of $N + 3$ features maps

$$\mathbf{F}^p = [f(\mathbf{F}^{p-1}), I^p_{i,j,k}], \qquad \mathbf{F}^1 = [\mathbf{0}, I_{i,j,k}].$$

which are the output label planes of the previous instance, and the scaled[2] version of the raw RGB squared patch surrounding the pixel at location (i, j) of the training image k. Note that the first network instance takes 0 label maps as previous label predictions.

As shown in Figure 2, the size of the input patch $I_{i,j,k}$ needed to label one pixel increases with the number of compositions of f. However, the capacity of the system remains constant, since the parameters of each network instance are shared.

The system is trained by maximizing the likelihood

$$L(f) + L(f \circ f) + ... + L(f \circ^P f)\,, \qquad (7)$$

where $L(f)$ is a shorthand for the likelihood introduced in (5) in the case of the plain CNN, and $\circ^p$ denotes the composition operation performed p times. This way, we ensure that each network instance is trained to output the correct label at location (i, j). In that respect, the system is able to learn to *correct its own mistakes* (made by earlier instances). It can also learn *label dependencies*, as an instance receives as input the label predictions made by the previous instance around location (i, j) (see Figure 2). Note that maximizing (7) is equivalent to randomly alternating (with equal weight) the maximization of each likelihood $L(f \circ^p f)$ (for $1 \leq p \leq P$). We chose this approach for simplicity of implementation.

The learning procedure is the same as for a standard CNN (stochastic gradient descent), where gradients are computed with the *backpropagation through time* (BPTT) algorithm – the network is first unfolded as shown in Figure 2 and then the standard backpropagation algorithm is applied.

[2] $I^p_{i,j,k}$ is $I_{i,j,k}$ scaled to the size of $f(F^{p-1})$.

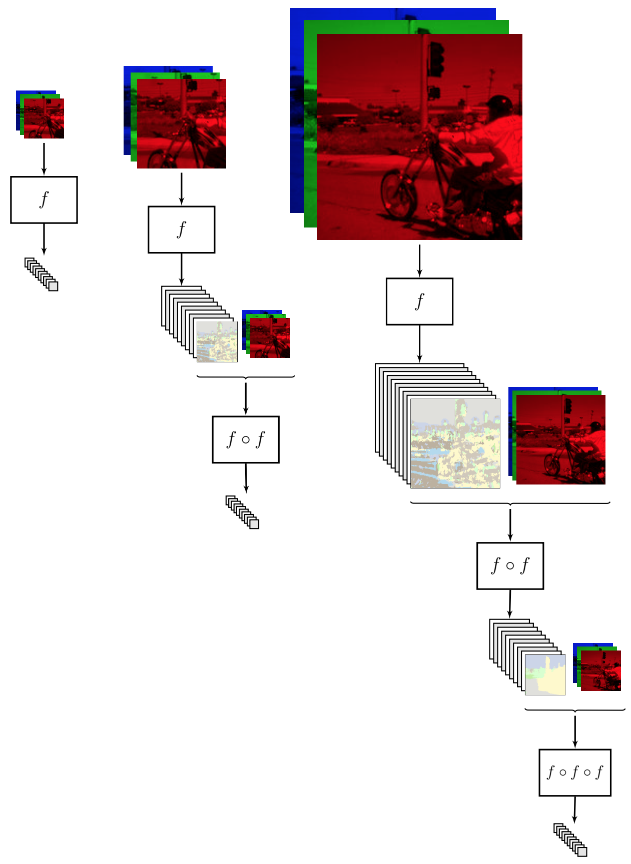

Figure 2. System considering one (f), two ($f \circ f$) and three ($f \circ f \circ f$) instances of the network. In all three cases, the architecture produces *labels* (1×1 output planes) *corresponding to the pixel at the center of the input patch.* Each network instance is fed with the previous label predictions, as well as a RGB patch surrounding the pixel of interest. For space constraints, we do not show the label maps of the first instances, as they are zero maps. Adding network instances increases the context patch size seen by the architecture (both RGB pixels and previous predicted labels).

3.3. Scene Inference

Given a test image I_k, for each pixel at location (i, j) the network predicts a label as:

$$\hat{l}_{i,j,k} = \underset{c \in \text{classes}}{\text{argmax}}\, p(c|I_{i,j,k}; (\mathbf{W}, \mathbf{b})) \,, \tag{8}$$

considering the context patch $I_{i,j,k}$. Note that this implies padding the input image when inferring label of pixels close to the image border. In practice, simply extracting patches $I_{i,j,k}$ and then feeding them through the network for all pixels of a test image is computationally very inefficient. Instead, it is better to feed the full test image (also properly padded) to the convolutional network: *applying one convolution to a large image is much faster than applying the same convolution many times to small patches.* When fed with the full input image, the network will output a plane of label scores. However, following (1), the plane size is smaller than the input image size: this is mainly due to pooling layers, but also due to border effects when applying the convolution. For example, if the network includes two 2×2 pooling layers, only 1 every 4 pixels of the input image will be labeled. Most convolutional network users (see for e.g. Farabet et al., 2013) upscale the label plane to the input image size.

In fact, it is possible to compute efficiently the label plane with a fine resolution by feeding to the network several versions of the input image, shifted on the X and Y axis. Figure 3 shows an example for a network which would have only one 2×2 pooling layer, and one output plane: low resolution label planes (coming out of the network for the input image shifted by $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$ pixels) are "merged" to form the high resolution label plane. Merging is a simple copy operation which matches a pixel in a low resolution label plane with the location of the corresponding original pixel to label in the (high resolution) input plane. The number of forwards is proportional to the number of pooling layers. However, this would be still much faster than forwarding patches at each location of the test image. We will see in Section 4.3 that having a finer label resolution can increase the classification performance.

4. Experiments

We tested our proposed method on two different fully-labeled datasets: the Stanford Background (Gould et al., 2009) and the SIFT Flow Dataset (Liu et al., 2011). The Stanford dataset has 715 images from rural and urban scenes composed of 8 classes. The scenes have approximately 320×240 pixels. As in (Gould et al., 2009), we performed a 5-fold cross-validation with the dataset randomly split into 572 training images and 143 test images in each fold. The SIFT Flow is a larger dataset composed of 2688 images of 256×256 pixels and 33 semantic labels. All the algorithms and experiments were implemented using Torch7 (Collobert et al., 2012).

Each image of the training set was properly padded and normalized such that they have zero mean and unit variance. All networks were trained by sampling patches surrounding a randomly chosen pixel from a randomly chosen image from the training set. Contrary to (Farabet et al., 2013) (i) we did not consider addition of any distortion on the images[3], (ii) we did not use contrastive normalization and (iii) we did not sample training patches according to balanced class frequencies.

[3]Which is known to improve the generalization accuracy by few extra percents.

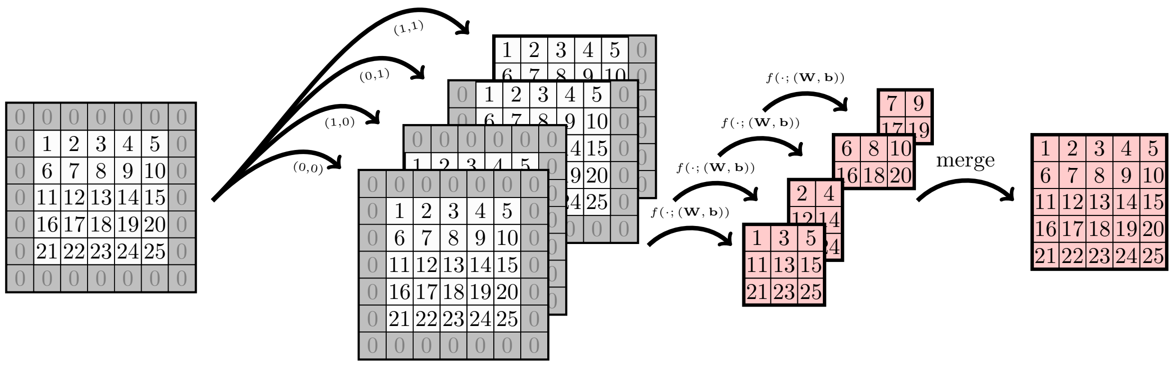

Figure 3. Convolutional neural networks output *downscaled label planes* (compared to the input image) due to pooling layers. To alleviate this problem, one can feed several shifted version of the input image (here represented by pixels 1...25) in the X and Y axis. In this example the network is assumed to have a single 2×2 pooling layer. Downscaled predicted label planes (here in red) are then merged to get back the full resolution label plane in an efficient manner. Note that pixels represented by 0 are adequate padding.

Table 3. Pixel and averaged per class accuracy and computing time of other methods and our proposed approaches on the Stanford Background Dataset. For recurrent networks, $\circ^n$ indicates the number of compositions.

METHOD	PIXEL/CLASS ACCURACY (%)	COMPUTING TIME (S)
(GOULD ET AL., 2009)	76.4 / -	10 TO 600
(TIGHE & LAZEBNIK, 2010)	77.5 / -	10 TO 300
(MUNOZ ET AL., 2010)[‡]	76.9 / 66.2	12
(KUMAR & KOLLER, 2010)	79.4 / -	< 600
(SOCHER ET AL., 2011)	78.1 / -	?
(LEMPITSKY ET AL., 2011)	81.9 / 72.4	> 60
(FARABET ET AL., 2013)[⋆]	78.8 / 72.4	0.6
(FARABET ET AL., 2013)[†]	81.4 / 76.0	60.5
PLAIN CNN$_1$	79.4 / 69.5	15
CNN$_2$ ($\circ^1$)	67.9 / 58.0	0.2
RCNN$_2$ ($\circ^2$)	79.5 / 69.5	2.6
CNN$_3$ ($\circ^1$)	15.3 / 14.7	0.06
RCNN$_3$ ($\circ^2$)	76.2 / 67.2	1.1
RCNN$_3$ 1/2 RESOLUTION ($\circ^3$)	79.8 / 69.3	2.15
RCNN$_3$ 1/1 RESOLUTION ($\circ^3$)	80.2 / 69.9	10.7

⋆ Multiscale CNN + superpixels
† Multiscale CNN + CRF
‡ Unpublished improved results have been recently reported by the authors

We considered two different accuracy measures to compare the performance of the proposed approach with other approaches. The first one is the accuracy per pixel of test images. This measure is simply the ratio of correct classified pixels of all images in the test set. However, in scene labeling (especially in datasets with large number of classes), classes which are much more frequent than others (*e.g.* the class "sky" is much more frequent than "moon") have more impact on this measure. Recent papers also consider the averaged per class accuracy on the test set (all classes have the same weight in the measure). Note that as mentioned above, we did not train with balanced class frequencies, which would have optimized this second measure.

Table 4. Pixel and averaged per class accuracy of other methods and our proposed approaches on the SIFT Flow Dataset. For recurrent networks, $\circ^n$ indicates the number of compositions.

METHOD	PIXEL/CLASS ACCURACY (%)
(LIU ET AL., 2011)	76.67 / -
(TIGHE & LAZEBNIK, 2013)	77.0 / 30.1
(FARABET ET AL., 2013)	78.5 / 29.6
PLAIN CNN$_1$	76.5 / 30.0
CNN$_2$ ($\circ^1$)	51.8 / 17.4
RCNN$_2$ ($\circ^2$)	76.2 / 29.2
RCNN$_3$ ($\circ^2$)	65.5 / 20.8
RCNN$_3$ ($\circ^3$)	77.7 / 29.8

We consider three CNNs architectures. A "plain CNN$_1$" was designed to take large input patches. CNN$_2$ and CNN$_3$ architectures were designed such that their recurrent versions (with respectively two or three compositions) would still lead to a reasonable input patch size. We denote rCNN$_i$ for the recurrent version of the regular convolutional network CNN$_i$. For rCNN$_3$, we show results considering both half resolution and full-resolution inference (see Section 3.3), in which we are able to achieve better results (at the cost of a higher computing time). Table 3 compares the performance of our architectures with related works on the Stanford Background Dataset and Table 4 compares the performance on the SIFT Flow Dataset. Note that the inference time in the second dataset does not change, since we exclude the need of any segmentation method. In the following, we provide additional technical details for each architecture used.

4.1. Plain Network

CNN$_1$ was trained with 133×133 input patches. The network was composed of a 6×6 convolution with nhu_1 output planes, followed by a 8×8 pooling layer, a $\tanh(\cdot)$ non-linearity, another 3×3 convolutional layer with nhu_2 output planes, a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, and a final 7×7 convolution to produce label scores. The hidden units were chosen to be $nhu_1 = 25$ and $nhu_2 = 50$ for the Stanford dataset, and $nhu_1 = 50$ and $nhu_2 = 50$ for the SIFT Flow dataset.

4.2. Recurrent Architectures

We consider two different recurrent convolutional network architectures.

The first architecture, rCNN$_2$, is composed of two consecutive instances of the convolutional network CNN$_2$ with shared parameters (system in the center of Figure 2). CNN$_2$ is composed of a 8×8 convolution with 25 output planes, followed by a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, another 8×8 convolutional layer with 50 output planes, a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, and a final 1×1 convolution to produce N label scores. As described in Section 3.2, rCNN$_2$ is trained by maximizing the likelihood given in (7). As shown in Figure 2, the input context patch size depends directly on the number of network instances in the recurrent architecture. In the case of rCNN$_2$, the input patch size is 25×25 when considering one instance (f) and 121×121 when considering two network instances ($f \circ f$).

The second recurrent convolutional neural network rCNN$_3$ is composed of a maximum of three instances of the convolutional network CNN$_3$ with shared parameters. Each instance of CNN$_3$ is composed of a 8×8 convolution with 25 output planes, followed by a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, another 8×8 convolution with 50 planes and a final 1×1 convolution which outputs the N label planes. Following (7), we aim at maximizing

$$L(f) + L(f \circ f) + L(f \circ f \circ f) \,. \tag{9}$$

This appeared too slow to train on a single computer in the case of rCNN$_3$. Instead, we initialized the system by first starting training with two network instances (maximizing $L(f \circ f)$). We then switched to the training of the full cost function (9). The input patch size is 23×23, 67×67 and 155×155 when considering one, two or three instances of the network (f, $f \circ f$ and $f \circ f \circ f$), respectively.

Figure 4 illustrates inference of the recurrent network with one and two instances. It can be seen that the network learns itself how to correct its own label prediction.

In all cases, the learning rate in (6) was equal to 10^{-4}. All hyper-parameters were tuned with a 10% held-out validation data.

4.3. Compute Time and Scene Inference

In Table 5, we analyze the trade off between computing time and test accuracy by running several experiments with different output resolutions for recurrent network rCNN$_3$ (see Section 3.3 and Figure 3). Labeling about $1/4^{th}$ of the pixels seems to be enough to lead to near state-of-the-art performance, while keeping a very fast inference time.

Table 5. Computing time and performance in pixel accuracy for the recurrent convolutional network rCNN$_3$ with different label resolution on the Stanford dataset. Our algorithms were run on a 4-core Intel i7.

Output Resolution	Computing Time Per Image	Pixel Accuracy
1/8	0.20s	78.4%
1/4	0.70s	79.3%
1/2	2.15s	79.8%
1/1	10.68s	80.2%

5. Conclusion

This paper presented a novel *feed-forward* approach for full scene labeling based on supervised deep learning strategies which model in a rather simple way non-local class dependencies in a scene from raw pixels. We demonstrated that the problem of scene labeling can be effectively achieved without the need of any expensive graphical model or segmentation technique to ensure labeling. The scene labeling is inferred simply by forward evaluation of a function applied to a RGB image.

In terms of accuracy, our system achieves state-of-the-art results on both Stanford Background and SIFT Flow datasets, while keeping a fast inference time. Future work includes investigation of unsupervised or semi-supervised pre-training of the models, as well as application to larger datasets such as the Barcelona dataset.

Acknowledgments

The authors thank the reviewers for their useful feedback and comments. This work was supported by the Swiss NSF through the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (www.im2.ch).

References

Collobert, R., Kavukcuoglu, K., and Farabet, C. Implementing neural networks efficiently. In *Neural Networks: Tricks of the Trade*. Springer, 2012.

Elman, J. L. Finding structure in time. In *Cognitive Sciences*, 1990.

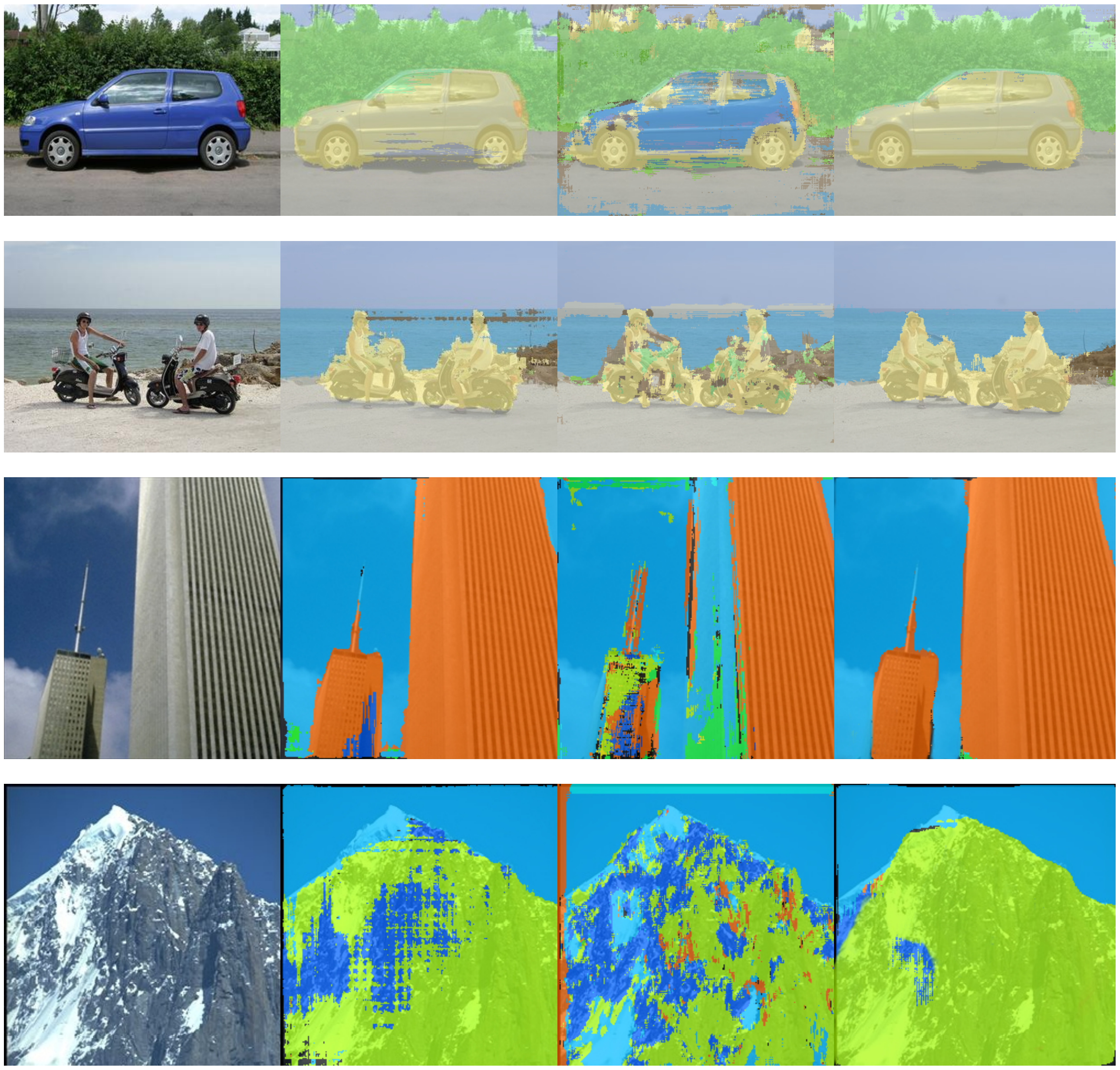

Figure 4. Inference results of our architectures. The two first examples (rows) are from the Stanford Background Dataset and the two last ones are from the SIFT Flow Dataset. First column is the input image. The second column represents the output of the "plain CNN_1" network, the third column illustrates results of $rCNN_2$ with one instance and the last column the result with the composition of two instances: most mistakes of first instance are corrected on the second one. Best viewed in color.

Farabet, C., Couprie, C., Najman, L., and LeCun, Y. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

Gould, S., Fulton, R., and Koller, D. Decomposing a scene into geometric and semantically consistent regions. In *International Conference on Computer Vision (ICCV)*, 2009.

Grangier, D., Bottou, L., and Collobert, R. Deep convolutional networks for scene parsing. In *International Conference on Machine Learning (ICML) Deep Learning Workshop*, 2009.

Graves, A. and Schmidhuber, J. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

Jarrett, K., Kavukcuoglu, K., Ranzato, MA., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *Proceedings International Conference on Computer Vision (ICCV'09)*, 2009.

Jordan, M. I. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Kumar, M.P. and Koller, D. Efficiently selecting regions for scene understanding. In *Computer Vision and Pattern Recognition (CVPR)*, 2010.

LeCun, Y. Generalization and network design strategies. In *Connectionism in Perspective*. 1989.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems (NIPS)*, 1990.

Lempitsky, V., Vedaldi, A., and Zisserman, A. A pylon model for semantic segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Liu, C., Yuen, J., and Torralba, A. Nonparametric scene parsing via label transfer. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011.

Munoz, D., Bagnell, J., and Hebert, M. Stacked hierarchical labeling. In *Proceedings European Conference on Computer Vision (ECCV)*, 2010.

Robinson, T. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5:298–305, 1994.

Schulz, H. and Behnke, S. Learning object-class segmentation with convolutional neural networks. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, 2012.

Socher, R., Lin, C., Ng, A., and Manning, C. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)*, 2011.

Socher, R., Huval, B., Bhat, B., Manning, C. D., and Ng, A. Y. Convolutional-recursive deep learning for 3d object classification. In *Advances in Neural Information Processing Systems (NIPS)*. 2012.

Stoianov, I., Nerbonne, J., and Bouma, H. Modelling the phonotactic structure of natural language words with simple recurrent networks. In *Computational Linguistics in the Netherlands*, 1997.

Tighe, J. and Lazebnik, S. Superparsing: scalable nonparametric image parsing with superpixels. In *European conference on Computer vision (ECCV)*, 2010.

Tighe, J. and Lazebnik, S. Superparsing - scalable nonparametric image parsing withsuperpixels. *International Journal of Computer Vision*, 2013.

Turaga, S. C., Murray, J. F., Jain, V., Roth, F., Helmstaedter, M., Briggman, K., Denk, W., and Seung, H. S. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 2010.

Verbeek, J. and Triggs, B. Scene segmentation with crfs learned from partially labeled images. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.