Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers

Clément Farabet [1,2] CFARABET@CS.NYU.EDU
Camille Couprie [1] CCOUPRIE@CS.NYU.EDU
Laurent Najman [2] L.NAJMAN@ESIEE.FR
Yann LeCun [1] YANN@CS.NYU.EDU
[1] Courant Institute of Mathematical Sciences, New York University, 715 Broadway, New York, NY 10003
[2] Université Paris-Est, Équipe A3SI - ESIEE Paris, 93160 Noisy-le-Grand, France

Abstract

Scene parsing consists in labeling each pixel in an image with the category of the object it belongs to. We propose a method that uses a multiscale convolutional network trained from raw pixels to extract dense feature vectors that encode regions of multiple sizes centered on each pixel. The method alleviates the need for engineered features. In parallel to feature extraction, a tree of segments is computed from a graph of pixel dissimilarities. The feature vectors associated with the segments covered by each node in the tree are aggregated and fed to a classifier which produces an estimate of the distribution of object categories contained in the segment. A subset of tree nodes that cover the image are then selected so as to maximize the average "purity" of the class distributions, hence maximizing the overall likelihood that each segment will contain a single object. The system yields record accuracies on the the Sift Flow Dataset (33 classes) and the Barcelona Dataset (170 classes) and near-record accuracy on the Stanford Background Dataset (8 classes), while being an order of magnitude faster than competing approaches, producing a 320×240 image labeling in less than 1 second, including feature extraction.

1. Overview

Scene parsing, or full scene labeling (FSL), is the task of labeling each pixel in a scene with the category of the object to which it belongs. FSL requires to solve the detection, segmentation, recognition and contextual integration problems simultaneously, so as to produce a globally consistent labeling. One of the obstacles to FSL is that the information necessary for the labeling of a given pixel may come from very distant pixels as well as their labels. The category of a pixel may depend on relatively short-range information (*e.g.* the presence of a human face generally indicates the presence of a human body nearby), as well as on very long-range dependencies (is this grey pixel part of a road, a building, or a cloud?).

This paper proposes a new method for FSL, depicted on Figure 1 that relies on five main ingredients:
1) Trainable, dense, multi-scale feature extraction: a multi-scale, dense feature extractor produces a series of feature vectors for regions of multiple sizes centered around every pixel in the image, covering a large context. The feature extractor is a three-stage convolutional network applied to a multi-scale contrast-normalized laplacian pyramid computed from the image. The convolutional network is fed with raw pixels and trained end to end, thereby alleviating the need for hand-engineered features.
2) Segmentation Tree: A graph over pixels is computed in which each pixel is connected to its 4 nearest neighbors through an edge whose weight is a measure of dissimilarity between the colors of the two pixels. A segmentation tree is then constructed using a classical region merging method, based on the minimum spanning tree of the graph. Each node in the tree corresponds to a potential image segment. The final image segmentation will be a judiciously chosen subset of nodes of the tree whose corresponding regions cover the entire image.
3) Region-wise feature aggregation: for each node in the tree, the corresponding image segment is encoded by a 3×3 spatial grid of aggregated feature vectors. The aggregated feature vector of each grid cell is computed by a component-wise max pooling of the feature vectors centered on all the pixels that fall into the grid cell; This produces a scale-invariant representation of the segment and its surrounding.
4) Class histogram estimation: a classifier is then applied to the aggregated feature grid of each node. The classifier is trained to estimate the histogram of all object categories present in its input segments.
5) **Optimal purity cover**: a subset of tree nodes is selected whose corresponding segments cover the entire image. The nodes are selected so as to maximize the average purity of

Appearing in *Proceedings of the 29^{th} International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

the class distribution. We define the class purity as a quantity that is inversely proportional to the entropy of the class distribution. The choice of the cover thus attempts to find a consistent overall segmentation in which each segment contains pixels belonging to only one of the learned categories.

All the steps in the process have a complexity linear (or almost linear) in the number of pixels. The bulk of the computation resides in the convolutional network feature extractor. The resulting system is very fast, producing a full parse of a 320×240 image in less than 1 second on a conventional CPU, and in less than 100ms using dedicated hardware, opening the door to real-time applications. Once trained, the system is parameter free, and requires no adjustment of thresholds or other knobs.

There are three key contributions in this paper **1)** using a *multi-scale convolutional net* to learn good features for region classification; While using a multiscale representation seems natural for FSL, it has rarely been used in the context of feature learning systems. **2)** using a *class purity criterion* to decide if a segment contains a single objet, as opposed to several objects, or part of an object; **3)** an efficient procedure to obtain a *cover that optimizes* the overall class purity of a segmentation.

2. Related work

The FSL problem has been approached with a wide variety of methods in recent years. Many methods rely on MRFs, CRFs, or other types of graphical models to ensure the consistency of the labeling and to account for context (He & Zemel, 2008; Russell et al., 2009; Gould et al., 2009; Kumar & Koller, 2010; Munoz et al., 2010; Tighe & Lazebnik, 2010; Lempitsky et al., 2011). Most methods rely on a pre-segmentation into super-pixels or other segment candidates, and extract features and categories from individual segments and from various combinations of neighboring segments. The graphical model inference pulls out the most consistent set of segments which covers the image.

Socher et al. (2011) propose a method to aggregate segments in a greedy fashion using a trained scoring function. The originality of the approach is that the feature vector of the combination of two segments is computed from the feature vectors of the individual segments through a trainable function. Like us, they use "deep learning" methods to train their feature extractor. But unlike us, their feature extractor operates on hand-engineered features.

One of the main question in scene parsing is how to take a wide context into account to make a local decision. Munoz et al. (2010) proposed to use the histogram of labels extracted from a coarse scale as input to the labeler that looks at finer scales. Our approach is somewhat simpler: our feature extractor is applied densely to an image pyramid. The coarse feature maps thereby generated are upsampled to match that of the finest scale. Hence with three scales, each feature vector has multiple fields which encode multiple regions of increasing sizes and decreasing resolutions,

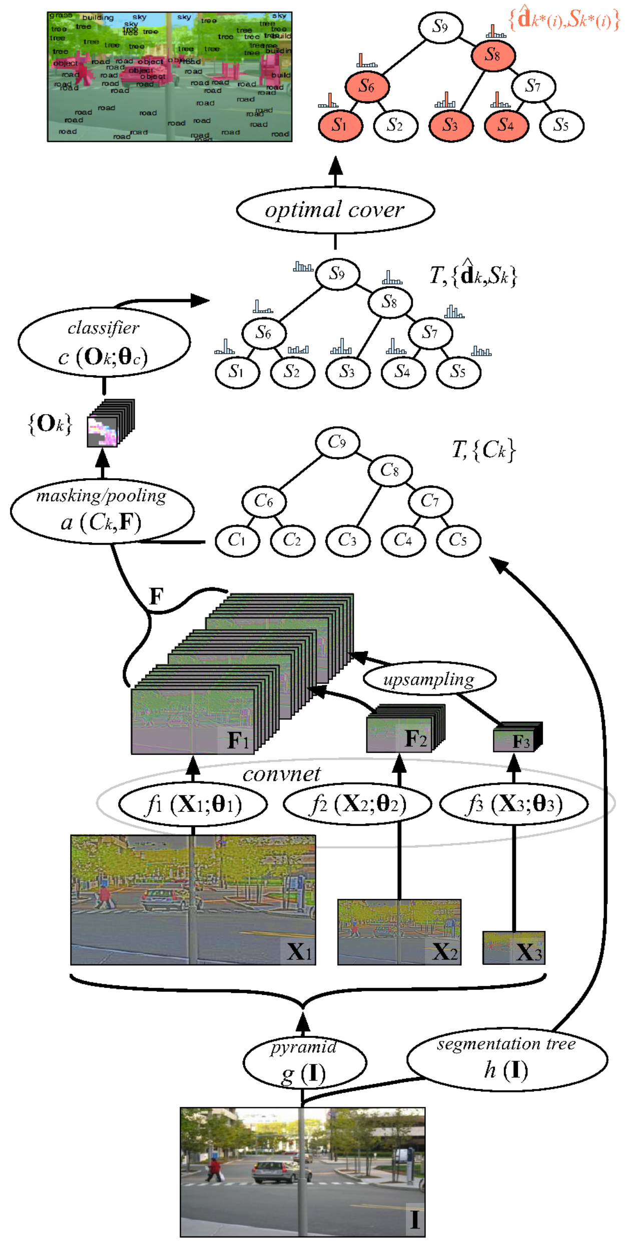

Figure 1. Diagram of the scene parsing system. The raw input image is transformed through a Laplacian pyramid. Each scale is fed to a 3-stage convolutional network, which produces a set of feature maps. The feature maps of all scales are concatenated, the coarser-scale maps being upsampled to match the size of the finest-scale map. Each feature vector thus represents a large contextual window around each pixel (184×184 in this paper). In parallel, a segmentation tree is computed via the minimum spanning tree of the dissimilarity graph of neighboring pixels. The segment associated with each node in the tree is encoded by a spatial grid of feature vectors pooled in the segment's region. A classifier is then applied to all the aggregated feature grids to produce a histogram of categories, the entropy of which measures the "impurity" of the segment. Each pixel is then labeled by the minimally-impure node above it, which is the segment that best explains the pixel's class.

centered on the same pixel location.

Like us, a number of authors have used trees to generate candidate segments by aggregating elementary segments, as in Russell et al. (2009); Lempitsky et al. (2011). These approaches rely on inference algorithms based on Graph Cuts or other methods. In this paper, we use an innovative method based on finding a set of tree nodes that covers the image while minimizing a class purity criterion, that allows us to label scenes in less that one second whereas the previously mentionned approaches require minutes.

Our system extracts features densely from a multiscale pyramid of images using a convolutional network (ConvNet) (LeCun et al., 1998a). ConvNets can be fed with raw pixels and can automatically learn low-level and mid-level features, alleviating the need for hand-engineered features. One big advantage of ConvNets is the ability to compute dense features efficiently over large images. ConvNets are best known for their applications to detection and recognition (Osadchy et al., 2007; Jarrett et al., 2009), but they have also been used for image segmentation, particularly for biological image segmentation (Ning et al., 2005; Jain et al., 2007; Turaga et al., 2009).

The only published work on using ConvNets for scene parsing is that of Grangier et al. (2009). While somewhat preliminary, their work showed that convolutional networks fed with raw pixels could be trained to perform scene parsing with decent accuracy. Unlike Grangier et al. (2009) however, our system uses a boundary-based hierarchy of segmentations to align the labels produced by the ConvNet to the boundaries in the image and thus produces representations that are independent of the size of the segments through feature pooling.

3. An end-to-end trainable model for scene parsing

The model proposed in this paper, depicted on Figure 1, relies on two complementary image representations. In the first representation, an image patch of size P is seen as a point in $\mathbb{R}^P$, and we seek to find a transform $f : \mathbb{R}^P \to \mathbb{R}^Q$ that maps each patch into $\mathbb{R}^Q$, a space where it can be classified linearly. This first representation typically suffers from two main problems when using a classical ConvNet, where the image is divided following a grid pattern: (1) the window considered rarely contains an object that is properly centered and scaled, and therefore offers a poor observation basis to predict the class of the underlying object, (2) integrating a large context involves increasing the grid size, and therefore the dimensionality P of the input; given a finite amount of training data, it is then necessary to enforce some invariance in the function f itself. This is usually achieved by using pooling/subsampling layers, which in turn degrades the ability of the model to precisely locate and delineate objects. In this paper, f is implemented by a multiscale convolutional network, which allows integrating large contexts (as large as the complete scene) into local decisions, yet still remaining manageable in terms of parameters/dimensionality. This multiscale model, in which weights are shared across scales, allows the model to capture long-range interactions, without the penalty of extra parameters to train. This model is described in Section 3.1.

In the second representation, the image is seen as an edge-weighted graph, on which a hierarchy of segmentations/clusterings can be constructed. This representation yields a natural abstraction of the original pixel grid, and provides a hierarchy of observation levels for all the objects in the image. It can be used as a solution to the first problem exposed above: assuming the capability of assessing the quality of all the components of this hierarchy, a system can automatically choose its components so as to produce the best set of predictions. Moreover, these components are spatially accurate, and naturally delineate the underlying objects, as this representation conserves pixel-level precision. Section 3.2 describes our methodology.

3.1. Scale-invariant, scene-level feature extraction

The feature extractor is a three-stage ConvNet. Each of the first two stages contains a bank of filters producing multiple feature maps, a point-wise non-linear mapping, and a spatial pooling and subsampling of each feature map; the third stage only contains a bank of filters and a point-wise non-linear mapping. The filters (convolution kernels) are subject to training. Each filter is applied to the input feature maps through a 2D convolution operation, which detects local features at all locations on the input. Each filter bank of a ConvNet produces features that are equivariant under shifts, *i.e.* if the input is shifted, the output is also shifted but otherwise unchanged.

While ConvNets have been successfully applied to a number of image labeling problems, image-level tasks such as full-scene understanding (pixel-wise labeling, or any dense feature estimation) require the system to model complex interactions at the scale of complete images, not simply within a patch. To view a large contextual window at full resolution, a ConvNet would have to be unmanageably large.

The solution is to use a multiscale approach. Our multiscale convolutional network overcomes these limitations by extending the concept of spatial weight replication to the scale space. Given an input image $\mathbf{I}$, a multiscale pyramid of images $\mathbf{X}_s,\ \forall s \in \{1, \dots, N\}$ is constructed, where $\mathbf{X}_1$ has the size of $\mathbf{I}$. The multiscale pyramid can be a Laplacian pyramid, and is typically pre-processed, so that local neighborhoods have zero mean and unit standard deviation. Given a classical convolutional network f_s with parameters θ_s, the multiscale network is obtained by instantiating one network per scale s, and sharing all parameters across scales: $\theta_s = \theta_1,\ \forall s \in \{1, \dots, N\}$.

The maps in the pyramid are computed using a scaling/normalizing function g_s as $\mathbf{X}_s = g_s(\mathbf{I})$, for all $s \in \{1, \dots, N\}$.

For each scale s, the convolutional network f_s can be described as a sequence of linear transforms, interspersed with non-linear symmetric squashing units (typically the tanh function (LeCun et al., 1998b)), and pooling/subsampling operators. For a network f_s with L layers, we have: $f_s(\mathbf{X}_s;\theta_s) = \mathbf{W}_L \mathbf{H}_{L-1}$, where the vector of hidden units at layer l is $\mathbf{H}_l = \text{pool}(\tanh(\mathbf{W}_l \mathbf{H}_{l-1} + \mathbf{b}_l))$ for all $l \in \{1, \ldots, L-1\}$, with $\mathbf{b}_l$ a vector of bias parameters, and $\mathbf{H}_0 = \mathbf{X}_s$. The matrices $\mathbf{W}_l$ are Toeplitz matrices, therefore each hidden unit vector $\mathbf{H}_l$ can be expressed as a regular convolution between kernels from $\mathbf{W}_l$ and the previous hidden unit vector $\mathbf{H}_{l-1}$, squashed through a tanh, and pooled spatially. More specifically, $\mathbf{H}_{lp} = \text{pool}(\tanh(\mathbf{b}_{lp} + \sum_{q \in \text{parents}(p)} \mathbf{w}_{lpq} * \mathbf{H}_{l-1,q}))$.

The filters $\mathbf{W}_l$ and the biases $\mathbf{b}_l$ constitute the trainable parameters of our model, and are collectively denoted θ_s. The function tanh is a point-wise non-linearity, while pool is a function that considers a neighborhood of activations, and produces one activation per neighborhood. In all our experiments, we use a max-pooling operator, which takes the maximum activation within the neighborhood. Pooling over a small neighborhood provides built-in invariance to small translations.

Finally, the outputs of the N networks are upsampled and concatenated so as to produce $\mathbf{F}$, a map of feature vectors of size N times the size of $\mathbf{F}_1$, which can be seen as local patch descriptors and scene-level descriptors: $\mathbf{F} = [\mathbf{F}_1, u(\mathbf{F}_2), \ldots, u(\mathbf{F}_N)]$, where u is an upsampling function.

As mentioned above, weights are shared between networks f_s. Intuitively, imposing complete weight sharing across scales is a natural way of forcing the network to learn scale invariant features, and at the same time reduce the chances of over-fitting. The more scales used to jointly train the models $f_s(\theta_s)$ the better the representation becomes for all scales. Because image content is, in principle, scale invariant, using the same function to extract features at each scale is justified. In fact, we observed a performance decrease when removing the weight-sharing.

3.2. Parameter-free hierarchical parsing

Predicting the class of a given pixel from its own feature vector is difficult, and not sufficient in practice. The task is easier if we consider a spatial grouping of feature vectors around the pixel, *i.e.* a neighborhood. Among all possible neighborhoods, one is the most suited to predict the pixel's class. In Section 3.2.1 we formulate the search for the most adapted neighborhood as an optimization problem. The construction of the cost function that is minimized is then described in Section 3.2.2.

3.2.1. Optimal purity cover

We define the neighborhood of a pixel as a connected component that contains this pixel. Let $C_k,\ \forall k \in \{1, \ldots, K\}$ be the set of all possible connected components of the lattice defined on image $\mathbf{I}$, and let S_k be a cost associated to each of these components. For each pixel i, we wish to find the index $k^*(i)$ of the component that best explains the class of the pixel, that is, the component with the minimal cost $S_{k^*(i)}$:

$$k^*(i) = \underset{k \mid i \in C_k}{\operatorname{argmin}} S_k \tag{1}$$

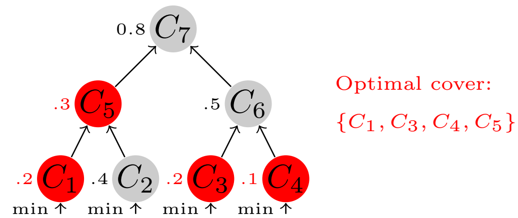

Figure 2. Finding the optimal cover. For each pixel (leaf) i, the optimal component $C_{k^*(i)}$ is the one along the path between the leaf and the root with minimal cost $S_{k^*(i)}$. The optimal cover is the union of all these components. In this example, the optimal cover $\{C_1, C_3, C_4, C_5\}$ will result in a segmentation in disjoint sets $\{C_1, C_2, C_3, C_4\}$, with the subtle difference that component C_2 will be labelled with the class of C_5, as C_5 is the best observation level for C_2.

Note that components $C_{k^*(i)}$ are non-disjoint sets that form a cover of the pixels (lattice). Note also that the overall cost $S^* = \sum_i S_{k^*(i)}$ is minimal.

In practice, the set of components C_k is too large, and only a subset of it can be considered. A classical technique to reduce the set of components is to consider a hierarchy of segmentations (Najman & Schmitt, 1996; Arbeláez et al., 2011), that can be represented as a tree T. Solving Eq 1 on T can be done simply by exploring the tree in a depth-first search manner, and finding the component with minimal weight along each branch. Figure 2 illustrates the procedure.

3.2.2. Producing the confidence costs

Given a set of components C_k, we explain how to produce all the confidence costs S_k. These costs represent the class purity of the associated components. Given the groundtruth segmentation, we can compute the cost as being the entropy of the distribution of classes present in the component. At test time, when no groundtruth is available, we need to define a function that can predict this cost by simply looking at the component. We now describe a way of achieving this, as illustrated in Figure 3.

Given the feature vectors in $\mathbf{F}$, we define a compact representation to describe objects as an adaptive spatial arrangement of such features. In other terms, an object, or category in general, can be best described as a spatial arrangement of features, or parts. We define a simple attention function a used to mask the feature vector map with each component C_k, producing a set of K masked feature vector patterns $\{\mathbf{F} \bigcap C_k\},\ \forall k \in \{1, \ldots, K\}$. The function a is called an attention function because it suppresses the background around the component being analyzed. The patterns $\{\mathbf{F} \bigcap C_k\}$ are resampled to produce fixed-size representations. In our model the sampling is done using an adaptive

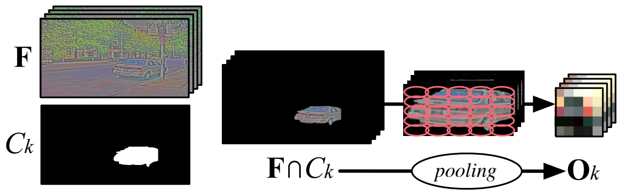

Figure 3. The shape-invariant attention function a. For each component C_k in the segmentation tree T, the corresponding image segment is encoded by a spatial grid of feature vectors that fall into this segment. The aggregated feature vector of each grid cell is computed by a component-wise max pooling of the feature vectors centered on all the pixels that fall into the grid cell; this produces a scale-invariant representation of the segment and its surroundings. The result, $\mathbf{O}_k$, is a descriptor that encodes spatial relations between the underlying object's parts. The grid size was set to 3×3 for all our experiments.

max-pooling function, which remaps input patterns of arbitrary size into a fixed $G \times G$ grid. This grid can be seen as a highly invariant representation that encodes spatial relations between an object's attributes/parts. This representation is denoted $\mathbf{O}_k$. Some nice properties of this encoding are: (1) elongated, or in general ill-shaped objects, are nicely handled, (2) the dominant features are used to represent the object, combined with background subtraction, the features pooled represent solid basis functions to recognize the underlying object.

Once we have the set of object descriptors $\mathbf{O}_k$, we define a classifier function $c : \mathbf{O}_k \rightarrow [0,1]^{N_c}$ (where N_c is the number of classes) as predicting the distribution of classes present in component C_k. We associate a cost S_k to this distribution. In this paper, c is implemented as a simple 2-layer neural network, and S_k is the entropy of the predicted distribution. More formally, let $\mathbf{x}_k$ be the feature vector associated with component C_k, $\hat{\mathbf{d}}_k$ the predicted class distribution, and S_k the cost associated to this distribution. We have

$$\begin{aligned} \mathbf{y}_k &= \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{x}_k + \mathbf{b}_1), && (2) \\ \hat{\mathbf{d}}_{k,a} &= \frac{e^{\mathbf{y}_{k,a}}}{\sum_{b \in \text{classes}} e^{\mathbf{y}_{k,b}}}, && (3) \\ S_k &= -\sum_{a \in \text{classes}} \hat{\mathbf{d}}_{k,a} \ln(\hat{\mathbf{d}}_{k,a}). && (4) \end{aligned}$$

Matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ are noted θ_c, and represent the trainable parameters of c. These parameters need to be learned over the complete set of hierarchies, computed on the entire training set available. The exact training procedure is described in Section 4.

4. Training procedure

Let $\mathcal{F}$ be the set of all feature maps in the training set, and $\mathcal{T}$ the set of all hierarchies. Training the model described in Section 3 can be done in two steps. First, we train the low-level feature extractor f in complete independence of the rest of the model. The goal of that first step is to produce features $(\mathbf{F})_{\mathbf{F} \in \mathcal{F}}$ that are maximally discriminative for pixelwise classification. Next, we construct the hierarchies $(T)_{T \in \mathcal{T}}$ on the entire training set, and, for all $T \in \mathcal{T}$ train the classifier c to predict the distribution of classes in component $C_k \in T$, as well as the costs S_k. Once this second part is done, all the functions in Figure 1 are defined, and inference can be performed on arbitrary images. In the next two sections we describe these two steps.

4.1. Learning discriminative scale-invariant features

As described in Section 3.1, feature vectors in $\mathbf{F}$ are obtained by concatenating the outputs of multiple networks f_s, each taking as input a different image in a multiscale pyramid. Ideally a linear classifier should produce the correct categorization for all pixel locations i, from the feature vectors $\mathbf{F}_i$. We train the parameters θ_s to achieve this goal, using the multiclass *cross entropy* loss function. Let $\hat{\mathbf{c}}_i$ be the normalized prediction vector from the linear classifier for pixel i. We compute normalized predicted probability distributions over classes $\hat{\mathbf{c}}_{i,a}$ using the *softmax* function, *i.e.*

$$\hat{\mathbf{c}}_{i,a} = \frac{e^{\mathbf{w}_a^T \mathbf{F}_i}}{\sum_{b \in \text{classes}} e^{\mathbf{w}_b^T \mathbf{F}_i}}, \tag{5}$$

where $\mathbf{w}$ is a temporary weight matrix only used to learn the features. The cross entropy between the predicted class distribution $\hat{\mathbf{c}}$ and the target class distribution $\mathbf{c}$ penalizes their deviation and is measured by

$$L_{\text{cat}} = -\sum_{i \in \text{pixels}} \sum_{a \in \text{classes}} \mathbf{c}_{i,a} \ln(\hat{\mathbf{c}}_{i,a}). \tag{6}$$

The true target probability $\mathbf{c}_{i,a}$ of class a to be present at location i can either be a distribution of classes at location i, in a given neighborhood or a hard target vector: $\mathbf{c}_{i,a} = 1$ if pixel i is labeled a, and 0 otherwise. For training maximally discriminative features, we use hard target vectors in this first stage. Once the parameters θ_s are trained, we discard the classifier in Eq 5.

4.2. Teaching a classifier to find its best observation level

Given the trained parameters θ_s, we build $\mathcal{F}$ and $\mathcal{T}$, *i.e.*we compute all vector maps $\mathbf{F}$ and hierarchies T on all the training data available, so as to produce a new training set of descriptors $\mathbf{O}_k$. This time, the parameters θ_c of the classifier c are trained to minimize the KL-divergence between the true (known) distributions of labels $\mathbf{d}_k$ in each component, and the prediction from the classifier $\hat{\mathbf{d}}_k$ (Eq 3):

$$l_{\text{div}} = \sum_{a \in \text{classes}} \hat{\mathbf{d}}_{k,a} \ln(\frac{\hat{\mathbf{d}}_{k,a}}{\mathbf{d}_{k,a}}). \tag{7}$$

In this setting, the groundtruth distributions $\mathbf{d}_k$ are not hard target vectors, but normalized histograms of the labels present in component C_k. Once the parameters θ_c are

	P / C	CT (sec.)
Gould et al. (2009)	76.4% / -	10 to 600s
Munoz et al. (2010)	76.9% / 66.2%	12s
Tighe & Lazebnik (2010)	77.5% / -	10 to 300s
Socher et al. (2011)	78.1% / -	? [2]
Kumar & Koller (2010)	79.4% / -	< 600s
Lempitsky et al. (2011)	**81.9%** / 72.4%[1]	> 60s
singlescale convnet	66.0 % / 56.5 %	0.3s
multiscale convnet	77.5 % / 70.0%	0.5s
multiscale net + cover	79.5% / **74.3%**	**1s**

Table 1. Performance of our system on the Stanford Background dataset (Gould et al., 2009): per-pixel / average per-class accuracy. The third column reports compute times, as reported by the authors. Our algorithms were computed using a 4-core Intel i7. [1] We would like to thank Victor Lempitsky who kindly provided us with his classification results. [2]Socher et al. (2011) only provide compute time for their inference (110ms), omitting the time to compute features.

	P / C
Liu et al. (2009)	74.75 % / -
Tighe & Lazebnik (2010)	76.9 % / 29.4 %
multiscale net + cover[1]	**78.5 % / 29.6 %**
multiscale net + cover[2]	74.2 % **/ 46.0 %**

Table 2. Performance of our system on the SIFT Flow dataset (Liu et al., 2009) Our multiscale network is trained using two sampling methods: [1]natural frequencies, [2]balanced frequencies.

trained, $\hat{\mathbf{d}}_k$ accurately predicts the distribution of labels, and Eq 4 is used to assign a purity cost to the component.

Note that the parameters of the feature extractor are not updated while training this classifier. This has been tried, and, interestingly, gave results that were slightly worse. We believe this is caused by the fact that learning all the parameters of the complete system jointly is more prone to overfitting. Moreover, the feature extractor and the classifier are trained on the same data, which can also be suboptimal in terms of generalization, as mentioned in (Munoz et al., 2010).

5. Experiments

We report our semantic scene understanding results on three different datasets: "Stanford Background" on which related state-of-the-art methods report classification errors, and two more challenging datasets with a larger number of classes: "SIFT Flow" and "Barcelona". The Stanford Background dataset Gould et al. (2009) contains 715 images of outdoor scenes composed of 8 classes, where each image contains at least one foreground object. We use the evaluation procedure introduced in Gould et al. (2009), 5-fold cross validation: 572 images used for training, and 143 for testing. The SIFT Flow dataset Liu et al. (2009) is composed of $2,688$ images, $2,488$ training images and 200 test images containing 33 semantic labels. The Barcelona dataset, as described in Tighe & Lazebnik (2010), is derived from the LabelMe subset and contains 170 unique labels. It has $14,871$ training and 279 test images. The test set consists of street scenes from Barcelona, while the training set ranges in scene types but has no street scenes from Barcelona.

For all experiments, we use a 3-stage convolutional network. Each layer of the network is composed of a bank of filters of size 7×7 followed by $\tanh$ units and 2×2 max-pooling operations. The input image $\mathbf{I}$ is transformed into a 16-dimension feature map, using a bank of 16 filters, the second layer transforms the 16-dimension feature map into a 64-dimension feature map, each component being produced by a combination of 8 filters, finally the 64-dimension feature map is transformed into a 256-dimension feature map, using a combination of 16 filters. The network is applied to a locally normalized Laplacian pyramid constructed on the input image. For these experiments, the pyramid consists of 3 rescaled versions of the input. All inputs are properly padded, and outputs of each of the 3 networks upsampled and concatenated, so as to produce a $256 \times 3 = 768$-dimension feature vector map $\mathbf{F}$. The field-of-view of the network at each scale is 46×46, such that the complete quarter-resolution field-of-view is 184×184. The network is trained on all 3 scales in parallel. Simple grid-search was performed to find the best learning rate and regularization parameters , using a holdout of 10% of the training dataset for validation. To ensure that the features do not overfit some irrelevant biases present in the data, jitter – horizontal flipping of all images, and rotations between -8 and 8 degrees – was used to artificially expand the size of the training data.

The segmentation tree used to find the optimal cover is computed from a 4-connexity graph built on the raw (unnormalized) RGB pixels. The edge weights are set to the Euclidean distance between two pixels. A minimum spanning tree is then computed, and the dendrogram of that spanning tree is our segmentation tree. Each edge in that tree is the weakest edge between the two components. Finally, the tree is filtered according to a morphologic volumetric criterion (Meyer & Najman, 2010; Cousty & Najman, 2011), completed by a removal of non-informative small components (less than 100 pixels). These two operations help produce larger, more stable components.

Classically segmentation methods find a partition of the segments rather than a cover. Partitioning the segments consists in finding an optimal cut in the tree (so that each terminal node in the pruned tree corresponds to a segment). We experimented with a number of graph based methods to do so, including graph-cuts (Ford & Fulkerson, 1955; Boykov & Jolly, 2001), but the results were less accurate than with our optimal cover method.

The $2-$layer neural network c (Eq 2) has $3 \times 3 \times 3 \times 256$ input units (using a 3×3 grid of feature vectors from $\mathbf{F}$), 512 hidden units; and the size of the output unit corresponds to the number of different classes in the dataset.

To evaluate the gain of each specific components of our purity tree approach, we report on the Stanford dataset three experiments: a system based on a convolutional network alone; the multiscale convolutional network, and the full model as described in Section 3. Results are reported in Table 1, and compared with related works. Our model achieves very good results in comparison with previous approaches. Methods of (Kumar & Koller, 2010; Lempitsky et al., 2011) achieve similar or better performances on this particular dataset but to the price of several minutes to parse one image. We demonstrate that our system scales nicely when augmenting the number of classes on two other datasets, in Tables 2 and 3. Example parses on the SIFT Flow dataset are shown on Figure 4.

Network and multiscale network: for our baseline, we trained a single-scale network and a three-scale network as raw site predictors, for each location i, using the classification loss L_{cat} defined in Eq 6. Table 1 shows the clear advantage of the multi-scale representation, which captures scene-level dependencies, and can classify more pixels accurately. Without an explicit segmentation model, the visual aspect of the predictions still suffers from poor spatial consistency, and poor object delineation.

Complete system, network and hierarchy: in this experiment, we use the complete model, as described in Section 3. Results are significantly better than the baseline method, in particular, much better delineation is achieved.

For the SIFT Flow dataset, we experimented with two sampling methods when learning the multiscale features: respecting natural frequencies of classes, and balancing them so that an equal amount of each class is shown to the network. Both results are reported in Table 2. Training with balanced frequencies allows better discrimination of small objects, and although it decreases the overall pixelwise accuracy, it is more correct from a recognition point of view. Frequency balancing is used on the Stanford Background dataset, as it consistently gives better results. For the Barcelona dataset, both sampling methods are used as well, but frequency balancing worked rather poorly in that case. This could be explained by the fact that this dataset has a large amount of classes with very few training examples. These classes are therefore extremely hard to model, and overfitting occurs much faster than for the SIFT Flow dataset. Results are shown on Table 3.

Results in Table 1 demonstrate the impressive computational advantage of convolutional networks over competing algorithms. Exploiting the parallel structure of this special network, by computing convolutions in parallel, allows us to parse an image of size 320×240 in less than one second on a 4-core Intel i7 laptop. Our scene parsing method shows promise of real time applications, which would constitute a breakthrough in the fields of machine learning and computer vision. Training time is also remarkably fast: results on the Stanford dataset were typically obtained in 48h on a regular server.

	P / C
Tighe & Lazebnik (2010)	66.9 % / 7.6 %
multiscale net + cover[1]	**67.8 % / 9.5 %**
multiscale net + cover[2]	39.1 % **10.7** %

Table 3. Performance of our system on the Barcelona dataset

6. Discussion

We introduced a discriminative framework for learning to identify and delineate objects in a scene. Our model does not rely on engineered features, and uses a multi-scale convolutional network operating on raw pixels to learn appropriate low-level and mid-level features. The convolutional network is trained in supervised mode to directly produce labels. Unlike many other scene parsing systems that rely on expensive graphical models to ensure consistent labelings, our system relies on a multiscale feature representation to consider large a large context for each local decision. It also relies on on a segmentation tree in which the nodes (corresponding to image segments) are labeled with the entropy of the distribution of classes contained in the corresponding segment. Instead of graph cuts or other inference methods, we use the new concept of *optimal cover* to extract the most consistent segmentation from the tree.

The complexity of each operation is linear in the number of pixels, except for the production of the tree, which is quasi-linear (meaning cheap in practice). The system produces state-of-the-art accuracy on the SIFT Flow and Barcelona datasets (both measured per pixel, or averaged per class) and state-of-the-art averaged per class accuracy on the Stanford Background dataset, while dramatically outperforming competing models in inference time.

Our current system relies on a single segmentation tree constructed from image gradients, and implicitly assumes that the correct segmentation is contained in the tree. Future work will involve searches over multiple segmentation trees, or will use other graphs than simple trees to encode the possible segmentations (since our optimal cover algorithm can work from other graphs than trees). Other directions for improvements include the use of structured learning criteria such as Maximin Learning (Turaga et al., 2009) to learn low-level feature vectors from which better segmentation trees can be produced.

References

Arbeláez, P., Maire, M., Fowlkes, C., and Malik, J. Contour Detection and Hierarchical Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, 2011.

Boykov, Y. and Jolly, M. P. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proceedings of International Conference of Computer Vision (ICCV)*, volume 1, pp. 105–112, 2001.

Cousty, J. and Najman, L. Incremental algorithm for hierarchical minimum spanning forests and saliency of watershed cuts. In *10th International Symposium on Mathematical Morphology (ISMM'11)*, LNCS, 2011.

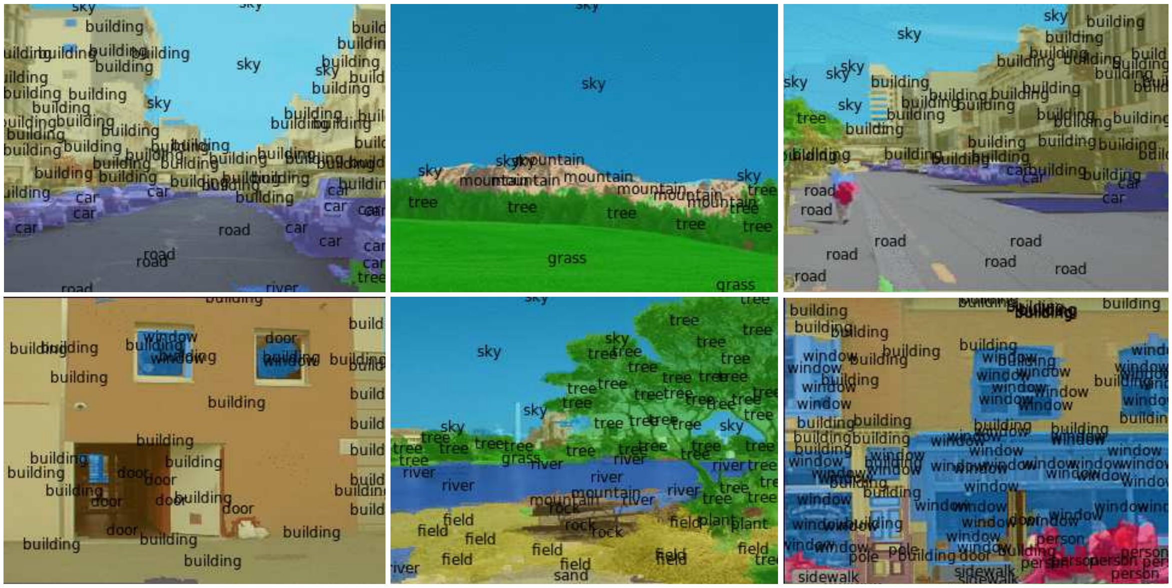

Figure 4. Typical results achieved on the SIFT Flow dataset.

Ford, L. R. and Fulkerson, D. R. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. Technical report, RAND Corp., Santa Monica, 1955.

Gould, S., Fulton, R., and Koller, D. Decomposing a scene into geometric and semantically consistent regions. *IEEE International Conference on Computer Vision*, pp. 1–8, Sept. 2009.

Grangier, D., Bottou, L., and Collobert, R. Deep Convolutional Networks for Scene Parsing. In *ICML 2009 Deep Learning Workshop*, 2009.

He, X. and Zemel, R.S. Learning hybrid models for image annotation with partially labeled data. *Advances in Neural Information Processing Systems*, 2008.

Jain, V., Murray, J. F., Roth, F., Turaga, S., Zhigulin, V., Briggman, K., Helmstaedter, M., Denk, W., and Seung, S. H. Supervised learning of image restoration with convolutional networks. In *ICCV*, 2007.

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

Kumar, M.P. and Koller, D. Efficiently selecting regions for scene understanding. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 3217–3224. IEEE, 2010.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. Efficient backprop. In Orr, G. and K., Muller (eds.), *Neural Networks: Tricks of the trade*. Springer, 1998b.

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

Liu, Ce, Yuen, Jenny, and Torralba, Antonio. Nonparametric scene parsing: Label transfer via dense scene alignment. *Artificial Intelligence*, 2009.

Meyer, F. and Najman, L. Segmentation, minimum spanning tree and hierarchies. In Najman, L. and Talbot, H. (eds.), *Mathematical Morphology: from theory to application*, chapter 9, pp. 229–261. ISTE-Wiley, London, 2010.

Munoz, D, Bagnell, J, and Hebert, M. Stacked hierarchical labeling. *ECCV 2010*, Jan 2010.

Najman, L. and Schmitt, M. Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(12):1163–1173, December 1996.

Ning, F., Delhomme, D., LeCun, Y., Piano, F., Bottou, L., and Barbano, P. Toward automatic phenotyping of developing embryos from videos. *IEEE Trans. on Image Processing*, 2005. Special issue on Molecular & Cellular Bioimaging.

Osadchy, M., LeCun, Y., and Miller, M. Synergistic face detection and pose estimation with energy-based models. *Journal of Machine Learning Research*, 8:1197–1215, 2007.

Russell, C., Torr, P. H. S., and Kohli, P. Associative hierarchical CRFs for object class image segmentation. In *Proc. ICCV*, 2009.

Socher, R., Lin, C. C., Ng, A. Y., and Manning, C. D. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

Tighe, J. and Lazebnik, S. Superparsing: scalable nonparametric image parsing with superpixels. *ECCV*, pp. 352–365, 2010.

Turaga, SC, Briggman, KL, Helmstaedter, M, Denk, W, and Seung, HS. Maximin affinity learning of image segmentation. *NIPS*, Jan 2009.