Hierarchically Gated Deep Networks for Semantic Segmentation

Guo-Jun Qi
Department of Computer Science
University of Central Florida
guojun.qi@ucf.edu

Abstract

Semantic segmentation aims to parse the scene structure of images by annotating the labels to each pixel so that images can be segmented into different regions. While image structures usually have various scales, it is difficult to use a single scale to model the spatial contexts for all individual pixels. Multi-scale Convolutional Neural Networks (CNNs) and their variants have made striking success for modeling the global scene structure for an image. However, they are limited in labeling fine-grained local structures like pixels and patches, since spatial contexts might be blindly *mixed up without appropriately customizing their scales. To address this challenge, we develop a novel paradigm of multi-scale deep network to model spatial contexts surrounding different pixels at various scales. It builds multiple layers of memory cells, learning feature representations for individual pixels at their customized scales by hierarchically absorbing relevant spatial contexts via memory gates between layers. Such Hierarchically Gated Deep Networks (HGDNs) can customize a suitable scale for each pixel, thereby delivering better performance on labeling scene structures of various scales. We conduct the experiments on two datasets, and show competitive results compared with the other multi-scale deep networks on the semantic segmentation task.*

1. Introduction

The goal of semantic segmentation [2, 17, 1] is to segment images into different regions usually by assigning one of semantic labels to each pixel. It is a crucial step towards understanding image scene structures. The label of an image pixel cannot be determined by its local features extracted from a small sliding windows or neighborhood surrounding it. Rather, pixel labels are usually defined in spatial contexts, whose scales often have large variation in sizes. For example, *sky* and *sea* have a large scale of spatial context, but *vessel* and *pedestrian* are more localized to a relatively small scale of context. Moreover, even the regions of the same category may have various sizes of spatial contexts, making it impossible to fix a scale to model each pixel. This inspires us to develop a model that is capable of *learning* to customize spatial contexts and their scales for individual pixels in an image.

To model the spatial context of a pixel, a typical approach is to model the dependencies between the adjacent local image structures based on 2D Markov Random Fields [4][12][13] and Conditional Random Fields [7][21][9]. These models usually capture the local similarity between adjacent image structures of various scales, ranging from pixels, patches to regions. Then, the scene labeling is performed by maximizing the consistency between the similar neighbors which are considered as being in the same spatial context.

On the other hand, the success of deep learning framework on ImageNet challenge [11] has inspired us to apply hierarchical neural networks to build the spatial context on various scales. Convolutional Neural Networks (CNNs) [14], among all deep learning models, have shown their striking performances on modeling the image structures on different levels with multiple layered convolutional kernels.

The CNN models have been generalized to scene labeling. For example, multi-scale CNNs are proposed in [3], which produce and concatenate the feature maps of all scales. Usually, the learned features have to be post-processed by up-sampling coarser-scale maps to match with finer-grained image pixels, and the global contextual coherence and spatial consistency are imposed by CRFs and segmentation trees. Long *et al.* [16] propose an alternative paradigm of fully convolutional networks to annotate images pixel-wise. They present an approach to de-convolve coarser scale of output maps to label the image pixels. Unlike these two CNN-based models, we are interested in a hierarchical network, which not only explores multi-scale structures of input images, but also avoids producing coarse results that have to be up-sampled for labeling pixels. Such a model is highlighted with customized scale of the spatial context for each individual pixels so that the local structures can be modeled on a suitable scale.

Recently, Long Short Term Memory (LSTM) recurrent neural networks [8] has been applied for scene labeling [1]. The LSTM networks are originally used to model sequential data, such as sentences [19] and videos [22]. The networks are composed of a series of recurrently connected memory cells, which are able to capture long-range dependencies between different time frames. All the information entering and leaving memory cells are controlled by several types of gates, which ensures only the information relevant to the task would be maintained in their memory spaces.

Recently, Byeon *et al.* [1] adapt the conventional LSTM networks to model 2D images along four directions – left-top, left-bottom, right-top and right-bottom, and a hidden feedforward layer is built upon 2D LSTM layer to combine the LSTM cells at each image location. Several 2D LSTM layers and feedforward layers are interleaved to capture the spatial contexts along different directions. However, unlike the CNNs, each memory cell is not hierarchically connected with a receptive field of pixels, making it incapable of modeling various scales of spatial contexts like the CNNs.

In this paper, we present a novel paradigm of Hierarchically Gated Deep Networks (HGDNs) to address the challenge of customizing spatial contexts for different pixels at suitable scales. The proposed network combines the advantage of both the CNNs and the LSTMs. Like CNNs, the networks have multiple layers of memory cells that model a growing scale of image structures in bottom up fashion. However, unlike CNNs, between two layers, there are memory gates which control whether spatial contexts from the lower layer should be annexed to form a larger scale of spatial context at the higher layer. In this way, at each layer, a memory cell takes the customized spatial context at its location. Going up the HGDN layers, the spatial context of a pixel could gradually grow up to an arbitrary shape by merging the relevant neighbors until it reaches a suitable scale.

The HGDN model has an intrinsic hierarchical structure, making it adequate in modeling multiple scales of spatial contexts. Unlike the 2D LSTMs, the HGDN model does not have any horizontal connections between memory cells in one layer, and all the gates are deployed to control the flows of spatial patterns moving vertically across layers. In this way, the HGDN can accurately label pixels by avoiding the risk of blindly mixing up different scales of spatial patterns. The HGDN model inherits the ability of the CNNs modeling various scales of image details, as well as the advantage of the LSTMs on capturing the long-range spatial dependency between relevant pixels.

The remainder of this paper is organized as follows. In the following section, we review the existing work related to the proposed method. In Section 3, we formulate the problem of scene labeling that motivates our model. We present the proposed network architecture in Section 4, followed by a discussion on the implementation details in Section 6. We present the experiment results in Section 7, and conclude the paper in Section 8.

2. Related Work

Most of existing scene labeling approaches fall into one of three major categories.

The first category uses probabilistic graphical models to reveal the dependencies between adjacent pixels, patches or regions. Tighe and Lazebnik [20] present a nonparametric approach to label superpixels by incorporating neighborhood context with efficient MRF optimization. On the contrary, Russell *et al.* [18] point out that there is no common optimal level like pixels and segments for labeling images, which is suitable for all scene categories. Thus they propose to use a hierarchical CRF model to integrate the features computed at different levels.

To explore the multiple scales of image spaces, Convolutional Neural Networks and their variants become successful and scalable in many computer vision applications. Among them, as aforementioned in the last section, multi-scale CNNs [2] and fully convolutional networks [16] have been proposed for scene labeling tasks. Both produce coarse-scaled output layer which has to be up-sampled to a higher resolution for labeling individual pixels. Kekeç *et al.* [10] propose to use two separate CNNs to combine the visual features and the contextual information for scene labeling.

Recently, LSTM recurrent neural networks have been applied to model the spatial contexts for labeling scenes [1]. This work is based on multi-dimensional LSTMs [5, 6], which model N-dimensional data along different directions. Specifically, 2D LSTMs are unfolded horizontally in four different directions (left-top, left-bottom, right-top and right-bottom) to model images. However, unlike the CNN models, multi-dimensional LSTMs do not model various scales of input images in a bottom-up fashion. This makes them inadequate in capturing flexible ranges of spatial contexts. Moreover, the 2D LSTMs work on pixel patches rather than the individual pixels. Thus, the labeling result on patches has to be interpolated to annotate individual pixels.

3. Problem and Motivation

In this section, we discuss the proposed Hierarchically Gated Deep Network (HGDN) and its application on scene labelling. Suppose we have an image $\mathbf{X} = \{\mathbf{x}_{i,j} | i = 1, \cdots, M, j = 1, \cdots, N\}$ of size $M \times N$, where $\mathbf{x}_{i,j}$ represents the local features extracted for the pixel located at (i, j) of the image. Then the goal of scene labeling is to assign each pixel with a label $y_{i,j}$ from a finite set

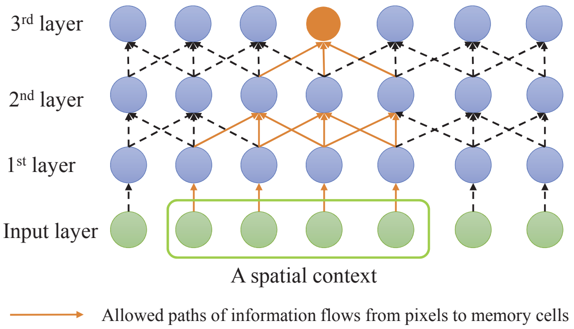

3rd layer
2nd layer
1st layer
Input layer
A spatial context
Allowed paths of information flows from pixels to memory cells

Figure 1: An one-dimensional example of the proposed Hierarchically Gated Deep Networks (HGDNs). The solid orange line represents the allowed information flows being gated between multiple successive layers. They eventually enter the highlighted node at the topmost layer.

$\mathcal{C} = \{1, 2, \cdots, C\}$.

The difficulty of labeling pixels lies in that we do not know a suitable scale of a pixel's spatial context containing a set of relevant pixels that form a local structure, such as object parts and scenery areas. Many existing methods attempt to explore the smoothness assumption over the labels assigned to the pixels in a local neighborhood. However, this assumption might not be able to explore the large variations in the scales of the spatial contexts. For example, some pixels have large scales of context like *grass* and *sky*; on the contrary, some pixels belong to an object part with a small scale of spatial context such as *wheel* and *window*. While multi-scale deep networks such as CNNs model various scales in a bottom-up fashion, however, they are inadequate in capturing location-variant scales surrounding pixels belonging to different scales of local structures. This inspires us to develop a model which can automatically handle flexible spatial contexts corresponding to different local structures.

To address this challenge, in this paper we propose a novel paradigm of multi-scale deep network, where a suitable scale of spatial context for each pixel is determined by gating varying sizes of neighborhood between two successive layers. It creates multi-layered gates to control the information flows between multiple successive layers. The gate only allows the information flows from the same spatial context to update the hidden memory state corresponding to each pixel at the higher layer. A suitable scale of the spatial context for each pixel gradually grows by annexing more and more relevant pixels bottom up through the network layers.

Figure 1 illustrates this idea with an one-dimensional example. The highlighted node at the topmost layer absorbs information flows from a spatial context of four nodes at the bottom layer. The solid arrows represent the allowed paths of information flows entering the highlighted node. In the next section, we will formalize this idea of gating information flows in the mathematical details.

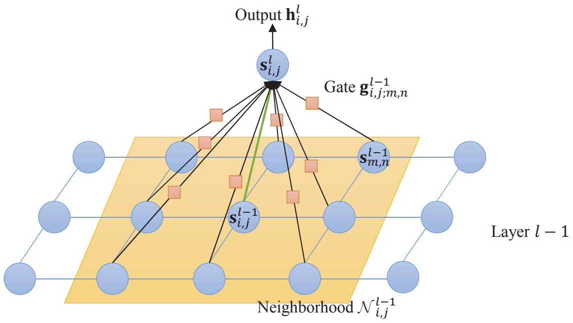

Output $\mathbf{h}_{i,j}^{l}$
$\mathbf{s}_{i,j}^{l}$
Gate $\mathbf{g}_{i,j;m,n}^{l-1}$
$\mathbf{s}_{m,n}^{l-1}$
$\mathbf{s}_{i,j}^{l-1}$
Layer $l-1$
Neighborhood $\mathcal{N}_{i,j}^{l-1}$

Figure 2: An illustration of the network structure between a memory cell at an upper layer and its connected cells in a neighborhood of the lower layer. Each memory cell has an internal state $\mathbf{s}_{i,j}^{l}$ representing the spatial patterns extracted from a context, and it produces an output $\mathbf{h}_{i,j}^{l}$ fed into the higher layer. Between two connected cells, there is a gate $\mathbf{g}_{i,j;m,n}^{l-1}$ (square nodes) controlling whether to allow information flows of lower-layer outputs to enter the memory cells at the higher layer. Only the information flows from the same spatial context should be allowed to pass through the gate.

4. The Proposed Approach

The proposed deep network has multiple layers of neurons which have the same size as the input image. This enables labeling each pixel directly at the output layer without having to up-sample the labels. Each layer is composed of a two-dimensional grid of memory cells indexed by (i, j) for $i = 1, \cdots, M$ and $j = 1, \cdots, N$, each corresponding to a pixel of input image.

As illustrated in Figure 2, between two layers, each memory cell (i, j) at an upper layer l is connected to a neighborhood $\mathcal{N}_{i,j}^{l-1}$ of memory cells around the same location (i, j) at the lower layer $l - 1$. Each memory cell has an internal state $\mathbf{s}_{i,j}^{l}$, which memorizes the patterns obtained from the spatial contexts up to the level l. Meanwhile, the memory cell produces the output $\mathbf{h}_{i,j}^{l}$, which is fed into the upper layer. All outputs are gated so that only the relevant patterns are allowed to update the memory cells at the upper layer.

In the following, we will explain different components of the proposed network architecture in detail.

Input Gates

First, we define an input gate to control the information flow between a memory cell (i, j) and its connected cells $(m, n) \in \mathcal{N}_{i,j}^{l-1}$ at the lower layer. The role of this gate is to filter out those irrelevant patterns to the pixel labels.

Specifically, for any connected cell $(m,n) \in \mathcal{N}^{l-1}_{i,j} \setminus \{(i,j)\}$ in the neighborhood of (i,j), we have an input gate defined as

$$\mathbf{g}^{l-1}_{i,j;m,n} = \sigma(\mathbf{U}_\mathrm{g}^{(i-m,j-n)}\mathbf{s}^{l-1}_{i,j} + \mathbf{W}_\mathrm{g}^{(i-m,j-n)}\mathbf{h}^{l-1}_{m,n} + \mathbf{b}_\mathrm{g}) \tag{1}$$

where $\sigma(\cdot)$ is the sigmoid activation over the range of $[0,1]$; $\mathbf{U}_\mathrm{g}^{(i-m,j-n)}$ and $\mathbf{W}_\mathrm{g}^{(i-m,j-n)}$ are transformation matrices from the state and input to the gate respectively, and $\mathbf{b}_\mathrm{g}$ is the bias vector [1]. These parameters are functions of the relative disposition $(i-m, j-n)$ between (i,j) at the center and (m,n) in the neighborhood $\mathcal{N}^{l-1}_{i,j} \setminus \{(i,j)\}$. So these parameters are recurrent across different neighborhoods at the same layer, which can save a huge amount of parameters.

The gate defines a spatial context in the neighborhood of (i,j) – its value decides to what extent the output from a memory cell (m,n) of a lower layer is contextually connected to a memory cell at (i,j) of the upper layer. It is a multiplicative gate, where a large value of the gate implies the output $\mathbf{h}^{l-1}_{m,n}$ from (m,n) can pass through it to update the memory state $\mathbf{s}^{l}_{i,j}$. In this sense, the gate can be viewed as an indicator of whether (m,n) belongs to the spatial context of (i,j).

It is worth noting that $\mathbf{g}^{l-1}_{i,j;m,n}$ is not a symmetric function of two locations (i,j) and (m,n). In other words, the assertion that (m,n) belongs to the spatial context of (i,j) does not necessarily imply (i,j) also belongs to the spatial context of (m,n). For example, for labeling a *ship*, the spatial context might include surrounding pixels of *sea* as they provide useful clues to label *ship*. However, on the converse, it is not necessary to include *ship* as the spatial context when labeling *sea*. This gives more flexibility to model the spatial contexts for different pixels.

Memory Cell States

With the input gate defined above, the state $\mathbf{s}^{l}_{i,j}$ of a memory cell can be updated by combining all the outputs from the connected memory cells at the lower layer through the relevant spatial context defined by the gates of $\mathbf{g}^{l-1}_{i,j;m,n}$

$$\mathbf{s}^{l}_{i,j} = \mathbf{s}^{l-1}_{i,j} + \sum\nolimits_{(m,n)\in\mathcal{N}^{l-1}_{i,j}\setminus\{(i,j)\}} \{\mathbf{g}^{l-1}_{i,j;m,n} \odot \mathbf{i}^{l-1}_{i,j;m,n}\} \tag{2}$$

where $\odot$ is the element-wise multiplication, and $\mathbf{i}^{l-1}_{i,j;m,n}$ is the input modulating $\mathbf{h}^{l-1}_{m,n}$ entering the memory cell (i,j):

$$\mathbf{i}^{l-1}_{i,j;m,n} = \tanh(\mathbf{W}_\mathrm{s}^{(i-m,j-n)}\mathbf{h}^{l-1}_{m,n} + \mathbf{b}_\mathrm{s}) \tag{3}$$

where the transformation matrix and bias vector $\mathbf{W}_\mathrm{s}^{(i-m,j-n)}$ and $\mathbf{b}_\mathrm{s}$ are functions of the relative location of $(i-m, j-n)$, and the hyperbolic tangent $\tanh(\cdot)$ is the activation function for this modulated input.

Note that, the first term of the RHS of Eq. (2) suggests that the memory state $\mathbf{s}^{l-1}_{i,j}$ from the lower layer does not need to be gated before entering the memory cell at the same location. This is based on the assumption that each memory cell must belong to its own spatial context at this same location. Thus, it allows the memory state to go straight up to label the pixel at that location.

Memory Cell Outputs

Once the state of a memory cell is updated, it is ready to produce the following output fed into the higher layer.

$$\mathbf{h}^{l}_{i,j} = \tanh(\mathbf{W}_\mathrm{h}\mathbf{s}^{l}_{i,j} + \mathbf{b}_\mathrm{h})$$

where $\mathbf{W}_\mathrm{h}$ and $\mathbf{b}_\mathrm{h}$ are the parameters for the memory cell output.

Topmost Output Layer

At the top of the network, we have an output layer where a softmax function is applied at each location (i,j) to classify the corresponding pixel to one of C labels:

$$\Pr(y_{ij} = c|\mathbf{h}^{L}_{i,j}) = \frac{\exp((\mathbf{w}_c * \mathbf{h}^L)_{i,j} + b_c)}{\exp(\sum_{c=1}^{C}(\mathbf{w}_c * \mathbf{h}^L)_{i,j} + b_c)}$$

where $*$ denotes the convolution between a kernel $\mathbf{w}_c$ of label c and the hidden output $\mathbf{h}^L$ from the last memory cell layer L, and b_c is bias for each label c.

5. Training the Deep Network

The training of the proposed deep network can be performed by back-propagating the errors up down through the whole network.

In particular, given a training image $\mathbf{X} = \{\mathbf{x}_{i,j}\}$ with the pixel labels $\mathbf{Y} = \{y_{i,j}\}$, the loss function can be defined as the negative log likelihood of softmax outputs

$$\ell(\mathbf{\Theta}) = \sum_{i,j=1}^{M,N} -\log \Pr(y_{i,j}|\mathbf{h}^{L}_{i,j})$$

where $\mathbf{\Theta}$ contains all model parameters.

It is worth noting that this loss function is separable between the pixel labels at different locations. Thus we can parallelize the calculation of the loss derivatives across different locations for an input image. This can speed up the back-propagation procedure on GPU with many cores.

Also, when we back-propagate the errors down the network, we will truncate those errors leaving a memory cell down to the lower layer, except the errors back-propagated along the states of the memory cells at the same location. This truncated back-propagation algorithm has worked well on training Long Short-Term Memory (LSTM) machine [8] to prevent the vanishing or exploding errors.

[1] These parameters should differ between different layers. However, for notational simplicity, we do not explicitly distinguish them for different layers.

It is also worth noting that all the model parameters $\mathbf{W}_*$ and $\mathbf{b}_*$ are recurrent over each layer. Like the convolutional kernels used in Convolutional Neural Networks (CNNs), this recurrent structure significantly reduces the size of the parameters that have to be estimated. This makes it possible to have the whole network fit in the GPU's device memory, without frequent swap of the network parameters between the host and device memory spaces. On the Nvidea Tesla K40 GPUs we used for the experiment, the model can fit into the device memory. This further speeds up the training of the proposed HGDN on the GPUs.

6. Further Discussions

6.1. GPU Implementation

Most computations of the model arise from the calculation of multiple layered memory cells and their gates. They can be accelerated by the GPUs. For example, each term in the argument of the memory gate function (1) can be parallelized on GPGPU. The first term can be vectorized by copying $\mathbf{s}^l_{i,j}$ to tile over the neighborhood and applying the matrix multiplication with $\mathbf{U}_g$ simultaneously; the second term can also be parallelized over the whole neighborhood, which can be conducted on GPUs very fast.

Then, in calculating the memory state in Eq. (2), the summation between the gate function $\mathbf{g}^{l-1}_{i,j;m,n}$ and the modulated input $\mathbf{i}^{l-1}_{i,j;m,n}$ can be considered as a variant of convolution [2] between these two terms. These multiplication and summation operations can be parallelized together with the computations of gate function (1) over each neighborhood in the similar way as the standard convolution. In our implementation on Nvidia Tesla K40 GPUs, an image of size 256×256 can be processed within 0.03 seconds on average, including both feed-forward pixel labeling for prediction and backward error propagation for model training.

6.2. How many layers should be added?

One of interesting questions is how many layers we should build for labeling an input image through the HGDN. Clearly, the answer depends on the size of the image, as well as the size of neighborhood.

Each memory cell at a particular layer has a maximal size of spatial context over the input image, if we assume all the gates are completely open. For example, in Figure 1, the highlighted memory cell at the top layer can cover a maximal number of five nodes in its spatial context at the input layer. But the actual scale of the spatial context for a memory cell can be much smaller than its possible maximal scale when not all gates on its path to input layers are open, as shown in Figure 1 where the highlighted cell only covers four rather than five nodes of input layer in its spatial context. As more layers are added, assuming all gates are open, the maximal scale of a spatial context will gradually grow until it covers the whole input image. How fast the scale of spatial context will grow will depend on the size of neighborhood – the larger the neighborhood, the more cells from the lower layer may be involved, and thus a spatial context can grow faster. Then the maximum number of layers required for labeling pixels can be defined as the one when the whole input image is covered by the memory cells at the output layer.

However, in practice, we only need spatial contexts to cover a relatively small part of an input image rather than the entire image, and thus the actual number of layers is often much smaller than the theoretical maximum number. For example, on the datasets we used in the experiments, we find with neighborhood sizes ranging from 11×11 to 5×5 pixels, five hidden layers (excluding input and output layers) should be enough to give a satisfactory performance on labeling pixels.

7. Experiments

In this section, we present our experiment results on scene labeling tasks.

7.1. Datasets

We test the proposed HGDN on two datasets – Stanford Background dataset [4] and the SIFT Flow dataset [15]. Both datasets have been fully labeled on individual pixel level.

The Stanford background dataset contains 715 images, which are annotated with 8 scene labels. The dataset is split into a training set of 572 images, and a test set of 143 images. It has a special foreground label, which denotes unknown objects. Each individual image has a different resolution, with an average size of 320×240 pixels.

On the other hand, SIFT Flow dataset has $2,488$ images, each having a fixed resolution of 256×256 pixels. The dataset is split into a training set of $2,488$ images and a test of the rest 200 images. It contains 33 labels annotated by LabelMe users.

7.2. Network Architecture

The HGDN model used in the experiment has seven layers, including one input layer, five hidden layers and one output layer. As shown in Table 1, five hidden layers are created to hierarchically gate the information flows originated from the input images. At the first hidden layer, convolutional filters of size 11×11 are applied to input image to initialize the states of memory cells at all locations. In other words, each memory cell at the first hidden layer is

[2] Here the convolutional kernel $\mathbf{g}$ is location-variant unlike the standard convolutional kernel that is location invariant. However, the location-variant convolution makes it possible to customize unique scale of spatial context for each pixel at different locations.

Table 1: Proposed HGDN architecture. Except the number of memory states, the same HGDN architecture is applied to both datasets. This table shows the neighborhood size of memory cells for each hidden layer. Among them, the first hidden layer initializes the memory cell state at each location by applying a convolution kernel of size 11×11 to input images. From the second hidden layer through the fifth hidden layer, varying sizes of neighborhoods are applied to connect the memory cells between two layers. Finally, the topmost output layer generates the pixel labels by applying a convolution computation to the last memory cell layer (i.e., the first layer) with a kernel of size 5×5.

(a) Hidden Layers

Layer	Neighborhood Size	No. of Mem. States	
		Stanford	SIFT Flow
1	11×11	32	256
2	9×9	32	256
3	5×5	32	256
4	5×5	32	256
5	5×5	32	256

(b) Output Layer

Layer	Conv. Kernel Size	No. of output labels	
		Stanford	SIFT Flow
Output	5×5	8	33

connected to a neighborhood (receptive field) of 11×11 pixels of the input image.

From the second hidden layer, each memory cell receives the gated inputs from a varying-sized neighborhood at the lower layer. The memory cells at the second hidden layer are connected to a neighborhood of 9×9 memory cells of the lower layer; from the third hidden layer, each memory cell is connected to a neighborhood of 5×5 memory cells from the lower layer. Finally, a convolutional kernel of size 5×5 is applied to the last memory cell layer, which generates the pixel labels at the topmost output layer.

All hidden layers have the same number of states for each memory cell, which ensures the representation of spatial contexts reside in the same state space across layers. A rule of thumb to choose the number of memory states is to use a multiple of 16 closest to four times of scene labels – on Stanford background dataset, we use 32 states for each memory cell, while 128 states are used for SIFT Flow dataset. Choosing the multiple of 16 for the number of memory states increases the computing performance on GPUs thanks to the data alignment. We find this rule works well for both datasets. Also, each layer is padded with the zeros to ensure all the layer to have a full size of resolution after being processed. All the layers of memory cells, from the input to the output layer, have the same size and thus we do not need to up-sample the output layer to label every pixel.

We adopt stochastic gradient method to train the model. All the model parameters are initialized with a zero-mean Gaussian distribution with a standard deviation of 0.1. The learning rate is fixed to 0.001, with a momentum of 0.9. With the GPU implementation, each epoch can be finished within less than 1.5 seconds with a batch of 128 training images.

7.3. Evaluation Metrics

The evaluation of scene labeling task is usually performed based on two metrics. The first is the pixel-wise accuracy, which measures the ratio of pixel-wise true positives over all the pixels. The second is the average label accuracy, which computes the average of label accuracies over all labels. The latter metric is more challenging because on both datasets, different labels are imbalanced, where some labels (e.g., *sky*) cover a much larger number of pixels than the other labels (e.g., *pedestrian*). To make a fair comparison with the other algorithms, we use all the pixel labels in their natural frequencies and do not balance them to train the model. We report both metrics when comparing with different algorithms.

7.4. Results

Table 2a and Table 2b report the pixel-wise accuracy and average label accuracy on Stanford Background dataset and SIFT Flow dataset respectively. We also compare the average computing time to process each image by different algorithms. Our implementation is very fast in processing images as we parallelize the computation across neighborhoods in each layer.

We compare with both CNN and LSTM paradigms of state-of-the-art methods. The results show that the proposed HGDN model achieves very competitive accuracy on both datasets, outperforming both CNN and LSTM paradigms when no extra training data are involved in pretraining the model. Note that the 2D LSTM model in [1] divided images into patches as inputs and has reported the performances with varying sizes of input patches. In the table, we report the best result achieved by the optimal size of input patch on two datasets.

Figure 3 illustrates the output maps of scene labels on some image examples. The brightness of output maps represents the activations of a pixel in respondence to different scene categories. It reflects the probability of each pixel belonging to those scene categories. The results are obtained from the last HGDN output layer. We can see that different scene categories occupy various scales of spatial contexts – for example, the *sky* and *road* usually span a larger scale of context, while the *tree* is often restricted to relatively s-

Table 2: Comparison of pixel-wise accuracy and label average accuracy on (a) Stanford Background and (b) SIFT FLOW datasets. We also report the average computing time to process each image by different algorithms. The proposed HGDN is implemented on Tesla K40 GPU, which reaches very fast computing speed. Not all compared models are implemented on the GPUs, and we denote them in the parenthesis after the reported computing time. Some results by the compared algorithms are missing in the literature, where we denote with N/A. We do not balance the label frequencies to improve the label average accuracy. † It is noted that the FCN uses extra ILSVRC data to pretrain a reference model before it is fine-tuned on the SIFT Flow dataset, whereas no extra data are involved to pretrain the HGDN.

(a) Stanford Background

Algorithm	Pixel-wise Accu.	Label Avg Accu.	Comp. Time (sec.)
Superparsing 2010 [20]	77.5	N/A	10 to 300
Singlescale ConvNet 2013 [2]	66	56.5	0.35 (GPU)
Multiscale net [2]	78.8	72.4	0.6 (GPU)
Multiscale net + superpixels [2]	80.4	74.56	0.7 (CPU)
Multiscale net + gPb + cover [2]	80.4	75.24	61 (CPU)
Multiscale net + CRF on gPb [2]	81.4	76.0	60.5
Augmented CNNs 2014 [10]	71.97	66.16	N/A
Recurrent CNNs [17]	76.2	67.2	1.1 (GPU)
2D LSTM networks [1]	78.56	68.26	1.3 (CPU)
Proposed HGDN	**82.41**	**72.98**	0.02 (GPU)

(b) SIFT Flow dataset

Algorithm	Pixel-wise Accu.	Label Avg Accu.	Comp. Time (sec.)
Multi-scale net (balanced frequency) [2]	72.3	50.8	N/A
Multi-scale net (natural frequency) [2]	78.5	29.6	N/A
Augmented CNNs 2014 [10]	49.39	44.54	N/A
Recurrent CNNs [17]	65.5	20.8	N/A
FCN† [16]	85.2	51.7	0.175 (GPUs)
2D LSTM networks [1]	70.11	22.59	1.2-3.1 (CPU)
Proposed HGDN	**79.68**	**51.26**	0.03 (GPU)

maller scale. Even for the same scene category, its scale of spatial context can vary a lot between different images. This confirms the necessity to customize the spatial context to various scales.

We also illustrate the feature maps of 32 memory states at 2-5 hidden layers in Figure 4. It shows that how image structures of various scales are captured through these layers – at the lower layers, small structures are captured, while at the higher layers, complex structures are modeled by gradually annexing the small structures from the lower layers. It is also worth noting that the activations of these 32 memory states are much sparse – some of memory states produce zero or saturated activations. This makes sense since not every memory state need to respond to all scene labels. Actually, the sparseness decouples the activations between different memory states, which can reduce the over-fitting risk.

Also, to verify the importance of gating the scales of spatial contexts across layers, we test a compared HGDN model by removing the multi-layered gates between layers. We found that the pixel-wise accuracy on the two datasets would drop to 63.84% and 56.75% respectively, and the average label accuracy would decline to 65.92% and 43.22%. This shows that without these gates, different scales of the spatial contexts might be mixed up, leading to bad performance on labeling pixels. This result justifies the necessity of adding gates to control the spatial scales.

Finally, it is worth noting that our pixel labeling result is the direct output from the HGDNs, without any postprocessing procedure like label smoothing or interpolation. Although we have found that postprocessing the labeling result via CRFs, over-segmented patches, and segmentation trees can further improve the performance, we do not report the post-processing results to ensure fair comparison with the other algorithms.

8. Conclusion

In this paper, we develop a novel paradigm of deep network which explores various scales of spatial contexts adjusted to pixels at different locations. Compared with the

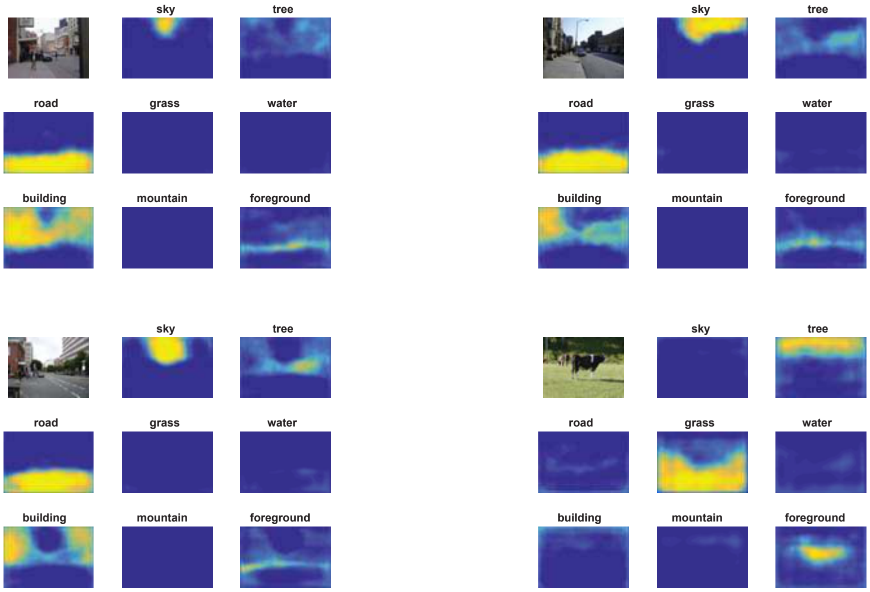
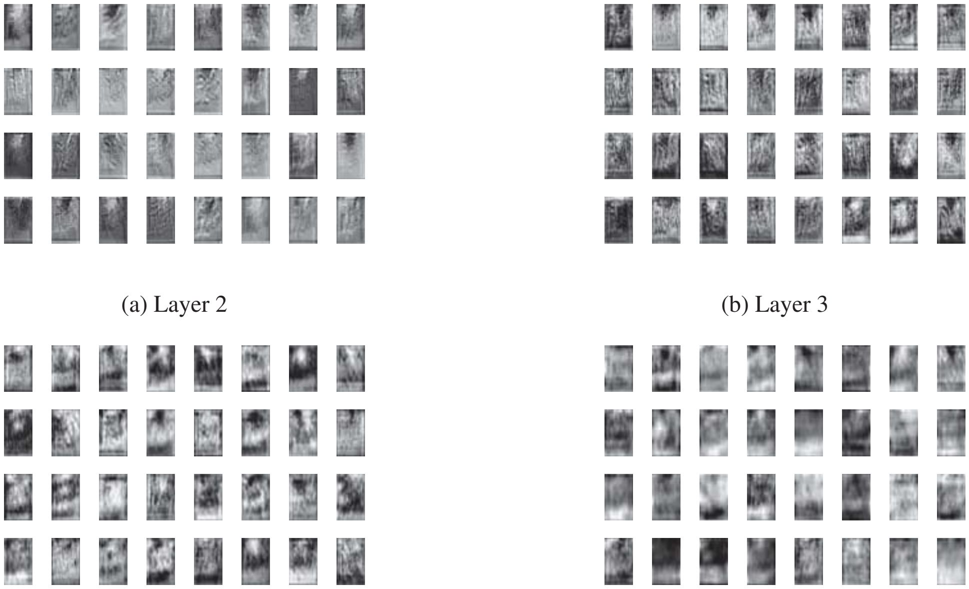


Figure 3: Illustration of output maps of scene labels on Stanford Background dataset, where the brightness represents the activations of a pixel in respondence to different categories. The results are obtained from the last HGDN output layer.



(a) Layer 2

(b) Layer 3

(c) Layer 4

(d) Layer 5

Figure 4: Illustration of the learned feature maps from the top four layers of memory cells (Layer 2-5). The input image corresponds to the first one in Figure 3.

other multi-scale deep network, the proposed model constructs multiple layers of memory cells, whose outputs are hierarchically gated on different scales before recursively feeding to higher layers. Then the pixel labels at different locations are decided based on the spatial contexts of the customized scales. The experiment results show that its architecture is more adequate in modeling the scene structures on different scales than the other compared deep networks.

References

[1] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki. Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3547–3555, 2015. 1, 2, 6, 7

[2] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013. 1, 2, 7

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014. 1

[4] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1–8. IEEE, 2009. 1, 5

[5] A. Graves, S. Fernndez, and J. Schmidhuber. Multidimensional recurrent neural networks. In *Artificial Neural Netwroks ICANN 2007*, pages 519–558, 2007. 2

[6] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 545–552, 2009. 2

[7] X. He, R. S. Zemel, and M. Á. Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *Computer vision and pattern recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE computer society conference on*, volume 2, pages II–695. IEEE, 2004. 1

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2, 4

[9] Q. Huang, M. Han, B. Wu, and S. Ioffe. A hierarchical conditional random field model for labeling and segmenting images of street scenes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1953–1960. IEEE, 2011. 1

[10] T. Kekeç, R. Emonet, E. Fromont, A. Trémeau, and C. Wolf. Contextually constrained deep networks for scene labeling. 2, 7

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[12] M. P. Kumar and D. Koller. Efficiently selecting regions for scene understanding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3217–3224. IEEE, 2010. 1

[13] D. Larlus and F. Jurie. Combining appearance models and markov random fields for category level object segmentation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE, 2008. 1

[14] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995. 1

[15] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):978–994, 2011. 5

[16] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014. 1, 2, 7

[17] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *Proceedings of The 31st International Conference on Machine Learning*, pages 82–90, 2014. 1, 7

[18] C. Russell, P. Kohli, P. H. Torr, et al. Associative hierarchical crfs for object class image segmentation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 739–746. IEEE, 2009. 2

[19] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. 2

[20] J. Tighe and S. Lazebnik. Superparsing: scalable nonparametric image parsing with superpixels. In *Computer Vision–ECCV 2010*, pages 352–365. Springer, 2010. 2, 7

[21] B. Triggs and J. J. Verbeek. Scene segmentation with crfs learned from partially labeled images. In *Advances in neural information processing systems*, pages 1553–1560, 2007. 1

[22] V. Veeriah, N. Zhuang, and G.-J. Qi. Differential recurrent neural networks for action recognition. *Proceedings of International Conference on Computer Vision*, 2015. 2