Object Class Segmentation of RGB-D Video using Recurrent Convolutional Neural Networks

Mircea Serban Pavel, Hannes Schulz, and Sven Behnke

Universität Bonn
Computer Science Institute VI
Friedrich-Ebert-Allee 144
53113 Bonn

pavel@cs.uni-bonn.de, schulzh@ais.uni-bonn.de, behnke@cs.uni-bonn.de

Abstract

Object class segmentation is a computer vision task which requires labeling each pixel of an image with the class of the object it belongs to. Deep convolutional neural networks (DNN) are able to learn and take advantage of local spatial correlations required for this task. They are, however, restricted by their small, fixed-sized filters, which limits their ability to learn long-range dependencies. Recurrent Neural Networks (RNN), on the other hand, do not suffer from this restriction. Their iterative interpretation allows them to model long-range dependencies by propagating activity. This property is be especially useful when labeling video sequences, where both spatial and temporal long-range dependencies occur. In this work, a novel RNN architecture for object class segmentation is presented. We investigate several ways to train such a network. We evaluate our models on the challenging NYU Depth v2 dataset for object class segmentation and obtain competitive results.

Keywords: recurrent neural networks, computer vision, object class-segmentation

1. Introduction

Current deep neural network architectures achieve superior performance on a number of computer vision tasks, such as image classification, object detection, and object class segmentation. Most of these tasks focus on extracting information from a single image. Deep neural networks compute increasingly abstract features, which simultaneously become more and more semantically meaningful, and incorporate larger contexts.

A real-world vision system will have to deal with the time dimension as well. Content is increasingly generated in the form of videos by Internet users, surveillance cameras, cars, or mobile robots. Video information can be helpful, as looking at a whole sequence instead of single frames may enable the interpretation of ambiguous measurements.
Figure 1: Architecture of our RNN. The layers are connected to each other with valid convolutions. Upward (forward) connections additionally include spatial max-pooling operations, while downward (backward) connections include a spatial upsampling. Delay $T$ is number of time frames between an input and corresponding output frame.

Similar to increasingly abstract features on images, we are interested in neural networks which produce high-level features on sequences. In a recursive computation, these high-level features should help to interpret the next frame in a sequence. In addition to a semantically meaningful localized content description, such features should form high-level descriptions of motions with increasing temporal context.

In this paper, we introduce a recurrent convolutional neural network architecture which produces high-level localized sequence features. We evaluate it on the NYU Depth v2 (NYUD) dataset, an RGB-D object class segmentation task, where every pixel of the input image must be labeled with the category of the object it belongs to. In this challenging and established benchmark, most methods focus on prediction based on single frames, while our method profits from image sequences.

In short, our contributions are as follows:

- We introduce a recurrent convolutional neural network model for processing image sequences.
- On toy datasets, we show that our recurrent models are able to keep an abstract state over time, track and interpret motion, and retain uncertainty.
- We show that our model improves RGB-D object class segmentation accuracy on the challenging NYUD dataset when compared to other CNN models. When combined with a CRF, RNN performance is close to carefully tuned transfer-learning approaches initialized on much larger datasets.
- We analyze the obtained networks and show that the improved performance of our network stems from the recurrent processing combined with
Figure 2: Recurrent connections as viewed from a single hidden layer. Activations of a hidden layer are the result of forward connections from below, backward connections from above, and lateral connections from the same layer at previous time steps. Inputs may be provided at all scales.

the exploitation of temporal, i.e. video, information.

The remainder of this paper is organized as follows. Section 2 introduces our architecture and learning methods. In Section 3, we discuss related work. We evaluate our model in Section 4, and discuss the results in Section 5.

2. Recurrent Convolutional Neural Networks for Image Labeling

In this section, we present our network architecture. It is inspired by the Neural Abstraction Pyramid of Behnke (2003), a hierarchical recurrent convolutional neural network architecture for image processing which extends feedforward convolutional neural networks (CNN (LeCun et al., 1998)). As CNNs, it retains the topological image structure and ensures that features are localized. A schematic overview is shown in Figures 1 and 2. Our base configuration (without the ability to process sequences) is a fairly standard small CNN with $L = 3$ convolutional layers, ReLU non-linearities, and interleaved spatial max-pooling.

In contrast to the Neural Abstraction Pyramid of Behnke (2003), which doubles the number of filters as the resolution of the representation is halved, we keep the number of filters to a constant 32 on all layer of the network. Initial experiments on our dataset have shown that doubling the number of filters leads to a too complex model, which takes longer to learn and is prone to overfitting.

2.1. Connection Types

To process sequences, we replicate our model for $T$ time steps and introduce connections between the temporal copies. Three types of connections
exist: forward, lateral, and backward. Computationally, all connections are valid convolution operations followed by a half-rectifying point-wise non-linearity \( f(x) = \max(0, x) \) (ReLU).

A hidden layer \( H(t, l) \), at time step \( t \) and abstraction level \( l \), is connected to layer \( H(t + 1, l + 1) \) using a forward connection. These connections allow the vertical flow of information from the bottom of the network to the top and thus the construction of high-level features based on the low-level features. The non-linearity of the forward connections are followed by a spatial 2×2 maximum pooling operation with stride 2.

Lateral projections connect layers \( H(t, l) \) and \( H(t + 1, l) \). These horizontal connections can incorporate immediate spatial context from the same activation level. The intermediate context is limited by the receptive field size of the convolution filters.

Backward projections connect layer \( H(t, l) \) to layer \( H(t + 1, l - 1) \), and can be interpreted as providing a high-level prior for the lower, more detailed layers. Since higher layers have a coarser spatial resolution, they also provide a convenient shortcut for information that needs to travel long distances. Backward connections are immediately followed by a factor two naive spatial upsampling operation (i.e. every pixel is replicated four times).

Due to padded convolutions and the opposing pooling and upsampling operations, all connections coinciding on a given hidden layer have the same size, and are simply summed elementwise.

We use a convolutional feature extraction layer between the inputs and the RNN, whose weights are initialized using symmetry-k-means (Konda et al., 2013). Since this unsupervised initialization prevents us from controlling the gain, we ensured that these weights are not part of a feedback loop. For similar reasons, the weights of the first forward pass are excluded from weight sharing as well.

All other hidden-to-hidden connections use temporal weight sharing, i.e. for all \( t \) and all \( k \in \{-1, 0, 1\} \), the weights used in the convolution from \( H(t, l) \) to \( H(t + 1, l + k) \) are identical across time steps.

2.2. Output in the Bottom Layer

In contrast to common CNN models, the output of our network is always obtained in the lowest layer of the network. This structural property allows us to produce detailed outputs at input resolution. We add a time delay between input and output, ensuring that the inputs from the last presented frame were able to reach the topmost layer and return to the bottom layer before evaluating the loss.

A final convolution without temporal weight sharing is used to extract one map per target class from the bottom layer. The cross-entropy loss is computed per pixel over all output maps.

2.3. Recurrent Convolutional Neural Networks

We process the images of videos sequentially, one image per time step. The state (i.e., the activations) at time \( t \) containing information about its past is
combined with the image at time $t$, producing an output and a new state. Since the last output benefits from learning from the whole sequence, it is natural to place the frame that we want to evaluate at the end.

The first temporal copy is special, since it contains regular feed-forward connections. This allows us to produce activations in each layer such that all connection types can be used in the transition from $t$ to $t + 1$.

**Network Depth.** When processing input at time $t$, we allow $L - 1$ time steps for the information to reach the top level of the network and the same amount for propagating back to the bottom layer, where the output corresponding to time $t$ is evaluated. Note that the last temporal steps do not need all the hidden layers, since their activation would no longer propagate to the output.

Our RNN is trained with backpropagation through time (BPTT), and can be interpreted as a very deep non-recurrent net after unfolding in time. In this non-recurrent network, multiple paths lead to the output, with the shortest path — from input $t = T$ to the final output — having only length $2L - 1$, and the longest $2L + t$, which amounts to a depth of 14 layers for our $L = 3$, $T = 8$ network.

**Weight Initialization and Optimization.** We initialize the weights and biases from a Gaussian distribution. It is important to ensure that the activations do not explode or vanish early during training. Ideally, activations in the first forward pass should have similar magnitudes. This is difficult to control, however. Instead, we choose the standard deviation of the weights for each layer $l$ according to the scheme proposed by He et al. (2015):

$$
\sigma = \sqrt{\frac{2}{k_l^2 \cdot d_{l-1}}},
$$

which takes into account the filter size $k_l$ and the number of filters of the last layer $d_{l-1}$. We determine the mean of the bias such that the average of the activations in every point of our network is positive and slightly decreasing over time. Liang and Hu (2015) use local contrast normalization at all layers to the same effect, which requires more GPU memory for the hidden layer activations. Due to our larger inputs and outputs and the increased number of time steps, current GPU memory restrictions prevent us from doing the same.

We learn the parameters of our network with backpropagation through time (BPTT) using RMSProp, which is a variant of resilient backpropagation (RPROP, Riedmiller and Braun 1993) suitable for mini-batch learning (Dauphin et al., 2015). RPROP and RMSProp to a large degree consider only the sign of the gradient, thus being robust against vanishing and exploding gradients, both common phenomena in RNN training.

During learning, we apply dropout (Srivastava et al., 2014). Combining dropout with RNNs is delicate, however. If it affects recurrent connections, their ability to learn long-range dependencies suffers (Pham et al., 2014). Thus, we apply dropout only to the final convolution with non-shared weights that
computes the network output and find that using dropout consistently improves our results.

3. Related Work

Multi-class semantic segmentation of images has a long history. Early approaches (e.g. He et al., 2004; Shotton et al., 2006) optimize continuity over pixels in a conditional random field (CRF) with pairwise affinity potentials and unary potentials from a learned classifier. Since pixels contain only a limited amount of information, classifiers take regions into account. Most works (e.g. Batra et al., 2008; Russell et al., 2009) start with an unsupervised method to oversegment images into homogeneous regions called superpixels. The neighborhood relations of superpixels define a graph, whose node labels can be optimized in a CRF. More recently, approaches using neural networks have been successful (Schulz and Behnke, 2012; Ciresan et al., 2012; Sermanet et al., 2014). These approaches can model classification and smoothness simultaneously, but—in line with our findings here—can still profit from CRF postprocessing (Chen et al., 2014, 2015) and oversegmentation (Gupta et al., 2014). Zheng et al. (2015) and Liu et al. (2015) even suggest choosing the last part of the neural network architecture in a way that it resembles trainable versions of these postprocessing mechanisms. Long et al. (2015a) then demonstrated that a high resolution output can also be obtained without postprocessing using fully convolutional networks, which are created by replicating ImageNet-trained image classification networks over the image and finetuning them on the segmentation task. Noh et al. (2015) further improve high resolution object delineation by refining and classifying object proposals with deconvolutional networks. While these approaches yield high accuracy on the PASCAL VOC segmentation challenge, they rely heavily on pre-training on a separate large classification dataset.

Several groups have used neural networks to process image sequences. Most works use stacks of frames to provide temporal context to the neural network. Le et al. (2011) and Taylor et al. (2010) learn hierarchical spatio-temporal features on video sequences using Gated Convolutional Restricted Boltzmann Machines (convGRBM) and Independent Subspace Analysis (ISA), respectively. Their image features are not learned discriminatively and the models do not allow localized predictions.

Simonyan and Zisserman (2014) use a two-stream architecture for action recognition, which separately creates high-level features from single-frame con-
tent and motion. Motion is provided through a stack of optical flow images, so that the modeled complexity is limited by the stack size. In our experiments, we found that increasing temporal context by providing frames consecutively yields improved performance.

More recently, Michalski et al. (2014) introduced a model designed to explicitly represent and predict the movement of entities in images. The architecture is chosen in a way that higher layers represent invariances of derivatives of positions (motions, accelerations, jerk). Our models do not explicitly model motion. However, our models can make use of deep layers even in the case no high-level position invariances exist, since in addition to motion, they also encode static content. Furthermore, in our model, deep layers have a lower resolution and facilitate transport of information across longer distances.

Jung et al. (2014) introduce a multiple timescale recurrent neural network for action recognition, which uses neurons with fixed time constants. The model uses leaky integrator neurons, which limits the rate at which higher layer activations can change. It is trained and evaluated on a simplified version of the Weizmann Human Action Recognition dataset.

Various architectures for processing video data are explored by Karpathy et al. (2014). The architecture most similar to our model, slow fusion, uses weight sharing between time steps and merges them in higher layers. In their study, slow fusion yields best results. In contrast to classifying video sequences with a single label, we produce label output at pixel level.

RNNs were successfully used for object class segmentation by Graves (2012) and Pinheiro and Collobert (2014). Both works use recurrence only to increase spatial context, whereas we extend processing to the temporal domain.

Object recognition is another task where RNNs achieved state of the art results. In a recent work, Liang and Hu (2015) use convolutional layers unfolded in time similar to ours. Their architecture consists of a stack of five such layers interleaved with pooling and dropout layers. Similarly to previous works, Liang and Hu (2015) only use static information. Nevertheless, their model obtained superior results classifying images of multiple datasets.

Long-Short Term Memory (LSTM) units are capable of carrying information, at the original resolution, over long temporal distances. LSTM is often used in speech recognition (Graves et al., 2013) or language understanding (Sundermeyer et al., 2012), where e.g. a specific property of a distant word or sound might influence the semantics of the current input. In this paper, we opt for simple neural units instead. While we are also interested in learning long-range dependencies, we do not provide spatial or temporal context at the original resolution. Instead, our architecture maintains expressive low resolution context information in higher layers. This is more realistic for natural images, where correlations are stronger between nearby pixels than those between distant ones. It also allows for sparser connectivity between units, since temporally distant units do not need to be wired.

Recent work of Bogun et al. (2015) uses LSTM units for object recognition in video sequences. They obtained state of the art results on the Washington Dataset (Lai et al., 2011) by incorporating information from several frames.
Their most successful strategy was to train the network unidirectionally and to use a bidirectional model, based on the same set of weights, during prediction. Sohn et al. (2015) introduce the Conditional Variational Autoencoder (CVAE). A CVAE is a conditional image label distribution \( p(y|\hat{y}, z) \), where \( x \) is the input image, \( \hat{y} \) is an initial guess of the output, and \( z \propto p(z|x, \hat{y}) \) is a latent variable. It is possible to sample from the CVAE and extend it to multiple scales. Our hierarchy of deterministic hidden states is also conditioned on the input and modulates a high resolution image label prediction. However, our proposed model employs recurrent connections not only to refine an initial guess in a single time step, but also to incorporate new knowledge from multiple consecutive video frames.

Pascanu et al. (2013) suggest that LSTM also addresses the problem of vanishing gradients. Here, we use RMSProp as optimization method, which—in addition to preventing vanishing gradients—also counteracts gradient explosion. Our architecture choices are motivated by the Neural Abstraction Pyramid of Behnke (2003), which performs pixel-wise classification tasks at input resolution as well. In contrast to our work, Behnke did not train on video sequences, but only on stationary patterns, which in some cases were corrupted by temporally changing noise. We also include modern architectural features, such as max-pooling and ReLU non-linearities, dropout, and use RMSProp to increase learning speed.

4. Experiments

We first conduct experiments on handcrafted datasets, which allow us to demonstrate important characteristics of our model. In a second step, we use our architecture for object class segmentation on a challenging RGB-D dataset.

4.1. Toy Experiments

We present three toy experiments, showing that our network is able to learn 1) filtering noisy information over time, 2) tracking and interpreting motion, and 3) retaining an internal state including uncertainty.

4.1.1. Denoising

In this experiment, we feed different degraded versions of the same binary image to the network. We use salt and pepper noise, uniformly distributed over the whole image. We also draw random black or white lines, to make the task more difficult. The task is to obtain the original image without noise. One way the network could solve this task would be to learn to average the image over time (which is our baseline). In addition, denoising filters learned by the neural network can remove high frequency noise.

To ensure that the network is able to generalize instead of learning an input by heart, we use different objects for training, validation and testing. Every split contains 100 independently generated sequences.
Figure 3: Toy experiment: Denoising. Rows represent, in order: the RGB input of the network for each time-step, the output of the softmax layer, the final outputs of the network, the evaluation (● True Positives ○ True Negatives ● False Positives ○ False Negatives). The last output is used for evaluation of pixelwise classification error.

Table 1: Denoising results for different models.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>93.3</td>
</tr>
<tr>
<td>CNN (all timesteps)</td>
<td>89.2</td>
</tr>
<tr>
<td>Thresholded average</td>
<td>81.6</td>
</tr>
</tbody>
</table>

Since the task has a low complexity, we opt for a simple convolutional model of only one hidden layer with 32 maps. A small filter size of 5x5 provides sufficient spacial context. We use $T=6$ temporal copies. During training, we optimize a weighted sum of the losses at all time steps, with a ten times larger weight placed on the final output. In all toy examples, we train for 12000 iterations with minibatches of size 16. For quantitative comparisons, we use the average class accuracy, i.e., the average of the diagonal of the confusion matrix between foreground and background pixels, after the confusion matrix rows have been normalized with the class prior.

Figure 3 shows an example from the test set. Our model is able to improve its prediction step by step, accumulating over time information even from the areas which are more affected by noise. After only two steps, the network is able to remove most of the false positives and to assemble together almost all features of the object. Table 1 and Figure 4 show that the RNN performance compares favorably to a naive thresholding of the average image.

We also train a baseline convolutional model without recurrent connections, but having access to the same amount of data. For this purpose, we concatenate
all images of the sequence and present them to the network simultaneously. The predictions produced by this baseline model (images not shown) indicate that it learned to combine information from multiple frames. However, the baseline accuracy of 89.2% is inferior to the accuracy of the recurrent architecture (93.2%).

4.1.2. Detecting Movement

In this experiment, we test the capabilities of the network to track a foreground object moving with constant speed through a noisy image. We use the same task as in the previous section (denoising a binary image), however, but this time we choose a smaller black object which is moving on a white canvas. Furthermore, to ensure that motion is the cue for tracking, we add two randomly placed distractor objects of the same shape and size in a random position at every time step. These distractors should be classified as background. To prevent the network from overfitting on motion direction and speed, we generate several sequences, each moving the object from a random position to another, with varying speed. The image size was 200×200, object at most 50 px in their largest dimension, and speed was between 9 and 23 pixels per time step.

Figure 5 shows the results obtained on this task using the unidirectional network. In the first time step, the network cannot decide which object is moving continuously. Already at \( t=2 \), however, the network detects a slight positional change from one of the objects, while the others are further away from their initial position. The softmax layer activations (second row) show
that the certainty of the hypothesis increases step by step. Also, one can notice that more details are added to the representation. Some false positives still exist when the new random position of a distractor object is close to its former position.

The baseline that we use for this experiment is also a convolutional neural network receiving the whole sequence input at once. We used a three layer architecture, as in the case of the recurrent counterpart. Our results show significant differences between the two models with respect to class accuracy. The recurrent network achieves a 89% accuracy, compared to 76.1% for the baseline.

4.1.3. Retaining Uncertainty

While in the previous experiments, we showed that the network is able to track a moving object, we now consider if a regular movement can be inferred from temporally distant information. We construct a dataset where two input frames are provided, with the same object present in different locations. Assuming linear motion, we want to predict the position of the object in the center. This task would be easy if both input frames were known at the same time, but we restrict access in our model as follows.

We use a bi-directional version of our model with weights shared between the past and future network parts. The two input images are provided to the first timestep of the “past” network and the first timestep of the “future” network, respectively. In this setup, the initial positions have to be remembered until
information from the past and future converges at the center time step. Since denoising is not an essential component of this task, we do not add noise.

Figure 6 depicts a sample sequence from the test set. As no motion information is provided, the best strategy of the network is to create a circular expanding hypothesis from the seen location. This time-dependent distribution over possible locations collapses when hypotheses from both timelines are combined. This is what we observe in the output maps of Figure 6. While the position is correctly identified, the shape of the object is largely lost.

As a baseline model, we also use a CNN with access to the same amount of data as our RNN. Due to the large spatial distance between objects, the baseline model is unable to determine the intermediate position in the motion. This leads to a class accuracy of merely 51.3%. Increasing the size of the filters from 7 to 13 leads to a class accuracy of 64.3%. Even with such an advantage, the baseline is more than 10% worse than the RNN, which achieves a class accuracy of 74.9% for this task.

4.2. RGB-D Object Class Segmentation

The NYU-Depth v2 (NYUD, (Silberman et al., 2012)) dataset is comprised of video sequences taken from 464 indoor scenes, annotated with a total of 894 categories. We use the popular relabeling into four high-level semantic categories, small objects that can be easily carried (“prop”), large objects that cannot be easily carried (“furniture”), non-floor parts of the room: walls, ceiling, columns (“structure”), and the floor of the room (“ground”).
Although NYUD was recorded as a video sequence, a subset of 1449 frames which were preprocessed and manually labeled is prevalent in the literature. The remainder—407,024 frames—consists of raw RGB-D camera images.

To transform the dataset into an image sequence dataset, but at the same time use the labeled frames for evaluation, we extract the past and future context of each labeled frame from the video stream and preprocess it similar to the labeled frames. For evaluation, we compare outputs corresponding to the labeled frame with the ground truth, retaining the same training/testing split as in the literature.

To preprocess the RGB and depth images, we follow standard procedure, sequentially applying lens correction\(^1\), projecting the depth readings into the RGB sensor frame, and filling-in missing depth readings (Levin et al., 2004).

To obtain ground truth information for unlabeled frames in our sequences, we propagate labels along optical flow directions. Due to manual labeling of the dataset, classes are often separated by small unlabeled regions, especially at edges which are crucial to the optical flow estimation. To address this problem, we use a colorization method (Levin et al., 2004) to determine missing labels close to labeled regions. Only the training set is modified, which allows direct comparison with other methods on the test set. After label fill-in, we compute optical flow (Farnebäck, 2003) on RGB image pairs. When label information is unavailable or ambiguous due to limitations in the optical flow computation, we write *unknown* labels, which are excluded from the computation of the loss and its gradient.

We also use optical flow to select which images are included in a sequence. Fast displacements are tricky to detect (Brox and Malik, 2011), while too slow motion results in quasi-static images, foiling our attempt to exploit the temporal context. Since the dataset was recorded at 30 Hz, taking every frame would result in no visible motion at input resolution. We thus decide to add a new image to the sequence once the optical flow shows, on average, a motion larger than half of our filter size.

Not all sequences in the dataset contain sufficient temporal context. In such cases, we complete them by replicating the first or last available frame.

### 4.2.1. Learning

We train the network with depth \(L=3\), an input resolution of 160×160 pixels, again using mini-batches of size 16, and a temporal context of 8 frames. As input, we use Histogram of Oriented Gradients (HOG) and Histogram of Oriented Depth (HOD) channels, the whitened version of the images (c.f. Höft et al., 2014), an estimated height map and the optical flow. The height map was computed by i) clustering normals in the depth map, ii) projecting onto the most vertical cluster mean, determining the lowest point, and iii) calculating the relative height of all other points in the depth map.

\(^1\)computed by OpenCV’s “Camera Calibration and 3D Reconstruction” package
Figure 7: Prediction for one of the NYUD dataset frames. Images (a) and (b) show RGB and depth, respectively, after being preprocessed. (c) and (d) represent the prediction and ground truth, respectively, where color codes “floor” ( ), “prop” ( ), “furniture” ( ), “structure” ( ) and “unknown” ( ). The network detects most of the pixels correctly, even some wrongly labeled ones (e.g., the third object on the table and the center of the wall-mounted piece). Note that in (c), “unknown” is superimposed from ground truth to aid comparison with ground truth.

We use randomly chosen 10% of the training set for validation (early stopping and model selection). While images are randomly transformed during training, we use a fixed, randomly picked set of transformations for validation to ensure stable estimates. The loss is measured at time steps $t=3$, 6, and 8. Training continues for 12,000 iterations, with an initial learning rate of $3 \cdot 10^{-4}$. The learning rate was automatically decreased three times when the validation error failed to improve. Once learning stopped, we retained the model which obtained the best performance on the validation set.

Figure 8 shows the evolution of the prediction for the structure class over the 8 input frames. The first row shows the RGB part of the input image, the second the softmax output for structure. The third row shows for which pixels the predicted class is structure, and the fourth row compares this prediction to ground truth when applicable (time steps 3, 6 and 8). As in Figure 7, we overlay “unknown” labels from ground truth to aid orientation in the image. Over time, the prediction is refined and false positives are reduced. It also seems that the network has learned to treat time steps 3 and 6, where the intermediate loss is measured, differently than other time steps.

4.2.2. Depth-Normalization

We integrate depth-normalized covering windows (Schulz et al., 2015a) into our learning algorithm. This approach evaluates the model on image patches at a spatial resolution which is dependent on the distance of the center pixel to the observer, effectively building scale invariance and adaptive output resolution into the model. Similar to Schulz et al. (2015a), this patch-based approach allows us to use smaller maps, which speeds up the training dramatically (days vs. weeks). As input, we used $80 \times 80$ instead of $160 \times 160$ pixels, and scaled hidden layers accordingly. At prediction time, the network has to process patches that fully cover the original image and combine their prediction afterwards. Thus,
Table 2: Comparison of NYUD classification performance to other state-of-the-art approaches without transfer learning. Our model with enabled covering windows (CW), unsupervised weight initialization (WI), and conditional random fields (CRF). Input consists of color/depth information, optionally with height (H). Baselines use convolutional neural networks (CNN), random forests (RF), and self-localization and mapping (SLAM). The “all-frames” model receives inputs from all frames at every time step.

<table>
<thead>
<tr>
<th>Method</th>
<th>Class Accuracies (%)</th>
<th>Average (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ground</td>
<td>struct</td>
</tr>
<tr>
<td>ours (CW, RGB-D only)</td>
<td>78.6</td>
<td>49.2</td>
</tr>
<tr>
<td>ours (CW)</td>
<td>95.8</td>
<td>74.6</td>
</tr>
<tr>
<td>ours (WI+CW)</td>
<td>94.9</td>
<td>76.8</td>
</tr>
<tr>
<td>ours (WI)</td>
<td>94.3</td>
<td>83.7</td>
</tr>
<tr>
<td>ours (WI+CW+CRF)</td>
<td>95.4</td>
<td>78.9</td>
</tr>
<tr>
<td>ours (WI+CRF)</td>
<td>94.2</td>
<td>83.9</td>
</tr>
<tr>
<td>all-frames</td>
<td>97.2</td>
<td>70.0</td>
</tr>
<tr>
<td>Schulz et al. (2015a) (CNN+CRF)</td>
<td>93.6</td>
<td>80.2</td>
</tr>
<tr>
<td>Müller and Behnke (2014) (RF+CRF)</td>
<td>94.9</td>
<td>78.9</td>
</tr>
<tr>
<td>Stückler et al. (2013) (RF+SLAM)</td>
<td>90.8</td>
<td>81.6</td>
</tr>
<tr>
<td>Couprie et al. (2013) (CNN)</td>
<td>87.3</td>
<td>86.1</td>
</tr>
<tr>
<td>Silberman et al. (2012) (RF)</td>
<td>68.0</td>
<td>59.0</td>
</tr>
</tbody>
</table>

Note that when using RGB-D input only, overall class accuracy drops to 56.2 %.

4.2.3. Comparison with State of the Art

Table 2 shows our result with and without depth-normalized covering windows (CW) together with state-of-the-art results on the same dataset. We compare against other methods which do not use transfer learning (e.g. by initializing filters using ImageNet-based transfer learning). Our RNN (WI+CW) outperforms all other published approaches. Overall, depth-normalization (CW) turns out to be worse for RNN, maybe indicating that there is not enough context to be interpreted in the chosen window sizes. However, CW is much better at identifying small objects (prop), at up to 64 % vs. 54.9 % for full frames.

Note that when using RGB-D input only, overall class accuracy drops to 56.2 %. With larger datasets, better RNN pretraining and growing GPU memory, larger models could be learned, removing the strong dependence on preprocessing.

We also investigated whether presenting video frames sequentially is an effective strategy. We used the best model and hyper-parameters, but provided all eight inputs at every RNN time step. This approach, labeled “all frames”, achieves significantly lower accuracy than the sequential model, despite the increased computational effort.

The outputs of our RNN can still be improved through other means. Instead of determining a pixel-wise maximum, we supply the probabilities as data term...
to the conditional random field (CRF) of Müller and Behnke (2014). The result (WI+CW+CRF and WI+CRF) improves performance even further. This indicates that our predictions are better suited for CRF postprocessing than CNN (Couprie et al., 2013; Schulz et al., 2015a) and random forests (Silberman et al., 2012; Müller and Behnke, 2014; Schulz et al., 2015b).

Comparison to Transfer Learning. Similar to non-recurrent neural networks (e.g. Erhan et al., 2010), our RNN profits from a weight initialization (CW vs. WI+CW). Supervised weight initialization using transfer learning on the large ImageNet database led to the (to the best of our knowledge) current top result on NYUD by Eigen and Fergus (2015), with a class accuracy of 82.0%. Our RNN only narrows the gap to the transfer learning approach. Weight initialization for recurrent connections is, therefore, an open but promising research direction.

4.2.4. Does Temporal Context Help?

Except for Stückler et al. (2013), none of the publications in Table 2 made use of temporal context to determine class labels. We would like to know whether our improvement is due to the fact that we use additional training data (albeit without labels), or through recurrent processing.

To check whether our network takes advantage of temporal context, we perform a static frame experiment. We use the same frame as input at all time steps during training and prediction. In this setting, the recurrent architecture is still able to learn long-range spatial dependencies, but cannot exploit temporal context, which results in accuracy reduction in both class accuracy
and pixel-wise accuracy (2.7 and 5.5 percentage points, respectively) relative to
the model which has access to temporal context. This model still outperforms
non-recurrent models, showing that spatial recurrent processing is the essential
improvement.

When using static frames as above, we remove both, temporal context and
additional ground truth generated through optical flow. Removing only addi-
tional ground truth during training results in a reduction of the class accuracy
by 2\%, while the pixel-wise accuracy decreases by 3.1\%. These controls suggest
that the superior accuracy of the RNN is mainly due to recurrent spatial and
temporal processing, and intermediate ground truth is important to help this
very deep — and mostly uninitialized — network learn.

5. Conclusion

In this work, we introduced a recurrent convolutional neural network archi-
tecture, which in addition to learning spatial relations is also able to exploit
temporal relations from video. We started with a series of toy examples that
showed that our networks are able to solve tasks that require denoising, detect-
ing movement, and retaining uncertainty.

We further carried out experiments on sequences of indoor RGB-D video se-
quences from the NYUD dataset. Combined with dropout, unsupervised weight
initialization, covering windows and conditional random fields, our proposed
model improves performance when compared to other non-recurrent baseline
models and random forests, obtaining the best results obtained so far without
transfer of features from ImageNet.

References

Batra, D., Sukthankar, R., Chen, T., 2008. Learning class-specific affinities for
image labelling. In: Conference on Computer Vision and Pattern Recognition
(CVPR).

2766 of Lecture Notes in Computer Science. Springer.


Brox, T., Malik, J., 2011. Large displacement optical flow: descriptor match-
ing in variational motion estimation. Transactions on Pattern Analysis and
Machine Intelligence 33.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A. L., 2014. Se-
matic image segmentation with deep convolutional nets and fully connected


