Semantic Segmentation of RGBD Videos with Recurrent Fully Convolutional Neural Networks

Ekrem Emre Yurdakul, Yücel Yemez
Computer Engineering Department, Koç University
Istanbul, Turkey
{ekyurdakul,yyemez}@ku.edu.tr

Abstract

Semantic segmentation of videos using neural networks is currently a popular task, the work done in this field is however mostly on RGB videos. The main reason for this is the lack of large RGBD video datasets, annotated with ground truth information at the pixel level. In this work, we use a synthetic RGBD video dataset to investigate the contribution of depth and temporal information to the video segmentation task using convolutional and recurrent neural network architectures. Our experiments show the addition of depth information improves semantic segmentation results and exploiting temporal information results in higher quality output segmentations.

1. Introduction

Semantic segmentation aims to assign an object class label to every pixel in a given image, and hence it is a key task in computer vision towards scene understanding. Before deep neural network architectures have been proved to be useful and have found practical applications in various vision tasks, semantic segmentation methods mostly relied on patch-wise training of hand-crafted features with conventional classifiers. The focus has recently shifted to methods that use neural networks. In particular, deep convolutional neural networks, when trained on large datasets, can achieve pixel-level segmentation on a given image and can solve the recognition and localization problems at the same time by globally analyzing the image as a whole.

With the introduction of fully convolutional neural networks [24], the use of deep neural network architectures has become popular for the semantic segmentation task. Most of the work done in this field is on RGB still images [1, 24, 26, 3, 23, 25, 37] with a few extensions to RGB videos, which make use of recurrent neural networks so as to incorporate temporal information into the segmentation task [1, 38, 30, 29]. Although there are numerous works on RGBD image segmentation [7, 14, 17, 8, 15], there exists currently no work in the literature, except [27], that addresses the problem of pixel-level segmentation on RGBD videos using neural networks. The primary reason for this is that while there exist a good number of RGB video and image datasets available with pixel-level annotation, such RGBD video datasets are fairly limited. Besides, most of the existing semantic segmentation methods transfer weights from pretrained deep convolutional neural networks such as the VGGnet [36] to train their architectures, whereas no such deep architectures pretrained on large datasets exist with RGBD data. We believe that the depth information contained in RGBD data can be exploited as an additional modality to improve the performance of the segmentation task.

In this work, we address the problem of pixel-level semantic segmentation on RGBD videos using both fully convolutional and recurrent fully convolutional neural networks. For our experiments, we use the Virtual KITTI dataset [11], which contains synthetic RGBD videos with pixel-level annotation. Our primary contribution is a fusion scheme based on existing recurrent and fully convolutional neural network architectures, that combines color and depth information in RGBD videos for semantic segmentation. We train our bimodal recurrent fully convolutional neural network with a progressive strategy based on weight transfer. We first convert depth images to color images, and initialize the convolutional layers with the weights of the pretrained VGGnet, and train two separate fully convolutional neural networks for color and depth modalities via backpropagation. We then transfer the weights of these unimodal architectures to the recurrent fully convolutional neural network to train our final bimodal architecture.

In Section 2, we review the related work on semantic segmentation particularly using neural networks. Section 3 describes the datasets that we use to test our neural network architectures which are then explained in Section 4. The experiments that we conducted are presented in Section 5, and the concluding remarks are finally given in Section 6.

2473-9944/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCVW.2017.51

2. Related Work

The ImageNet challenge [32] has led to (very) deep convolutional neural network architectures such as AlexNet [20], VGGnet [36] and ResNet [16]. Although these networks were each originally trained with the goal of RGB-based image classification, since the RGB image dataset on which they were trained was very large, they can easily be generalized to other datasets as well as to various other vision tasks. The most common approach to transfer learning from these pretrained deep convolutional neural networks (CNNs) is either to use them as feature extractors or to transfer weights for network initialization [41]. This type of transfer learning from pre-trained networks is commonly employed by the state of the art semantic segmentation methods.

Semantic segmentation networks existing in the literature differ in their architectures, such that they may be fully convolutional [24], composed of a single deconvolutional layer [24] or mirror the convolutional section of the network with the corresponding deconvolution and unpool operations [1, 26]. Multiple processing streams in a single network is also a common choice of network architecture [35, 40]. If temporal data is present, recurrent layers can be added to exploit temporal relationships.

There exist relatively few works in the literature, that address the problem of semantic pixel-level RGB video segmentation [29, 38, 30, 33] with experiments conducted on various datasets such as [28, 6, 12, 21]. Convolutional recurrent layers are a crucial building block of the semantic segmentation networks that exploit temporal data. Whereas recurrent layers accept vector inputs and produce vector outputs, convolutional recurrent layers take matrices as inputs and produce matrices as outputs. As a result, spatial information in convolutional recurrent layers is preserved and can be exploited as opposed to standard recurrent layers [22]. In our work, we use convolutional recurrent layers similar to [33, 38].

Although there are numerous works on RGBD image segmentation [7, 14, 17, 8, 15], to the best of our knowledge, there exists no semantic segmentation method on RGBD video data using neural networks, except maybe [27], which can be viewed as a preliminary work in this field, that uses a small dataset *NYUv2* [34] of only 1449 frames and an ad-hoc simplistic network architecture with no transfer learning but with handcrafted features as input. Furthermore, in [27], the ground-truth segmentation for all frames of the *NYUv2* dataset is estimated from the available sparse annotation based on optical flow. We also note that there exist few other segmentation methods, such as [17], which take RGBD videos as input, but then use the extracted temporal information to enhance segmentation on sparsely distributed individual frames (for which annotations are available).

Since producing RGBD video datasets with pixel-level annotation is a much harder task compared to RGB video datasets, RGBD video segmentation remains as an unexplored field. There exist however very recent RGBD video datasets [11, 31], which provide segmentation annotation at pixel level and has hence enabled us to conduct research on this problem.

3. Datasets

We use the *Virtual KITTI* dataset [11], which contains photo-realistic synthetic videos with pixel accurate ground truth for scene- and instance-level segmentation, and depth. Aimed at providing data for autonomous driving, this dataset contains video sequences of cars navigating roads. There are 5 different worlds, each rendered in 10 variations under different camera angles, weather conditions and time of day; 30-degrees-left, 30-degrees-right, 15-degrees-left, 15-degrees-right, clone; morning, sunset; overcast, fog and rain. Each world has a different number of video frames, however the number of video frames remains the same in a world. The number of video frames in each world is 447, 233, 270, 339 and 837 respectively. We regroup all of the objects in the dataset into a total number of 13 classes, ignoring the instance level categorization of the original annotation, where for example the ”car” class had several instances. Ground truth segmentations in the dataset are provided as RGB color images, which we convert into the one-hot matrix representation compatible with the softmax classifier.

The images in this dataset have a resolution of 1242×375 pixels, however we resize all of the images to a resolution of 224×224 pixels without preserving their aspect ratios. This downsampling enables us to train our networks more efficiently in regards to memory and computation.

From the gray-scale depth images provided in the dataset, we generate additional data which is colorized depth images by applying a jet colormap ranging from red (near) over green to blue (far) as in [9] to be able to benefit from the pretrained VGG weights. The standard jet colormap is however linear, hence does not fully exploit the color spectrum. Thus, we adopt a non-linear version of the jet colormap. For this, we quantize the available range of depth values nonuniformly and assign the same color to multiple depth values. This nonlinear assignment is achieved by first building a histogram of the available depth values and then merging bins such that the new histogram has roughly the same number of occurrences in all the bins. We find the depth value that has the most number of occurrences and use this number as the bin size, which results in a total of 412 bins in our RGBD dataset. Hence each depth image in the dataset is eventually represented by using 412 depth levels, i.e. colors. We note that this nonlinear encoding scheme is dataset specific and computed once over the training set. We do not process the original RGB and depth images in any other additional way.

We also use the *DAVIS* dataset [28] to show that our method works on real videos, even though the dataset is composed of RGB-only videos. There are 50 Full HD videos totaling in 3455 frames with challenges such as occlusion, motion-blur and appearance changes. The dataset provides pixel-level annotation into 2 categories (background, foreground) and we apply the same downsampling operation performed in the case of the *Virtual KITTI* dataset.

4. Method

We use custom layers in our recurrent convolutional networks, which we call C-RNN, C-LSTM and C-GRU. Simply put, these layers are extensions of the standard RNN [10], LSTM [18] and GRU [4] layers; which take matrices as inputs and produce matrices as outputs, rather than taking vectors as inputs and outputting vectors. This extension from vectors to matrices is achieved by replacing multiplication operations with convolutions.

4.1. Recurrent Convolutional Layers

Convolutional RNN Layer (C-RNN) is the convolutional counterpart of the RNN layer. It is the most simple recurrent convolutional layer and is dictated by a single equation:

$$\boldsymbol{H}_t = \text{relu}(\boldsymbol{W}_1 * \boldsymbol{X} + \boldsymbol{W}_2 * \boldsymbol{H}_{t-1} + \boldsymbol{b}) \tag{1}$$

where $*$ denotes the convolution operation, $\boldsymbol{H}_t$ is the hidden state matrix, *relu* is the rectified linear unit (ReLU), $\boldsymbol{X}$ is the input matrix, $\boldsymbol{W}_1$ *and* $\boldsymbol{W}_2$ are the parameter matrices, and $\boldsymbol{b}$ is the bias vector.

Convolutional LSTM Layer (C-LSTM) is the convolutional counterpart of the LSTM layer and calculates its gates and state according as follows:

$$\begin{aligned}
\boldsymbol{I} &= \text{sigm}(\boldsymbol{W}_1 * \boldsymbol{X} + \boldsymbol{W}_2 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_1)\\
\boldsymbol{F} &= \text{sigm}(\boldsymbol{W}_3 * \boldsymbol{X} + \boldsymbol{W}_4 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_2)\\
\boldsymbol{O} &= \text{sigm}(\boldsymbol{W}_5 * \boldsymbol{X} + \boldsymbol{W}_6 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_3)\\
\boldsymbol{J} &= \tanh(\boldsymbol{W}_7 * \boldsymbol{X} + \boldsymbol{W}_8 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_4)\\
\boldsymbol{C}_t &= \boldsymbol{C}_{t-1} \circ \boldsymbol{F} + \boldsymbol{I} \circ \boldsymbol{J}\\
\boldsymbol{H}_t &= \tanh(\boldsymbol{C}_t) \circ \boldsymbol{O}
\end{aligned} \tag{2}$$

where $\circ$ denotes the Hadamard product, $\boldsymbol{I}$ is the input gate matrix, $\boldsymbol{F}$ the forget gate matrix, $\boldsymbol{O}$ the output gate matrix, $\boldsymbol{C}_t$ the cell state matrix, $\boldsymbol{H}_t$ the output matrix, $\boldsymbol{X}$ the input matrix, *sigm* the sigmoid function, *tanh* the hyperbolic tangent function, $\boldsymbol{W}_1$ to $\boldsymbol{W}_8$ are parameter matrices and $\boldsymbol{b}_1$ to $\boldsymbol{b}_4$ are bias vectors.

Convolutional GRU Layer (C-GRU) is the counterpart of the GRU layer, defined by the following equations:

$$\begin{aligned}
\boldsymbol{Z} &= \text{sigm}(\boldsymbol{W}_1 * \boldsymbol{X} + \boldsymbol{W}_2 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_1)\\
\boldsymbol{R} &= \text{sigm}(\boldsymbol{W}_3 * \boldsymbol{X} + \boldsymbol{W}_4 * \boldsymbol{H}_{t-1} + \boldsymbol{b}_2)\\
\boldsymbol{H}_h &= \tanh(\boldsymbol{W}_5 * \boldsymbol{X} + \boldsymbol{W}_6 * (\boldsymbol{R} \circ \boldsymbol{H}_{t-1}) + \boldsymbol{b}_3)\\
\boldsymbol{H}_t &= \boldsymbol{Z} \circ \boldsymbol{H}_{t-1} + (1 - \boldsymbol{Z}) \circ \boldsymbol{H}_h
\end{aligned} \tag{3}$$

where $\boldsymbol{Z}$ is the update gate matrix, $\boldsymbol{R}$ the reset gate matrix, $\boldsymbol{H}_t$ the output matrix, $\boldsymbol{X}$ the input matrix, *sigm* the sigmoid function, *tanh* the hyperbolic tangent function, $\boldsymbol{W}_1$ to $\boldsymbol{W}_6$ the parameter matrices and $\boldsymbol{b}_1$ to $\boldsymbol{b}_3$ are the bias vectors.

C-RNN is the easiest layer to train, however has the least number of parameters. In order to increase the learning capacity of networks, C-LSTM is a more preferable choice, since it has more parameters than C-RNN with internal memory, but increases the time required for training. To speed up training C-GRU can be used, which provides similar performance compared to C-LSTM with less parameters than C-LSTM and more parameters than C-RNN [5]. In the semantic segmentation task we focus on, we expect C-LSTM to perform better than the C-RNN and C-GRU variants, since it has the most number of parameters and possesses internal memory that is potentially capable of learning more information from successive video frames in dealing with challenging conditions such as occlusions and uncovered regions.

Although the parameters of the convolution operations (stride, window size, padding...) in our custom layers are tunable, we set them the same as the convolution layers in our networks, since the results of our initial experiments presented the most performance boost with this configuration.

4.2. Networks

Our networks have different architectures to accommodate various input data types shown in Table 1.

Network	Input Type	Input Channels
D	Colorized Depth	3
RGB	Color	3
RGBD	Color and Colorized Depth	3 + 3

Table 1. Input types accepted by our networks. *-D* and *-RGB* networks accept a single input, whereas the *-RGBD* networks accept dual inputs.

Network Architectures We base our network architectures on the VGG-19 (Model E) [36] network by mirroring its first 12 convolution and 3 pooling layers. *-D* and *-RGB* networks share the same architecture; accept a single input of size 224×224×3, but their input types are different. *-RGBD* networks on the other hand, accept two inputs of the same size 224×224×3, where each input is processed

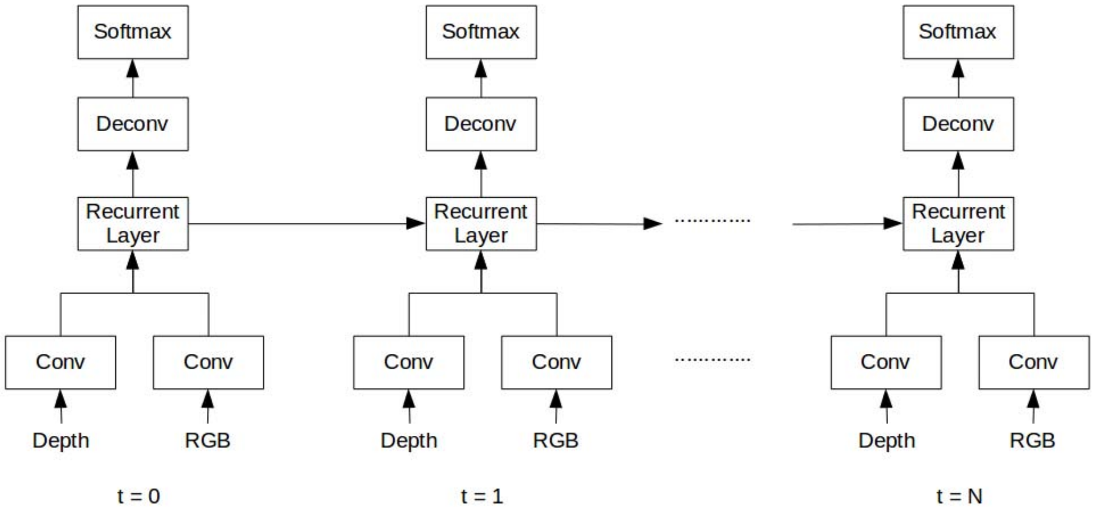

Figure 1. Block diagram of the *-RGBD* networks unrolled in time starting from *t=0* to sequence length *t=N*.

TVGG-19
conv3-64 (×2)
pool
conv3-128 (×2)
pool
conv3-256 (×4)
pool
conv3-512 (×4)

Table 2. Layers of *TVGG-19*, which are repeated consecutively by the number of times written in parenthesis.

Conv	RNN	LSTM	GRU
Input	Input	Input	Input
TVGG-19	TVGG-19	TVGG-19	TVGG-19
	C-RNN	**C-LSTM**	**C-GRU**
deconv	deconv	deconv	deconv
Softmax	Softmax	Softmax	Softmax

Table 3. Architectures the of *-D* and *-RGB* networks, layer by layer.

by a different stream in the network. In the end, the two streams are fused together in the deconvolution layer [24] by element-wise addition. Tables 3 and 4 show the architectures of our networks layer by layer, and Figure 1 visualizes the structure of the *-RGBD* networks. Only the *Conv* network is fully convolutional, whereas the *RNN, LSTM, GRU* networks are recurrent fully convolutional networks. In the *-RGBD* networks, depth and RGB are processed by two different steams of convolution layers. Next, they are fused by elementwise addition and fed to the convolutional recurrent layer, which is either a C-RNN, C-LSTM or C-GRU layer. The output of the convolutional recurrent layer is then fed to the deconvolution layer [24] and softmax is applied, which determines the final output.

In all of our networks, all convolution and deconvolution layers are followed by ReLU activations, which are not shown for brevity. All convolution layers have padding=1 and stride=1. Deconvolution layers have padding=2 and stride=4. Pooling operations are maximum pooling with stride=window=2. We call the first 15 layers of the *VGG-19* network *TVGG-19* (Trimmed VGG-19) and use it as a building block in our networks. The architecture of *TVGG-19* is given in Table 2.

Initialization and Weight Transfer Initial weights of the *Conv-D* and *Conv-RGB* networks are picked as the weights of the convolution layers of the VGG-19 network (Model E) pretrained on ImageNet. With this initialization, we benefit from the data learned by the VGG-19 network, since our data is composed of RGB and colorized depth. In training the *Conv-RGBD* network, we use the trained weights from the *Conv-D* and *Conv-RGB* networks as the initial weights of the convolutional layers in the corresponding streams. This transfer of weights reduces training time by increasing the convergence rate and improves pixel-wise accuracy. The recurrent fully convolutional networks are then initialized with the trained weights of the fully convolutional networks. This second transfer makes it easier for the recurrent fully convolutional networks to learn. For example, after having trained the *Conv-D* and *Conv-RGB* networks, *LSTM-D* and *LSTM-RGB* can be trained. It is possible to train *LSTM-RGBD* only after the trainings of *LSTM-D* and *LSTM-RGB* are completed. This requirement is the

Conv		RNN		LSTM		GRU	
Depth	RGB	Depth	RGB	Depth	RGB	Depth	RGB
TVGG-19	TVGG-19	TVGG-19	TVGG-19	TVGG-19	TVGG-19	TVGG-19	TVGG-19
		C-RNN		**C-LSTM**		**C-GRU**	
deconv		deconv		deconv		deconv	
Softmax		Softmax		Softmax		Softmax	

Table 4. Structure of the *-RGBD* networks layer by layer. In each network, there are two different streams (TVGG-19) one for depth and one for RGB data.

same for the *RNN* and *GRU* networks. As opposed to no weight transfer at all, our weight transfer scheme provided about a 10% accuracy boost in our initial experiments. In all of our networks, we initialize the deconvolution weights with the bilinear distribution [24], the weight matrices and biases of the C-RNN, C-LSTM and C-GRU layers with the Xavier distribution [13] and zeros, respectively.

5. Experiments

We employ end-to-end training by minimizing the pixel-wise negative log-likelihood with Adam [19] and use pixel-wise accuracy as our quantitative evaluation metric. In all settings, the learning rate is 10^{-5}. We train *Conv-D* and *Conv-RGB* networks for 100, *Conv-RGBD* for 50 and the remaining networks for 25 epochs. *Conv-D* and *Conv-RGB* networks have a minibatch of size 24, whereas *Conv-RGBD* has of size 16. The remaining networks do not have mini-batching. The recurrent networks are trained with back-propagation through time (BPTT) algorithm [39], where the number of time steps *D* is 1.

We implemented all of our networks in the programming language Julia [2] with the deep learning package Knet [42], which provides GPU support. We used a single NVIDIA K80 in a high performance cluster to train each network, which took about between 6 hours and 2.5 days for a single network, depending on the configuration and amount of training data. At test time, our *Conv-RGBD* and *LSTM-RGBD* networks perform semantic segmentation at 18 and 13 FPS (frames per second) on average for a single frame, respectively. Our code and pretrained models will be made available on GitHub.

We have two different setups for the *Virtual KITTI* dataset. In all setups, we use all of the worlds in the dataset. **Setup 1** is designed to measure the variance of the networks to the angle of the camera. *30-deg-left, 30-deg-right, clone* variations are used for training; *15-deg-left, 15-deg-right* variations are used for testing. **Setup 2** is more challenging than the first setup; the first halves of all the variations are used for training and the second halves are used for testing. In the first setup there are 6378 train and 4252 test images, and in the second setup there are 10630 train and test images. We choose the best performing network on the first setup and continue the rest of our experiments with that network, which is the *LSTM-RGBD* network. Therefore, we no longer train the RNN and GRU networks for setup 2.

The results acquired are provided in Table 6. In each setup, the best performing network is written in bold. We test both the linear and nonlinear depth encoding schemes (described in Section 3) in setup 2, while in setup 1 we experiment only using the linear depth encoding scheme. We first observe that *LSTM-RGBD* and *Conv-RGBD* outperform all other networks in the first and second setups, respectively. Although we expected *LSTM-RGBD* to be the winner also in the second setup, this is not the case. This might be due to the fact that the number of training images in setup 2 is much higher than in setup 1 and the *LSTM-RGBD* network cannot fully exploit the temporal information on a scale this large. Despite slightly lower accuracy, the *LSTM-RGBD* network outputs higher quality segmentations as in Figure 2, where the boundaries around the cars are sharper and finer details are much more visible. We also observe that, in setup 2, the nonlinear depth encoding scheme increases the accuracy of depth-only *LSTM-D* by 5.43% and reduces the gap between *Conv-RGBD* and *LSTM-RGBD* from 0.48% to 0.14%. In any case, *-RGBD* networks perform better in test accuracy and show the addition of depth information improves segmentation results and quality.

Network	Test Accuracy (%)
Conv-RGB	94.56
LSTM-RGB	**95.22**

Table 5. Results acquired on the DAVIS dataset.

For the experiments on the *DAVIS* dataset, we use all of the video sequences provided in the dataset and train our networks with the first halves of the sequences. The remaining second halves are used for testing. Since there is no depth data provided in this dataset, we can not train *-D* and *-RGBD* networks and only train *-RGB* networks.

Table 5 presents our results on the *DAVIS* dataset. *LSTM-RGB* network architecture performs better in test accuracy and provides higher quality visual segmentation outputs compared to the *Conv-RGB* network as in Figure 3. The

	Test Accuracy (%)		
Network	Setup 1	Setup 2 (linear encoding)	Setup 2 (nonlinear encoding)
Conv-D	73.73	73.75	76.99
Conv-RGB	84.87	80.01	80.01
Conv-RGBD	85.81	**82.70**	**82.76**
RNN-D	69.58	-	-
RNN-RGB	84.09	-	-
RNN-RGBD	85.81	-	-
LSTM-D	68.68	71.49	76.92
LSTM-RGB	85.26	79.95	79.95
LSTM-RGBD	**86.23**	82.22	82.62
GRU-D	69.81	-	-
GRU-RGB	85.14	-	-
GRU-RGBD	85.89	-	-

Table 6. Results acquired on the *Virtual KITTI* dataset.

boundaries are sharper and the general shape of objects are much more precise. Finer details are captured better by the *LSTM-RGB* network, while the *Conv-RGB* network generates coarser segmentations.

6. Conclusion

In this paper, we have explored the performance of recurrent fully convolutional networks in the domain of RGB and RGBD videos for the semantic segmentation task. Through experiments, we have shown that fully convolutional networks possess the potential to benefit from depth information when combined with RGB data. In adding recurrent layers to fully convolutional networks, we observe further improvement in quantitative results and achieve better quality in visual segmentation results. By transferring weights from fully convolutional networks to recurrent convolutional networks, we manage to reduce the training time of recurrent convolutional networks and also improve the accuracy.

Our future goals are to explore the effects of increasing the number of time steps used in backpropagation through time in our recurrent networks. We also aim to improve the input resolution, deepen our networks with more convolutional and recurrent layers and develop different fusion methods by transfer learning from RGB to depth. Performing experiments on real RGBD video datasets, rather than synthetic as in our current experiments, may also prove interesting.

7. Acknowledgements

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) Grants 114E628 and 215E201.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, abs/1411.1607, 2014.

[3] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[5] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[7] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. *CoRR*, abs/1301.3572, 2013.

[8] Z. Deng, S. Todorovic, and L. J. Latecki. Semantic segmentation of rgbd images with mutex constraints. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1733–1741, 2015.

[9] A. Eitel, J. T. Springenberg, L. Spinello, M. A. Riedmiller, and W. Burgard. Multimodal deep learning for robust RGB-D object recognition. *CoRR*, abs/1507.06821, 2015.

[10] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[11] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. *CoRR*, abs/1605.06457, 2016.

[12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[14] S. Gupta, R. B. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. *CoRR*, abs/1407.5736, 2014.

[15] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *ACCV*, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[17] Y. He, W. Chiu, M. Keuper, and M. Fritz. RGBD semantic segmentation using spatio-temporal data-driven pooling. *CoRR*, abs/1604.02388, 2016.

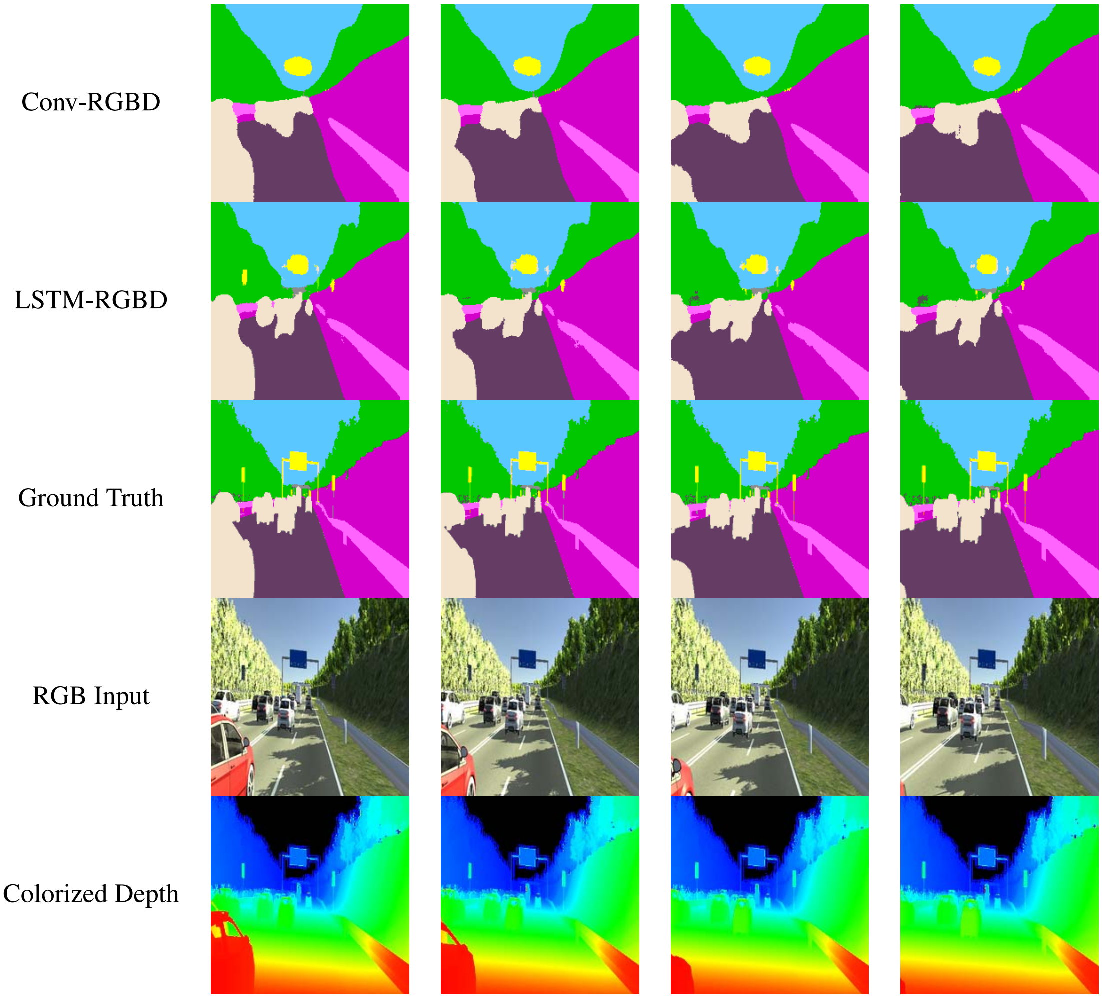

Figure 2. Comparison of output segmentations of the *Conv-RGBD* and *LSTM-RGBD* networks obtained on the *clone* sequence of world *0020* in the *Virtual KITTI* dataset for setup 2 with nonlinear depth encoding. The third row shows the ground truth annotations and the last two rows are inputs to the networks at time *t*. The horizontal axis (from left to right) depicts time.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory, 1997.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] F. Li, T. Kim, A. Humayun, D. Tsai, and J. M. Rehg. Video segmentation by tracking many figure-ground segments. In *ICCV*, 2013.

[22] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[23] Z. Liu, X. Li, P. Luo, C. Loy, and X. Tang. Semantic image segmentation via deep parsing network. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[25] D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla. Semantic segmentation of aerial images with an ensemble of cnns. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 473–480, June 2016.

[26] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[27] M. S. Pavel, H. Schulz, and S. Behnke. Recurrent convolutional neural networks for object-class segmentation of rgb-d video. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015.

[28] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool,

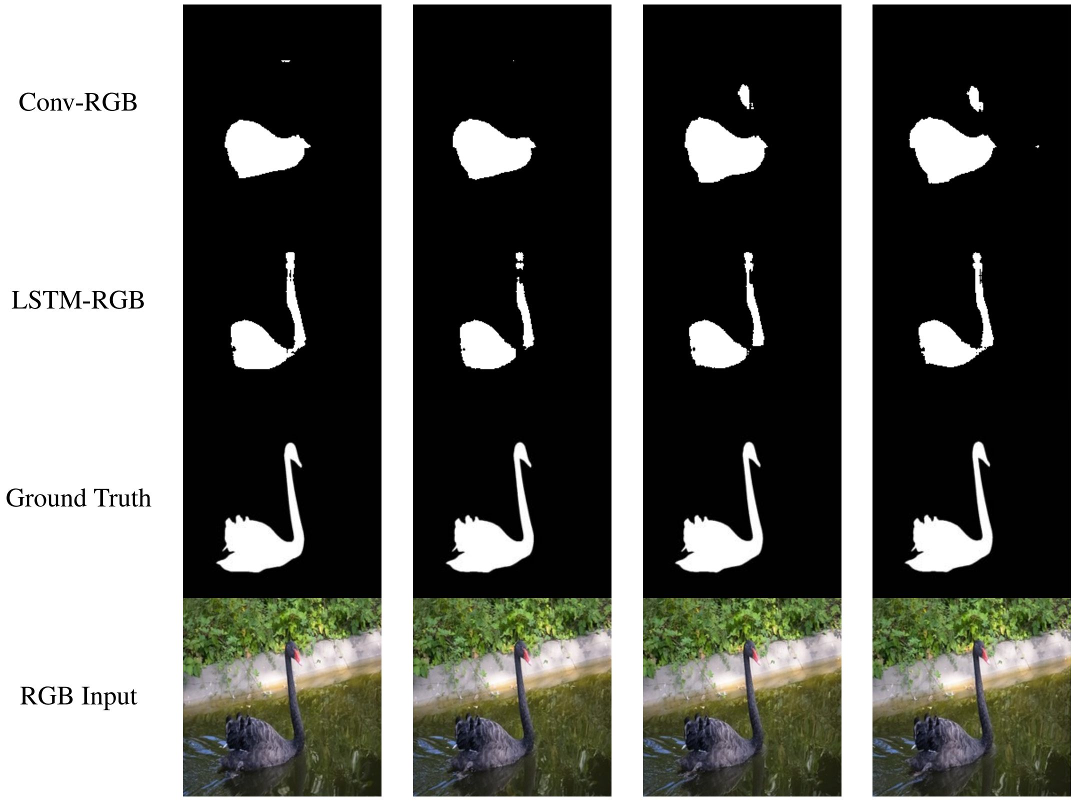

Figure 3. Comparison of output segmentations of the *Conv-RGB* and *LSTM-RGB* networks on the *blackswan* sequence in the *DAVIS* dataset. The first and second rows are outputs of the *Conv-RGB* and *LSTM-RGB* networks, respectively. The third row shows the ground truth annotations and the last row the input at time *t*. The horizontal axis (from left to right) depicts time.

M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Computer Vision and Pattern Recognition*, 2016.

[29] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. Full-resolution residual networks for semantic segmentation in street scenes. *CoRR*, abs/1611.08323, 2016.

[30] M. Ren and R. S. Zemel. End-to-end instance segmentation and counting with recurrent attention. *CoRR*, abs/1605.09410, 2016.

[31] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[33] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[34] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[35] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[37] S. Tsogkas, I. Kokkinos, G. Papandreou, and A. Vedaldi. Semantic part segmentation with deep learning. *CoRR*, abs/1505.02438, 2015.

[38] S. Valipour, M. Siam, M. Jägersand, and N. Ray. Recurrent fully convolutional networks for video segmentation. *CoRR*, abs/1606.00487, 2016.

[39] P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[40] Z. Wu, Y. Jiang, X. Wang, H. Ye, X. Xue, and J. Wang. Fusing multi-stream deep networks for video classification. *CoRR*, abs/1509.06086, 2015.

[41] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[42] D. Yuret. Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS 2016*, 2016.