Effects of Spatial Transformer Location on Segmentation Performance of a Dense Transformer Network

David Abou Chacra University of Waterloo, ON, Canada
John Zelek University of Waterloo, ON, Canada

Abstract

Semantic segmentation solves the task of labelling every pixel in an image with its class label, and remains an important unsolved problem. While significant work has gone into using deep learning to solve this problem, almost all the existing research uses methods that do not make modifications on spatial context considered for the pixel being labelled. Spatial information is an important cue in tasks such as segmentation, reusing the same spatial span for every pixel and every label may not be the best approach. Spatial Transformer Networks have shown promising results in improving classification performance of existing networks by allowing networks to actively manipulate their input data to achieve better performance. Our work shows the benefit of incorporating Spatial Transformer Networks and their corresponding decoders into networks tailored to semantic segmentation. Our experiments show an improvement in performance over baseline networks when using networks augmented with Spatial Transformers.

1 Introduction

Deep learning has shown promise on a vast variety of tasks: image classification, autonomous driving, natural language processing, and text translation are a few notable examples of fields where deep learning is the current state-of-the-art (and by a wide margin). Deep learning has also been applied to the task of semantic image segmentation. Semantic segmentation is an image-to-image translation task that takes an image as input and must learn to output another image of the same size, where each pixel in the output image describes the class of that same pixel in the input image. Semantic segmentation is still largely an open problem; larger still is the problem instance segmentation, which aims to classify different instances of the same class separately. Semantic segmentation can be used in a variety of different applications, making it an interesting and useful problem to tackle.

A variety of different methodologies, reflected in a large array of different network architectures, have been applied to this problem. Region-based approaches, namely R-CNN and its successors, encoder-decoder methods, including FCNs [5], Segnet [1] and U-Net [6], and methods using dilated convolutions, such as Deeplab and its derivatives. While significant research has gone into the the application of deep learning techniques to image segmentation, one common underlying trait in all of these segmentation approaches is the use of a regular, mainly square, area around a pixel to define its class. Every image in the train and test sets is treated in a similar manner, with no regard for image content or class context. Even architectures that are applied at multiple scales, still only take square regions, except at finer or coarser scales.

As a solution to this, researchers at Google DeepMind developed Spatial Transformer Networks (STNs)[3]. The aim of their work was to allow networks to spatially manipulate their input data in a way that improves their performance. STN modules vary in shape, but are trainable modules that can be inserted into a network in different locations, and learn to transform input data spatially into convenient representations that make the subsequent network tasks simpler, thereby allowing for better performance. They showed that the inclusion of STNs into existing networks improves their performance, and outperformed the state-of-the-art on challenging classification datasets, namely the CUB-200-2001 bird classification dataset [4].

As of yet, there is only one other work applying STNs to image segmentation: Dense Transformer Networks [4]. In their work, Li et al. develop the dense transformer decoder layer, which maps STN transformed feature vectors back to their original space, allowing spatial correspondence to be preserved between the feature maps before the transformer, and after the decoder [4]. Their work verifies the differentiability of this 'decoder' function, allowing it to be used in deep networks and trained by back-propagation.

The aim of this work is to explore the utility of STNs inside encoder-decoder frameworks for image segmentation. While this was briefly explored in [4], our work further explores the effects of the STN location, along with the overall network depth. We show that properly placed STN modules inside of an encoder-decoder architecture can in fact have a positive effect on network performance for segmentation tasks (see figure 1).

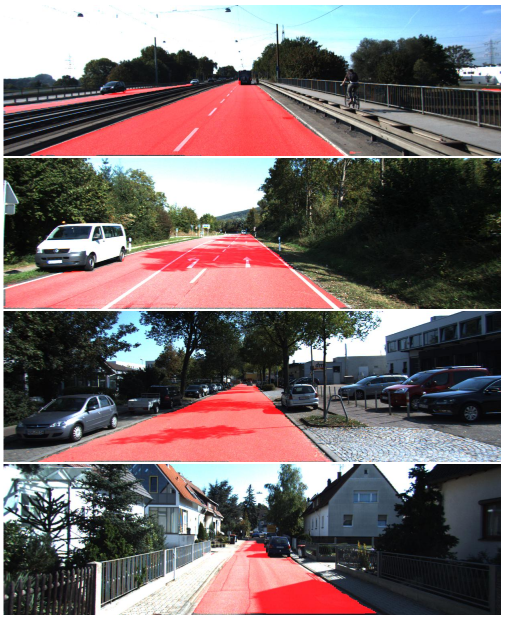
Fig. 1: Sample results on a test images from the KITTI dataset. The top two rows are from a less challenging marked road class, whereas the bottom two rows are from a more challenging class where road is unmarked.

2 Background

2.1 Deep Learning for Semantic Segmentation

Initial deep learning approaches to semantic segmentation were simply patch classification networks, where patches around each input pixel were fed into the network, and this pixel's label was predicted. This meant that networks with fully connected layers could be used in the task of segmentation, but it also meant that segmentation required forward passes equalling the number of pixels for every image, which meant even for small input images at test time, these networks would require significant processing power and time: A 256x256 image for example requires 65,536 forward passes to semantically segment it.

This led to the use of Fully Convolutional Networks (FCN) [5] which do away completely with the fully connected layers, allowing networks to output where class activations happened. FCNs were the first deep learning architectures to be tailored specifically for semantic segmentation, and they used transpose convolution layers to get back to an image the same size as the input, and also introduce skip connections to improve the final output. Transpose convolutions (contentiously referred to as 'deconvolutions') are actually already done when training any convolutional neural network with backpropagation, as the backwards pass of the training algorithm applies transpose convolutions. Despite the usefullness of the FCN architechture, the reliance on the coarsely downsampled activations led to a coarse segmentation. Follwing FCN's, Segnet [1] utilized an encoder-decoder architecture with increased skip connections between max-pooling and un-pooling layers (where the fine details were usually lost) to preserve more of the fine detail in the final segmentation. Finally, the U-Net [6] architecture also employs this encoder-decoder framework, doubling the feature map depth with each max-pool and uses skip connections between encoder feature maps and the corresponding decoder feature maps to generate a

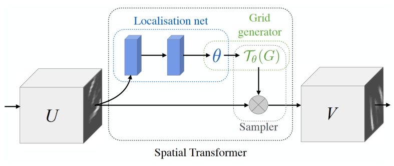

Fig. 2: The spatial transformer module is composed of three parts. The localization network learns to output transformation parameters θ of the affine transformation to be applied to the input feature map, this network can be a fully connected network, a convolutional network or a mix of both. The grid generator uses the generated θ to generate a sampling grid specifying which points from the input feature map must be sampled to produced the transformed output. Finally, the sampler uses bilinear interpolation, or alternately another differentiable interpolation, to sample the input feature map and generate a transformed feature map.

segmentation map for every class in the training set. U-Net originally was used on medical images, and required few training examples (as little as 30 per class), however it is usable with natural images as well [4]. We modify U-Net in our work, as we desired a network employing encoder-decoder architecture, as well as one that can be trained rapidly and using a relatively small dataset.

2.2 Spatial Transformer Networks

All of the segmentation approaches discussed in the previous section share a common trait: they operate on predefined, regularly shaped patches. Whether they employ an encoder-decoder framework, or atrous convolutions and transpose convolutions, or region of interest alignment, these networks operate on square patches of the input images. Frequently, researchers use data augmentations on the training set to teach these networks spatial invariance, which range from random crops, rotations, even deformed grids [6], yet the networks themselves are never taught to understand these spatial invariances. This is what makes Spatial Transformer Networks (STNs) so timely and useful [3], as they offer the key to networks gaining spatial understanding through several of their properties. Most importantly they are differentiable, this allows STNs to be trained with ordinary backpropagation, just like any other part of the network. Furthermore, they can be placed in one or more streams of any existing network, and simply treated as a black-box of sorts. Their main property is their ability to learn how spatially to transform the activations of the previous feature layer, and output a transformed feature tensor in a manner that boosts network performance, or at worst. keeps it constant by virtue of being capable of learning the identity mapping. The architecture of the STN module is presented in figure 2.

STNs have been applied to various classification tasks, and proved that they do improve network performance when included [3]. However, they have only been applied to semantic segmentation in one other work: Dense Transformer Networks (DTNs) [4]. The authors of [4] insert STNs into an encoder-decoder architecture, as well and show an improvement in performance of the modified network on the PASCAL 2012 dataset. They prove the differentiability of the inverse of a spatial transformer sampler, a spatial decoder sampler, which allows the use of STNs for convolution as this decoder now transforms feature maps that have passed through an STN back into their previous space. This can be done by sharing the transformation parameters θ between the STN and the STN decoder. In their work, the authors of [4] provide only basic experimentation results using DTNs for segmentation, only including the results of a single architechture with an STN module at the deepest layer only. We build on the work of [4] by utilizing their STN decoder to experiment with different positions of the STN-STN decoder pairs in the U-Net network,with different network depths, and their effect on the final segmentations.

3 Network Architecture

Our work aims to prove the positive effect STNs can have on network performance in the case of semantic segmentation. Our baseline network is U-Net [6]. We modify the network architecture to include spatial transformers and their corresponding decoders, an example of our modified networks is shown in figure 3. For this work we limit our experimentation to one STN module inside of the network, however, multiple STN modules can also be used. We place an STN module at different depths of the encoder portion of the U-Net, as explained in table 1. We place the corresponding STN decoder at the appropriate layer of the decoder portion of the U-Net, allowing the network to transform the intermediate decoded feature maps back to the space of the input image to get a non-transformed semantic segmentation.

Table 1: A sample modified U-Net network of depth 6, with an STN at depth 4 (layer Down 3 and Up 3), this is shown in figure 3. Convolutions (denoted by conv) are followed by batch normalization. Single asterisks denote the connection via a skip layer between convolution and transpose convolution(tconv) layers in corresponding down and up layers, these are present in the original U-Net. Double asterisks denote the skip layer connections between the spatial transformer and decoder which share the information on the transformation θ. A similar scheme is employed in all of our experiments, while we vary U-Net depth, and STN/STN decoder location.

Part			
Layer	Layer Name	Operations	Output Size
0	Input	-	256x256x3
1	Down 0	c1, c2*, pool	128x128x64
2	Down 1	c1, c2*, pool	64x64x128
3	Down 2	c1, c2*, pool	32x32x256
4	Down 3	c1, Spatial Transformer**, c2*, pool	16x16x512
5	Down 4	c1, c2*, pool	8x8x1024
6	Bottom	c1, c2	8x8x1024
7	Up 4	tc1, concatenate*, tc2, tc3	16x16x512
8	Up 3	tc1, concatenate*, tc2, Spatial Decoder**, tc3	32x32x256
9	Up 2	tc1, concatenate*, tc2, tc3	64x64x128
10	Up 1	tc1, concatenate*, tc2, tc3	128x128x64
11	Up 0	tc1, concatenate*, tc2, tc3	256x256x2
12	Output	-	256x256x2

4 Experiments

We use tensorflow for our experimentation. We utilize and modify the STN implementation of [7], the U-Net implementation of [6], and the DTN implementation of [4] to generate our final framework. Our experiments are run on a Microsoft Azure virtual machine with one half of a Tesla K-80 GPU.

We experiment with varying the depth of the U-Net, along with placing STN modules in different locations of the U-Net, or the U-Net without STN modules, for our baseline measure. We experiment with a U-Net of depths 4, 5 and 6. We place STNs and their corresponding decoders at the second layer of the network, and onwards till the deepest layer. We were unable to experiment with STNs after the first layer due to memory constraints (as the feature map size would still be large at 128x128x64). We trained our network for 4000 epochs with a batch size of 10 images, and evaluated the networks on the test set at epochs 3000, 3500, and 4000 and chose the best performing of them to represent the class of networks.

Our experiments are done using the KITTI road dataset [2]. The dataset is composed of 289 labelled images of which we used 150 for training, 79 for validation, and 60 for testing. The metrics used for evaluation are precision, recall, false-positive rate (FPR), and F1-measure. Precision reflects how much of the proposed road segmentation is truly road, recall measures how much of the overall road in the image was detected, the false-positive rate reflects

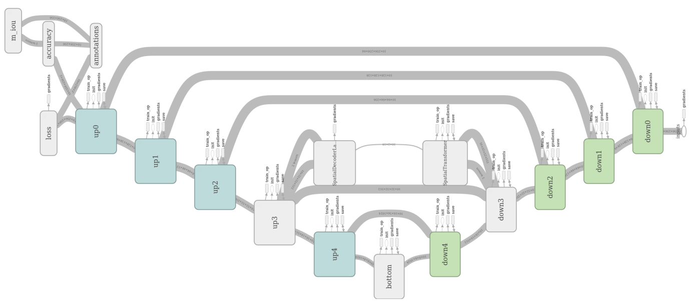

Fig. 3: Our modified U-Net architecture (as a tensorflow graph) with a depth of 5 + 1 (6) encoder layers, and their corresponding decoder layers. Note that in the above figure, the data flows from left to right. We place an STN in the path of the encoder, while the STN decoders are placed in the decoder paths. In the above example, the STN is at the 4th 'layer' (down3), note that each encoder layer is formed of two convolutions (with normalization) followed by a pooling layer. The transformation via STN happens in between the first and second convolutions. The encoded feature maps are also shared via skip connections of the layers that follow them. The decoder layers concatenate the encoded feature maps (via skip connections) with the transpose-convolved feature maps via the regular pipeline. This is followed by two other transpose convolutions, the spatial decoder acts on the feature map before the final convolution of a decoder layer. Our input images are of size 256x256x3. Table 1 describes the encoder layers in more detail.

Table 2: Results, in percentages, on the test subset of the KITTI road dataset. Results are separated based on their base network depth. The top performance metric for each network depth is bolded, while the overall best metrics are italicized and bolded.

Network Architecture					
Depth	STN Position	F1-Score	Precision	Recall	FPR
4	2	92.92	93.11	92.74	1.52
4	3	**96.37**	***96.46***	**96.28**	**0.78**
4	None	95.73	95.46	96.01	1.01
5	2	92.86	92.91	92.81	1.57
5	3	**95.95**	**96.45**	95.46	***0.78***
5	4	95.71	95.25	96.17	1.06
5	None	95.75	94.65	***96.88***	1.22
6	2	93.86	93.69	94.03	1.40
6	3	95.74	95.83	95.66	0.92
6	4	***96.64***	**96.42**	**96.86**	**0.80**
6	5	96.49	96.29	96.69	0.83
6	None	95.55	95.80	95.31	0.93

how much of the background class was improperly classified, and the F-measure is the harmonic mean that measures a trade-off between precision and recall for the foreground classes. The key performance metric we rely on is the F1-measure, as it shows the best all-encompassing result of the network, however, individual metrics may be of interest depending on the application. Numerical results are shown in table 2, while sample analytical results are shown in figure 1.

5 Discussions and Conclusions

5.1 Discussions

Based on the experimental results shown, we verified the performance gains that can be achieved by using STNs inside existing architectures. The baseline U-Net networks of various sizes already perform well on their own, yet the modified U-Nets with added STN modules still manage to outperform the baseline. We can clearly see that the top performing networks for each depth in terms of the F-measure are always STN-modified networks, and this is true for every metric, with the one exception of the recall rate attained by the baseline U-Net of depth 5, which the STN modified networks underperform by a small percentage.

Another noteworthy observation is that STN modules usually yield better performance when placed towards the bottom-half of the encoder and decoder frameworks, while STNs placed in shallow layers did not improve performance. This could be due to those networks requiring more training epochs or data. We use the same STN architecture regardless of layer position, hence it could also be the case that the STN modules themselves require different internal architectures depending on where they are placed in a network.

Finally, shallower STN modified networks performed better than deeper baseline networks, which could be an indication that STN modules can be used to improve computational efficiency and decrease train time of a network by making it smaller. We noted that train time increased when STNs were placed in shallower portions of the network (as a result of them dealing with larger feature maps), however, training time for an STN-modified network did not exceed that of a deeper baseline U-net. However, due to the different depth baselines performing comparably, it could mean that the U-Net networks require more training to reach their full capacity, despite a plateau seen in the validation loss.

5.2 Conclusions and Future Work

Overall, we can conclude that the addition of STNs does yield tangible performance gains, and sometimes can allow shallower networks to outperform deeper ones that haven't been modified with an STN. We did not expect the baseline U-Net to perform so well on the test set, and experimenting with a more challenging dataset may further prove the utility of STN modules for segmentation, especially if the baseline network doesn't already perform well on it. While it's clear that STNs provide improved performance, they still cannot be used as blackboxes, and must be incorporated carefully. We noted that in some cases STNs also decreased the performance of a baseline network, which could mean the STN has not converged to an identity transformation. This is an indication that STN location and train time must also be studied carefully.

For our future work, we plan on testing on more challenging datasets to verify the performance gains using STNs in networks used for segmentation. Furthermore the use of multiple STN modules inside a network architecture has never been explored, and it would be interesting to see if further performance gains could be achieved by the use of multiple STN modules of different types and shapes within the same network. We also would like to explore the effect of STNs on other types of segmentation networks, such as ones that utilize atrous convolutions, and regional CNNs.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[2] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[3] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[4] Jun Li, Yongjun Chen, Lei Cai, Ian Davidson, and Shuiwang Ji. Dense transformer networks. *arXiv preprint arXiv:1705.08881*, 2017.

[5] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[7] Kevin Zakka. Tensorflow implementation of spatial transformer networks. `https://github.com/kevinzakka/spatial_transformer_network`, 2017.