Implementing Fully Convolutional Networks for Semantic Segmentation

Based on the paper

Shelhamer, Long, and Darrell - "Fully Convolutional Networks for Semantic Segmentation" (2016)

Student Practical at Heidelberg University Computer Vision Group, SS 16

Supervisor: Artsiom Sanakoyeu

Simon Guist

1. October 2016

Abstract

In their paper "Fully Convolutional Networks for Semantic Segmentation" Shelhamer et al. show that Convolutional Neural Networks trained end-to-end for pixelwise prediction (Fully Convolutional Networks) can lead to improved accuracy in multiple segmentation tasks. We evaluate their pretrained models on the PASCAL Dataset. We also use their training procedure to train models for the Cityscapes Dataset for Semantic Urban Scene Understanding and compare our results to the benchmark results published by the authors of the Cityscapes Dataset.

Introduction

Convolutional Neural Networks (CNNs) are biologically inspired artificial networks with a wide range of applications, in particular in the field Computer Vision and Object Recognition.
Shelhamer et al. show that standard Convolutional Networks for Object Recognition can be reinterpreted as “Fully Convolutional” networks (FCNs) and be used for semantic segmentation [1]. These Networks can take input of arbitrary size and produce output of the same size.

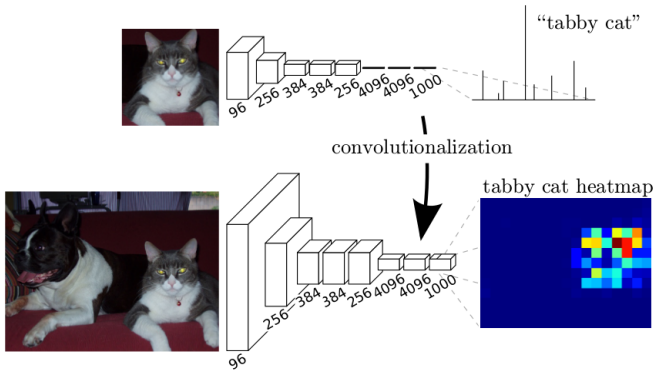

Figure 1: Transforming fully connected layers into convolution layers that can output a heatmap. [1]

Typical CNNs for recognition take input of fixed size and produce non-spacial output, having fully connected layers that throw away all spacial information. The fully-connected layers can easily be interpreted as convolutions with kernels that cover their entire input region. Figure 1 shows how to transform a standard CNN into a network that produces coarse output maps that can be used for semantic segmentation. Dense outputs are thus obtained by stitching together coarse outputs from the input.

Architecture

For their further study, the authors compared several convolutional networks for classification and choose the VGG 16-layer net [2] to transform it into a fully connected network.
All fully connected layers of the VGG net are converted to convolutions. The final classification layer is removed and instead 1x1 convolutions are added for each class at each coarse output location. Also, a deconvolution layer is added to bilinearly upsample to pixel-dense outputs. This

gives the first fully convolutional classifier, that archievs good results on standard metrics, but still has a fairly coarse output.

A common challenge with Image Segmentation is the fact that global information resolves “what” whereas local information shows “where”. Convolutional Neural Networks are build in a way to go from local to increasingly global information. In order to preserve some of the local information in the coarse output layers, the authors introduce skip layers. Skip layers combine the final prediction layer with earlier layers, that have a finer stride and more local information. By doing that they transform the network from a line topology to a directed acyclic graph (DAG). Figure 2 shows how the transformation is done.

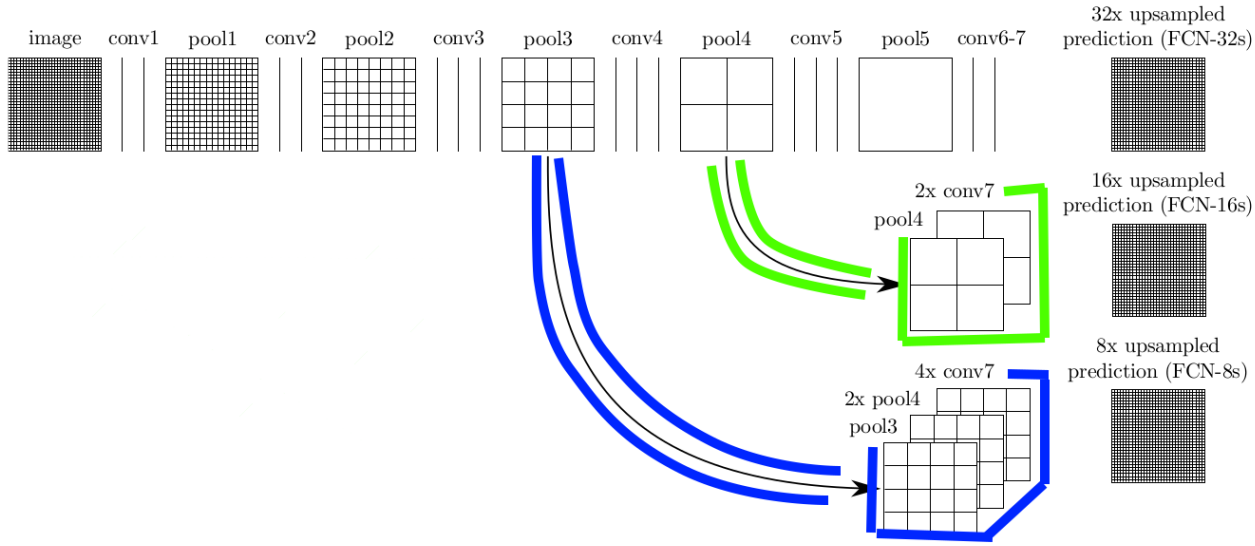

Figure 2: Combining coarse and fine information. The figure shows the general network architecture of FCN-32s, FCN-16s and FCN-8s. Pooling and prediction layers are represented by grids that show relative spatial coarseness and intermediate layers are represented by vertical lines. The layers additionally added to FCN-32s to obtain FCN16 are highlighted in green, the layers additionally added to FCN-16s to obtain FCN-8s are highlighted in blue. [1]

They first divide the output stride in half and fuse information from a 16 pixel stride layer. They combine predictions from pool4 with upsampled predictions computed on top of conv7 and sum them up. Then the stride 16 predictions are upsampled to image size. The resulting network is called FCN-16s. The same process is done for a 8 pixel stride layer, combining skip layers and the predictions from conv7, which they call FCN-8s. The original network without skip layers is subsequently called FCN-32s. Both FCN-16s and FCN-8s are also learned end-to-end and initialized with the last coarser net (FCN-32s and FCN-16s respectively).

The authors evaluate their FCN on semantic segmentation and scene parsing, exploring PASCAL VOC, NYUDv2, and SIFT Flow and achieve results that exceed the state-of-the-art in semantic segmentation.

All of the models are trained and tested in Caffe [3] and their code, as well as the Caffe Models are publicly available on GitHub (https://fcn.berkeleyvision.org).

In the following we are going to further evaluate the FCNs published by Shelhamer et al. For a more detailed discussion of FCNs refer to the paper [1].

Evaluating Pretrained Models on PASCAL VOC 2011

The goal of the PASCAL VOC 2011 Challange [5] is to recognize objects from 20 different categories (e.g. person, dog, bus, sofa...) in realistic scenes. One of the three main competitions is the segmentation competition, in which the intention is to generate pixelwiese semantic segmentation with the class of the object visible or "background" otherwise.
In order to get familiar with the FCN models, we started by evaluating the public available pre-trained Caffe models also using and adapting code for testing provided by the authors. In Table 1 we report the five metrics net loss, overall accuracy, mean accuracy, mean pixel intersection over union and frequency weighted intersection over union. All metrics are evaluated on the validation set with full resolution images. Mean IU is also reported by the authors on GitHub and is consistent with our evaluation.

	Loss	Overall acc.	Mean acc.	Mean IU	Freq. Weight. IU
	Average loss over one image	$\frac{\sum_i n_{ii}}{\sum_i t_i}$	$(\frac{1}{n_{cl}}) \sum_i n_{ii}/t$	$\frac{\sum_i (\frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}})}{n_{cl}}$	$\frac{\sum_i (\frac{t_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}})}{(\sum_k t_k)}$
FCN-32s PASCAL	59390	90.48	76.47	63.62	83.45
FCN-16s PASCAL	59110	90.99	78.03	64.97	84.25
FCN-8s PASCAL	58200	91.22	77.57	65.49	84.54

Table 1: Evaluating Pretrained Models on PASCAL VOC 2011

n_{ij} – number of pixels of class i predicted to belong to class j
n_{cl} – number of different classes
$t_i = \sum_j n_{ij}$ – total number of pixels of class i

Training for the Cityscapes Dataset

The Cityscapes Dataset [6] is a dataset that focuses on semantic understanding of urban street scenes. It contains among other data, high-quality pixel-level annotations of 5000 frames from 50 cities with 30 different classes. The annotations are divided into three sets: training (2975 images), validation (500 images) and test (1525 images). Only the training and the validation set are public. Results from the test set are posted on the cityscapes website (http://cityscapes-dataset.com/).

The authors of the dataset already trained FCN models [7]. Their models are not public, so we decided to train our own models.
They use (except the learning rate) the training parameters described by Shelhamer et al. for PASCAL-Context. We also use these training parameters.
One problem that arose at the beginning of the training was that training with full images (2048x1024 pixels) exceeds the largest GPU memory available. Table 2 shows the Memory usage during training and evaluation of the FCN models for the Cityscapes Dataset.
A linear model with respect to the number of pixels seems to be a good fit as an approximation for the GPU Memory Usage. Applying linear regression (Mem. Usg. MB ~ 2780 + 0.0135 * Num. Pixels) leads to an estimated 28300 MB of GPU Memory needed for full resolution images. The largest GPU that was available was a GeForce GTX TITAN X with 12GB of GPU memory.

	Downscaling factor	GPU Memory Usage Training and Evaluation combined [MB]	GPU Memory Usage Training [MB]	GPU Memory Usage Evaluation [MB]
FCN-32s	x6	5012	3467	2195
FCN-16s	x6	5034	3479	2198
	x4	7212	4659	3201
	x2	x	9856	7627
	x1 – images cut half	x	x (Estim. 16280)	x
	x1	x	x (Estim. 28300)	x

Table 2: GPU Memory Usage while Training for the Cityscapes Dataset

Subsequently we decided to go with 2 times downscaled images (1024x512 pixels instead of 2048x1024). Cordts et al. [7] used half cut images but also did a test with 2 times downscaled images.
Another idea to overcome this limitation was to train on Amazon Web Services (https://aws.amazon.com/) using Bitfusion Boost Ubuntu 14 Caffe from the AWS Marketplace (there is free student credit for AWS). Using Bitfusion Boost it is possible to combine multiple AWS instances into a single virtual instance. Combining two g2.8xlarge instances would, for example, create a virtual machine with 8 graphics cards and a total GPU memory of 32 GB, enough for the estimated amount needed to train on full images. Unfortunately Caffe supports multi-GPU training so far only for the C++ interface, but not for Python Layers (a Python Layers is part of the FCN model). So Instead of training on the GPU, some CPU training was done on an AWS machine featuring 36 Intel Haswell processors and using a Caffe Version optimized for Intel Processors [4].

The CPU training is about 50 slower than GPU training on the GeForce GTX TITAN X with downscaled images or about 12.5 times slower, adjusted for the four times larger images.
We train on the training set and report on the validation set. Like Shelhamer et al. [1] we initialize FCN-32s with the VGG 16-layer net weights [2] and FCN-8s and FCN-16s with weights from previous stages. For the CPU training, we initialize with FCN-8s from downscaled images.
The mean BGR pixel for the model is calculated only on the training set.

Results

With images downscaled by factor two, we did 550000 training iterations (one iteration corresponds to forward- and backpropagation of one image). Figure 2 shows how the loss decreases, there are visibly faster drops at about 350000 iterations and at about 450000 iterations, corresponding to the change of network architecture from FCN32S to FCN16S and from FCN16S to FCN8S.

Figure 2: Training with downscaled images: Loss on the training set (blue, averaged over 500 random images) and on the validation set (red).

Table 3 shows Mean IU for the models. For the final FCN8 model we archive an overall accuracy of 92.5 %, mean accuracy of 71.2%, mean IU of 60.8% and freq. weighted IU of 86.9%.
Cordts et al. report 61.9 Mean IU on the test set for FCN-8 being trained with 2 times downscaled images on the training and validation set and evaluated on the test set [7]. Our result of 60.8 Mean IU on the validation set and 59.1 on the test set (for FCN-8 being trained only on the training set, thus using fewer training data) seems comparable. Cordts et al. also did training with FCN only on the 500 images of the validation set (with full resolution images) and observe a 7 point drop in Mean IU, compared to training on validation and training set (4475 images in total).

As for the difference between results on the test set and the validation set, this is probably random variation, because we didn't do training on the validation set. Looking at the IoU for each class instead of Mean IU, the difference between results on test and validation set also seems to be less significant: We observe better results in 9 classes on the validation set but better results in 10 classes on the test set.

	Downscaling factor	Number Iterations	Learning Rate	Mean IU validation set – downscaled images (x2)	Mean IU test set - downscaled images (x2)
FCN32S CITYSCAPES-2X	2	350000	2e-12	53.2	
FCN16S CITYSCAPES-2X	2	50000	2e-13	59.0	
		50000	2e-14		
FCN8S CITYSCAPES-2X	2	50000	2e-15	60.8	59.1
		50000	2e-16		

Table 3: Results training FCN for the Cityscapes Dataset with downscaled images

Because we use fully convolution networks we can also use full resolution images instead of upscaling the result after forwarding the downscaled image through the network. Doing this with the final FCN8 layer we get mean IU of 54.0 on the validation set. (At this point we tried training on the CPU with full images, but saw only moderately fast improvements, and archived mean IU 56.7 on the test set after 7200 iterations).

This indicates that the network can be used for images of different sizes, but it still shows significantly better performance with images the size the network was originally trained for.

Conclusion

Converting standard classification nets to fully convolutional networks is a relatively simple, but powerful approach.

We use fully-convolutional networks for semantic segmentation, both to evaluate pretrained models and to train our own models for the cityscapes dataset. We look at the problem of limited GPU memory and discuss possible solutions: downscaling, cutting images in half, multi-GPU training and training on CPUs. We archive comparable results on the cityscapes dataset.

The practical helped to improve Python and Linux skills and to get familiar with the caffe framework.

References

[1] Shelhamer, Evan, Jonathon Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." PAMI, 2016.

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[3] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014, URL https://github.com/BVLC/caffe

[4] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014, URL https://github.com/intel/caffe

[5] Everingham, M., et al. "The PASCAL Visual Object Classes Challenge 2011 (VOC 2011)" URL http://www. pascal-network.org/challenges/VOC/voc2011/workshop/index.Html

[6] Cordts, Marius, et al. "The cityscapes dataset." CVPR Workshop on The Future of Datasets in Vision. 2015, URL https://www.cityscapes-dataset.com

[7] Cordts, Marius, et al. "The cityscapes dataset for semantic urban scene understanding." arXiv preprint arXiv:1604.01685 (2016).