SegNet Evaluation

Algorithm Brief Summary

- Many neural net that segment to pixel level have similar encoder -> decoder (deconvolution) steps, but SegNet have much more efficient (in speed and RAM) decoder step with little tradeoff in accuracy.
- SegNet's decoder uses data from encoder in a clever way. Other algorithm, decoder "learn" how to upsampling but it is slow and difficult to train. SegNet remembers just pointer data ("indices") that encoder produce as an input for decoder to upsampling, not the full output like some other networks.
- This means SegNet is more RAM friendly when use. It is upto 11 times more efficient than others.
- SegNet can be trained end-to-end. Some algorithm contains separate parts that you have to train individually, like the one that throw neural net output into separate CRF algorithm.
- SegNet removed fully connected final layers from VGG16 (very deep conv. net for image recognition) to reduce parameters, enable better end to end training. (VGG16 = 13 conv. + 3 fully connected layers, layer count down from 134M to 14.7M)
- "Competitive" performance on standard tests, but significantly faster.
- FCN model (one of the early pixel-wise recognitor) has weakness in complex training and decoder is too simple.
- Max-pooling can adds translation invariance, but too much decrease edge details.
- But memory during inference is low, we can't remember all the encoder outputs. We remember only (max) indices. (FCN did not remeber, but uses all the encoder feature maps)
- "SegNet-**Basic**" is a simpler model that the author made. The different is it has 4 encoders and decoders instead of 13 encoders and decoders.
- The author found that larger decoder is better. And using just indices from encoder as an input for decoder is a better way to upsampling.

Advantages

Better in exploiting co-occurence of labels or "scene understanding" than competitors. Other algorithm might be better in competitive scene recognition contest, but real road scene have more co-occurence than contests. (For examples, car must be around the road label)

Real-time?

SegNet's quality is indeed very good. But the inference time, even though the improvement is significant, the paper said nothing about real-time performance. (They only say they are improving the network **towards** practical application) So it is questionable if Qoncept could **directly** use this in production.

SegNet Performance

Training

Macbook's CPU is not suitable to train SegNet at all. You definitely needs GPU to train it.
It is so slow that I cannot see a single update in half an hour.

Inference

The model used in this inference test is the same one that they use on their web demo : http://mi.eng.cam.ac.uk/projects/segnet/

It was trained with data specifically for road scene and classify to **12 classes.** (The standard CamVid training data (http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/) have 32 classes.)

SegNet-Basic's performance on Titan X GPU

They have this data in their paper. Real-time speed is around 33ms, and SegNet-Basic can achieve 52.6 ms which is about half-real time. Note that this is on the very best GPU in the market, the Nvidia Titan and still this model is the basic one, not the same as in their web demo.

				Median frequency balancing						Natural frequency balancing					
		Encoder	Infer	Test			Train			Test			Train		
Variant	Params (M)	storage (MB)	time (ms)	G	C	I/U	G	C	I/U	G	C	I/U	G	C	I/U
Fixed upsampling															
Bilinear-Interpolation	0.625	0	24.2	77.9	61.1	43.3	89.1	90.2	82.7	82.7	52.5	43.8	93.5	74.1	59.9
Upsampling using max-pooling indices															
SegNet-Basic	1.425	1x	52.6	82.7	62.0	47.7	94.7	96. 2	92.7	84.0	54.6	46.3	96.1	83.9	73.3
SegNet-Basic-EncoderAddition	1.425	64x	53.0	83.4	**63.6**	48.5	94.3	95.8	92.0	**84.2**	56.5	**47.7**	95.3	80.9	68.9
SegNet-Basic-SingleChannelDecoder	0.625	1x	33.1	81.2	60.7	46.1	93.2	94.8	90.3	83.5	53.9	45.2	92.6	68.4	52.8
Learning to upsample (bilinear initialisation)															
FCN-Basic	0.65	11x	24.2	81.7	62.4	47.3	92.8	93.6	88.1	83.9	55.6	45.0	92.0	66.8	50.7
FCN-Basic-NoAddition	0.65	n/a	23.8	80.5	58.6	44.1	92.5	93.0	87.2	82.3	53.9	44.2	93.1	72.8	57.6
FCN-Basic-NoDimReduction	1.625	64x	44.8	**84.1**	63.4	**50.1**	**95.1**	**96.5**	**93.2**	83.5	**57.3**	47.0	**97.2**	**91.7**	**84.8**
FCN-Basic-NoAddition-NoDimReduction	1.625	0	43.9	80.5	61.6	45.9	92.5	94.6	89.9	83.7	54.8	45.5	95.0	80.2	67.8

SegNet (regular)'s performance on my Macbook Pro (CPU only)

Network was trained with 360x480 pixels image so the input for inference must match this. Any input will be resized to that size. This model is the same as their web demo.

Average inference time : 10 seconds

I made a video of segmentation in action. I tried 2 size of video to confirm some assumption that there is no difference in segmentation time.

Input 1280x720 : https://www.youtube.com/watch?v=HsYaGp0KfFo
Input 480x270 : https://www.youtube.com/watch?v=dCajkBePRj0

Definitely CPU inference is not good for real time application in car.

If I have Titan GPU I could evaluate it and compare with SegNet-Basic, unfortunately Macbook Pro cannot uses any graphic cards.

My thoughts on improvement

Currently SegNet is not real time even with very high-end GPGPU like Titan that this paper used (Only the newest GTX1080/70 is better than this) plus CUDNN acceleration. (I am not sure if you can have all of this in a car?) Ways to improve this towards usage in real car I think includes :

- Reduce class that network can segment. This will greatly reduce parameters in all layers. (requires retraining)
- Reduce the size of input image. This will impact the final quality of segmentation.
- Increase the specs of GPU used. With newest

- Fall back to simpler but inaccurate algorithm like FCN. (They have multiple varients like FCN-8, FCN-16) FCN have **near real-time** performance, 50ms on 500x500 input with NVIDIA Tesla K40c. (Real time is around 33ms)
- Preprocess input image with only part that we need. I don't know the real intention of Qoncept's system, but for example, if you just want to **find people on the street,** then preprocessing input and crop the input to likely area that have a person before feeding to neural network is good. (I have taken this technique idea from this work http://jacobsschool.ucsd.edu/news/news_releases/release.sfe?id=1883) But if you want to label the entire scene this won't help.

Note on FCN, I have tried FCN in my thesis before and compared with newer algorithm the result from FCN is very blocky. (It is like the decoder step just use straight line to connect low resolution segmentation map, you can clearly see the sharp boundary.) Below is the comparison.

FCN : https://www.youtube.com/watch?v=80oZOOeQEHs
Better NN (I used CRF-RNN here, but SegNet is more efficient than CRF-RNN) : https://www.youtube.com/watch?v=9X9UdHGW1ps please look at second picture on left column.

Miscellenous Technical Note

- Caffe allows you to use one from 3 kind of BLAS (basid linear algebra program) that is OpenBLAS, ATLAS and Intel MKL.
- Intel MKL takes 1.5GB and very long install time. And it only works with Intel CPU.
- You cannot install ATLAS with make- j4 flag. With normal make it succeeds, but took 8 hours for compiling and testing. (If you want to try my pre-built lib file and doesn't want to wait, I have uploaded them here https://github.com/5argon/AtlasOSXBuild but I am not sure if it will works with different Macbook model or not)
- MKL and ATLAS default directory failed, you must manually input manually to Makefile.config
- Using CPU mode (Macbook does not have GPU), by default the training does not work. It results in error like this which probably is a bug :

  F0615 23:12:33.120666 2056376320 upsample_layer.cpp:127] upsample top index 0 out of range - check scale settings match input pooling layer's downsample setup

  However if you comment out the code that produce this fatal warning in Caffe source code and build Caffe again the training can continue as expeted.