Notes - Fully Convolutional Network (FCN)

Wednesday, December 12, 2018 10:32 PM

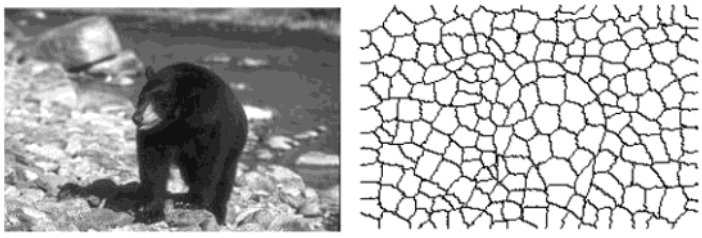
What is FCN?
It is a machine learning algorithm used for the computer vision application especially semantic segmentation.

What is the purpose?
The purpose of FCN is to segment the images and assign pixels belonging to an object to a specific class label.

What is the advantage of FCN?
When compared to the approaches that were proposed prior to FCN, FCN exceeds the performance of the state of the art algorithm and can be trained end to end.

What is the common pipeline of previous approaches?
Previous approaches have used pre-processing and post-processing. Pre-processing includes superpixels, proposals and post-processing includes refinement by random-fields or local classifiers.

What is superpixel? (One of the techniques used by previous approach)
Superpixel is a perceptually meaningful entity that can be obtained from a low-level grouping process. Normalized cuts is a classical region segmentation algorithm which used spectral clustering to exploit pairwise brightness, color and texture affinities between pixels. This algorithm can be applied to oversegment images to obtain superpixels. Following example shows such a superpixel map with the number of clusters 200.



Original Image Superpixel map with k=200

How Fully Convolutional network uses the CNN architecture for semantic segmentation?
Deep feature hierarchies created by Convolutional Neural Network encode location and semantics in a nonlinear local-to-global pyramid i.e. the deep layers (feature layers near the end) in the CNN can produce the coarse semantic information and the shallow layers (initial feature layers) provide the fine, appearance related information. These two layers are combined using the skip architecture.

What are the previous approaches related to FCN?
The approach used by FCN combines the successes of deep nets for image classification and transfer learning.
<u>Fully convolution approach:</u>
This approach is initially used for the extension of convnets to arbitrary sized inputs, where the classic LeNet is used to recognize the string of digits. Here the input is considered as one dimensional strings and Viterbi decoding is used to obtain the results. The approach is also used in a lot of coarse multi-class segmentation in medical applications, sliding window detection, image restoration etc.,
<u>Dense predictions with convnets:</u>
Several approaches have used the convnets to do the semantic segmentation. Common elements used by them include the following:

1. Having small models and restricting the capacity and receptive fields
2. Patch wise training
3. Post-processing using the super-pixel projection, random field regularization, filtering or local classification
4. Input shifting and output interlacing for dense output
5. Multi-scale pyramid processing
6. Saturating tanh nonlinearities
7. Ensembles

But the FCN does not use any of these machineries. FCN uses patch wise training and shift-and-stitch dense output, in-network up-sampling.
FCN uses fusion architecture that fuses the features across layers to define a nonlinear local-to-global representation which is then tuned end to end.

How does the convolution layer can be described mathematically?
The basic components of convnets are convolution, pooling and activation functions. They operate on local input regions and depend only on relative spatial coordinates.
x_{ij} for the data vector at location (i; j) in a particular layer, and y_{ij} for the following layer can be written using the following expression

$$\mathbf{y}_{ij} = f_{ks}\left(\{\mathbf{x}_{si+\delta i, sj+\delta j}\}_{0 \le \delta i, \delta j \le k}\right)$$

Where k is the kernel size, s is the stride or subsampling factor and f_{ks} determine the layer type

- Matrix multiplication for convolution or average pooling
- Spatial max for max pooling
- Elementwise nonlinearity for an activation function.

How the FCN differs from a common deep network?
A general deep net computes a general nonlinear function, but FCN with only layers of convolutional form computes a nonlinear filter. So an FCN operates on an input of any size and produces an output of corresponding spatial dimensions.

How does the loss function and corresponding gradient looks in the FCN?
Loss function is a sum over the spatial dimension of the final layer,
$\ell(x; \theta) = \sum_{ij} \ell`(x_{ij}; \theta)$ , then the gradient will be the sum over the gradients of each of its spatial components. Then the stochastic gradient descent on $\ell$ computed on whole images will be the same as stochastic gradient descent on $\ell`$, taking all of the final layer receptive fields as a minibatch.

How to adapt classifiers for the dense prediction?
Recognition nets like LeNet, AlexNet and its successors take fixed size inputs and produce non-spatial outputs. Fully connected layers of the above models have fixed dimensions and throw away spatial coordinates. These fully connected layers can be viewed as convolutions with kernels that cover their entire input regions which then converts them into fully convolutional networks and output classification maps.
The spatial output maps of this convolutional models make them a natural choice for dense problems like semantic segmentation. Ground truth is available at every output cell, so, both the forward and backward passes are straight forward.
While converting these fully connected layers to fully convolution ones, the output dimensions are reduced by subsampling. This coarsens the output by reducing it from the size of the input by a factor equal to the pixel stride of the receptive fields of the output units.

How the dense predictions can be obtained from the coarse outputs?

Shift and stitch:

Dense predictions can be obtained by stitching together the output from the shifted versions of the input.

If the output is down sampled by a factor of f, shift the input x pixels to the right and y pixels down, once for every (x, y) s.t. $0 \leq x; y < f$. Process each of these f^2 inputs, and interlace the outputs so that the predictions correspond to the pixels at the centers of their receptive fields.

Filter Rarefaction:

Performing shift and stitch increases the cost by a factor of f^2. in order to avoid this and also produce identical results a-trous algorithms can be used.

For example:

Consider a layer (convolution or pooling) with input strides, and a subsequent convolution layer with filter weights f_{ij} (eliding the irrelevant feature dimensions). Setting the lower layer's input stride to 1 upsamples its output by a factor of s. However, convolving the original filter with the upsampled output does not produce the same result as shift-and-stitch, because the original filter only sees a reduced portion of its (now upsampled) input. To reproduce the trick, rarefy the filter by enlarging it as

$$f'_{ij} = \begin{cases} f_{i/s,j/s} & \text{if } s \text{ divides both } i \text{ and } j; \\ 0 & \text{otherwise,} \end{cases}$$

This filter enlargement layer-by-layer has to be repeated until all the subsampling is removed.

Drawbacks of the above methods:

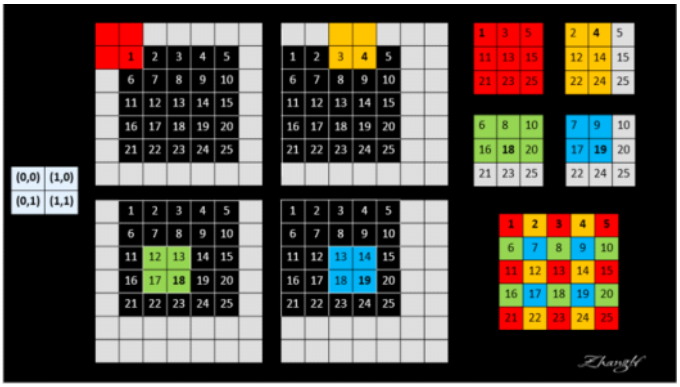
Decreasing subsampling in a net is a trade-off: the filter can see the finer information, but because of the smaller receptive fields it will take longer to compute. But in the shift and stick, the output is denser without decreasing the receptive field sizes of filters, but filters cannot observe finer information.

Hence FCN has proposed and used learning through upsampling which is more effective and efficient, especially when combined with skip layer fusion.

Additional Explanation for Shift and stitch:

In FCN, the final output you get (by default without utilizing any tricks for up sampling) is at a lower resolution compared to the input. Assuming you have an input image of 100x100 and you get an output (from the network) of 10x10. Mapping the output directly to the input resolution will look patchy (even with high order interpolation). Now, you take the same input and shift it a bit and get the output and repeat this process multiple times. You end up with a set of output images and a vector of shifts corresponding to each output. These output images with the shift vectors can be utilized (stitch) to get better resolution in the final schematic map.

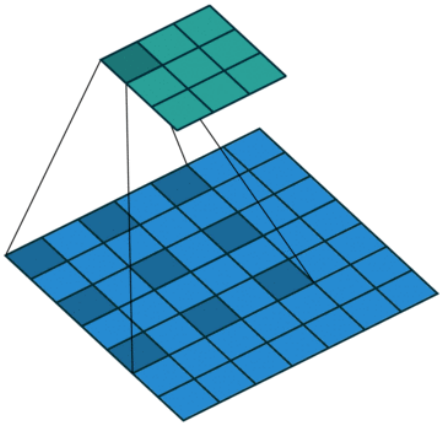
One might think of it as taking multiple (shifted) low resolution images of an object and combining (stitch) them to get a higher resolution image. The following diagram clearly depicts the process of the shift and stitch.



Additional Explanation for Filter Rarefaction / Atrous Convolutions:

Dilated convolutions introduce another parameter to convolutional layers called the dilation rate. This defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters. Imagine taking a 5x5 kernel and deleting every second column and row.

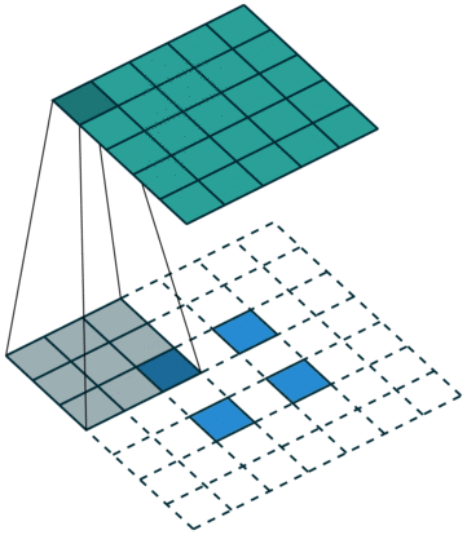
This delivers a wider field of view at the same computational cost. Dilated convolutions are particularly popular in the field of real-time segmentation. Use them if you need a wide field of view and cannot afford multiple convolutions or larger kernels.



What are the other ways to connect coarse output to dense pixel predictions?

Interpolation is another approach. In a simple bilinear interpolation, each output y_{ij} is computed from the nearest four inputs by a linear map and depends only on the relative positions of the input and output cells.

So upsampling with factor f is convolution with a fractional input stride of 1/f . As long as f is integral, backwards convolution sometimes called as deconvolution is a way for upsampling with an output stride of 'f'. This operation simply reverses the forward and backward passes of convolution. It enables the upsampling to be performed in -network for end-to-end learning by backpropagation from the pixelwise loss. The following diagram gives an idea about the transposed convolution.



This deconvolution filter is need not to be having fixed weights, they can be learned. So, a stack of deconvolution layers and activation functions can learn a nonlinear upsampling. This approach seems to be effective for learning dense prediction and hence it is used in FCN.

What is patchwise training and how it is used in FCN?

In patchwise training each image is divided into patches or sub images and each of them will be passed through the network to create the results. Whole image convolutional training is where a whole image is passed in a single pass and the backward pass which computes the gradient and the weights will be updated. Whole image convolutional training can be similar to patchwise training if the patches in a mini-batch is having the sub images produced from a single image itself. We are having a choice to create the mini-batch containing the images from different large images. If the image patches in the mini-batch have significant overlap then it can speed up the training. If the gradients are accumulated over multiple backward passes then batches can include patches from several images. This kind of patchwise training can mitigate the spatial correlation of dense patches and decrease the effect of class imbalance. Other ways to do class balance is to weight the loss and loss sampling. But FCN paper hasn't found any significant improvement provided by the patchwise training. The paper proposes that the whole image training is efficient and effective.

References:

Fully Convolutional Network
Link: https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Superpixel
Link: http://ttic.uchicago.edu/~xren/research/superpixel/