Convolutional Neural Networks

Vladimir Feinberg

July 18, 2017

A convolution is a mathematical operation that operates as layer in an ANN. Content is mostly from Dr. Goodfellow's Deep Learning Book, but also taken from Dr. Hinton's Coursera Class, lecture week 5.

Convolutional neural nets (CNNs) are principally used for computer vision, but they can be viewed as a broader form of parameter sharing that's applicable to many domains. In particular, CNNs find "local patterns" in grid topologies, possibly of 1 dimension (like time series: joint angle data for a robot or audio data) or more than two (such as 3D fMRI imaging). Multiple channels (such as colors, or later feature maps) can just be viewed as separate image maps or new dimensions in the data.

1 Visual Difficulties

Visual difficulties we'd like to avoid include segmentation (teasing apart an object from its background and other object), deformation, lighting, and viewpoint.

2 Convolution

Convolution is a linear operation that replaces a fully-connected layer in a neural network. Thus, whereas a fully-connected layer with a vector input $\mathbf{x}$ and activation f may look like $f(W\mathbf{x}+\mathbf{b})$ for weight W and bias $\mathbf{b}$, a convolutional layer would behave like $f(\mathbf{w}*\mathbf{x})$. In this context, the weight $\mathbf{w}$ is referred to as a kernel, and f is typically a rectifier followed by a pooling operation. It is common for multiple kernels to be applied to the same $\mathbf{x}$, the pooled results of which form the output of the layer as feature maps. The terminology follows similarly when replacing the vectors $\mathbf{x}, \mathbf{w}$ with arbitrary tensors X, W; convolutions are well-defined on them.

Convolution can be viewed as a linear operation. A 1D convolution is multiplication by a circulant matrix, a 2D convolution is multiplication by a doubly-circulant block matrix, and so on (example derivation). Thus, indeed convolution is a form of parameter sharing and sparse interactions. Convolutions are typically implemented as cross-correlations. Even though this loses the commutativity property, these networks are just as expressive since we can just transpose the kernel. For small kernel sizes, convolution can be implemented much more efficiently than with their associated matrix multiplies.

Some nuance is required to deal with channels. A 2D image with width w and height h might have three RGB values at each pixel, so its input is $w \times h \times 3$. A convolutional layer with k kernels applied to a greyscale image might also result in k output feature maps of size $w \times h$. Given a 3D volume of 2D images, with size $w \times h \times d$, what does it mean to apply a 2D convolution with k kernels?

- Usually, this "full convolution" refers to using kd filters total, such that the j-th output layer, is the sum of d independent kernels applied to each of the d 2D inputs. For all $j \in [k]$, this gives our full volume output. Then, if our kernel size is s, the total number of parameters in this layer is s^2kd (assuming no bias is added after convolution - per-kernel bias is very redundant in full convolutions).
- A 1×1 convolution, then, reduces to a matrix multiply from the space of the input channels to that of the output channels (i.e., it creates a linear relationship between the input and output).

- Separable convolutions (Chollet et al 2016 have only d regular kernels for each of the input feature maps, which run a convolution. Then a full convolution with 1×1 kernels (which has kd scalars total) is run. Thus, separable convolutions are much slimmer. More complicated filter connections do exist, but are sparingly used. See this overview and this LeNet analysis.

Modifications or common enhancements to convolution (see Dumoulin and Visin 2016, Fig. 1):

- Padding. Various paddings add zeros around the inputs and result in different output sizes. These dictate the role that partially applied filters at the boundary will play.
- Strides. A stride of greater than one skips applications of the kernel filter, which can be viewed as a form of regularization through downsampling. Along a similar vein, diluted convolutions apply the kernel itself at strides along the input. A transposed form of convolution, deconvolution, reverses this process, upsampling compressed input (Zeiler 2010).
- Detectors. These are the nonlinearities applied after our linear convolutional transform.
- Degrees of parameter sharing. Unshared convolutions learn a new kernel for each application of the kernel to an input feature map, these have no sharing and are called locally connected layers. Tiled convolutions rotate through a small set of kernels as a middle ground proposed by Le et al 2010, but they're usually replaced by having many parallel regular convolutional layers now.

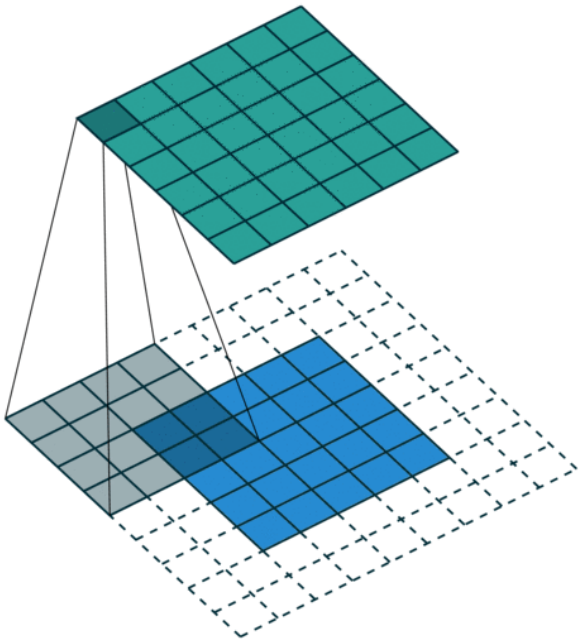

Figure 1: Demonstration of a convolutional layer with padding, no strides, and no dilation retrieved from Theano docs.

3 Pooling

Pooling is typically applied after a convolutional layer/stage. It may be a nonlinear operation which is applied to a local neighborhood of the input, like a convolution. For various types of pooling and when to

use them, see Boureau et al 2010. Max-pools, for instance, encourage invariance between adjacent kernel outputs, since only one of them needs to be activated to activate the associated max pool.

Pools have a specified pool width (determining the size of the neighborhood that is pooled) and stride, which has the same effect as kernel stride.

Pooling typically reduces image width, so in cases where we would like to classify input pixels themselves, one solution is to simply produce an output at a lower resolution than the input (Fig. 2).

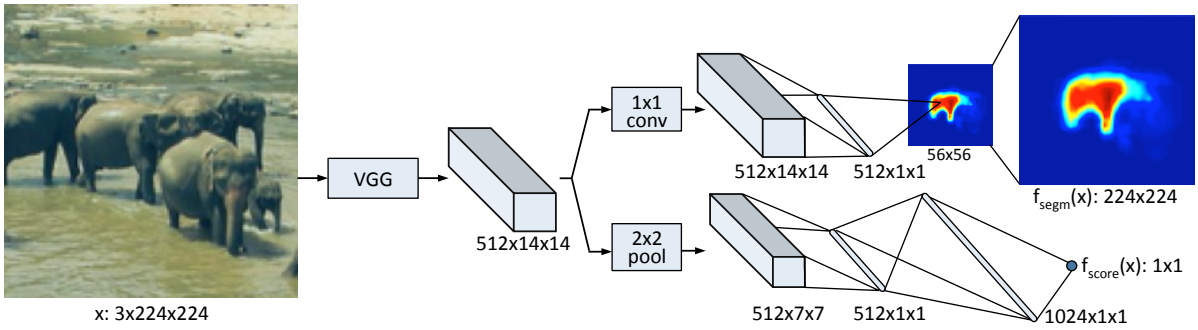

Figure 2: Architecture of segmentation architecture from Fig. 1 of Pinheiro et al 2015)

Pooling isn't always necessary (Springenberg et al 2014), and maxout-like cross-channel pooling is an interesting response (Liu et al 2015).

4 Initialization

4.1 Random Kernel Intialization

Random kernel initialization is surprisingly good, so much so that Coates et al 2011 even states that a network trained only on the last layer can be used as a selection method for CNN architectures.

Saxe et al 2010 finds that architectures with random weight initialization in CNNs can be further improved with unsupervised pretraining. See the initialization document for details.

5 Preprocessing

- Normalize data input channels from integer pixel values $[0, 255]$ to floats on $[0, 1]$.
- Train on many cropped versions of the data and have classifiers vote (can this be re-interpreted as parallel CNNs, and if so how does sharing affect this technique?)
- Denoise with respect to some cannoncial form of the data (rotations, skew). This can enable smaller models but some caution should be used. It is sometimes better to let the models figure out the cannonical form for themselves.
- Global contrast normalization (GCN) standardizes pixel inputs; local contrast normalization (LCN) scopes deviation to a window (helps make edges stand out). GCN reduces pressure on the model to handle different scales of input intensity.
- Random perturbations to input such as color changes or translations or rotations can help augment the dataset for better generalization (LeCun et al 1998b, Krizhevsky et al 2012)

6 Common Architectural Approaches

Szegedy et al 2015 give several general strategies for CNN design:

1. Avoid representation bottlenecks. MLPs should gently decrease in hidden layer size (though representation complexity is more nuanced than just width). Correspondingly, for CNNs, use high-dimensional embeddings as outputs of CNN layers. In other words, the number of convolution kernels should increase to maintain information density (with the advantage being that the kernels will tease out individual features). This doesn't conflict with the first item of advice, since a convolutional layer doesn't have as much representational density as a fully connected one.

2. Convolutions can be preceded by low-dimensional embeddings, since strong local correlation in inputs can be compressed.

3. Width and depth should be balanced: optimal performance to compute is achieved by increasing both in parallel rather than one a lot.

An excellent overview of individual techniques proposed in CNN architecture can be found in a blog post by Adit Deshpand