Neural Networks and Computer Vision

Vladimir Feinberg

July 11, 2017

We review techniques for computer vision with DNNs. Content is mostly from Dr. Goodfellow's [Deep Learning Book](#), but also taken from [Dr. Hinton's Coursera Class](#), lecture week 5.

DNN vision techniques rely critically on convolutional neural nets (CNNs). Note that the techniques here can generally be applied to other inputs with grid topology, 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.

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 Ngiam 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).

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

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.

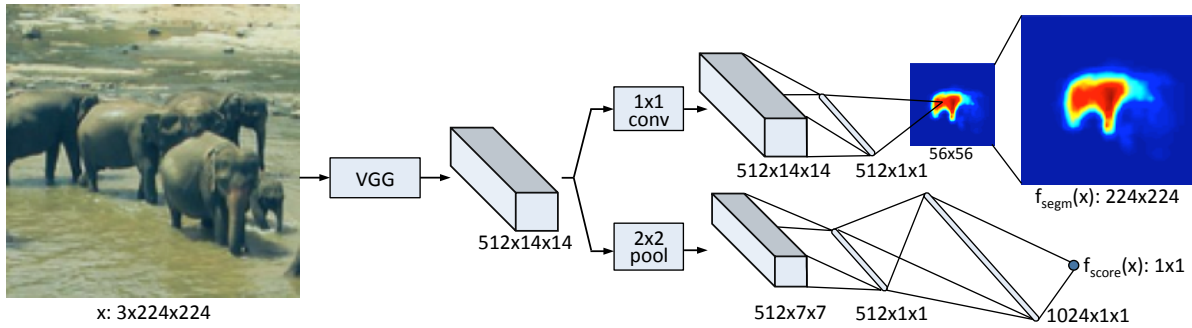Saxe et al 2010 explores unsupervised initialization approaches.

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

5 Common Architectural Families

These change very frequently and it's more useful to just study common families of networks. See this overview post. Missing from that post but selected as important by Dr. Goodfellow is the spatial transducer network.

6 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)