6.036 Spring 2018 : Week 6

March 19, 2018

7 Convolutional Neural Networks

We will discuss here a new neural network architecture that is specifically tailored to images. As a starting point, let's try to solve an image classification problem with a simple feed-forward architecture that we already know about and see what types of problems we run into. If we view a digital image as a matrix of pixels, we can vectorize it, and feed it "as is" to a feed-forward neural network. Thus we designate an input unit to hold the value of each pixel in the image. If the image has millions of pixels, and we use a few hundred hidden units, then the first layer in the feed-forward architecture already has over 100 million parameters. While the number is large, and it seems that we will need lots of labeled images to learn the associated parameters, the main problem is actually in what these weights are supposed to do. For example, it is often useful to first identify small features in the image such as edges, and later combine such features to detect objects etc. But in this architecture, we would have to separately learn the weights for recognizing the same simple feature in each possible location in the image (nothing carries over from one part of the image to another). Another issue is accounting for scale. Features are sometimes small, sometimes large, as are objects in the image. How do we vary the resolution at which we look at the image? Moreover, the objects should eventually be classified correctly regardless of where they appear in the image, i.e., our classifier should be (at least partly) translation invariant. Do we have to separately learn to recognize the same object in each position in the image? The straightforward feed-forward architecture doesn't seem well-suited for this task.

A convolutional neural network (CNN for short) is a feed-forward neural network but it has quite a bit of special structure. It consists of interspersed layers of convolution and pooling operations (which we will outline in more detail below). The convolution layer applies simple local image "filters" across the image, producing new images (known as feature maps) where the "pixels" in the feature map represent how much the simple features were present in corresponding location. You can think of this process as first breaking the image into overlapping little image patches, and applying a linear classifier to each patch. The size of the patches, and how much they overlap, can vary. The pooling layer, on the other hand, abstracts away from the location where the features are, only storing the maximum activation within each local area, capturing "what" is there rather than "where" it is.

7.1 Convolution

To make the ideas a little more concrete, let's assume that our image is 1-dimensional with n pixels, $x_1, \ldots, x_n$, as shown in figure 1 below. The filter applied to the image could be just $[w_1, w_2, \ldots, w_k]^T$, i.e., of size k (an odd number). Of course, the filter size can vary but we will use $k = 3$ for simplicity. Figure below illustrates how the filter is applied to obtain the corresponding feature map (another 1-dimensional image). For example, if stride is set to one, filter size is k, and the added non-linearity is ReLU, then the i^{th} coordinate value of the feature map is obtained as

$$f_i = \max\left\{0,\ \sum_{j=1}^{k} x_{i+j-\lfloor k/2 \rfloor - 1}\, w_j \right\} \tag{1}$$

where $i = 1, \ldots, n$ (the same size as the original image). So, for example, when $k = 3$, then the index $i + j - \lfloor k/2 \rfloor - 1$ runs through $\{i-1, i, i+1\}$ when $j = 1, 2, 3$, i.e., you apply the classifier with weights w_1, w_2, w_3 to a patch in the original image centered at i. We assume that the image is padded with zeros when the index $i + j - \lfloor k/2 \rfloor - 1$ extends beyond the limits. So, for example, we define $x_0 = 0$. Note that if we increase the stride beyond one, the resulting feature map will be of size n/stride.

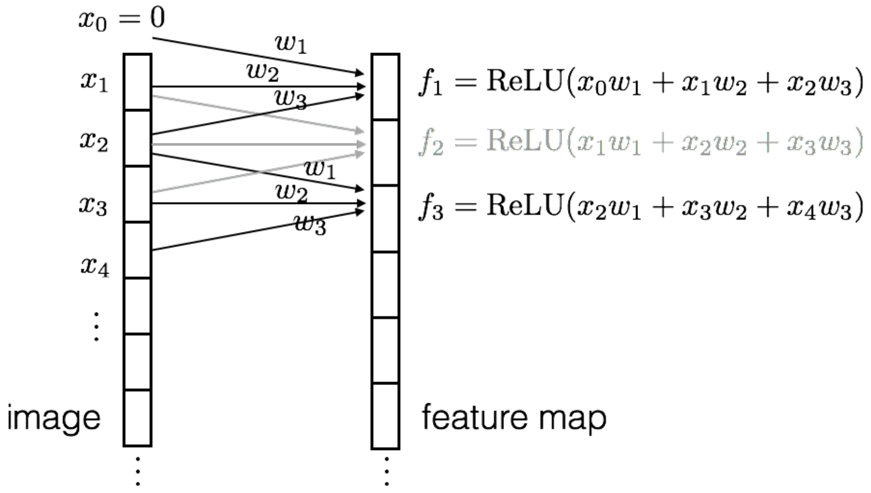

$x_0 = 0$
x_1
x_2
x_3
x_4
w_1
w_2
w_3
$f_1 = \text{ReLU}(x_0w_1 + x_1w_2 + x_2w_3)$
$f_2 = \text{ReLU}(x_1w_1 + x_2w_2 + x_3w_3)$
$f_3 = \text{ReLU}(x_2w_1 + x_3w_2 + x_4w_3)$
image
feature map

Figure 1: A simple one dimensional convolution layer with $k = 3$ and stride one. The image is padded with zeros at the ends when the convolution operation (linear feature detection) would otherwise extend beyond the image boundaries.

The feature map is parameterized just as any feed-forward layer in a neural network. However, if viewed as a feed-forward layer, the weights in this layer are *shared* in the sense that one unit in the feature map layer uses the exact same weights to calculate its net input as any other unit in that layer. Thus the only parameters that need to be estimated are $[w_1, w_2, \ldots, w_k]^T$, i.e., the parameters of the filter. This makes the mapping very compact with number of parameters that is independent of the image size (cf. straightforward feed-forward layer). Nevertheless, the filter can be adjusted to look for different features and the resulting feature map identifies where the chosen feature is active in the image. Of course, we don't just wish to look for a single feature (e.g., horizontal edge). Instead, we typically introduce a number of different feature maps (also called channels) and the parameters in the associated filters are estimated jointly in the overall CNN model. The feature maps may also be introduced across channels.

7.2 Pooling

Another key operation in CNNs is pooling. There are many different types of pooling operations but the simplest one is just max-pooling. This is similar to convolution in terms of the filter size, and how it is applied across the image. However, there are no parameters, and the operation analogous to the weighted average in the convolution is just the maximum. In other words, if the pooling filter size is k with stride one, then

$$f_i = \max_{j\in\{1,\ldots,k\}} x_{i+j-\lfloor k/2\rfloor-1} \tag{2}$$

for $i = 1, \ldots, n$. Max-pooling is typically applied with a relatively small k (can be an even number) but with larger stride (e.g., 2) so as to reduce the resulting feature map size. The figure 2 below illustrates max-pooling with $k = 3$ and stride two. Note that the resulting pooling layer activations retain the highest value within the patch that they consider but no longer who produced it – a step towards translation invariance.

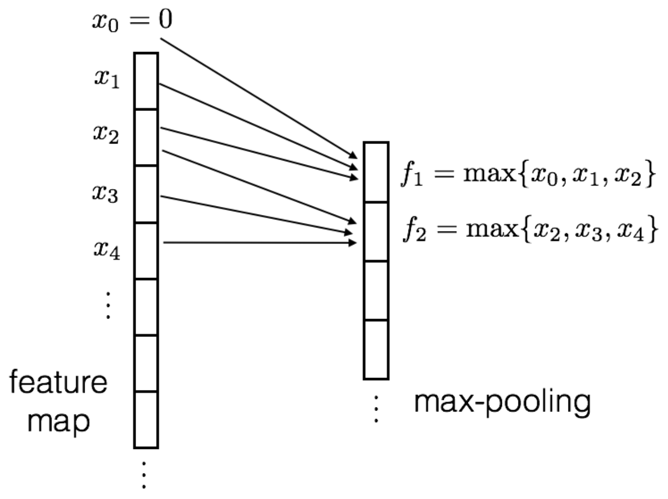
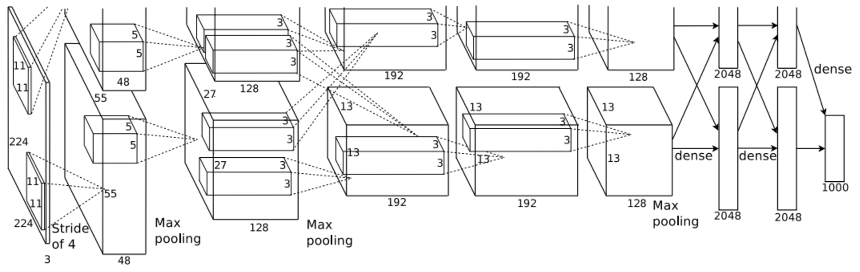


Figure 2: A simple one dimensional max-pooling layer with $k = 3$ and stride two.

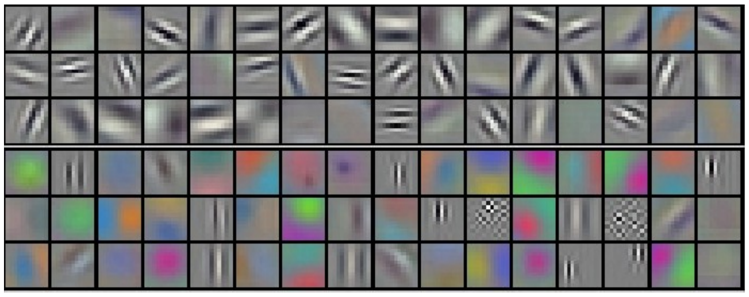
7.3 CNN architecture

There are a large number of possible CNN architectures that combine convolution and pooling operations (among other things). For example, the architecture that won the 2012 imageNet image classification challenge (1.2 million images, 1000 categories), begins by mapping the image to 96 feature maps using 11x11x3 convolution filters. The filters consider 11x11 image patches along with 3 color coordinates for each pixel. So, the original image is a cube (when we add the color coordinates), and each filter is a small cube. The filters involve ReLU non-linearity and stride equal to 4. This is followed by contrast normalization and then 2x2 max pooling. Further layers operate similarly but vary in terms of filter and pooling sizes. Towards the output layer, the model introduces fully connected layers that combine the information from all the channels. The final output layer is a softmax layer with 1000 output units, one unit per category (see the appendix). The model is trained end-to-end, i.e., as a differentiable map from the image all the way to the output units similarly to any feed-forwrd network. A simple schematic is shown below.



(Krizhevsky et al. 2012)

We can visualize what the resulting filters look like in the final trained model. Figure below shows the first 96 filters 11x11x3 as color images (color image patches that would maximize the ReLU filter response).



(Krizhevsky et al. 2012)

More details about the model and tricks used for training it (data augmentation, dropout) can be found in Krizhevsky et al. (2012).