1 CNN Layers

1.1 Dense layer (fully connected layer)

As the name suggests, every output neuron of the inner product layer has full connection to the input neurons. See here for a detailed explanation. The output is the multiplication of the input with a weight matrix plus a bias offset, i.e.:

$$f(x) = Wx + b. \tag{1}$$

This is simply a linear transformation of the input. The weight parameter W and bias parameter b are learnable in this layer. The input x is a d dimensional vector, and W is an $n \times d$ matrix and b is n dimensional vector.

1.2 ReLU layer

We add nonlinear functions after the inner product layers to model the nonlinearity of real data. One of the activation functions found to work well in image classification is the rectified linear unit (ReLU):

$$f(x) = max(0, x). \tag{2}$$

There are many other activation functions such as the sigmoid and tanh function. See here for a detailed comparison among them. There is no learnable parameter in the ReLU layer.

The ReLU layer is sometimes combined with inner product layer as a single layer; here we separate them in order to make the code modular.

1.3 Pooling layer

It is common to use pooling layers after convolutional layers to reduce the spatial size of feature maps. Pooling layers are also called down-sample layers. With a pooling layer, we can extract more salient feature maps and reduce the number of parameters of CNNs to reduce over-fitting. Like a convolution layer, the pooling operation also acts locally on the feature maps, and there are also several hyper parameters that controls the pooling operation including the windows size k and stride s. The pooling operation is typically applied independently within each channel of the input feature map. There are two types of pooling operations: max pooling and average pooling. For max pooling, for each window of size $k \times k$ on the input feature map, we take the max value of the window. For average pooling, we take the average of the window. We can also use zero padding on the input feature maps. If the padding size is p, the first two dimension of output feature map are $[(h + 2p - k)/s + 1] \times [(w + 2p - k)/s + 1]$. This is the same as in the convolutional layer. Since pooling operation is channel-wise independent, the output feature map channel size is the same as the input feature map channel size.

1.4 Convolution layer

See here for a detailed explanation of the convolution layer.

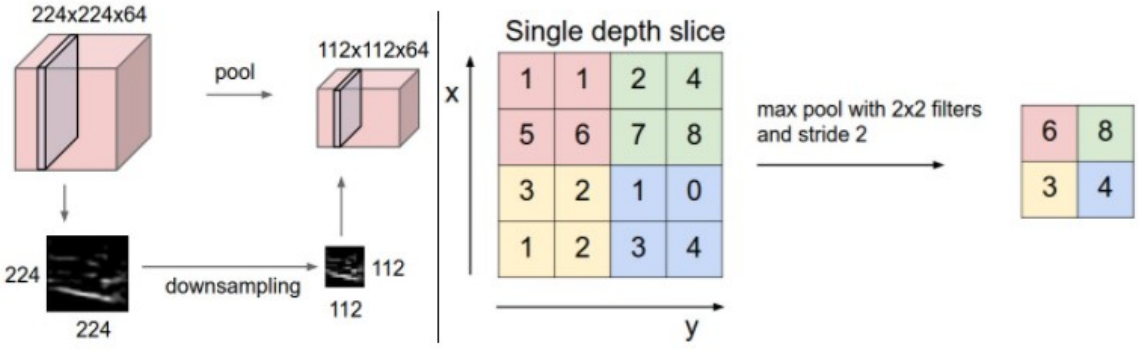

Figure 1: pooling layer

The convolution layer is the core building block of CNNs. Different from the inner product layer, each output neuron of a convolution layer is only connected to some input neurons. As the name suggests, in the convolution layer, we apply a convolution operation with filters on input feature maps (or images). Recall that in image processing, there are many types of kernels (filters) that can be used to blur, sharpen an image or to detect edges in an image. See the wiki page if you are not familiar with the convolution operation. In a convolution layer, the filter (or kernel) parameters are learnable and we want to adapt the filters to data. There is also more than one filter at each convolution layer. The input is a three dimensional tensor, rather than a vector as in the inner product layer. We represent the input feature maps (it can be the output from a previous layer, or an image from the original data) as a three dimensional tensor with height h, width w and channel c (for a color image, it has three channels).

Fig. 1 shows the detailed convolution operation. The input is a feature map, i.e., a three dimensional tensor with size $h \times w \times c$. Assume the (square) window size is k, then each filter is of shape $k \times k \times c$ since we use the filter across all input channels. We use n filters in a convolution layer, then the dimension of the filter parameter is $k \times k \times c \times n$. Another two hyper-parameters in the convolution operation, are the padding size p and stride step s. Zero padding is typically used; after padding, the first two dimensions of input feature maps are $(h + 2p) \times (w + 2p)$. The stride s controls the step size of the convolution operation. As Fig. 1 shows, the red square on the left is a filter applied locally on the input feature map. We multiply the filter weights (of size $k \times k \times c$) with a local region of the input filter map and then sum the product to get the output feature map. Hence, the first two dimensions of the output feature map are $[(h+2p-k)/s+1] \times [(w+2p-k)/s+1]$. Since we have n filters in a convolution layer, the output feature map is of size $[(h+2p-k)/s+1] \times [(w+2p-k)/s+1] \times n$.

Besides the filter weight parameters, we also have the filter bias parameters which is a vector of size n, that is, we add one scalar to each channel of the output feature map.