1 Convolutional Networks

By: Jussi Martin

1.1 Preface

This blog post is written for the *Image and Video Statistics* study group held by the AIHelsinki Academia. It is based on the Chapter 9 of the book *Deep Learning* by Ian Goodfellow, Yoshua Bengio and Aaron Courville, which is available online at `http://www.deeplearningbook.org`.

We will assume the reader to be familiar with basic components of artificial neural networks, such as weights and biases. Some mathematical preliminaries are introduced in the text. Our goal is to introduce basics of convolutional neural networks and explain some of their properties.

Convolutional networks are particularly useful in image classification, since they can learn to be invariant to local translations in the input. For example, convolutional network can learn to classify image of a face correctly despite individual faces having slight variation in the positions of facial features.

Besides the classificational aspects, convolution is also effective in the sense that it uses fewer connections and shared parameters, which reduces the computational cost and memory requirements and improves the statistical efficiency. In deep networks this difference is significant.

1.2 Definition

We begin by recalling the mathematical notion of convolution. Since we are dealing with images, the input data is two dimensional and the convolution $(I * K)$ is defined as

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

where I is an images, K is a filter applied to it, i and j are the pixel coordinates. One special property of convolution is that it is commutative, that is

$$(K * I) = (K * I).$$

Which is relatively easy to derive.

In practice the input grid is finite, the filter is defined on a smaller grid and usually only applied when its coordinates are fully contained in the input grid.

Many machine learning libraries actually defined convolution as

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n),$$

which is also known as cross-correlation or convolution without kernel-flipping. It is not commutative, but from a practical point of view this dose not make any difference since commutativity is not needed and the network can learn its parameters with respect to either of these convolutions.

We say that a neural network is convolutional if it uses convolution in place of general matrix multiplication in at least one of its layers.

1.3 Convolutional Networks Structure

Here we give an example layout for convolutional networks structure. We will also consider what type of outputs convolutional network can have. Individual layers are presented later.

Typically the first layer in a convolutional network is convolutional. After this the output is passed trough a nonlinearity, at this point there might be a pooling layer. This type of structure can then be repeated multiple times. The final layers are usually fully connected.

Here is an example network, which was one of the networks used by the VGG team in the ImageNet 2014 Challenge:

input, 224 $\times$ 224 RGB image
convolutional, kernel width 3, channels 64
convolutional, kernel width 3, channels 64
max pooling
convolutional, kernel width 3, channels 128
convolutional, kernel width 3, channels 128
max pooling
convolutional, kernel width 3, channels 256
convolutional, kernel width 3, channels 256
convolutional, kernel width 3, channels 256
convolutional, kernel width 3, channels 256
max pooling
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
max pooling
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
convolutional, kernel width 3, channels 512
max pooling
fully connected, nodes 4096
fully connected, nodes 4096
fully connected, nodes 1000
soft-max

All the hidden layers are followed by ReLu (defined later), which is not shown for the sake of brevity. For details, see `https://arxiv.org/pdf/1409.1556.pdf`.

1.4 Tensors and Convolutional Layer

We adopt the Deep Learning book's convention of defining tensors as multidimensional arrays of real numbers.

Namely, 0-D tensors are just real numbers, 1-D tensors are arrays

$$T = (T_1, \ldots, T_n)$$

of real numbers, 2-D tensors are arrays

$$T = ((T_{1,1}, \ldots, T_{1,n}), \ldots (T_{m,1}, \ldots, T_{m,n}))$$

of arrays of real numbers (with mutually equal length), and so on.

Basically our tensors can be viewed as D-dimensional grids of real numbers. Example in 2-D:

$$((1,2,3),(4,5,6),(7,8,9)) \quad \simeq \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

In general case the sizes of the axes do not need to match.

Typically convolutional network has several channels which may correspond to different colors in the image or they may originate from same input image which is processed in different ways in parallel.

We now assume for the sake of simplicity that the input grid has spatial shape of $a \times a$ grid and the kernel grid has spatial shape of $b \times b$ grid, where a and b are independent of the channel index and

$$a = bN \quad \text{for some } N > 1.$$

We also denote by c and d the number of input and output channels in the layer.

Output tensor Z of the convolutional layer at pixel (j, k) of the channel i is now defined by

$$Z_{i,j,k} = \sum_{l=1}^{c} \sum_{m=1}^{a} \sum_{n=1}^{b} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

where V is the input tensor with l as its channel index, K is the weight tensor with $K_{i,l,m,n}$ representing the weight of the connection from node in output channel i to node in input channel l, with an offset of m rows and n columns between the channels.

Also, the output node $Z_{i,j,k}$ is only defined when

$$1 \leq j + m - 1 \leq a \quad \text{and} \quad 1 \leq k + n - 1 \leq a$$

for all terms in the summation. This is due to the requirement that the kernel fits in to the input grid, which now looks different since we are using convolution without kernel-flipping. Since m and n are at most b, we see that

$$1 = \max\{2 - b, 1\} \leq j \leq a - b + 1 \quad \text{and the same way} \quad 1 \leq k \leq a - b + 1.$$

Namely, this means that the convolution has shrunk the spatial width of the data grid from a to $a - b + 1$. This is a problem since it opposes a limit to the depth of the network. It can however be compensated by adding artificially zeros to the boundary of the input layer. This way it is possible to obtain deeper networks. The method is known as *zero padding*.

The convolution can also be chosen to accept input from a periodical subgrid. This is known as convolution with a stride s, where we require that

$$a = sM \quad \text{for some } M > 1$$

and defined by

$$Z_{i,j,k} = \sum_{l=1}^{c} \sum_{m=1}^{M} \sum_{n=1}^{M} V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n}$$

with the same type of validity conditions required as before. Similar stride is used also for pooling which we define later.

1.5 Nonlinearities

Some of the typical nonlinearities used in convolutional networks are:

Rectified linear unit also known as *ReLu*, defined by

$$f(x) = \max(x, 0).$$

Logistic funtion also known as *sigmoid*, defined by

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

Hyperbolic tangent function also known as *tanh*, defined by

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

A nonlinearity is applied after convolutional layer. Its output tensor T is defined by

$$T_{i,j,k} = f(Z_{i,j,k} + b_{i,j,k})$$

where Z is the output tensor of the convolutional layer and $b_{i,j,k}$ is a bias unit, which is often shared between units in the same channel, that is

$$b_{i,j,k} = b_i \quad \text{for all } j \text{ and } k$$

where j and k are the pixel indices and i is the channel index.

In some case, for example, if objects in a set of images are known to be in the center of the images, it may be more convenient to allow the biases to vary with respect to location.

1.6 Pooling

Typically pooling is used for down sampling, which means that the grid size of the data is reduced. Method that is most often used nowadays is max pooling and its output tensor T is defined by

$$T_{i,j,k} = \max\{V_{i,sj+\alpha,sk+\beta} | \text{ for } (\alpha, \beta) \in \{1, \ldots, b\} \times \{1, \ldots, b\}\}$$

where s is the stride, b is the spatial width of the max pooling kernel. Often max pooling is used with $s = b > 1$, which has the effect of shrinking the data grid by factor s.

Example in 2-D, max pooling with both s and kernel width set to 2:

$$\begin{array}{|c|c|c|c|} \hline 0 & 1 & 3 & 9 \\ \hline 4 & 8 & 6 & 2 \\ \hline 7 & 4 & 2 & 0 \\ \hline 4 & 3 & 5 & 1 \\ \hline \end{array} \xrightarrow{\text{Pooling}} \begin{array}{|c|c|} \hline 8 & 9 \\ \hline 7 & 5 \\ \hline \end{array}$$

1.7 Sparse Connections

One may notice that an output node in a convolutional or a pooling layer is only connected to a set of nodes of the size of the convolution or pooling kernel. Similarly, the input nodes are connected only as many output nodes as is the size of the kernel (or some times even less if a stride is used). This is in great contrast to fully connected layers where every node in the input is connected to every node in the output nodes.

1.8 Receptive Field

For a given node the nodes in the previous layers which are directly connected to it are known as receptive field.

Although the receptive field for a node in the early layers of the network is small due to sparse connections, it is larger for nodes in the deeper layers of the network and it can even cover the whole input image if the network has enough layers.