Motivation [GBC16]

- sparse interactions
- parameter sharing
- equivariant representations (i.e. with respect to translation)
- efficiency
- Train deeper networks.

Sparse interactions: From dense to block circulant matrix.

Parameter sharing: Use the same parameters for more than one job.

Equvariance: Translations of an input should not change the outcome.

Defining cross-correlation

$$S(i,j) = (K * I) = \sum_m^M \sum_n^N I(i+m, j+n)K(m,n) \tag{3}$$

Cross-correlation is convolution without flipping the kernel [GBC16]. Many machine learning libraries implement cross-correlation and call it convolution. In this course we will follow their example.

A convolution example on the board:

$$\mathbf{I} = \begin{pmatrix} 1 & 3 & -1 \\ 2 & 1 & 0 \\ 0 & 2 & -1 \end{pmatrix}, \mathbf{K} = \begin{pmatrix} 1 & 0 \\ 2 & -1 \end{pmatrix} \tag{4}$$

Computing $\mathbf{I} * \mathbf{K}$:

$$\mathbf{I} * \mathbf{K} = \begin{pmatrix} 1 \cdot 1 + 3 \cdot 0 + 2 \cdot 2 + 1 \cdot (-1) & 3 \cdot 1 + (-1) \cdot 0 + 1 \cdot 2 + 0 \cdot (-1) \\ 2 \cdot 1 + 1 \cdot 0 + 0 \cdot 2 + 2 \cdot (-1) & 1 \cdot 1 + 0 \cdot 0 + 2 \cdot 2 + (-1) \cdot (-1) \end{pmatrix} \tag{5}$$

$$= \begin{pmatrix} 4 & 5 \\ 0 & 6 \end{pmatrix} \tag{6}$$

Multichannel convolution

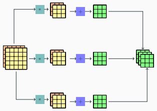

Figure: The plot shows a convolution computation using a 3x2x3x3 kernel on a 2x5x5 input. The kernel pairs convolve with the input, producing 3x3 results. + adds the two channels for each of the three tensors. Finally, everything is stacked. Inspired by [DV16, page 9].

On the board: Explain the effect of the input and output shapes.

I.e:

Kernel (O, I, H, W): Out-Channels, In-Channels, Height, Width

Image (N, C, H, W): Batch-Size, Channels, Height, Width

Results in:

Result (N, O, H_n, W_n)

Image to column and the forward pass

We already know how to train dense network layers using matrix multiplication. Training a CNN the same way requires restructuring the image to express convolution as matrix multiplication,

$$\overline{\mathbf{h}} = \mathbf{K}_f \mathbf{v}_I + \mathbf{b}, \tag{8}$$

$$\mathbf{h}_f = f(\overline{\mathbf{h}}). \tag{9}$$

$\mathbf{v}_I \in \mathbb{R}$ denotes the restructured image input. $\mathbf{K}_f \in \mathbb{R}^{k_o, k_i \cdot k_h \cdot k_w}$ the flattened restructured kernel. o, i, h, w denote the output, input, height, and width dimensions, respectively.

im2col demonstrate in the board: Idea collect the image convolution patches in the columns of a matrix. Use python indexing to set it up. For a 3×3 matrix and a 2×2 kernel without padding this would lead to the index matrices:

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 3 & 4 \\ 1 & 2 & 4 & 5 \\ 3 & 4 & 6 & 7 \\ 4 & 5 & 7 & 8 \end{pmatrix} \tag{10}$$

Pooling

Max pooling layers choose maximum values in predefined regions. Two by two max pooling, for example, picks the maximum in neighboring areas of four pixels. If an input is shifted by two pixels, the result will remain the same! Pooling layers are used repeatedly for a cumulative effect.

→ Draw the effect of max pooling on the board.

Deep convolutional neural networks

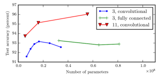

Figure: Comparing deep networks with and without convolutional structures on the Google-Street view dataset [GBC16, page 199].

[GBC16] tells us:

Effect of number of parameters. Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow et al. (2014d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance, as illustrated in this figure.