Convolutional neural networks

Victor Kitov

v.v.kitov@yandex.ru

Convolutional neural networks

- Convolutional neural network:
  - Used for image analysis
  - Uses convolutional layers followed by non-linearity functions and sub-sampling (pooling) layers.
  - Multi-layer perceptron at the end if regression or classification task.

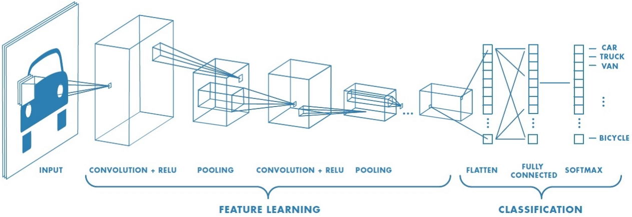
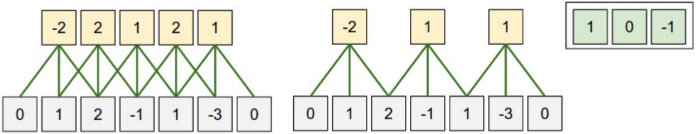


Table of Contents

1-D Convolution operation



$$out1D(x,y)=\sum_{i=-n}^{n}K(i+n+1)in(x+i)+b,\quad Kernel\in\mathbb{R}^{2n+1},\ b\in\mathbb{R}$$

Parameters[1]:

- W - length of input; $2n+1$ - kernel size
- P - amount of padding (to increase dimensionality)
- Type of padding (valid[absent], zero, same [extend], mirror)
- S - stride (offset of kernel); D - dilation (offset inside kernel)

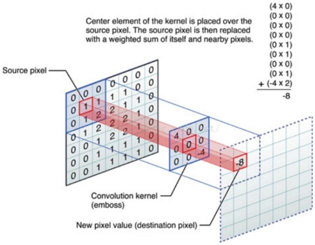
[1] Depending on these parameters, what would be the size of output layer?

Predefined convolution examples

- $K = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5})$ - uniform averaging.
- $K = (0.1, 0.2, 0.4, 0.2, 0.1)$ - averaging with decaying weights.
- $K = (-1, 0, +1)$ - change detection ($f' \approx z_{t+1} - z_{t-1}$)
- $K = (+1, -2, +1)$ - change of change detection ($f'' \approx (z_{t+1} - z_t) - (z_t - z_{t-1})$)

2D convolution

2-D convolution



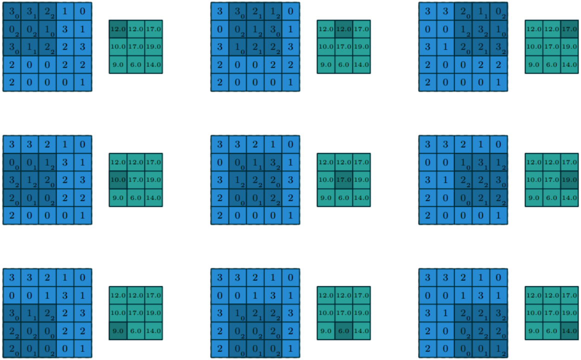
$$out2D(x,y) =$$

$$\sum_{i=-n}^{n}\sum_{j=-n}^{n} K(i+n+1, j+n+1) in(x+i, y+j) + b,$$

$$K \in \mathbb{R}^{(2n+1)x(2n+1)}, \;\; b \in \mathbb{R}$$

Convolution demo[2]

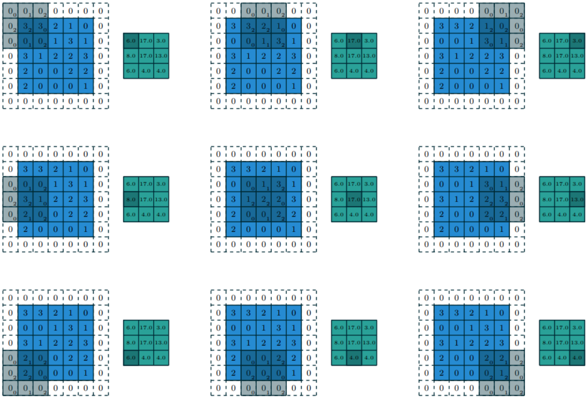
Single layer convolution:



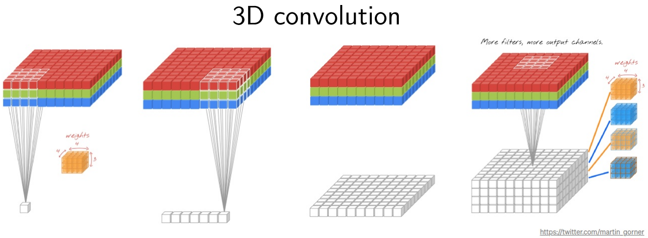
[2] Illustrations from Dumoulin et al. 2018.

Padding allows to increase output feature map

Convolution with stride and zero-padding:



3D convolution



$$out3D(x,y,c) =$$

$$\sum_{i=-n}^{n}\sum_{j=-n}^{n}\sum_{c=1}^{C} K(i+n+1, j+n+1, c)in(x+i, y+j, c) + b,$$

$$K \in \mathbb{R}^{(2n+1)x(2n+1)}, \quad b \in \mathbb{R}$$

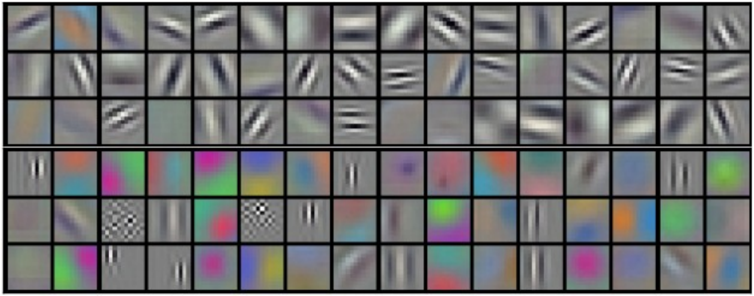
Convolution comments

Comments:

- #parameters↓ because neuron is connected only to some neighborhood of input neurons
- #parameters↓ further, because we apply the same transformation in different locations
  - #parameters=$((2n+1)x(2n+1)+1)\,C_{in}C_{out}$ for set of C_{out} convolutions, applied to feature map with C_{in} channels.
- Visibility region of convolution output is $\pm n$ in terms of previous layer.
  - visibility region increases in terms of earlier layers.
- Stride>1 reduces output spatial dimensionality.
- Convolution should be followed by non-linear transformation.
- Typical to use pretrained convolutions from some large dataset (e.g. ImageNet)

Visualized convolutions from 1st layer of AlexNet

1st layer of AlexNet



- 1st layer convolution kernels $\in \mathbb{R}^{(2n+1)x(2n+1)x3}$, so can be visualized as images.
- Later convolutions use more channels, so they can be visualized by
  - taking actual patches, maximizing convolution activation
  - deriving patches giving maximal activation with

Convolution pyramid

- Convolution applied to input image extracts features.
- Convolution, applied to feature map (intermediate output) extracts more abstract features.
- Typical to gradually decrease spatial resolution, extracting more abstract features from wider areas of input image.
  - downsampling performed by strided convolutions or strided pooling layers.

Convolution pyramid

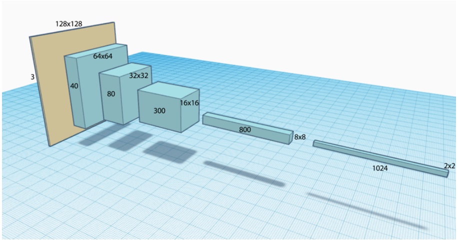
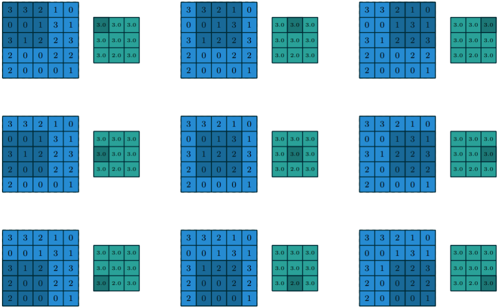


Table of Contents

Max pooling

- Max pooling: "feature is present somewhere in the region" (e.g. for corner detector)
- Introduces invariance to small transitions on the image.
- Stride>1 reduces output spatial dimensionality.

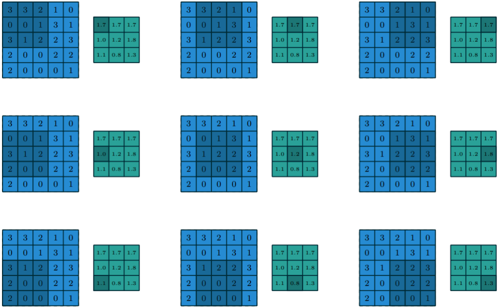
Max 3x3 pooling:



Average pooling

- Avg. pooling: "average feature presence", e.g. for isolated point detector.
- Introduces invariance to small transitions on the image.
- Stride>1 reduces output spatial dimensionality.

Average 3x3 pooling:



Upscaling

- Output size can be increased by applying convolution to enlarged input
  - transposed convolution (padding input values, "bed of nails")

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

  - simple scaling (nearest neighbours or rescaling with smoothing)

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \longrightarrow \begin{pmatrix} a & a & b & b \\ a & a & b & b \\ c & c & d & d \\ c & c & d & d \end{pmatrix}$$

Intermediary tensor->vector of fixed size

- Typical to use multi-layer perceptron at the end.
- It requires input of fixed size, whereas input image may be of various shapes.
- Solutions:
  - rescale&crop input image to fixed size
  - perform global channel-wise pooling $\mathbb{R}^{CxWxH} \to \mathbb{R}^{C}$
  - pyramid pooling:
    1. split output into fixed grid mxm.
    2. perform channel-wise pooling for each fragment selected by the grid
    3. stack results, so $\mathbb{R}^{CxWxH} \to \mathbb{R}^{Cm^2}$

Conclusion

- Fully-connected layers have too many parameters.
- Convolutions solve this by:
  - taking dependencies only from small neighborhood
  - applying the same transformation to different locations
- Avg. and max pooling implement invariance to small transitions on the image.
- Subsequent convolutions extract more abstract features from wider area of input image.