Lecture 23

[Convolutional neural networks (CNN)]

NN for images

Continuous convolution:

$$[f \times g](\vec{z}) = \int d\vec{u}\, f(\vec{u})\, g(\vec{z}-\vec{u})$$

Discrete convolution: ↙ 1D example

if $f(\vec{u})$ is defined at $\{-L, \dots, 0, \dots, L\}$:

$f(-L) = w_{-L}, \dots, f(L) = w_L$ filter/kernel

if $g(\vec{u})$ is defined at $\{-N, \dots, 0, \dots, N\}$:

$g(-N) = x_{-N}, \dots, g(N) = x_N$ input

Then $[w \times x](i) = w_{-L} x_{i+L} + \dots + w_0 x_i + \dots + w_L x_{i-L}$ (1)

$i+L$ & $i-L$ must be within the $[-N, N]$ range

Similarly, one can define a cross-correlation:

$$[w \otimes x](i) = w_{-L} x_{i-L} + \dots + w_{-1} x_{i-1} + \dots + w_L x_{i+L} \quad (2)$$

(1) & (2) are the same if $w_{-k} = w_k$.

One can restrict Eq. (2) to non-negative indices:

$$[w \otimes x](i) = \sum_{u=0}^{L-1} w_u x_{i+u}$$ filter of length L

Ex.

0	1	2	3	4	5	6

input

$\times$

1	2

kernel

$=$

2	5	8	11	14	17

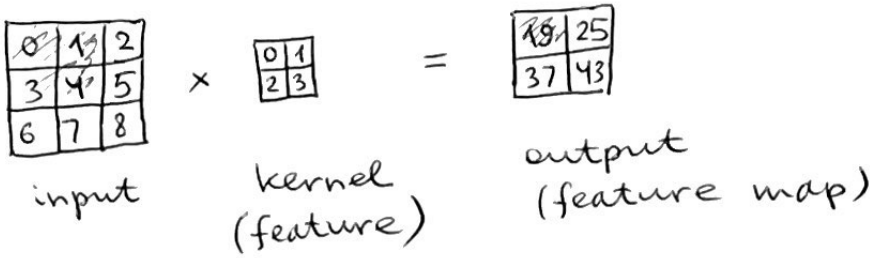
output

Similarly, in 2D

$$[w \circledast x](i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v}\, x_{i+u,j+v}$$

2D $H \times W$ filter

Ex.



In matrix notation,

$$\begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \times \begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} = \begin{pmatrix} w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 \dots \\ w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 \dots \end{pmatrix} \quad (11)$$

2×2

$$(=) \begin{pmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_9 \end{pmatrix}$$

4×9 (sparse weight matrices, shared weights)

9 (flattened)

In general, $f_h \times f_w$ filter applied to an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$: use zero padding of the image (borders of zeros) so that the output retains the $x_h \times x_w$ size.

With symmetric padding of size p_h, p_w we have:

$$x_h \times x_w \xrightarrow[\text{padding}]{} (x_h + 2p_h) \times (x_w + 2p_w) \xrightarrow[\text{convolution}]{}$$

$$\to (x_h + 2p_h - f_h + 1) \times (x_w + 2p_w - f_w + 1) \to$$

$$\begin{cases} 2p_h = f_h - 1 \\ 2p_w = f_w - 1 \end{cases}$$

$$\to x_h \times x_w .$$

Strided convolution: apply filters with step sizes > 1 or, equivalently, ~~[illegible]~~ keep only $1/(s_h s_w)$ convolutions

(s_h, s_w: step sizes)

Output size:

$$\left\lfloor \frac{x_h + 2p_h - f_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{x_w + 2p_w - f_w + s_w}{s_w} \right\rfloor$$

smaller than input

Multiple channels:

For ex., RGB channels in color images

Then $$z_{i,j} = b + \sum_{c=0}^{C-1} \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v,c} \, x_{si+u, sj+v, c}$$

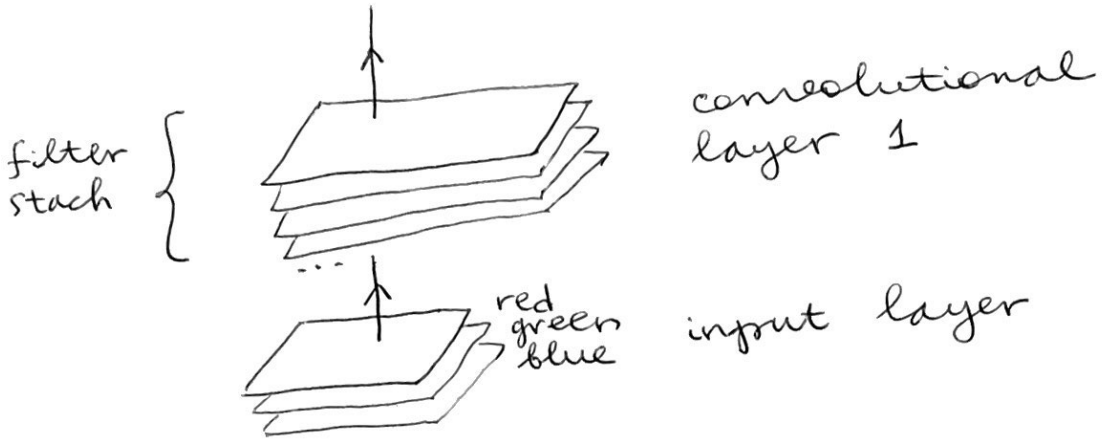
(*) (b: bias)

$s = s_h = s_w \leftarrow$ stride

Eq. (*) provides a single filter; with multiple filters, we have:

$$z_{i,j,\boxed{d}} = b_d + \sum_c \sum_u \sum_v w_{u,v,c,d}\, x_{s_i+u,\, s_j+v,\, c}$$

(**) filter index: $d = 0, \ldots, D-1$



Pointwise convolution:
no "averaging" across locations

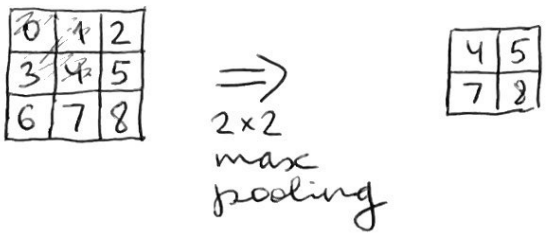
$$z_{i,j,d} = b_d + \sum_{c=0}^{C-1} x_{i,j,c}\, w_{0,0,c,d}$$

Same spatial dimensions: $H \times W$,
but the number of channels
goes from C to D.

Pooling layers: convolutions preserve spatial information (at reduced resolution)

Idea: "pool" incoming values by taking a max (max pooling) or an average (ave pooling).

Ex.



[pooling is usually done separately on each feature channel.]

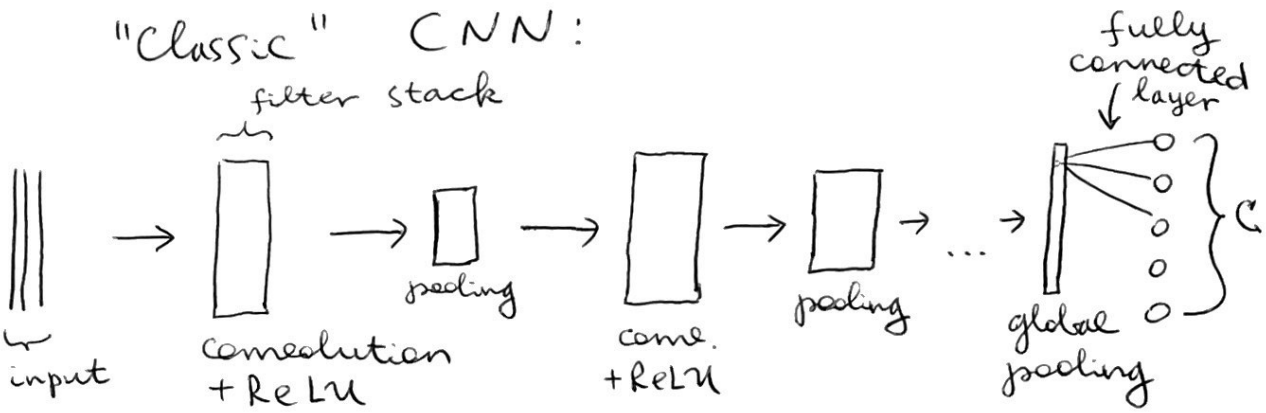
At the end of the CNN, one can do global average pooling (pooling filter size = H×W):

$$H \times W \times D \rightarrow 1 \times 1 \times D = D$$

(D ↑ # channels)

This D-dim vector can be fed into a softmax output layer with C nodes.

"Classic" CNN:



Deep CNNs may suffer from vanishing or exploding grads.

Idea: add normalization layers to the CNN architecture.

Batch normalization (BN):

ensures the distribution of activations has mean = 0 & var = 1 when averaged over a minibatch.

$\vec{z}$ ← vector of activations in a layer

$$\vec{\mu}_B = \frac{1}{|B|}\sum_{\vec{z}\in B}\vec{z}, \qquad \vec{\sigma}_B^2 = \frac{1}{|B|}\sum_{\vec{z}\in B}(\vec{z}-\vec{\mu}_B)^2$$

B is the minibatch of size $|B|$ containing example n (datapoint)

$$\vec{z}_n' = \frac{\vec{z}_n - \vec{\mu}_B}{\sqrt{\vec{\sigma}_B^2 + \epsilon}}$$ standardized; $\epsilon > 0$ = const

$$\vec{z}_n'' = \vec{\gamma} \times \vec{z}_n' + \vec{\beta}$$ shifted & scaled [output of the BN layer]

($\times$: element-wise)

$\vec{\gamma}$ & $\vec{\beta}$ are fitted prms.

In the context of CNN, $\vec{\mu}_B$ is averaged across both spatial locations and examples $\Rightarrow$ $\dim\{\vec{\mu}_B\}$ = # channels.

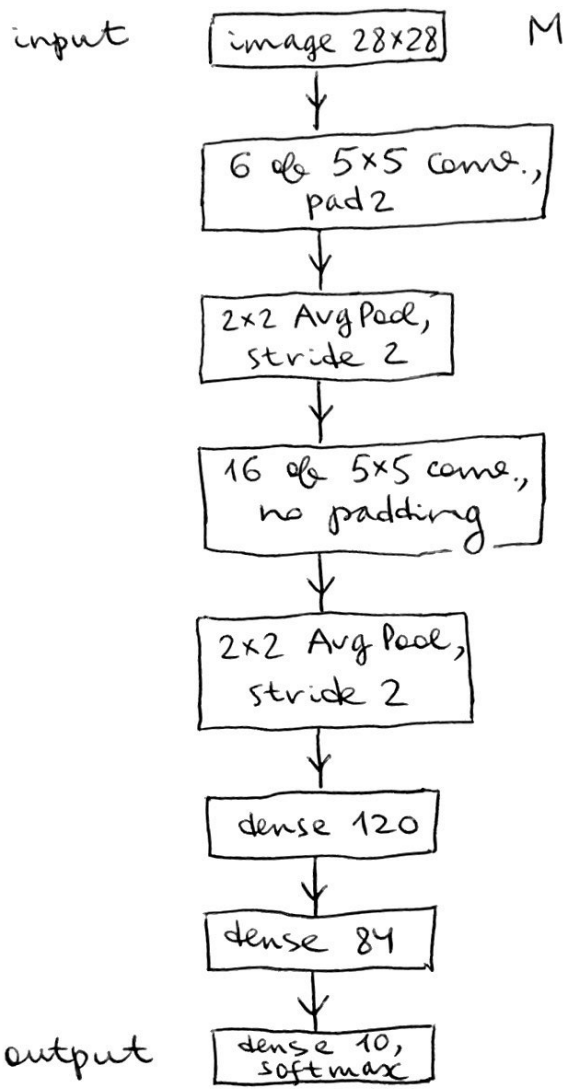
At test time, first compute $\vec{\mu}_e$ & $\vec{\sigma}_e^2$ on the entire training set and then 'freeze' these prms for testing.

(e: layer index)

Adaptive grad clipping: instead of BN layers, use grad clipping with c = fitting prm.

CNN architectures

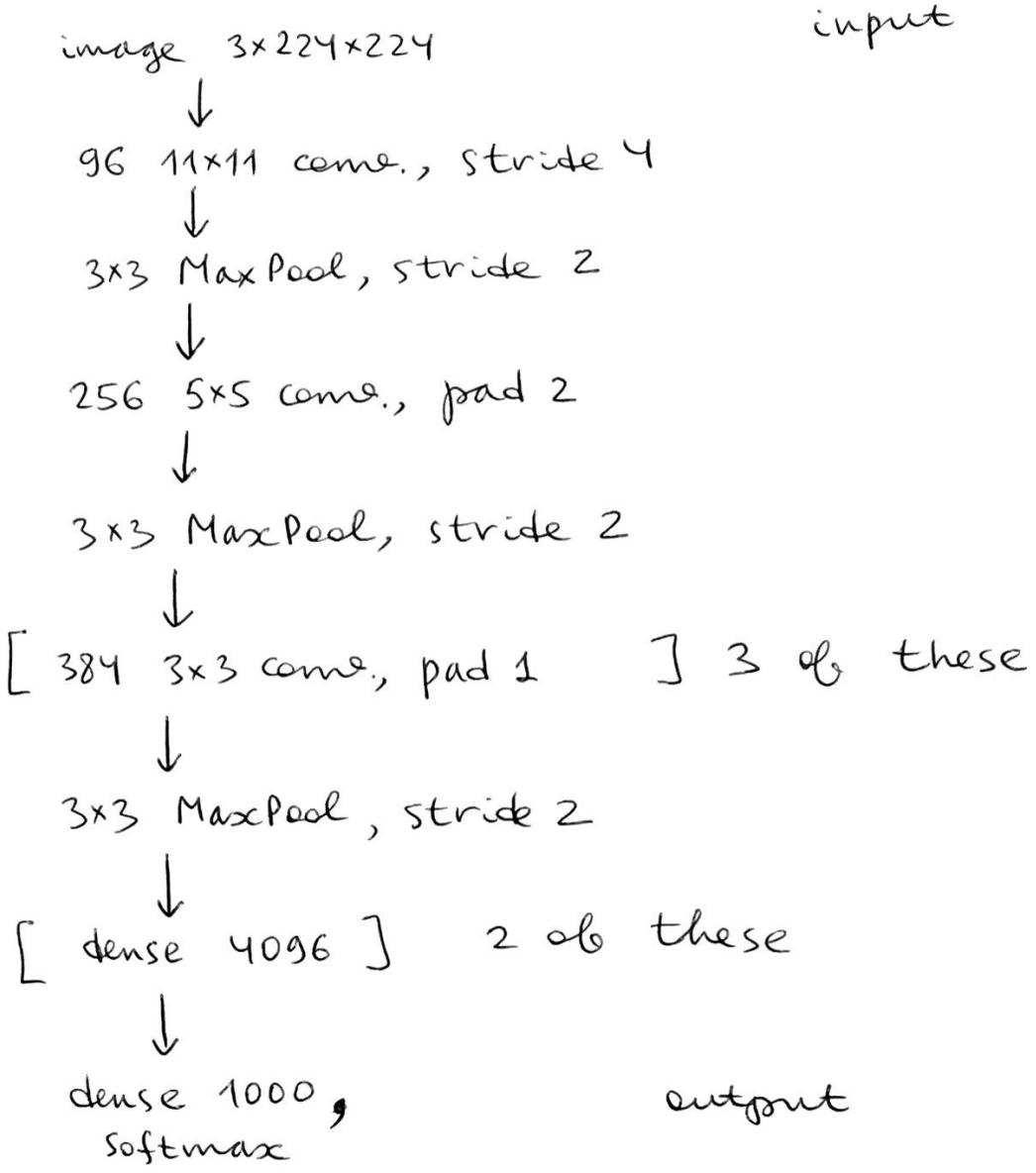
LeNet5



MNIST: 28×28, 1 channel

test accuracy = 98.8% after 1 epoch

AlexNet



~60 M prms (mostly due to dense layers)

ImageNet : ~1M labeled examples

Top 5 error rate was reduced from ~26% (SOTA) to ~15%.