Convolutional neural network

2024年9月12日 22:23

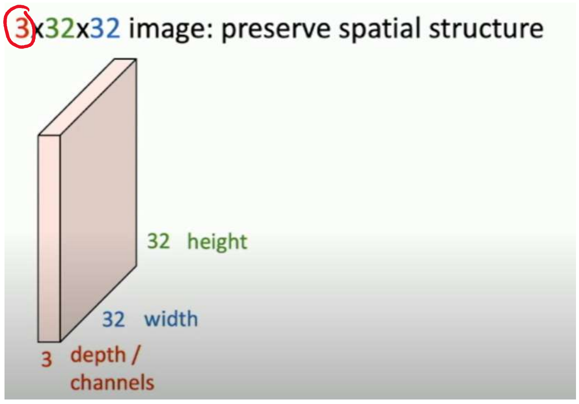
$\underbrace{32 \times 32}_{1024} \times 3 \to 3072 \times 1$ fully connected

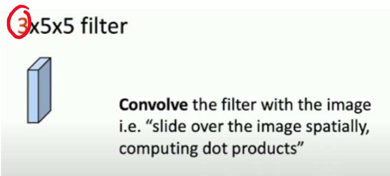
3 Components: Conv layers, Pooling, Normalization

1 Convolution Layers

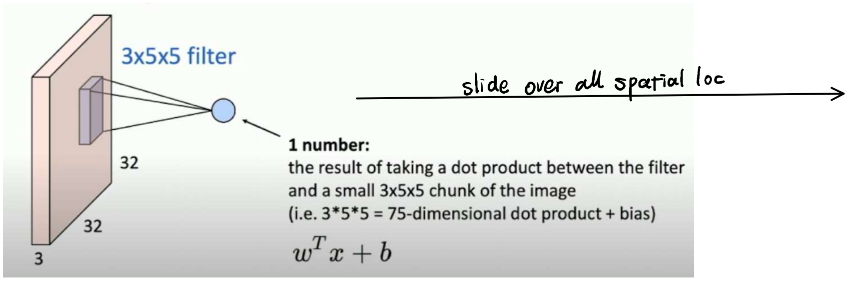
1)

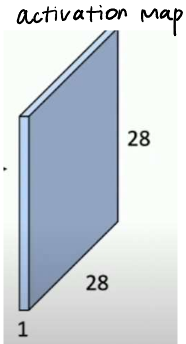
1°





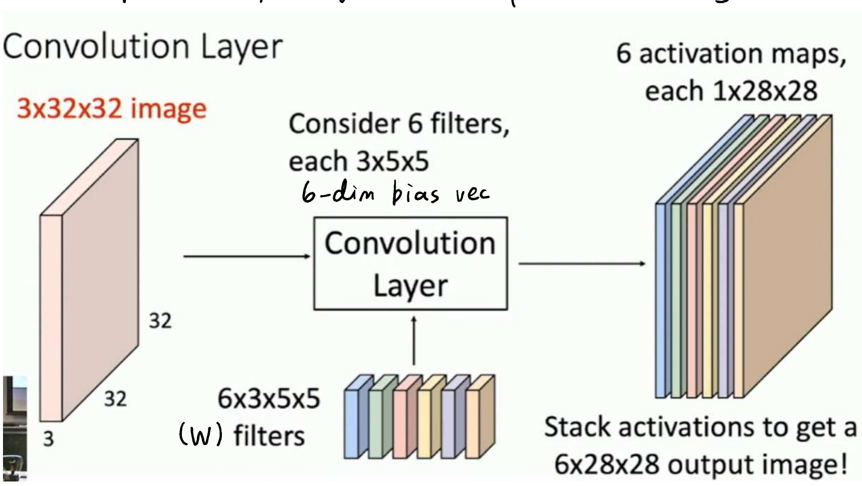
activation map





$3 \times 32 \times 32$ $\qquad$ $1 \times 28 \times 28$

2° 可以有更多 filters，提取不同特征 → act- maps, stack them together.



3° Batch of img $2 \times 3 \times 32 \times 32 \xrightarrow{6 \times 3 \times 5 \times 5 \text{ filters}} 2 \times 6 \times 28 \times 28$ Batch of outp

$N \times C_{in} \times H \times W$ $\qquad$ $C_{out} \times C_{in} \times K_w \times K_h$ $\qquad$ $N \times C_{out} \times H' \times W'$

3° Batch of img $2\times3\times32\times32 \xrightarrow{6\times3\times5\times5 \text{ filters}} 2\times6\times28\times28$ Batch of outp

$N\times C_{in}\times H\times W$ $\quad C_{out}\times C_{in}\times K_w\times K_h$ $\quad N\times C_{out}\times H'\times W'$

inp channels $\quad$ mayb dif

2) Stacking Convolutions

1° inp $\longrightarrow$ Conv $\longrightarrow$ 1st hid $\longrightarrow$ Conv $\longrightarrow$ 2nd hid $\longrightarrow$ Conv ...

inp	Conv	1st hid	Conv	2nd hid
$N\times3\times32\times32$	W_1: $6\times3\times5\times5$	$N\times6\times28\times28$	W_2: $10\times6\times3\times3$	$N\times10\times26\times26$
	b_1: 6	$32-5+1$	b_2: 10	$28-3+1$

oriented edges, opposing colors

Stacking 2 conv → another conv $\quad y=W_2W_1x$ linear classifier

2° Size

① In: W $\quad 7\times7$

Fil: K $\quad 3\times3$

Out $W-K+1$ $\quad 5\times5$

Feature maps shrink, lim #layers, sol:

② Padding, + 0s around the input

In: W

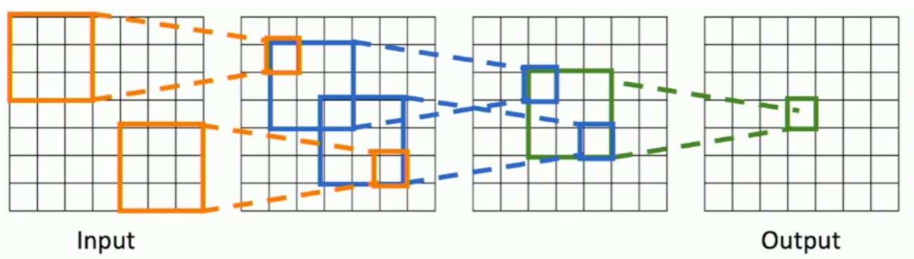
Fil: K

Pad: P

Out: $W-K+1+2P$

(common: $P=(K-1)/2$ to make size in = out)

3° Receptive Fields



Layer	3	2	1	0
Field size	$1+3\times2=7$	$1+2\times2=5$	$1+1\times2=3$	1

L Layer receptive field size $= 1 + L\times(K-1)$

fil size

e.g. input 1000×1000, $K=3$, $L=?$

$1000 = 1 + L\times(3-1)$

$L = 999\div2 = 499.5$

Large imgs need many layers for each outps to "see" the whole img, sol:

(gloabal context)

4° Downsample inside the network

Controlling the stride: indirectly downsampling ∵ fewer data pt are being processed.

① Strided conv:

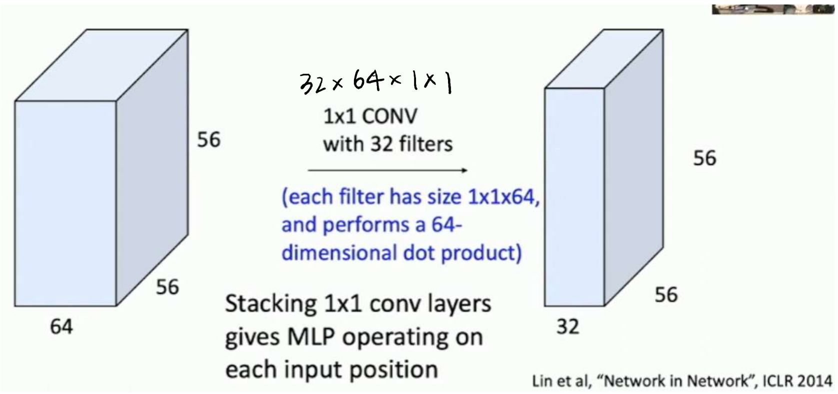
In W $\quad 3\times32\times32$

Fil K $\quad 10\times5\times5$

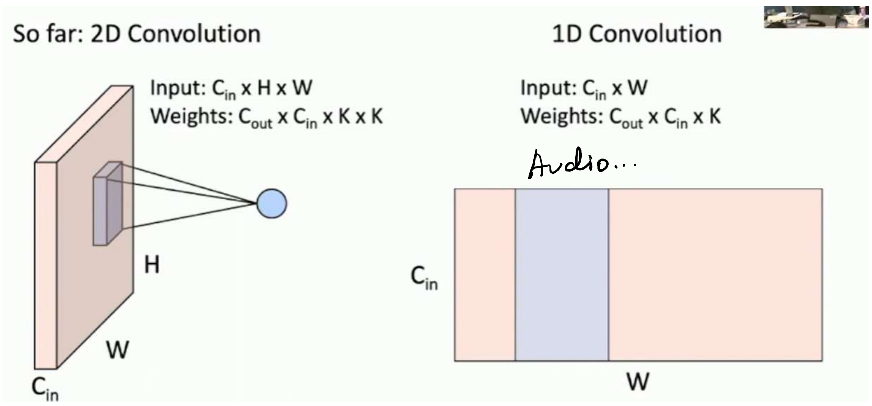
In	W	$3\times32\times32$
Fil	K	$10\times5\times5$
Pad	P	2
Str	S	1
Out	$(W-K+2P)/S+1$	$(32-5+2\times2)/1+1=32$ $10\times32\times32$

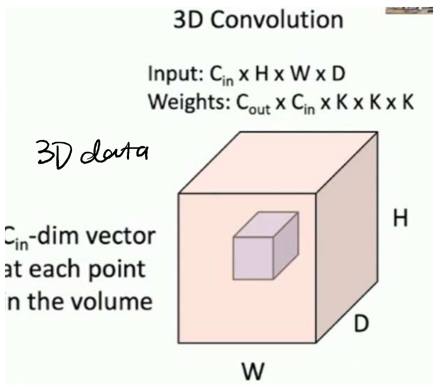
\# learnable para
- ① para/fl: $3\times5\times5+\underset{\text{bias}}{1} = 76$
- ② × #fl: $76\times10 = 760$

\# multiply-add oper
- #outs: $10\times32\times32 = 10240$
- inner product/out: $3\times5\times5 = 75$
- total: $75\times10240 = 768\text{K}$

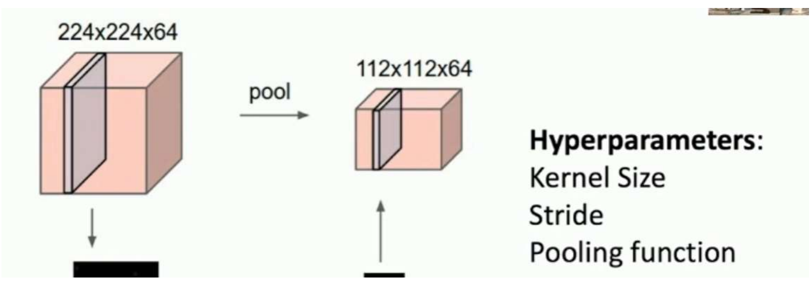


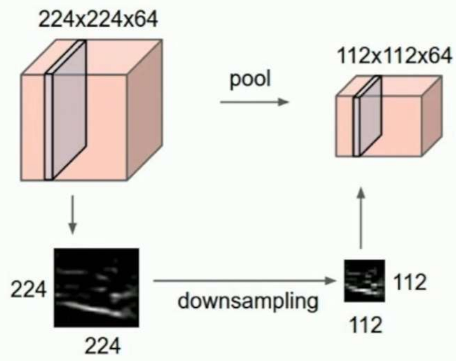
5° Other conv





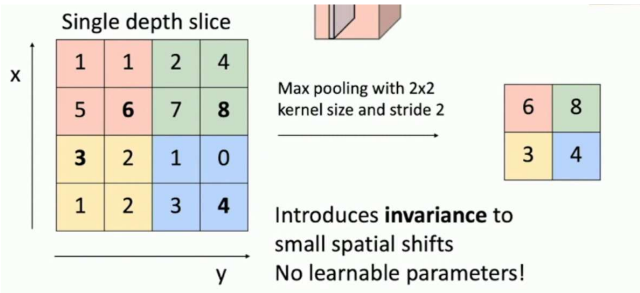
2 Pooling Layer: downsample 另一法





Hyperparameters:
Kernel Size
Stride
Pooling function

1° Max Pooling

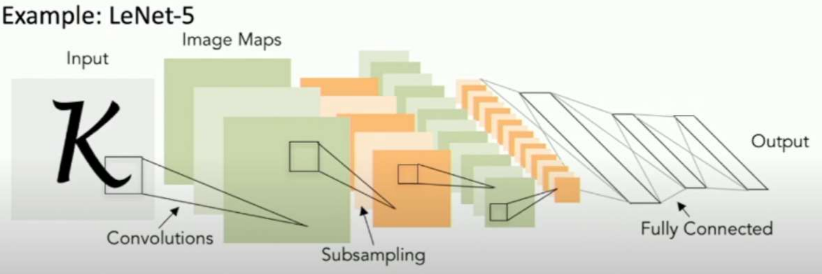


Introduces **invariance** to small spatial shifts
No learnable parameters!

3 Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5



Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	

Spatial size ↓ (pl & strided)

channels ↑ (total volume is preserved)

ReLU 不一定需要.

Deep NN: hard to train (converge), sol:

4 Normalization

1) Batch Nor in fully-connected

1° Idea: "Normalize" the outputs of a layer so they have zero mean and unit variance

Why? Helps reduce "internal covariate shift", improves optimization

We can normalize a batch of activations like this:

Why? Helps reduce "internal covariate shift", improves optimization

We can normalize a batch of activations like this:

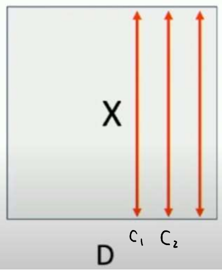
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!

convert inp → more standardized dist

Input: $x : N \times D$

batch dim i N



D

dim each vec j

per channel mean, shape D $\qquad \mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$ ①

~ std, shape D $\qquad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$ ②

Normalized X, $N \times D$ $\qquad \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ ③

What if $\mu = 0$, unit vec: too hard of a constrained, sol:

2° + Learnable scale & shift para: γ, β: D

Learning $\gamma = \sigma$, $\beta = \mu$, recover identity func.

① ② ③

$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$ ④ $N \times D$

① ~ ③ Estimated depend on minibatch; × do this at test-time! sol:

3° Test-Time use μ_j, σ_j^2 got from training

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel std, shape is D

$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ — Normalized x, Shape is N x D

$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$ — Output, Shape is N x D

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

v. 融合.

2) Batch N~ in Conv

Batch Normalization for **fully-connected** networks

$\mathbf{x}: N \times D$

Normalize ↓

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times D$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

only on batch

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$\mathbf{x}: N \times C \times H \times W$

Normalize ↓ ↓ ↓

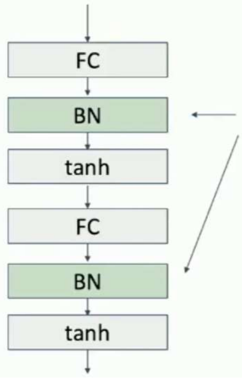
$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1$

$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times C \times 1 \times 1$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

on batch, spatial dims

3)

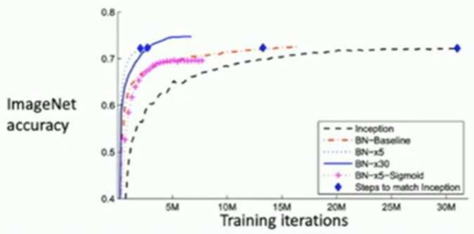


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

⊕

- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!



⊖

- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

4) Others Norm