Course number: 80240743

Deep Learning

Xiaolin Hu (胡晓林) & Jun Zhu (朱军)
Dept. of Computer Science and Technology
Tsinghua University

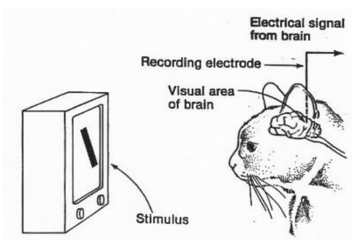
Lecture 5: Convolutional Neural Networks-II

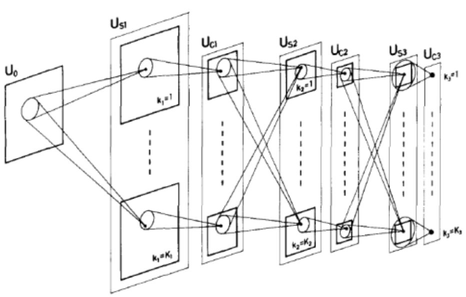
Xiaolin Hu
Dept. of Computer Science and Technology
Tsinghua University

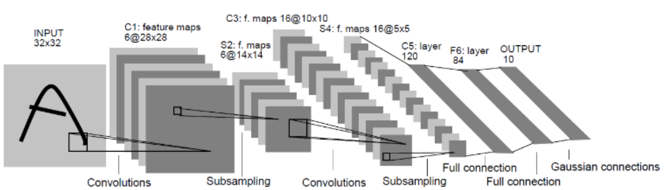
Last lecture review

1. Introduction

History







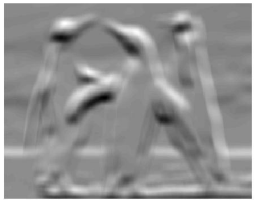
Convolution

inpu

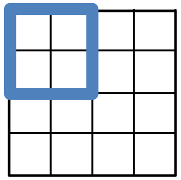
kernel

*

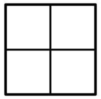
feature



Pooling

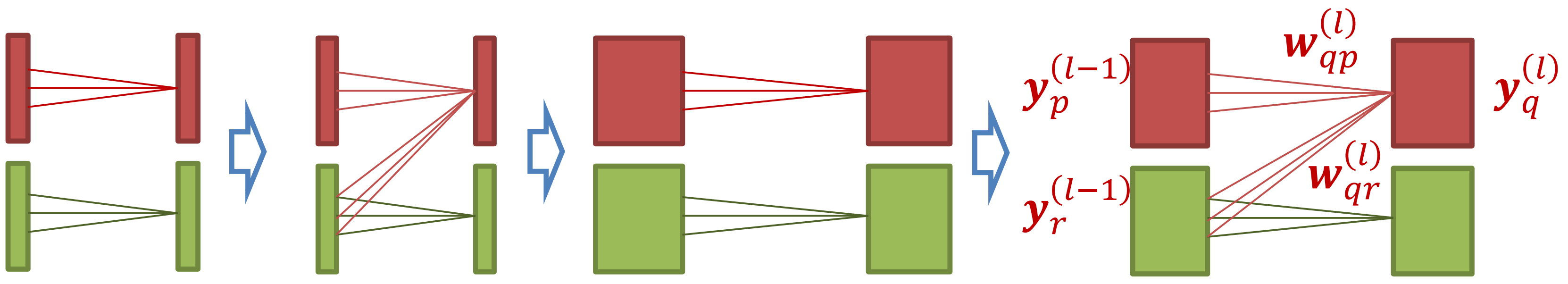


average or max



Last lecture review

2. Convolutional layer



- Forward pass

$$\boldsymbol{y}_q^{(l)} = \sum_{p \in \mathcal{M}_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{w}_{qp}^{(l)}) + b_q^{(l)}$$

- Backward pass

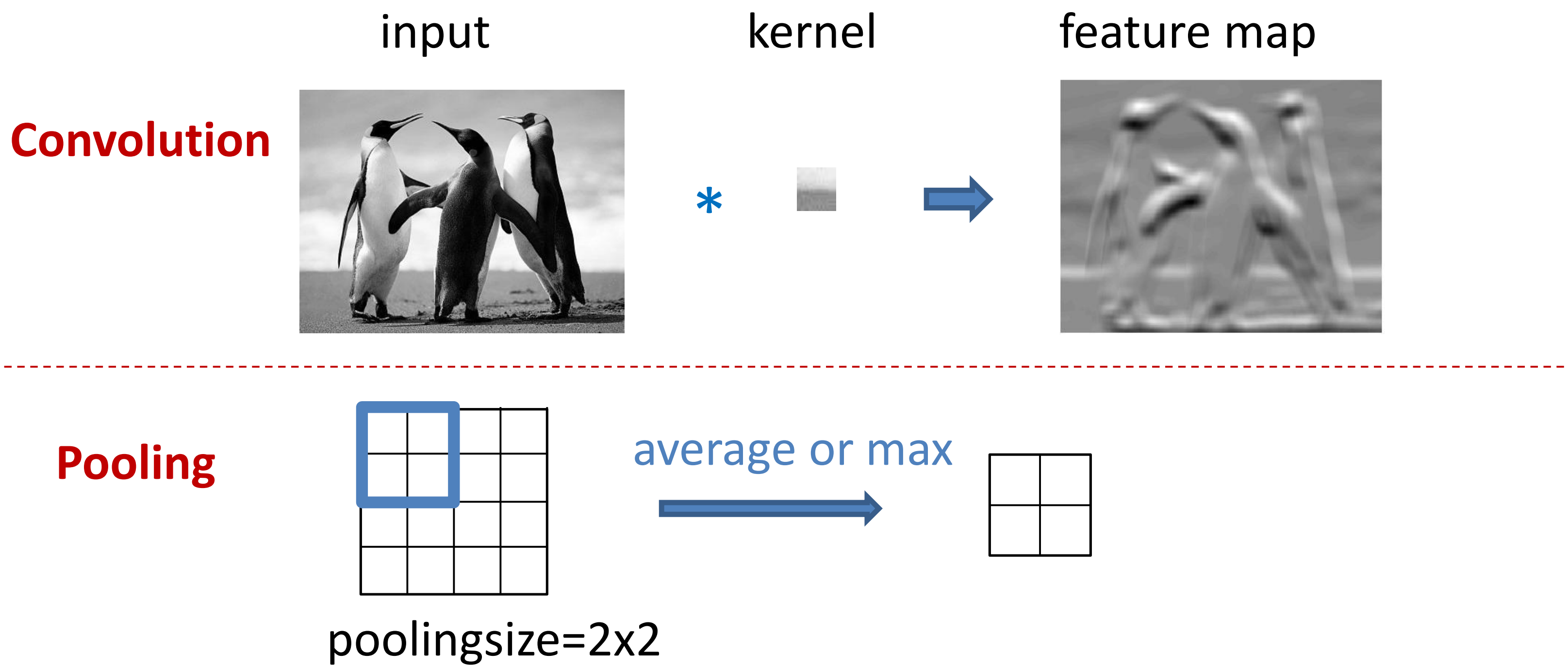
Gradient:

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_{ij}$$

Local sensitivity:

$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \tilde{\mathcal{M}}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

Last lecture review

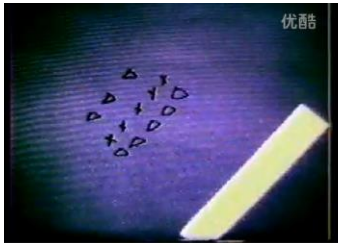


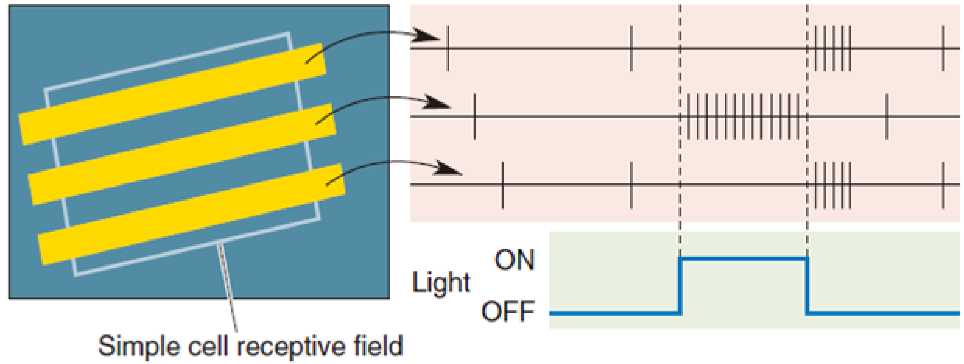
- Convolutional layer and pooling layer
  - Define two additional layers with forward computation and backward computation

Outline

1. Pooling
2. Standard CNN
3. Typical CNNs
4. Training techniques-II
5. Summary

Simple cell and complex cell



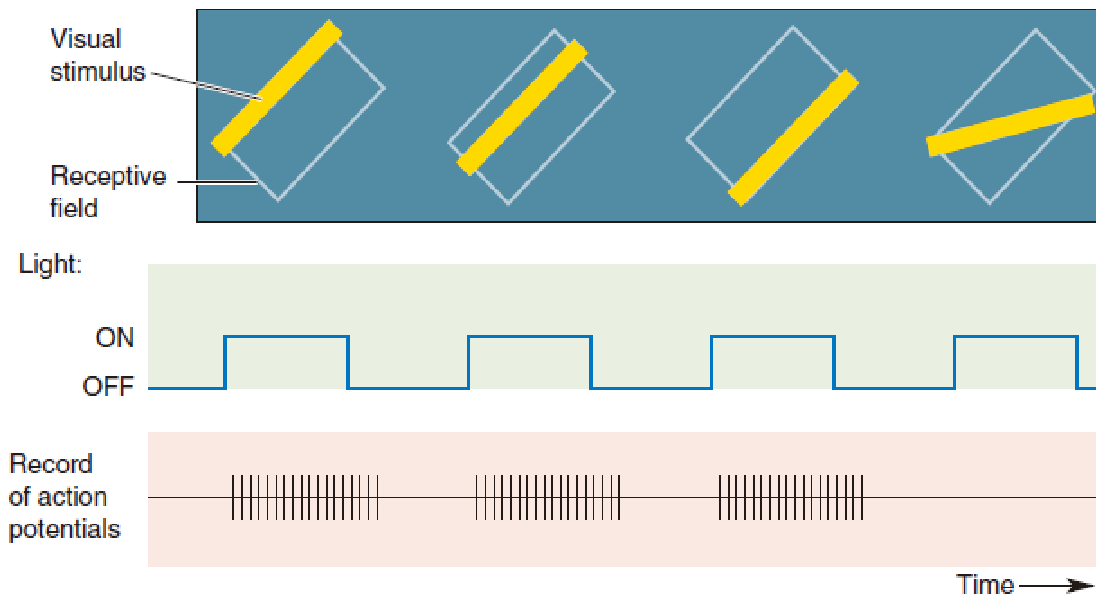


Simple cells can detect local features

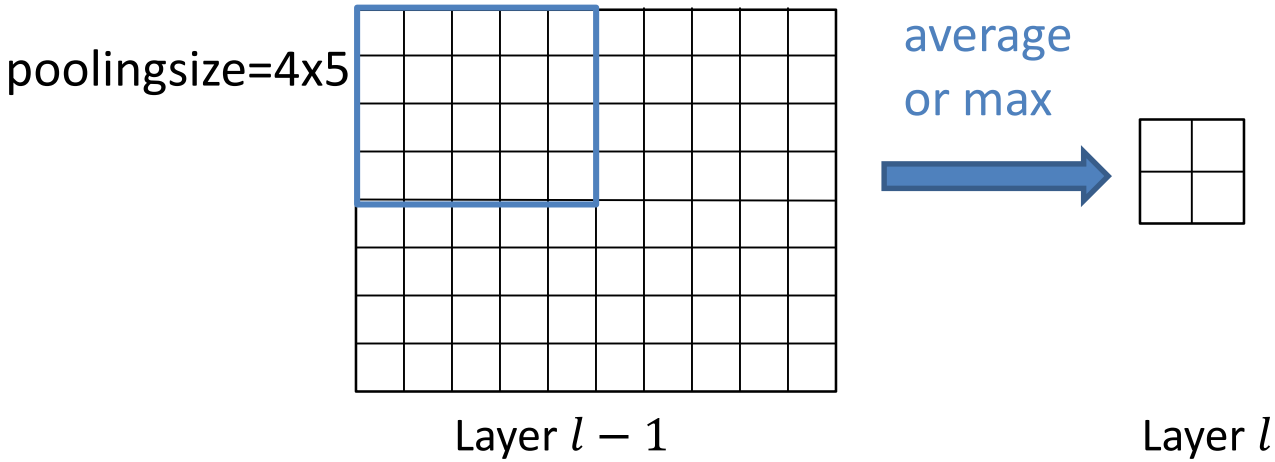
What's the advantage of complex cells?

Translation invariance!

How do we model complex cells?



Pooling in local regions



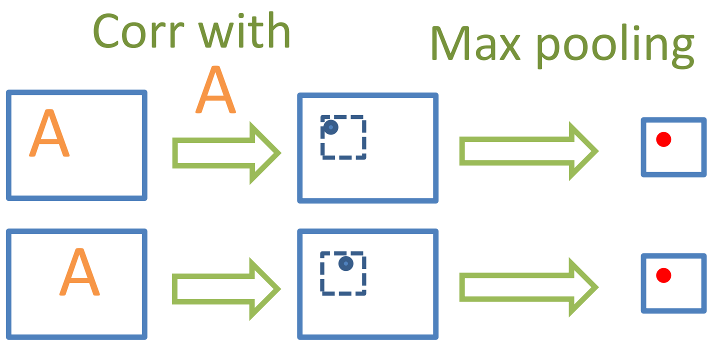
$$\boldsymbol{y}^{(l)} = \frac{1}{poolingsize}\mathrm{downsample}(\boldsymbol{y}^{(l-1)})$$

- Divide the convolved features into *disjoint* $m \times n$ regions, and take the mean (or maximum) feature activation over these regions
- Similar operations on 1D input

How about 3D input?

Channel-wise pooling

Can pooling model the function of complex cells?



This neuron is insensitive to the position of “A”

Simple cells: feature detector → Convolution

complex cells: translation invariant → Pooling

- Other advantages of max pooling?

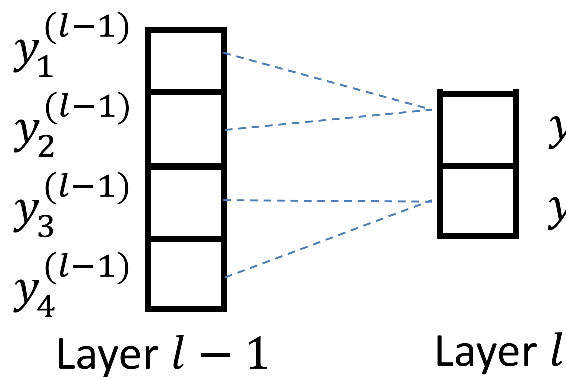
Other advantages

- Reduce the number of features for final classification
  - Consider images of 96×96 pixels. Suppose we have learned 400 features over 8×8 inputs. This results in an output of size $(96 - 8 + 1)^2 \times 400 = 3{,}168{,}400$ features per example
- Enlarge the effective region of features in the next layer
  - A feature learned in the pooled maps will have larger effective regions in the pixel space

This is similar to the receptive fields of visual neurons, whose sizes increase along the visual hierarchy

Average pooling layer

If layer l is an average pooling layer. Consider one single feature map



$$y_1^{(l)} = (y_1^{(l-1)} + y_2^{(l-1)})/2$$
$$y_2^{(l)} = (y_3^{(l-1)} + y_4^{(l-1)})/2$$

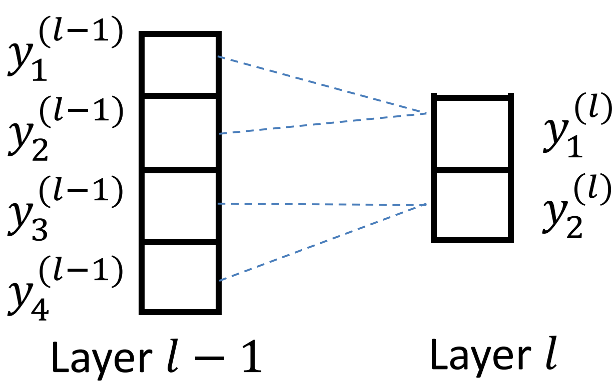
- Local sensitivity in the scalar form

$$\delta_1^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_1^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial y_1^{(l-1)}} = \frac{1}{2}\delta_1^{(l)}$$

$$\delta_2^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_2^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial y_2^{(l-1)}} = \frac{1}{2}\delta_1^{(l)}$$

Similarly we can obtain $\delta_3^{(l-1)} = \frac{1}{2}\delta_2^{(l)}, \quad \delta_4^{(l-1)} = \frac{1}{2}\delta_2^{(l)}$

Average pooling layer



$$\delta_1^{(l-1)} = \delta_1^{(l)}/2$$
$$\delta_2^{(l-1)} = \delta_1^{(l)}/2$$
$$\delta_3^{(l-1)} = \delta_2^{(l)}/2$$
$$\delta_4^{(l-1)} = \delta_2^{(l)}/2$$

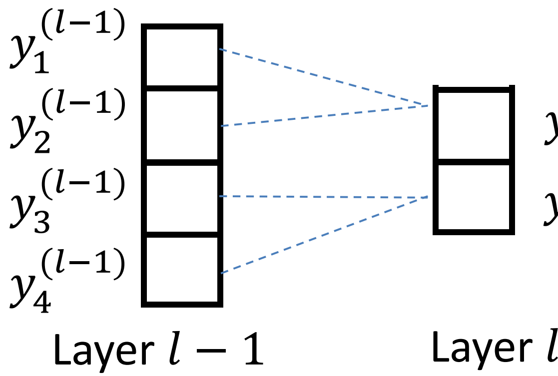
- In general, local sensitivity in the vector form

$$\boldsymbol{\delta}^{(l-1)} = \frac{1}{poolingsize}\mathrm{upsample}(\boldsymbol{\delta}^{(l)})$$

$$\mathrm{upsample}(\boldsymbol{a}) \triangleq \begin{pmatrix} \left.\begin{matrix} a_1 \\ a_1 \end{matrix}\right\} \text{Poolingsize} \\ \hline \vdots \\ \hline \left.\begin{matrix} a_n \\ a_n \end{matrix}\right\} \text{Poolingsize} \end{pmatrix} \qquad \text{where} \quad \boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

Max pooling layer

If layer l is a max pooling layer. Consider one single feature map



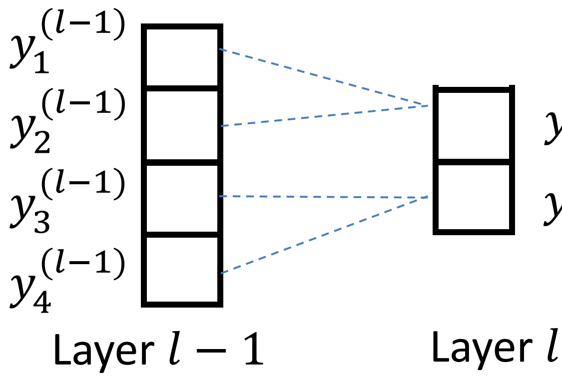
$$y_1^{(l)} = \max\{y_1^{(l-1)}, y_2^{(l-1)}\}$$
$$y_2^{(l)} = \max\{y_3^{(l-1)}, y_4^{(l-1)}\}$$

- What are $\delta_i^{(l-1)}$ for $i = 1, \ldots, 4$?

The solutions are different for different values of $y_i^{(l-1)}$

Max pooling layer

If layer l is a max pooling layer. Consider one single feature map



$$y_1^{(l)} = \max\{y_1^{(l-1)}, y_2^{(l-1)}\}$$
$$y_2^{(l)} = \max\{y_3^{(l-1)}, y_4^{(l-1)}\}$$

- If $y_1^{(l-1)} \geq y_2^{(l-1)}$,

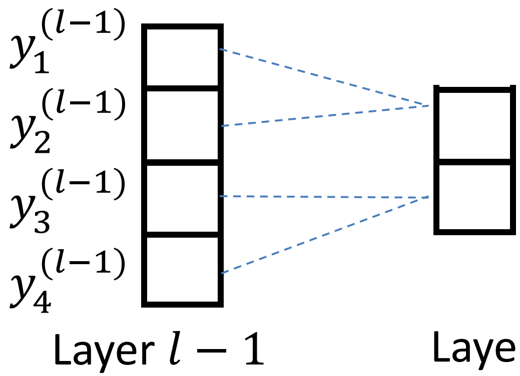
$$\delta_1^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_1^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial y_1^{(l-1)}} = \delta_1^{(l)}, \qquad \delta_2^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_2^{(l-1)}} = 0.$$

- Else

$$\delta_1^{(l-1)} = 0, \qquad \delta_2^{(l-1)} = \delta_1^{(l)}.$$

Max pooling layer

If layer l is a max pooling layer. Consider one single feature map



$$y_1^{(l)} = \max\{y_1^{(l-1)}, y_2^{(l-1)}\}$$
$$y_2^{(l)} = \max\{y_3^{(l-1)}, y_4^{(l-1)}\}$$

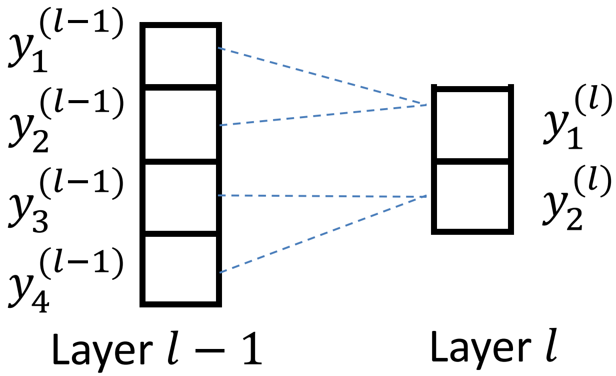
- If $y_3^{(l-1)} \geq y_4^{(l-1)}$,

$$\delta_3^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_2^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_2^{(l)}} \frac{\partial y_2^{(l)}}{\partial y_3^{(l-1)}} = \delta_2^{(l)}, \qquad \delta_4^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_4^{(l-1)}} = 0.$$

- Else

$$\delta_3^{(l-1)} = 0, \qquad \delta_4^{(l-1)} = \delta_2^{(l)}.$$

Max pooling layer



Suppose $y_1^{(l-1)} \geq y_2^{(l-1)}$ and $y_3^{(l-1)} \geq y_4^{(l-1)}$

$$\delta_1^{(l-1)} = \delta_1^{(l)}, \quad \delta_2^{(l-1)} = 0.$$

$$\delta_3^{(l-1)} = \delta_2^{(l)}, \quad \delta_4^{(l-1)} = 0.$$

- In general, local sensitivity in the vector form

$$\boldsymbol{\delta}^{(l-1)} = \Gamma(\boldsymbol{y}^{(l-1)}) \odot \mathrm{upsample}(\boldsymbol{\delta}^{(l)}),$$

$$\text{where } \Gamma(\boldsymbol{y}^{(l-1)}) = \left(\begin{array}{c} 1 \\ 0 \\ \hline \vdots \\ \hline 1 \\ 0 \end{array}\right) \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} \textit{Poolingsize} \\ \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} \textit{Poolingsize} \end{array}$$

In each pooling region of $\boldsymbol{y}^{(l-1)}$, the location with max elements is 1 and other locations are 0

2D Pooling layers

- Forward pass

$$\boldsymbol{y}^{(l)} = \frac{1}{poolingsize}\mathrm{downsample}(\boldsymbol{y}^{(l-1)})$$

- Backward pass
  - Average pooling:

$$\boldsymbol{\delta}^{(l-1)} = \frac{1}{poolingsize}\mathrm{upsample}(\boldsymbol{\delta}^{(l)})$$

  - Max pooling:

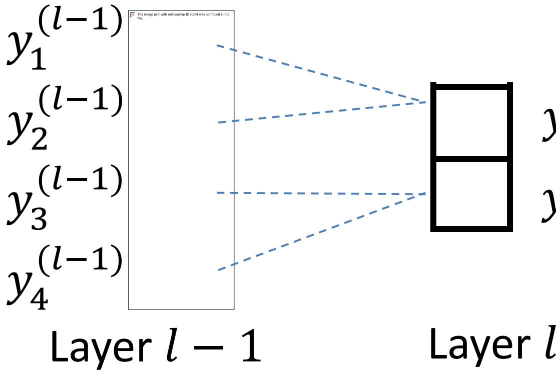
$$\boldsymbol{\delta}^{(l-1)} = \Gamma(\boldsymbol{y}^{(l-1)}) \odot \mathrm{upsample}(\boldsymbol{\delta}^{(l)})$$

$$\mathrm{upsample}(\boldsymbol{a}) \triangleq \left(\begin{array}{cc|c|cc} a_{11} & a_{11} & \cdots & a_{1m} & a_{1m} \\ a_{11} & a_{11} & \cdots & a_{1m} & a_{1m} \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline a_{n1} & a_{n1} & \cdots & a_{nm} & a_{nm} \\ a_{n1} & a_{n1} & \cdots & a_{nm} & a_{nm} \end{array}\right)$$

Only one element=1

$$\Gamma(\boldsymbol{c}) = \left(\begin{array}{cc|c|cc} 0 & 1 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{array}\right) \in R^{r\times s}$$

where $\boldsymbol{a} \in R^{n\times m}$, $\boldsymbol{c} \in R^{r\times s}$, Setting



$$y_1^{(l)} = \max\{y_1^{(l-1)}, y_2^{(l-1)}\}$$
$$y_2^{(l)} = \max\{y_3^{(l-1)}, y_4^{(l-1)}\}$$

Suppose $y_1^{(l-1)} \geq y_2^{(l-1)}$ and $y_3^{(l-1)} \leq y_4^{(l-1)}$. Which is (are) correct?

A $\delta_1^{(l-1)} = \delta_2^{(l)}$

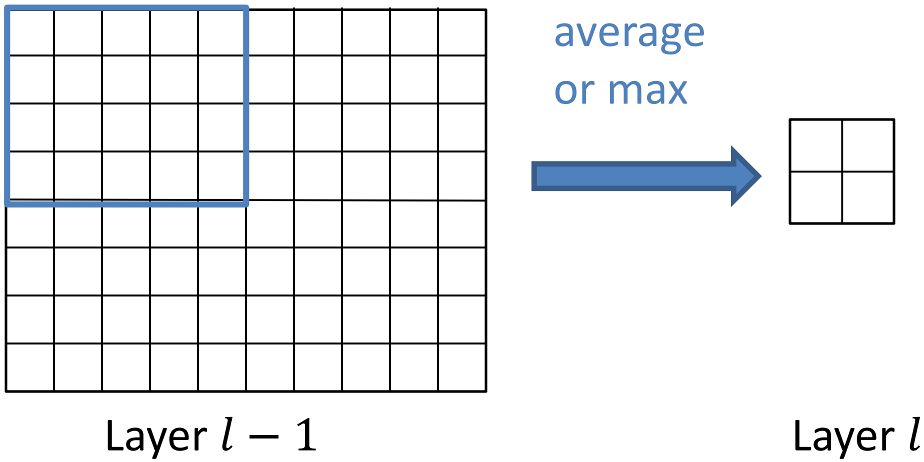
B $\delta_2^{(l-1)} = \delta_1^{(l)}$

C $\delta_3^{(l-1)} = 0$

D $\delta_4^{(l-1)} = \delta_2^{(l)}$

Submit

Summary of Part 1



- Realize translation invariance
- Reduce the number of features
- Enlarge RF

Forward pass

$$\boldsymbol{y}^{(l)} = \frac{1}{poolingsize}\mathrm{downsample}(\boldsymbol{y}^{(l-1)})$$

Backward pass

$$\boldsymbol{\delta}^{(l-1)} = \frac{1}{poolingsize}\mathrm{upsample}(\boldsymbol{\delta}^{(l)})$$

$$\boldsymbol{\delta}^{(l-1)} = \Gamma(\boldsymbol{y}^{(l-1)}) \odot \mathrm{upsample}(\boldsymbol{\delta}^{(l)})$$

Outline

1. Pooling
2. Standard CNN
3. Typical CNNs
4. Training techniques-II
5. Summary

Construction of CNN

- The convolutional layers and pooling layers can be combined freely with other layers that we have discussed
  - Fully connected layer
  - Sigmoid layer, ReLU layer or other activation layers
  - Euclidean loss layer
  - Cross-entropy loss layer

  as well as other layers that we haven't discussed, e.g.,
  - Local response normalization layer (Krizhevsky et al. 2012)
  - Dropout layer (Srivastava et al., 2014)
  - Batch normalization layer (Ioffe and Szegedy, 2015)

CNN Implementation

- Implement each *type* of layer as a class and provide functions for forward calculation and backward calculation, respectively
- Design a CNN structure by specifying layer modules in a main file
- Run forward process
  - Calculate the output $\boldsymbol{y}^{(l)}$ for $l = 1,2, \dots, L$
- Run backward process
  - Calculate $\partial E/\partial \boldsymbol{W}^{(l)}$ and $\partial E/\partial \boldsymbol{b}^{(l)}$ if any, and $\boldsymbol{\delta}^{(l)}$ for $l = L, L-1, \dots, 1$
- Update $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ for $l = 1,2, \dots, L$

Extensions

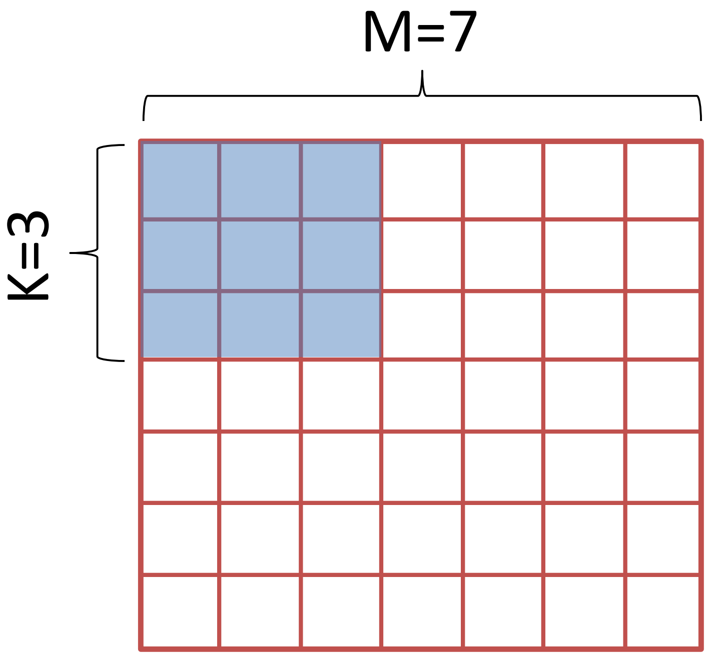
a) Preserving the spatial size with “same” mode convolution

In many DL toolbox, there is no “same” mode for convolution; all convolution has just one mode: “valid”

b) Convolution with stride$\neq$1

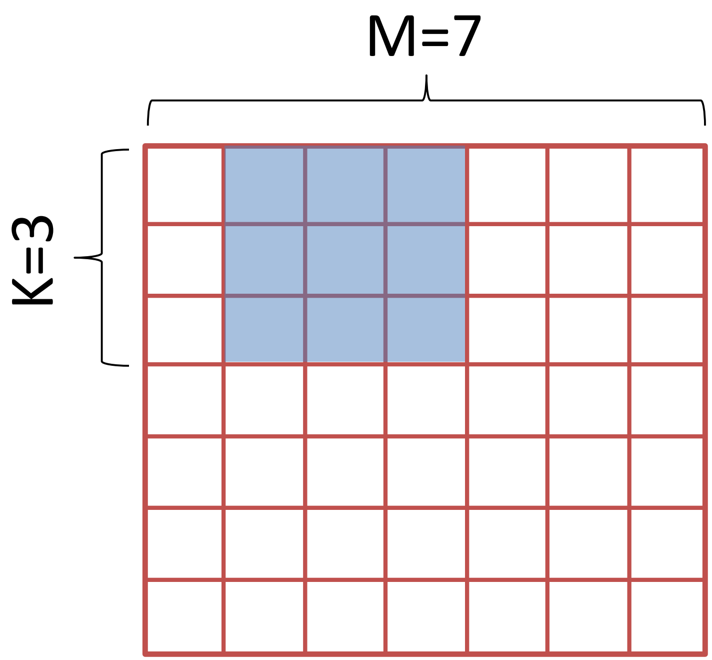
c) Polling with stride $\neq$ poolingsize

a) Preserving the spatial size



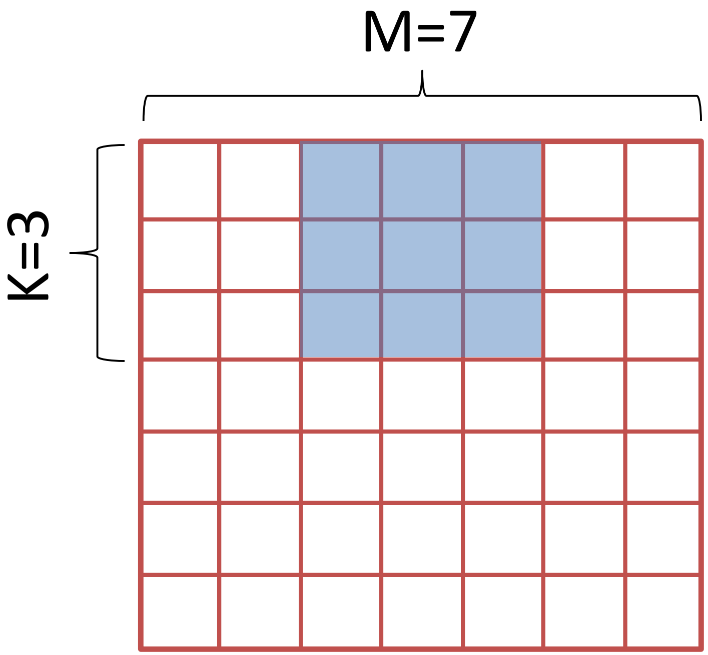
- Input size M=7x7
- Kernel size K=3x3
- Stride=1
- Output size (valid mode):

a) Preserving the spatial size



- Input size M=7x7
- Kernel size K=3x3
- Stride=1
- Output size (valid mode):

a) Preserving the spatial size



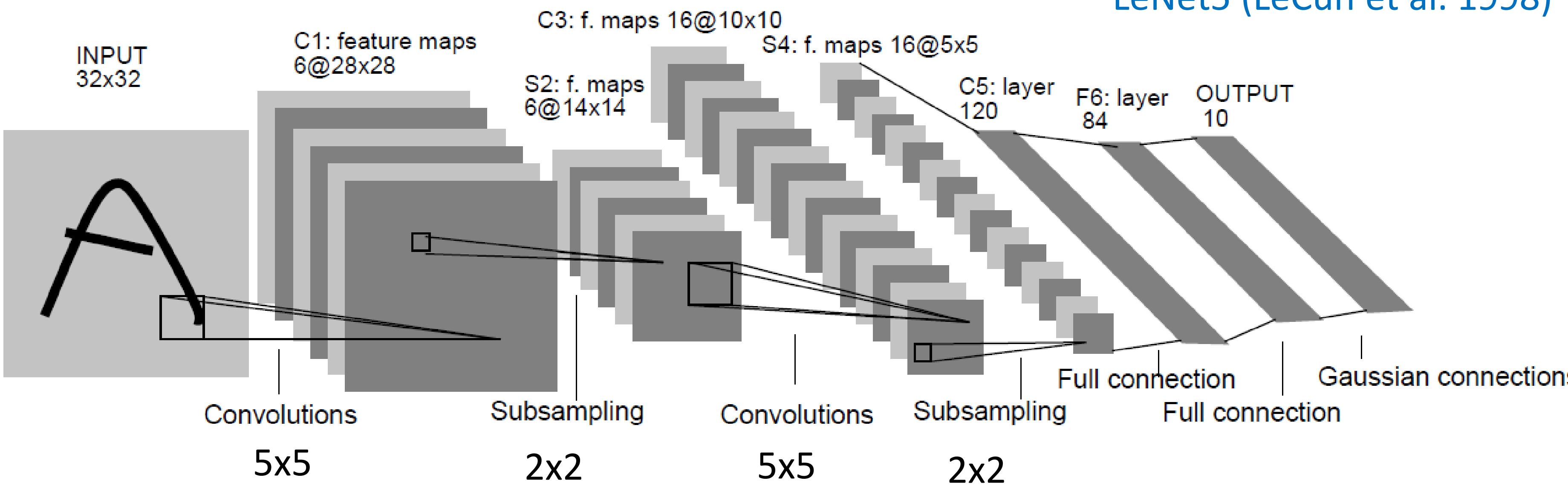
- Input size M=7x7
- Kernel size K=3x3
- Stride=1
- Output size (valid mode): 5x5

The input is shrunk

a) Preserving the spatial size

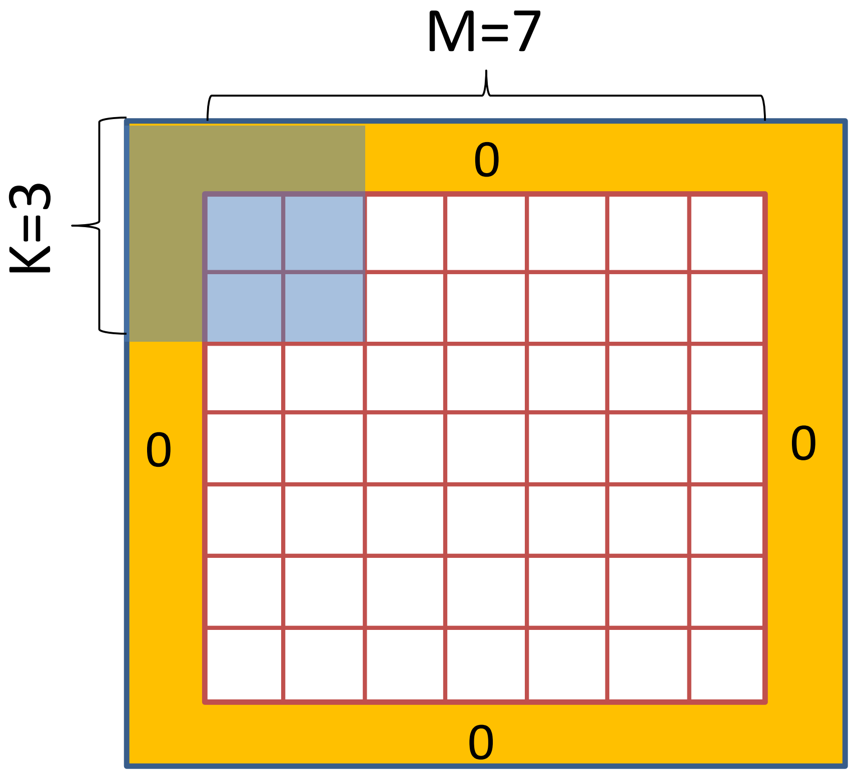
This is the case in LeNet 5

LeNet5 (LeCun et al. 1998)



If we don't want to shrink the input, what shall we do?

a) Preserving the spatial size



- Input size M=7x7
- Kernel size K=3x3
- Stride=1
- Pad with 1 pixel border
- Output size: 7x7

The "same" mode conv

- Usually, K is odd
- To keep the output size the same as input size, with stride=1, what is the pad size (on each side)?

单选题 1分

Setting

- Input size MxM. Kernel size KxK. Stride=1.
- If K is odd. To keep the output size the same as the input size, what is the pad size (on each side)?

A (M-K)/4

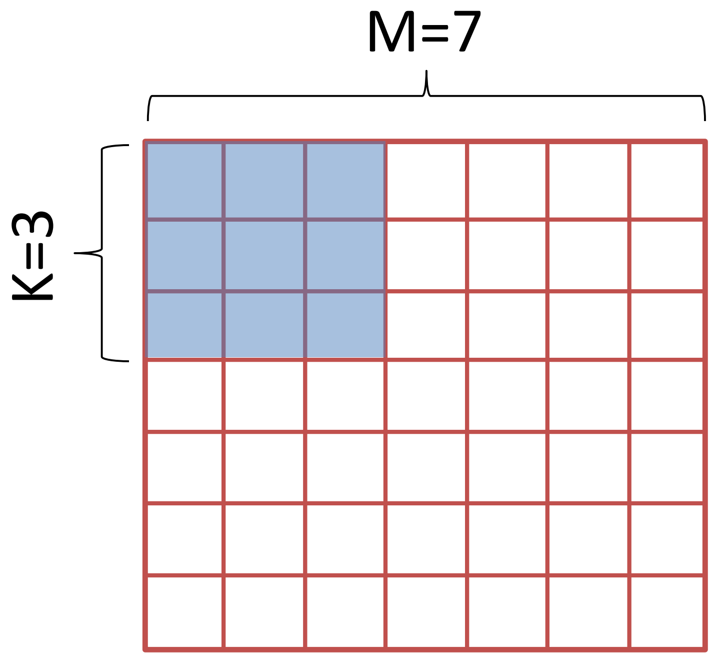
B (K+1)/2

C (K-1)/2

D (M-1)/2

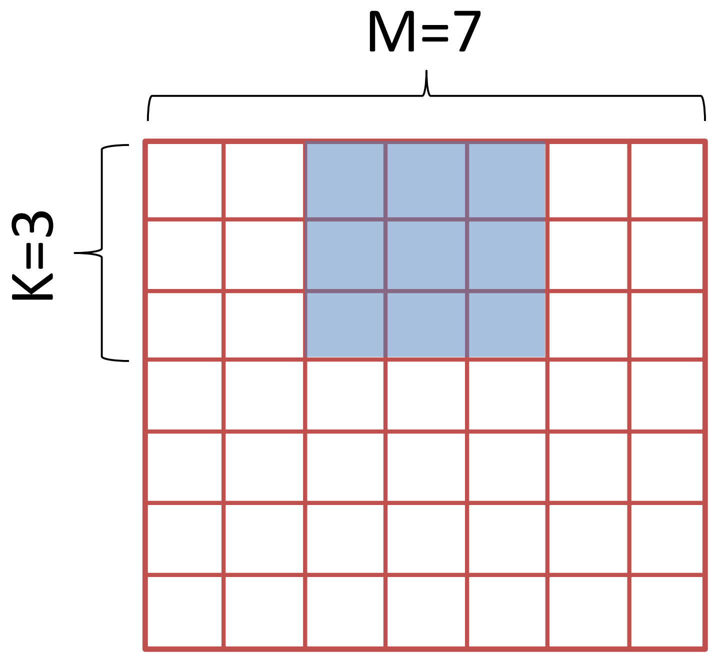
Submit

b) Convolution with stride≠1



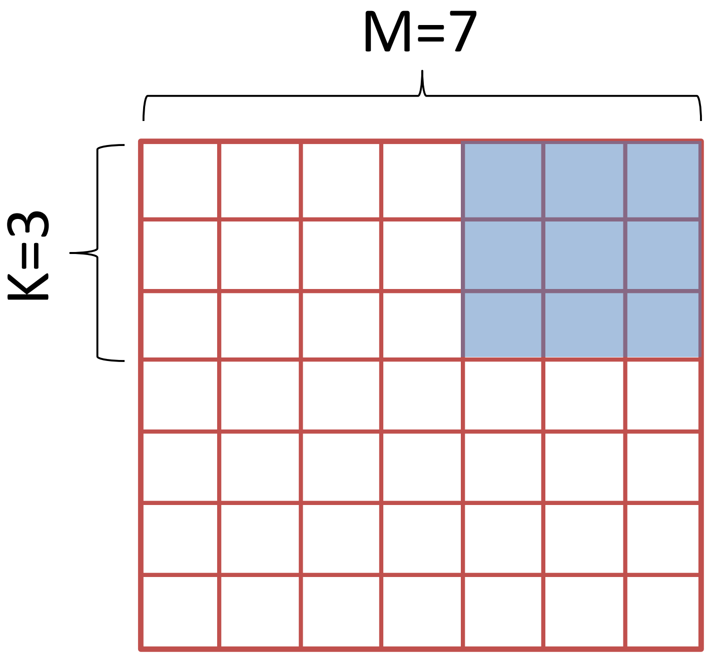
- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size (valid mode):

b) Convolution with stride≠1



- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size (valid mode):

b) Convolution with stride≠1

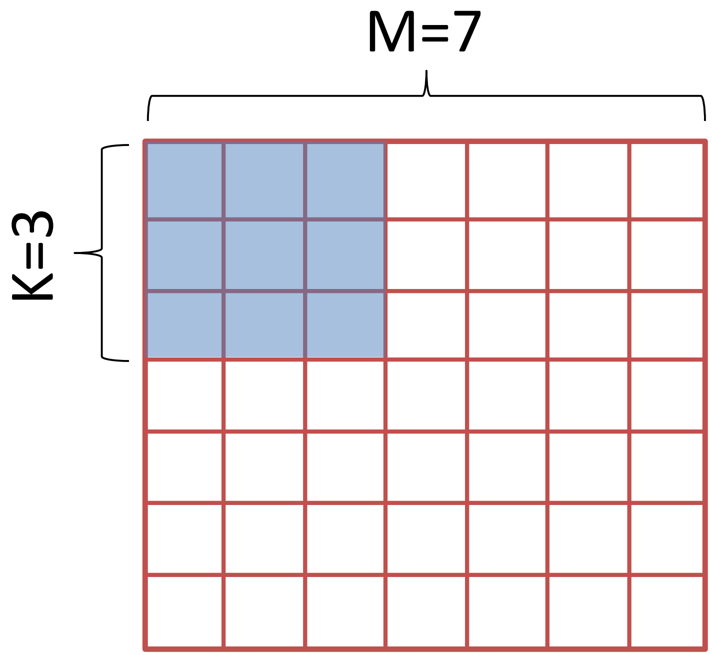


- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size (valid mode): 3x3

In general, output size: (M-K)/stride+1

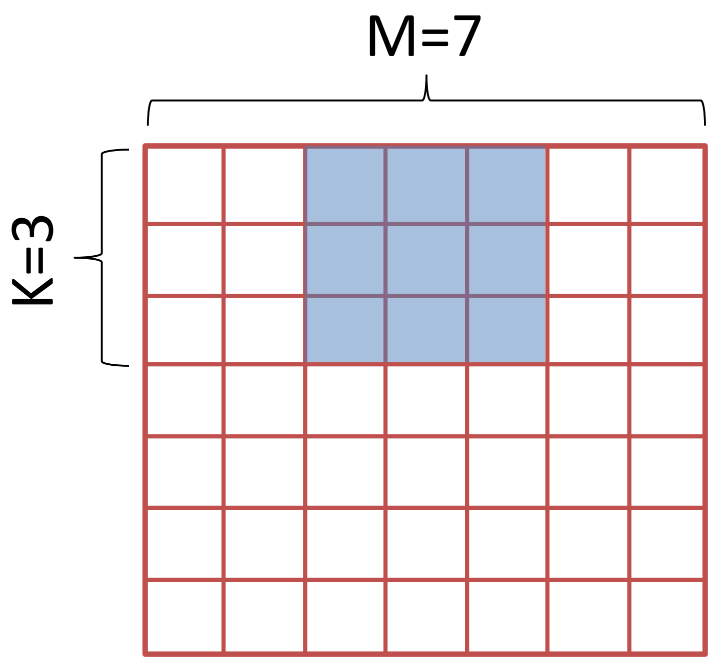
What if (M-K)/stride is not an integer?

c) Polling with stride ≠ poolingsize



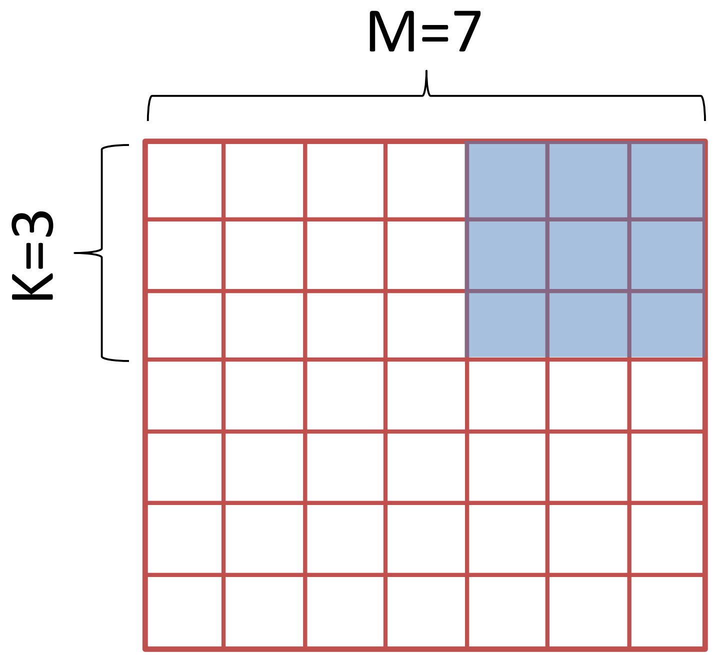
- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size:

c) Polling with stride ≠ poolingsize



- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size:

c) Polling with stride ≠ poolingsize



- Input size M=7x7
- Kernel size K=3x3
- Stride=2
- Output size: 3x3

In general, output size: (M-K)/stride+1

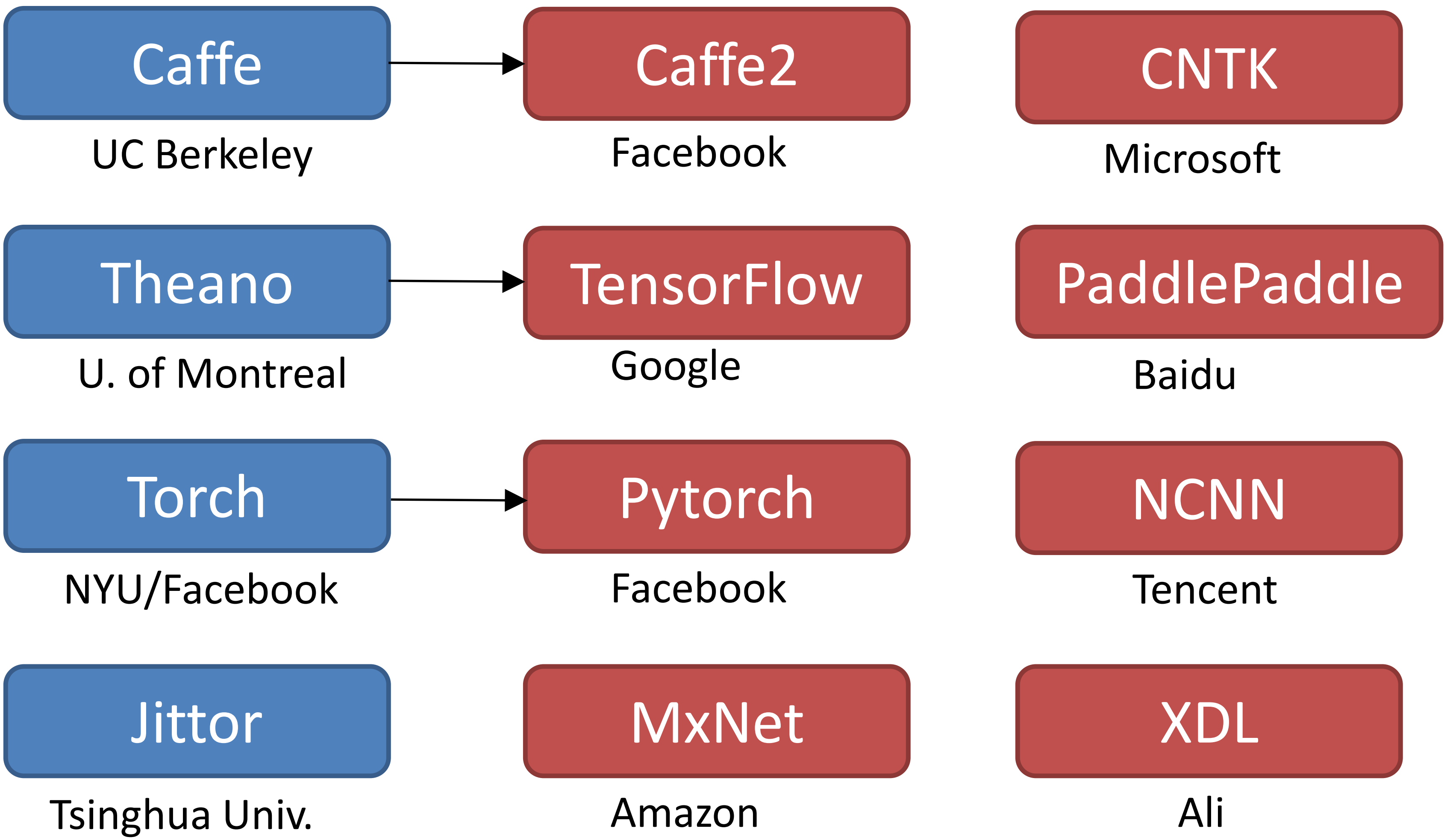
What if (M-K)/stride is not an integer?

Extensions

a) Preserving the spatial size with “same” mode convolution
b) Convolution with stride$\neq$1
c) Polling with stride $\neq$ poolingsize

What are the backward calculations in these cases?

Frameworks



Caffe
UC Berkeley
Caffe2
Facebook
CNTK
Microsoft
Theano
U. of Montreal
TensorFlow
Google
PaddlePaddle
Baidu
Torch
NYU/Facebook
Pytorch
Facebook
NCNN
Tencent
Jittor
Tsinghua Univ.
MxNet
Amazon
XDL
Ali

Demo: MNIST classification

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

MNIST

- 60,000 training images and 10,000 test images
- 28x28 black and white images

Network setting

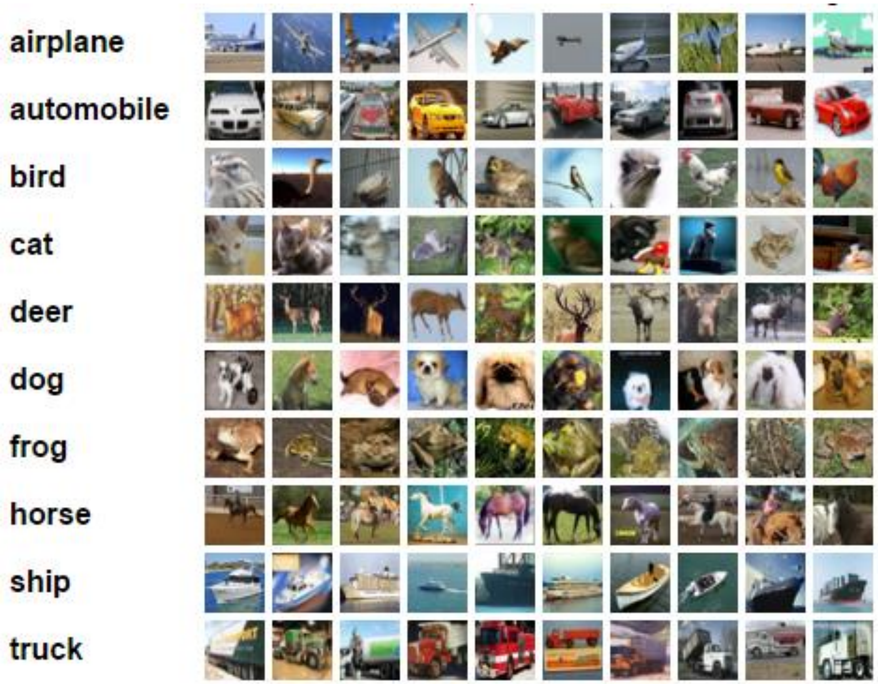
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24,
out_sy:24, out_depth:1});
layer_defs.push({type:'conv', sx:5, filters:8,
stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:16,
stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:3, stride:3});
layer_defs.push({type:'softmax',
num_classes:10});
```

Demo: CIFAR-10 classification

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

CIFAR-10 & CIFAR100

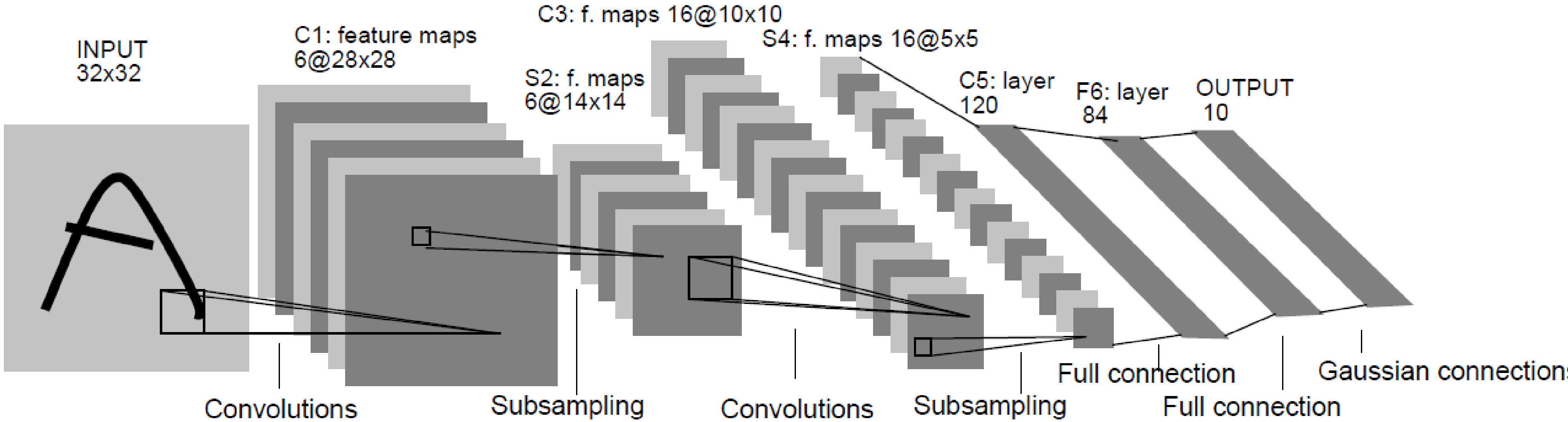
- 50,000 training images and 10,000 test images
- 32x32 colour images




```
layer_defs = [];
layer_defs.push({type:'input', out_sx:32, out_sy:32, out_depth:3});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'softmax', num_classes:10});
```

Summary of part 2

Standard CNN

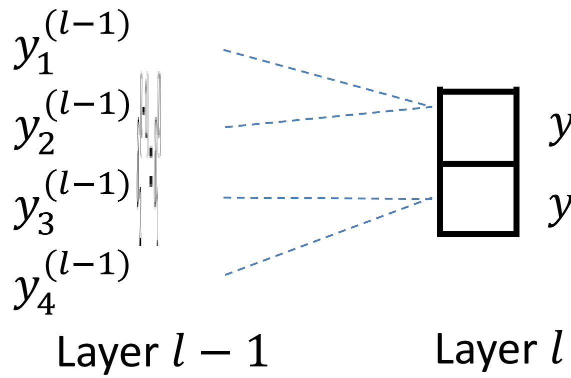


Extensions

Padding for “same” mode conv

Conv with stride ≠ 1

Polling with stride ≠ poolingsize



$$y_1^{(l)} = \max\{y_1^{(l-1)}, y_2^{(l-1)}\}$$
$$y_2^{(l)} = \max\{y_3^{(l-1)}, y_4^{(l-1)}\}$$

Suppose $y_1^{(l-1)} \geq y_2^{(l-1)}$ and $y_3^{(l-1)} \geq y_4^{(l-1)}$. Which is (are) correct?

A $\delta_1^{(l-1)} = \delta_2^{(l)}$

B $\delta_2^{(l-1)} = \delta_1^{(l)}$

C $\delta_3^{(l-1)} = \delta_2^{(l)}$

D $\delta_4^{(l-1)} = 0$

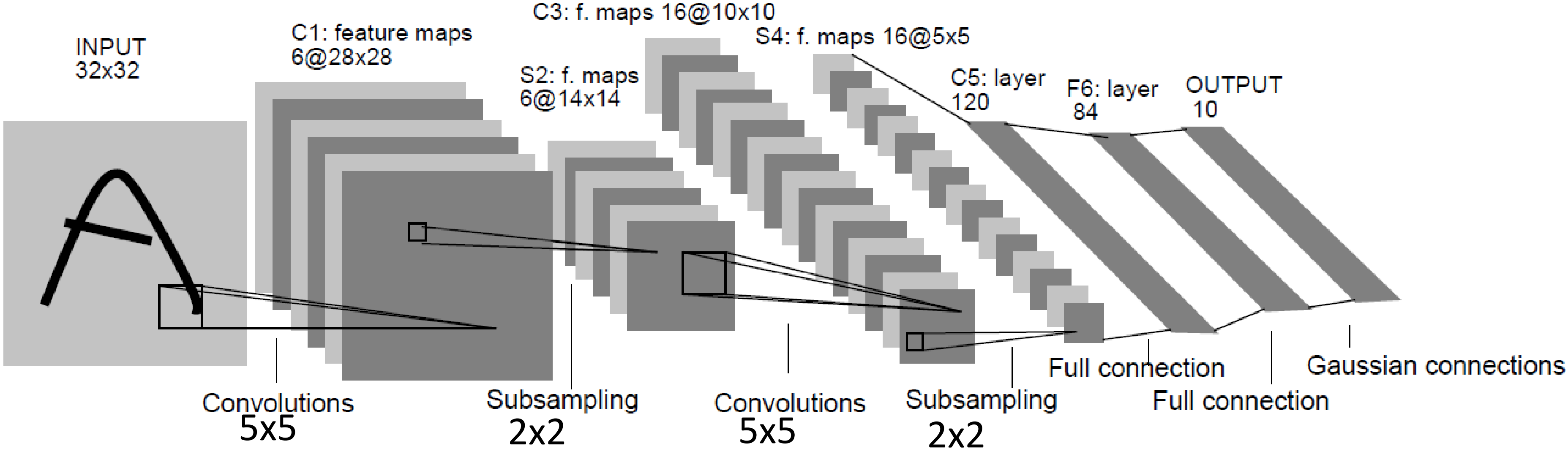
Submit

Outline

1. Pooling
2. Standard CNN
3. Typical CNNs
4. Training techniques-II
5. Summary

LeNet-5

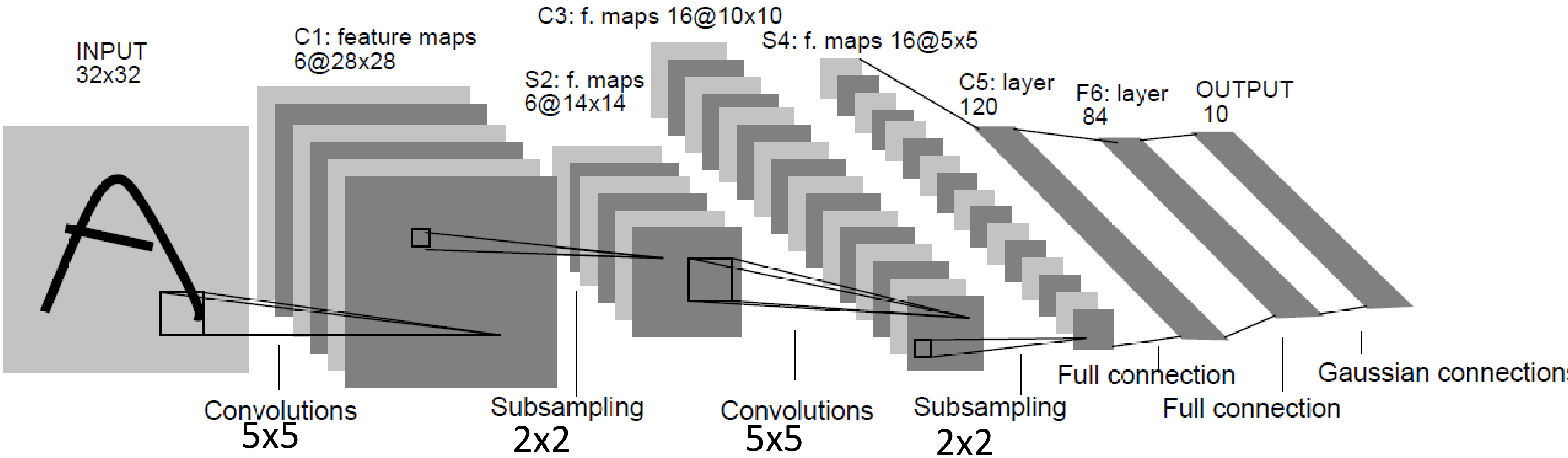
LeCun, Bottou, Bengio, Haffner, 1998



- C layers: convolution
  - Output $y_i = f(\sum_\Omega w_j x_j + b)$ where Ω is the patch size, is the sigmoid function, w and b are parameters $f(\cdot)$
- S layers: subsampling (avg pooling)
  - Output $y_i = f(w \sum_\Omega x_j + b)$ where Ω is the pooling size

LeNet-5

LeCun, Bottou, Bengio, Haffner, 1998



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

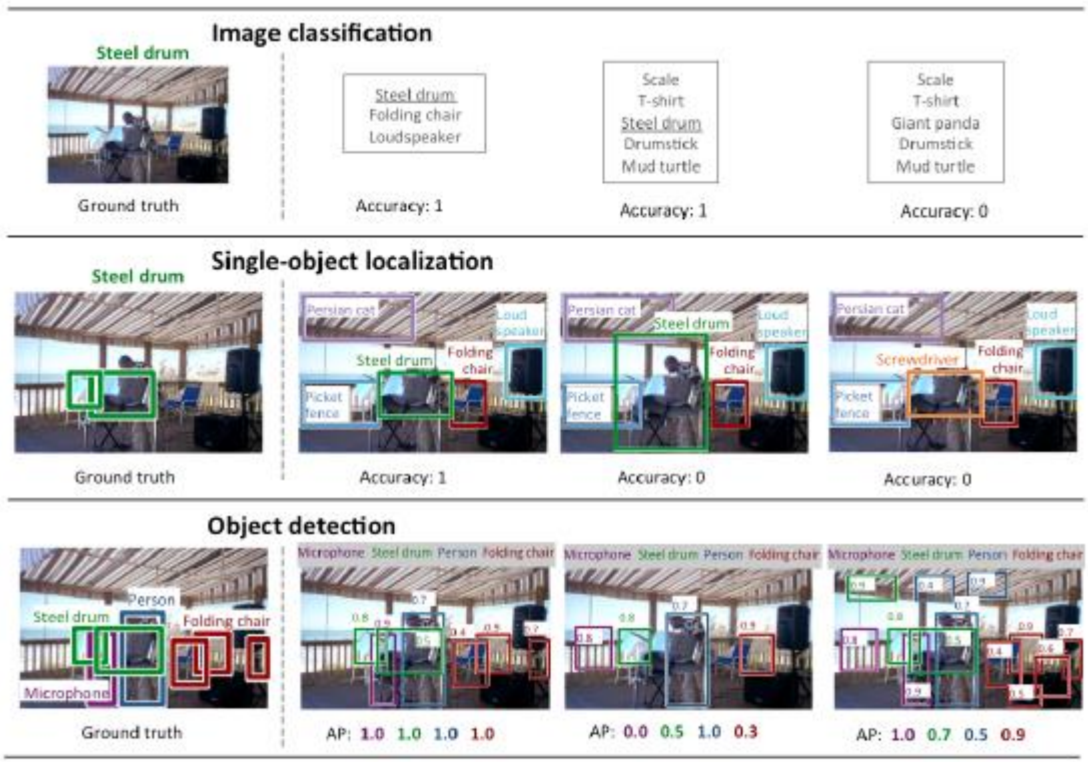
ImageNet comptition (ILSVRC)

Tasks

2010-

2011-

2013-



Top-1

Top-5 (preferred)

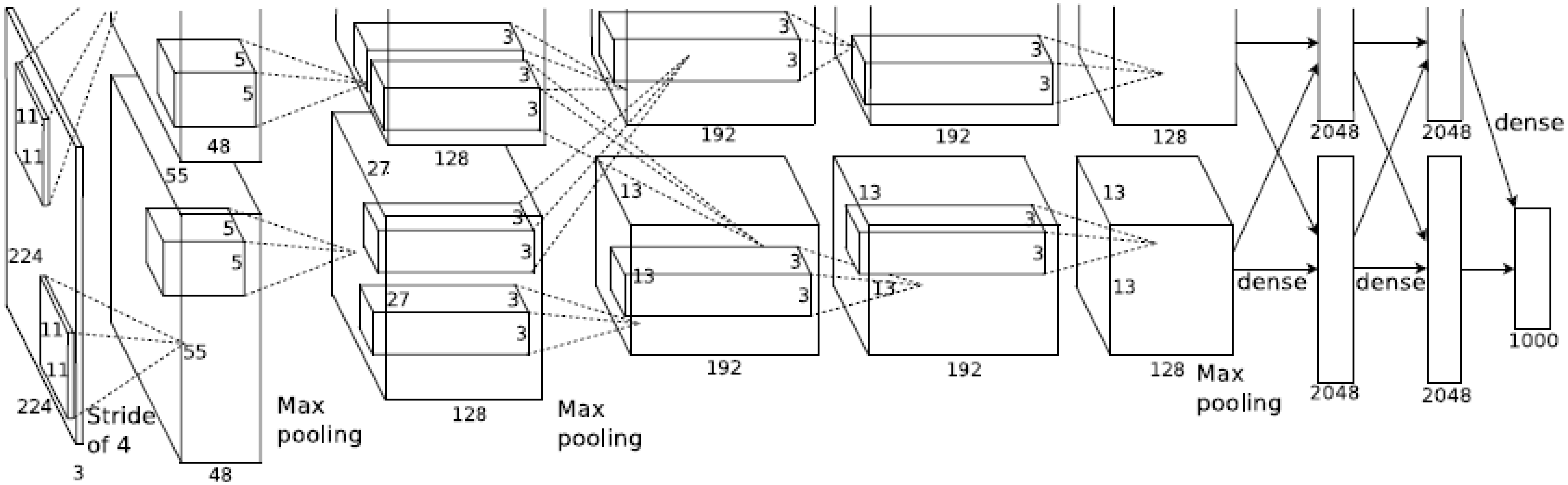
Two human expert: 5.1%, 12%

The first column shows the ground truth labeling on an example image, and the next three show three sample outputs with the corresponding evaluation score.

Russakovsky, et al., 2014

AlexNet

Krizhevsky, Sutskever and Hinton, NIPS, 2012



- Classification: 1000 classes, 1.2 million training images
- In total: 60 million parameters

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Since then, CNN dominates computer vision society

- In 2013, the vast majority of teams used CNN.
- In 2014 & 2015, almost all teams used convolutional neural networks.

VGG net

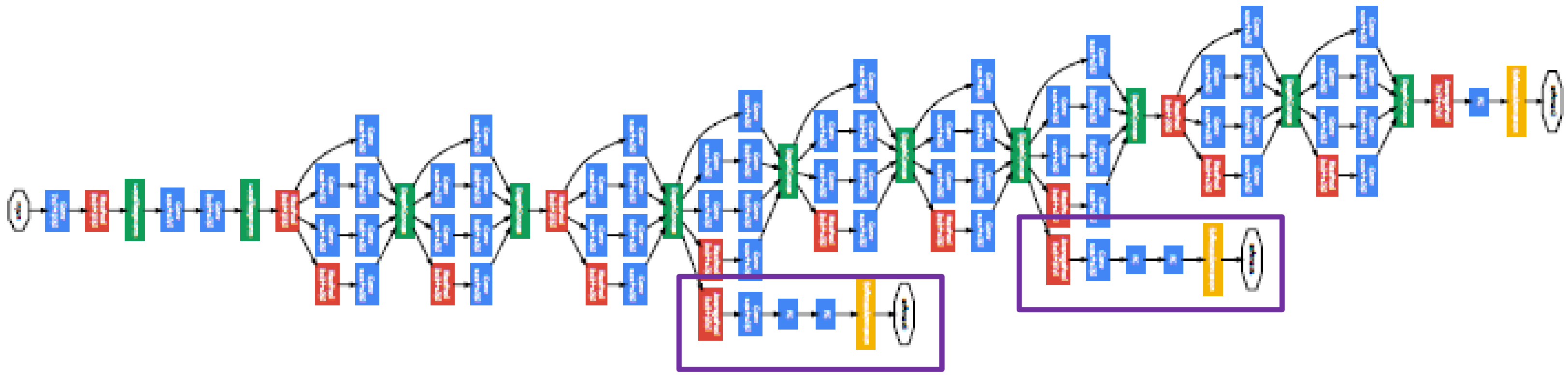
Simonyan, Zisserman, 2015

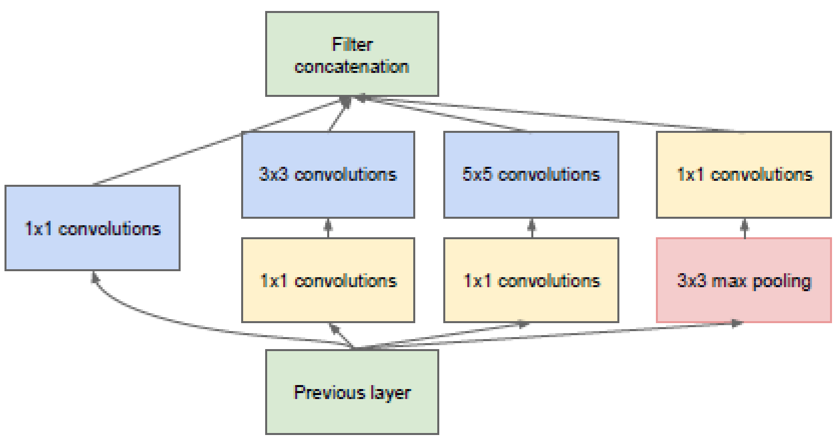
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- 3*3 filters are extensively used
- GPU implementation

GoogLeNet (Inception-v1)

Szegedy, et al., 2014





- Multiple sizes in the same layer
- 1×1 conv are used to reduce the number of channels

- 22 weight layers
- Small filters (1x1, 3x3, 5x5)
- Two auxiliary classifiers connected to intermediate layers are used to increase the gradient signal for BP algorithm
- A cpu-based implementation on distributed system

Extensions

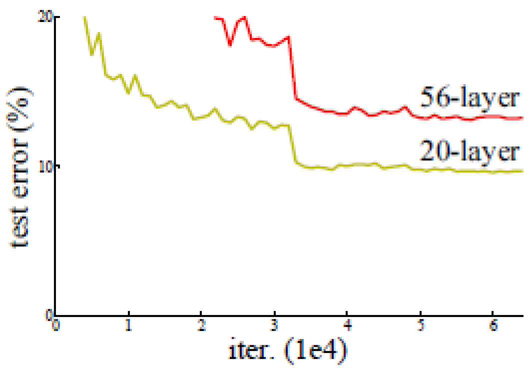
- There are extensions to this model
  - Inception-v2
  - Inception-v3

Szegedy, Vanhoucke, Ioffe et al., Rethinking the Inception Architecture for Computer Vision, CVPR, 2016

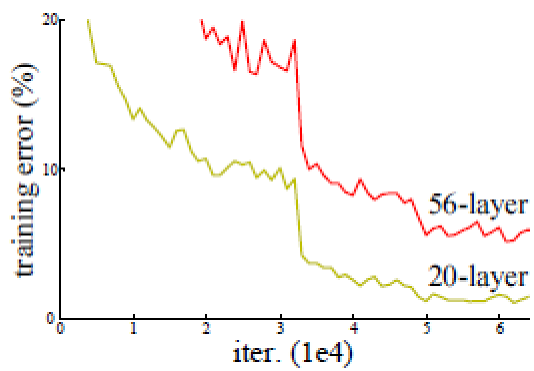
More layers, better results?

	Weight layers	Top-5 error rate
AlexNet	8	17.0%
VggNet	19	7.5%
GoogLeNet	22	6.67%

More layers, better results?



Is it caused by over-fitting?

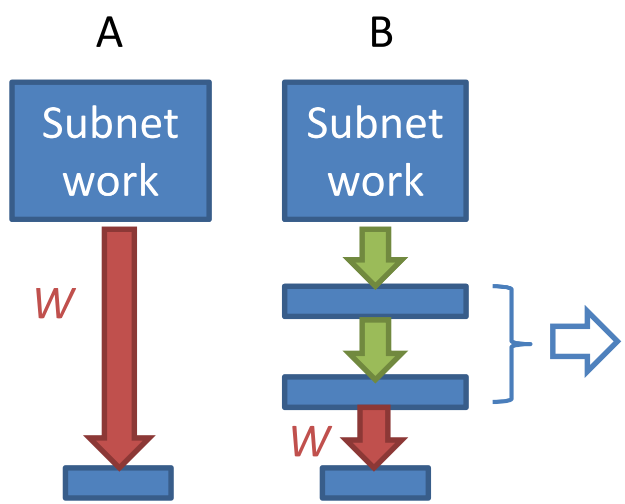


He et al., 2016

Deep residual network (ResNet)

He et al., 2016

Consider two models



Error of B should not be larger than that of A!

- If they are identity mappings, then A and B are equivalent
- If they include identity mapping as special cases, the capacity of B is larger than A

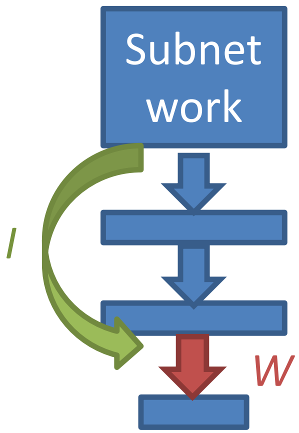
The assumptions may not hold

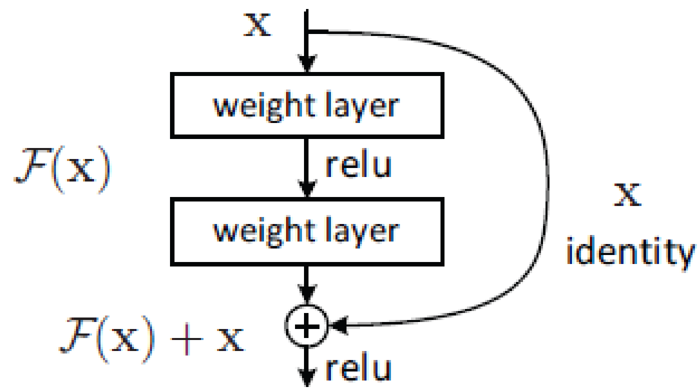
It might be difficult for nonlinear layers to approx. the identity mapping

Deep residual network (ResNet)

He et al., 2016

- If this is the case, let's explicitly use the identity mapping





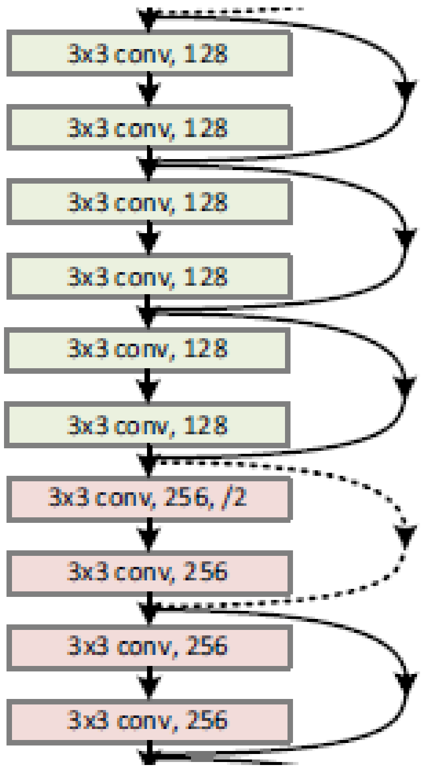
- The nonlinear mapping from input to output $H(x)$ has two parts

$$H(x) = F(x) + x$$

- Then the (two) weight layers are learning $F(x)$, that is the residual $H(x) - x$

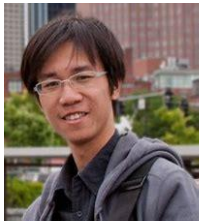
Deep residual network (ResNet)

He et al., 2016



A 152-layer network achieves 3.57% error rate

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

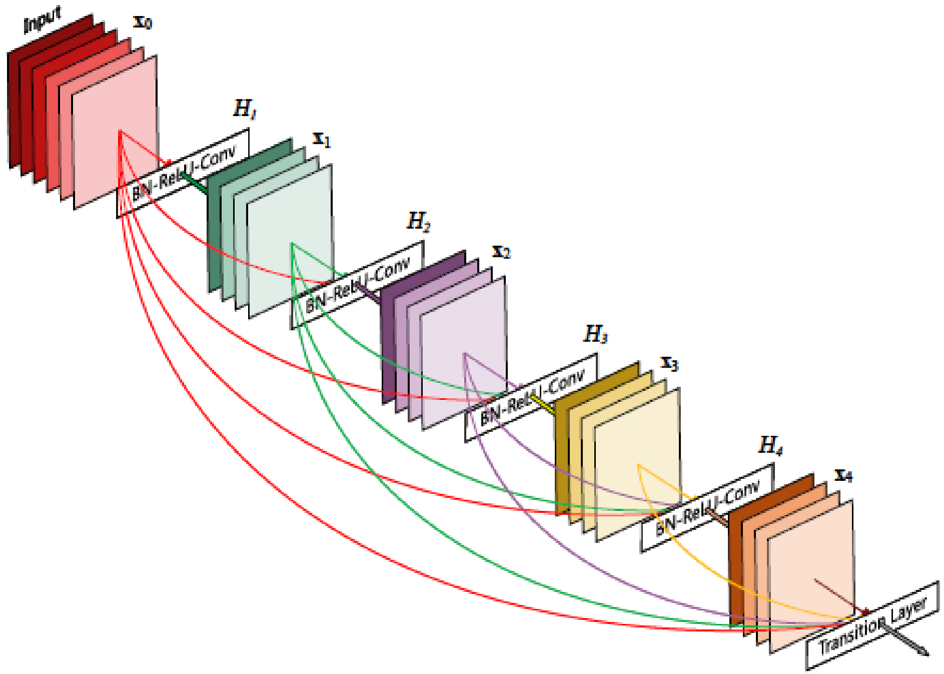


Kaiming He

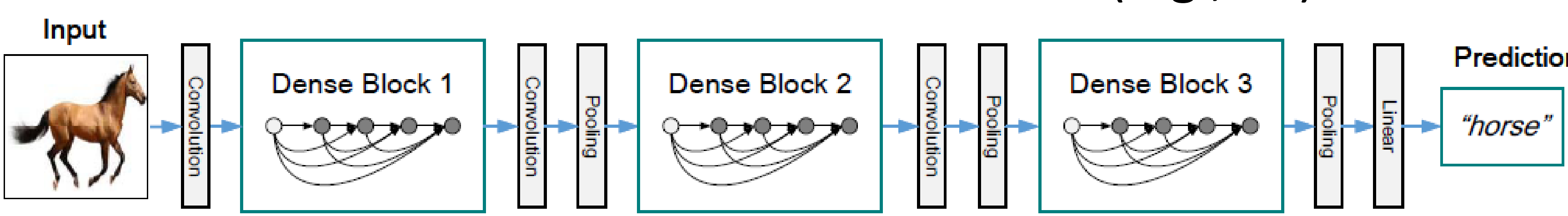
Best CVPR 2016 paper

DenseNet

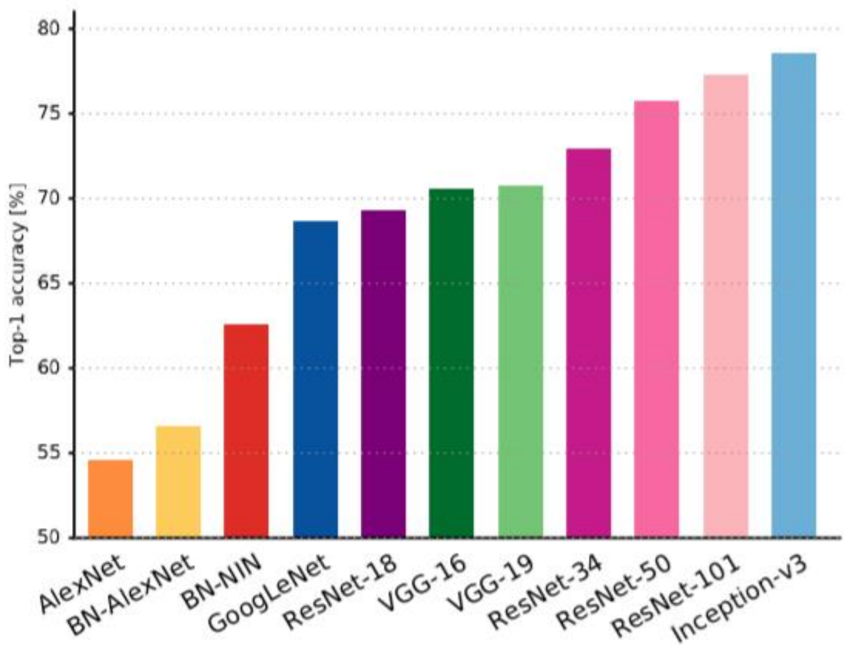
Huang et al., CVPR 2017

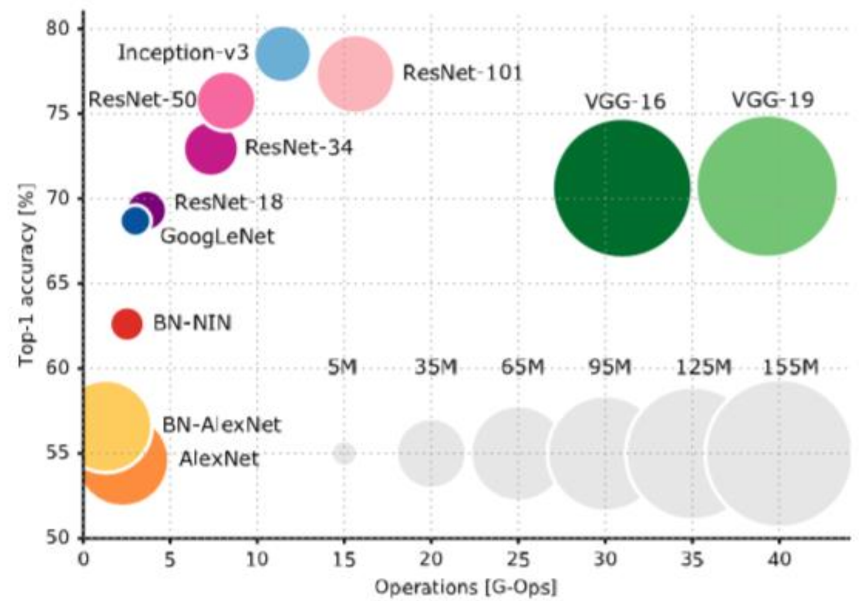


- Each layer takes all preceding feature-maps as input, which are *concatenated* together
- An L-layer net has $\frac{L(L+1)}{2}$ connections
- Each layer outputs k feature maps and k is small (e.g., 12)



Results on ImageNet challenge

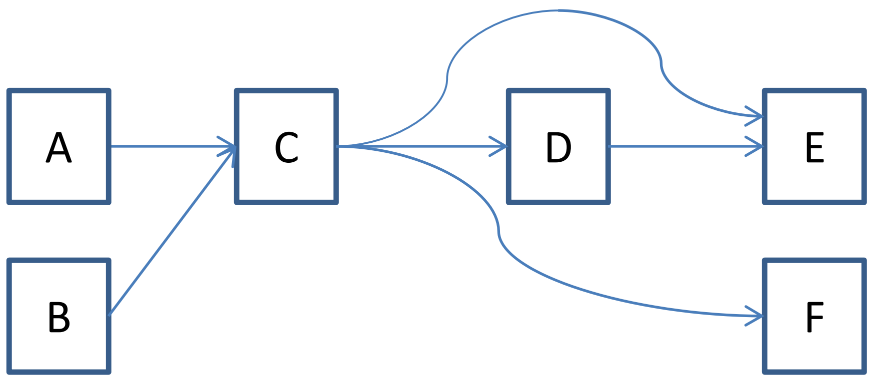




Alfredo Canziani, Adam Paszke, Eugenio Culurciello, arXiv:1605.07678v4, 2017

Calculation in complex architectures

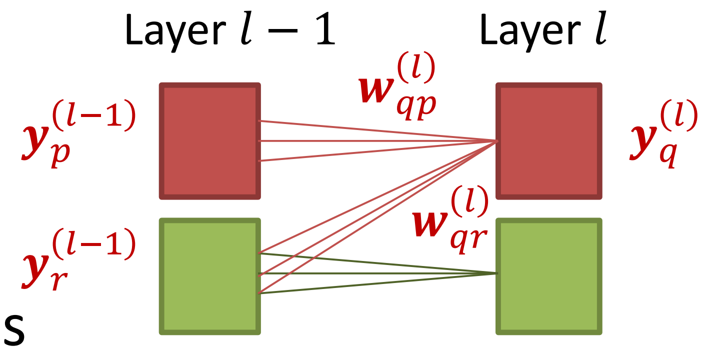
- We have seen many models having complex architectures



- Suppose the combinations use summation
- Feedforward calculation is straightforward
- During backward pass
  - What are needed to be calculated at each block?
  - And how?

Recap: 2D convolution with feature combination

- Suppose that the l-th layer is a convolutional layer



- Forward pass

$$\boldsymbol{y}_q^{(l)} = \sum_{p \in \mathcal{M}_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{w}_{qp}^{(l)}) + b_q^{(l)}$$

- Backward pass
  - Gradient:

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_{ij}$$

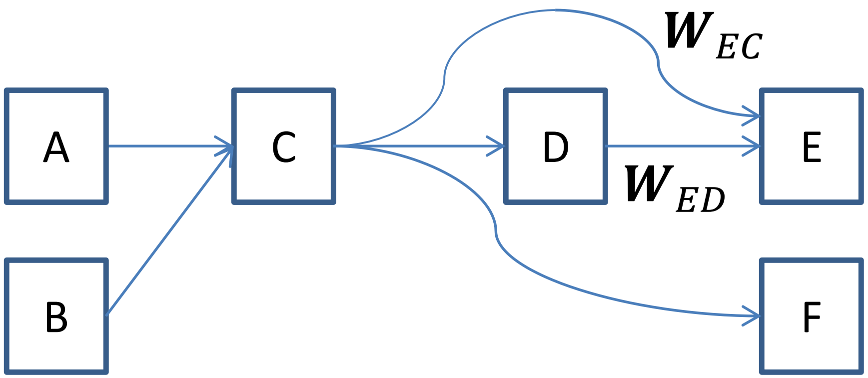
  - Local sensitivity:

$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \tilde{\mathcal{M}}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

($\mathcal{M}_q$ and $\widetilde{\mathcal{M}}_p$ are defined before)

多选题 1分

Setting



Suppose the combinations use summation. Which of the following are true?

A $\frac{\partial E^{(n)}}{\partial W_{EC}}$ depends on y_C and δ_E only

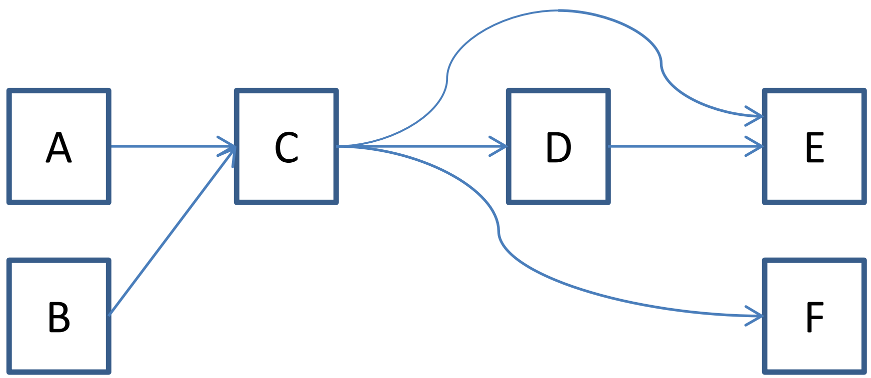
C δ_D depends on δ_E and W_{ED} only

B δ_C depends on δ_D and W_{DC} only

Submit

Calculation in complex architectures

- We have seen many models having complex architectures

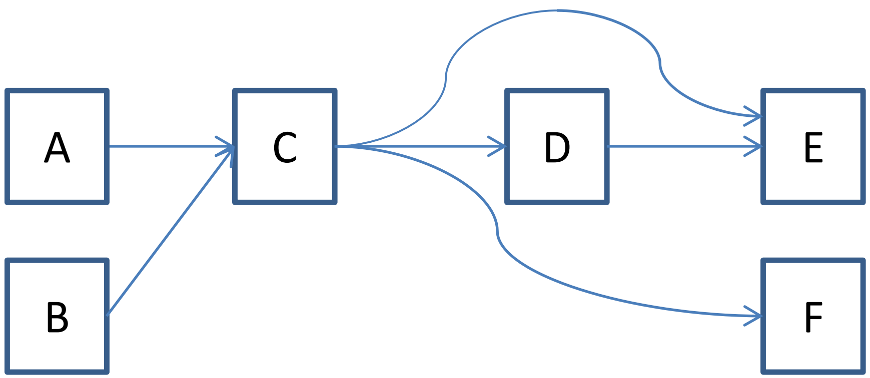


- Suppose the combinations use summation
- Feedforward calculation is straightforward
- During backward pass
  - What are needed to be calculated at each block?
  - And how?

During backward pass, $\delta^{(C)}$ can be only obtained after E, F and D have all been computed!

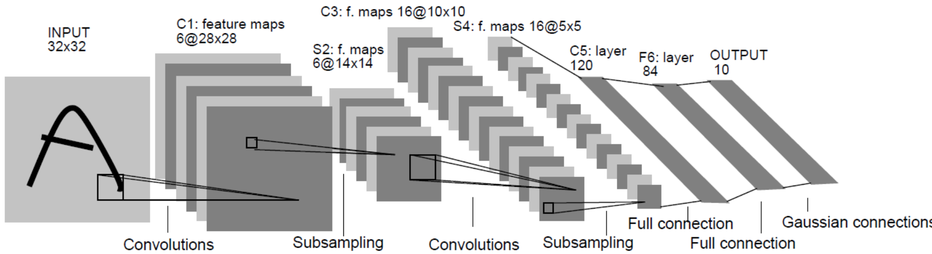
Calculation in complex architectures

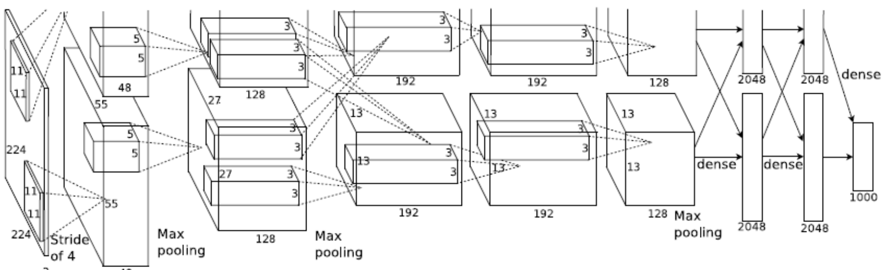
- What if the combination is not summation but "concatenation"?

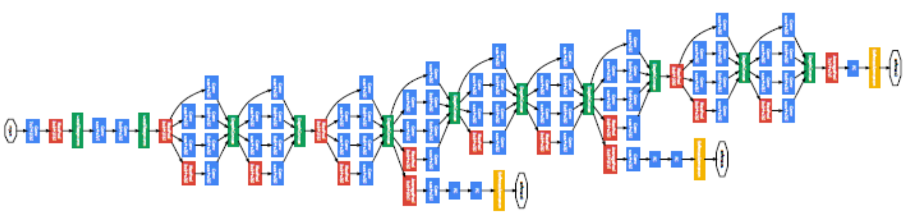


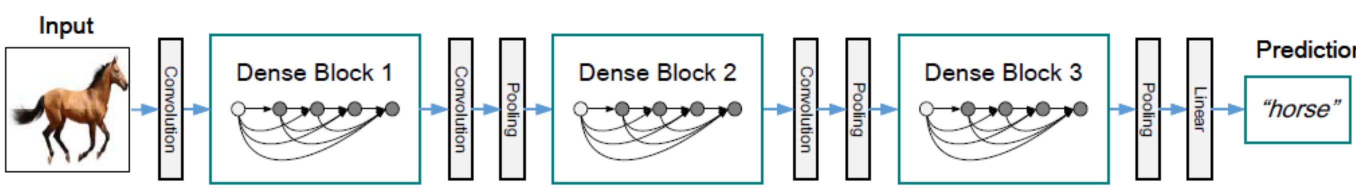
Same as the case without feature combination discussed before!

Summary of part 3

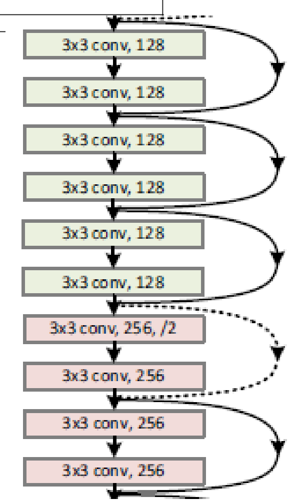








ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



Outline

1. Pooling
2. Standard CNN
3. Typical CNNs
4. Training techniques-II
5. Summary

Training techniques-II

a) Optimizers
b) Prevent overfitting

Recap: SGD and momentum

- SGD optimizes over individual minibatches $\left(\boldsymbol{X}^{(i)}, \boldsymbol{t}^{(i)}\right)$ at each iteration

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)}\right)$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{g}$$

- The momentum update is given by,

$$\boldsymbol{v} = \gamma \boldsymbol{v} - \eta \boldsymbol{g}$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$$

- Problem 1: We need to adjust learning rates η during training
  - Recall different strategies
  - Tuning the learning rates is expensive!

Are there any methods that can adaptively tune the learning rates?

Per-parameter adaptive learning rate methods

- Problem 2: The previous method manipulates the learning rate globally and equally for all parameters

  Is it possible to adaptively tune the learning rates for individual parameters?

- Many of these methods may still require other hyperparameter settings, but the argument is that they are well-behaved for a broader range of hyperparameter values than the raw learning rate

Adagrad

- An adaptive learning rate method (Duchi et al. 2011)
- Denote the gradient by $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)}\right)$
- The updating rule

$$\left.\begin{aligned} \boldsymbol{c} &= \boldsymbol{c} + \boldsymbol{g}^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}} \end{aligned}\right\} \text{Elementwise operations}$$

  where ϵ is usually set between 10^{-4} and 10^{-8}
- $\boldsymbol{c}$ is used to normalize the parameter update step

Suppose at a certain iteration, $c_3 = 90, \mathrm{c}_9 = 2$, then

$$\Delta\theta_3 = -\frac{\eta}{\sqrt{90 + \epsilon}} g_3 \quad \Delta\theta_9 = -\frac{\eta}{\sqrt{2 + \epsilon}} g_9$$

Effective learning rate

Parameters received small updates will have larger effective learning rates

Adagrad

- An adaptive learning rate method (Duchi et al. 2011)
- Denote the gradient by $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)}\right)$
- The updating rule

$$\left.\begin{aligned} \boldsymbol{c} &= \boldsymbol{c} + \boldsymbol{g}^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}} \end{aligned}\right\} \text{Elementwise operations}$$

  where ϵ is usually set between 10^{-4} and 10^{-8}
- Any problem with this method?

Adagrad

- An adaptive learning rate method (Duchi et al. 2011)
- Denote the gradient by $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)}\right)$
- The updating rule

$$\left.\begin{aligned} \boldsymbol{c} &= \boldsymbol{c} + \boldsymbol{g}^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}} \end{aligned}\right\} \text{Elementwise operations}$$

  where ϵ is usually set between 10^{-4} and 10^{-8}
- Any problem with this method?
  - The effective learning rates are monotonically decreasing, which may leads to early stopping

RMSProp

- Adagrad: let $\boldsymbol{c}$ accumulate $\boldsymbol{g}^2$ at all previous steps
- RMSProp: let $\boldsymbol{c}$ accumulate the recent $\boldsymbol{g}^2$
- In practice

$$\boldsymbol{c} = \gamma \boldsymbol{c} + (1-\gamma)\boldsymbol{g}^2$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}}$$

- Typical values for γ are 0.9, 0.99, 0.999
- RMSProp still modulates the learning rate of each parameter based on the magnitudes of its gradients, but unlike Adagrad the updates do not get monotonically smaller.

Adagrad versus RMSProp

Adagrad: $\boldsymbol{c}(t) = \boldsymbol{c}(t-1) + \boldsymbol{g}(t)^2$

$$= \boldsymbol{c}(0) + \boldsymbol{g}(1)^2 + \boldsymbol{g}(2)^2 + \ldots + \boldsymbol{g}(t)^2$$

All $\boldsymbol{g}(1)^2, \ldots, \boldsymbol{g}(t)^2$ contribute equally to $\boldsymbol{c}(t)$

RMSProp: $\boldsymbol{c}(t) = \gamma\boldsymbol{c}(t-1) + (1-\gamma)\boldsymbol{g}(t)^2$

$$= \gamma^t \boldsymbol{c}(0) + \boxed{\gamma^{t-1}(1-\gamma)}\boldsymbol{g}(1)^2 + \boxed{\gamma^{t-2}(1-\gamma)}\boldsymbol{g}(2)^2 + \ldots + \gamma(1-\gamma)\boldsymbol{g}(t-1)^2 + (1-\gamma)\boldsymbol{g}(t)^2$$

Product of t numbers between (0,1)

Product of $t-1$ numbers between (0,1)

Contributions of $\boldsymbol{g}(k)^2$ to $\boldsymbol{c}(t)$ far away from t decay exponentially

Adam

- This method is proposed by Kingma and Ba (2015). Default algorithm!
- The simplified version

$$\boldsymbol{m} = \beta_1 \boldsymbol{m} + (1 - \beta_1)\boldsymbol{g}$$
$$\boldsymbol{c} = \beta_2 \boldsymbol{c} + (1 - \beta_2)\boldsymbol{g}^2$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{m}}{\sqrt{\boldsymbol{c} + \epsilon}}$$

Recommended values: $\epsilon = 10^{-8}, \beta_1 = 0.9, \beta_2 = 0.999$

  - Compared with RMSProp, it uses the "smooth" version of the gradient $\boldsymbol{m}$. This is like a momentum.
- The full version ("warm up" version)

$$\boldsymbol{m} = \beta_1 \boldsymbol{m} + (1 - \beta_1)\boldsymbol{g}, \qquad \boldsymbol{m}_t = \boldsymbol{m}/(\boldsymbol{1} - \beta_1^t)$$
$$\boldsymbol{c} = \beta_2 \boldsymbol{c} + (1 - \beta_2)\boldsymbol{g}^2, \qquad \boldsymbol{c}_t = \boldsymbol{c}/(\boldsymbol{1} - \beta_2^t)$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{c}_t + \epsilon}}$$

where t denotes the iteration

More details about optimization techniques

- http://cs231n.github.io/neural-networks-3/#update

Summary of Part 4a

SGD

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)}\right)$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{g}$$

$\Rightarrow$

SGD+momentum

$$\boldsymbol{v} = \gamma \boldsymbol{v} - \eta \boldsymbol{g}$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$$

Adagrad

$$\boldsymbol{c} = \boldsymbol{c} + \boldsymbol{g}^2$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}}$$

$\Rightarrow$

RMSProp

$$\boldsymbol{c} = \gamma \boldsymbol{c} + (1 - \gamma) \boldsymbol{g}^2$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{c} + \epsilon}}$$

$\Rightarrow$

Adam

$$\boldsymbol{m} = \beta_1 \boldsymbol{m} + (1 - \beta_1) \boldsymbol{g}$$
$$\boldsymbol{c} = \beta_2 \boldsymbol{c} + (1 - \beta_2) \boldsymbol{g}^2$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\boldsymbol{m}}{\sqrt{\boldsymbol{c} + \epsilon}}$$

- Which is the best?
- All optimizers have a learning rate η
  - Learning rate decay is always a good strategy

Training techniques-II

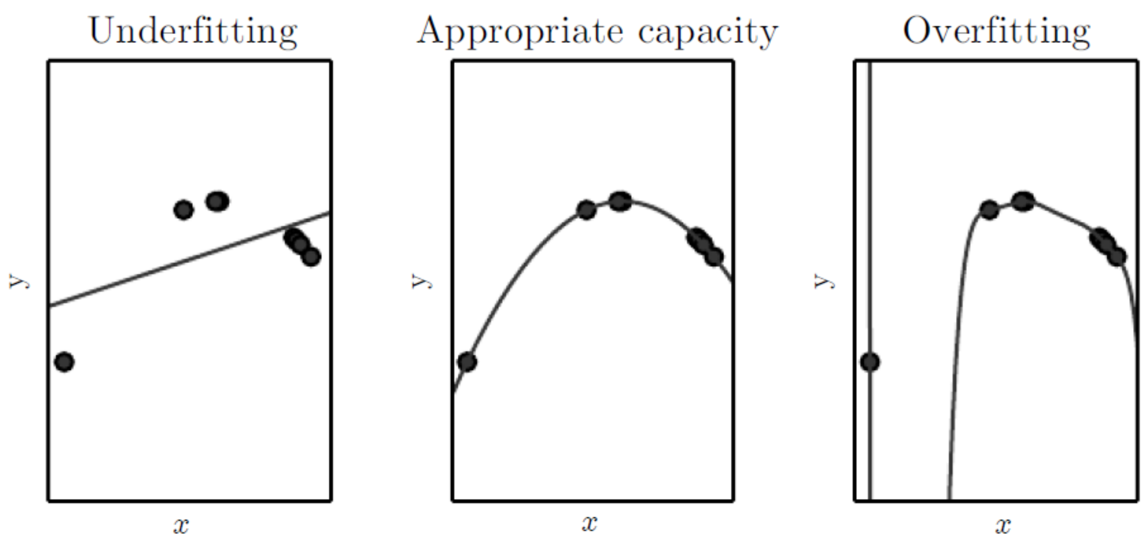
a) Optimizers
b) Prevent overfitting

Recall: polynomial regression example

- Consider a regression problem in which the input x and output y are both scalars. Find a function $f: \mathbb{R} \to \mathbb{R}$ to fit the data
  - $f(x) = b + wx$
  - $f(x) = b + w_1 x + w_2 x^2$
  - $f(x) = b + \sum_{i=1}^{9} w_i x^i$

MSE training:

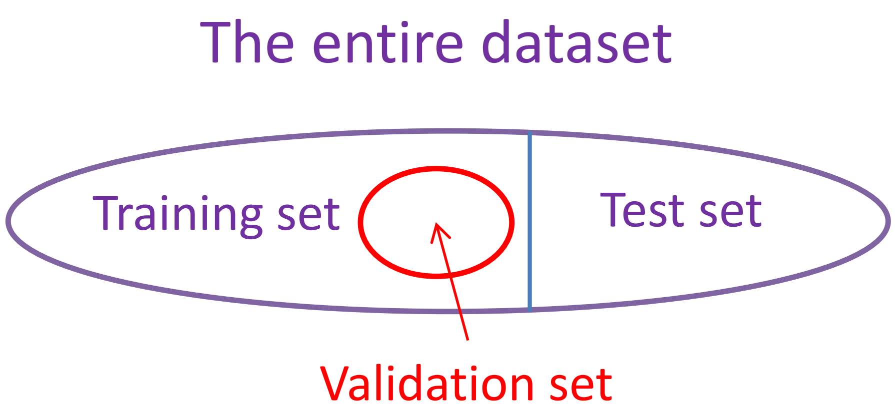
$$\min_w \frac{1}{N} \sum_{n=1}^{N} \left\| f\left(x^{(n)}\right) - y^{(n)} \right\|_2^2$$

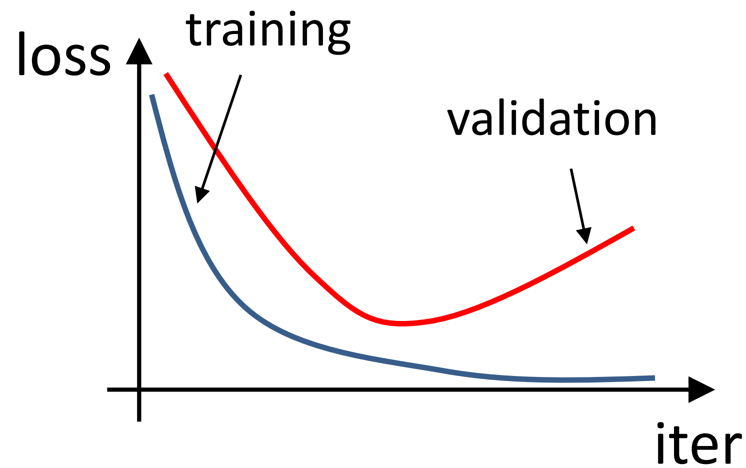


Overfitting

- A neural network (as well as other machine learning models) typically contains many parameters to learn (e.g., millions to billions) which tends to overfit the training data
- What's overfitting?
  - Fits the training data well but performs poorly on held-out test data
- Weight regularization is one method to handle this
- Other techniques
  - Early stopping
  - Dropout proposed by Hinton et al. (2012)
  - Data augmentation

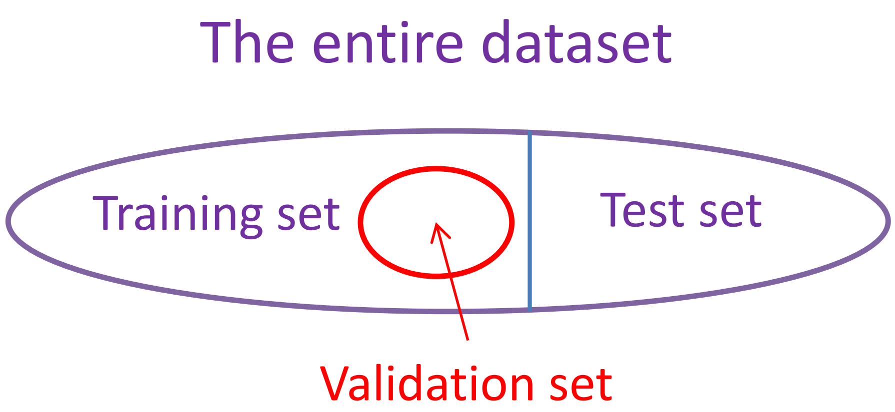
Early stopping

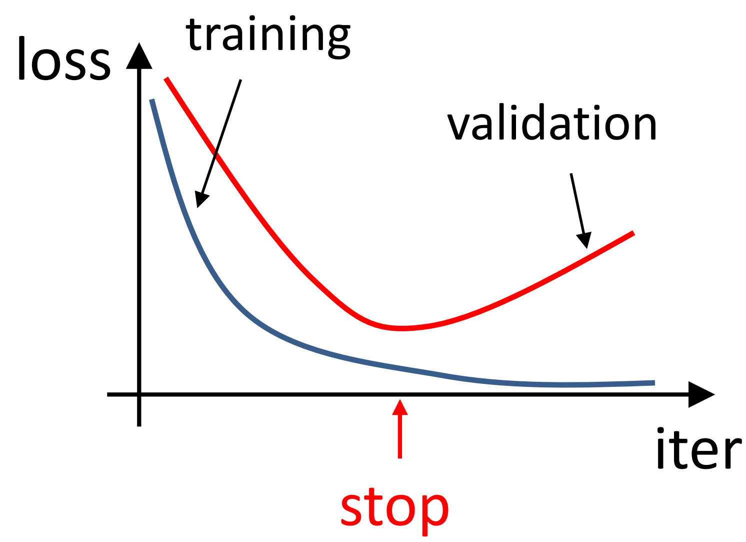




- When you see the loss on the training set is decreasing, but the loss on the validation set begins to increase…

Early stopping

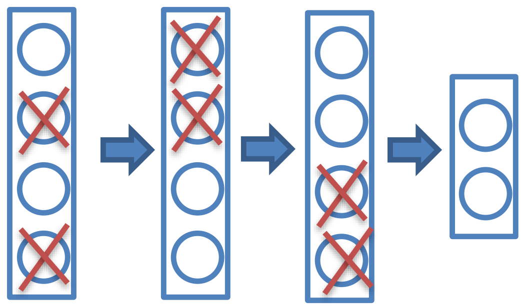




- When you see the loss on the training set is decreasing, but the loss on the validation set begins to increase…STOP there!

Dropout

- On each presentation of each training case, each hidden unit is randomly omitted (zero its output) from the network with a probability p (e.g., 0.5)

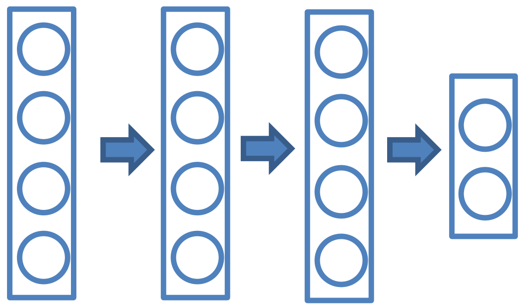


These zero values are used in the backward pass propagating the derivatives to the parameters

- Advantage
  - A hidden unit cannot rely on other hidden units being present, therefore we prevent complex co-adaptations of the neurons on the training data
  - It trains a huge number of different networks in a reasonable time, then average their predictions

Testing phase

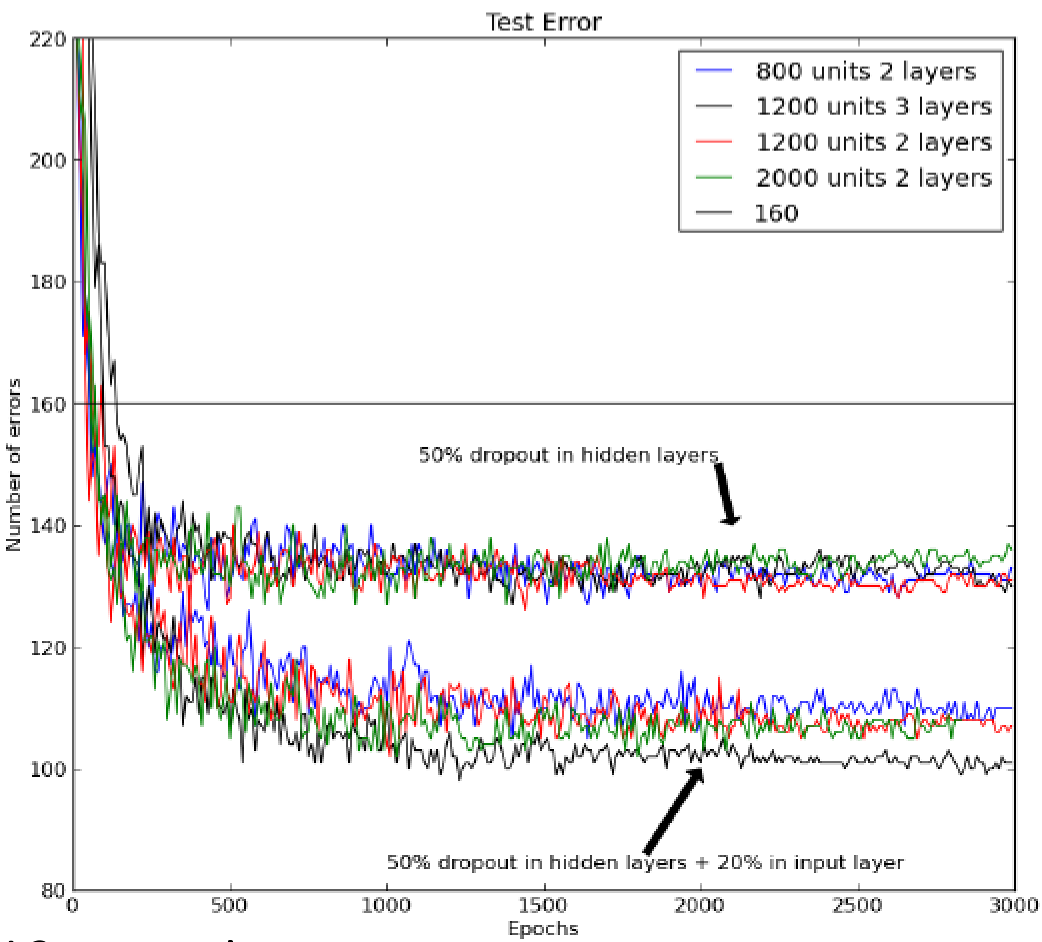
- Use the "mean network" that contains all of the hidden units



- But we need to adjust the outgoing weights of neurons to compensate for the fact that in training only a portion of them are active
  - If $p = 0.5$, we halve the weights
  - If $p = 0.1$, we multiply the weights with $1 - p$, i.e., 0.9
- In practice, this gives very similar performance to averaging over a large number of dropout networks.

Results on MNIST

- Standard MLP without the tricks
  – Enhancing training data with transformed images
  – Generative pre-training
- In this setting without dropout the best results is 160 errors
- Dropout can significantly reduce errors



Another trick is used: separate L2 constraints on the incoming weights of each hidden unit

Hinton et al., 2012

Remarks

- In some implementations, during test, $(1-p)$ is multiplied with the output of the activation function, say $f(Wx+b)$, instead of the weights W. Then
$$a_{\text{test}} = (1-p)f(Wx+b)$$
- In some implementations, the output of the activation function is changed as follows
$$a_{\text{train}} = \frac{1}{(1-p)}f(Wx+b)$$
Inverted dropout
during training, while the output of the activation function is unchanged during test
- In practice, p is set lower in lower layers, e.g., 0.2, but higher in higher layers, e.g., 0.5
- In the literature or some software, the dropout rate is sometimes defined as the probability for retaining the output of each node, i.e., $(1-p)$

Discussion

- Inspired by dropout, can you figure out other techniques to alleviate overfitting?
  - Dropconnect by Wan, Zeiler, Zhang, LeCun, Fergus (2013)
  - Drop pixels: a data augmentation method

Data augmentation

Let $\{\boldsymbol{x}^{(n)}, \boldsymbol{t}^{(n)}\}$ denote the original training set

① Add variations to the input data

$$\boldsymbol{x}^{(n)} \rightarrow \widehat{\boldsymbol{x}}^{(n)}$$

while keep the label $\boldsymbol{t}^{(n)}$ unchanged

② Use the augmented training set

$$\{\boldsymbol{x}^{(n)}, \boldsymbol{t}^{(n)}\} \cup \{\widehat{\boldsymbol{x}}^{(n)}, \boldsymbol{t}^{(n)}\}$$

to train the model

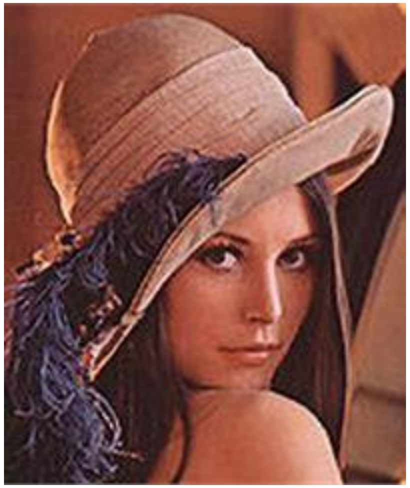
Different data (text, image, video) type can use different variations

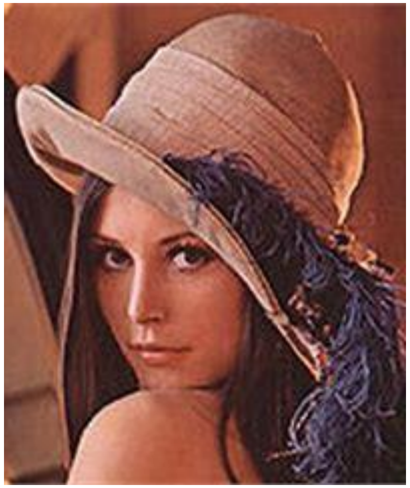
What's the motivation?

Commonly used variations for images

- Random
  - flips
  - translations
  - crops and scales
  - stretching
  - shearing
  - Cutout or erasing
  - mixup
  - color jittering
  - etc.
- A combination of above

Random flip



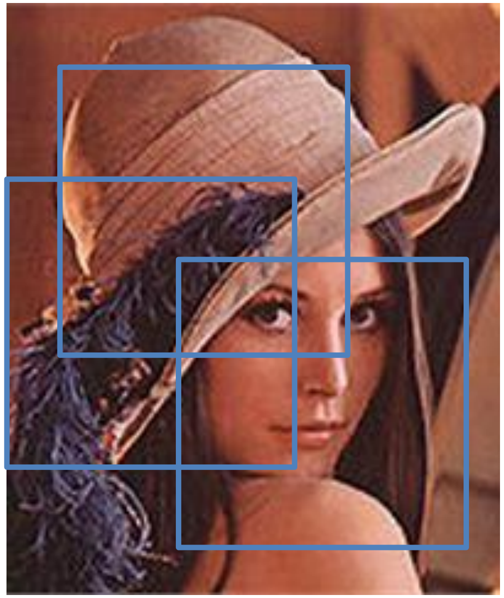


Random crops and scales

During training, you can…

crops

Scale then crop



What would you do for test?

主观题 10分

Setting

If you use random crops and scales during training, how do you get the prediction for a test image?

Open Question is only supported on Version 2.0 or newer.

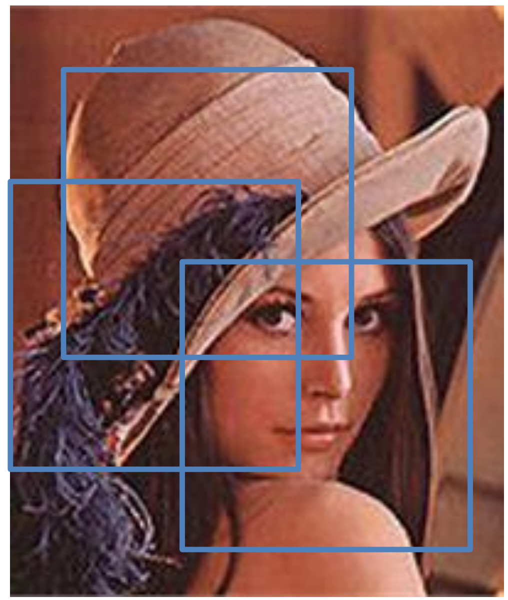
Answer

Random crops and scales

During training, you can…

crops

Scale then crop

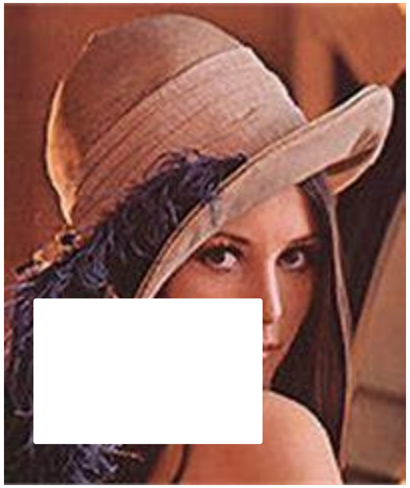


During test, you can

- Resize the test images to the required input size of the model
- Crop a region (usually the center one) and input to the model
- Crop multiple regions and input to the model, then average the predictions of the model

Random cutout or erasing

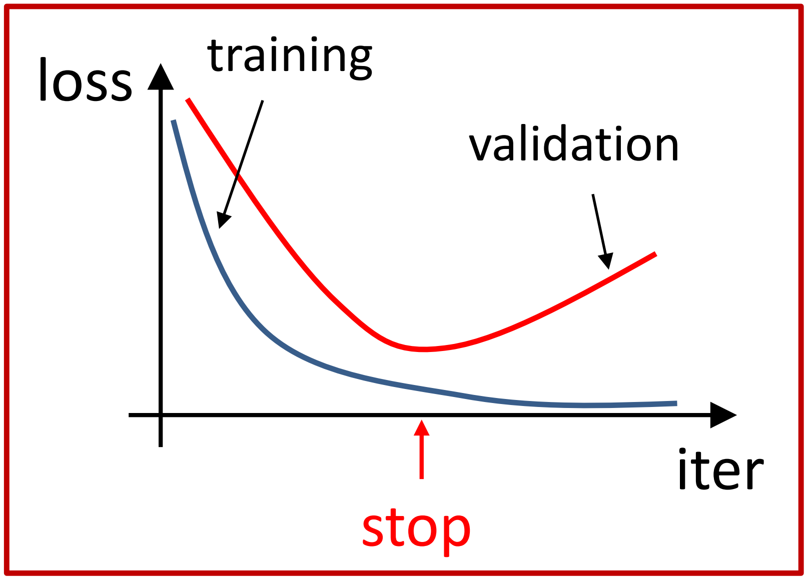
During training, you can...

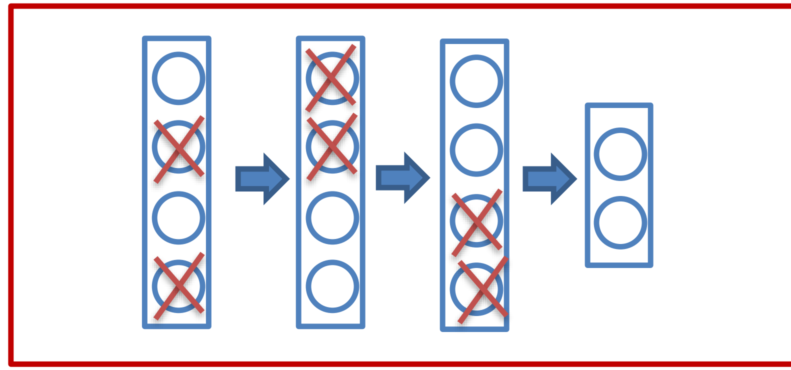
During test, you input the whole image to the model

Summary of Part 4b

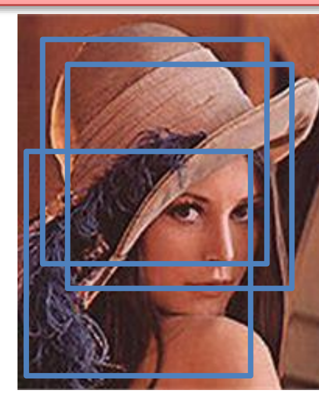
Early stop



Dropout



Data augmentation



多选题 1分

Setting

Which updating rule(s) allow different learning rates for different parameters?

- A Stochastic gradient decent (SGD)
- B SGD+momentum
- C Adagrad
- D RMSProp
- E Adam

Submit

单选题 1分

Setting

Which updating rule(s) has the early stopping problem?

A Adagrad

B RMSprop

C Adam

Submit

Outline

1. Pooling
2. Standard CNN
3. Typical CNNs
4. Training techniques-II
5. Summary

Summary of this lecture

Knowledge

Part 1

Part 2

Conv | Max pooling | Avg pooling | Full connection

ReLU layer | Sigmoid layer | Cross-entropy loss | …

Part 3

AlexNet | VGGNet | GoogLeNet | ResNet

Inception v2, v3 | DenseNet | …

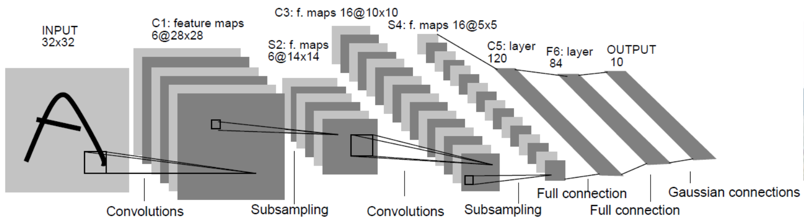
Part 4

Adagrad → RMSProp → Adam

Early stop → Dropout → Data augmentation

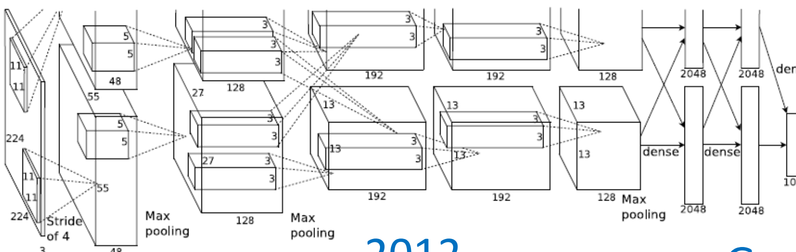
Summary of this lecture

Capability and value



1989-1998

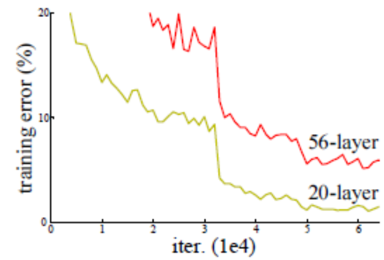
Yann LeCun



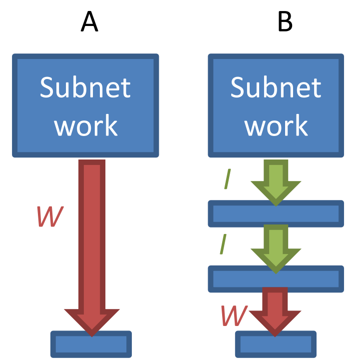
2012

Geoffrey Hinton

Mark the resurgence of deep learning



Kaiming He



Best CVPR 2016 paper

- Solve the problem, though there seems to be no novelty!
- Make a prediction, do experiment, analyze the disagreement, solve the problem

Recommended reading

- Krizhevsky, Sutskever, Hinton (2012)
  ImageNet Classification with Deep Convolutional Neural Networks
  NeurIPS
- Szegedy, Liu, Jia et al. (2015)
  Going deeper with convolutions
  CVPR
- He, Zhang, Ren, Sun (2016)
  Deep Residual Learning for Image Recognition
  CVPR
- Huang, Liu, van der Maaten, Weinberger (2017)
  Densely Connected Convolutional Networks
  CVPR
- http://cs231n.github.io/neural-networks-3/#update

Prepare for the next lecture

1. Read the following paper
   – Szegedy, Vanhoucke, Ioffe et al. (2016) Rethinking the Inception Architecture for Computer Vision, CVPR
2. Form groups of 2 and every group prepares a 5-minute presentation with slides for one of the following papers
   – Hu, Shen, Sun (2018) Squeeze-and-Excitation Networks, CVPR
   – Li, Wang, Hu, Yang (2019) Selective Kernel Networks, CVPR