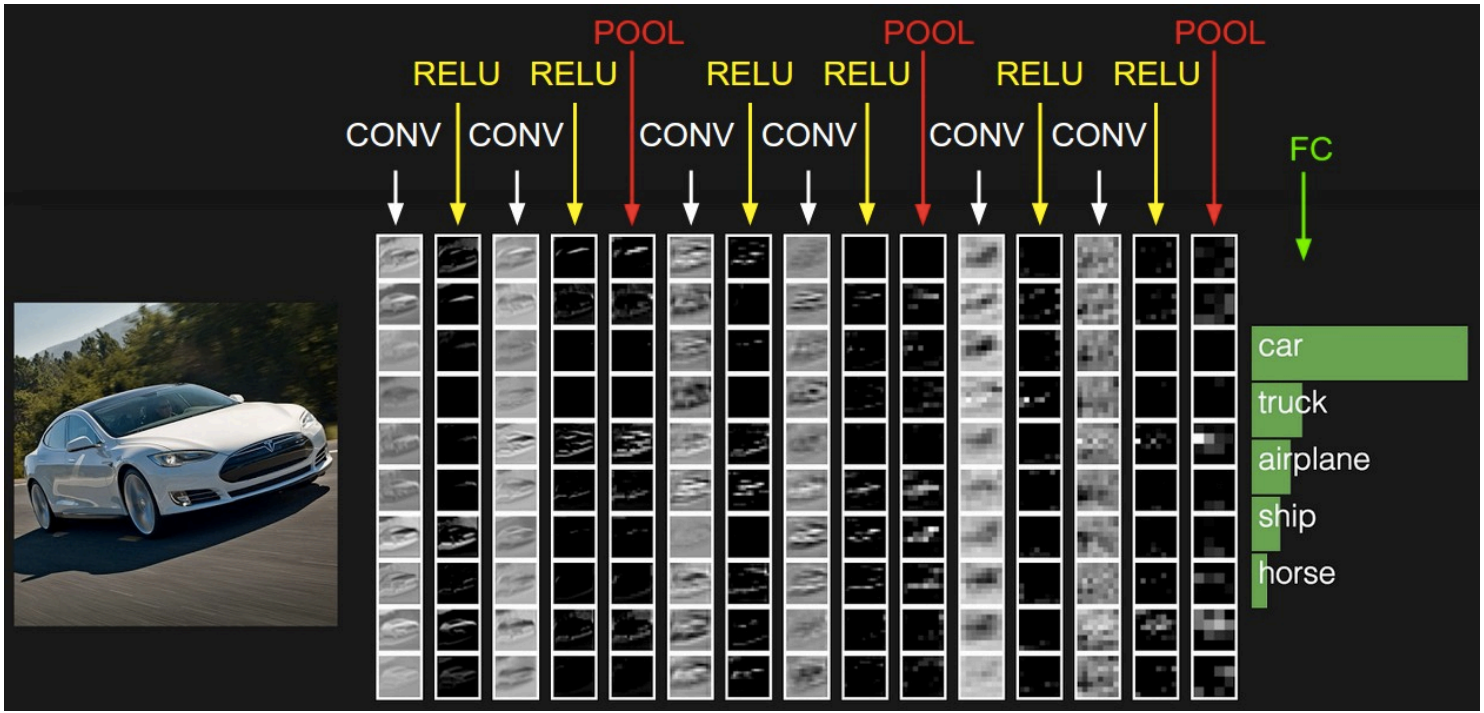
Convolutional neural network

Hung-Hsuan Chen

Many slides are taken from Fei-Fei Li @Stanford

Convolutional Neural Network (CNN)

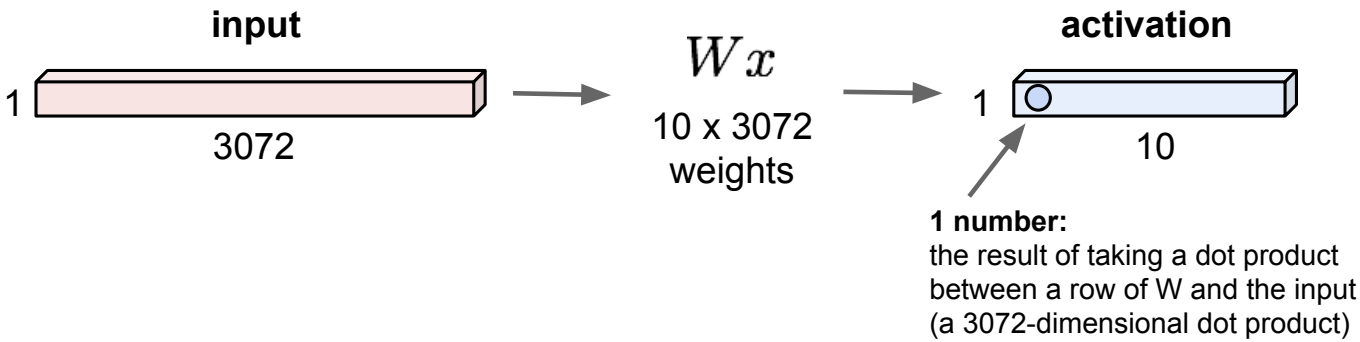
- Widely used in computer vision
  - Image recognition
    - Given a photo, name the objects in the photo
- Its variations are used in other fields
  - Natural language processing
  - Signal processing
  - AlphaGo


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Naïve version: use fully connected feedforward neural network for object recognition

32x32x3 image -> stretch to 3072 x 1

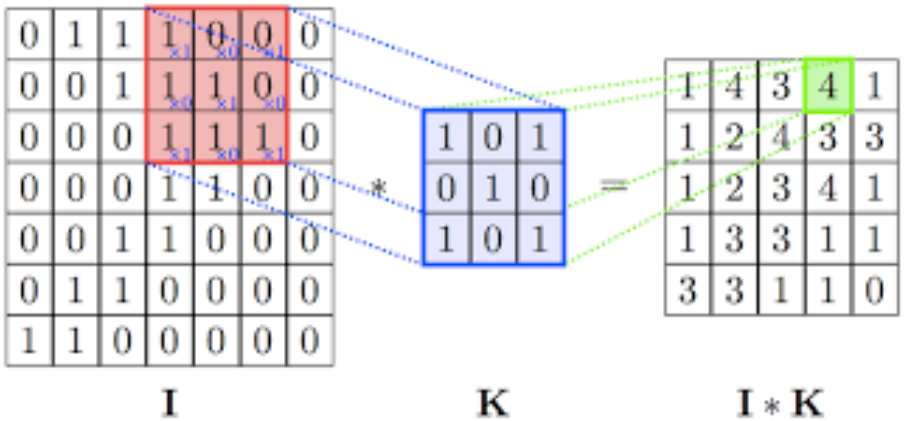
10-class classificaton



Possible problem

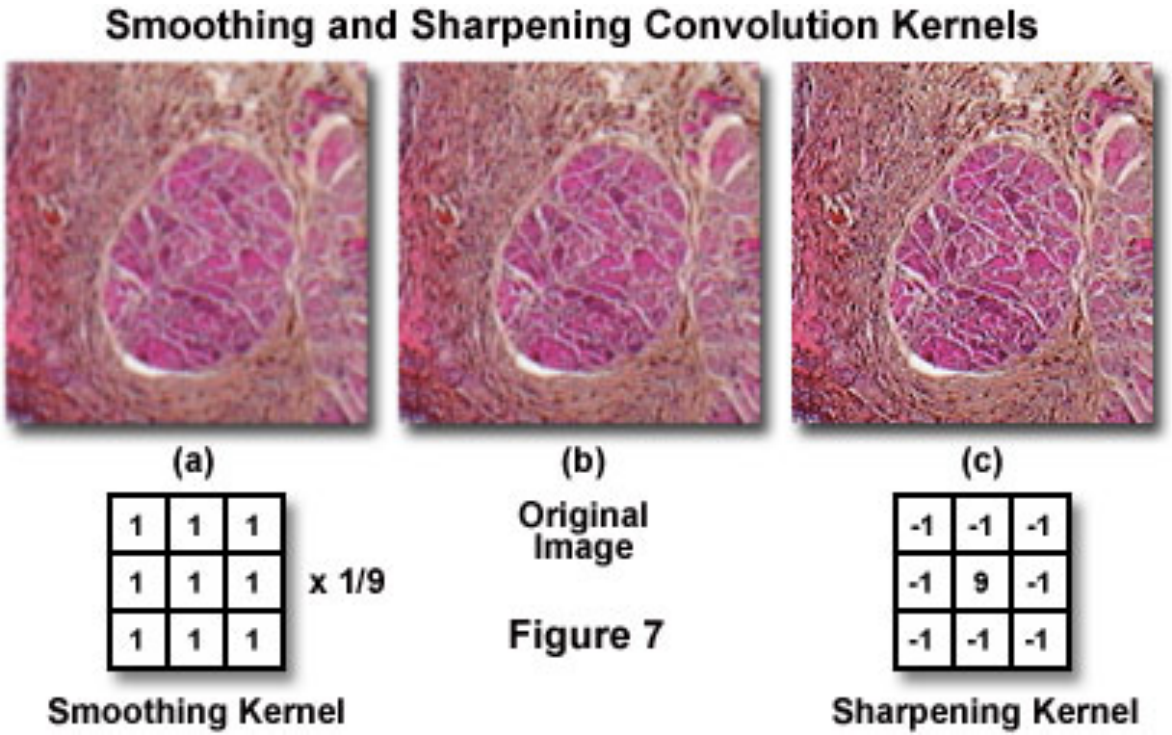
- # of features is huge in single layer
  - A 256 * 256 image has 256 * 256 * 3 = 196,608 features
  - The number of parameters to learn may explode if we have more layers
- Spatial features are not preserved
  - For object recognition, neighboring pixels are probably more important than the pixels that are farther away

Convolution kernel



- **Convolution kernel** a.k.a. **convolution** a.k.a. **kernel** a.k.a. **filter**
- The kernel slides through the entire image (mostly pixel by pixel)

Examples of human designed convolution kernels

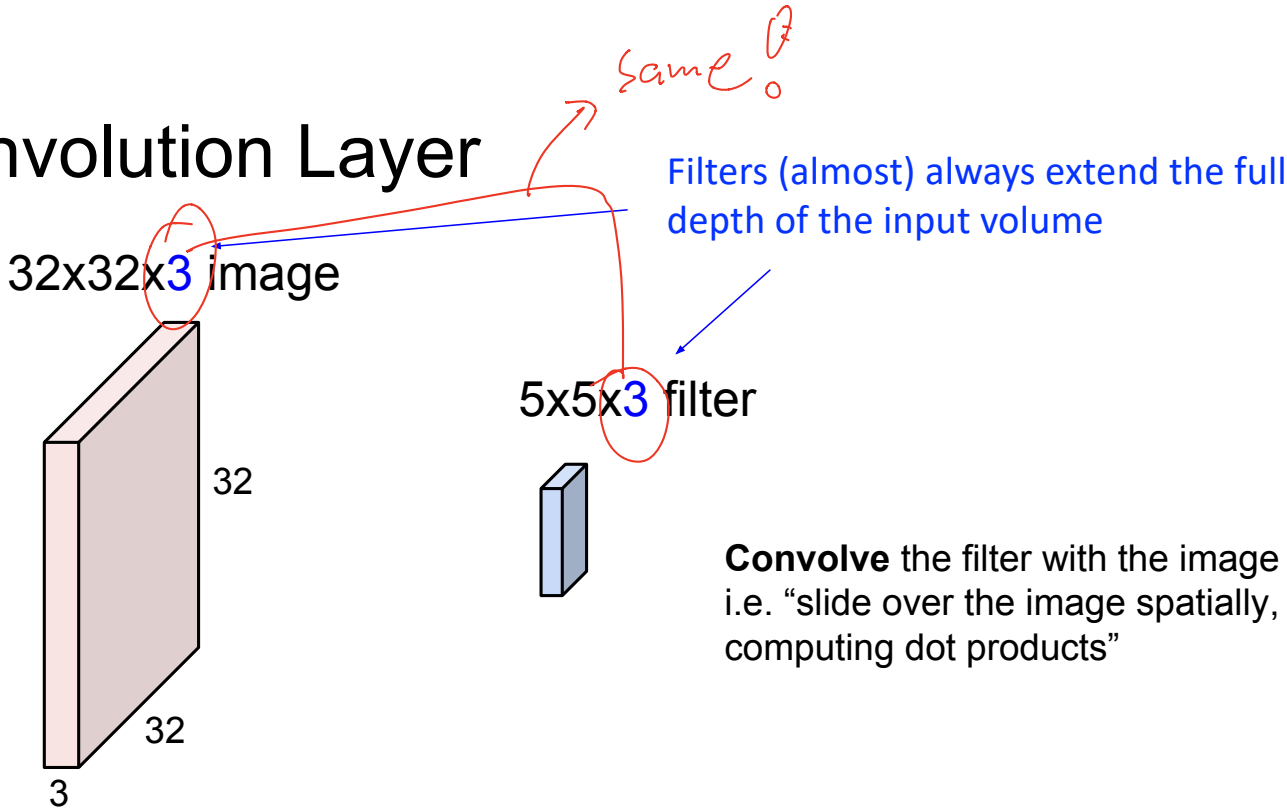


Convolutional neural network

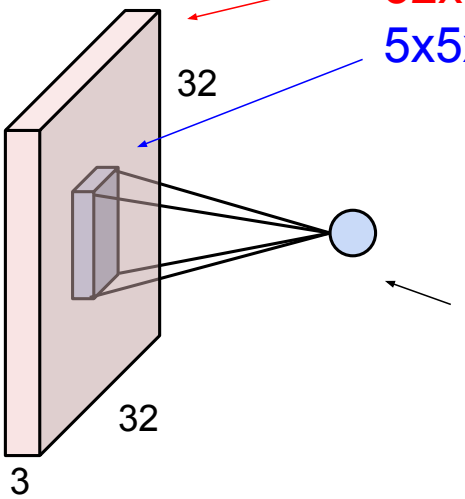
- Design simple convolution kernels is probably possible
  - E.g., sharpen, smoothing (blur), etc.
- Design complex convolution kernels is difficult
  - How to design a kernel to identify the motorcycles?
- Convolutional neural network
  - Learning to find the convolution kernels to identify objects

find kernal by itsself.

Convolution Layer



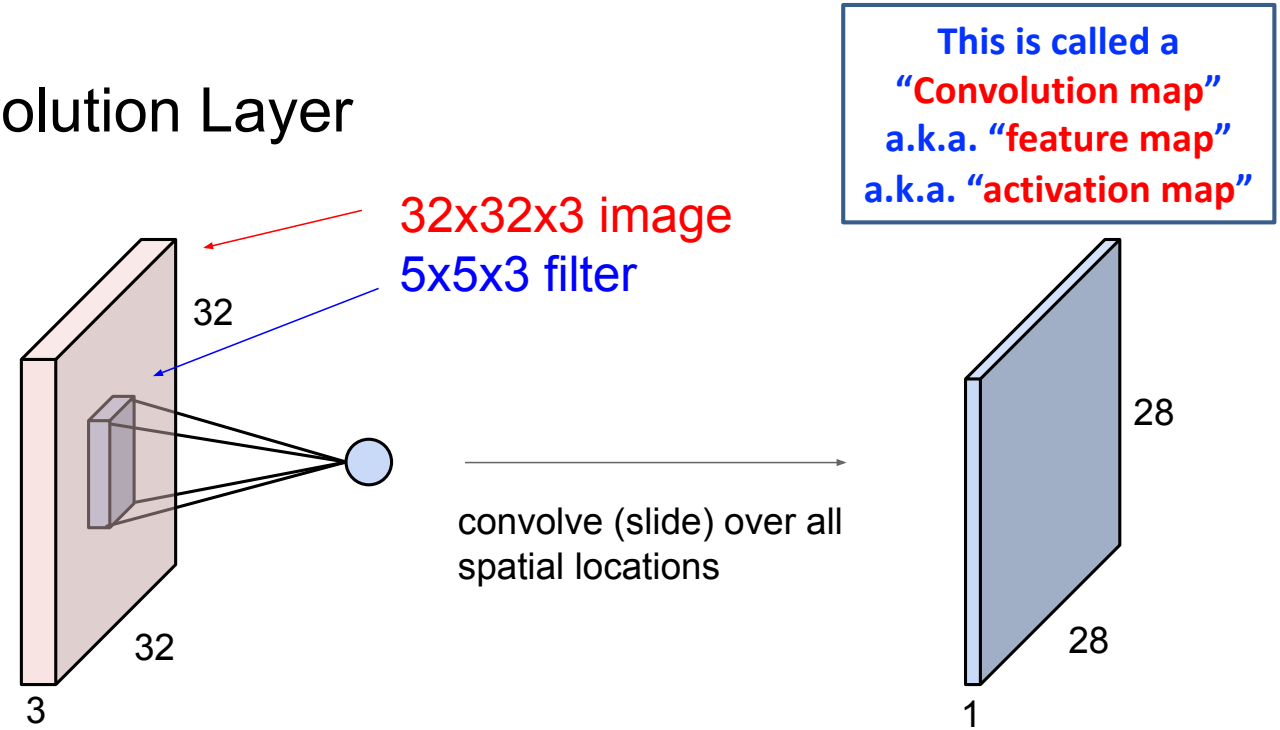
Convolution Layer



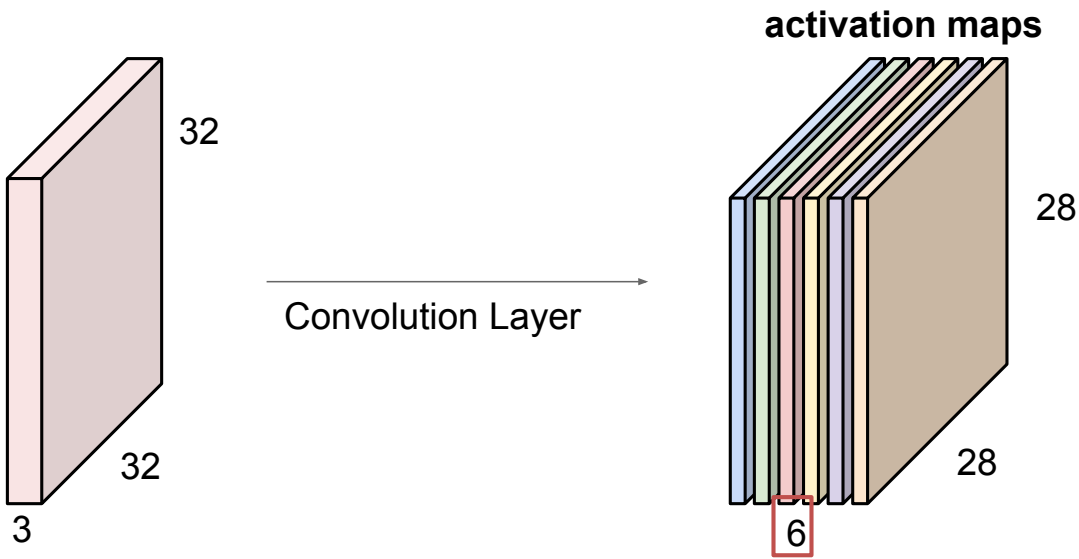
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

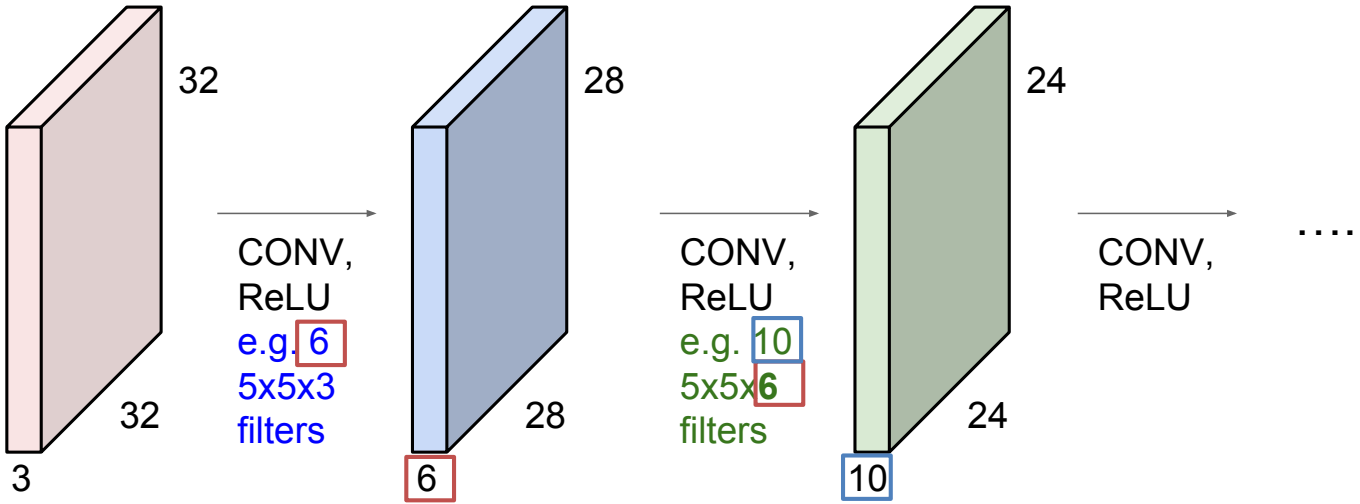


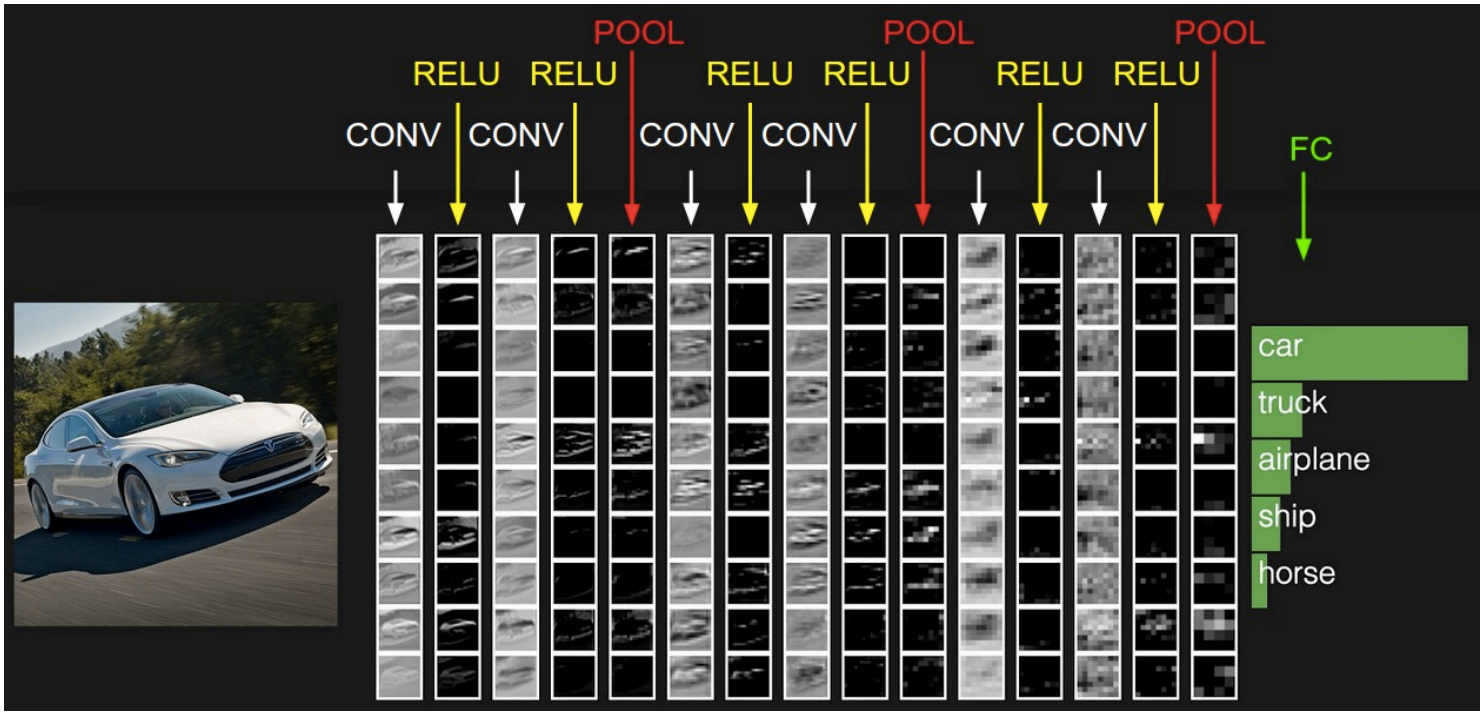
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



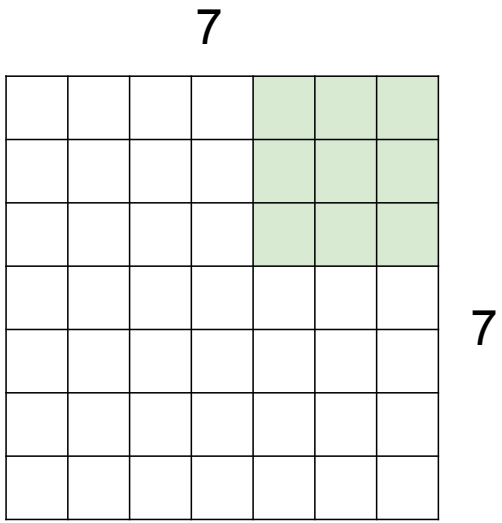
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions




POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

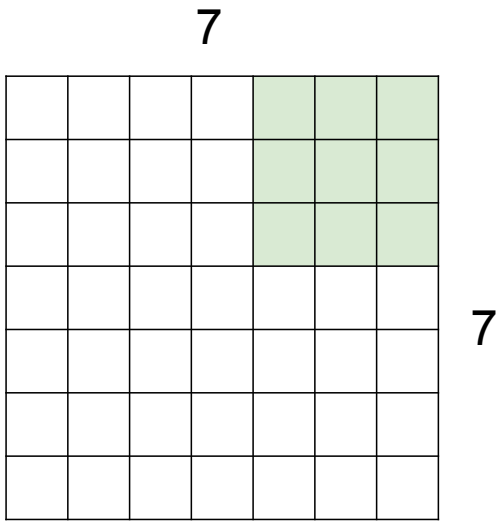
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

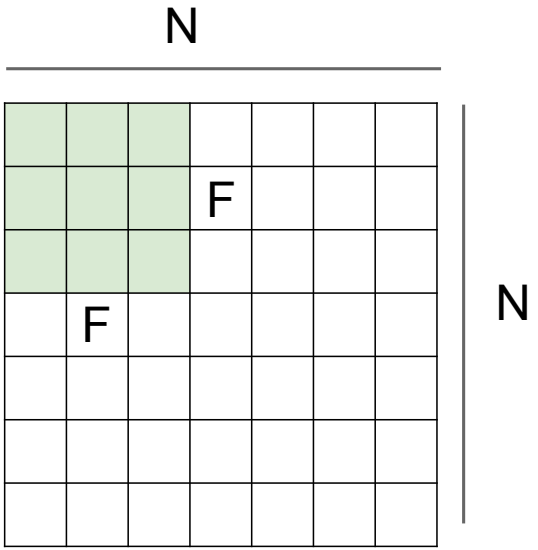
=> 5x5 output

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

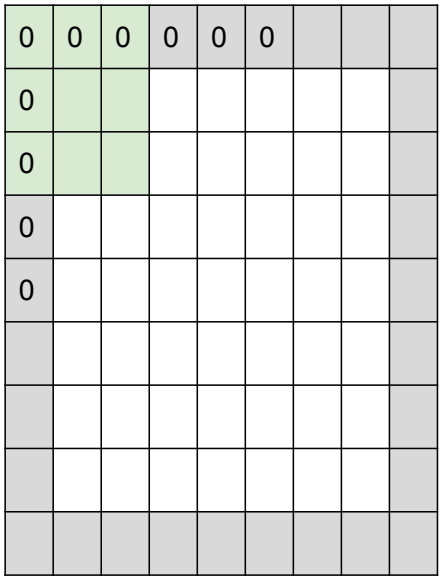
Stride 2: jump 2 grids at a time



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

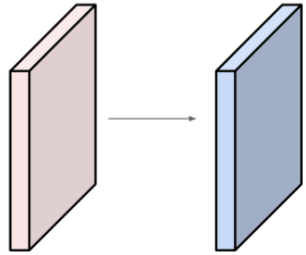
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Quiz

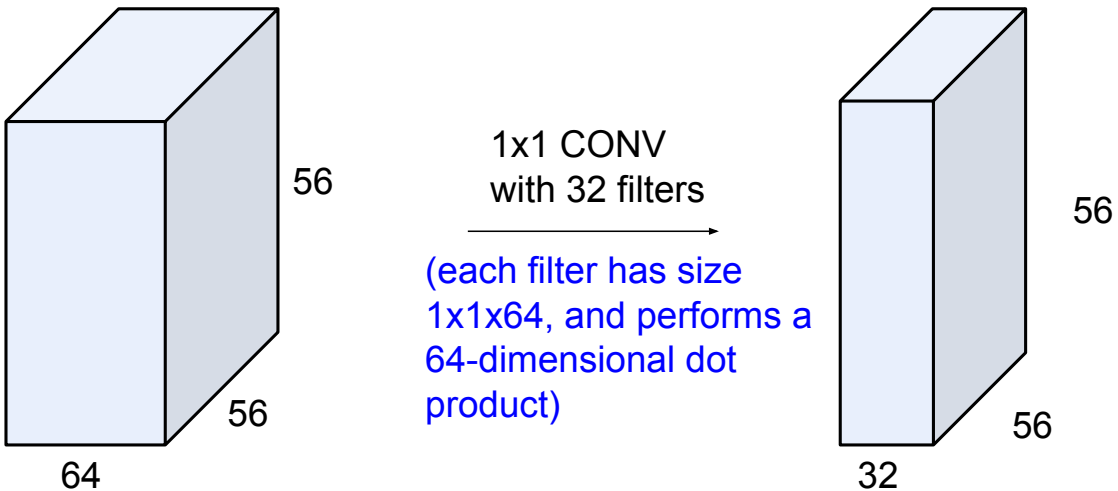


- Q: Input volume: 32x32x3; apply 10 5x5 filters with stride 1 and pad 2
  - What is the output volume size?
    - A: (1) for each filter, the output size is 32x32x1; (2) 10 such filters ➔ 32 x 32 x 10
  - How many parameters to learn in this layer?
    - A: (1) each filter contains 5x5x3+1 parameters (the +1 is the bias term); (2) 10 filters ➔ 76 x 10 = 760

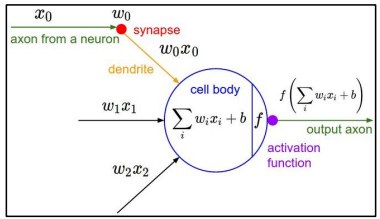
Quiz

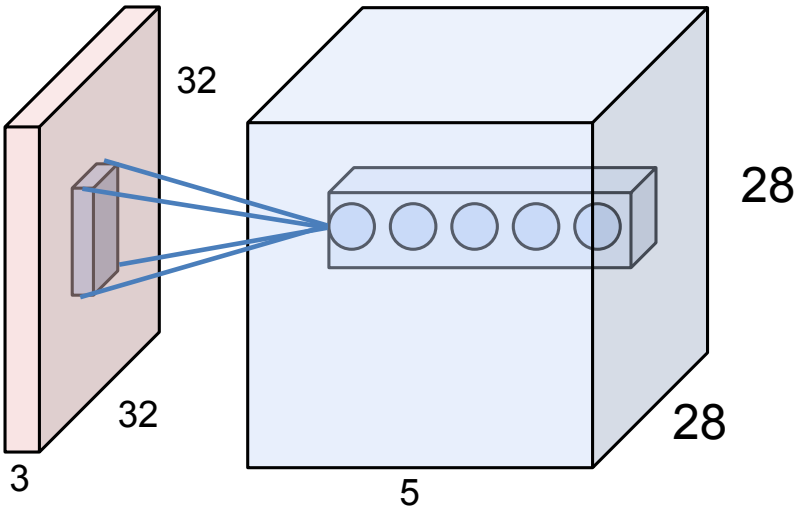
- What is the relationship between a kernel and a feature map?

(btw, 1x1 convolution layers make perfect sense)



The brain/neuron view of CONV Layer



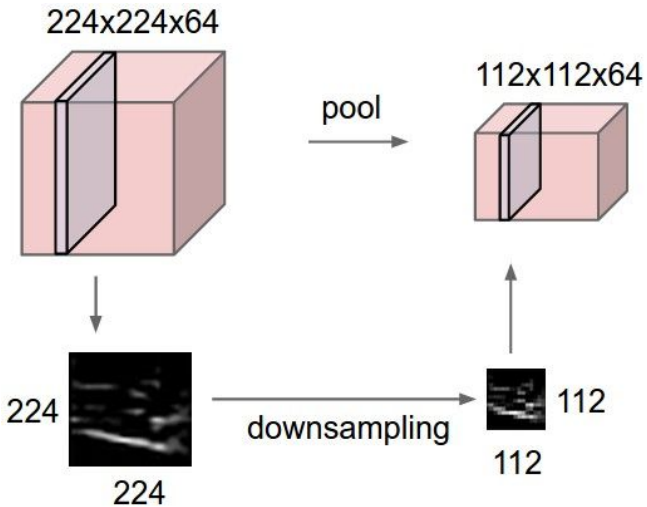


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

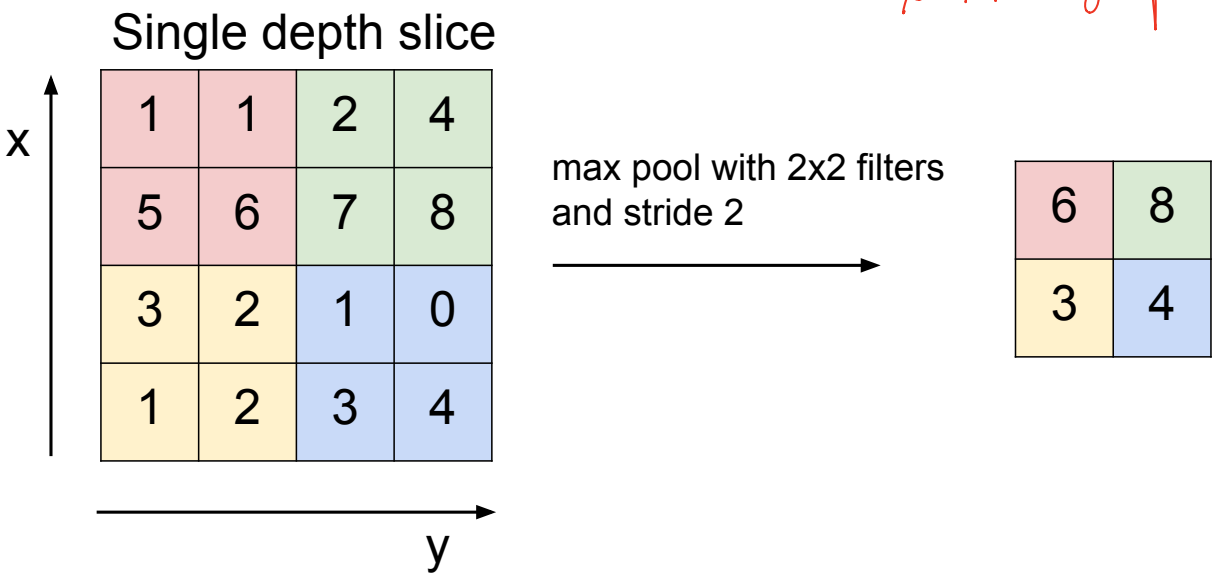
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING

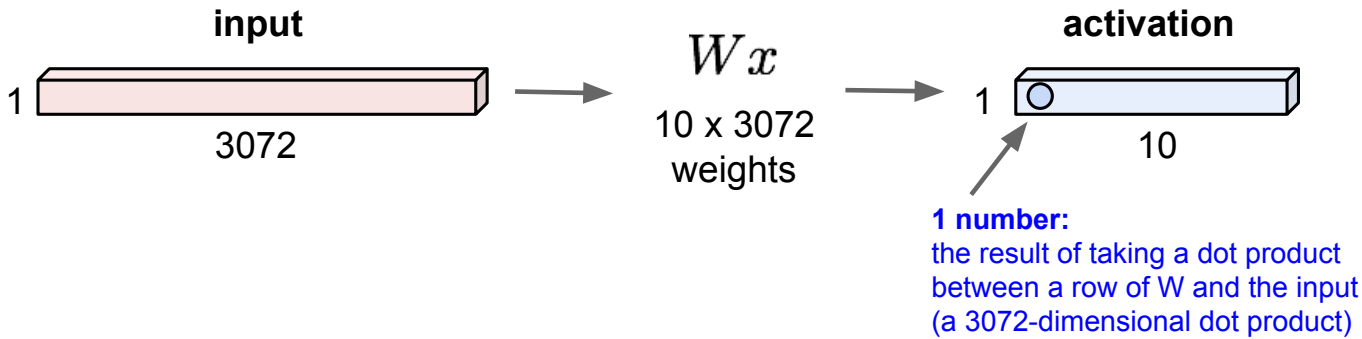
↳ 另有 avg pooling



Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

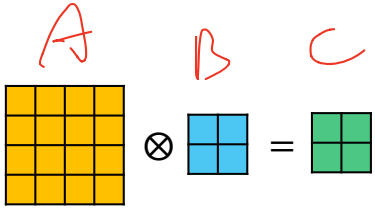
Each neuron looks at the full input volume



How to learn the parameters?

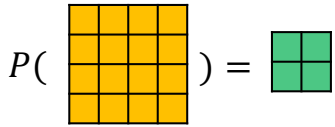
- Compute the “local gradient” for each operator
- Operators
  - Convolution
  - Pooling
  - Fully connected

How to learn the parameters -- convolution



- Convolution example: input A: 4x4x1; apply 10 2x2 filters B_i, padding 0, stride 2
  - $A \bigotimes B_i = C_i, A = [a_{p,q}] \in R^{4\times4\times1}, B_i = [b_{r,s}] \in R^{2\times2\times1}, C_i = [c_{t,u}] \in R^{2\times2\times1}, i = 1, \dots, 10$
  - $a_{p,q}b_{1,1} + a_{p+1,q}b_{2,1} + a_{p,q+1}b_{1,2} + a_{p+1,q+1}b_{2,2} + b_{\text{bias}} = c_{(p+1)/2,(q+1)/2}$
    - $p = 1,3; q = 1,3$
  - $(a_{1,1} + a_{1,3} + a_{2,1} + a_{2,3})b_{1,1} + (a_{1,2} + a_{1,4} + a_{3,2} + a_{3,4})b_{1,2} + (a_{2,1} + a_{2,3} + a_{4,1} + a_{4,3})b_{2,1} + (a_{2,2} + a_{2,4} + a_{4,2} + a_{4,4})b_{2,2} + 4b_{\text{bias}} = c_{1,1} + c_{1,2} + c_{2,1} + c_{2,2} \equiv y$
  - $\frac{\partial y}{\partial b_{1,1}} = a_{1,1} + a_{1,3} + a_{2,1} + a_{2,3}, \frac{\partial y}{\partial b_{1,2}} = a_{1,2} + a_{1,4} + a_{3,2} + a_{3,4}, \frac{\partial y}{\partial b_{2,1}} = a_{2,1} + a_{2,3} + a_{4,1} + a_{4,3}, \frac{\partial y}{\partial b_{2,2}} = a_{2,2} + a_{2,4} + a_{4,2} + a_{4,4}$

How to learn the parameters -- pooling



- Pooling example: input A: 4x4; apply 2x2 pooling P_i, stride 2
  - $P(A) = C, A = [a_{p,q}] \in R^{4\times4}, C = [c_{t,u}] \in R^{2\times2}$
  - $P(a_{p,q}, a_{p,q+1}, a_{p+1,q}, a_{p+1,q+1}) = c_{(p+1)/2,(q+1)/2} \equiv y$
    - $p = 1,3; q = 1,3$
  - $\frac{\partial y}{\partial a_{p,q}} = \begin{cases} 1 & \text{if } a_{p,q} \text{ is max} \\ 0 & \text{otherwise} \end{cases}$

How to learn the parameters – fully connected

- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}, \boldsymbol{y} \in R^{n\times 1}, \boldsymbol{W} \in R^{n\times d}, \boldsymbol{x} \in R^{d\times 1}$
  - Let $\boldsymbol{w}_i = \begin{bmatrix} w_{i,1} \\ \vdots \\ w_{i,d} \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,d} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_n^T \end{bmatrix}$
  - $y_i = \boldsymbol{w}_i^T \boldsymbol{x} \Rightarrow \frac{\partial y_i}{\partial w_{i,j}} = x_j$

LeNet-5 – the “classic CNN”

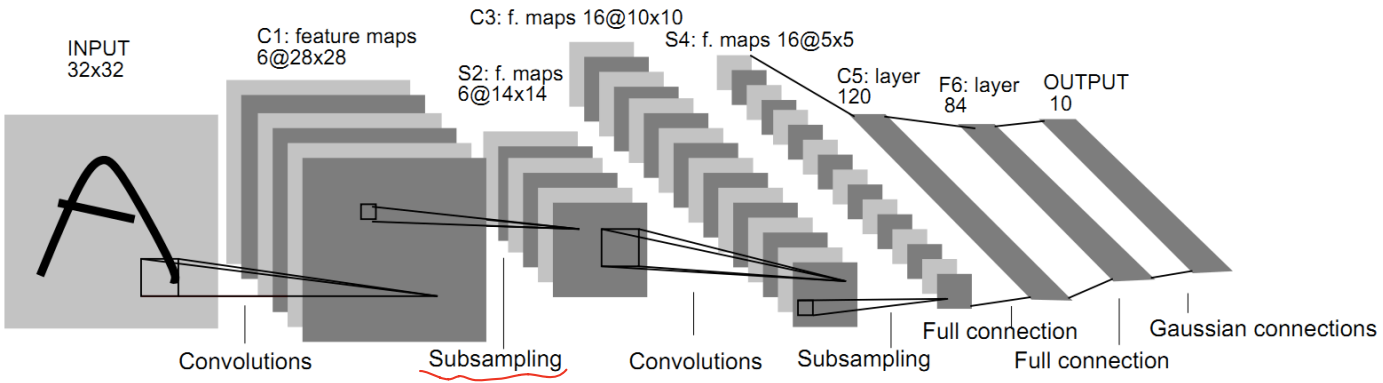
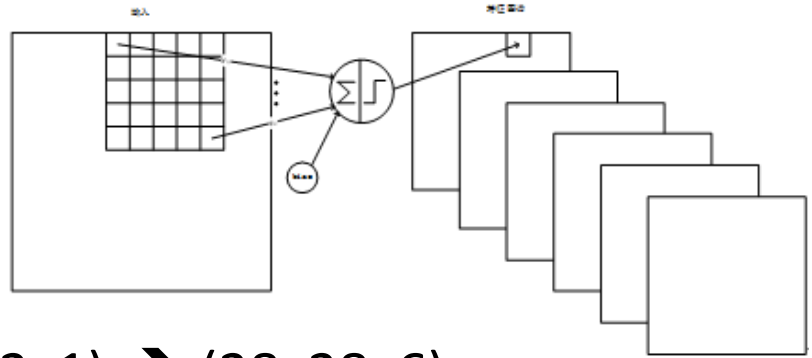


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

LeNet-5: C1 (convolution)



- (32x32x1) ➔ (28x28x6)
  - Kernel size: 5x5
  - # kernels: 6
  - Activation function: sigmoid
  - # parameters: (5x5x1+1)x6 = 156
  - # connections: 156x28x28 = 122,304

bias

Why # parameters and # connections are different

(not fully connect)

- Example: a 4x4 image with a 2x2 kernel
  - A 2x2 kernel has 2*2+1 (bias) = 5 parameters
  - The kernel is applied on 9 different regions of the 4x4 image, each location has 5 connections, so in total we have 9*5=45 connections

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

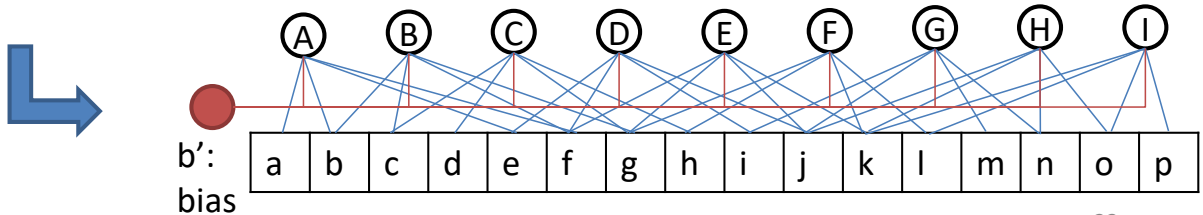
$\otimes$

α	β
γ	δ

$=$

A	B	C
D	E	F
G	H	I

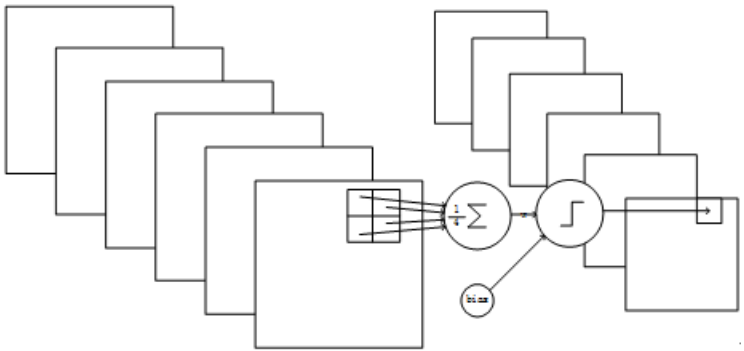
- A= αa+βb+γe+ δf+b'
- B= αb+βc+γf+ δg+b'
- C= αc+βd+γg+ δh+b'
- D= αe+βf+γi+ δj+b'
- E= αf+βg+γj+ δk+b'
- F= αg+βh+γk+ δl+b'
- G= αi+βj+γm+ δn+b'
- H= αj+βk+γn+ δo+b'
- I= αk+βl+γo+ δp+b'



LeNet-5: S2 (subsampling)

shrink the size by 2

- (28x28x6) ➔ (14x14x6)
  - Kernel size: 2x2, stride=2
  - # kernels: 6
  - Activation function: sigmoid in the original paper, but often ignored nowadays
  - # parameters: (weighting, bias)x6 = 12
  - # connections: (2x2+1)x(14x14x6) = 5880



Remark: LeNet-5 use **summation pooling**, but nowadays we usually use max pooling

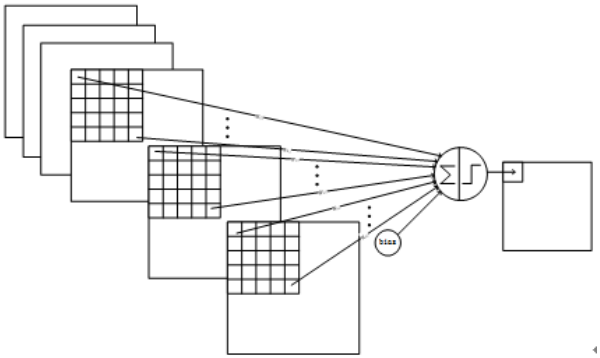
- Summation pooling

  $s = \text{sum of entries in a } 2\times2 \text{ grid}$

  $\text{output} = w_1 \times s + w_2$

highly share

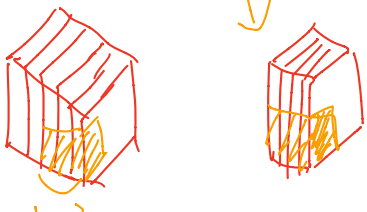
LeNet-5: C3



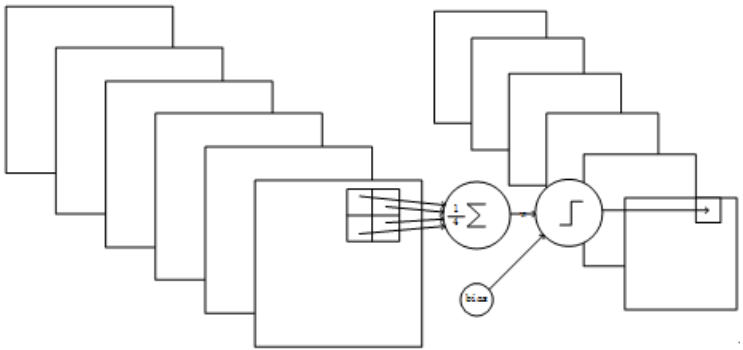
16 kernels

6 input channels	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
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
6				x	x	x			x	x	x	x		x	x	x

- (14x14x6) ➔ (10x10x16)
  - Kernel size: 5x5
  - # kernels: 16
    - Each kernel is only applied to different parts of input channels
  - Activation function: sigmoid
  - # parameters: (5x5x3+1)x6+(5x5x4+1)x6+(5x5x4+1)x3+(5x5x6+1)x1 = 1516
    - The colors map to the above table
  - # connections: 1516x10x10 = 151,600



LeNet-5: S4

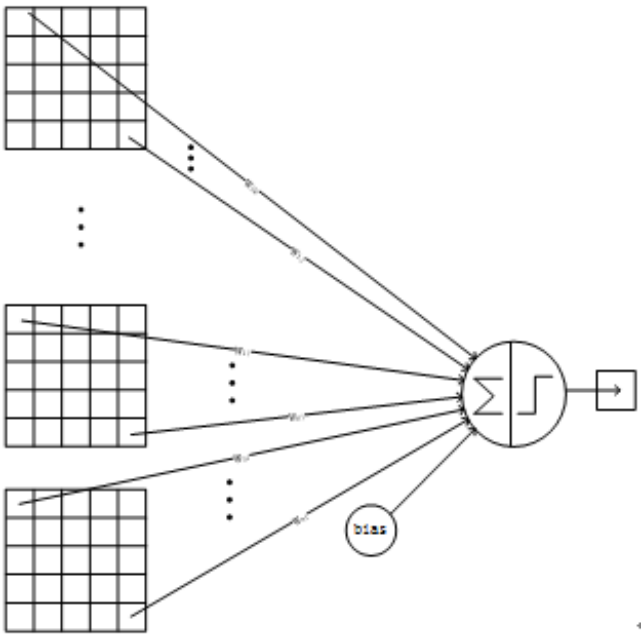


- (10x10x16) ➔ (5x5x16)
  - Kernel size: 2x2, stride=2
  - # kernels: 16
  - Activation function: sigmoid in the original paper, but often ignored nowadays
  - # parameters: (weighting, bias)x16 = 32
  - # connections: (2x2+1)x(5x5x16) = 2000

LeNet-5 use summation pooling, but nowadays we usually use max pooling

LeNet-5: C5

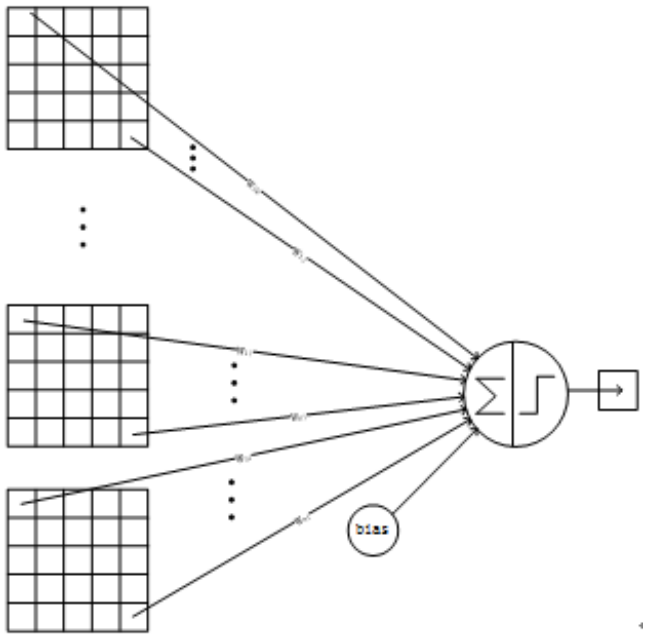
- (5x5x16) ➔ (1x1x120)
  - Kernel size: 5x5
  - # kernels: 120
  - Activation function: sigmoid
  - # parameters: (5x5x16+1)x120=48120
  - # connections: 48120 (fully connected)



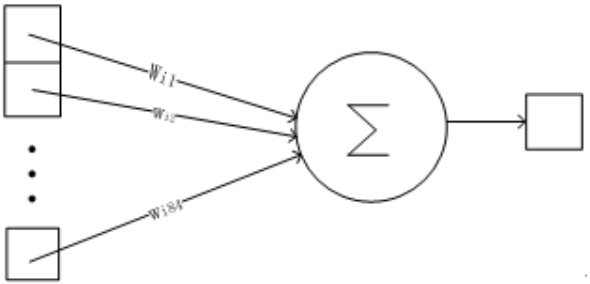
FC

LeNet-5: F6 (fully connected)

- (1x1x120) ➔ (1x1x84)
  - Kernel size: 1x1
  - # kernels: 84
  - Activation function: a customized sigmoid
    - $f(z) = A\tanh(Sz)$
      - *A*: amplifier of the function
      - *S*: slope at the origin
  - # parameters: (1x1x120+1)x84=10164
  - # connections: 10164 (fully connected)



LeNet-5: output layer



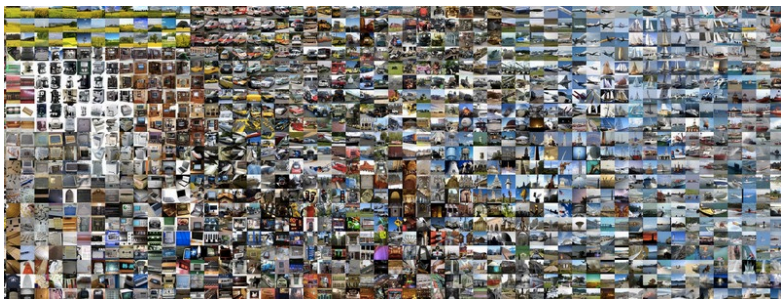
- (1x1x84) ➔ (1x1x10)
  - Kernel size: 1x1
  - # kernels: 10
  - Activation function: none
  - # parameters: (1x1x84)x10=840
    - No bias term because RBF network is applied in the original paper (details ignored). Nowadays usually use softmax function

Summary of CNN

- A special type of feedforward neural network
  - Conv layer
    - Connect to only a small subset of neurons in the neighboring layers
  - Subsampling (pooling) layer
    - Connect to only a small subset of neurons in the neighboring layers
    - Subsampling
  - Fully connected layer
    - Multi-class classification

ImageNet

- # images: ~14M
- # high level categories: 27
- # sub-categories: ~21K
- Collected from Amazon Mechanical Turk



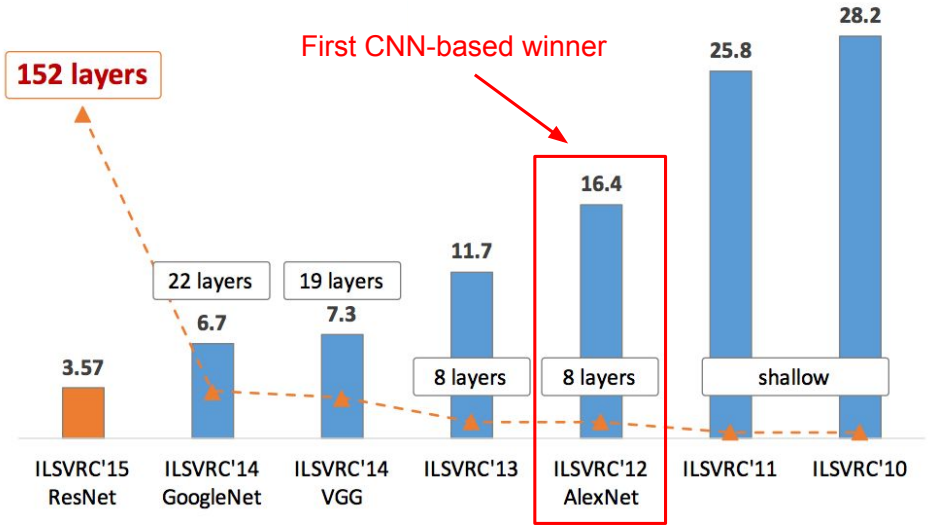
http://image-net.org/about-stats

ILSVRC

- ImageNet Large Scale Visual Recognition Competition
- Annual competition (2010 – 2017)

Top-5 error rate

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

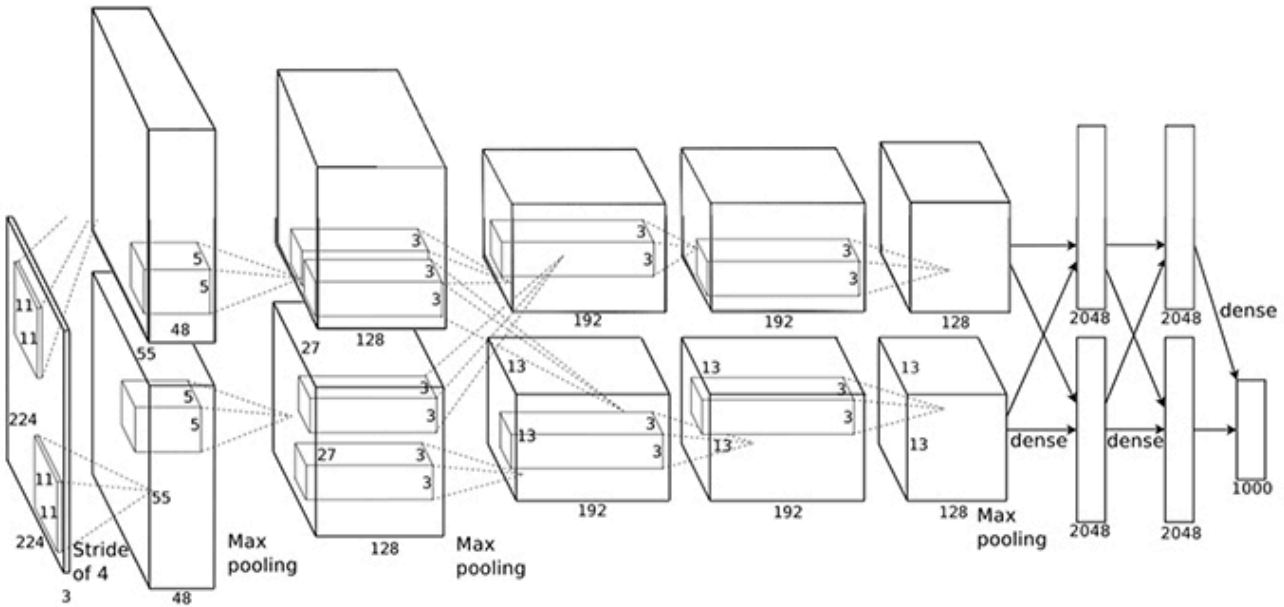


Top-5 error: check if the target label is one of the top 5 predictions

Case study: AlexNet

- Won 2012 ImageNet ILSVRC challenge
  - Top-5 error rate: 16.4% (second place: 26%)
- Important features
  - A deeper LeNet5
  - First use of ReLU
  - Heavy data augmentation
  - Dropout 0.5
  - SGD with momentum 0.9
  - Learning rate 0.01, reduce by 10 manually when val accuracy plateaus
  - Proposed Local Response Normalization Layers (LRN), but this is rarely used recently since usually not helpful

AlexNet architecture (1/2)



AlexNet architecture (2/2)

Layer	Type	# maps	Size	Kernel size	Stride	Padding	Activation
In	Input	3	227X227	-	-	-	-
C1	Conv.	96	55X55	11X11	4	valid	ReLU
S2	Max pool	96	27X27	3X3	2	valid	-
C3	Conv.	256	27X27	5X5	1	same	ReLU
S4	Max pool.	256	13X13	3X3	2	valid	-
C5	Conv.	384	13X13	3X3	1	same	ReLU
C6	Conv.	384	13X13	3X3	1	same	ReLU
C7	Conv.	256	13X13	3X3	1	same	ReLU
F8	Fully conn.	-	4,096	-	-	-	ReLU
F9	Fully conn.	-	4,096	-	-	-	ReLU
out	Fully conn.	-	1,000	-	-	-	Softmax

ReLU vs tanh (reported in AlexNet paper)

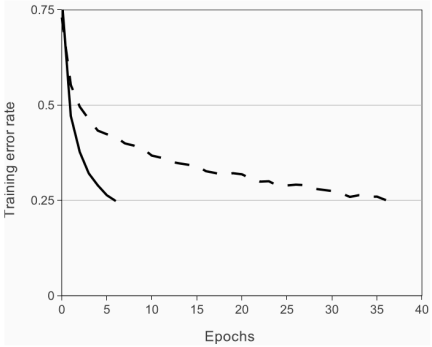


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Case study: ZFNet

- Won 2013 IMageNet ILSVRC challenge
  - Top-5 error rate: 11.7%
- A variant of AlexNet with improved hyper-parameters

ZFNet architecture

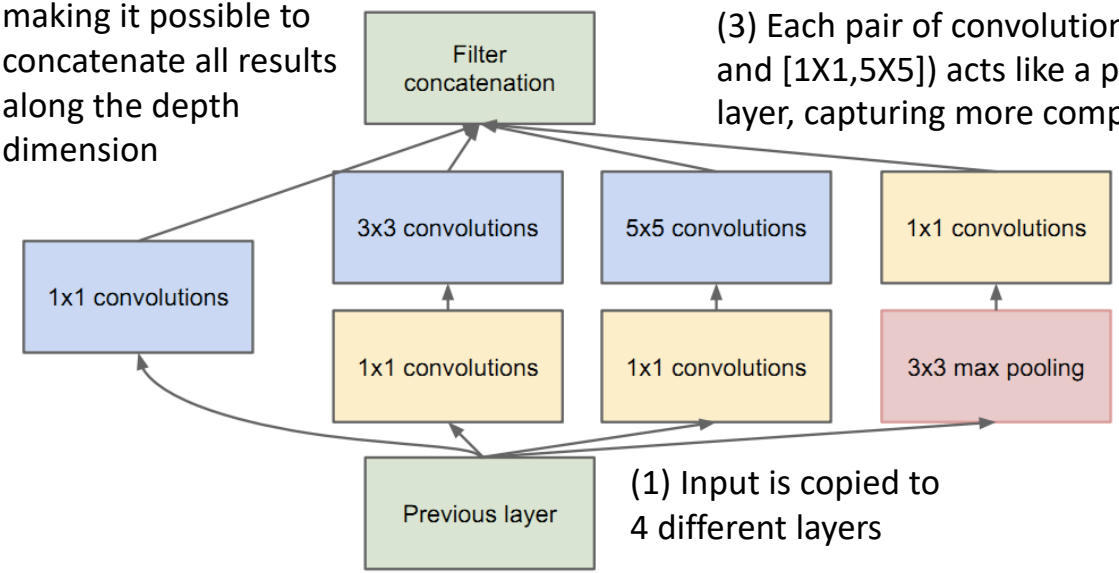
- AlexNet, but
  - Conv1: change from (11X11) stride 4 to (7X7) stride 2
  - Conv3, 4, 5: increase # filters

Case study: GoogLeNet

- Won 2014 ImageNet ILSVRC challenge
  - Top-5 error rate: 6.7%
- Important features
  - Much deeper network (22 layers)
  - Invent inception module to reduce parameters
    - 10 times fewer than AlexNet (6 million vs 60 million)
  - No fully connected layers

Inception module

(4) All outputs have the same height and width, making it possible to concatenate all results along the depth dimension

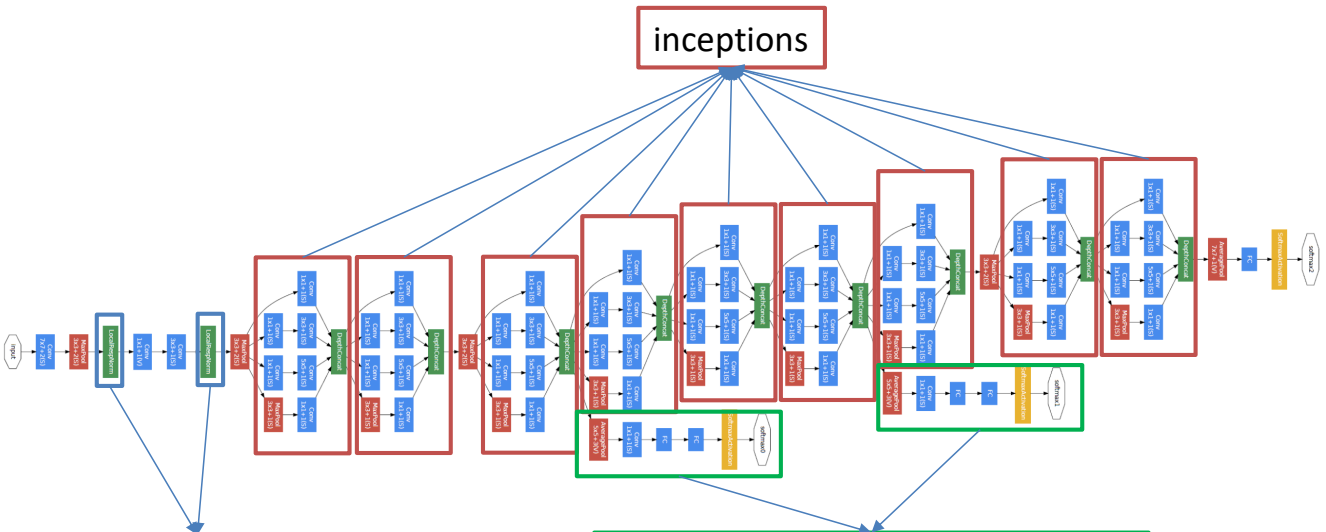


(3) Each pair of convolution layers ([1X1,3X3] and [1X1,5X5]) acts like a powerful convolution layer, capturing more complex patterns

(2) Different kernel size (1X1,3X3,5X5) to capture patterns at different scales

(1) Input is copied to 4 different layers

Full GoogLeNet



Parameters of each layer

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

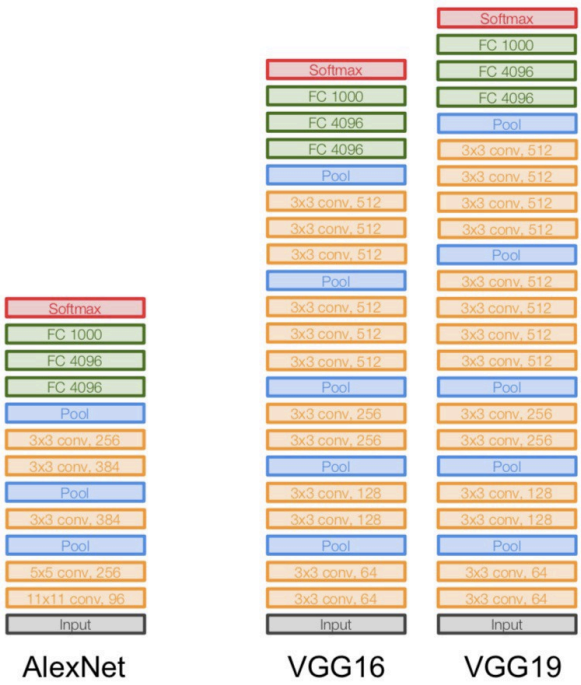
Source: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7

Auxiliary classifier for training

- Each auxiliary classifier is consists of
  - 5x5 average pooling (stride 3)
  - 1x1 conv (128 filters)
  - 1024 FC
  - 1000 FC
  - Softmax
- Used in training, but not used in testing

Case study: VGGNet (Visual Geometry Group)

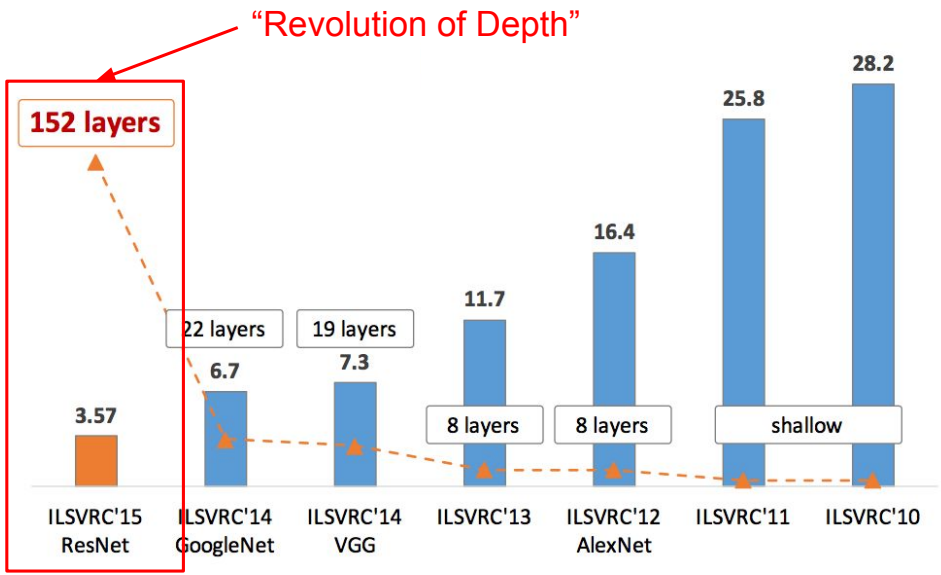
- 2nd in 2014 ImageNet ILSVRC challenge
  - Top-5 error rate: 7.3%
- Important features
  - Small filters
    - Only 3X3 conv, stride 1, pad 1 and 3X3 max pool stride 2
  - deeper networks
    - 16 – 19 layers



Case study: ResNet (Residual Network)

- Won 2015 ImageNet ILSVRC challenge
  - Top-5 error rate: 3.57%
- Important features
  - Extremely deep network (152 layers)
  - Skip connection (a.k.a., shortcut)

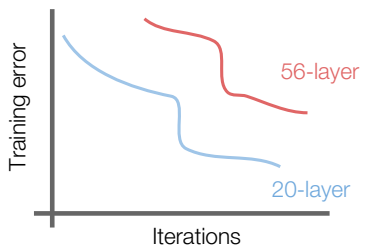
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

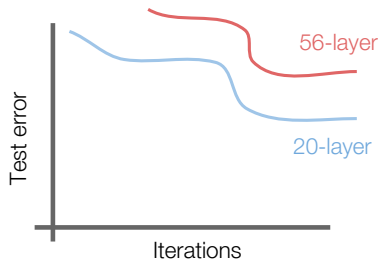


Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

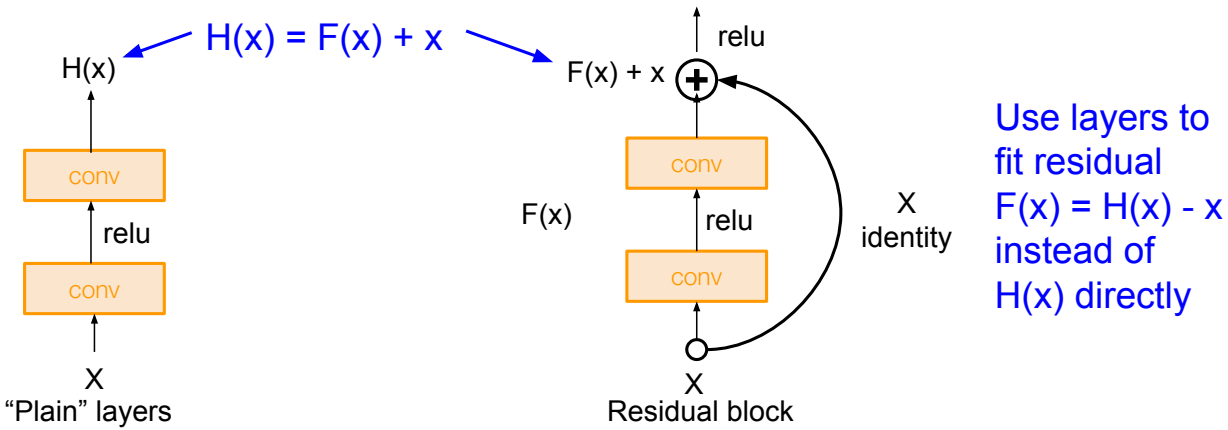
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

[He et al., 2015]

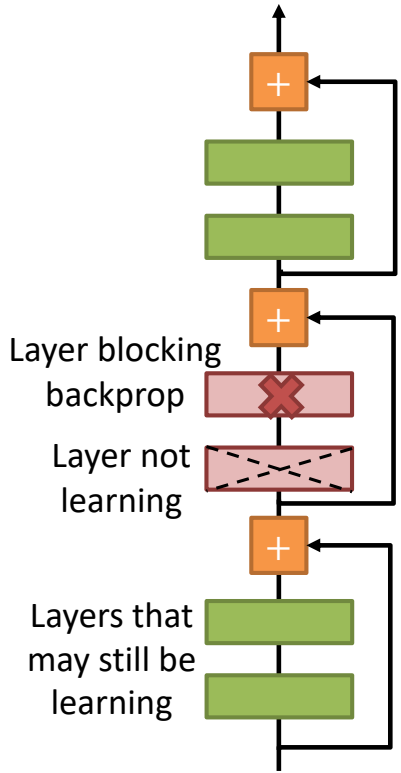
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

Why skip connection works?

- For a regular network, initial weights are close to 0, so outputs are close to 0
- If add skip connections, network outputs a copy of its input, may accelerate training
- If add many skip connections, even if some layers have not started learning yet, the network can start making progress

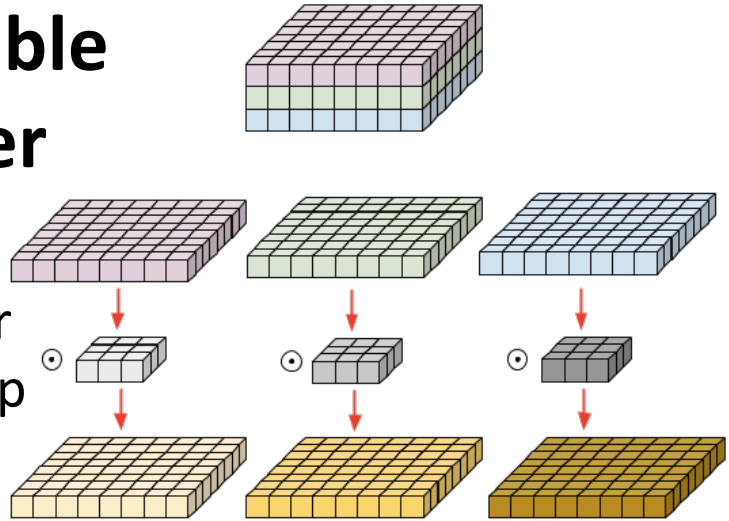


Case study: Xception

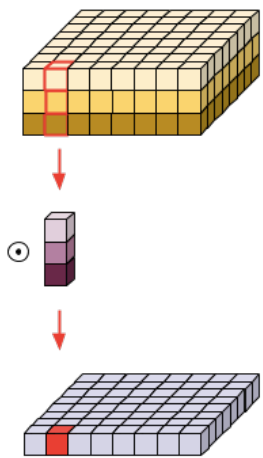
- Important features
  - Merges the idea of GoogLeNet and ResNet
  - Replace the inception module with a special type of layer called "**depthwise separable convolution layer**", with two parts
    - A single spatial filter for each input feature map
    - A regular convolutional layer with 1×1 filters
- Idea: spatial patterns and cross-channel patterns can be modeled separately

Depthwise separable convolution layer

1. Apply a single spatial filter for each input feature map (looks exclusively for spatial patterns)



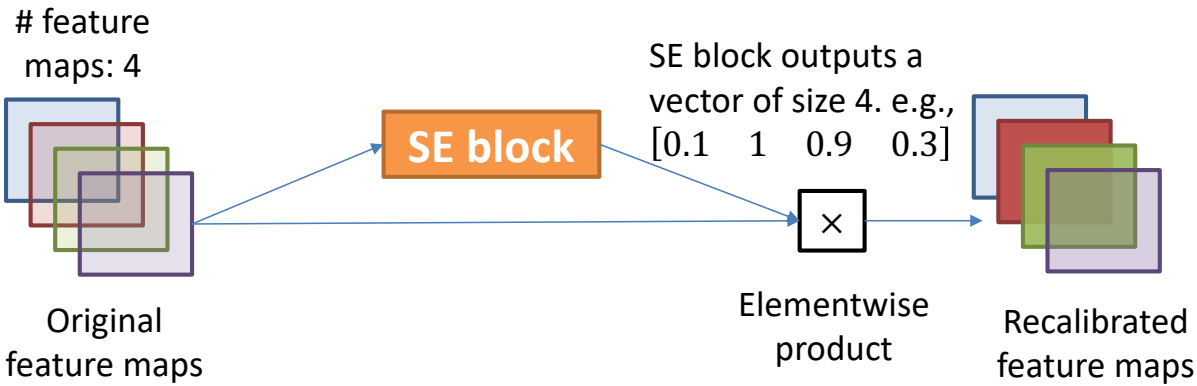
2. Apply a convolutional layer with 1×1 filters (looks exclusively for cross-channel patterns)



Case study: SENet (Squeeze-and Excitation Network)

- Won 2017 ImageNet ILSVRC challenge
  - Top-5 error rate: 2.25%
- Important features
  - Extends existing architectures such as inception and ResNet
    - SE-Inception and SE-ResNet
  - Adds a SE block (a small NN) to every unit in the original architecture

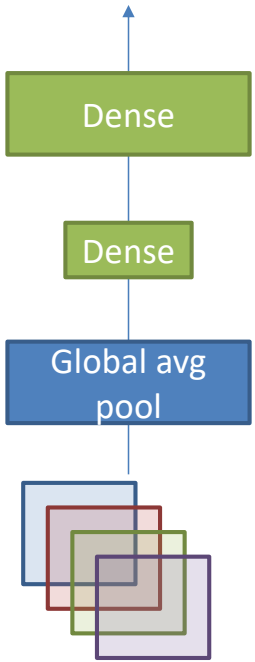
Integrate SE block into existing network architecture



- SE block outputs a vector $\boldsymbol{v} = [v_1, \ldots, v_d]$
  - $d =$ # feature maps
- Recalibrate the feature map $M^{(i)}_{\text{original}}$ by
  - $M^{(i)}_{\text{recalibrated}} = v_i \times M^{(i)}_{\text{original}}$

SE block architecture

- Global avg pool: mean for each feature map
  - If d feature maps, get d avg pools
    - E.g., 256 maps, avg pools $\boldsymbol{a} = [a_1, \dots, a_{256}]^T$
- Consider SE block as an autoencoder
  - Learn $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ such that $\boldsymbol{W}_2 \mathrm{Relu}(\boldsymbol{W}_1 \boldsymbol{a}) \approx \boldsymbol{a}$
    - $\boldsymbol{W}_1 \in R^{c \times d}, \boldsymbol{W}_2 \in R^{d \times c}, c < d$
      - E.g., $\boldsymbol{W}_1 \in R^{16 \times 256}, \boldsymbol{W}_2 \in R^{256 \times 16}$
  - The small vector, $\boldsymbol{m} = \mathrm{Relu}(\boldsymbol{W}_1 \boldsymbol{a})$, is a compact representation of global avg pool $\boldsymbol{a}$
    - So relationship among a_is are captured by $\boldsymbol{m}$
- The output of SE block is $\sigma\big(\boldsymbol{W}_2 \mathrm{Relu}(\boldsymbol{W}_1 \boldsymbol{a})\big)$



Why SE works?

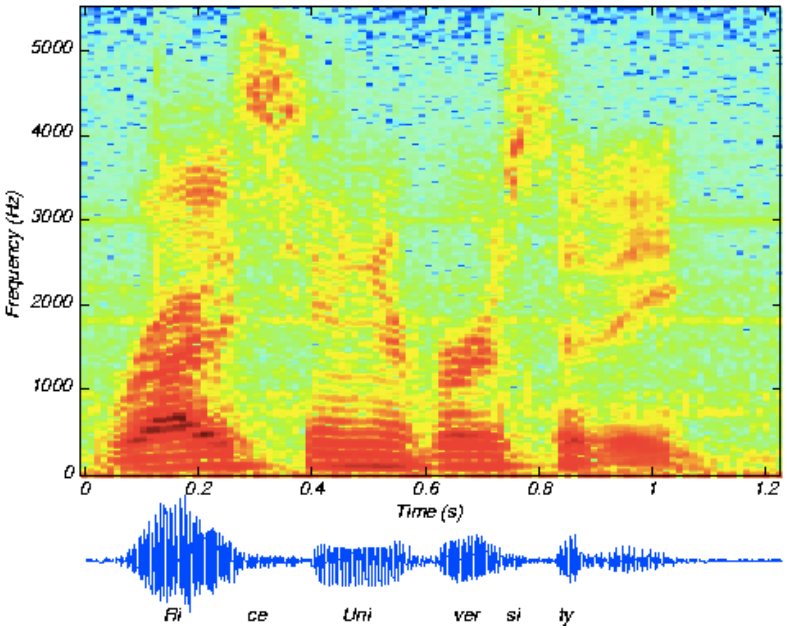
- Assume 3 feature maps capture the patterns of “eyes”, “noses”, and “mouths”
- An SE block may learn they usually appear together in pictures
- If SE sees strong signal for eyes and nose feature maps, SE may boost the strength of the mouth map

AlphaGo

AlphaGo and CNN

- AlphaGo is partially based on a deep CNN
  - Input: raw board position s_t
  - Output:
    1. the probability distribution over moves
    2. the probability of the current player winning in position s_t
  - Model: deep CNN
- Go is viewed as a 19x19 image
  - Many conv layers
  - Seems no pooling layer
- AlphaGo employed many other strategies in addition to CNN

CNN on sound wave



Sound wave in time domain can be transformed into “spectrogram”

CNN on sound wave

- Apply variants of (Deep-)CNN on the spectrogram

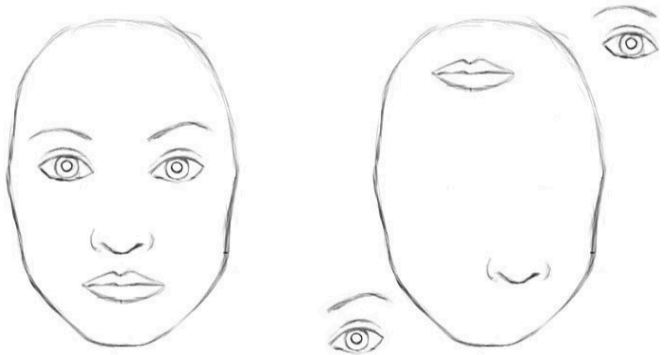
CNN on traffic flow prediction

- Traffic flow on the highway can be considered as a “1D” image
  - # cars within a area is considered as “pixel” values
- Apply variants of (Deep-)CNN

Critics on CNN (by G. Hinton)

- The orientational information is missing
- The relative spatial relationship between components is missing
- Why? ➔ Higher-level features are the weighted sum of the lower-level features
- Hinton proposed Capsule Network (CapsNet) to address some of these issues (in Nov. 2017)

Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in Neural Information Processing Systems. 2017.



Conclusion

- (Deep-)CNN is a special type of feedforward neural network
- Variations of (Deep-)CNN perform extremely well on various computer vision tasks
  - Trend: deeper, but fewer parameters
- Training extra-deep network is difficult
  - Mainly due to optimization strategy
  - Faster and more stable training
    - Improving gradient flow
    - Improving network structure