



LG전자 Deep Learning 과정

Convolutional Neural Networks

Gunhee Kim

Computer Science and Engineering



서울대학교
SEOUL NATIONAL UNIVERSITY

Outline

- Filters and Convolution
- Convolutional Neural network
- Training Tips
- Case Studies

Filters in Images

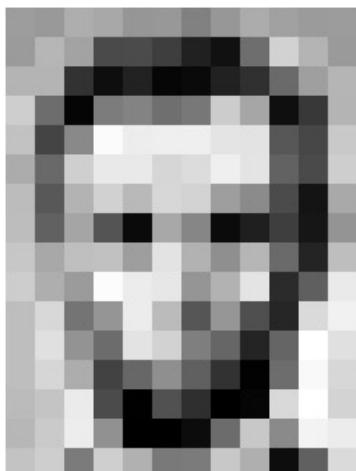
It is a basic tool for image processing

- Transform images, or extract meaningful information from digital images via algorithms using computers

Digital image as a function

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}$$

- $f(x, y)$ gives the intensity [0,255] at position (x, y)



157	153	174	168	150	152	129	151	172	161	155	166
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	48	105	159	181
206	109	5	124	151	111	120	204	166	15	56	180
194	68	137	251	257	259	259	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	177	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	167	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	297	177	121	123	200	175	13	96	218

157	153	174	168	180	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	257	259	259	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	177	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	167	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	297	177	121	123	200	175	13	96	218

cf. A color image
is a function

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

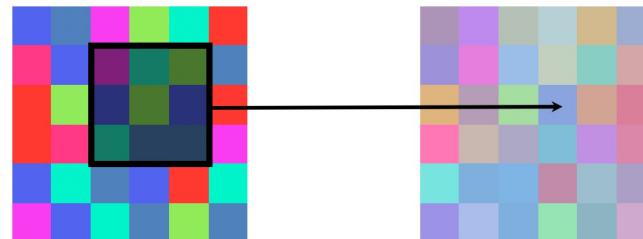
$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Filters in Images

Filters change the pixel value of images

- Apply a function to digital image
- Modify image properties and/or to extract valuable information
- ex. Noise reduction, blurring, sharpening, edge detection,

Neighborhood operation



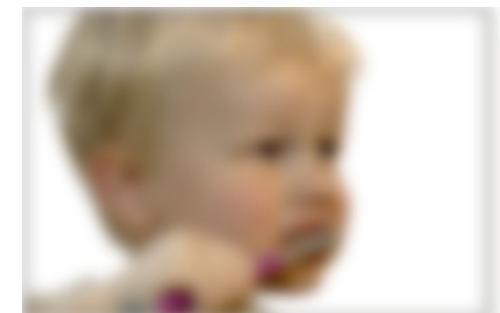
Convolution!



Image



Edge detection

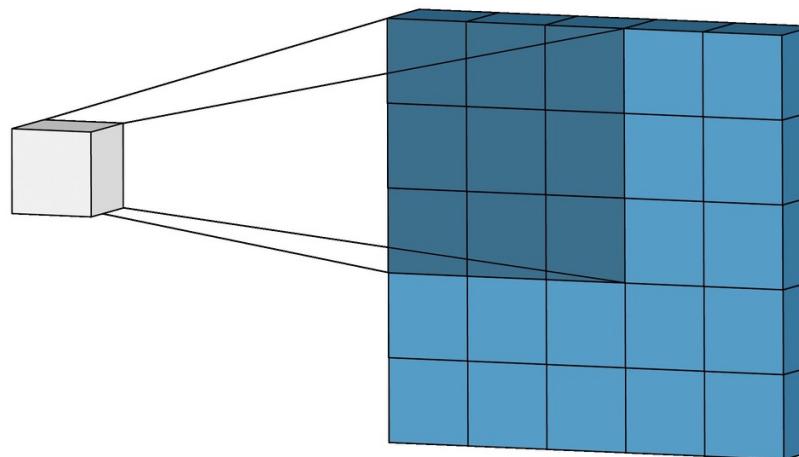


Blur

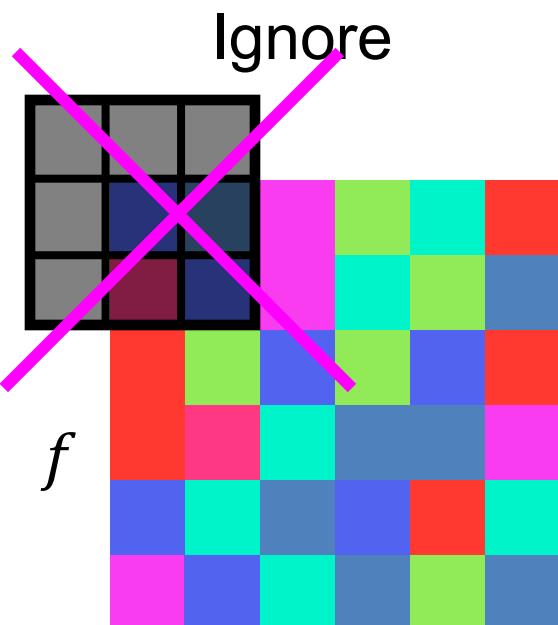
Convolution

Multiply each kernel (filter) value by the corresponding pixel value

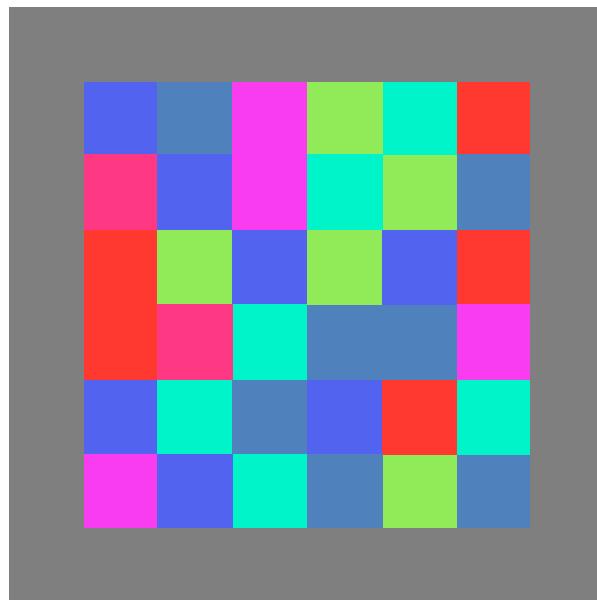
Input	Kernel	Output													
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	\ast	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3
0	1	2													
3	4	5													
6	7	8													
0	1														
2	3														
	=	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43									
19	25														
37	43														



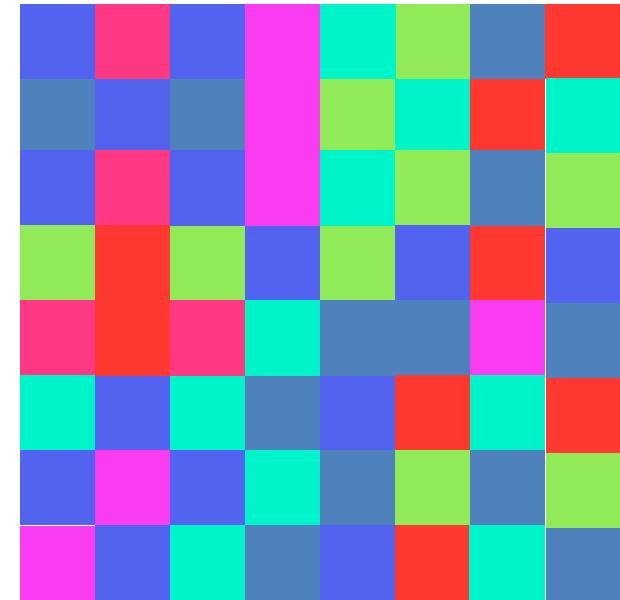
Convolution – Border Problem



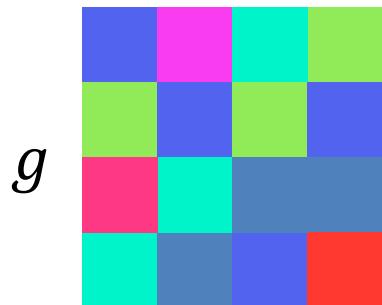
Pad with
constant values



Pad with
reflection



EX) Pad with zeros



Convolution

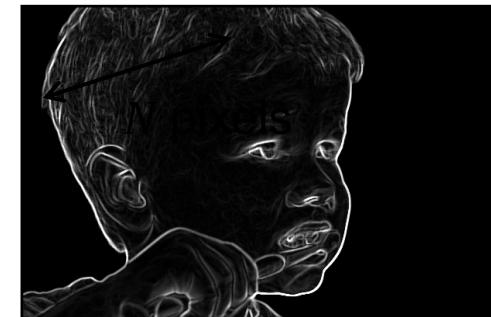
Image processing can be done by convolution with filters (kernels)



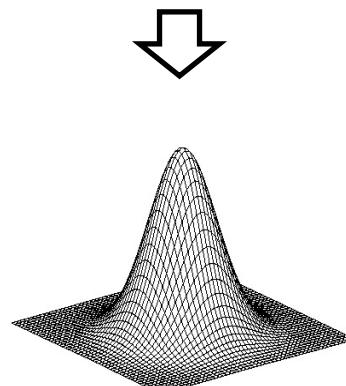
Image



-1	0	1	1	2	1
-2	0	2	0	0	0
-1	0	1	-1	-2	-1

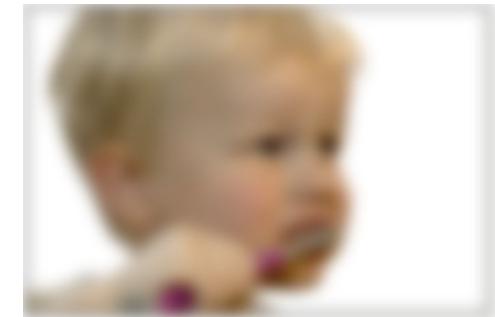


Edge detection



1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1



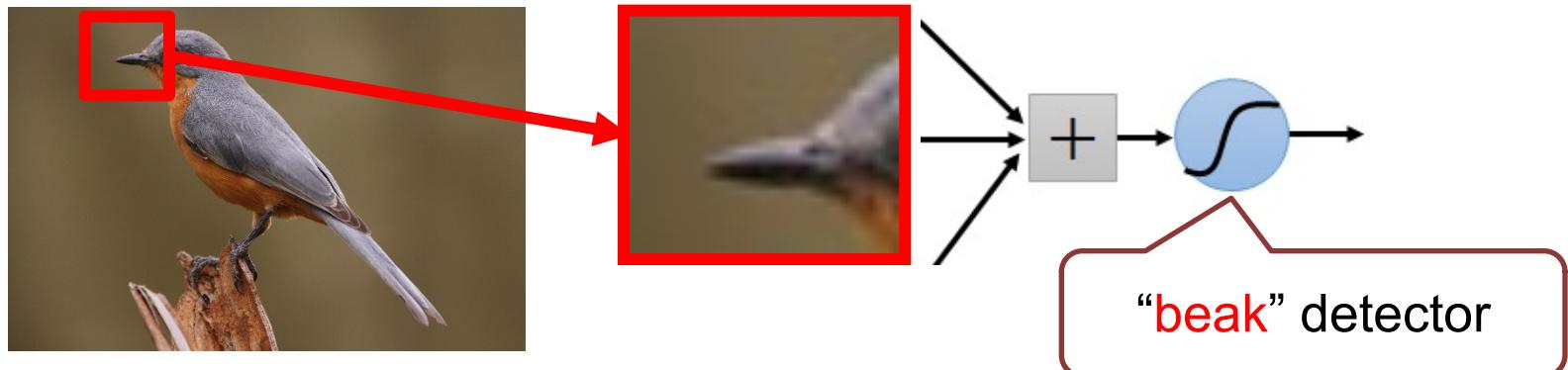
Blur

Why Do CNNs Need Filters?

Some patterns for recognition are much smaller than the whole image

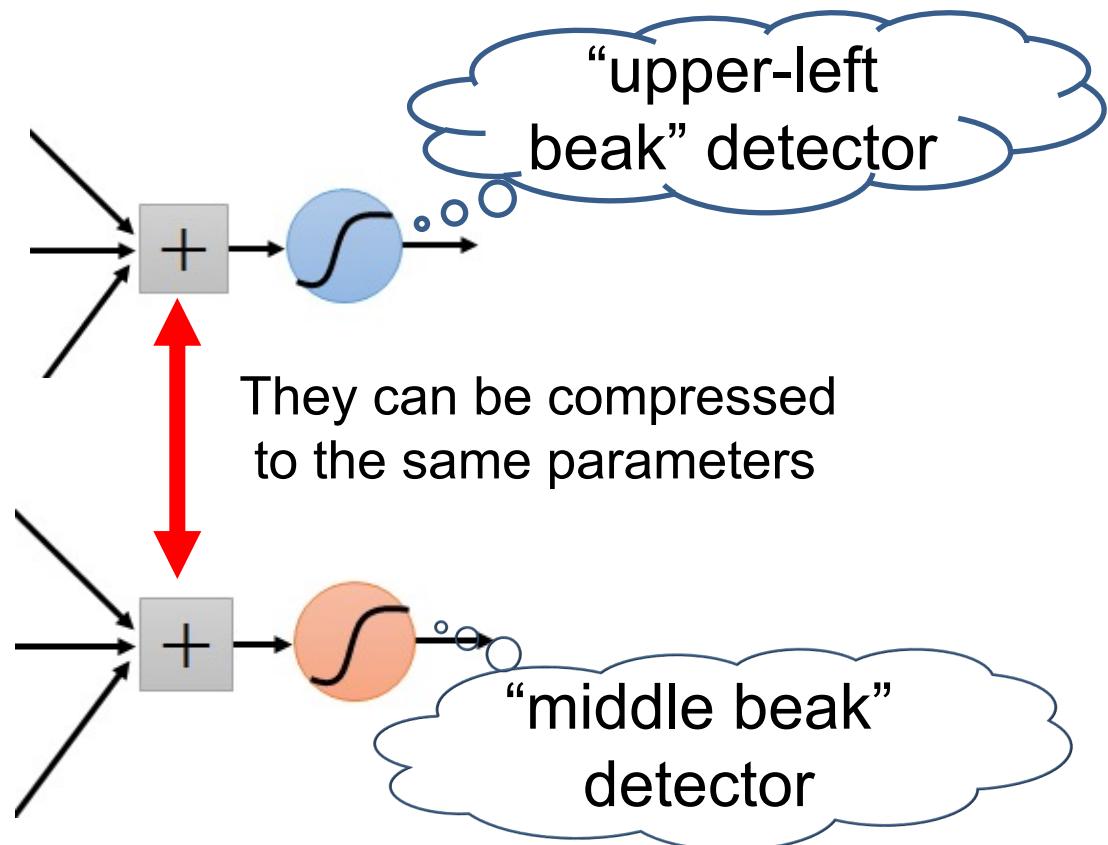
Also want to represent a small region with fewer parameters

Beak detector is a filter in some upper layers of CNN!



Why Do CNNs Need Filters?

What about training a lot of such **small** detectors and each detector must move around

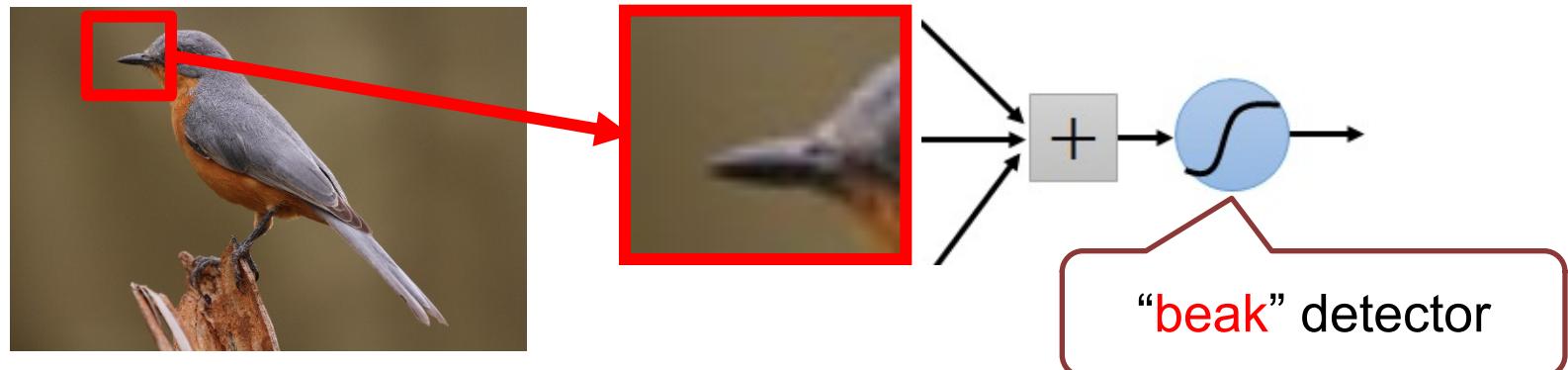


Why Do CNN Need Filters?

However, we do not know which patterns are important to detect a bird beforehand

Let CNNs learn a set of filters!

- More correctly, a hierarchy of sets of filters



Outline

- Convolutional Neural network
 - Training Tips
 - Parameter Updates
 - Case Studies

Convolutional Neural Network (ConvNet)

Some background knowledge

- Convolution of the image with filters
- Back propagation + gradient based optimization

A ConvNet consists of a number of convolutional + subsampling (i.e. pooling) + fully-connected layers

- In most cases, input is a 2D images and output is class labels

ConvNets are everywhere in computer vision

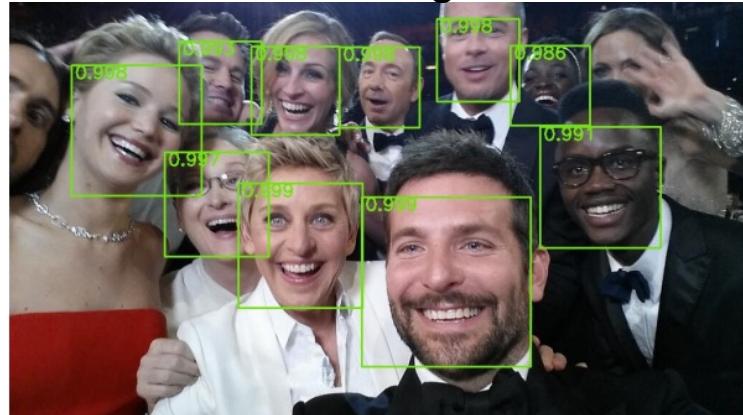
Classification



Retrieval

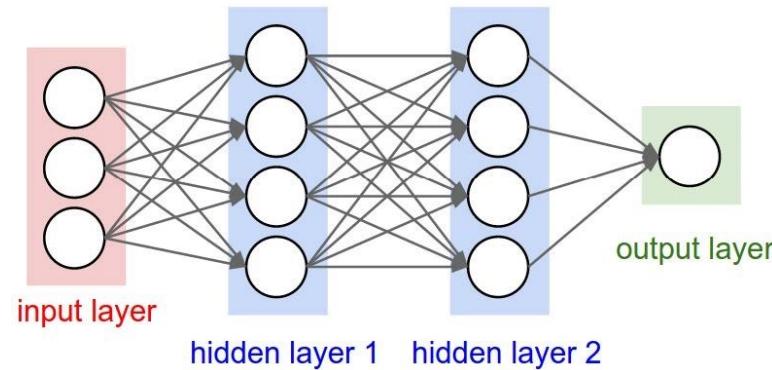


Face recognition

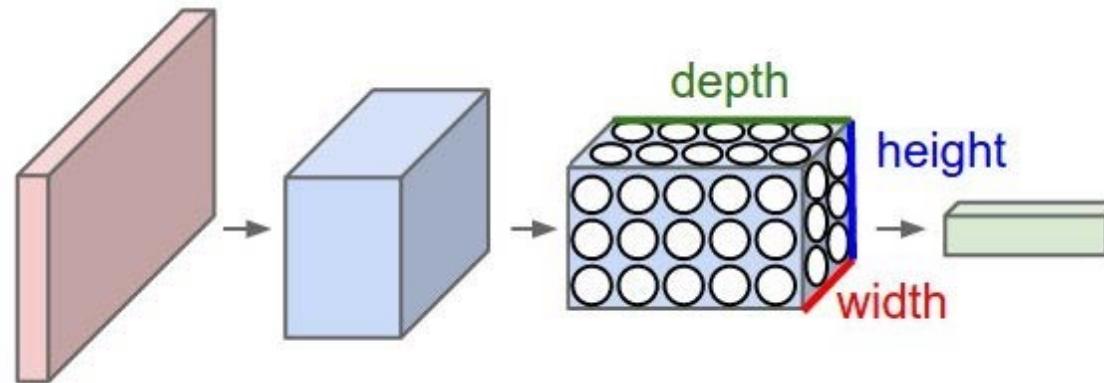


ConvNet Representation

In previous multi-layer NN



Each layer of ConvNet is represented by 3D volume of neurons (width, height, depth)



ConvNet Representation

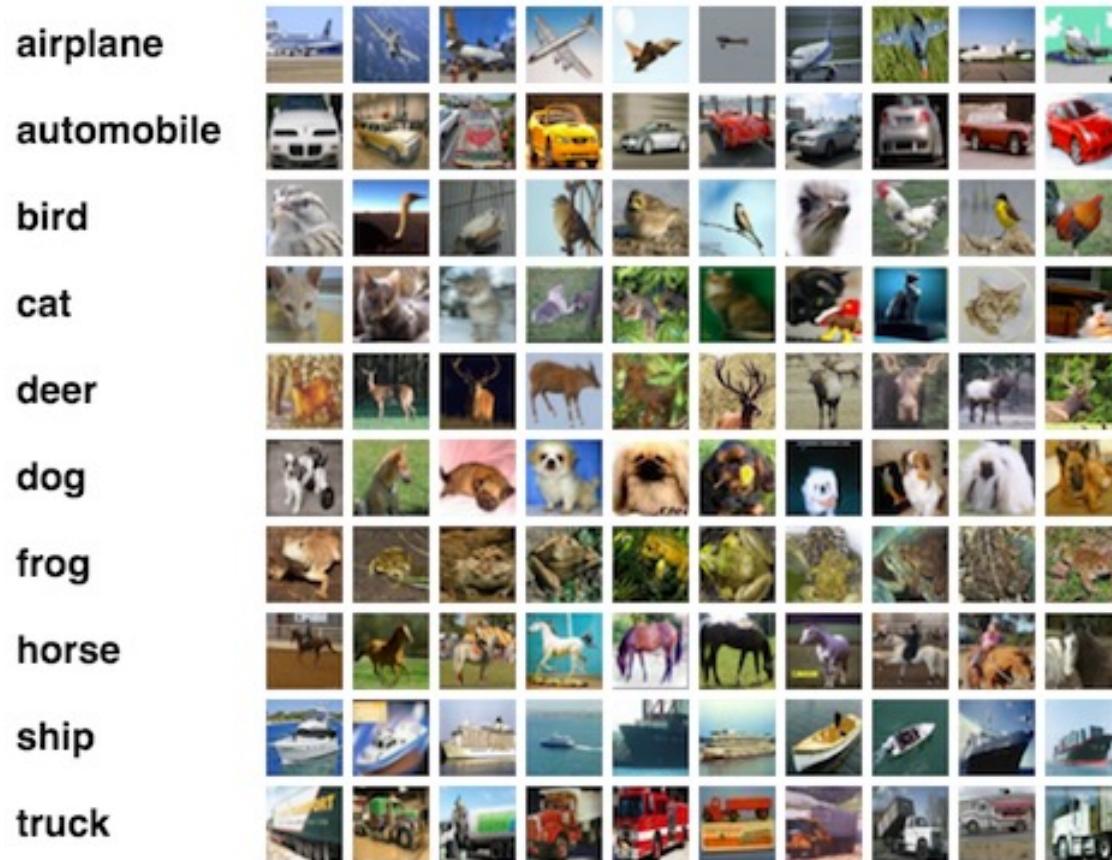
1. INPUT layer: Raw pixel values of the image
2. COV (Convolutional) layer: Output of neurons that are connected to local regions in the input
 - Computing a dot product between their weights and the region they are connected to in the input volume
3. Activation layer (RELU): Element-wise activation function
4. POOL layer: Downsampling (summation) operation
5. FC (Fully-connected) Layer: Computing class score

Parameters

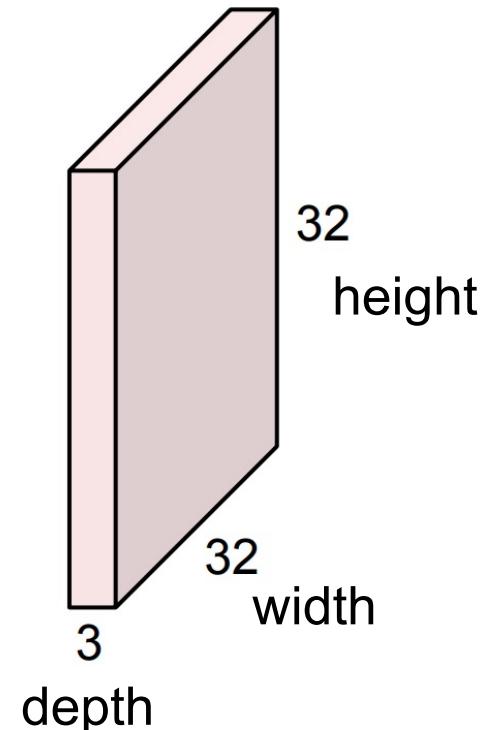
- CONV/FC have parameters (weights and bias of neurons)
- RELU/POOL have NO parameters

Input Layer

Ex. CIFAR-10: Images of 10 classes with 32x32x3 (RGB)



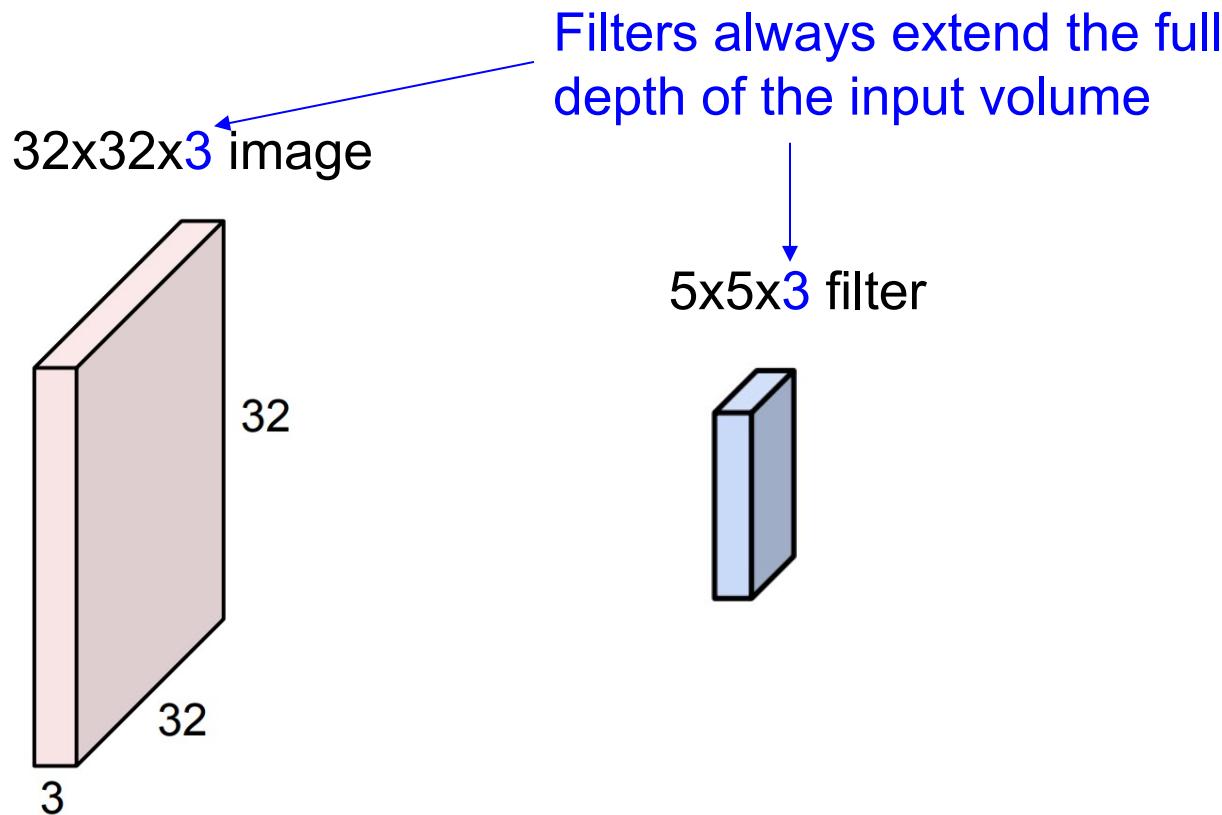
- Input layer (Raw pixels of an image)



Convolutional Layer

Convolve the filter with the image

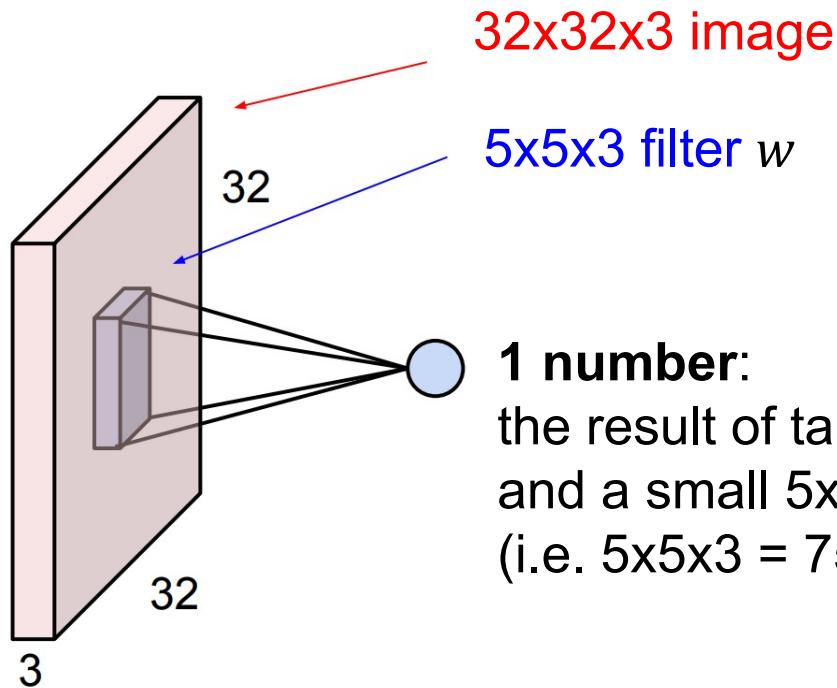
- Slide over the image spatially, computing dot products



Convolutional Layer

Convolve the filter with the image

- Slide over the image spatially, computing dot products



1 number:

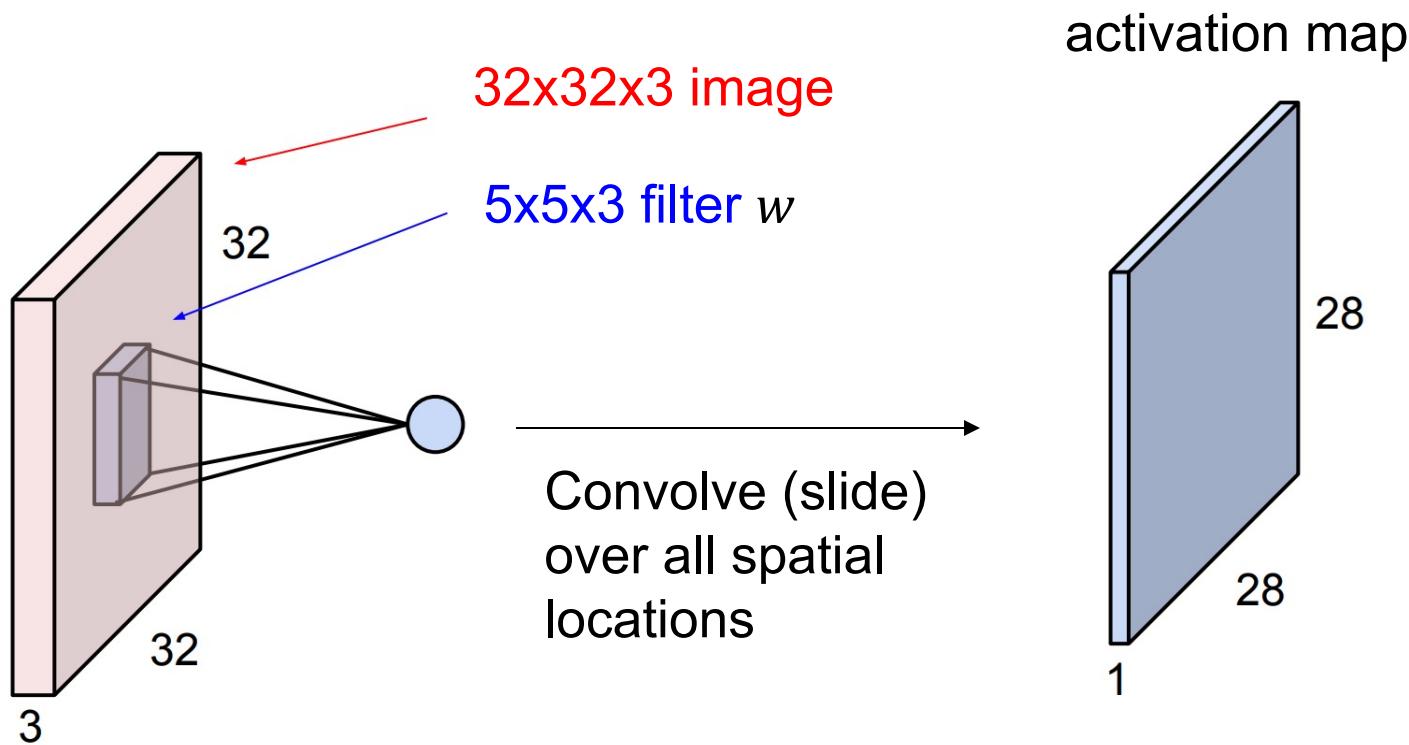
the result of taking a dot product between the filter
and a small $5 \times 5 \times 3$ chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

Convolutional Layer

Convolve the filter with the image

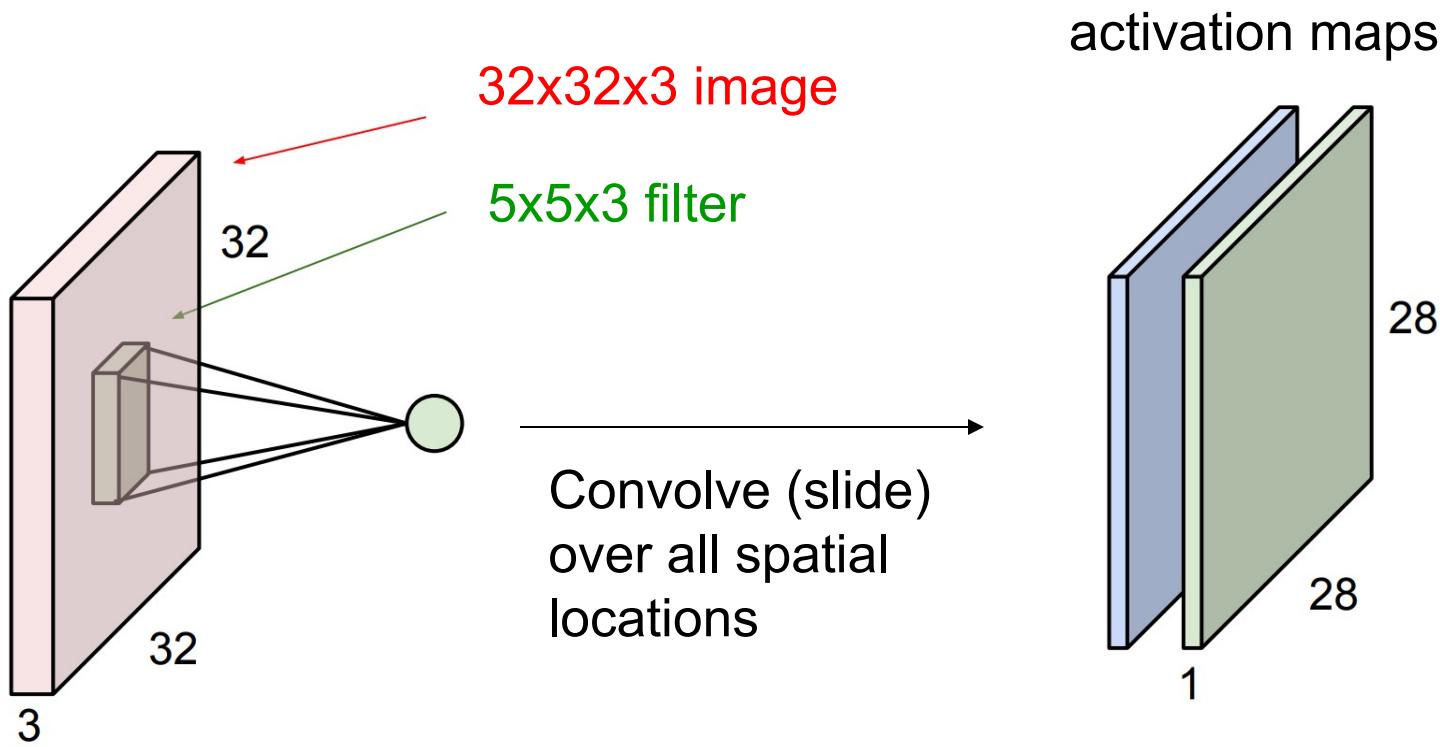
- Slide over the image spatially, computing dot products



Convolutional Layer

Convolve the filter with the image

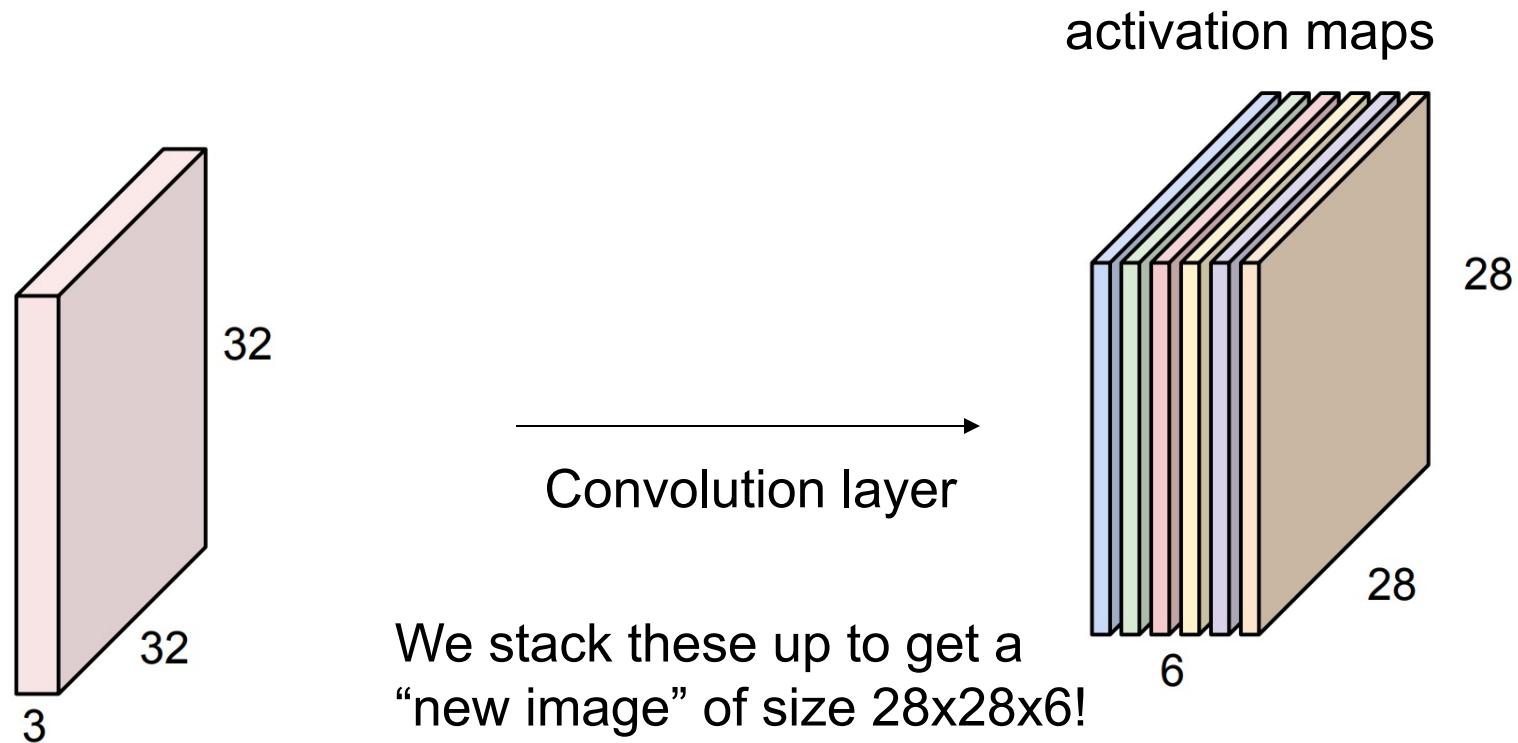
- Slide over the image spatially, computing dot products
- Consider a second, green filter



Convolutional Layer

If we had 6 5x5 filters, we'll get 6 separate activation maps

- The 6 5x5 filters should be learned from data during training
- a.k.a. kernels, receptive fields

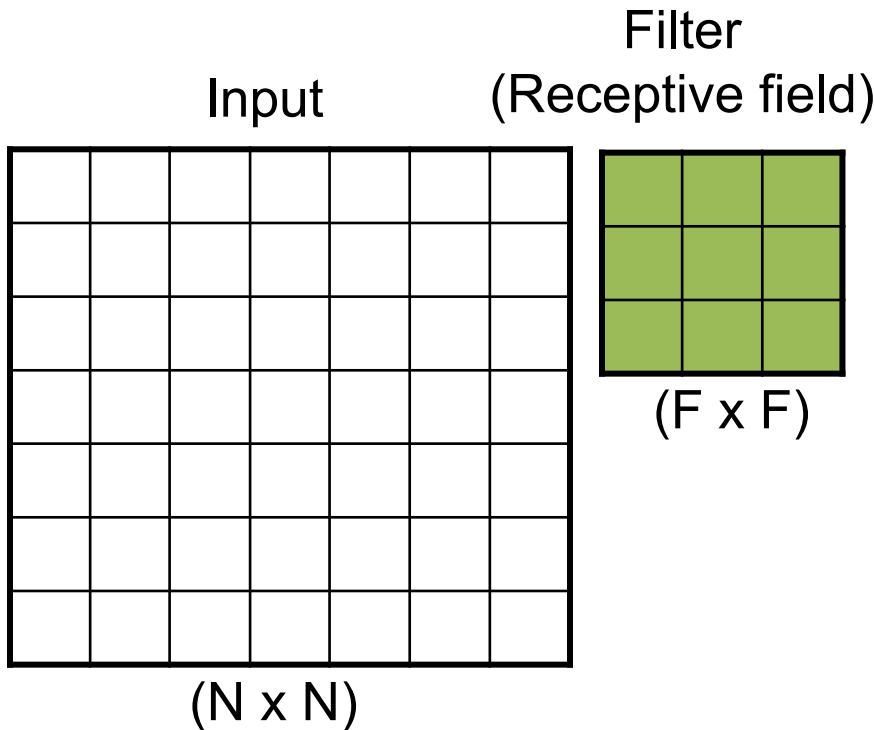


Convolutional Layer

A closer look at spatial dimension

Stride: distance between consecutive filter application

An example



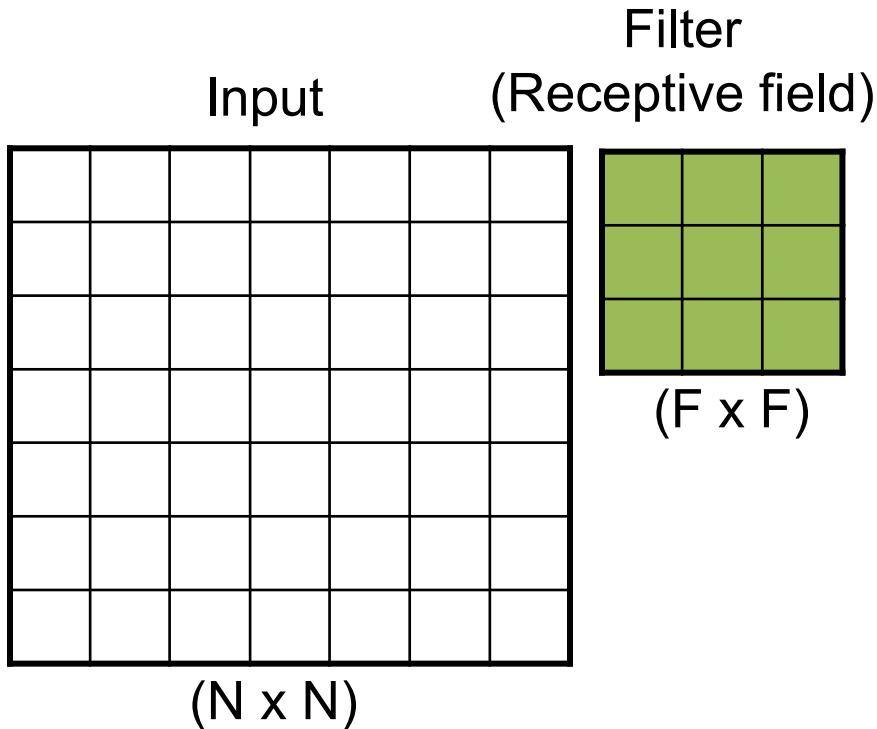
- 7x7 input (spatially) and 3x3 filter
- with stride 1?
→ 5x5 output
- with stride 2?
→ 3x3 output
- with stride 3?
→ **doesn't fit!**
cannot apply 3x3 filter on 7x7 input with stride 3

Convolutional Layer

A closer look at spatial dimension

Stride: distance between consecutive filter application

An example



- 7x7 input (spatially) and 3x3 filter
- Output size: $(N - F) / \text{Stride} + 1$
- e.g. Stride 1 $\rightarrow (7-3)/1+1=5$
Stride 2 $\rightarrow (7-3)/1+1=3$
Stride 3 $\rightarrow (7-3)/1+1=2.33?$

Convolutional Layer

In practice: common to zero pad the border

- Pad the input with zeros spatially on the border
- In general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$ (will preserve size spatially)

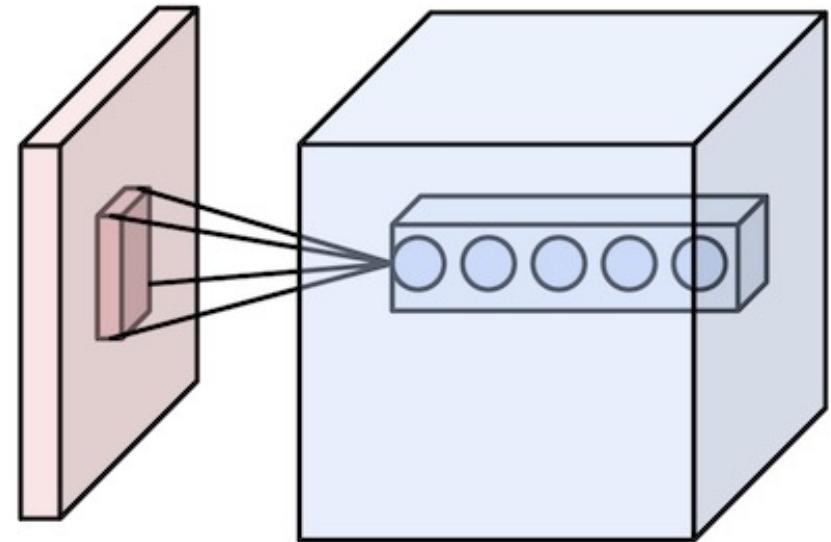
0	0	0	0	0				
0								
0								
0								
0								

- Input 7×7 , filter 3×3 , stride 1, zero-pad 1. What is the output?
- $7 \times 7 \rightarrow$ Preserved size!
- e.g. $F=3 \rightarrow$ zero-pad with 1
 $F=5 \rightarrow$ zero-pad with 2
 $F=7 \rightarrow$ zero-pad with 3
- No headaches when sizing architecture

Example of Convolutional Layer

Question

- Input volume: **32x32x3**
- Filters: **5x5**, stride **1**
- Number of filters: **5**



Answer

- Output size: $(32 - 5)/1 + 1 = 28$
- Output volume: $28 \times 28 \times 5$
- How many weights for each of $28 \times 28 \times 5$ filters? $5 \times 5 \times 3 = 75$
- Total # of weights: 294,000 ($= (28 \times 28 \times 5) \times (5 \times 5 \times 3)$)

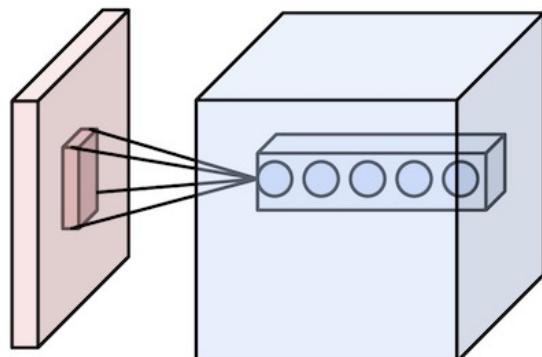
Parameter Sharing

Input layer of [32x32x3] with 30 5x5 filters at stride of 1 and pad 2

- Output volume: [32x32x30]
- Each neuron has 5x5x3 (=75) weights (Too high!)
- Total number of weights: $(32 \times 32 \times 30) \times 75 \sim \mathbf{2.3 \text{ millions}}$

Parameter sharing constraints the neurons in each depth slide to use the same weights

- Total number of weights: $30 \times 75 = \mathbf{2250}$



30 Examples of trained weights of size [5x5x3]

Use the same filter across all spatial location!

Activation Layer

An activation layer follows every CONV layer

- The most common activation function is ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

- ReLU's purpose is to introduce non-linearity in our ConvNet
- Point-wise operation with no parameter

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

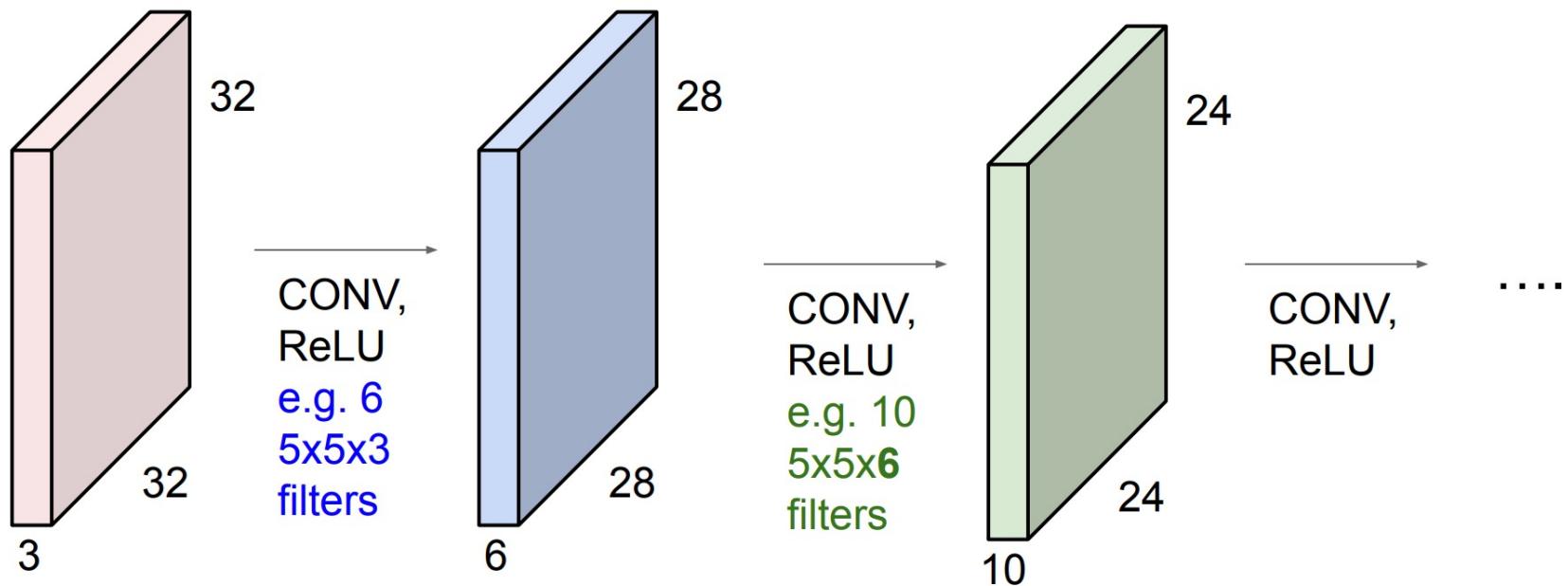


0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77



ConvNet

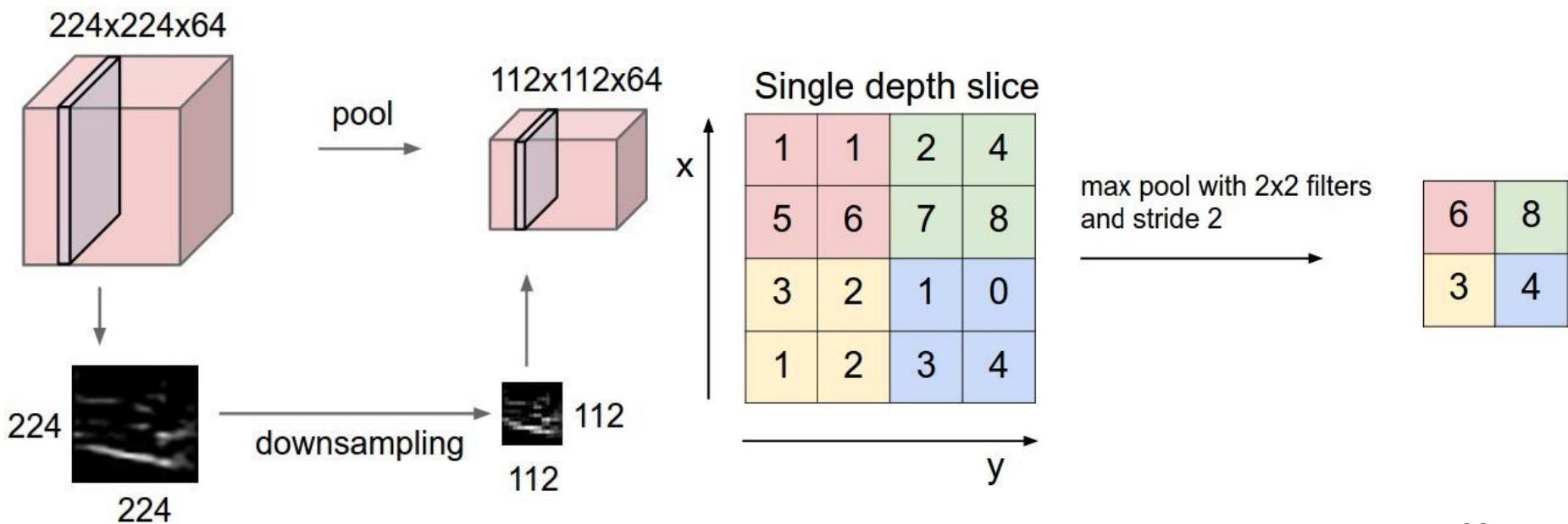
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Pooling Layer

Make the representation smaller and more manageable without losing too much information

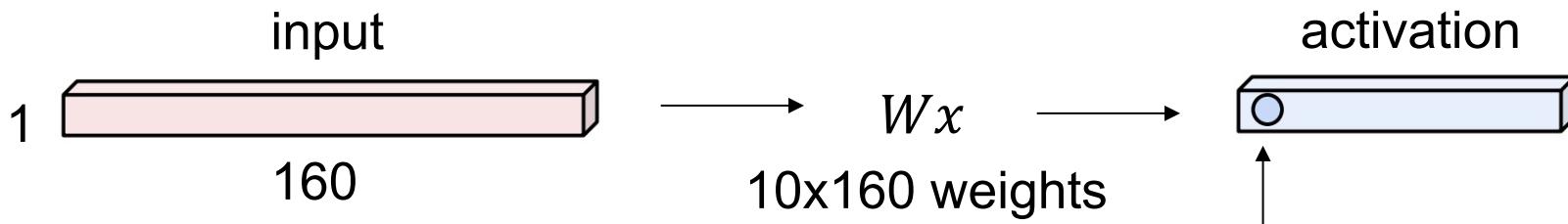
- Operates over each activation map independently:
- Computes MAX operation (most common)
- Average pooling or L2-norm pooling (less used)



Fully Connected Layer

Takes the feature map (the output of Conv/POOL layers) as input and predicts the label to describe the image

- e.g. Given a $4 \times 4 \times 10$ feature map as input, stretch to 160×1

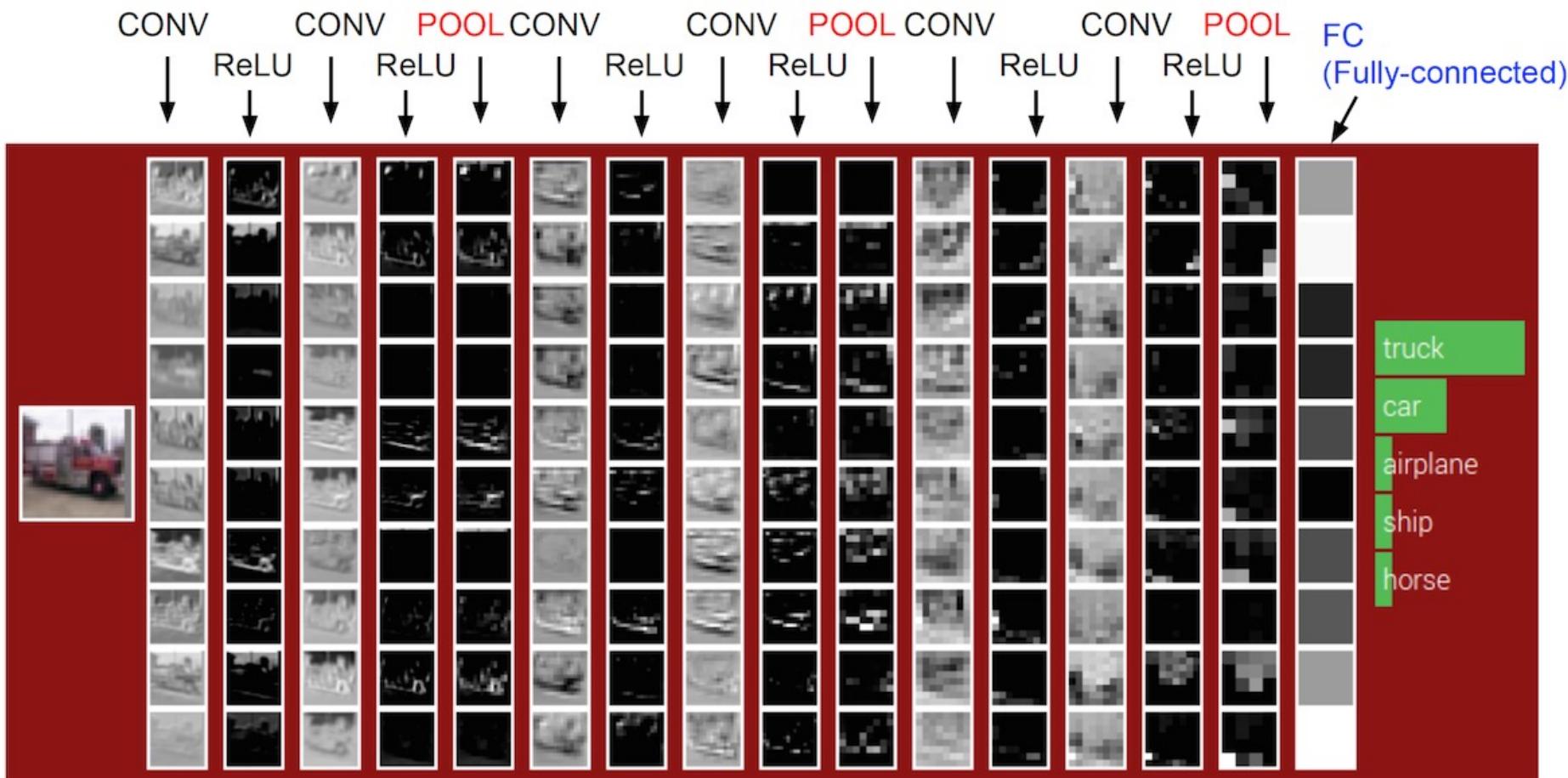


1 number:
the result of taking a dot product between a row of W and the input (a 160-dimensional dot product)

Layer Patterns

Most common form of ConvNet architecture

- Stacks a few CONV-RELU layers, follows them with POOL layers, repeats this pattern, finally adds FC layer



CNNs In Practice

Prefer a stack of small receptive field CONV to one large CONV layer

- Ex) Three CONV layer consisting of 3x3 filters vs one CONV layer consisting 7x7 filters
 - Which is better? The former is BETTER!
1. Filtering is a linear function over the input. Thus, three stacks of CONV layer contain non-linearity to make features more expressive
 2. Fast computation!

Ex. If all volumes have depths of C ,

$$7 \times 7 \text{ CONV layer: } C \times (7 \times 7 \times C) = 49C^2$$

$$3 \text{ } 3 \times 3 \text{ CONV layer: } 3 \times (C \times (3 \times 3 \times C)) = 27C^2$$

Rule of Thumbs for Sizing CNNs

Input layer: divisible by 2 many times

- 32 for CIFAR-10, 64 or 96 for STL-10, 224, 384, 512 for AlexNet

CONV layer: Use small filters (3x3 or at most 5x5) with zero-padding of P=1 at stride of S=1

- Smaller strides work better in practice
- Ex. Input volume of size $[W_i \times H_i \times D_i]$ using K number of FxF filters at stride of S
- Output volume: $[W_o \times H_o \times D_o]$
- $W_o = (W_i - F)/S + 1, H_o = (H_i - F)/S + 1, D_o = K$

POOL layer: Use pooling size of 2x2 with stride of S=2

- 3/4 of the activations will be discarded

Go deep! Use small ones multiple times instead of big

Sizing Example

Input layer: [32x32x3]

1. CONV with 10 3x3 filters, stride 1, pad 1 → [32x32x10]
 - New parameters: $(3*3*3)*10+10 = 280$
2. RELU
3. CONV with 10 3x3 filters, stride 1, pad 1 → [32x32x10]
 - New parameters: $(3*3*10)*10+10 = 910$
4. RELU
5. POOL with 2x2 filters, stride 2 → [16x16x10]
 - New parameters: 0



Sizing Example

6. CONV with 10 3x3 filters, stride 1, pad 1 → [16x16x10]

- New parameters: $(3 \times 3 \times 10) \times 10 + 10 = 910$

7. RELU

8. CONV with 10 3x3 filters, stride 1, pad 1 → [16x16x10]

- New parameters: $(3 \times 3 \times 10) \times 10 + 10 = 910$

9. RELU

10. POOL with 2x2 filters, stride 2 → [8x8x10]

- New parameters: 0



Sizing Example

11~15. Once again CONV-RELU-CONV-RELU-POOL → [4x4x10]

16. FC layer to 10 neurons (each of which for class score)

- New parameters: $10 \times 4 \times 4 \times 10 + 10 = 1600$

Done!

GPU memory constraints!

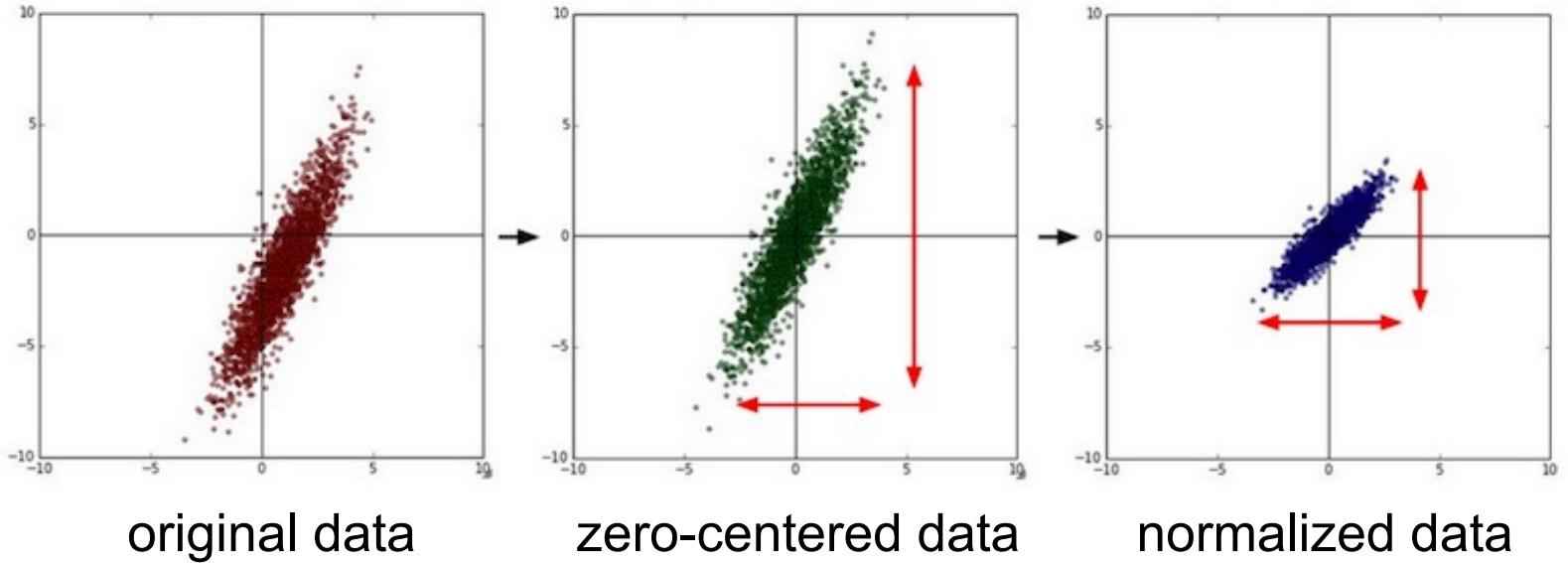
- Ex. 224x224x3 image with three 3x3 CONV layers with 64 filters each and padding 1 → [224x224x64] = 72MB
- Most GPUs have 3/4/6GB memory. Tesla K40 = 12GB!



Outline

- Convolutional Neural network
- Training Tips
- Case Studies

Data Preprocessing



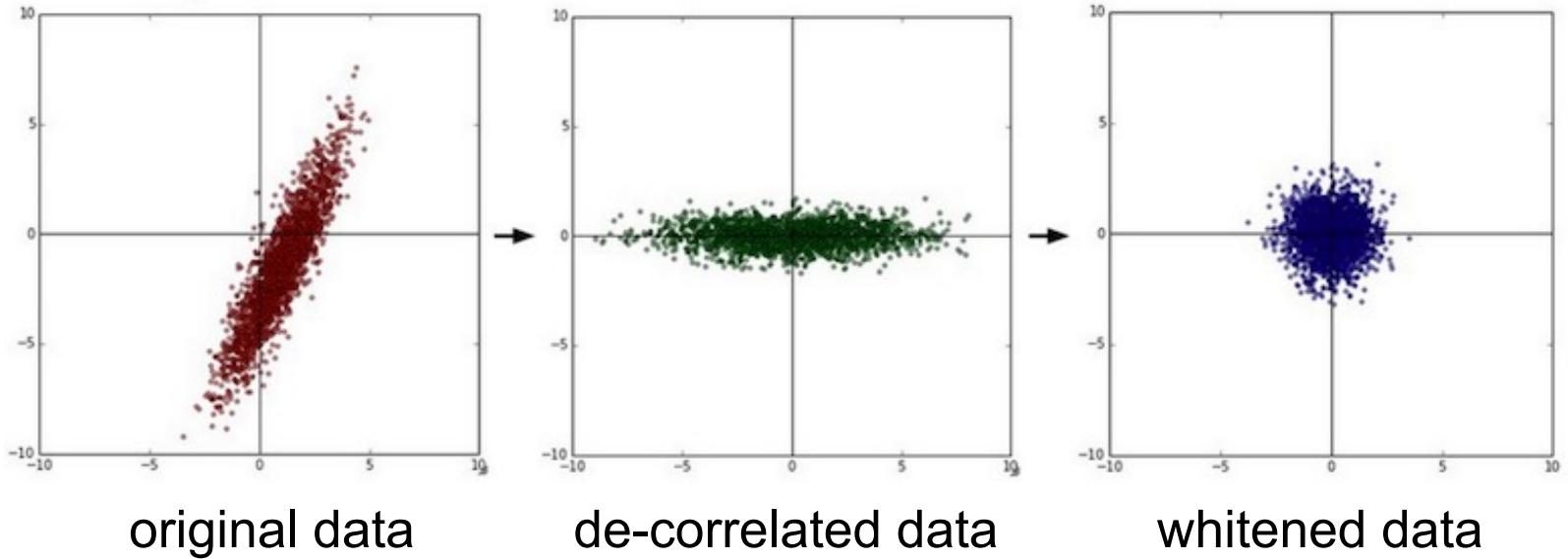
Mean-subtraction

- Zero-centered by subtracting the mean in each dimension

Normalization

- Each dimension is additionally scaled by its standard deviation
- The red lines indicate the extent of the data

Data Preprocessing



PCA decorrelates the data

- The covariance matrix is diagonal (rotated into the eigenbasis)

Each dimension is additionally scaled by eigenvalues

- Transform the data covariance matrix into the identity matrix

Don't PCA/whiten the data with ConvNet

Weight Initialization and Regularization

Do not set all the initial weights to zero

- May lead **exact** same parameter updates

One heuristic recommendation

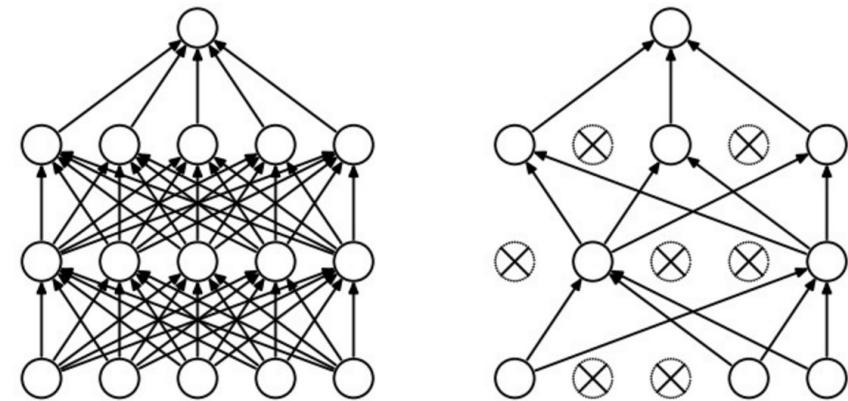
- uniform (0,1) / \sqrt{n} where n is the number of input
- Can initialize all biases to zero (or small random numbers)

Dropout

- Training: Only update the parameters of the sampled nodes
- Test: No dropout applied

In practice

- Use dropout to all layers with 0.5 rate
- L2 norm can be used



Loss Functions

An average over the data losses for individual examples

$$L = \sum_i L_i / N$$

Classification

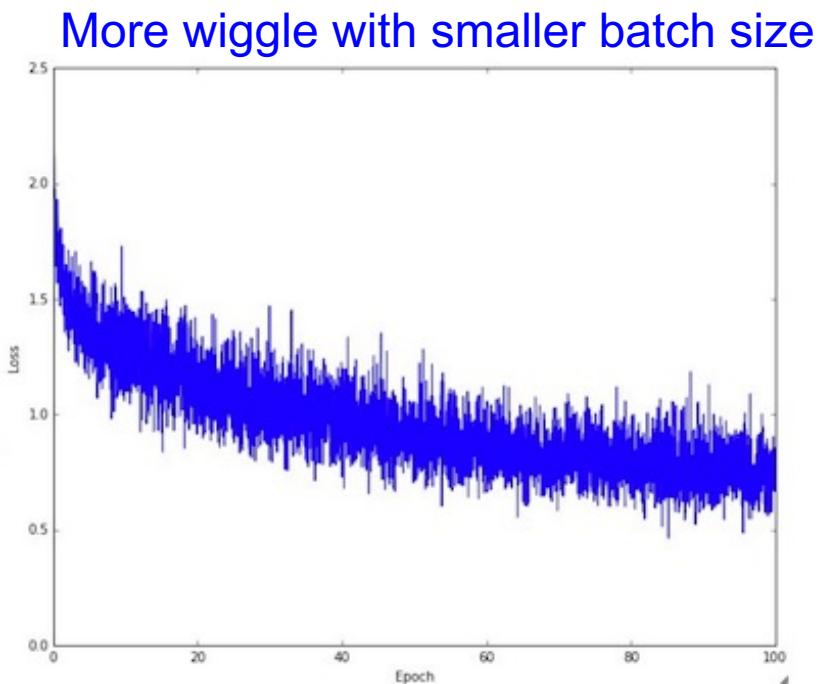
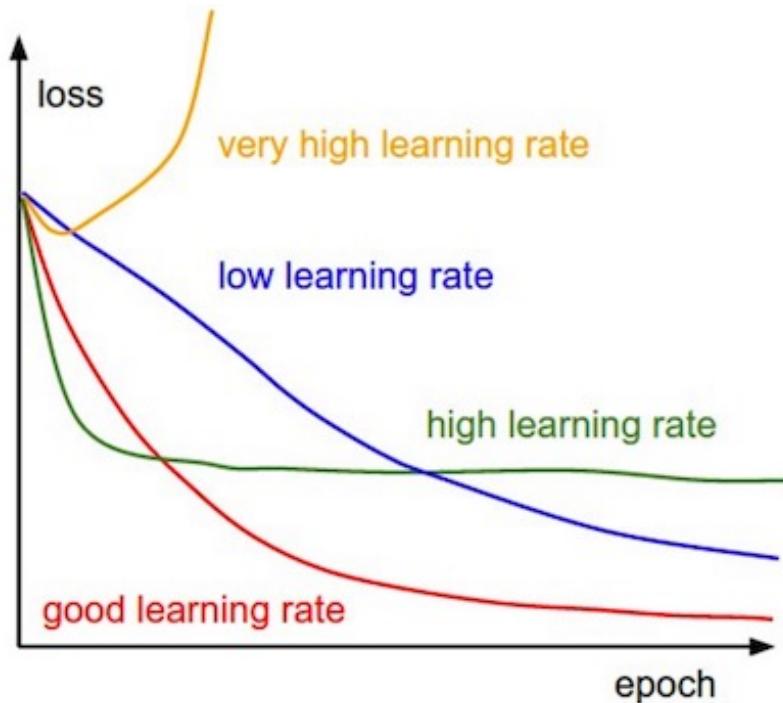
- Hinge loss $L_i = \sum_{j \neq y_i} \max(0, f_i - f_{y_i} + 1)$ (f : activation of output layer)
- Cross-entropy loss $L_i = -\log(\frac{\exp(f_{y_i})}{\sum_j \exp(f_j)})$
- Hierarchical loss when the set of labels is very large

Regression

- L2 (L1) loss $L_i = \|f - y_i\|_2^2$ ($L_i = \|f - y_i\|_1$)
- The L2 loss is much harder to optimize than a more stable loss such as Softmax

Monitoring Learning Process

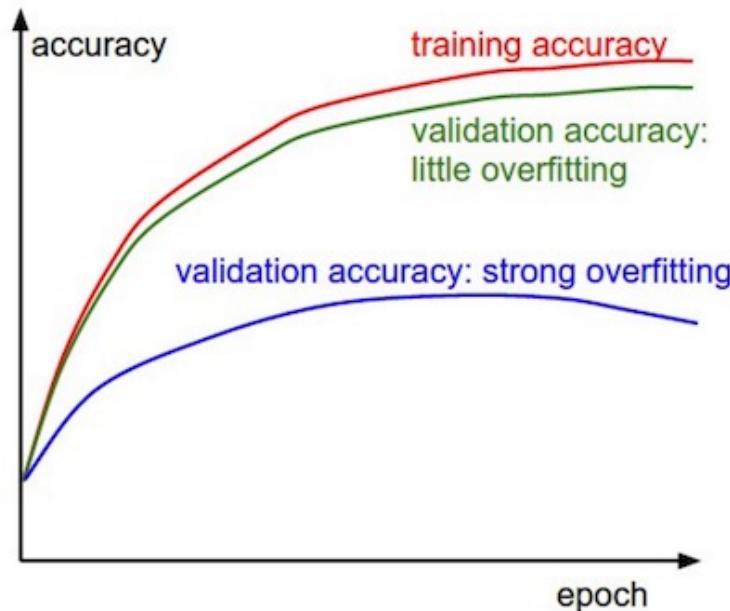
1. Loss function



- Low learning rate: slow convergence
- High learning rate: decay the loss faster, but they get stuck at worse values of loss

Monitoring Learning Process

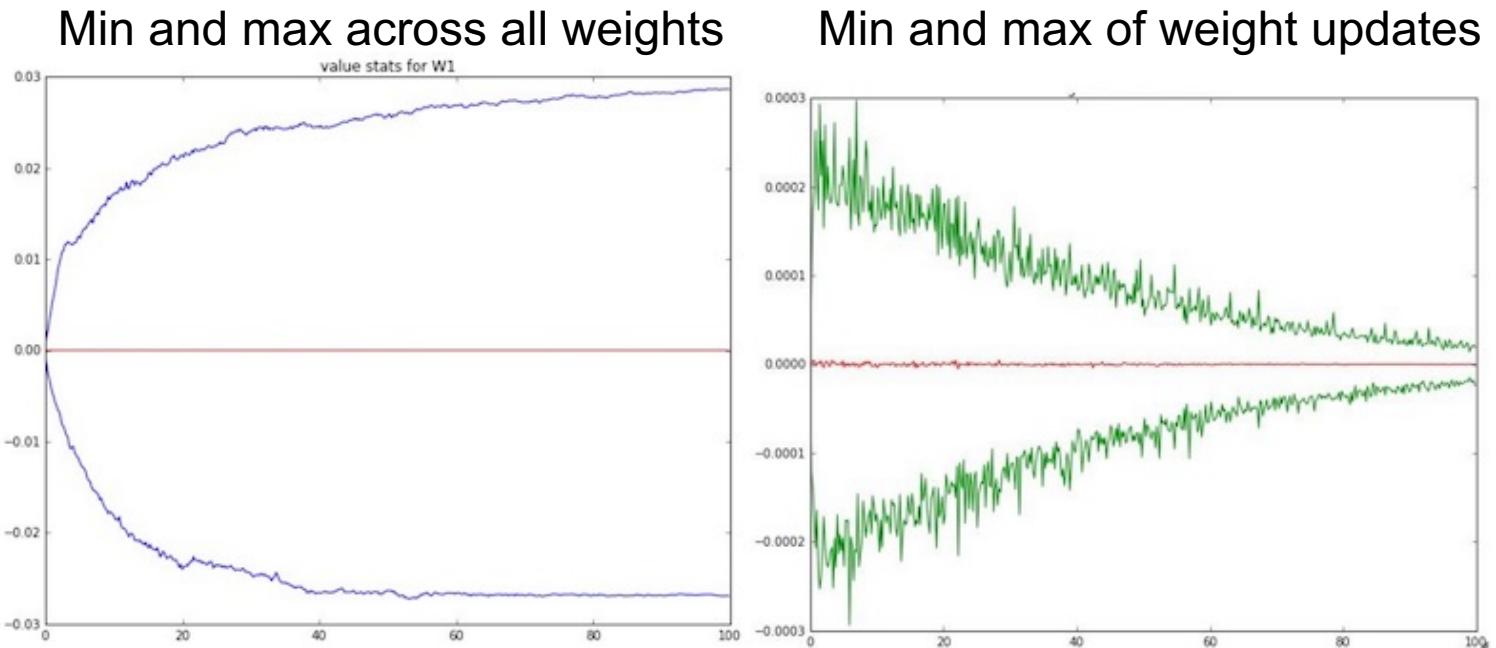
2. Train/Val accuracy



- The gap between the training and validation accuracy indicates the amount of overfitting
- When overfitting occurs, increase regularization (stronger L2 weight penalty, more dropout, etc.) or collect more data

Monitoring Learning Process

3. Ratio of updates / weights



- A rough heuristic is that the ratio is around $1\text{e-}3$
- If the ratio is lower than that, increase the learning rate
- Ex. $0.0002 / 0.02 \sim 1\text{e-}2$: within a relatively healthy limit

Monitoring Learning Process

4. Activation / Gradient distributions per layer

- Variance of the activations on each layer

Bad

```
Layer 0: Variance: 1.005315e+00
Layer 1: Variance: 3.123429e-04
Layer 2: Variance: 1.159213e-06
Layer 3: Variance: 5.467721e-10
Layer 4: Variance: 2.757210e-13
Layer 5: Variance: 3.316570e-16
Layer 6: Variance: 3.123025e-19
Layer 7: Variance: 6.199031e-22
Layer 8: Variance: 6.623673e-25
```

Good

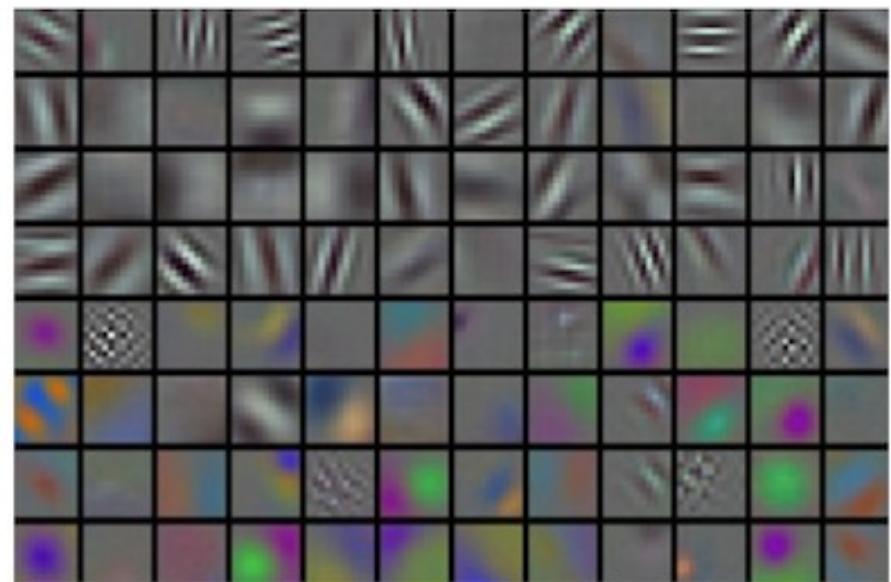
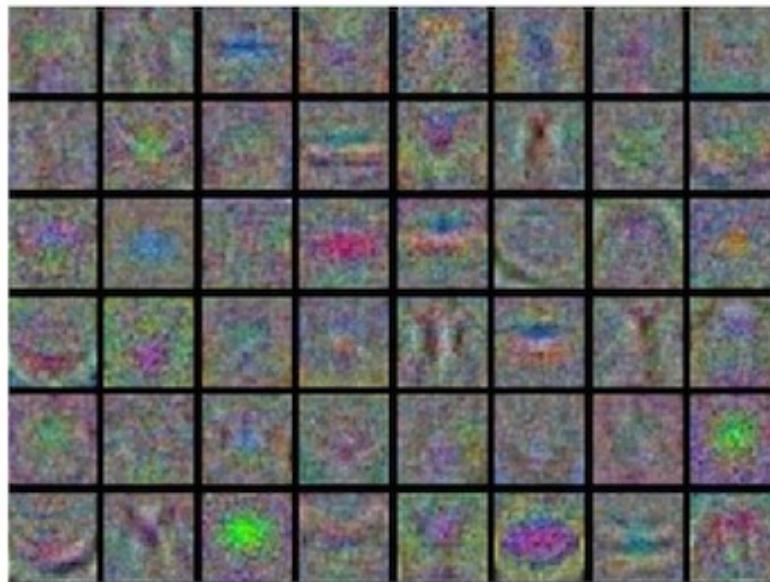
```
Layer 0: Variance: 1.002860e+00
Layer 1: Variance: 7.015103e-01
Layer 2: Variance: 6.048625e-01
Layer 3: Variance: 8.517882e-01
Layer 4: Variance: 6.362898e-01
Layer 5: Variance: 4.329555e-01
Layer 6: Variance: 3.539950e-01
Layer 7: Variance: 3.809120e-01
Layer 8: Variance: 2.497737e-01
```

- (Left) Activations vanish extremely quickly in the higher layers
- (Right) Variance through the network is preserved

Monitoring Learning Process

5. Visualization of each layer's feature map

- Two extreme cases of feature maps for the first layer



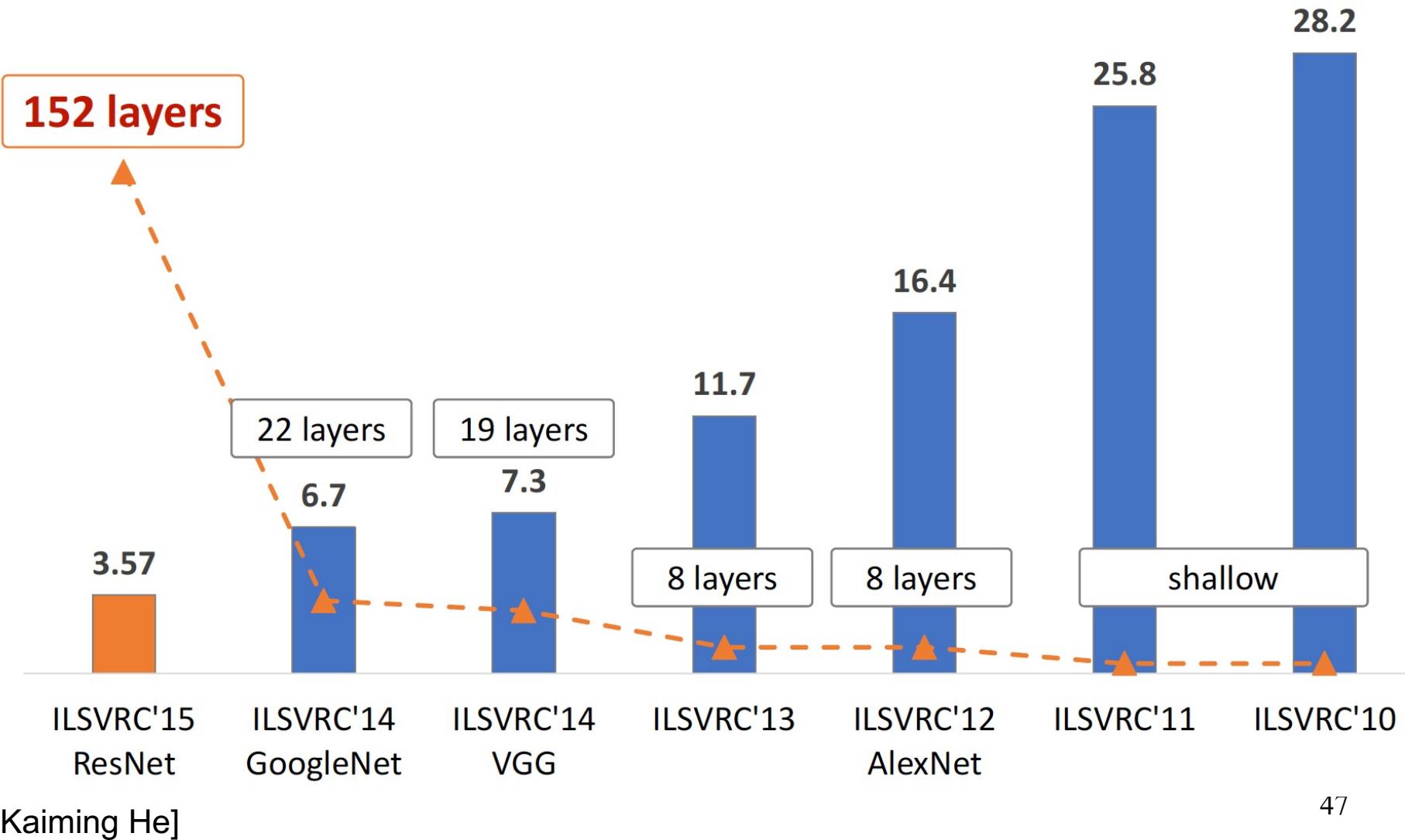
- (Left) Problematic! (e.g. Unconverged network, improperly set learning rate, very low weight regularization penalty)
- (Right) Good indication that the training is proceeding well

Outline

- Convolutional Neural network
- Training Tips
- Case Studies

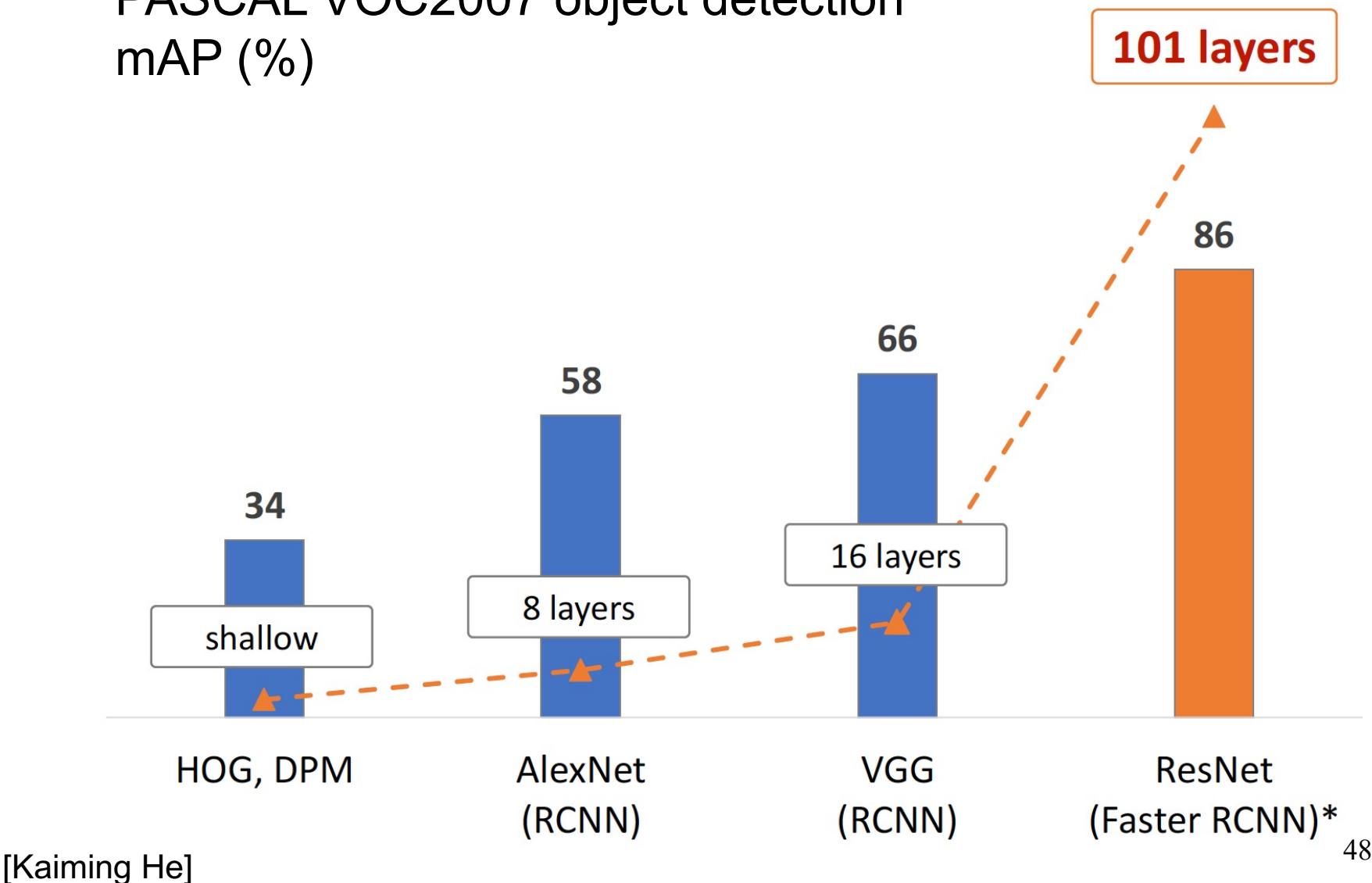
Revolution of Depth

ImageNet classification top-5 error (%)



Revolution of Depth

PASCAL VOC2007 object detection mAP (%)

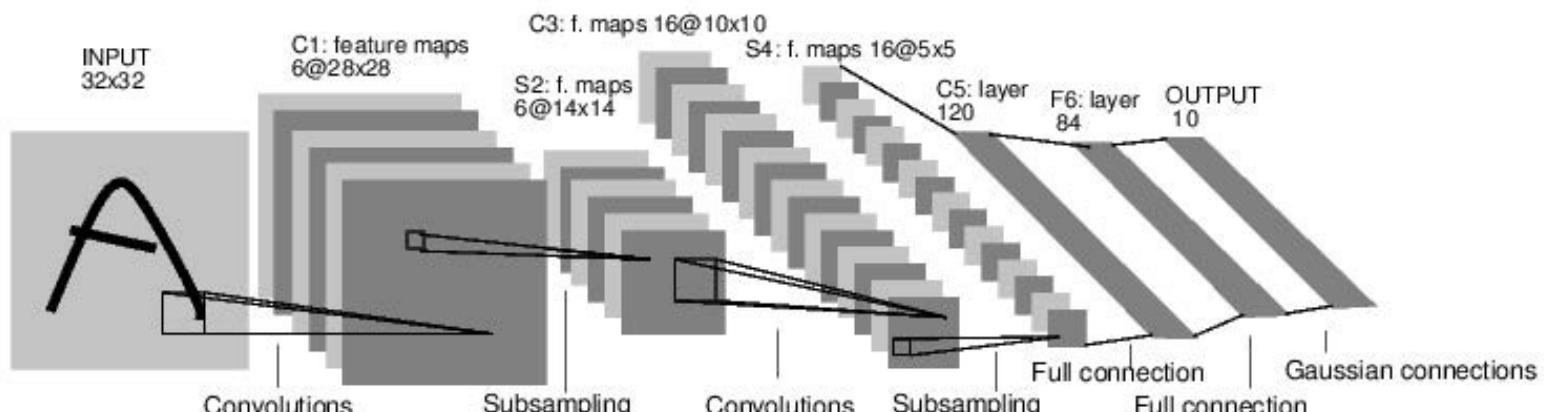


Case Study – LeNet

First successful CNN for recognition of hand-written digits

Propose basic components of modern ConvNets

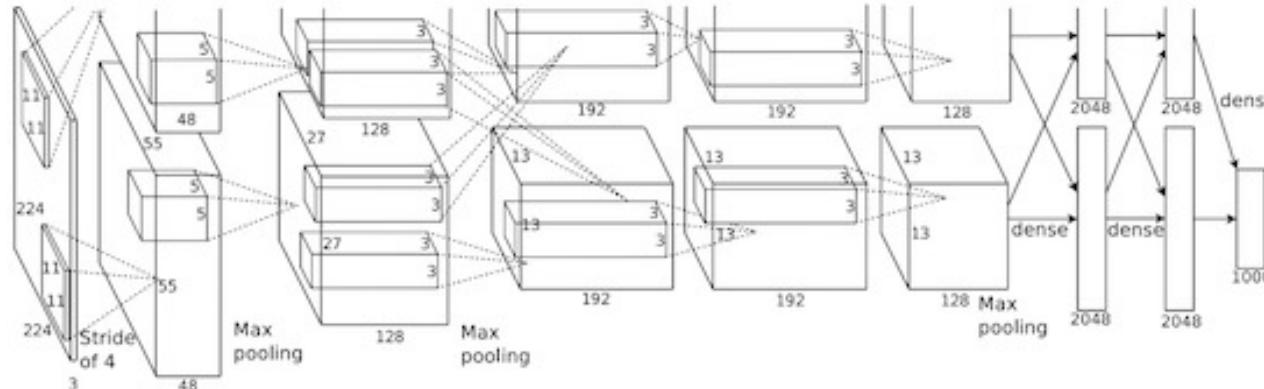
- Convolution: locally-connected, spatially weight-sharing (5x5 at stride 1)
- Subsampling (pooling) (2x2 at stride 2)
- [CONV-POOL-CONV-POOL-FC-FC]
- Training by Backpropagation



Case Study – AlexNet

The first work revolutionized CNN in computer Vision

- Similar to LeNet, but deeper, bigger, and featured layers of CONV, RELU, POOL
- In ImageNet ILSVRC challenge 2012, significantly outperform the runner-up (top 5 error of 16% < 26%)
- ReLU: accelerate training and better grad prop (vs. tanh)
- Dropout: In-network ensembling for reducing overfitting
- Heavy data augmentation: Label-preserving transformation



Case Study – AlexNet

Architecture details

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

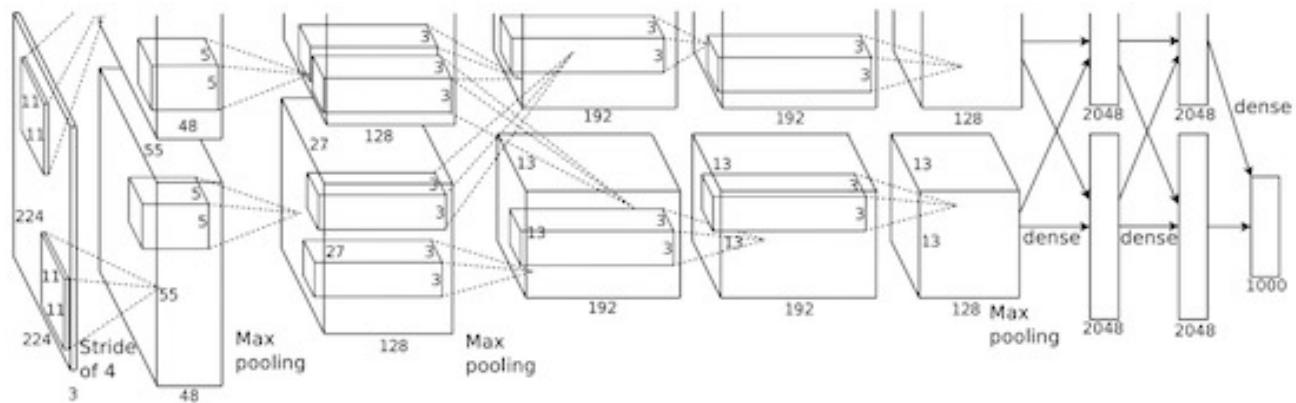
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons
(class scores)

Other details

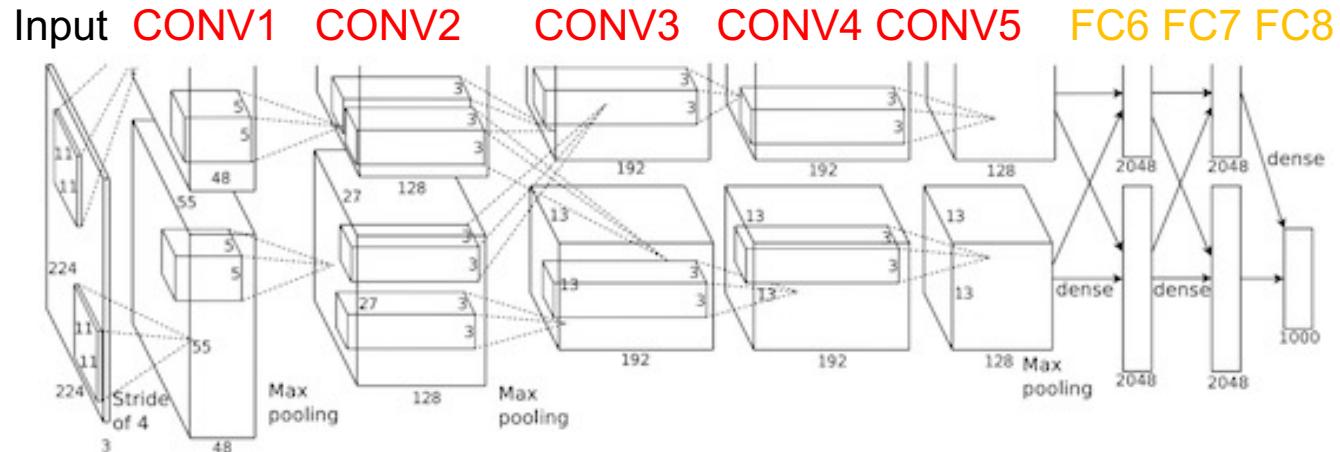
- Used Norm layers (not common anymore)
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val acc. plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%



Case Study – AlexNet

Implementation details

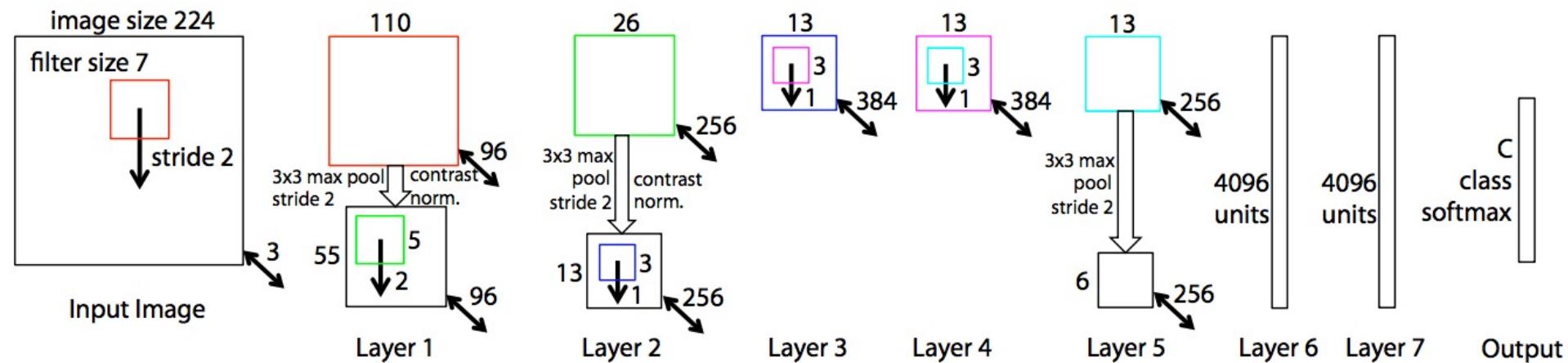
- Trained on GTX 580 GPU with only 3 GB of memory
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU
- CONV1/2/4/5: Connections only with feature maps on same GPU
- CONV3, FC6/7/8: Connections with all feature maps in preceding layer, communication across GPUs



Case Study – ZFNet

Engineered AlexNet

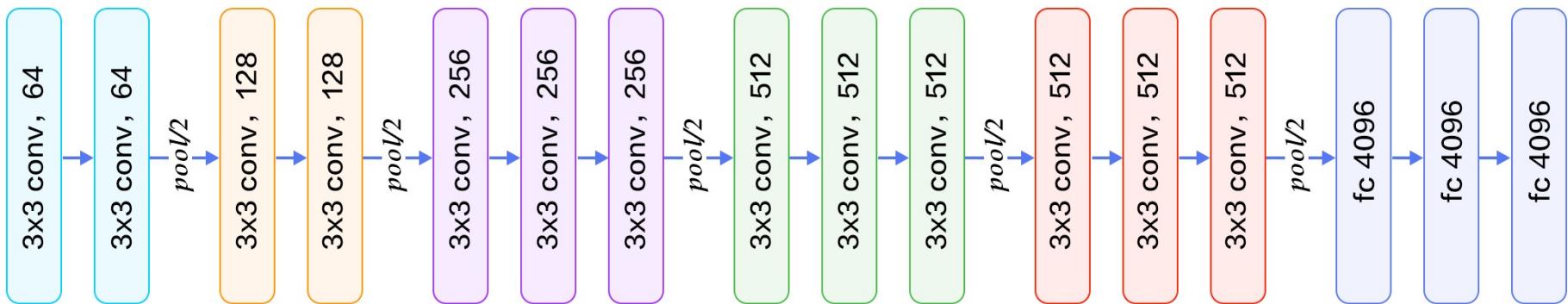
- The winner of ILSVRC 2013 (ImageNet Top-5: 16.4% → 11.7%)
- e.g. Compared to AlexNet,
CONV1: (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: (384, 384, 256) filters to (512, 1024, 512)
- Not popularly used these days



Case Study – VGGNet

Deeper versions of AlexNet (8 layers)

- The runner-up of ILSVRC 2014 (ImageNet Top-5: 11.7% → 7.3%)
- Good performance but not so efficient
- Modularized design:
 - Stack the same module (3×3 Conv stride 1, pad 1)
 - Same computation per module (1/2 spatial size → 2x filters)
- Stage-wise training: VGG-11 → VGG-13 → VGG-16



K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. ICLR 2015

Case Study – VGGNet

Most memory is in early CONV

INPUT: [224x224x3]

CONV3-64: [224x224x64]

CONV3-64: [224x224x64]

POOL2: [112x112x64]

CONV3-128: [112x112x128]

CONV3-128: [112x112x128]

POOL2: [56x56x128]

CONV3-256: [56x56x256]

CONV3-256: [56x56x256]

CONV3-256: [56x56x256]

POOL2: [28x28x256]

CONV3-512: [28x28x512]

CONV3-512: [28x28x512]

CONV3-512: [28x28x512]

POOL2: [14x14x512]

CONV3-512: [14x14x512]

CONV3-512: [14x14x512]

CONV3-512: [14x14x512]

POOL2: [7x7x512]

FC: [1x1x4096]

FC: [1x1x4096]

FC: [1x1x1000]

memory: $224 \times 224 \times 3 = 150\text{K}$

memory: $224 \times 224 \times 64 = 3.2\text{M}$

memory: $224 \times 224 \times 64 = 3.2\text{M}$

memory: $112 \times 112 \times 64 = 800\text{K}$

memory: $112 \times 112 \times 128 = 1.6\text{M}$

memory: $112 \times 112 \times 128 = 1.6\text{M}$

memory: $56 \times 56 \times 128 = 400\text{K}$

memory: $56 \times 56 \times 256 = 800\text{K}$

memory: $56 \times 56 \times 256 = 800\text{K}$

memory: $56 \times 56 \times 256 = 800\text{K}$

memory: $28 \times 28 \times 256 = 200\text{K}$

memory: $28 \times 28 \times 512 = 400\text{K}$

memory: $28 \times 28 \times 512 = 400\text{K}$

memory: $28 \times 28 \times 512 = 400\text{K}$

memory: $14 \times 14 \times 512 = 100\text{K}$

memory: $7 \times 7 \times 512 = 25\text{K}$

memory: 4096

memory: 4096

memory: 1000

params: 0

params: $(3 \times 3 \times 3) \times 64 = 1,728$

params: $(3 \times 3 \times 64) \times 64 = 36,864$

params: 0

params: $(3 \times 3 \times 64) \times 128 = 73,728$

params: $(3 \times 3 \times 128) \times 128 = 147,456$

params: 0

params: $(3 \times 3 \times 128) \times 256 = 294,912$

params: $(3 \times 3 \times 256) \times 256 = 589,824$

params: $(3 \times 3 \times 256) \times 256 = 589,824$

params: 0

params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

params: 0

params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

params: 0

params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

params: $4096 \times 4096 = 16,777,216$

params: $4096 \times 1000 = 4,096,000$

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}/\text{image}$

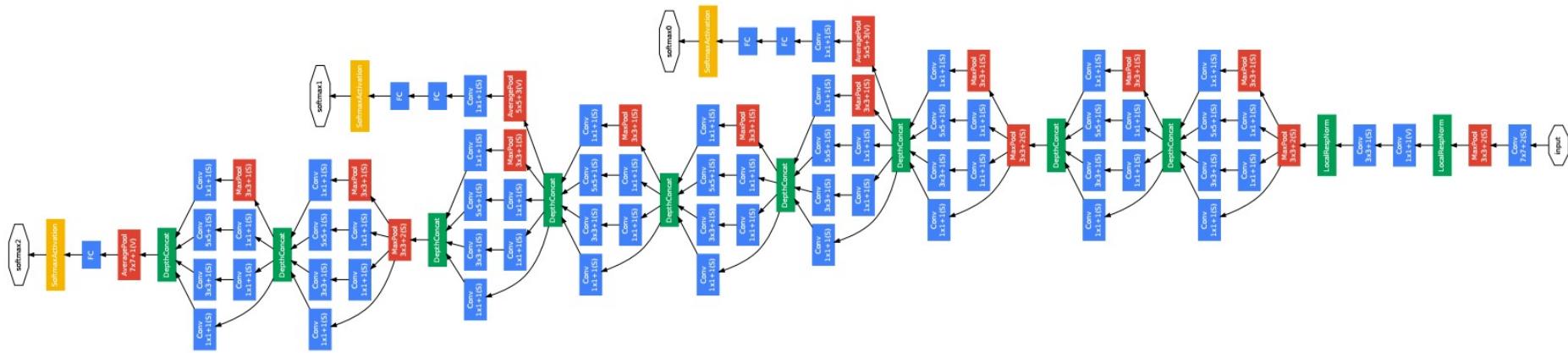
TOTAL params: 138M parameters

Most params
are in late FC

Case Study – GoogLeNet

Inception (network in network) structure

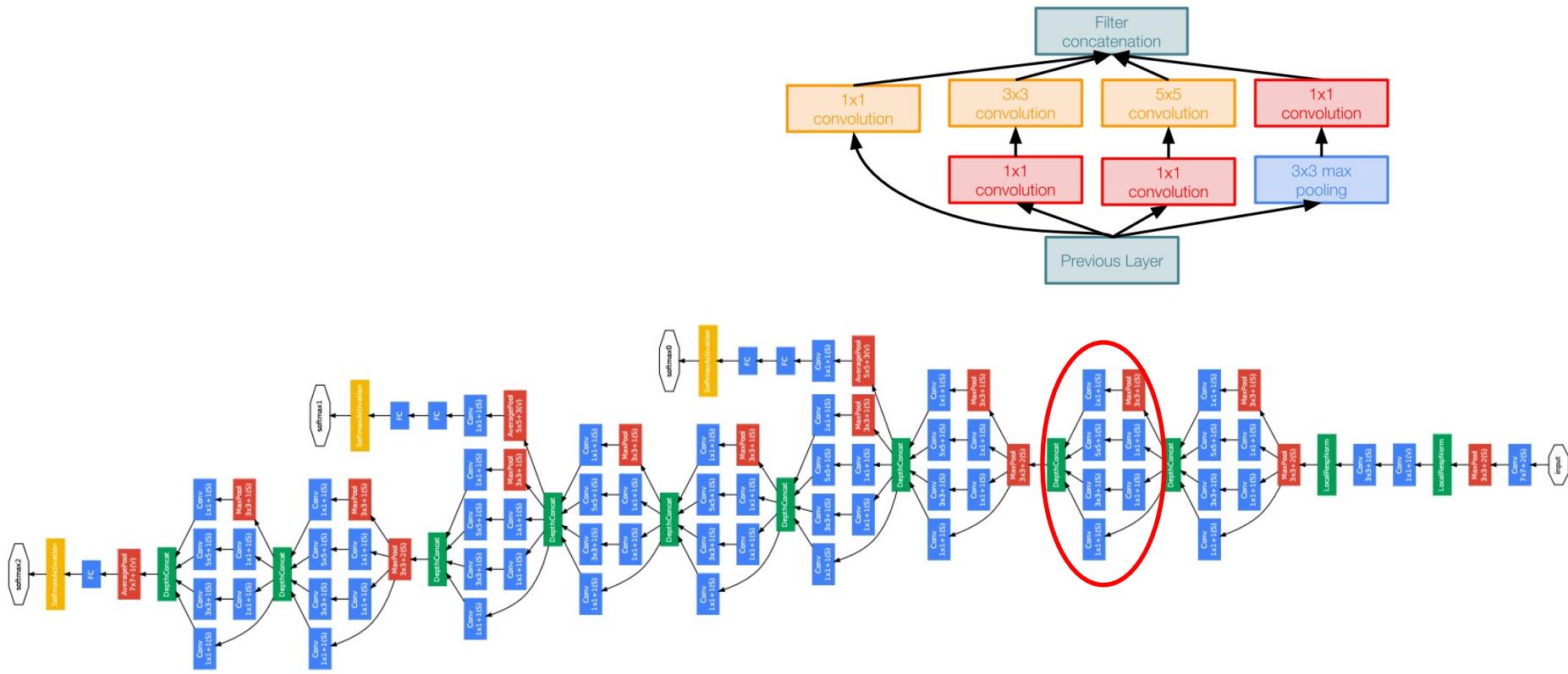
- The winner of ILSVRC 2014 (ImageNet Top-5: 7.3% → 6.7%)
 - Much fewer parameters of 5M compared to 60M of AlexNet
 - No FC layers and 22 layers in total (v1)
 - Keep updating... to Inception-v4 (Inception-ResNet)



Case Study – GoogLeNet

Inception module

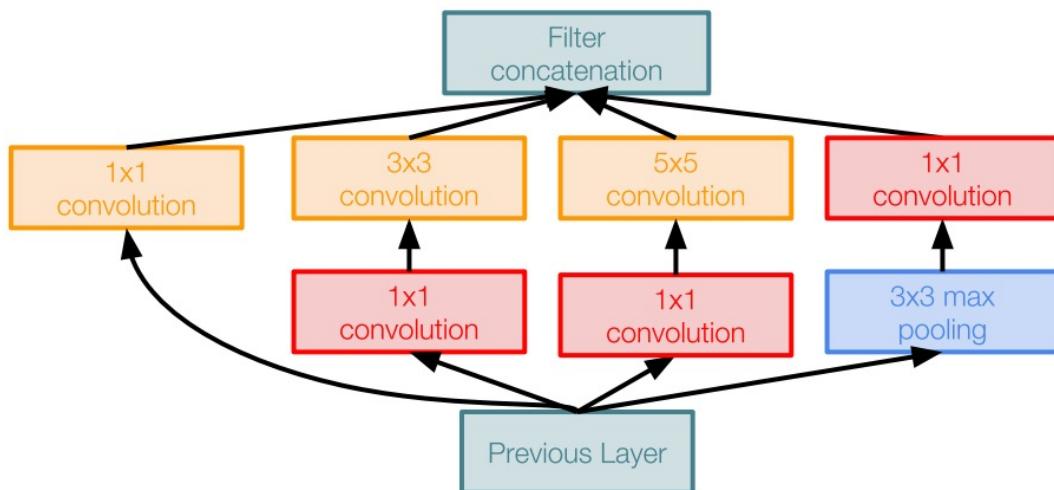
- Design a good local network topology (network within a network) and then stack these modules on top of each other



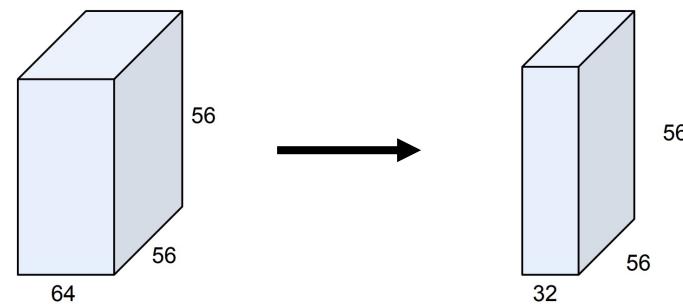
Case Study – GoogLeNet

Key ideas of Inception module

- Multiple branches: e.g., 1x1, 3x3, 5x5, pool
- Shortcuts: stand-alone 1x1
- Bottleneck: reduce dim by 1x1 before expensive 3x3/5x5 conv
- Concatenate all filter outputs together depth-wise



cf) 1x1 CONV with 32 filters

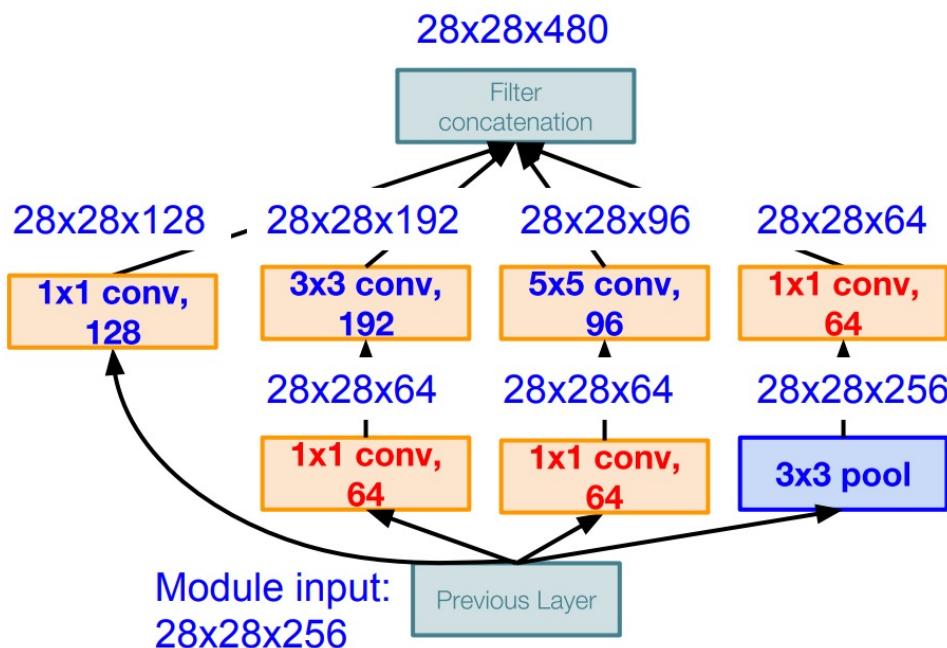


Each filter has size $1 \times 1 \times 64$,
and performs a 64-dim dot
product

Case Study – GoogLeNet

Inception module with dimension reduction

- Compared to 854M ops for naive version Bottleneck can reduce depth after pooling layer



Conv Ops:
[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1×1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3×3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
[5×5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
Total: 358M ops

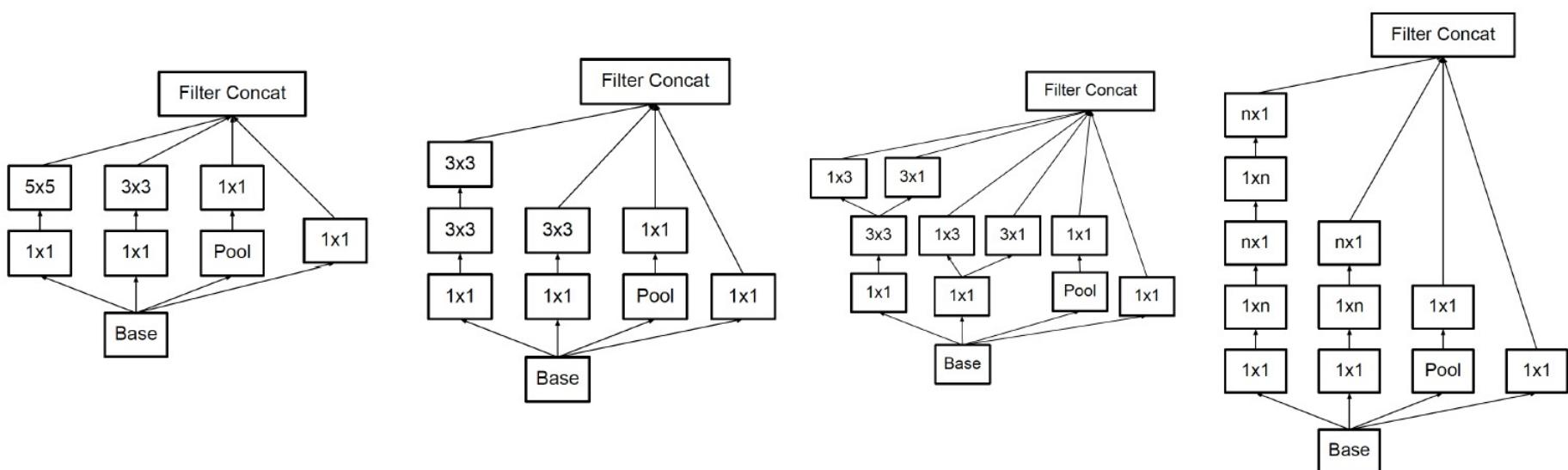
Case Study – GoogLeNet / Inception v1-v3

More templates, but the same 3 properties are kept

- Multiple branches, shortcuts, bottleneck

Optimizing multi-branch CNNs largely benefits from BN

- Including all Inceptions and ResNets

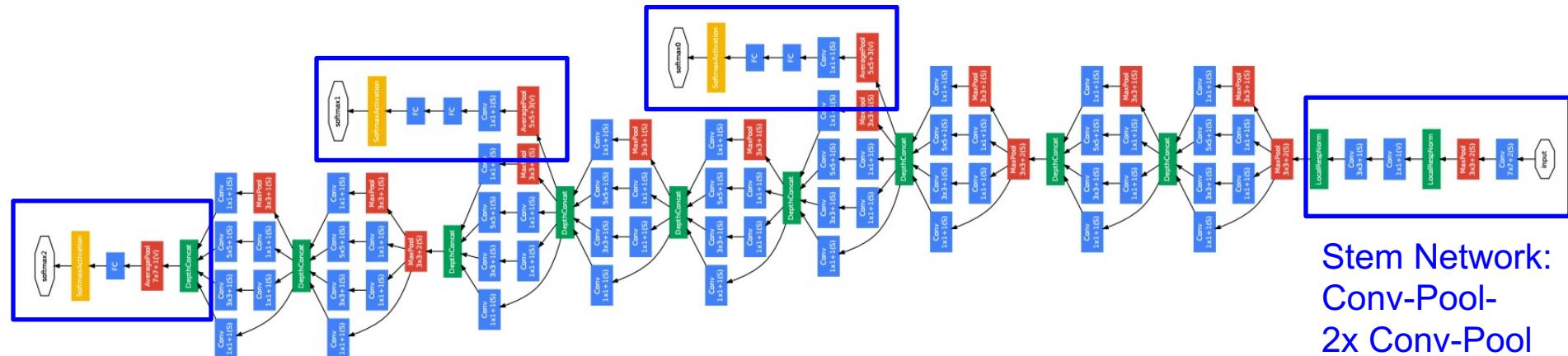


Case Study – GoogLeNet

Full GoogLeNet architecture

- Stack Inception modules with dim reduction on top of each other
- 22 total layers with weights (parallel layers count as 1 → 2 layers per Inception module. Don't count auxiliary output layers)

Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)



Classifier output
(removed expensive
FC layers!)

Stacked inception modules

ResNet – Degradation Problem

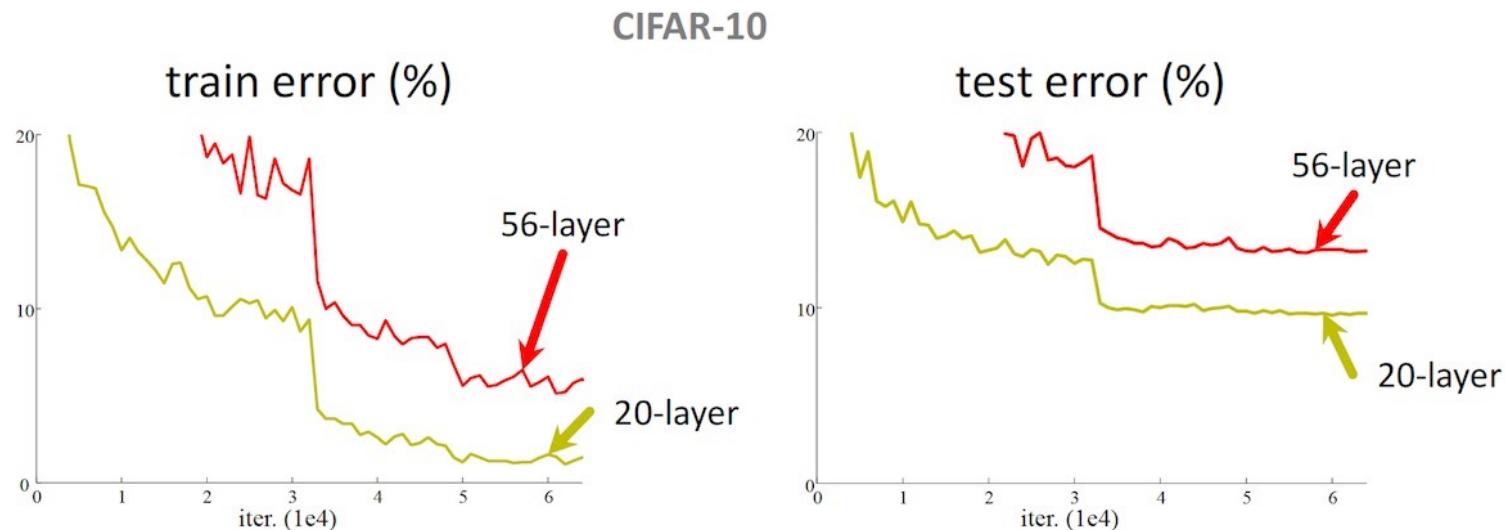
Learn better networks by simply stacking more layers?

Plain nets: stacking 3x3 conv layers...

- With the network depth increasing, accuracy gets saturated and then degraded rapidly

NOT caused by **overfitting**

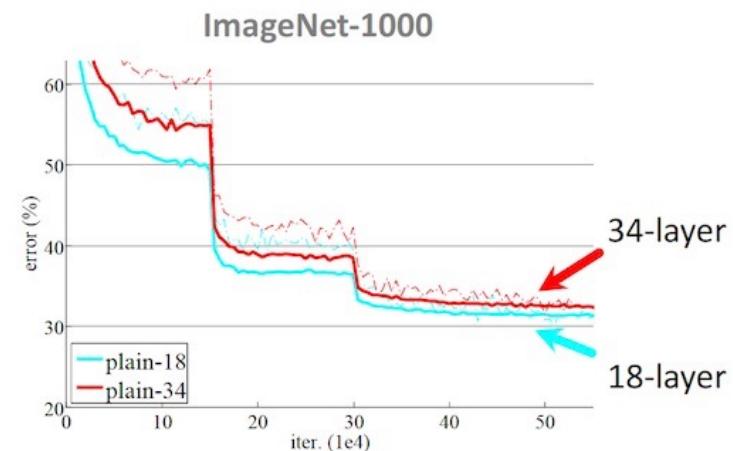
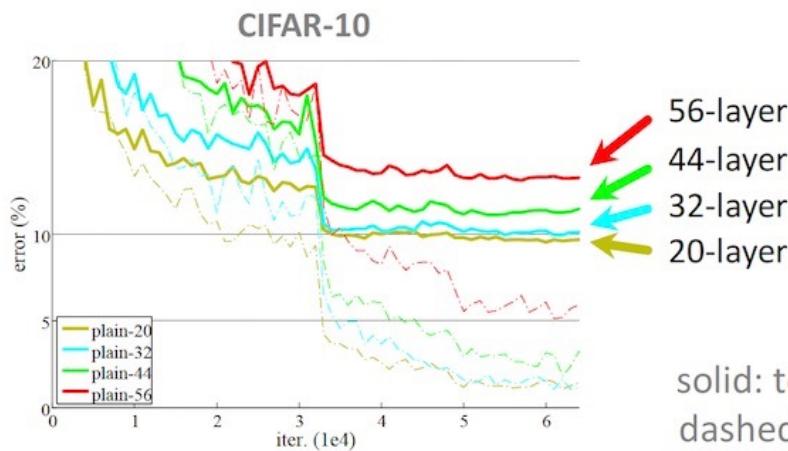
- 56-layer net has higher *training and test error* than 20-layer net



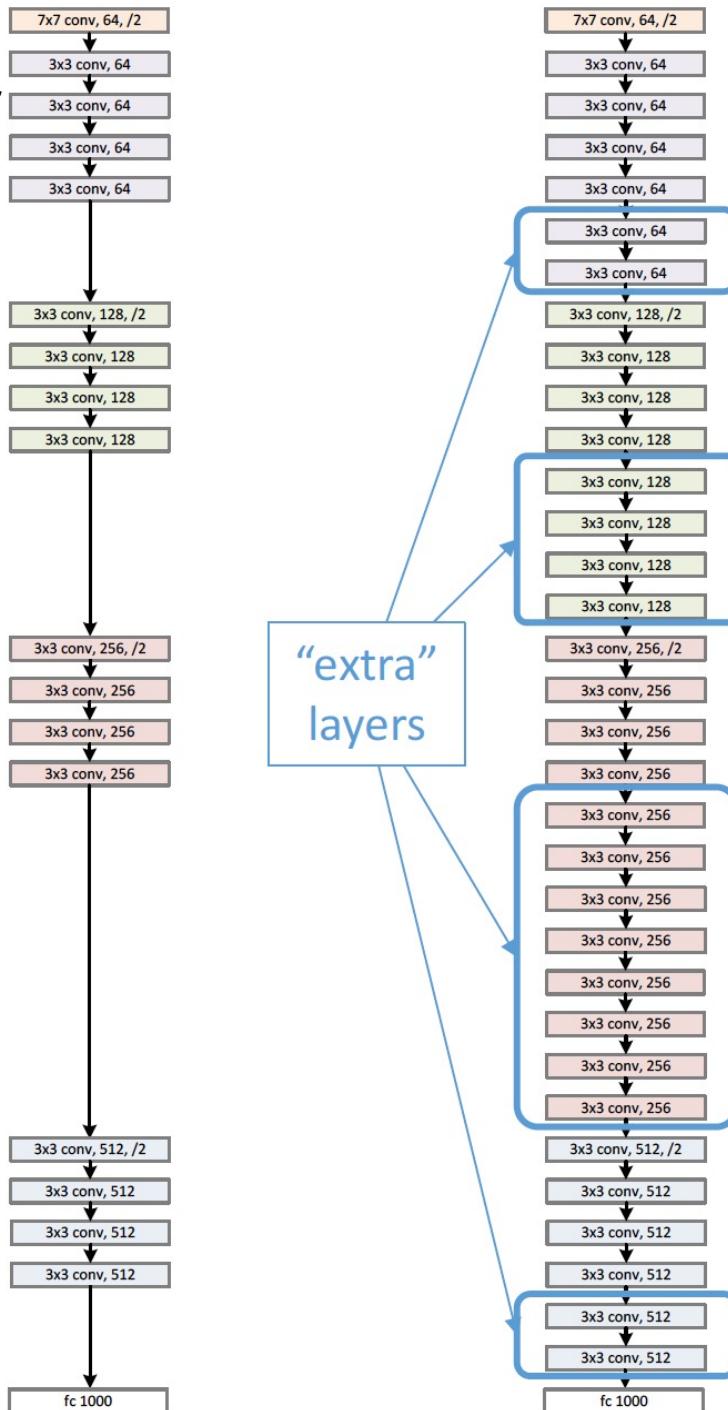
ResNet – Degradation Problem

Overly deep plain nets have **higher training error**

A general phenomenon, observed in many datasets



A shallower
model
(18 layers)



A deeper
counterpart
(34 layers)

A richer solution space

A deeper model should not have higher **training error**

A solution by construction

- Original layers: copied from a learned shallower model
- Extra layers: set as **identity**
- At least same training error

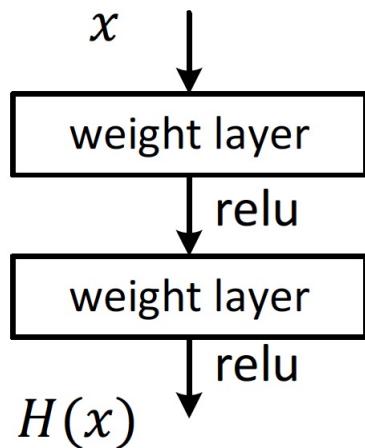
(Hypothesis)

Optimization difficulties:
solvers cannot find the
solution when going
deeper...

Deep Residual Learning

Feedforward neural networks with *shortcut connections*
(i.e. identity mapping)

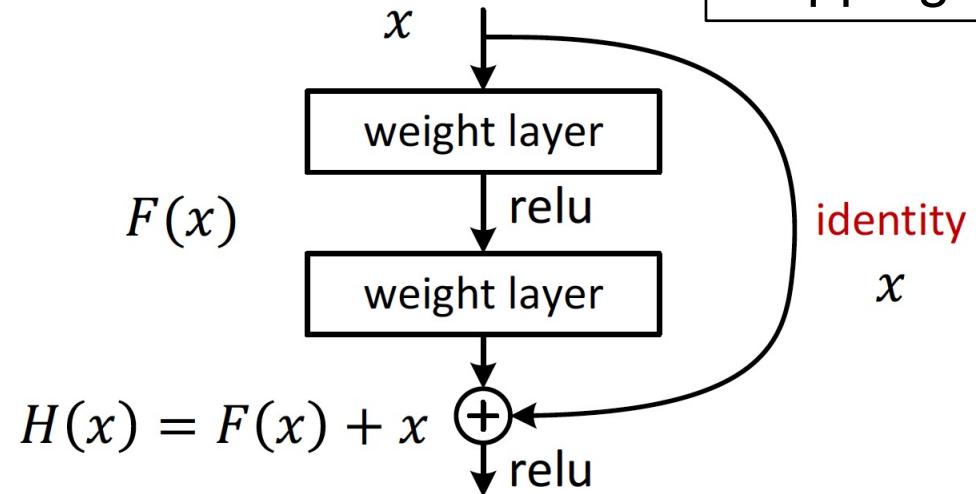
Plain net



Any small
subnet

Let $H(x)$ is
a desired
mapping

Residual net



Hope the small net fit $H(x)$

Hope the small net fit $F(x)$

Then $H(x) \cong F(x) + x$

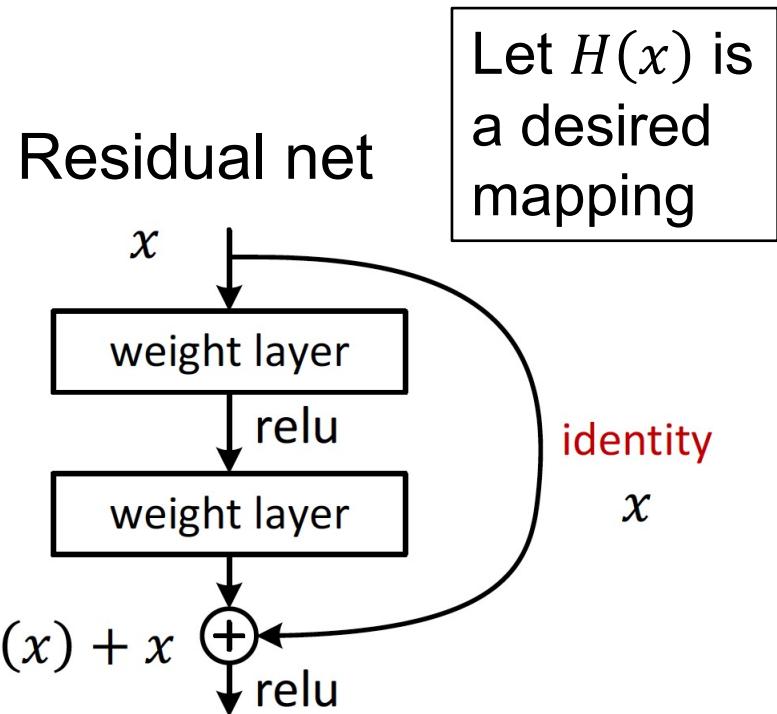
Deep Residual Learning

Feedforward neural networks with *shortcut connections* (i.e. identity mapping)

$F(x)$ is a residual mapping w.r.t. **identity**

- Focus on representing additional info from what previous layers already learned
- If identity were optimal, easy to set weights as 0 (cf. Identity mappings that plain models have difficulty in representing)

$$H(x)$$



Hope the small net fit $F(x)$

Then $H(x) \cong F(x) + x$

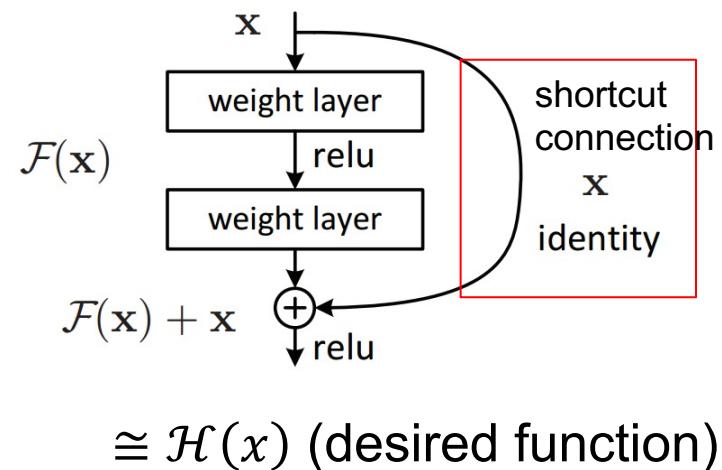
Advantage of Residual Learning

Identity shortcut connections add neither extra parameter nor computational complexity

Easy end-to-end optimization by backpropagation

Accuracy gains from greatly increased depth

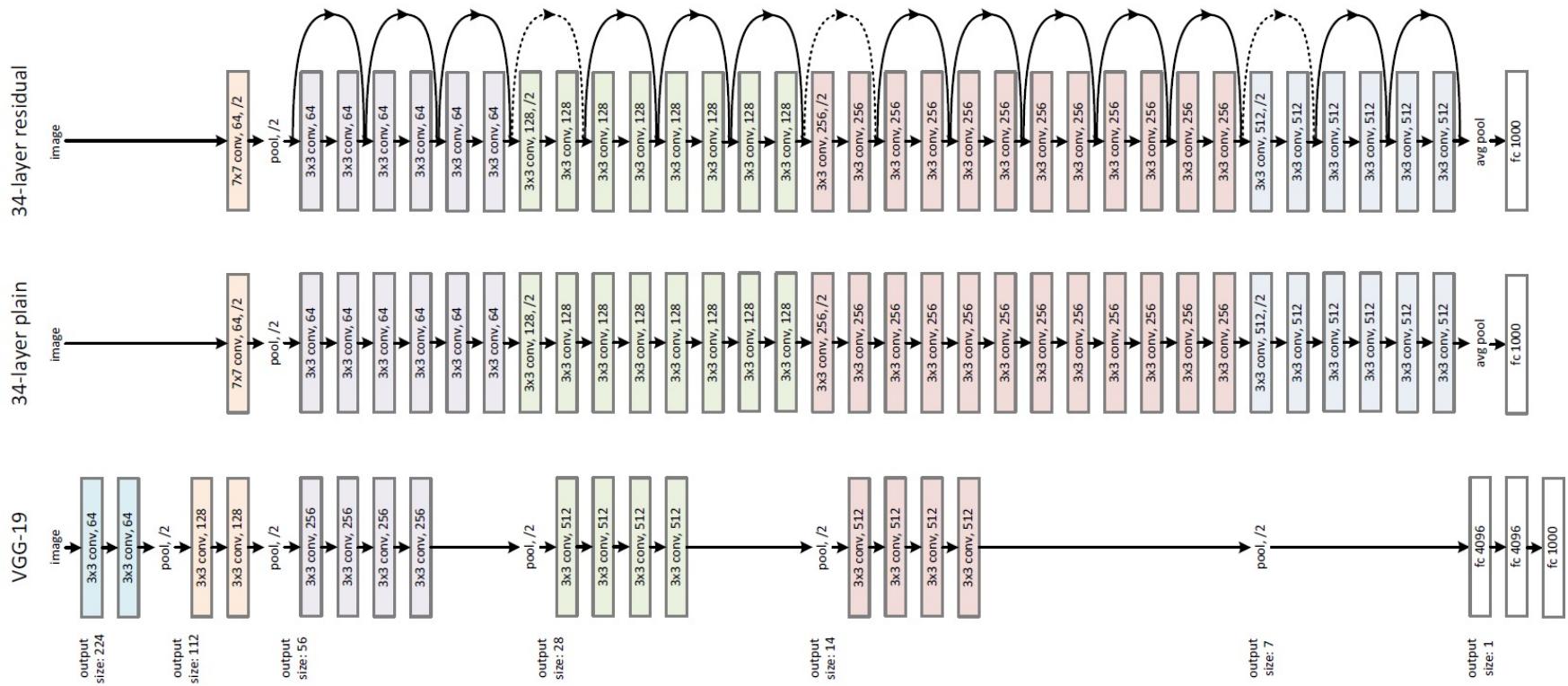
- 152-layer ResNet achieves 3.57% top-5 errors on ImageNet datasets with lower complexity than VGGNet



34-Layer ResNet

Design principles – similar to VGG-19

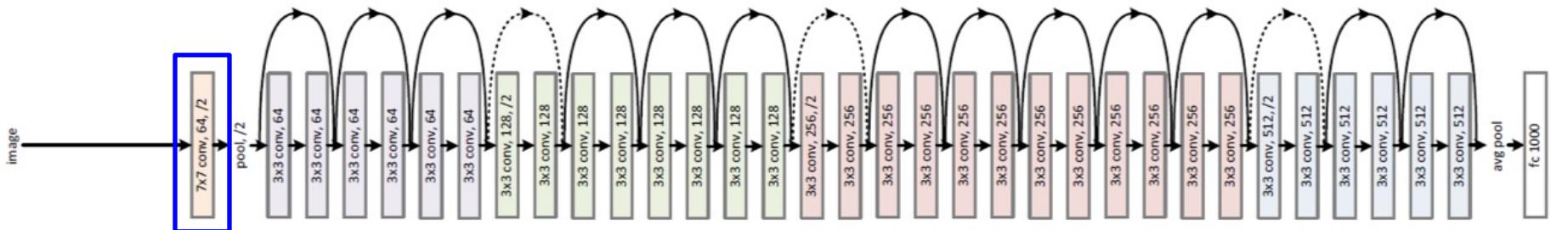
- Same feature map size, same # filters (channels)
 - Whenever feature map size is halved, # filters is doubled to preserve the time complexity per layer



34-Layer ResNet

Full ResNet architecture (stack residual blocks)

- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- 34-layer ResNet has only 19% FLOPs of VGG-19:
 - (i) 1/2 input size (224/14 -> 112/7), ii) No hidden FC layer
- Total depths of 34, 50, 101, or 152 layers for ImageNet



Additional conv layer
at the beginning

No FC layers except
fc1000 to output
classes

34-Layer ResNet

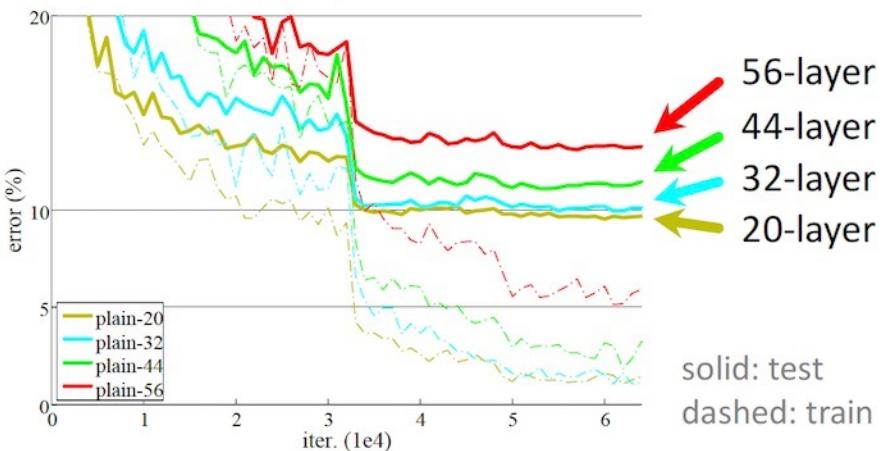
Comparing 18/34-layer residual/plain on ImageNet

- Plain-18/34: degradation problem
- ResNet-18/34: ResNet-34 better
- Plain/ResNet-18: ResNet-18 converges faster
- Deeper ResNets have **lower training error**, and also lower test error

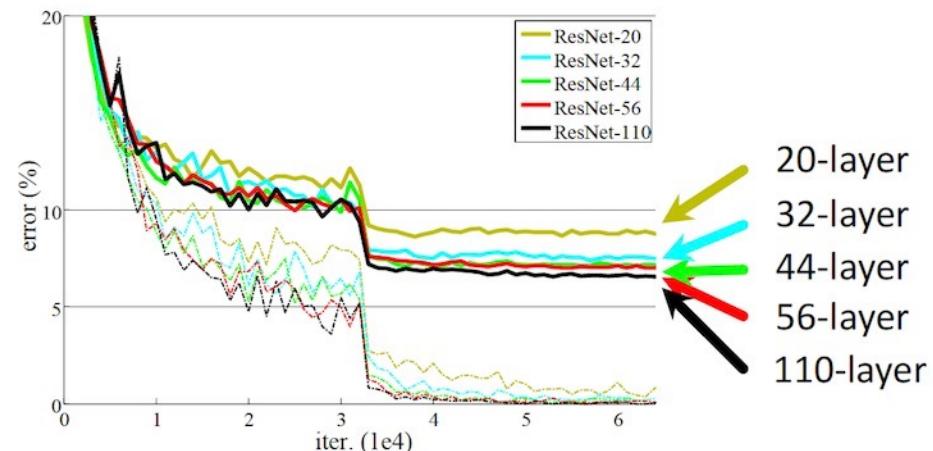
Top-1 error on ImageNet validation

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	25.03

CIFAR-10 plain nets



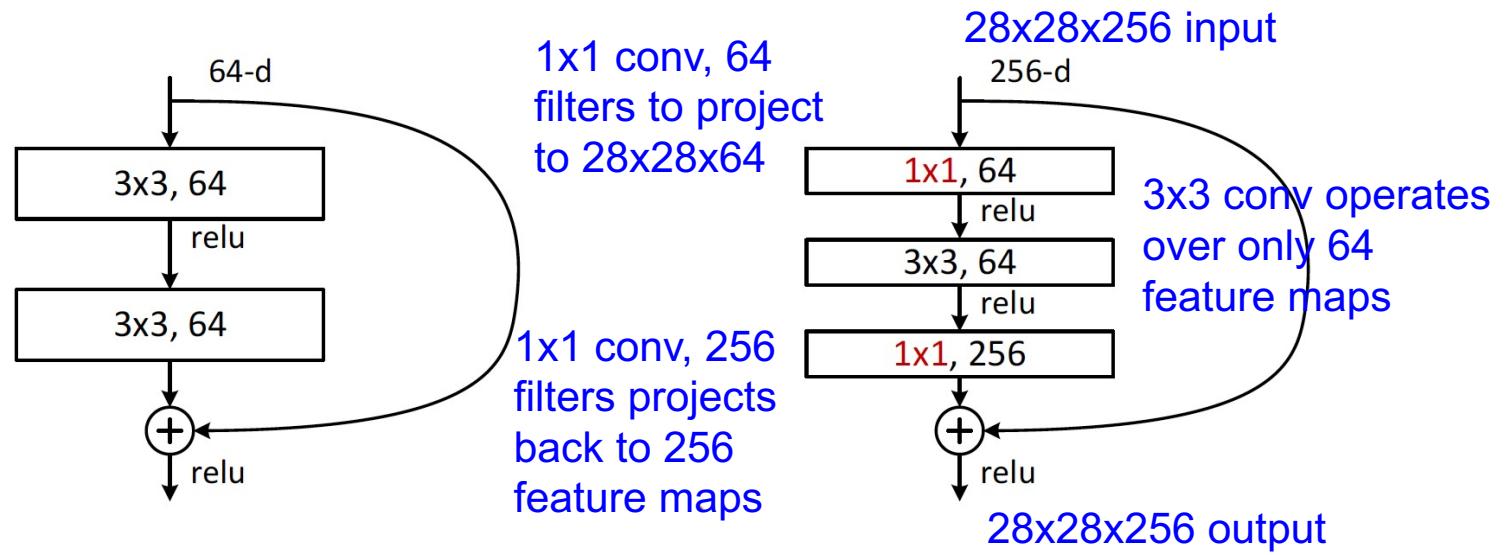
CIFAR-10 ResNets



A Practical Design of Deeper ResNet

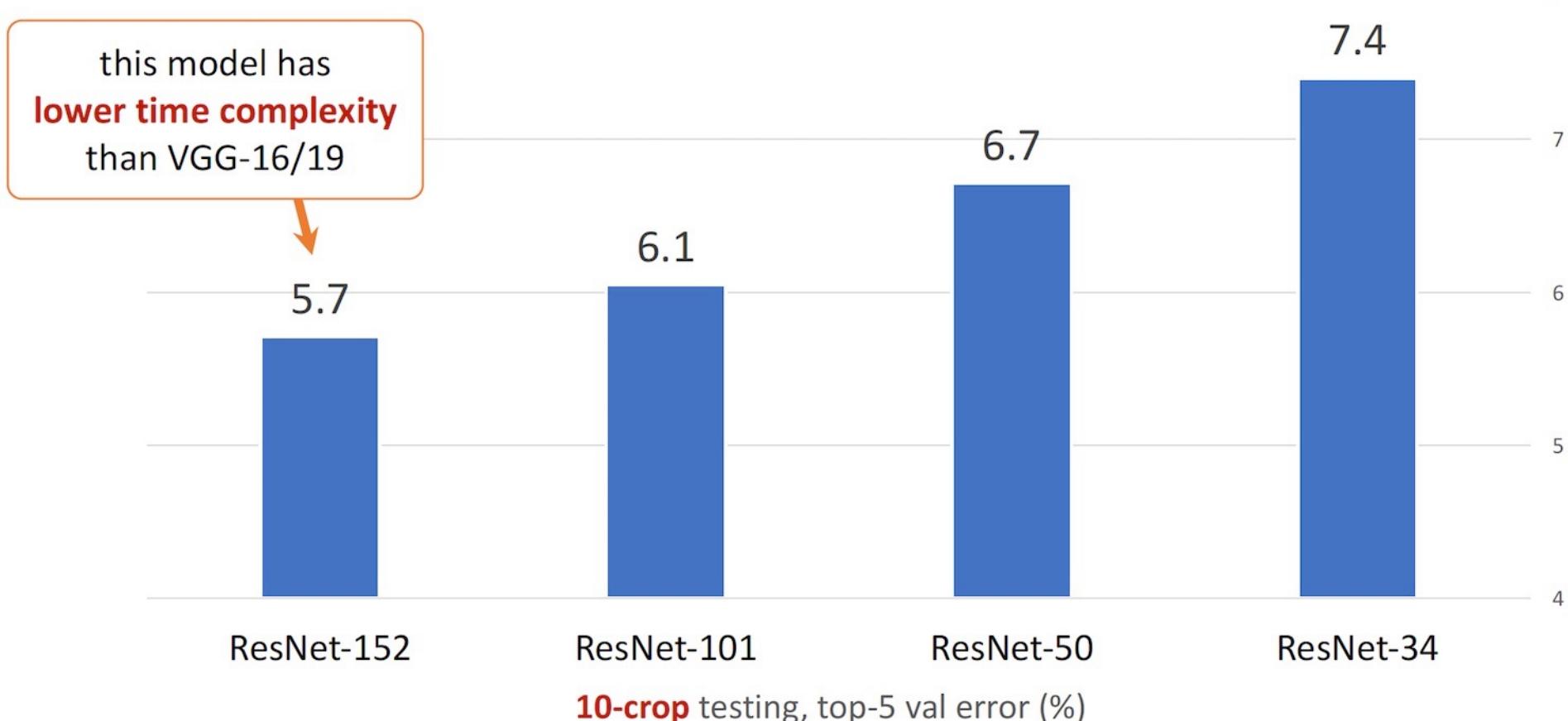
Deeper bottleneck architectures (ResNet-50+)

- Two 3x3 CONV layers → 1x1, 3x3, 1x1 layers
- Use bottleneck 1x1 layers – reducing and increasing dimensions
- Both designs have similar time complexity
- For 50/101/152-layers: 152-layer ResNet still has lower complexity than VGG-19 (15.3/19.6 BFLOPs)



ImageNet Experiments

Deeper ResNet have **lower** errors



ResNet Summary

Swept 1st place in all ILSVRC&COCO 2015 competitions

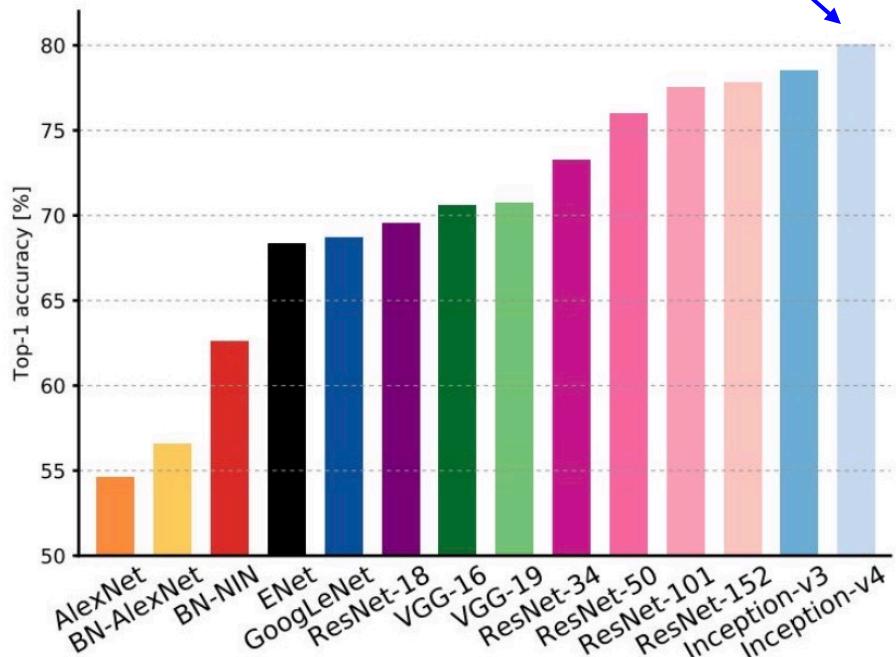
- ImageNet classification/detection/localization, COCO detection/segmentation
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on CIFAR)
- ImageNet Top-5: 6.7% → 3.5% better than human (5.1%)

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
 - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
 - ImageNet Detection: **16%** better than 2nd
 - ImageNet Localization: **27%** better than 2nd
 - COCO Detection: **11%** better than 2nd
 - COCO Segmentation: **12%** better than 2nd

Comparing Complexity

Inception-v4: Resnet + Inception



- VGG: Highest memory, most operations
- GoogLeNet: most efficient
- AlexNet: Smaller compute, still memory heavy, lower accuracy

