Deep Learning for Computer Vision

Spring 2019

http://vllab.ee.ntu.edu.tw/dlcv.html (primary)

https://ceiba.ntu.edu.tw/1072CommE5052 (grade, etc.)

FB: DLCV Spring 2019

Yu-Chiang Frank Wang 王鈺強, Associate Professor

Dept. Electrical Engineering, National Taiwan University

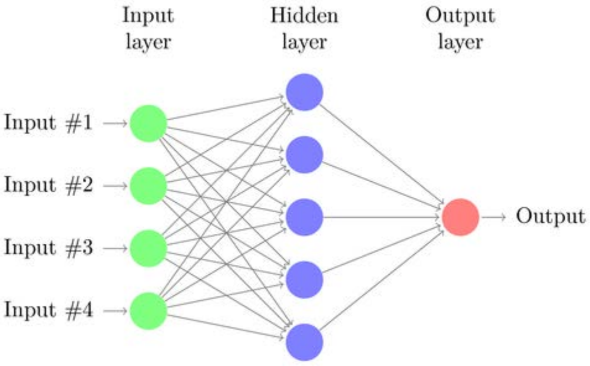
2019/03/20

What's to Be Covered Today…

- Intro to Neural Networks & CNN
  - Linear Classification
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks
- Pytorch Framework Tutorial (by TAs)

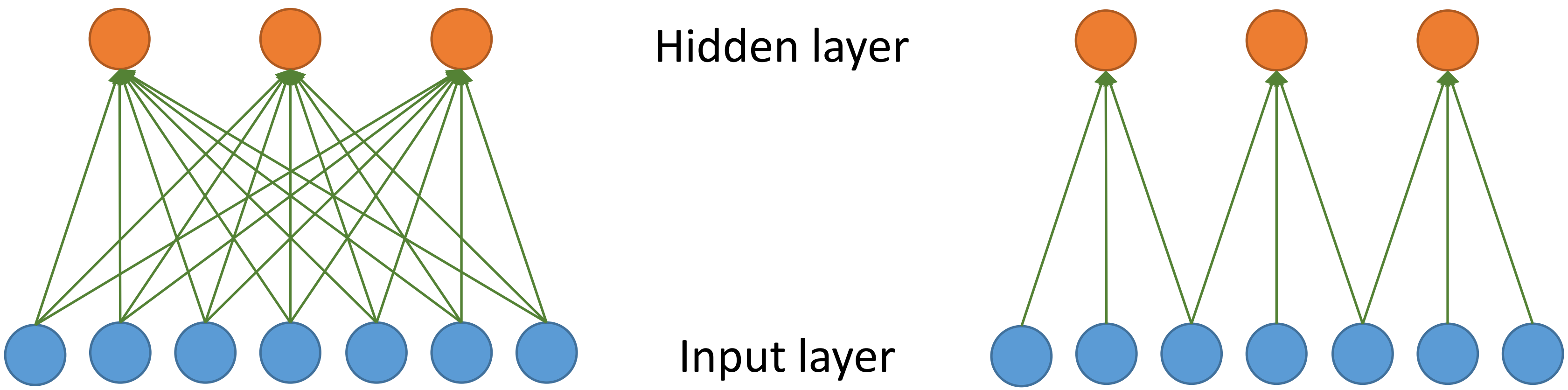
吳致緯

劉致廷



Many slides from Richard Turner, Fei-Fei Li, Yaser Sheikh, Simon Lucey, Kaiming He, and J.-B. Huang

CNN: Local Connectivity

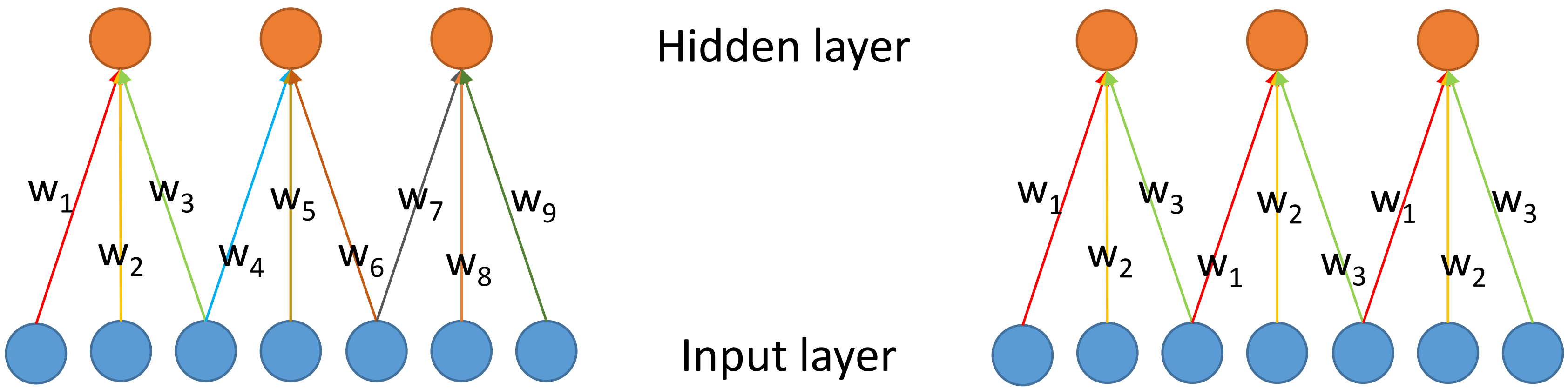


Global connectivity

Local connectivity

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Global connectivity: $7 \times 3 = 21$
  - Local connectivity: $3 \times 3 = 9$

CNN: Weight Sharing



Without weight sharing

With weight sharing

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Without weight sharing: 9
  - With weight sharing : 3

Putting them together

- Local connectivity
- Weight sharing
- Handling multiple input channels
- Handling multiple output maps

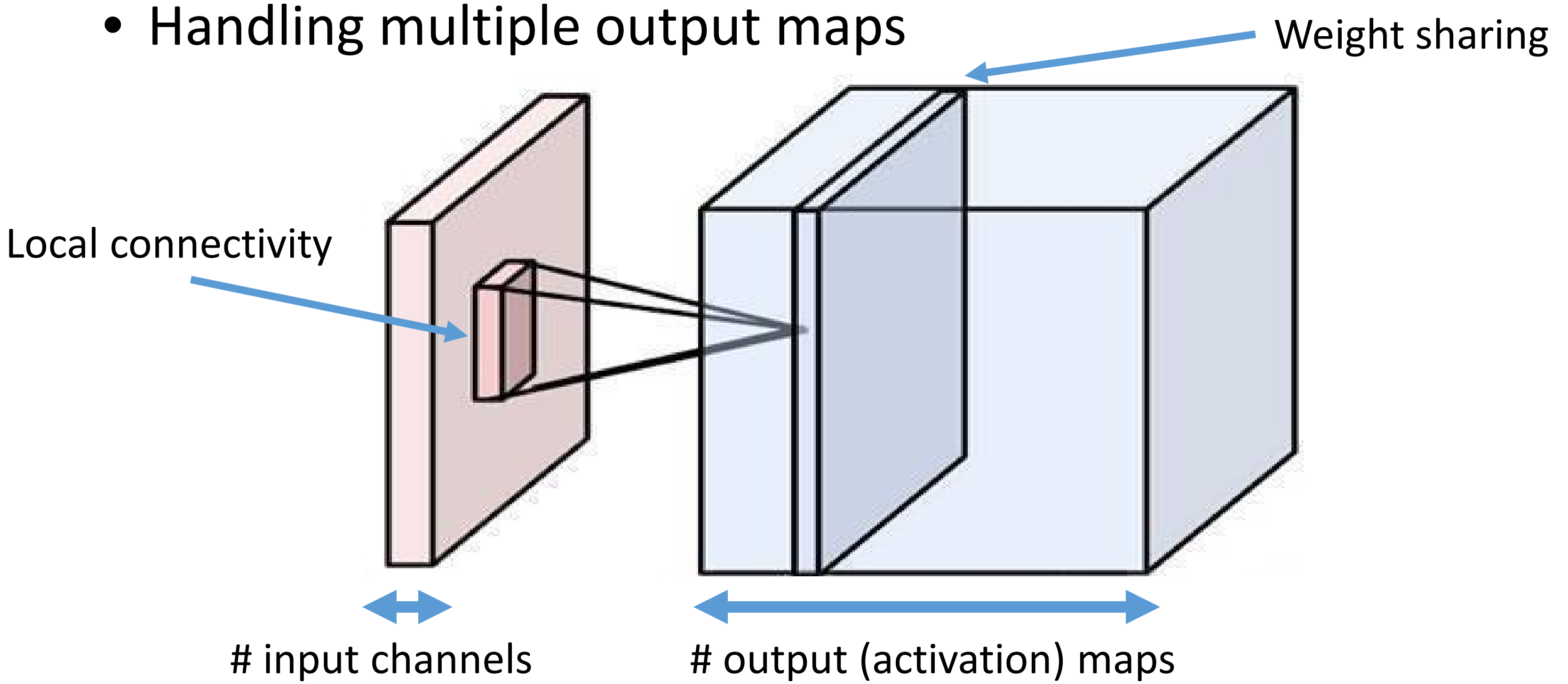
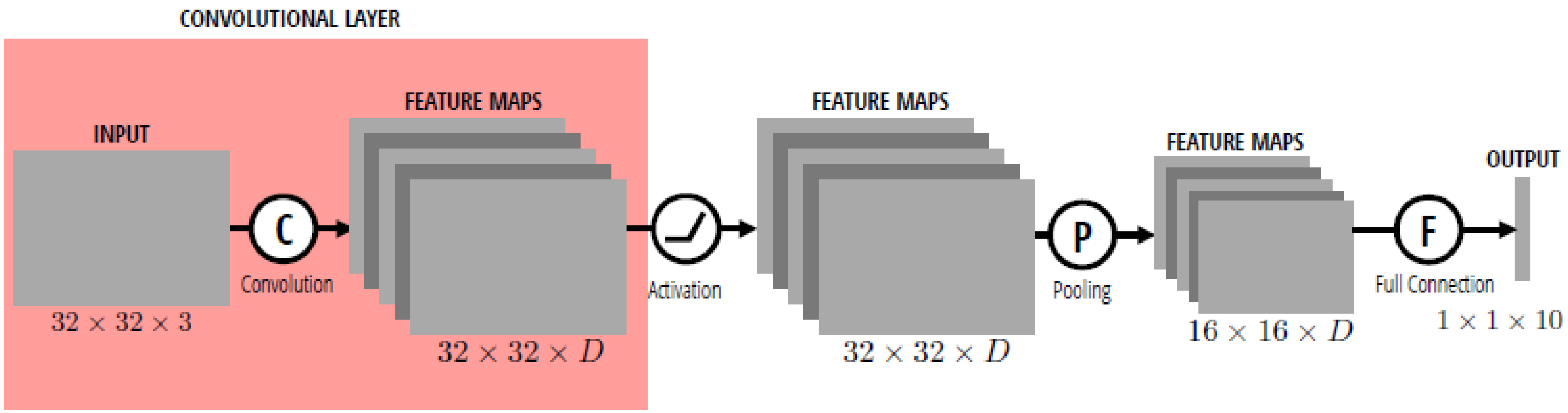


Image credit: A. Karpathy

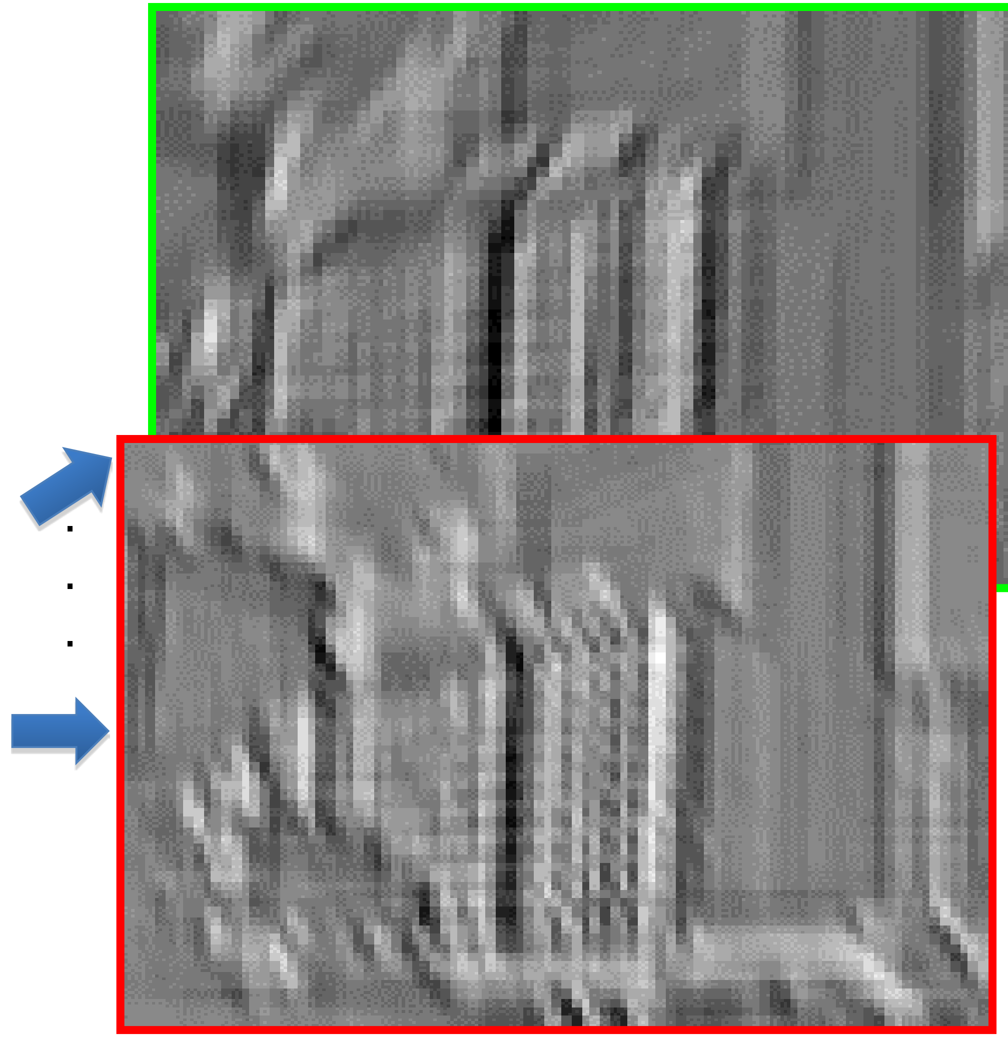
Convolution Layer in CNN



What is a Convolution?

- Weighted moving sum

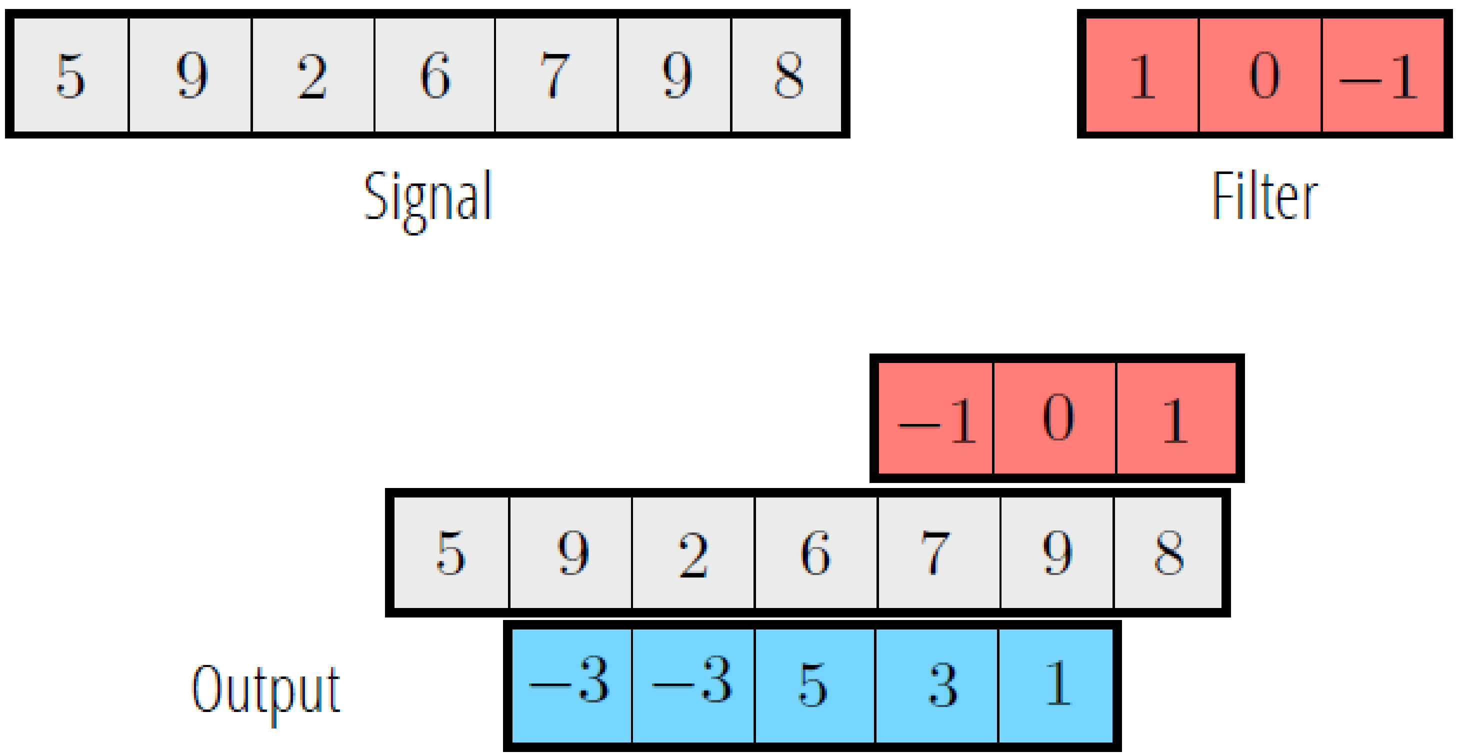
Input



Feature Activation Map

slide credit: S. Lazebnik

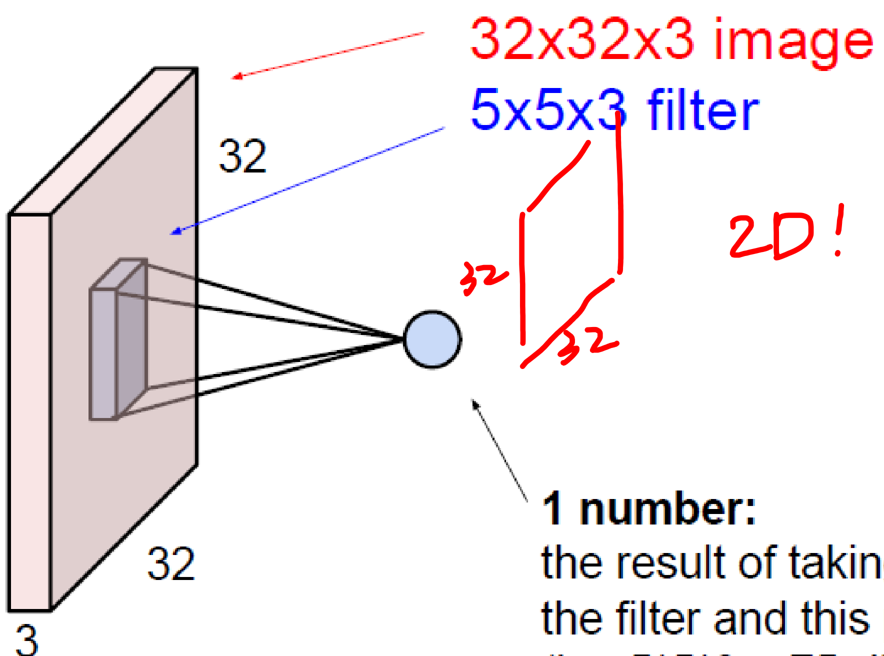
What is a Convolution?



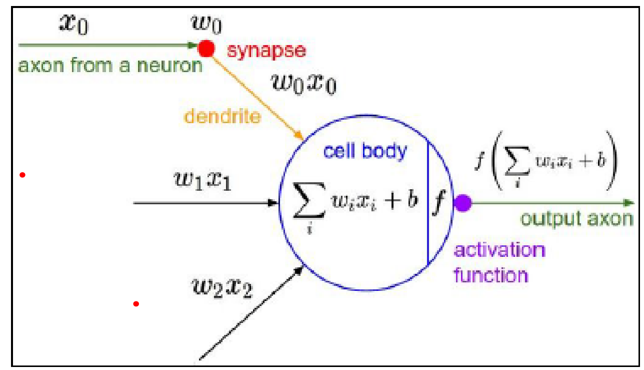
Convolution is a local linear operator

Putting them together (cont'd)

- The brain/neuron view of CONV layer



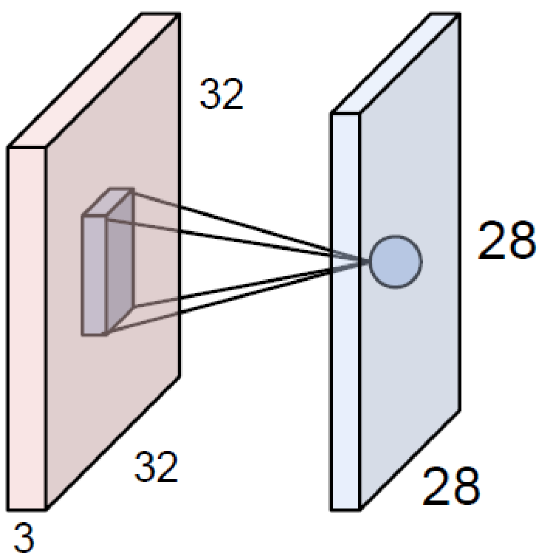
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



It's just a neuron with local connectivity...

Putting them together (cont'd)

- The brain/neuron view of CONV layer



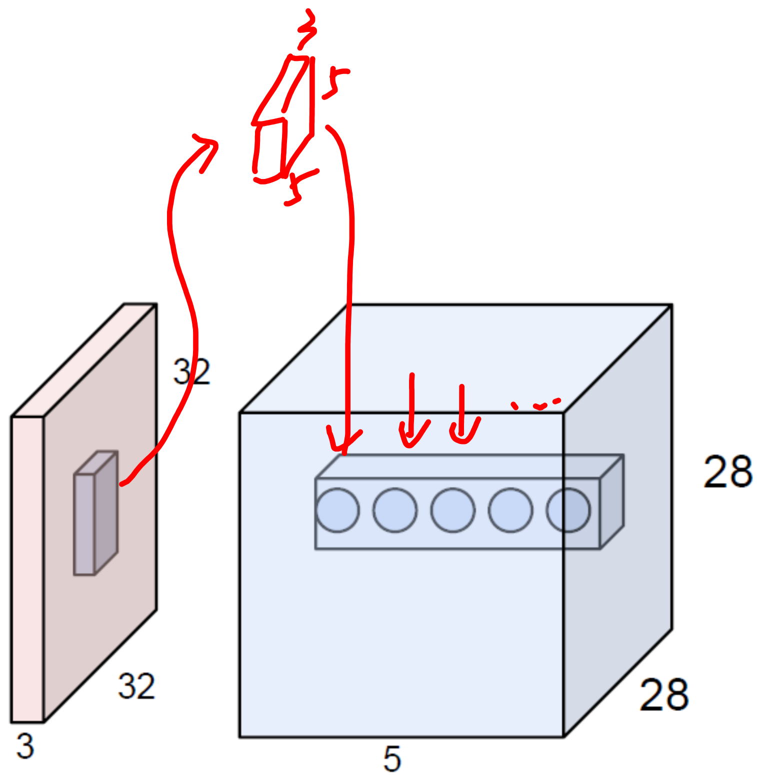
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

Putting them together (cont'd)

- The brain/neuron view of CONV layer

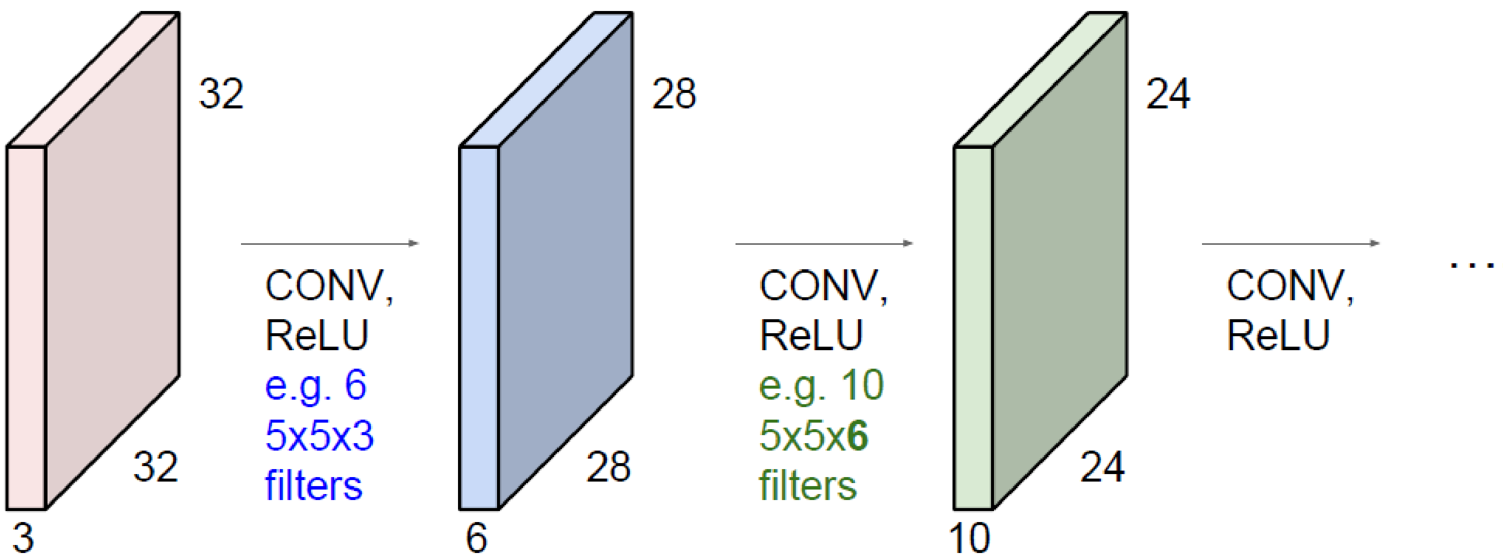


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

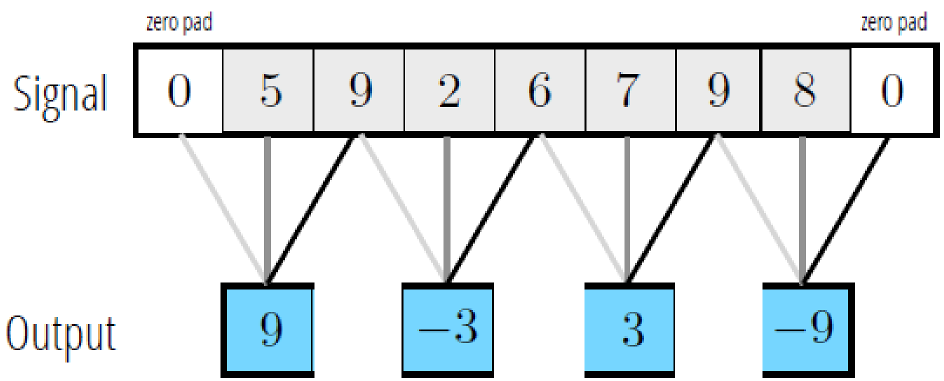
Putting them together (cont'd)

- Image input with 32 x 32 pixels convolved repeatedly with 5 x 5 x 3 filters shrinks volumes spatially (32 -> 28 -> 24 -> …).



What is a Convolution?

- Stride
  - Step size across signals



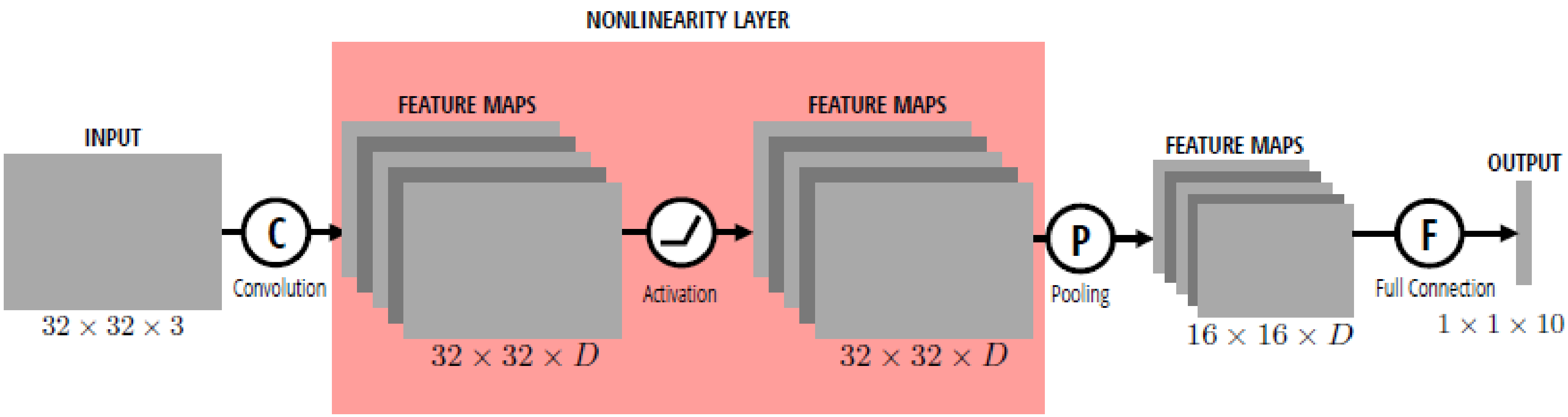
Input Size

Filter Size

Output Size $$\frac{N - c}{s} + 1$$

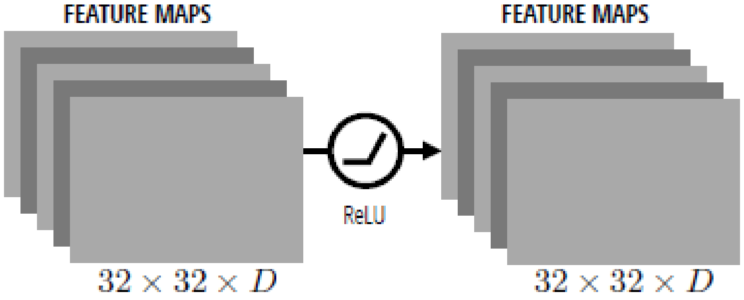
Stride: step size across the signal

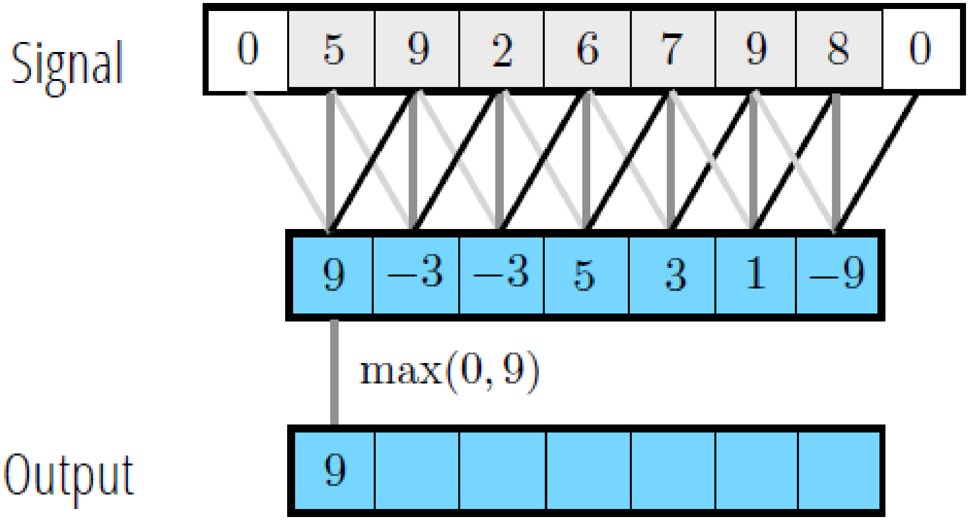
Nonlinearity Layer in CNN



Nonlinearity Layer

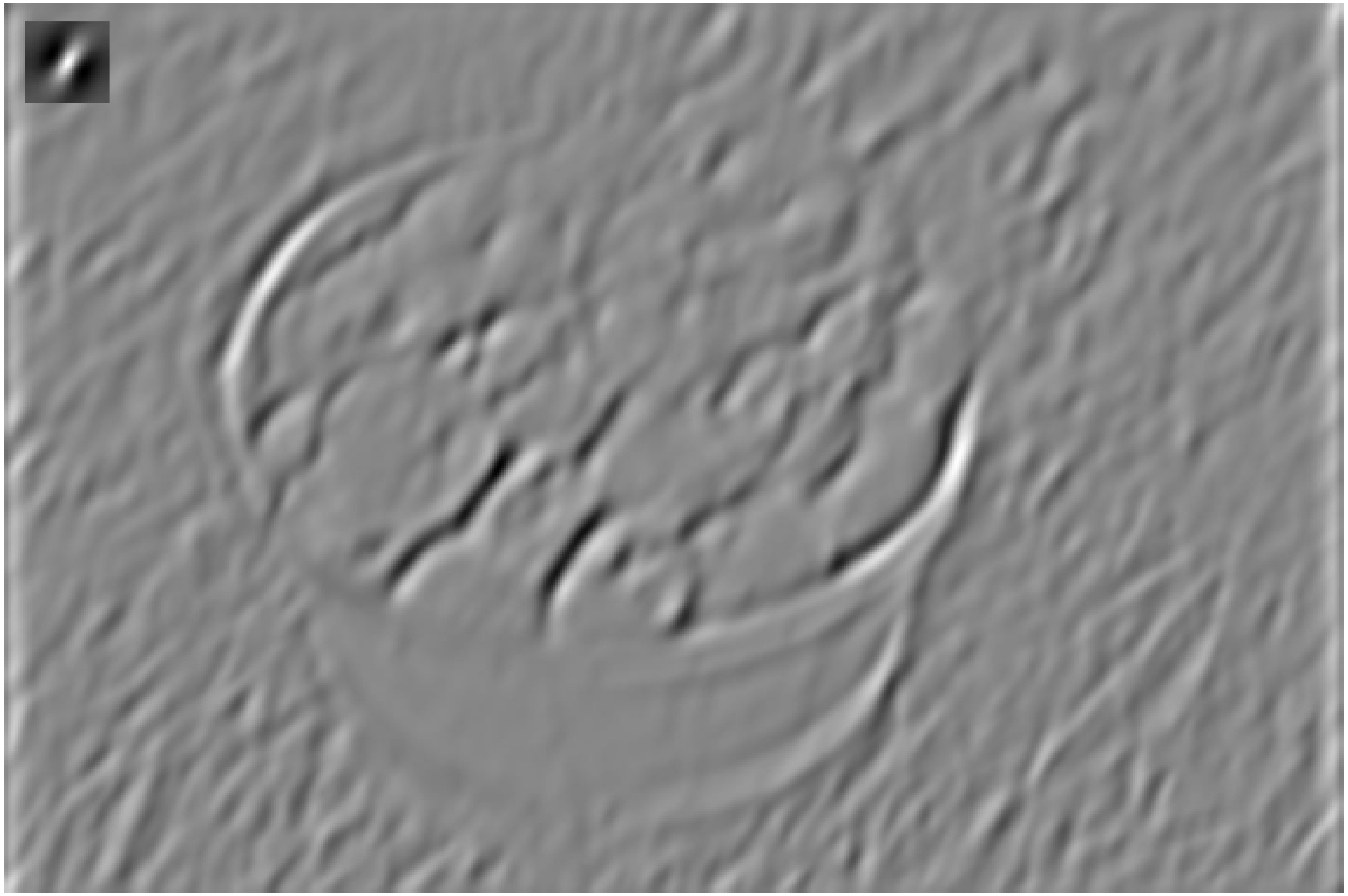
- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)





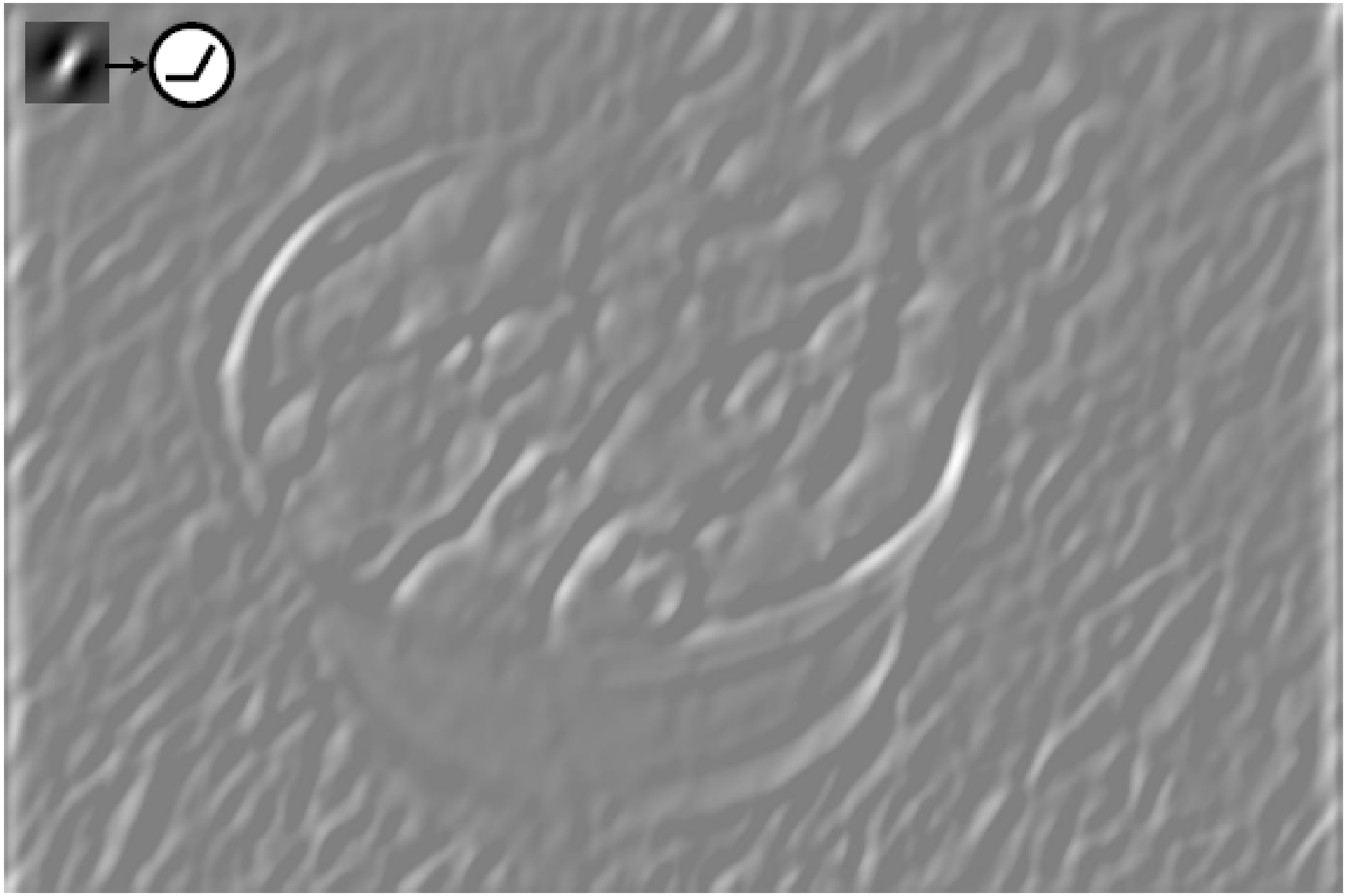
Nonlinearity Layer

- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)

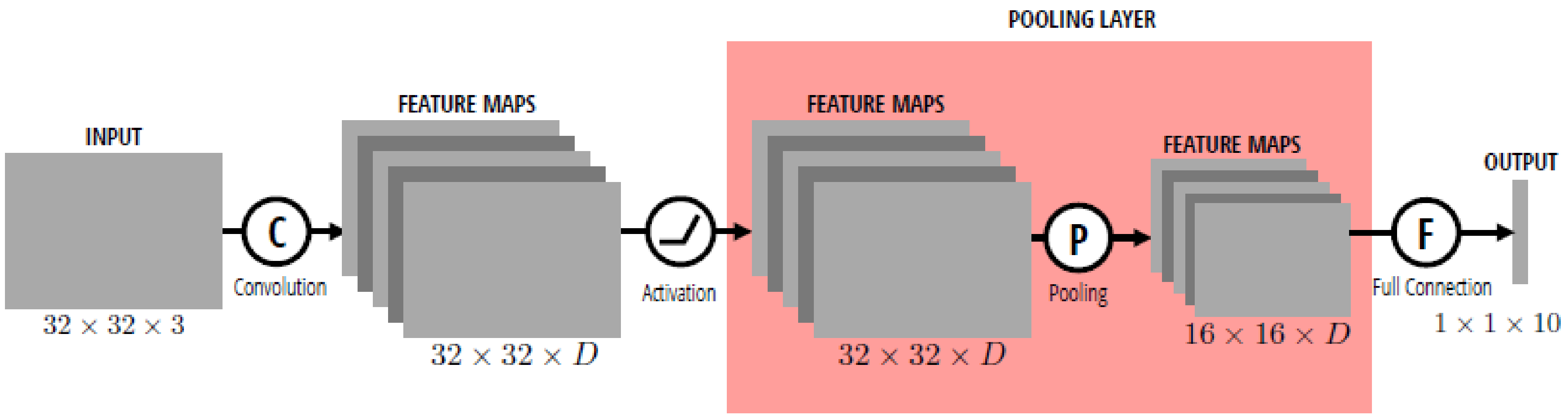


Nonlinearity Layer

- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)

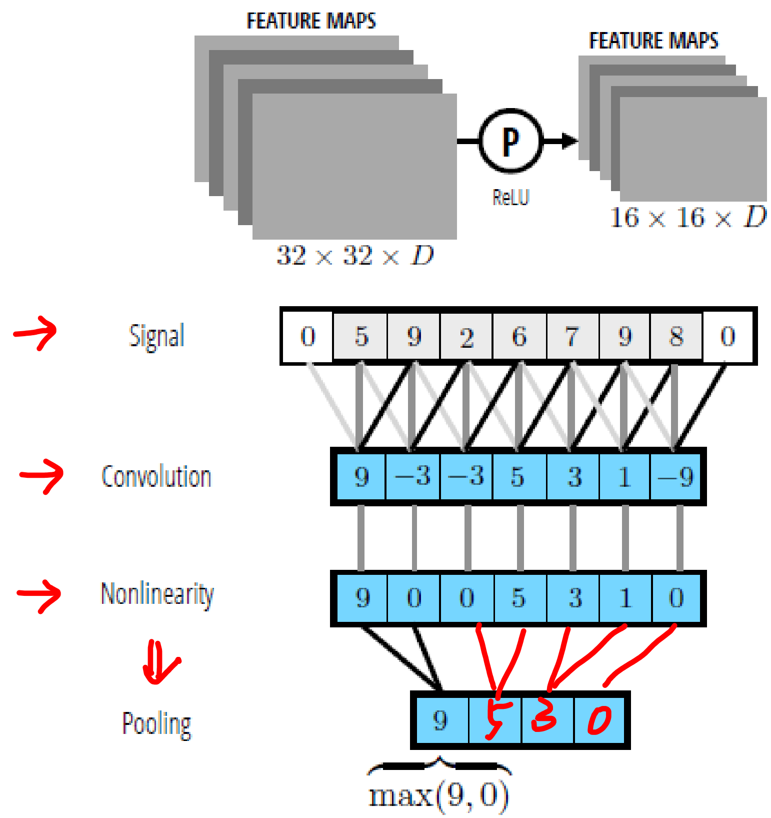


Pooling Layer in CNN



Pooling Layer

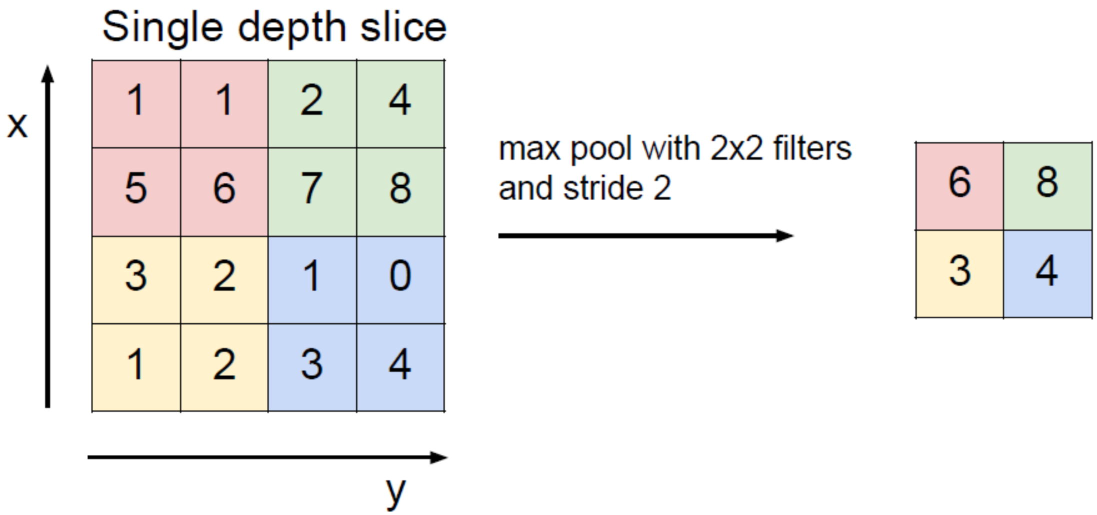
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling

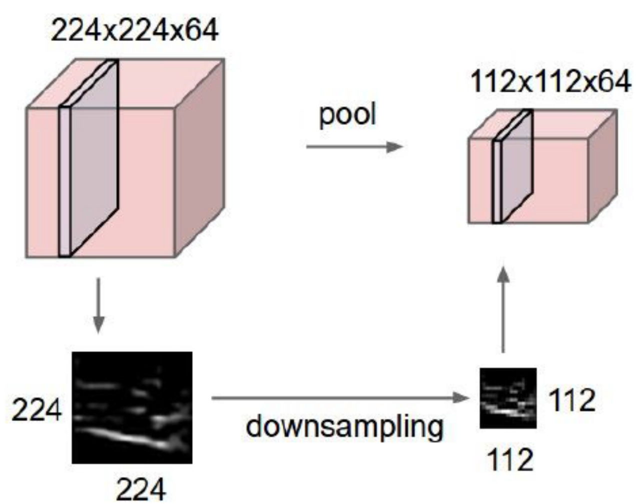


fine
↓
coarse

Pooling Layer

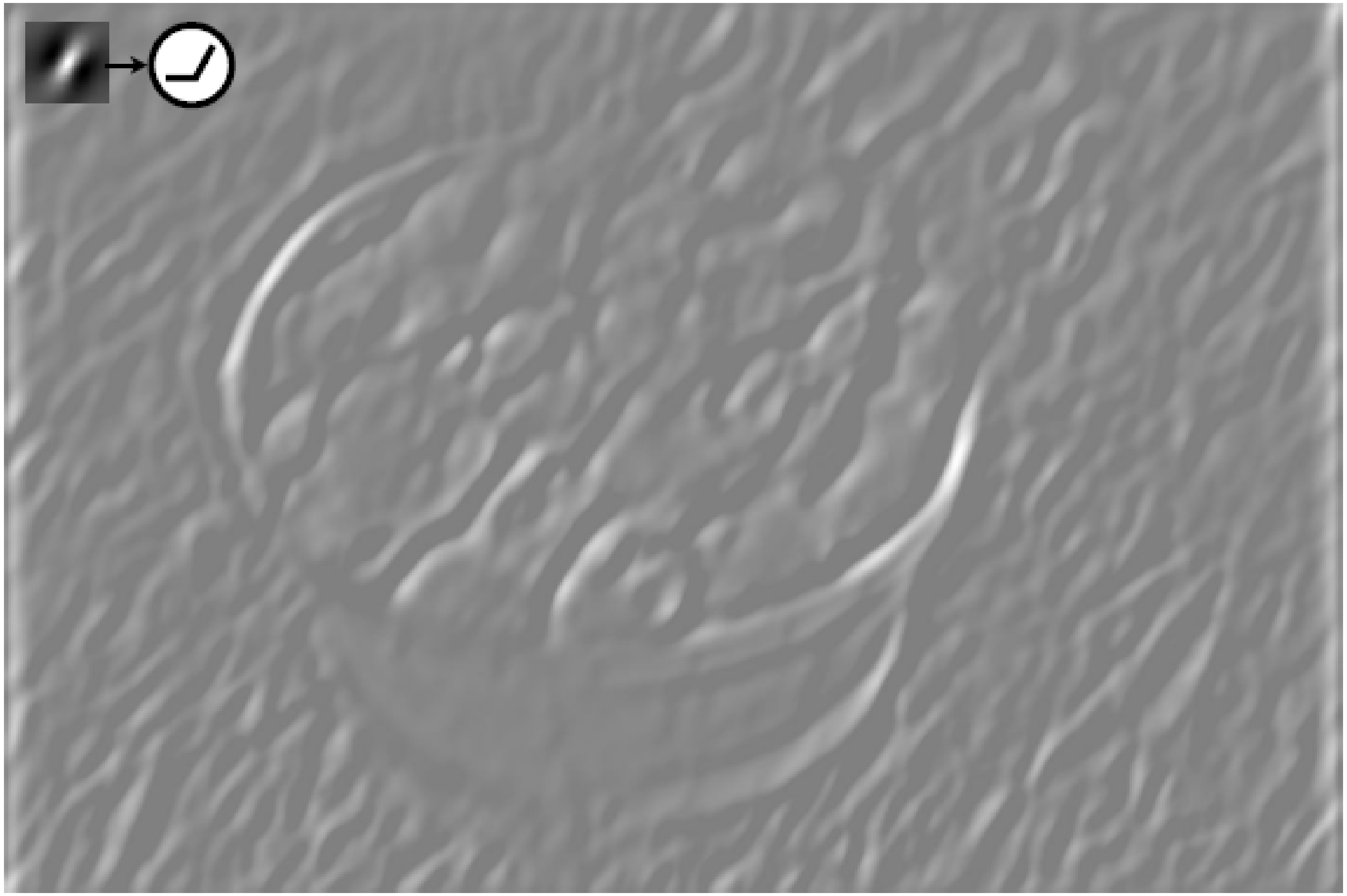
- Reduces the spatial size and provides spatial invariance



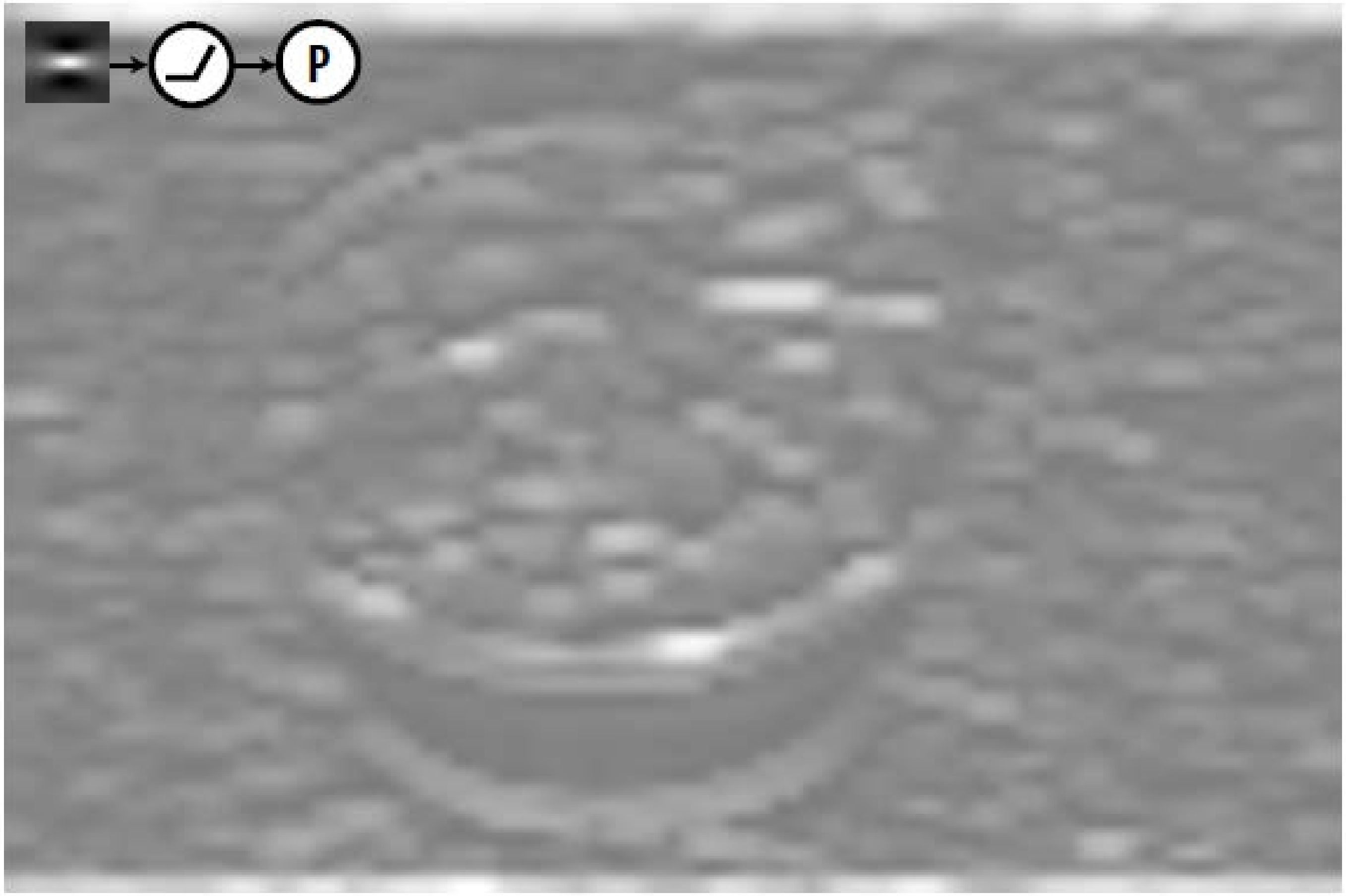


scale up!

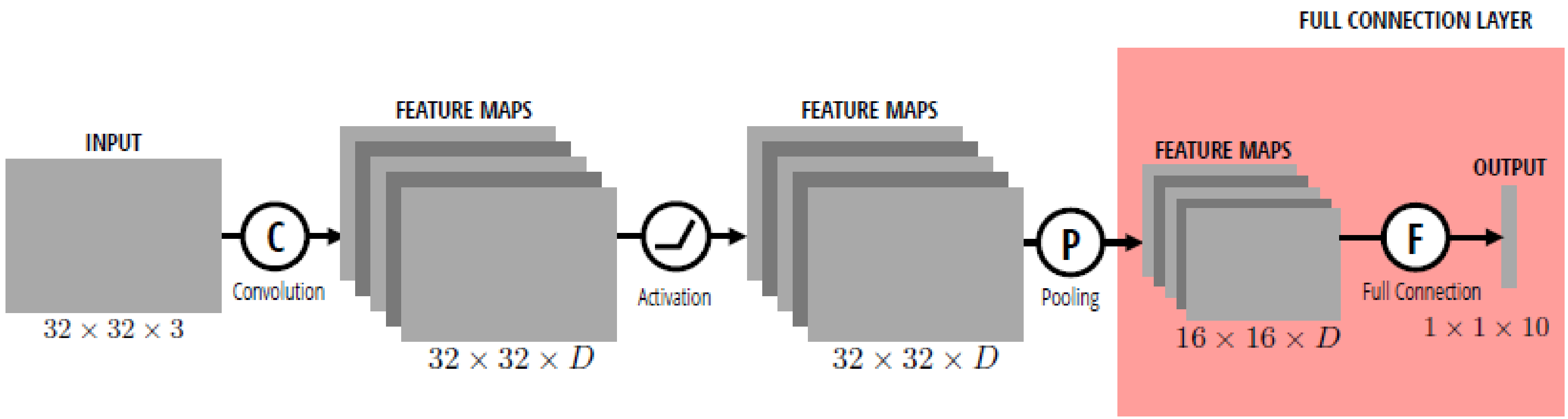
- Example
  - Nonlinearity by ReLU



- Example
  - Max pooling

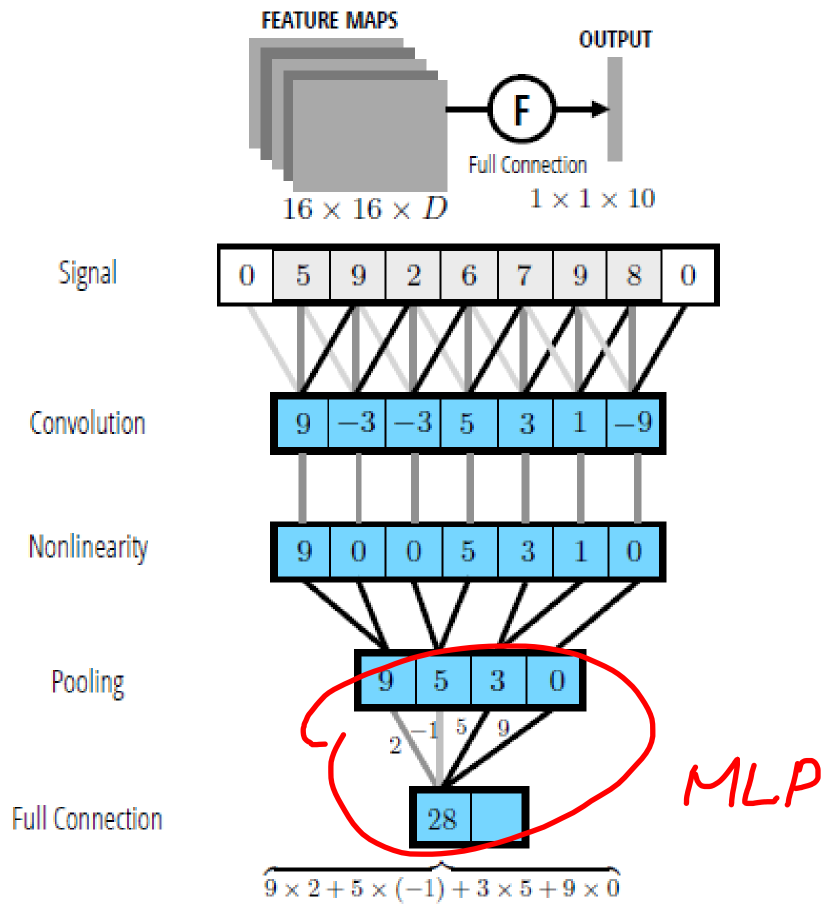


Fully Connected (FC) Layer in CNN



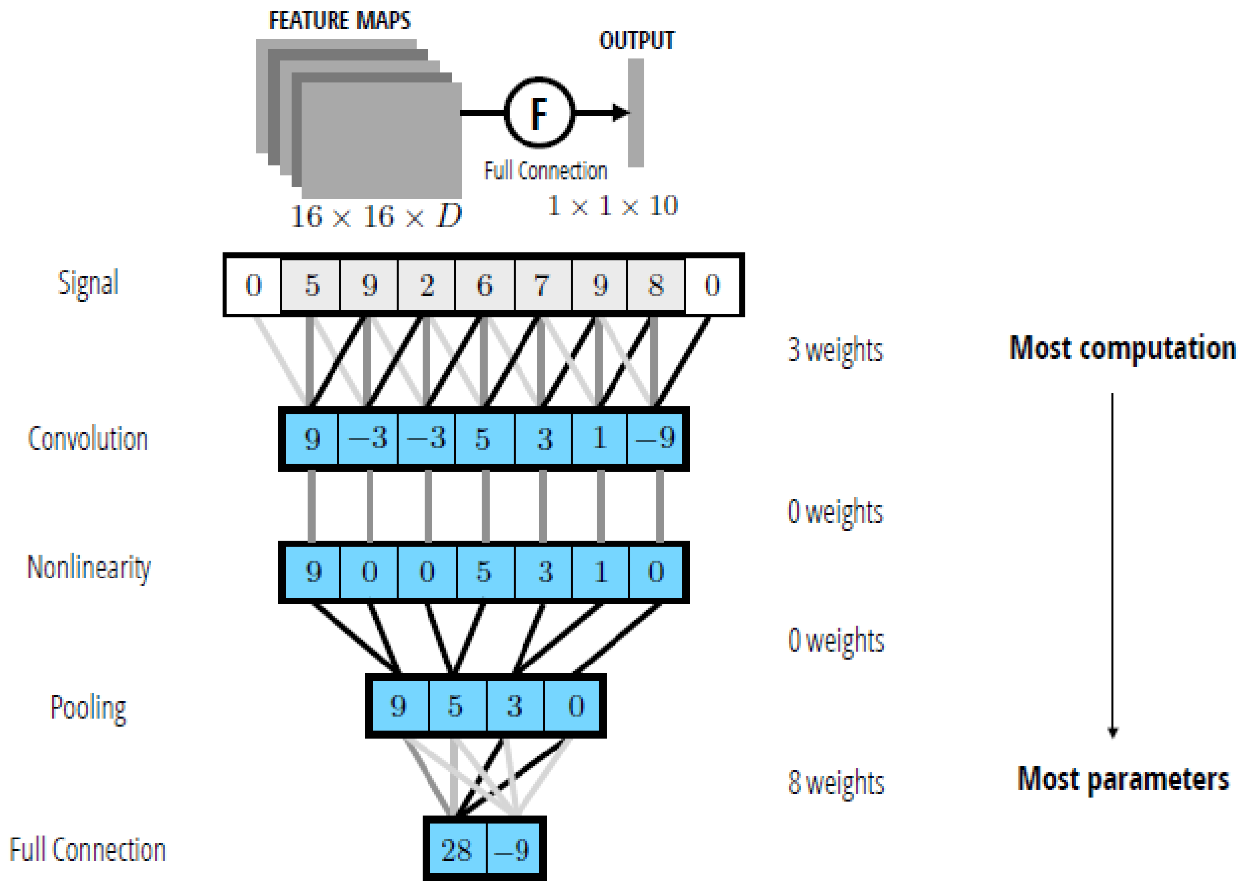
FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks

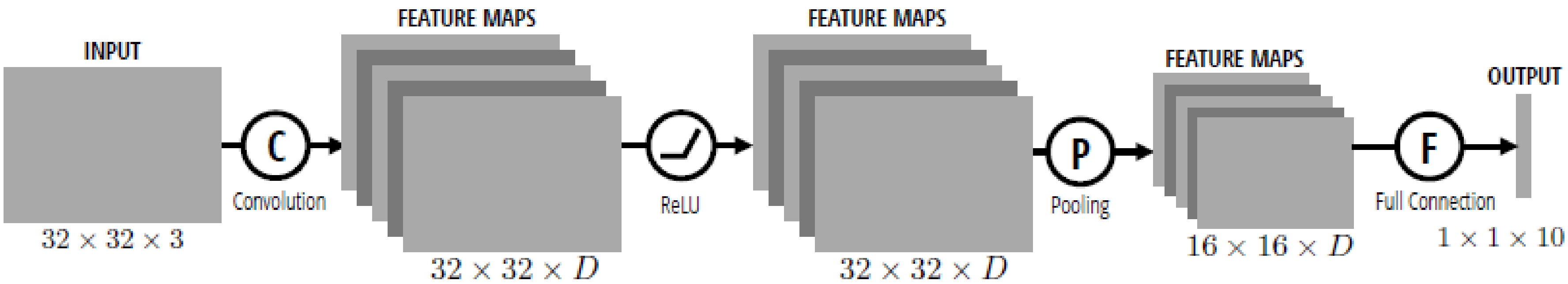


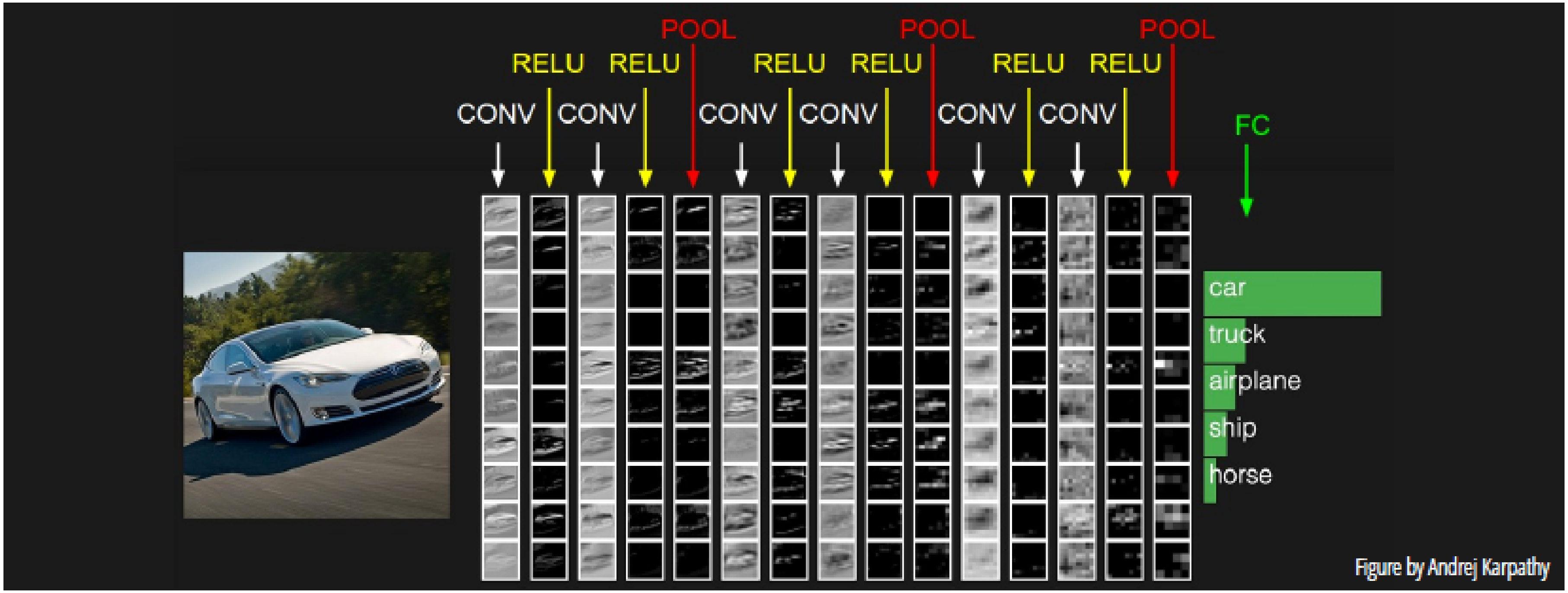
FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks



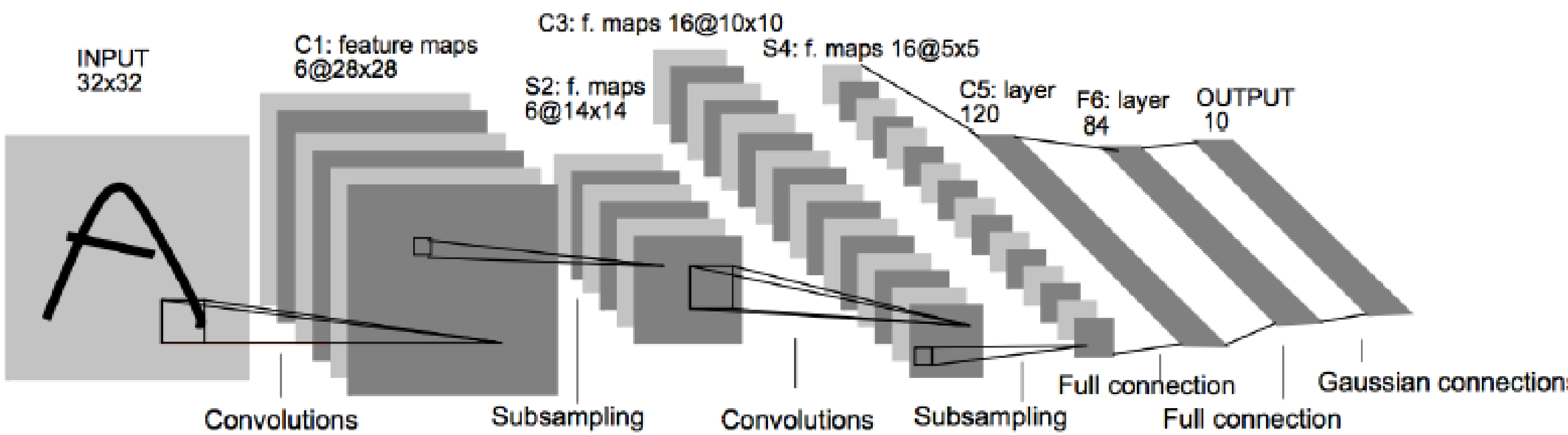
CNN


INPUT
Convolution
$32 \times 32 \times 3$
FEATURE MAPS
$32 \times 32 \times D$
ReLU
FEATURE MAPS
$32 \times 32 \times D$
Pooling
FEATURE MAPS
$16 \times 16 \times D$
Full Connection
OUTPUT
$1 \times 1 \times 10$


POOL
RELU
CONV
FC
car
truck
airplane
ship
horse
Figure by Andrej Karpathy

LeNet

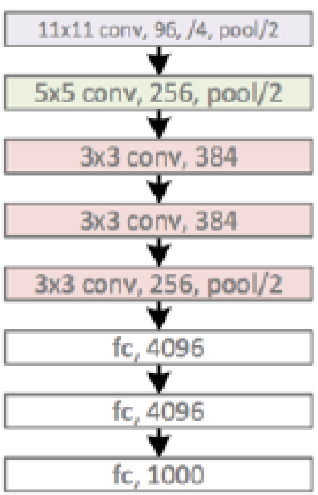
- Presented by Yann LeCun during the 1990s for reading digits
- Has the elements of modern architectures

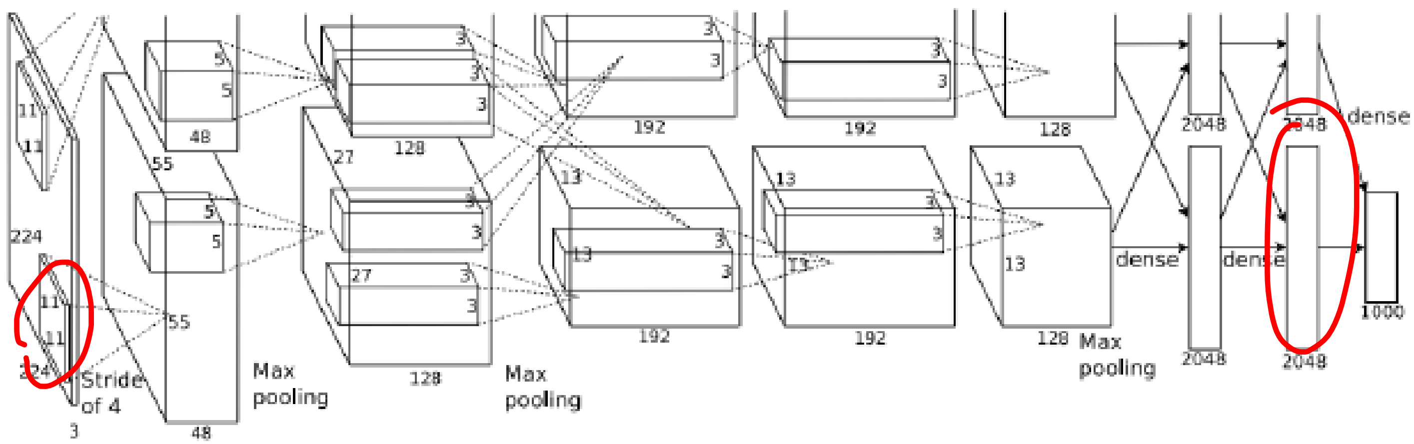


AlexNet [Krizhevsky et al., 2012]

- Repopularized CNN
  by winning the ImageNet Challenge 2012
- 7 hidden layers, 650,000 neurons,
  60M parameters
- Error rate of 16% vs. 26% for 2nd place.

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)





Deep or Not?

- Depth of the network is critical for performance.

AlexNet

AlexNet: 8 Layers with 18.2% top-5 error

Removing Layer 7 reduces 16 million parameters, but only 1.1% drop in performance!

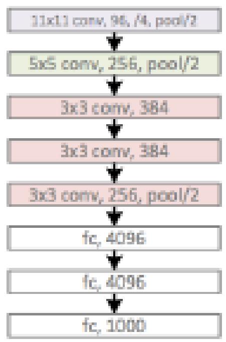
Removing Layer 6 and 7 reduces 50 million parameters, but only 5.7% drop in performance

Removing middle conv layers reduces 1 million parameters, but only 3% drop in performance

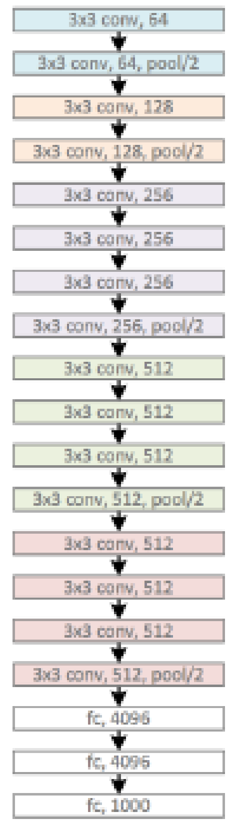
Removing feature & conv layers produces a **33% drop** in performance

CNN: A Revolution of Depth

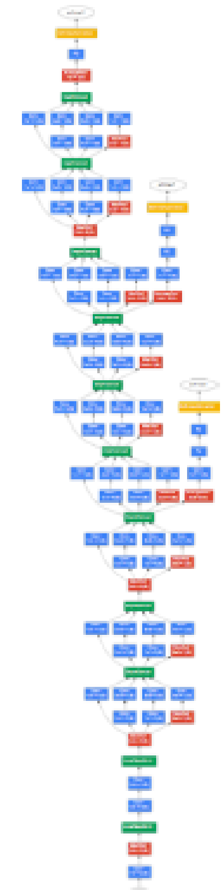
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

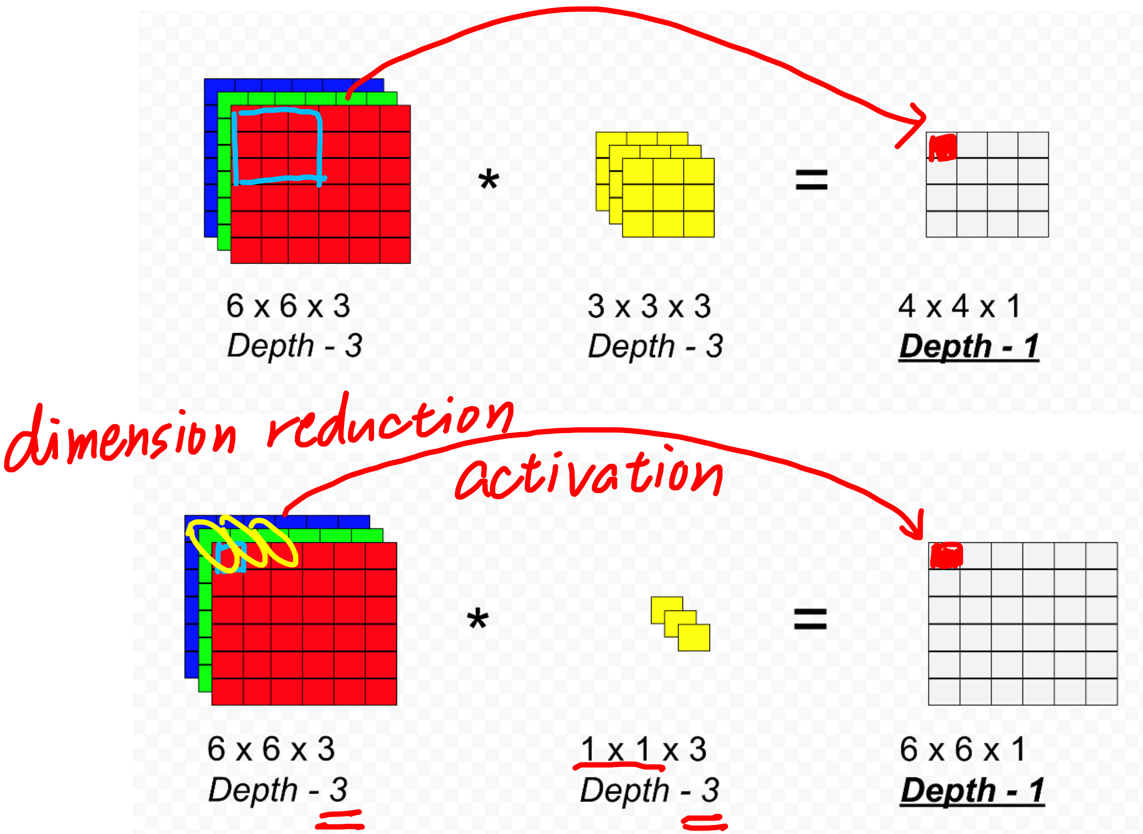


GoogleNet, 22 layers
(ILSVRC 2014)



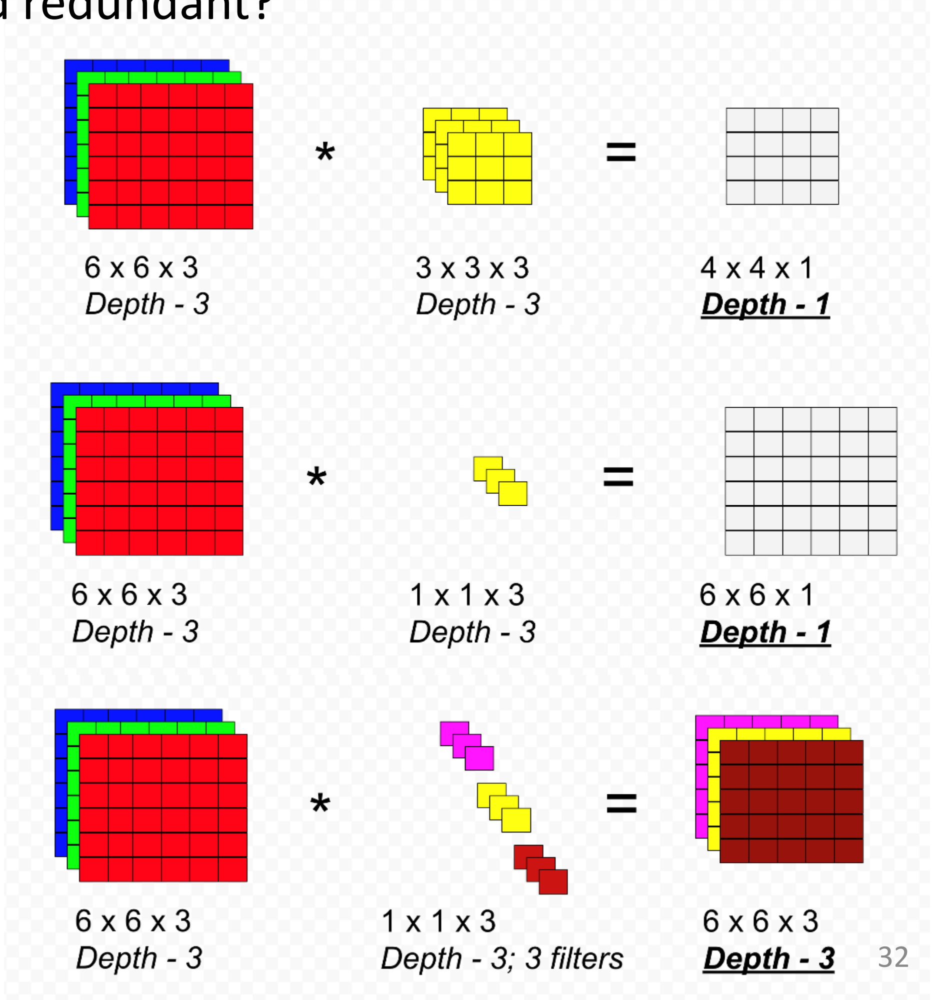
What is 1x1 Convolution?

- Doesn’t 1x1 convolution sound redundant?



What is 1x1 Convolution? (cont'd)

- Doesn't 1x1 convolution sound redundant?
- Simply speaking, it provides…
    - Dimension reduction (?)
    - Nonlinearity

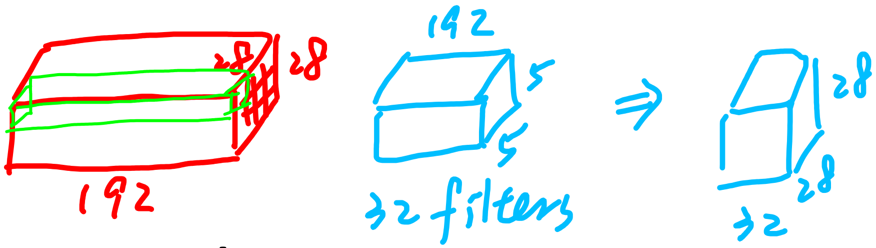


What is 1x1 Convolution? (cont'd)

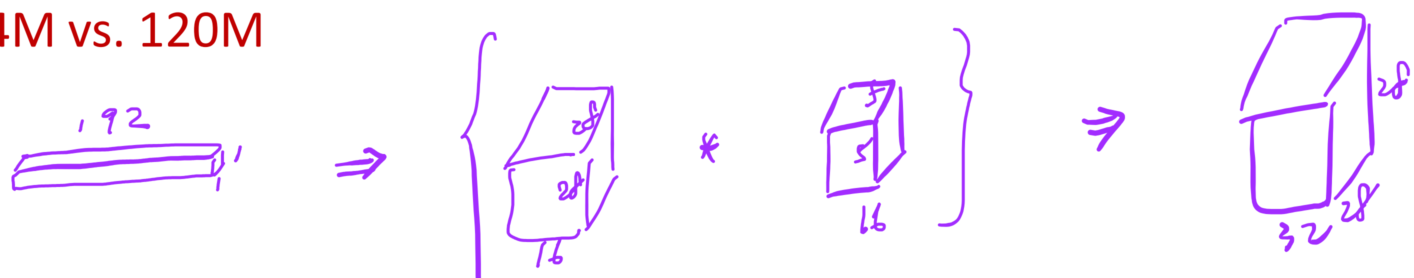
output at i^{th} layer

output at $i{+}1^{th}$ layer

- **Example 1**
  {28 x 28 x 192} convolved with 32 {5 x 5x 192} kernels into {28 x 28 x 32}
- (5 x 5 x 192) muls x (28 x 28) pixels x 32 kernels ~ 120M muls

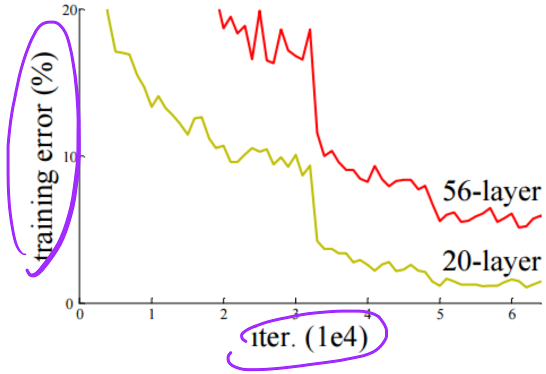


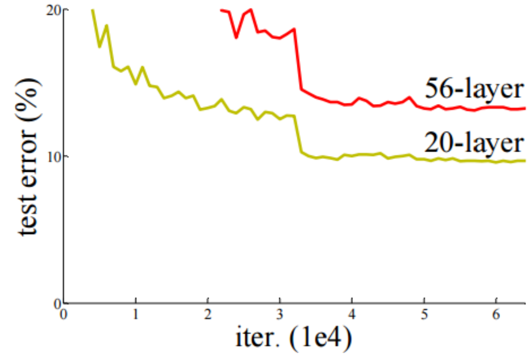
- **Example 2**
  {28 x 28 x 192} convolved with 1 16 {1 x 1x 192} kernels into {28 x 28 x 16}, followed by convolution with into 32 {5 x 5 x 16} kernels 2 into {28 x 28 x 32}
- 192 mul x (28 x 28) pixels x 16 kernels ~ 2.4M
- (5 x 5 x 16) muls x (28 x 28) pixels x 32 kernels ~ 10M
- 12.4M vs. 120M



ResNet

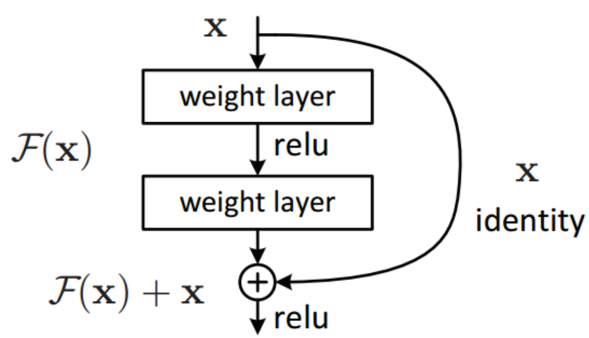
- Can we just increase the #layer?





gradient vanishing

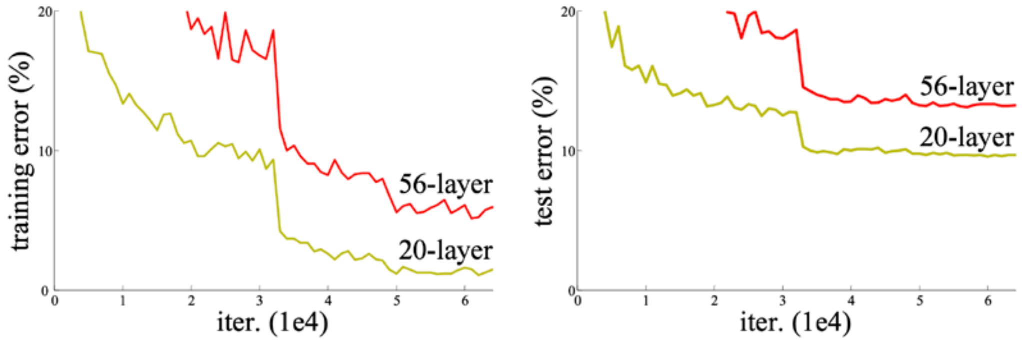
- How can we train very deep network?
  - Residual learning



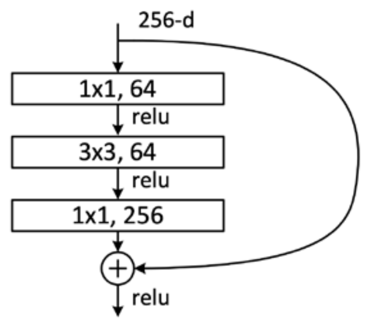
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

ResNet (cont'd)

- Can we just increase # of layers?



- How to train very deep networks?
  - Residual learning



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**

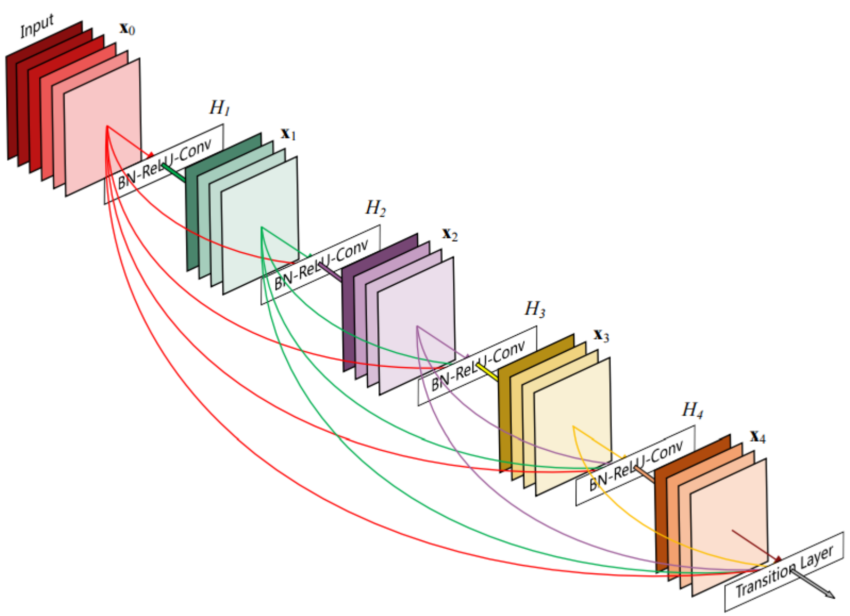
Non-Bottleneck (ResNet-18, 34)

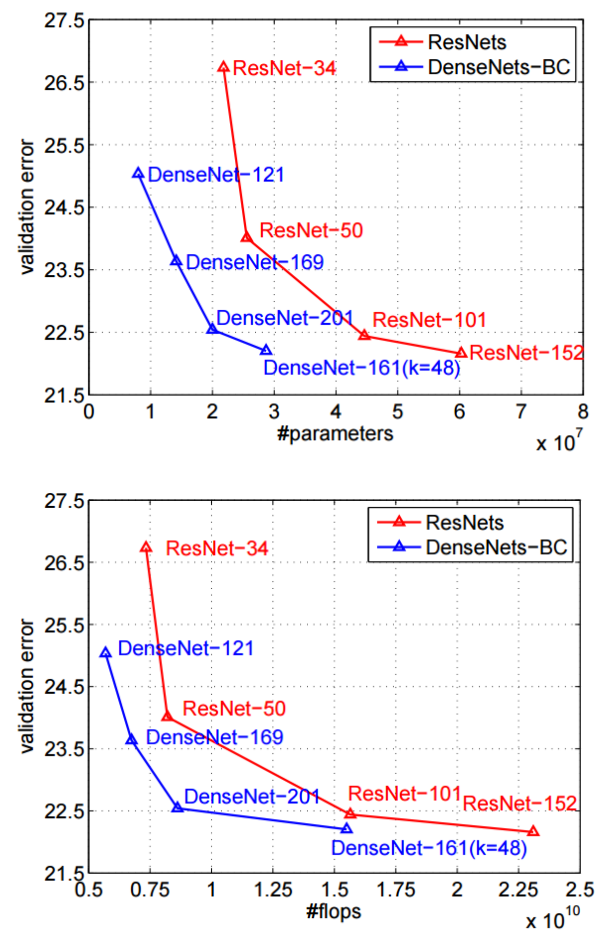
Bottleneck (ResNet-50, 101, 152)

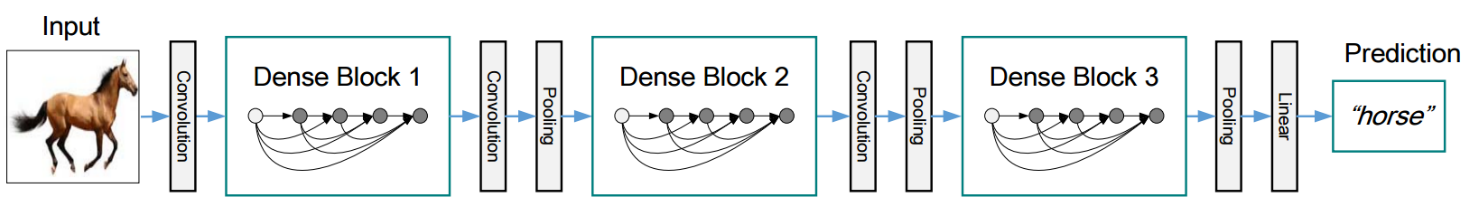
Ref: He, Kaiming, et al. "Deep residual learning for image recognition." *CVPR*, 2016.

DenseNet

- Shorter connections (like ResNet) help
- Why not just connect them all?

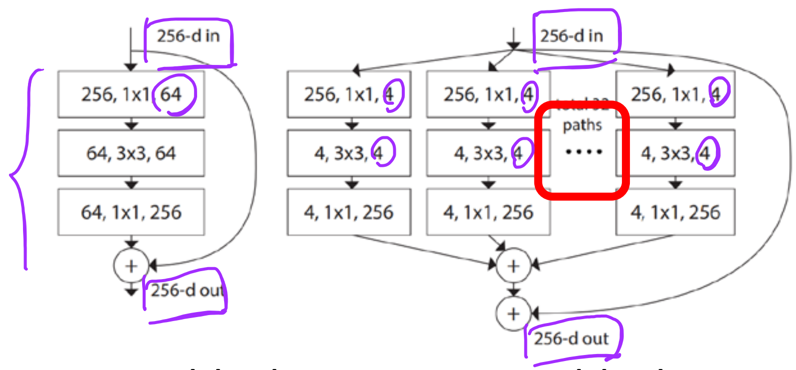






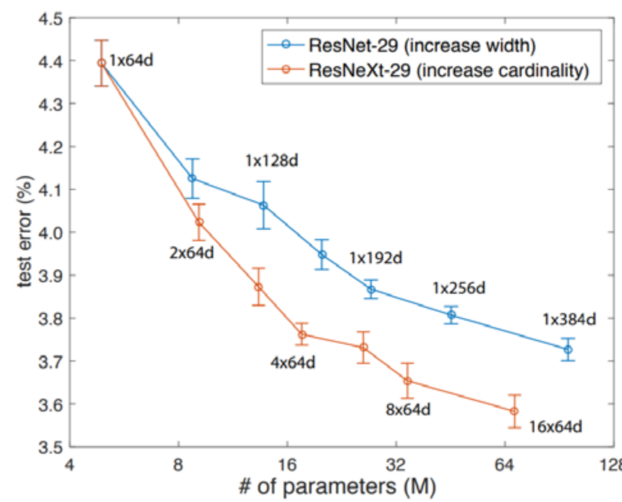
ResNeXT

- Deeper and wider → better…what else?
  - Increase cardinality



ResNet block ResNeXt block

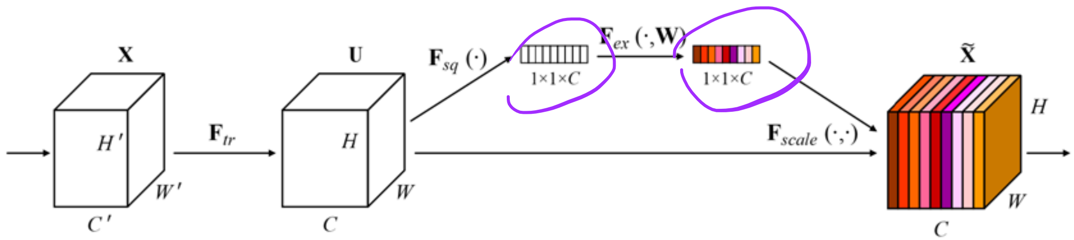
	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

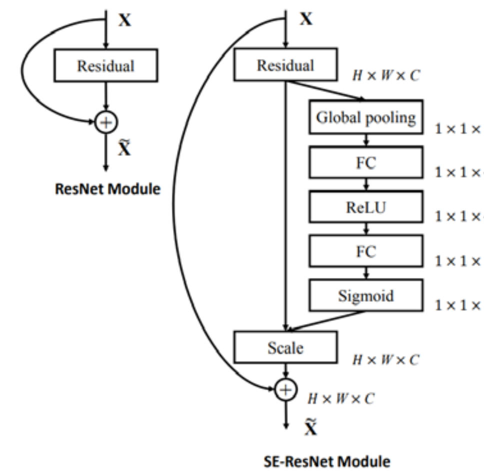


Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *CVPR,* 2017.

Squeeze-and-Excitation Net (SENet)

- How to improve acc. without much overhead?
  - Feature recalibration (channel attention)





	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [21]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." *CVPR,* 2018.

Remarks

- CNN:
  - Reduce the number of parameters
  - Reduce the memory requirements
  - Make computation independent of the size of the image
- Neuroscience provides strong inspiration on the NN design, but little guidance on how to train CNNs.
- Few structures discussed: convolution, nonlinearity, pooling

Training Convolutional Neural Networks

- Backpropagation +
  stochastic gradient descent with momentum
  - [Neural Networks: Tricks of the Trade]
- Dropout
- Data augmentation
- Batch normalization

An Illustrative Example

$$f(x,y) = xy, \qquad \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x$$

Example: $x = 4, y = -3 \Rightarrow f(x,y) = -12$

Partial derivatives

$$\frac{\partial f}{\partial x} = -3, \qquad \frac{\partial f}{\partial y} = 4$$

Gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

Example credit: Andrej Karpathy

$$f(x, y, z) = (x + y)z = qz$$

$q = x + y$

$$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

$f = qz$

$$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$$

Goal: compute the gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]$$

Example credit: Andrej Karpathy

$$f(x,y,z) = (x+y)z = qz$$

$$q = x + y$$

$$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

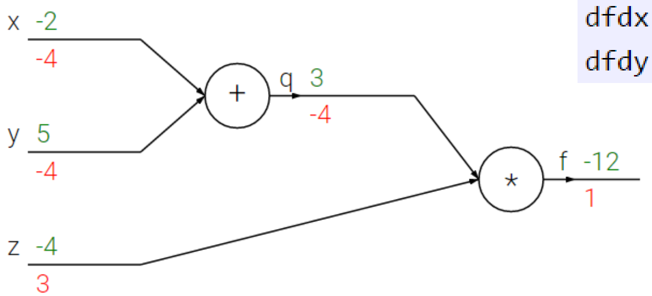
$$f = qz$$

$$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$$

Chain rule:

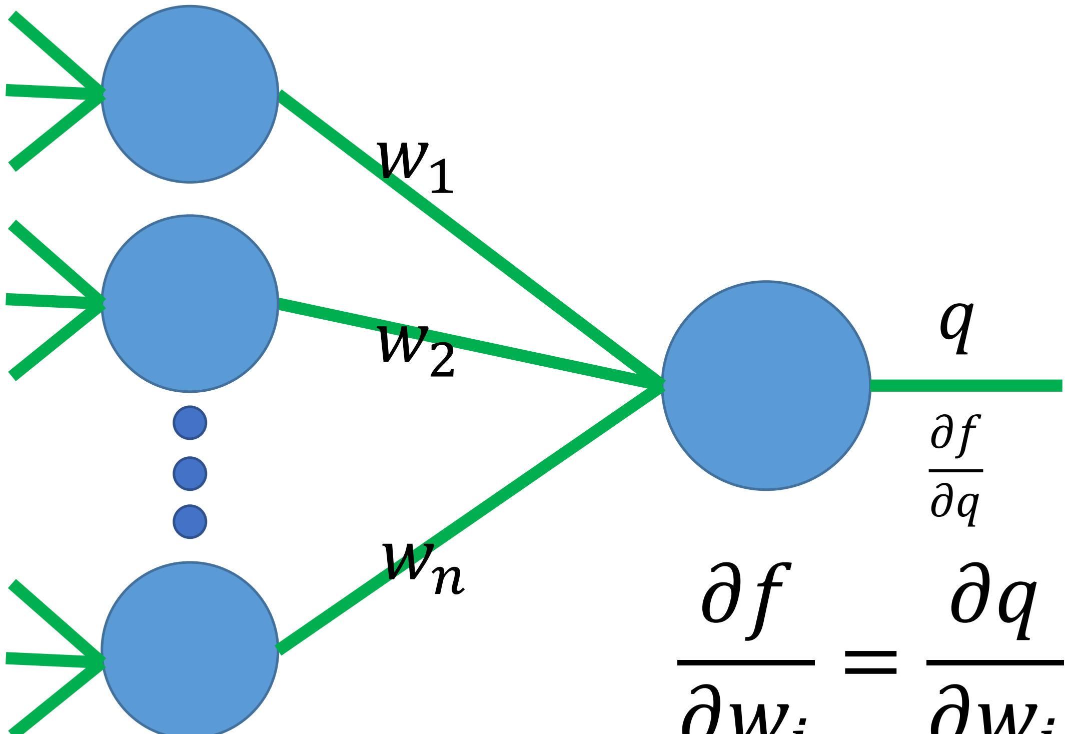
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```



Example credit: Andrej Karpathy

Backpropagation (recursive chain rule)



$$\frac{\partial f}{\partial w_i} = \frac{\partial q}{\partial w_i}\frac{\partial f}{\partial q}$$

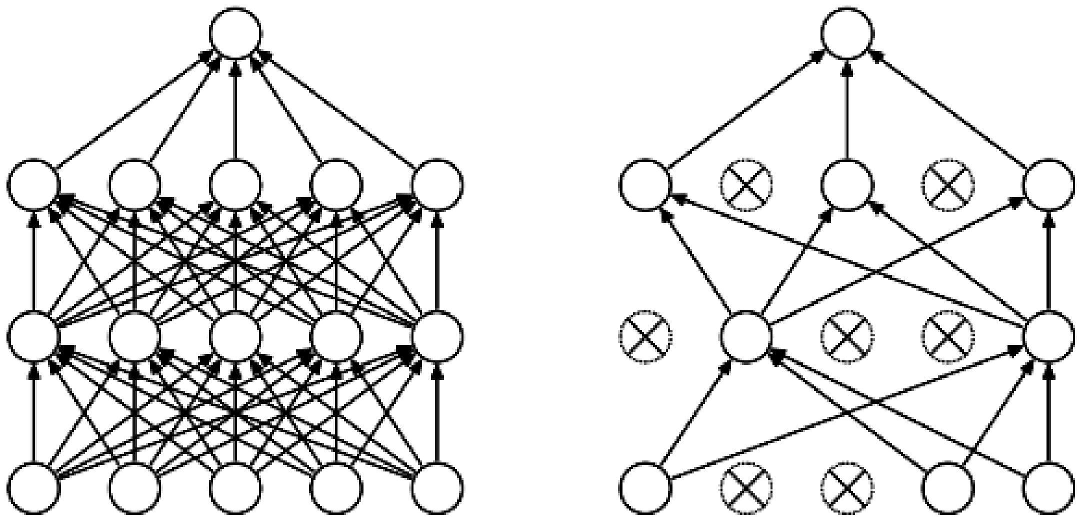
Local gradient

Gate gradient

Can be computed during forward pass

The gate receives this during backprop

Dropout



(a) Standard Neural Net (b) After applying dropout.

Intuition: successful conspiracies

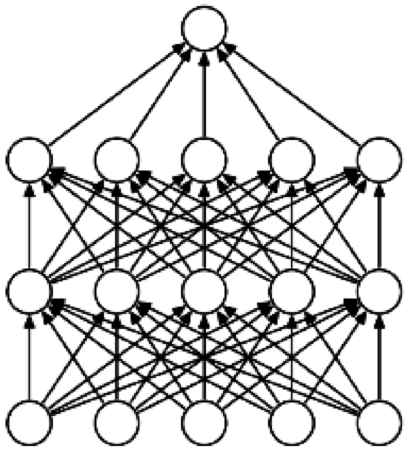
- 50 people planning a conspiracy

- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.

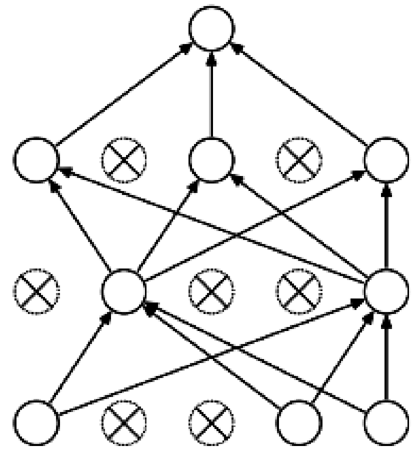
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

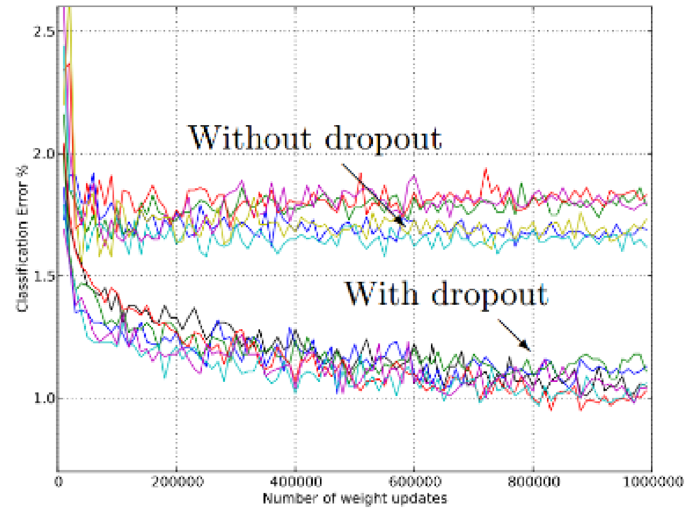
Dropout

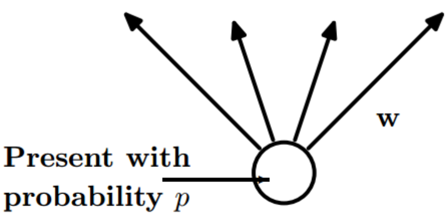


(a) Standard Neural Net

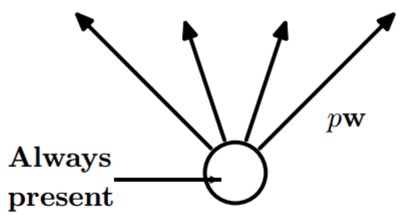


(b) After applying dropout.





(a) At training time



(b) At test time

Main Idea: approximately combining exponentially many different neural network architectures efficiently

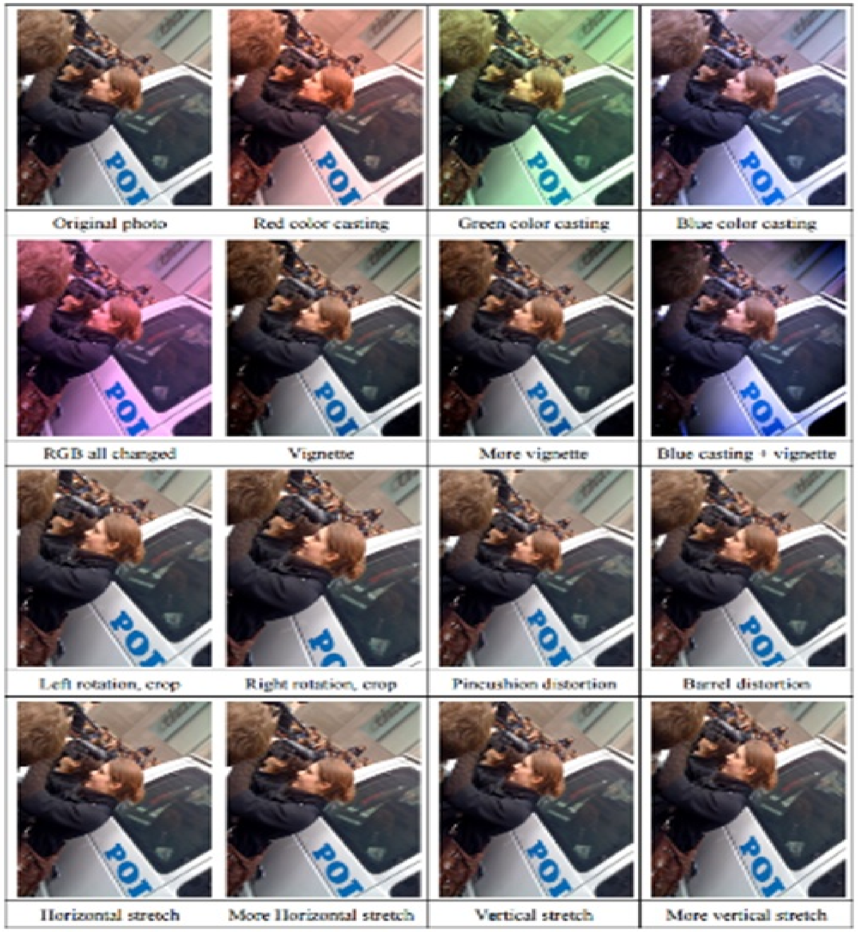
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

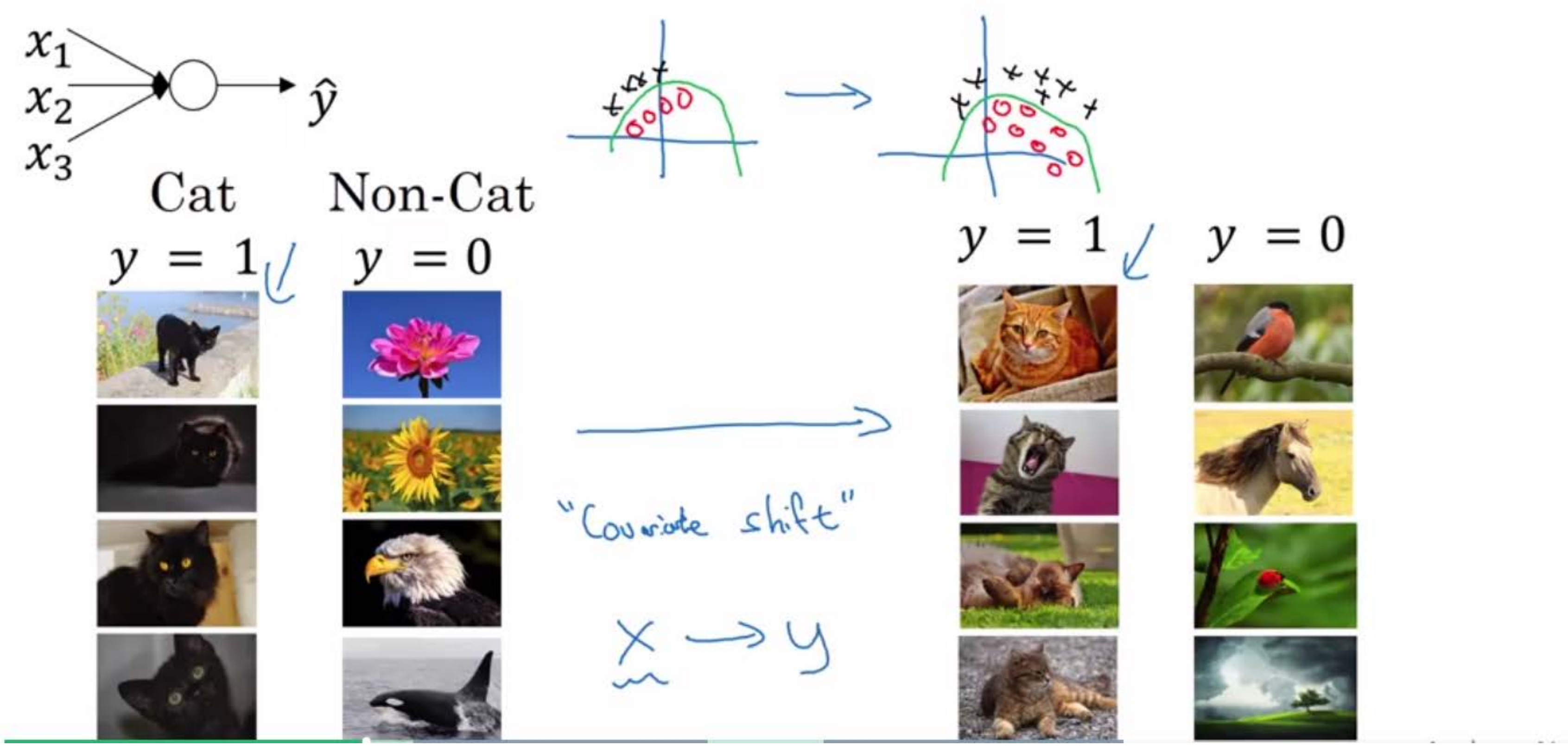
Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

Batch Normalization



Credit: Andrew Ng

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

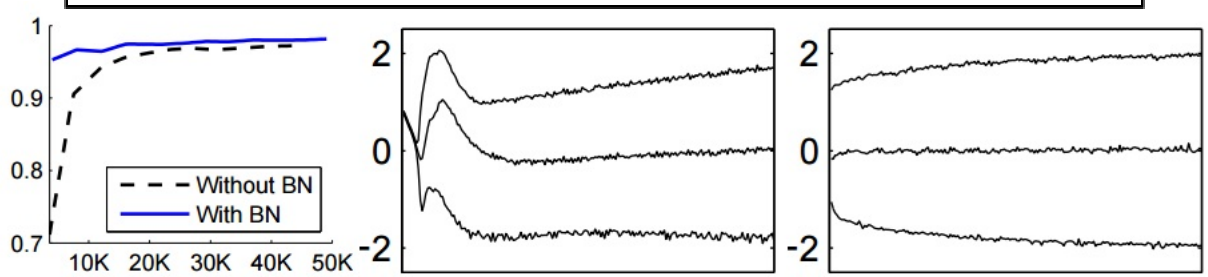
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



(a) (b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

What Will We Cover Next Week?

- **Pytorch Framework Tutorial** (for those who are not familiar with Pytorch)
  - Introduction to Pytorch
    - Installation guide
    - Basic concept of computation graph and back propagation
  - Basic: Module Class
    - How to build complex model with pytorch built-in classes.
  - Basic: DataSet & DataLoader Class
    - How to load data efficiently with pytorch built-in classes.
  - Hands on example : Image Classification Task **(bring your own laptop!)**
  - Advance :
    - Finetuning with pretrained model.
    - Data augmentation
    - Training with multiple GPU
    - Exporting models to other platforms.

- **HW #1 is due 3/23 Sat 3AM & no late submission!!**