Deep Learning for Computer Vision

113-1/Fall 2024; Classroom ~~BL112~~ -> **9:30am @ BL113**

https://cool.ntu.edu.tw/courses/41702 (NTU COOL)

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

Yu-Chiang Frank Wang 王鈺強, Professor

Dept. Electrical Engineering, National Taiwan University

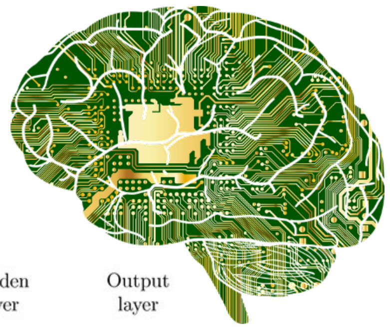
2024/09/10

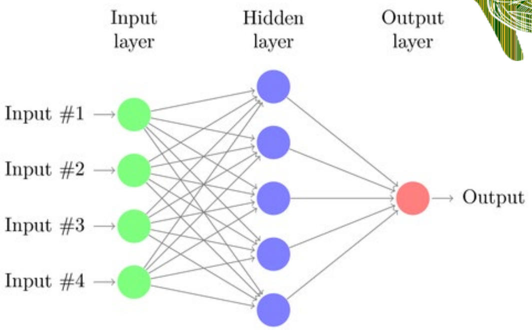
Slightly updated syllabys

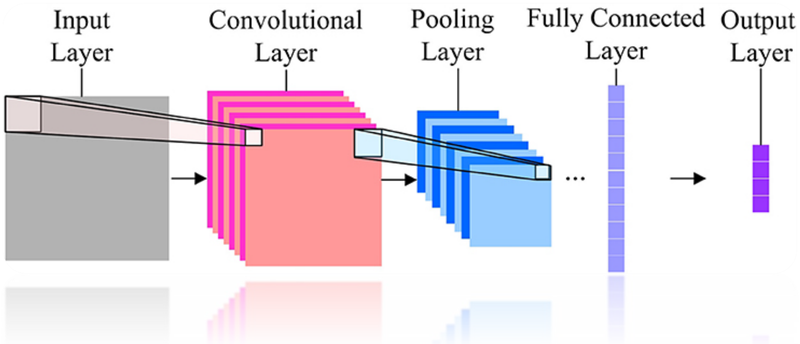
Week	Date	Topic	Course Materials	Remarks
1	09/03	Course Logistics & Registration; Intro to Neural Nets	W1-1 W1-2	
2	09/10	Convolutional Neural Networks & Image Segmentation		**HW #1 out**
3	09/17	**No class**		Mid-Autumn Festival
4	09/24	Generative Models (I) - Diffusion Model		**HW #1 due**
5	10/01	**Guest Lecture:** Dr. Jun-Cheng Chen, Academia Sinica		ECCV week
6	10/8	Generative Models (II) - AE, VAE & GAN		**HW # 2 out**
7	10/15	Recurrent Neural Networks & Transformer		
8	10/22	Transformer; Vision & Language Models		
9	10/29	Vision & Language Models; Multi-Modal Learning		**HW #2 due; HW #3 out**
10	11/05	Parameter-Efficient Finetuning; Unlearning, Debiasing, and Interoperability		
11	11/12	**Guest Lecture:** Linda Huang, Senior Dir., GeValyn Associates		
12	11/19	3D Vision		**HW #3 due; HW #4 out**
13	11/26	Object Detection		**Final Project Announcement**
14	12/03	**Guest Lecture:** Prof. Ming-Ching Chang, SUNY, Albany; Federated Learning and advanced topics in DLCV		**HW #4 due**
15	12/10	**Progress Check for Final Projects**		NeurIPS week
TBD 17	12/25 Wed	**Final Project Presentation**		

What to Cover Today…

- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
  - Self-Supervised Learning for CNN
- Image Segmentation

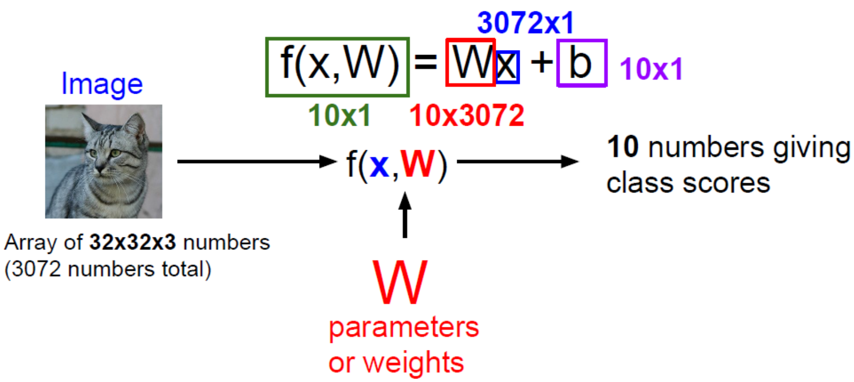


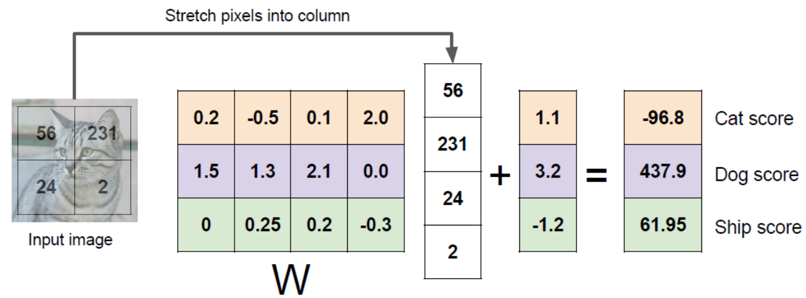




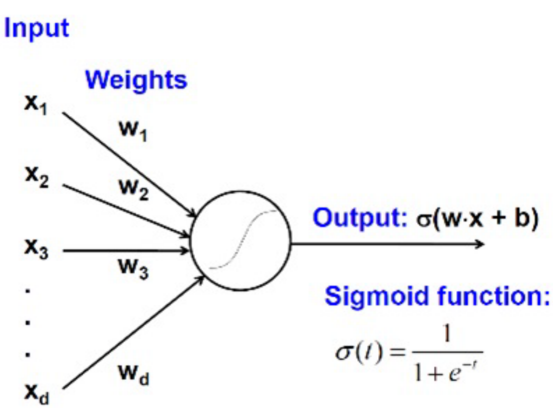
Recap: From Linear Classifiers to Neural Nets

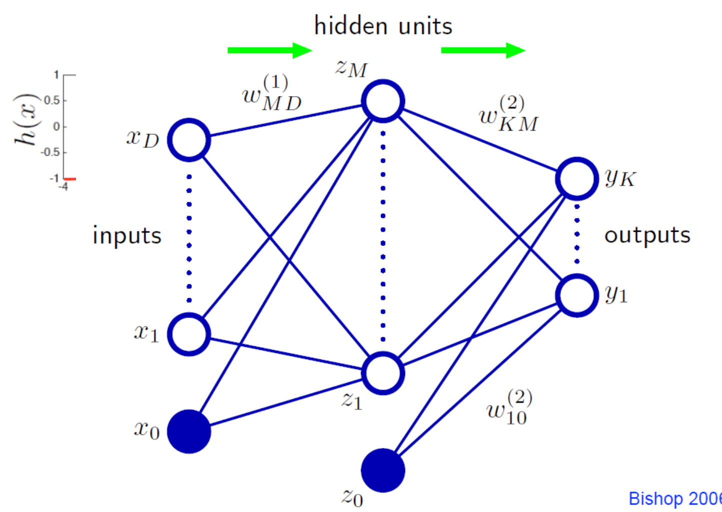
- Linear Classifier





- Neural Network (Multilayer Perceptron)





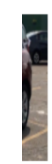
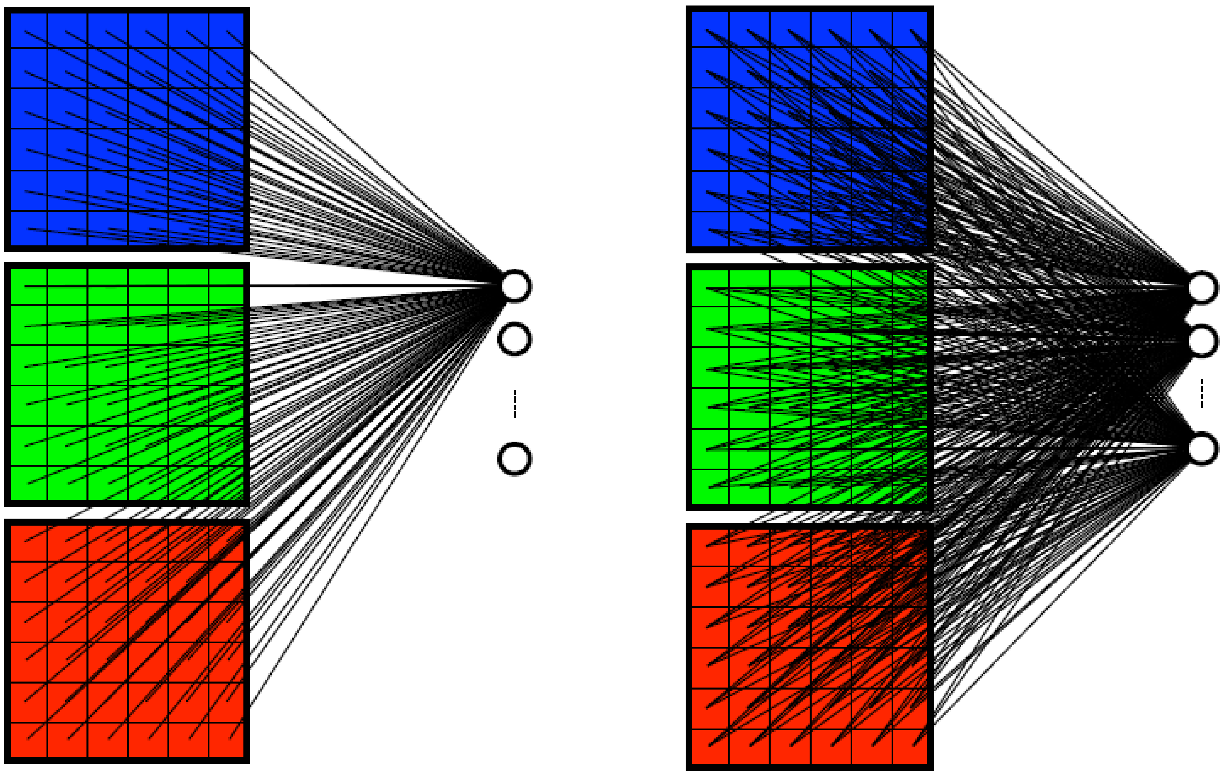


Image credit: Stanford CS231n

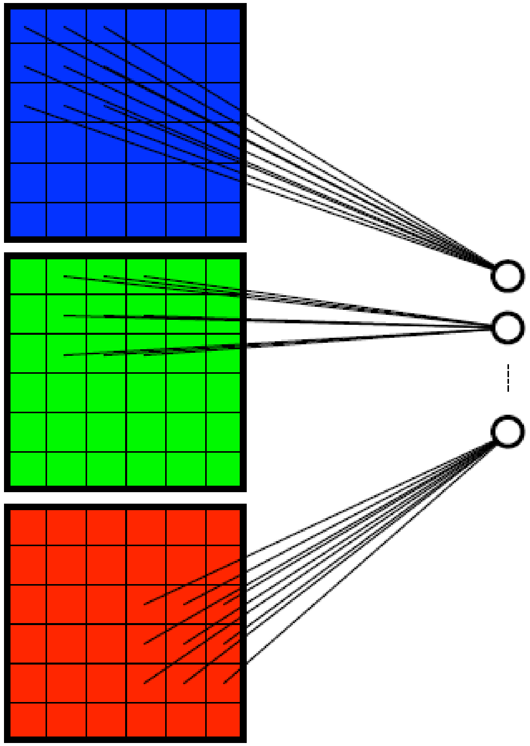
Convolutional Neural Networks

- How many weights for MLPs for images?



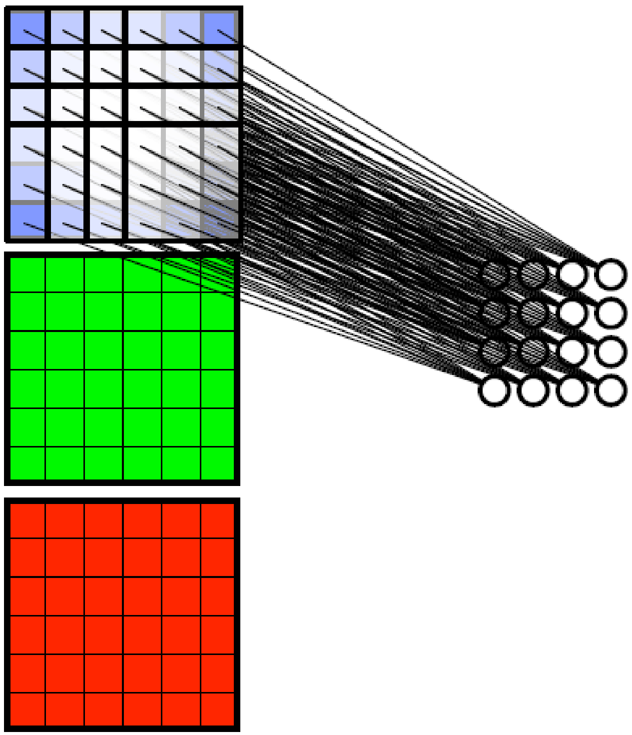
Convolutional Neural Networks

- Property I of CNN: Local Connectivity
  - Each neuron takes info only from a neighborhood of pixels.

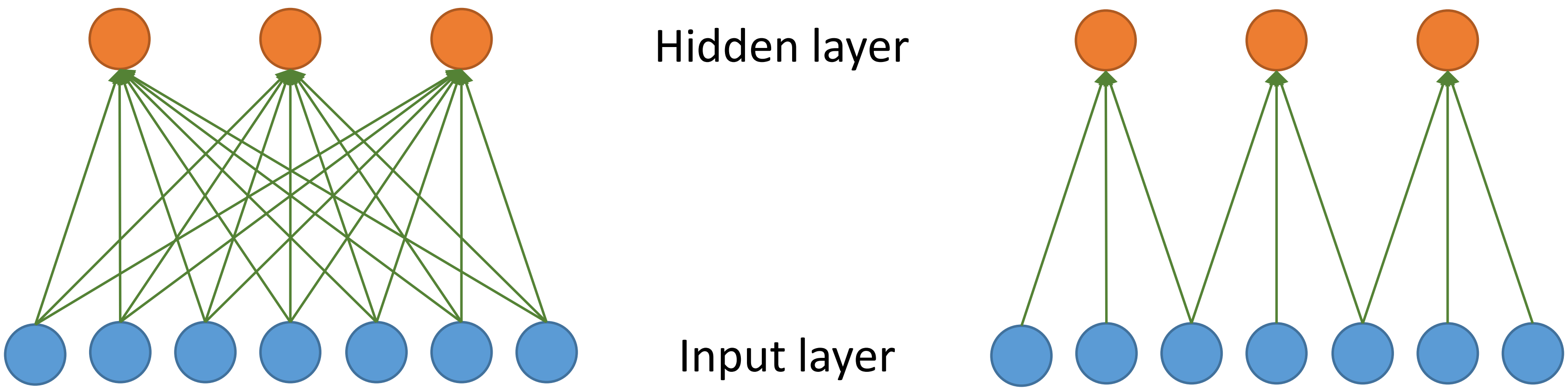


Convolutional Neural Networks

- Property II of CNN: Weight Sharing
  - Neurons connecting each pixel and its neighborhoods have identical weights.



CNN: Local Connectivity

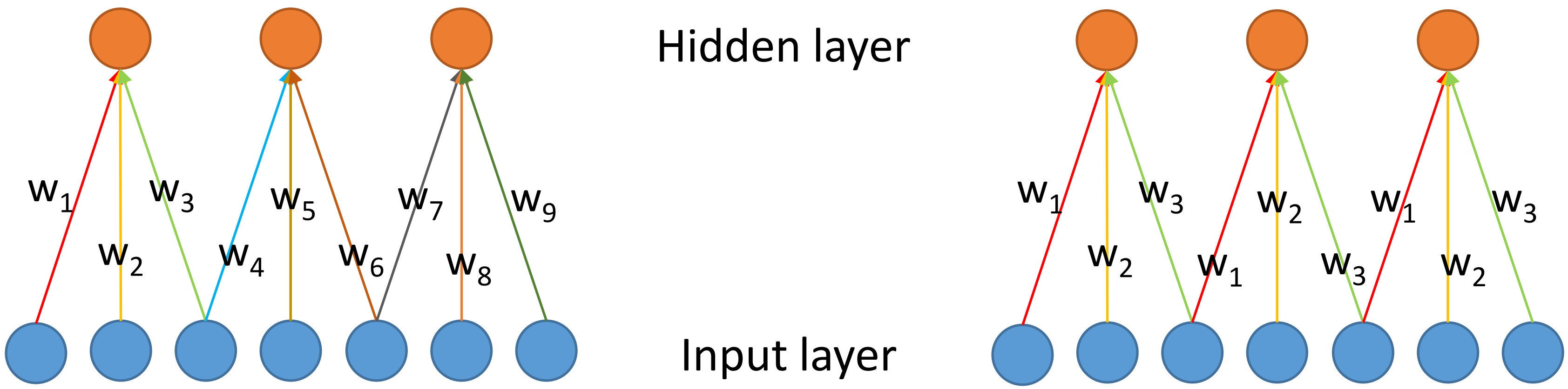


Global connectivity

Local connectivity

- # of input dimensions/units (neurons): 7
- # of output/hidden units: 3
- Number of parameters
    - Global connectivity:
    - Local connectivity:

CNN: Weight Sharing

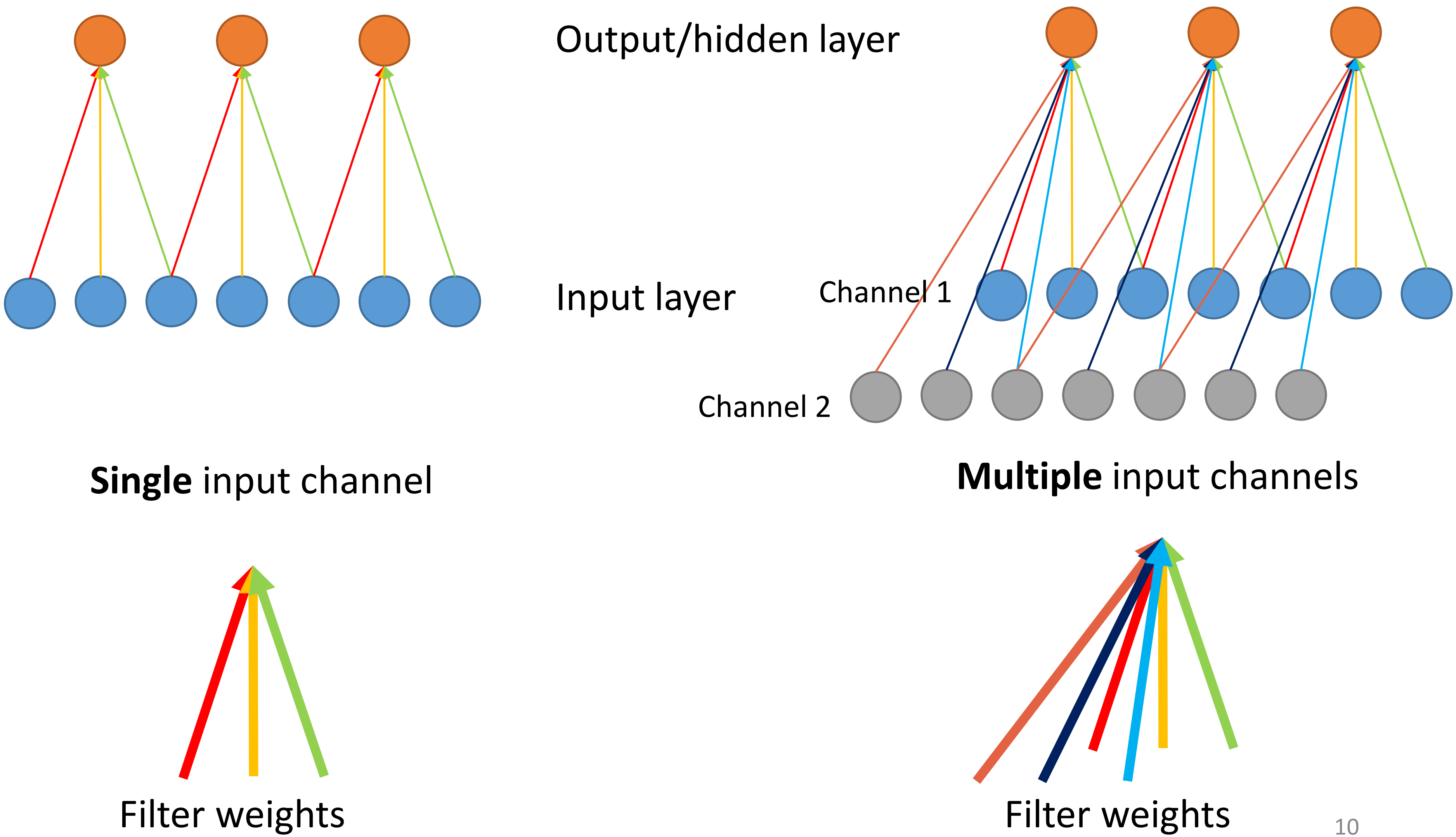


Without weight sharing

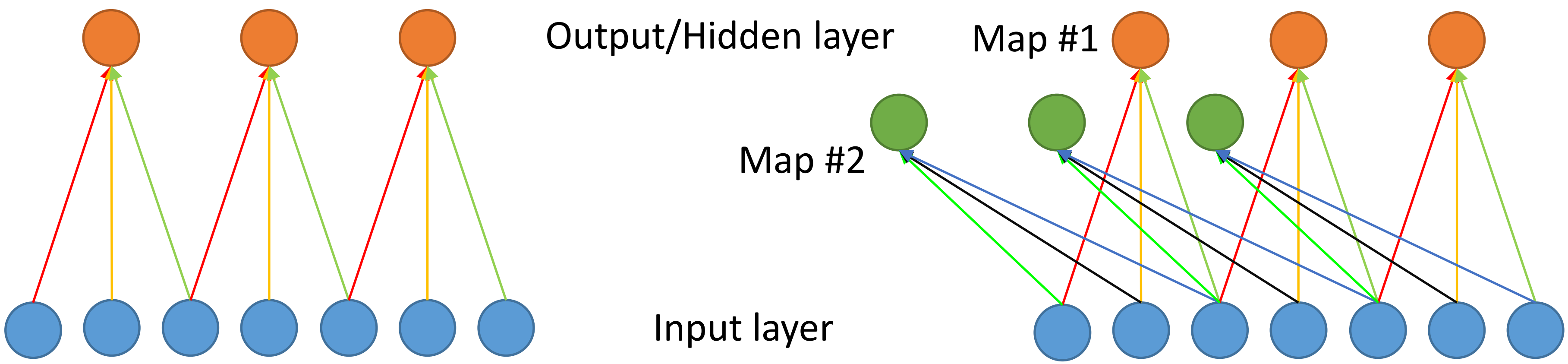
With weight sharing

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

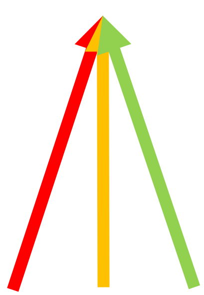
CNN with Multiple Input Channels



CNN with Multiple Output Maps

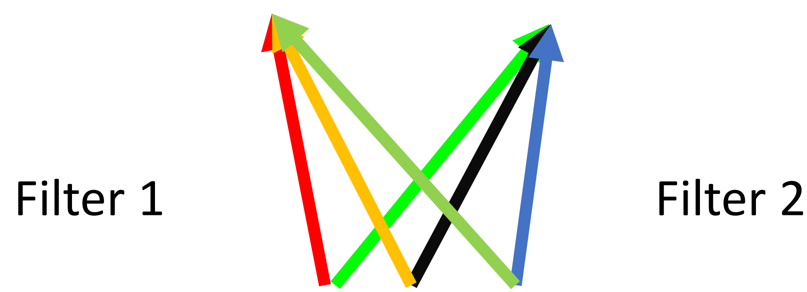


Single output map



Filter weights

Multiple output maps



Filter weights

Putting the ideas together → CNN

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

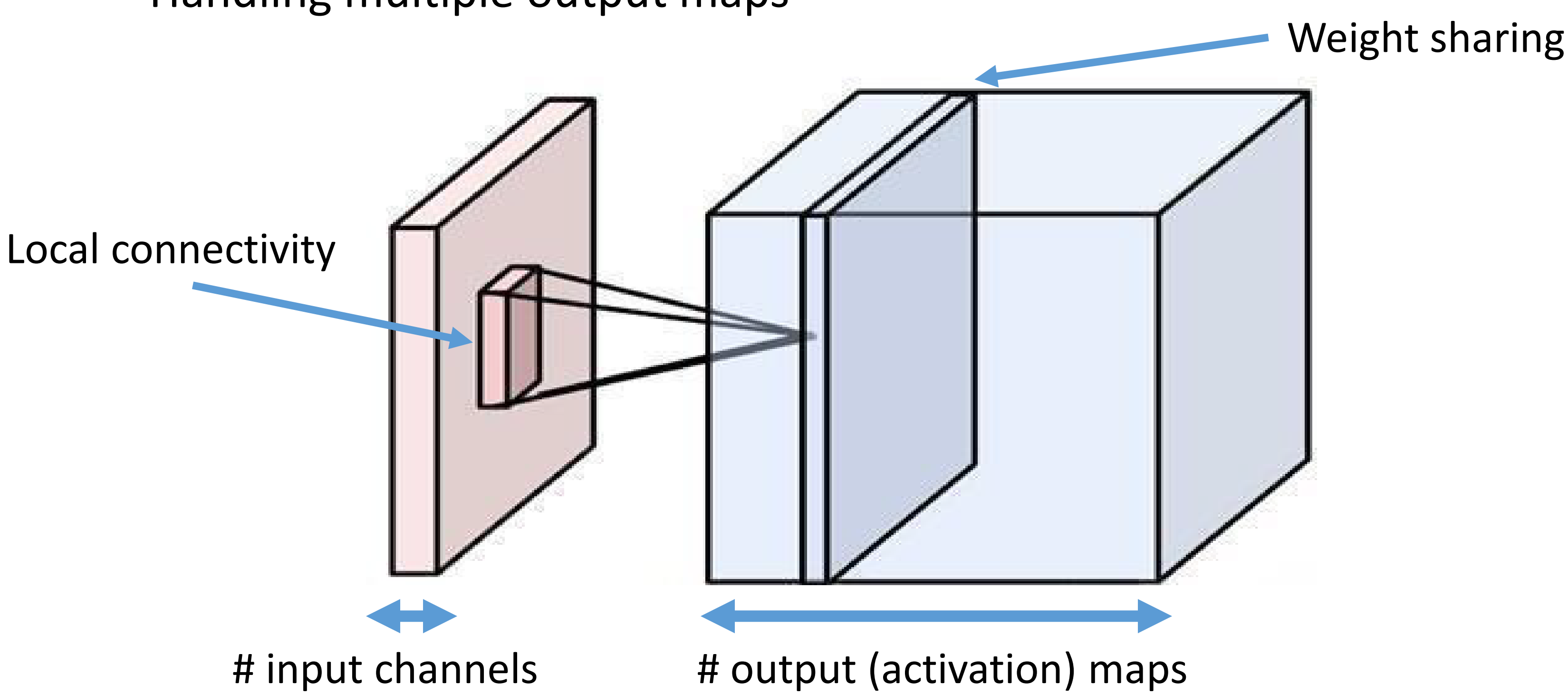
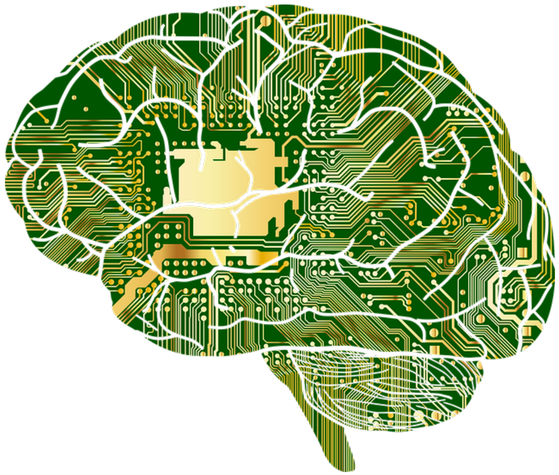
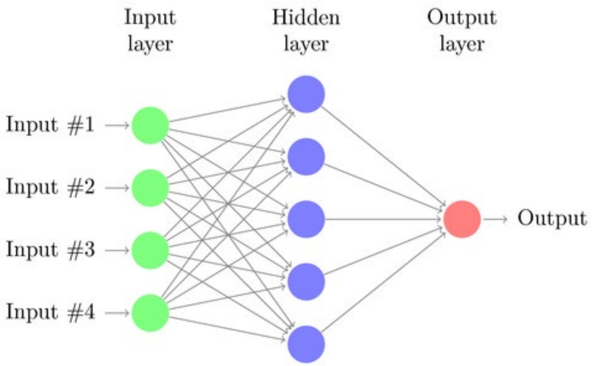


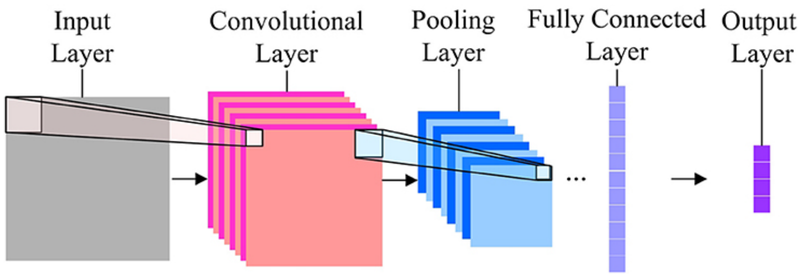
Image credit: A. Karpathy

What's to Be Covered Today…

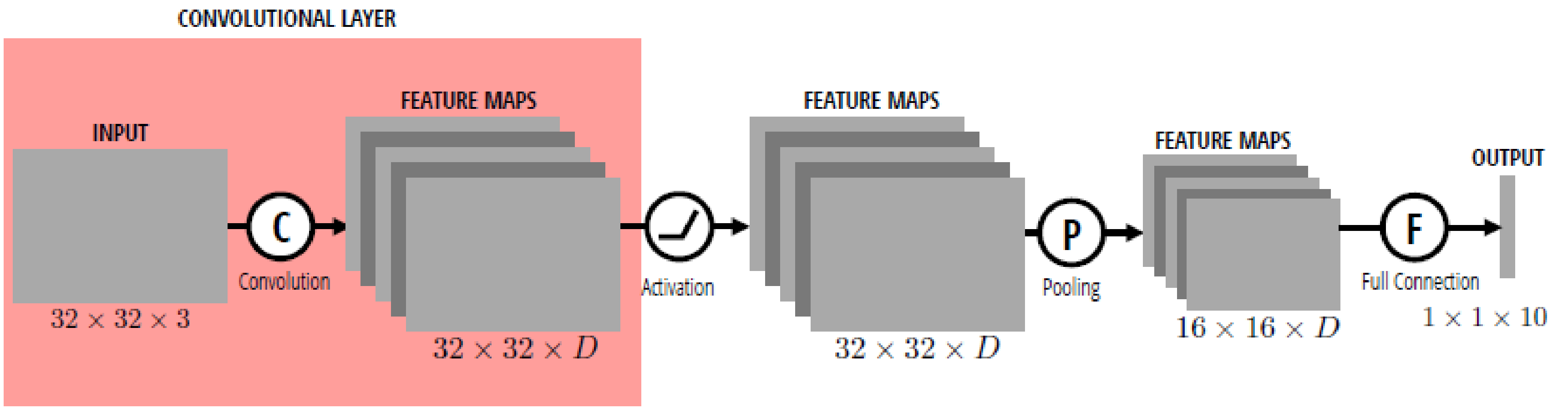
- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN







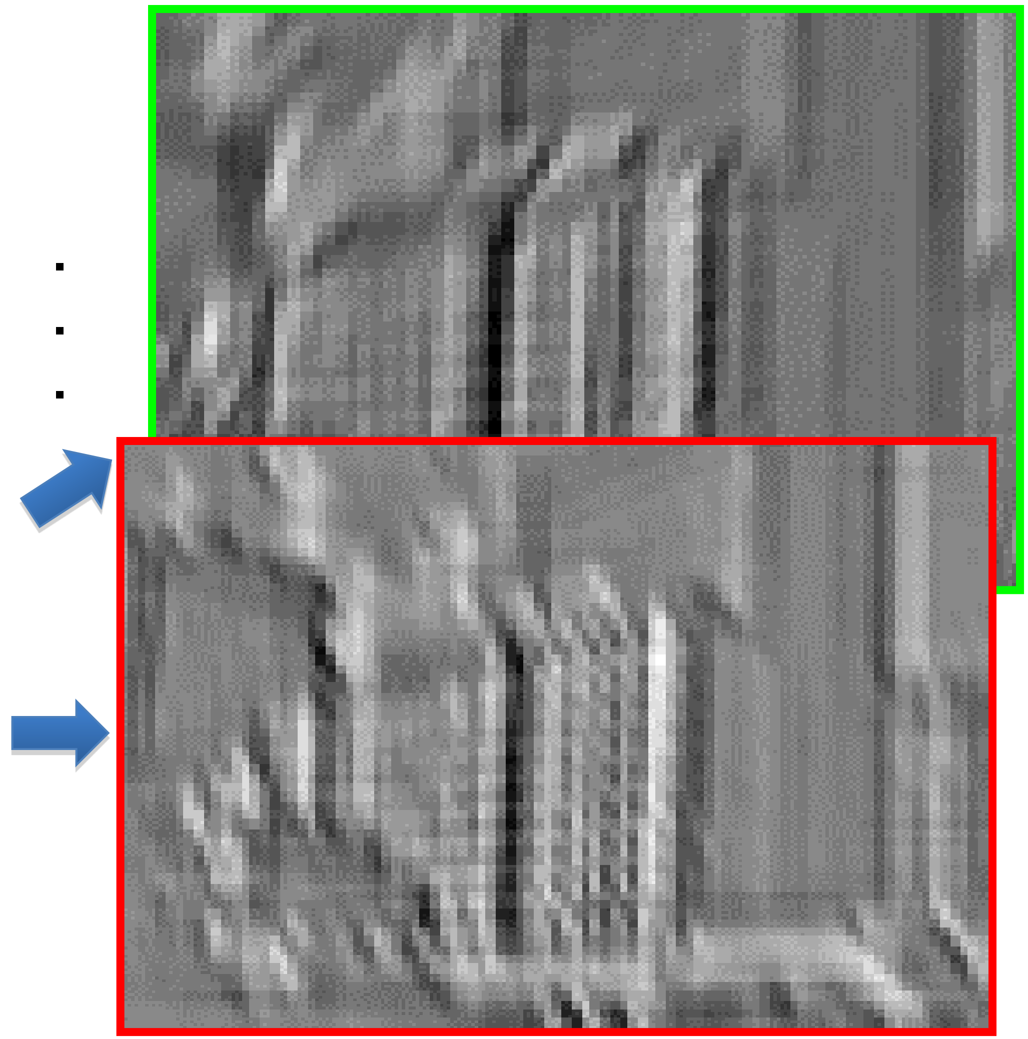
Convolution Layer in CNN



What is a Convolution?

- Weighted moving sum

Feature Activation Map #2

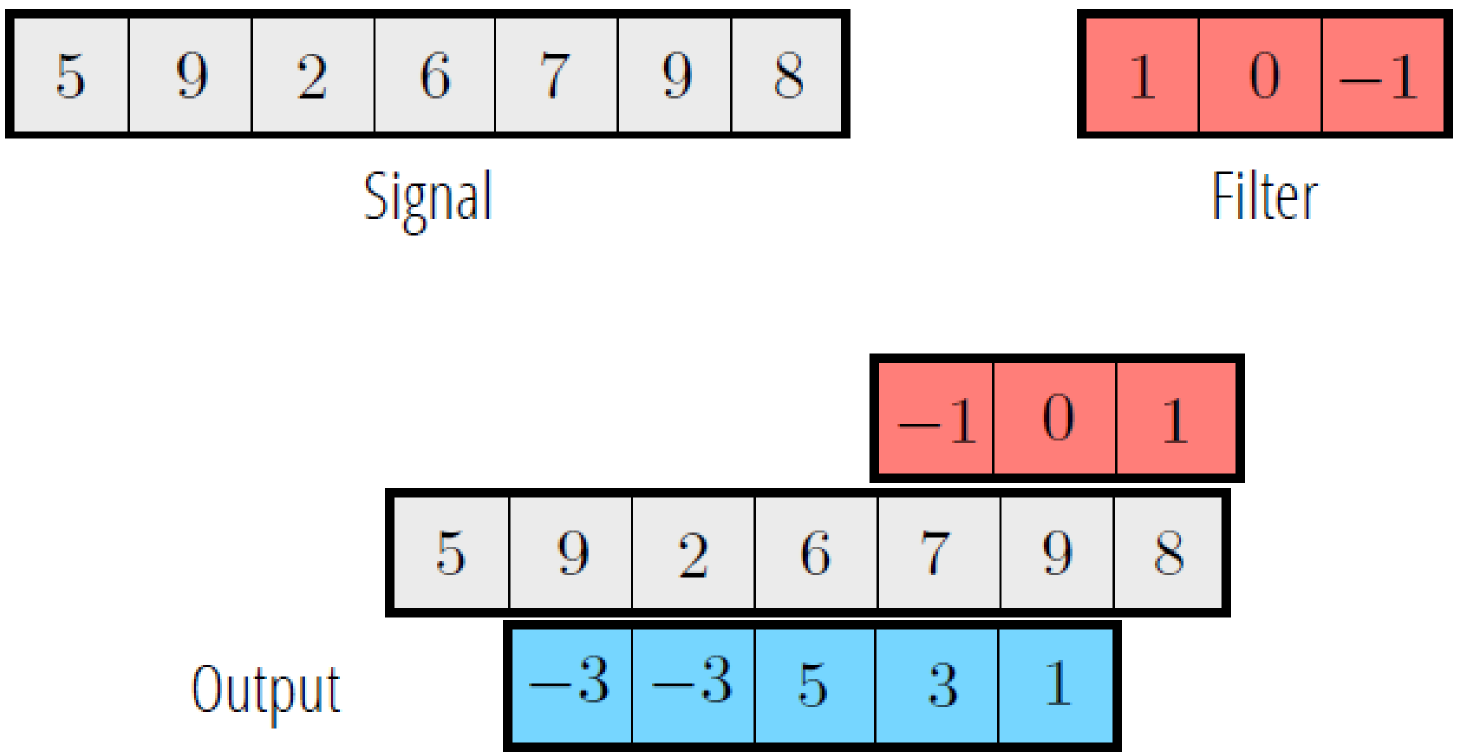


Input

Feature Activation Map #1

slide credit: S. Lazebnik

What is a Convolution?



Convolution is a local linear operator

What is a Convolution?

- Toeplitz Matrix Form

$$
\begin{bmatrix}
\ddots & & & & & \\
& w_1 & w_2 & w_3 & & \\
& & w_1 & w_2 & w_3 & \\
& & & w_1 & w_2 & w_3 \\
& & & & & \ddots
\end{bmatrix}
\begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{MN} \end{bmatrix}
+
\begin{bmatrix} b \\ b \\ \vdots \\ b \end{bmatrix}
=
\begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{MN} \end{bmatrix}
$$

Filter

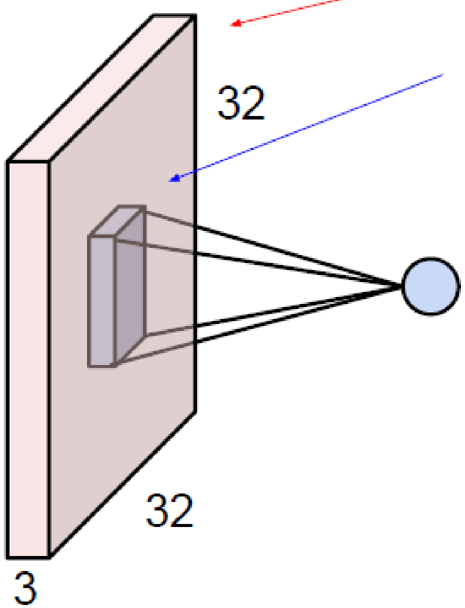
Bias

Putting them together (cont'd)

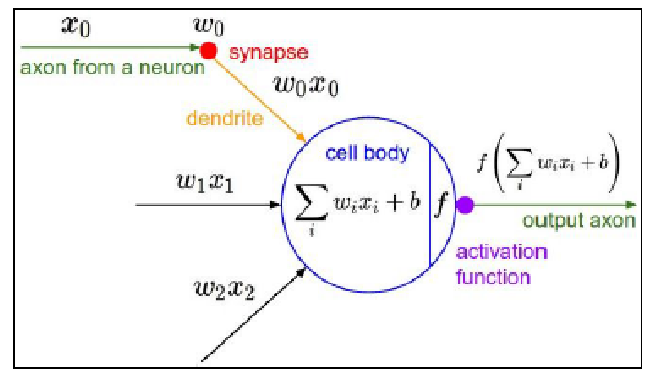
- The neuron view of a CONV layer

32x32x3 image

5x5x3 filter



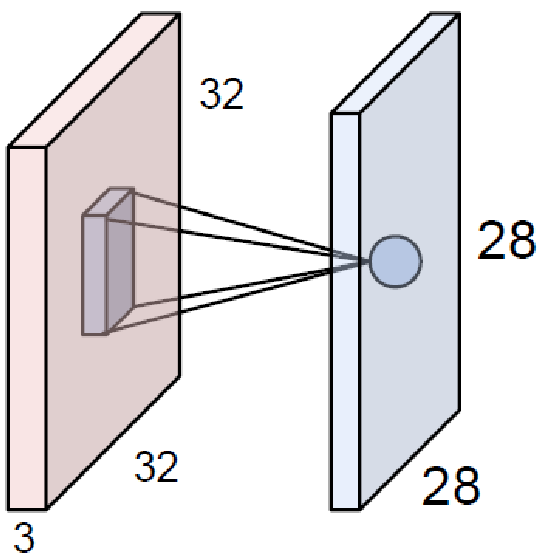
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



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

Putting them together (cont'd)

- The neuron view of CONV layer (cont'd)



➢ **Feature map** (at an intermediate layer):

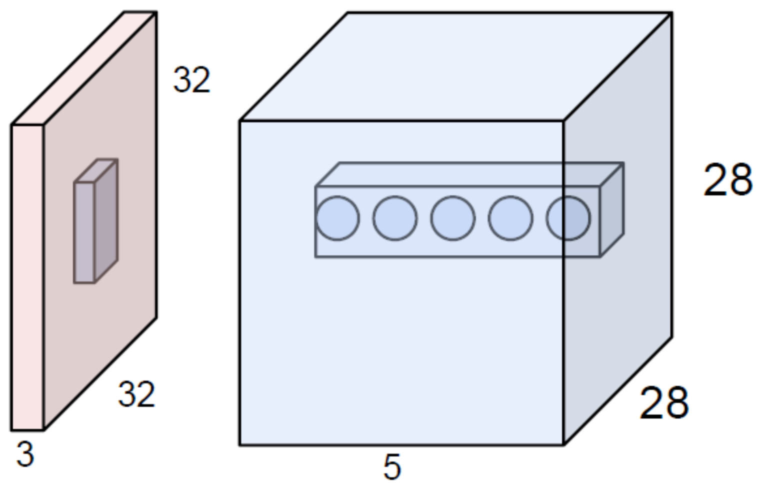
An activation map is a 28x28 sheet of neuron outputs:

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 neuron view of CONV layer
  - Typically, more than 1 filter is learned in CNN…

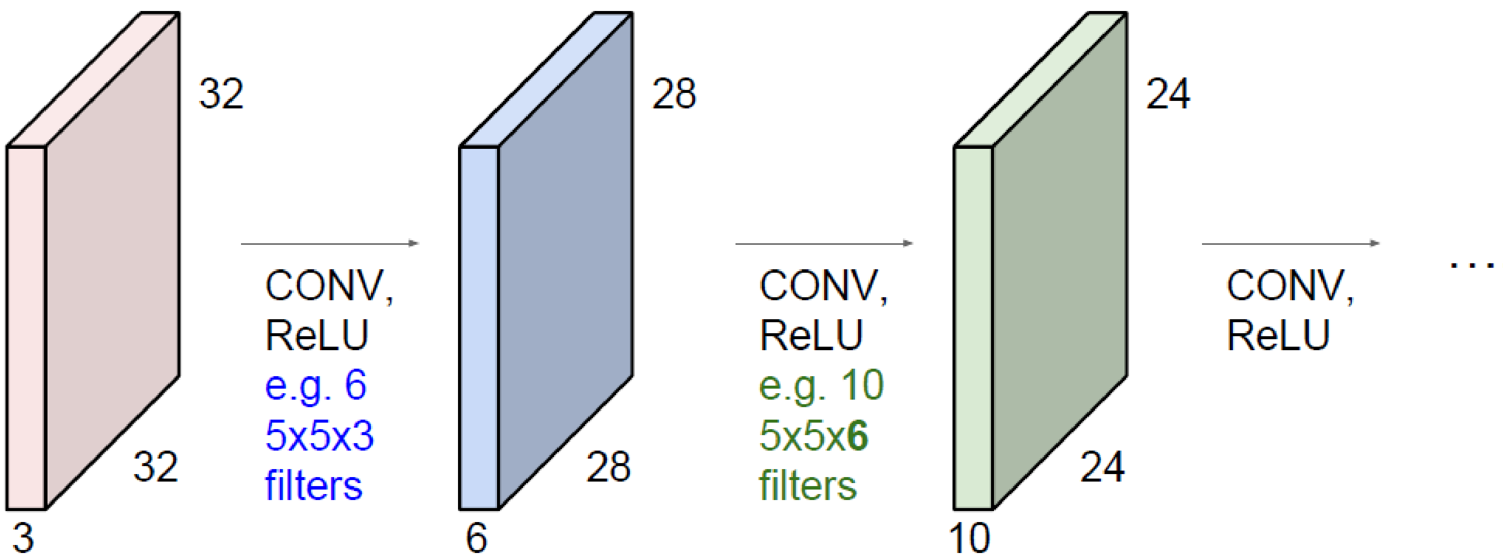


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

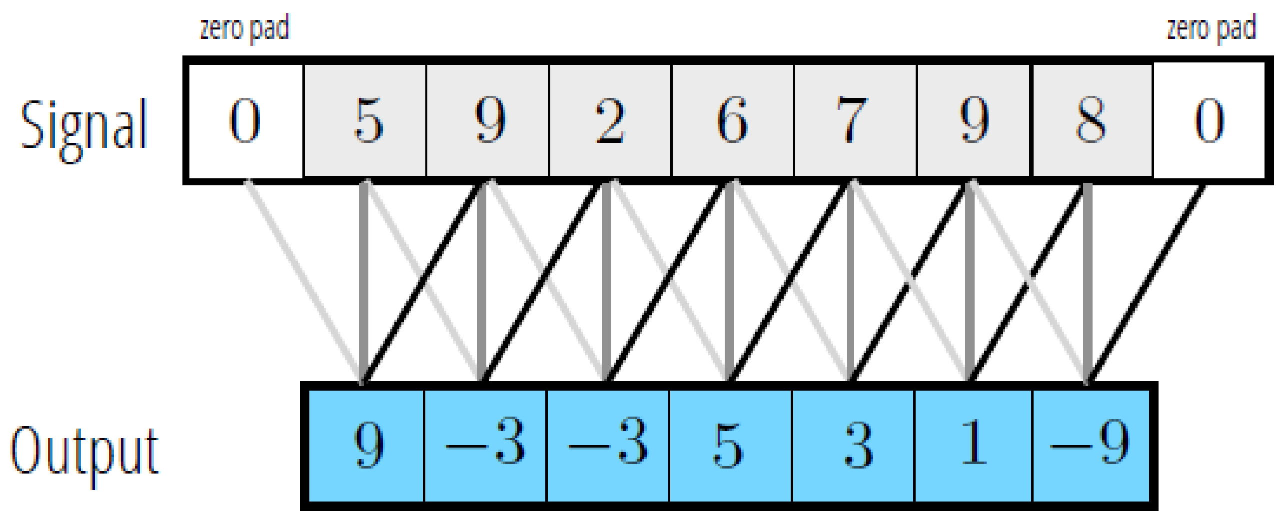
Putting them together (cont'd)

- Image input with 32 x 32 pixels convolved repeatedly with 5 x 5 x 3 filters would shrink feature mape volumes spatially
  - 32 -> 28 -> 24 -> …



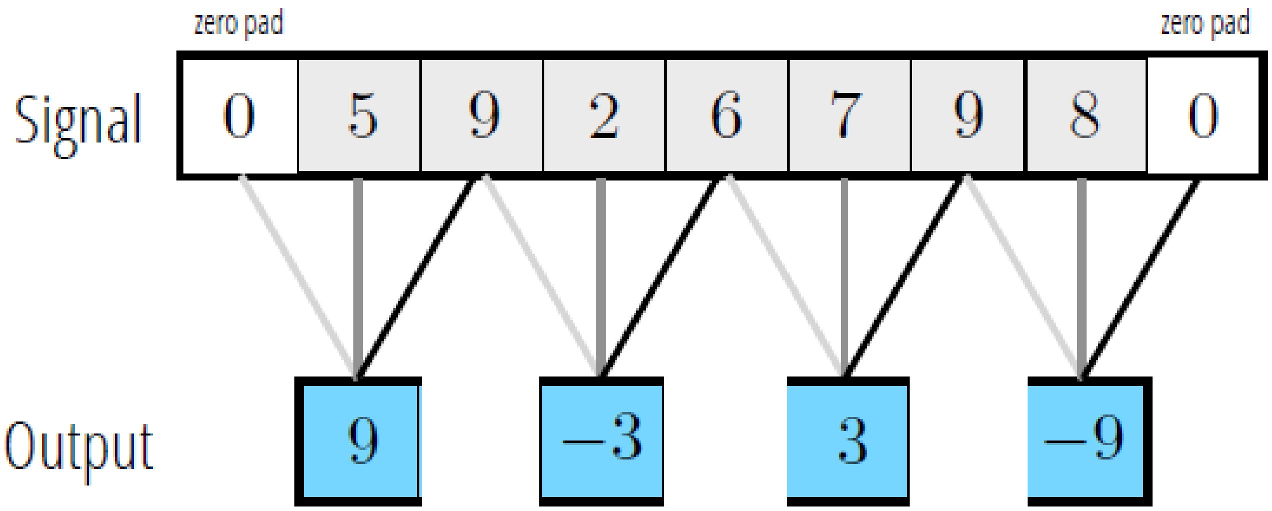
What is a Convolution? (cont'd)

- Zero Padding
  - Output is the same size as that of the input
    - That is, conv will not shrink as the network gets deeper.



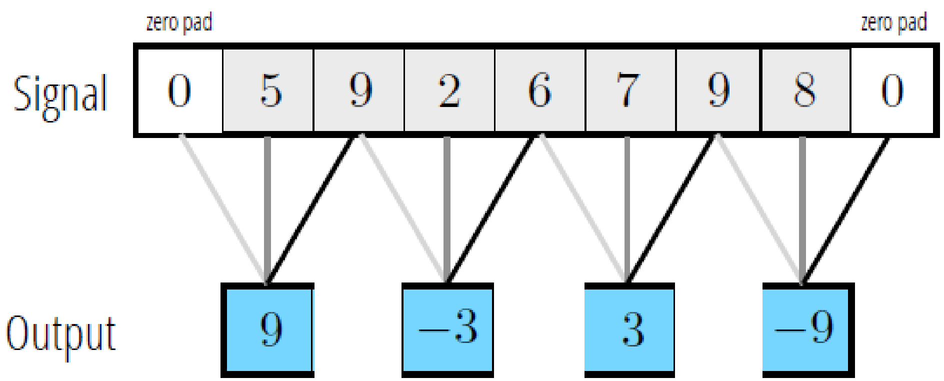
What is a Convolution? (cont'd)

- Stride
  - Step size across signals
  - Why & when preferable?



What is a Convolution? (cont'd)

- Stride
  - Step size across signals
  - See example below:



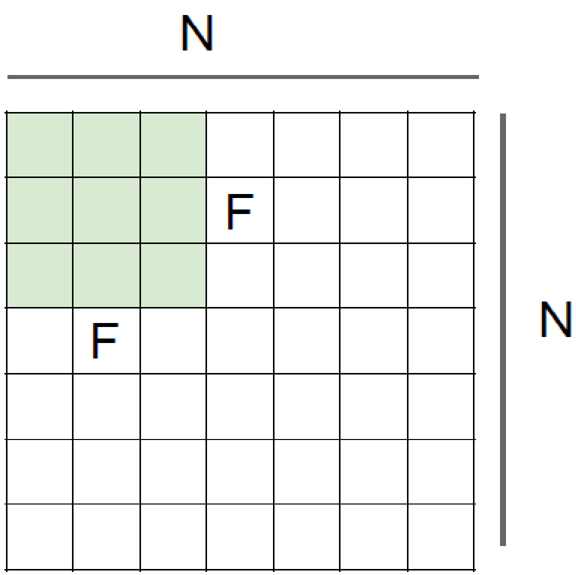
Input Size **Filter Size**

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

Stride: step size across the signal

What is a Convolution? (cont'd)

- Stride
  - Step size across signals
  - See a 2D example below:

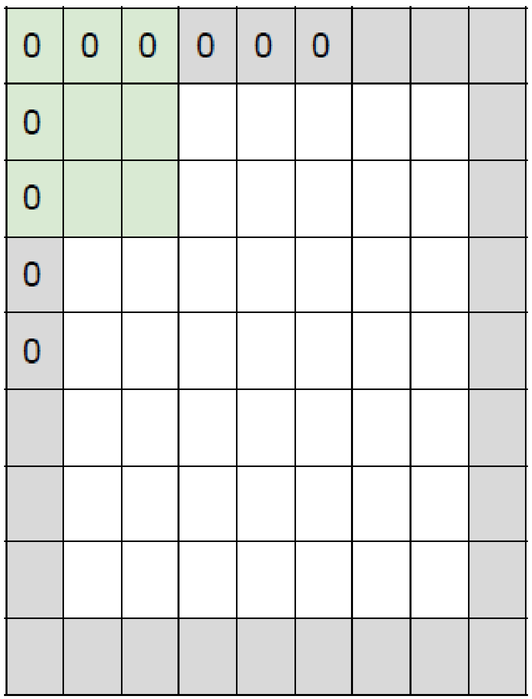


Output size:
(N - F) / stride + 1

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 :\

What is a Convolution?

- Zero Padding + Stride

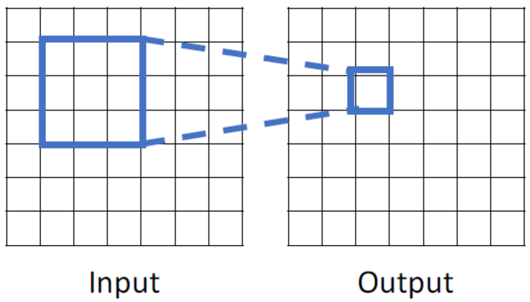


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

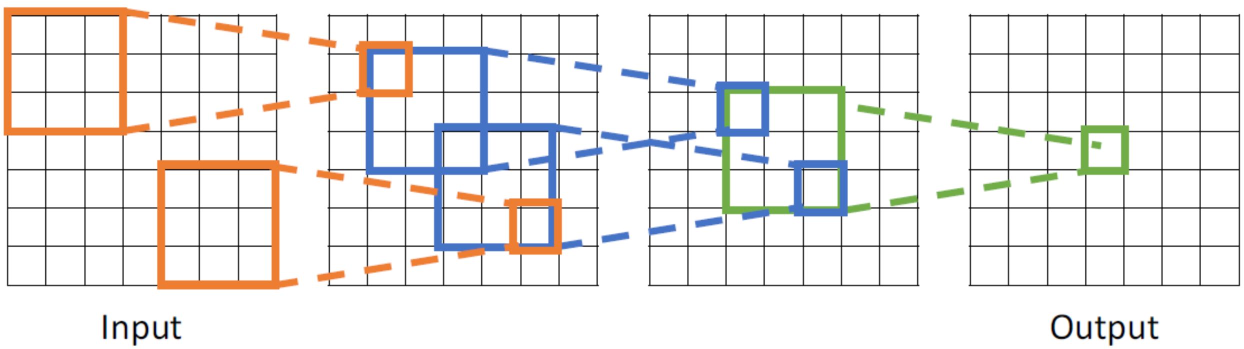
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remarks: Receptive Field

- For convolution with kernel size n x n,
  each entry in the output layer depends on a n x n receptive field in the input layer.



- Each successive convolution adds n-1 to the receptive field size.
  With a total of L layers, the receptive field size would be 1 + (L-1) * (n-1).

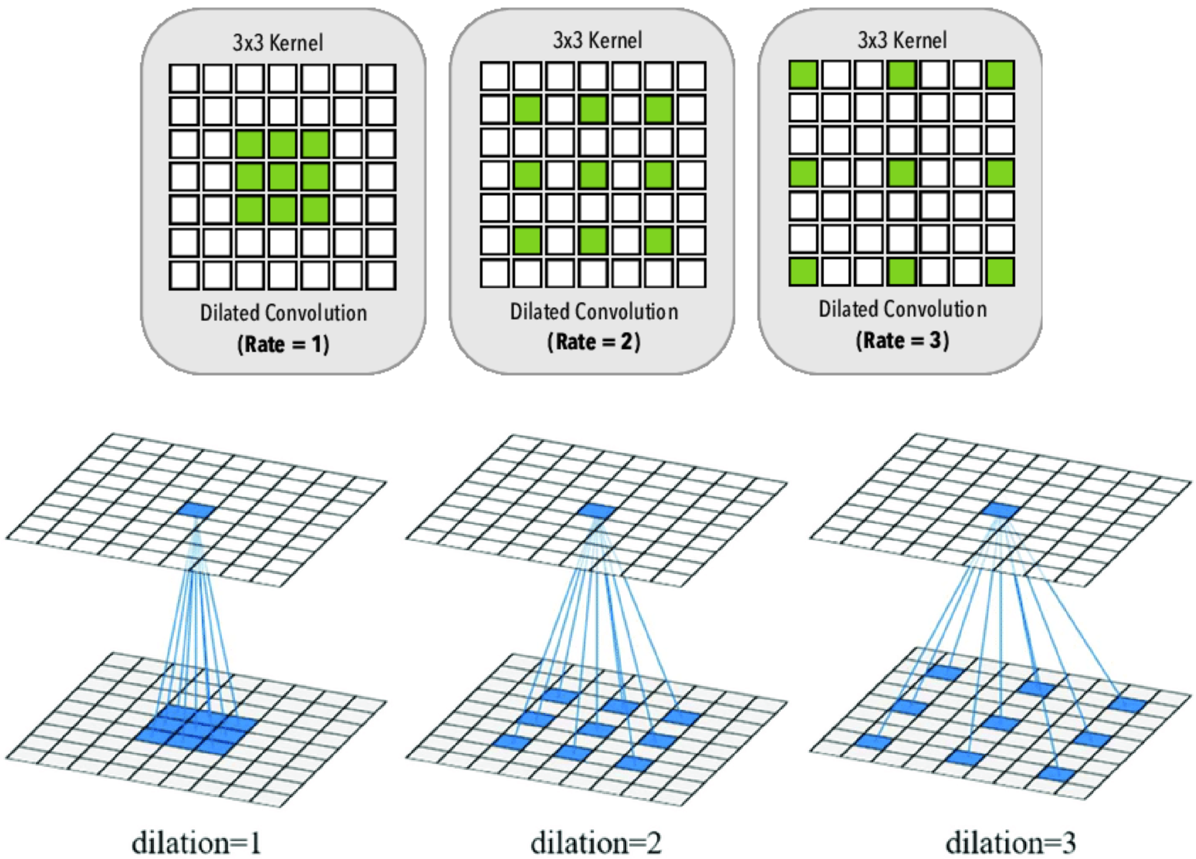


- For an image w/ high resolution, we need to deploy multiple CNN layers
  for the output to "see" the entire input image.
- Other alternatives: downsample the image/feature map (see pooling layer next)

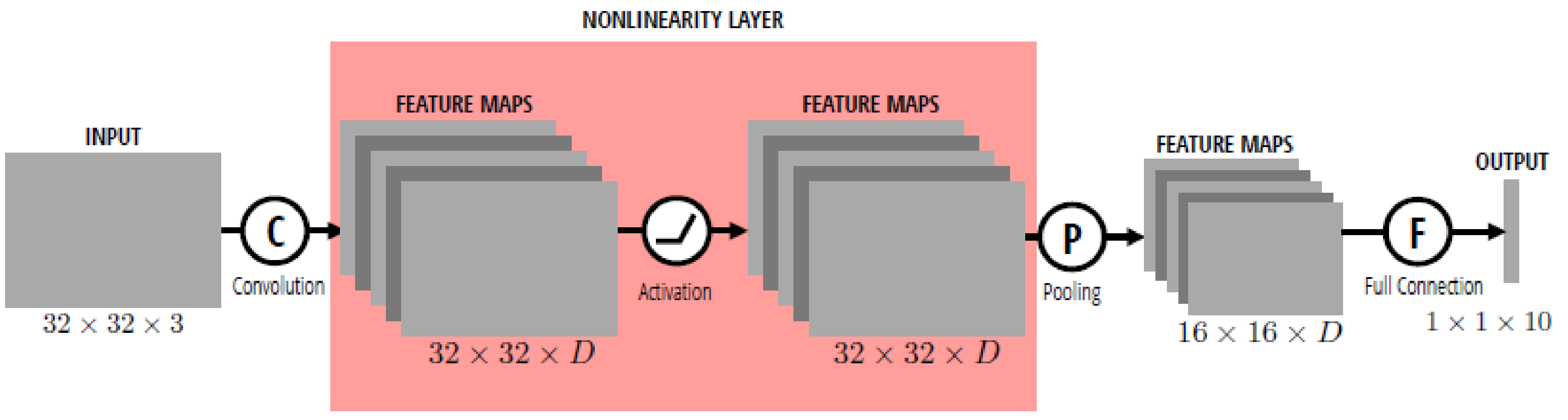
Slide credit: UMich EECS 498-007

A Variant of Convolution

- Dilated Convolution
  - Kernel in the same size but capable of handling a larger receptive field

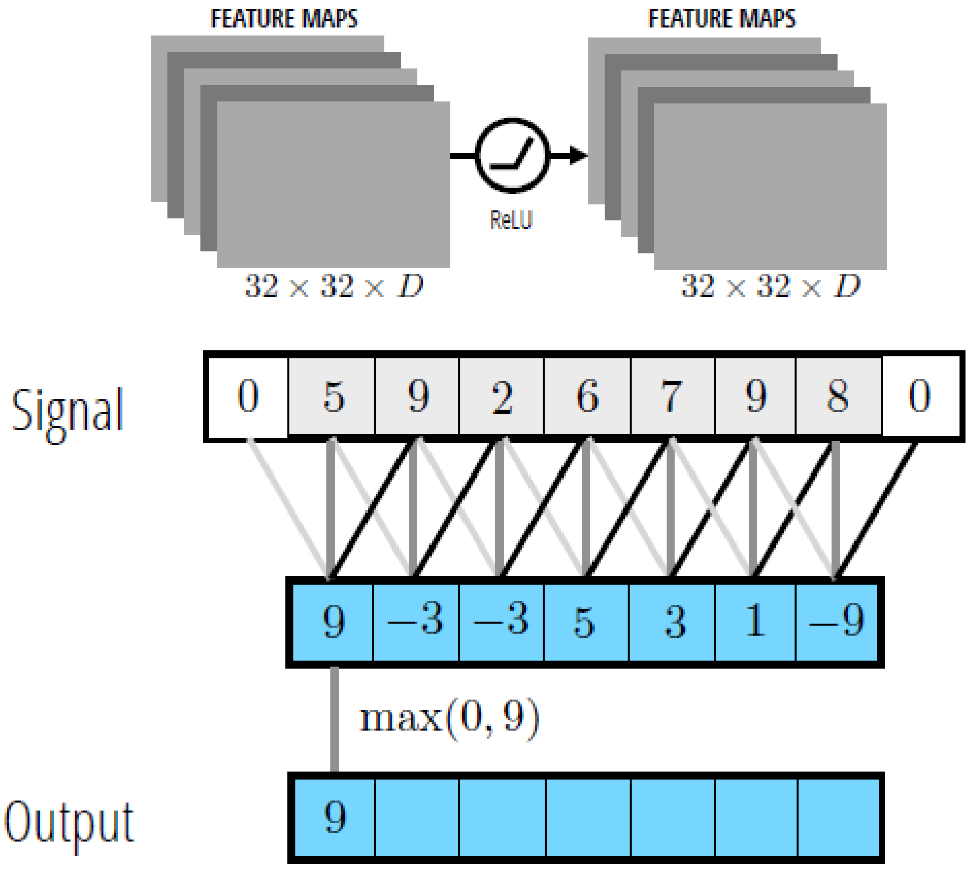


Nonlinearity Layer in CNN



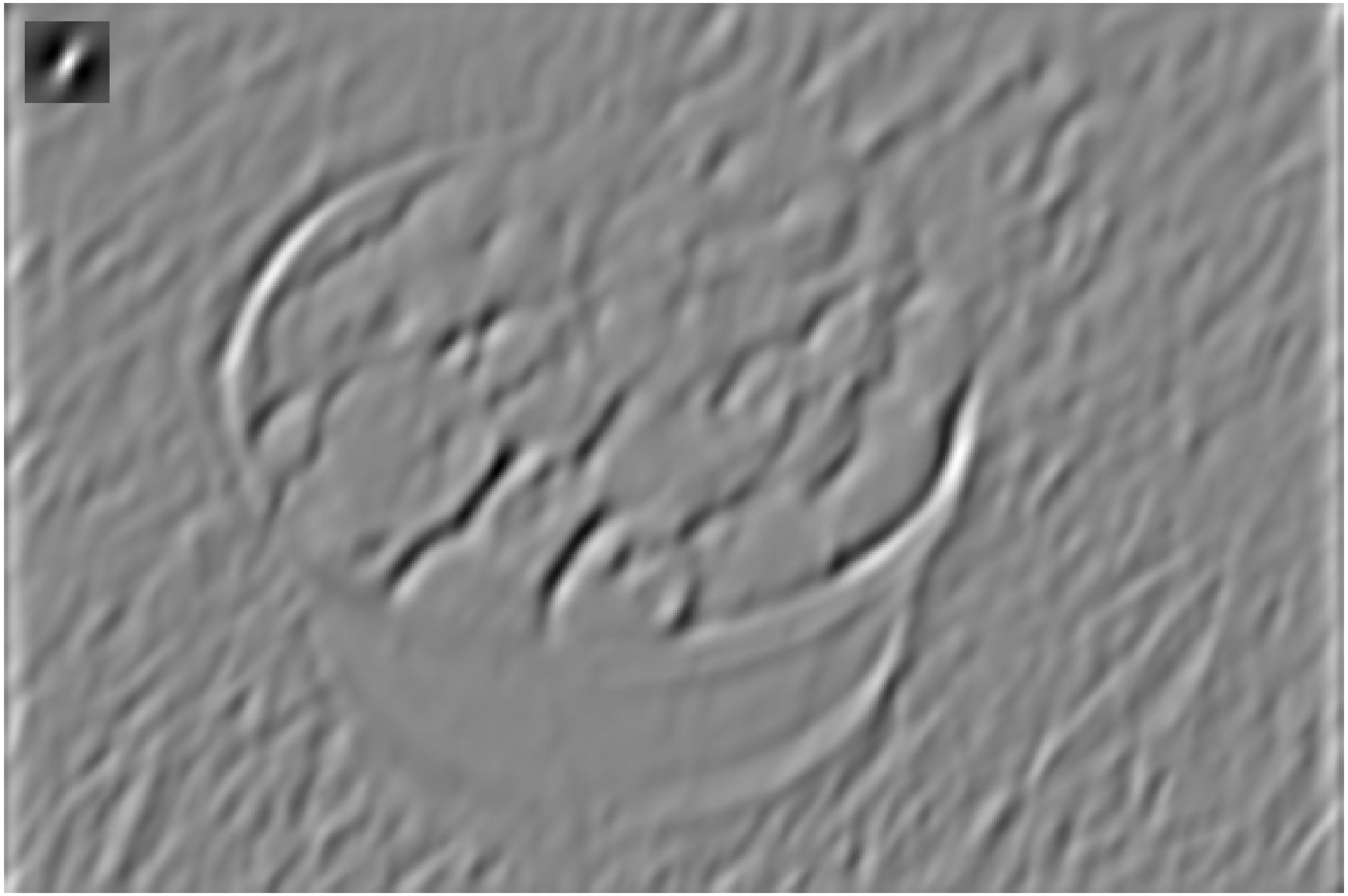
Nonlinearity Layer

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



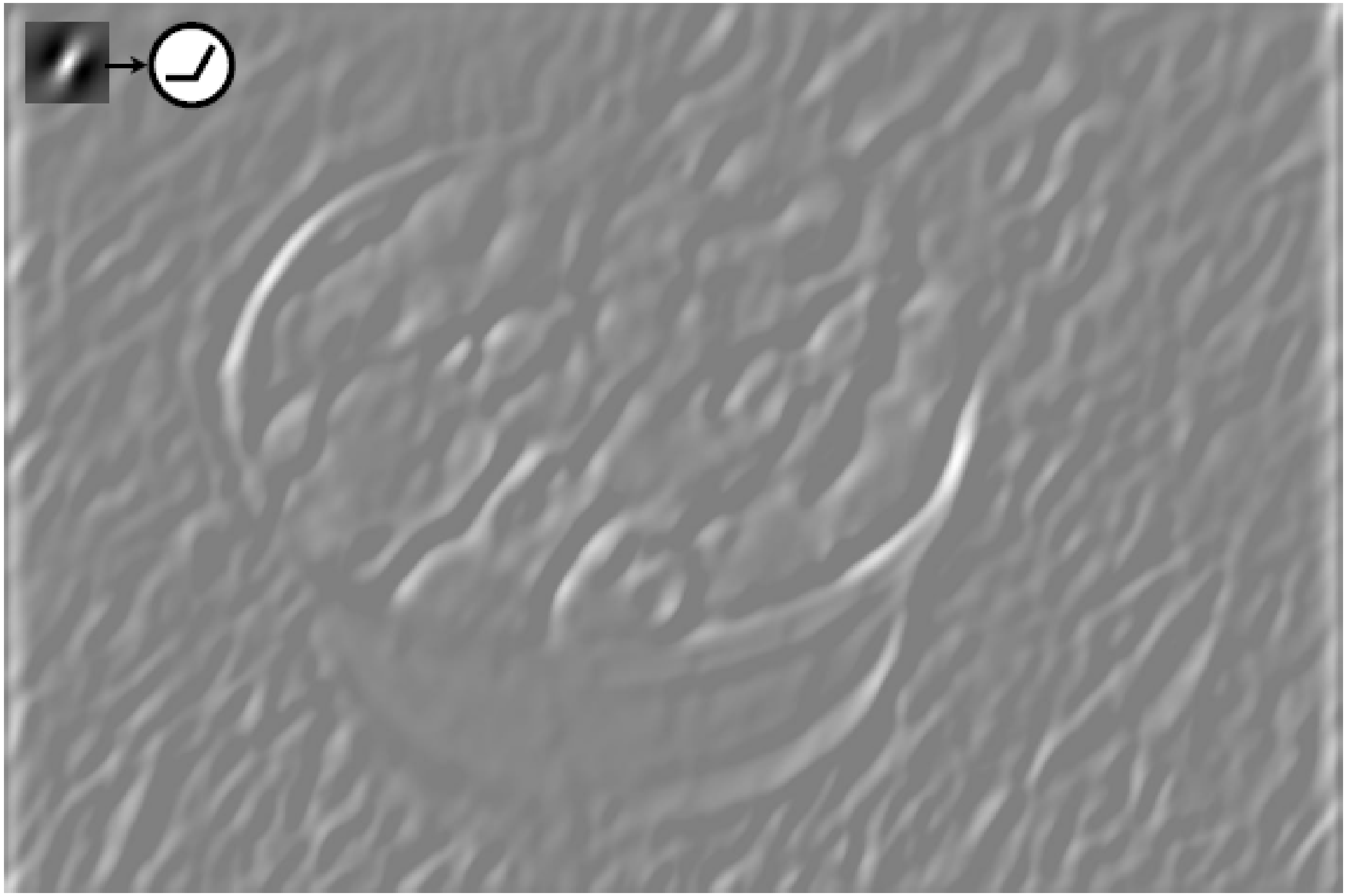
Nonlinearity Layer

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

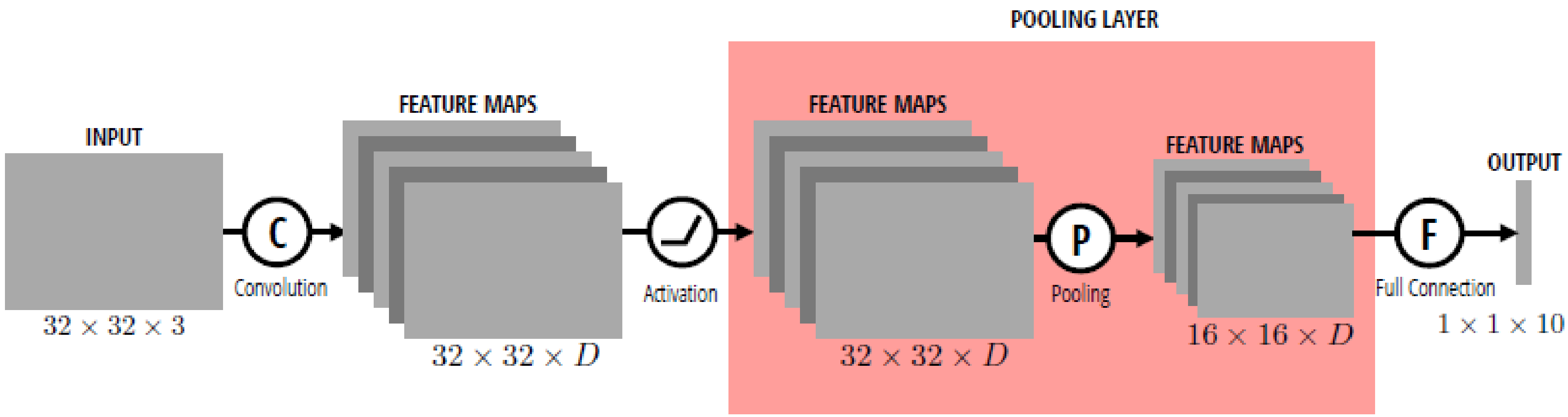


Nonlinearity Layer

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

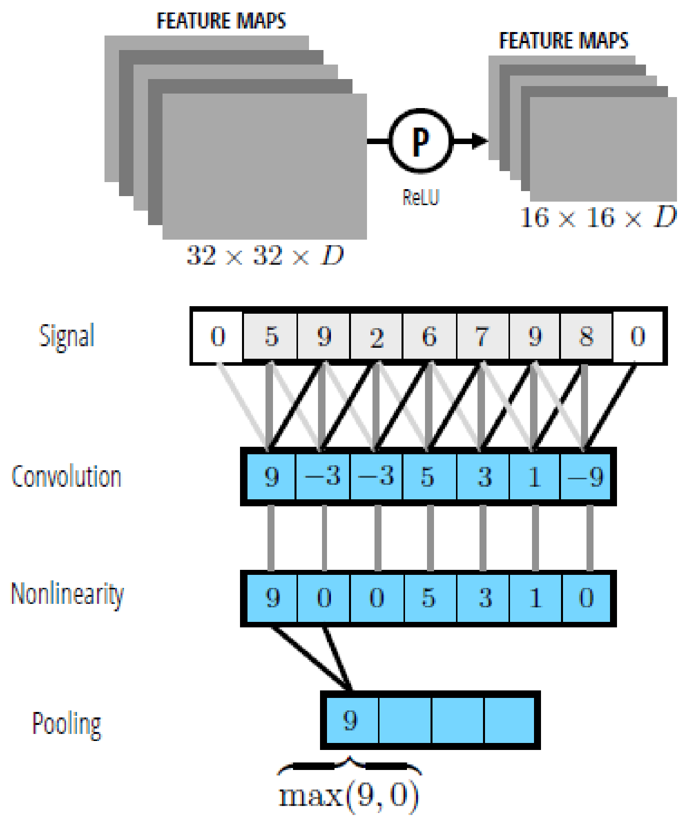


Pooling Layer in CNN



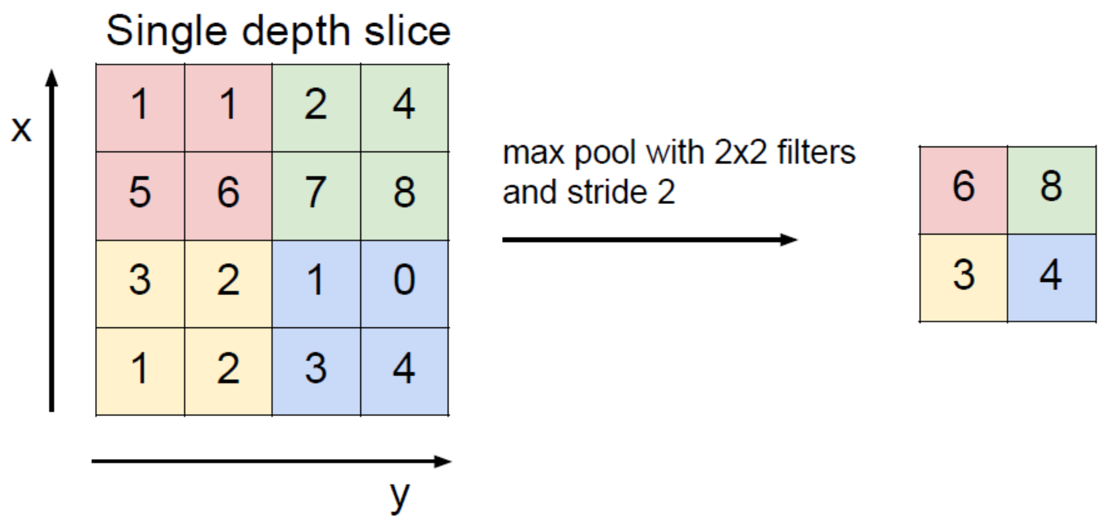
Pooling Layer

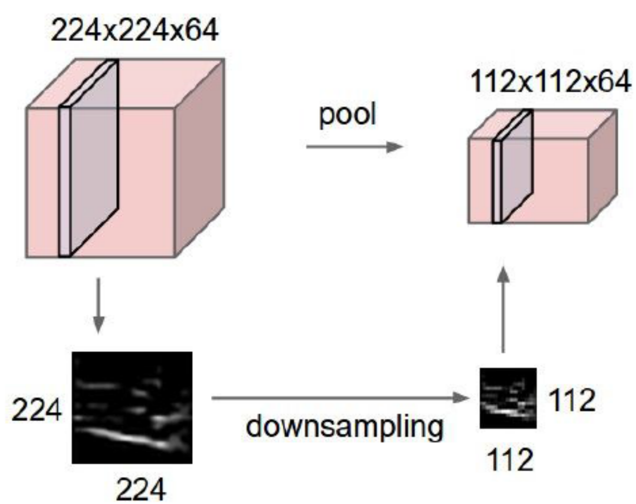
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling



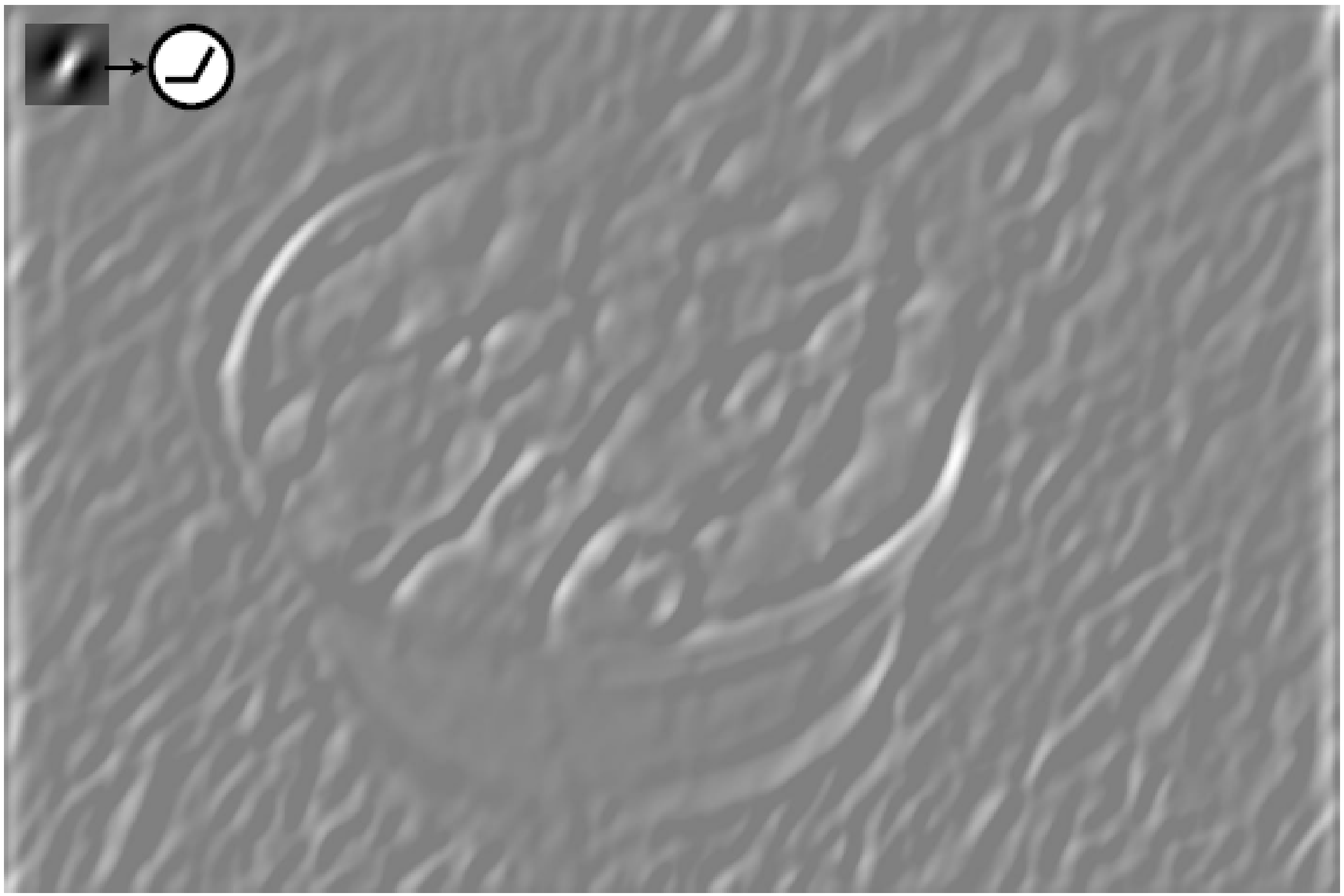
Pooling Layer

- Reduces the spatial size and provides spatial invariance

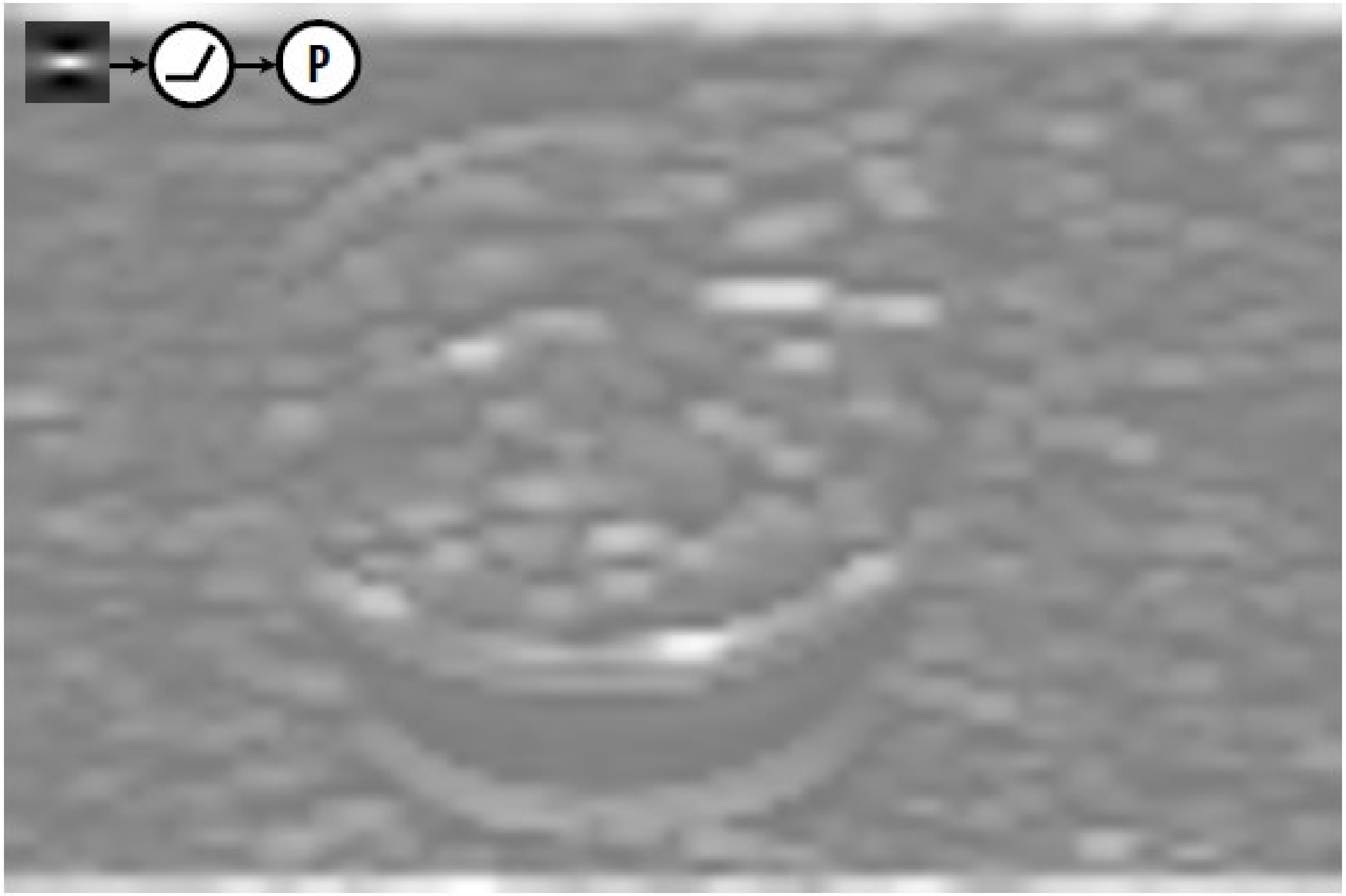




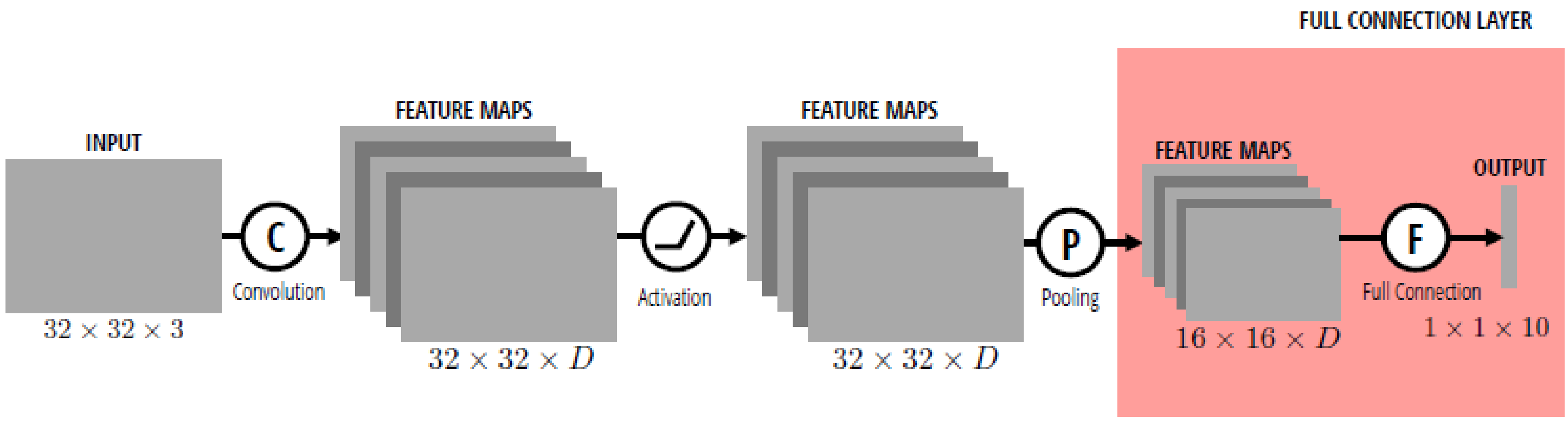
- Example
  - Nonlinearity by ReLU



- Example
  - Max pooling

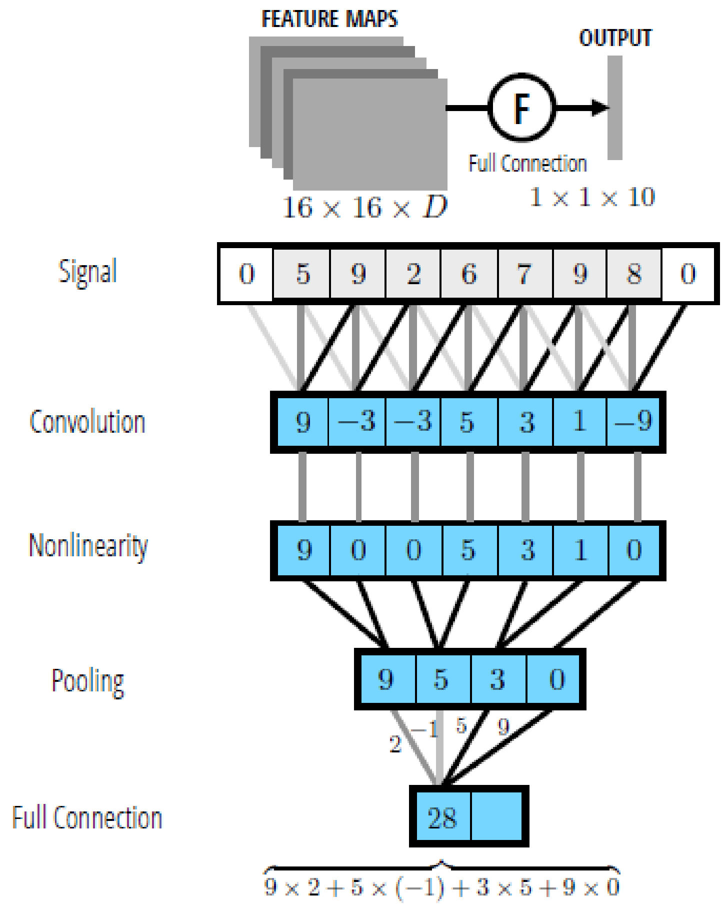


Fully Connected (FC) Layer in CNN



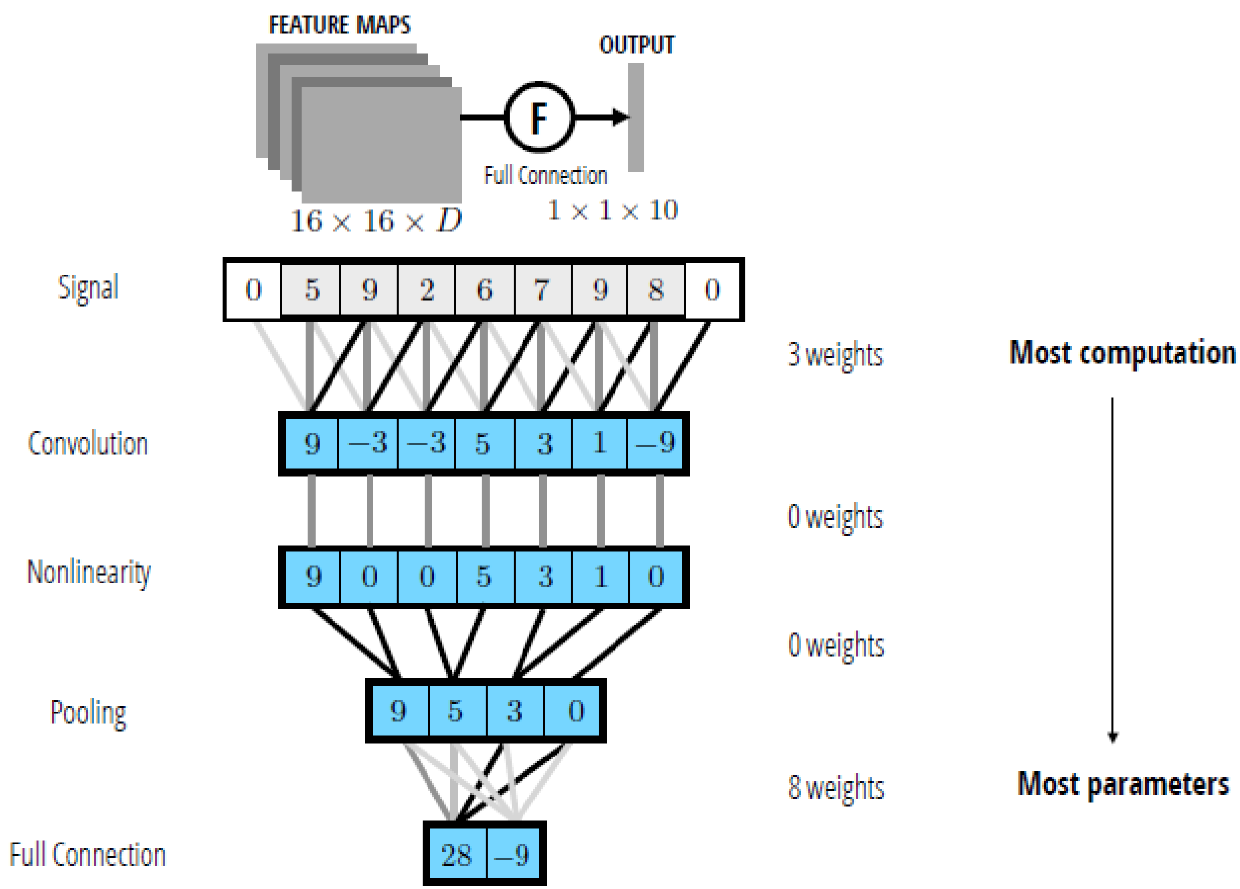
FC Layer

- Mapping features/neurons that connect to the entire input volume to the desirable output (e.g., predicted scores for each class)

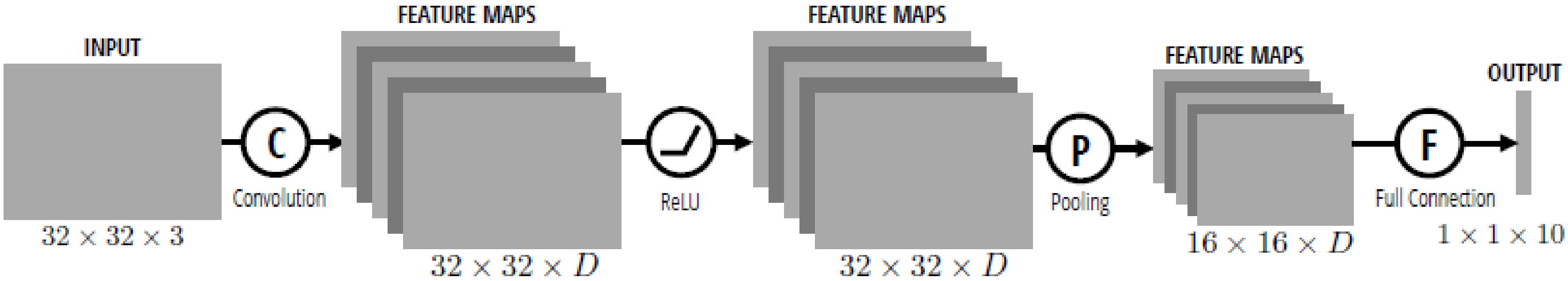


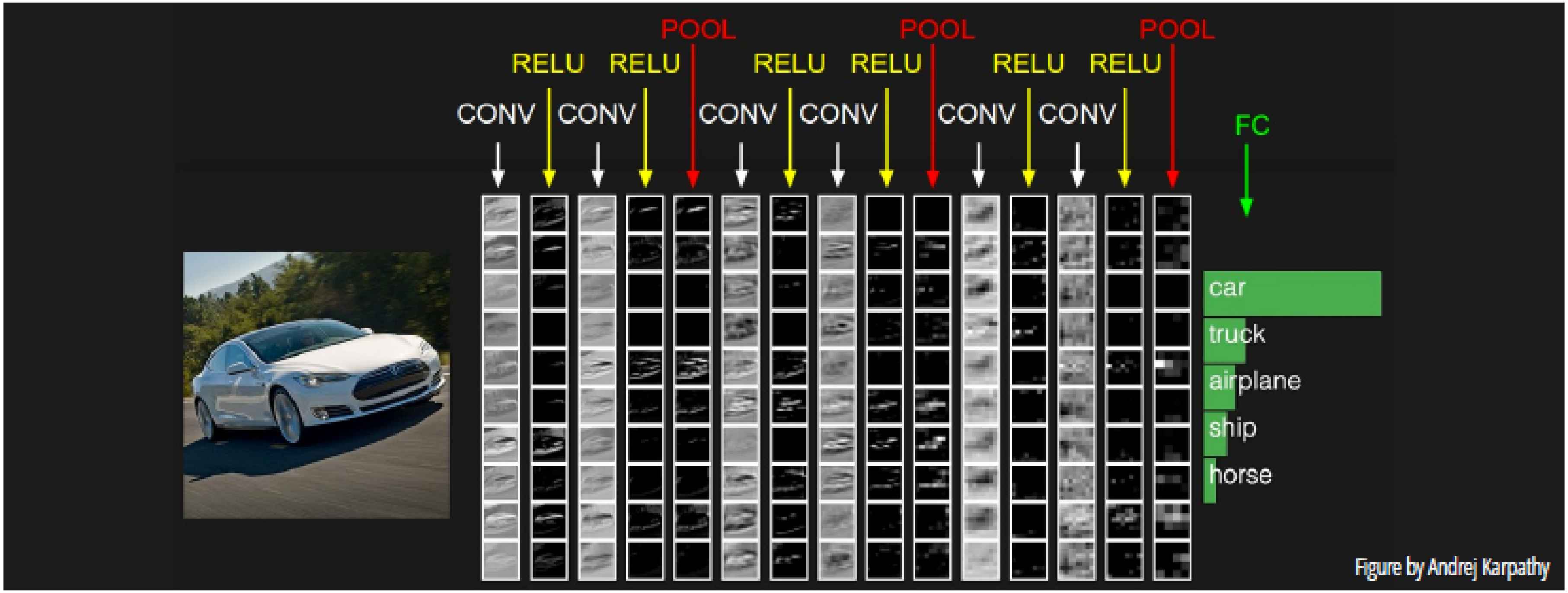
FC Layer (cont'd)

- Required computation vs. Learnable parameters



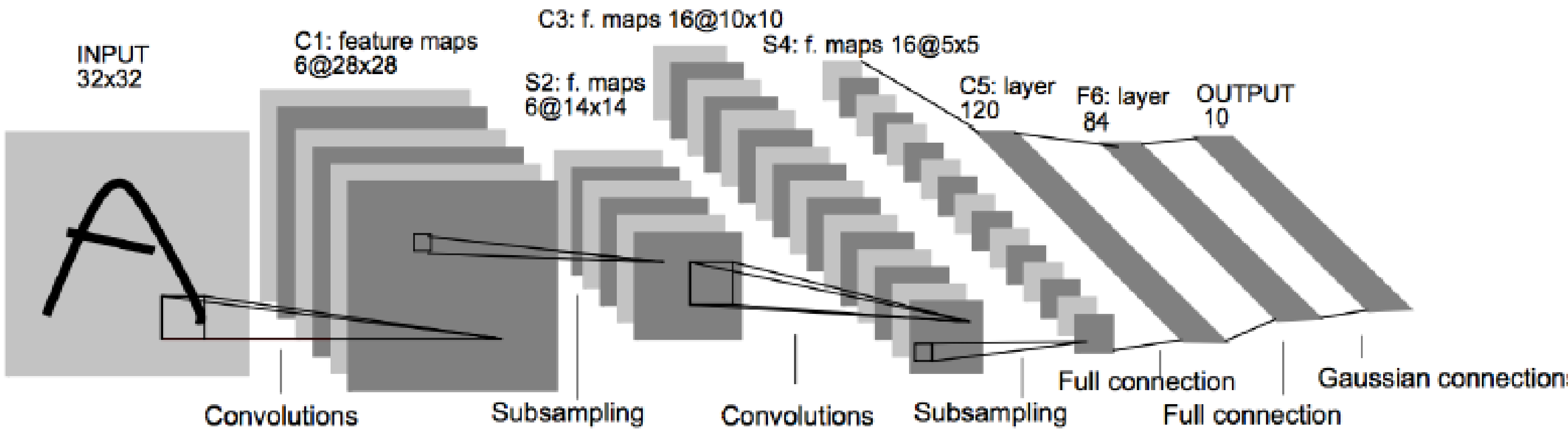
CNN


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

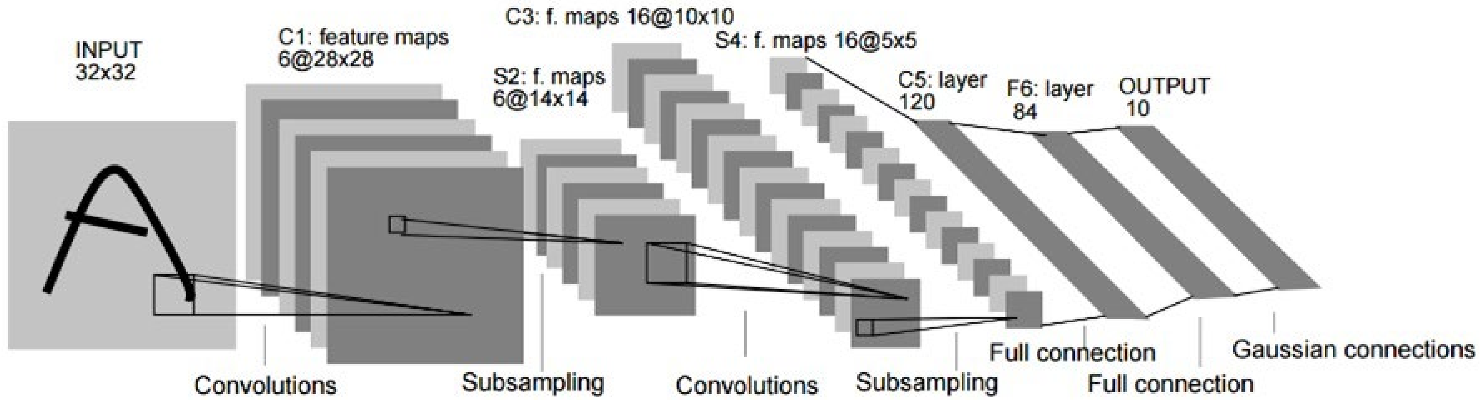

POOL
POOL
POOL
RELU RELU RELU RELU RELU RELU
CONV CONV CONV CONV CONV CONV
FC
car
truck
airplane
ship
horse
Figure by Andrej Karpathy

LeNet

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



LeNet [LeCun et al. 1998]



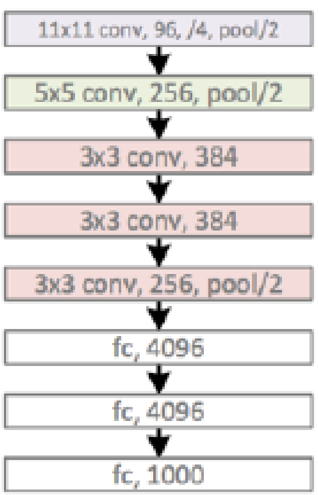

LeNet-1 from 1993

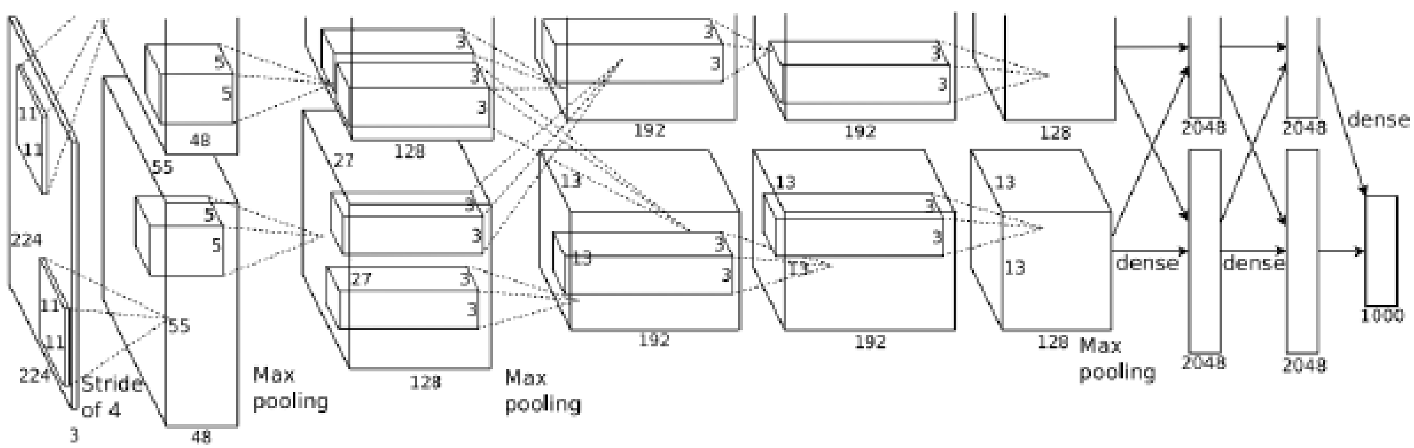
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

AlexNet [Krizhevsky et al., 2012]

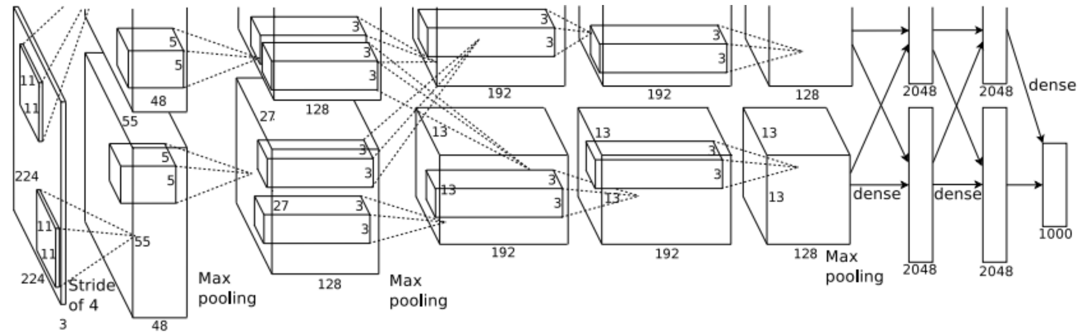
- Repopularized CNN
  by winning the ImageNet Challenge 2012
- 7 hidden layers, 650,000 neurons,
  60M parameters
- Error rate of 16% vs. 26% for 2nd place.

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)





of Hyperparameters in AlexNet (cont'd)



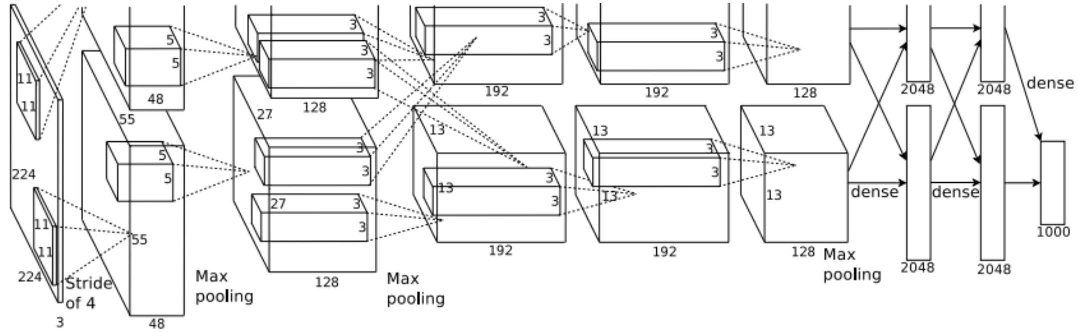
	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	784

Number of output elements = C * H' * W'
= 64*56*56 = 200,704

Bytes per element = 4 (for 32-bit floating point)

KB = (number of elements) * (bytes per elem) / 1024
= 200704 * 4 / 1024
= **784**

of Hyperparameters in AlexNet (cont'd)



	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

Weight shape = C_{out} x C_{in} x K x K

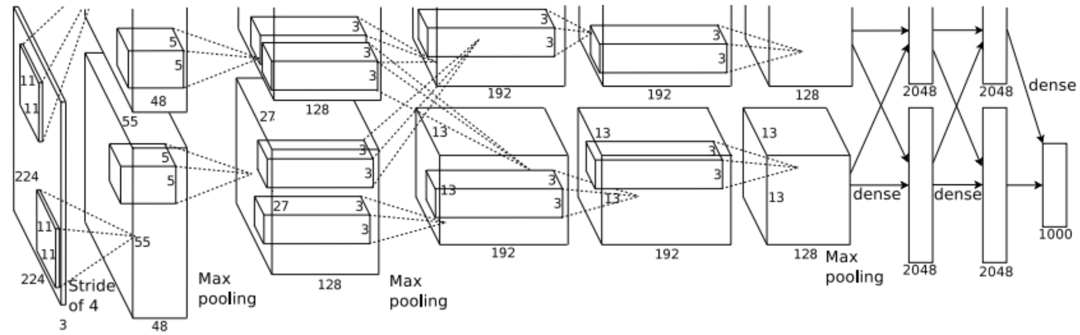
= 64 x 3 x 11 x 11

Bias shape = C_{out} = 64

Number of weights = 64*3*11*11 + 64

= **23,296**

of Hyperparameters in AlexNet (cont'd)



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73

Number of floating point operations (multiply+add)

= (number of output elements) * (ops per output elem)

= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$

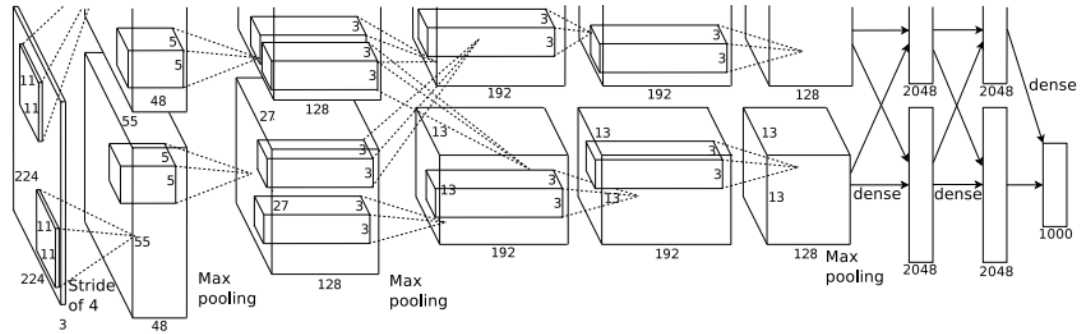
= (64 * 56 * 56) * (3 * 11 * 11)

= 200,704 * 363

= **72,855,552**

Slide credit: UMich EECS 498-007

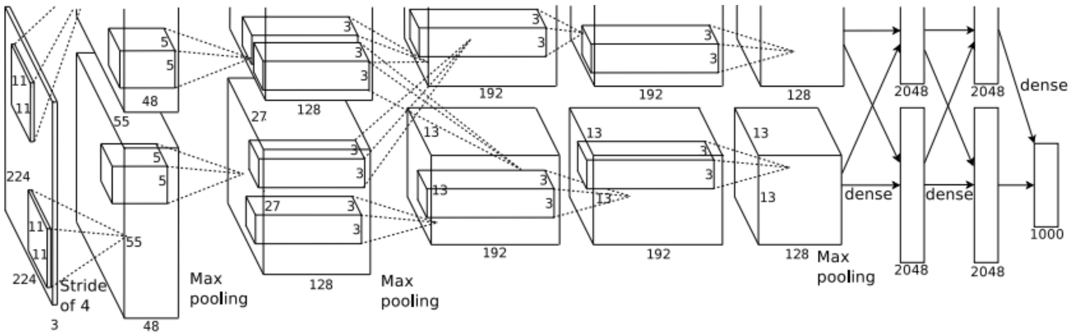
of Hyperparameters in AlexNet (cont'd)

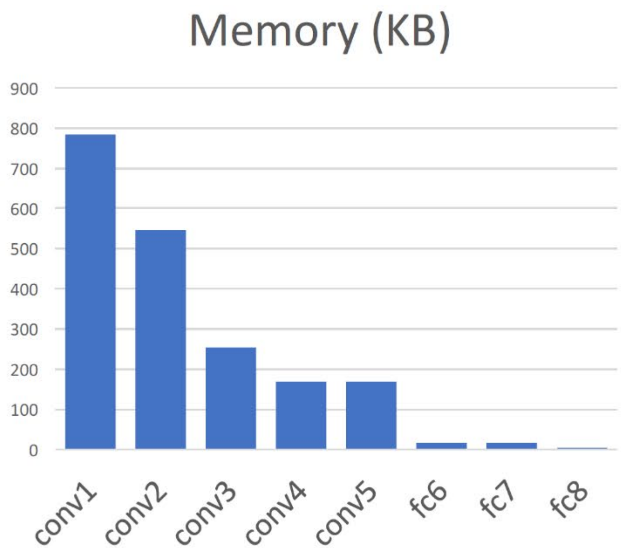


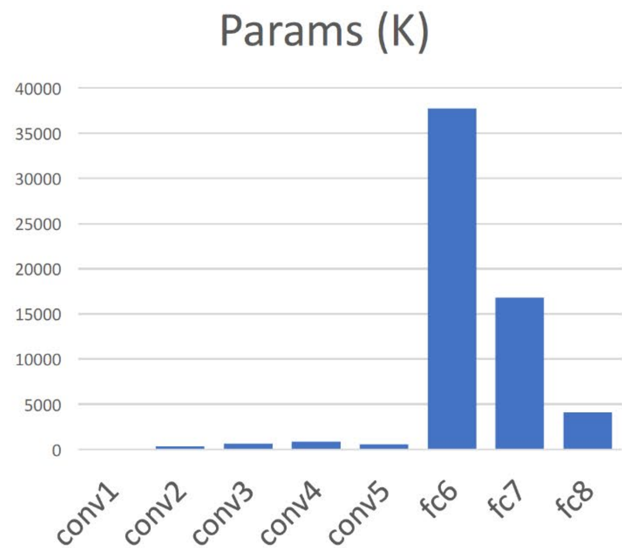
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

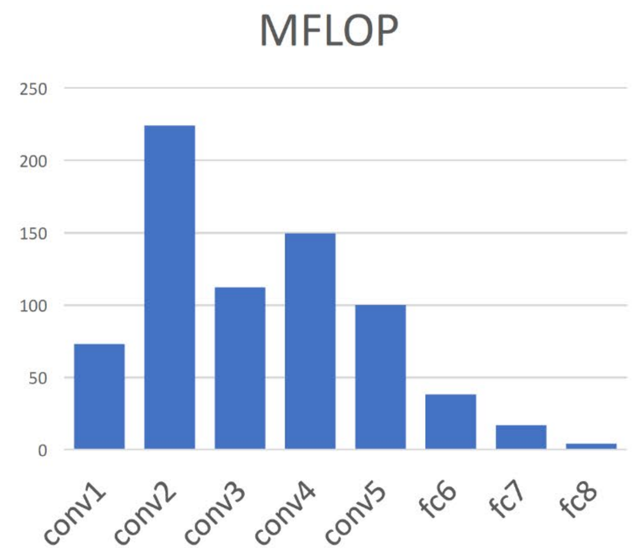
Slide credit: UMich EECS 498-007

Additional Remarks on AlexNet









Most of the memory usage in early convolution layers

Nearly all the parameters are in the fully connected layers

Most floating-point operations occur in the convolution layers

Deep or Not?

- Depth of the network is critical for performance.

AlexNet: 8 Layers with 18.2% top-5 error

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

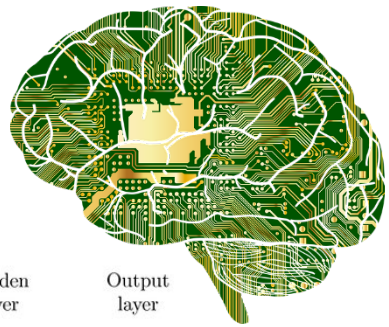
Removing Layer 6 and 7 reduces 50 million parameters, but only 5.7% drop in performance

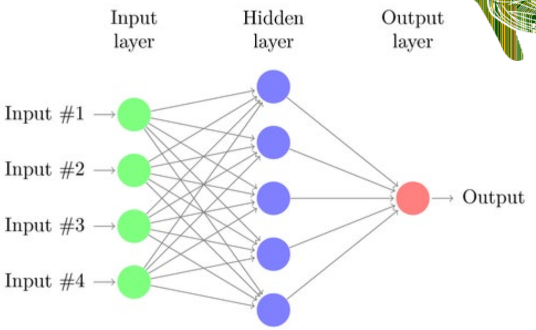
Removing middle conv layers reduces 1 million parameters, but only 3% drop in performance

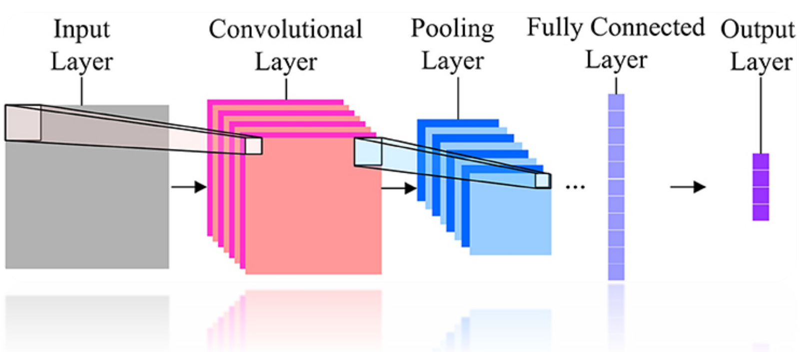
Removing feature & conv layers produces a **33% drop** in performance

What to Cover Today…

- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
- Image Segmentation

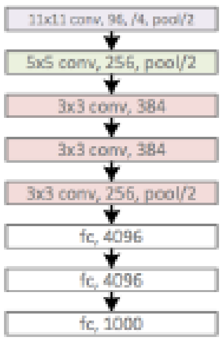




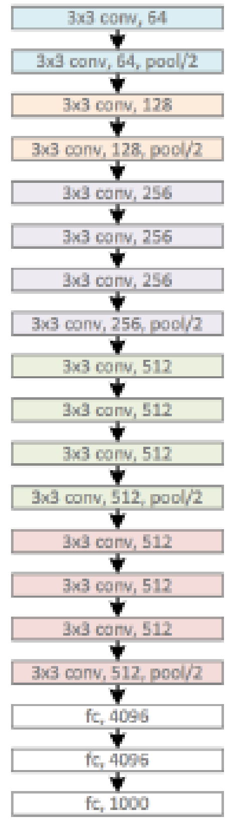


CNN: A Revolution of Depth

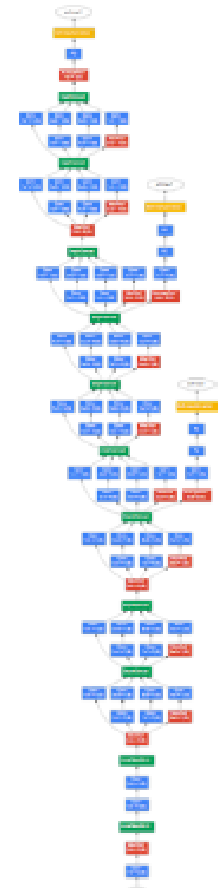
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

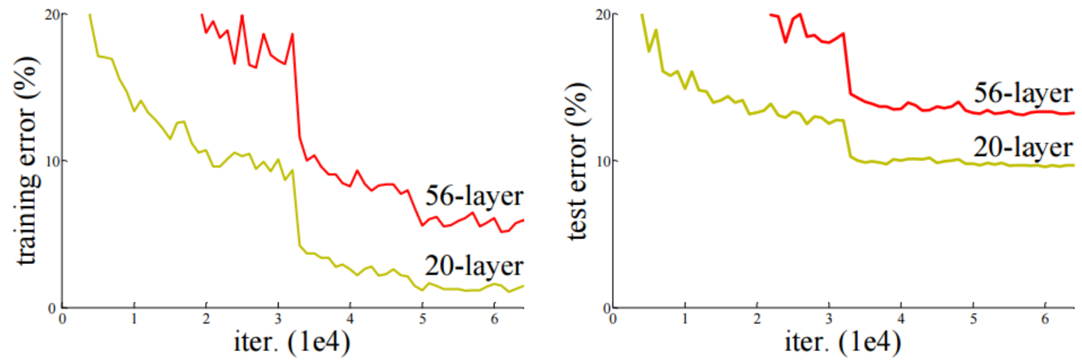


GoogleNet, 22 layers
(ILSVRC 2014)

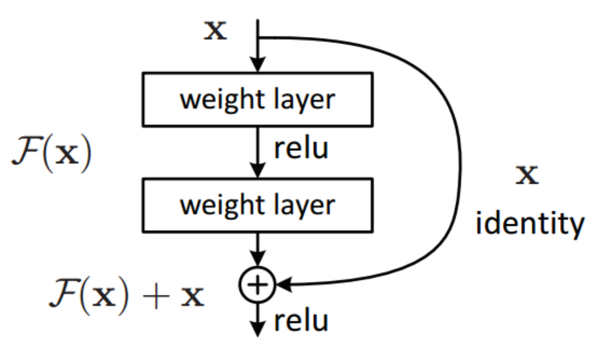


ResNet

- Can we just increase the #layer? What are the potential risks?



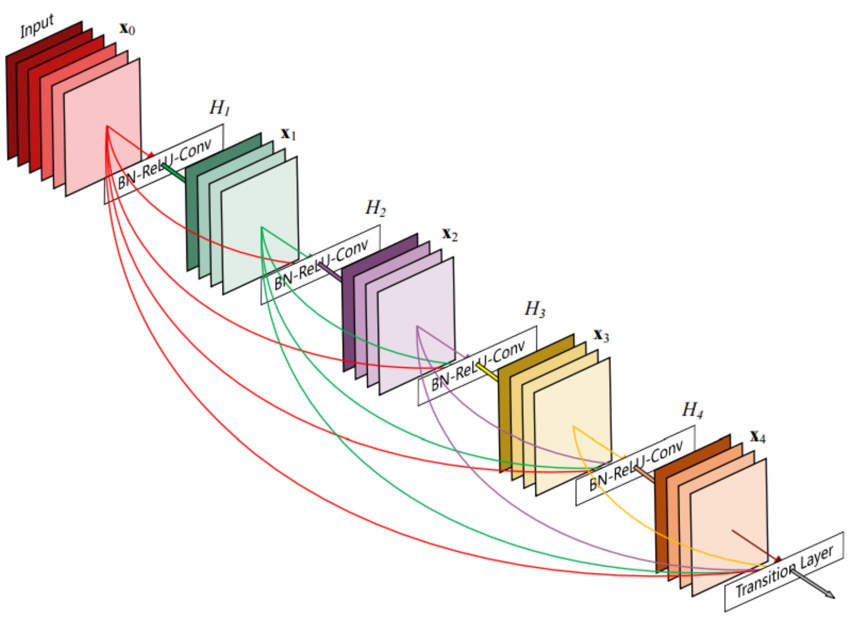
- How can we train very deep network?
  - Residual learning

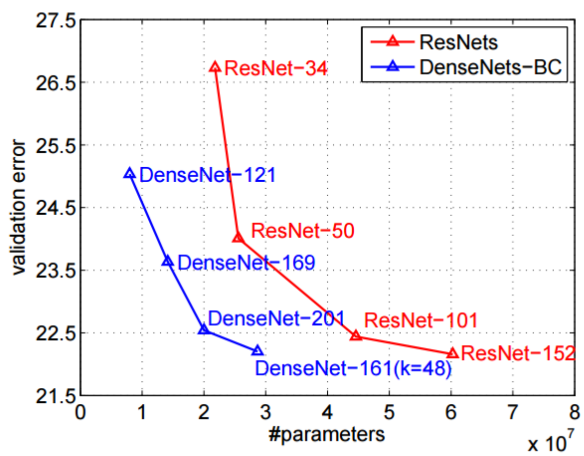


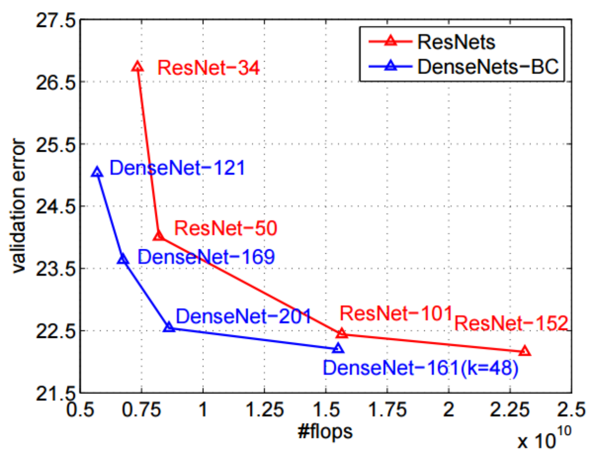
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

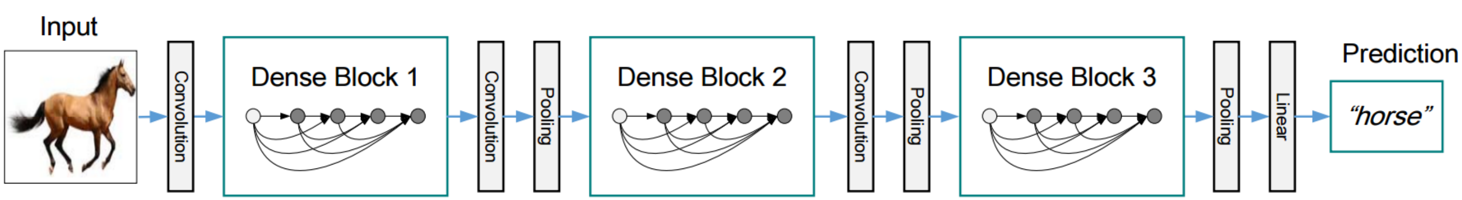
DenseNet [CVPR'17]

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



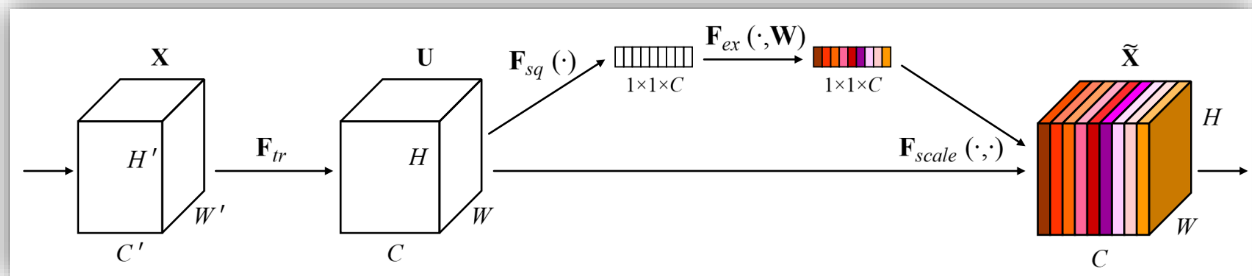


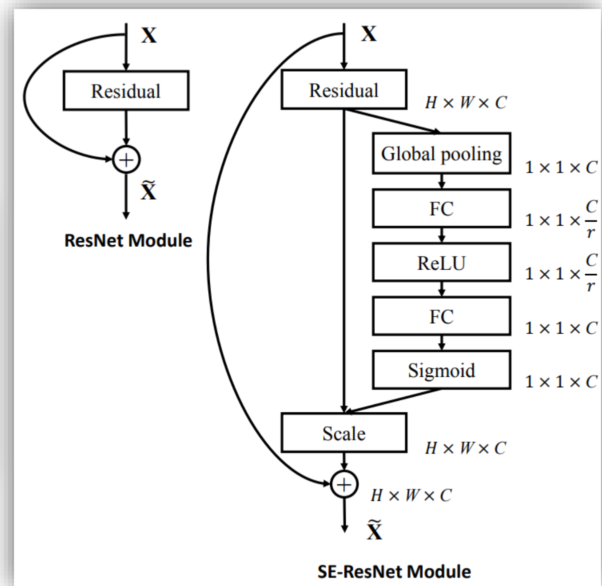




Squeeze-and-Excitation Net (SENet)

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



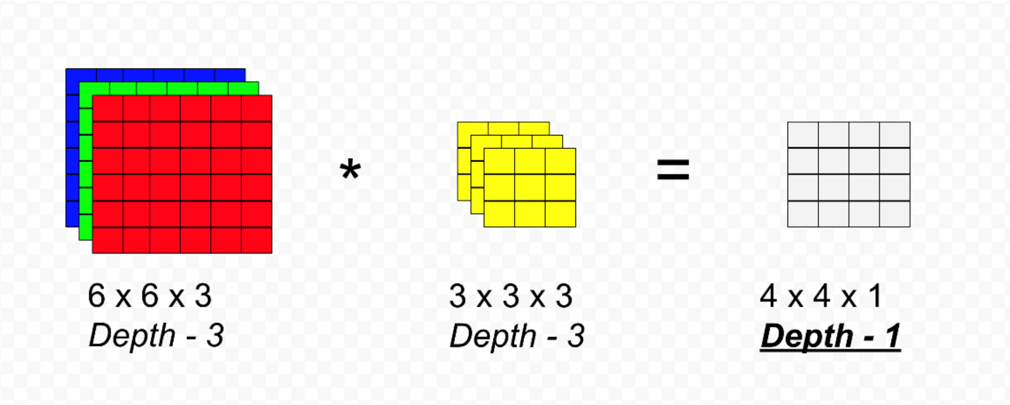


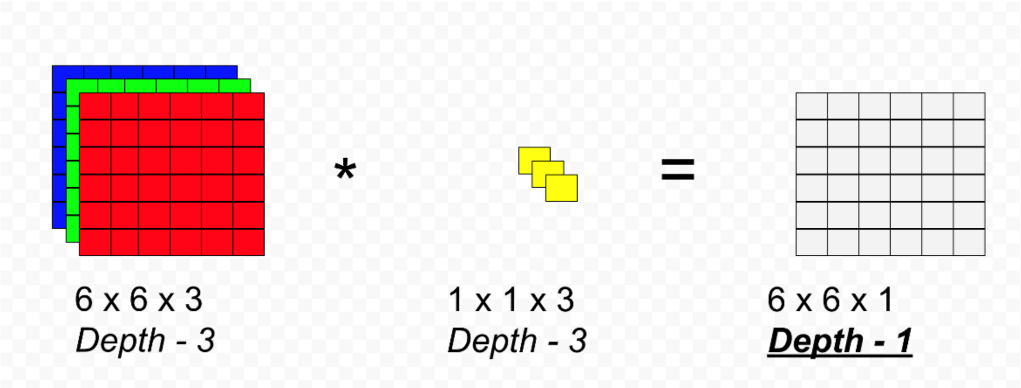
	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	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.

Btw, what is 1x1 Convolution?

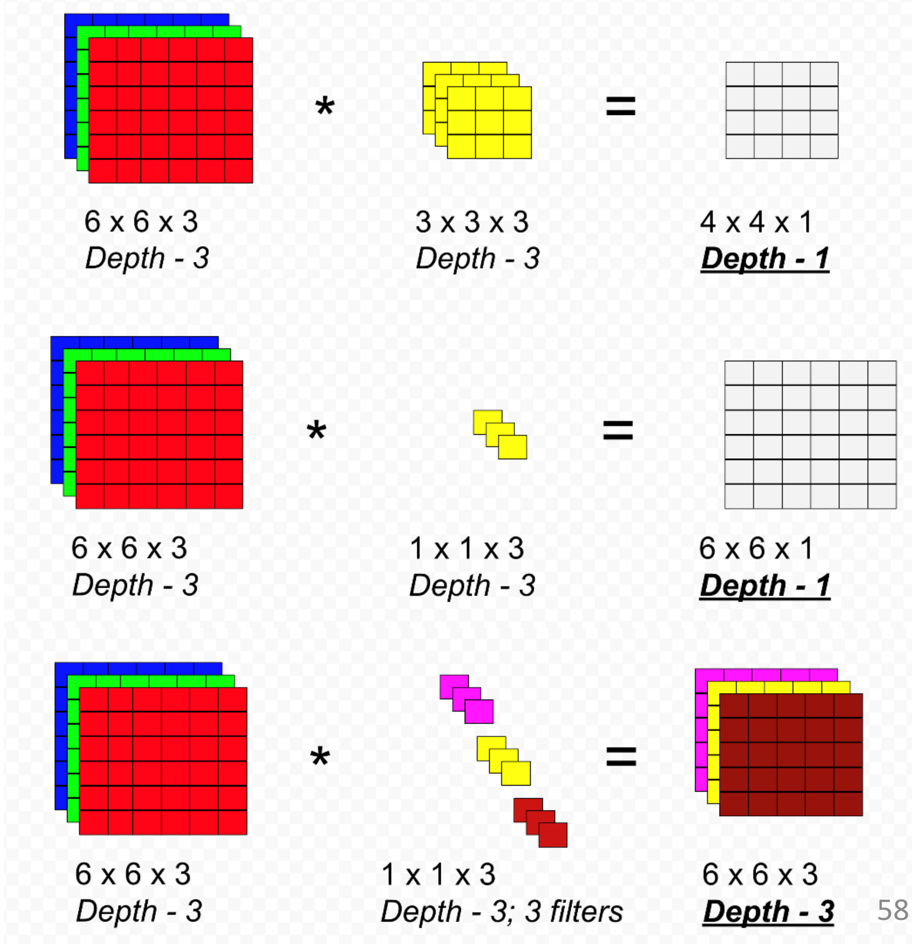
- Doesn’t 1x1 convolution sound redundant?
- Actually, it’s for accelerating computation purposes





What is 1x1 Convolution? (cont'd)

- Doesn't 1x1 convolution sound redundant?
- Simply speaking, it provides…
  - Dimension reduction
  - Additional nonlinearity



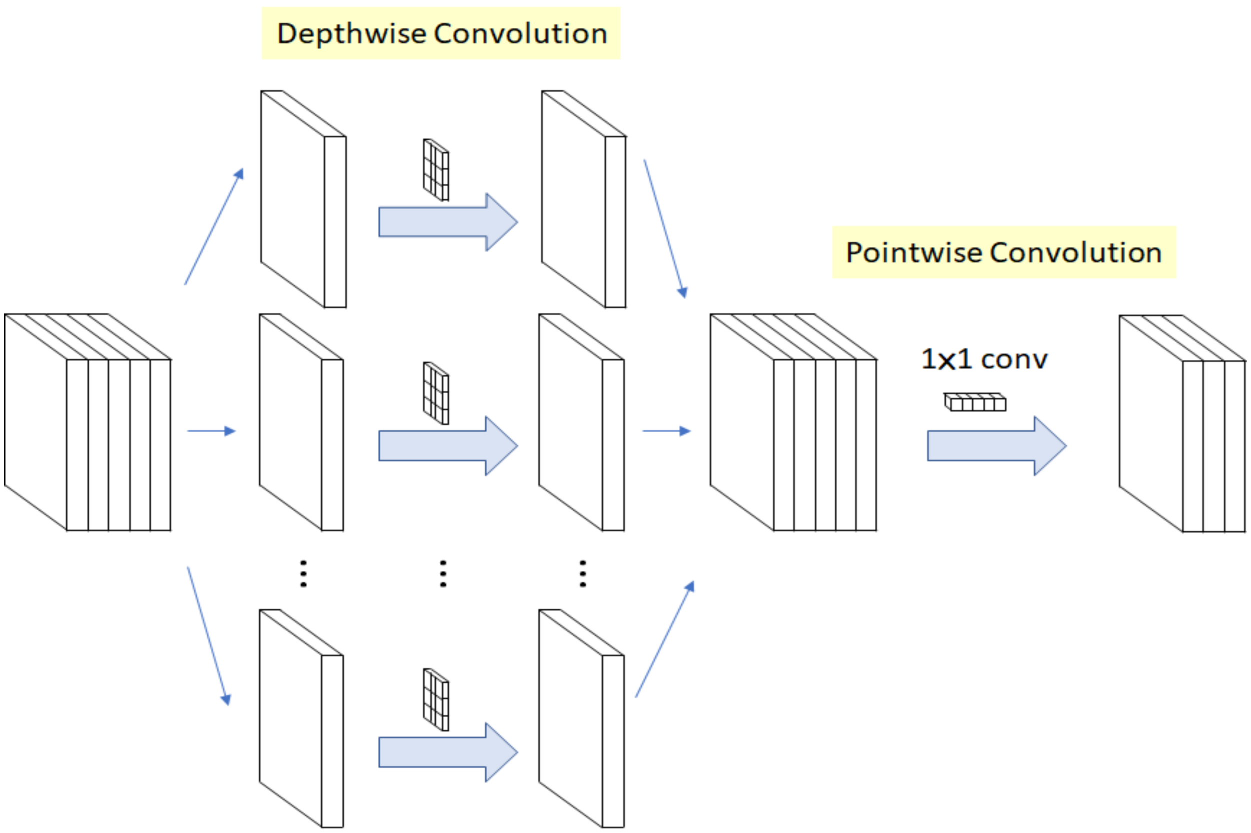
What is 1x1 Convolution? (cont'd)

- **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 16 {1 x 1x 192} kernels into {28 x 28 x 16}, followed by convolution with into 32 {5 x 5 x 16} kernels 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 (2.4M + 10M) << 120M; what's the price to pay?

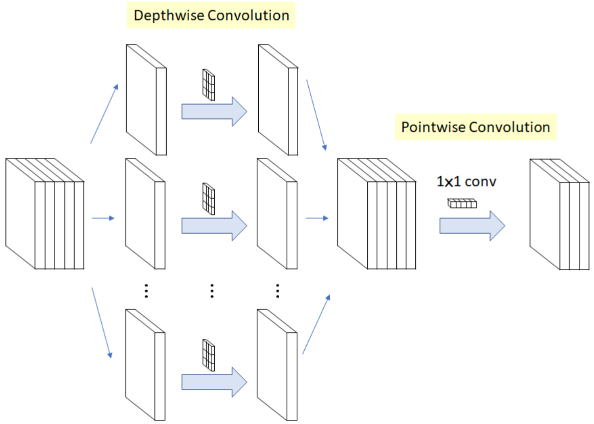
MobileNets: Tiny Networks for End Devices

- MobileNet V1
  - Depthwise & pointwise convolution



Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017

MobileNets (cont'd)



- MobileNet V1
  - Depthwise & pointwise convolution
  - Reduced Computation
    - Input feature map D_F x D_F pixels with M channels, kernel size D_K, & output with N channels
    - The ratio of required computation of depth+pointwise conv. and standard conv. is :

$$\frac{\underbrace{\boxed{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F}}_{\text{Depthwise Convolution}} + \underbrace{\boxed{M \cdot N \cdot D_F \cdot D_F}}_{\text{Pointwise Convolution}}}{\underbrace{\boxed{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}}_{\text{Standard Convolution}}} = \frac{1}{N} + \frac{1}{D_K^2}$$

    - Thus, depth+pointwise convolution requires only **$1/N + 1/D_K^2$** of the computation cost compared with that of standard convolution.

  - Variants of MobileNets are available!

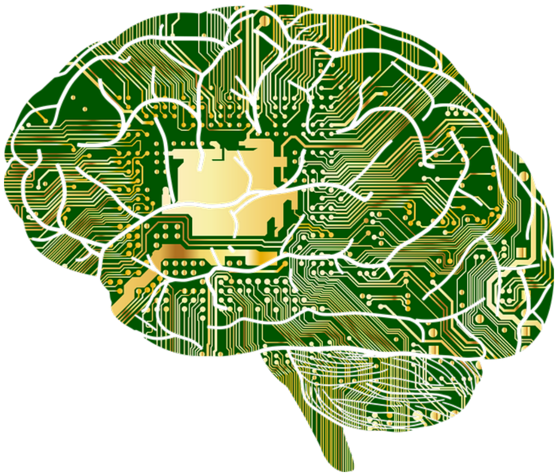
Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017

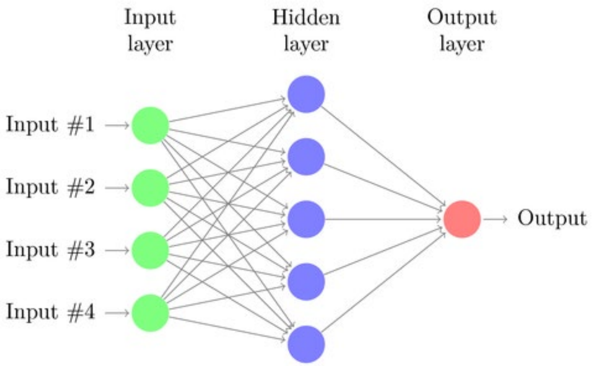
Remarks

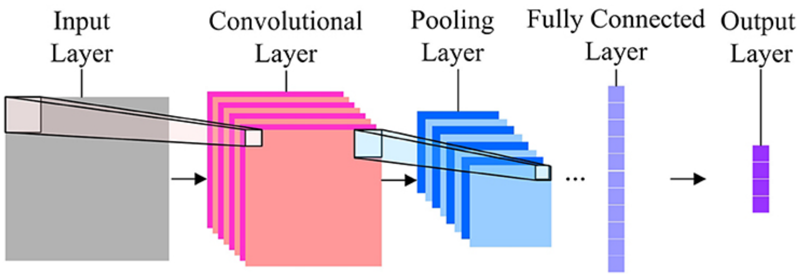
- CNN:
  - Reduce the number of parameters
  - Reduce the memory requirements
  - Make computation independent of the size of the image
- Neuroscience provides strong inspiration on the NN design, but little guidance on how to train CNNs.
- Few structures discussed: convolution, nonlinearity, pooling

What's to Be Covered Today…

- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
- Image Segmentation



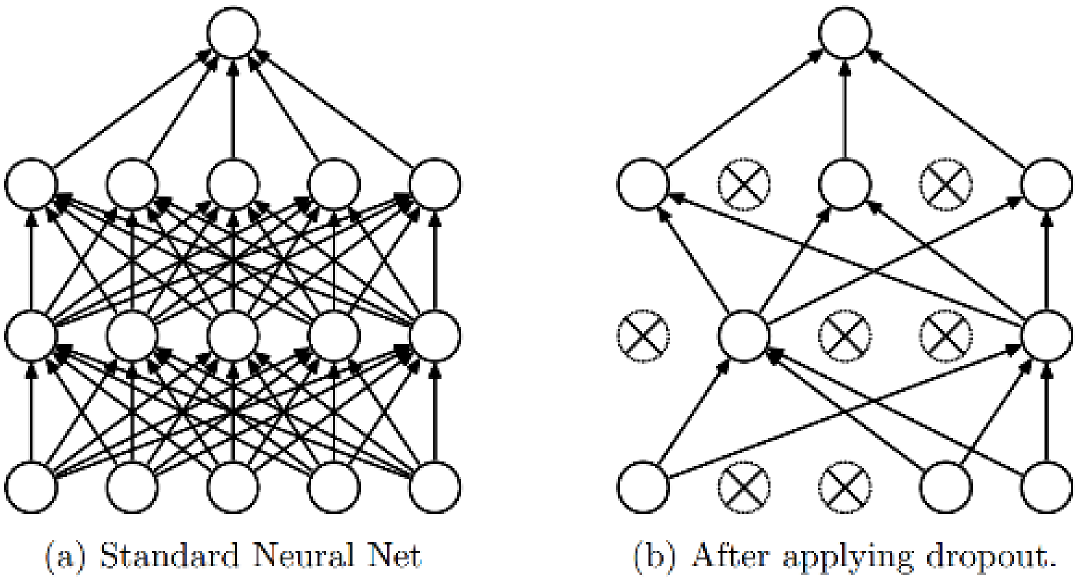




Selected Tricks for Training Deep Learning Models

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

Dropout



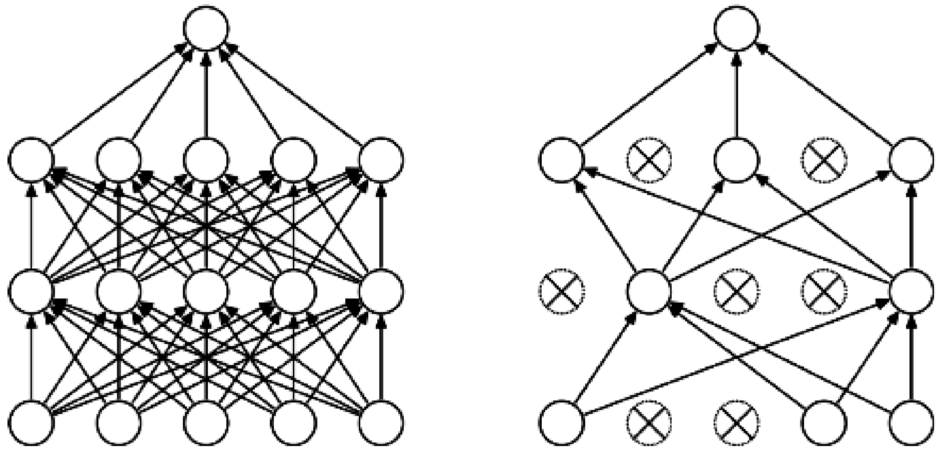
Intuition: successful **conspiracies**

Example: 50 people planning a conspiracy

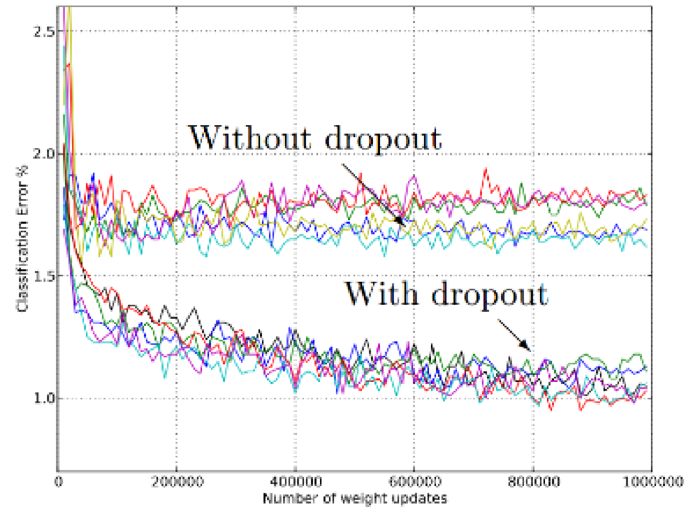
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

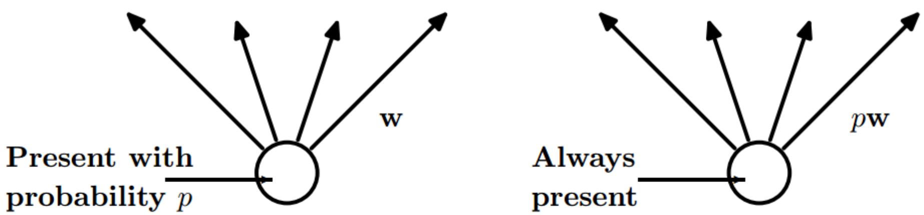
Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

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

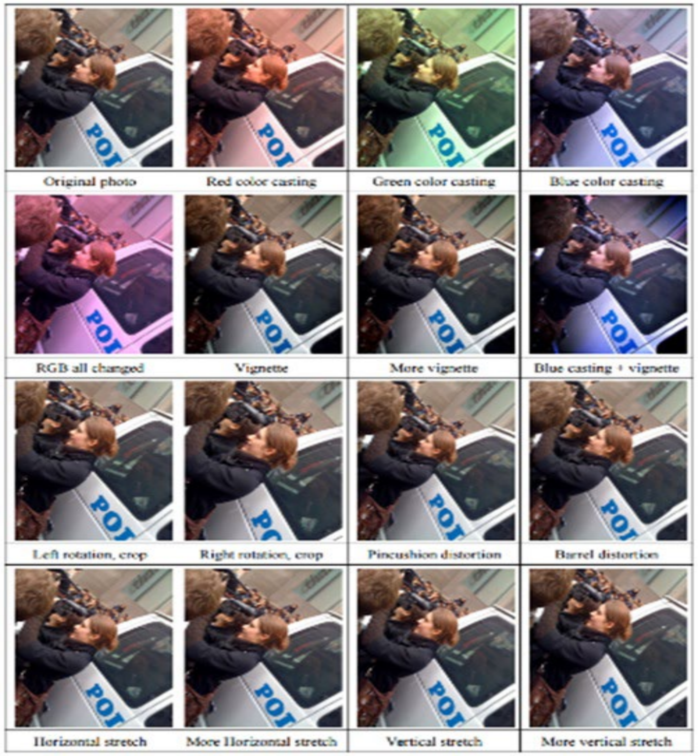
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

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

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

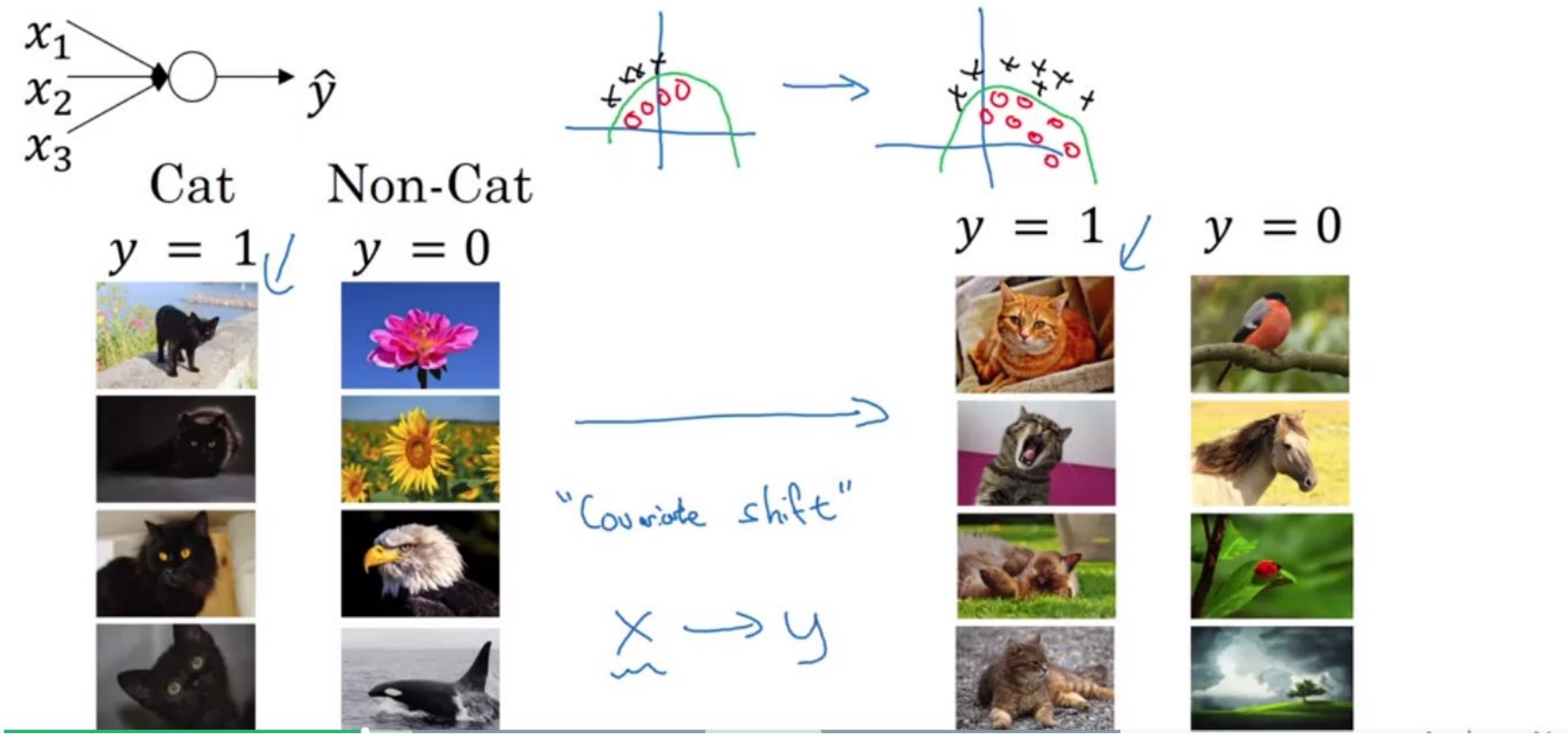
Data Augmentation (Jittering)

- DL typically requires larger # of data for training
- Collecting data is time and cost consuming…
- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion
    and so on…
  - See any concerns?



Deep Image [Wu et al. 2015]

Batch Normalization



Credit: Andrew Ng

Batch Normalization

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

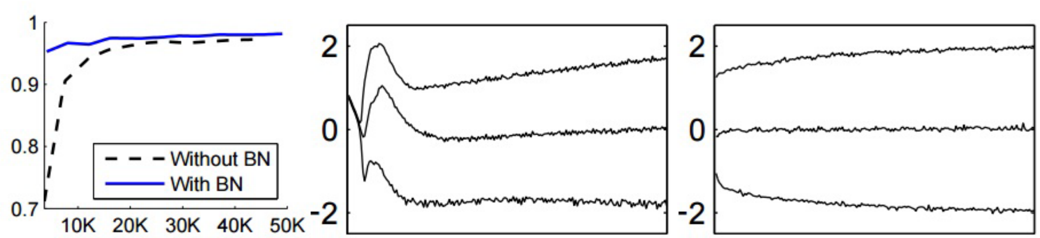
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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



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

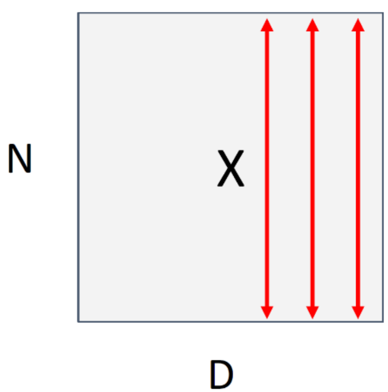
Batch Normalization (cont'd)

- Remarks
  - Differentiable function; back propagation OK

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

  - Procedure

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean across N samples

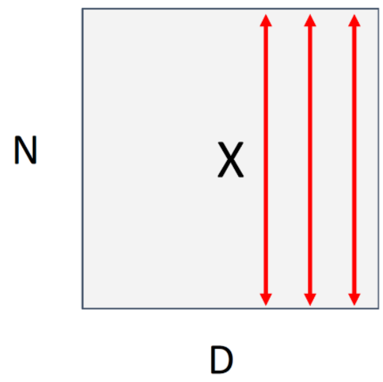
$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel std across N samples

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Batch Normalization (cont'd)



- Remarks
  - Differentiable function; back propagation OK

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

  - Procedure (cont'd)
    - With learnable scale and shift parameters γ and β
      to alleviate the hard constraint of zero-mean and unit variance

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

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

  - Mean and variance estimated from each mini-batch during training
    - What about inference/testing?

$\mu_j =$ (Running) average of values seen during training — Per-channel mean across N samples

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel std across N samples

Instance Normalization in CNN

Batch Normalization for convolutional networks

$$\boxed{\begin{array}{l} \mathbf{x}:\ N \times C \times H \times W \\ \text{Normalize} \\ \boldsymbol{\mu}, \boldsymbol{\sigma}:\ 1 \times C \times 1 \times 1 \end{array}}$$

$$\gamma, \beta:\ 1 \times C \times 1 \times 1$$

$$y = \gamma (x - \mu) / \sigma + \beta$$

Instance Normalization for convolutional networks
Same behavior at train / test!

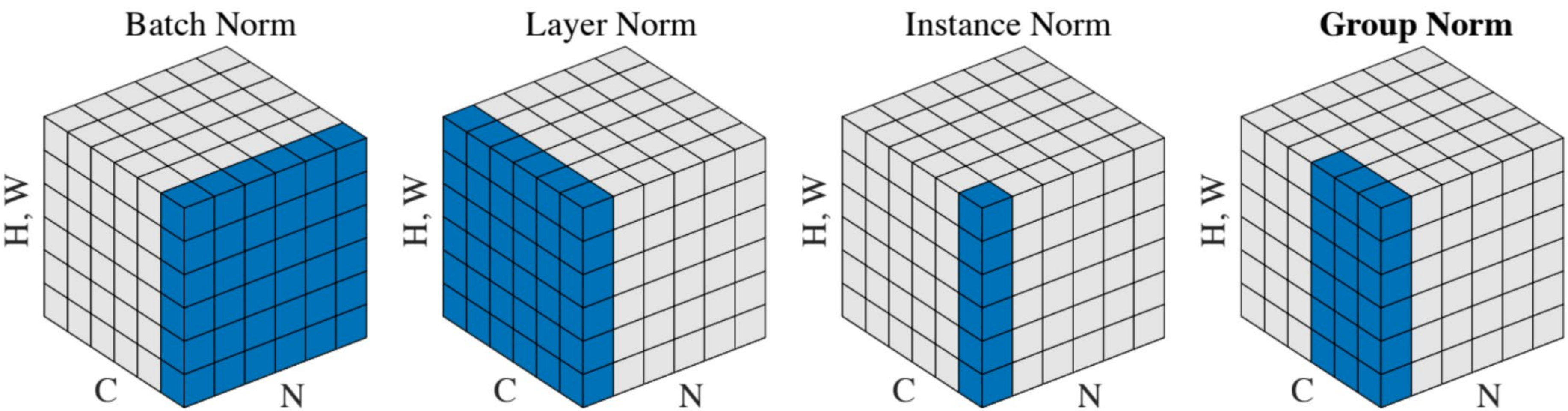
$$\boxed{\begin{array}{l} \mathbf{x}:\ N \times C \times H \times W \\ \text{Normalize} \\ \boldsymbol{\mu}, \boldsymbol{\sigma}:\ N \times C \times 1 \times 1 \end{array}}$$

$$\gamma, \beta:\ 1 \times C \times 1 \times 1$$

$$y = \gamma (x - \mu) / \sigma + \beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

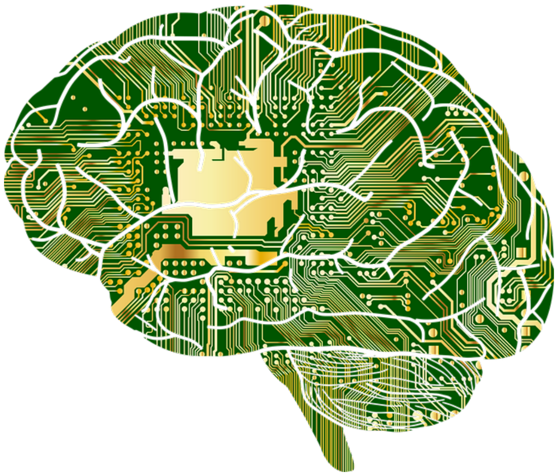
Variants of Normalization in Training CNN

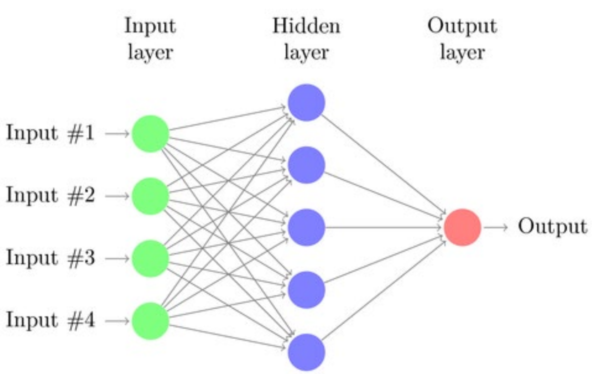


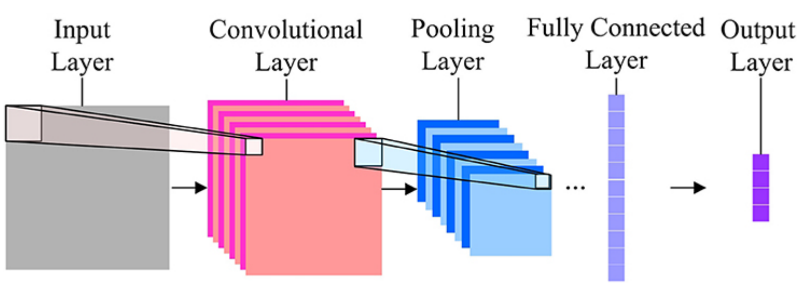
Wu and He, “Group Normalization”, ECCV 2018

What's to Be Covered Today…

- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
  - Self-Supervised Learning for CNN
- Image Segmentation







Supervised Learning

- Most DL models are learned in a supervised fashion…

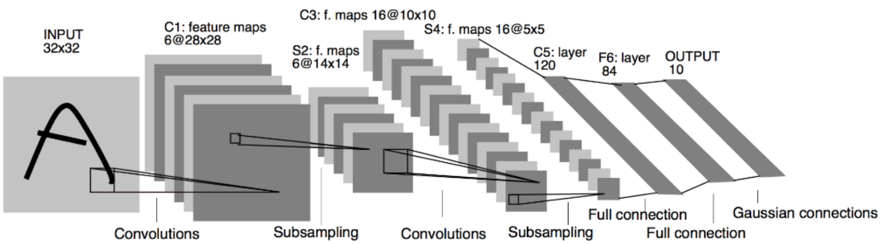
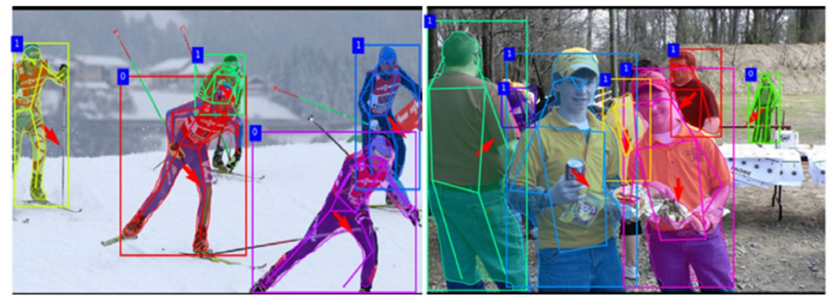
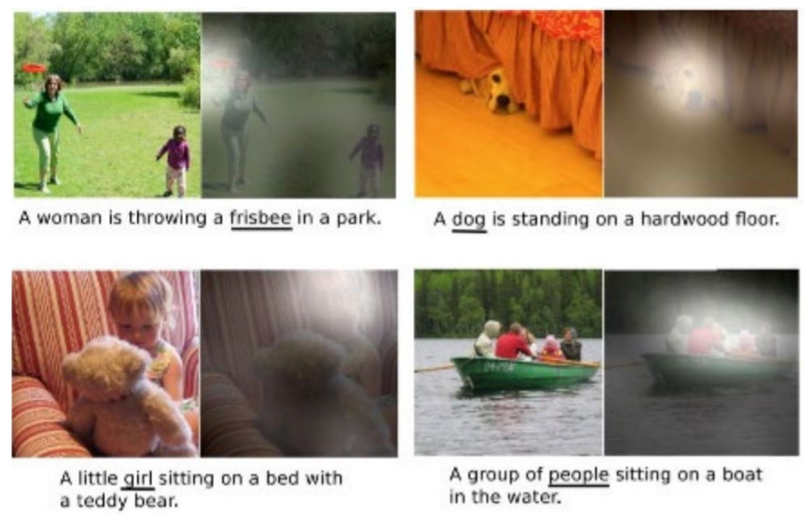


Image classification



Object detection

Semantic segmentation



Visual question answering

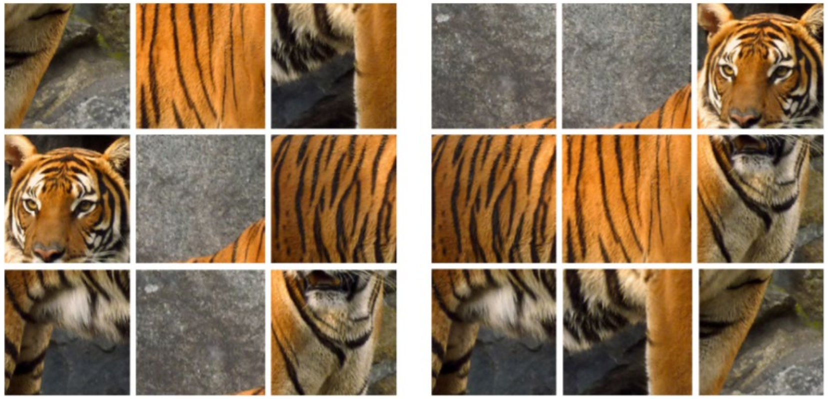
- In real world scenarios, data-annotation is quite **time-consuming**
- Could one exploit supervised signals from **unlabeled** data?

Self-Supervised Learning (SSL)

- Learning (somewhat) discriminative feature representations from **unlabeled** data
- Create self-supervised tasks via **data augmentation**

Colorization

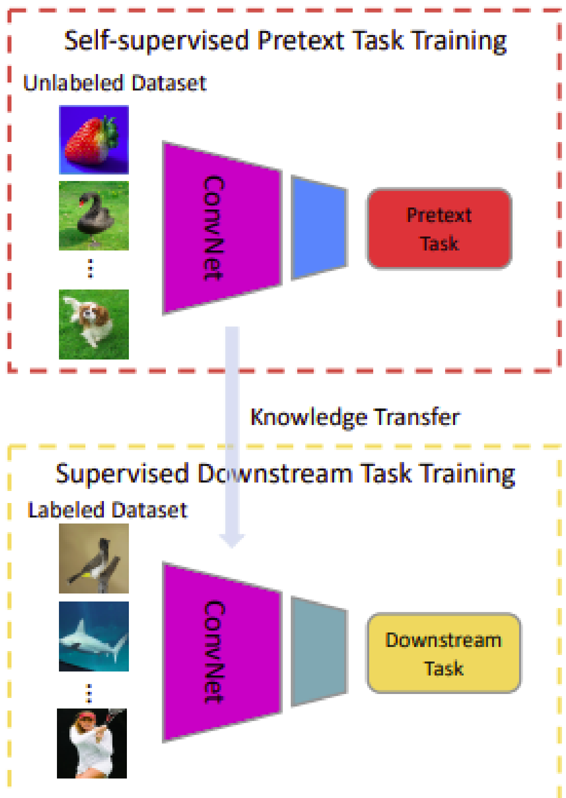
Rotation



Jigsaw Puzzle

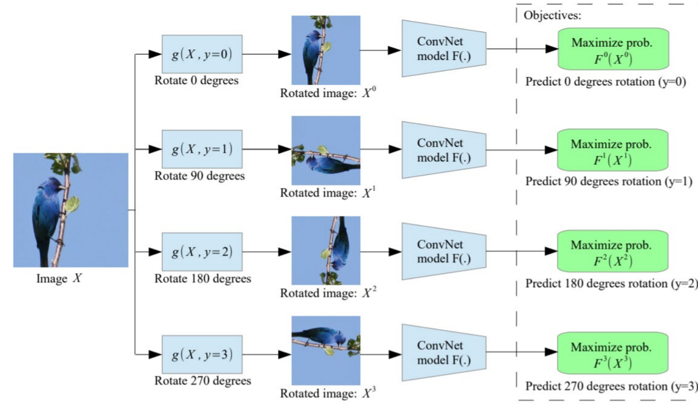
A Typical SSL Procedure

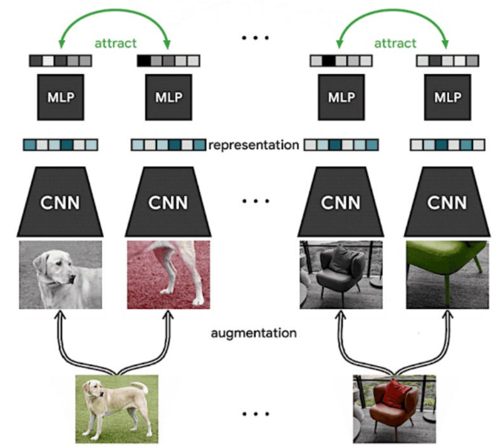
- Stage 1: Self-Supervised Pretraining (w/ a *large* # of unlabeled data)
- Stage 2: Supervised Fine-tuning (w/ a *small* # of labeled data)
- Often performs favorably against fullysupervised trained models

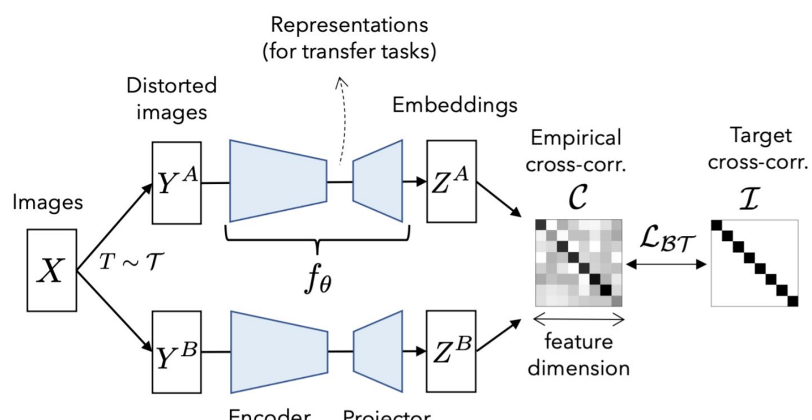


Selected SSL Techniques

- Pretext Tasks
  - Jigsaw (ECCV'16)
  - RotNet (ICLR'18)
- Contrastive Learning
  - CPC (ICML'20)
  - SimCLR (ICML'20)
- Learning w/o negative samples
  - BYOL (NeurIPS'20)
  - Barlow Twins (ICML'21)

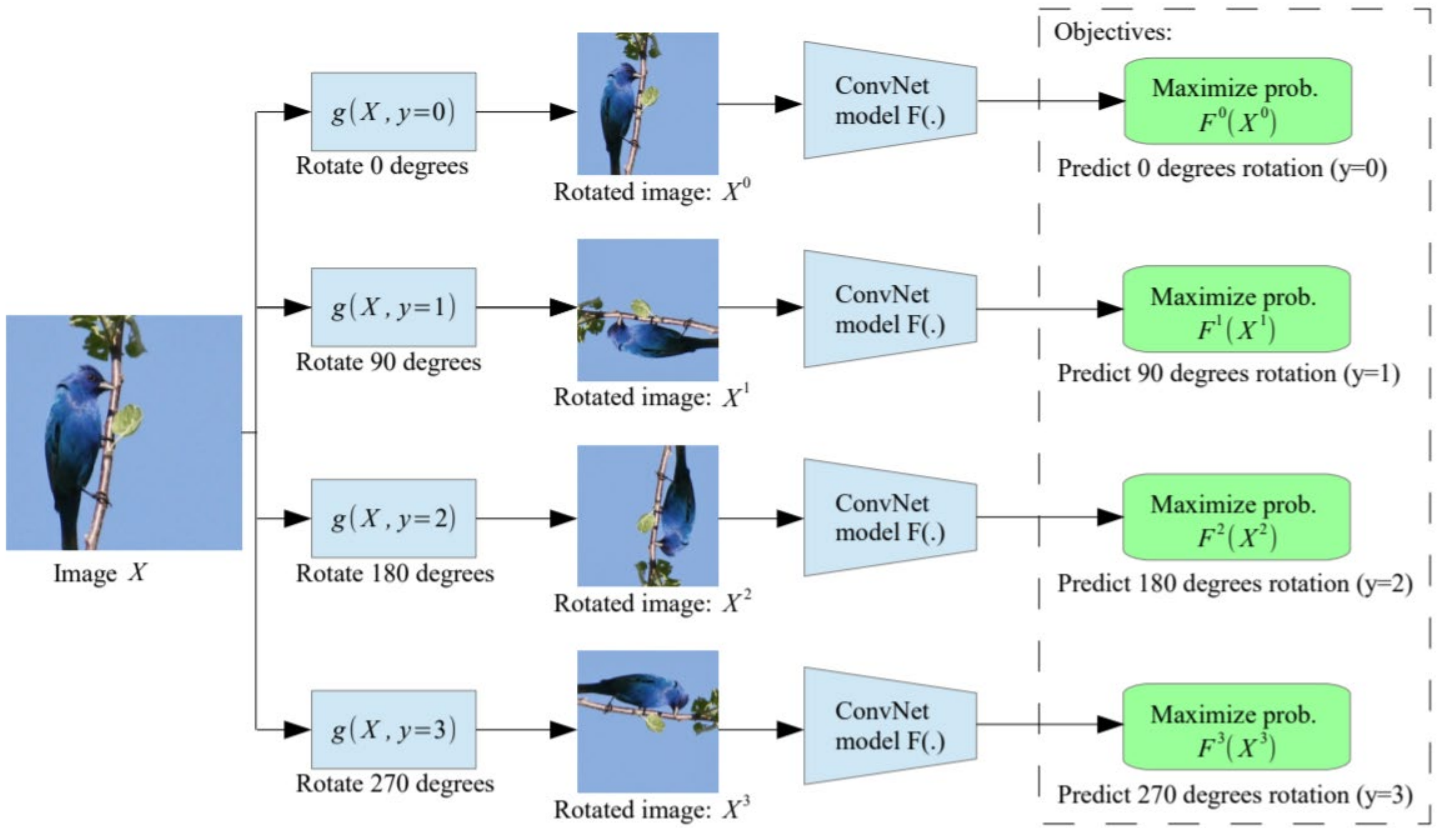






RotNet

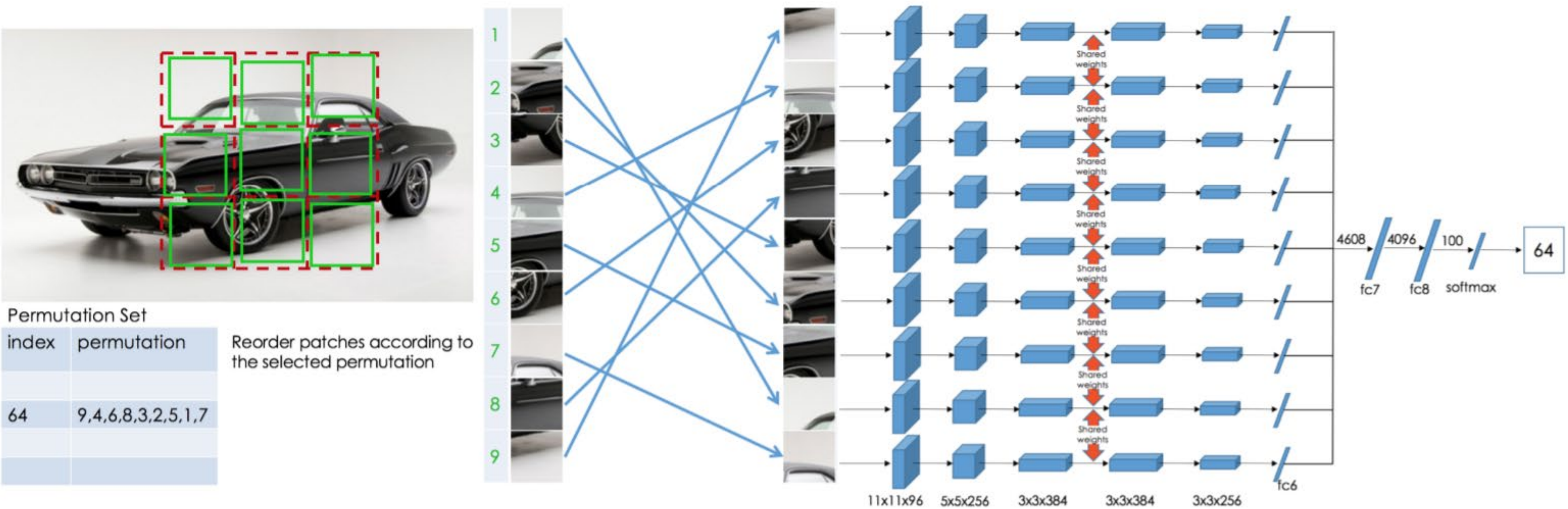
- Learning to predict the **rotation** angle



Gidaris et al. “Unsupervised Representation Learning by Predicting Image Rotations.” ICLR 2018

Jigsaw Puzzle

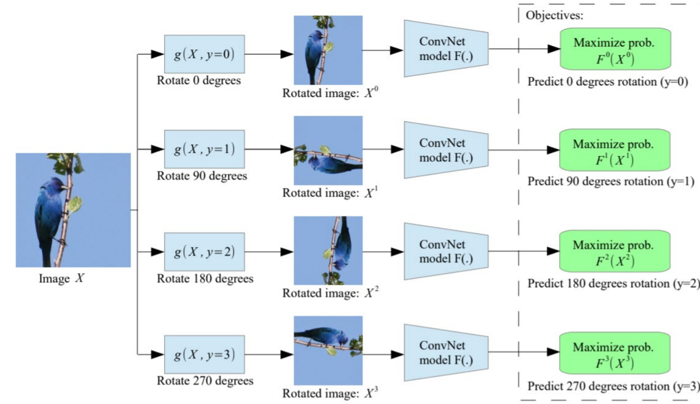
- Assign the **permutation index** and perform augmentation
- Solve jigsaw puzzle by predicting the permutation index

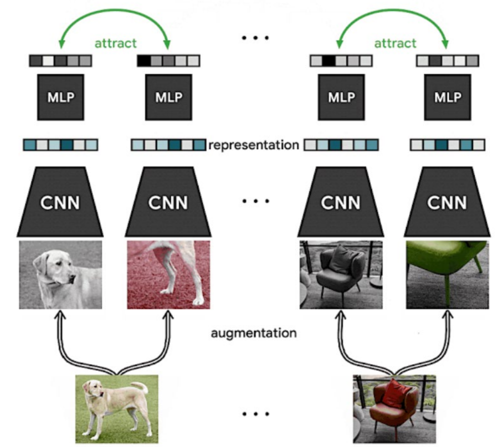


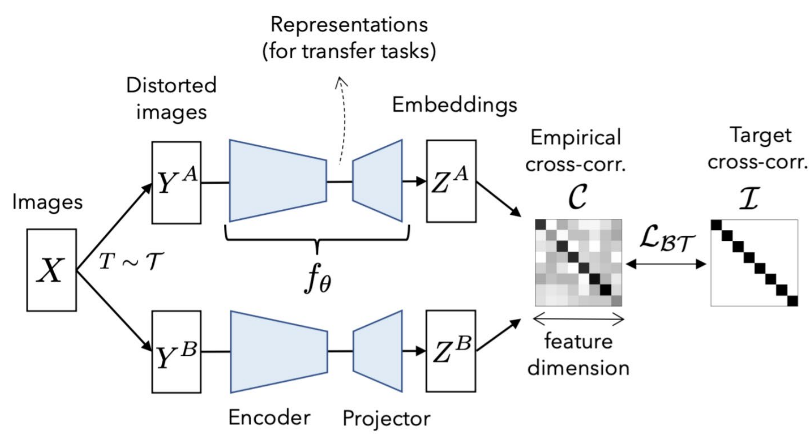
Noroozi et al. “Unsupervised learning of visual representations by solving jigsaw puzzles.” ECCV 2016

Selected SSL Techniques

- Pretext Tasks
  - Jigsaw (ECCV’16)
  - RotNet (ICLR’18)
- Contrastive Learning
  - CPC (ICML’20)
  - SimCLR (ICML’20)
- Learning w/o negative samples
  - BYOL (NeurIPS’20)
  - Barlow Twins (ICML’21)

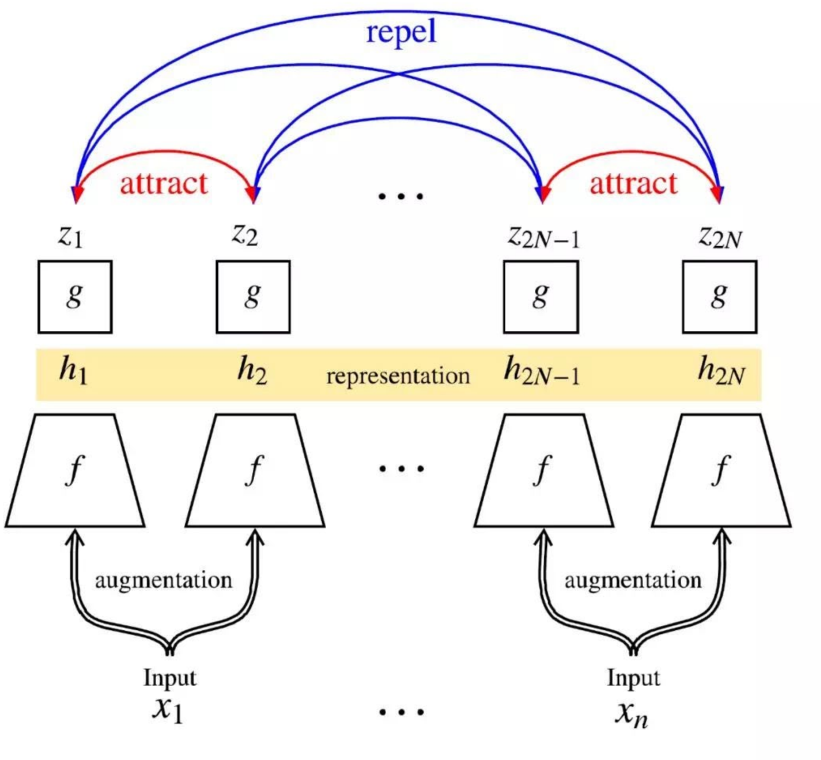






SimCLR

- **Attract** augmented images and **repel** negative samples
- Improve the representation quality with **projection heads** (*g*)…why?



Chen et al. "A simple framework for contrastive learning of visual representations." ICML 2020

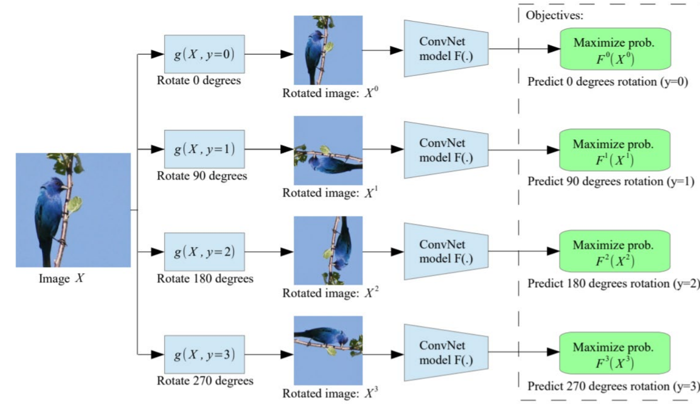
SimCLR

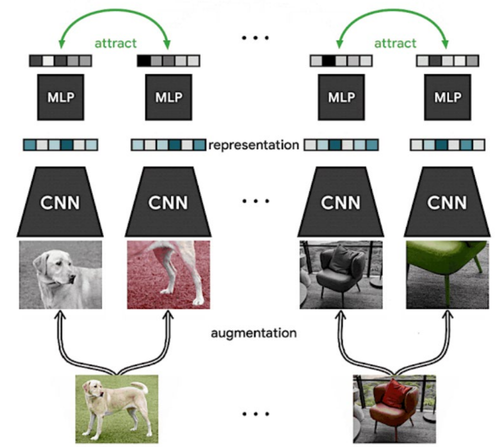
- Experiments on semi-supervised settings

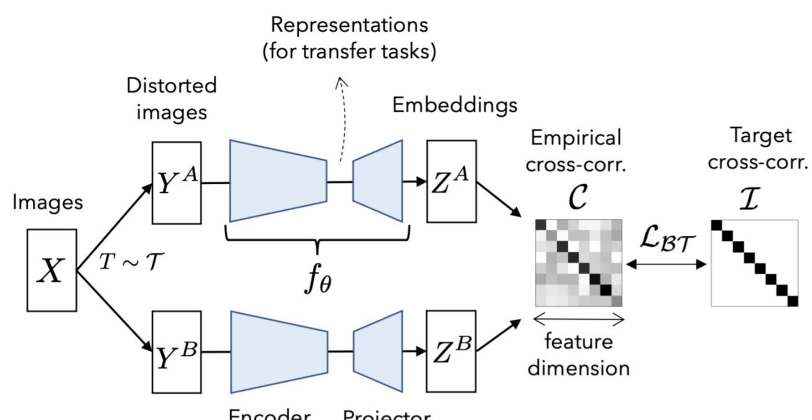
Method	Architecture	Label fraction 1%	Label fraction 10%
		Top 5	
Supervised baseline	ResNet-50	48.4	80.4
Methods using other label-propagation:			
Pseudo-label	ResNet-50	51.6	82.4
VAT+Entropy Min.	ResNet-50	47.0	83.4
UDA (w. RandAug)	ResNet-50	-	88.5
FixMatch (w. RandAug)	ResNet-50	-	89.1
S4L (Rot+VAT+En. M.)	ResNet-50 (4×)	-	91.2
Methods using representation learning only:			
InstDisc	ResNet-50	39.2	77.4
BigBiGAN	RevNet-50 (4×)	55.2	78.8
PIRL	ResNet-50	57.2	83.8
CPC v2	ResNet-161(∗)	77.9	91.2
SimCLR (ours)	ResNet-50	75.5	87.8
SimCLR (ours)	ResNet-50 (2×)	83.0	91.2
SimCLR (ours)	ResNet-50 (4×)	**85.8**	**92.6**

Selected SSL Techniques

- Pretext Tasks
  - Jigsaw (ECCV'16)
  - RotNet (ICLR'18)
- Contrastive Learning
  - CPC (ICML'20)
  - SimCLR (ICML'20)
- Learning w/o negative samples
  - BYOL (NeurIPS'20)
  - Barlow Twins (ICML'21)

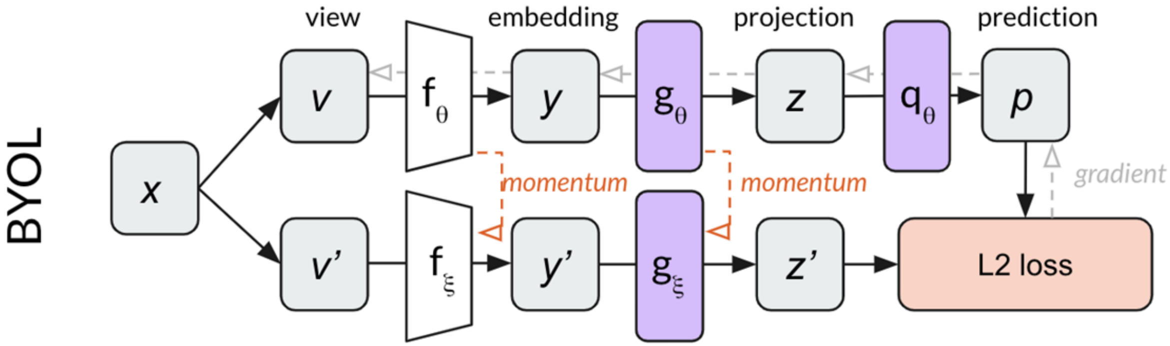


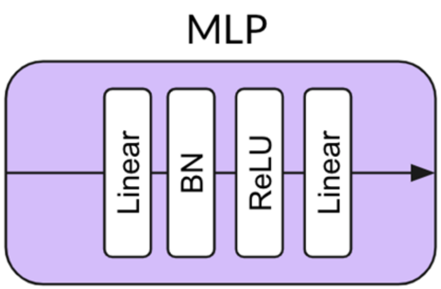




BYOL (Bootstrap Your Own Latent)

- No need of negative pairs
- Introduce the **predictor** for architecture asymmetry to avoid model collapse
- Model update via Exponential Moving Average (**EMA**)



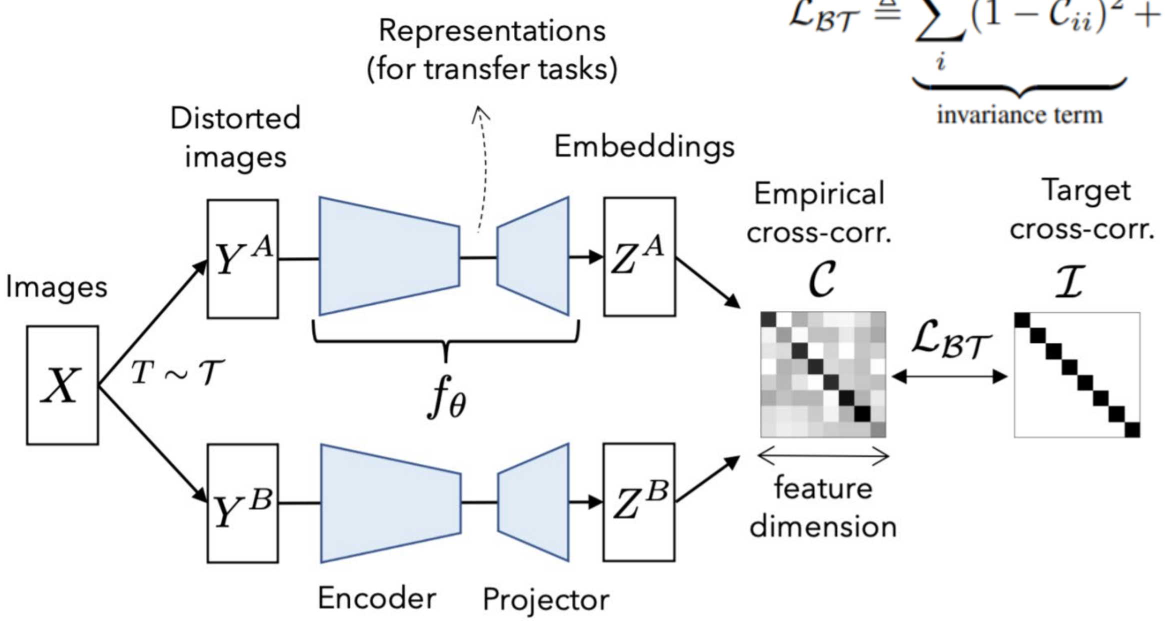


Grill et al. "Bootstrap your own latent: A new approach to self-supervised learning." NeurIPS 2020

Barlow Twins

- Enforce **diversity** among **feature dimensions**
- Maximize diagonal terms and minimize off-diagonal ones
- No need of negative pairs, predictor network, gradient stopping or moving average techniques

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} {\mathcal{C}_{ij}}^2}_{\text{redundancy reduction term}}$$



Zbontar et al. “Barlow twins: Self-supervised learning via redundancy reduction.” ICML 2021

Barlow Twins

- Experiments on classification

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised	25.4	56.4	48.4	80.4
PIRL	-	-	57.2	83.8
SimCLR	48.3	65.6	75.5	87.8
BYOL	53.2	68.8	78.4	89.0
SwAV	53.9	**70.2**	78.5	**89.9**
Barlow Twins (ours)	**55.0**	69.7	**79.2**	89.3

Barlow Twins

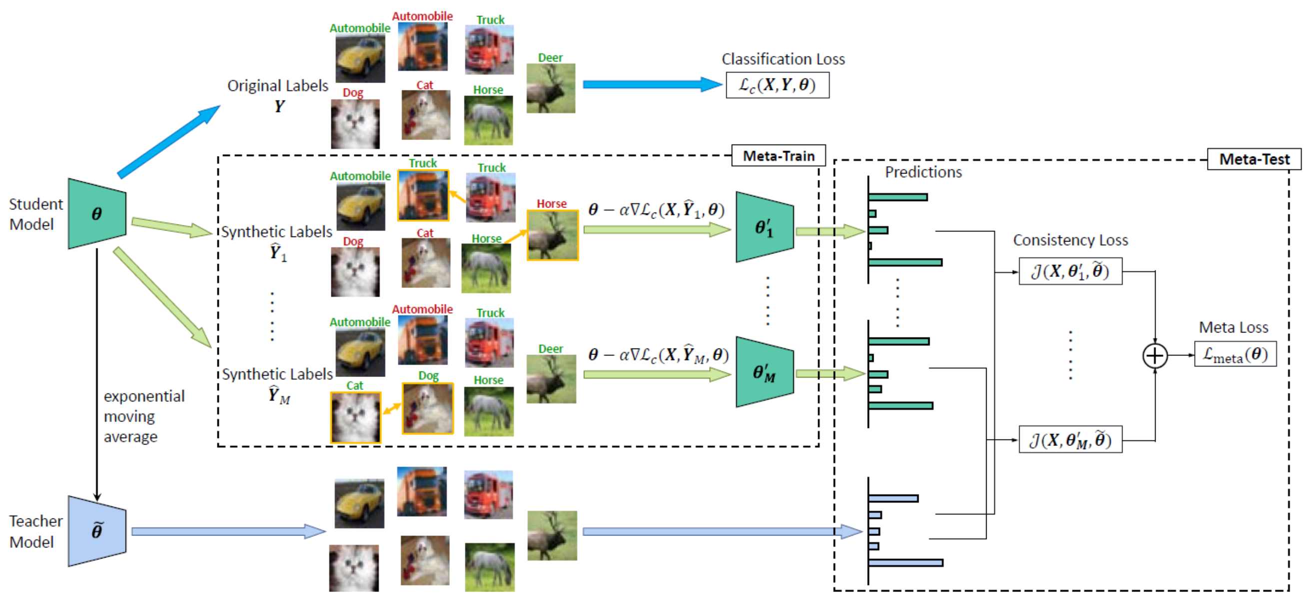
- Experiments on detection and segmentation

Method	VOC07+12 det			COCO det			COCO instance seg		
	AP_{all}	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Sup.	53.5	81.3	58.8	38.2	58.2	41.2	33.3	54.7	35.2
MoCo-v2	**57.4**	82.5	**64.0**	**39.3**	58.9	**42.5**	**34.4**	55.8	36.5
SwAV	56.1	**82.6**	62.7	38.4	58.6	41.3	33.8	55.2	35.9
SimSiam	57	82.4	63.7	39.2	**59.3**	42.1	**34.4**	**56.0**	**36.7**
BT (ours)	56.8	**82.6**	63.4	39.2	59.0	**42.5**	34.3	**56.0**	36.5

SSL Beyond Image Data

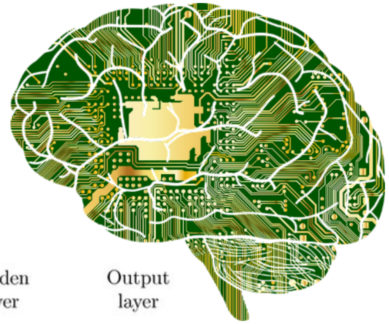
- What about videos?

- What about noisy data? J. Li et al., Learning to Learn from Noisy Labeled Data, *CVPR* 2019

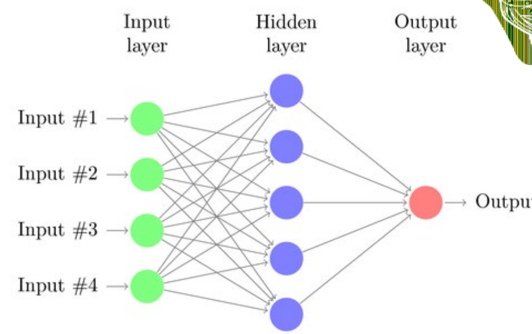


- You can come up with your own SSL strategy!

What to Cover Today…



- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
  - SSL for CNN



- Image Segmentation

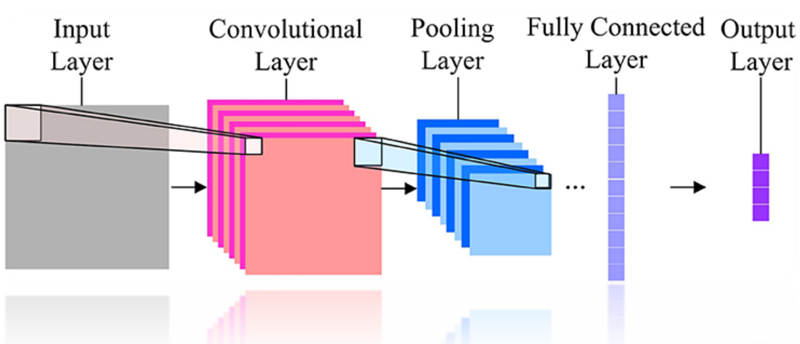
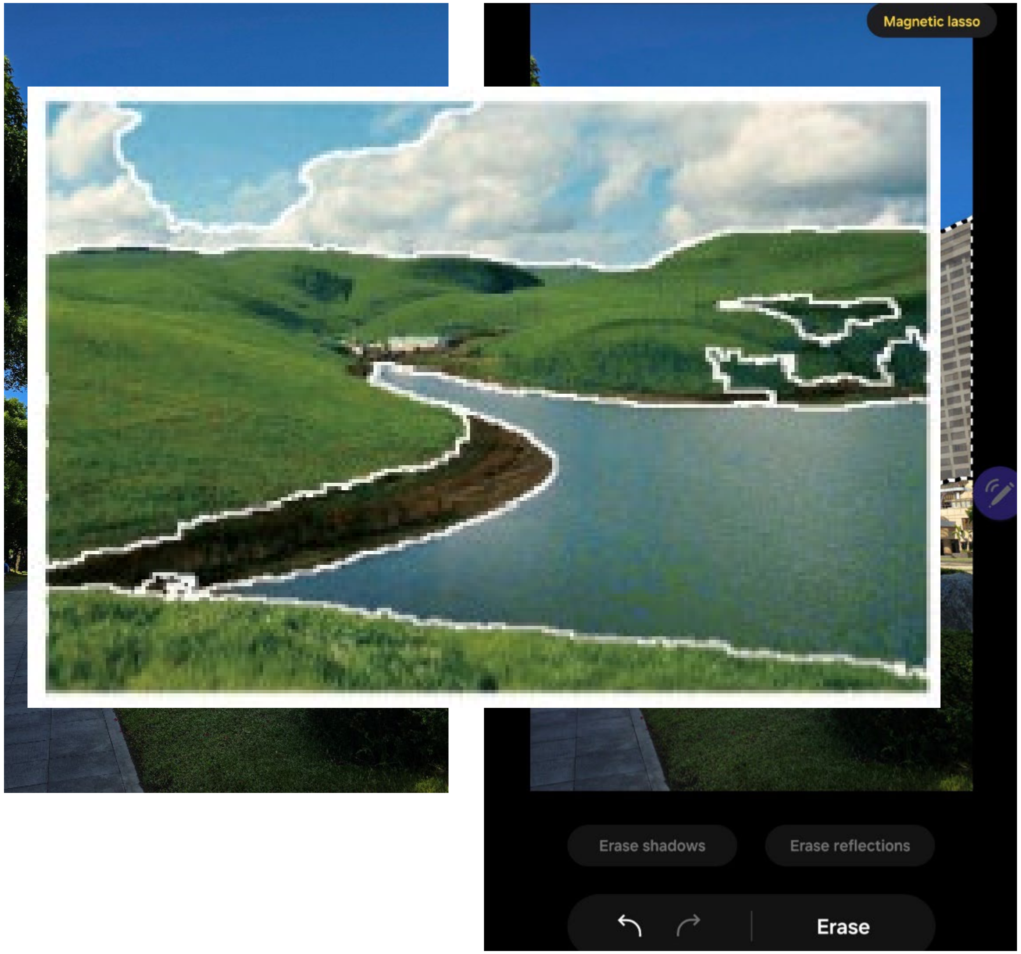
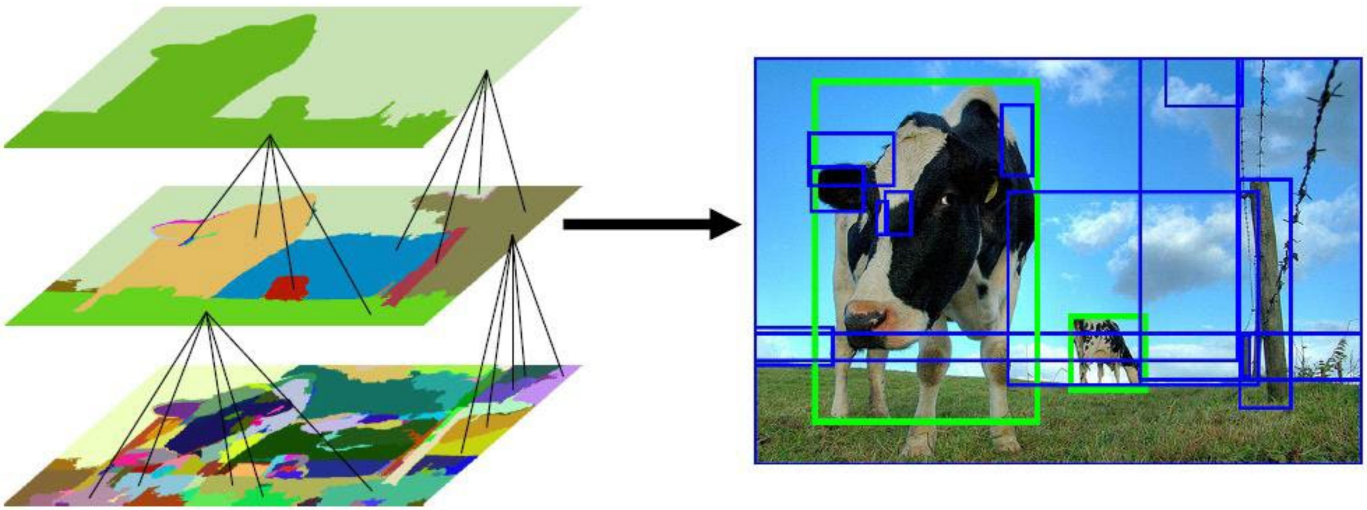


Image Segmentation

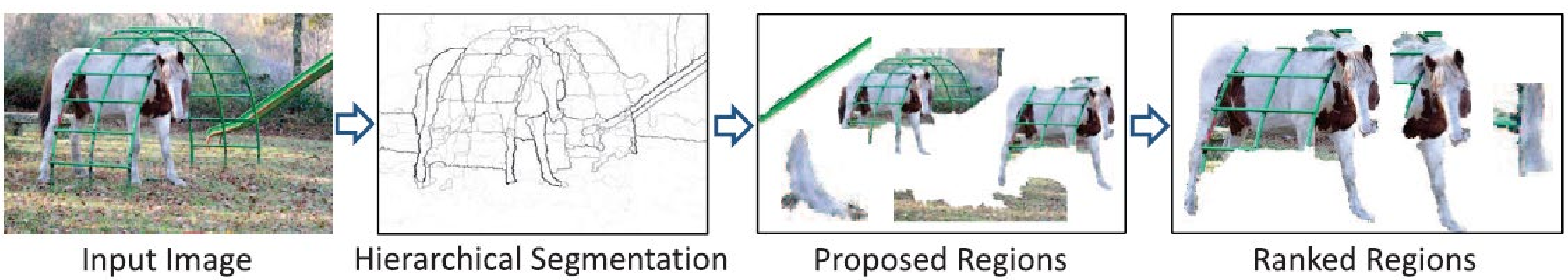
- Goal: Group pixels into meaningful or perceptually similar regions
- Any recent smart phone applications?



Segmentation for Object Proposal



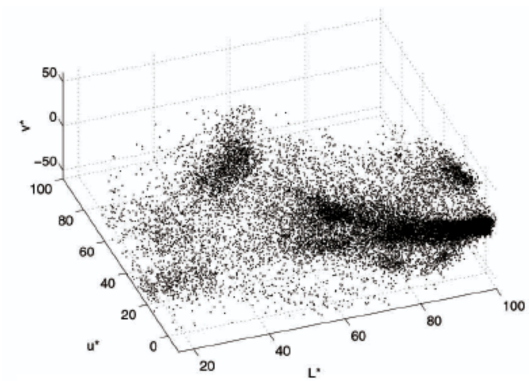
“Selective Search” [Sande, Uijlings et al. ICCV 2011, IJCV 2013]

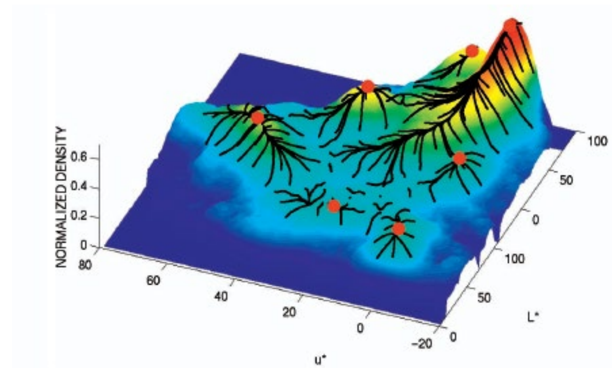


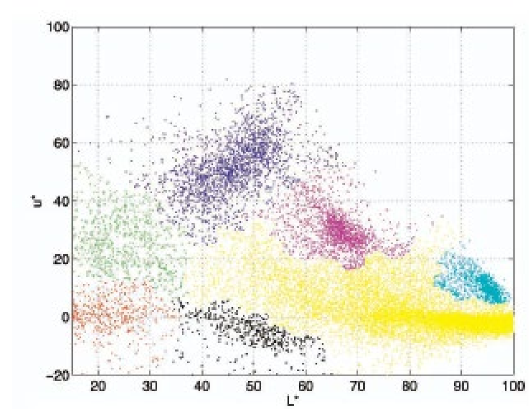
[Endres Hoiem ECCV 2010, IJCV 2014]

Segmentation via Clustering – Unsupervised Learning based Approaches

- K-means clustering -> [R, G, B, x, y] as pixel features
- Mean-shift
  - Find modes of the following non-parametric density



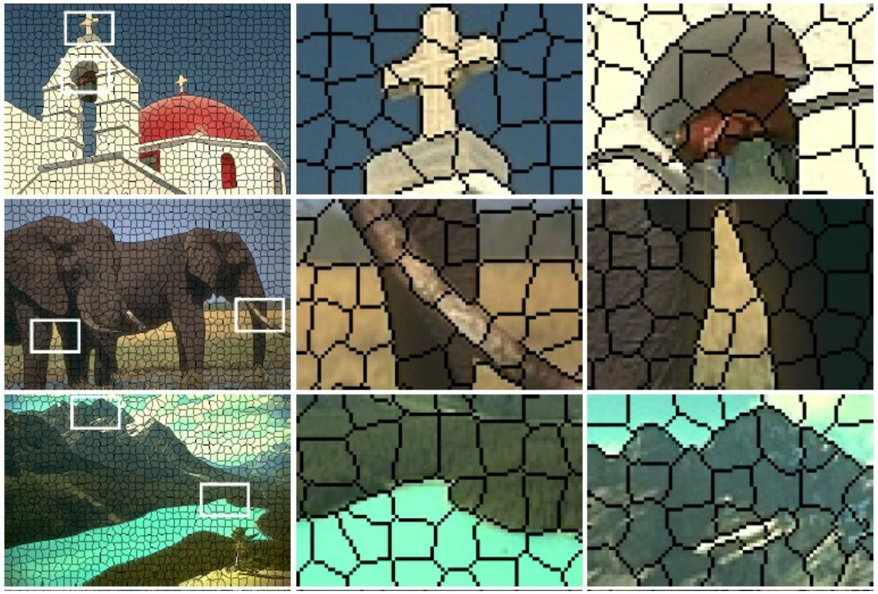




*D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, IEEE PAMI 2002.

Superpixels

- A relatively simpler task of image segmentation
- Divide an image into a large number of image regions, such that each region lies within object boundaries.
- Examples
  - Watershed
  - Felzenszwalb and Huttenlocher graph-based
  - Turbopixels
  - SLIC

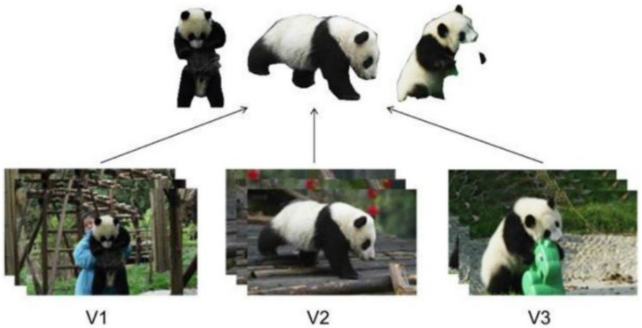


Semantic Segmentation –
Supervised Learning based Approaches

- Semantic Segmentation
  - Assign a class label to each pixel in the input image (i.e., pixel-level classification)
  - Not like instance segmentation, do not differentiate instances;
    only care about pixel labels

More Tasks in Segmentation

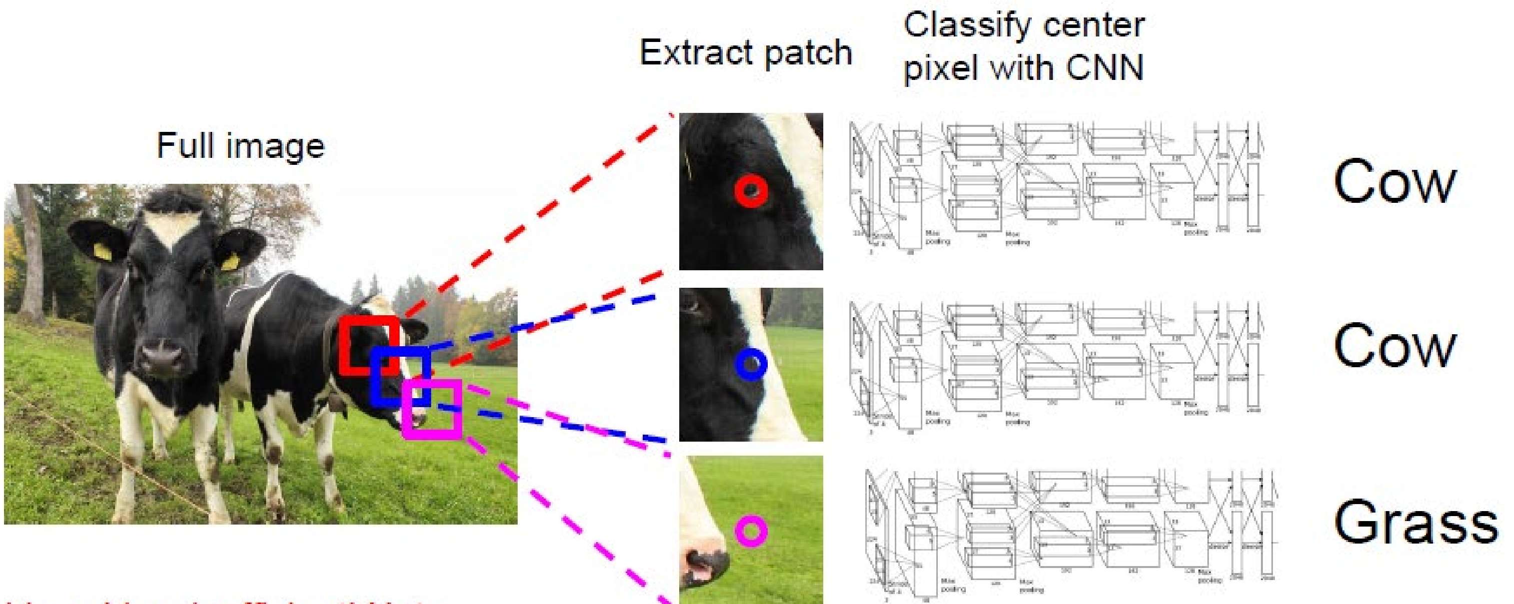
- Cosegmentation
  - Segmenting common objects from multiple images
  - Unsup. or supervised? Why preferable?



- Instance Segmentation
  - Assign a particular class label for each object instance
  - Unsuper. or supervised?

Semantic Segmentation

- Sliding Window
  - Patch or pixel-level classification
  - Any concern?

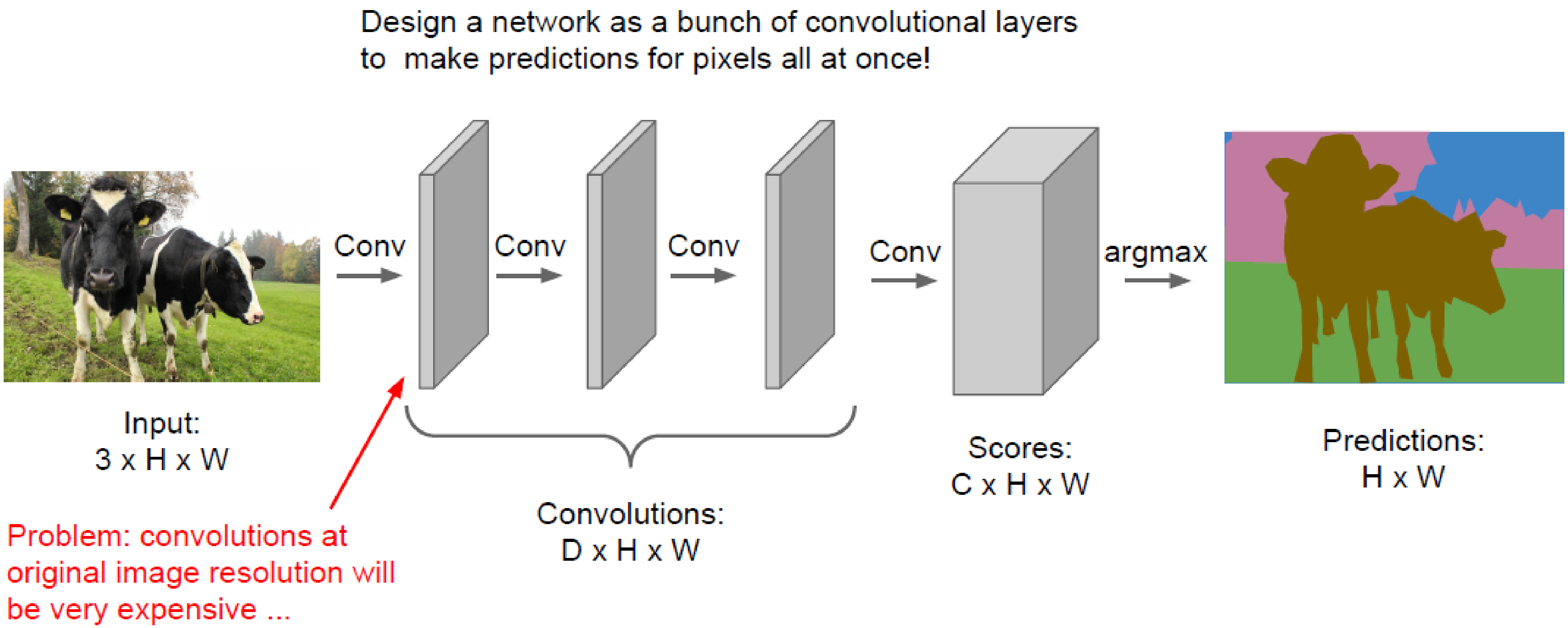


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

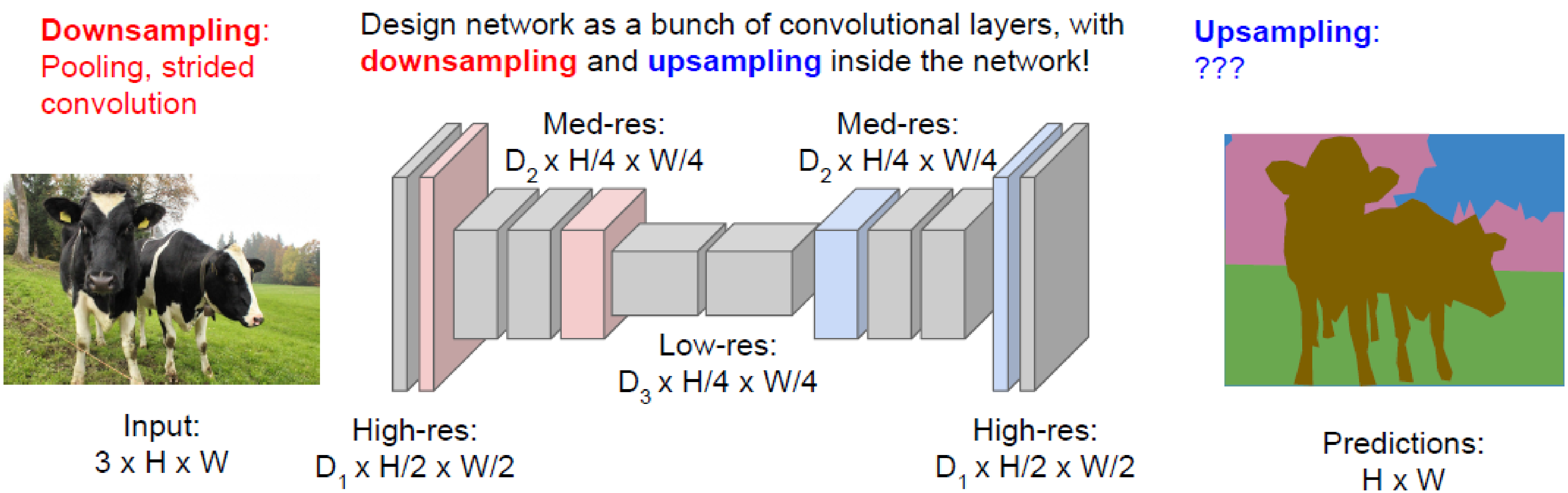
Semantic Segmentation

- Fully Convolutional Nets
  - The prediction output is a H x W map,
    which can be view as a C x H x W class-label matrix.
  - Performing pixel-level classification
    by mapping the output feature map (C x H x W) to a class-label matrix (C x H x W).



Semantic Segmentation

- Fully Convolutional Nets (cont'd)



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network Upsampling

- Unpooling

Nearest Neighbor

Input: 2 x 2

1	2
3	4

Output: 4 x 4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

“Bed of Nails”

Input: 2 x 2

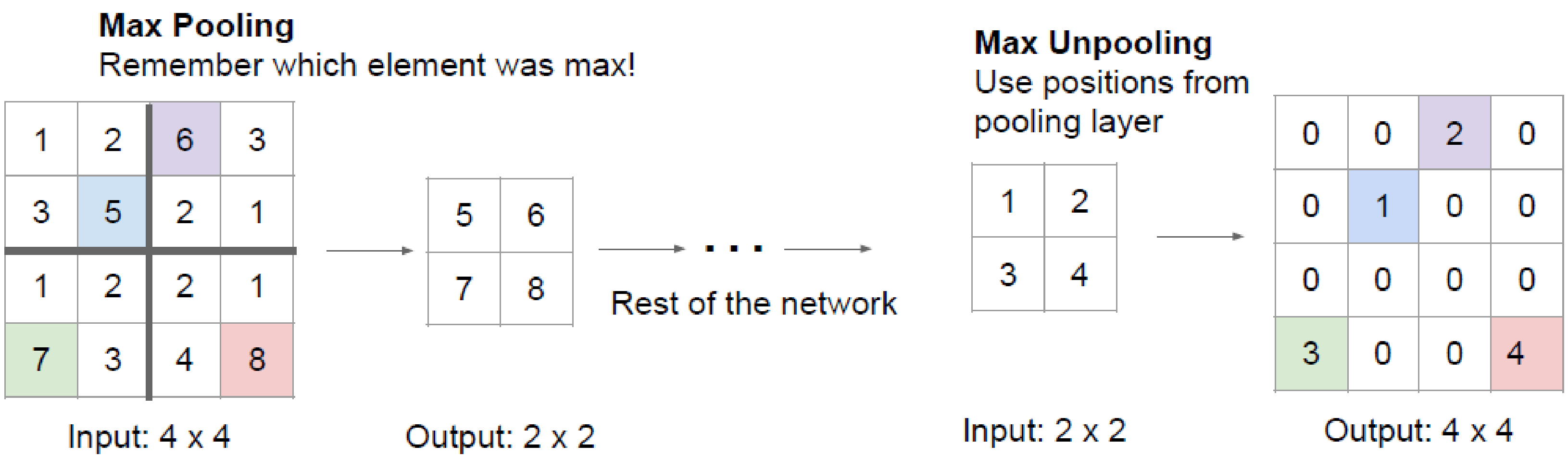
1	2
3	4

Output: 4 x 4

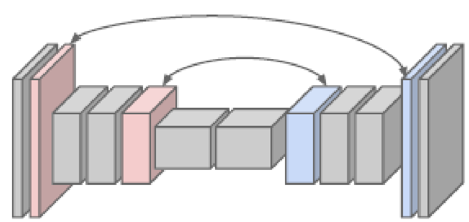
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

In-Network Upsampling

- Max Unpooling
  - What's the price to pay?



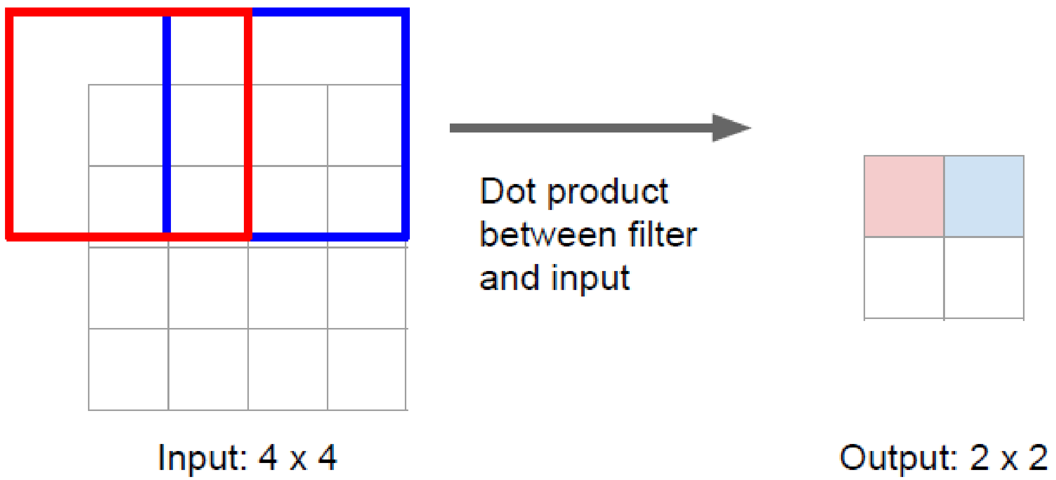
Corresponding pairs of downsampling and upsampling layers



In-Network Upsampling

- Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

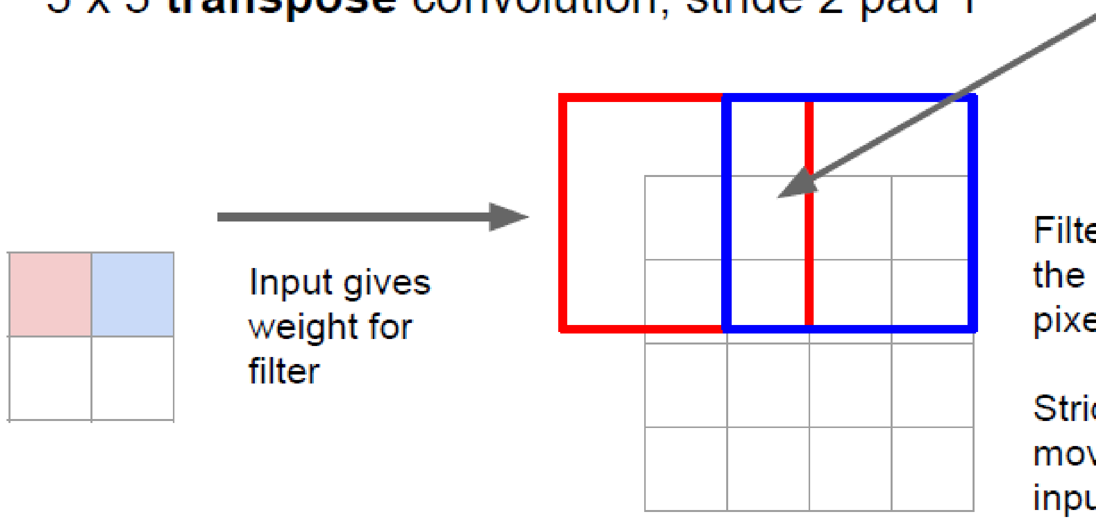
In-Network Upsampling

- *Transpose* Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1

Sum where output overlaps



Input: 2 x 2

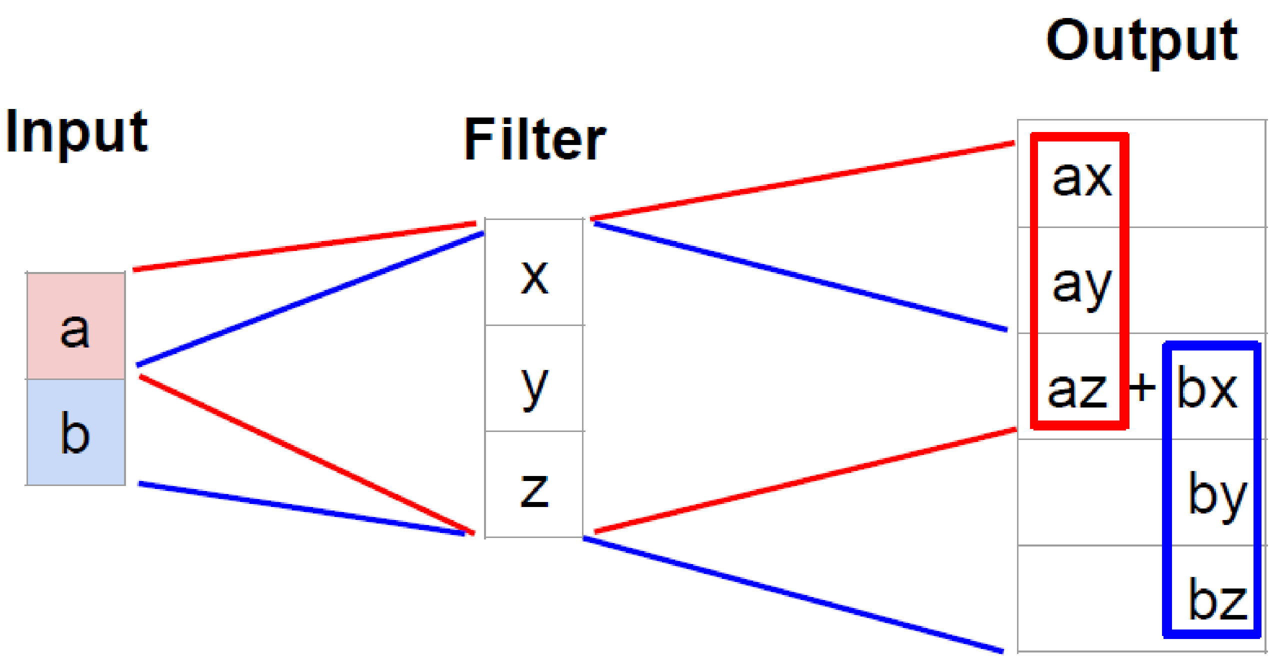
Output: 4 x 4

Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

In-Network Upsampling

- Transpose Convolution
  - See a 1D example below:



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

In-Network Upsampling

- Transpose Convolution
  - Example as matrix multiplication

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

In-Network Upsampling

- Transpose Convolution
  - Example as matrix multiplication

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

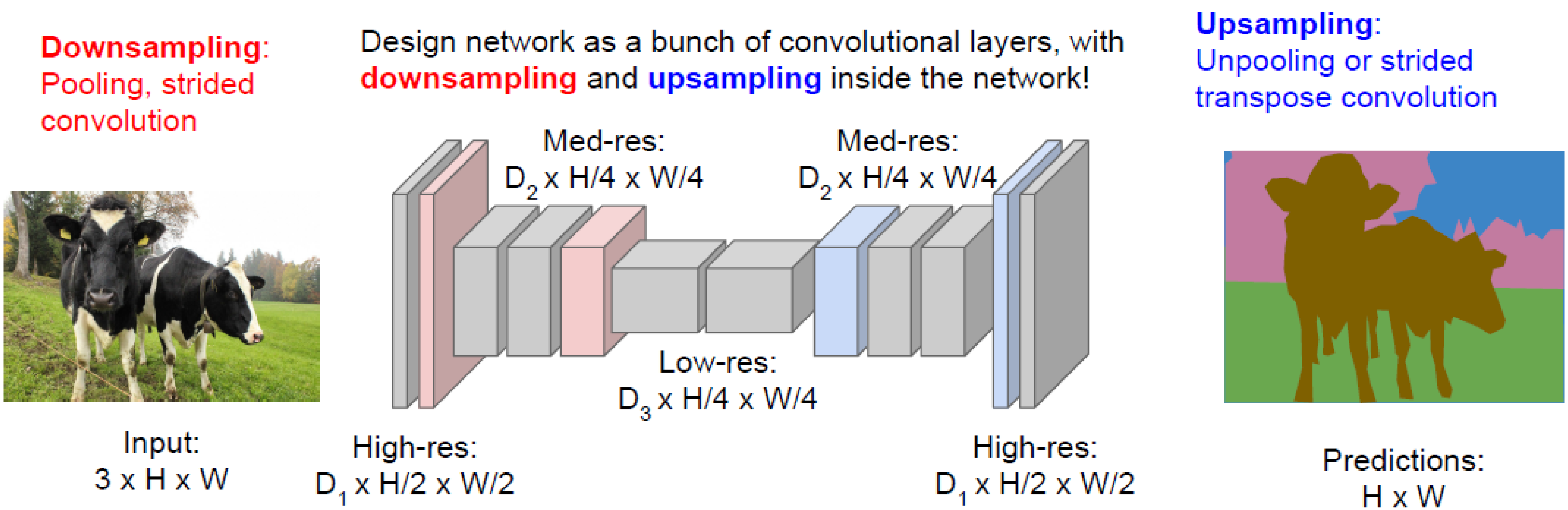
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

Fully Convolutional Networks (FCN)

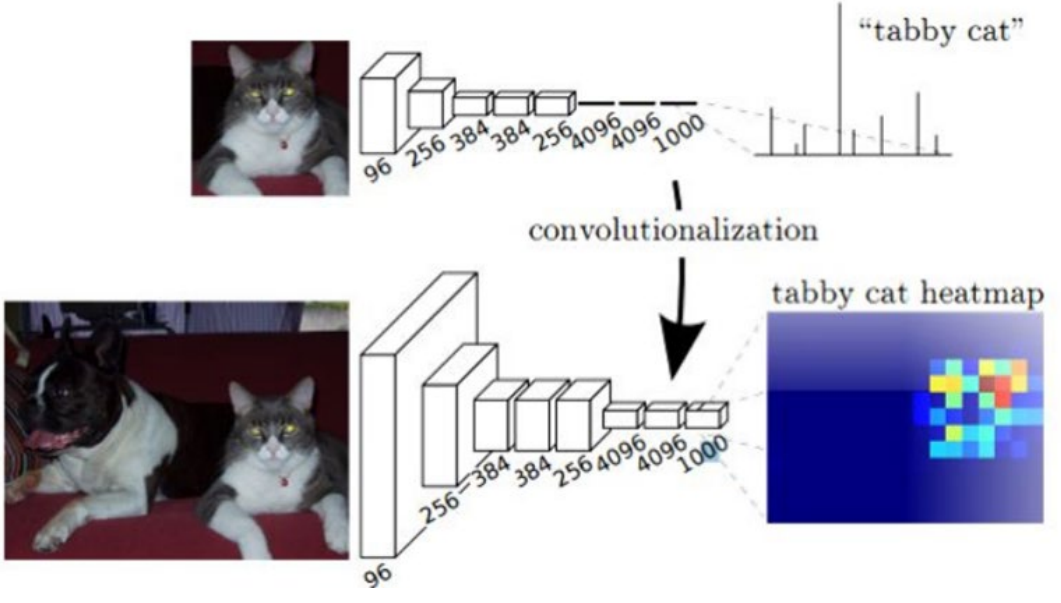
- Remarks
  - All layers are convolutional
  - End-to-end training



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Fully Convolutional Networks (FCN)

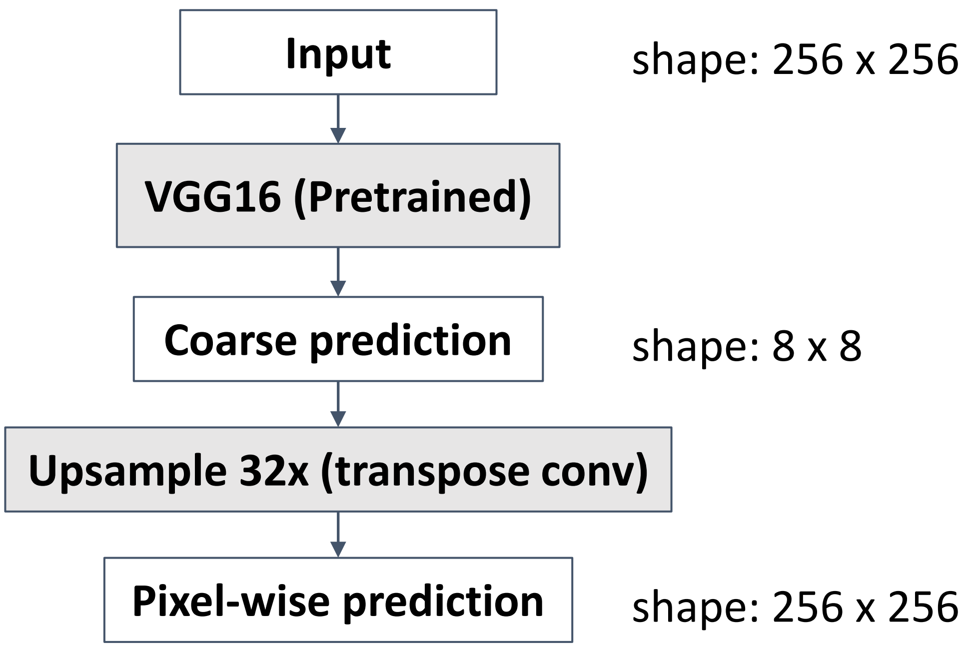
- More details
  - Adapt existing classification network to fully convolutional forms
  - Remove flatten layer and replace fully connected layers with conv layers
  - Use transpose convolution to upsample pixel-wise classification results



Fully Convolutional Networks (FCN)

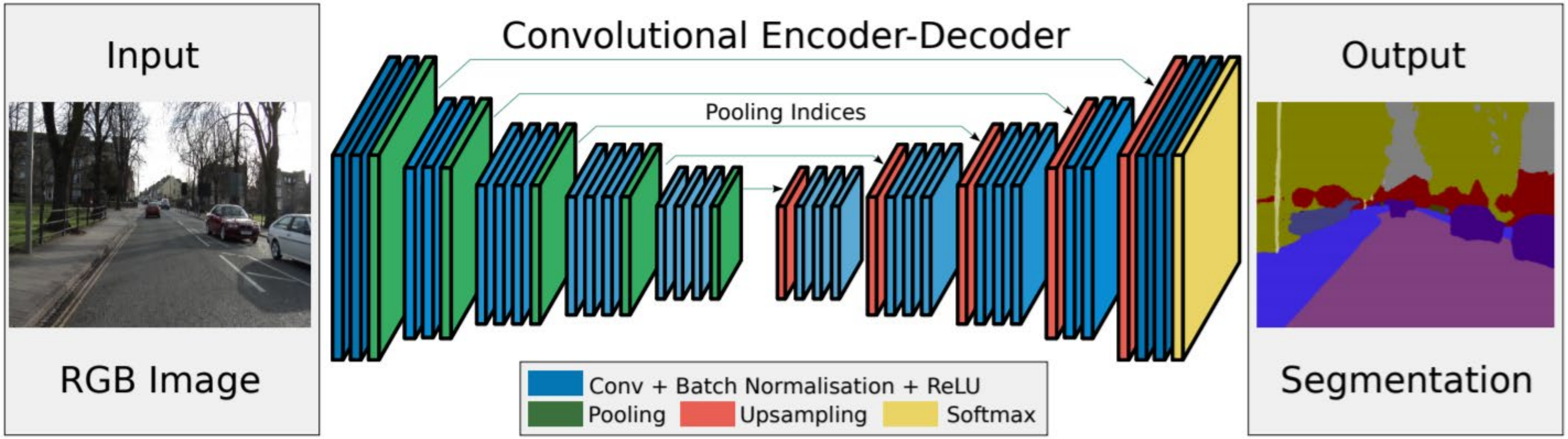
- Example
  - **VGG16-FCN32s**
  - Loss: pixel-wise cross-entropy

  i.e., compute cross-entropy between each pixel and its label, and average over all of them



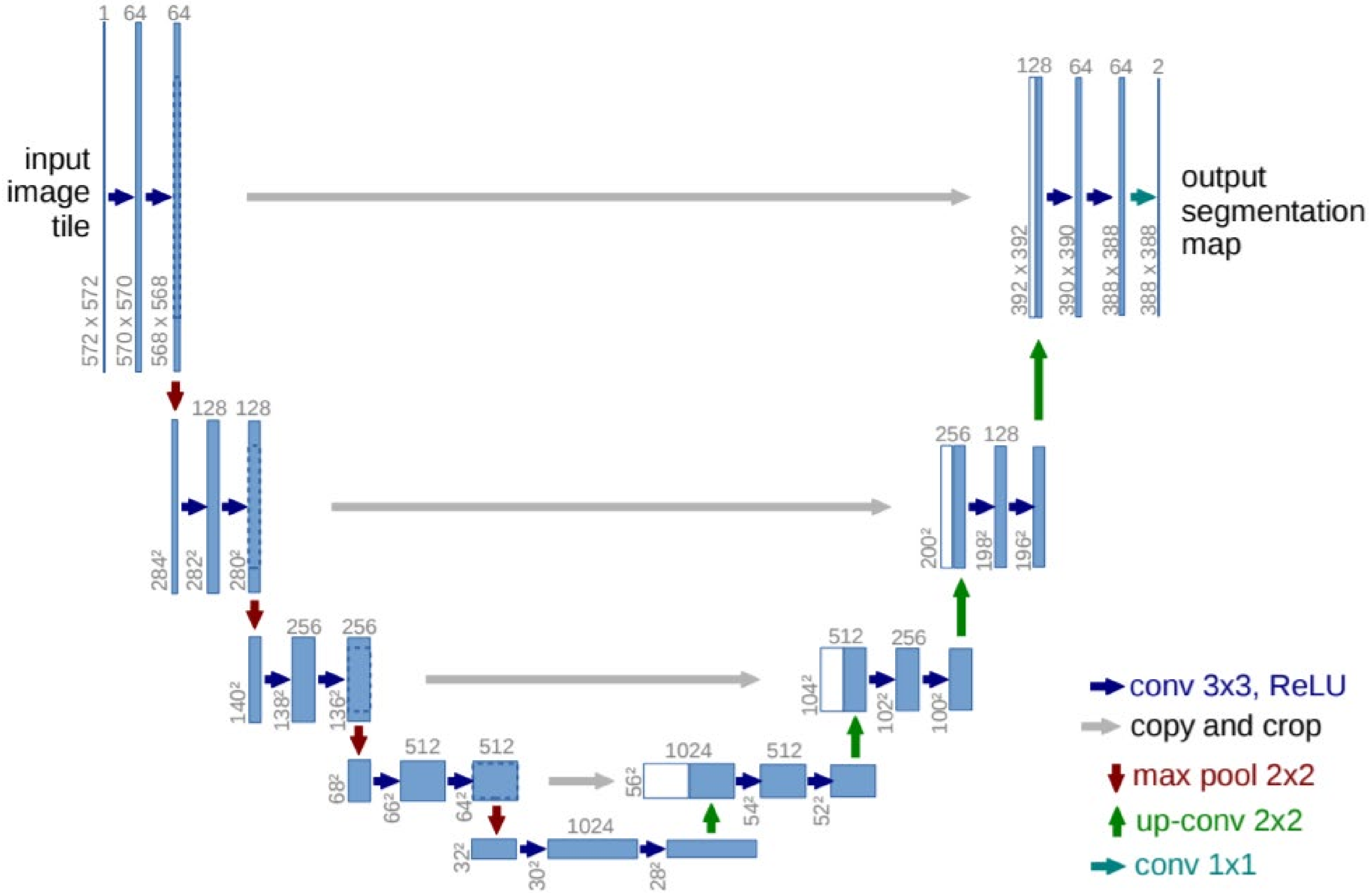
SegNet

- Efficient architecture (memory + computation time)
- Upsampling reusing max-unpooling indices
- Reasonable results without performance boosting addition
- Comparable to FCN



“SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation” [link]

U-Net

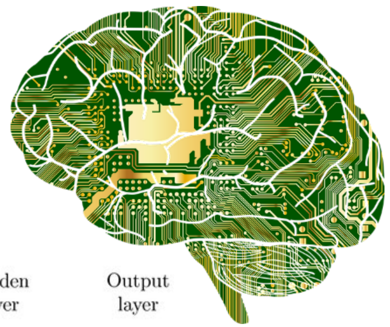


U-Net: Convolutional Networks for Biomedical Image Segmentation [link]

Additional Remarks: Enhanced Spatial Information

- For semantic segmentation, **spatial information** is of great importance
- It is desirable for the model to observe
  both the target pixel/region and its **neighboring areas**
  - Atrous (or dilated) convolution
- Features across **different scales** should be considered
  - Spatial pyramid pooling
- **Will comment on this part in future lectures (e.g., object detection)**

What We've Covered Today…



- Convolution Neural Networks (CNN)
  - Design of CNN
  - Variants of CNNs
  - Training Techniques for CNN
  - Self-Supervised Learning for CNN
- Image Segmentation
- HW #1 is out and due 9/27 Fri 23:59