Deep Learning for Computer Vision

Fall 2022

https://cool.ntu.edu.tw/courses/189345 (NTU COOL)

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

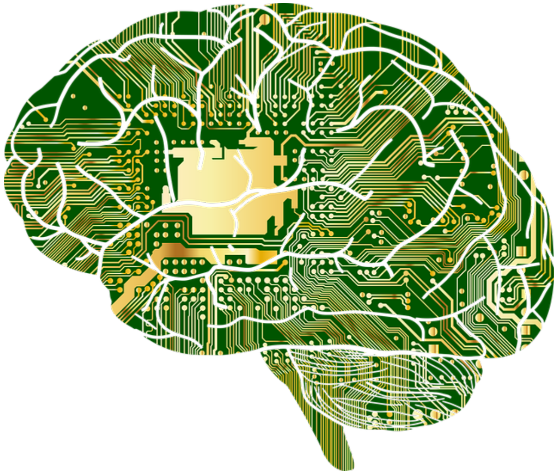
Yu-Chiang Frank Wang 王鈺強, Professor

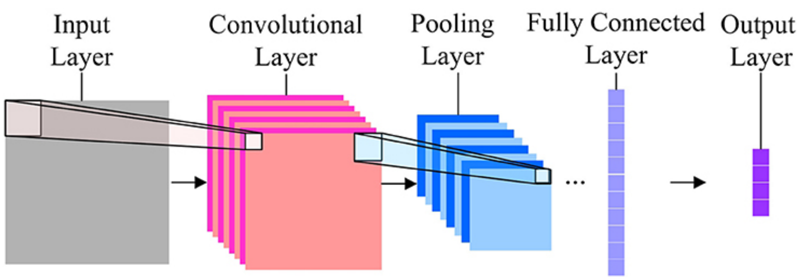
Dept. Electrical Engineering, National Taiwan University

2022/09/20

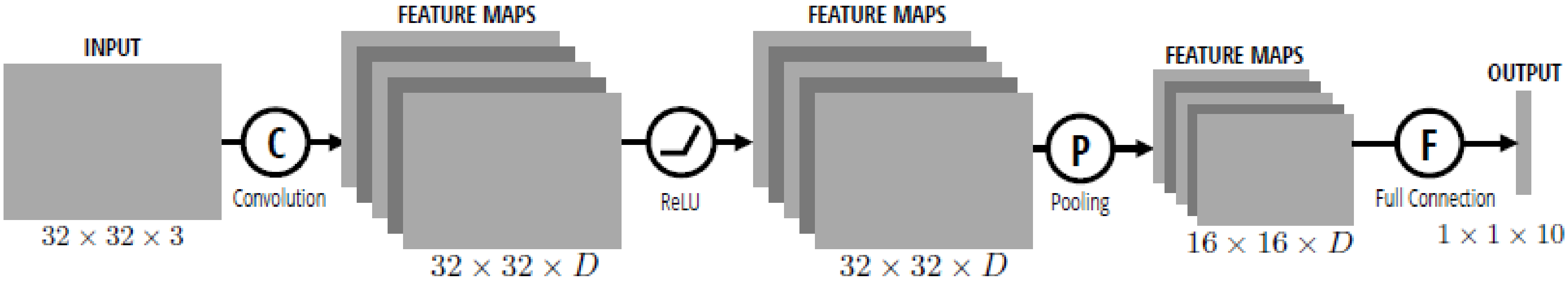
What's to Be Covered Today…

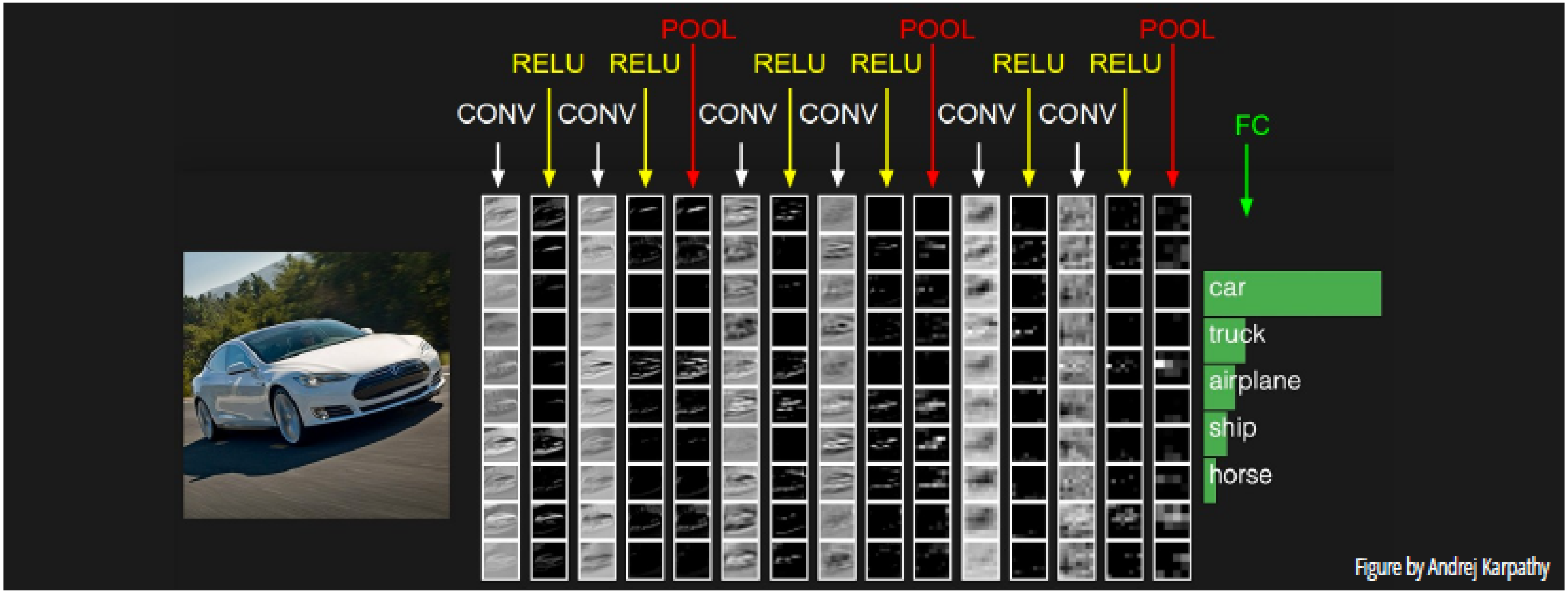
- Convolutional Neural Networks
  - Properties of CNN
  - Selected variants of CNN
  - Training CNN
  - Visualizing CNN
- Segmentation
- HW #1 is out & due Oct. 10^{th} Mon 23:59





CNN

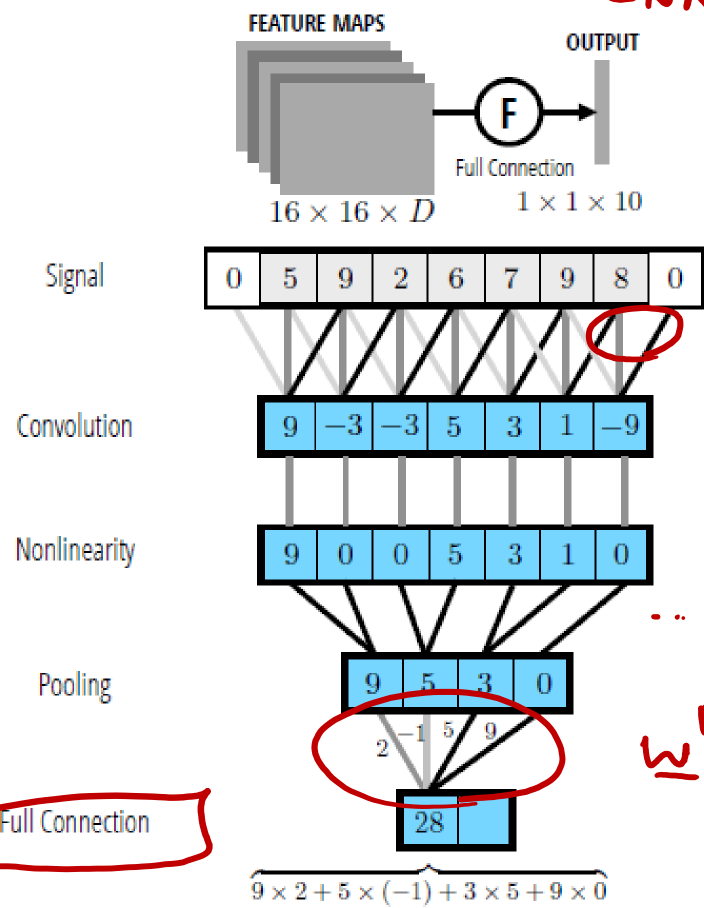

INPUT
$32 \times 32 \times 3$
C
Convolution
FEATURE MAPS
$32 \times 32 \times D$
ReLU
FEATURE MAPS
$32 \times 32 \times D$
P
Pooling
FEATURE MAPS
$16 \times 16 \times D$
F
Full Connection
OUTPUT
$1 \times 1 \times 10$


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

FC Layer

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

CNN: $\underbrace{\{ \underline{w}^1, \underline{w}^2, \dots, \underline{w}^{MLP} \}}_{\underline{w}}$



Kernel w_i^1 ← hyper para.

w_i^2

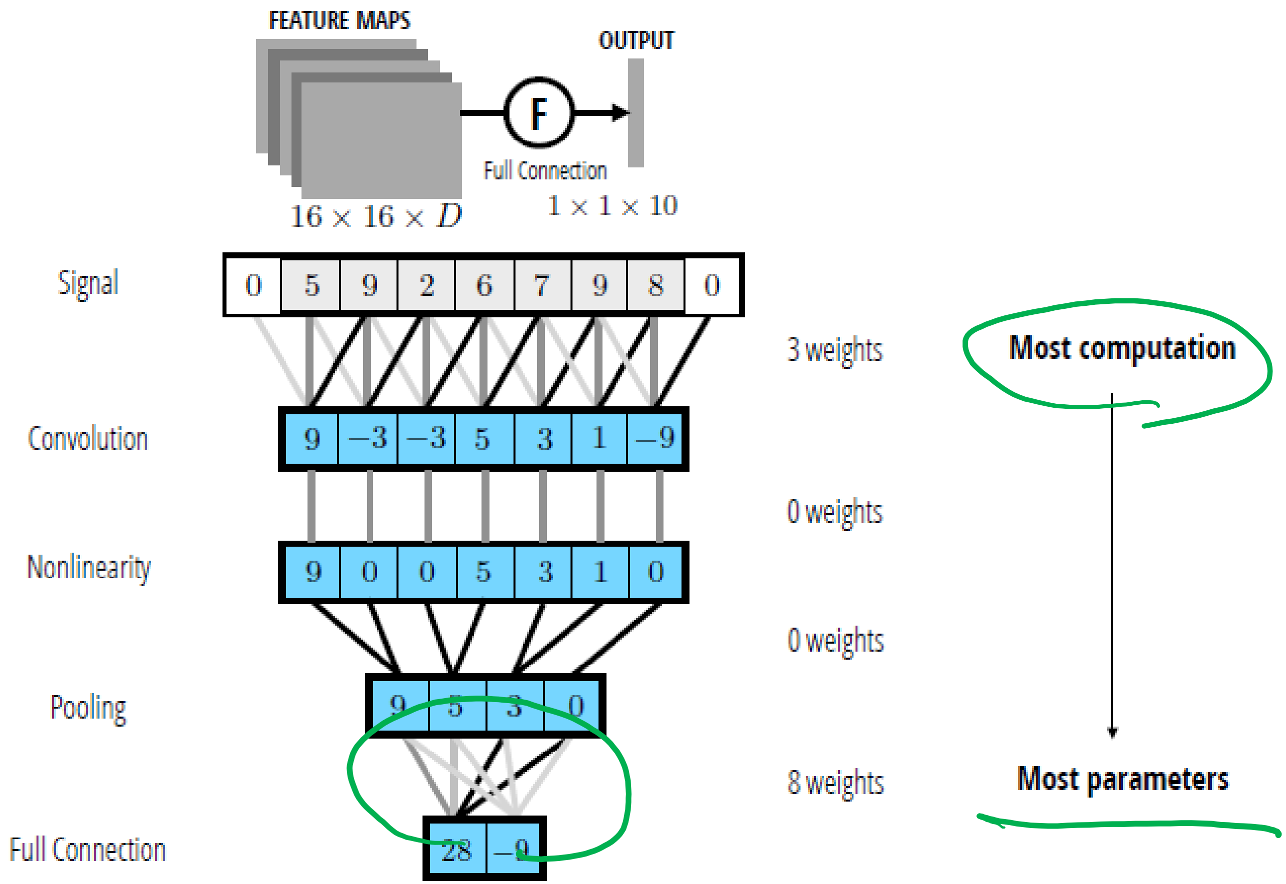
w_i^3

$\underline{w}^{MLP}$

MLP

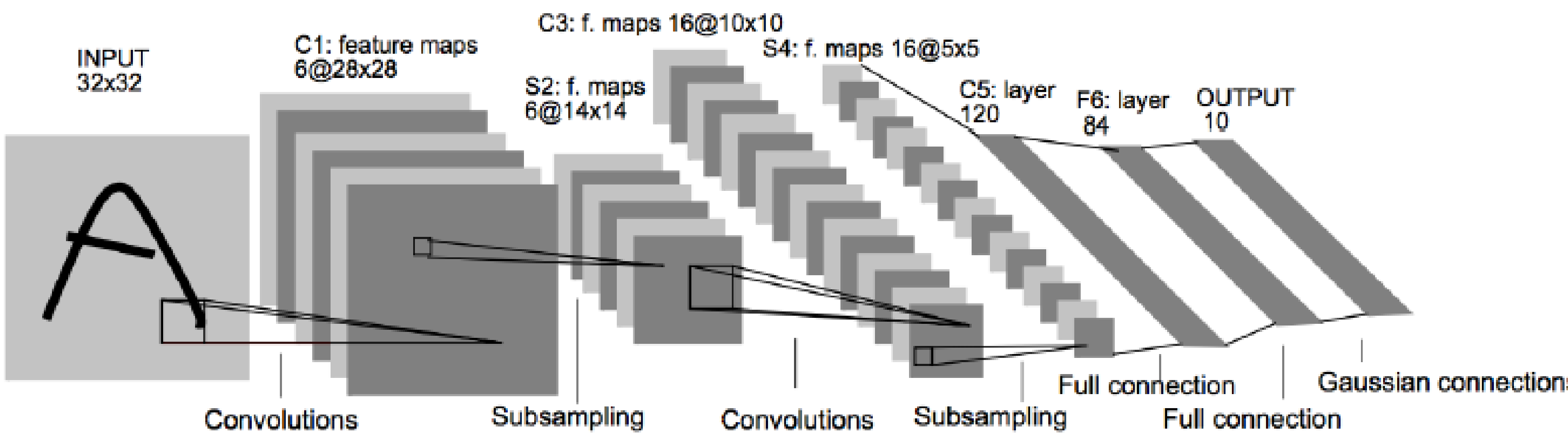
FC Layer

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

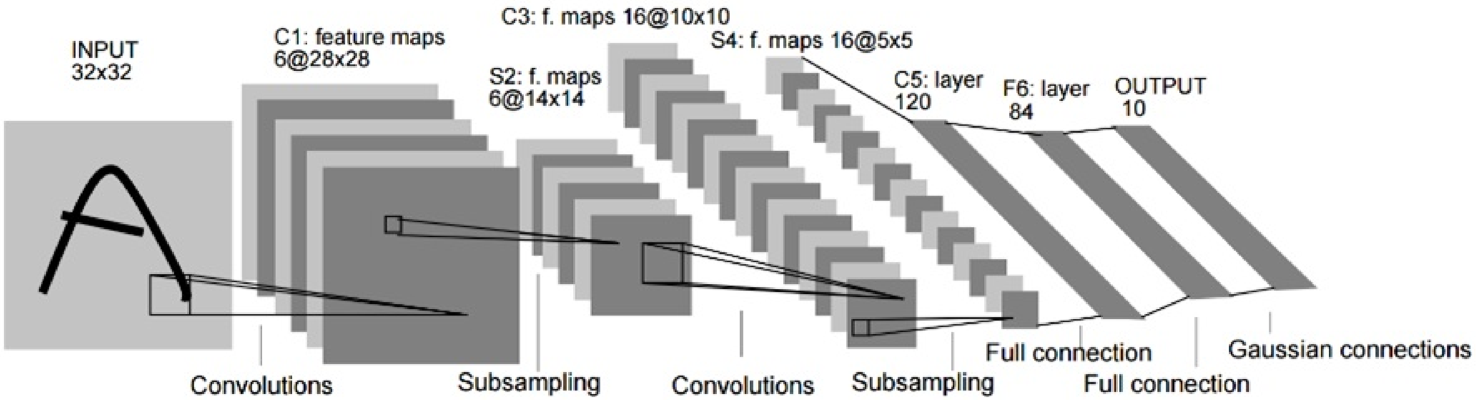


LeNet

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



LeNet [LeCun et al. 1998]

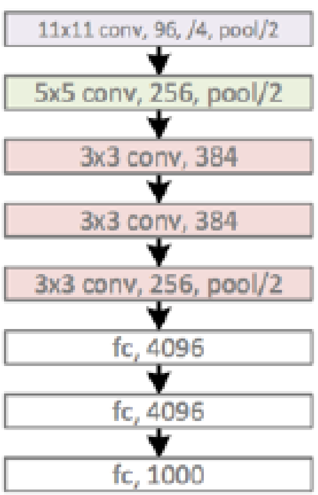


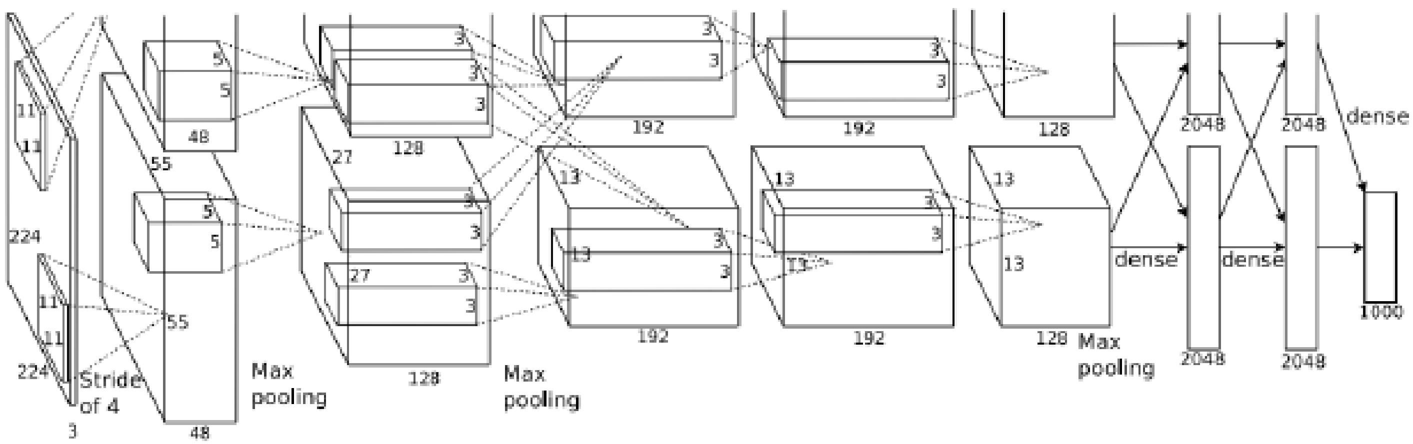
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

AlexNet [Krizhevsky et al., 2012]

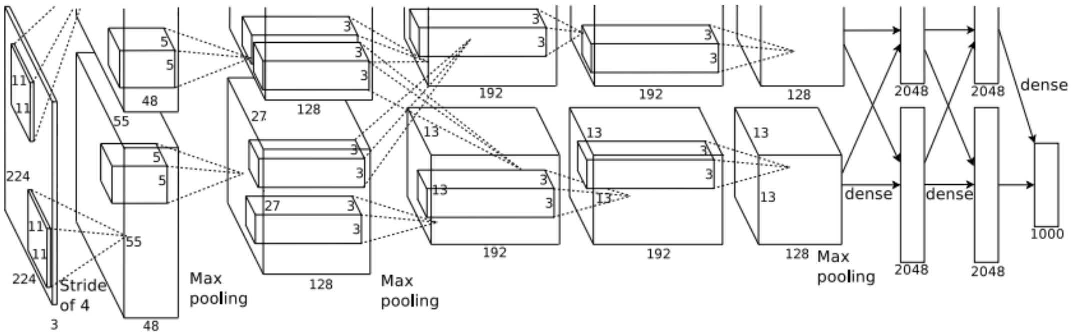
- Repopularized CNN
  by winning the ImageNet Challenge 2012
- 7 hidden layers, 650,000 neurons,
  60M parameters
- Error rate of 16% vs. 26% for 2nd place.

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)

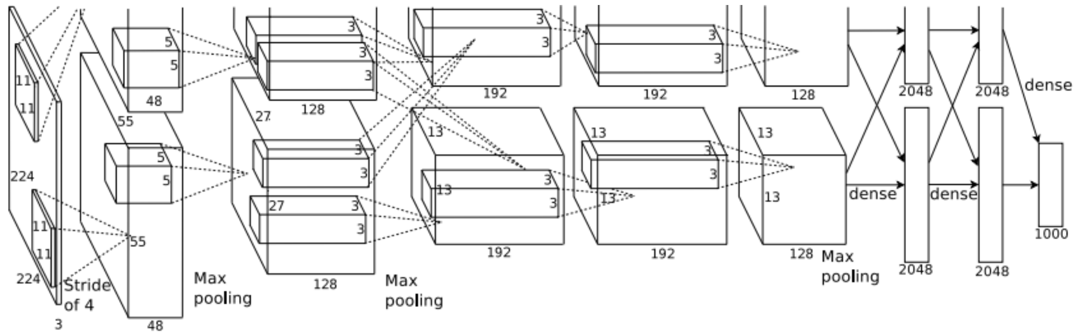
KB = (number of elements) * (bytes per elem) / 1024

= 200704 * 4 / 1024

= **784**

Slide credit: UMich EECS 498-007

of Hyperparameters in AlexNet (cont'd)

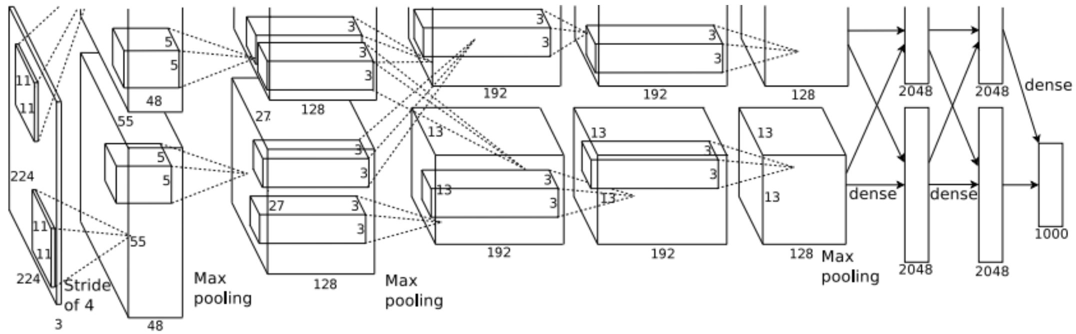


	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

$$\text{Weight shape} = C_{out} \times C_{in} \times K \times K$$
$$= 64 \times 3 \times 11 \times 11$$
$$\text{Bias shape} = C_{out} = 64$$
$$\text{Number of weights} = 64*3*11*11 + 64$$
$$= \mathbf{23{,}296}$$

Slide credit: UMich EECS 498-007

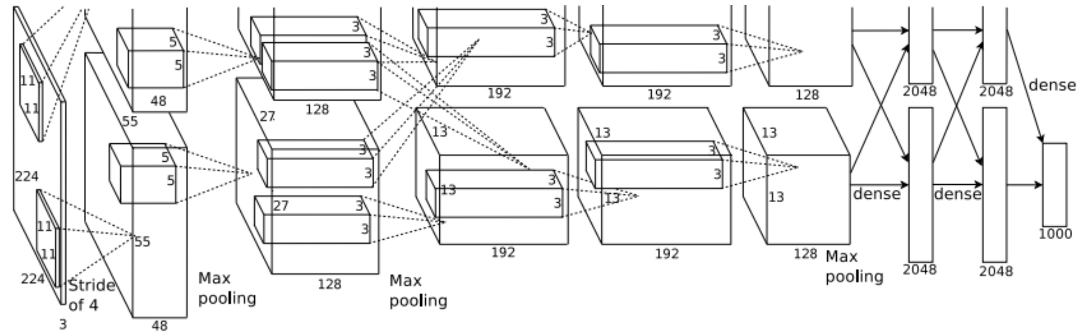
of Hyperparameters in AlexNet (cont'd)



	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

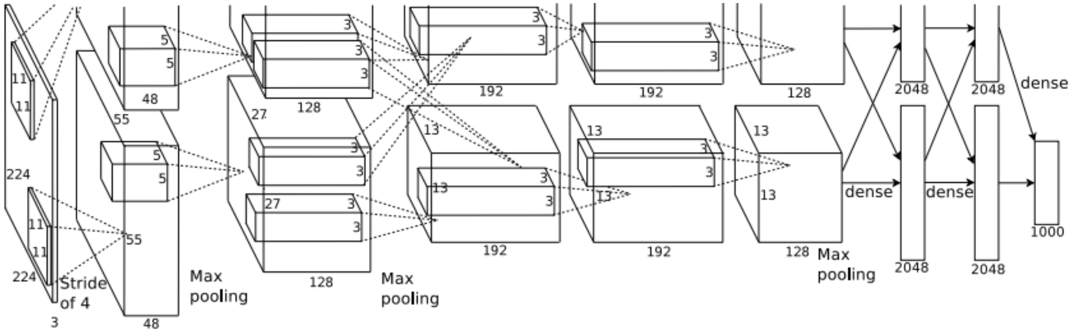
Number of floating point operations (multiply+add)
= (number of output elements) * (ops per output elem)
= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$
= (64 * 56 * 56) * (3 * 11 * 11)
= 200,704 * 363
= **72,855,552**

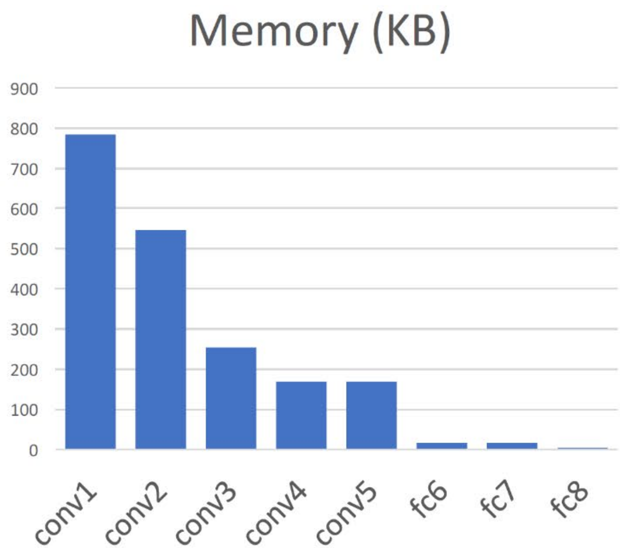
of Hyperparameters in AlexNet (cont'd)

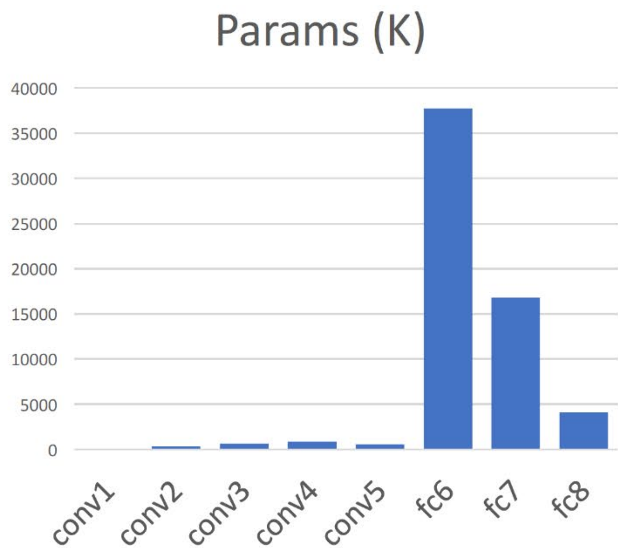


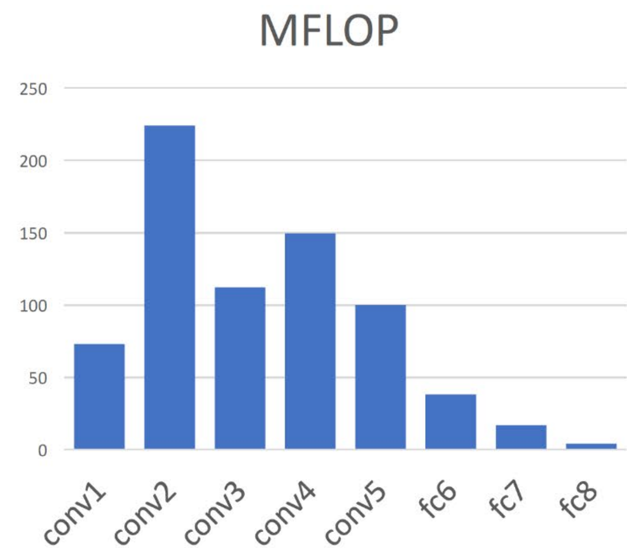
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	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

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

Slide credit: UMich EECS 498-007

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!

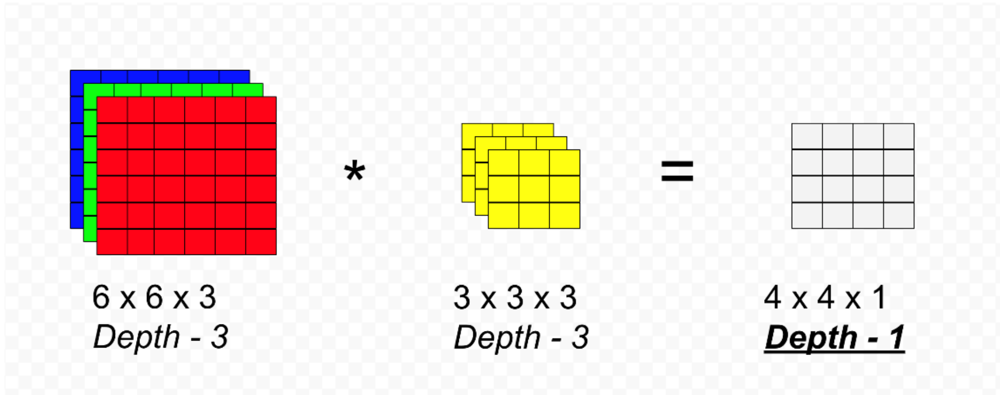
Removing Layer 6 and 7 reduces 50 million parameters, but only 5.7% drop in performance

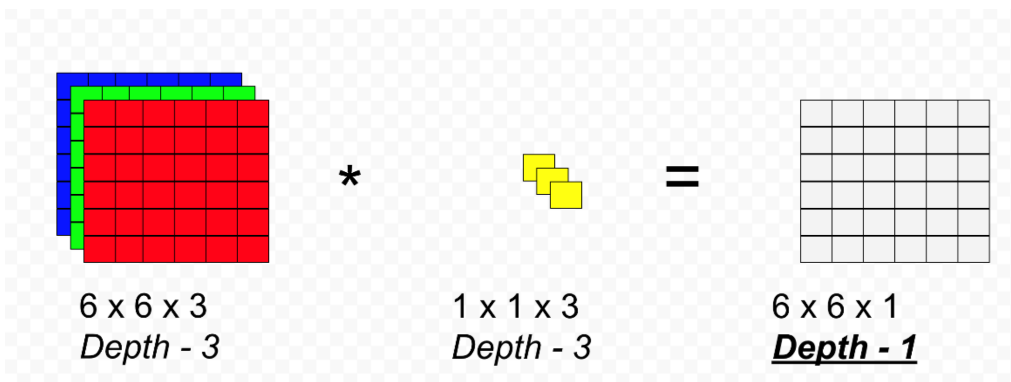
Removing middle conv layers reduces 1 million parameters, but only 3% drop in performance

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

Btw, what is 1x1 Convolution?

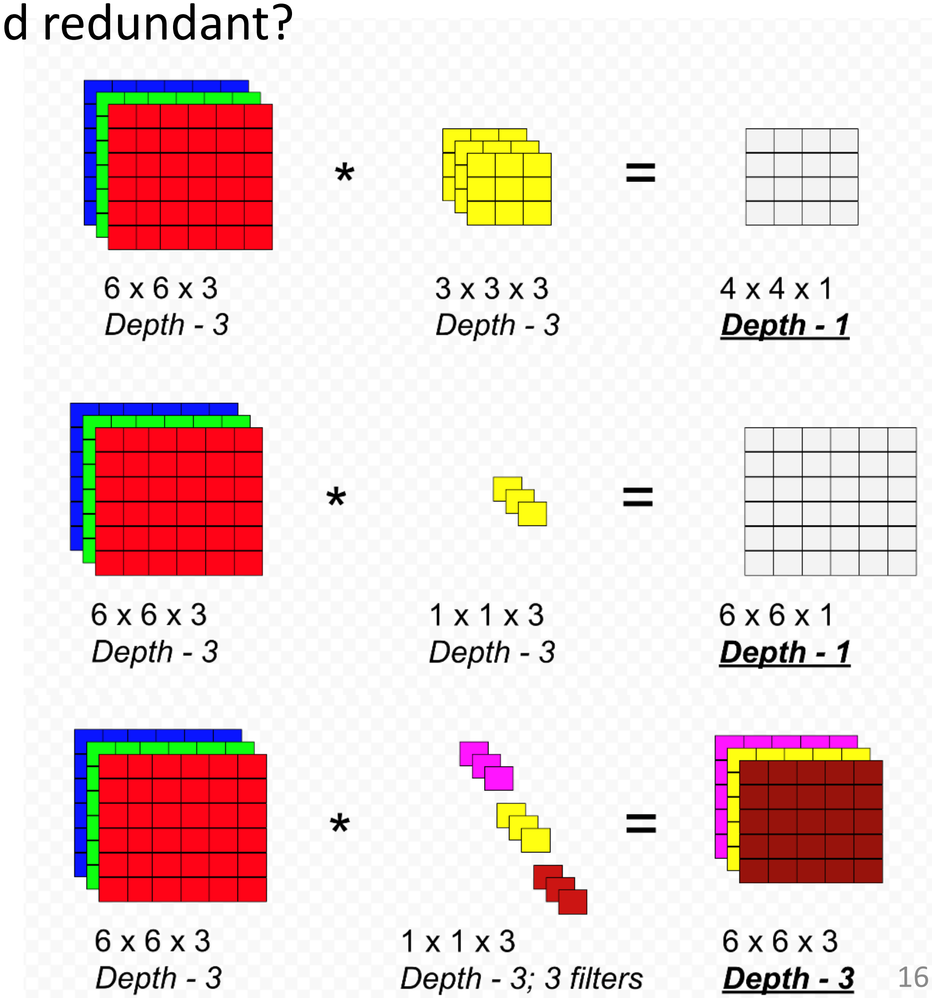
- Doesn't 1x1 convolution sound redundant?





What is 1x1 Convolution? (cont'd)

- Doesn't 1x1 convolution sound redundant?
- Simply speaking, it allows…
  - DR:
  - NL:



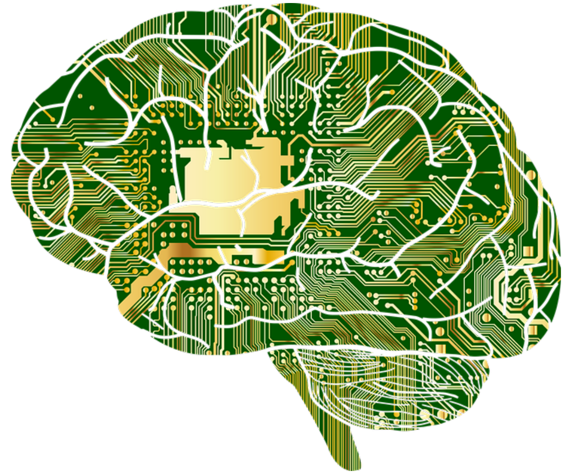
What is 1x1 Convolution? (cont'd)

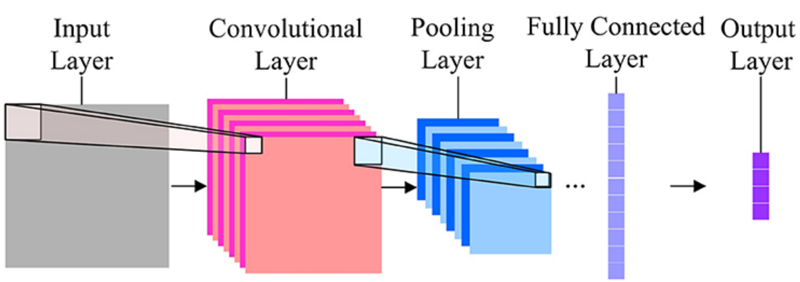
- **Example 1**
  {28 x 28 x 192} convolved with 32 {5 x 5 x 192} kernels into **{28 x 28 x 32}**
- (5 x 5 x 192) muls x (28 x 28) pixels x 32 kernels ~ 120M muls

- **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 vs. 120M

What’s to Be Covered Today…

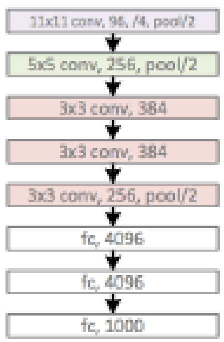
- Convolutional Neural Networks
    - Properties of CNN
    - Selected variants of CNN
    - Training CNN
    - Visualizing CNN
- Segmentation



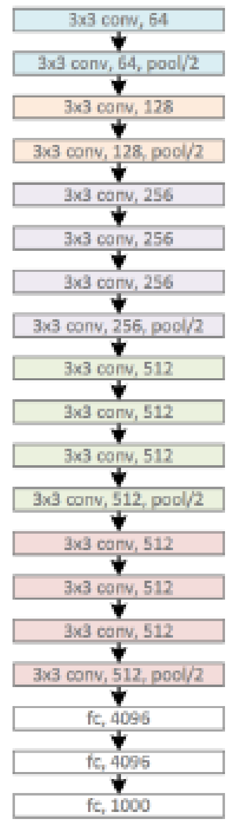


CNN: A Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



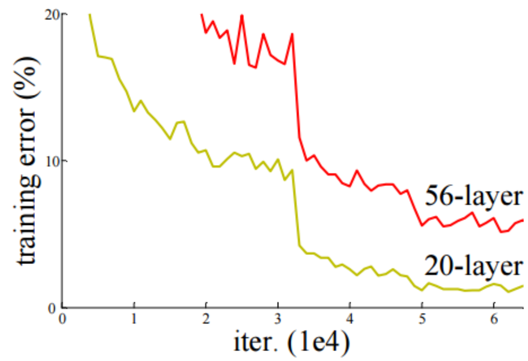
VGG, 19 layers
(ILSVRC 2014)

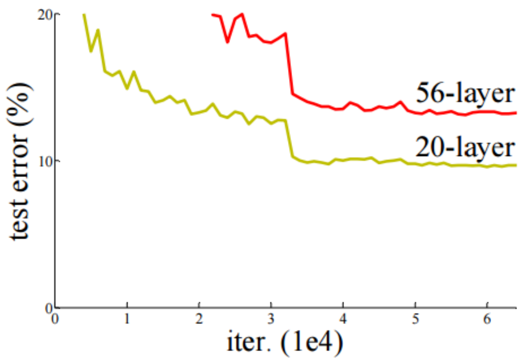


GoogleNet, 22 layers
(ILSVRC 2014)

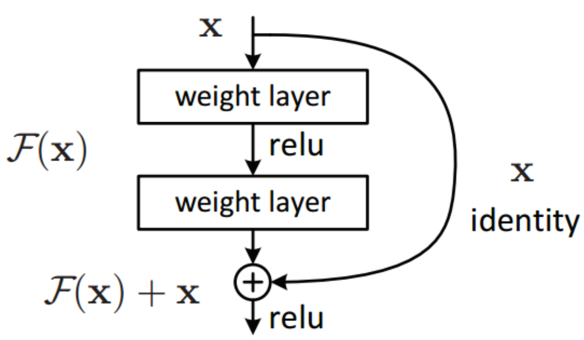
ResNet

- Can we just increase the #layer?





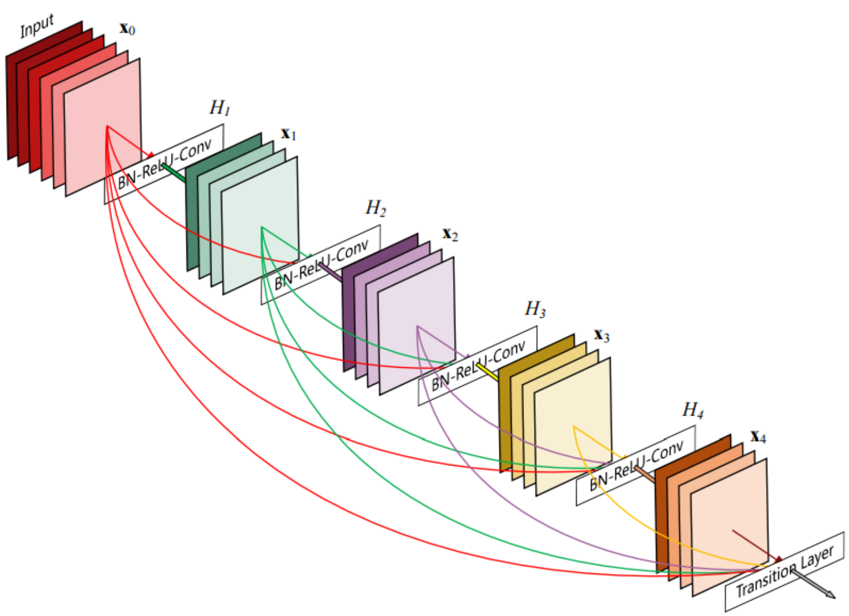
- How can we train very deep network?
  - Residual learning

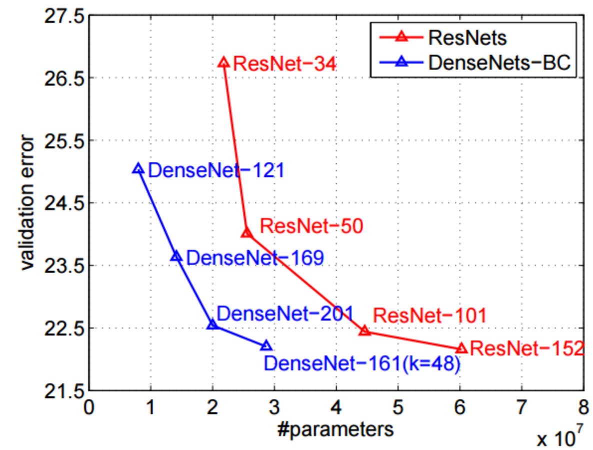


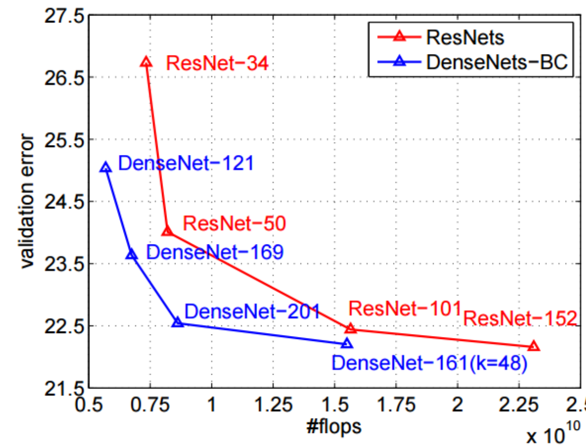
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

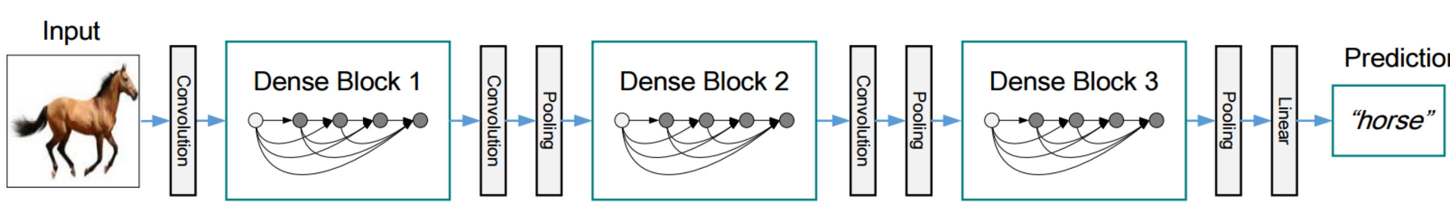
DenseNet

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

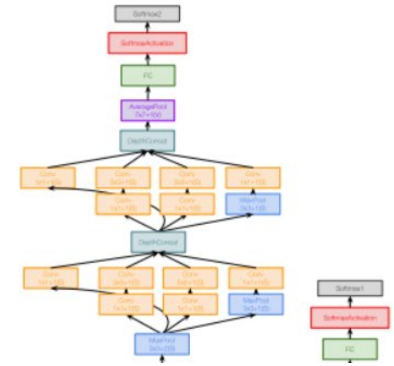






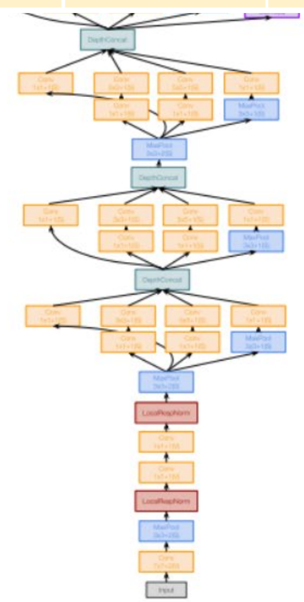


GoogleNet



- Focus on Efficiency

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



- Aggressively downsample the input

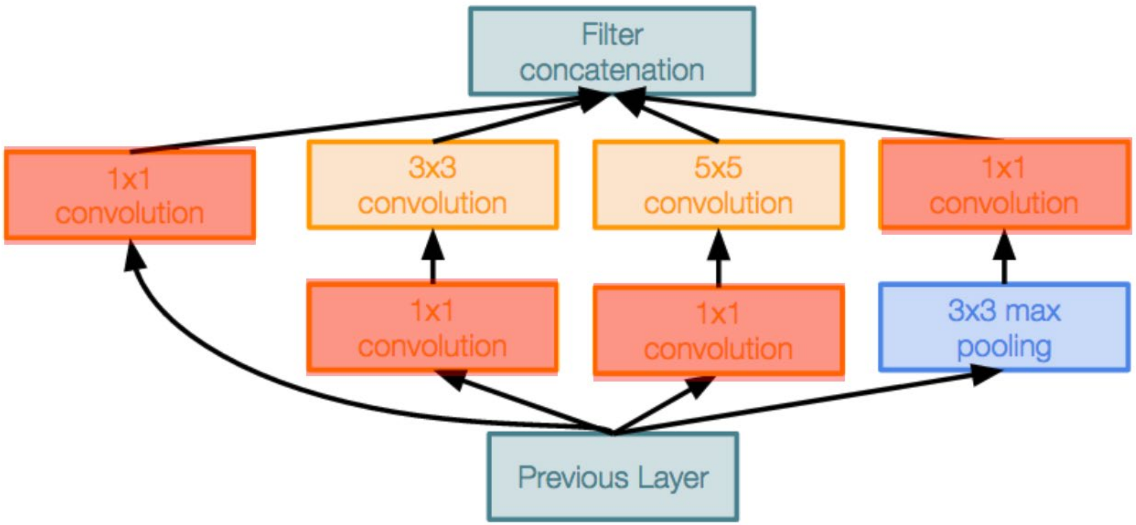
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

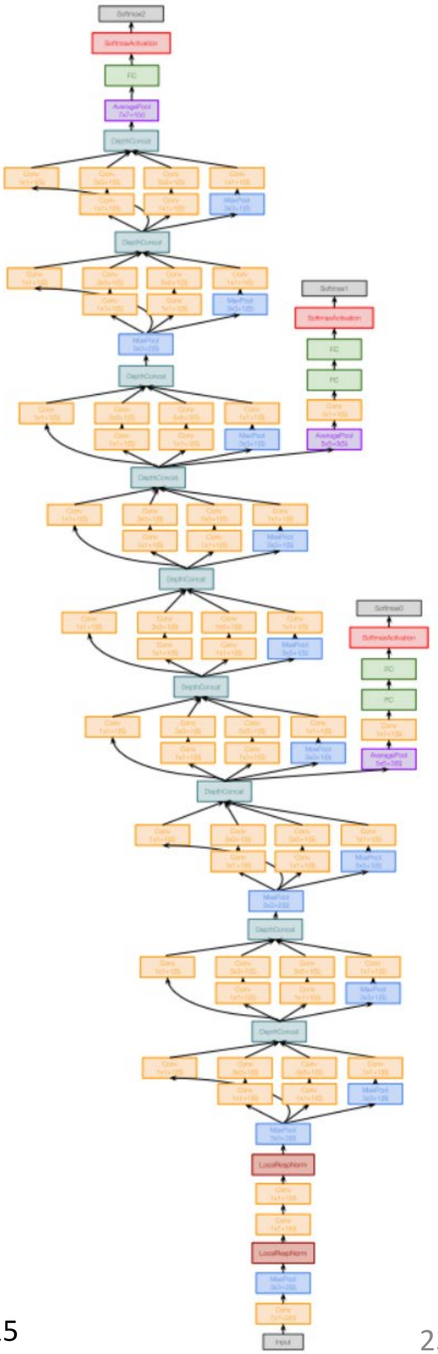
Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

Szegedy et al., Going Deeper with Convolutions, CVPR 2015

GoogleNet (cont'd)

- Inception Module
  - Local units with parallel branches
  - Repeat multiple times
  - Use 1x1 bottleneck layers to reduce channel dims





Szegedy et al., Going Deeper with Convolutions, CVPR 2015

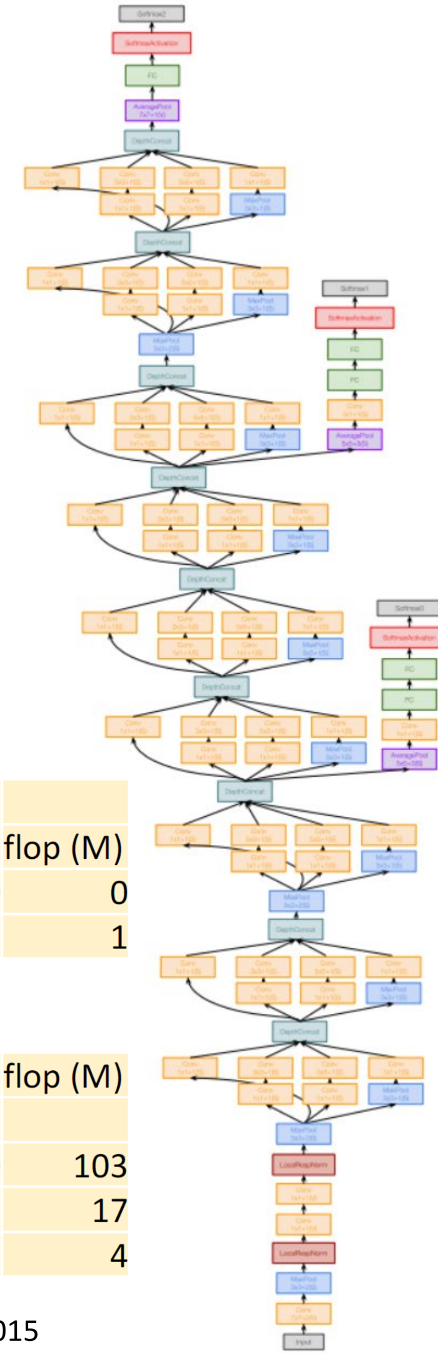
GoogleNet (cont'd)

- Inception Module
  - Local units with parallel branches
  - Repeat multiple times
  - Use 1x1 bottleneck layers to reduce channel dims
- Global Average Pooling
  - Avoid large FC layers

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Compare with VGG-16:

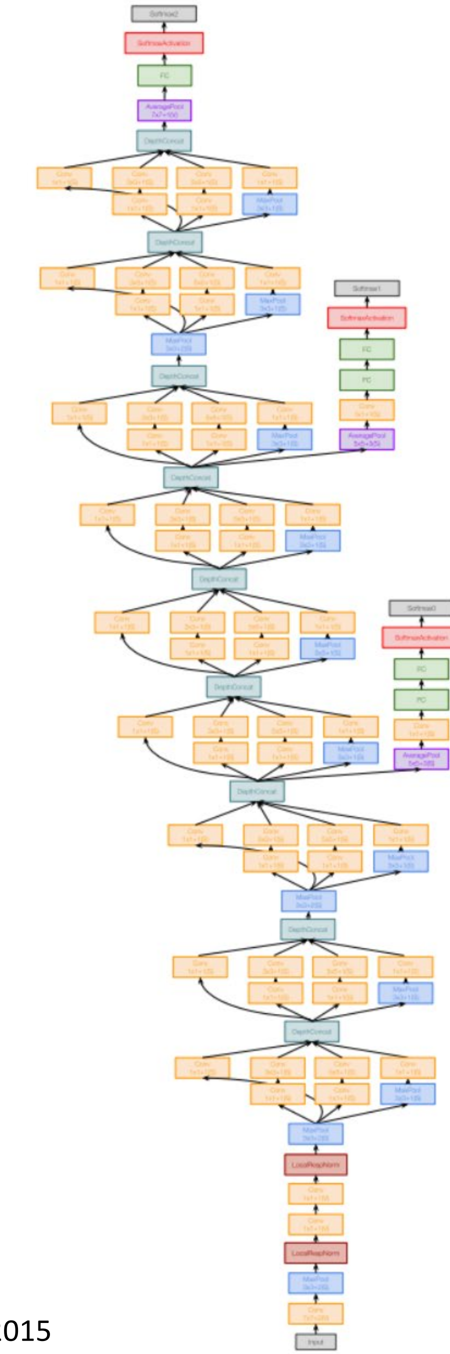
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



Szegedy et al., Going Deeper with Convolutions, CVPR 2015

GoogleNet (cont'd)

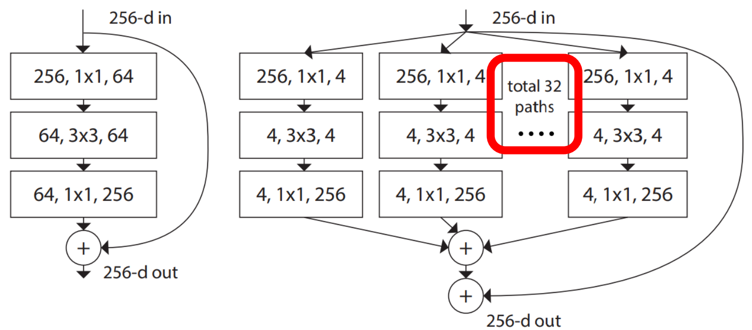
- Inception Module
  - Local units with parallel branches
  - Repeat multiple times
  - Use 1x1 bottleneck layers to reduce channel dims
- Global Average Pooling
  - Avoid large FC layers
- Auxiliary Classifier
  - Guidance to intermediate layers
  - Avoid deep layer with vanishing gradients



Szegedy et al., Going Deeper with Convolutions, CVPR 2015

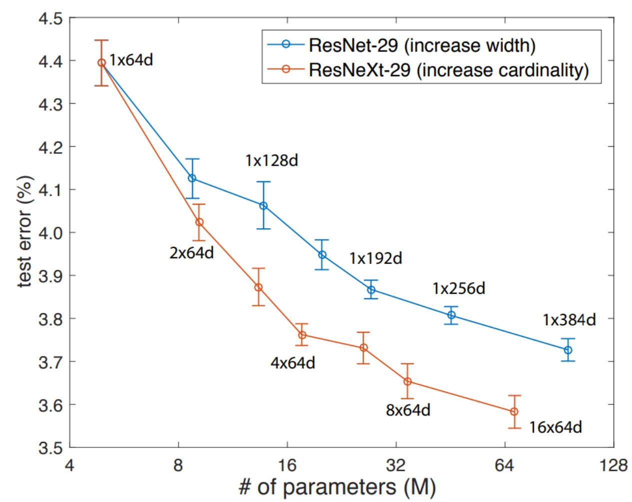
ResNeXT

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



ResNet block ResNeXt block

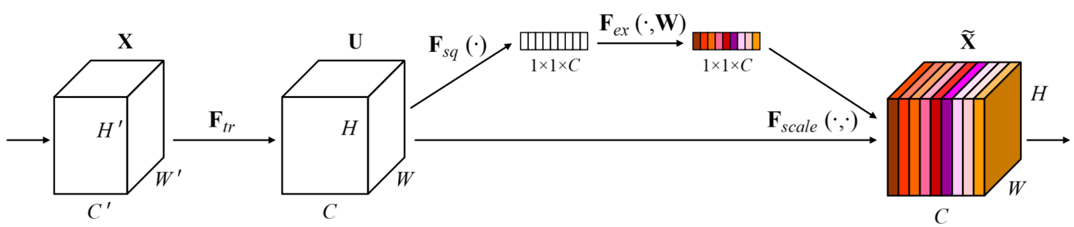
	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

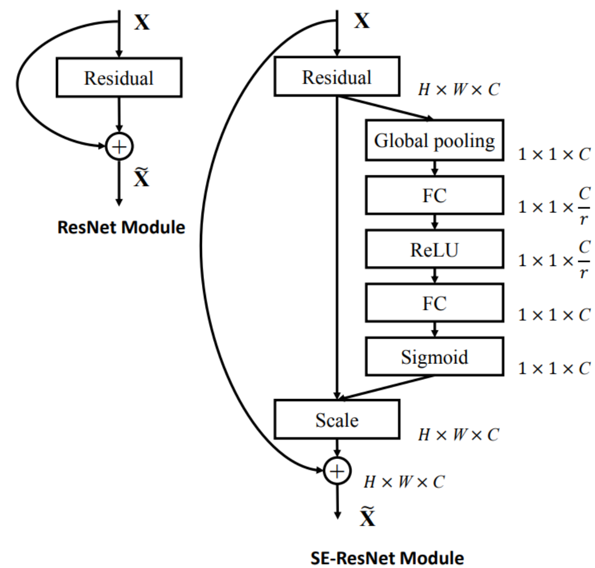


Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *CVPR,* 2017.

Squeeze-and-Excitation Net (SENet)

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



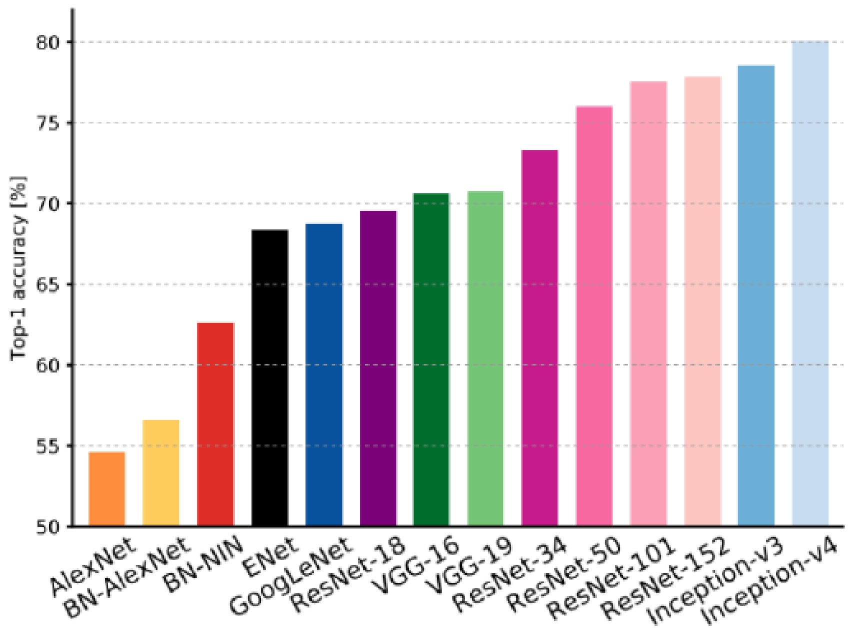


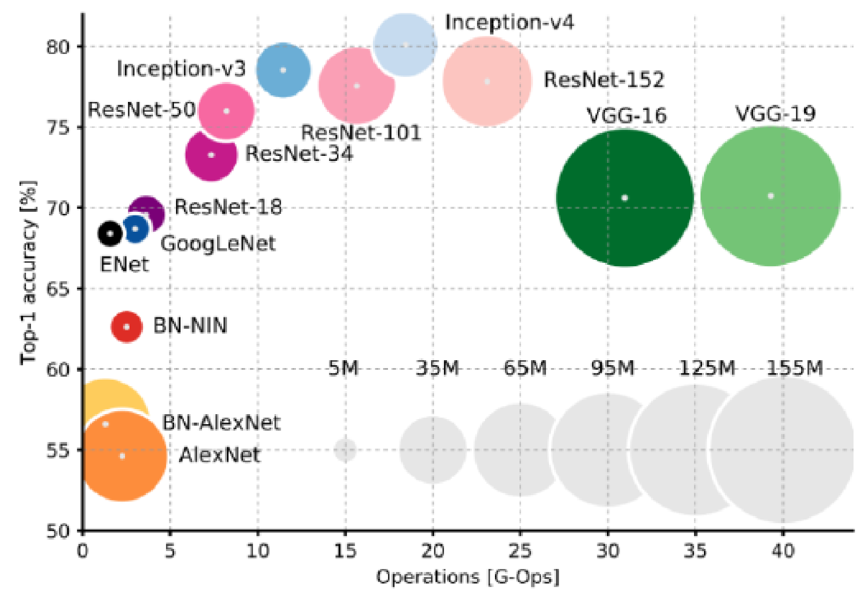
	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7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

Comparing Complexity

- ✓ Highest memory, most ops:
- ✓ Very efficient with moderate acc:
- ✓ Few ops but lots of parameters:
- ✓ Simple design, moderate efficiency yet high accuracy:

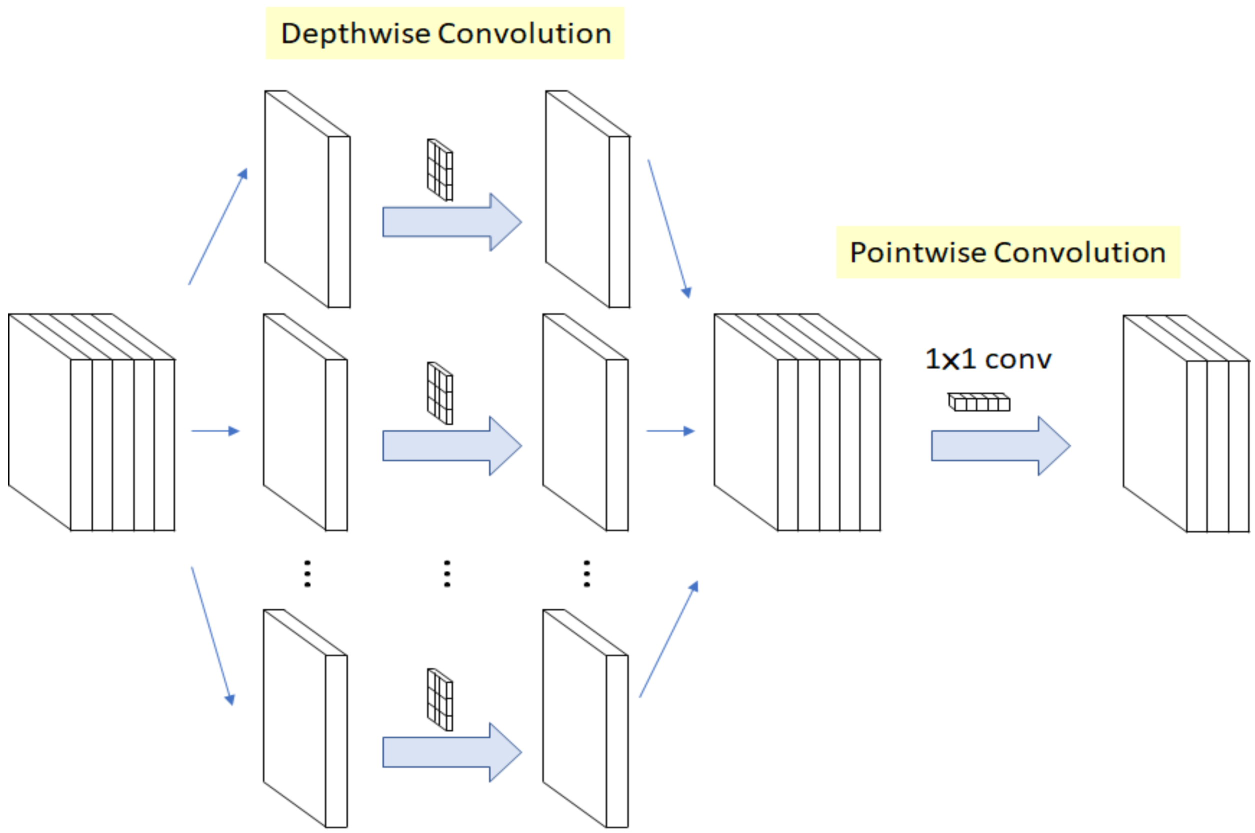




Canziani et al., An analysis of DNN for practical applications, 2017

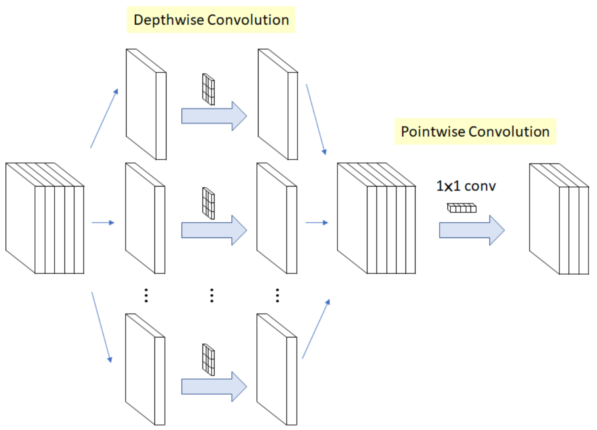
MobileNets: Tiny Networks for End Devices

- 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 :

Depthwise Convolution Pointwise Convolution

$$\frac{\boxed{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F} + \boxed{M \cdot N \cdot D_F \cdot D_F}}{\boxed{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}}$$

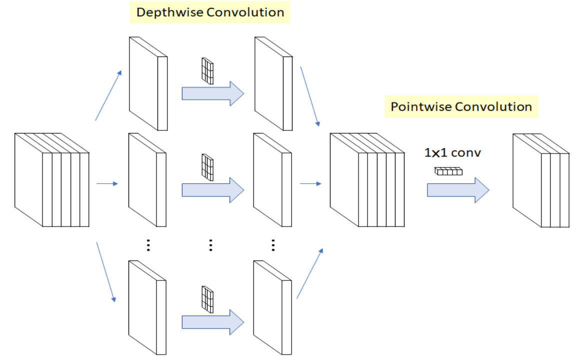
Standard Convolution

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

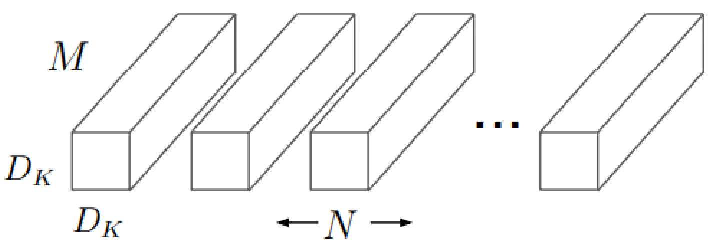
- Thus, depth+pointwise convolution requires only **1/N + 1/D_K^2** of the computation cost compared with that of standard convolution.

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

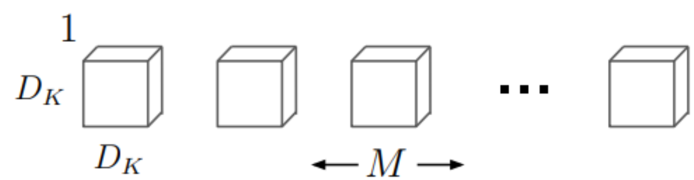
MobileNets (cont'd)



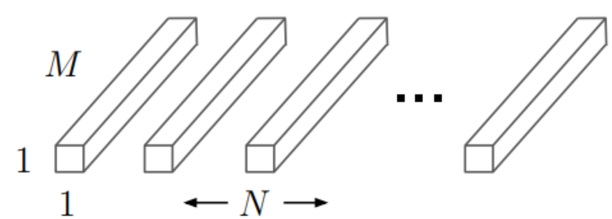
- MobileNet V1
  - Reduced Memory Size
    - Take a standard convolution which kernel size is D_K (with M input and N output channels).
    - The operation can be separated into a depthwise convolution which kernel (filter) size is D_K , and a pointwise convolution where input and output channels are M and N, respectively.
    - Therefore, the memory reduction is also $1/N + 1/D_K^2$.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



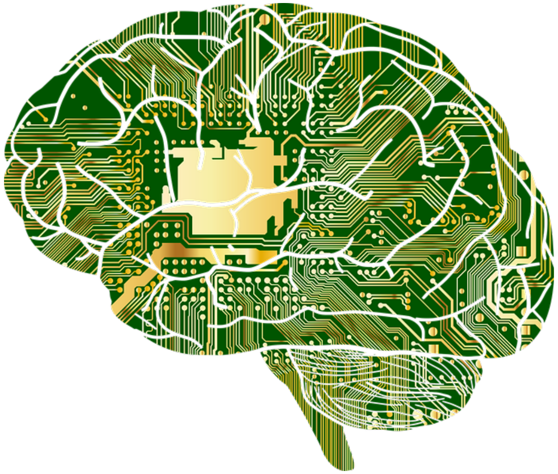
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

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

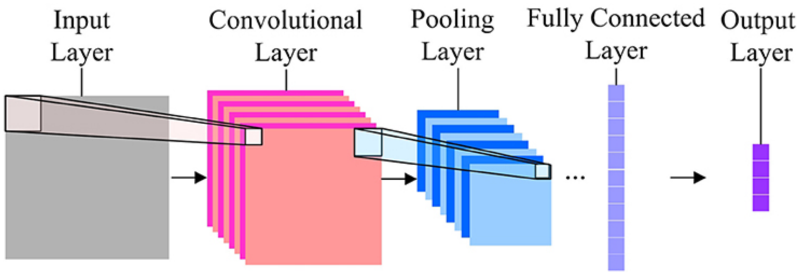
Remarks

- CNN: convolution, nonlinearity, pooling & FC
  - 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.

What's to Be Covered Today…



- Convolutional Neural Networks
  - Properties of CNN
  - Selected variants of CNN
  - Training CNN
  - Visualizing CNN
- Segmentation



Training Convolutional Neural Networks

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

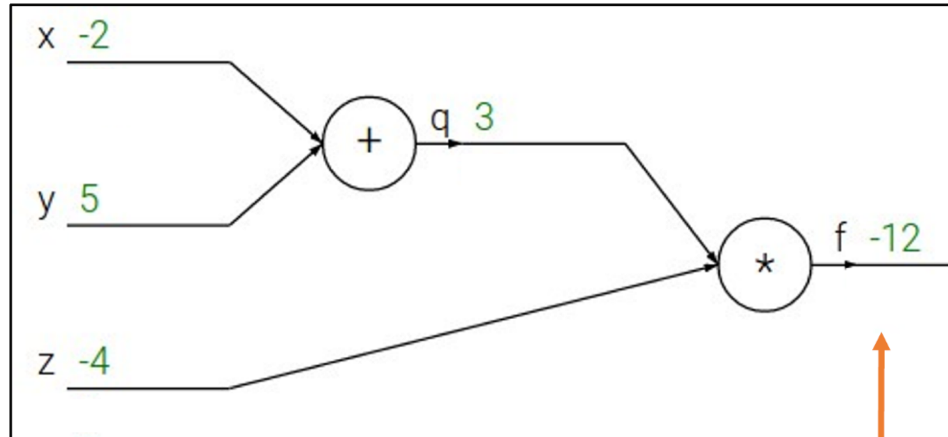
Training Convolutional Neural Networks

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

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{\frac{\partial f}{\partial f}}$$

1. Forward pass: Compute outputs

$$q = x + y \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

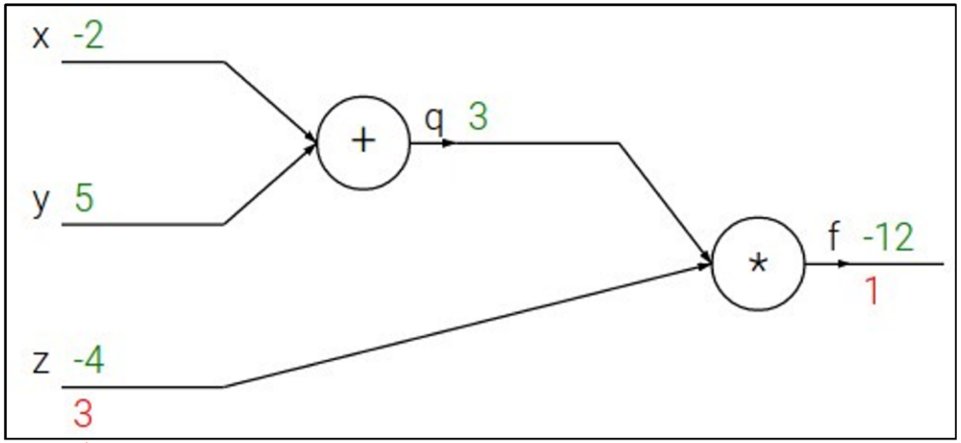
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide credit: UMich EECS 498-007

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial z} = q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

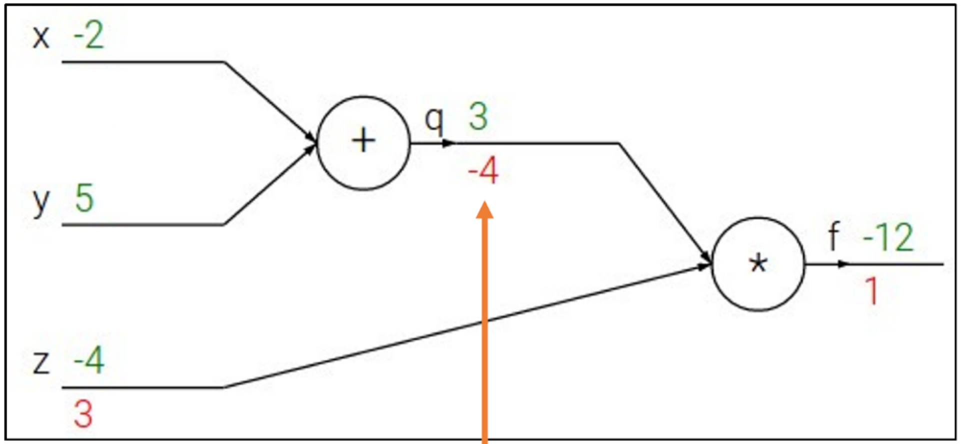
e.g. x = -2, y = 5, z = -4

1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q} = z}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

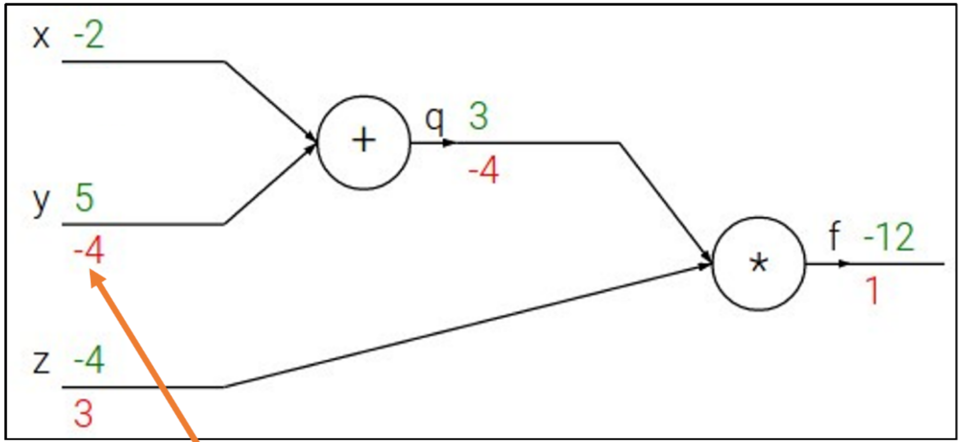
e.g. x = -2, y = 5, z = -4

1. Forward pass: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



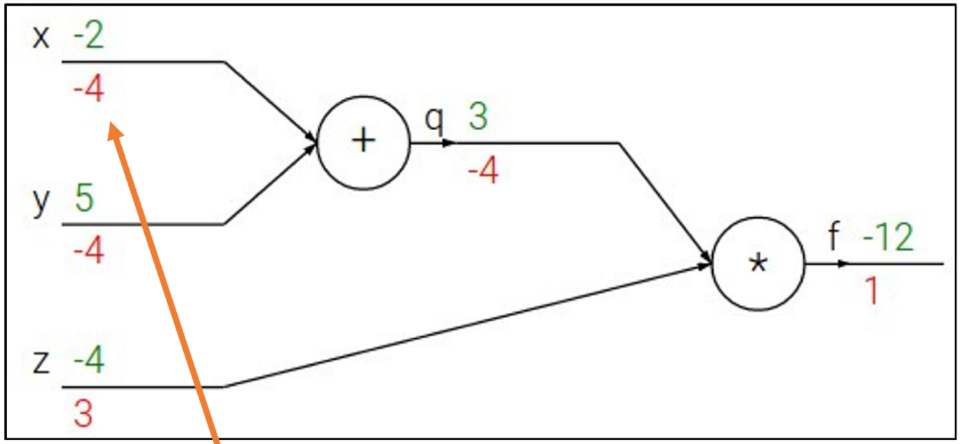
Chain Rule

$$\boxed{\underbrace{\frac{\partial f}{\partial y}}_{\text{Downstream Gradient}} = \underbrace{\frac{\partial q}{\partial y}}_{\text{Local Gradient}} \underbrace{\frac{\partial f}{\partial q}}_{\text{Upstream Gradient}}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

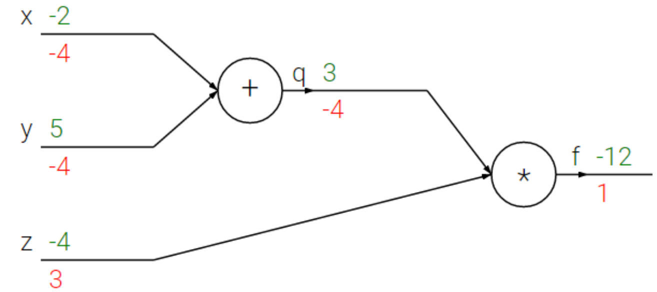
Chain Rule

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

Slide credit: UMich EECS 498-007

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



$q = x + y$

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

$f = qz$

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

Chain rule:

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

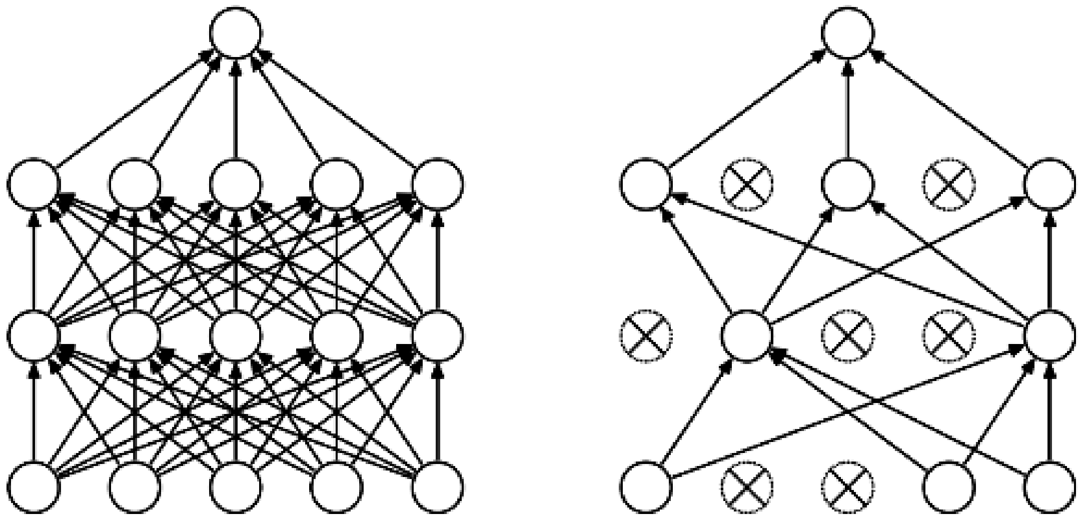
```
# set some inputs
x = -2; y = 5; z = -4

# 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
```

Slide credit: Andrej Karpathy

Dropout



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

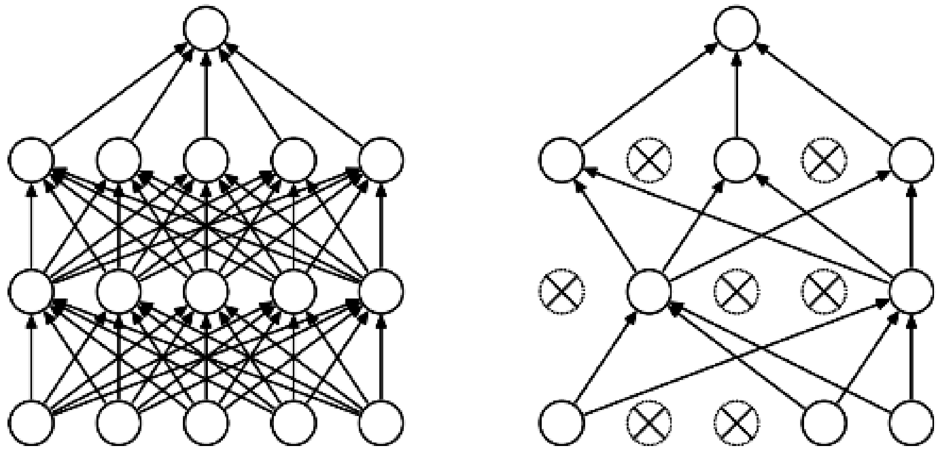
Intuition: successful **conspiracies**

Example: 50 people planning a conspiracy

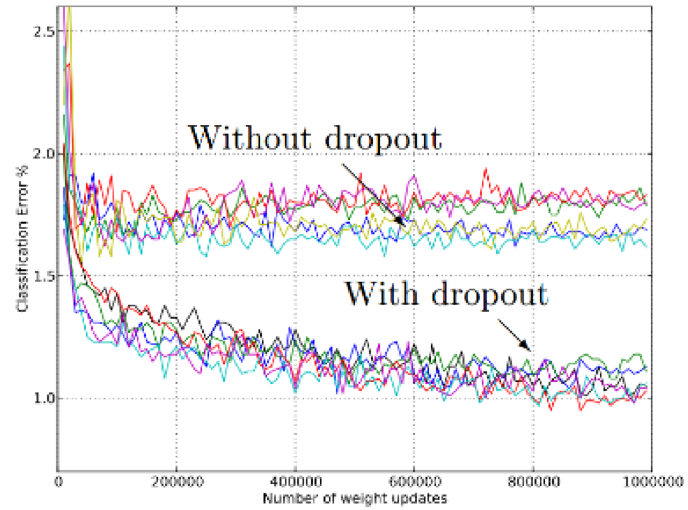
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

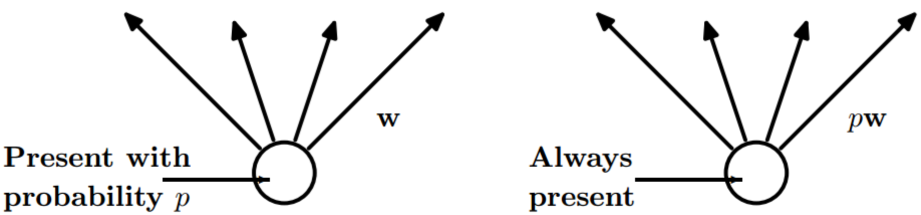
Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

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

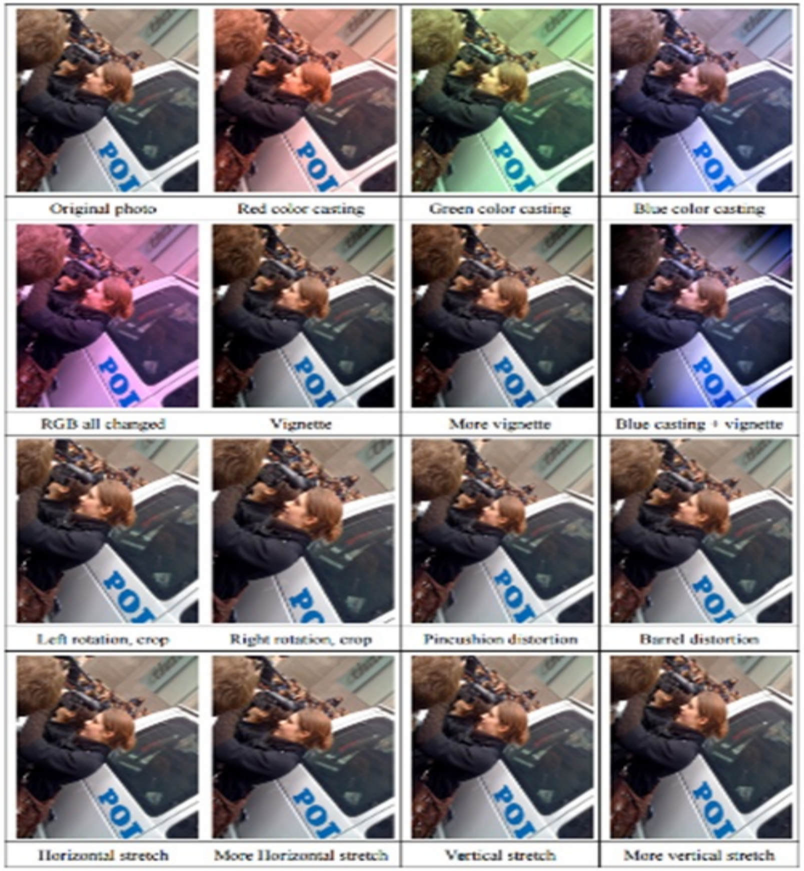
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

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

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

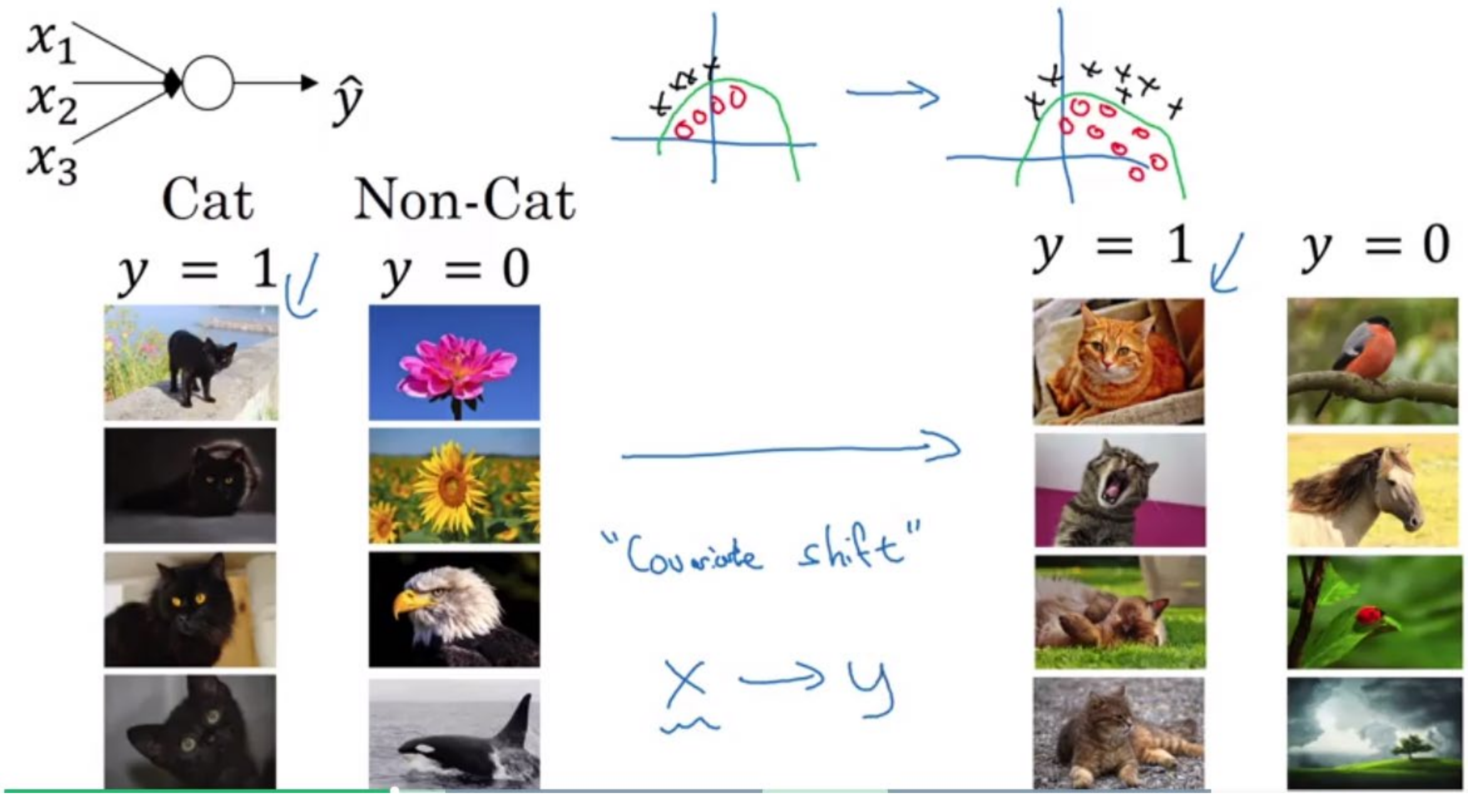
Data Augmentation (Jittering)

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

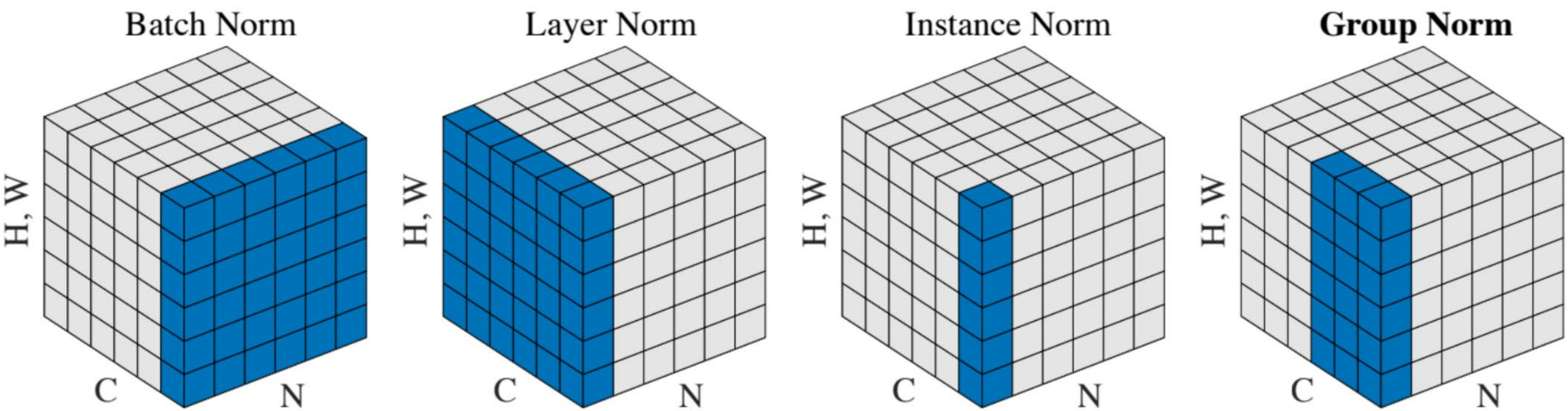


Deep Image [Wu et al. 2015]

Batch Normalization



Variants of Normalization in Training CNN



Wu and He, "Group Normalization", ECCV 2018

Batch Normalization

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

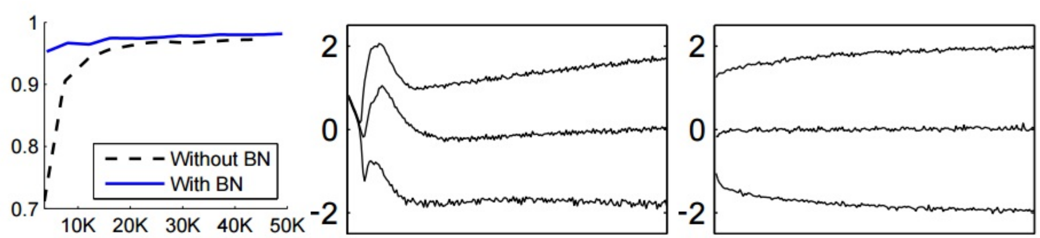
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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



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

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

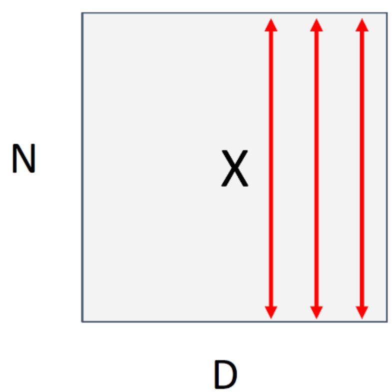
Batch Normalization (cont'd)

- 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

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

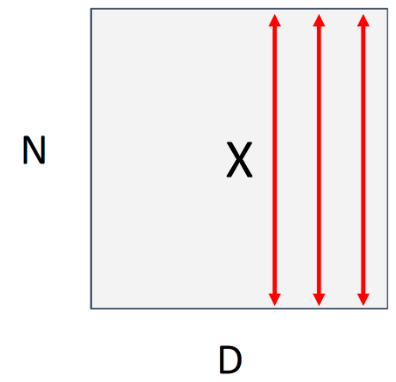
Per-channel std across N samples

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

Normalized x, Shape is N x D

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

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

Batch Normalization in CNN

Batch Normalization for **fully-connected** networks

$$\mathbf{x}: \; N \times D$$

Normalize

$$\mu, \sigma: \; 1 \times D$$

$$\gamma, \beta: \; 1 \times D$$

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

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize

$$\mu, \sigma: \; 1 \times C \times 1 \times 1$$

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

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

Instance Normalization in CNN

Batch Normalization for convolutional networks

$$\mathbf{x}:\ N \times C \times H \times W$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}:\ 1 \times C \times 1 \times 1$$

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

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

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

$$\mathbf{x}:\ N \times C \times H \times W$$

Normalize

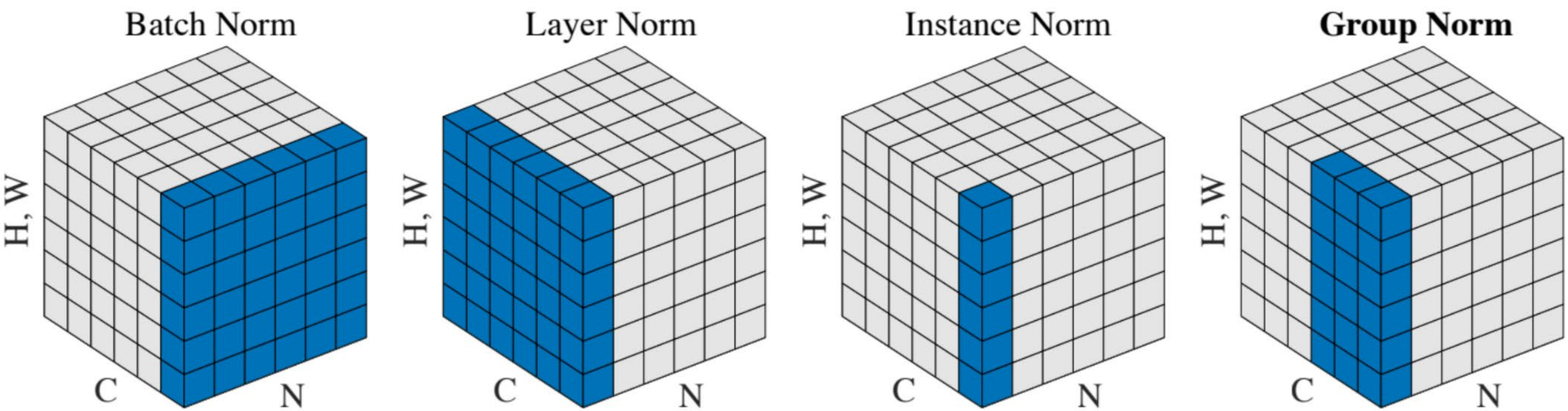
$$\boldsymbol{\mu}, \boldsymbol{\sigma}:\ N \times C \times 1 \times 1$$

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

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

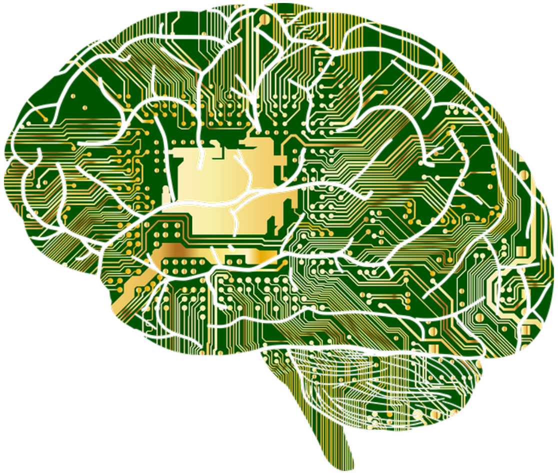
Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Variants of Normalization in Training CNN

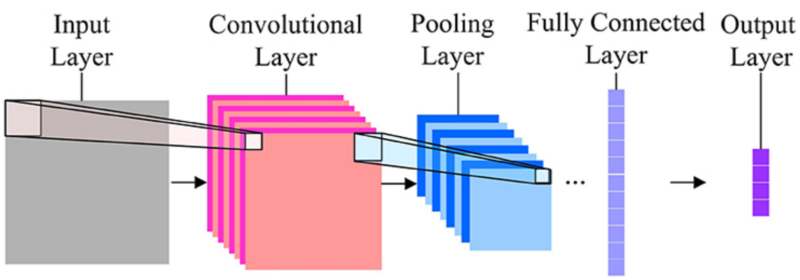


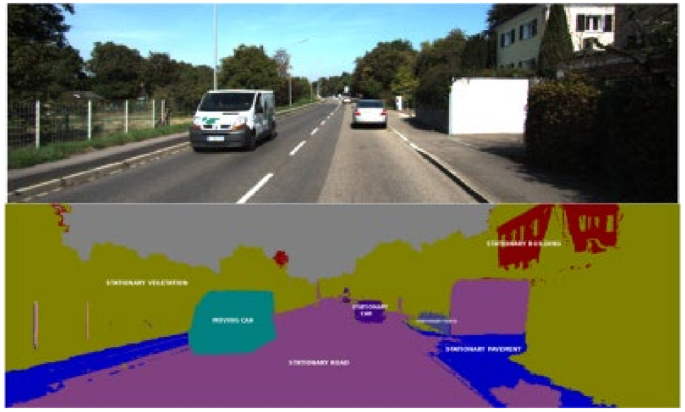
Wu and He, "Group Normalization", ECCV 2018

What’s to Be Covered Today…

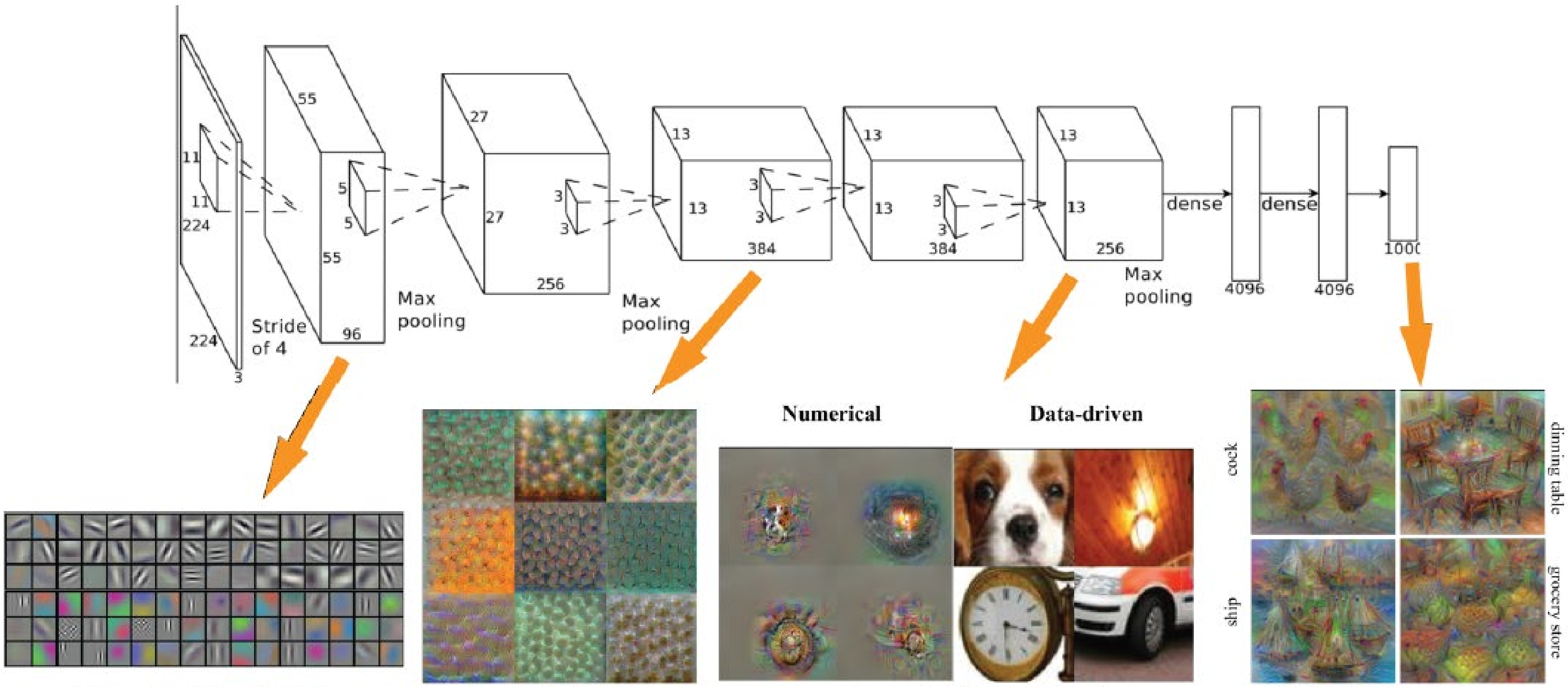


- Convolutional Neural Networks
    - Properties of CNN
    - Selected variants of CNN
    - Training CNN
    - Visualizing CNN
- Segmentation





Visualizing CNN (if time permits): What's Going on Inside CNNs?

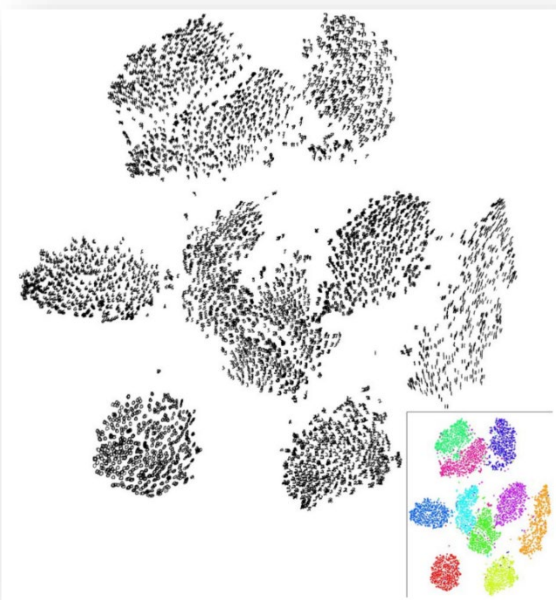


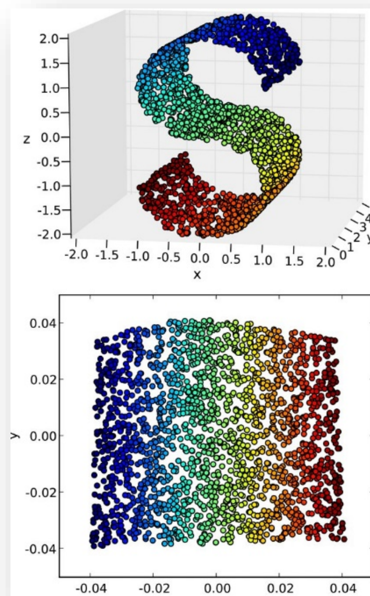
Conv 1: Edge+Blob Conv 3: Texture Conv 5: Object Parts Fc8: Object Classes

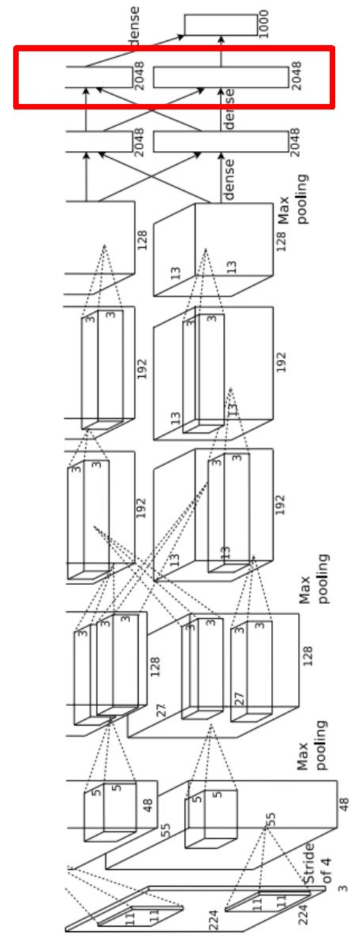
http://vision03.csail.mit.edu/cnn_art/data/single_layer.png

Recap:
Visualizing CNN Features (at the final layer)

- Visualization via Dimension Reduction
  - Linear DR: PCA
  - Non-linear DR:
    t-distributed stochastic neighbor embedding (t-SNE)
    (by G. Hinton & L. van der Maaten)
  - For classification purposes, FC layers are applied.

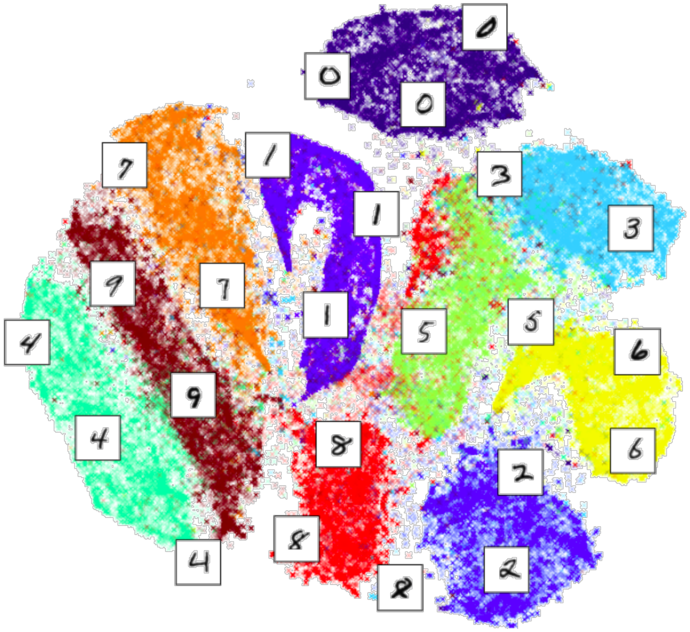






t-Distributed Stochastic Neighbor Embedding (t-SNE)

- **Remarks**
  - Powerful tool for data visualization
  - Help understand black-box algorithms like CNN ☺
  - Alleviate crowding problem
  - Great resources/tools available
    e.g., https://distill.pub/2016/misread-tsne



Recap:
Visualizing CNN Features (at the final layer)

- Visualization via Dimension Reduction
  - Linear DR: PCA
  - Non-linear DR: t-SNE
  - For classification purposes, FCN is applied.
- What about other layers?

First layer weights: 16 x 3 x 7 x 7

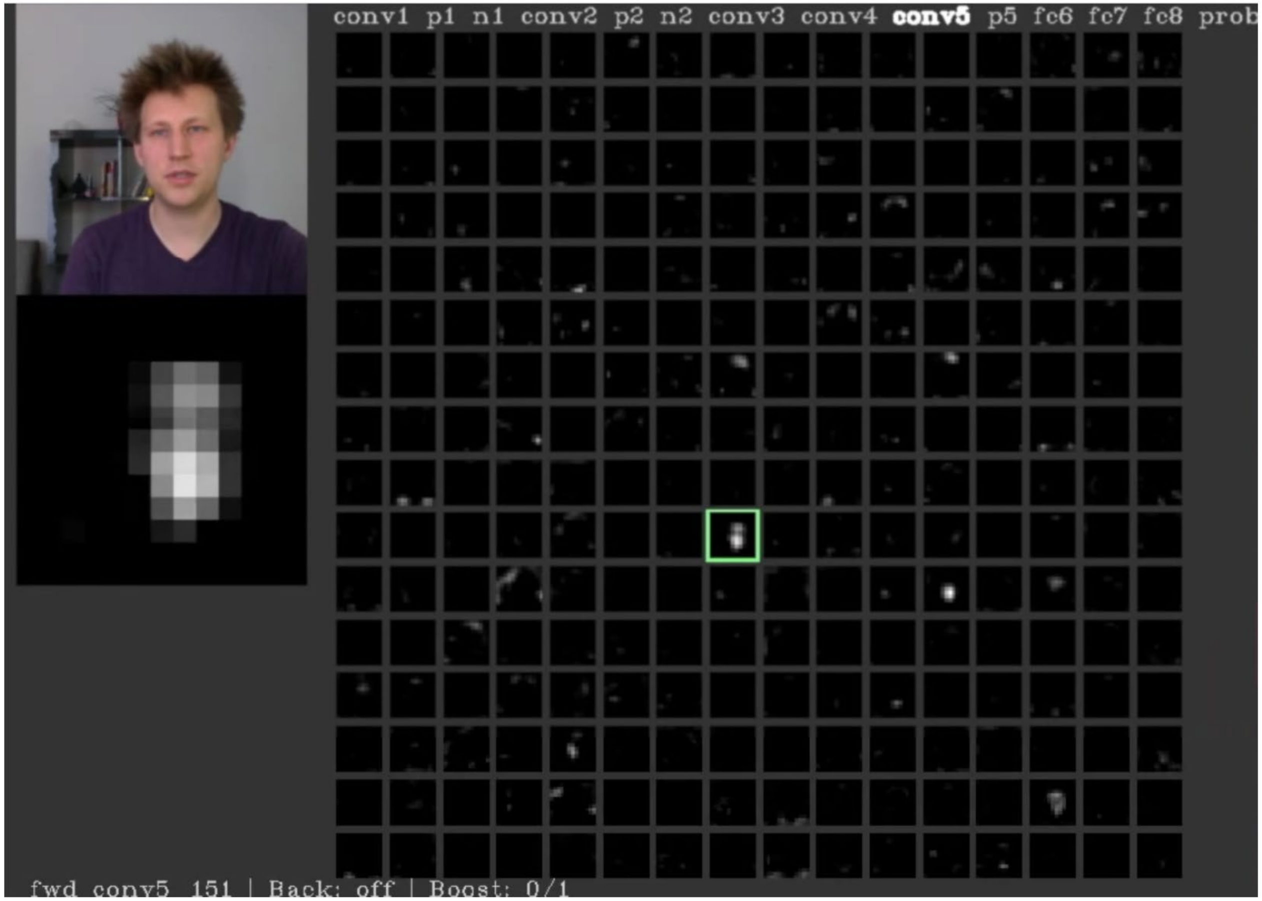
Second layer weights: 20 x 16 x 7 x 7

Third layer weights: 20 x 20 x 7 x 7

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

Visualization of Activations

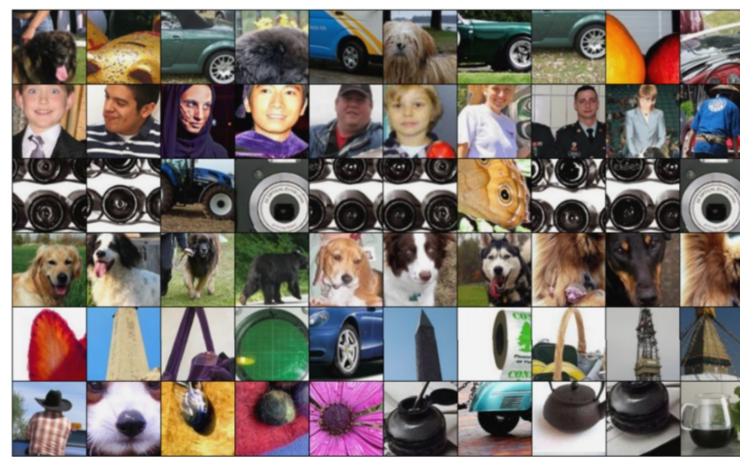
- Take conv5 feature map (13 x 13 x 128) as an example
  - Visualize as 128 grayscale images with size 13 x 13

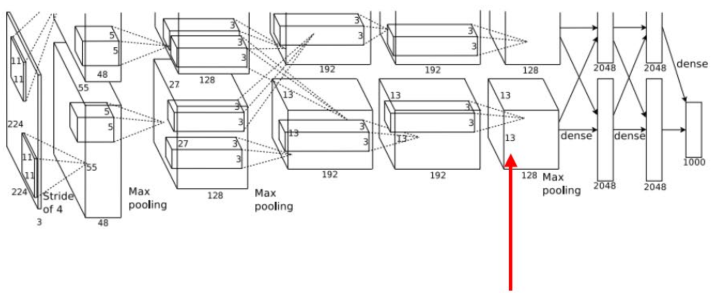


Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.

Visualization of Activations (cont'd)

- Patches with maximum activation
  - Run images through the network
  - Record values of the selected channel
  - Visualize image patches that correspond to maximal activations

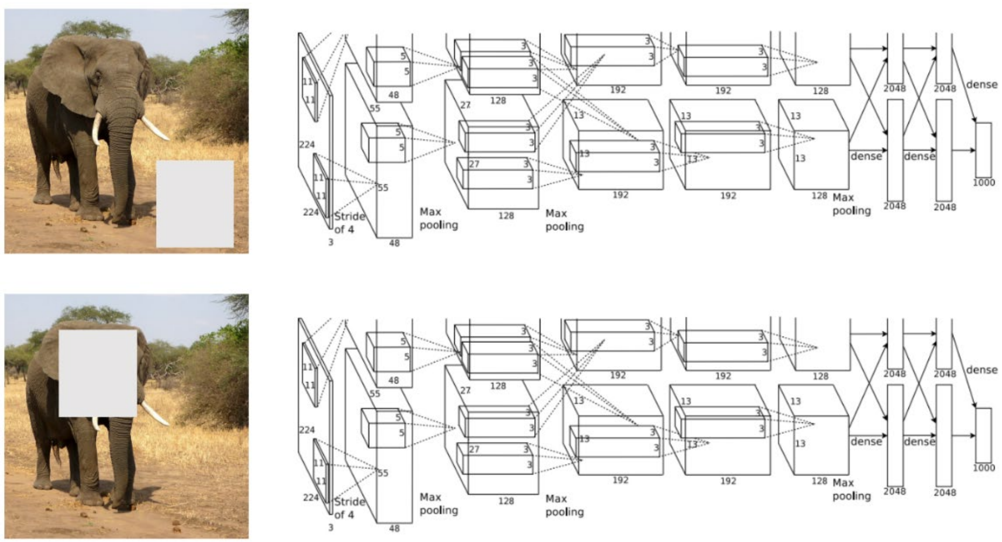


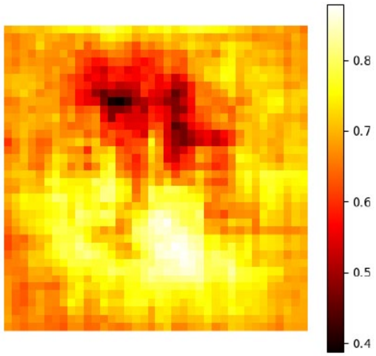


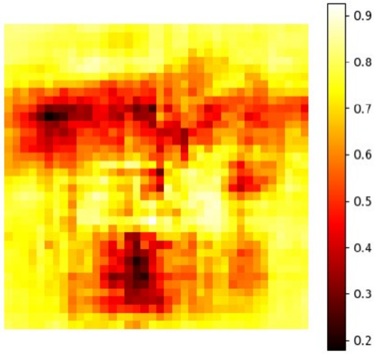
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Visualization of Activations (cont'd)

- Which pixels matter? Saliency via occlusion!
  - Mask parts of the input image before feeding to CNN
  - Check how much predicted probabilities change
  - Are the results as exactly what you expect?

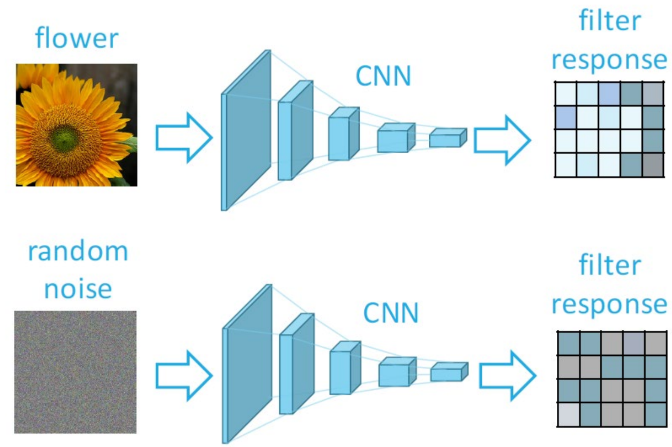






Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Activation maximization



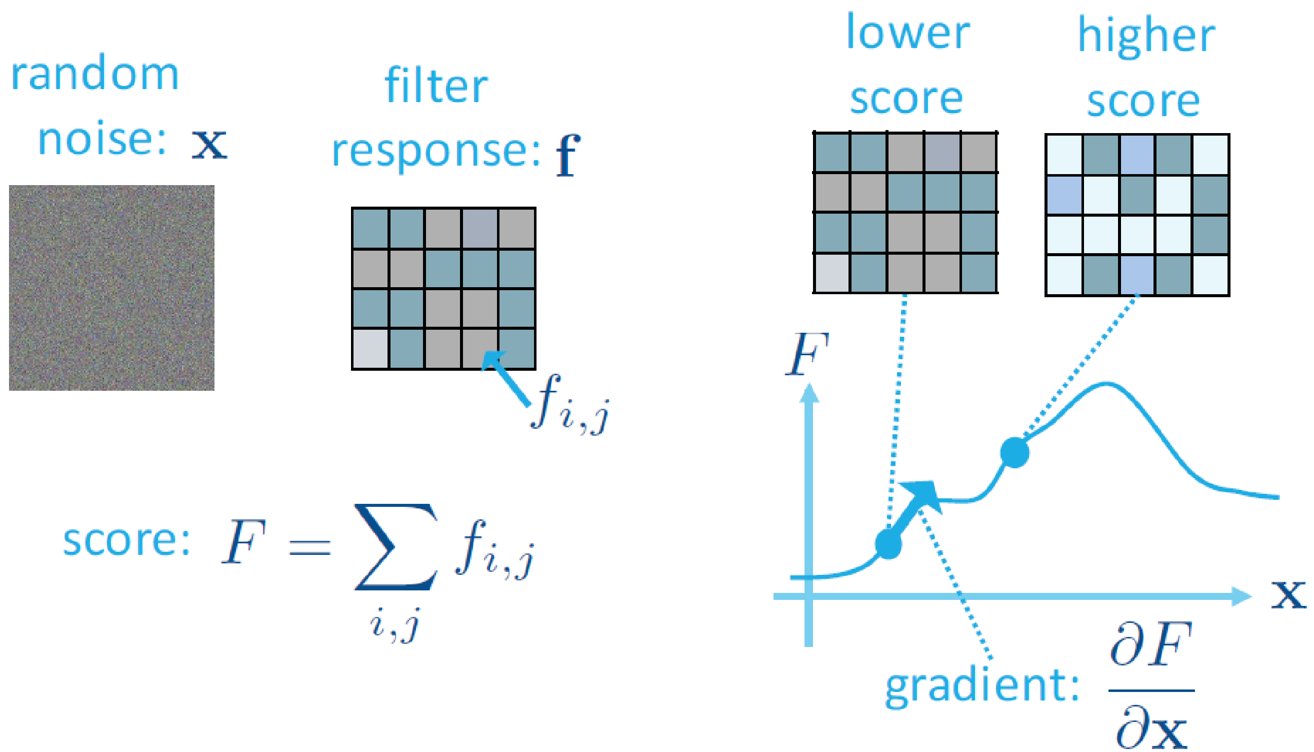
$$\mathbf{x}^* = \arg \max_{\mathbf{x} \ s.t. \ ||\mathbf{x}||=\rho} h_{ij}(\theta, \mathbf{x})$$

- θ: model parameters (already trained and fixed!)
- h_{ij}: the activation of a given unit i from a given layer j in the network
- x: input sample

- For a fix model θ, performing gradient ascent in the input space
- Hyperparameters: learning rate / stopping criterion

Gradient Ascent

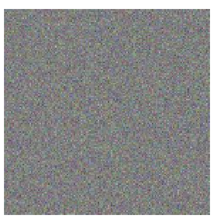
- Magnifying the filter response!



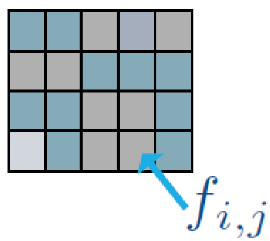
Gradient Ascent

- Magnifying the filter response!

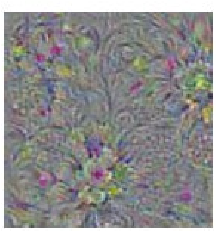
random noise: $\mathbf{x}$



filter response: $\mathbf{f}$

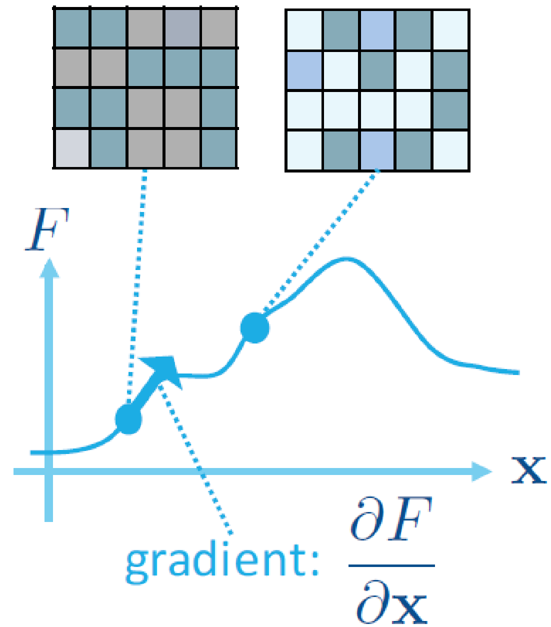


update $\mathbf{x}$



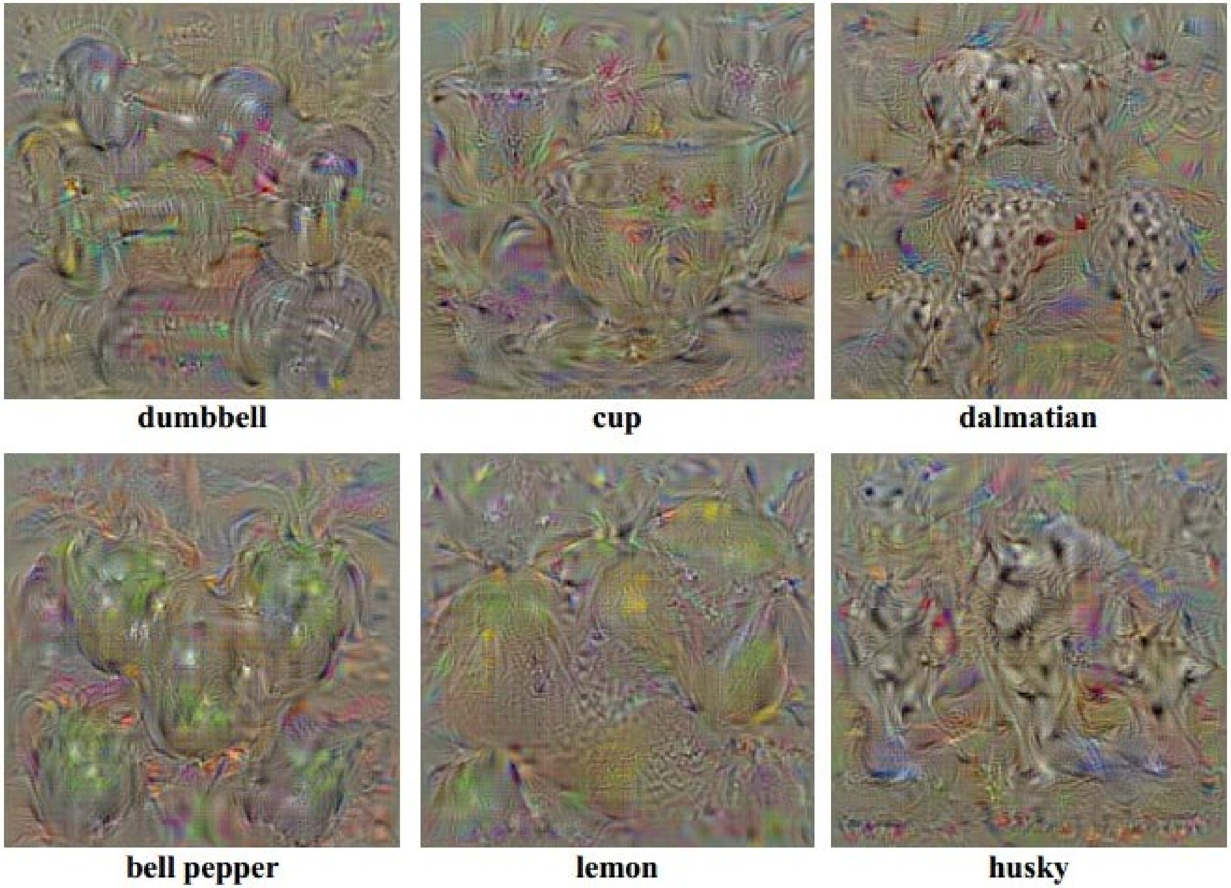
$$\mathbf{x} \leftarrow \mathbf{x} + \eta \frac{\partial F}{\partial \mathbf{x}}$$

learning rate



Example Visualization of CNN Features

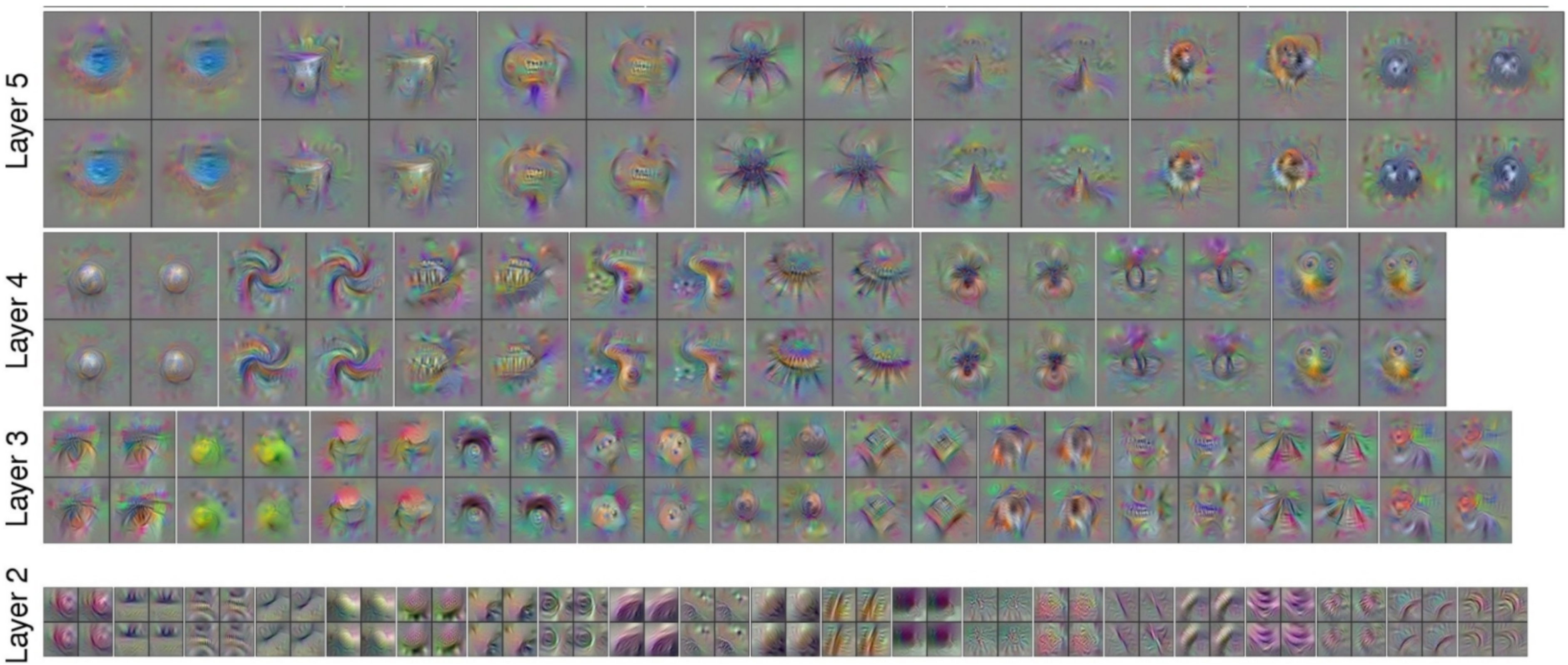
- Saliency via back propagation (by gradient ascent)



Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models andSaliency Maps”, ICLR WS 2014.

Example Visualization of CNN Features

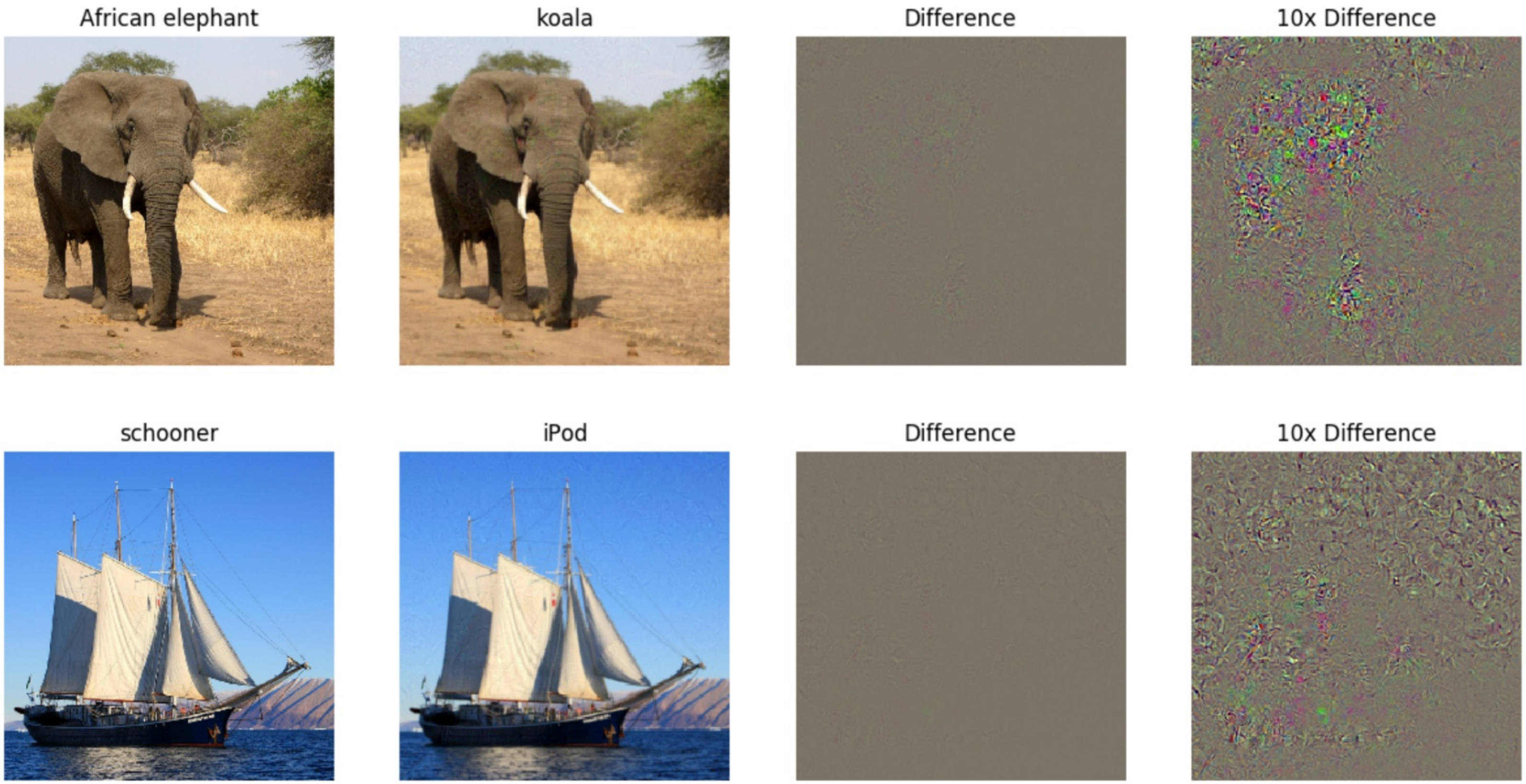
- Saliency via back propagation (by gradient ascent)
  - If visualizing intermediate feature maps



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014

One More Remark Before Moving Forward: **Adversarial Examples**

- Adversarial attack

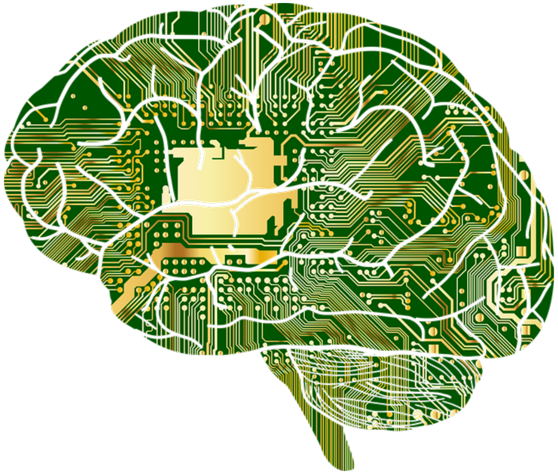


One More Remark Before Moving Forward: **Adversarial Examples**

- Basic ideas of producing adversarial attack
  - Start from an arbitrary image (e.g., stop sign)
  - Pick an category of interest (e.g., green light);
    modify the input image via gradient ascent to max the score of that category
  - Stop when the CNN is fooled 

What’s to Be Covered Today…

- Convolutional Neural Networks
  - Properties of CNN
  - Selected variants of CNN
  - Training CNN
  - Visualizing CNN
- Segmentation



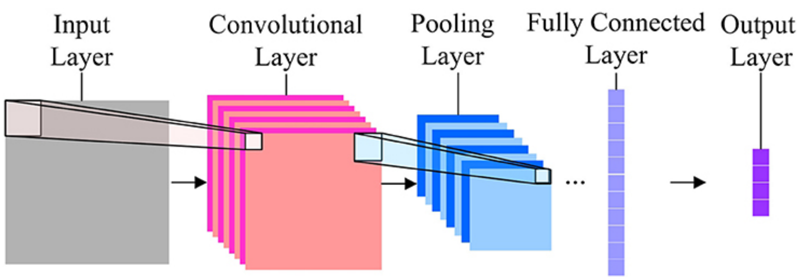
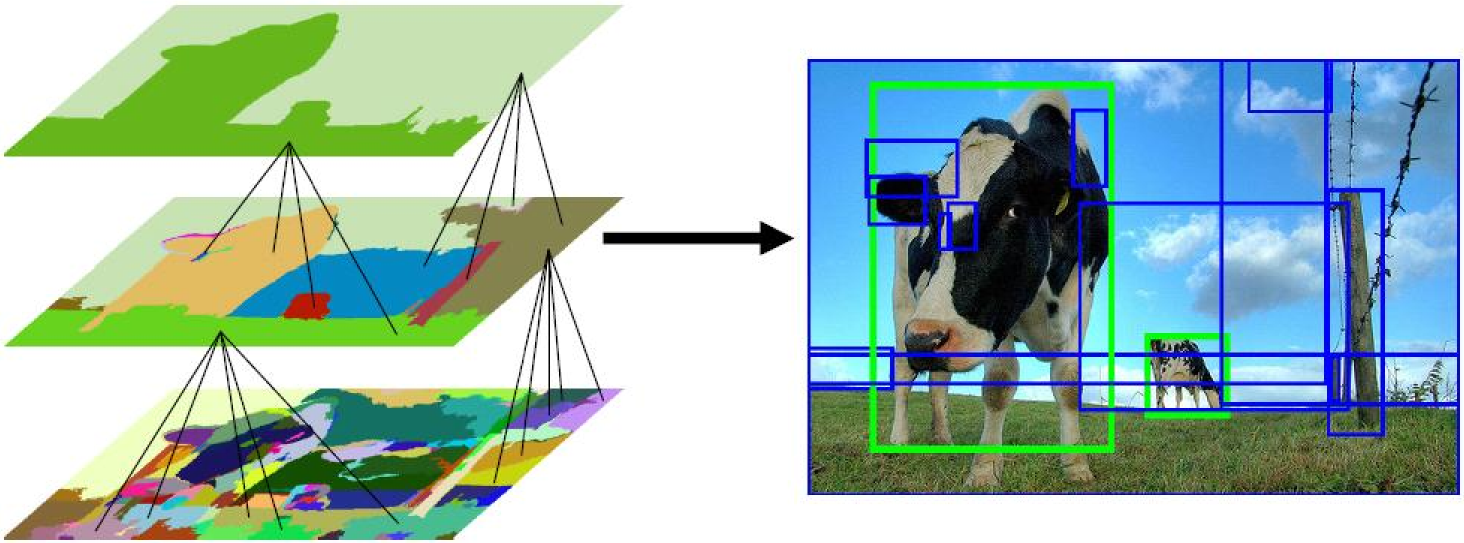


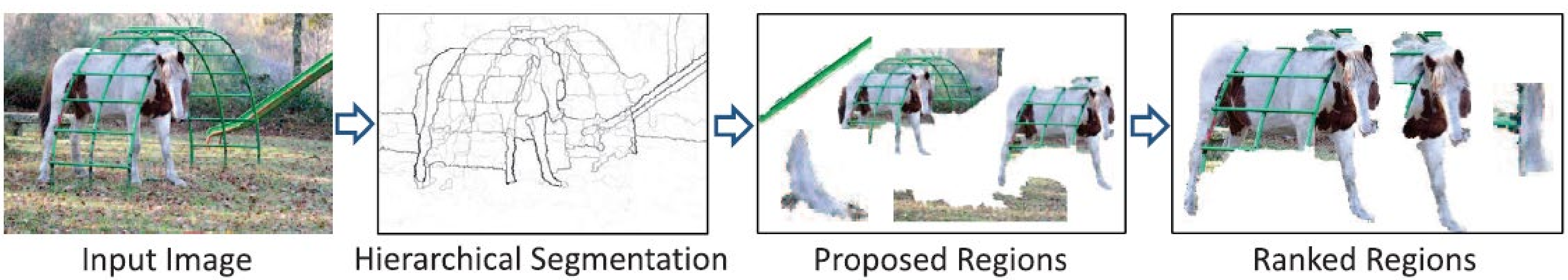
Image Segmentation

- Goal:
  Group pixels into meaningful or perceptually similar regions

Segmentation for Object Proposal



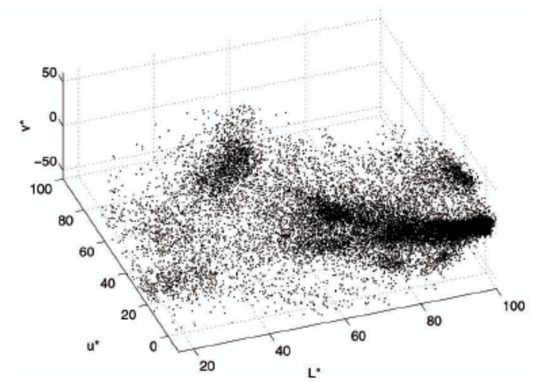
"Selective Search" [Sande, Uijlings et al. ICCV 2011, IJCV 2013]

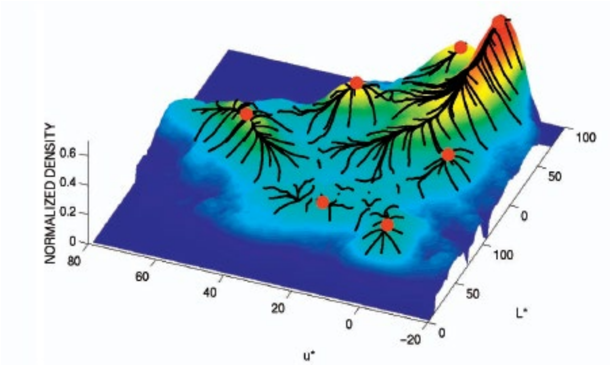


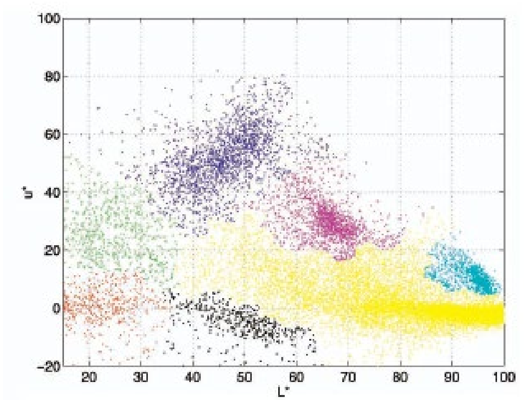
[Endres Hoiem ECCV 2010, IJCV 2014]

Segmentation via Clustering

- K-means clustering
- Mean-shift*
  - Find modes of the following non-parametric density



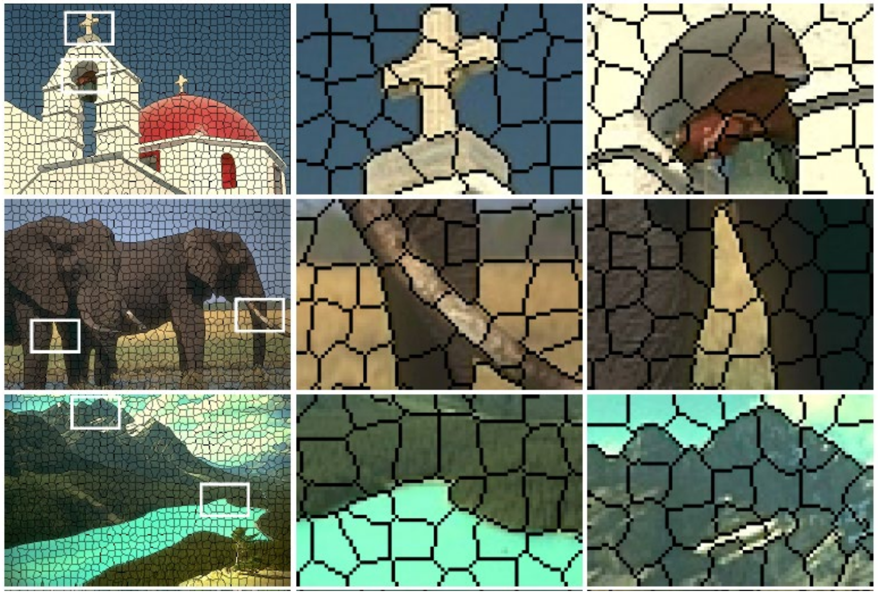




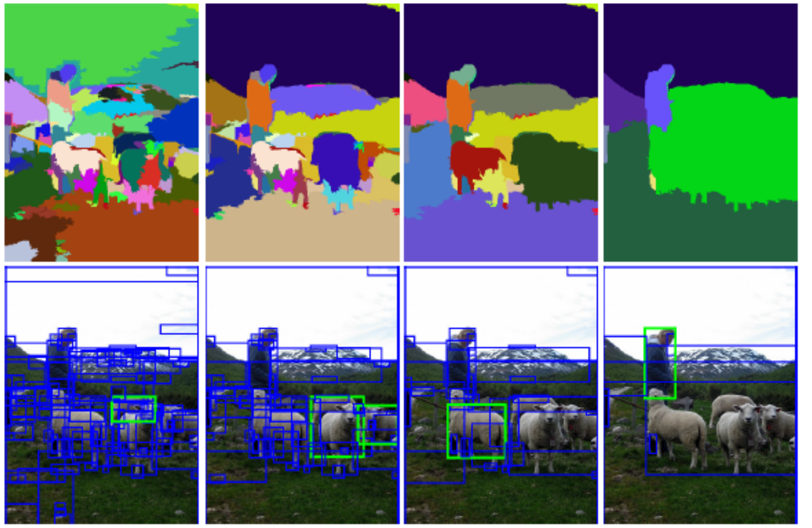
*D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, IEEE PAMI 2002.

Superpixels

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



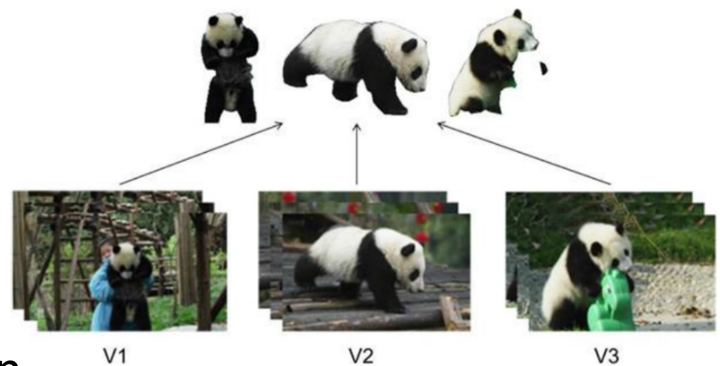
Multiple Segmentations



- Don’t commit to one partitioning
- Hierarchical segmentation
  - Occlusion boundaries hierarchy:
    Hoiem et al. IJCV 2011 (uses trained classifier to merge)
  - Pb+watershed hierarchy: Arbeleaz et al. CVPR 2009
  - Selective search: FH + agglomerative clustering
  - Superpixel hierarchy
- Varying segmentation parameters
  - E.g., multiple graph-based segmentations or mean-shift segmentations
- Region proposals
  - Propose seed superpixel, try to segment out object that contains it
    (Endres Hoiem ECCV 2010, Carreira Sminchisescu CVPR 2010)

More Tasks in Segmentation

- Cosegmentation
  - Segmenting common objects from multiple images



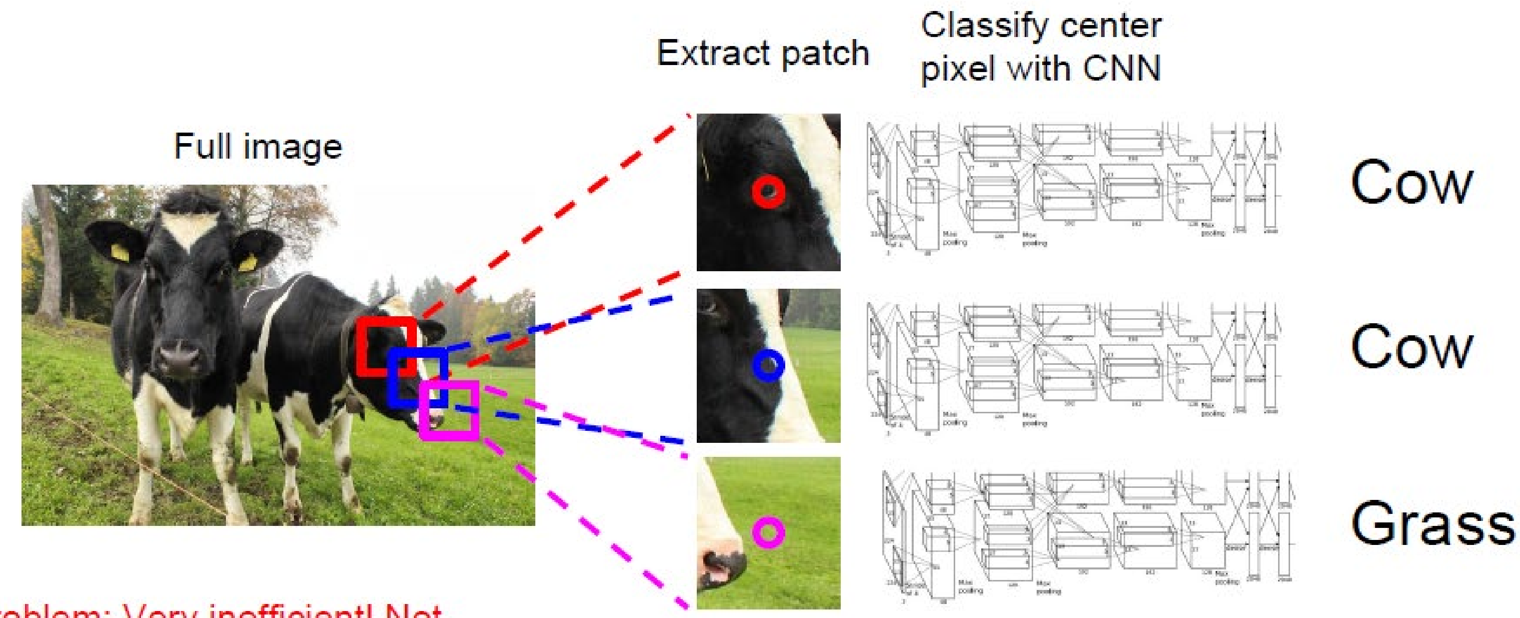
- Instance Segmentation
  - Assign each pixel an object instance

A Practical Segmentation Task

- Semantic Segmentation
  - Supervised learning
  - 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

Semantic Segmentation

- Sliding Window

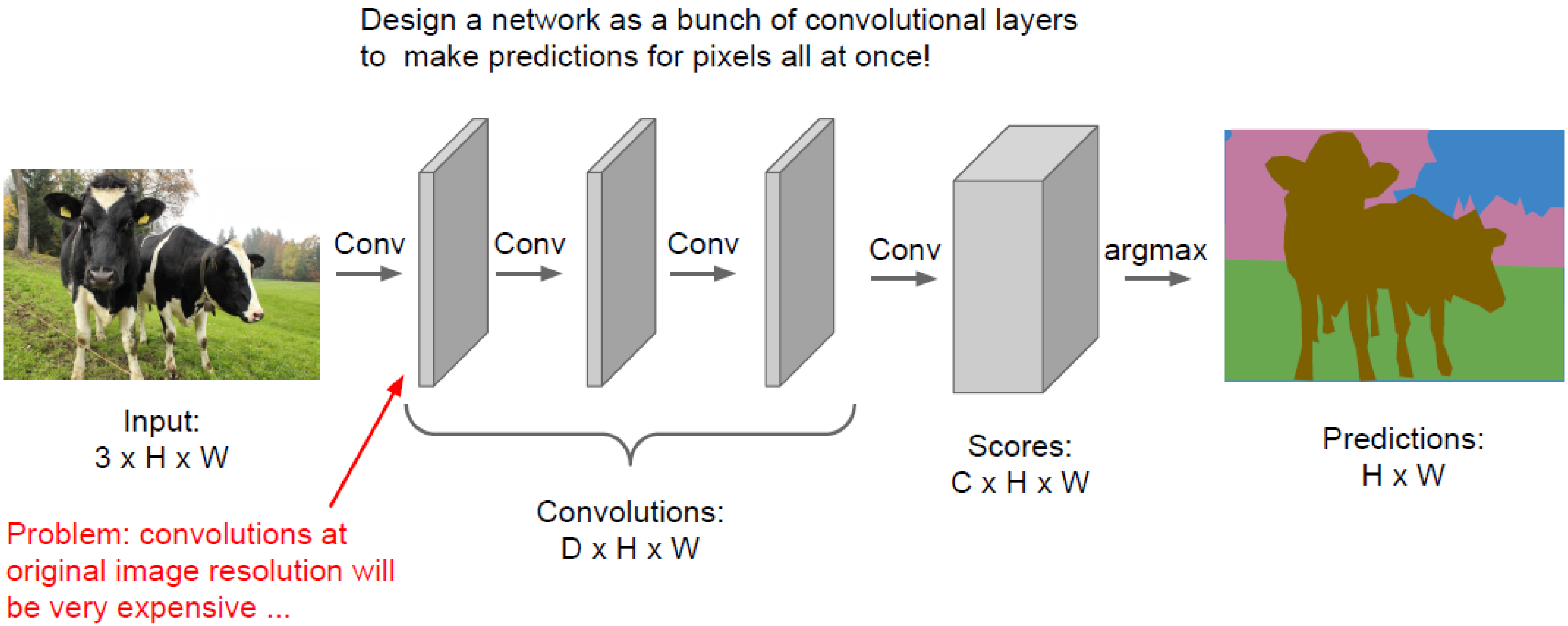


Problem: Very inefficient! Not reusing shared features between overlapping patches

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



Semantic Segmentation

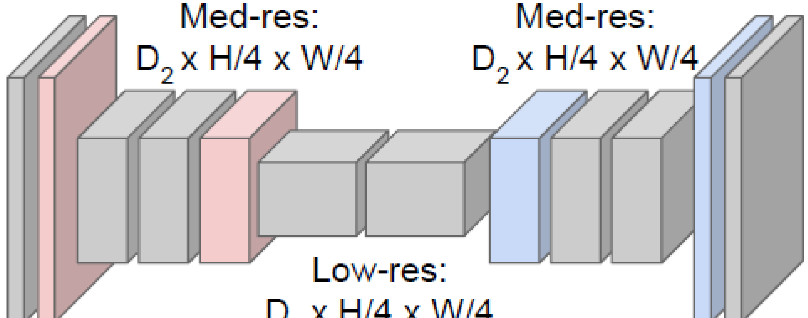
- Fully Convolutional Nets (cont'd)

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: ???

Input: $3 \times H \times W$



Med-res: $D_2 \times H/4 \times W/4$

Med-res: $D_2 \times H/4 \times W/4$

Low-res: $D_3 \times H/4 \times W/4$

High-res: $D_1 \times H/2 \times W/2$

High-res: $D_1 \times H/2 \times W/2$

Predictions: $H \times W$

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

1	2
3	4

Input: 2 x 2

→

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

Output: 4 x 4

“Bed of Nails”

1	2
3	4

Input: 2 x 2

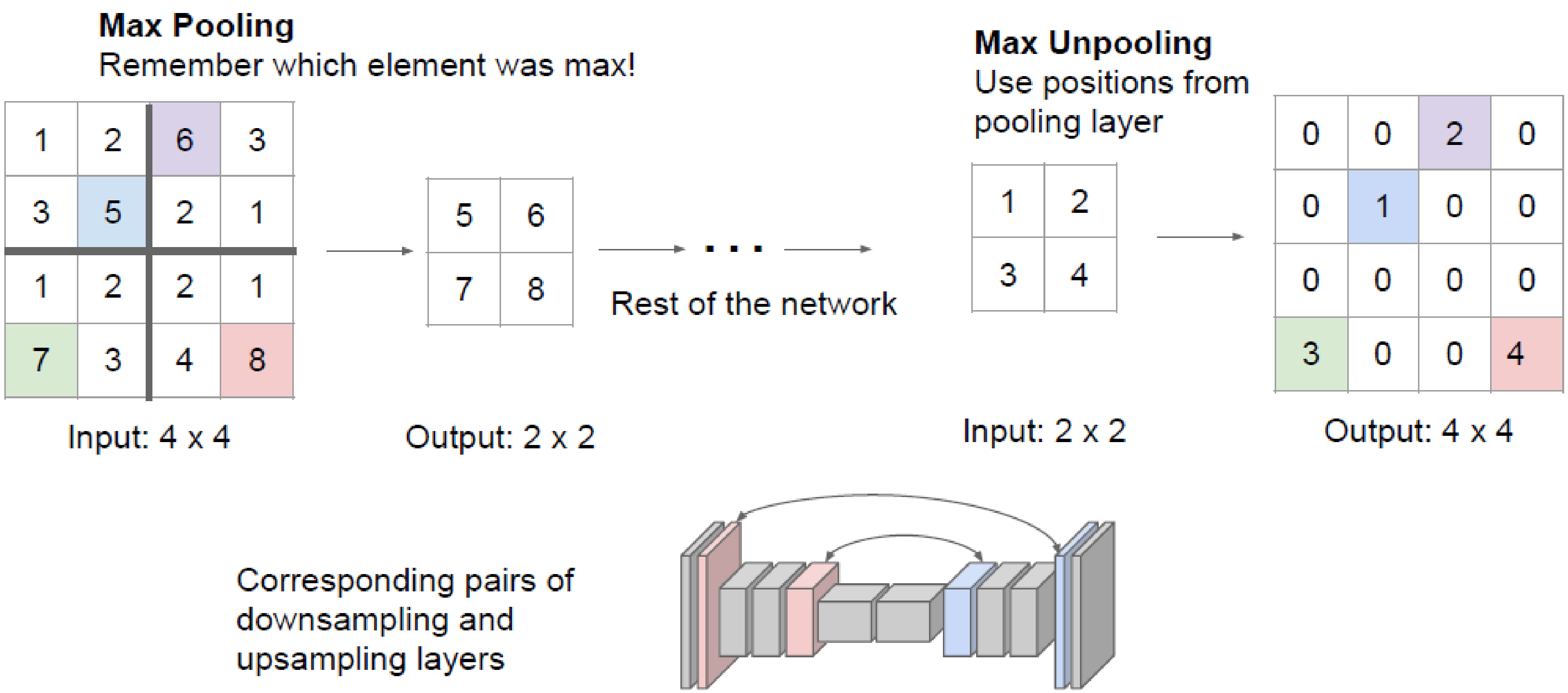
→

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

Output: 4 x 4

In-Network Upsampling

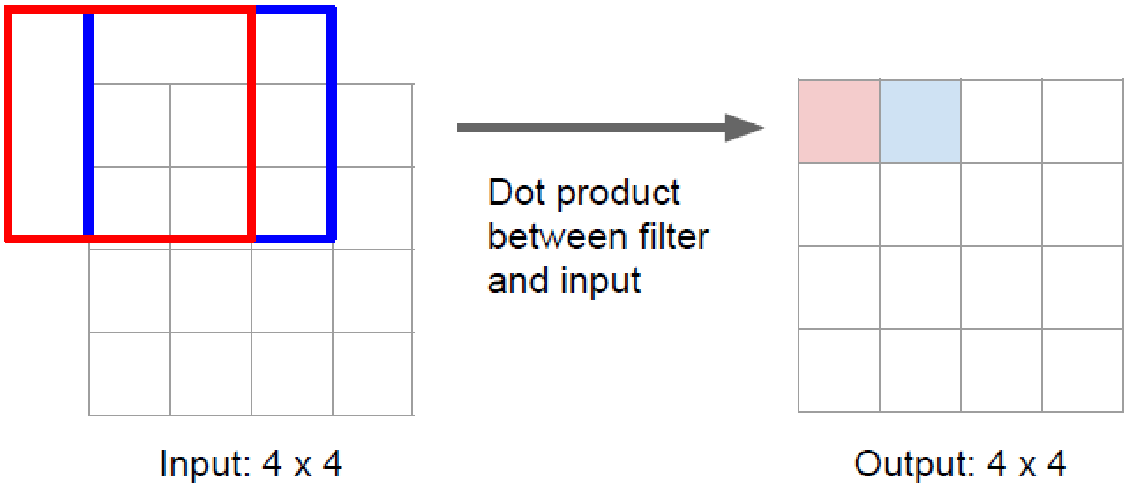
- Max Unpooling



In-Network Upsampling

- Learnable Upsampling: Transpose Convolution

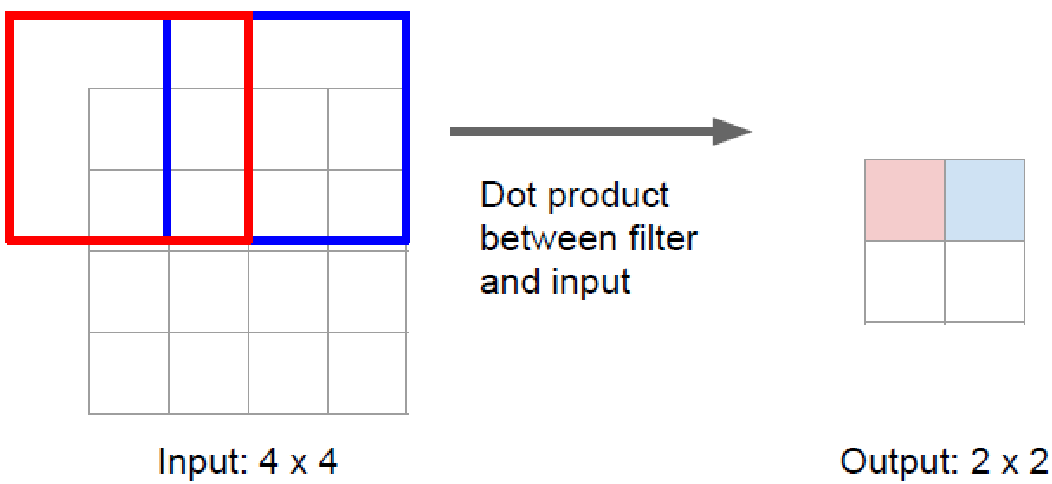
Recall: Normal 3 x 3 convolution, stride 1 pad 1



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

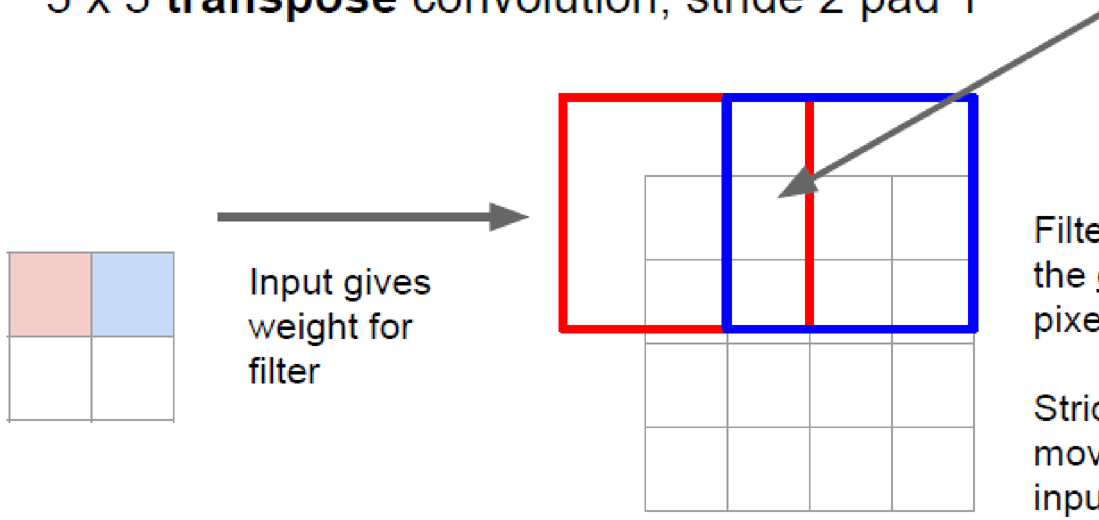
Stride gives ratio between movement in input and output

In-Network Upsampling

- Transpose Convolution

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

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



Input: 2 x 2

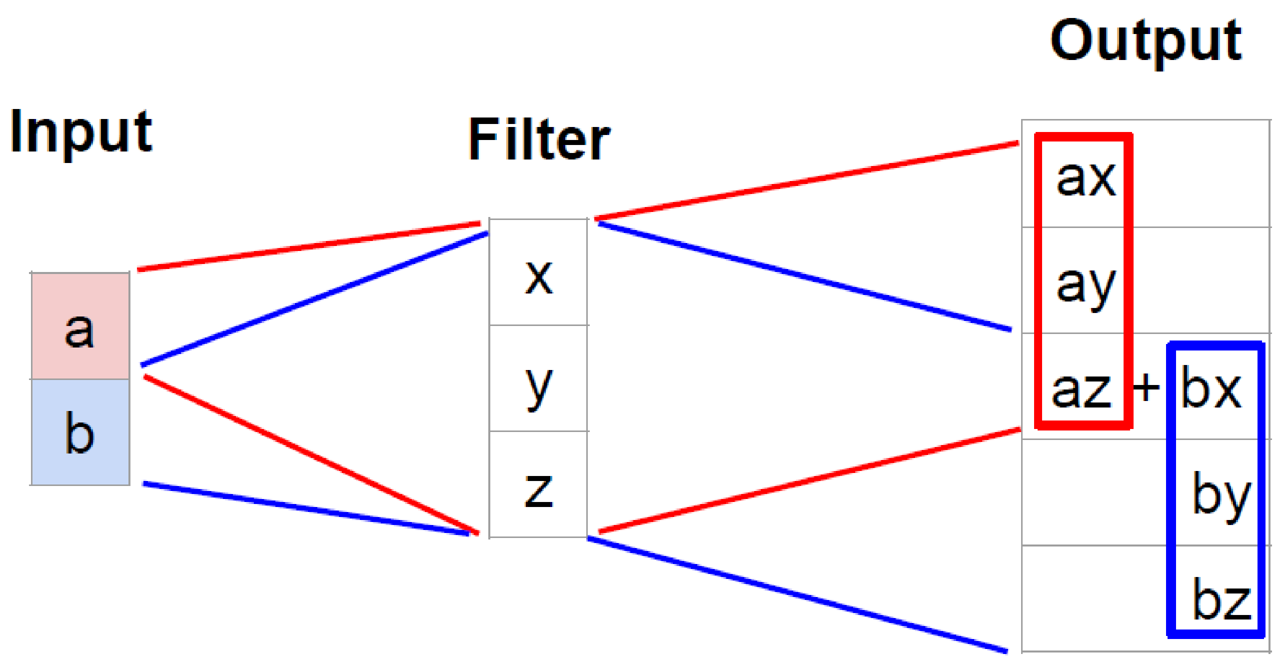
Output: 4 x 4

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
  - 1D example



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:

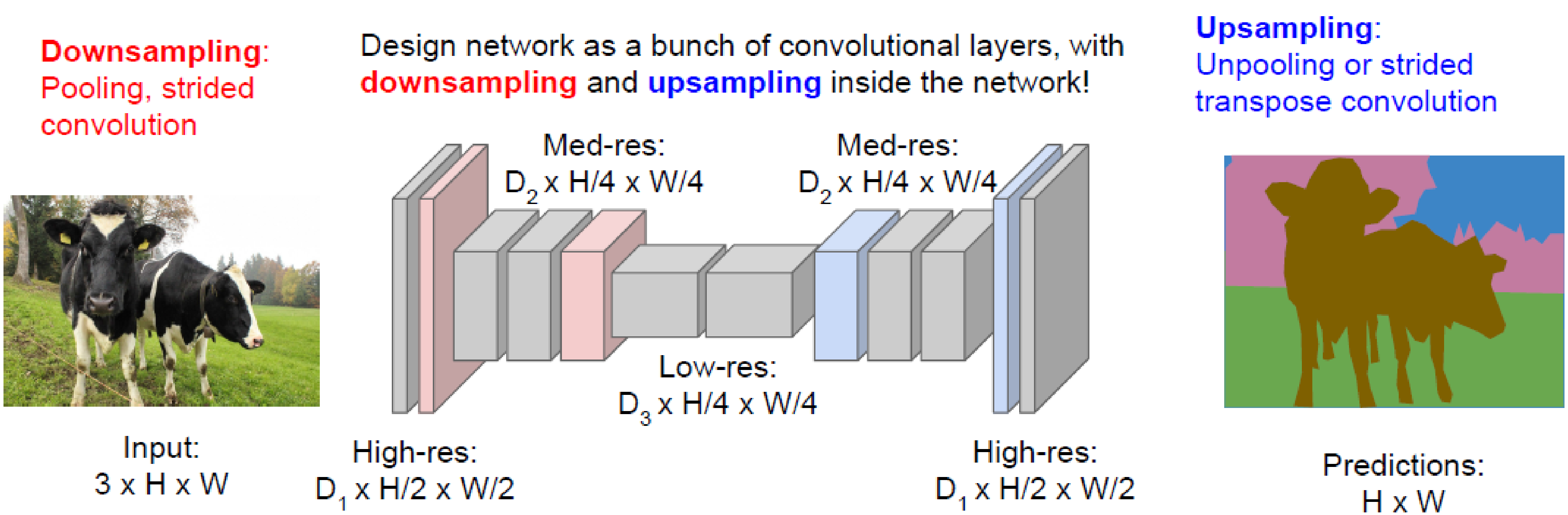
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

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

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

Fully Convolutional Networks (FCN)

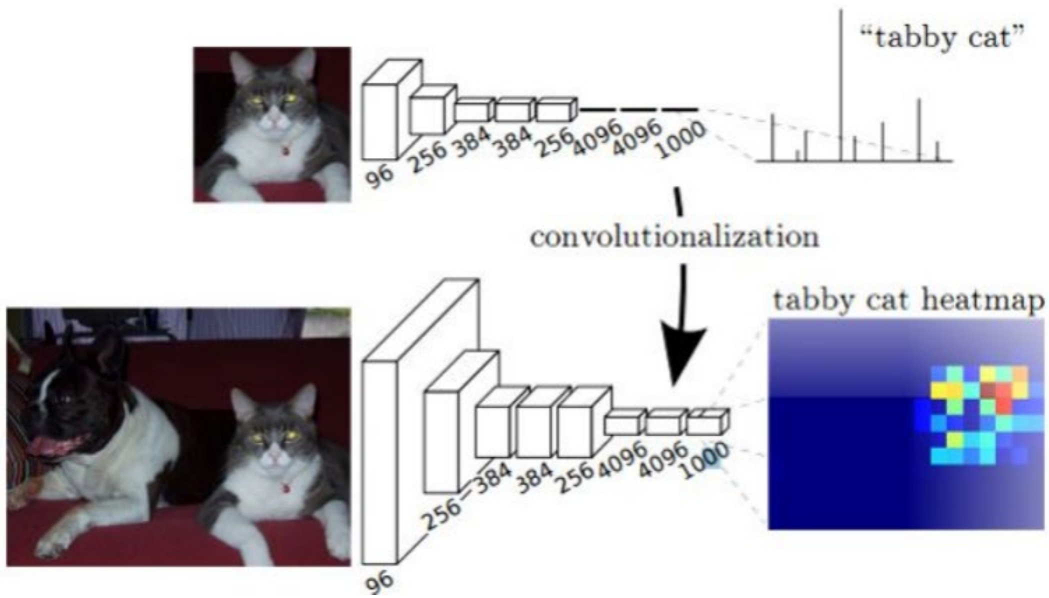
- Remarks
  - All layers are convolutional
  - End-to-end training



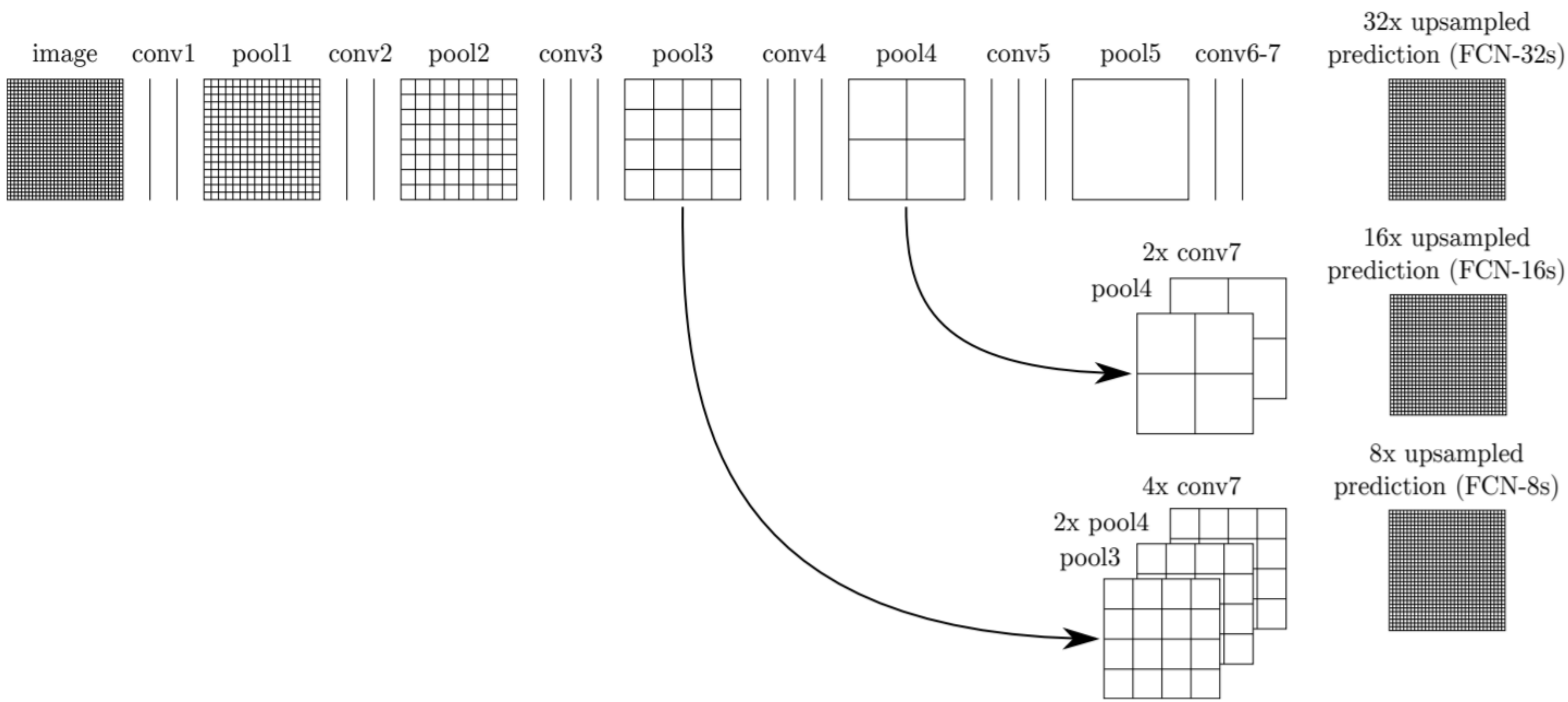
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

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



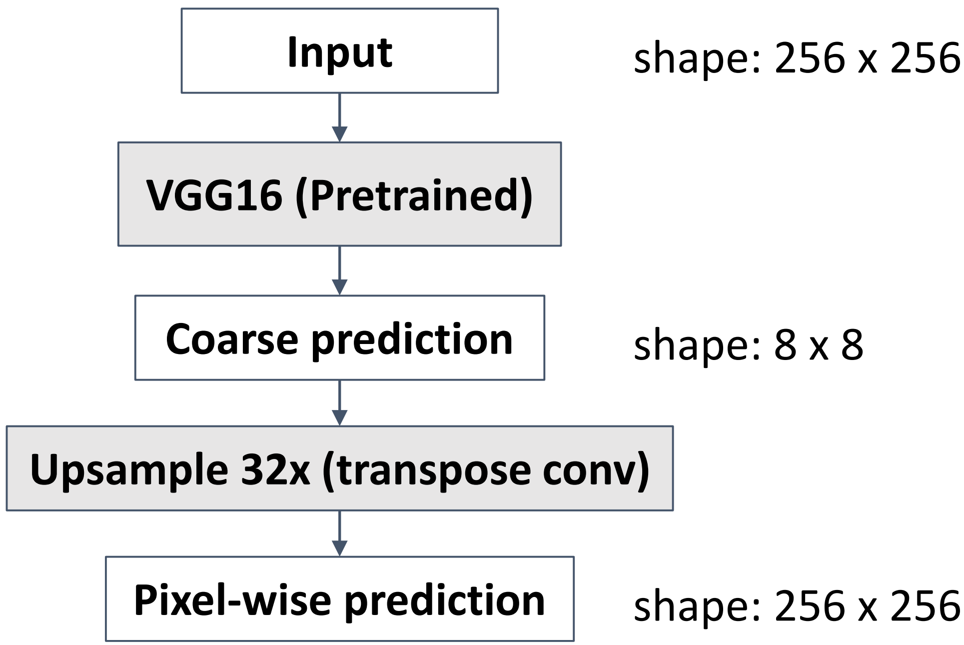
Fully Convolutional Networks (FCN)



Fully Convolutional Networks (FCN)

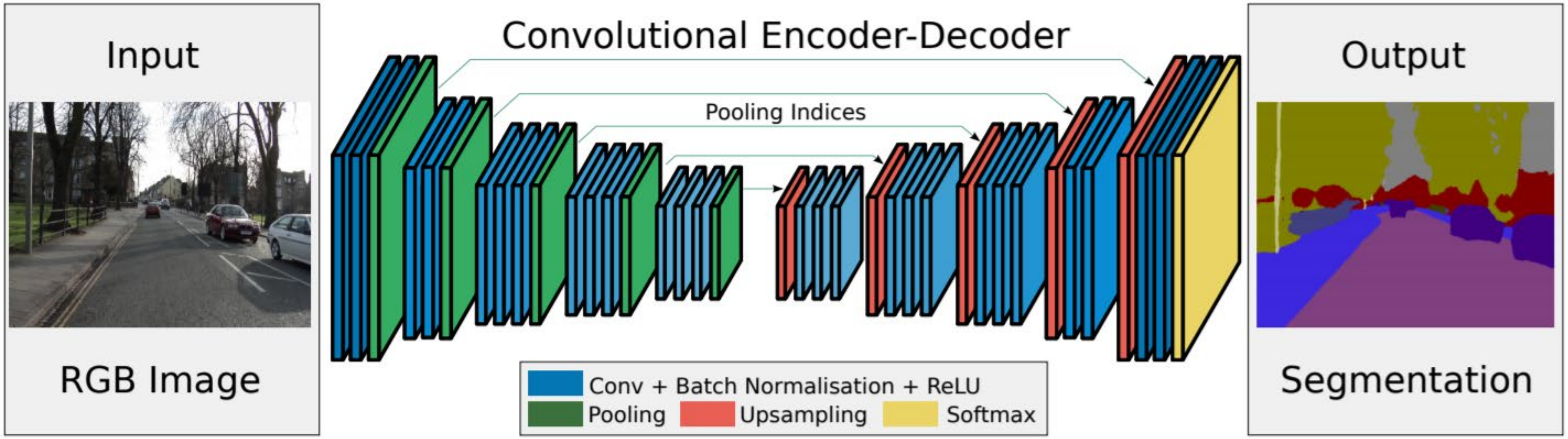
- Example
  - **VGG16-FCN32s**
  - Loss: pixel-wise cross-entropy

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



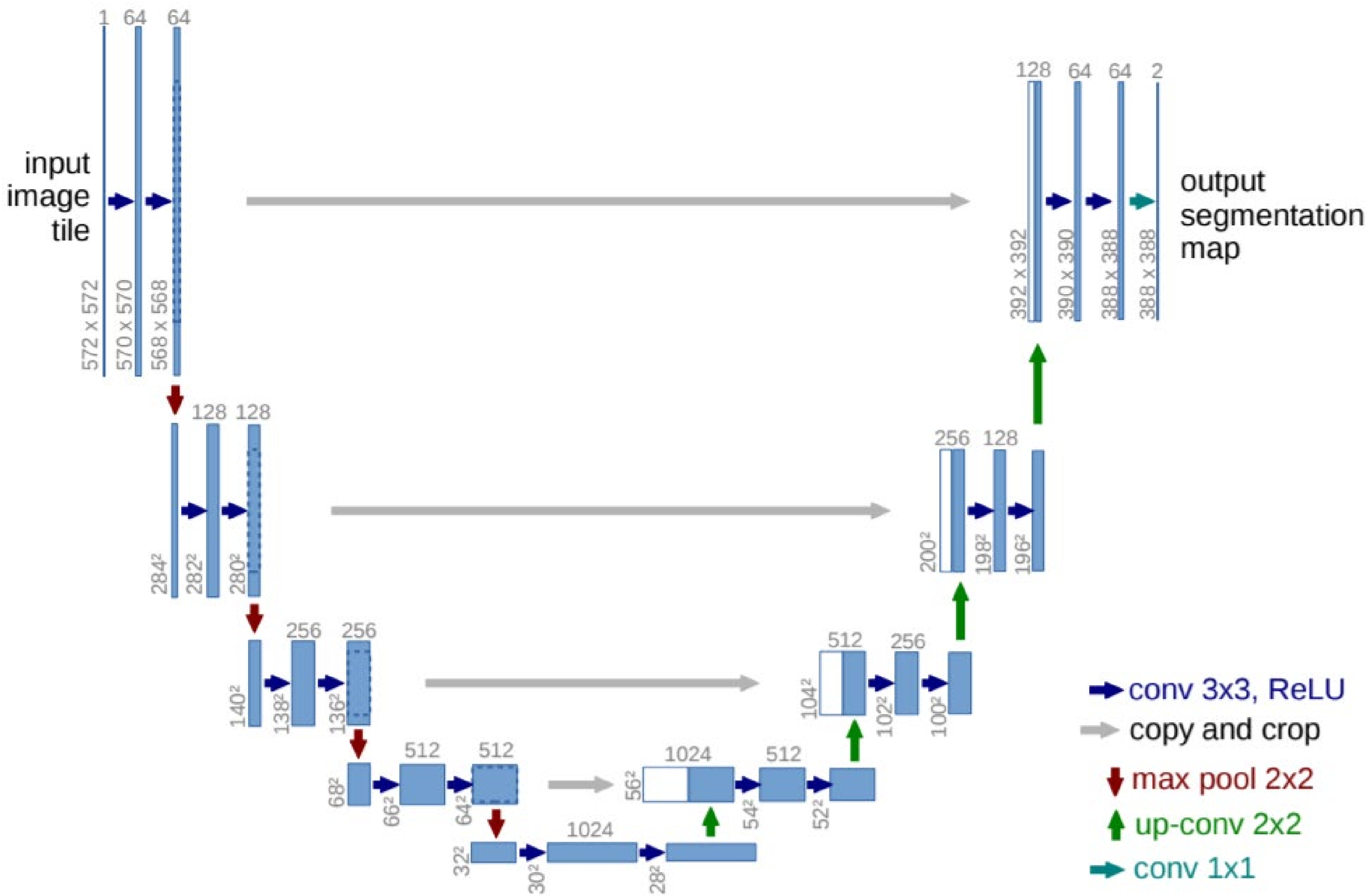
SegNet

- 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



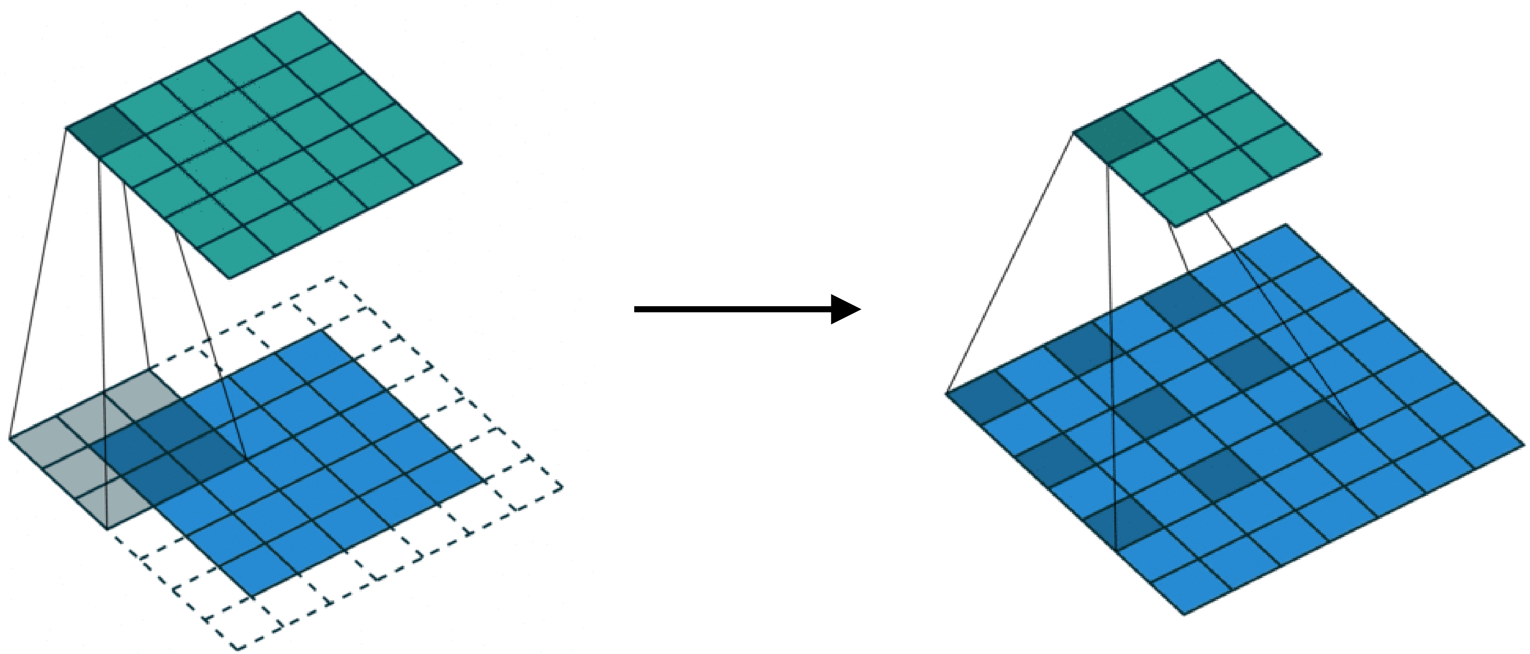
U-Net: Convolutional Networks for Biomedical Image Segmentation [link]

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

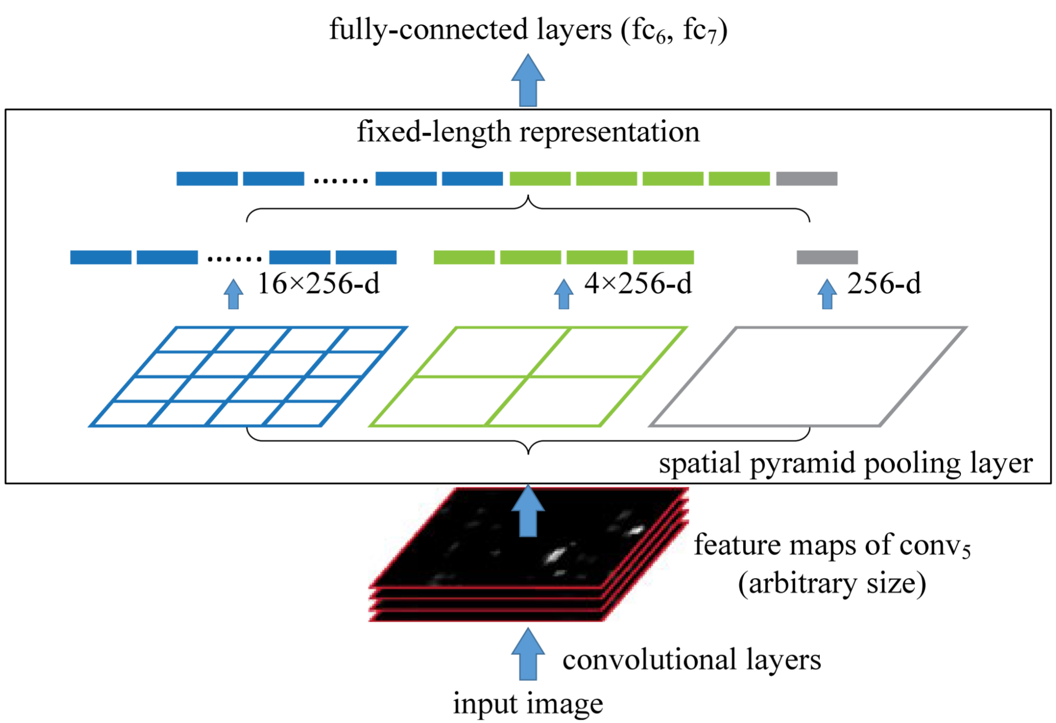
Recap (1)

- Atrous (Dilated) Convolution
  - Larger receptive field with the same kernel size



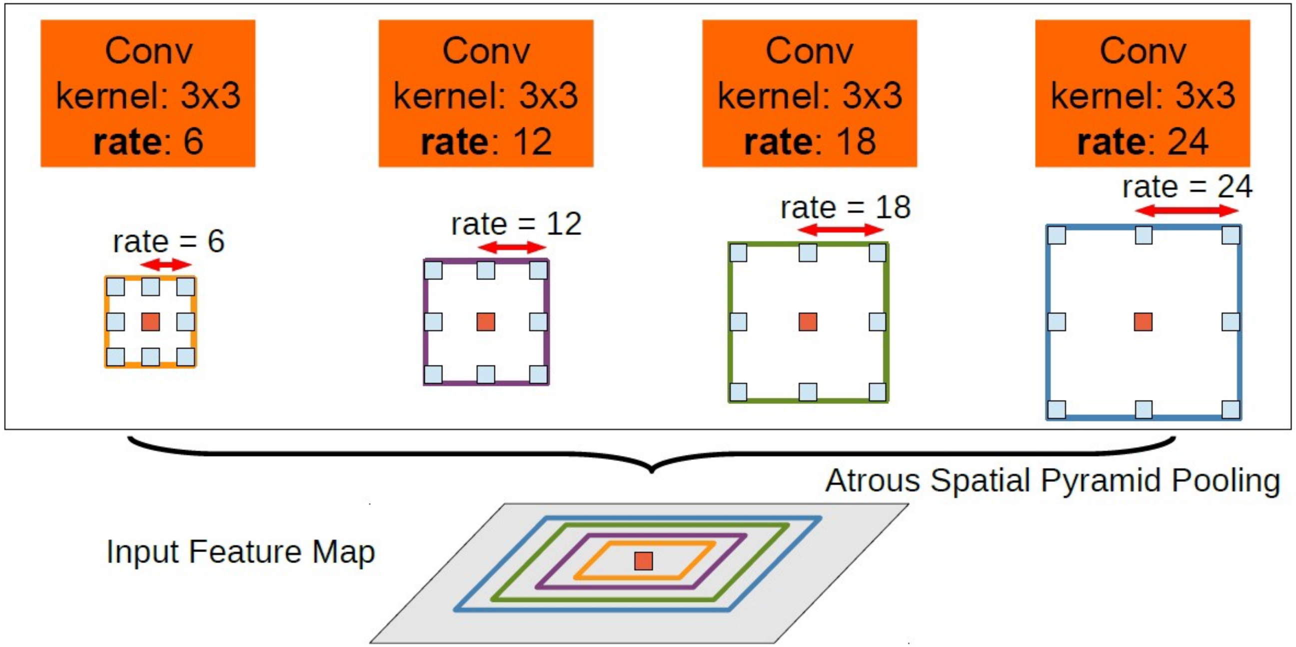
Recap (2)

- Spatial Pyramid Pooling
  - Integrating information viewed under different scales

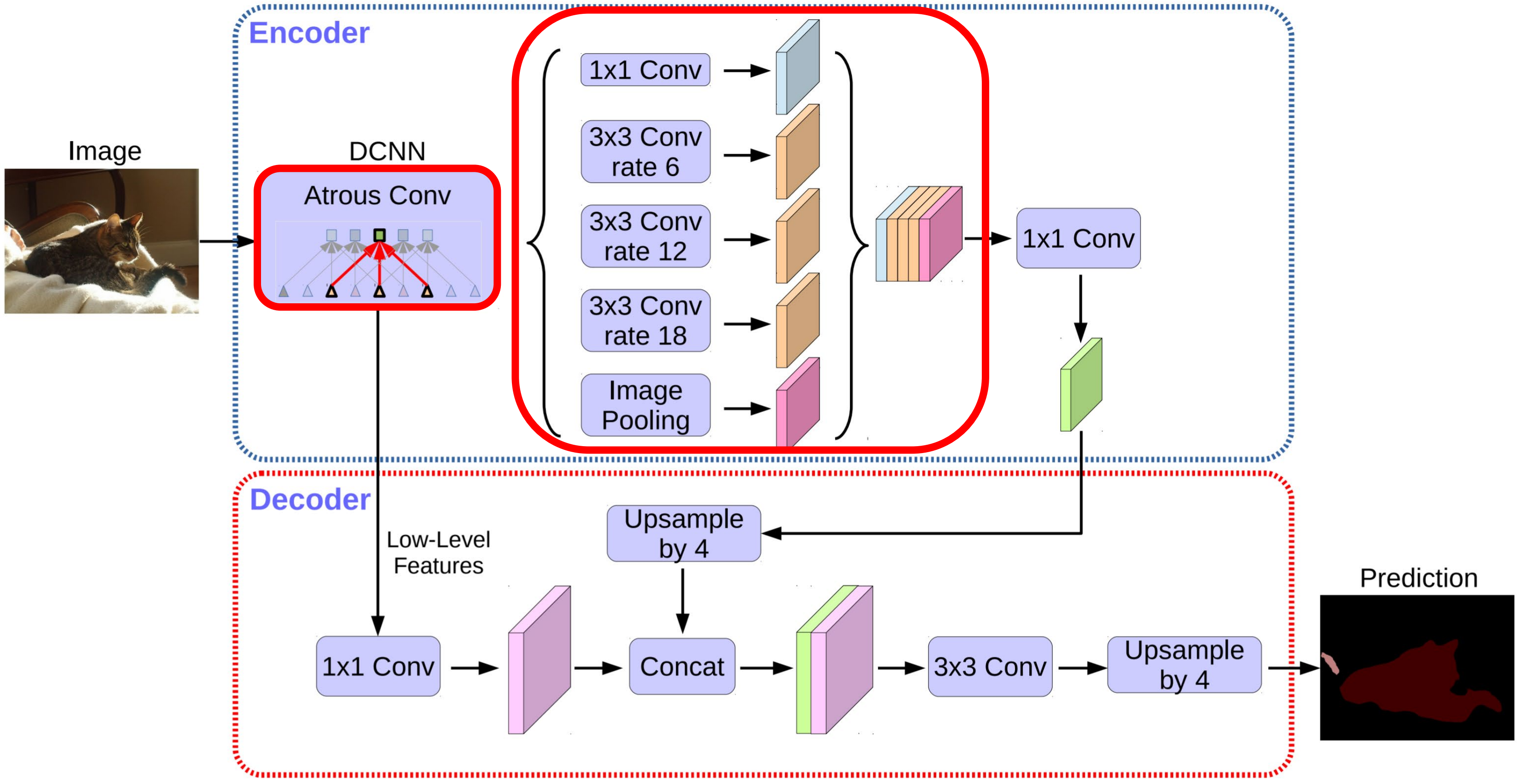


Thus, we have…

- Atrous Spatial Pyramid Pooling
  - Combines both techniques for producing enhanced spatial info



DeepLabv3+



Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation," *ECCV* 2018

What's to Be Covered Today…

- Convolutional Neural Networks
  - Properties of CNN
  - Selected variants of CNN
  - Training CNN
  - Visualizing CNN
- Segmentation
- HW #1 is out & due Oct. 10th Mon 23:59