Convolutional Neural Networks and Image Segmentation

Ivan
Sep. 3rd, 2017

Outline

- Review of Convolutional Neural Networks
- Introduction to AlexNet, VGG, GoogLeNet and ResNet
- Image Segmentation
- Semantic Segmentation with Fully Convolutional Neural Networks

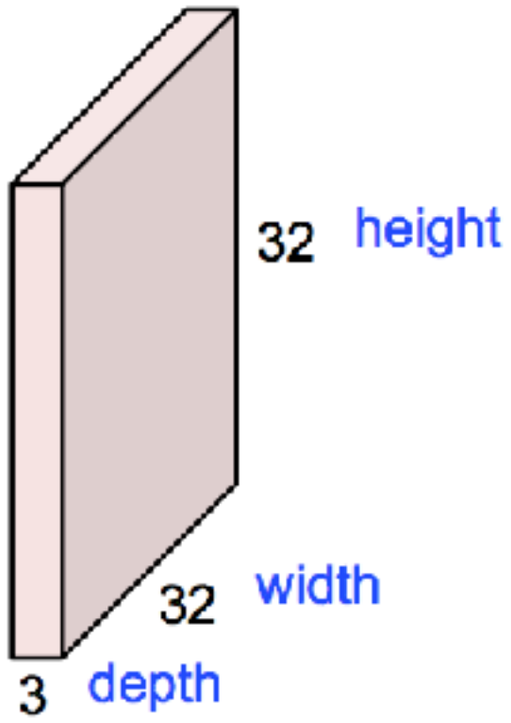
Convolutional Neural Networks

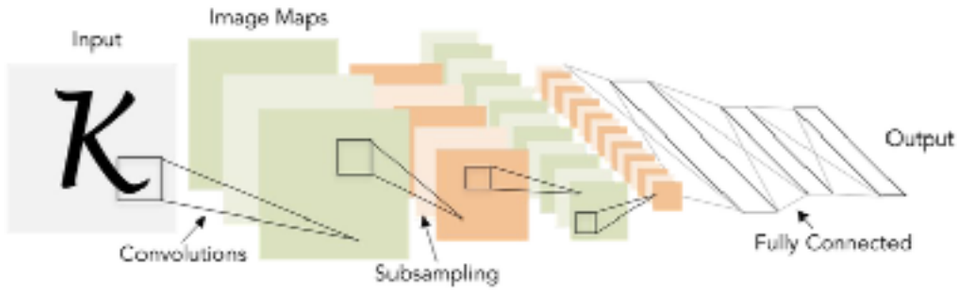
- CNN is everywhere in computer vision
- Object classification
- Object tracking
- Image captioning
- Image retrieval
- **Image segmentation**
- Style transfer
- ……

Convolutional Neural Networks

Convolution Layer

32x32x3 image -> preserve spatial structure

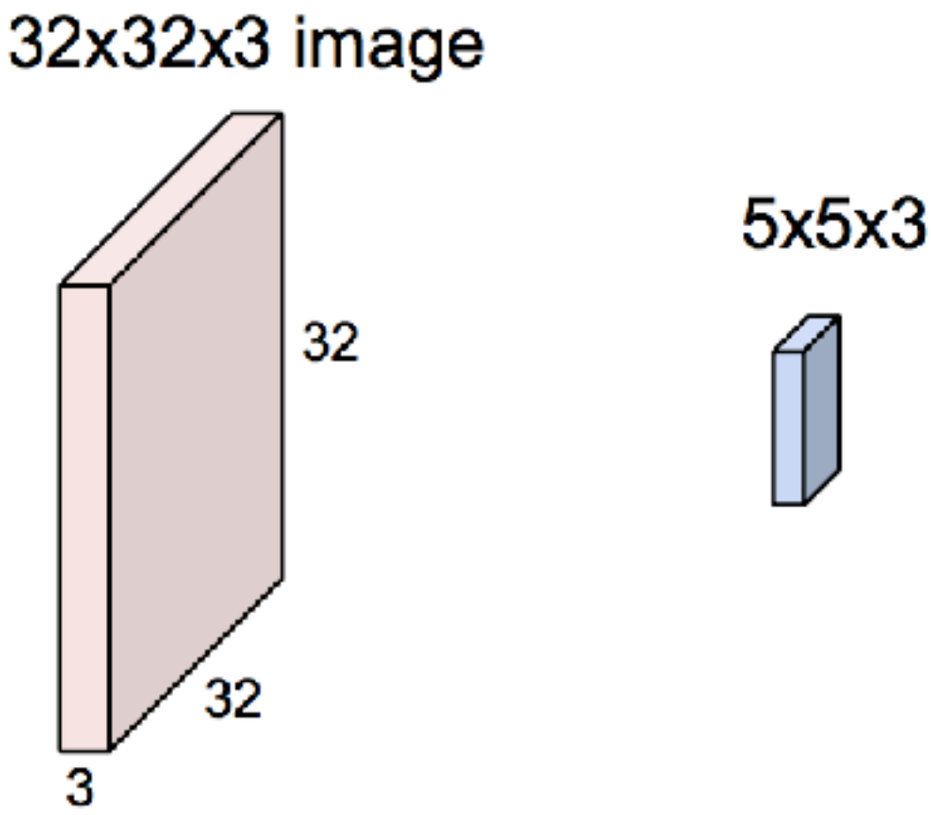




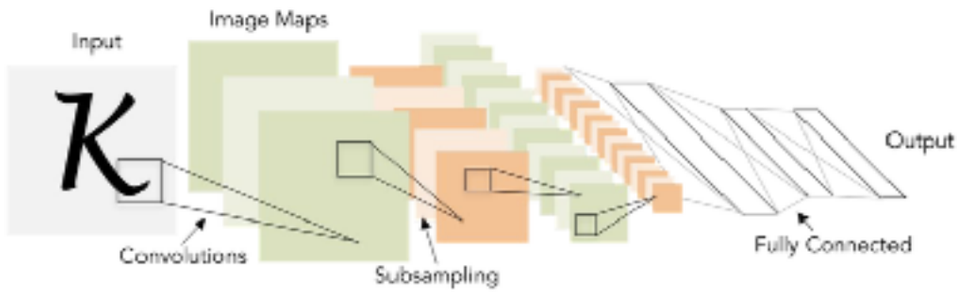
LeNet-5

Convolutional Neural Networks

Convolution Layer



Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"



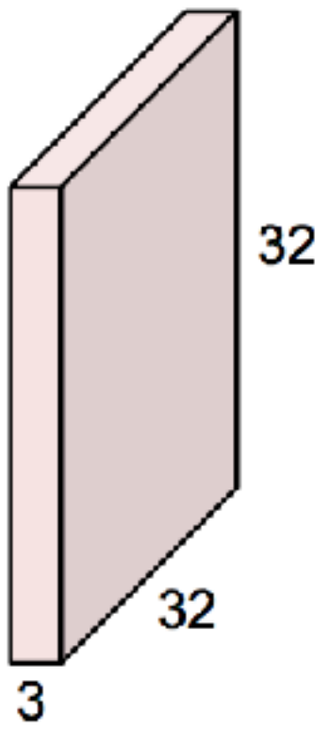
LeNet-5

Convolutional Neural Networks

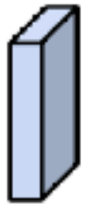
Convolution Layer

Filters always extend the full depth of the input volume

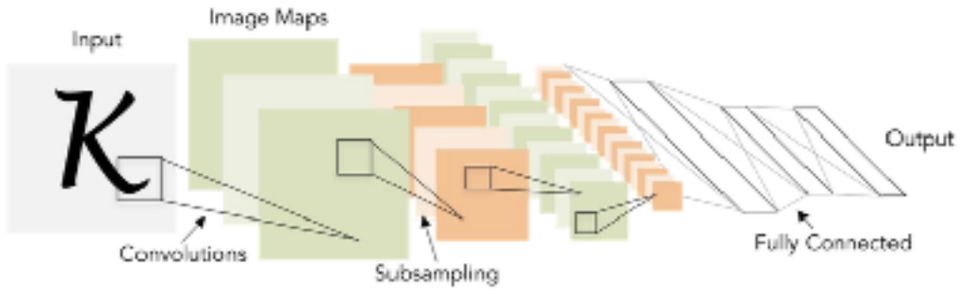
32x32x3 image



5x5x3 filter



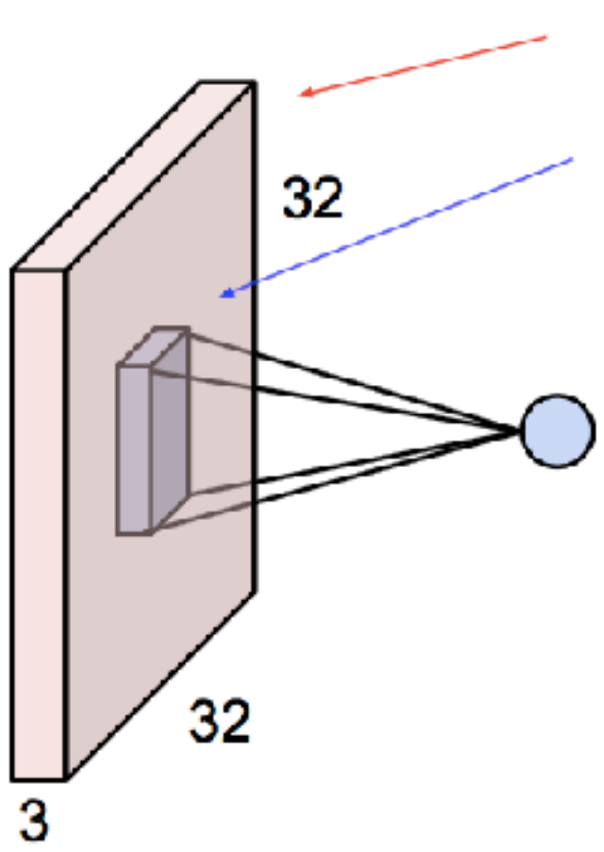
Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"



LeNet-5

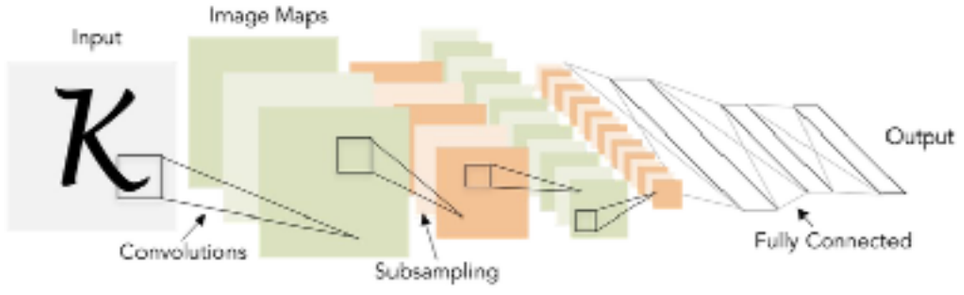
Convolutional Neural Networks

Convolution Layer



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

$$w^T x + b$$



LeNet-5

Convolutional Neural Networks

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	2	1	0	2	0
0	2	1	1	2	2	0
0	1	2	0	1	0	0
0	2	0	1	0	2	0
0	1	2	2	0	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	1	2	0	0
0	1	1	1	1	2	0
0	1	2	2	1	0	0
0	0	1	1	2	0	0
0	2	1	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	0	2	1	0
0	2	1	2	2	2	0
0	1	2	1	0	1	0
0	2	2	1	1	0	0
0	1	0	0	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	1	-1
-1	0	1
0	1	-1

w0[:,:,1]

1	-1	-1
0	1	-1
1	-1	1

w0[:,:,2]

0	1	1
1	-1	0
-1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	-1	1
0	0	0
1	0	-1

w1[:,:,1]

1	1	1
-1	0	-1
0	-1	-1

w1[:,:,2]

-1	-1	0
-1	1	-1
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

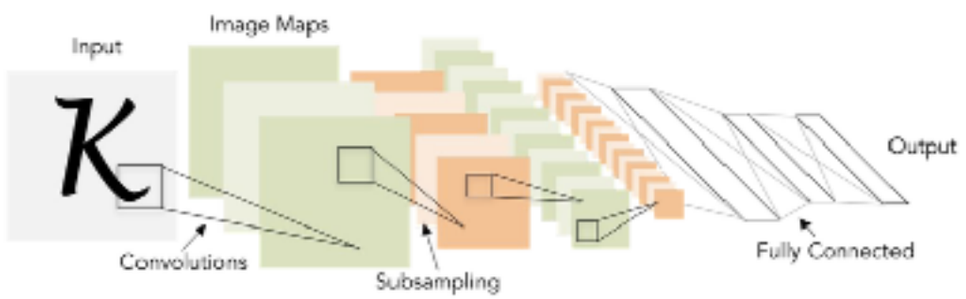
Output Volume (3x3x2)

o[:,:,0]

1	-6	-1
2	2	3
8	-2	5

o[:,:,1]

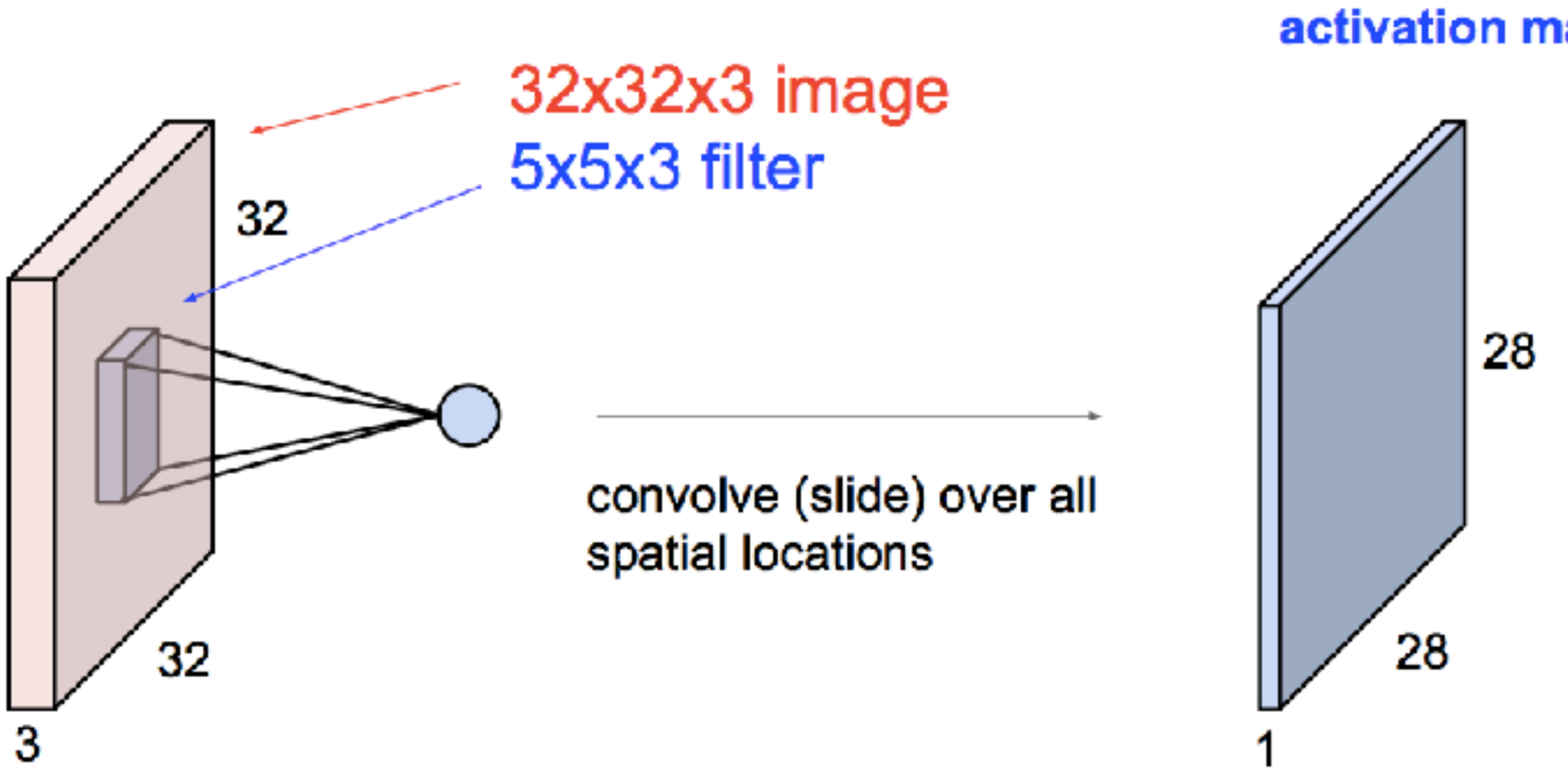
-2	-8	-3
-5	-5	-1
-3	-4	-3

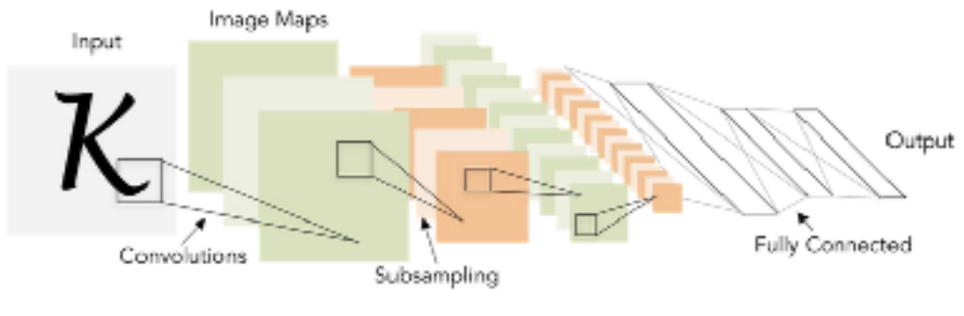


LeNet-5

Convolutional Neural Networks

Convolution Layer

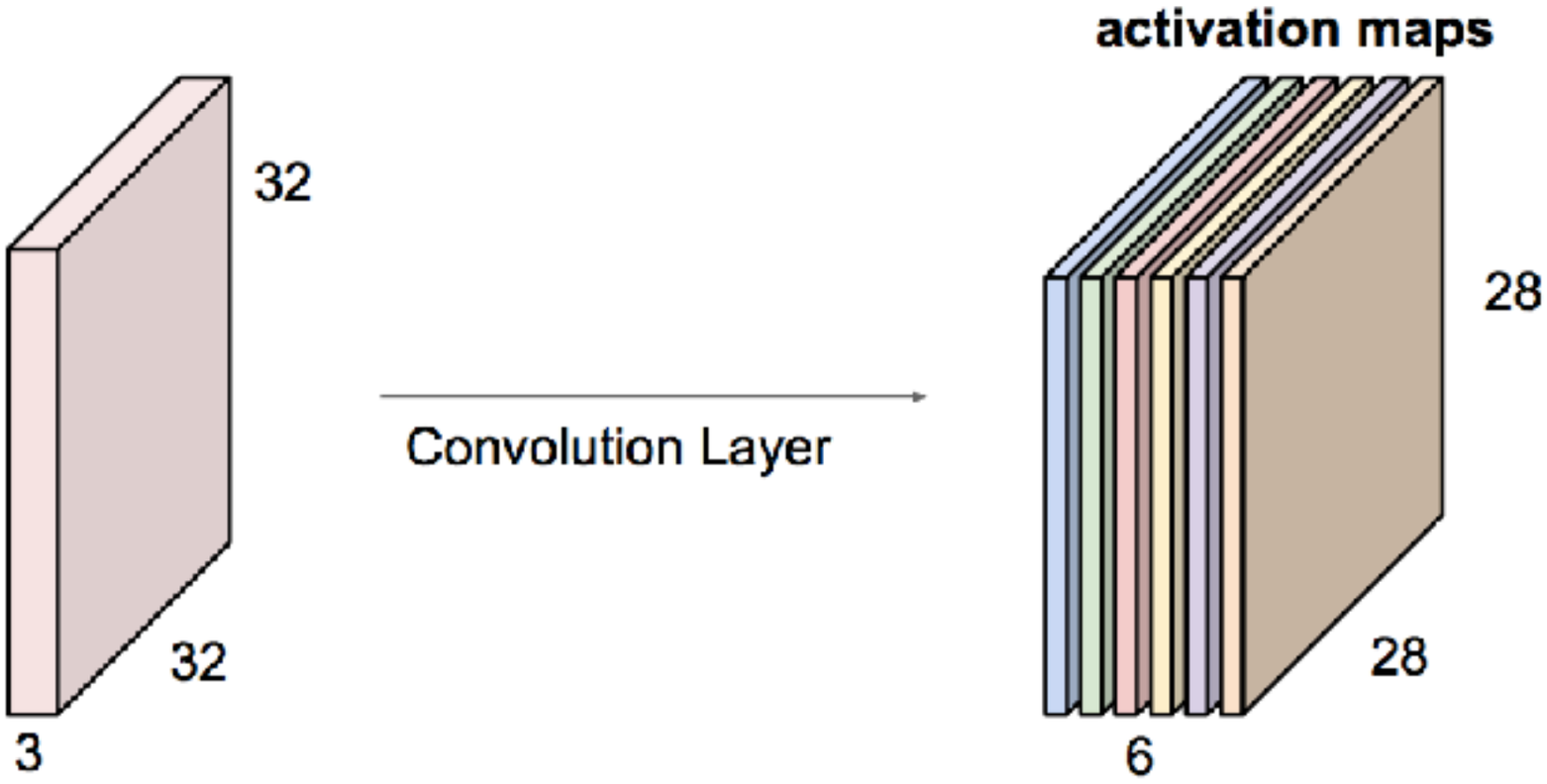




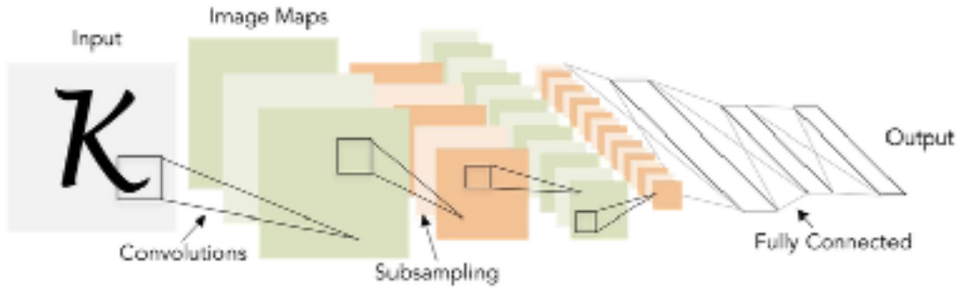
LeNet-5

Convolutional Neural Networks

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



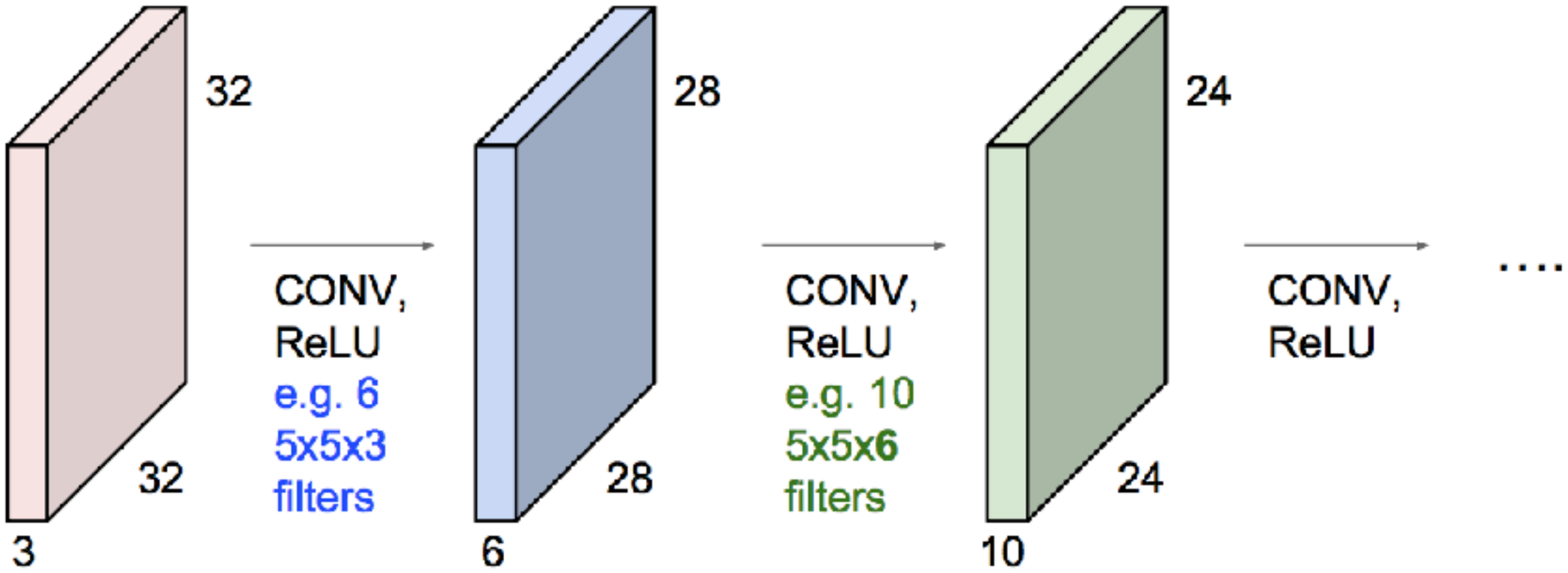
We stack these up to get a "new image" of size 28x28x6!

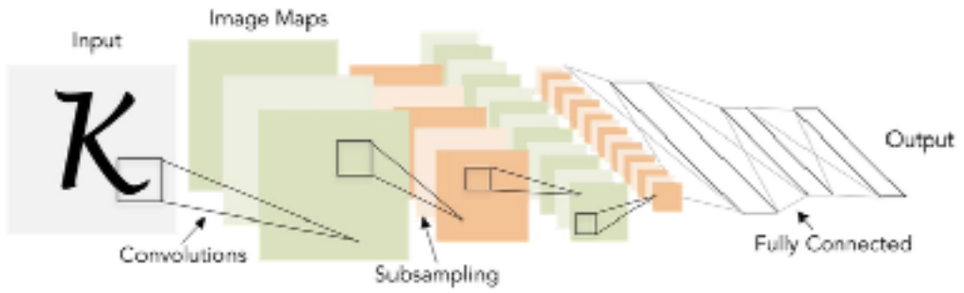


LeNet-5

Convolutional Neural Networks

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



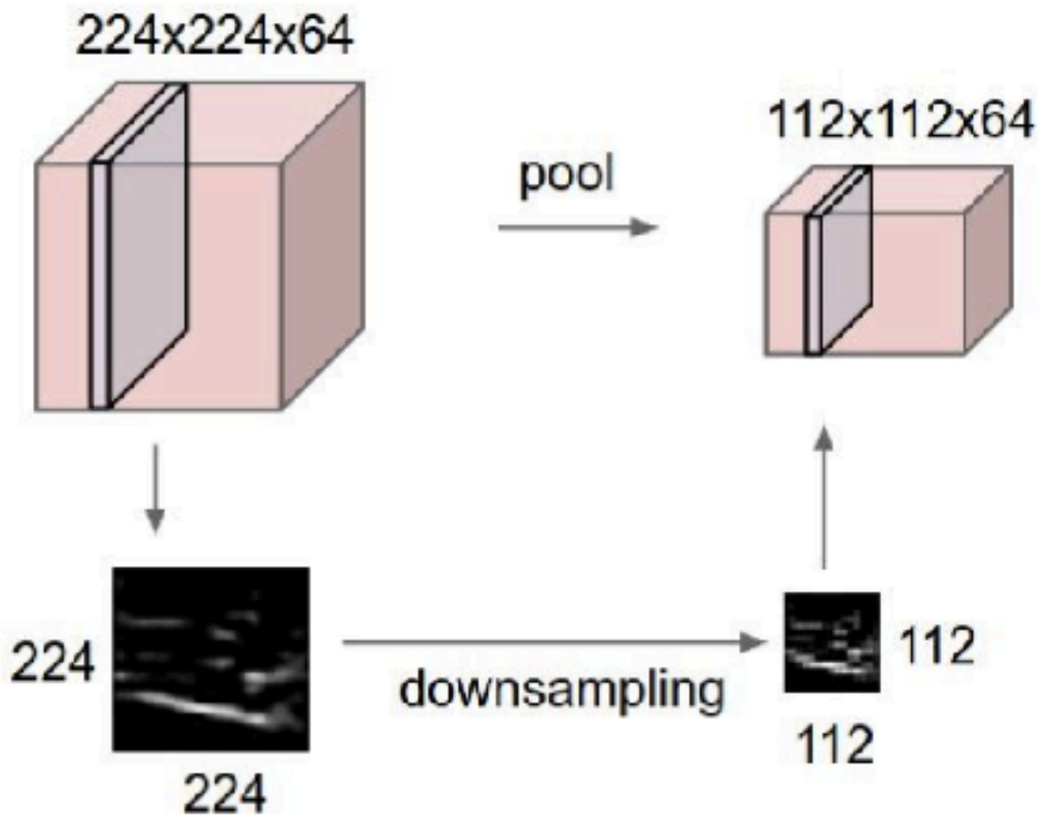


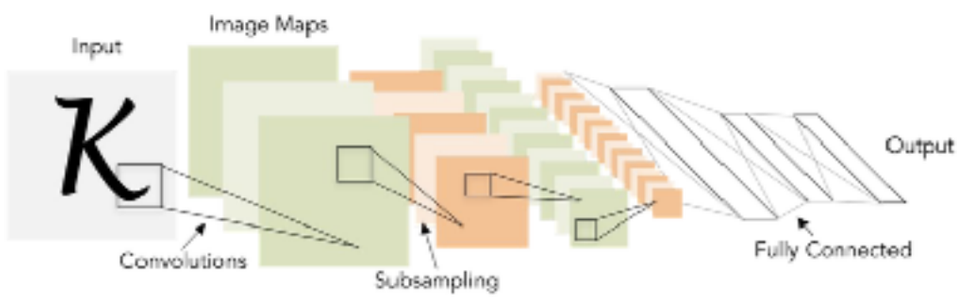
LeNet-5

Convolutional Neural Networks

Pooling layer

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





LeNet-5

Convolutional Neural Networks

MAX POOLING

Single depth slice

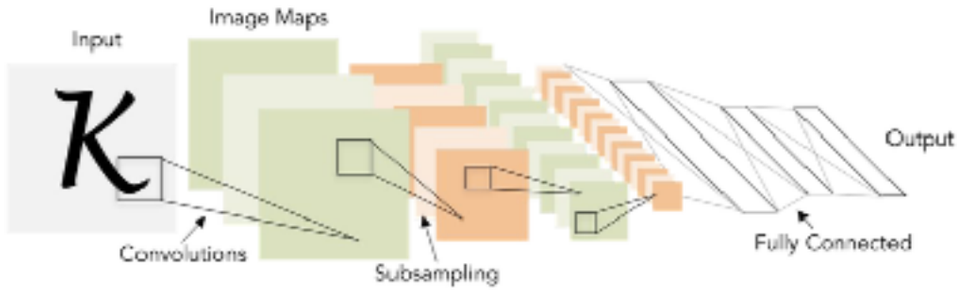
x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters and stride 2

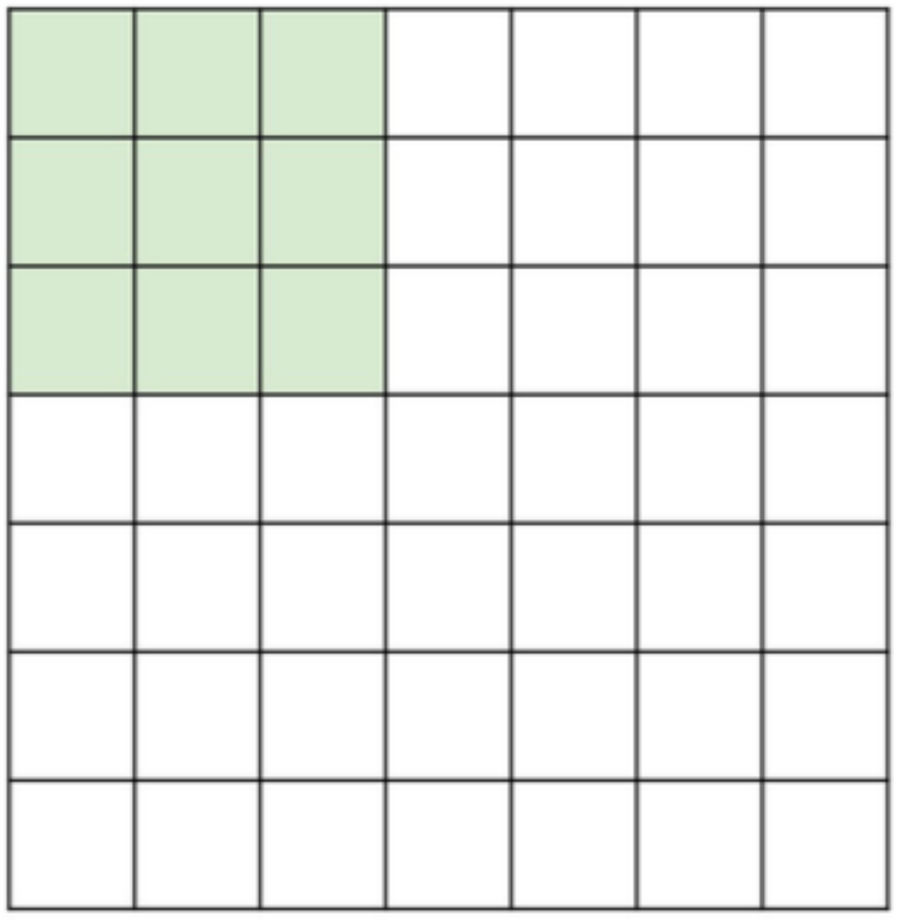
6	8
3	4

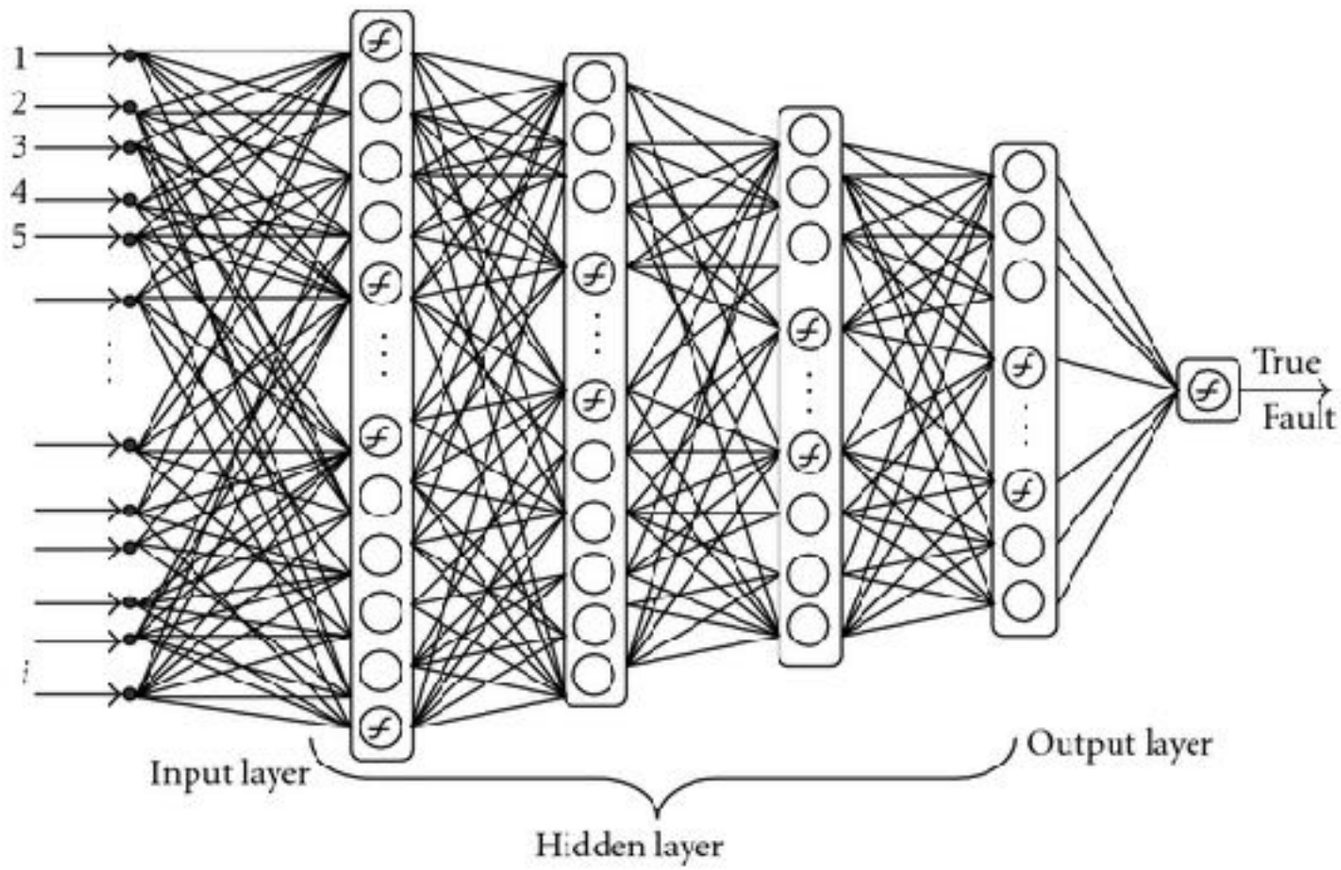


LeNet-5

Convolutional Neural Networks

- Reduction from CNN to MLP:

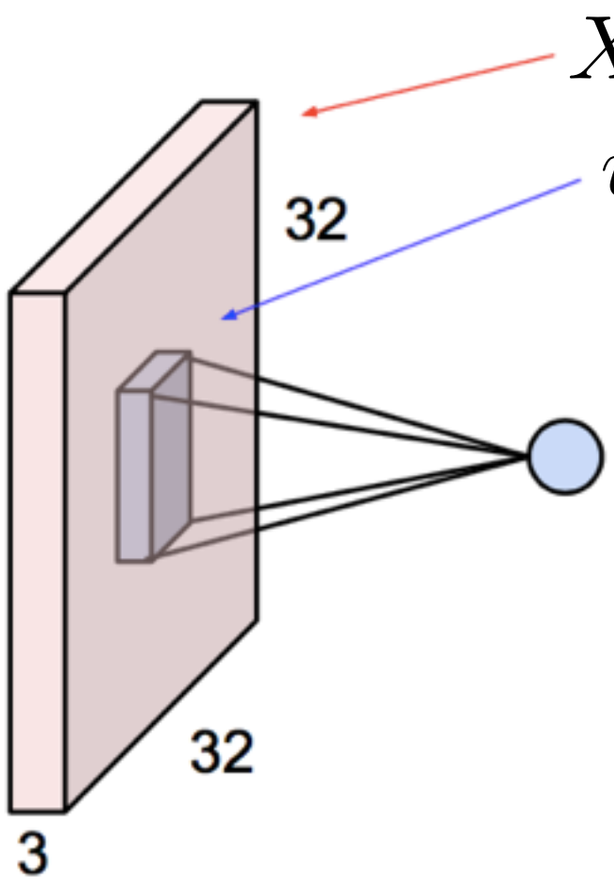




VS

Convolutional Neural Networks

- Reduction from CNN to MLP:

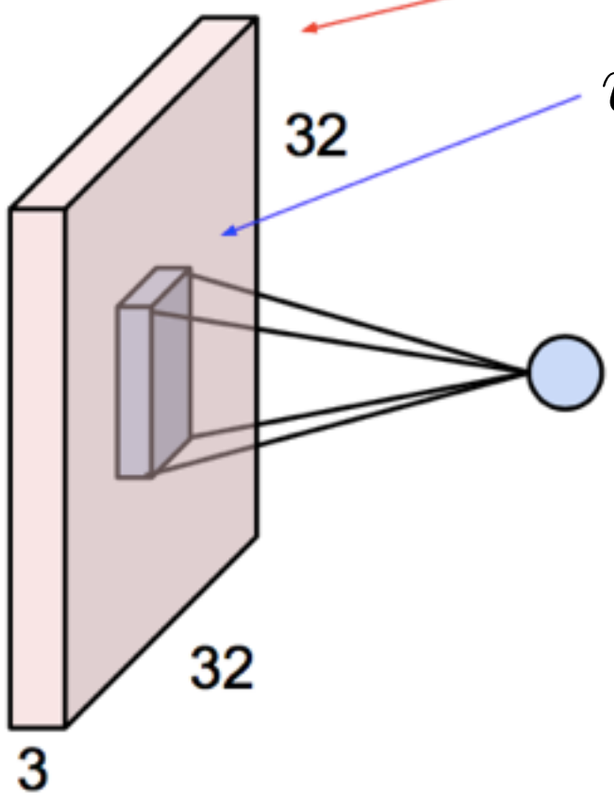


$X \in \mathbb{R}^{32\times32}, w \in \mathbb{R}^{5\times5}$

pre-activation of first local batch = $\sum_{i=1}^{5}\sum_{j=1}^{5} X_{ij}w_{ij}$

Convolutional Neural Networks

- Reduction from CNN to MLP:



$$X \in \mathbb{R}^{32\times32}, w \in \mathbb{R}^{5\times5}$$

$$W \in \mathbb{R}^{32\times32}$$

$$W_{1:5,1:5} = w, W_{6:32,6:32} = 0$$

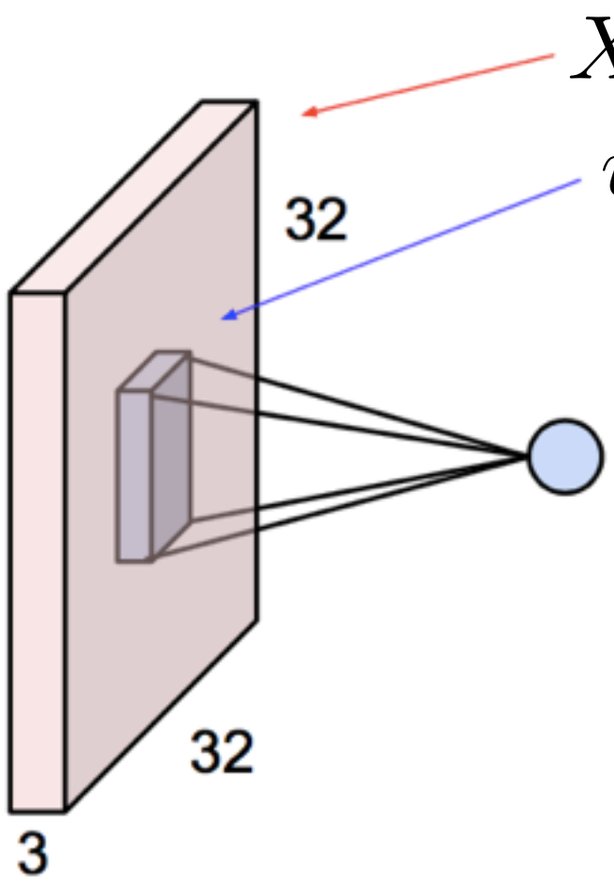
pre-activation of first local batch = $\sum_{i=1}^{5}\sum_{j=1}^{5} X_{ij}w_{ij}$

pre-activation of fully-connected unit =

$$\sum_{i=1}^{32}\sum_{j=1}^{32} X_{ij}W_{ij} = \sum_{i=1}^{5}\sum_{j=1}^{5} X_{ij}w_{ij}$$

Convolutional Neural Networks

- Reduction from CNN to MLP:



$$X \in \mathbb{R}^{32\times32}, w \in \mathbb{R}^{5\times5}$$

$$W \in \mathbb{R}^{32\times32}$$

$$W^{(1)}_{1:5,1:5} = w, W^{(1)}_{6:32,6:32} = 0$$
$$W^{(2)}_{1:5,2:6} = w, W^{(2)}_{others} = 0$$
$$\vdots$$
$$W^{(28\times28)}_{28:32,28:32} = w, W^{(28\times28)}_{1:27,1:27} = 0$$

Convolutional Neural Networks

- Backprop in CNN

- Since CNN is a special case of MLP, we should be able to derive back-propagation rules for CNN as well

- Define the convolution operation:

$$a = X * w$$

$$X \in \mathbb{R}^{32\times32}, w \in \mathbb{R}^{5\times5} \Rightarrow a \in \mathbb{R}^{28\times28}$$

Convolutional Neural Networks

- Backprop in CNN:
- Suppose we have $\nabla_Y \ell = \frac{\partial \ell}{\partial Y} \in \mathbb{R}^{y \times y}$, where

$$Y = g(a) = g(w * X), \quad X \in \mathbb{R}^{p \times p}, w \in \mathbb{R}^{k \times k}, y = p - k + 1$$

- How to compute $\nabla_w \ell = \frac{\partial \ell}{\partial w} \in \mathbb{R}^{k \times k}$?

Convolutional Neural Networks

- Backprop in CNN:
- Suppose we have $\nabla_Y \ell = \frac{\partial \ell}{\partial Y} \in \mathbb{R}^{y \times y}$, where

$$Y = g(a) = g(w * X), \quad X \in \mathbb{R}^{p \times p}, w \in \mathbb{R}^{k \times k}, y = p - k + 1$$

- How to compute $\nabla_w \ell = \frac{\partial \ell}{\partial w} \in \mathbb{R}^{k \times k}$?

$$\frac{\partial \ell}{\partial w} = \sum_{i,j=1}^{y} \frac{\partial \ell}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial a_{ij}} \frac{\partial a_{ij}}{\partial w} = \sum_{i,j=1}^{p} \nabla_{Y_{ij}} \ell \cdot g'(a_{ij}) \cdot X_{ij}$$

where X_{ij} is the i,j th local image patch in X of size k-by-k.

Convolutional Neural Networks

- Backprop in CNN:
- Suppose we have $\nabla_Y \ell = \frac{\partial \ell}{\partial Y} \in \mathbb{R}^{y \times y}$, where

$$Y = g(a) = g(w * X), \quad X \in \mathbb{R}^{p \times p}, w \in \mathbb{R}^{k \times k}, y = p - k + 1$$

- How to compute $\nabla_w \ell = \frac{\partial \ell}{\partial w} \in \mathbb{R}^{k \times k}$?

$$\frac{\partial \ell}{\partial w} = \sum_{i,j=1}^{y} \frac{\partial \ell}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial a_{ij}} \frac{\partial a_{ij}}{\partial w} = \sum_{i,j=1}^{p} \nabla_{Y_{ij}} \ell \cdot g'(a_{ij}) \cdot X_{ij}$$

where X_{ij} is the i,j th local image patch in X of size k-by-k.

Put everything into matrix form:

$$\nabla_w \ell = X * (\nabla_Y \ell \odot g'(a)) \in \mathbb{R}^{k \times k}$$

Back-prop can be computed using convolution!

Convolutional Neural Networks

- Summary and Practical Tricks:

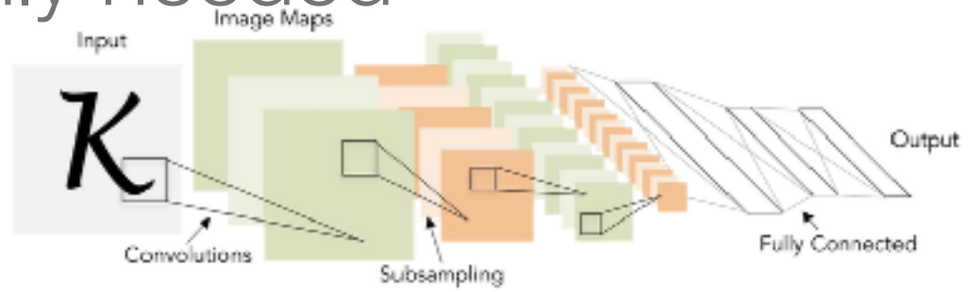
- Convolution $*$ is a linear operator

- Convolution is a special case of MLP with sparse connection and weight sharing

- CNN is a stack of Conv, Pool and Fully-connected layers

- Usually deeper models help

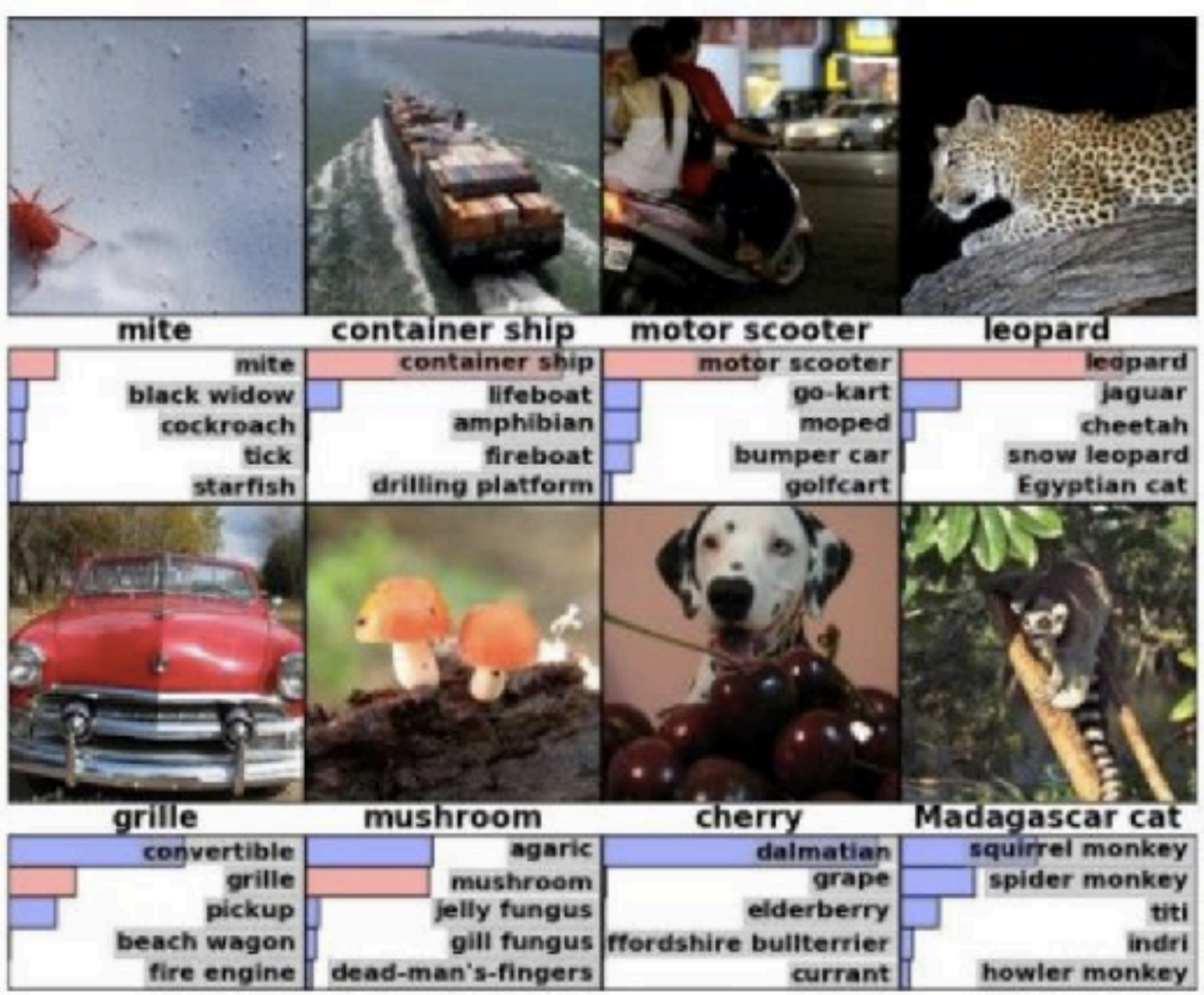
- Pooling layers are not necessarily needed



LeNet-5

Popular CNN Architectures

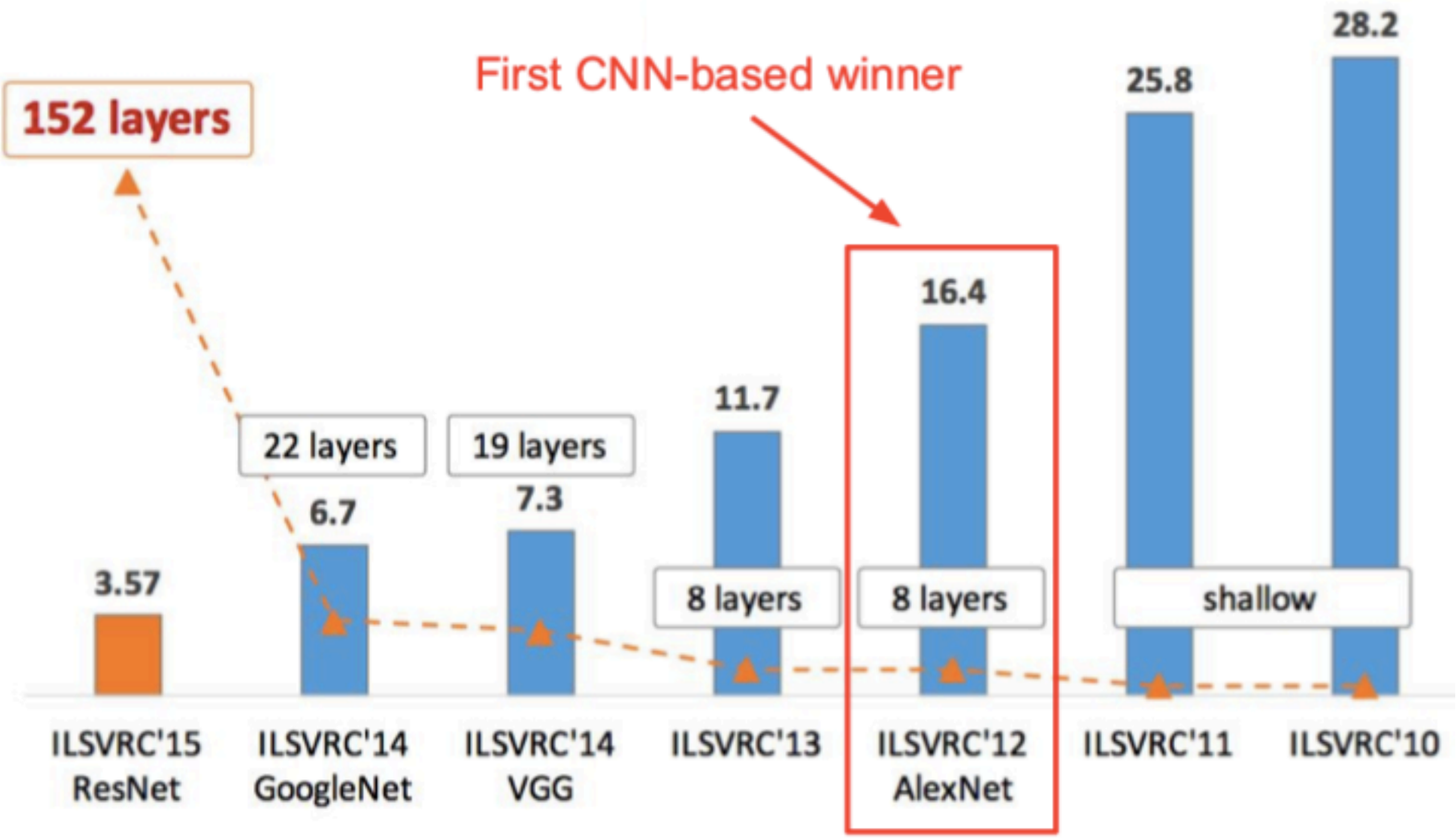
- ImageNet competition: 1000 classes, ~1.4M images in total, normalized to 256x256x3 size



http://www.image-net.org/

Popular CNN Architectures

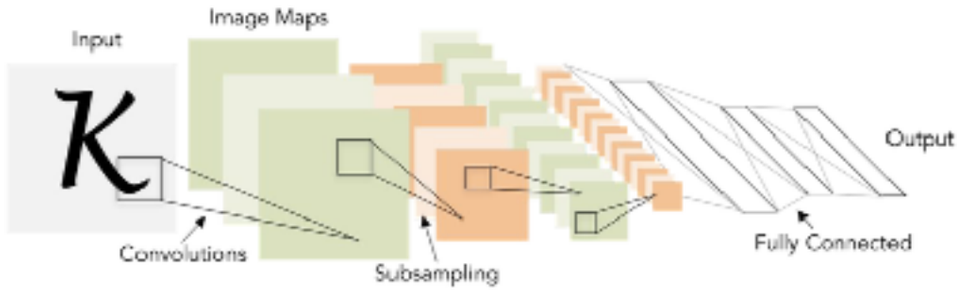
- ImageNet competition: 1000 classes, ~1.4M images in total, normalized to 256x256x3 size



Human performance on ImageNet: 5.1%

Popular CNN Architectures

- AlexNet (2012 ImageNet Champion)
- VGG (2014 ImageNet Champion)
- GoogLeNet (2014 ImageNet Champion)
- ResNet (2015 ImageNet Champion)

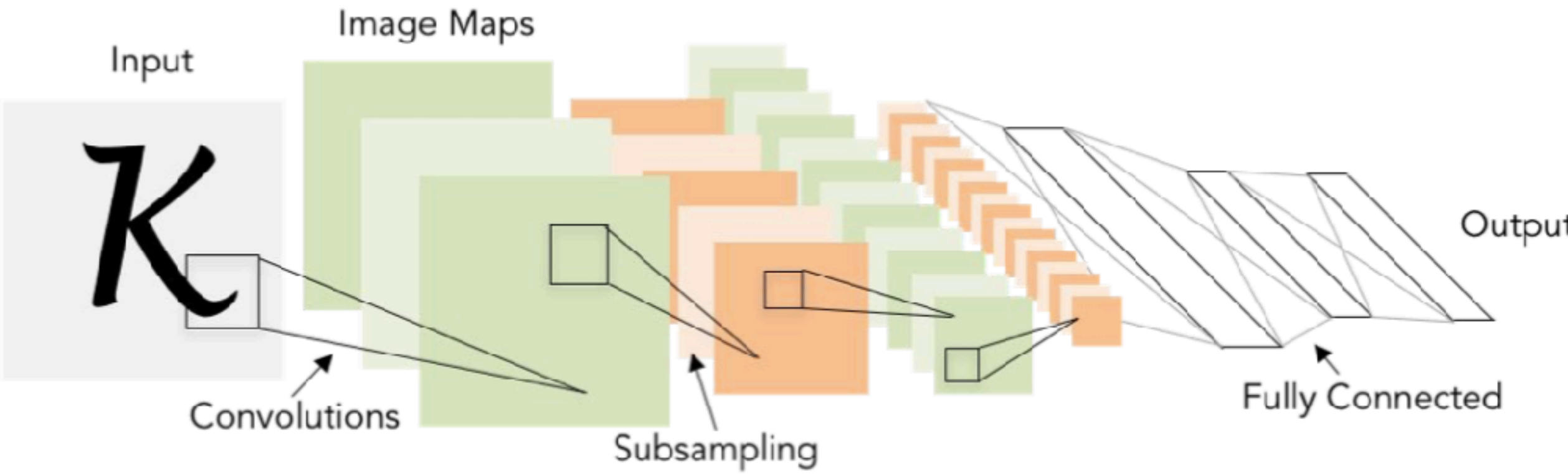


LeNet-5

Popular CNN Architectures

Review: LeNet-5

[LeCun et al., 1998]

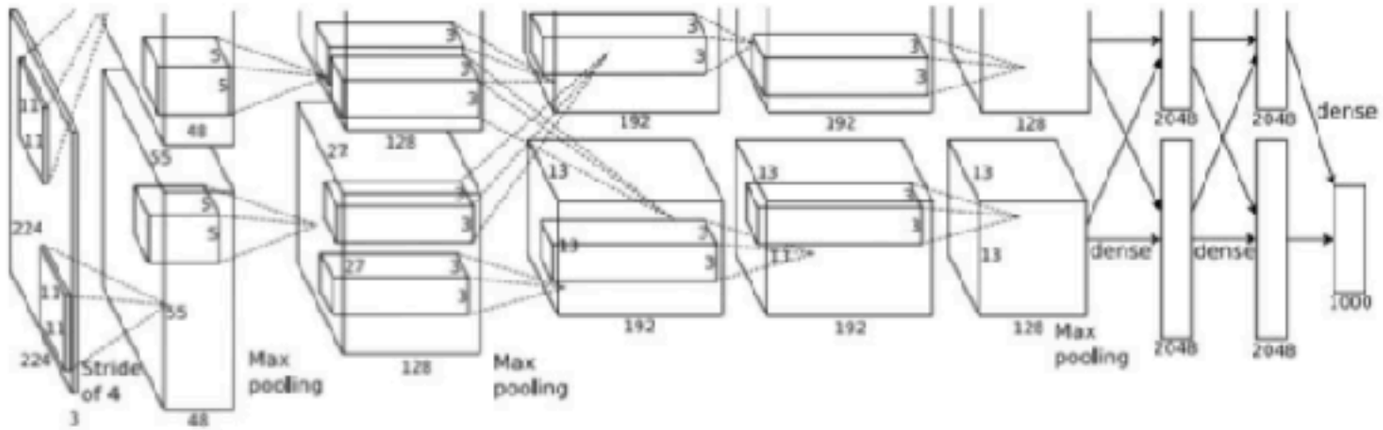


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Popular CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



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)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Popular CNN Architectures

Case Study: VGGNet

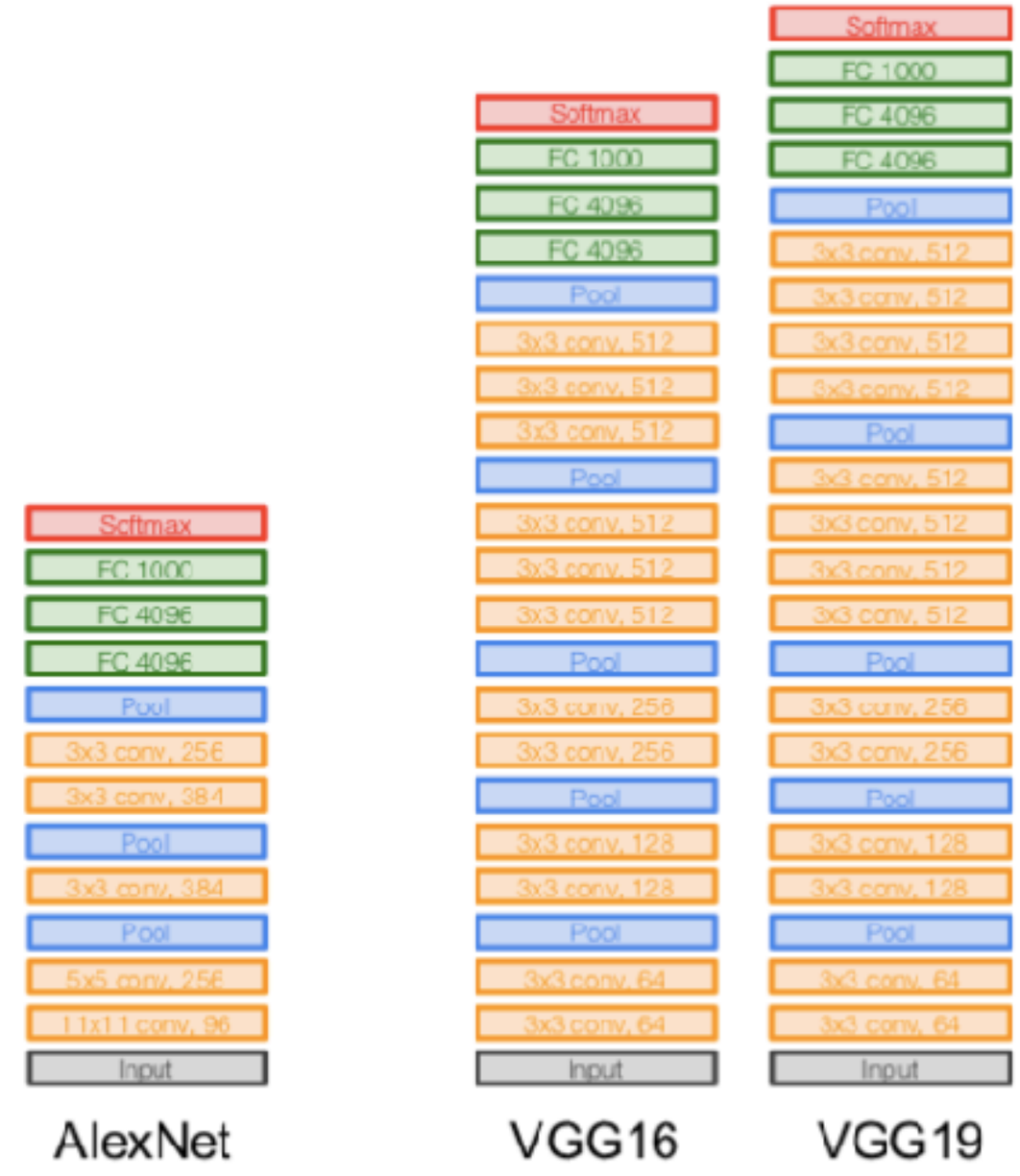
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



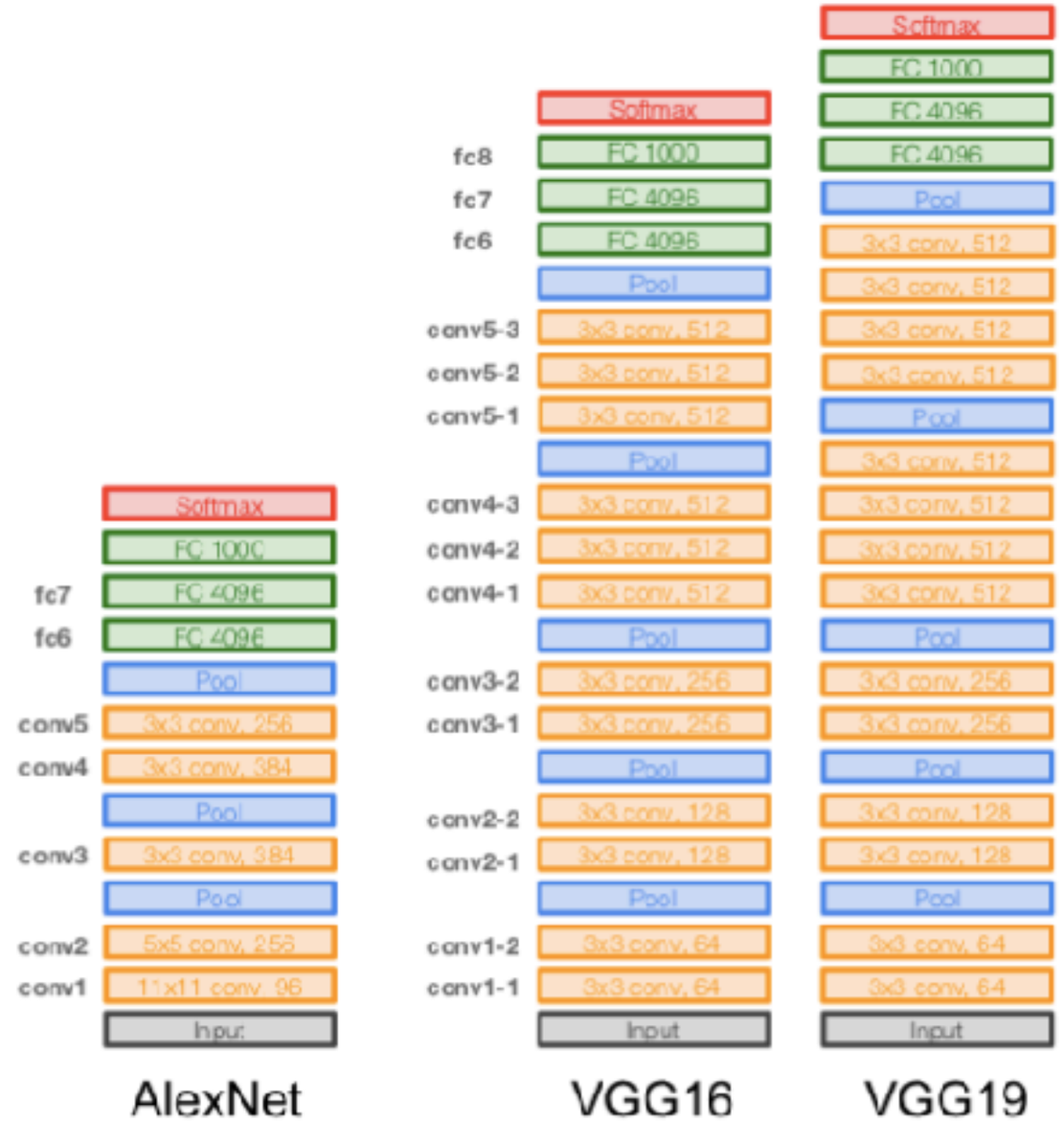
Popular CNN Architectures

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



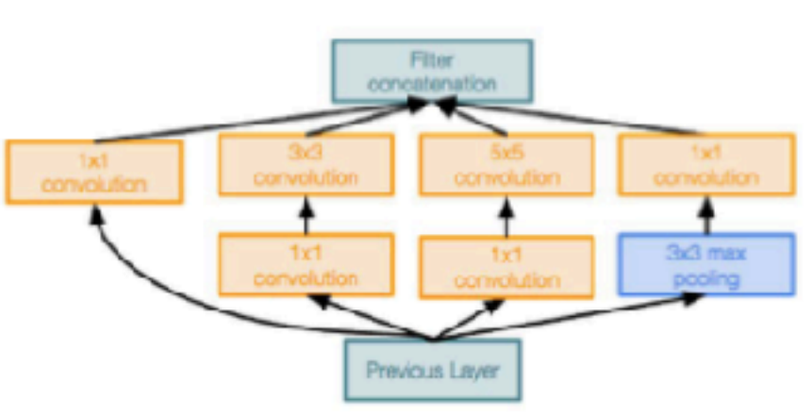
Popular CNN Architectures

Case Study: GoogLeNet

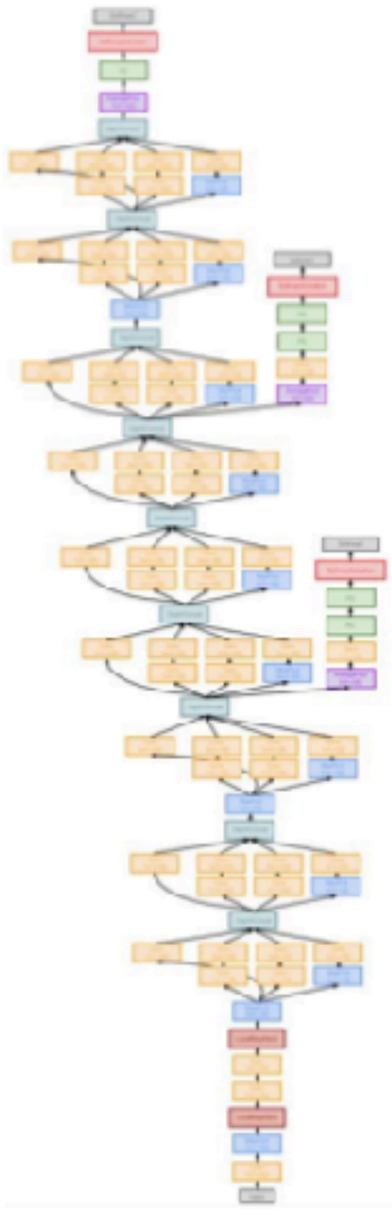
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)



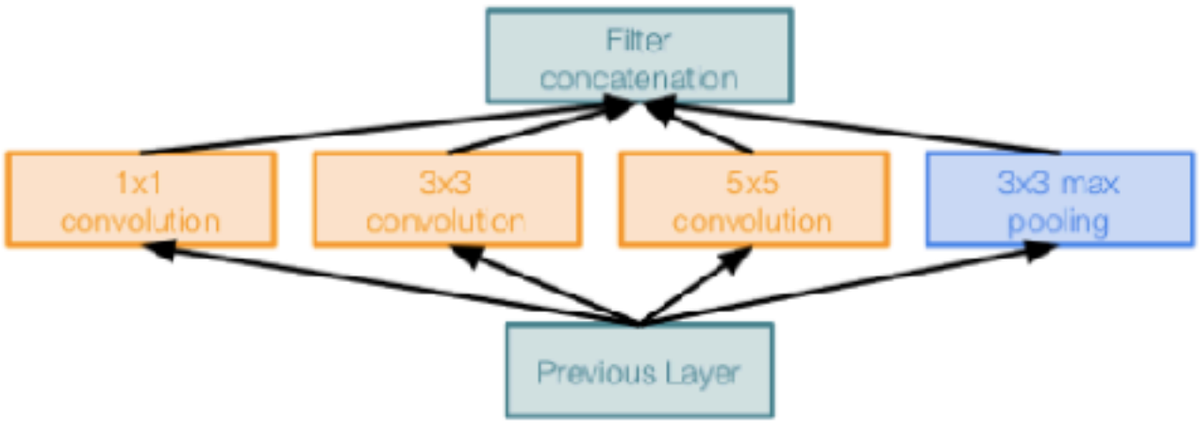
Inception module



Popular CNN Architectures

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

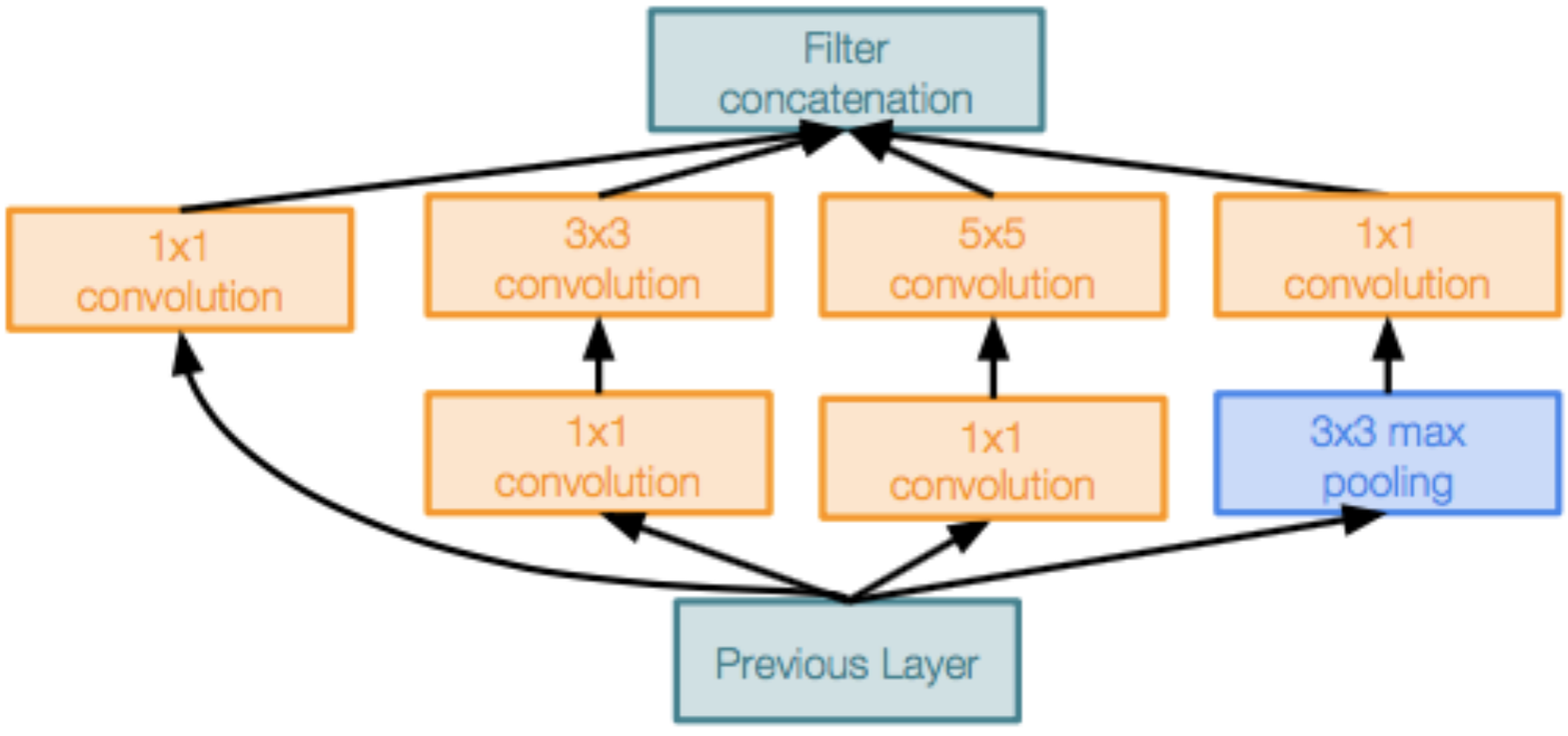
Popular CNN Architectures

- Key: 1x1 convolution for dimensionality reduction

Compression Ratio = 64 / 32 = 2: $\frac{\text{depth of input}}{\text{number of 1x1 convs}}$

Popular CNN Architectures

- Key: 1x1 convolution for dimensionality reduction

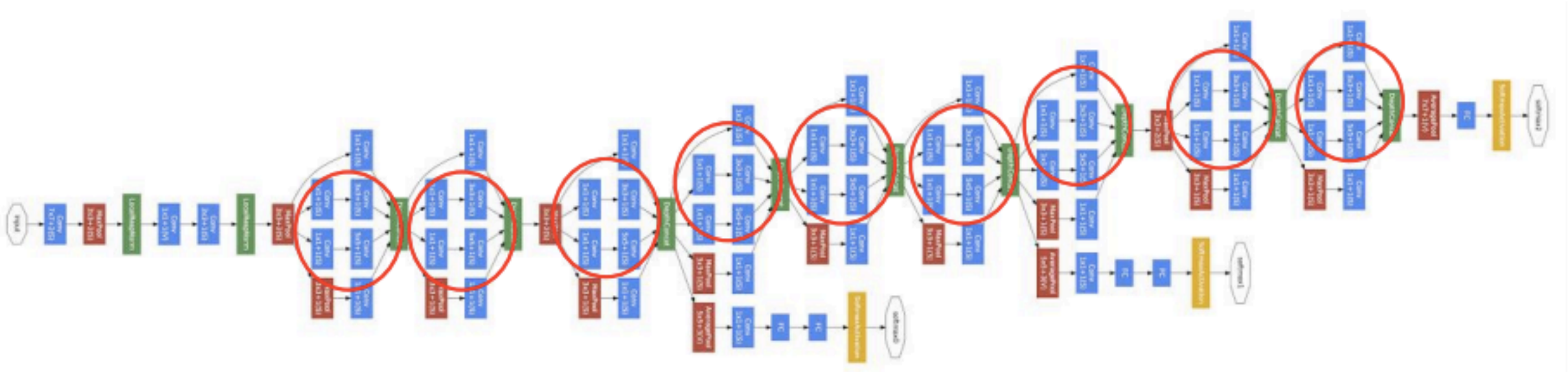


Inception module with dimension reduction

$$\text{Compression Ratio} = \frac{\text{depth of input}}{\text{number of 1x1 convs}}$$

Popular CNN Architectures

- Inception Module: Network in Network



9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other

Popular CNN Architectures

- Inception Module: Intermediate Multi-objectives

- Key for training deep nets

- Encourage to learn useful intermediate representations

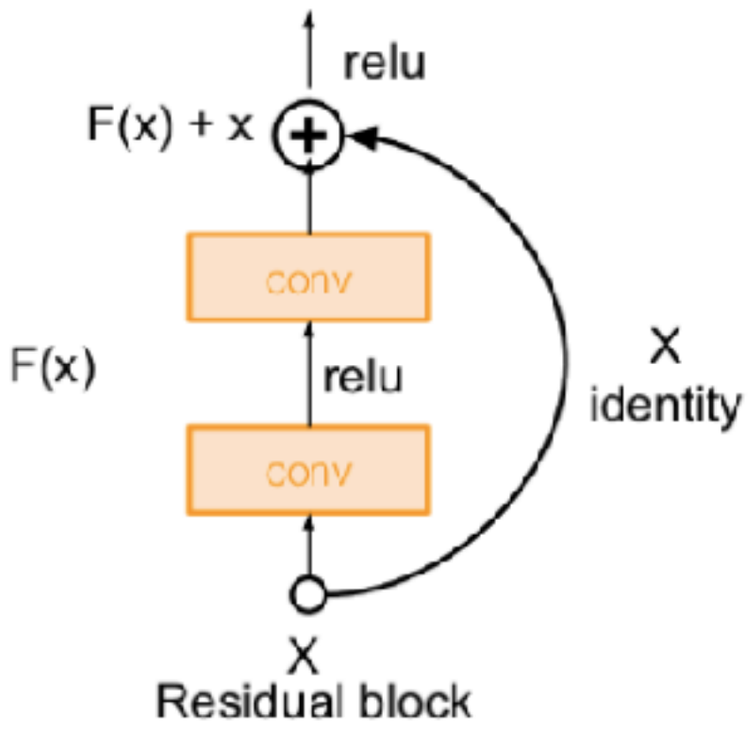
Popular CNN Architectures

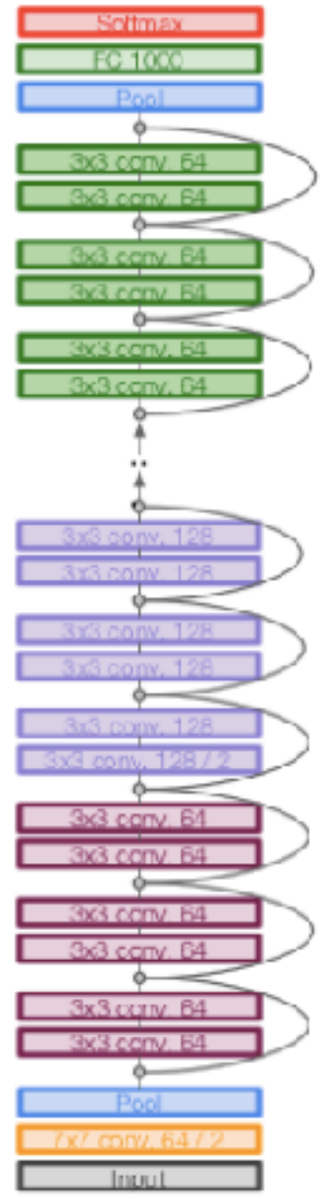
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

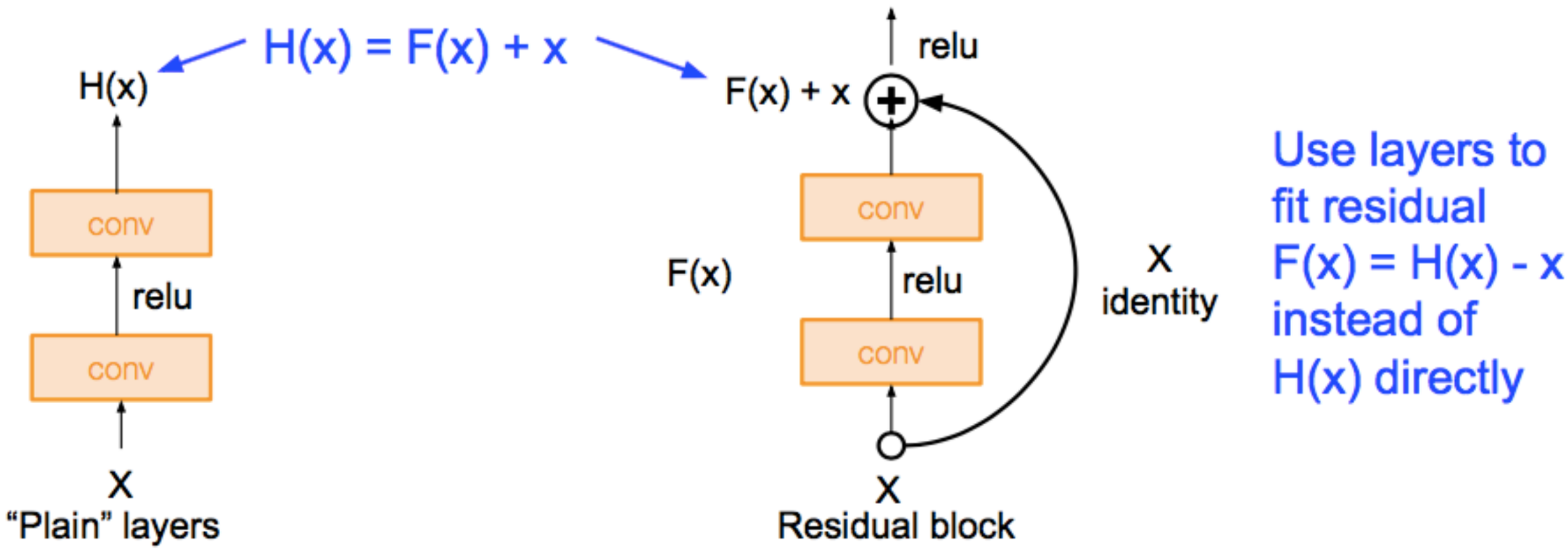




The model that outperforms human recognition rate

Popular CNN Architectures

- Why we need residual connection: **optimization is hard for deep nets**



The effective depth is halved by residual connection

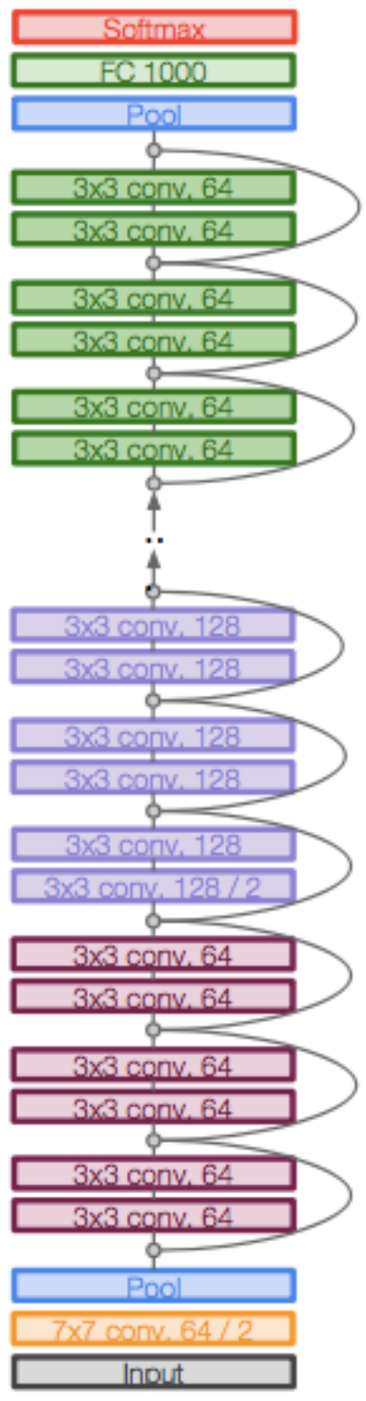
Popular CNN Architectures

- Tricks learned from ResNet:

- Residual connection to help optimization

- Avoid pooling layers when computational resources are available

- Conceptually easier than GoogLeNet: optimization > model architecture



Popular CNN Architectures

- Summary:

- Models we covered today: AlexNet, VGG, GoogLeNet, ResNet, …

- Original models are trained on multi-GPUs

- Existing models are available in TensorFlow

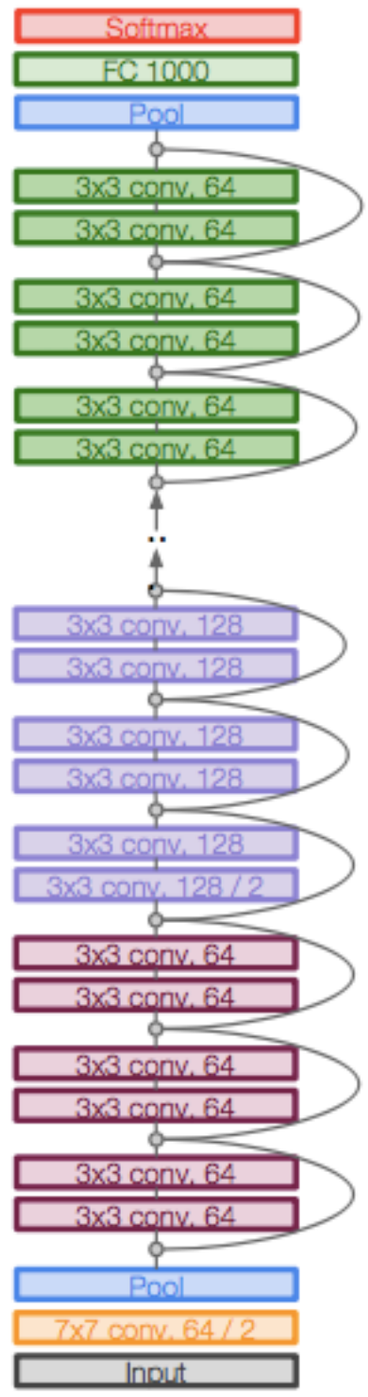


Image Saliency Detection

- Saliency: what we quickly focus on when seeing an image

- Useful in UX design to understand customers' focus
- Useful in advertisement to better design ads
- Useful in image resizing to automatically adjust its size

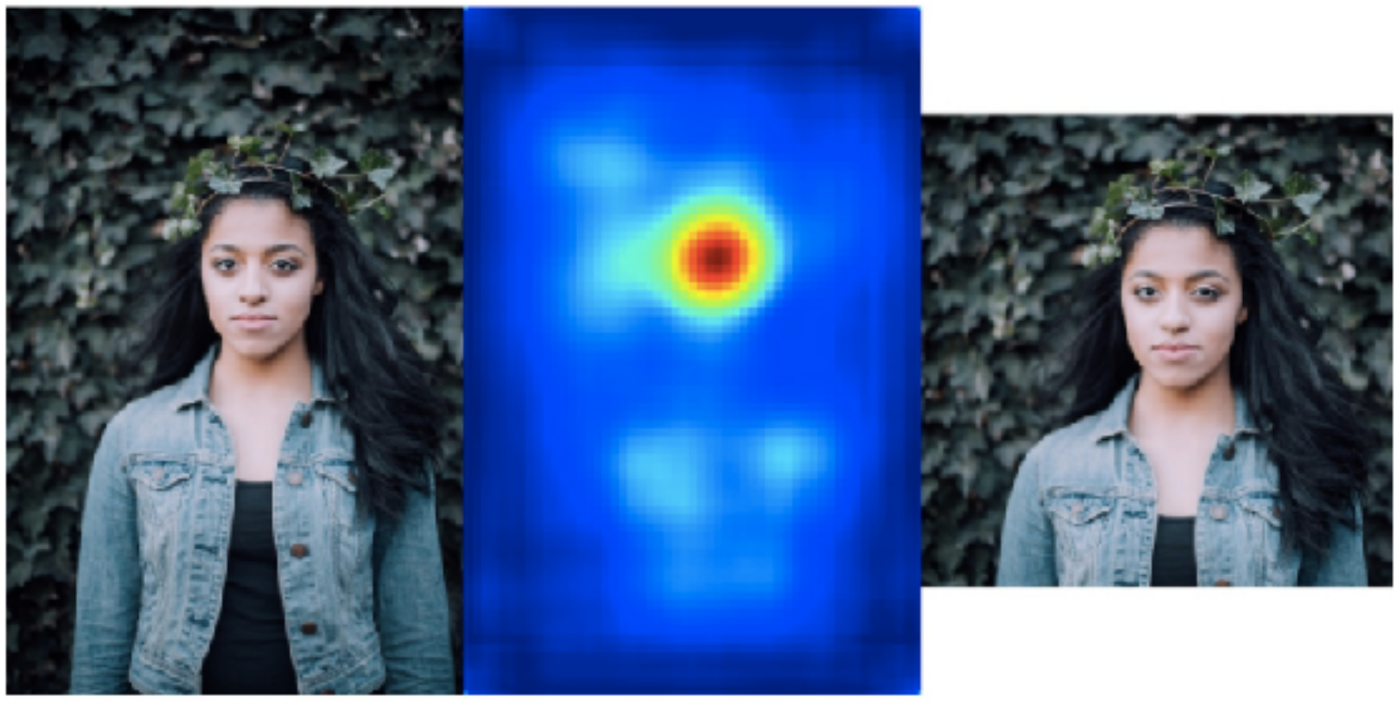
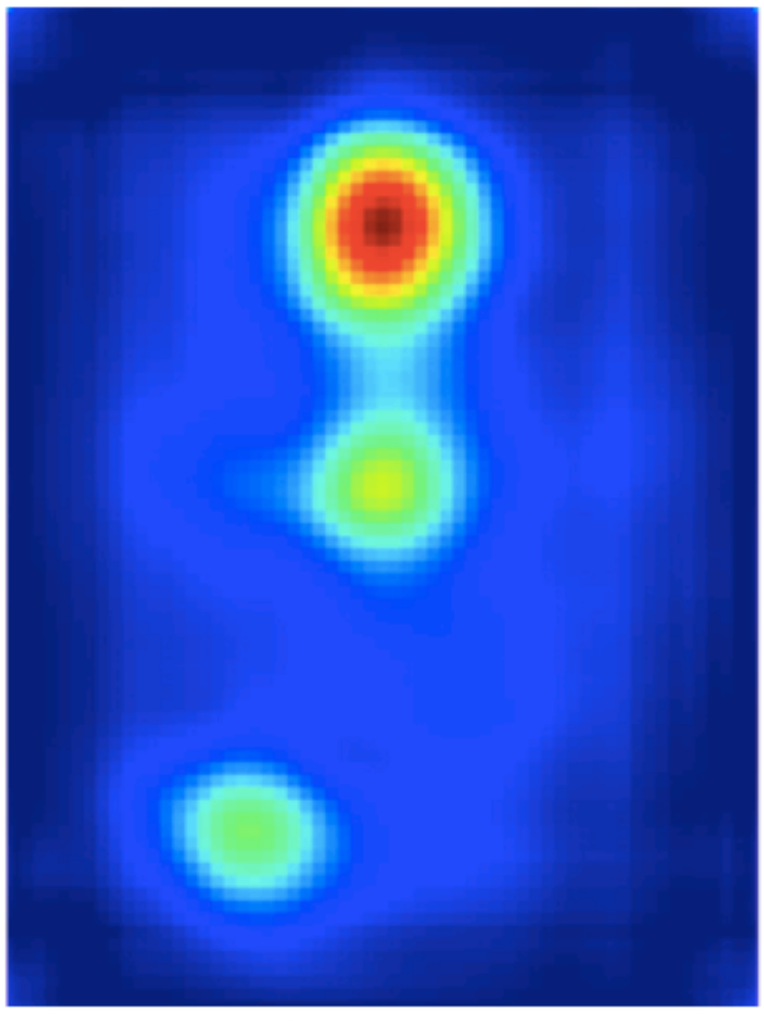


Image Saliency Detection

- Saliency detection: automatically detects the most relevant part of an image

Input



Output

Image Saliency Detection

- CNN for saliency detection: automatically detects the most relevant part of an image

- Dataset: SALICON (http://salicon.net/), a part of the MS-COCO (Microsoft Common Objects in Context) dataset: ~330K images (more details at: https://arxiv.org/pdf/1405.0312.pdf)

- We will introduce CNN-based models for saliency detection: Shallow and Deep Convolutional Networks for Saliency Prediction (CVPR 2016, https://arxiv.org/pdf/1603.00845v1.pdf)

- Github repo: https://github.com/imatge-upc/saliency-2016-cvpr

- Recent advances in saliency detection: http://saliency.mit.edu/ECCVTutorial/ECCV_saliency.htm

Image Saliency Detection

- Main difference from previous CNNs: convnet for regression rather than classification

$$\ell(\theta) = \frac{1}{n}\sum_{i=1}^{n} ||Y_i - f(X_i;\theta)||_F^2 \qquad Y_i, f(X_i;\theta) \in \mathbb{R}^{p_1 \times p_2}$$

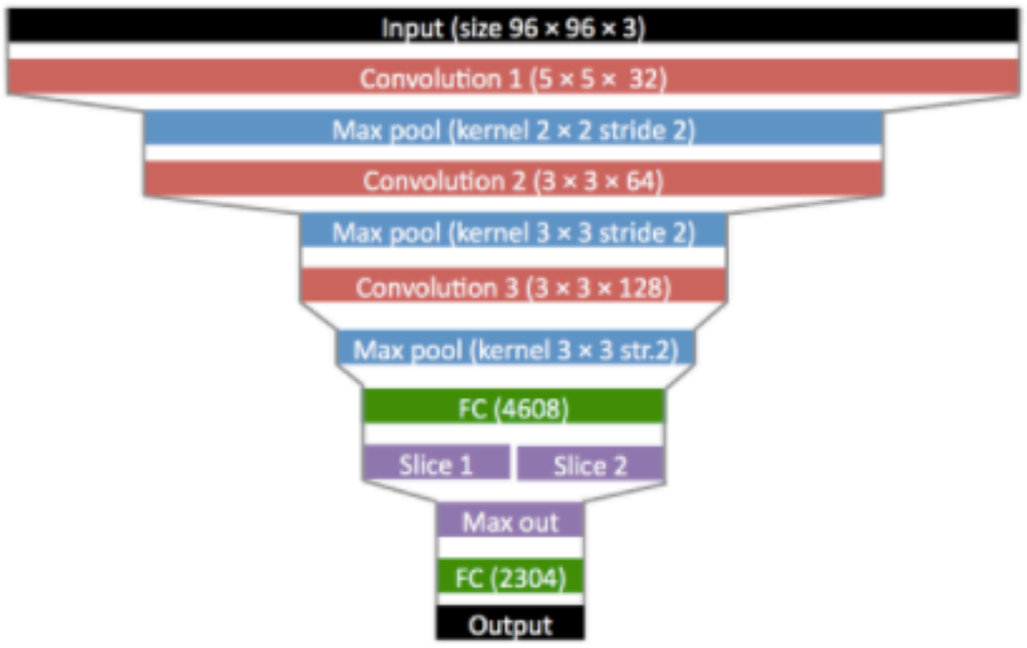
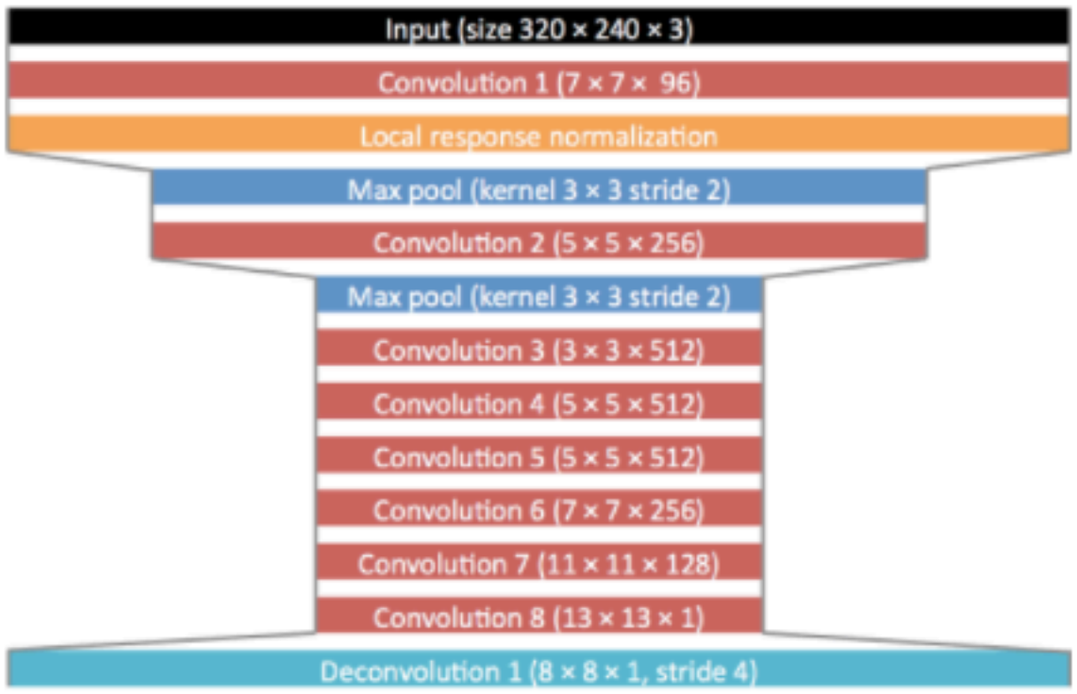
Shallow ConvNet (aka JuntingNet)	Deep ConvNet (aka SalNet)
	
[Lasagne Model (2.5 GB)]	[Caffe Model (99 MB)] [Caffe Prototxt]
Training from scratch	First 3 layers from VGG

Image Saliency Detection

- Model comparisons:

	Shallow	Deep
Data	2.29 MB	123.65 MB
Parameters	244.64 MB	98.44 MB
Total (train)	249.22 MB	345.74 MB
Total (test)	246.93 MB	222.09 MB

Table 1. Approximate memory requirements for each convnet.

Image Saliency Detection

- Model comparisons:

	Similarity	CC	AUC shuffled	AUC Borji	AUC Judd
Shallow Convnet	**0.5198**	**0.5957**	**0.6698**	**0.8291**	**0.8364**
WHU IIP	0.4908	0.4569	0.6064	0.7759	0.7923
Rare 2012 Improved [30]	0.5017	0.5108	0.6644	0.8047	0.8148
Xidian	0.4617	0.4811	0.6809	0.7990	0.8051
Baseline: BMS [45]	0.4542	0.4268	0.6935	0.7699	0.7899
Baseline: GBVS [14]	0.4460	0.4212	0.6303	0.7816	0.7899
Baseline: Itti [17]	0.3777	0.2046	0.6101	0.6603	0.6669

Table 5. Results for the SALICON test set, according to the LSUN Challenge 2015.

	Similarity	CC	AUC shuffled	AUC Borji	AUC Judd
Baseline: Infinite Humans	1.00	1.00	0.80	0.87	0.91
SALICON [16] (*)	0.60	0.74	0.74	0.85	0.87
DeepFix [37] (**)	0.67	0.78	0.71	0.80	0.87
Deep Gaze 1 [24]	0.39	0.48	0.66	0.83	0.84
Deep Convnet	**0.52**	**0.58**	**0.69**	**0.82**	**0.83**
BMS [45]	0.51	0.55	0.65	0.82	0.83
eDN [40]	0.41	0.45	0.62	0.81	0.82
GBVS [14]	0.48	0.48	0.63	0.80	0.81
Judd [21]	0.42	0.47	0.60	0.80	0.81
Shallow Convnet	**0.46**	**0.53**	**0.64**	**0.78**	**0.80**
Mr-CNN [28]	0.48	0.48	0.69	0.75	0.79
Rare 2012 Improved [30]	0.46	0.42	0.67	0.75	0.77
Baseline: One human	0.38 – 0.46	0.52 – 0.65	0.63 – 0.67	0.66 – 0.71	0.80 – 0.83

Table 6. Results of the MIT300 dataset. (*-to be published, (**-non-peer reviewed

Image Saliency Detection

- Examples:

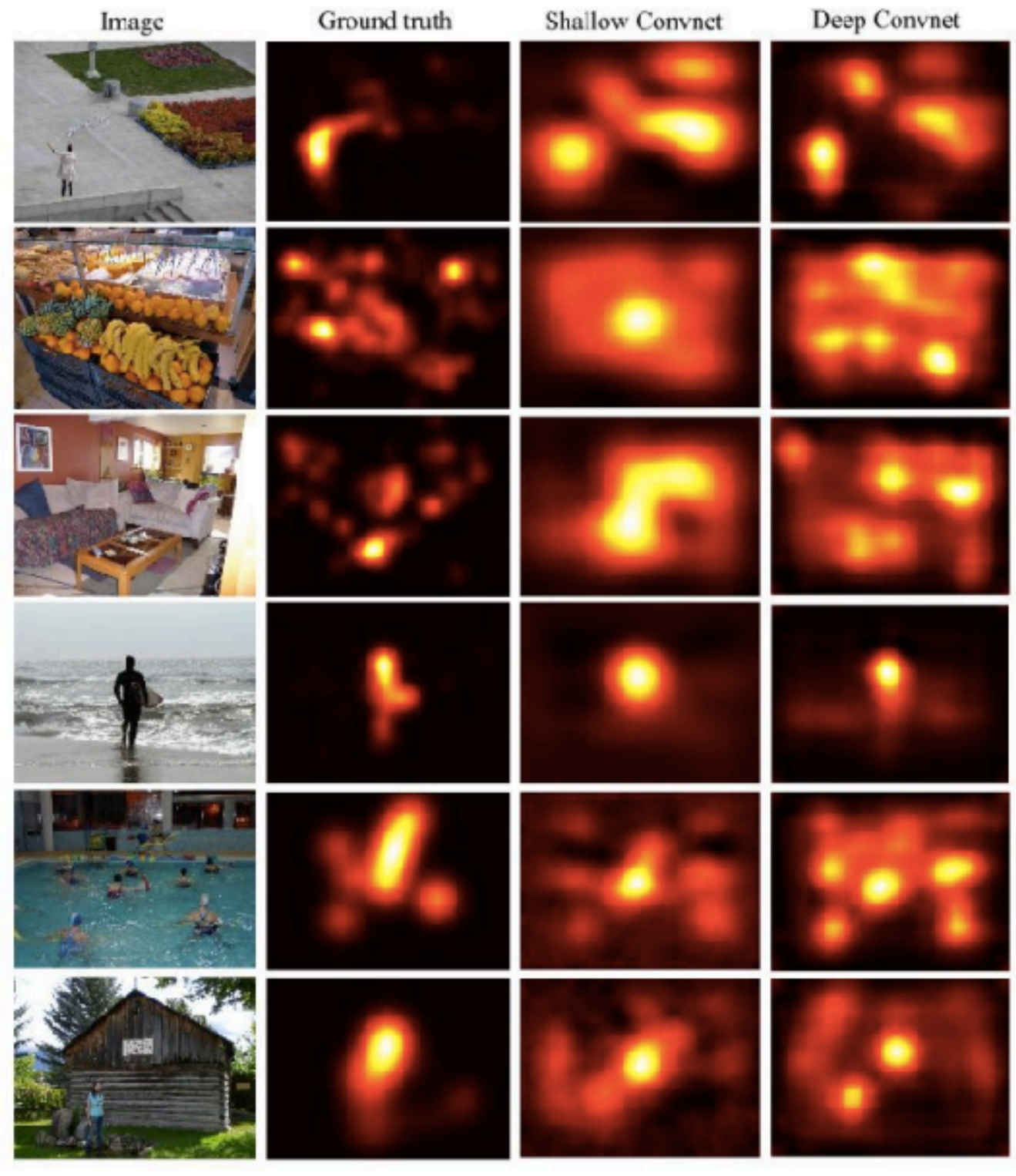


Image Segmentation

- What is image segmentation: partition an image into multiple parts

- Uses in locating objects
- Uses in boundary detection
- Goal: assign a label to every pixel in the image

Input

Output

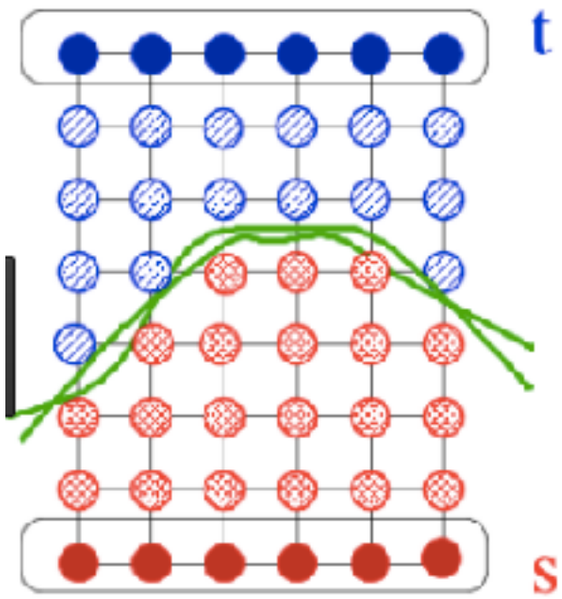
Image Segmentation

- Classic approaches for image segmentation:

- Binary thresholding
- Clustering (K-means++, flat clustering)
- Bottom-up region growing (hierarchical clustering)
- Differential equation based approach (level set method)
- **Graph partitioning (graph cut, grab cut)**
- **Probabilistic graphical model (Markov random field)**
- **Deep learning approach (convnet)**

Image Segmentation

- Graph cut: formulate as max-flow-min-cut problem/ normalized cut
- Original paper: Normalized cut and image segmentation (http://www.cis.upenn.edu/~jshi/papers/pami_ncut.pdf)
- G = (V, E), each image pixel = vertex, weights = difference between pixels gray values
- Image segmentation = finding minimum cut in a graph G



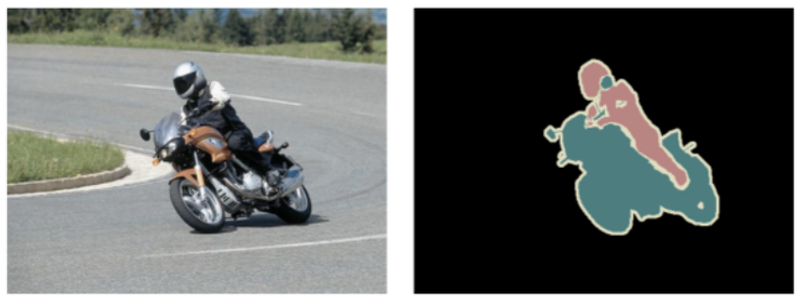


Image Segmentation

- Graph cut: max-flow-min-cut problem/normalized cut

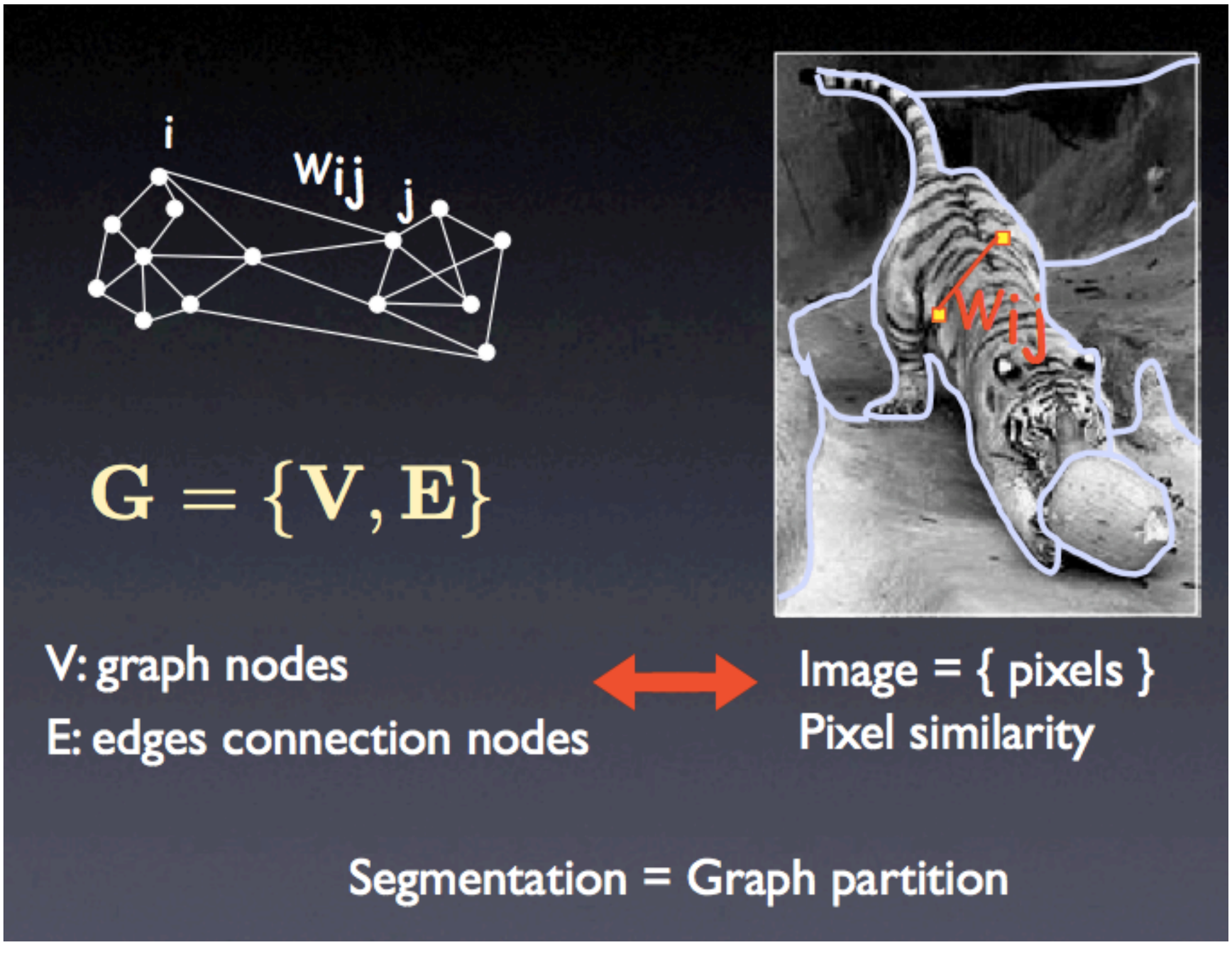


Image Segmentation

- Graph cut: max-flow-min-cut problem/normalized cut

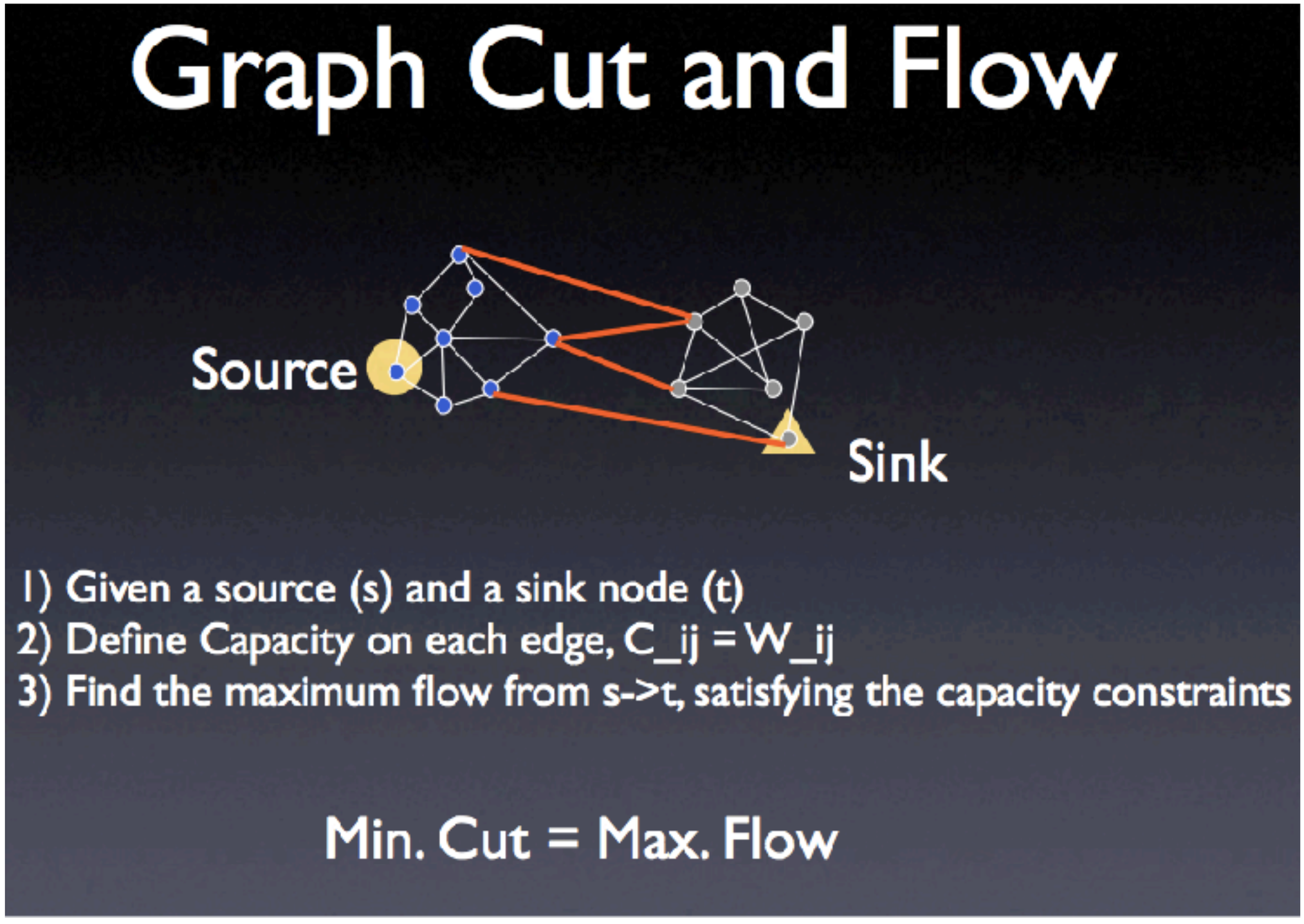


Image Segmentation

- Problem with min-cut:

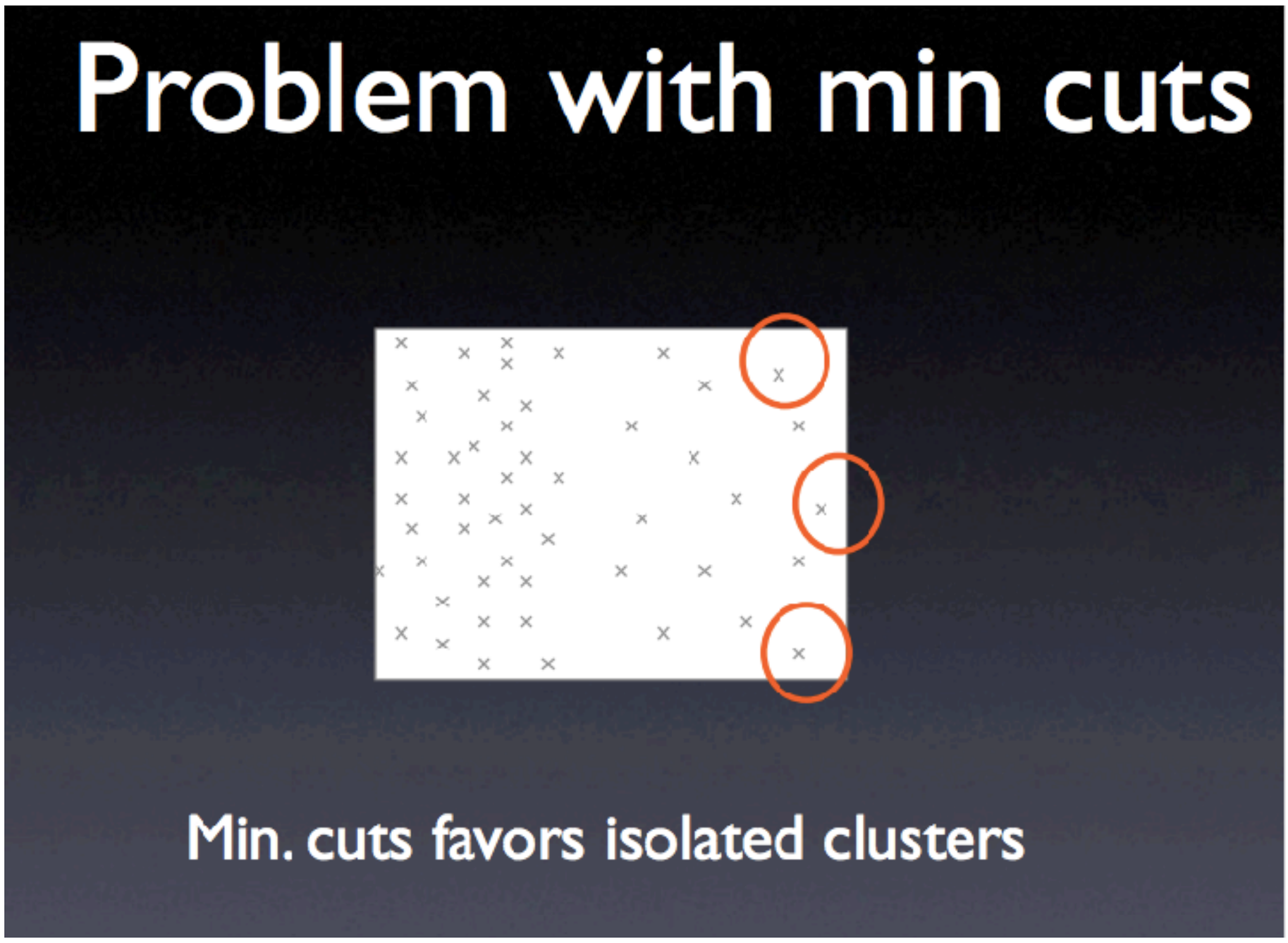
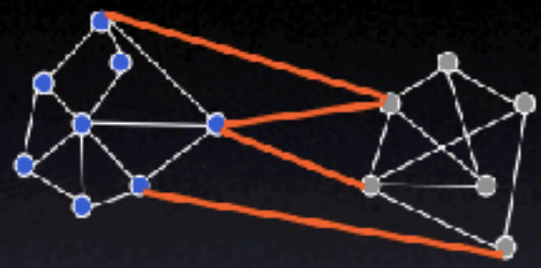


Image Segmentation

- Improved cost function: Normalized-Cut (Ncut)

Normalize cuts in a graph

- (edge) Ncut = balanced cut



$$Ncut(A,B) = cut(A,B)(\frac{1}{vol(A)} + \frac{1}{vol(B)})$$

NP-Hard!

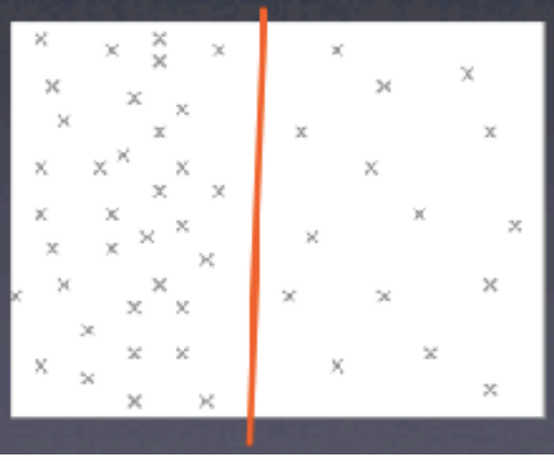


Image Segmentation

- Problem with Ncut: finding the Ncut is NP-hard!
- Approximation using spectral method: spectral clustering for image segmentation (further reading: http://www.cis.upenn.edu/~jshi/papers/pami_ncut.pdf)
- Further improvement: Grab-Cut
  - Based on graph-cut
  - Include user interaction (user specifies a bounding box)
  - Incorporate Markov random fields to define an energy function
  - Further reading: http://pages.cs.wisc.edu/~dyer/cs534-fall11/papers/grabcut-rother.pdf, http://docs.opencv.org/3.2.0/d8/d83/tutorial_py_grabcut.html

Fully Convolutional Network

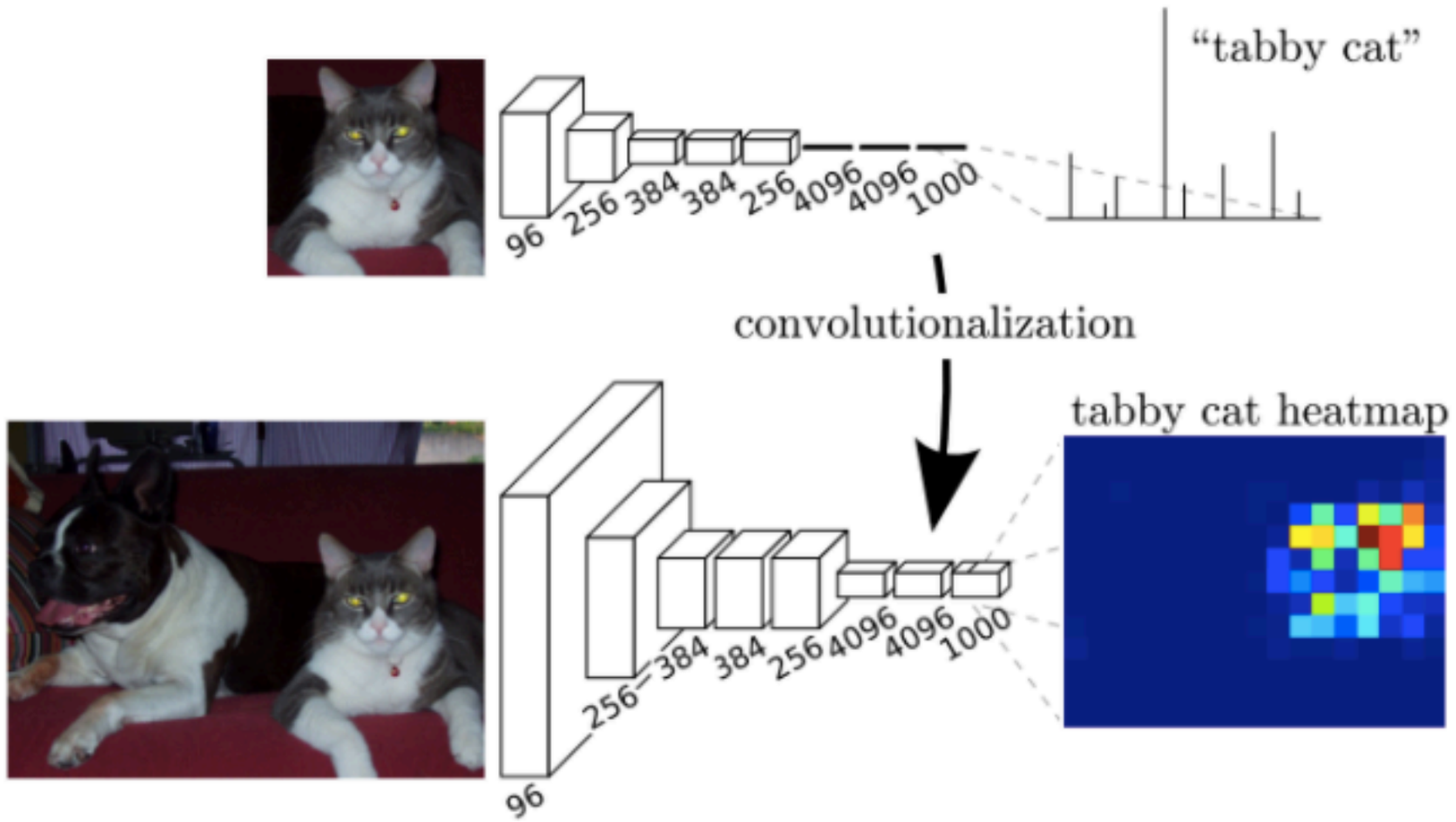
- Fully convolutional Networks for Semantic Segmentation: https://arxiv.org/abs/1411.4038

- The first work applying end-to-end CNN to pixel-wise prediction (semantic segmentation)
  - Use pre-trained network on ImageNet (VGG)
  - No need on pre-processing of image patches
  - FCN: build a classifier for each pixel in the image

Key idea: replace fully-connected layer with convolution!

Fully Convolutional Network

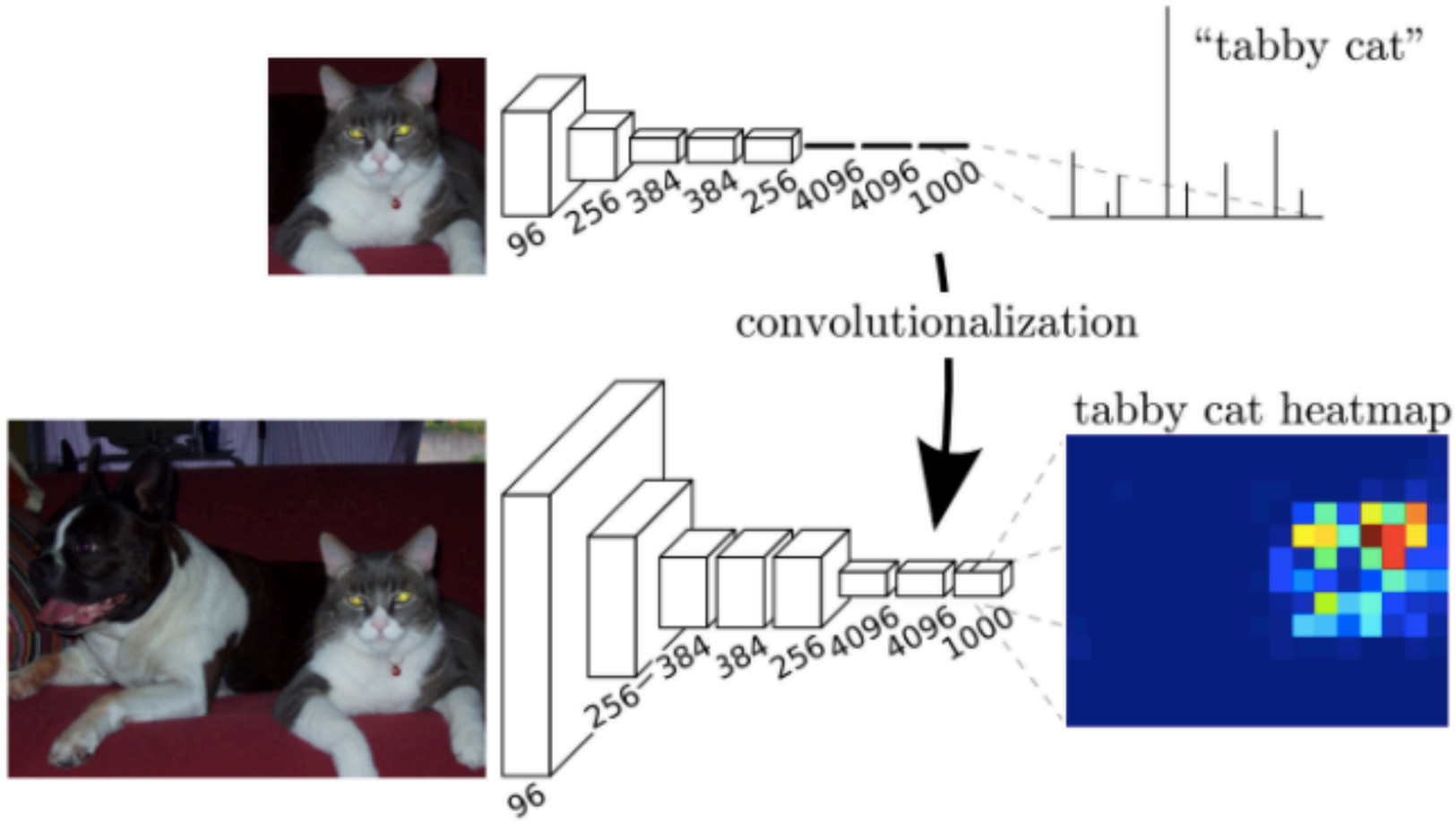
- Fully convolutional Networks for Semantic Segmentation



Every layer in the network is convolution, no fully-connected, hence the name: fully convolutional network

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

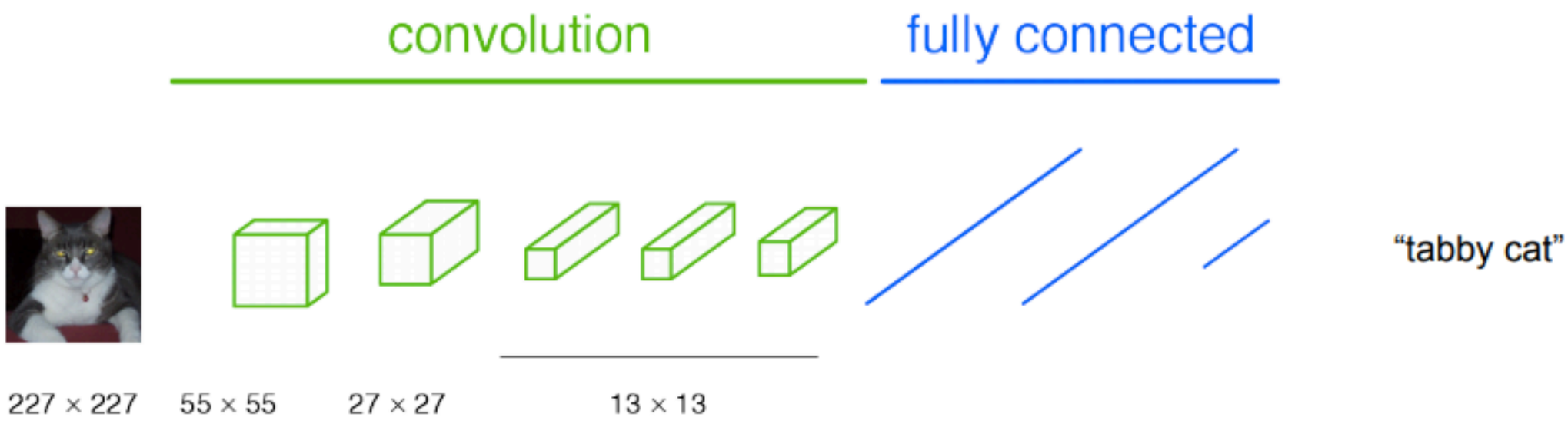


$$4096 \rightarrow 4096 \times 1 \times 1$$

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

a classification network



Usual convnet: image with specific size

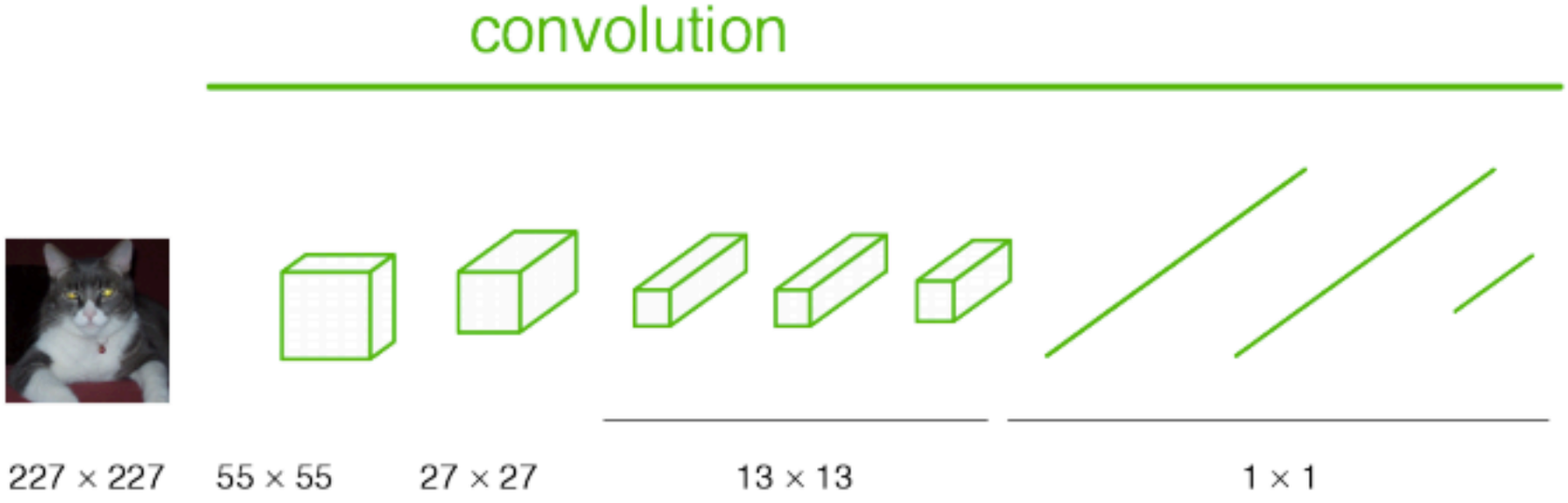
FCN: image with arbitrary size

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

becoming fully convolutional

$$4096 \rightarrow 4096 \times 1 \times 1$$



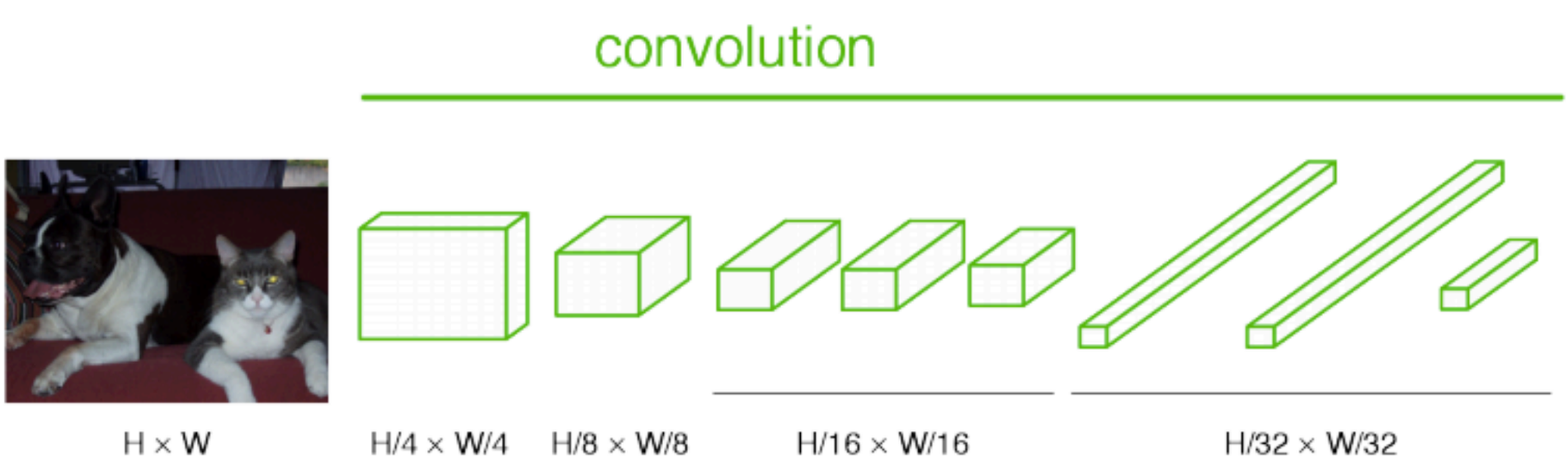
Usual convnet: image with specific size

FCN: image with arbitrary size

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

becoming fully convolutional



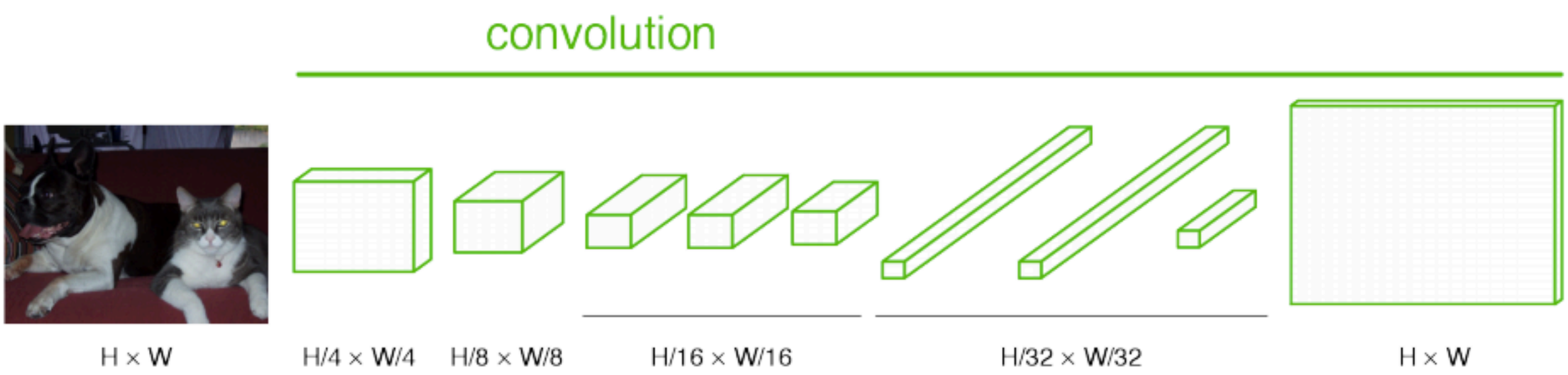
Usual convnet: image with specific size

FCN: image with arbitrary size

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

upsampling output



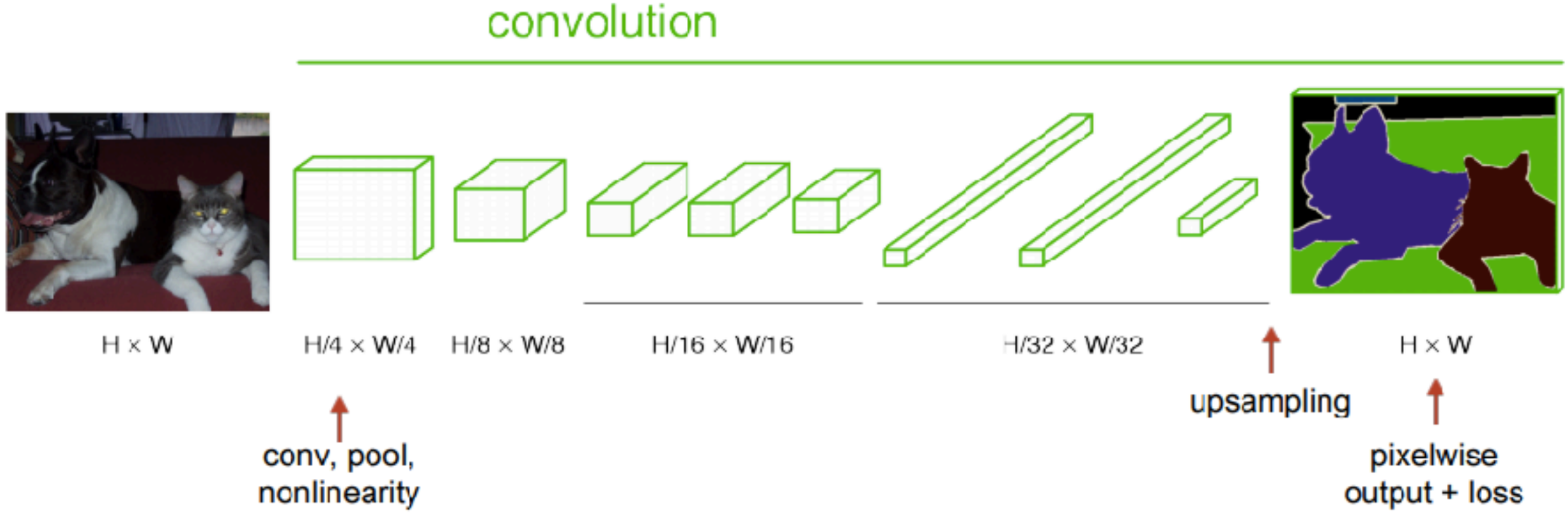
Usual convnet: image with specific size

FCN: image with arbitrary size

Fully Convolutional Network

- Fully convolutional Networks for Semantic Segmentation
- Observation: each layer of data in a convnet: $h \times w \times d$

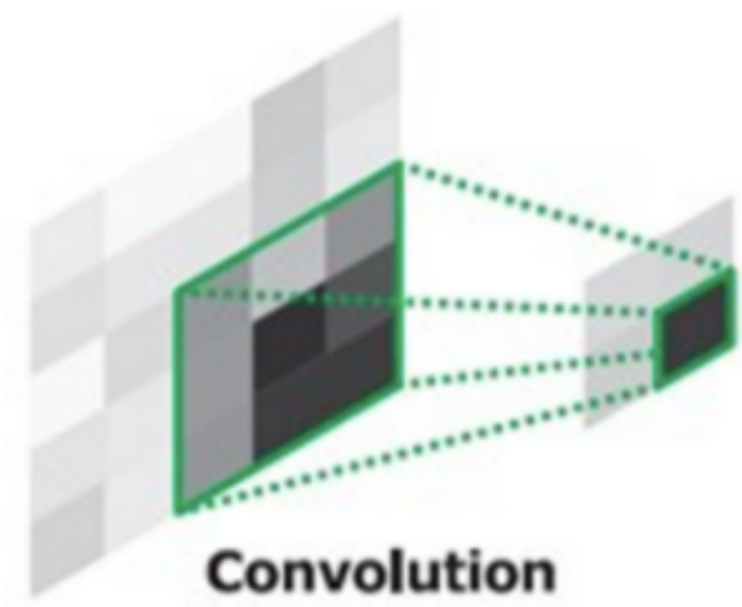
end-to-end, pixels-to-pixels network

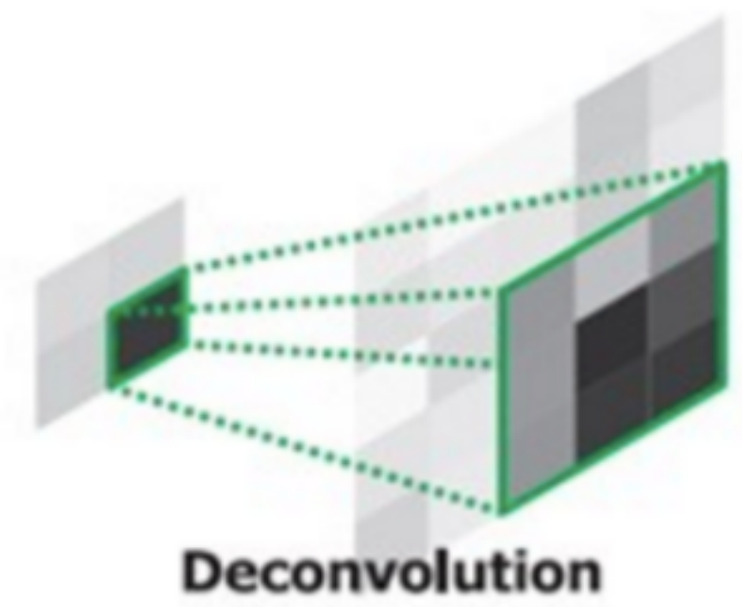


FCN: image with arbitrary size, loss = classification

Fully Convolutional Network

- Deconvolution = Transposed convolution = Upsampling

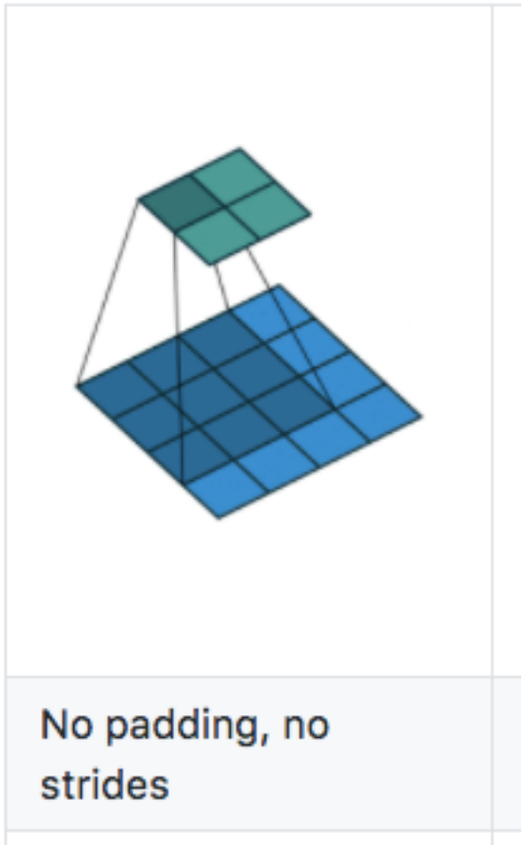




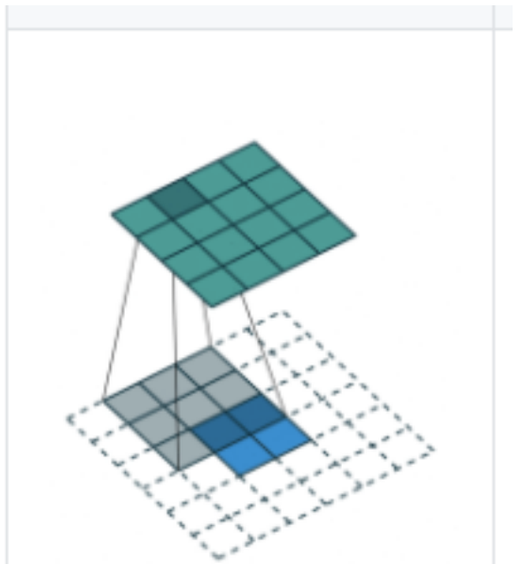
Key: To scale up the feature maps from previous convs

Fully Convolutional Network

- Deconvolution = Transposed convolution = Upsampling
- Actually the same as conv, but with fully zero-padded border



No padding, no strides



No padding, no strides, transposed

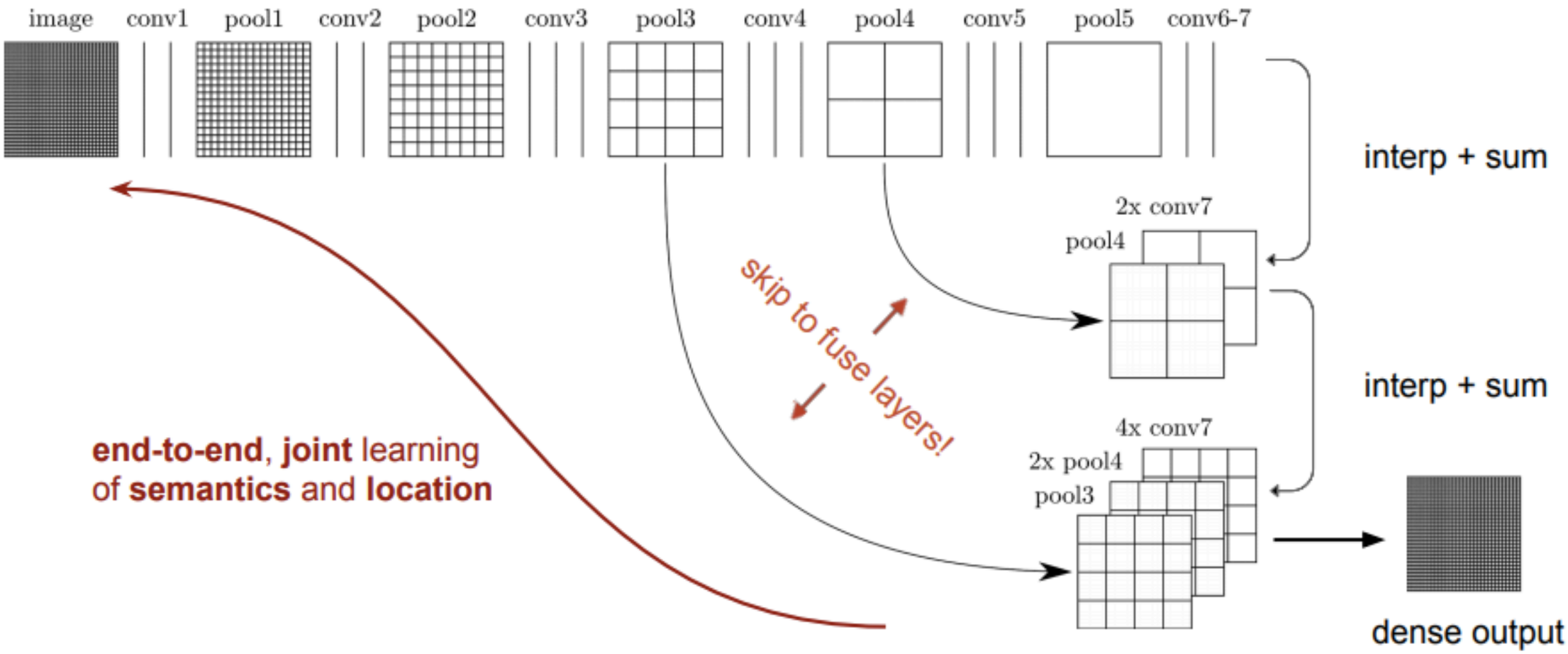
Animation for cons arithmetic: https://github.com/vdumoulin/conv_arithmetic

Detailed intro: https://arxiv.org/pdf/1603.07285.pdf

Key: To scale up the feature maps from previous convs

Fully Convolutional Network

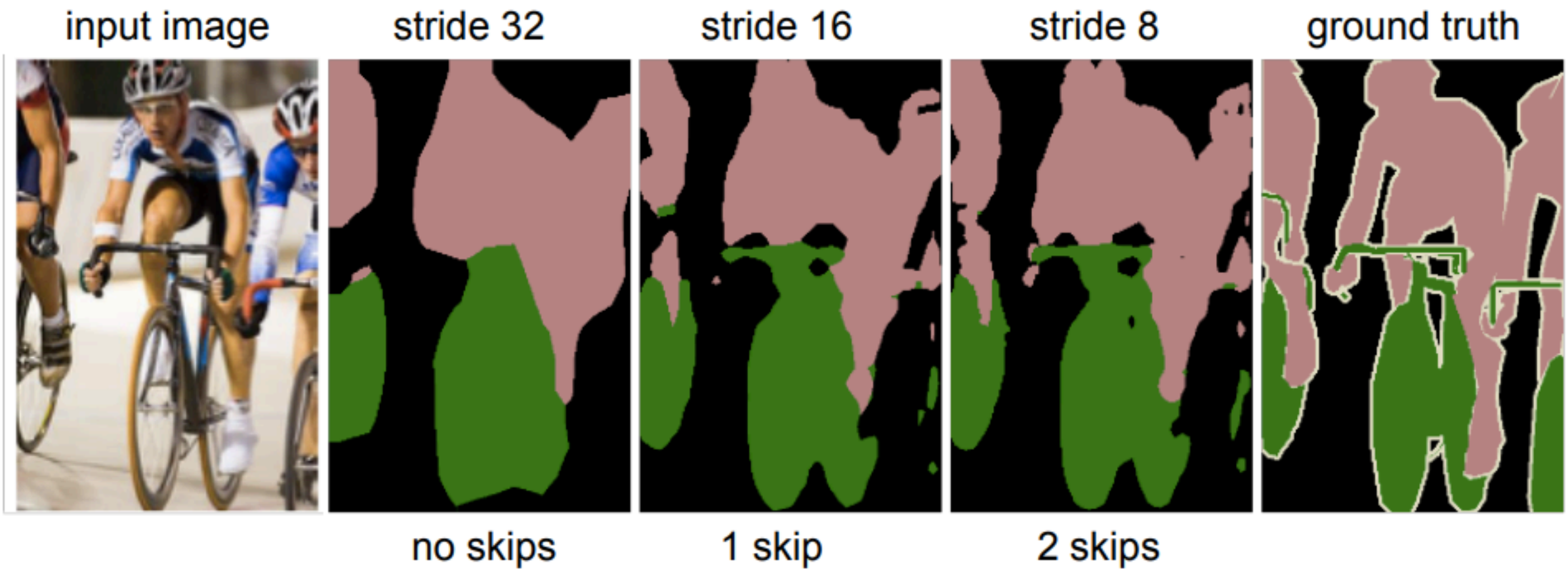
- Skip connection:



Convolutions make feature more coarse, loss of information

Fully Convolutional Network

- Skip connection:



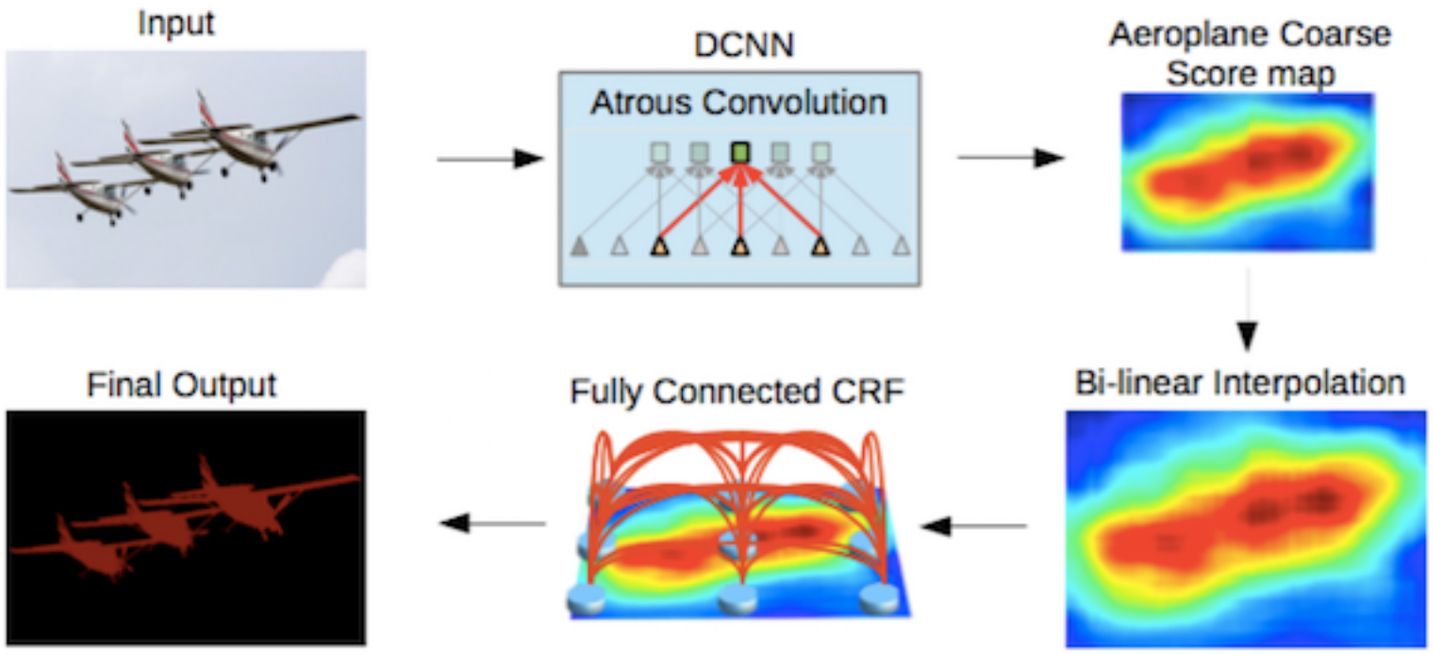
Convolutions make feature more coarse, loss of information

Github repo for FCN: https://github.com/shekkizh/FCN.tensorflow

DeepLab

- DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs (https://arxiv.org/abs/1606.00915)

- Further improvement over FCN
- Atrous convolution + conditional random field



Summary

- Abstract and high-level understanding of CNNs
- Derivation of back-propagation for CNNs
- Introduction about popular CNN models and their key ideas
- Image segmentation: from graph based algorithms to deep learning models
- Fully convolutional networks: transposed convolution + skip connection + fully convolution architecture
- DeepLab: combination of FCN and CRF (Deep learning + Probabilistic graphical models)