Lecture 8: Convolutional Neural Networks

Zerrin Yumak

Utrecht University

Announcements

- Project proposals will be presented next week!
  - 19 Dec Wed: Teams 1-8
  - 21 Dec Friday: Teams 9-16
  - Each team will have around 8-10 mins maximum.

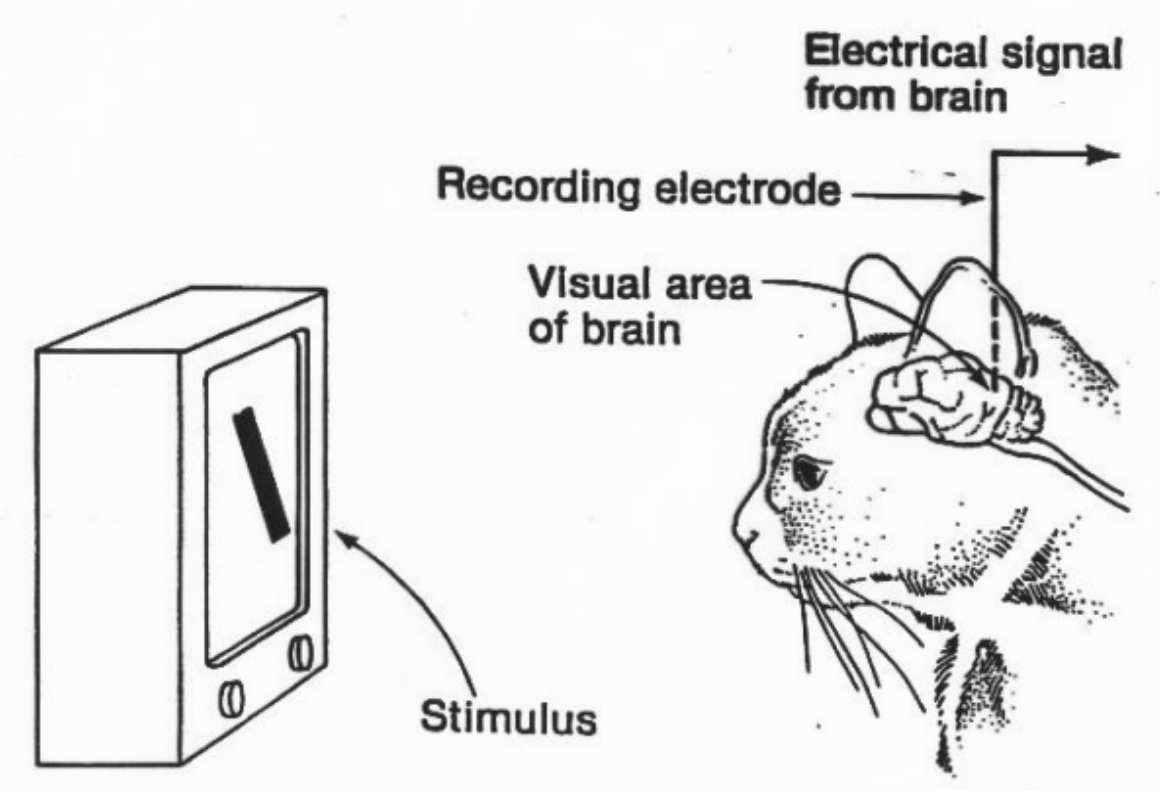
- Project proposal deadline: **21 Dec Friday 23:59pm**

- If you have any questions, one or two representative from each team can go to BBG-175 and meet the TA during lab hours, starting from Dec 21.

In this lecture

- A bit of history
- CNN basics
  - Convolutions, strides, pooling
- CNN architectures
  - AlexNext
  - ZFNet
  - VGGNet
  - GoogLeNet
  - Residual Networks
- Applications of CNN
  - Object detection
  - Object segmentation
  - …

A bit of history



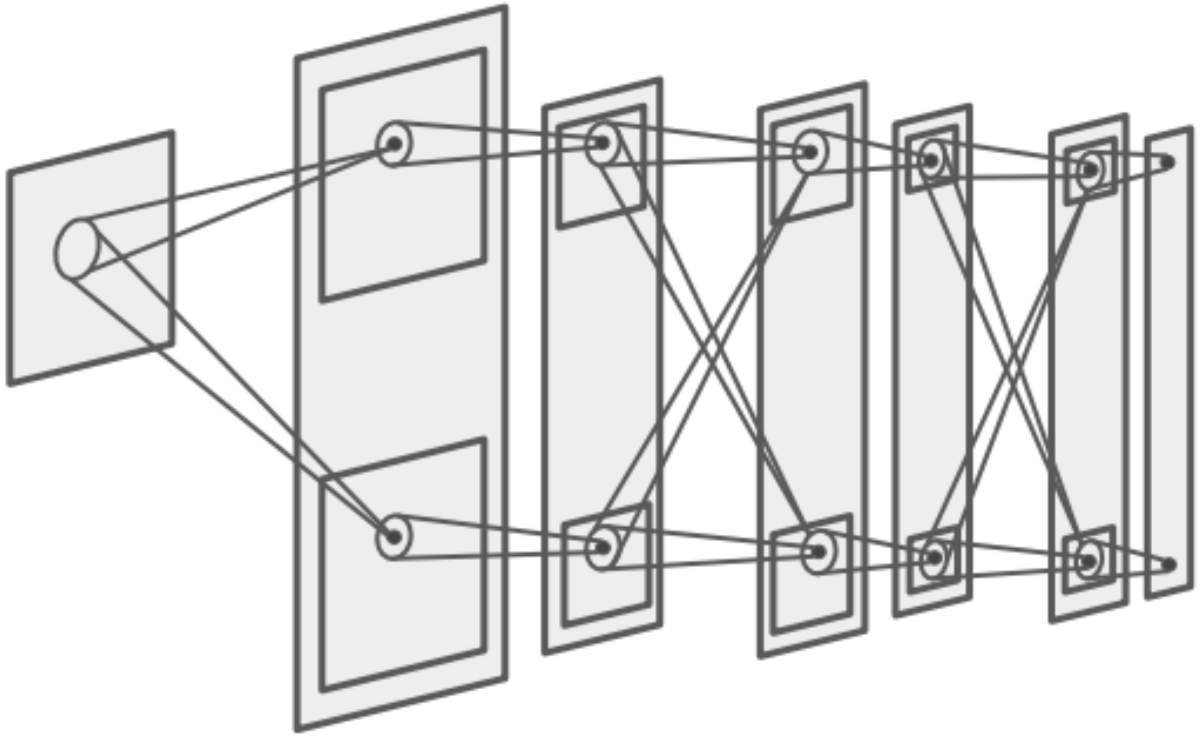
1) Spatial Invariance

2) Receptive Field

3) Hierarchy

A bit of history

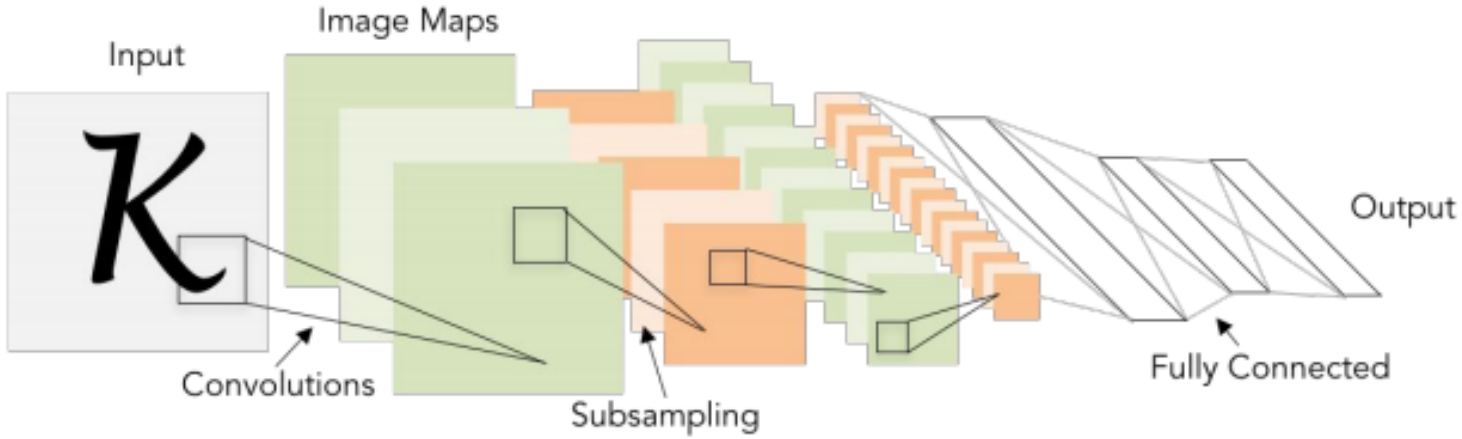
Neocognitron
[Fukushima 1980]



"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history

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

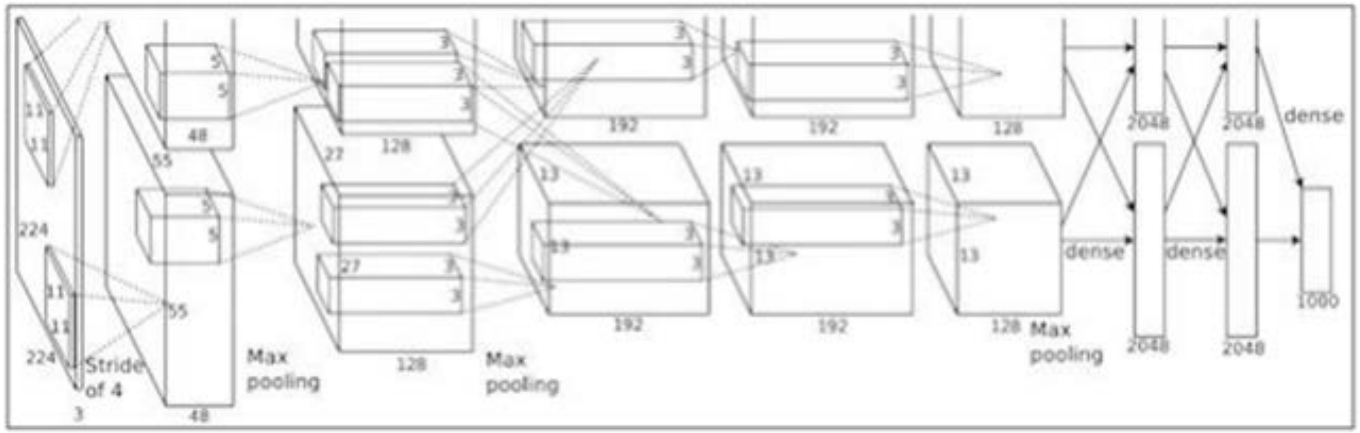


LeNet-5

A bit of history

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]



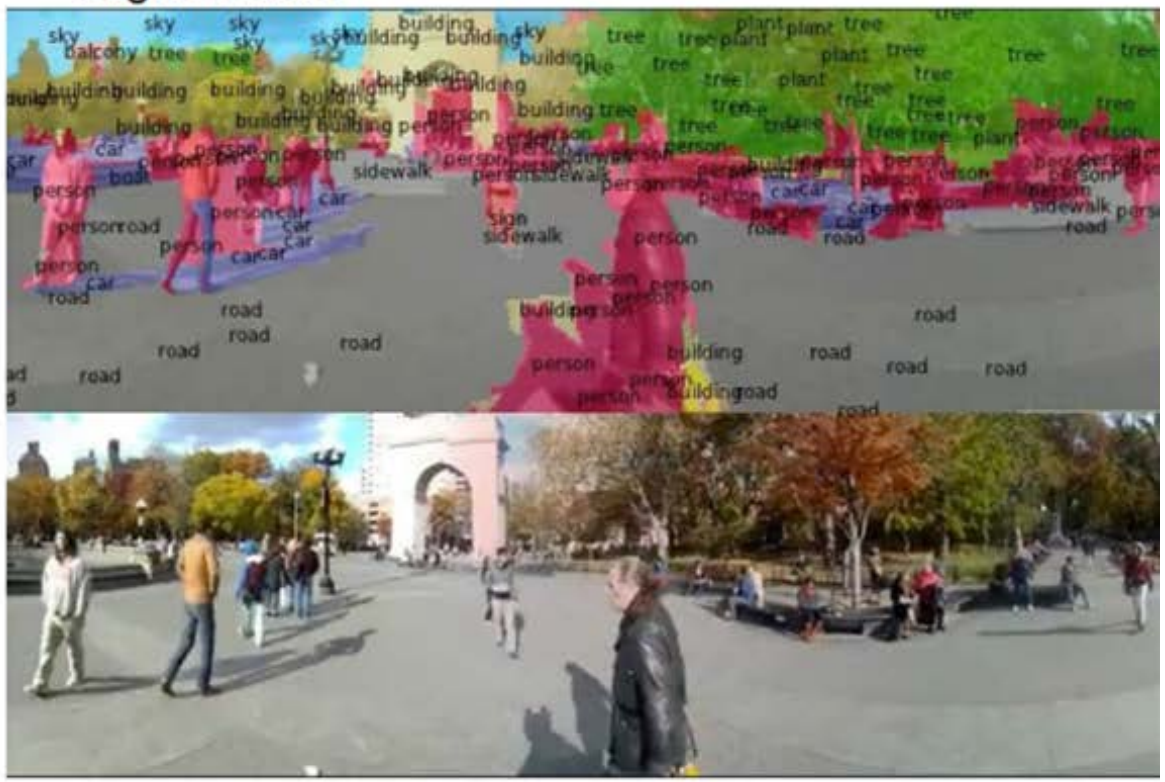
"AlexNet"

CNNs are widely used now..

Detection

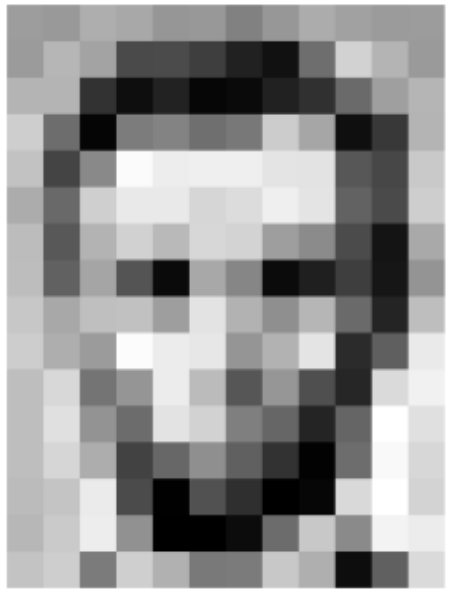
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

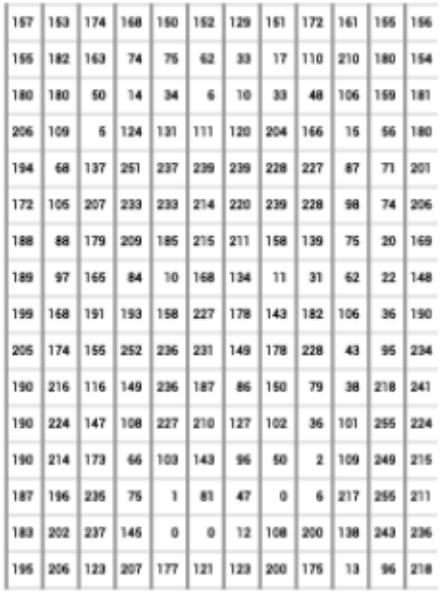


[Farabet et al., 2012]

Images are numbers

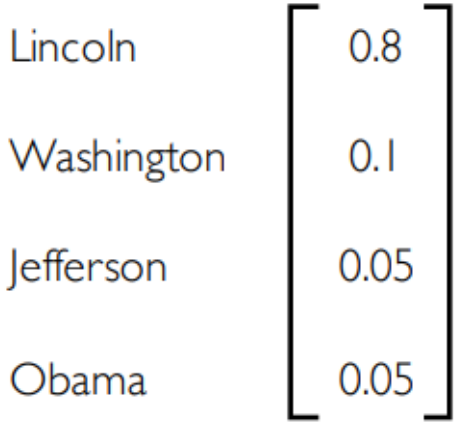


Input Image



Pixel Representation

classification

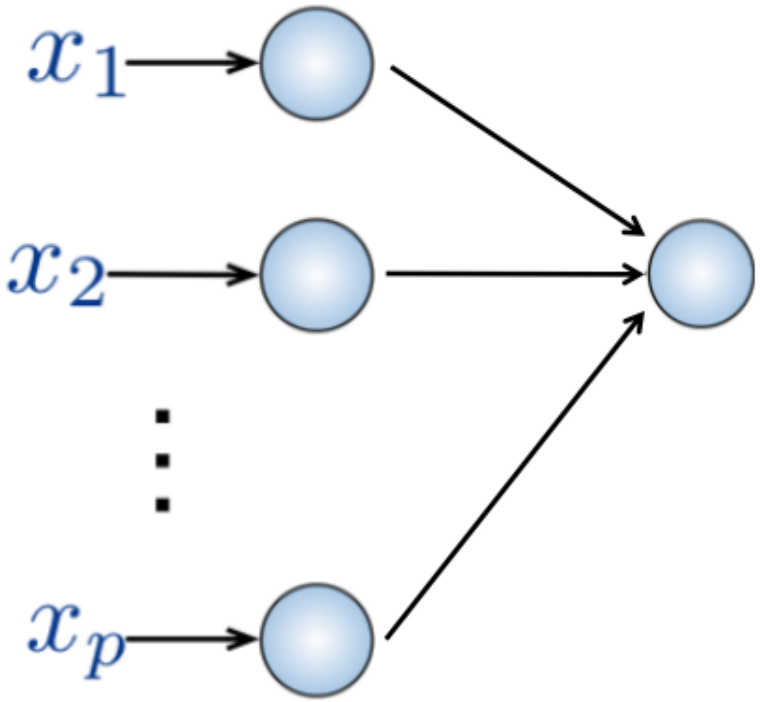


- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



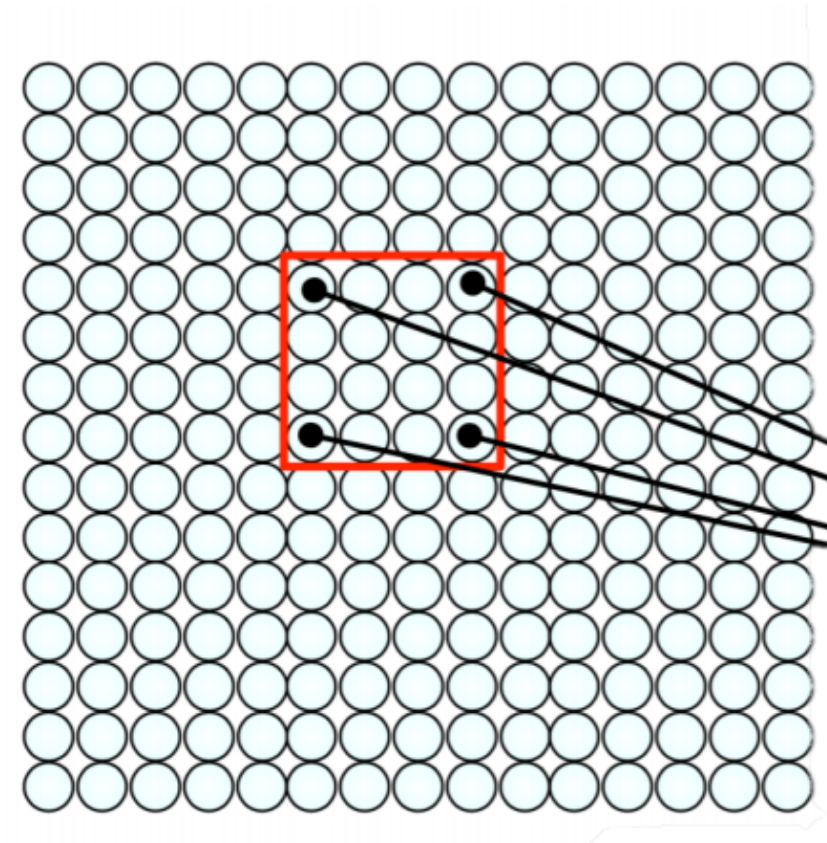
Fully connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Using Spatial Structure

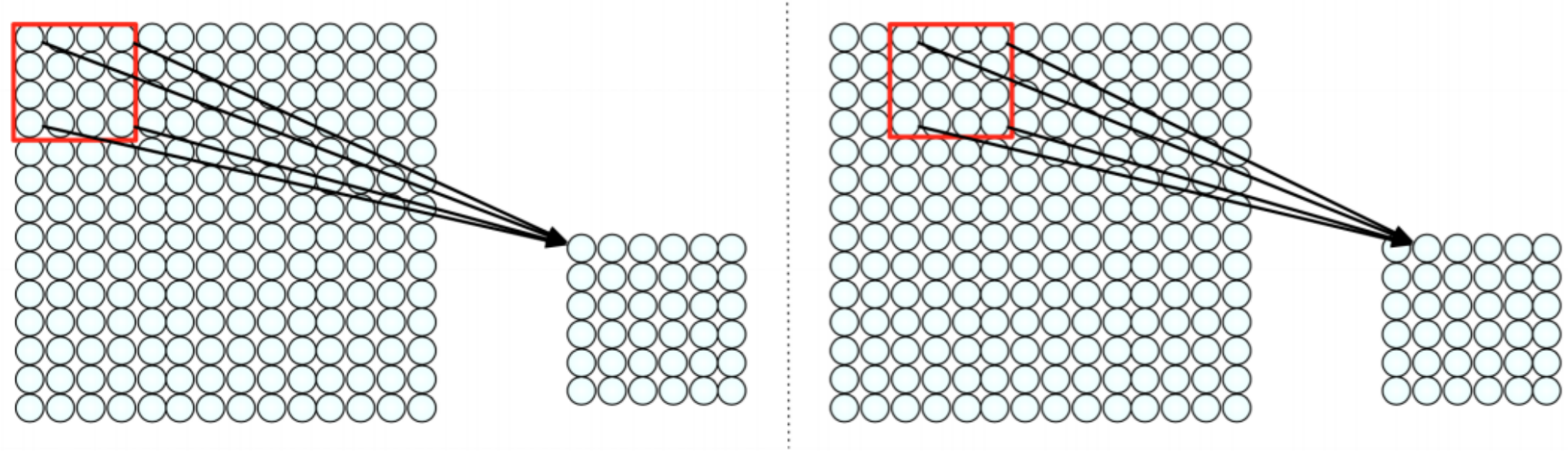
Input: 2D image.
Array of pixel values



Idea: connect patches of input to neurons in hidden layer.

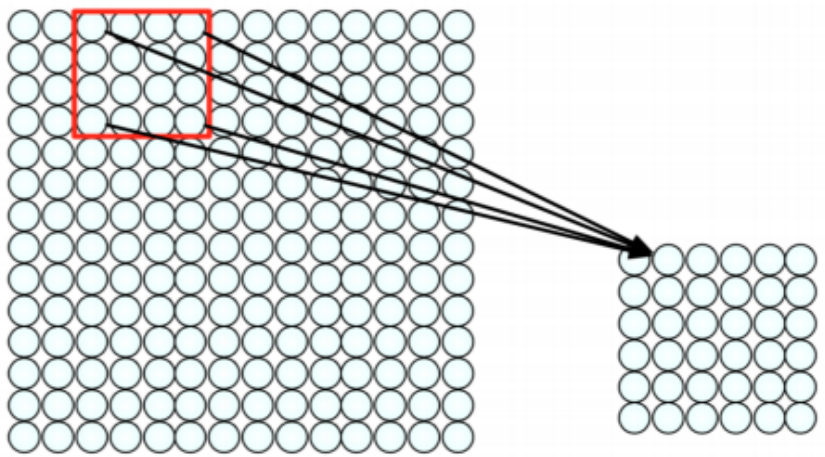
Neuron connected to region of input. Only "sees" these values.

Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Feature Extraction and Convolution

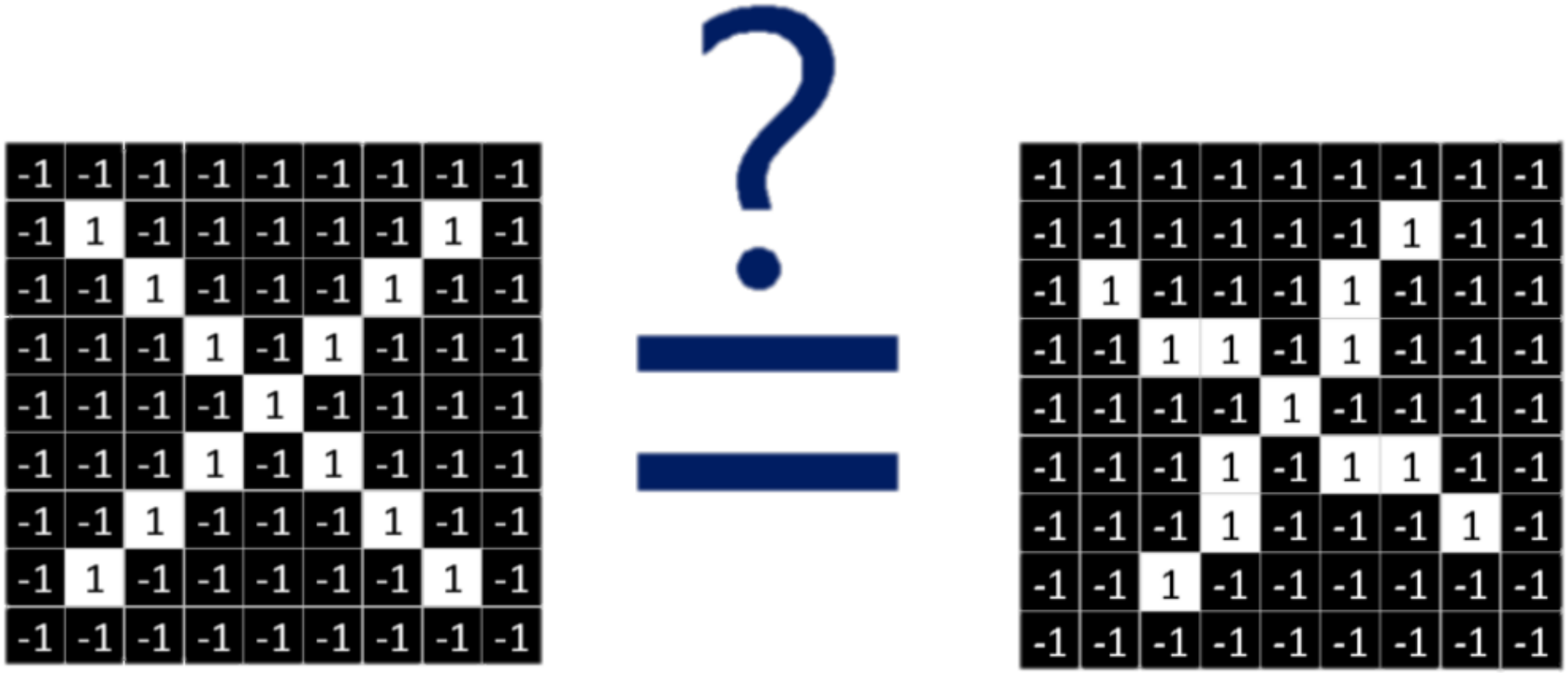
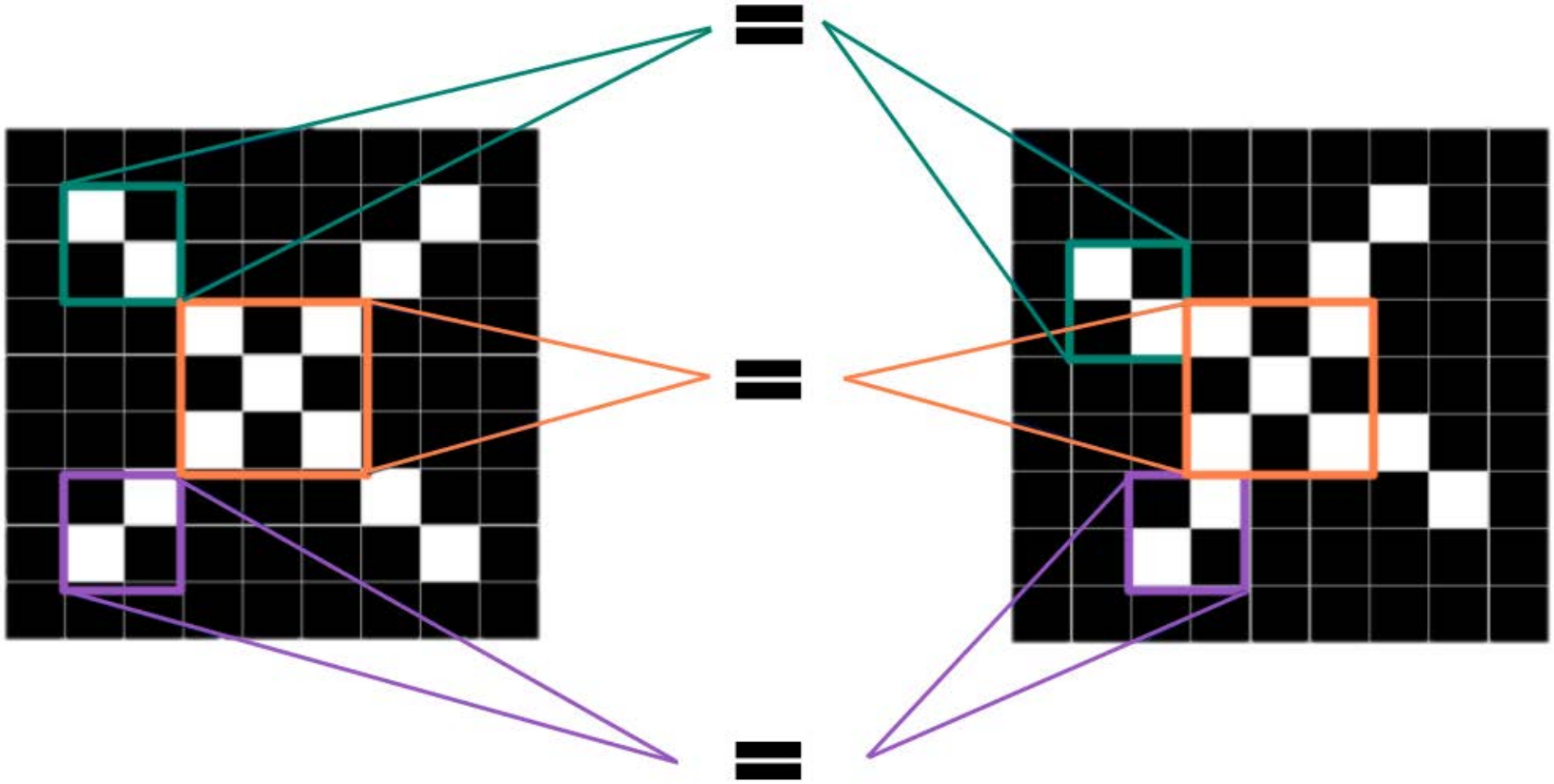


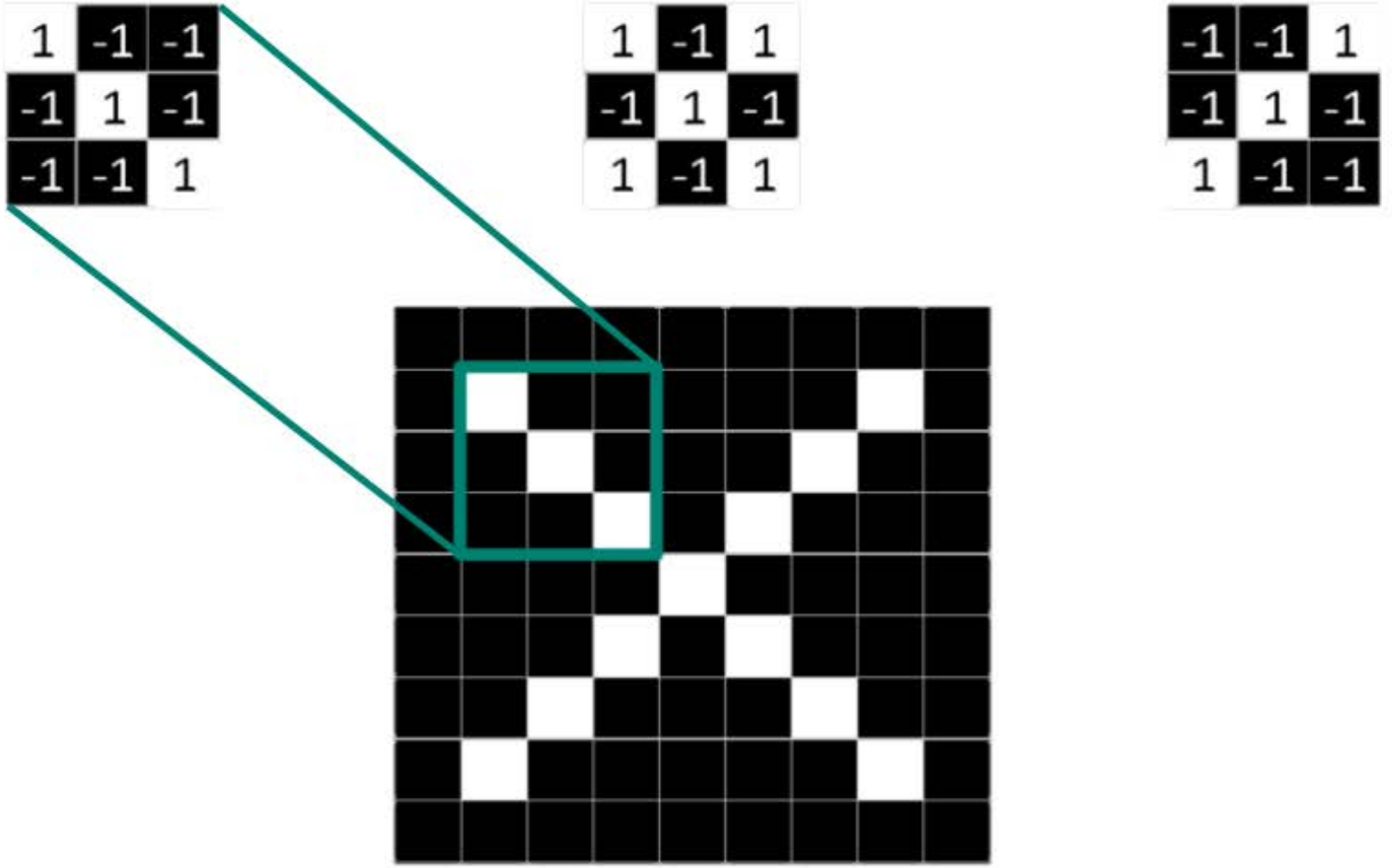
Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Features of X

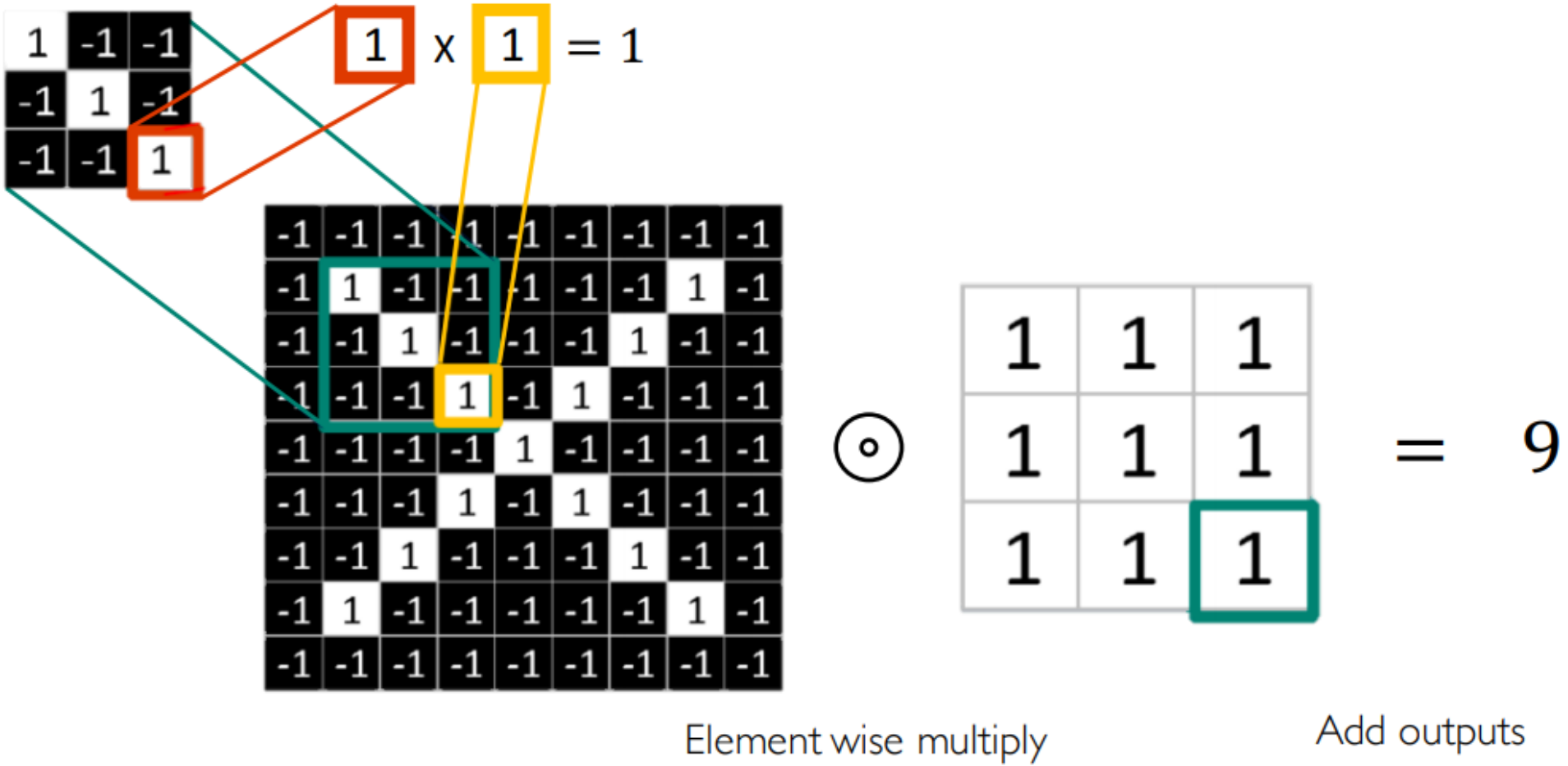


Filters to Detect X Features

filters

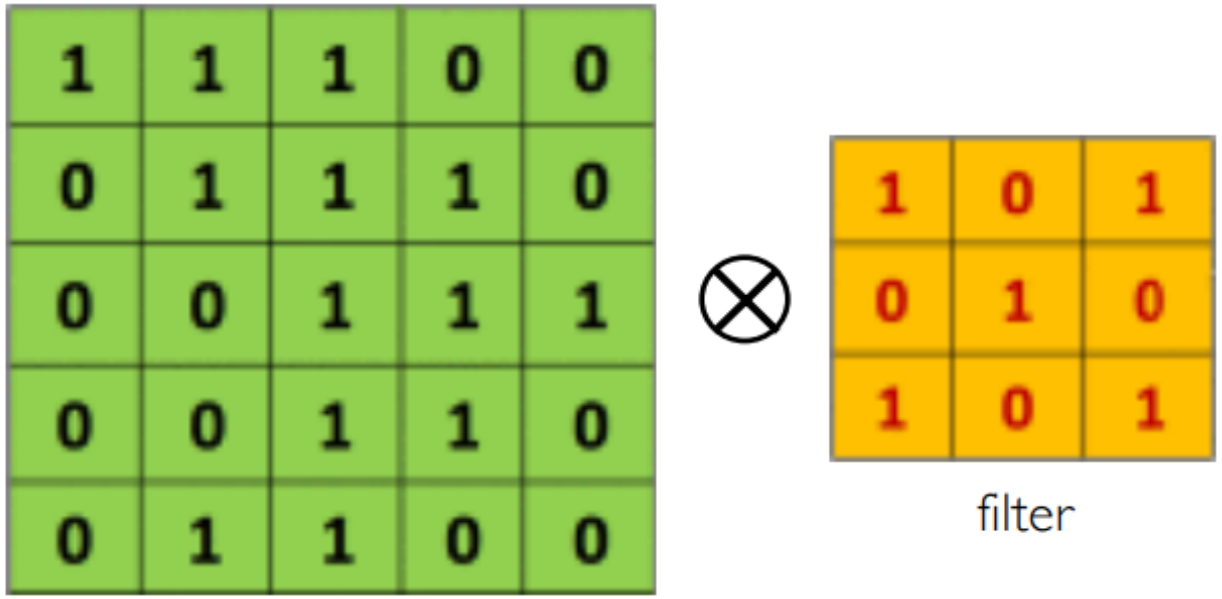


The Convolution Operation



The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



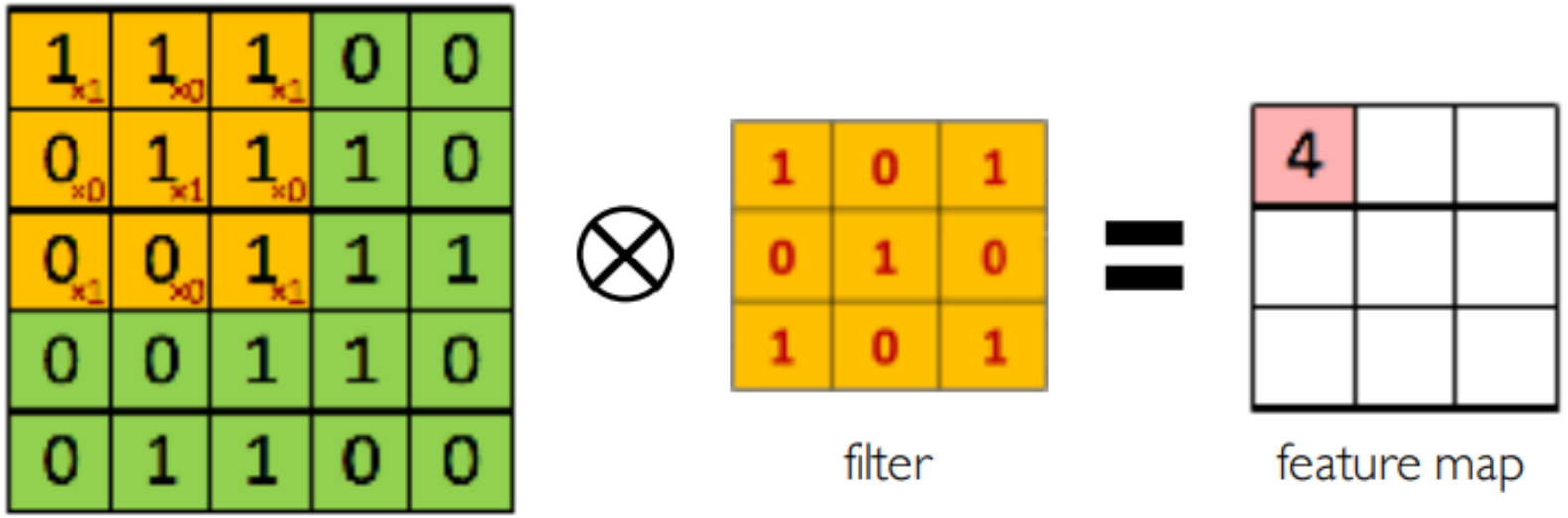
image

filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

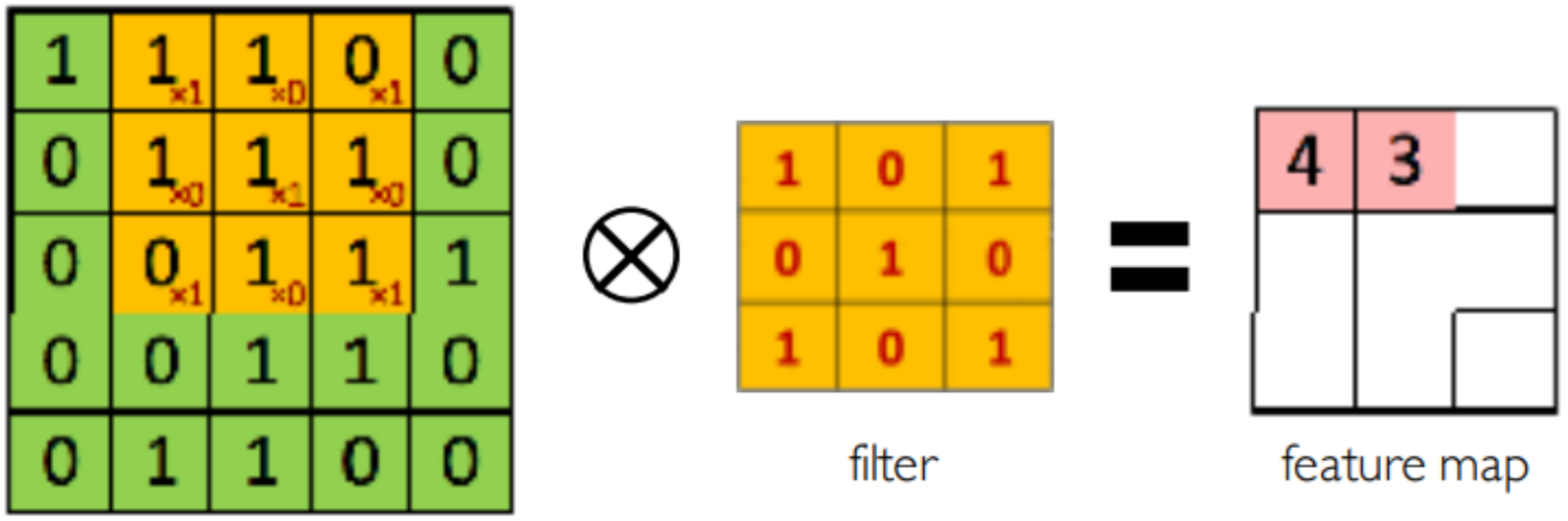
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



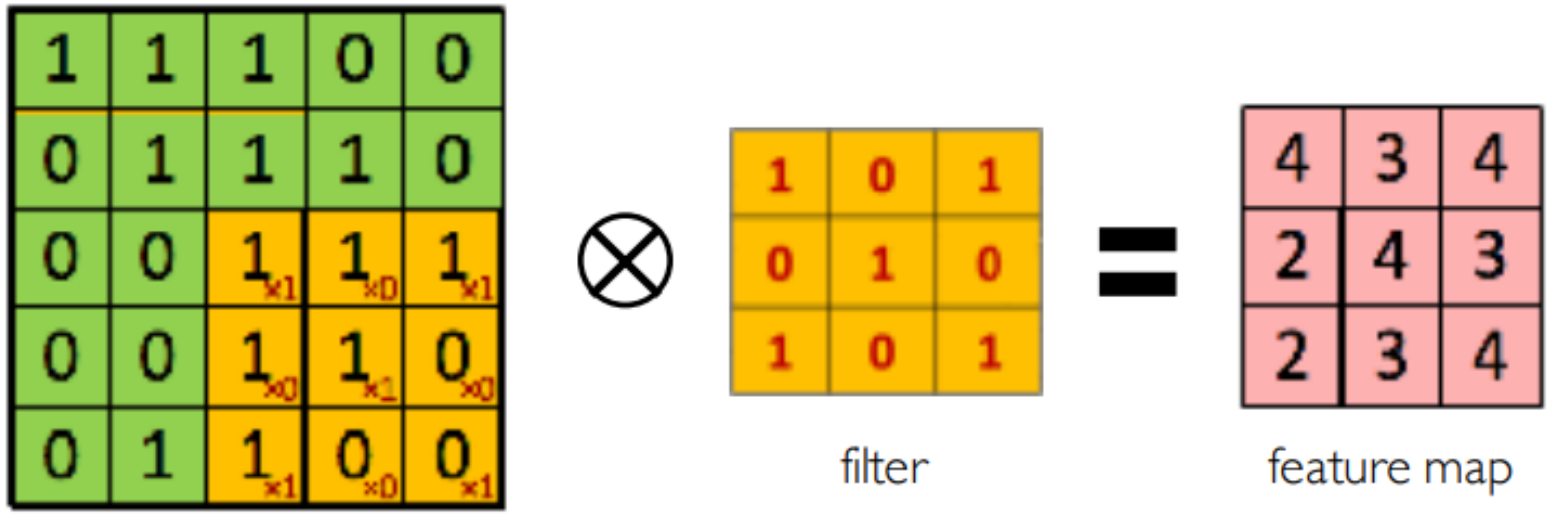
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Producing Feature Maps

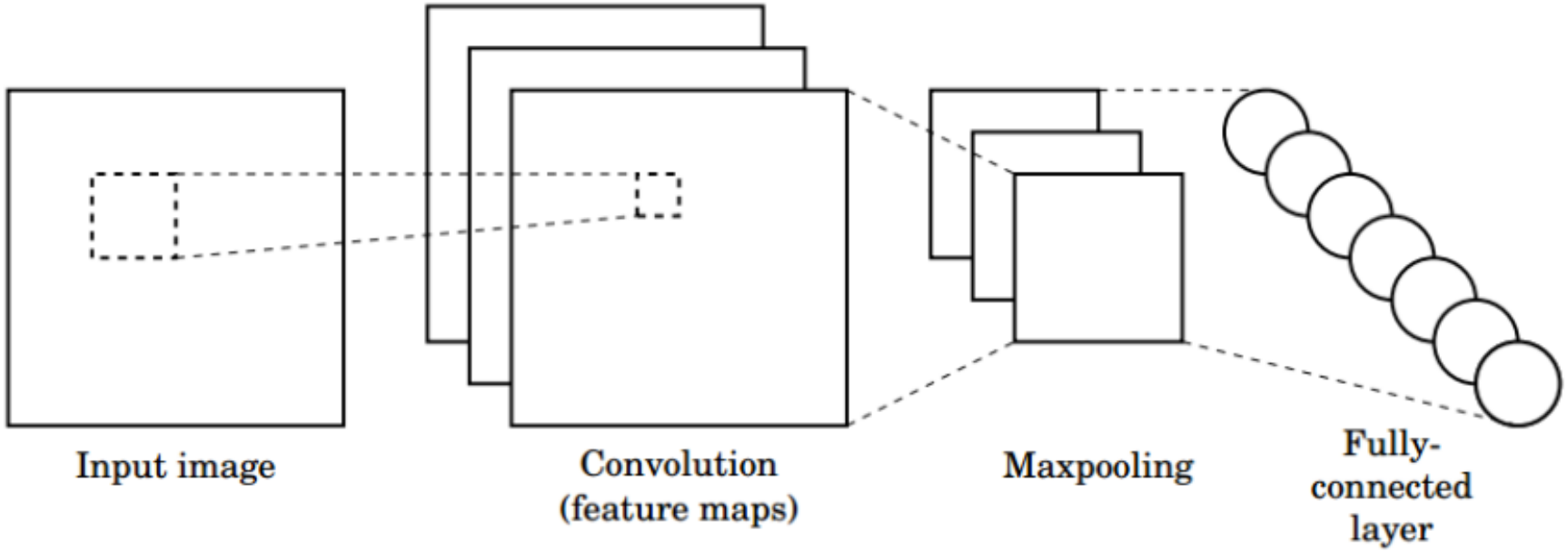
Original

Sharpen

Edge Detect

"Strong" Edge Detect

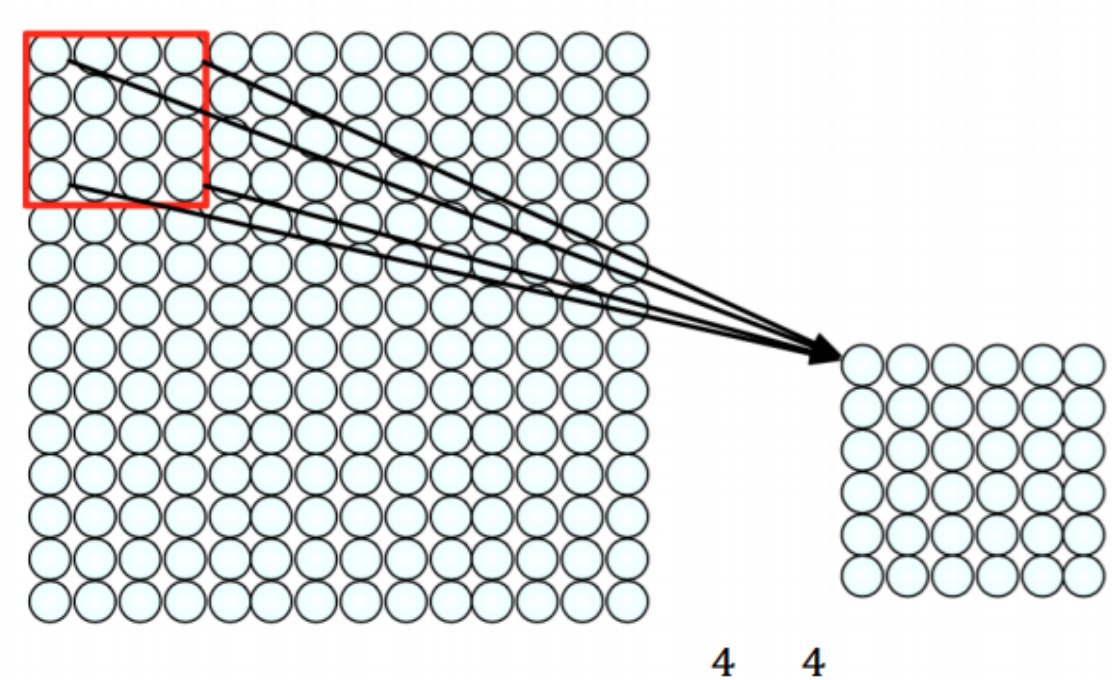
CNNs for Classification



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity



For a neuron in hidden layer:

- Take inputs from patch the neuron "sees"
- Compute weighted sum
- Apply bias

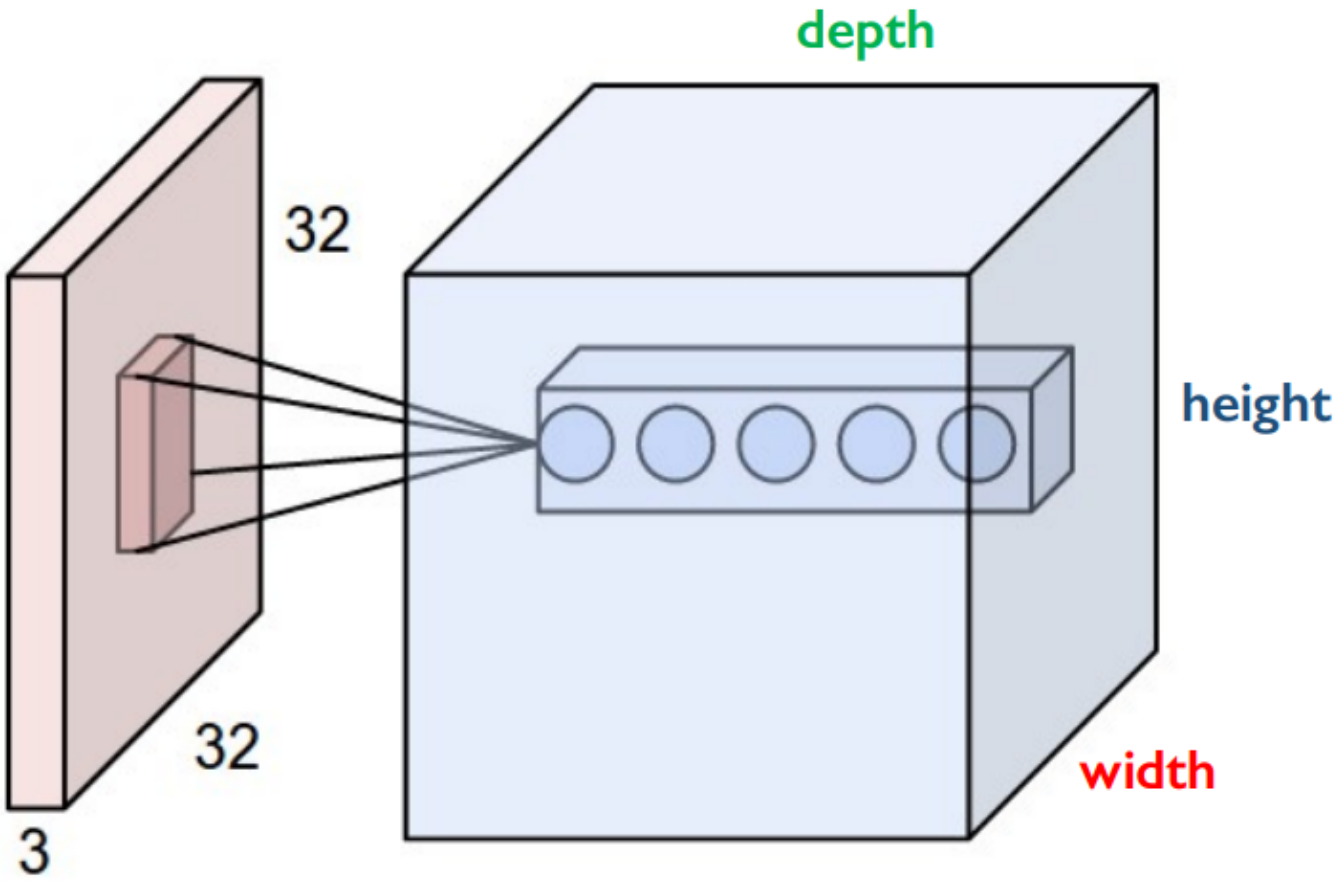
4x4 filter: matrix of weights θ_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} \theta_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

applying a window of weights
computing linear combinations
activating with non-linear function

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \; x \; w \; x \; d$

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

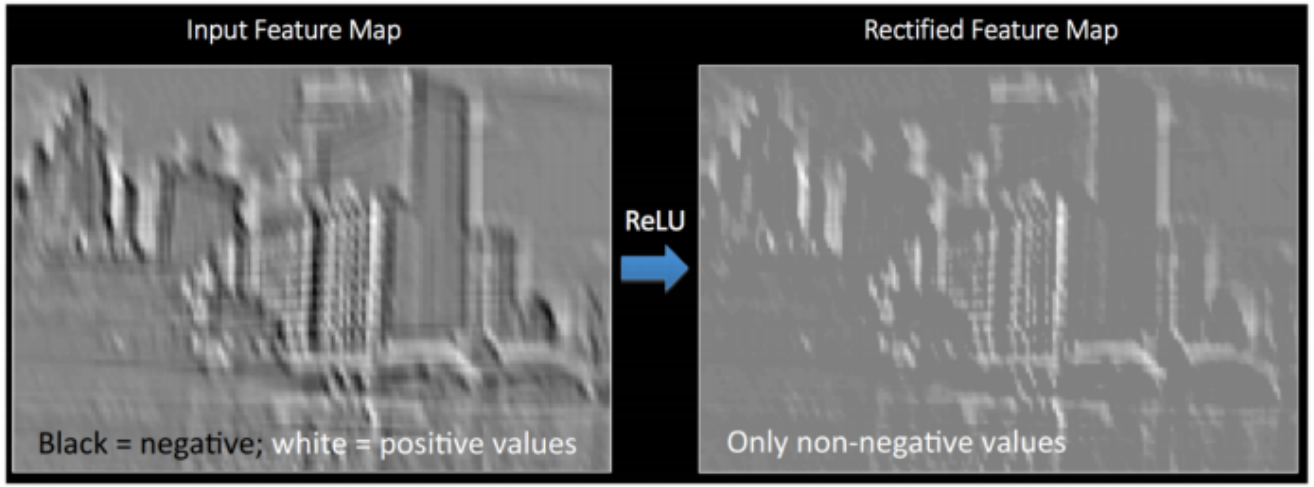
Filter step size

Receptive Field:

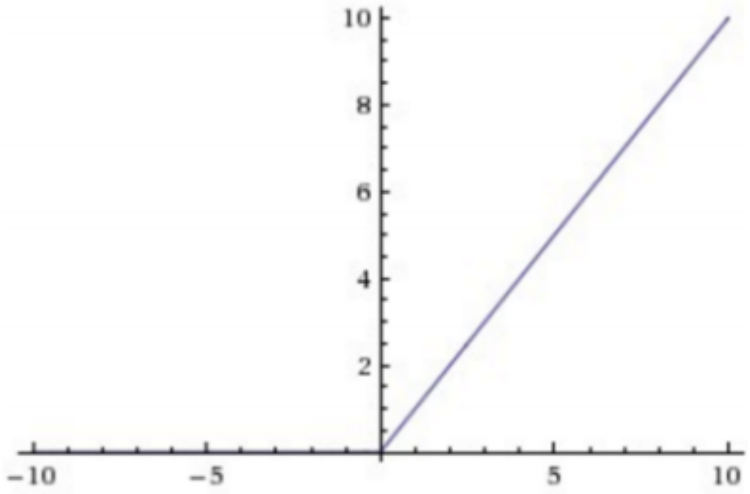
Locations in input image that
a node is path connected to

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation**

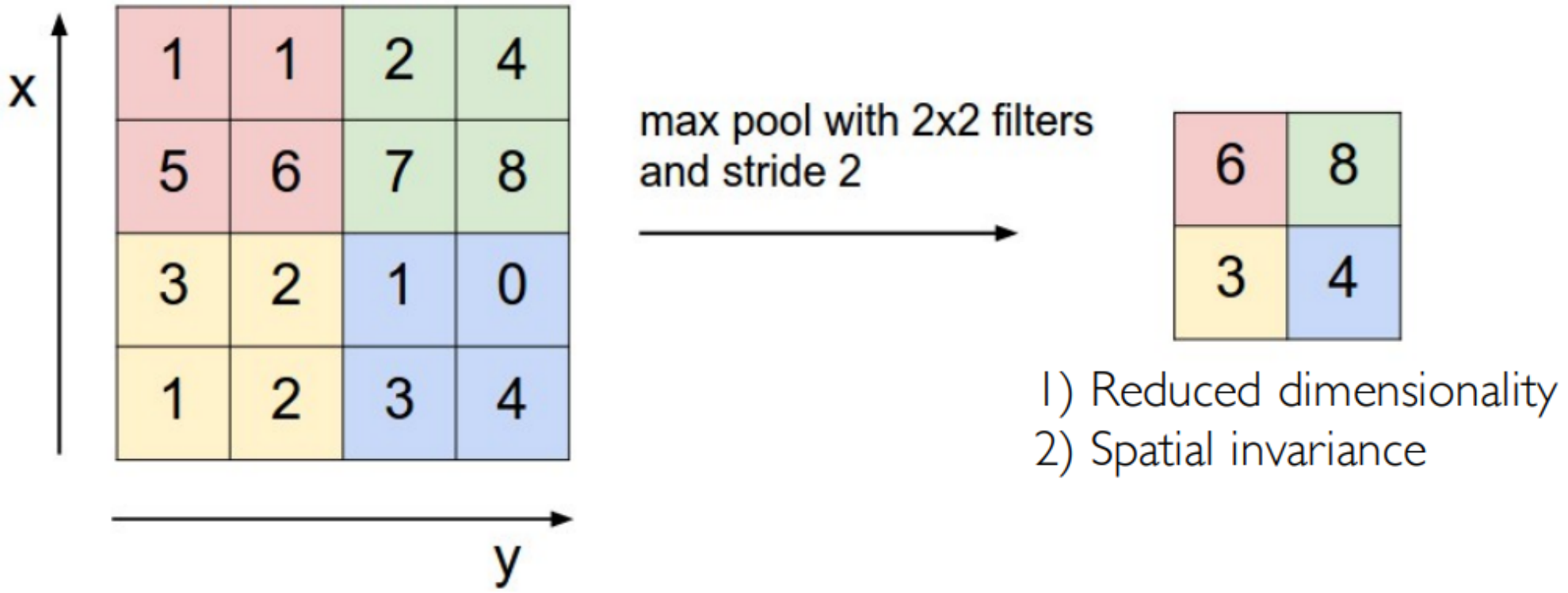


Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

Pooling

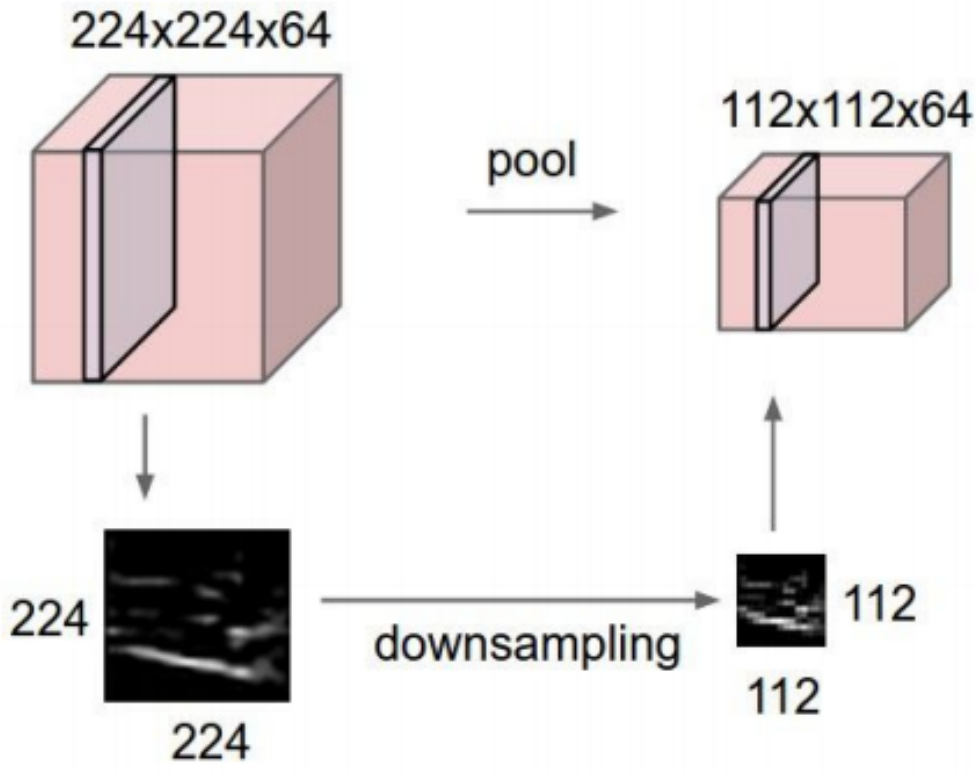


1) Reduced dimensionality
2) Spatial invariance

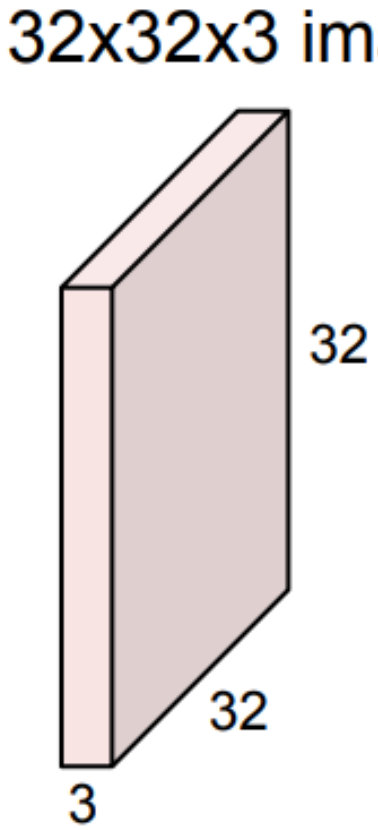
How else can we downsample and preserve spatial invariance?

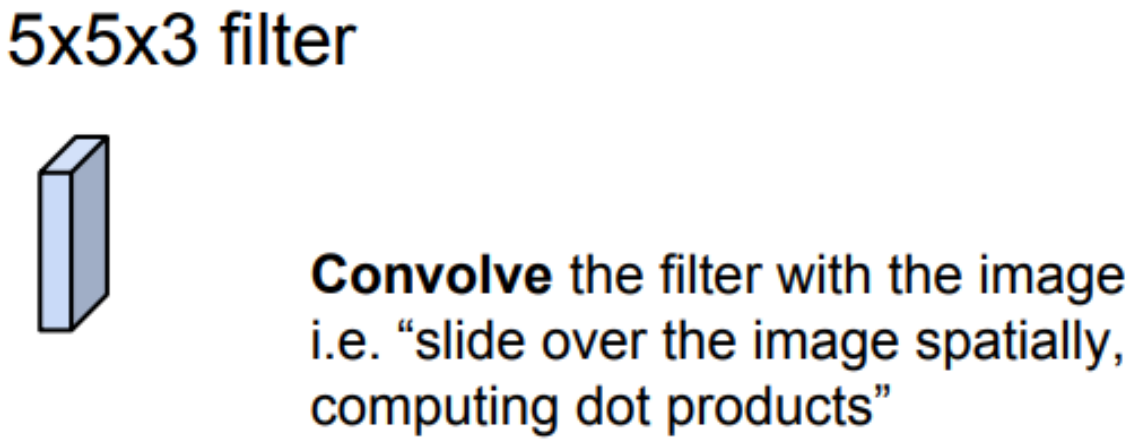
Pooling

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

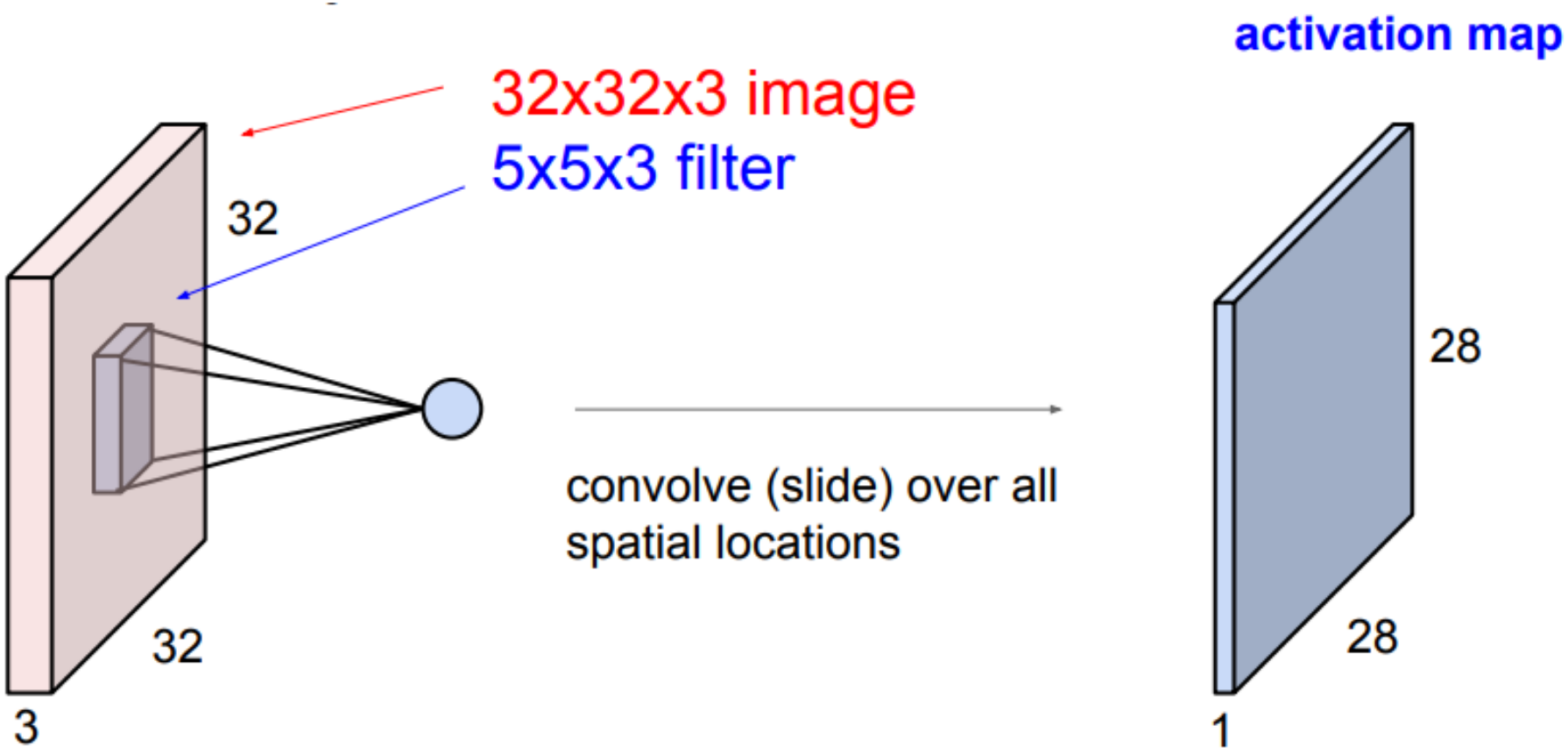


Convolution Layer

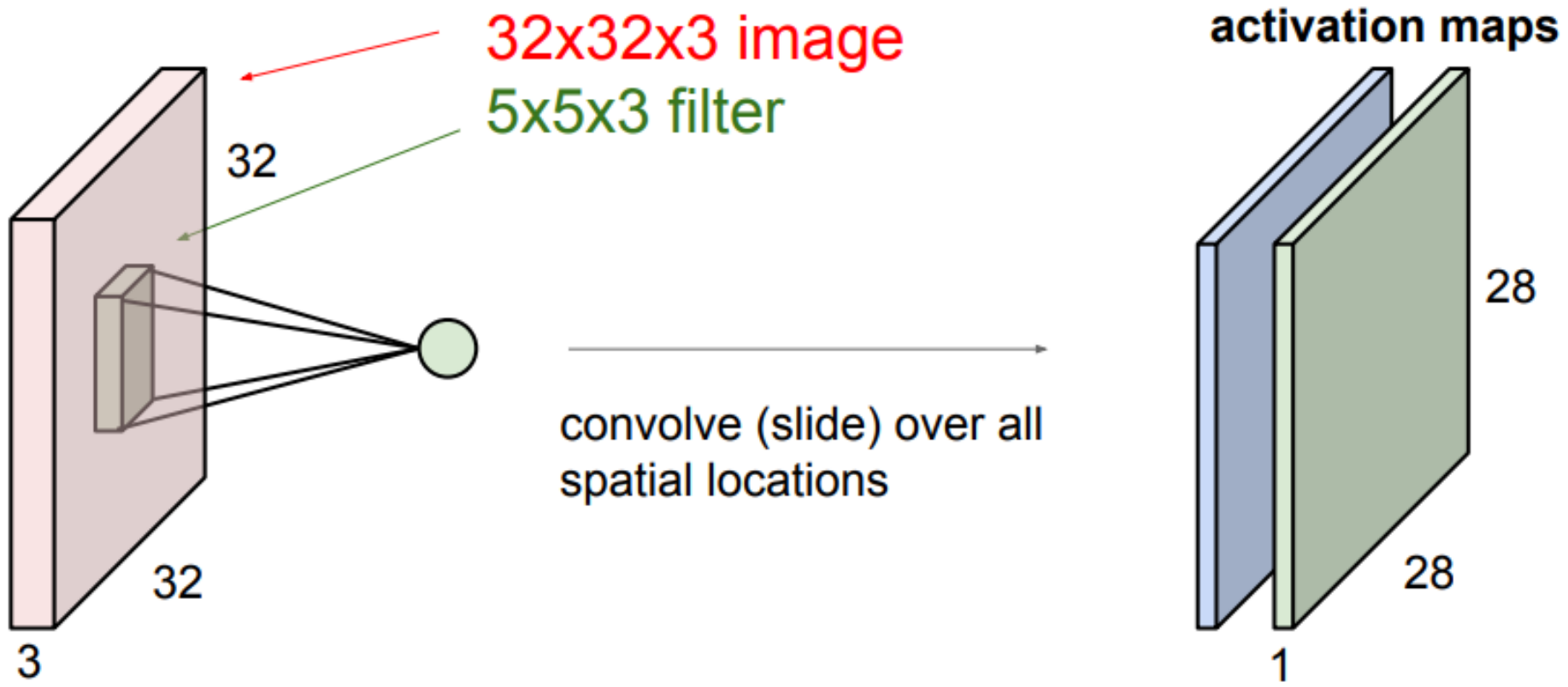




Convolution Layer

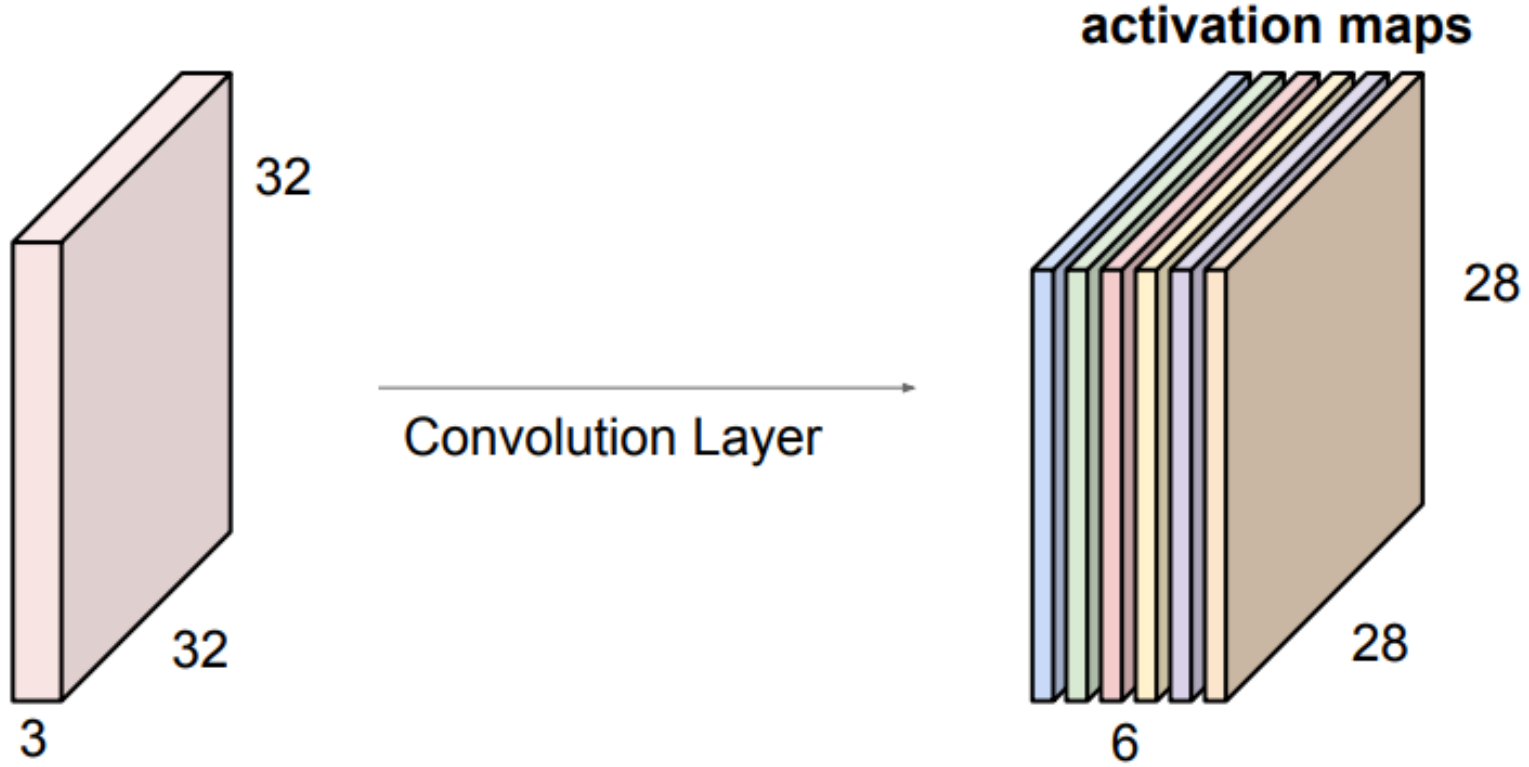


Convolution Layer



Convolution Layer

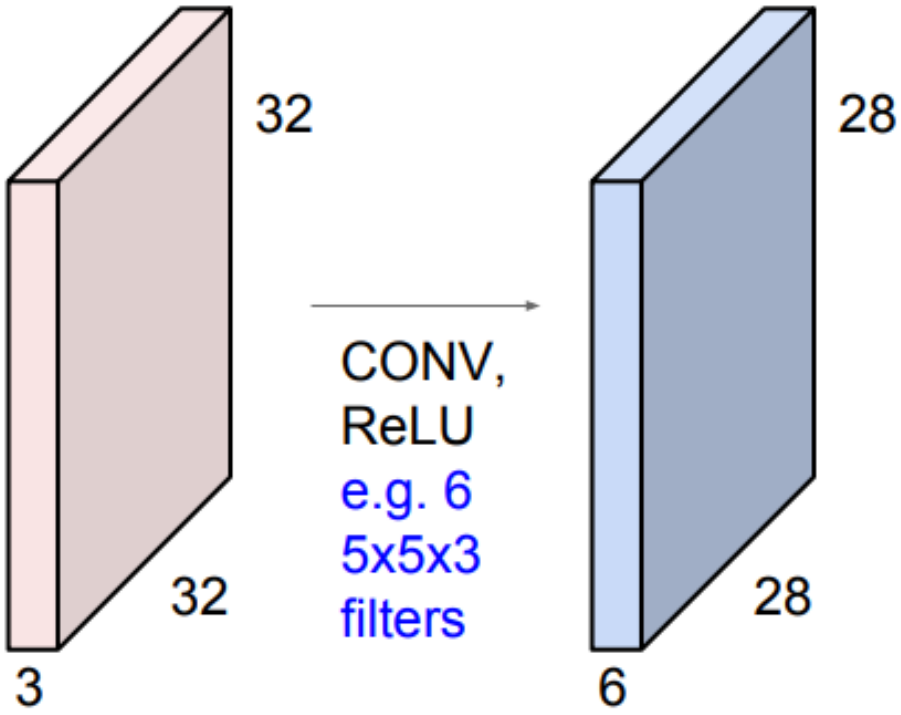
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

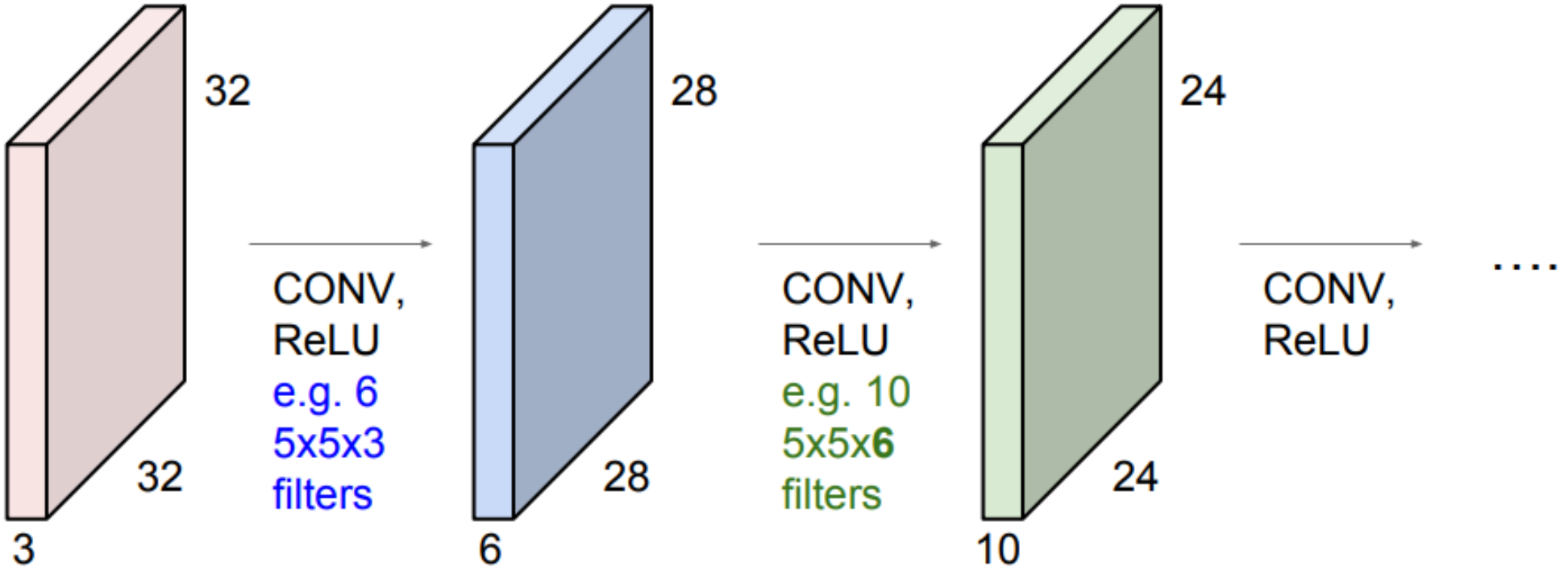
Convolution Layer

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



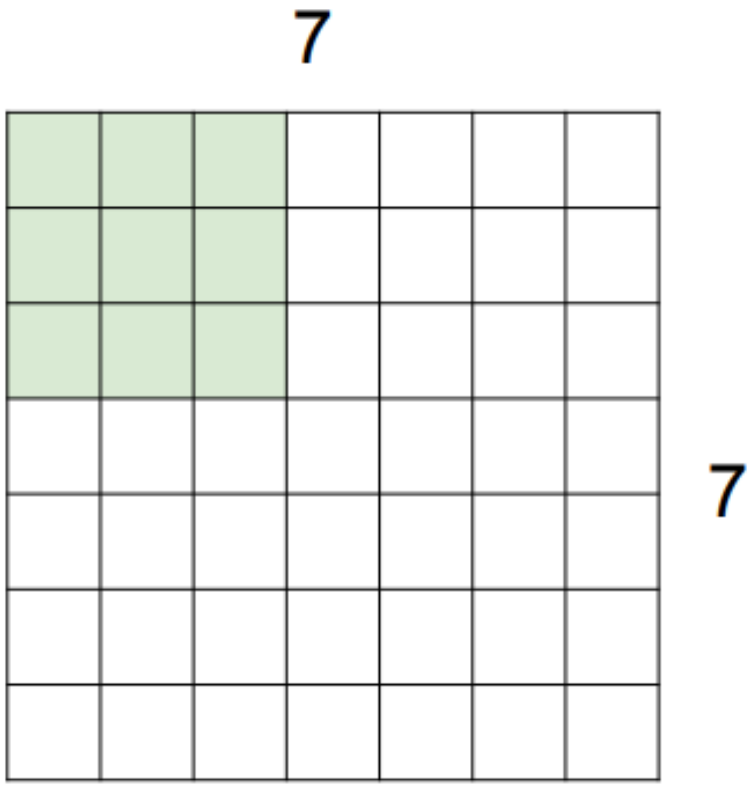
Convolution Layer

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



Stride

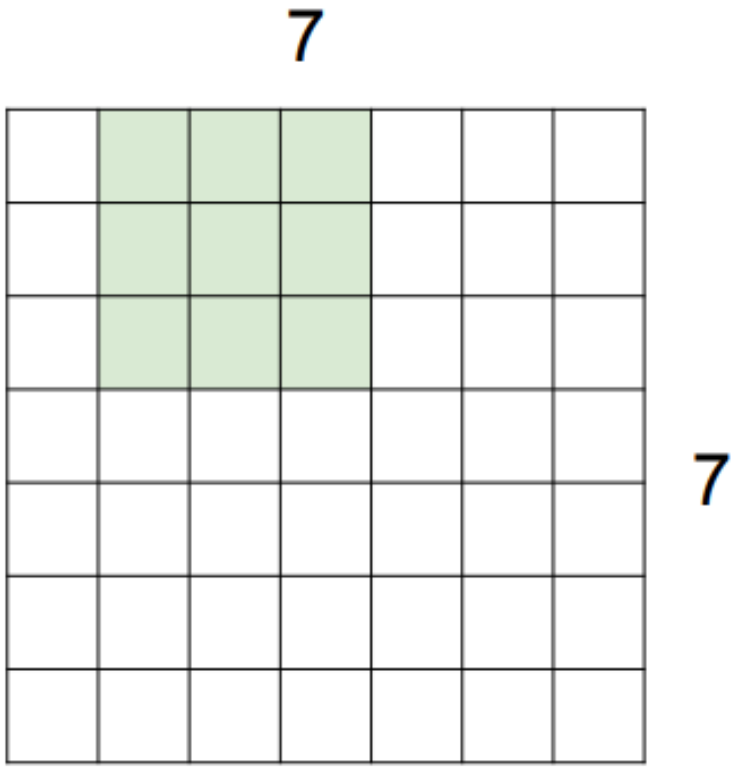
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Stride

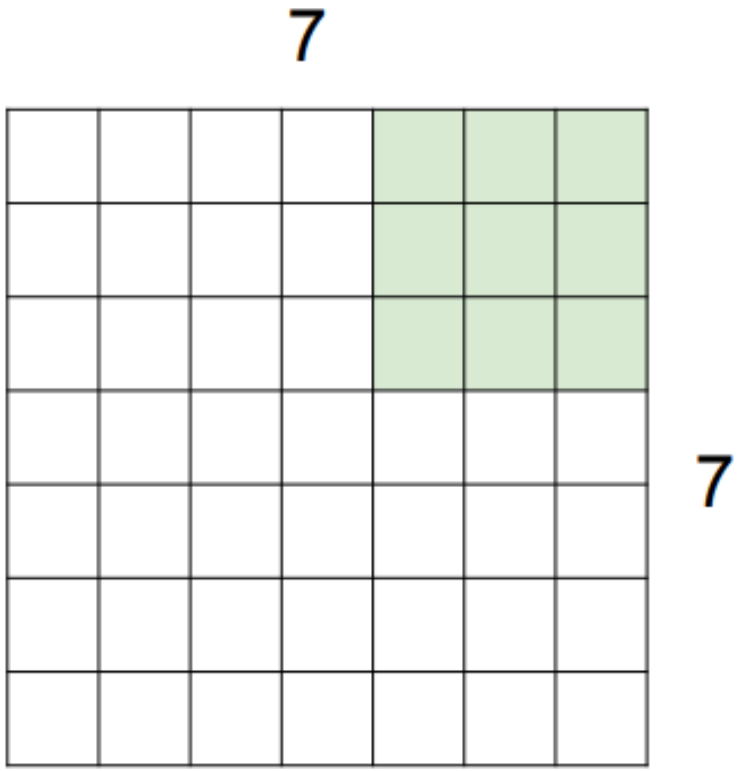
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Stride

A closer look at spatial dimensions:

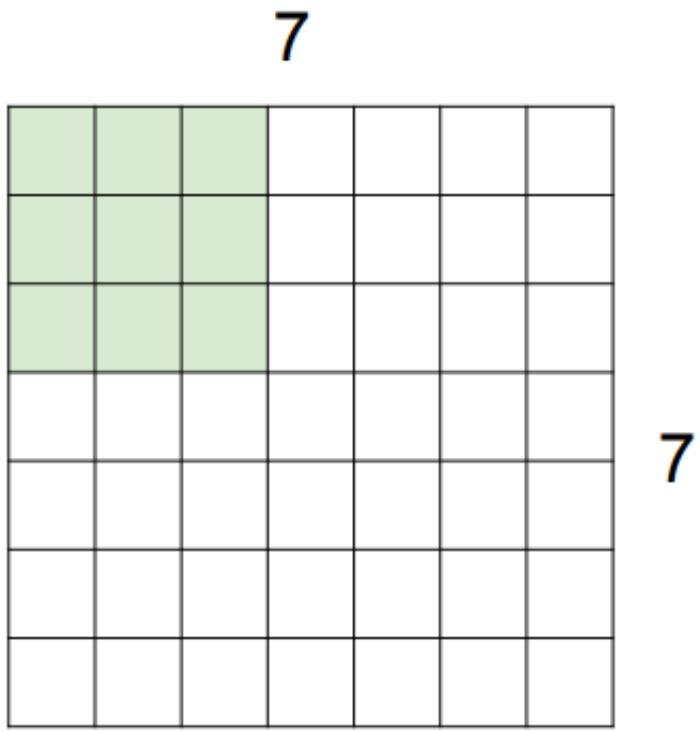


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride

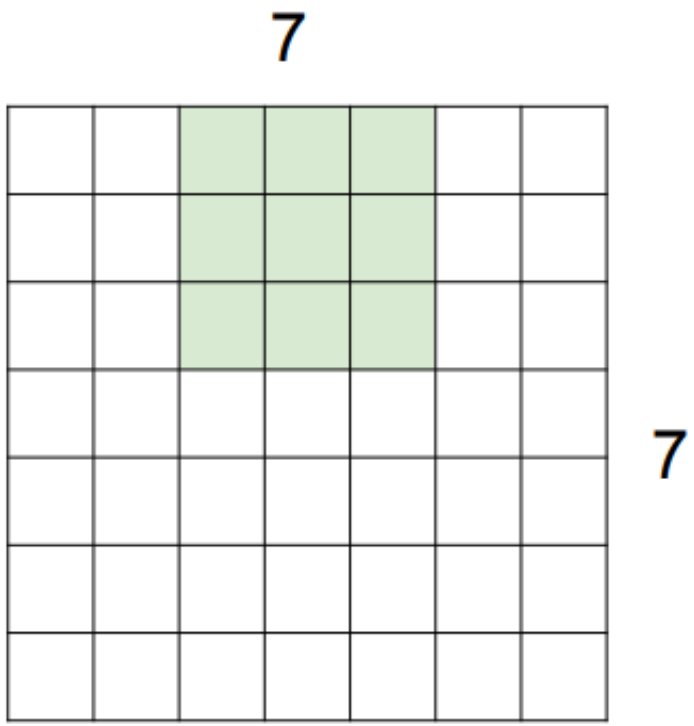
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride

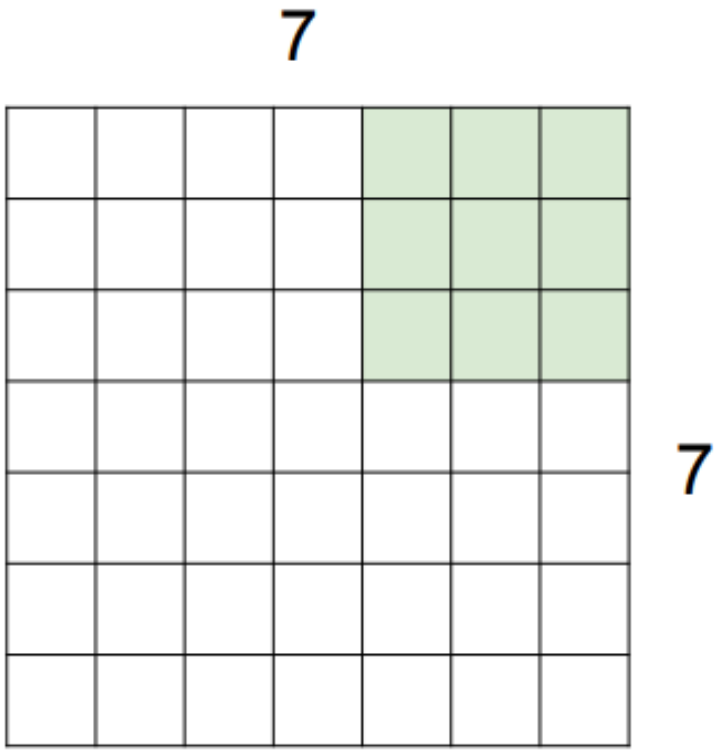
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride

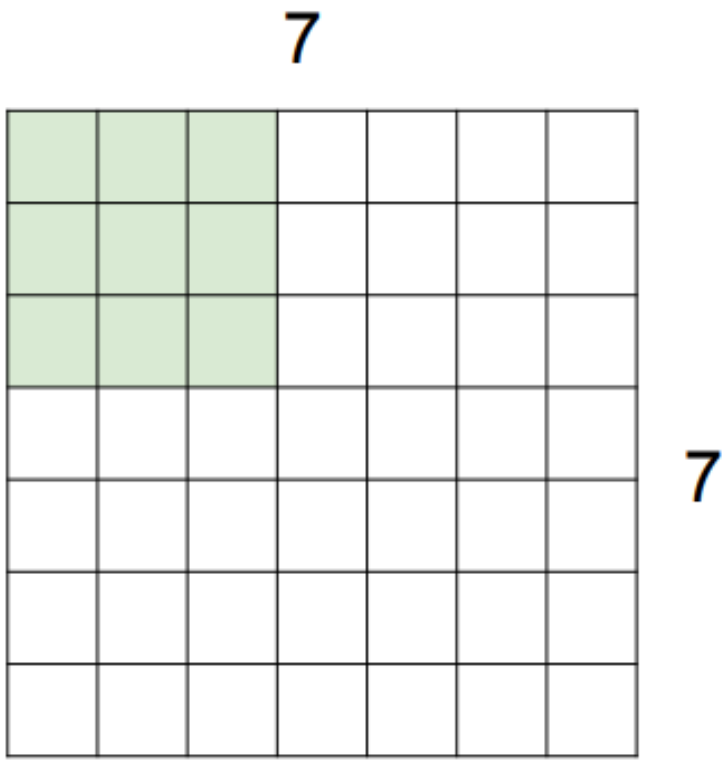
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Stride

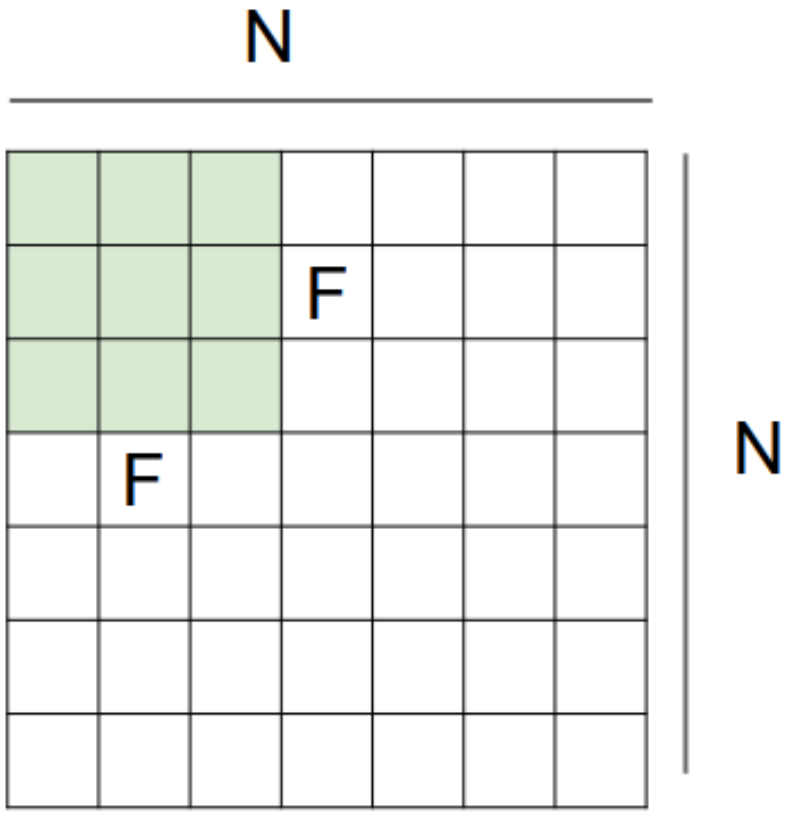
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

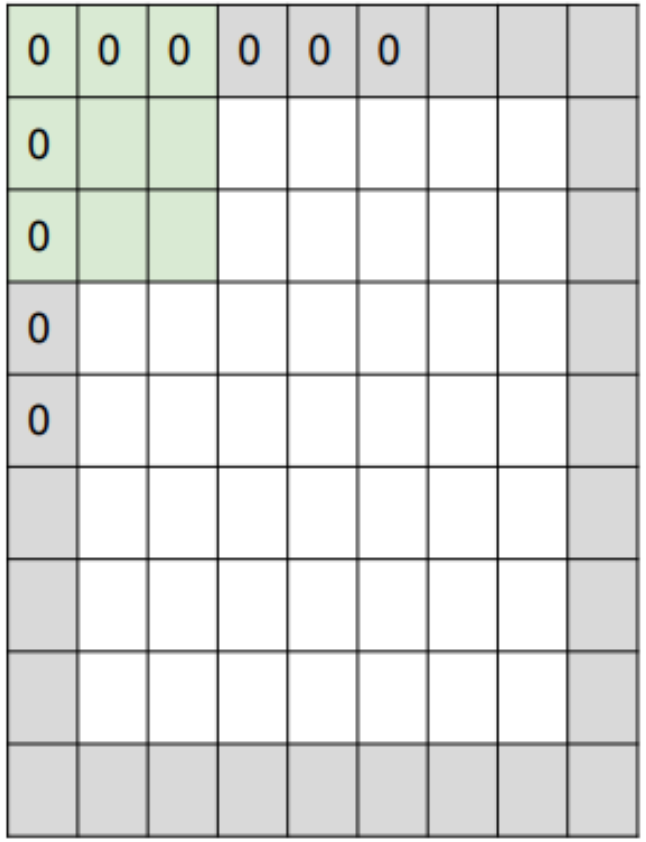
Stride



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

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

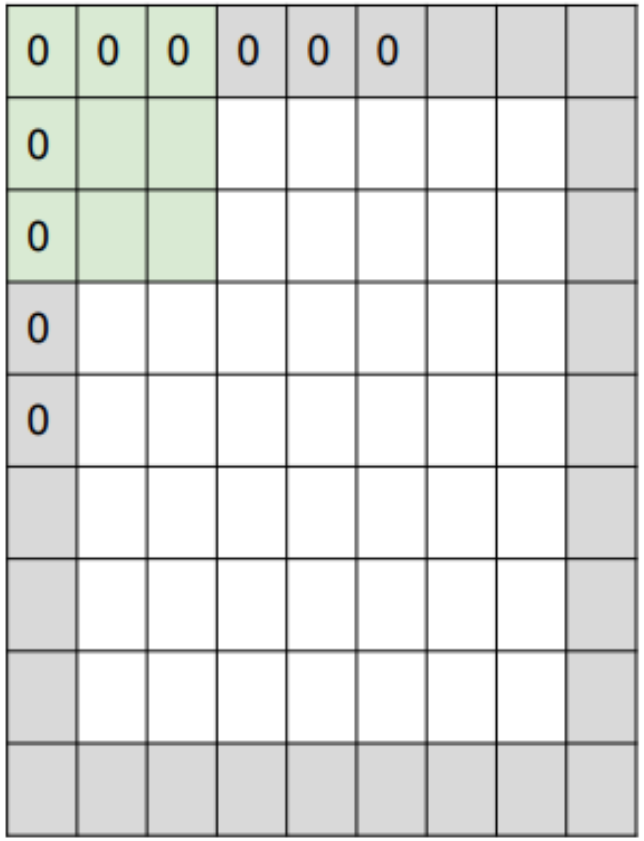
In practice: Common to zero pad the border



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

(recall:)
(N - F) / stride + 1

In practice: Common to zero pad the border

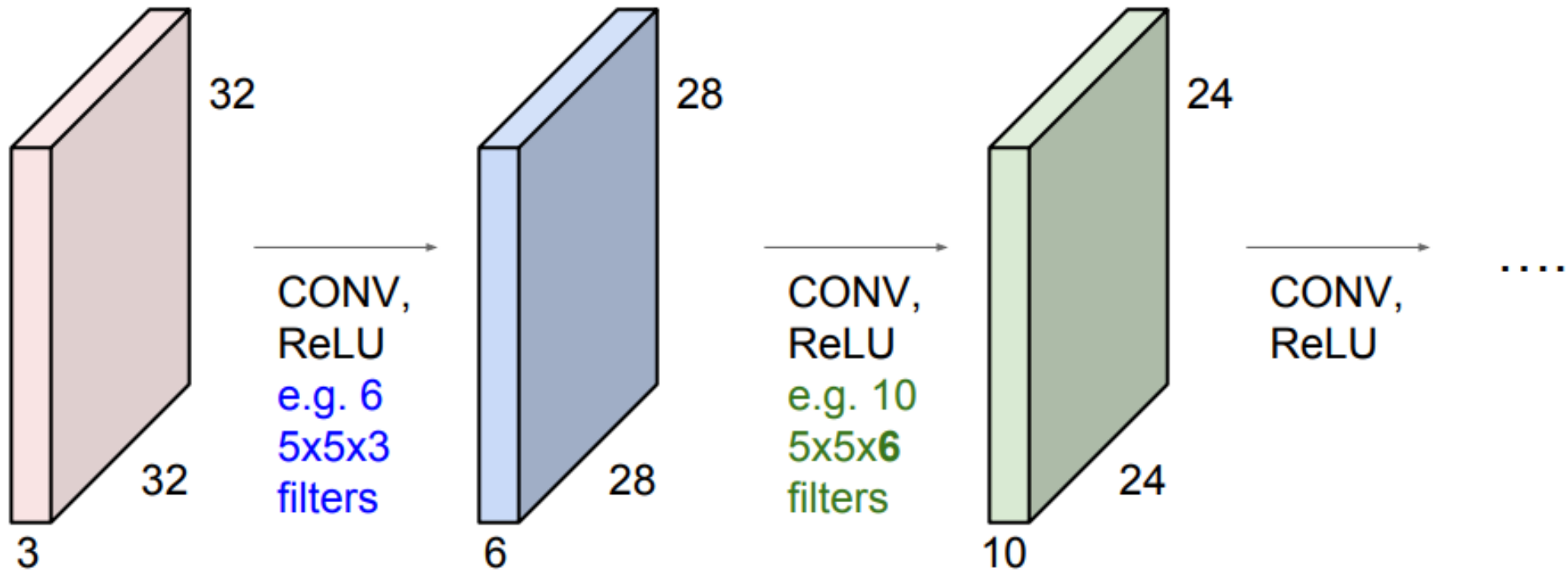


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

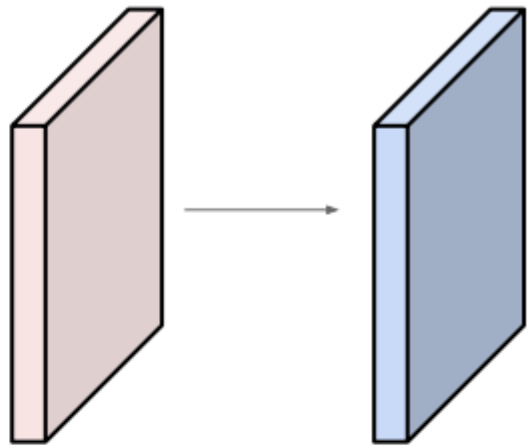
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



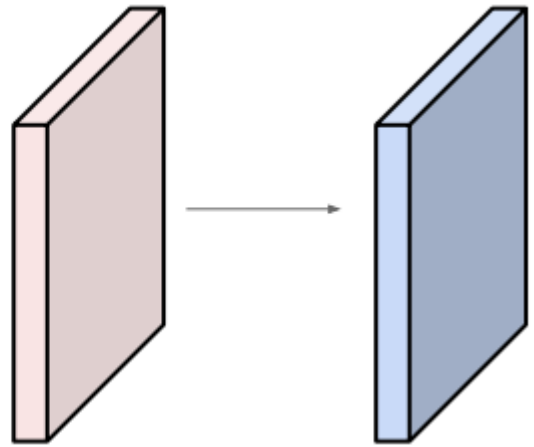
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



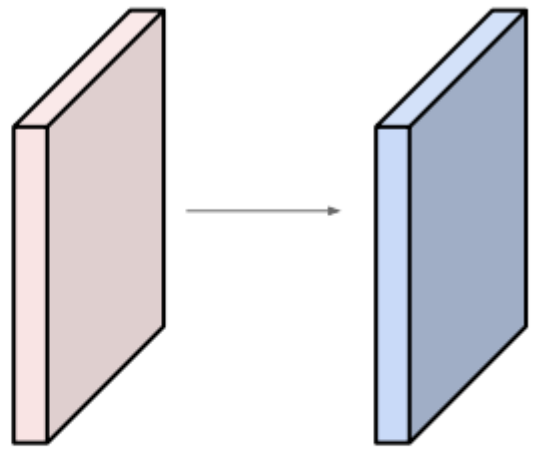
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

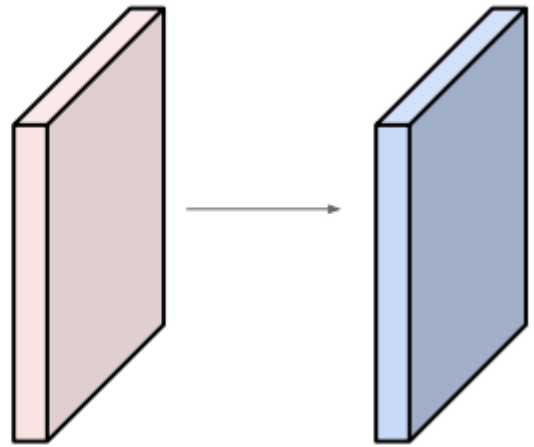
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Examples time:



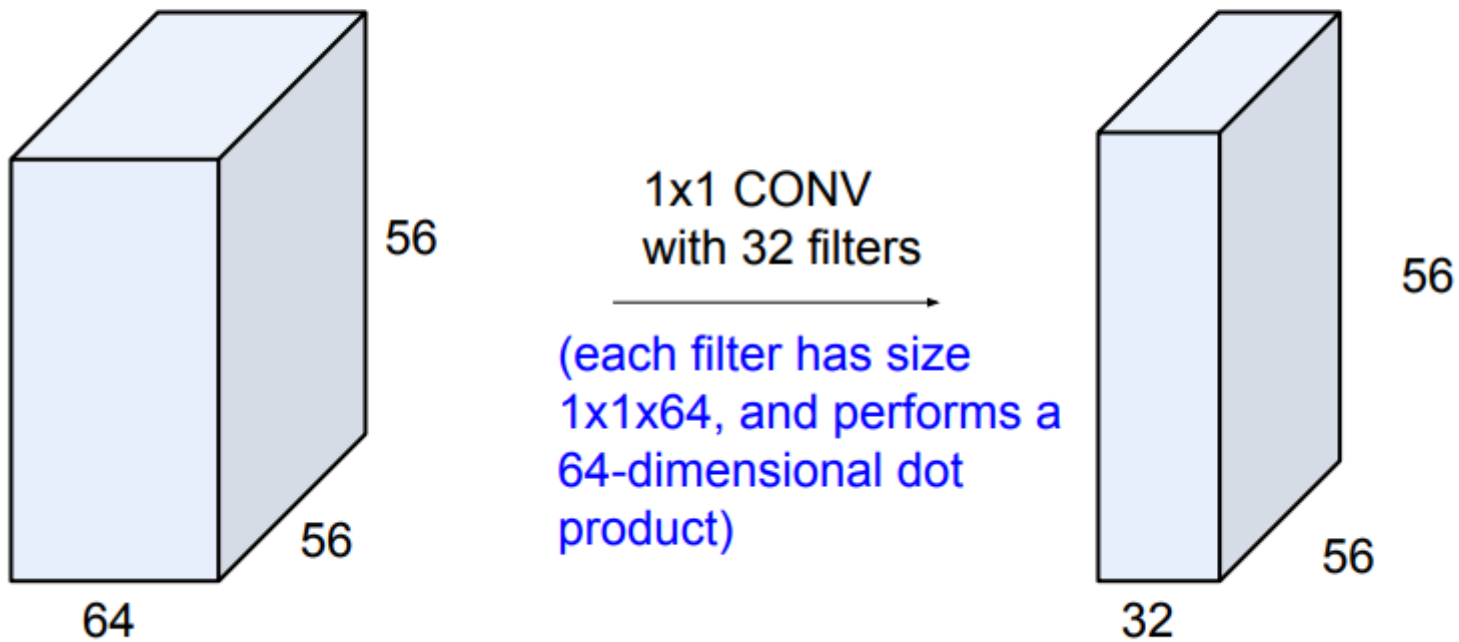
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

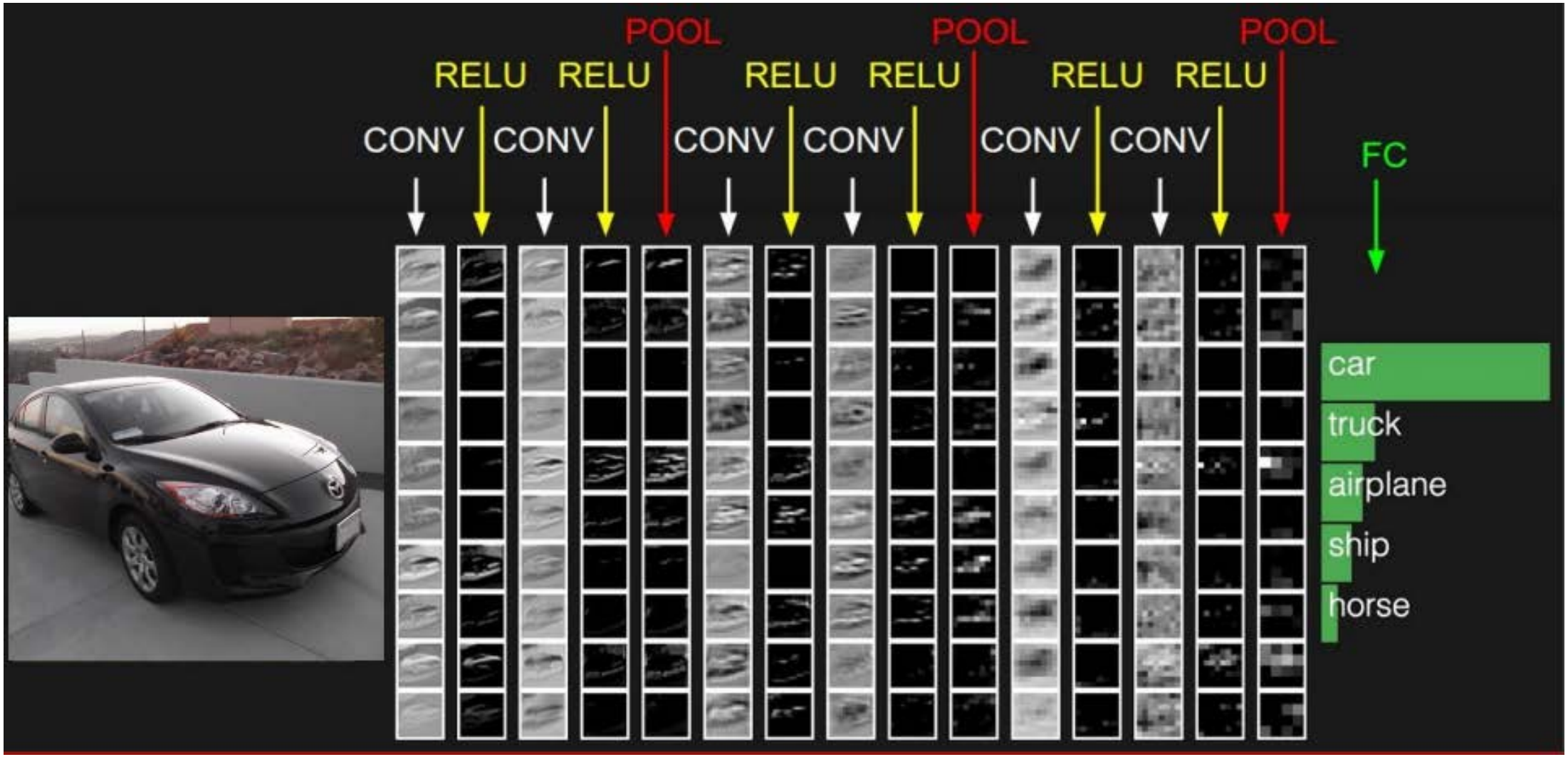
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

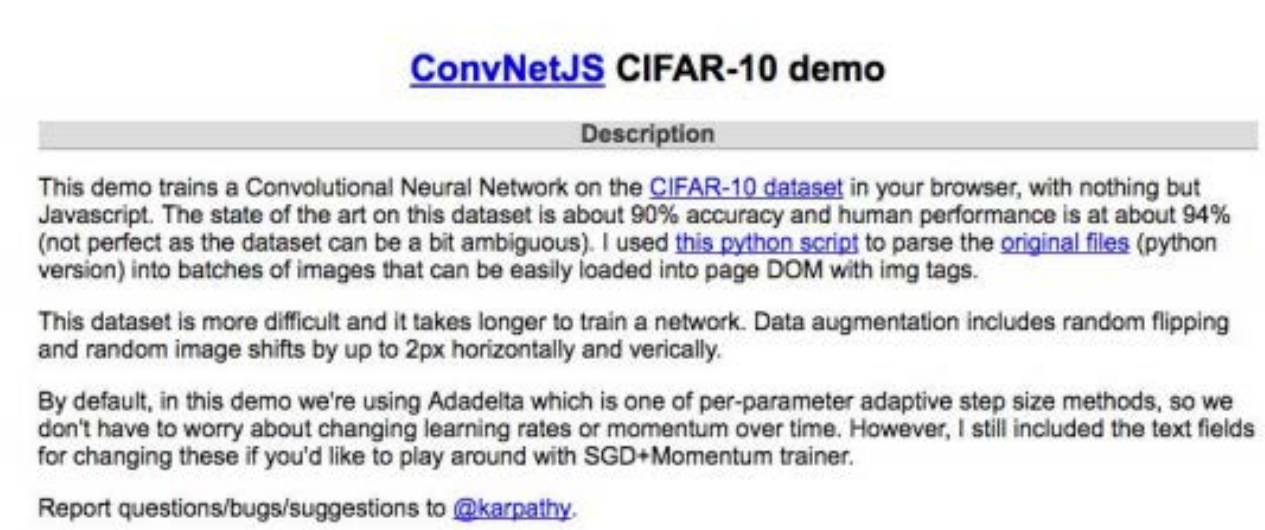
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

(btw, 1x1 convolution layers make perfect sense)



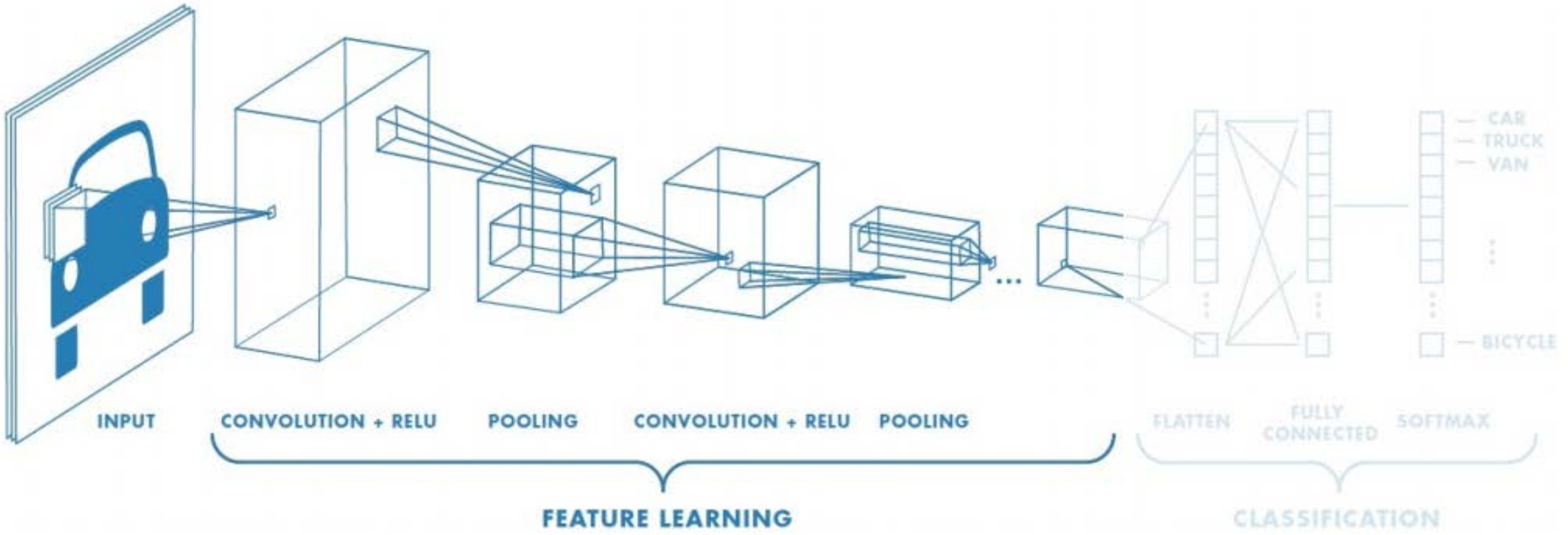

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

[ConvNetJS demo: training on CIFAR-10]



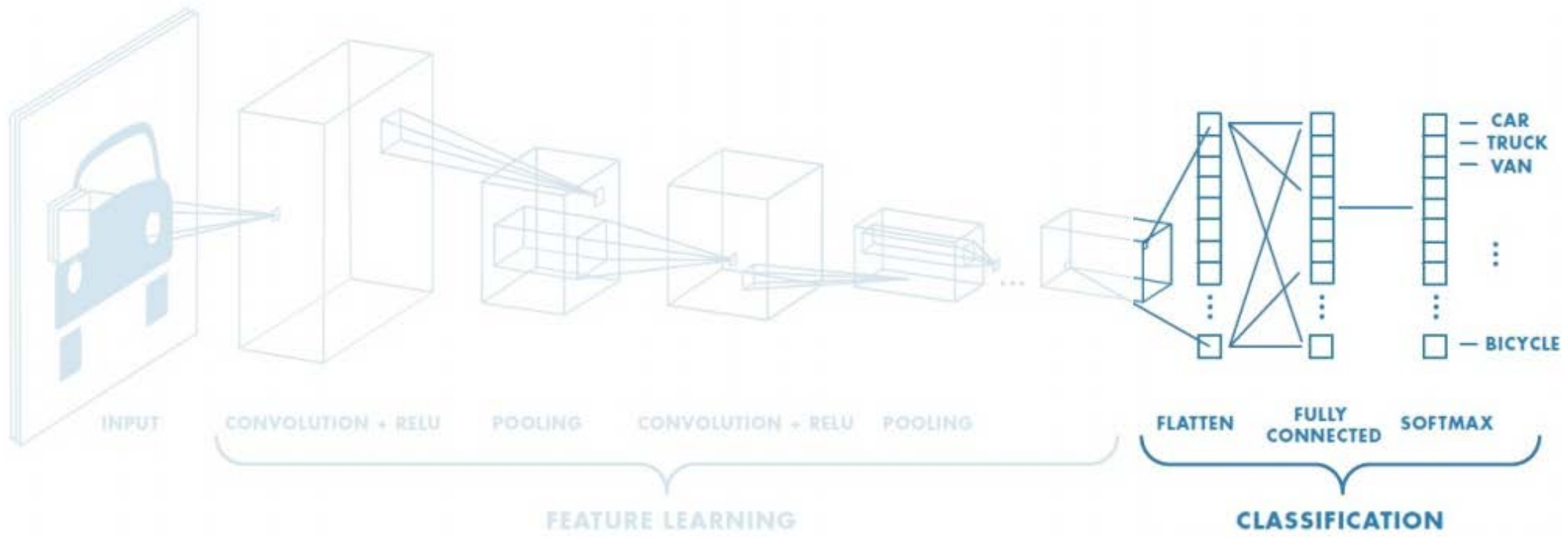
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

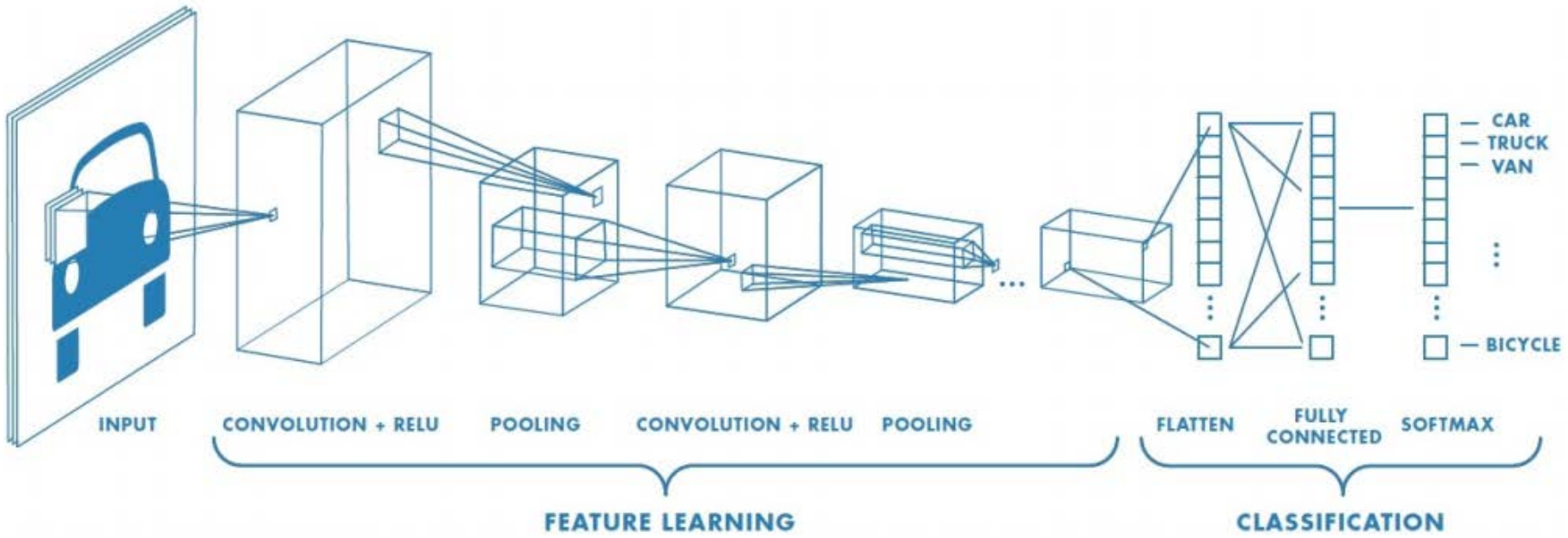
CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNNs: Training with Backpropagation



Learn weights for convolutional filters and fully connected layers

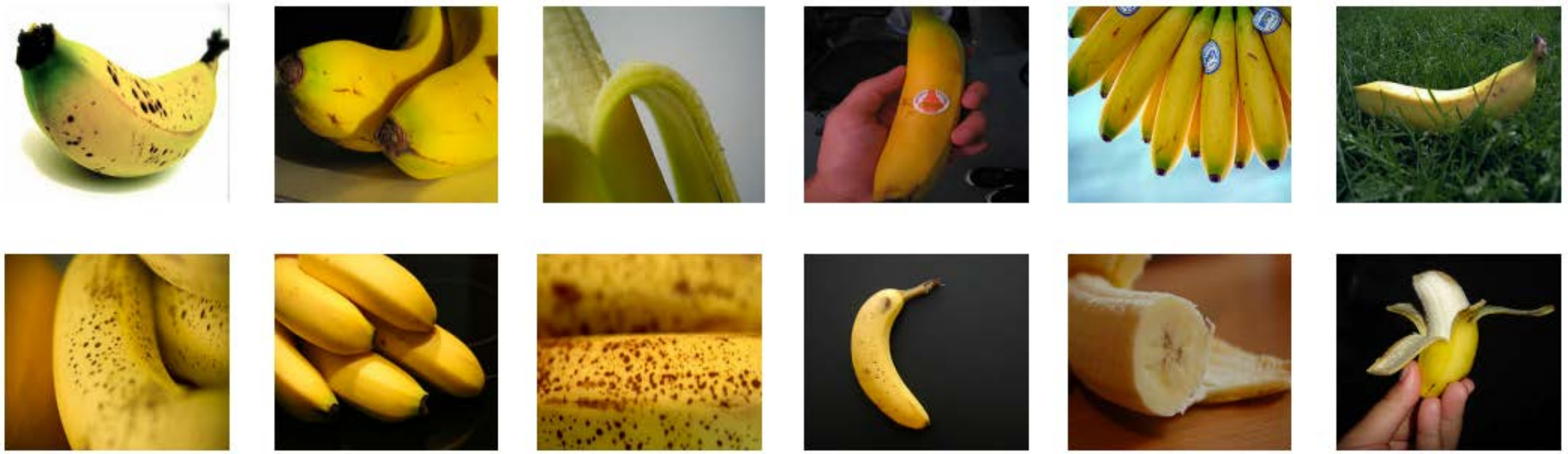
Backpropagation: cross-entropy loss

$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$

ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

"Elongated crescent-shaped yellow fruit with soft sweet flesh"



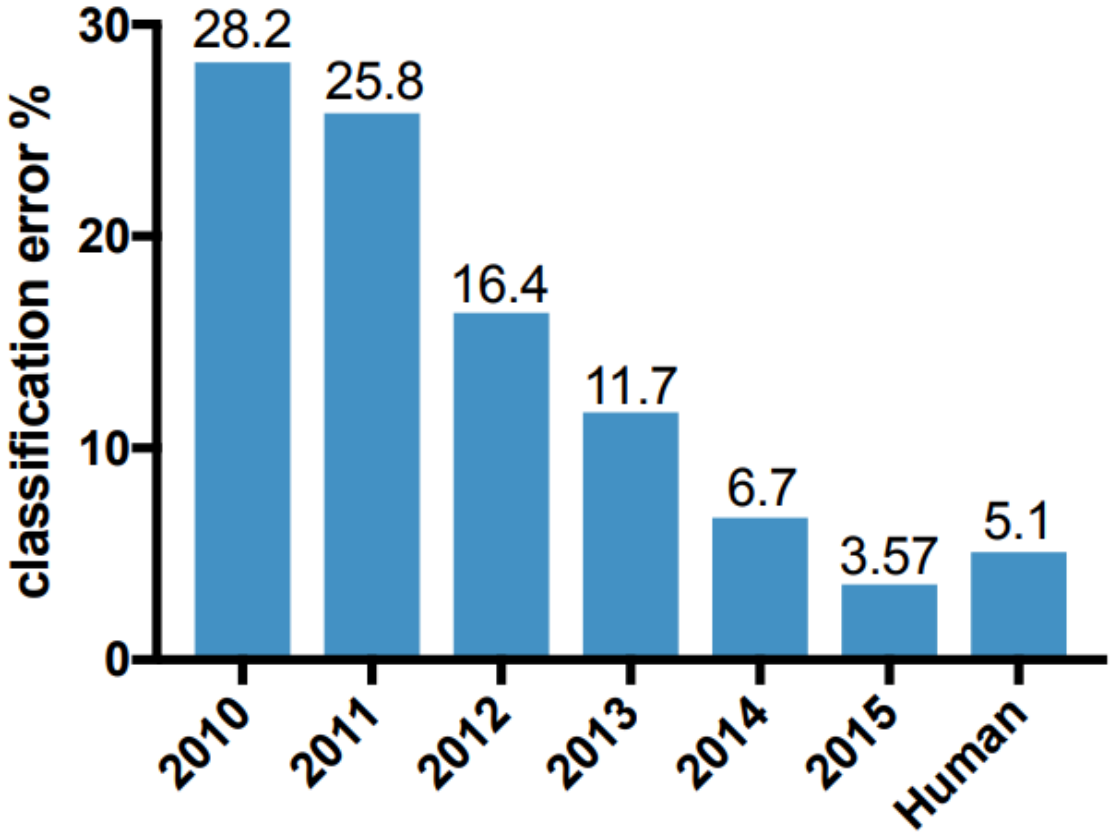
1409 pictures of bananas.

ImageNet Challenge

Classification task: produce a list of object categories present in image. 1000 categories.
"Top 5 error": rate at which the model does not output correct label in top 5 predictions

Other tasks include:
single-object localization, object detection from video/image, scene classification, scene parsing

ImageNet Challenge: Classification Task



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

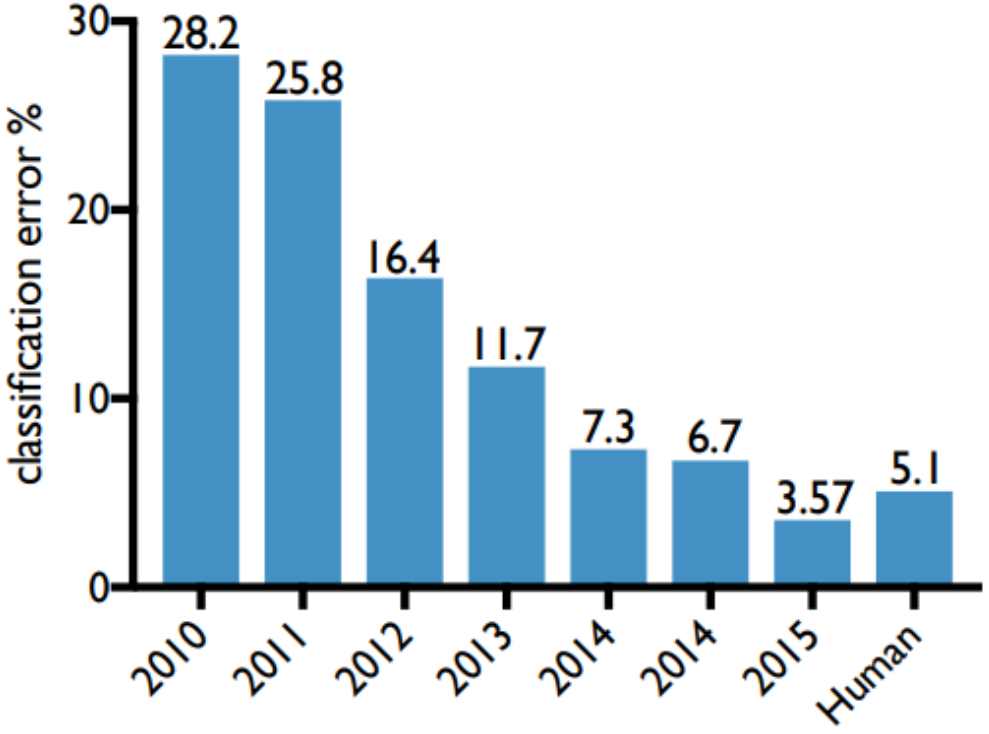
2013: ZFNet
- 8 layers, more filters

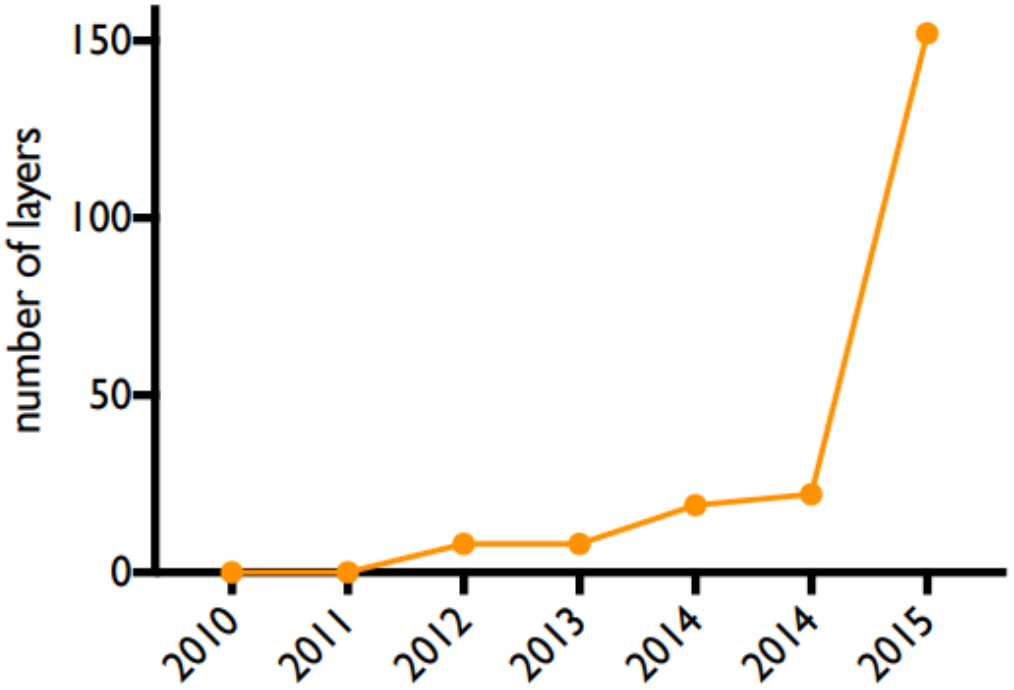
2014: VGG
- 19 layers

2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

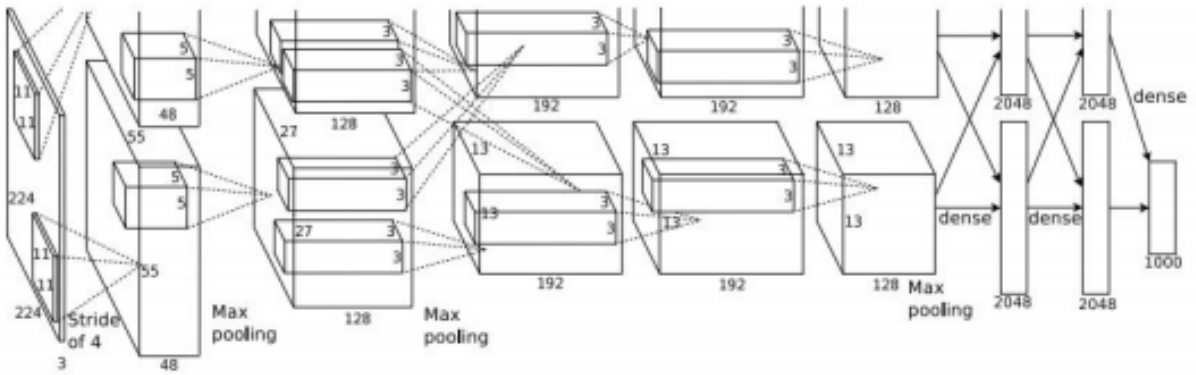
ImageNet Challenge: Classification Task





Case Study: AlexNet

[Krizhevsky et al. 2012]



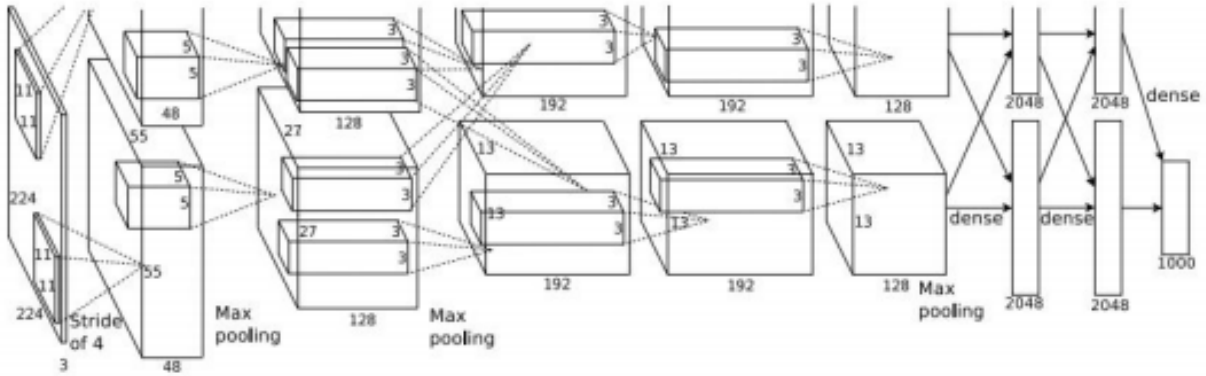
Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (May 2017), 84-90

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

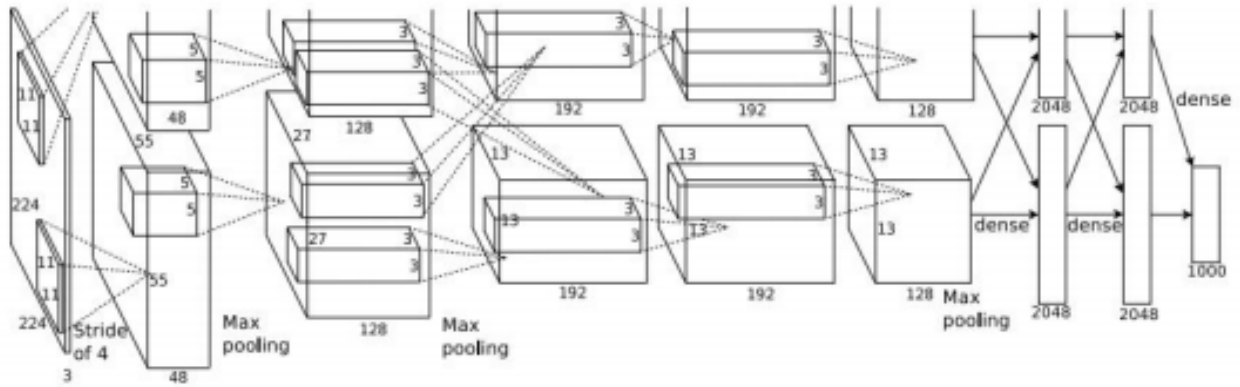
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

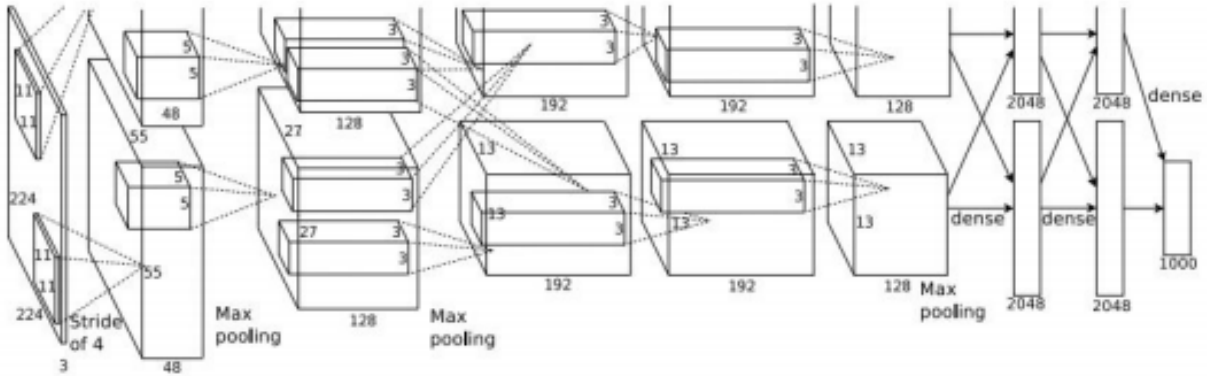
=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

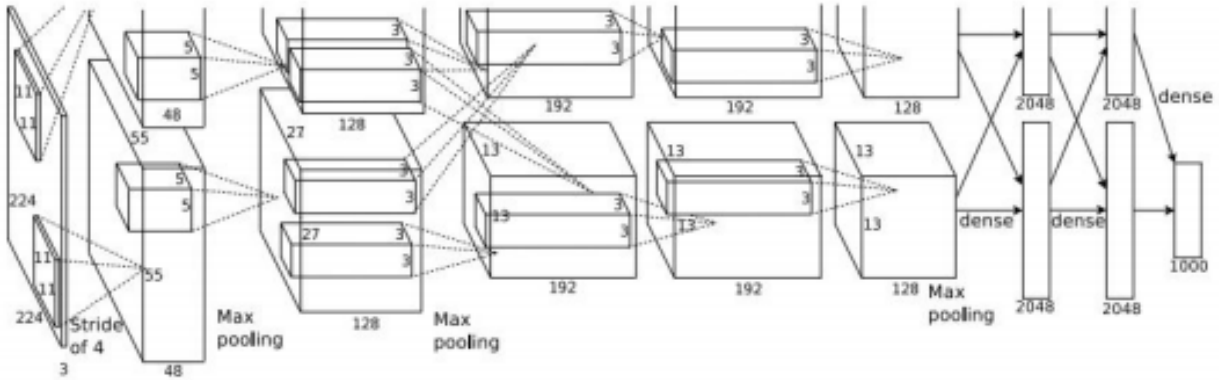


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



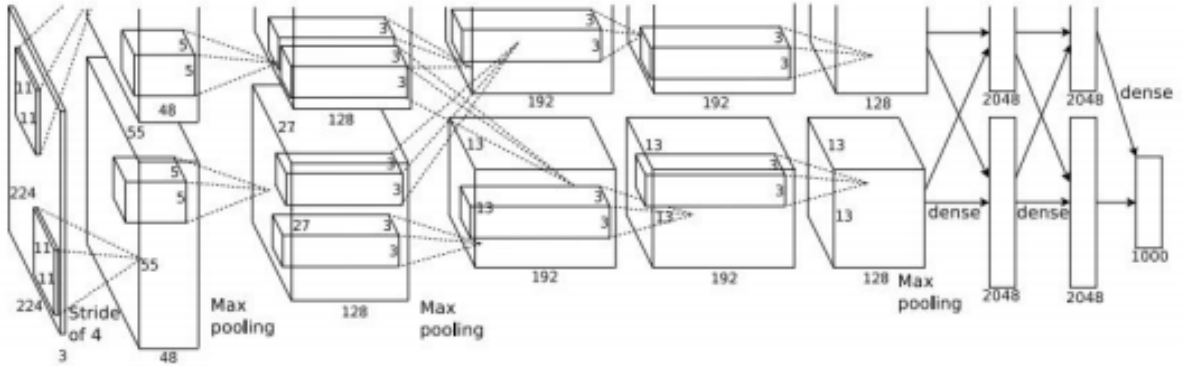
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



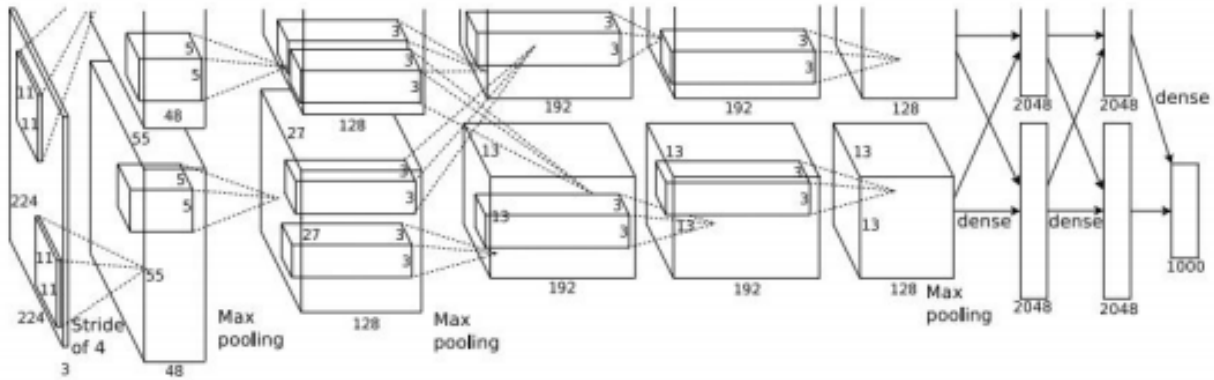
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

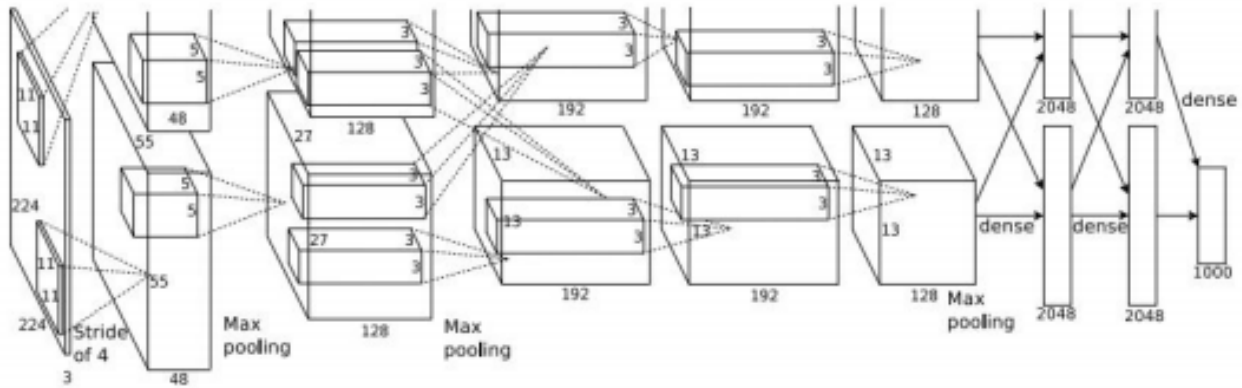


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

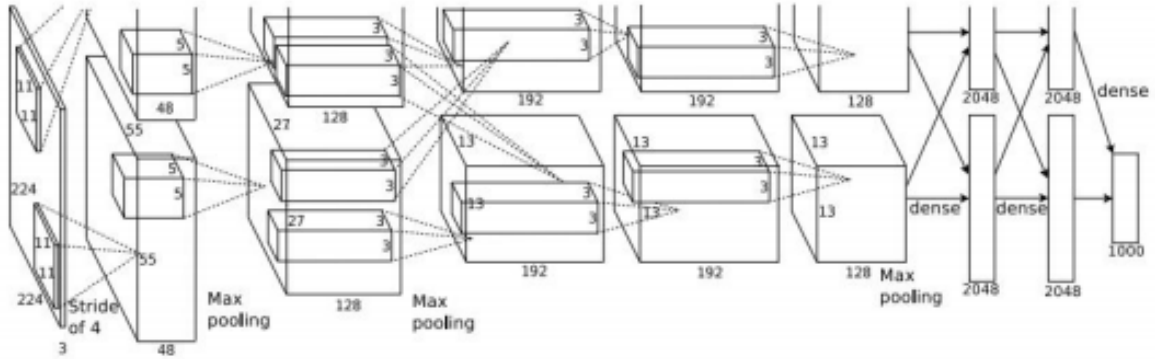
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

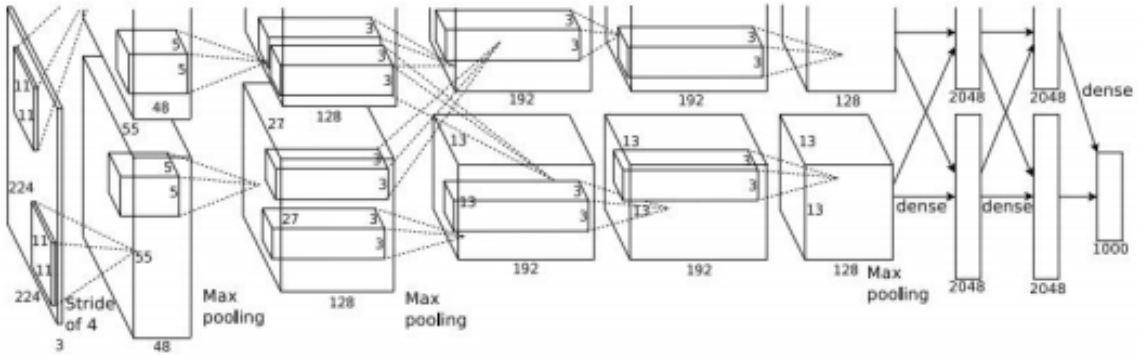
[Krizhevsky et al. 2012]



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)

Case Study: AlexNet

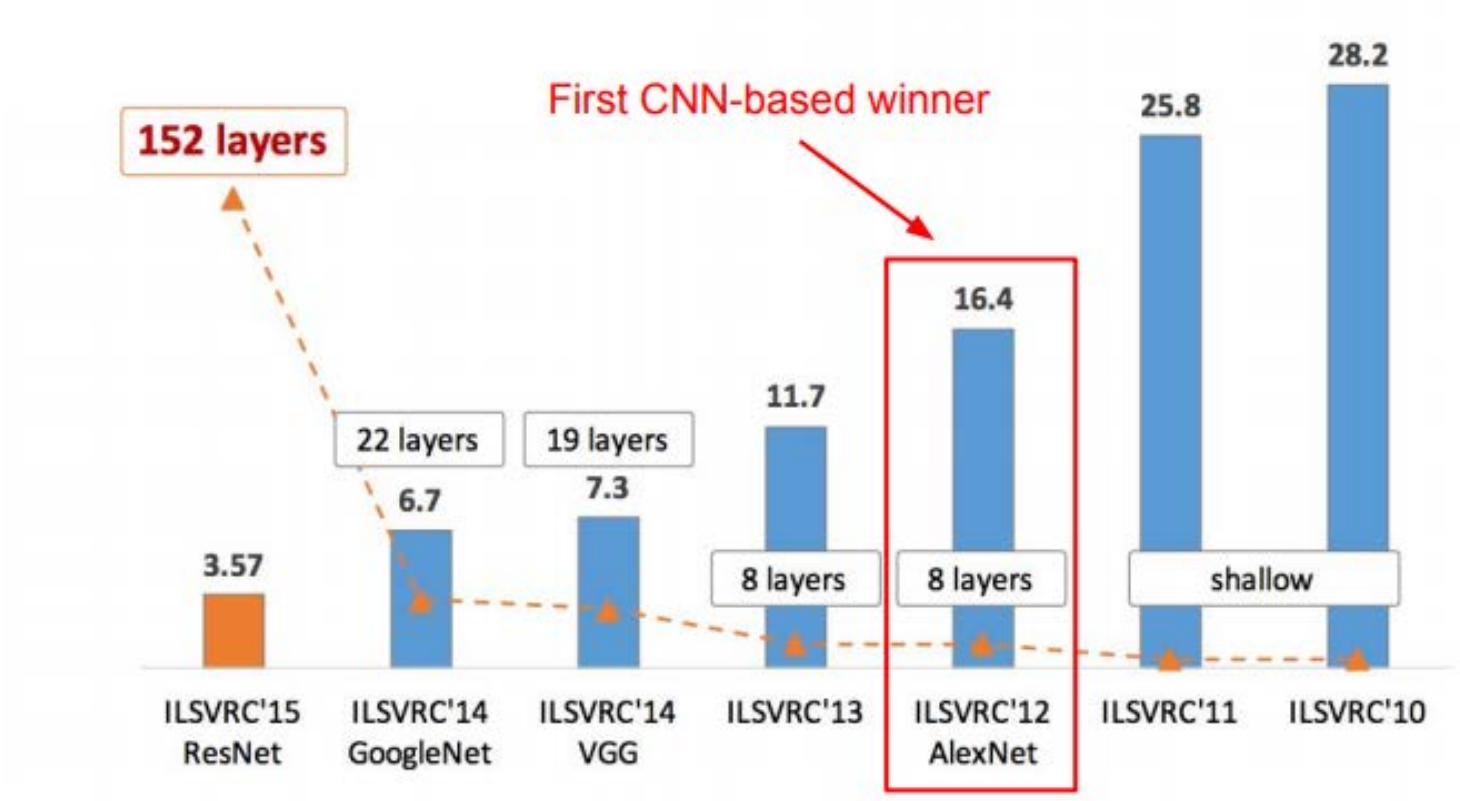
[Krizhevsky et al. 2012]



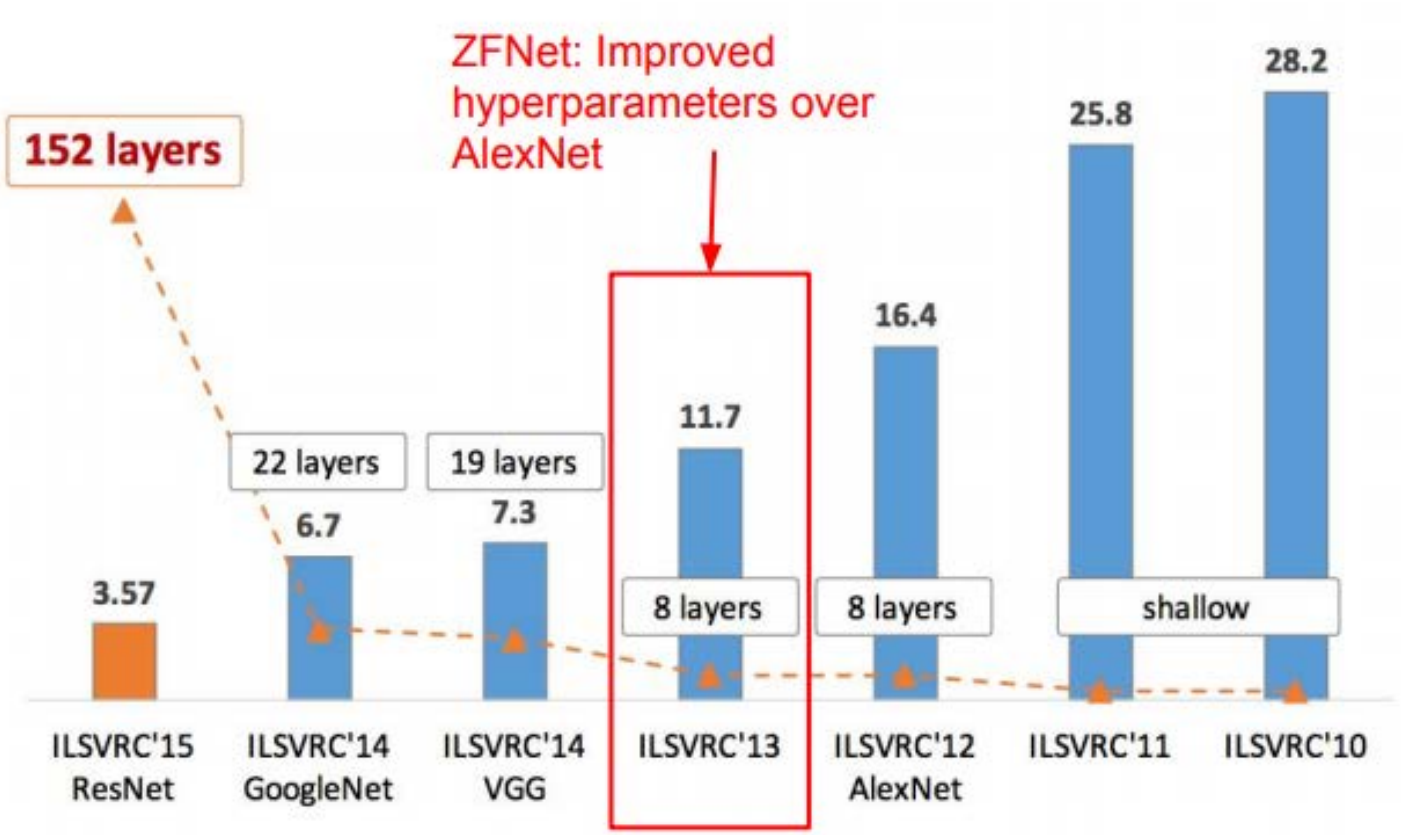
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

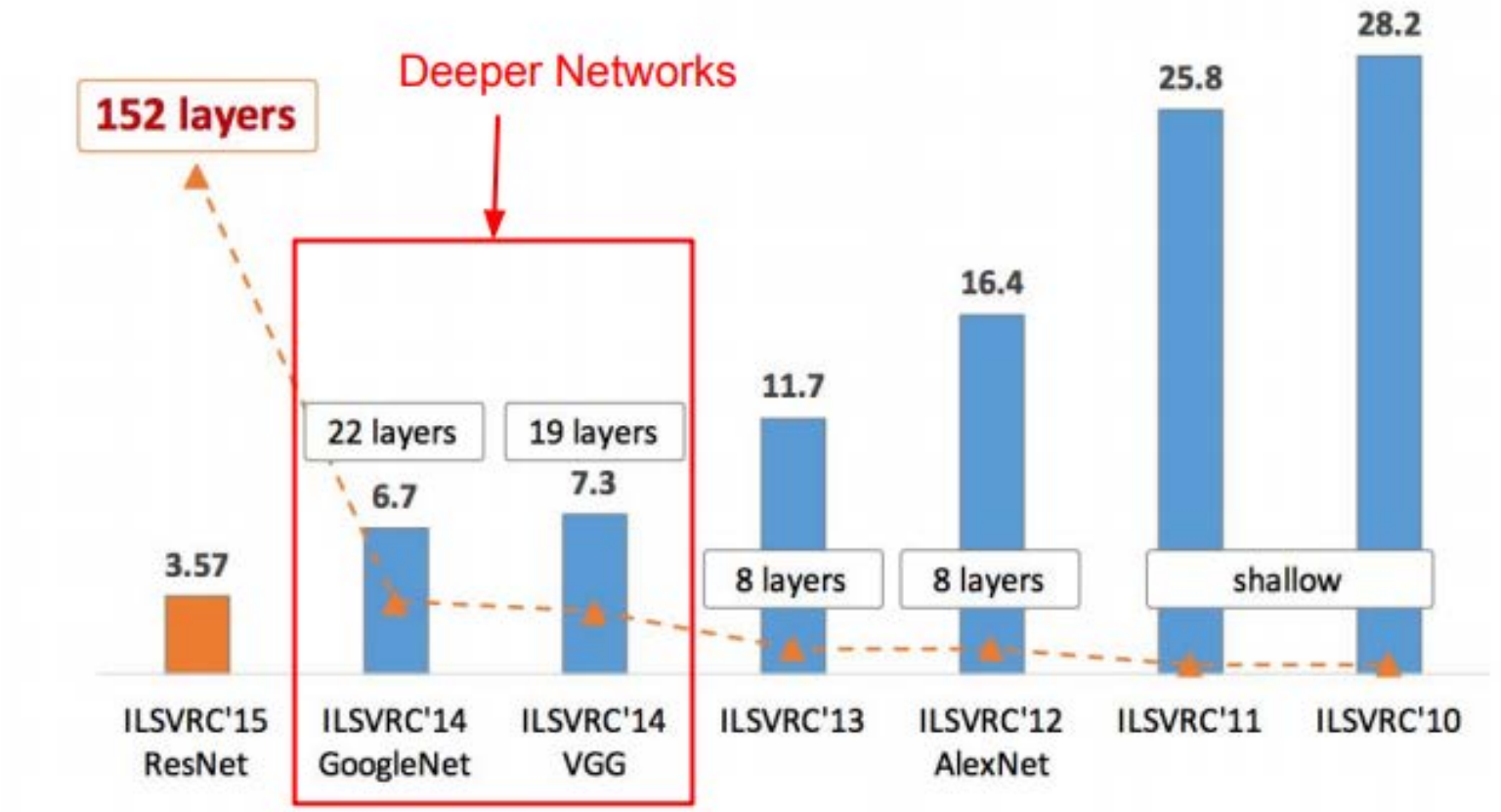


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



M. D. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, European conference on computer vision, 818-833, 2013

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



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

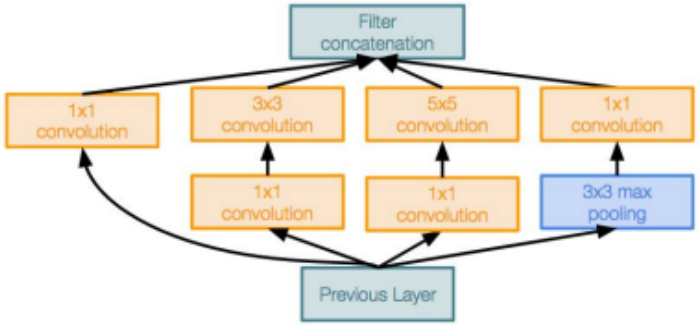
K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, Computer Vision and Pattern Recognition, 2014

Case Study: GoogLeNet

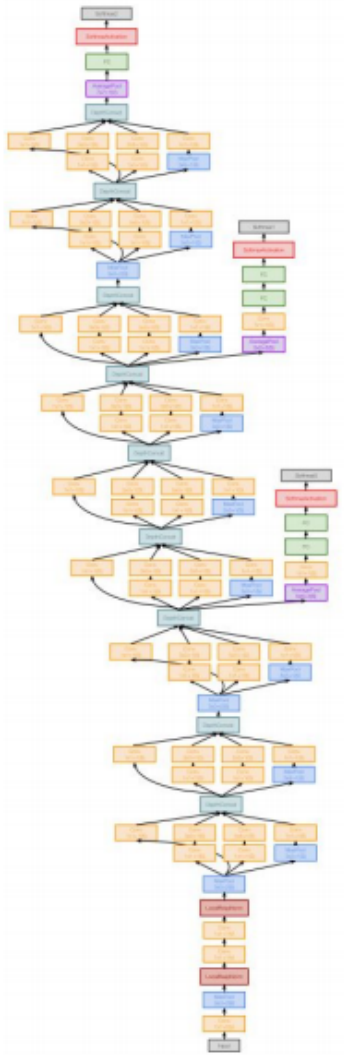
[Szegedy et al., 2014]

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

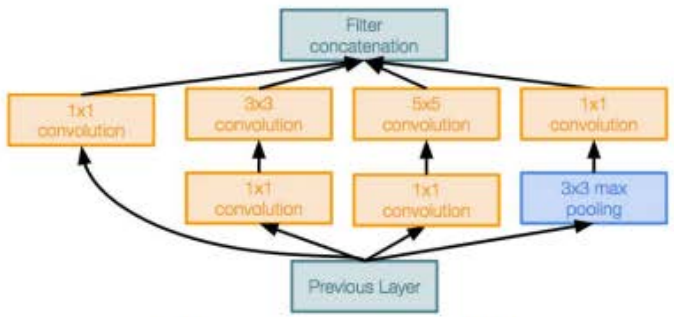


C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelv, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, IEEE Conference on Computer Vision and Pattern Recognition, 2015

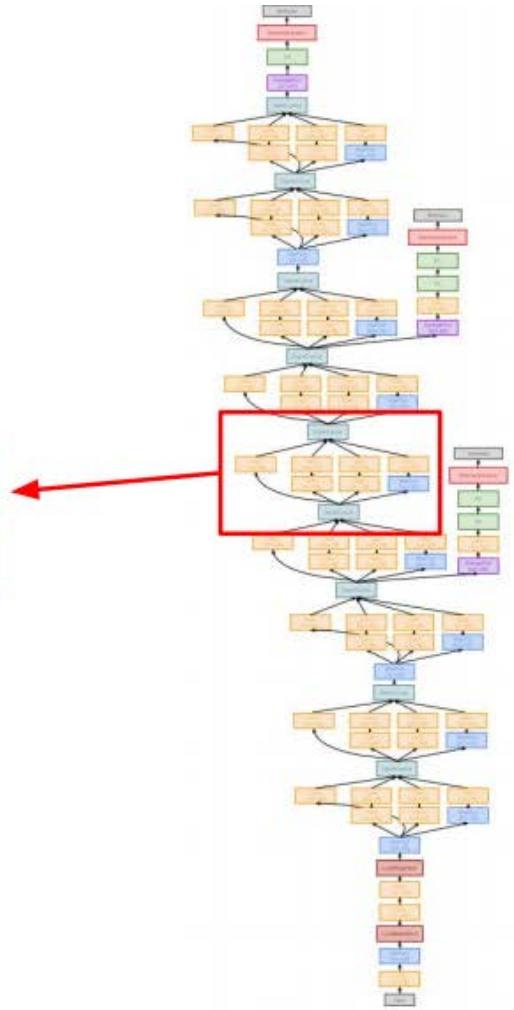
Case Study: GoogLeNet

[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other

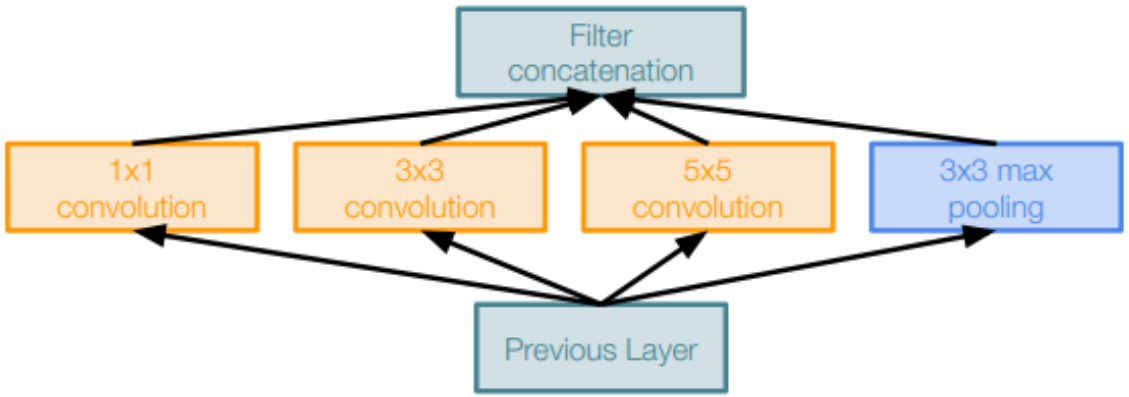


Inception module



Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

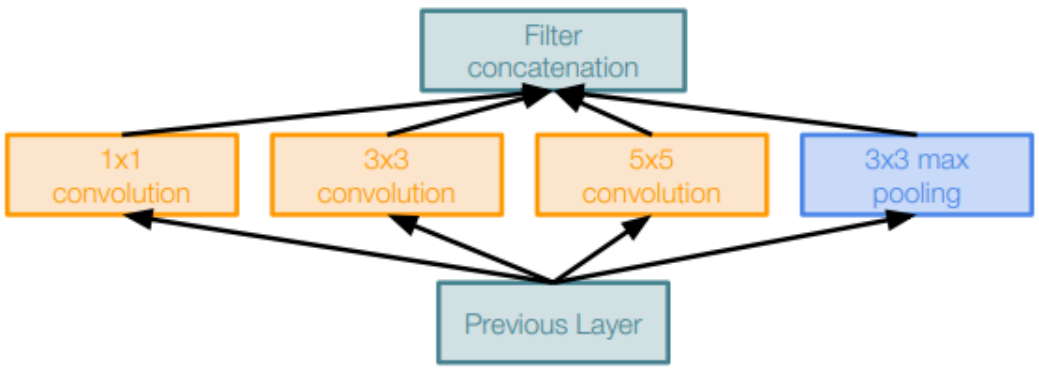
Apply parallel filter operations on the input from previous layer:

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

Concatenate all filter outputs together depth-wise

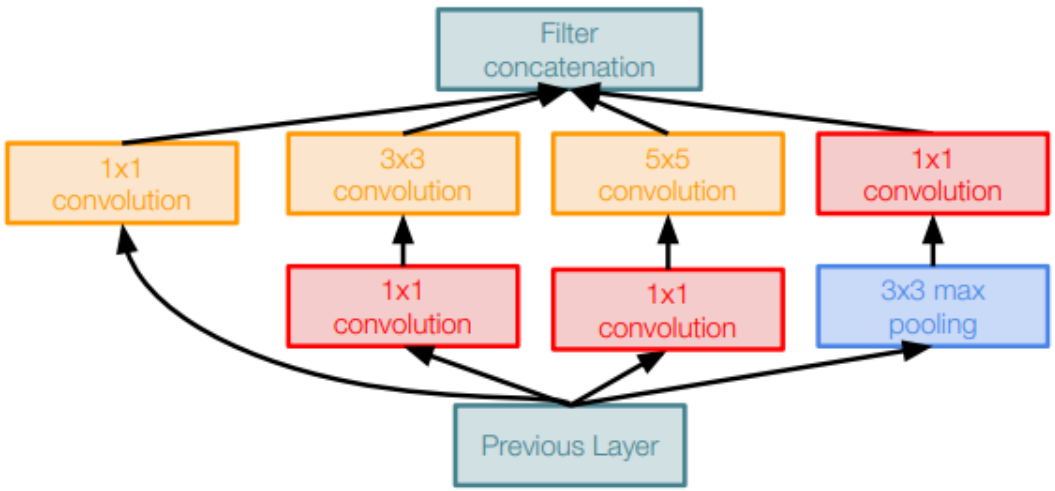
Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

1x1 conv "bottleneck" layers

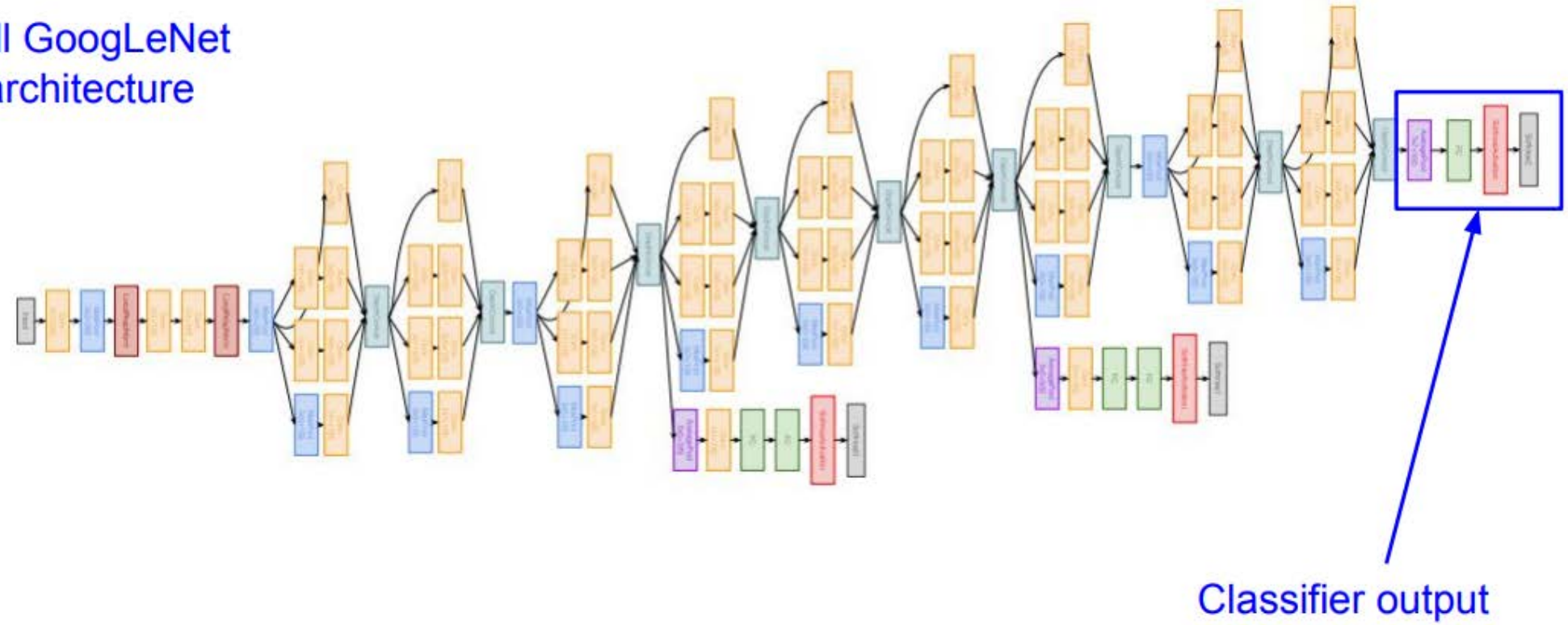


Inception module with dimension reduction

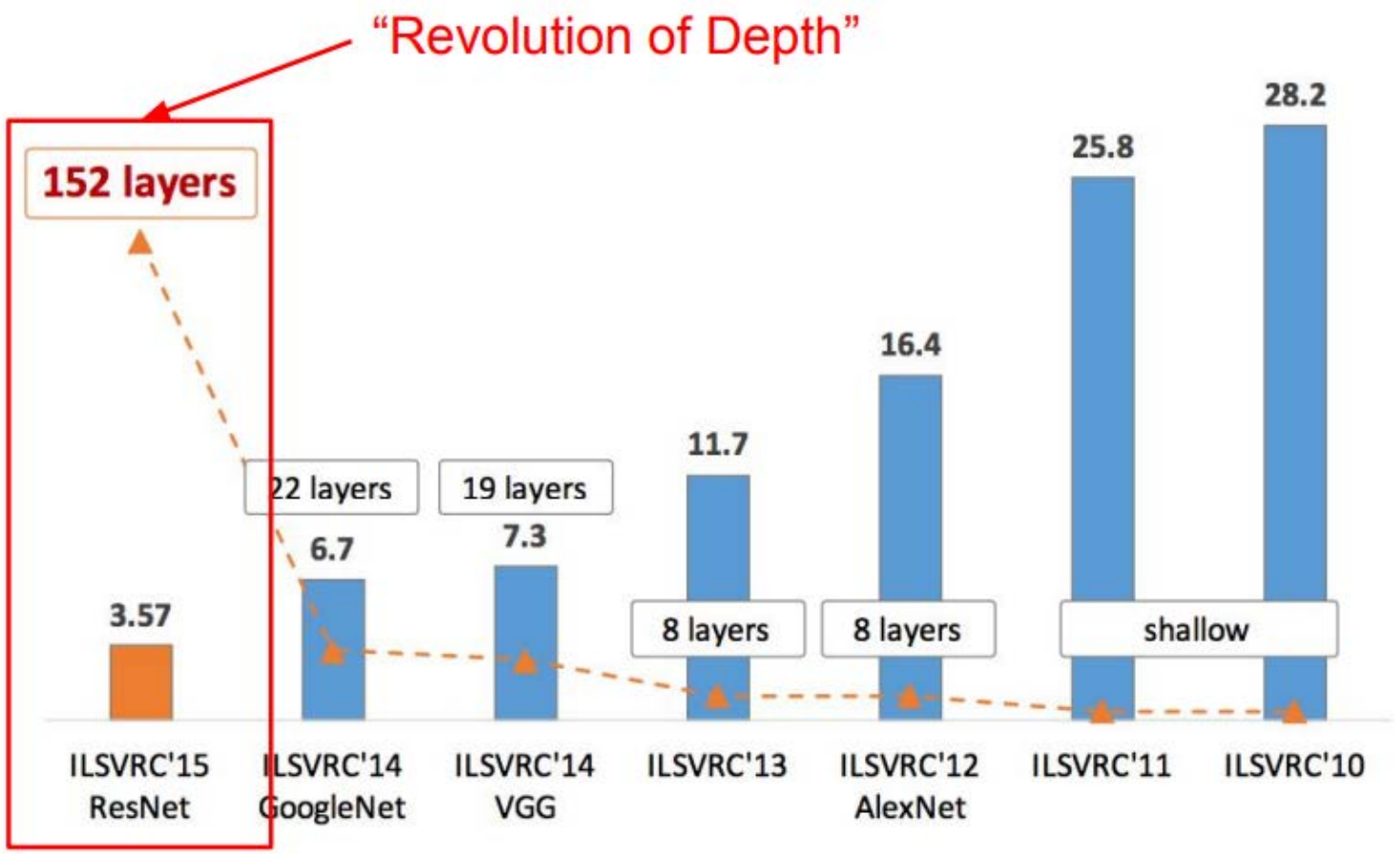
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

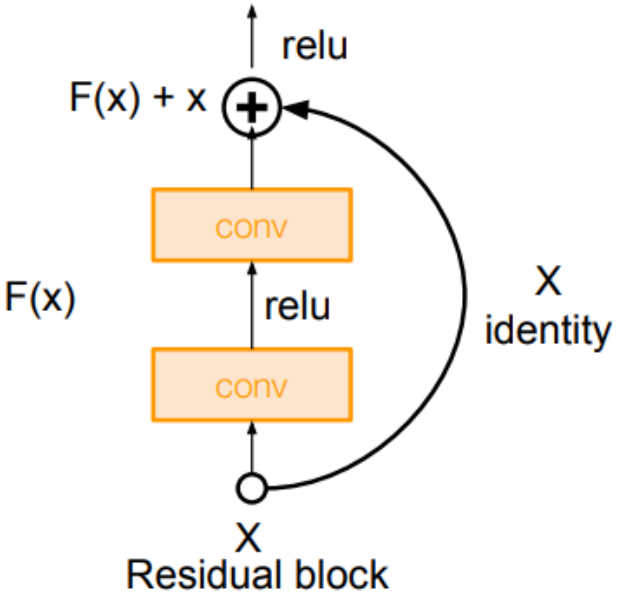


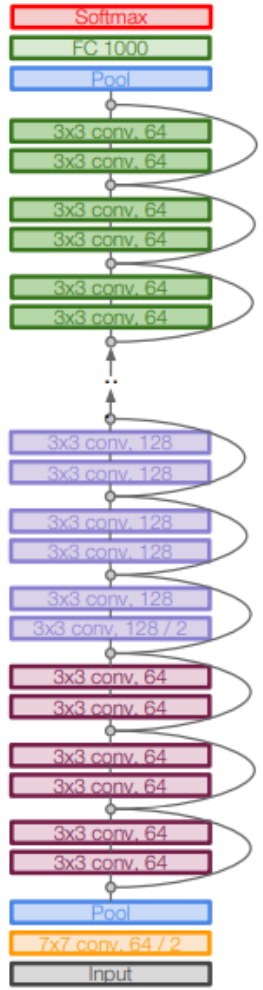
Case Study: ResNet

[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!





K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, IEEE Conference on Computer Vision and Pattern Recognition, 2016

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

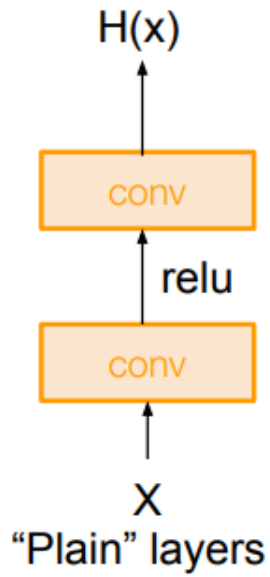
The deeper model should be able to perform at least as well as the shallower model.

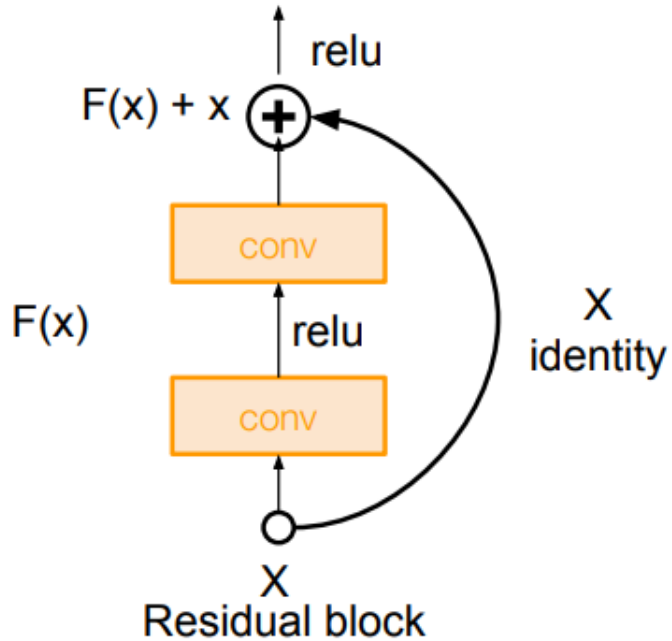
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

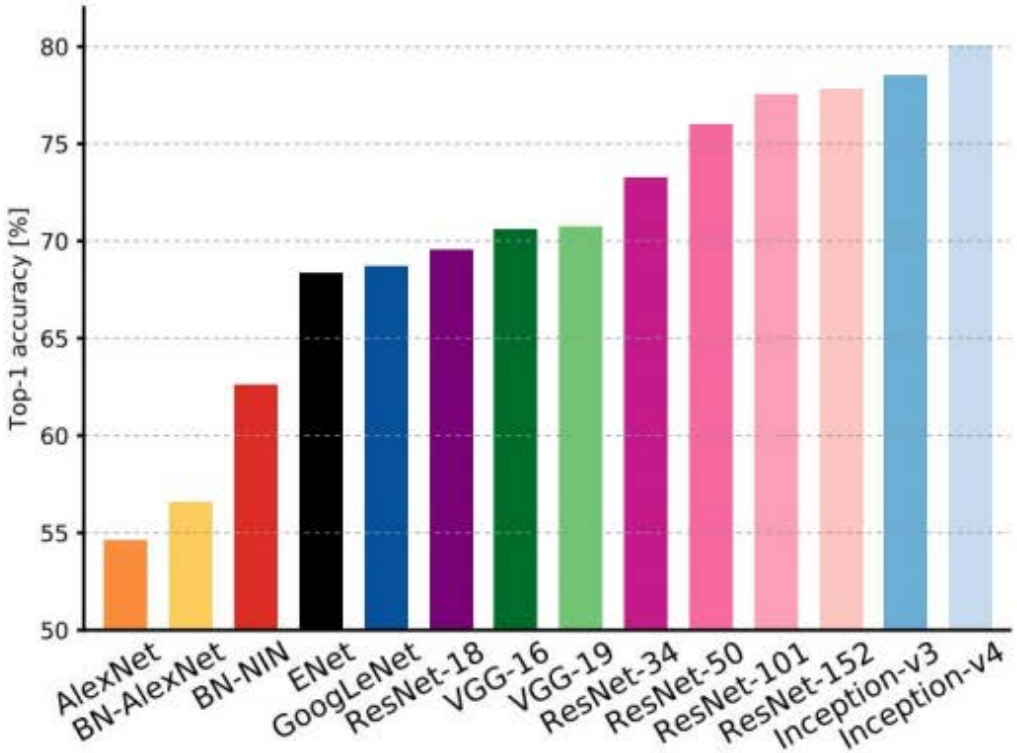
[He et al., 2015]

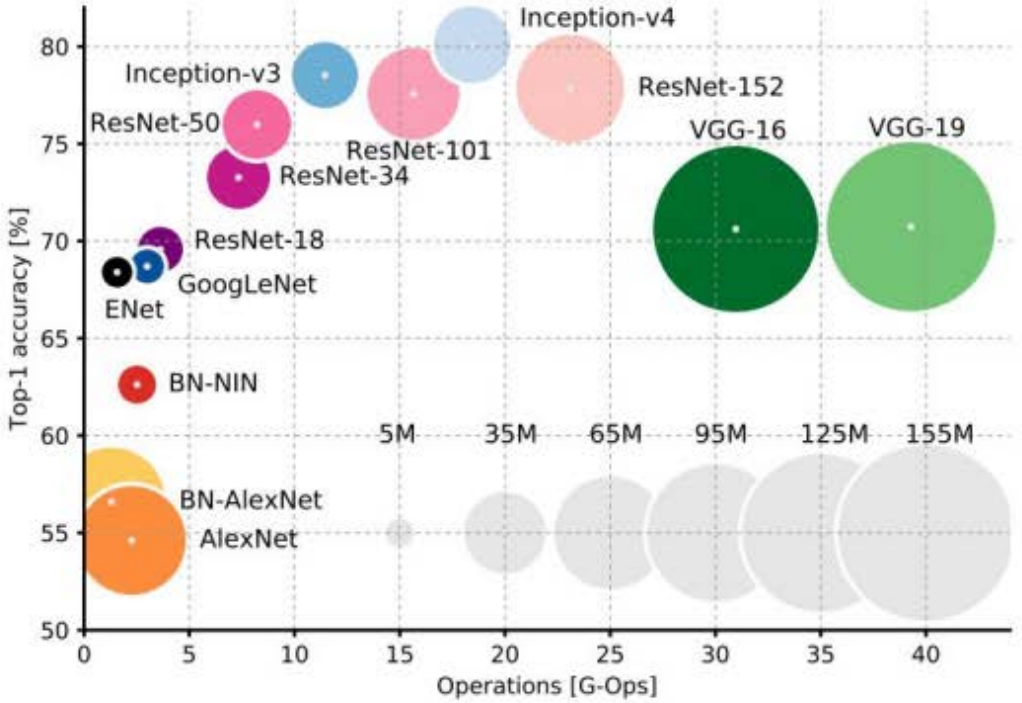
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping





Comparing complexity

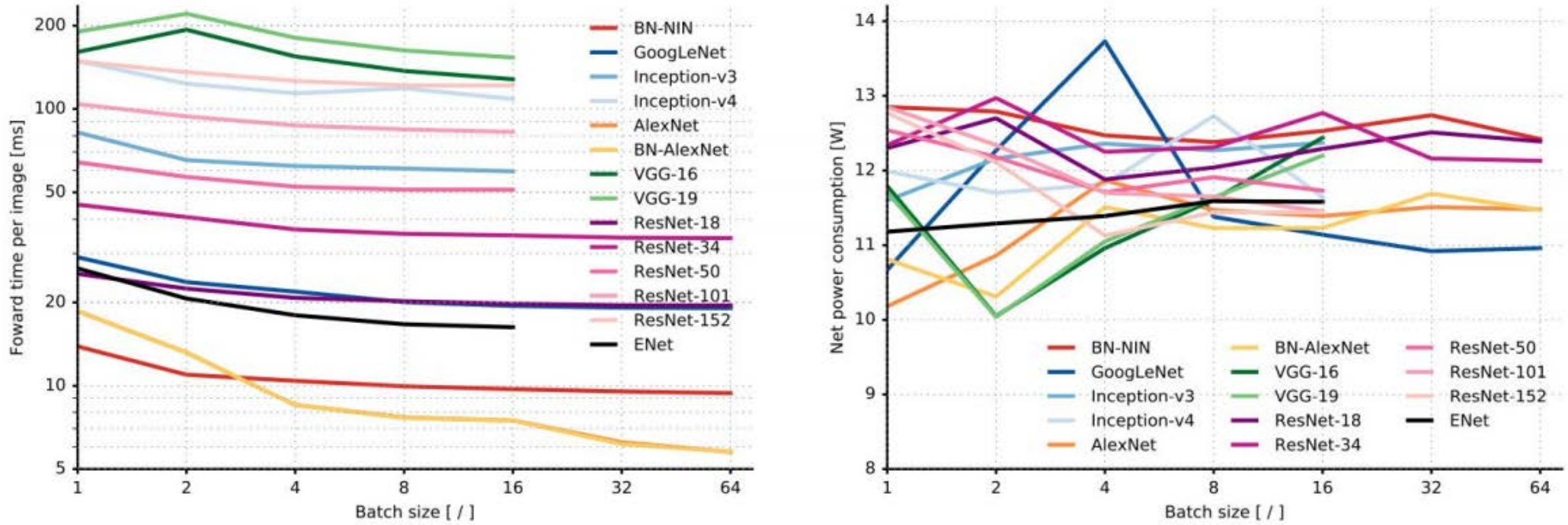




An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity

Forward pass time and power consumption



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Data

Do not use MNIST in your projects! Use it only for practising with Tensorflow.

ImageNet:
22K categories. 14M images.

MNIST: handwritten digits

places: natural scenes

Airplane

Automobile

Bird

Cat

Deer

Dog

Frog

Horse

Ship

Truck

CIFAR-10

So far: Image Classification

This image is CC0 public domain

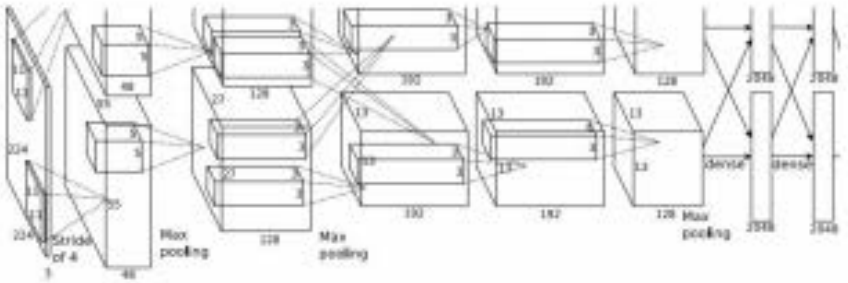


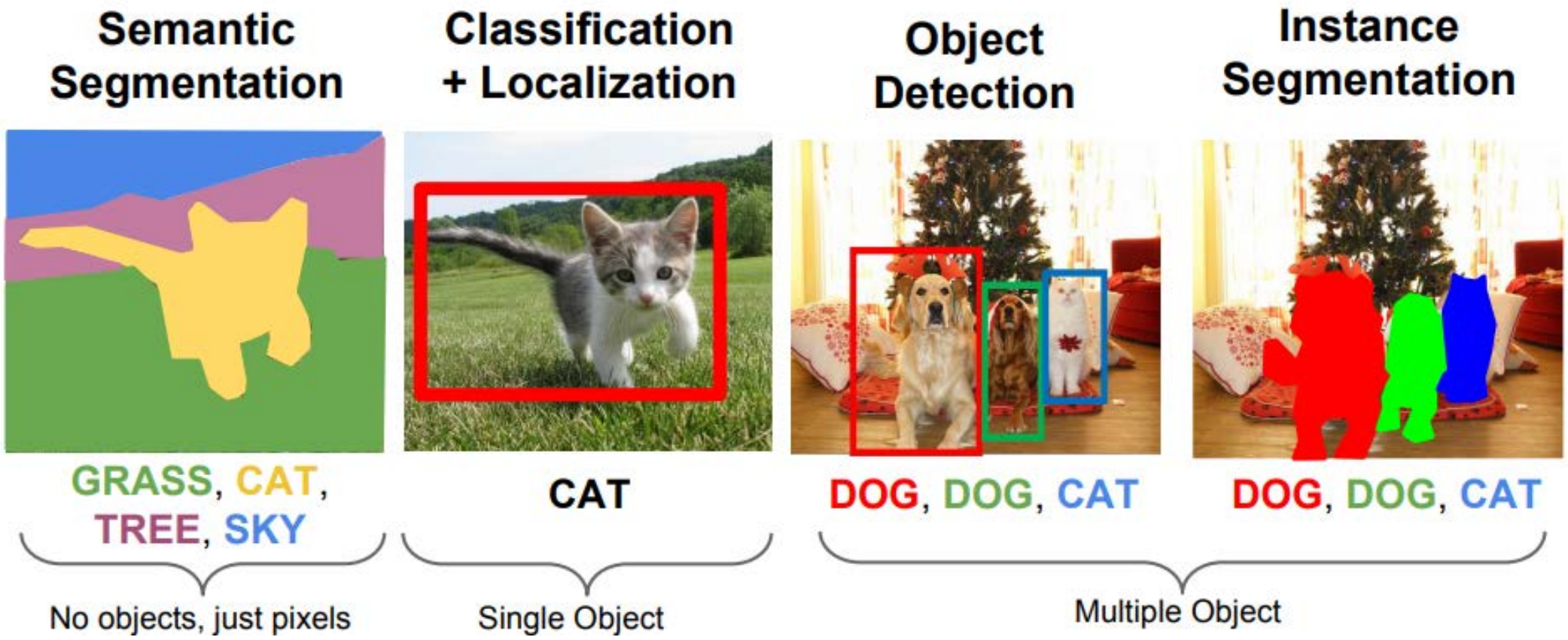
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

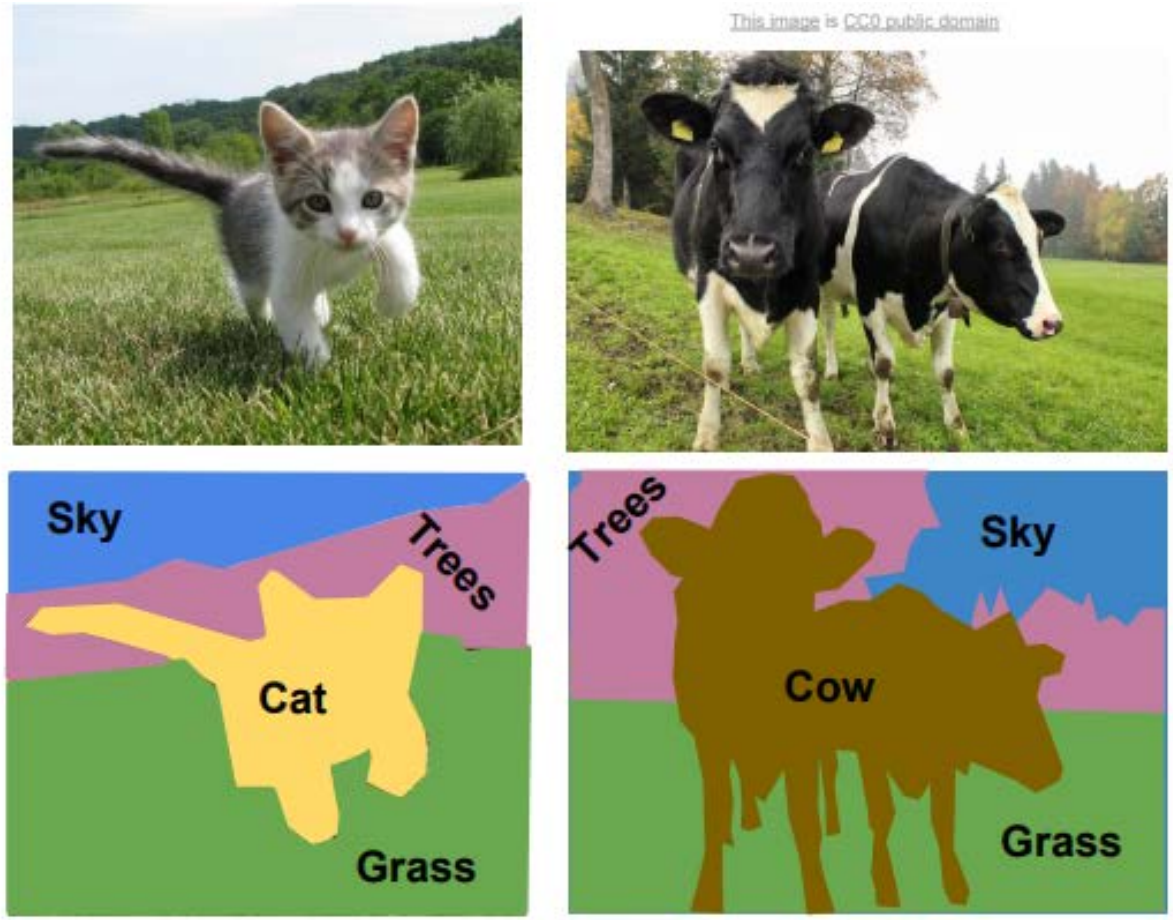
Other Computer Vision Tasks



Semantic Segmentation

Label each pixel in the image with a category label

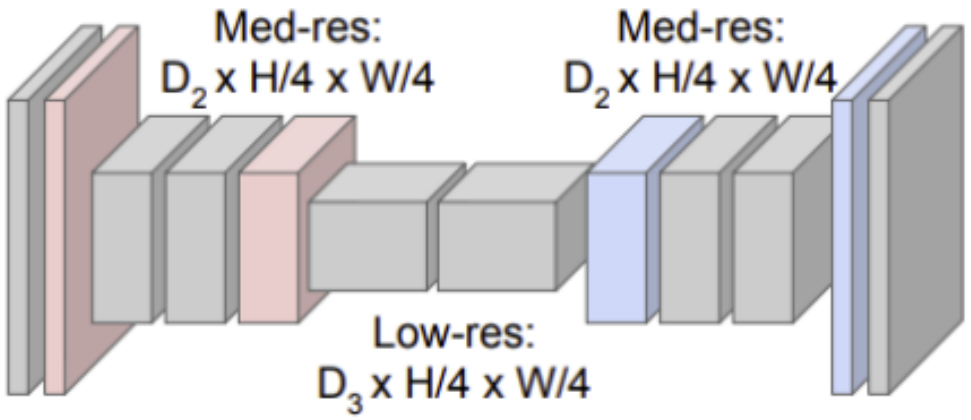
Don't differentiate instances, only care about pixels



Semantic Segmentation: FCNs

FCN: Fully Convolutional Network
Network designed with all convolutional layers,
with **downsampling** and **upsampling** operations

Input:
3 x H x W



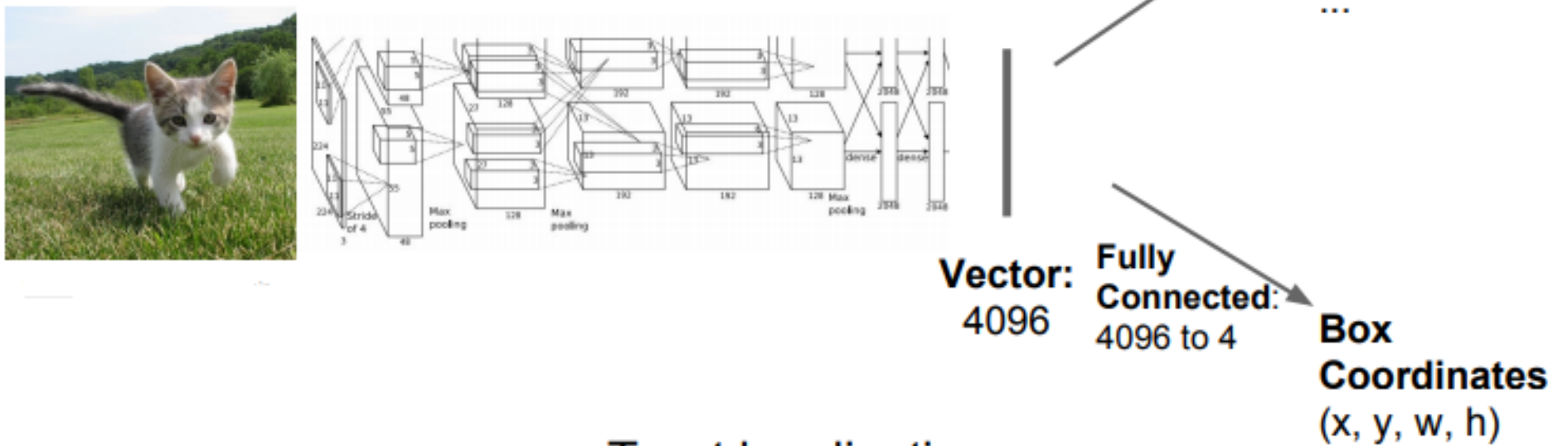
High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

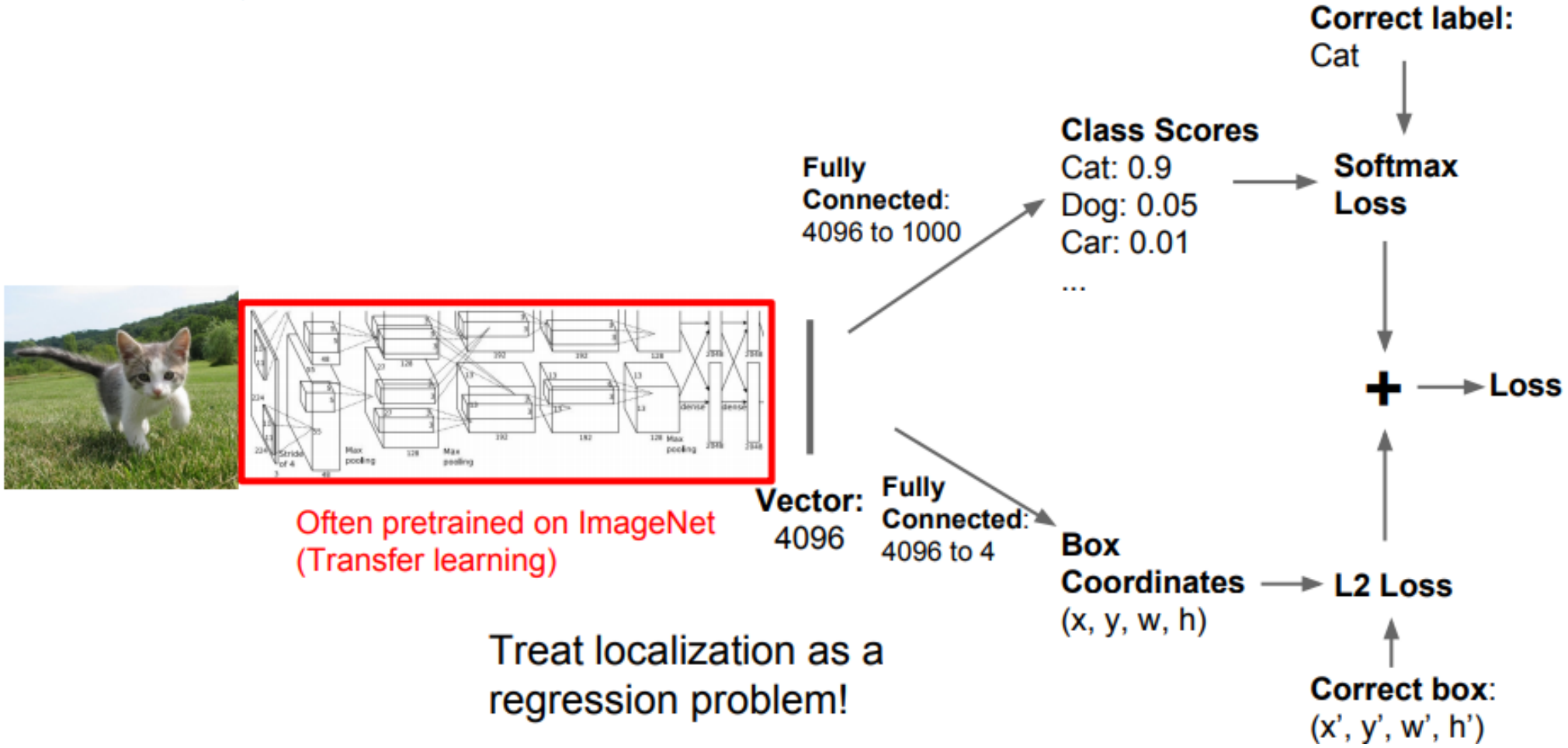
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Classification + Localization

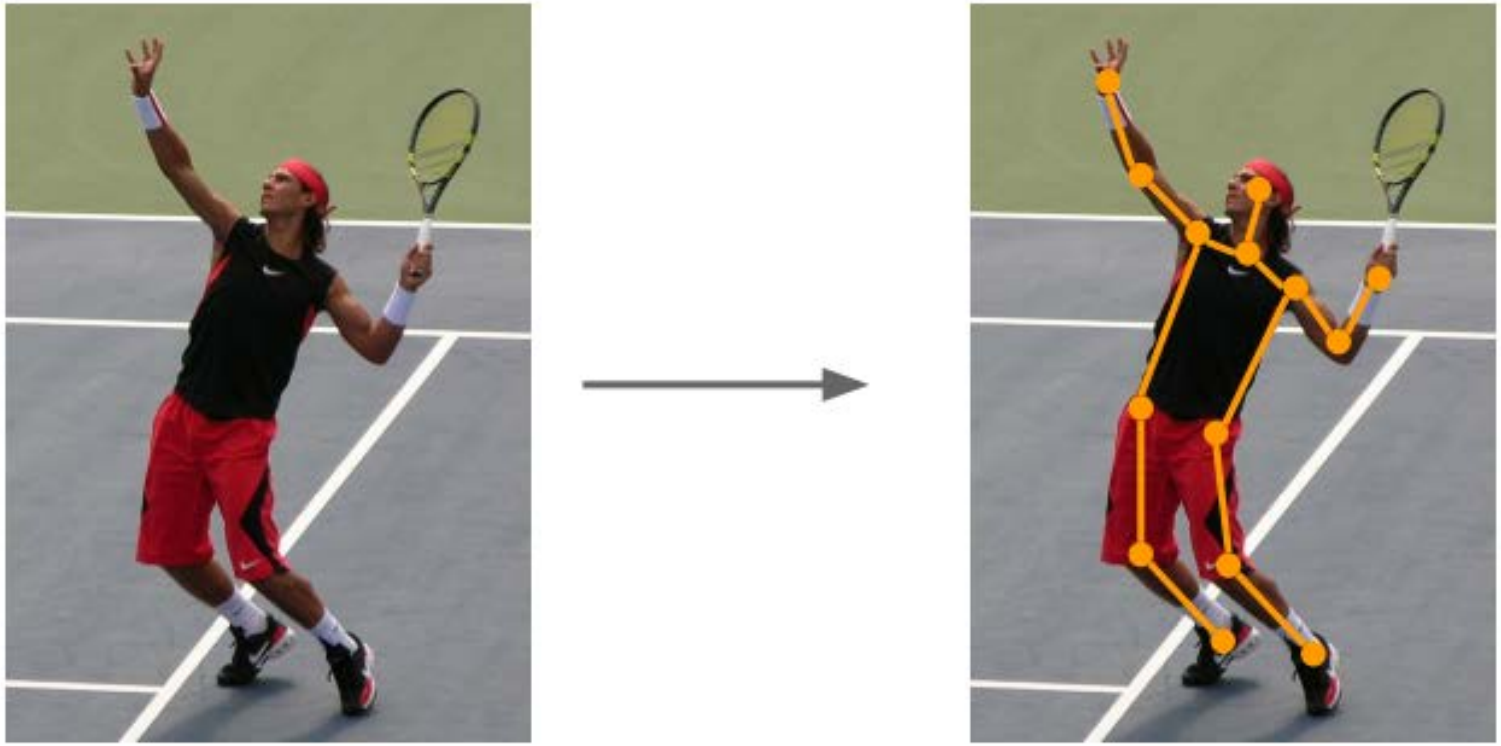


Treat localization as a regression problem!

Classification + Localization



Human Pose Estimation

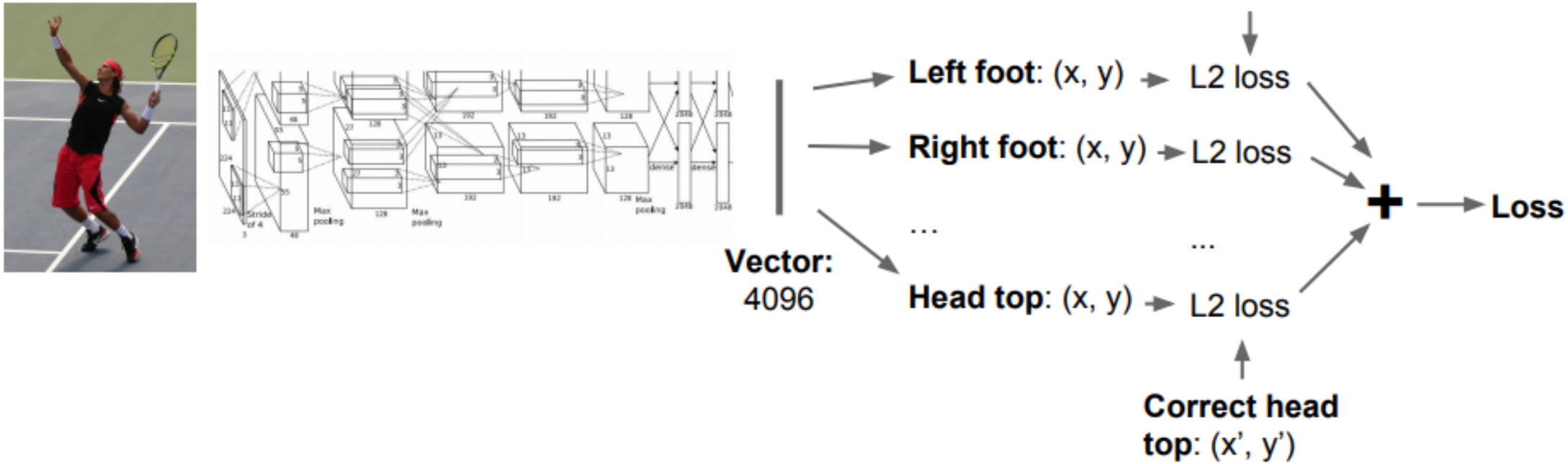


Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

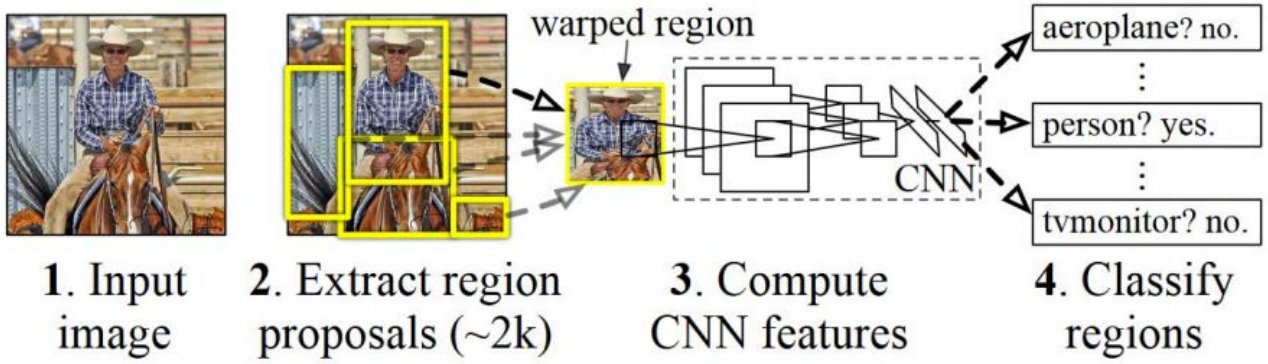
Johnson and Everingham, "Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation", BMVC 2010
Toshev and Szegedy, “DeepPose: Human Pose Estimation via Deep Neural Networks”, CVPR 2014

Human Pose Estimation



Object Detection/Instance Segmentation

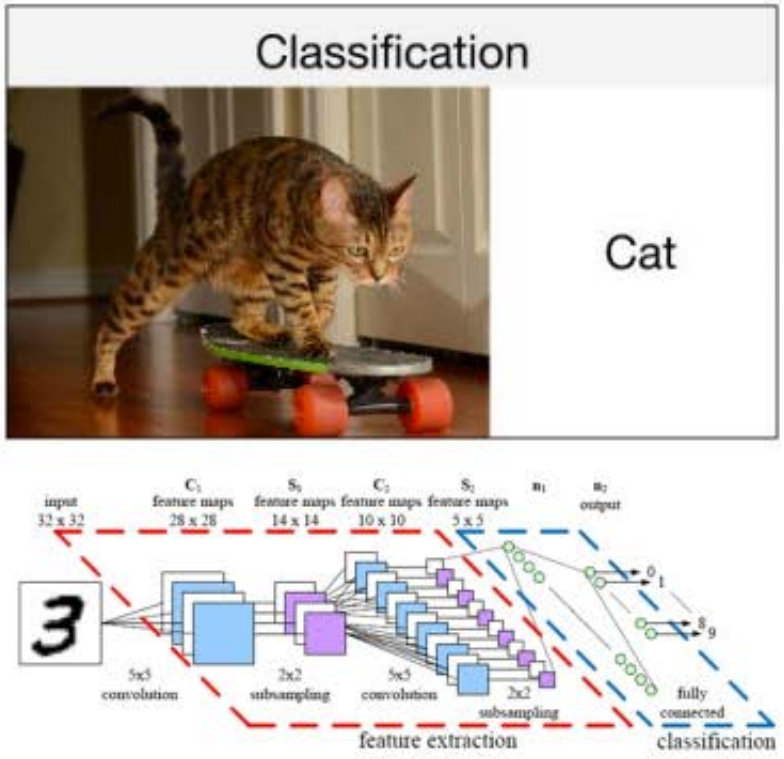
R-CNN: Find regions that we think have objects. Use CNN to classify.



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016

He et al, "Mask R-CNN", arXiv 2017

Image Captioning using RNNs



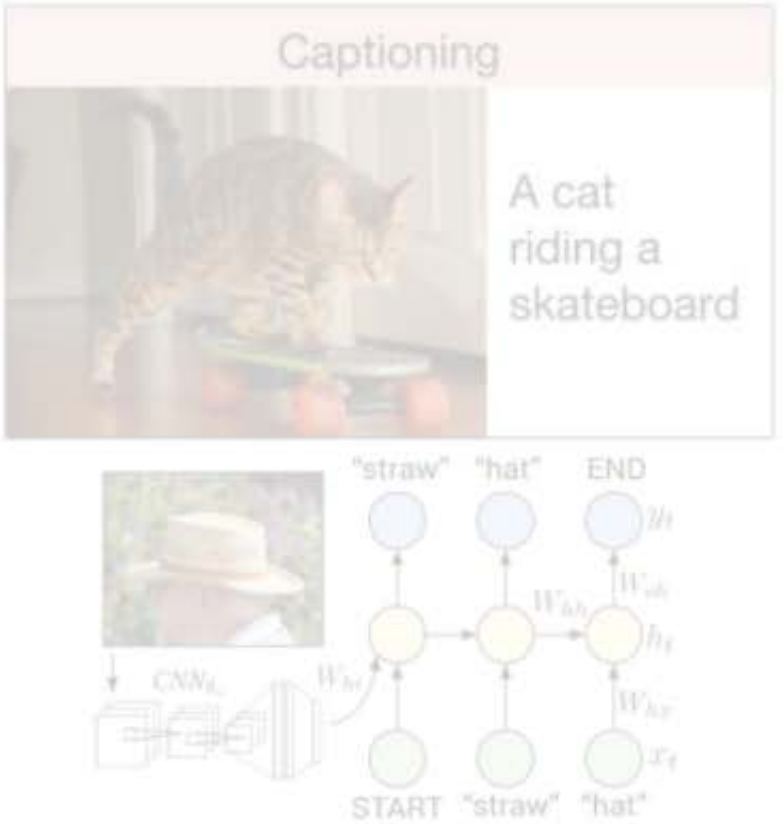
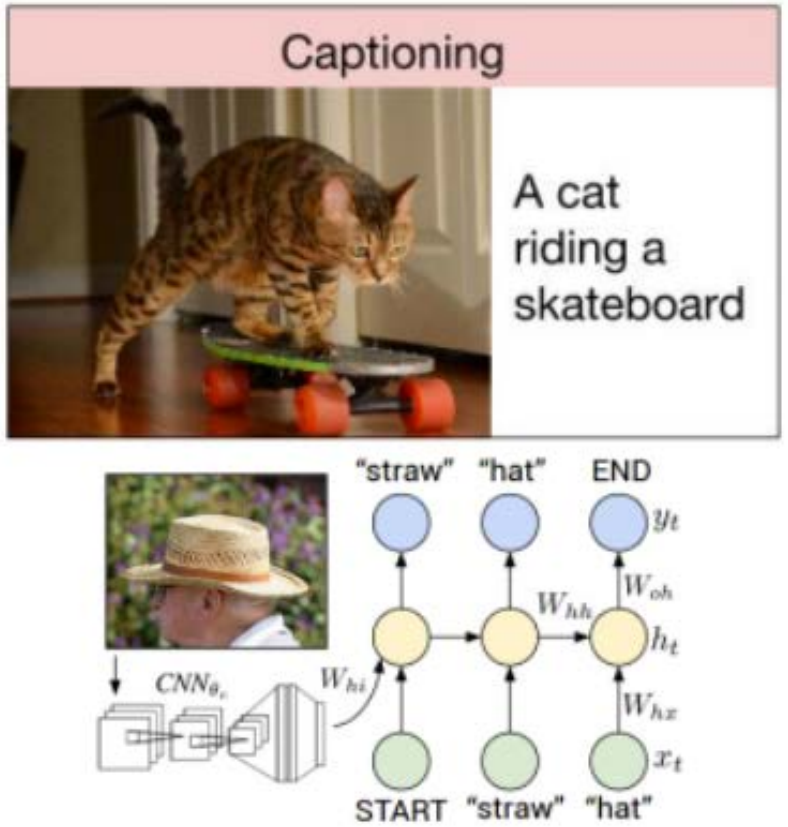
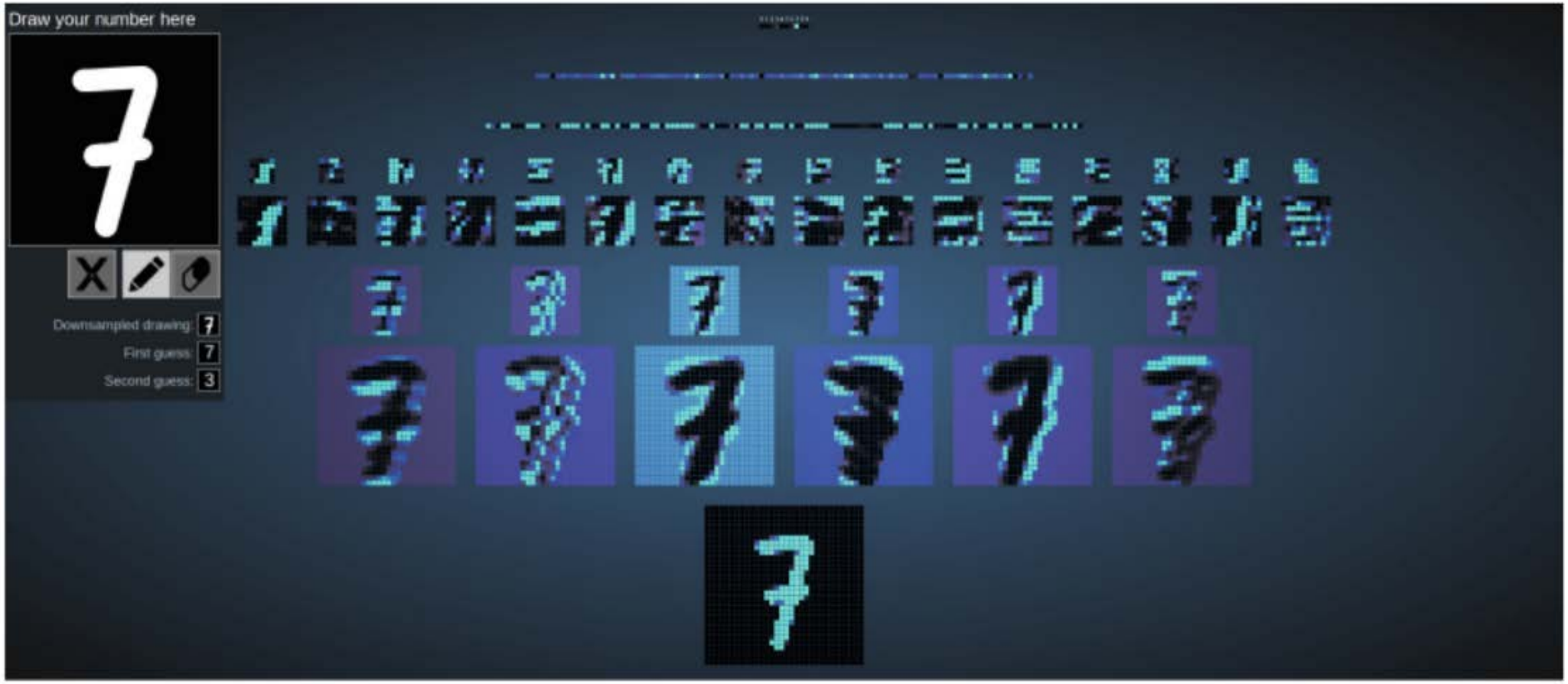


Image Captioning using RNNs



A. Karpathy, L. Fei-Fei, Deep Visual-Semantic Alignments for Generating Image Descriptions, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017
https://cs.stanford.edu/people/karpathy/deepimagesent/

Visualizing Convolutional Networks



http://www.cs.cmu.edu/~aharley/vis/
For more info: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture12.pdf

For your projects..

- Use research papers mentioned in the slides and other papers you found
- Do your research and investigate what is possible and what you like to do
- Choose a problem
  - e.g. object classification, object detection, segmentation, face detection/recognition, pose estimation, image/video captioning (RNNs are not yet explained but you can still use them in your projects if you want)
- Choose a dataset
  - e.g. CIFAR, ImageNet, MSCoco (Do not use MNIST)
- Modify/reimplement different architectures
  - e.g. AlexNet, GoogLeNet, ResNet etc.
- Try different hyperparameters
- More challenging, state-of-the-art problems and in depth analysis result in higher grade!

Supplementary material and references

- Deep Learning book, Chapter 9

- CS231N: Convolutional Neural Networks, Stanford University
- http://cs231n.stanford.edu/2017/syllabus

- 6.S191: Introduction to Deep Learning, MIT
- http://introtodeeplearning.com/2018/#schedule

- Coursera Deeplearning.ai on YouTube: https://www.youtube.com/channel/UCcIXc5mJsHVYTZR1maL5l9w/videos

- (Slides are mainly adopted from the above courses)