References

CS231n: Convolutional Neural Networks for Visual Recognition

Fei-Fei Li & Andrej Karpathy & Justin Johnson

Fast R-CNN

Ross Girshick
Microsoft Research
rbg@microsoft.com

Rich feature hierarchies for accurate object detection and semantic segmentation

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

Selective Search for Object Recognition

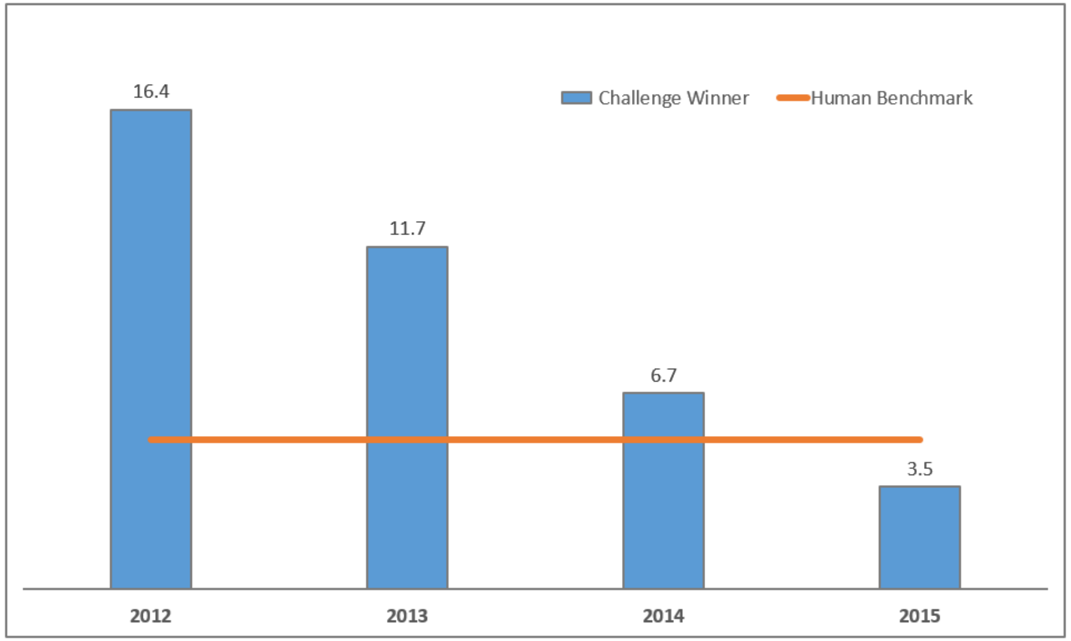
J.R.R. Uijlings[*1,2], K.E.A. van de Sande[†2], T. Gevers[2], and A.W.M. Smeulders[2]
[1]University of Trento, Italy
[2]University of Amsterdam, the Netherlands

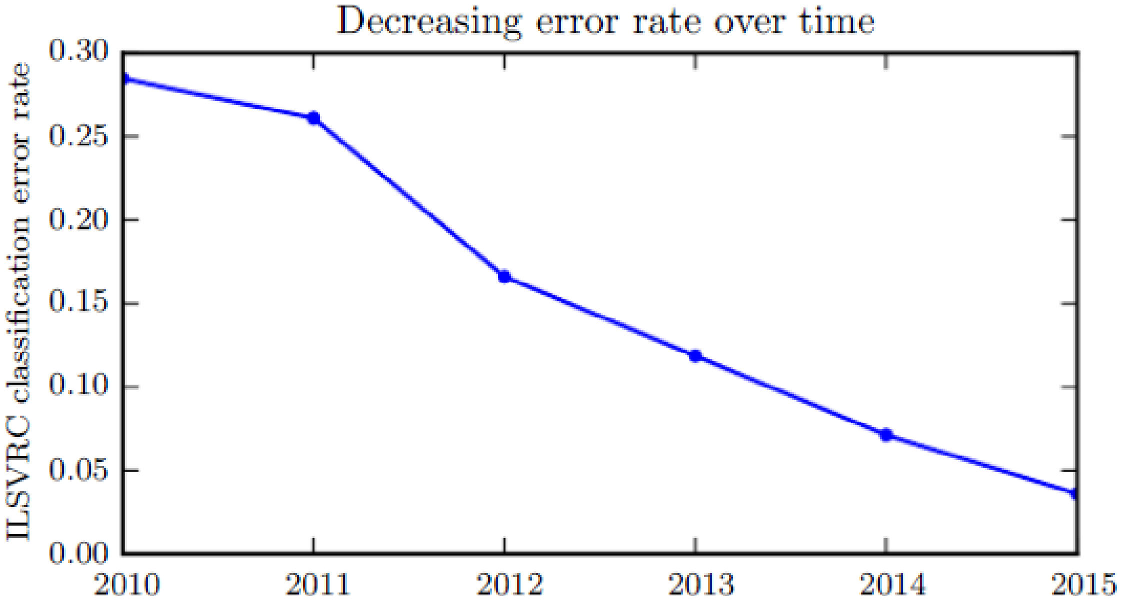
Technical Report 2012, submitted to IJCV

Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley
{jonlong,shelhamer,trevor}@cs.berkeley.edu

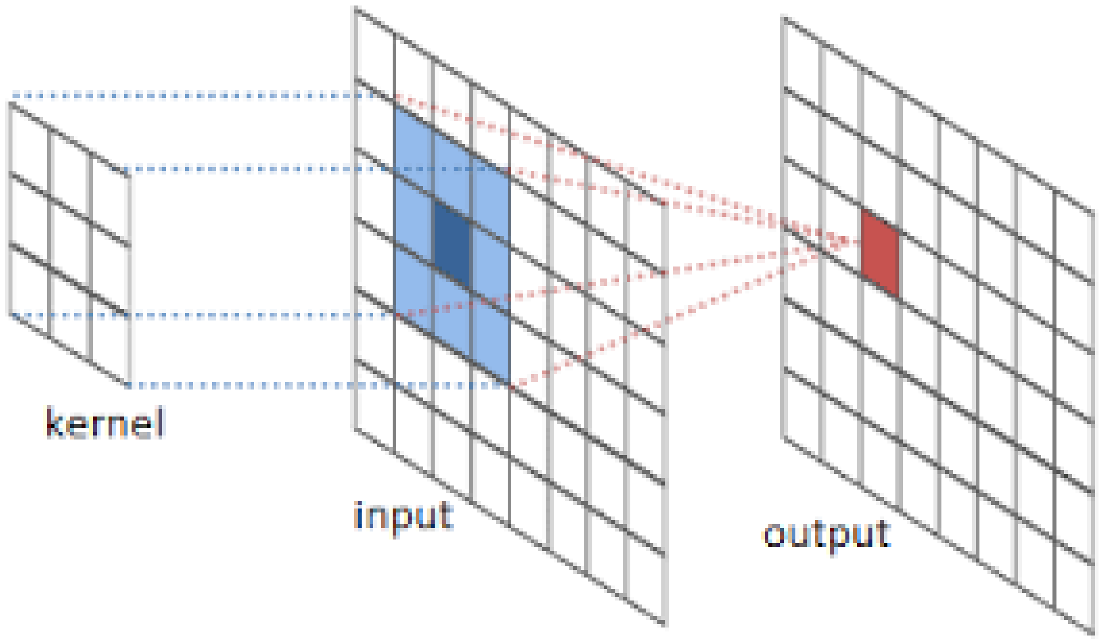
"A dramatic moment in the meteoric rise of deep learning came when a convolutional network won this challenge for the first time and by a wide margin, bringing down the state-of-the-art top-5 error rate from 26.1% to 15.3% (Krizhevsky *et al.*, 2012), meaning that the convolutional network produces a ranked list of possible categories for each image and the correct category appeared in the first five entries of this list for all but 15.3% of the test examples. **Since then, these competitions are consistently won by deep convolutional nets**, and as of this writing, advances in deep learning have brought the latest top-5 error rate in this contest down to 3.6%" – Ref: Deep Learning Book by Y Bengio et al





What is a convolutional neural network?

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



- Convolution is a mathematical operation having a linear form

Types of inputs

- Inputs have a structure
  - Color images are three dimensional and so have a volume
  - Time domain speech signals are 1-d while the frequency domain representations (e.g. MFCC vectors) take a 2d form. They can also be looked at as a time sequence.
  - Medical images (such as CT/MR/etc) are multidimensional
  - Videos have the additional temporal dimension compared to stationary images
  - Speech signals can be modelled as 2 dimensional
  - Variable length sequences and time series data are again multidimensional

- Hence it makes sense to **model them as tensors** instead of vectors.

- The classifier then needs to accept a tensor as input and perform the necessary machine learning task. In the case of an image, this tensor represents a volume.

CNNs are everywhere

- Image retrieval
- Detection
- Self driving cars
- Semantic segmentation
- Face recognition (FB tagging)
- Pose estimation
- Detect diseases
- Speech Recognition
- Text processing
- Analysing satellite data

CNNs for applications that involve images

- Why CNNs are more suitable to process images?

- Pixels in an image correlate to each other. However, nearby pixels correlate stronger and distant pixels don’t influence much
    - Local features are important: Local Receptive Fields

- Affine transformations: The class of an image doesn’t change with translation. We can build a feature detector that can look for a particular feature (e.g. an edge) anywhere in the image plane by moving across. A convolutional layer may have several such filters constituting the depth dimension of the layer.

Fully connected layers

- Fully connected layers (such as the hidden layers of a traditional neural network) are agnostic to the structure of the input
  - They take inputs as vectors and generate an output vector
  - There is no requirement to share parameters unless forced upon in specific architectures. This blows up the number of parameters as the input and/or output dimensions increase.
- Suppose we are to perform classification on an image of 100x100x3 dimensions.
- If we implement using a feed forward neural network that has an input, hidden and an output layer, where: hidden units (nh) = 1000, output classes = 10 :
  - Input layer = 10k pixels * 3 = 30k, weight matrix for hidden to input layer = 1k * 30k = 30 M and output layer matrix size = 10 * 1000 = 10k
- We may handle this is by extracting the features using pre processing and presenting a lower dimensional input to the Neural Network. But this requires expert engineered features and hence domain knowledge

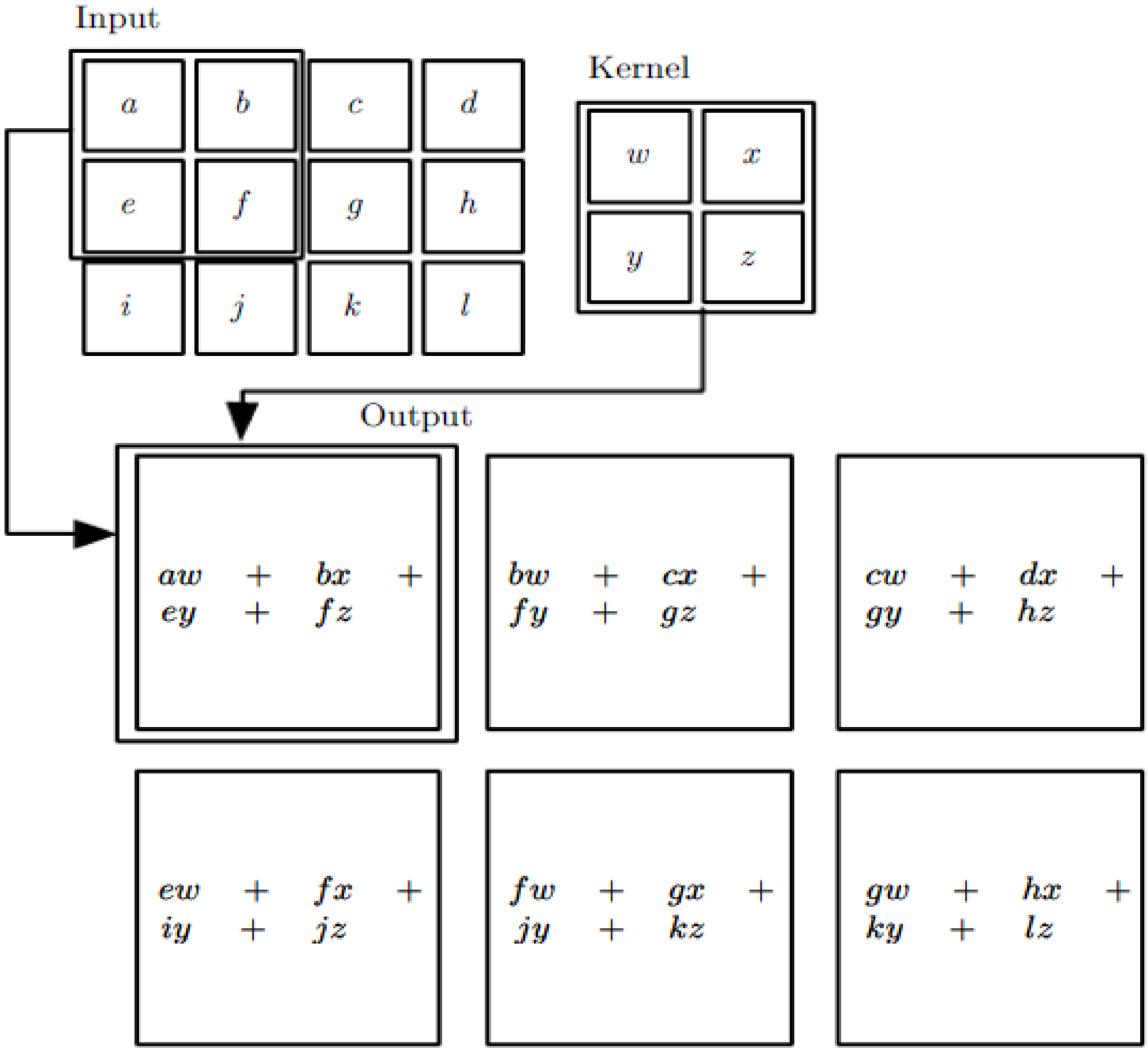
Convolution

$Convolution\ in\ 1\ Dimension$:

$$y[n] = \sum_{k=-\infty}^{k=\infty} x[k]h[n-k]$$

$Convolution\ in\ 2\ Dimensions$:

$$y[n_1, n_2] = \sum_{k1=-\infty}^{k1=-\infty} \sum_{k2=-\infty}^{k2=\infty} x[k_1, k_2]h[(n_1 - k_1), (n_2 - k_2)]$$


Input
a
b
c
d
e
f
g
h
i
j
k
l
Kernel
w
x
y
z
Output
aw + bx + ey + fz
bw + cx + fy + gz
cw + dx + gy + hz
ew + fx + iy + jz
fw + gx + jy + kz
gw + hx + ky + lz

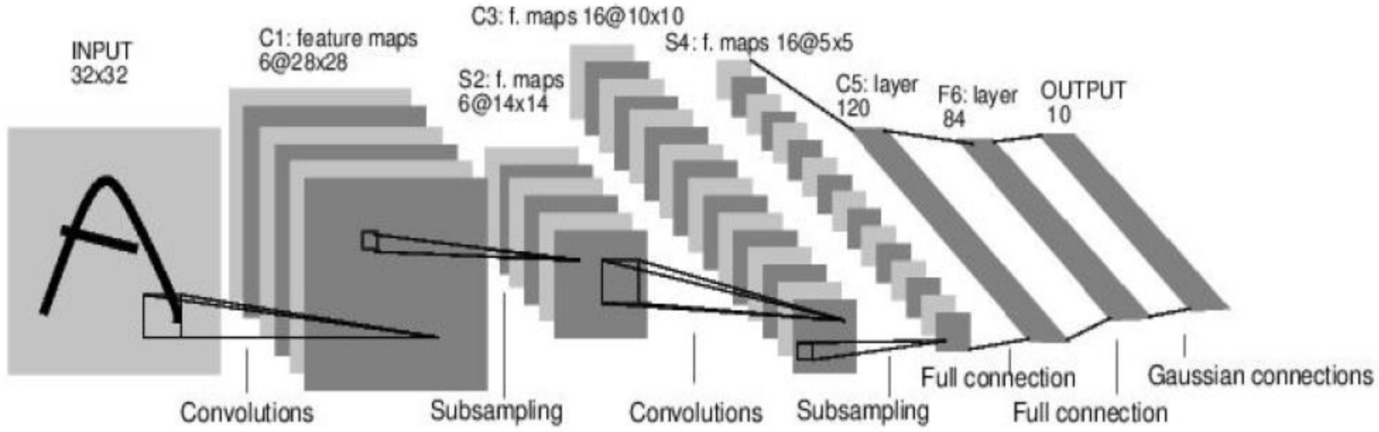
CNNs

Types of layers in a CNN:

- Convolution Layer
- Pooling Layer
- Fully Connected Layer

Case Study: LeNet-5

[LeCun et al., 1998]

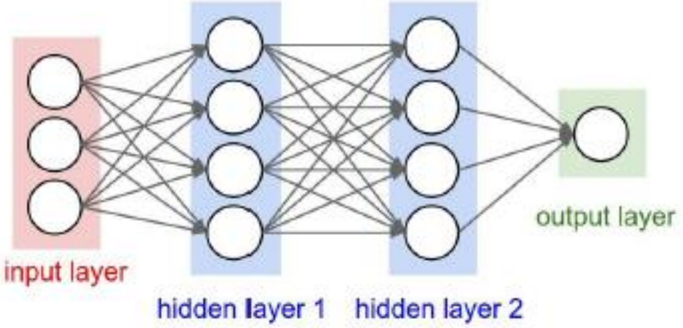


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Convolution Layer

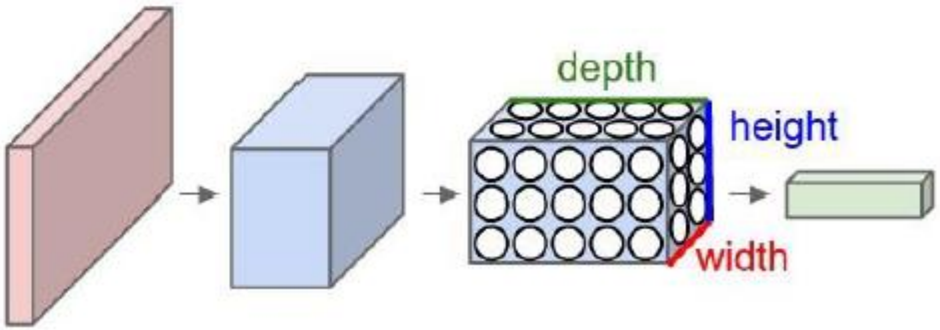
- A layer in a regular neural network take vector as input and output a vector.

Regular neural network (fully connected):



- A convolution layer takes a tensor (3d volume for RGB images) as input and generates a tensor as output

Convolutional neural network:

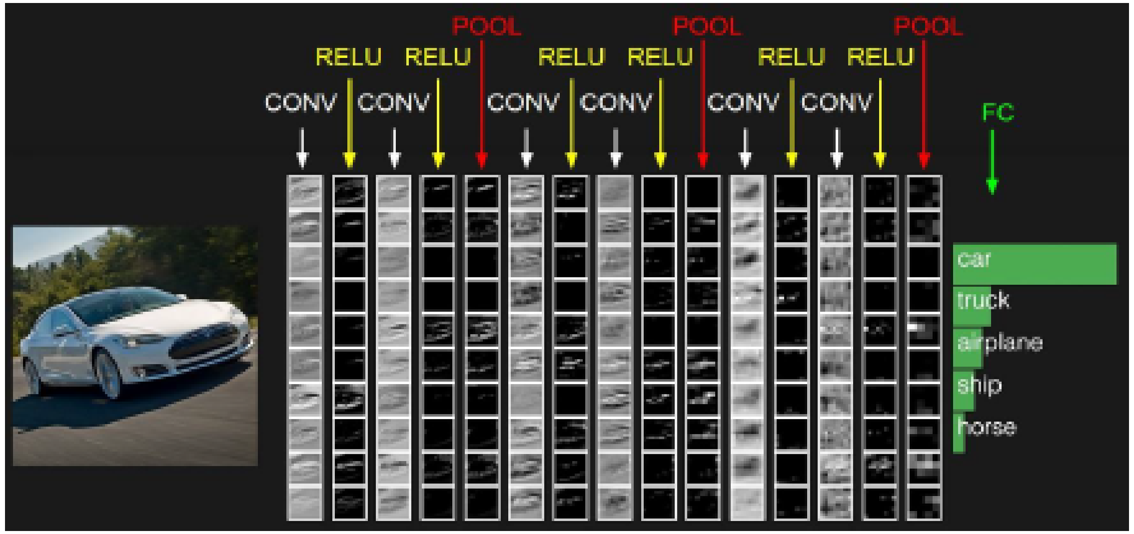


Each layer takes a 3d volume, produces 3d volume with some smooth function that may or may not have parameters.

Fig Credit: Lex Fridman, MIT, 6.S094

Convolutional Neural Networks: Layers

- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the *max(0,x)* thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

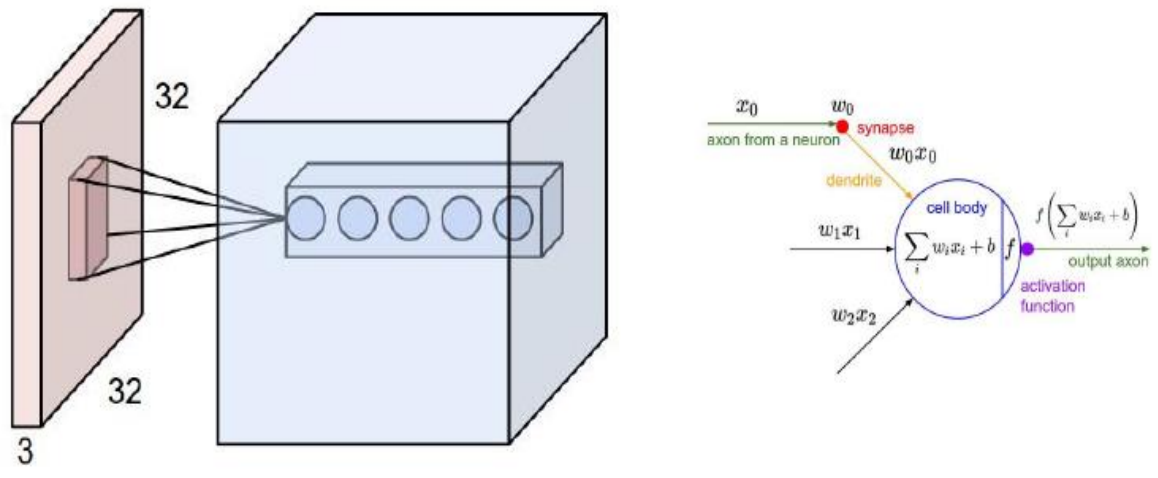


Slide Credit: Lex Fridman, MIT, 6.S094

Local Receptive Fields

- Filter (Kernel) is applied on the input image like a moving window along width and height
- The depth of a filter matches that of the input.
- For each position of the filter, the dot product of filter and the input are computed (Activation)
- The 2d arrangement of these activations is called an activation map.
- The number of such filters constitute the depth of the convolution layer

Dealing with Images: Local Connectivity



Same neuron. Just more focused (narrow “receptive field”).

The parameters on a each filter are spatially “shared”

(if a feature is useful in one place, it’s useful elsewhere)

Fig Credit: Lex Fridman, MIT, 6.S094

Convolution Operation between filter and image

- The convolution layer computes dot products between the filter and a piece of image as it slides along the image
- The step size of slide is called stride
- Without any padding, the convolution process decreases the spatial dimensions of the output

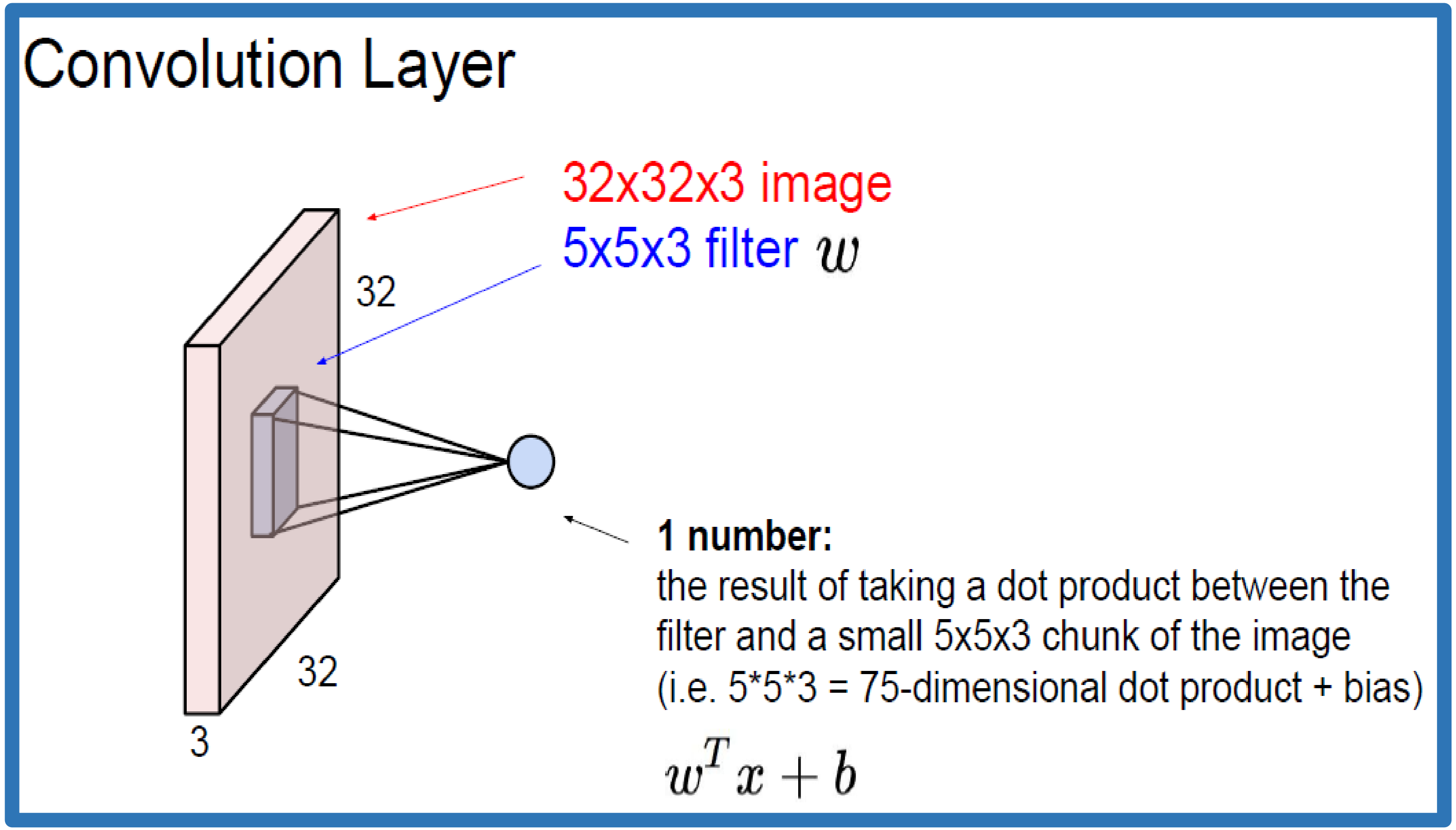


Fig Credit: A Karpathy, CS231n

Activation Maps

- Example:
  - Consider an image 32 x 32 x 3 and a 5 x 5 x 3 filter.
  - The convolution happens between a 5 x 5 x 3 chunk of the image with the filter: $w^T x + b$
  - In this example we get 75 dimensional vector and a bias term
  - In this example, with a stride of 1, we get 28 x 28 x 1 activation for 1 filter without padding
  - If we have 6 filters, we would get 28 x 28 x 6 without padding

- In the above example we have an activation map of 28 x 28 per filter.

- Activation maps are feature inputs to the subsequent layer of the network

- Without any padding, the 2D surface area of the activation map is smaller than the input surface area for a stride of >= 1

Stacking Convolution Layers

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

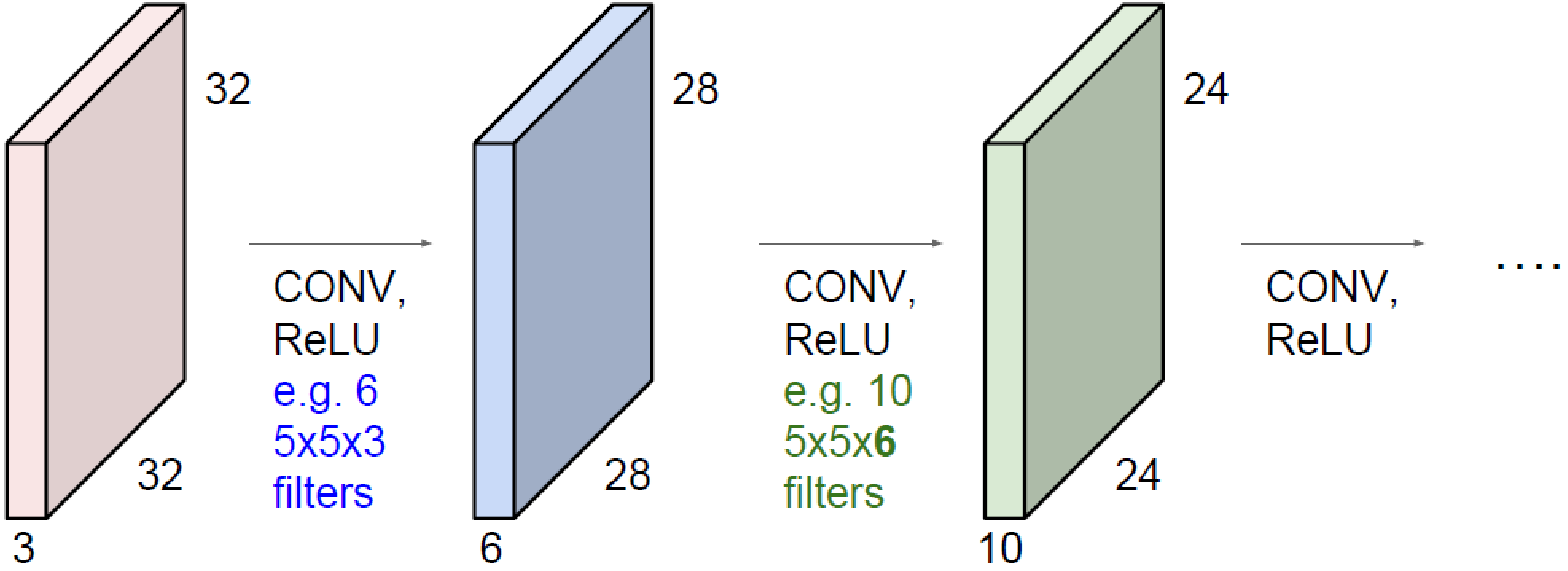
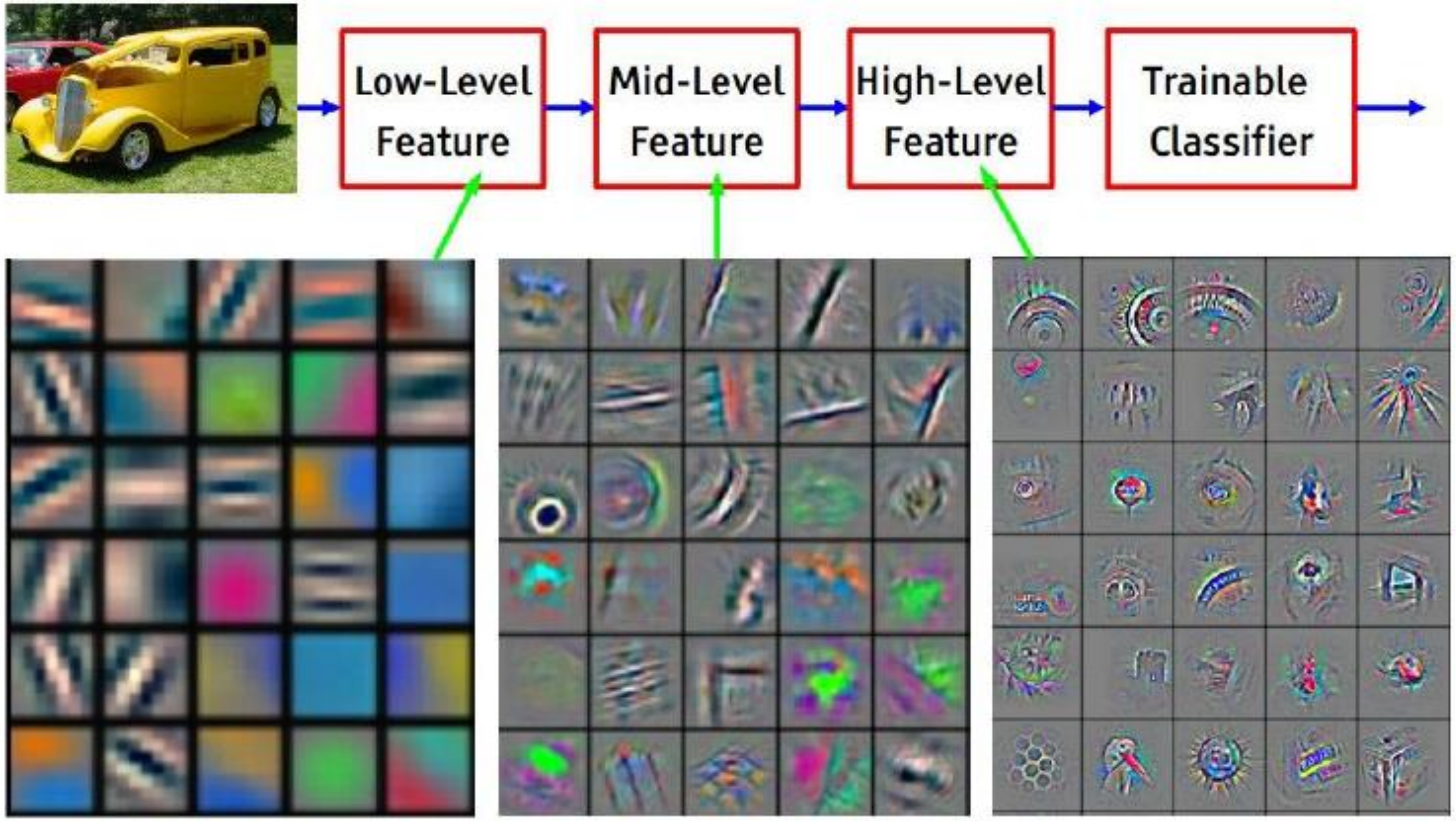


Fig Credit: A Karpathy, CS231n

Feature Representation as a hierarchy

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Padding

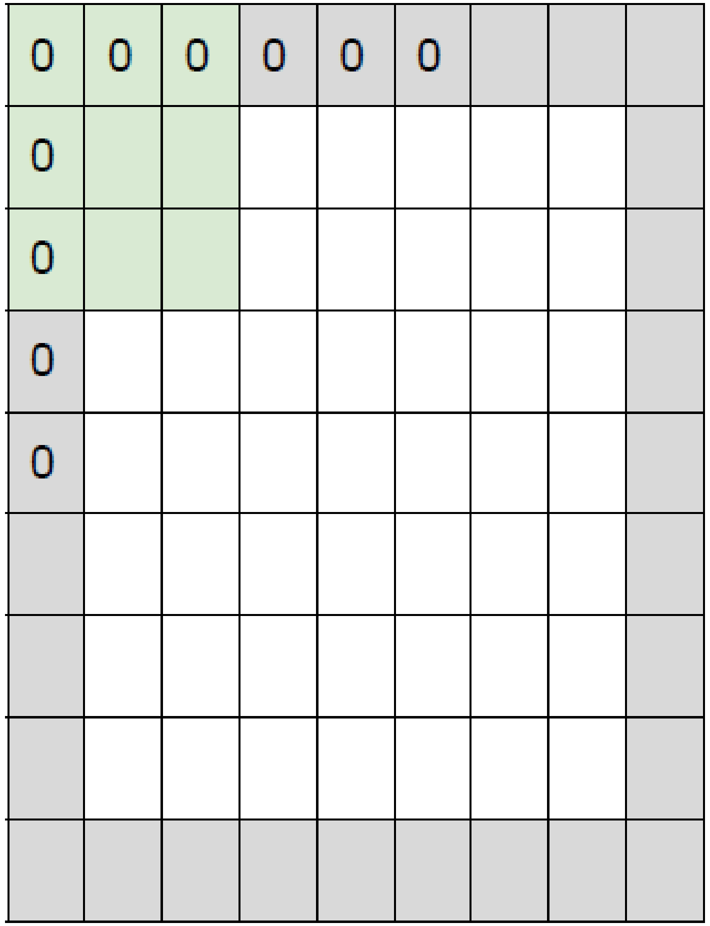
- The spatial (x, y) extent of the output produced by the convolutional layer is less than the respective dimensions of the input (except for the special case of 1 x 1 filter with a stride 1).

- As we add more layers and use larger strides, the output surface dimensions keep reducing and this may impact the accuracy.

- Often, we may want to preserve the spatial extent during the initial layers and downsample them at a later time.

- Padding the input with suitable values (padding with zero is common) helps to preserve the spatial size

Zero Padding 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

Fig Credit: A Karpathy, CS231n

Hyperparameters of the convolution layer

- Filter Size
- # Filters
- Stride
- Padding

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

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.

Fig Credit: A Karpathy, CS231n

Pooling Layer

- Pooling is a downsampling operation
- The rationale is that the “meaning” embedded in a piece of image can be captured using a small subset of “important” pixels
- Max pooling and average pooling are the two most common operations
- Pooling layer doesn’t have any trainable parameters

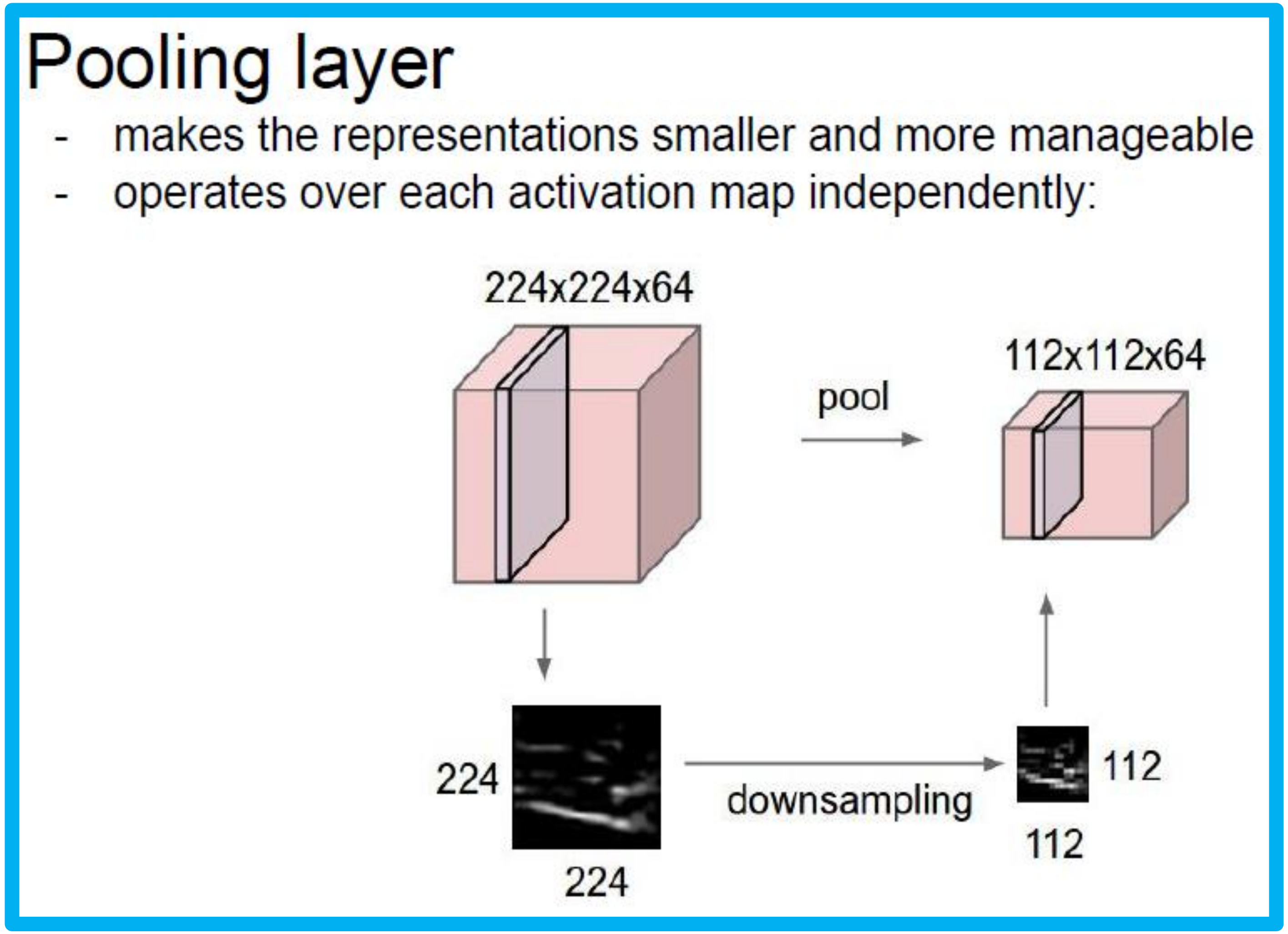
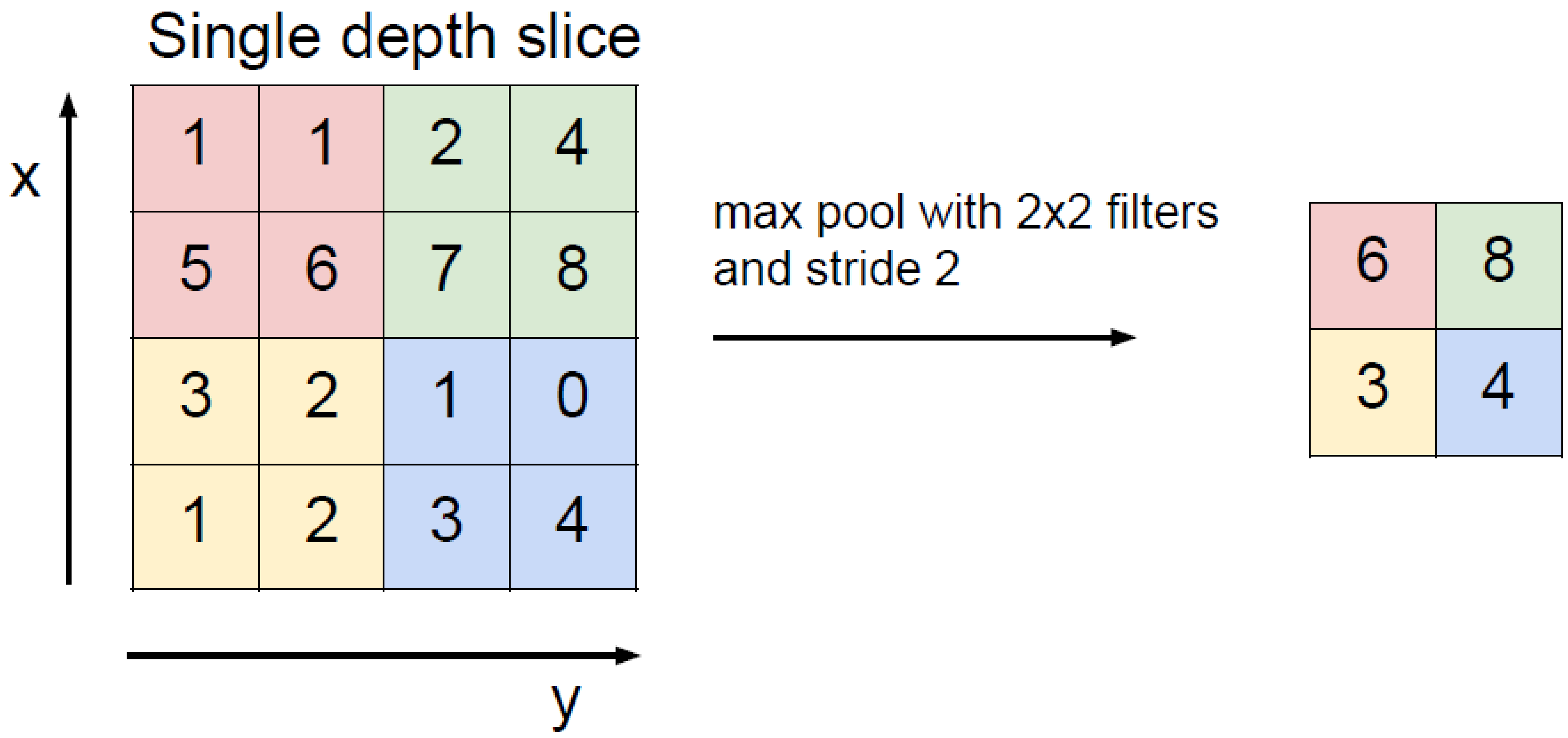


Fig Credit: A Karpathy, CS231n

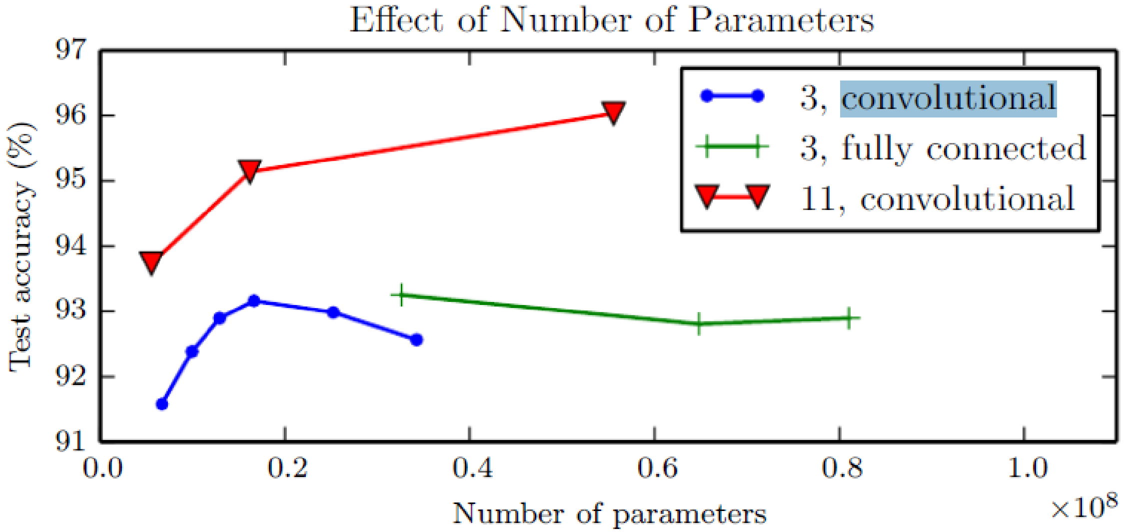
Max Pooling Illustration



Popular Network Architectures

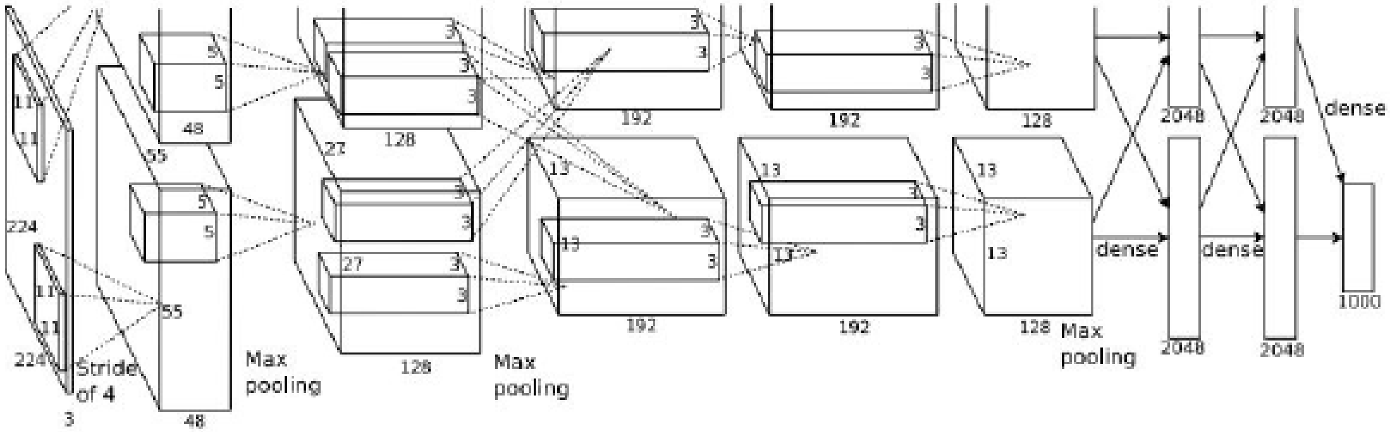
Current trend: Deeper Models

- CNNs consistently outperform other approaches for the core tasks of CV
- Deeper models work better
- Increasing the number of parameters in layers of CNN without increasing their depth is not effective at increasing test set performance.
- Shallow models overfit at around 20 million parameters while deep ones can benefit from having over 60 million.
- Key insight: Model performs better when it is architected to reflect composition of simpler functions than a single complex function. This may also be explained off viewing the computation as a chain of dependencies



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%

VGG Net

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64
maxpool		
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

VGG net

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

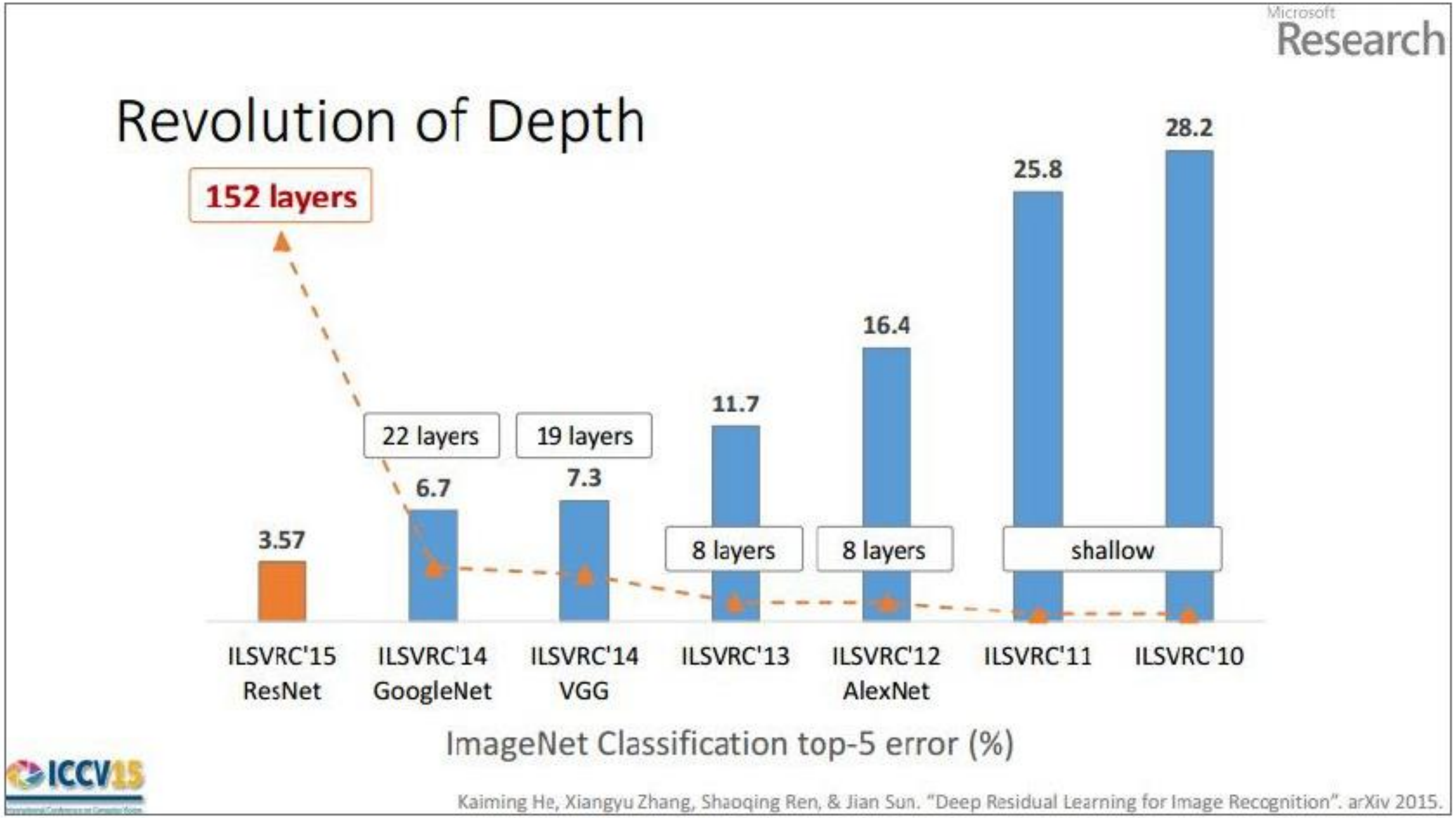
Note:

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ResNet



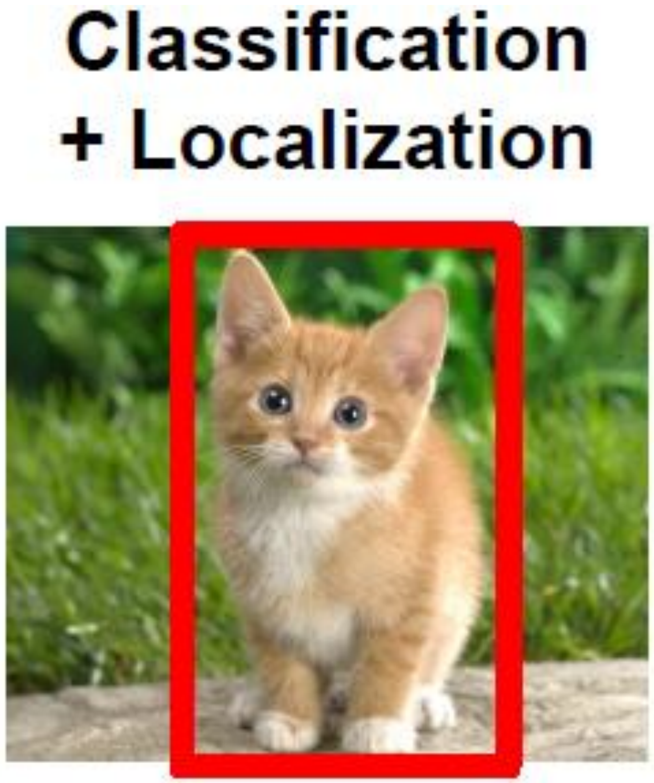
(slide from Kaiming He's recent presentation)

Core Tasks of Computer Vision

Core CV Task	Task Description	Output	Metrics
Classification	Given an image, assign a label	Class Label	Accuracy
Localization	Determine the bounding box containing the object in the given image	Box given by (x1, y1, x2, y2)	Ratio of intersection to the union (Overlap) between the ground truth and bounding box
Object Detection	Given an image, detect all the objects and their locations in the image	For each object: (Label, Box)	Mean Avg Best Overlap (MABO,) mean Average Precision (mAP)
Semantic Segmentation	Given an image, assign each pixel to a class label, so that we can look at the image as a set of labelled segments	A set of image segments	Classification metrics, Intersection by Union overlap
Instance Segmentation	Same as semantic segmentation, but each instance of a segment class is determined uniquely	A set of image segments	

Object Localization

- Given an image containing an object of interest, determine the bounding box for the object



- Classify the object

Classification + Localization: Task

Classification: C classes
- **Input:** Image
- **Output:** Class label
- **Evaluation metric:** Accuracy

Localization:
- **Input:** Image
- **Output**: Box in the image (x, y, w, h)
- **Evaluation metric:** Intersection over Union

Classification + Localization: Do both

Slide Credit: A Karpathy, CS231n

Classification + Localization: ImageNet

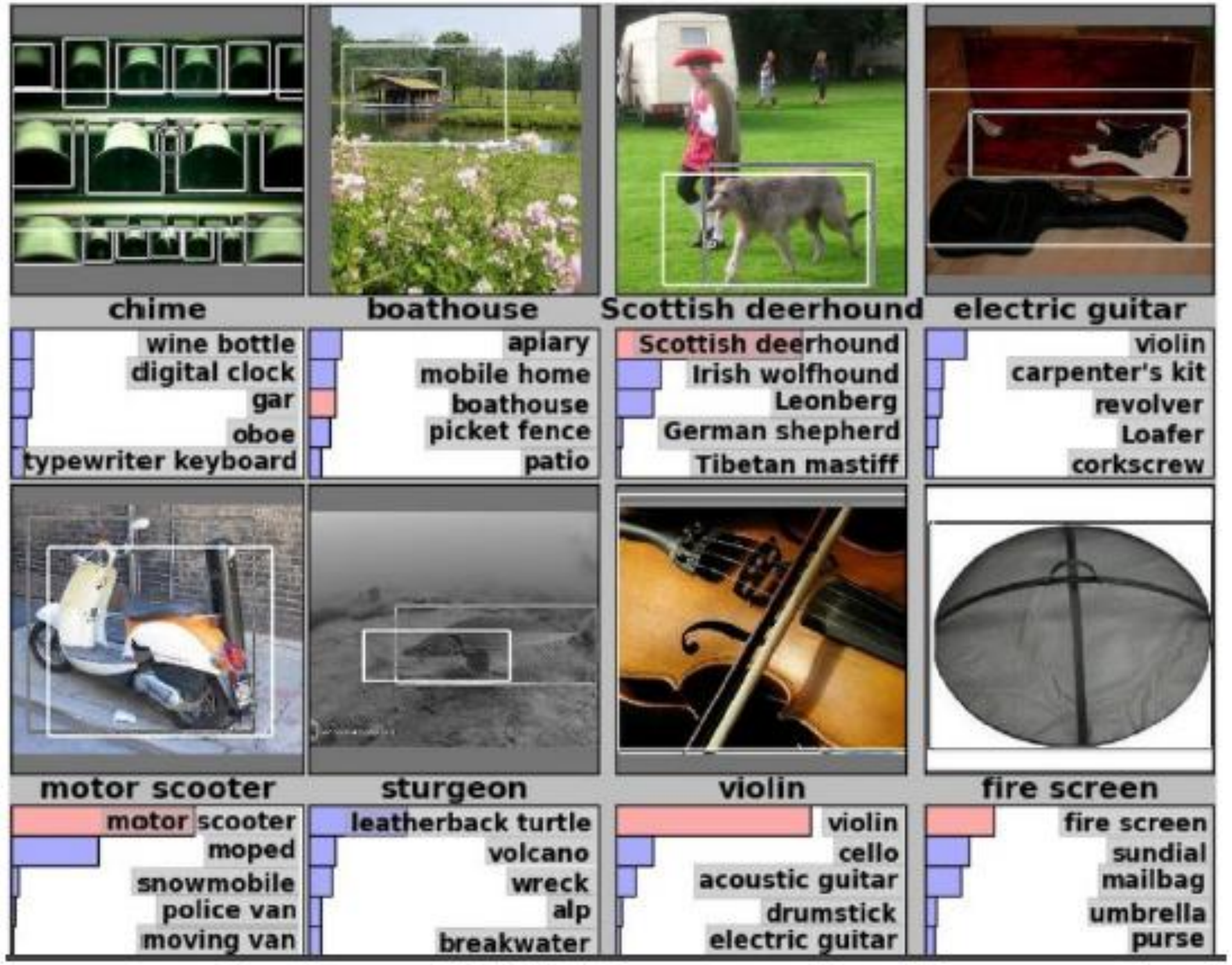
1000 classes (same as classification)

Each image has 1 class, at least one bounding box

~800 training images per class

Algorithm produces 5 (class, box) guesses

Example is correct if at least one one guess has correct class AND bounding box at least 0.5 intersection over union (IoU)



Krizhevsky et. al. 2012

Idea #1: Localization as Regression

Input: image

Neural Net

Output:
Box coordinates
(4 numbers)

Correct output:
box coordinates
(4 numbers)

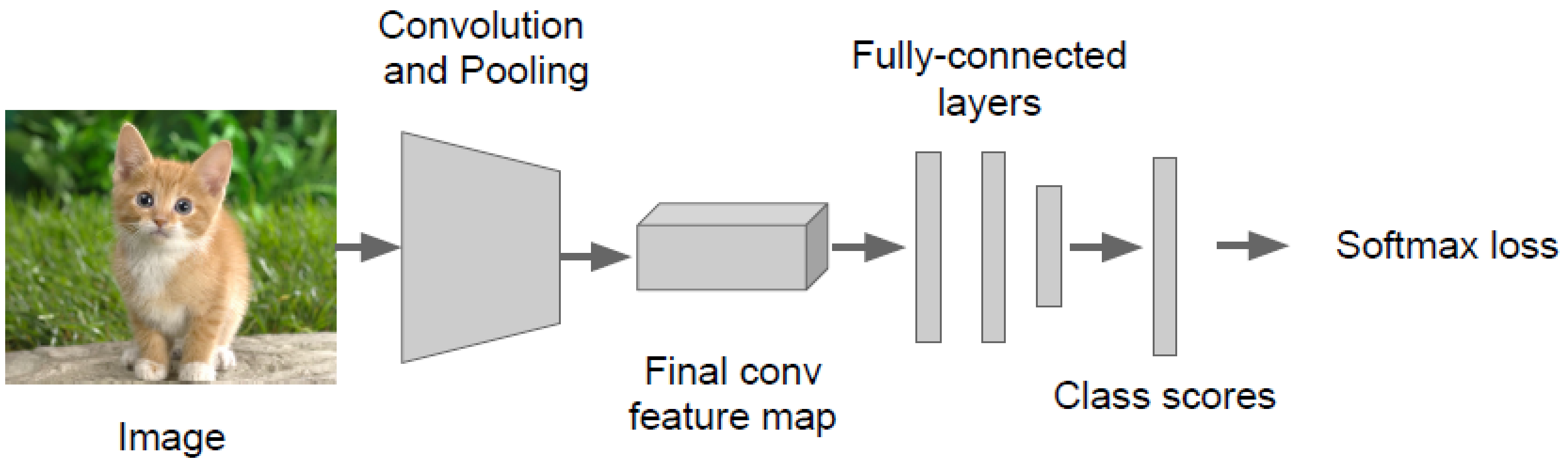
Loss:
L2 distance

Only one object,
simpler than detection

Slide Credit: A Karpathy, CS231n

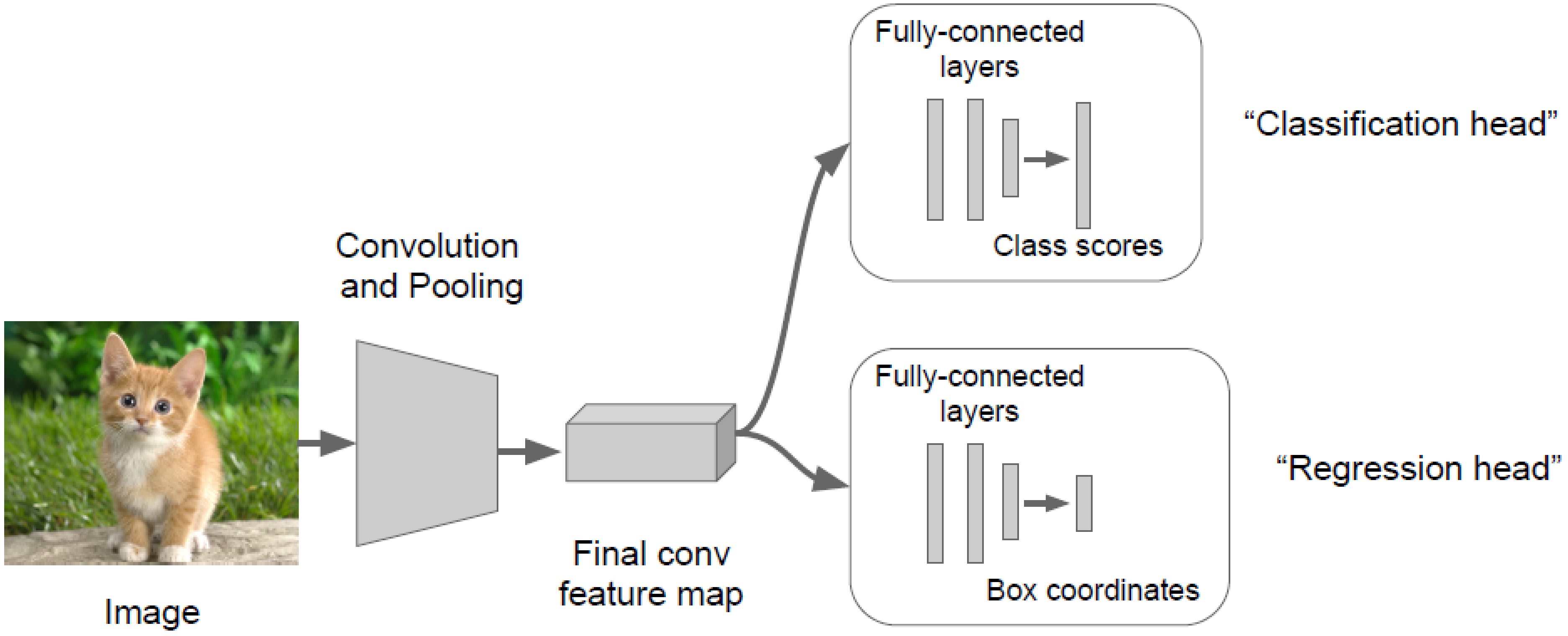
Simple Recipe for Classification + Localization

Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



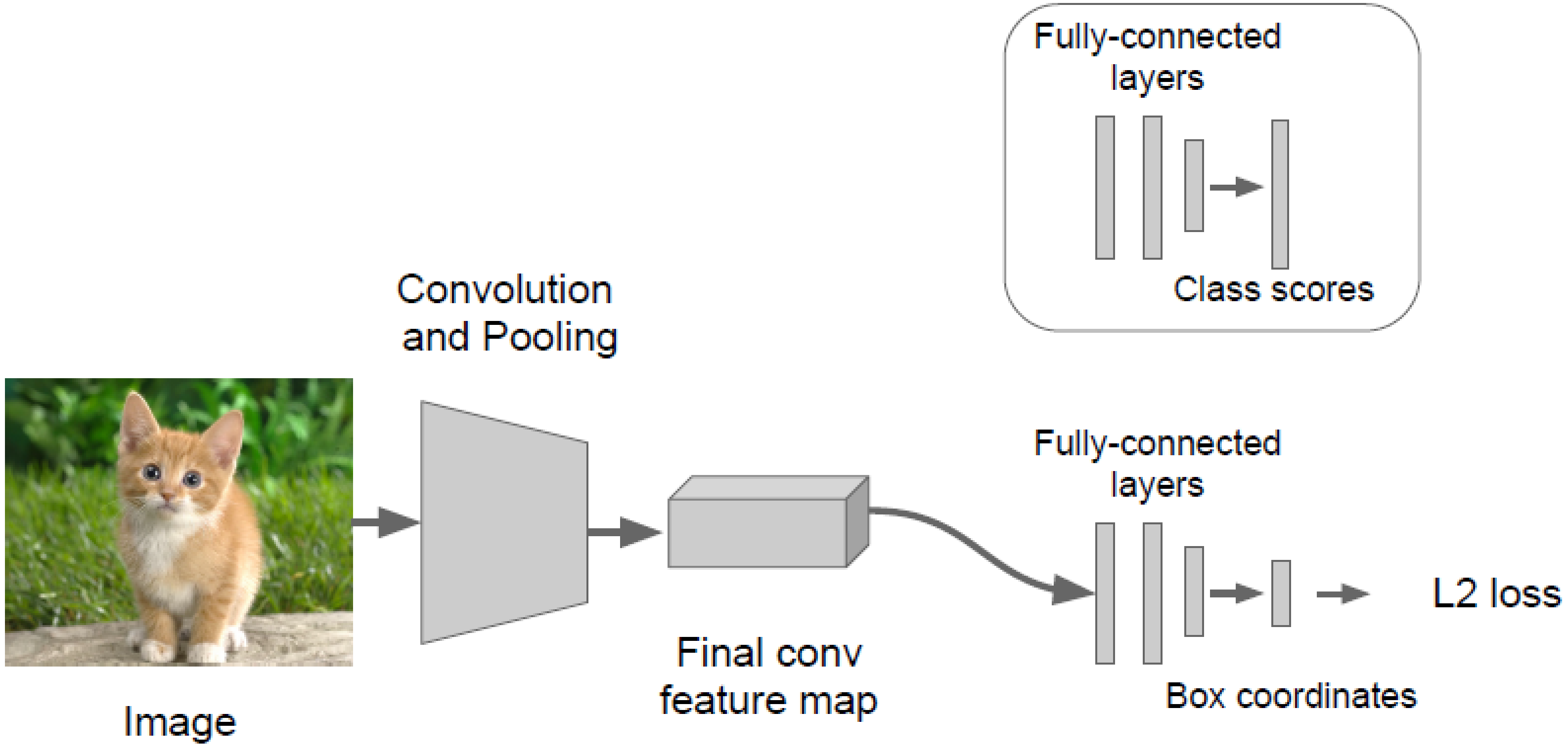
Simple Recipe for Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network



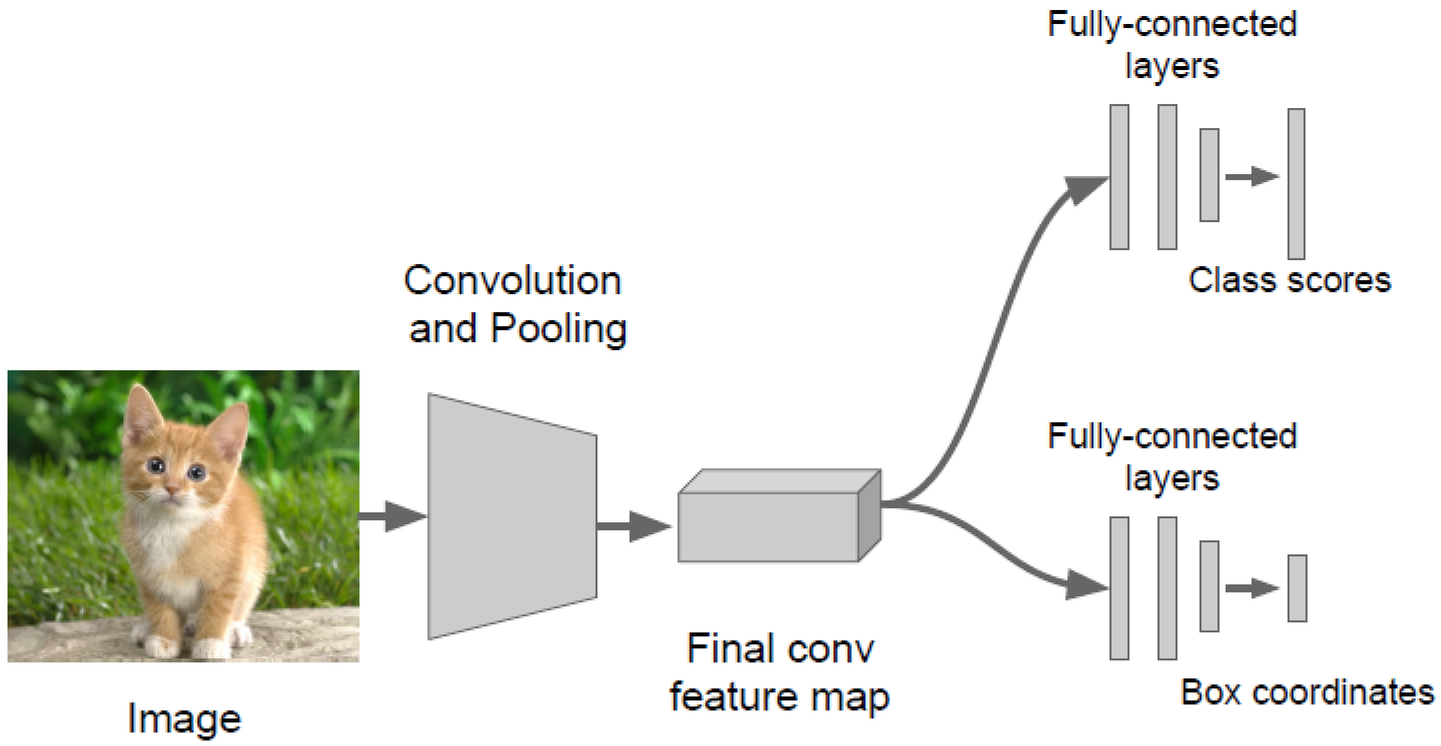
Simple Recipe for Classification + Localization

Step 3: Train the regression head only with SGD and L2 loss



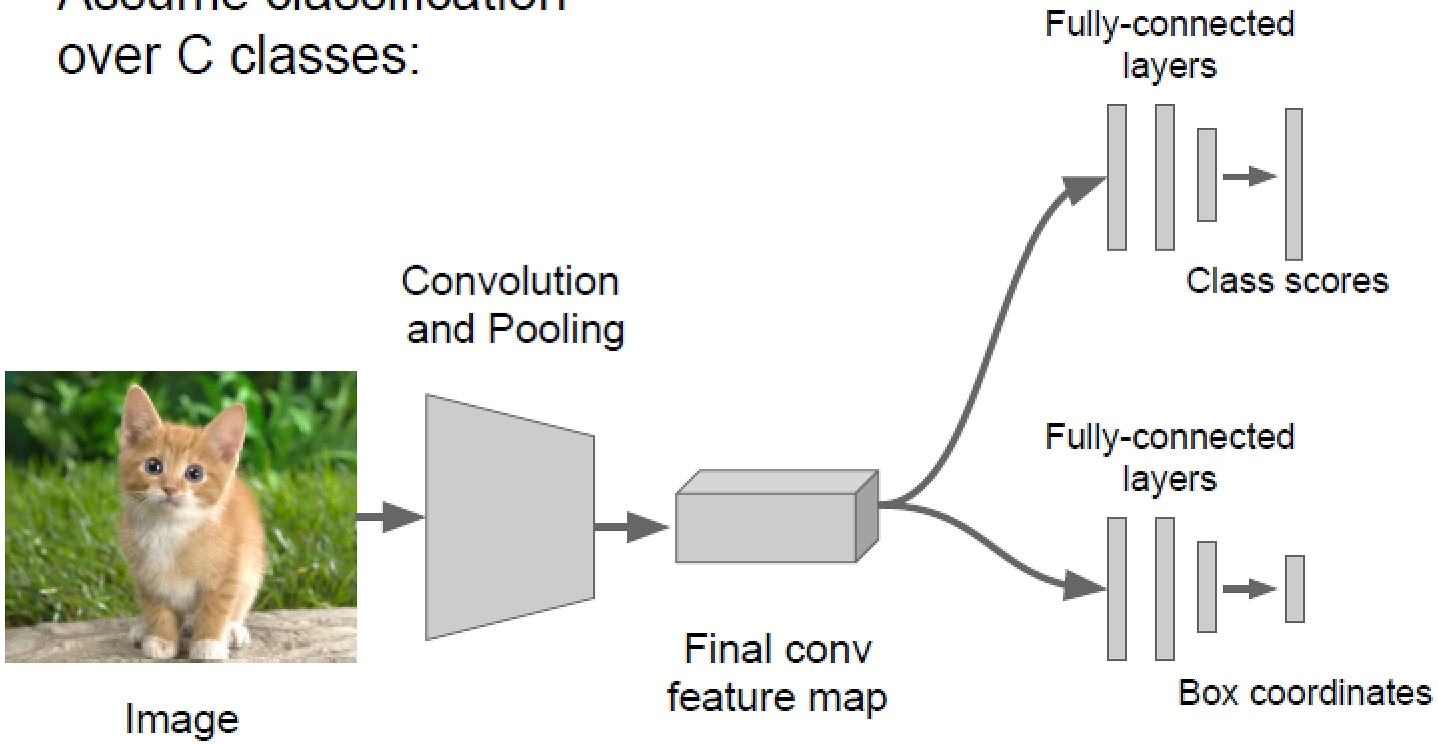
Simple Recipe for Classification + Localization

Step 4: At test time use both heads



Per-class vs class agnostic regression

Assume classification
over C classes:

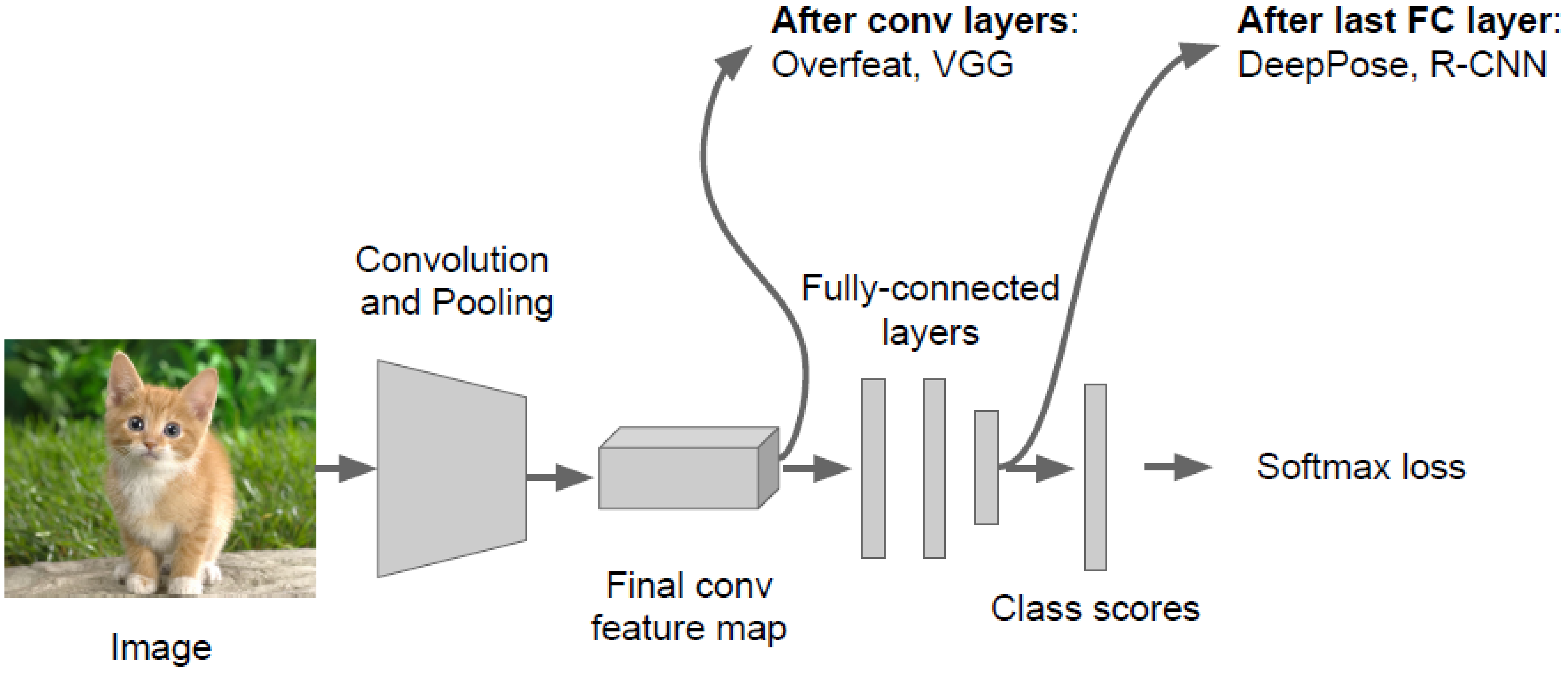


Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)

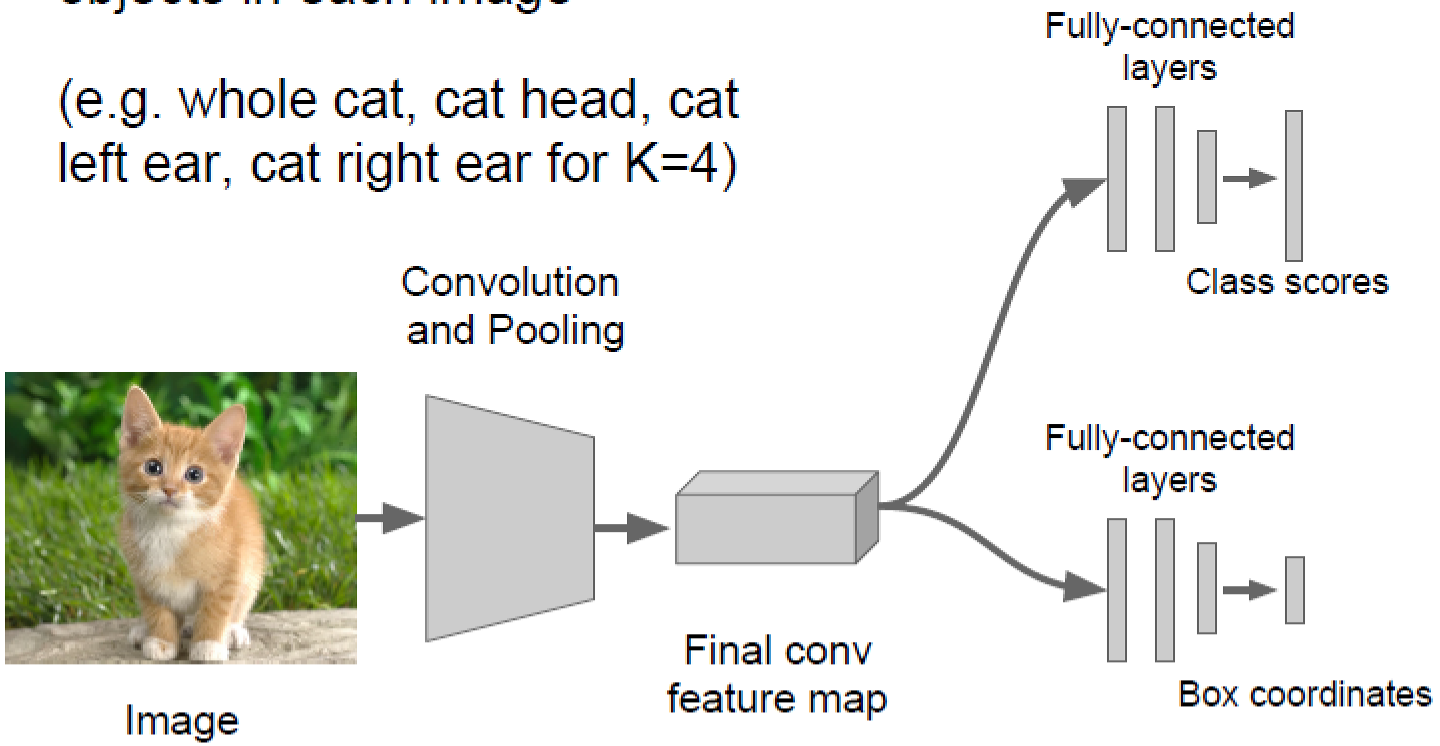
Where to attach the regression head?



Aside: Localizing multiple objects

Want to localize **exactly** K objects in each image

(e.g. whole cat, cat head, cat left ear, cat right ear for K=4)



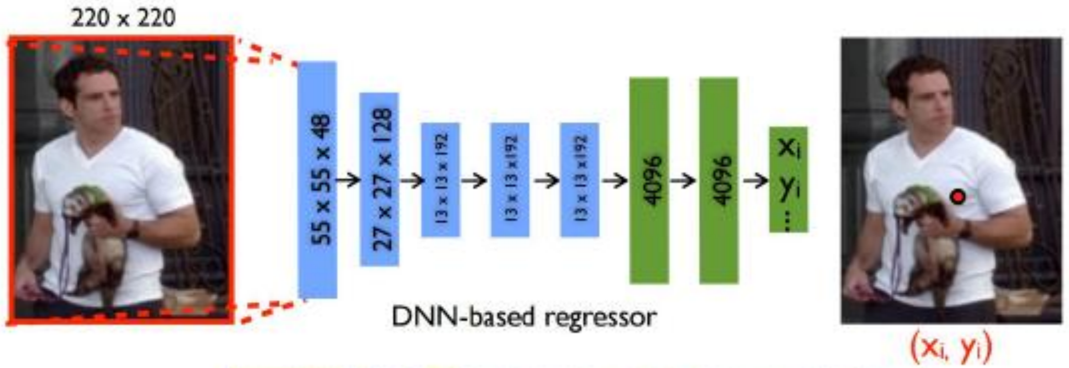
K x 4 numbers
(one box per object)

Aside: Human Pose Estimation

Represent a person by K joints

Regress (x, y) for each joint from last fully-connected layer of AlexNet

(Details: Normalized coordinates, iterative refinement)



Toshev and Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

Datasets for evaluation

- Imagenet challenges provide a platform for researchers to benchmark their novel algorithms
- PASCAL VOC 2010 is great for small scale experiments. About 1.3 GB download size.
- MS COCO datasets are available for tasks like Image Captioning. Download size is huge but selective download is possible.

	PASCAL VOC (2010)	ImageNet Detection (ILSVRC 2014)	MS-COCO (2014)
Number of classes	20	**200**	80
Number of images (train + val)	~20k	**~470k**	~120k
Mean objects per image	2.4	1.1	**7.2**