Deep Learning and Computer Vision

Applied AI, 6COSC020W

Aleka Psarrou

Acknowledgements

This material has been put together using multiple sources from WWW and slides from colleagues in the field.

Main sources of material:

- CS231n: Convolutional Neural Networks for Visual Recognition

http://cs231n.stanford.edu/schedule.html

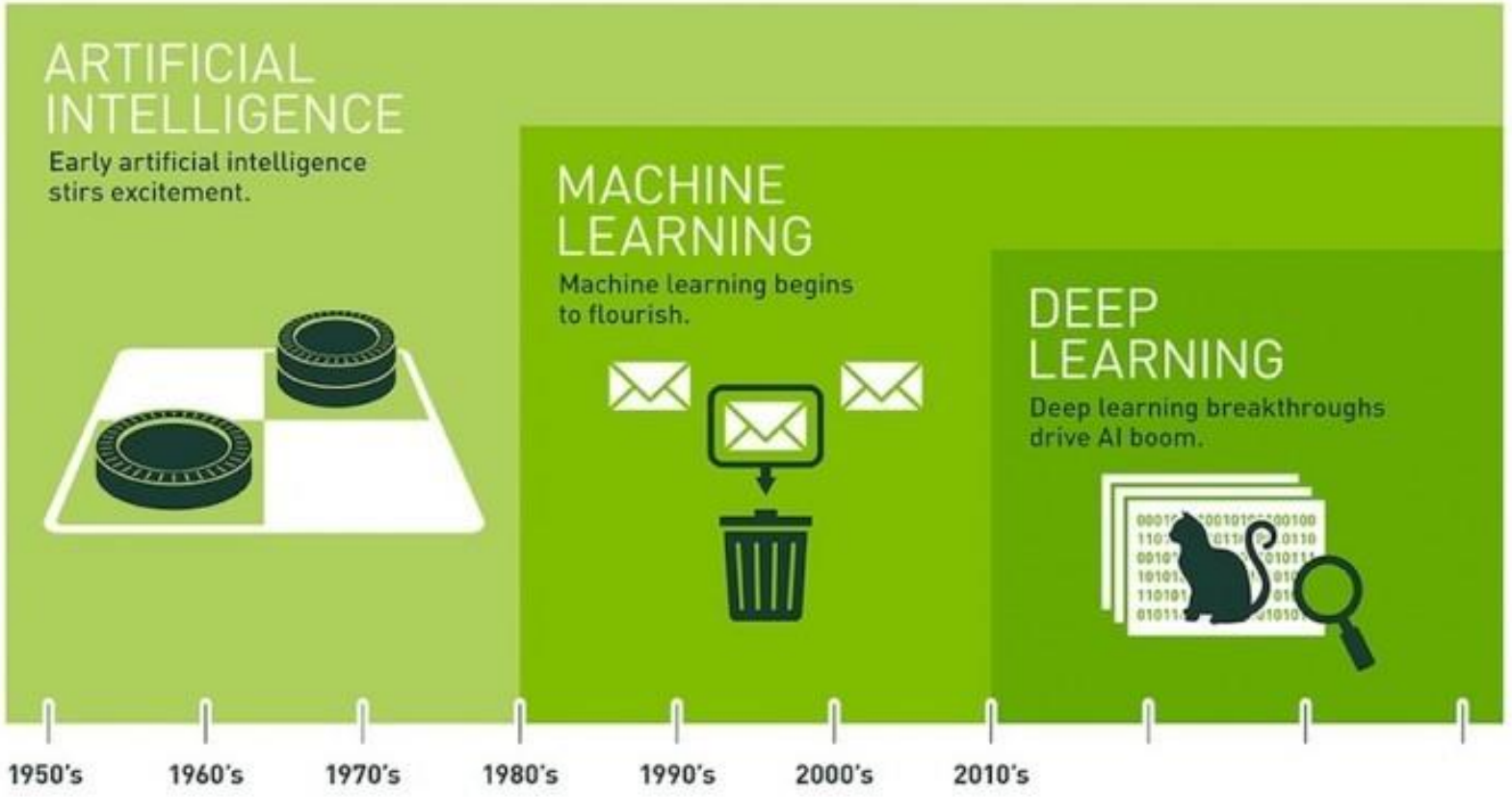
- MIT 6.S191 Introduction to Deep Learning
- Deep Learning and Computer Vision, Queen Mary, University of London
- Convolutional Neural Networks

https://medium.com/@eltronicsvilla17/convolutional-neural-network-1a02f472a90c

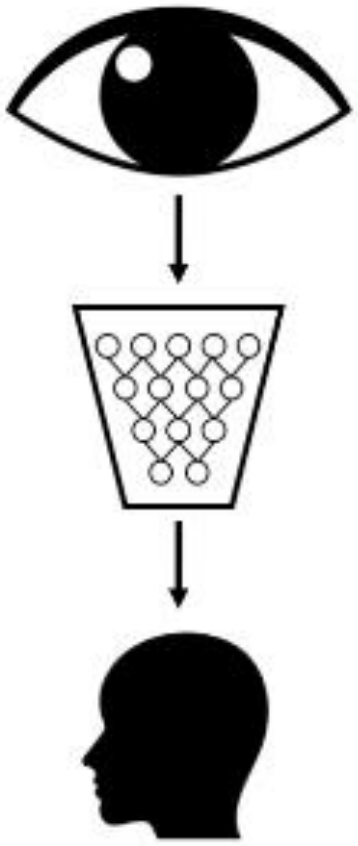
Deep Learning and Computer Vision

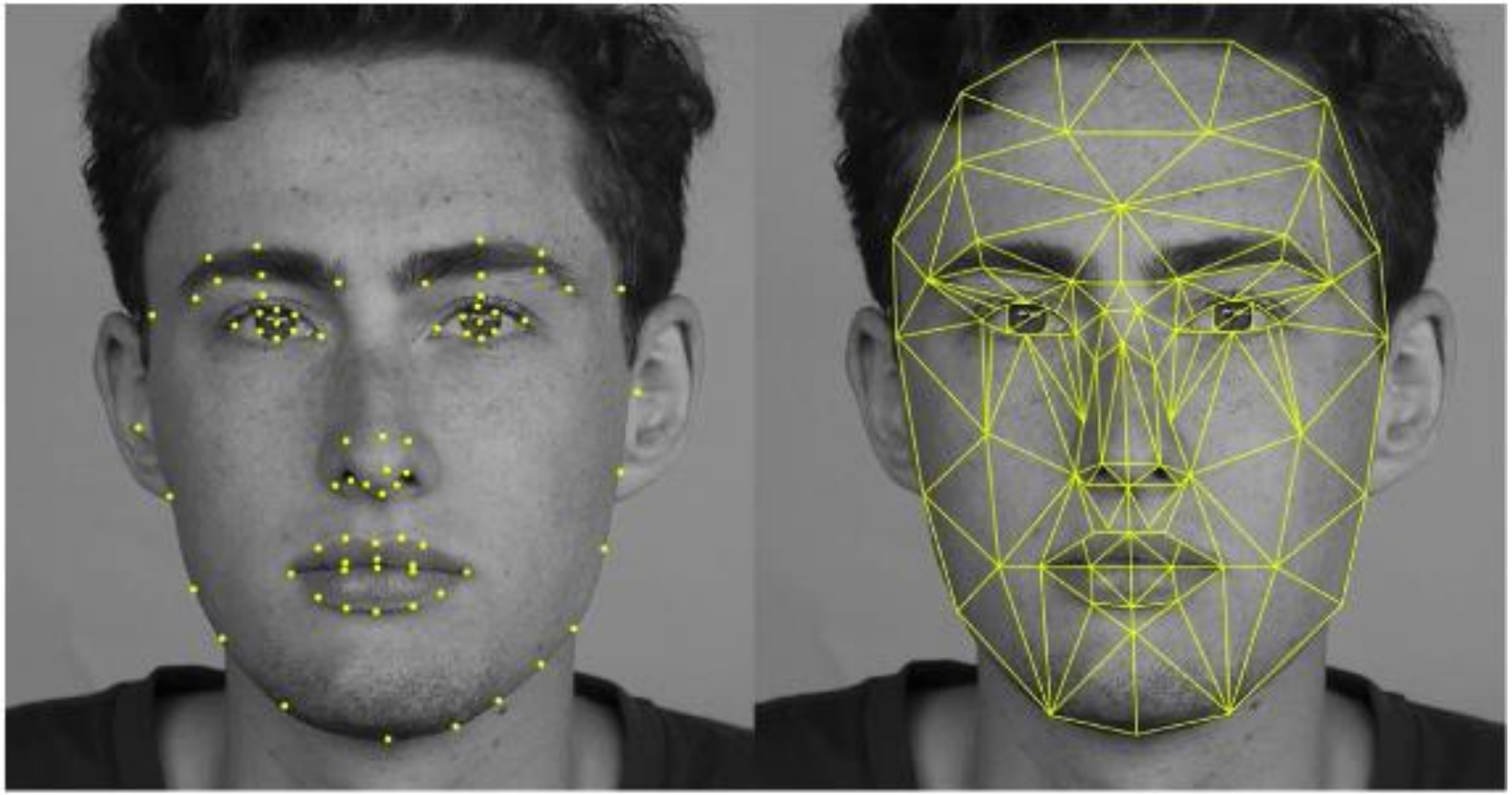
- Why is computer vision important?
  - What computers see?
  - How are features created and used in image representation
- What is deep learning?
  - How can deep learning (convolutional networks) help
  - Why deep learning now
  - Some deep learning (CNN) architectures

What is Deep Learning?

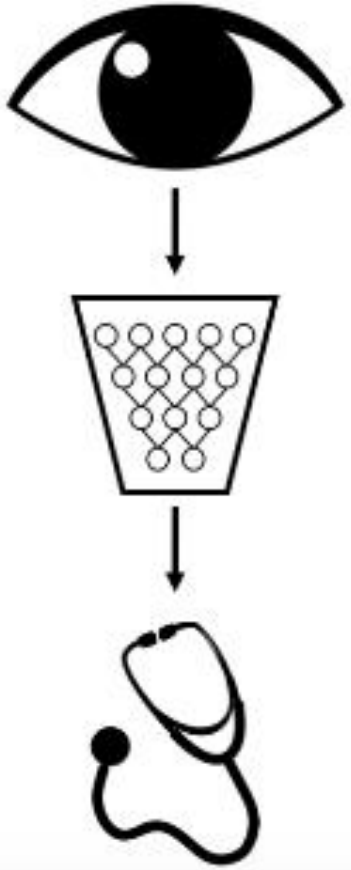


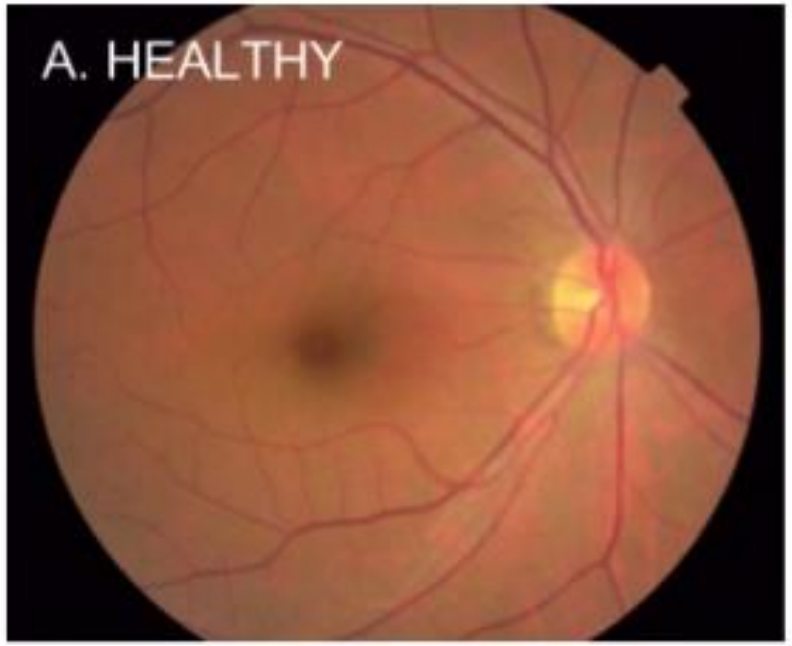
Face recognition

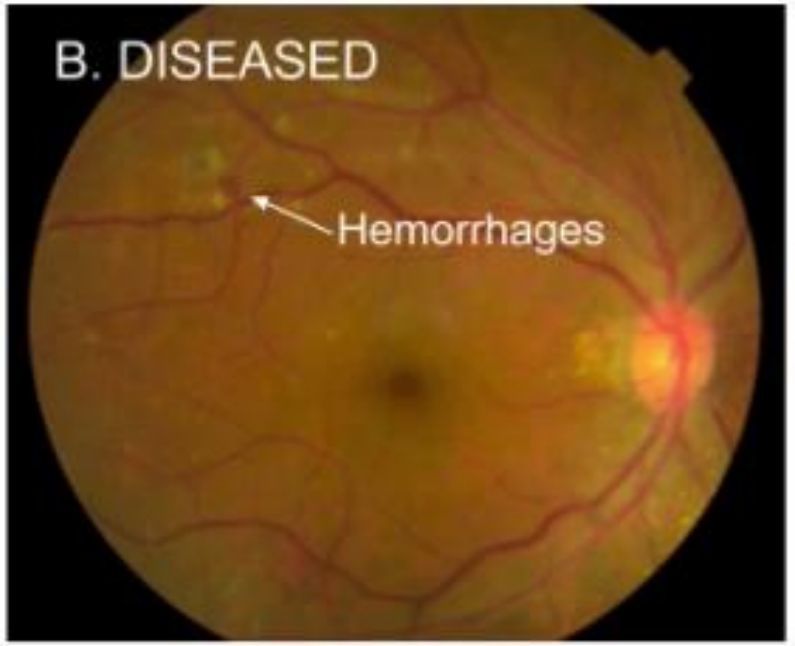




Detecting haemorrhages in retinas







Why is object detection important?

- Perception is one of the biggest bottlenecks of
  - Robotics

  - Self-driving cars

  - Surveillance

What is object detection?

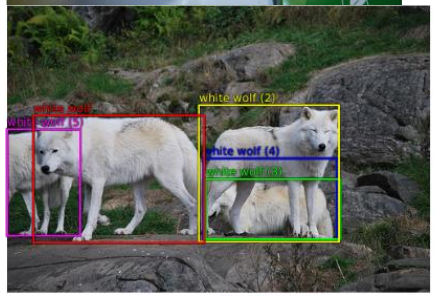
- classification

Top 5:
pencil sharpener
pool table
hand blower
oil filter
packet

Groundtruth:
pencil sharpener

ILSVRC2012_val_00010000.JPEG

- localization

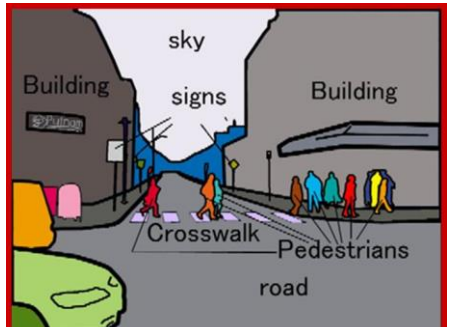


Groundtruth:
white wolf
white wolf (2)
white wolf (3)
white wolf (4)
white wolf (5)

- detection

Groundtruth:
tv or monitor
tv or monitor (2)
tv or monitor (3)
person
remote control
remote control (2)

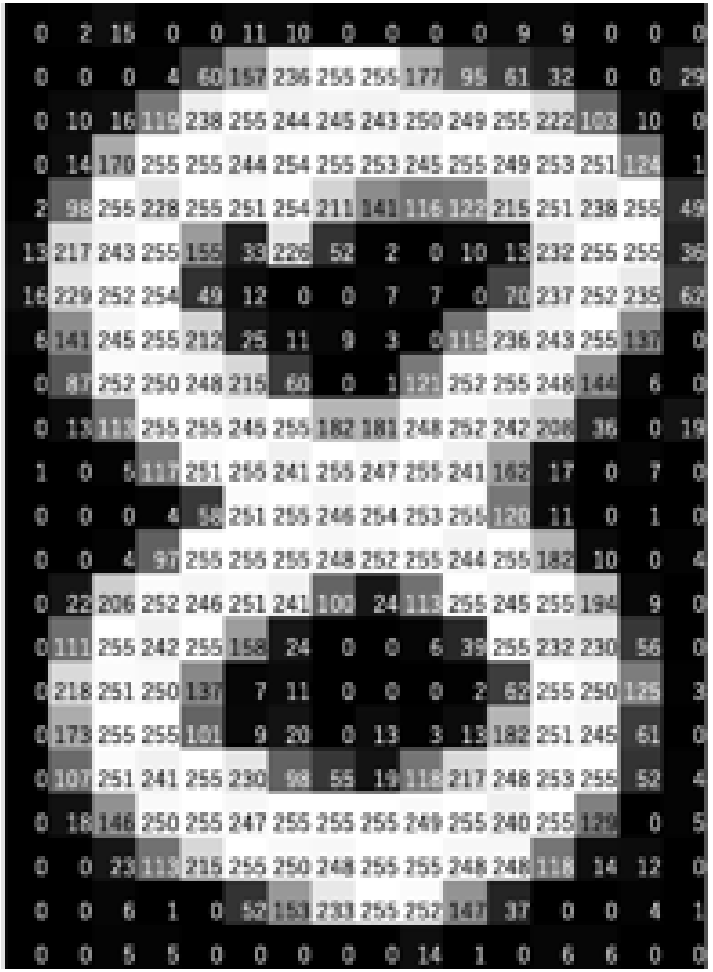
- segmentation



difficulty

What Computers See

What Computer Sees



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Image classification

Input Image

Pixel Representation

Classification

One	0.8
Five	0.1
Eight	0.05
Six	0.0.5

Features are characteristic for each class

Nose,
Eyes,
Mouth

Wheels,
Number Plate
Headlights

Door,
Windows
Steps

Traditional Recognition Approach

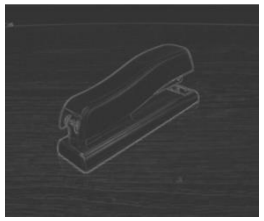
Input data (pixels)

feature representation (hand-crafted)

Learning Algorithm (e.g., SVM)

Features are not learned

Image



Low-level vision features (edges, SIFT, HOG, etc.)

Object detection / classification

Manual Feature extraction

Viewpoint variation

Scale variation

Deformation

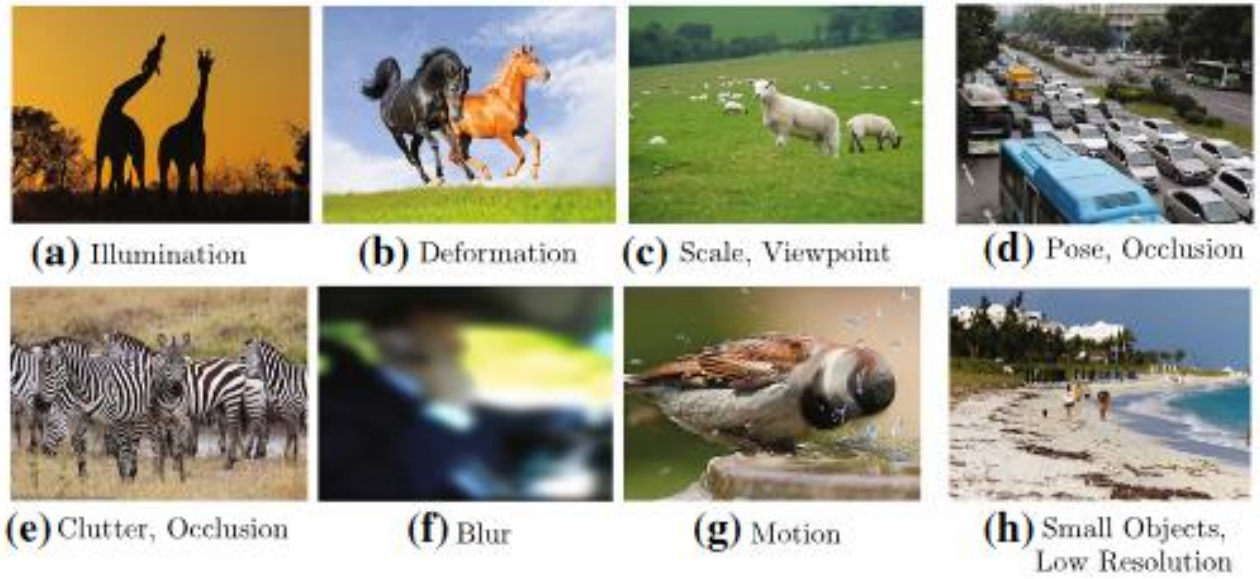
Occlusion

Illumination conditions

Background clutter

Intra-class variation

Intraclass and Interclass variations

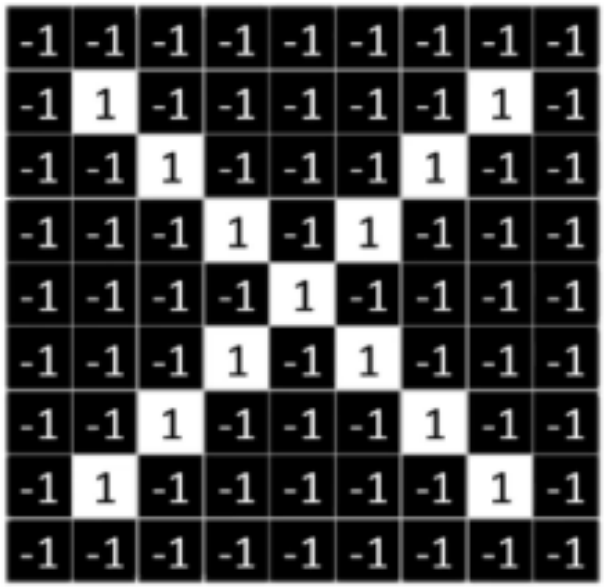


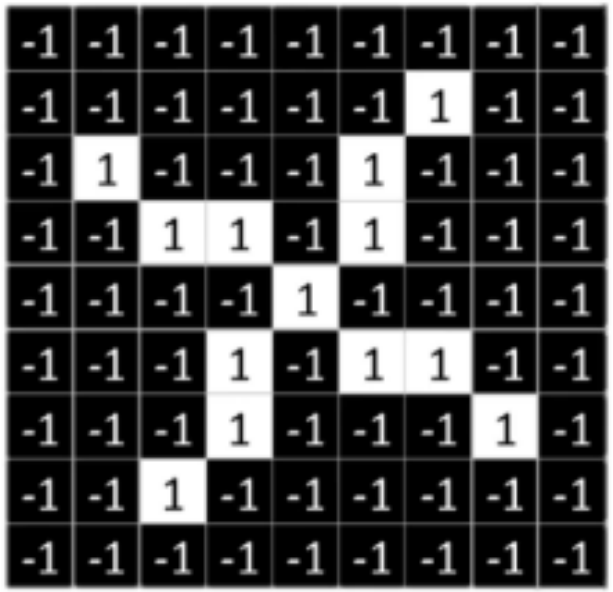
Interclass Variation: Marilyn Monroe and lookalikes

Interclass variation: Species of animals who look similar but in fact, are from four different object classes

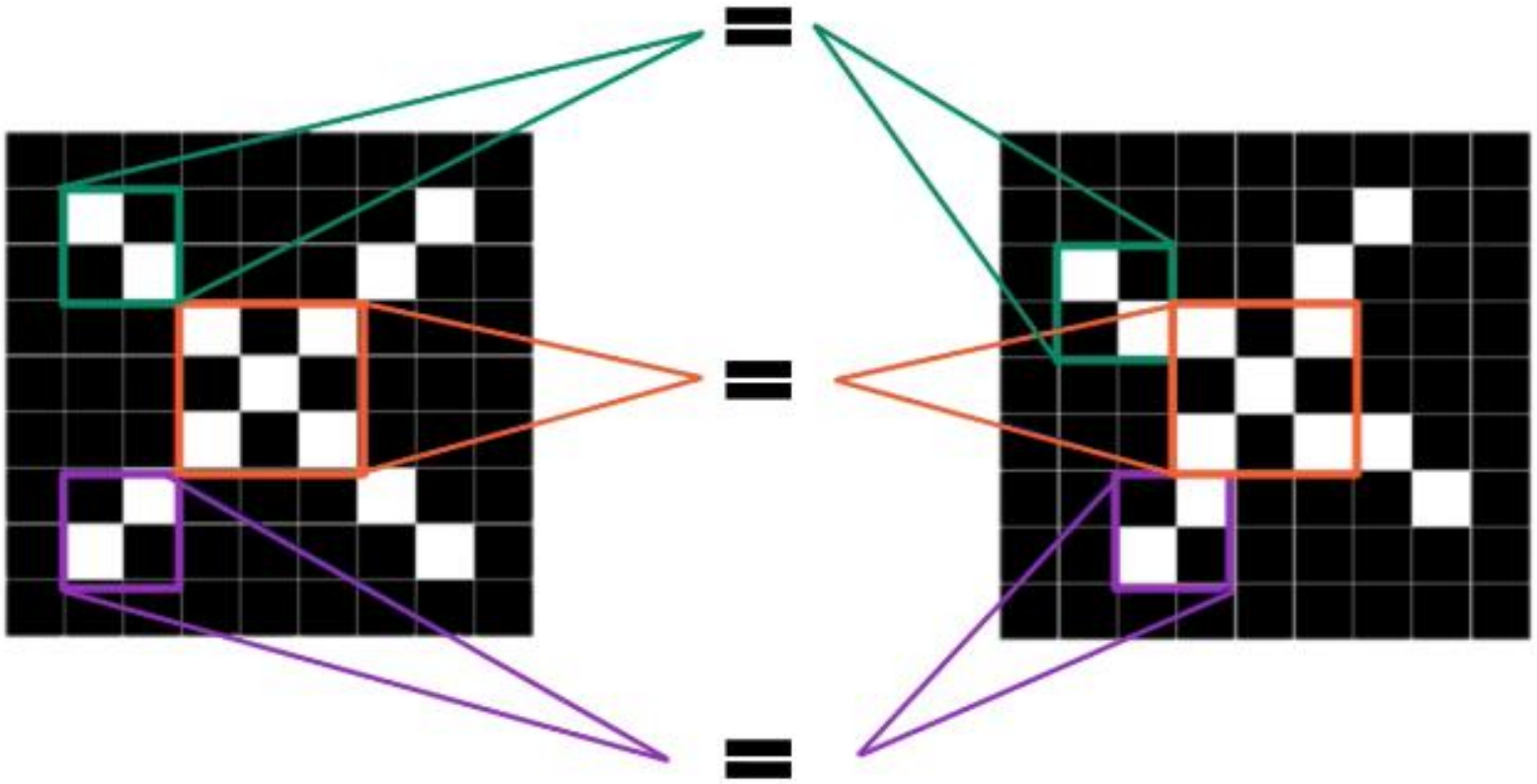
Feature Detection and Convolution:
A case study

X or X?

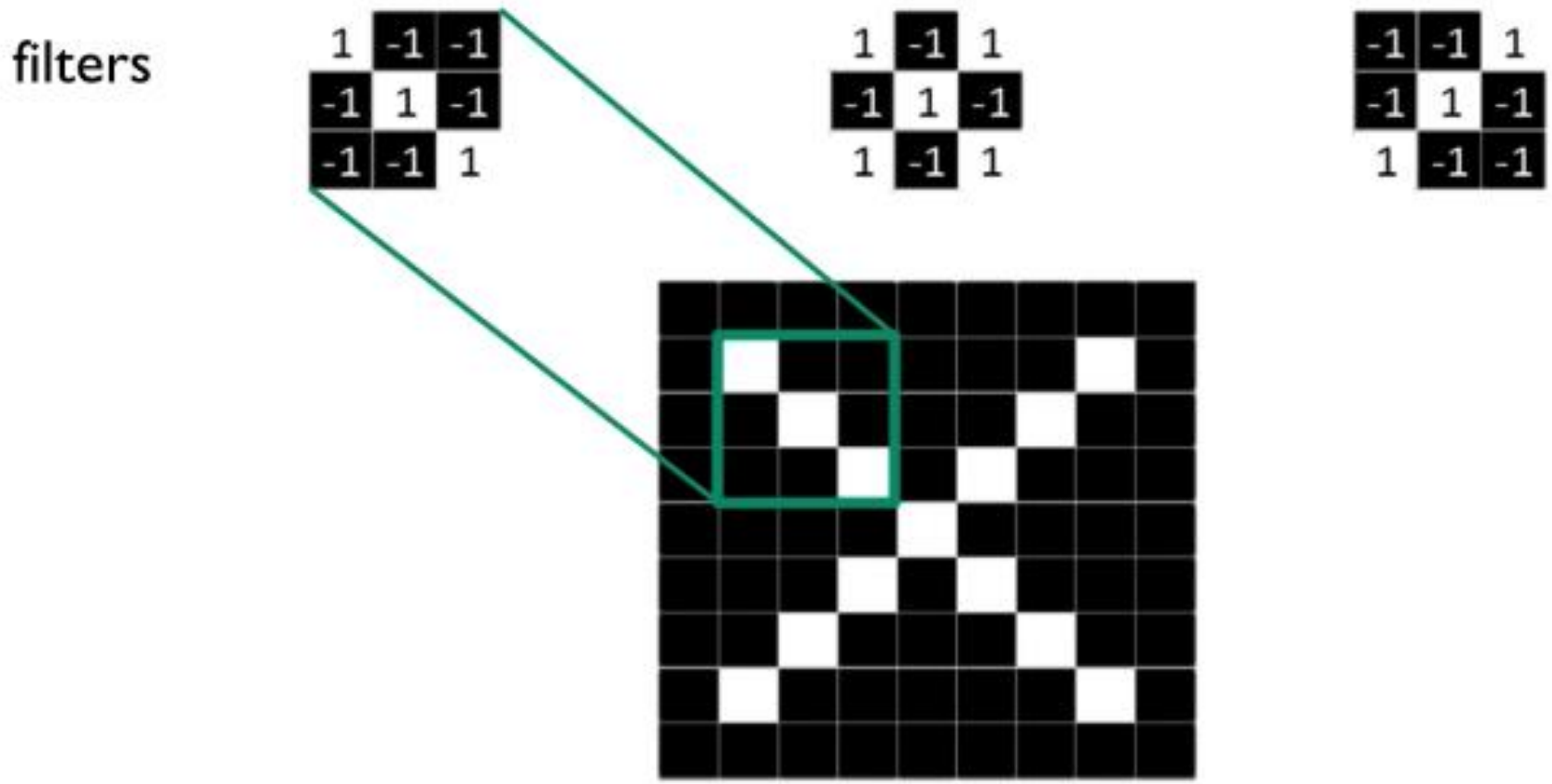




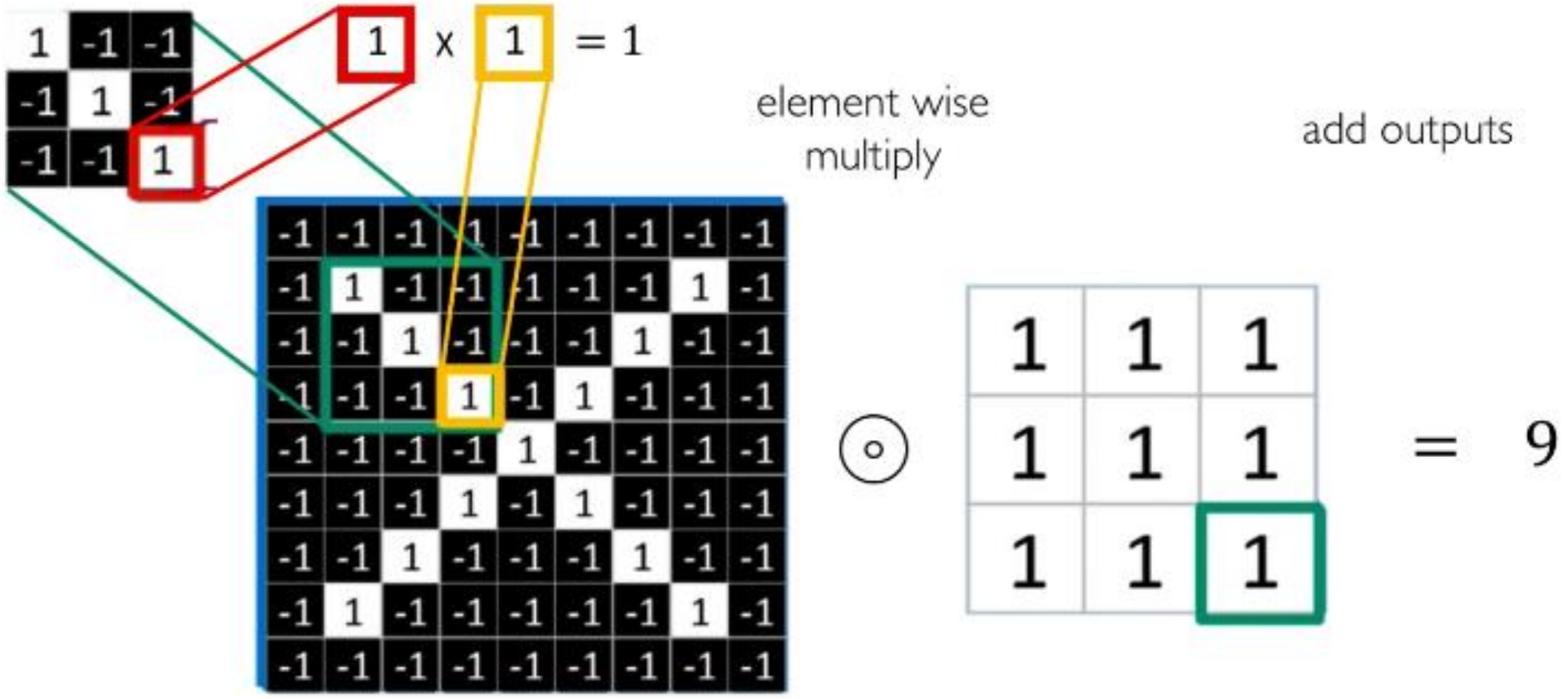
Features of X



Filters to detect Features of “X”

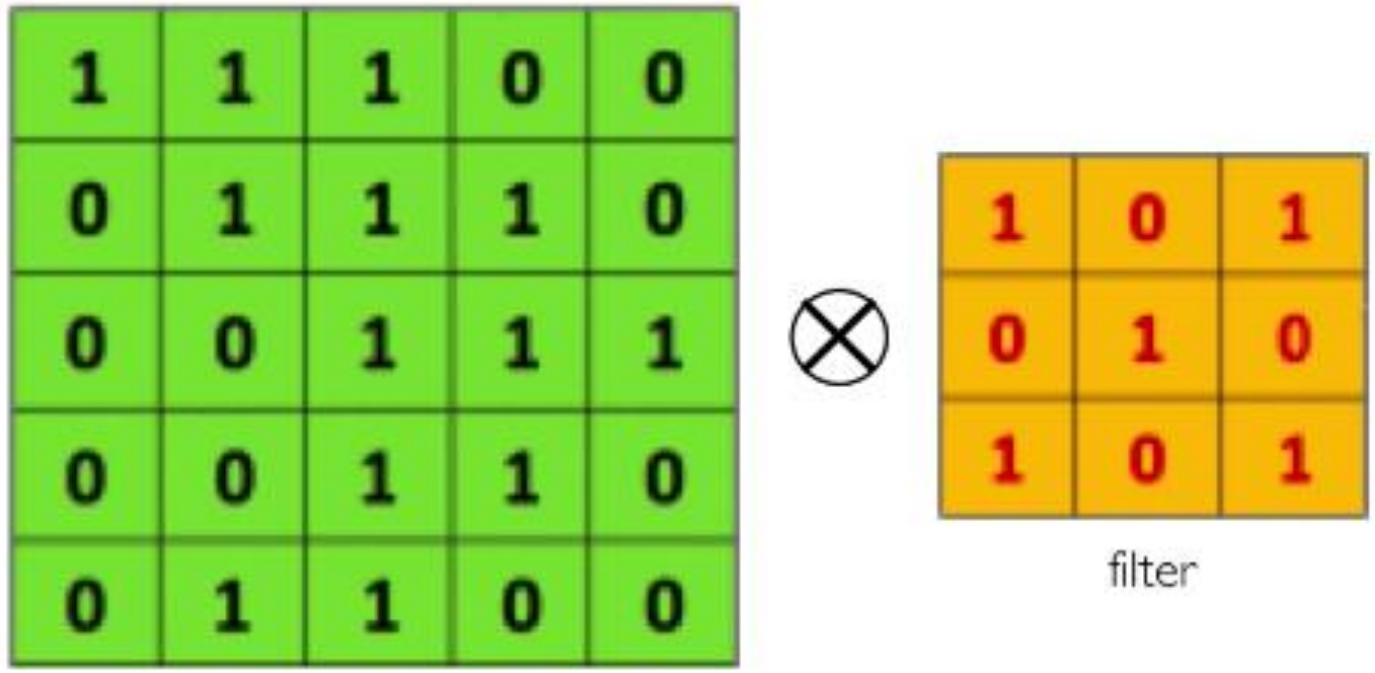


The convolution operation

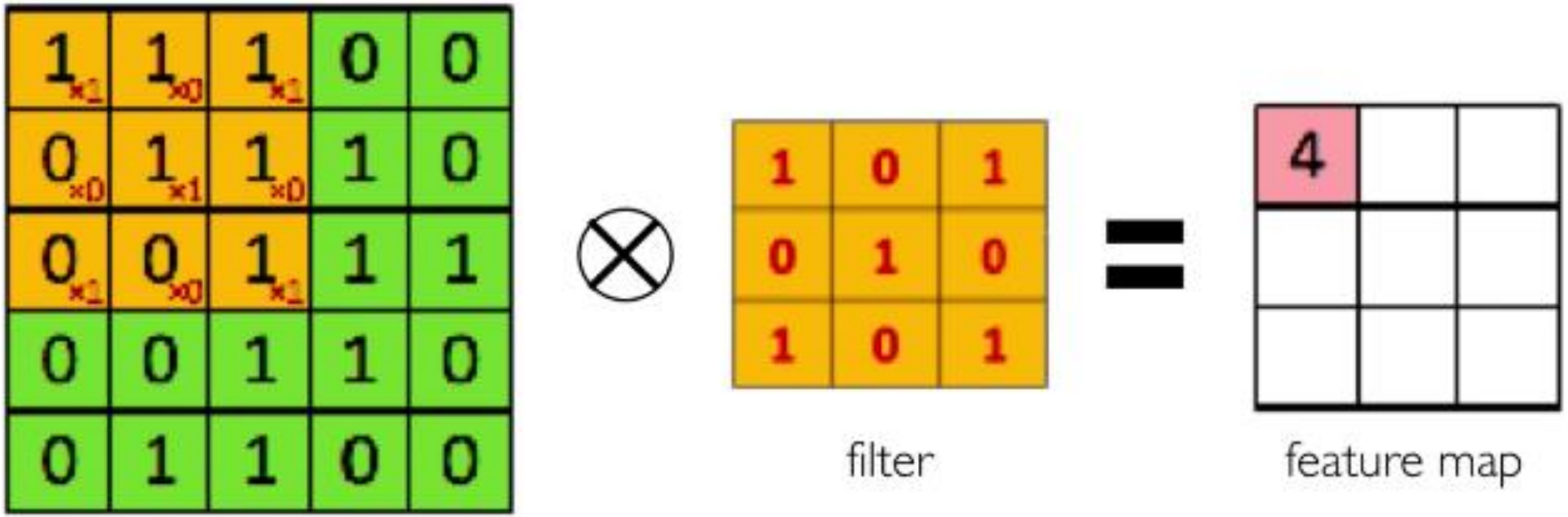


The convolution operation

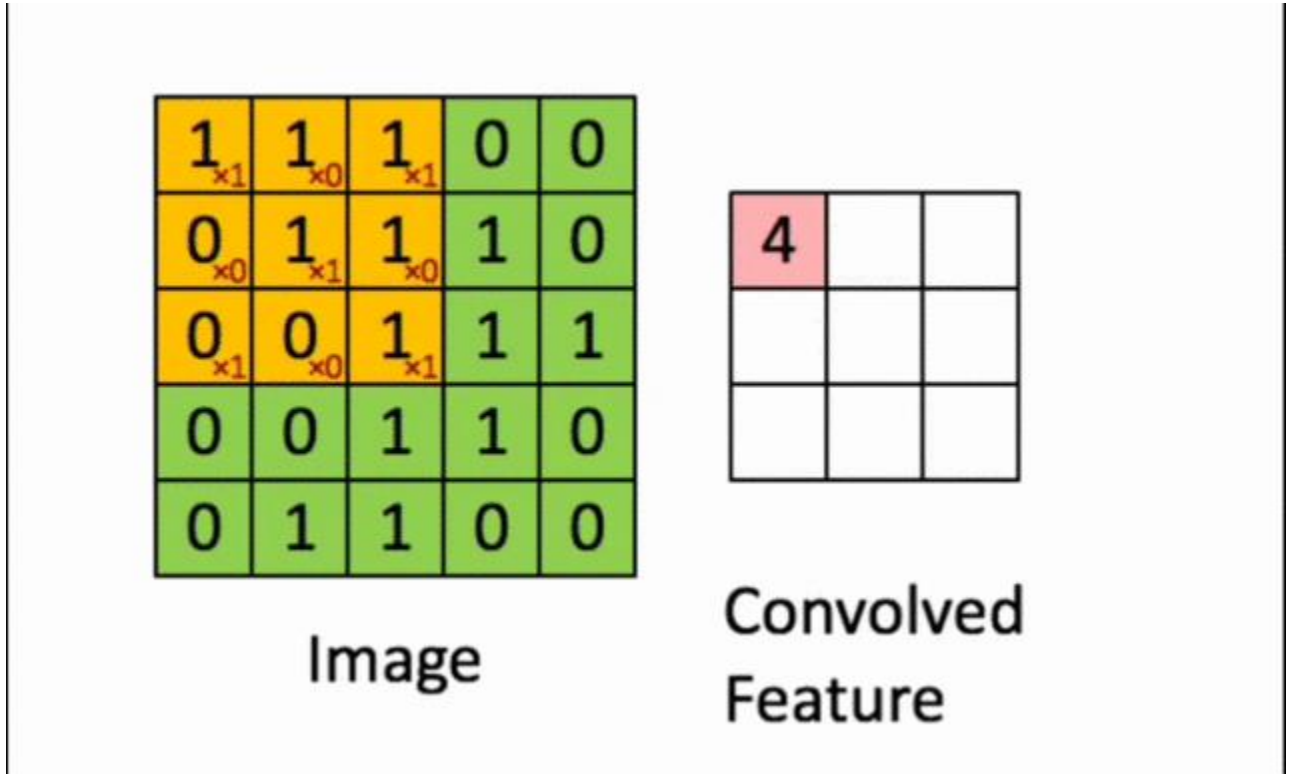
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



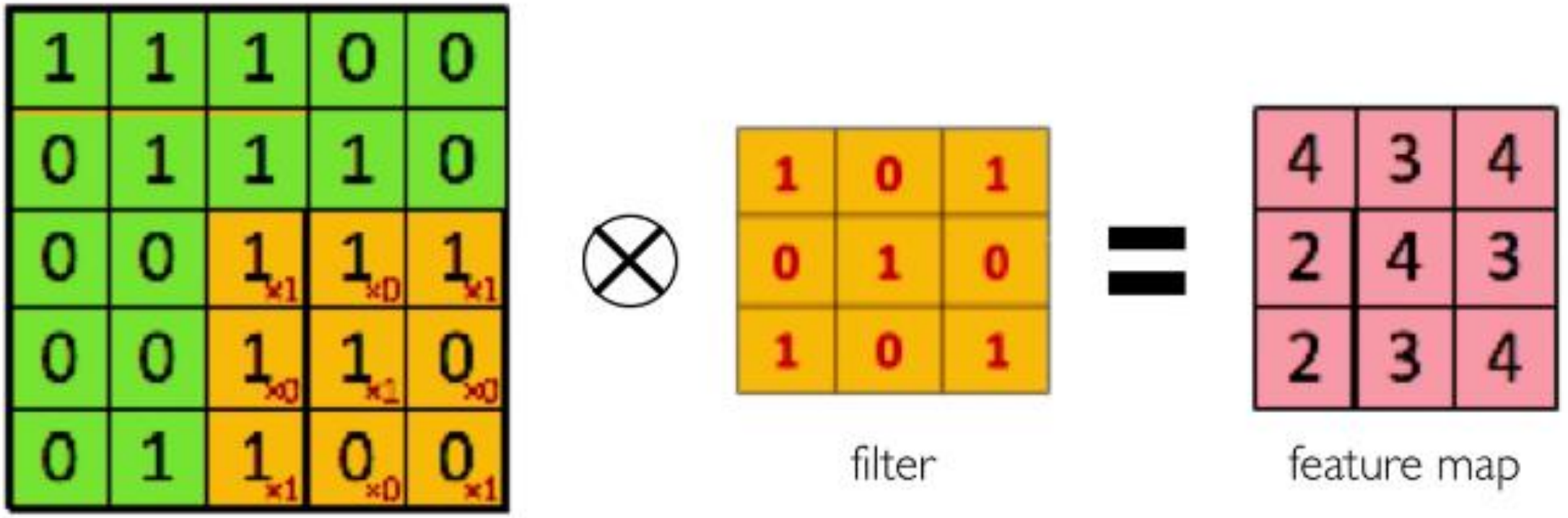
The Convolution operation



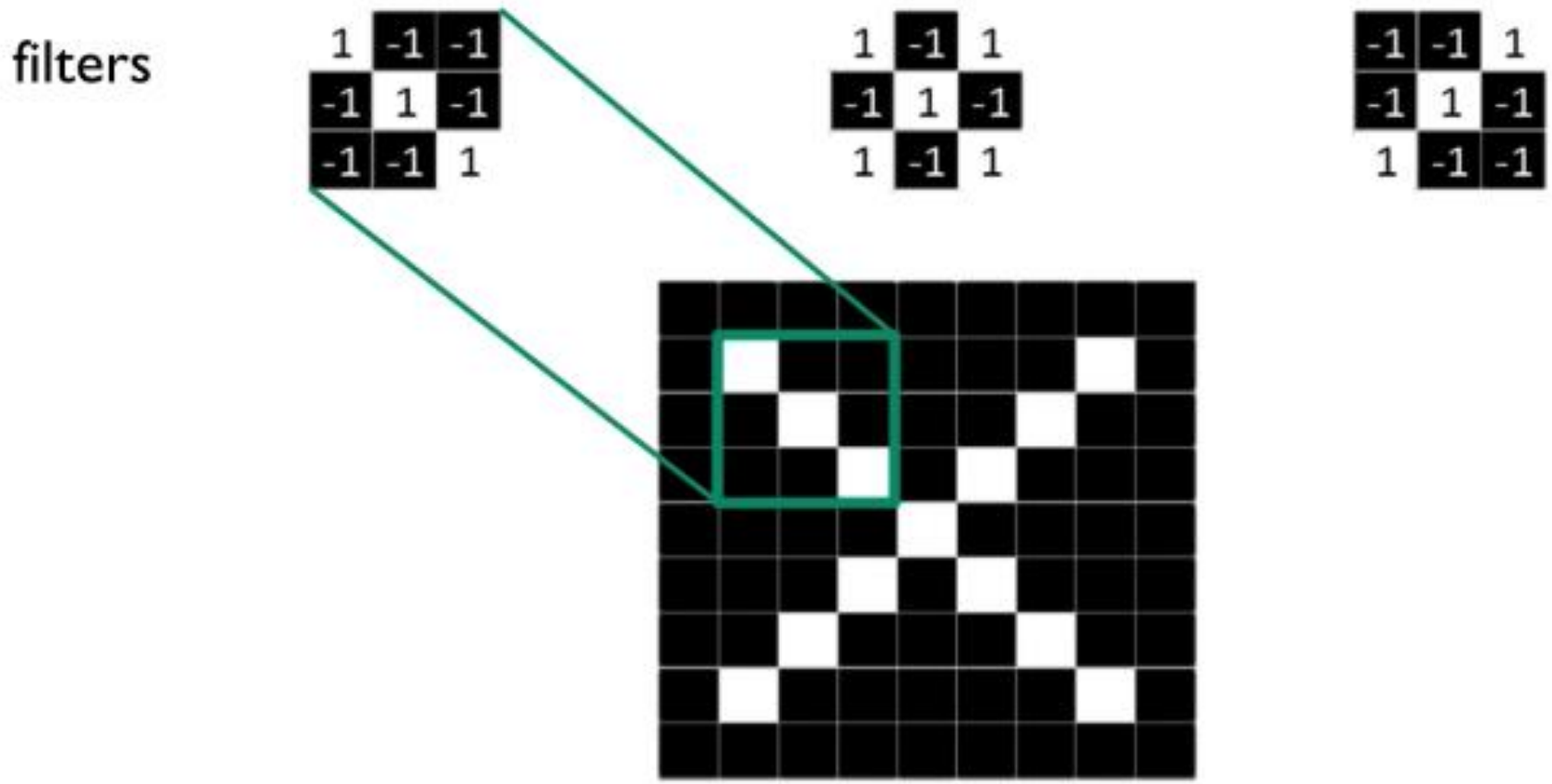
The Convolution operation



The Convolution operation



Filters to detect Features of “X”



Examples od edge detection filters

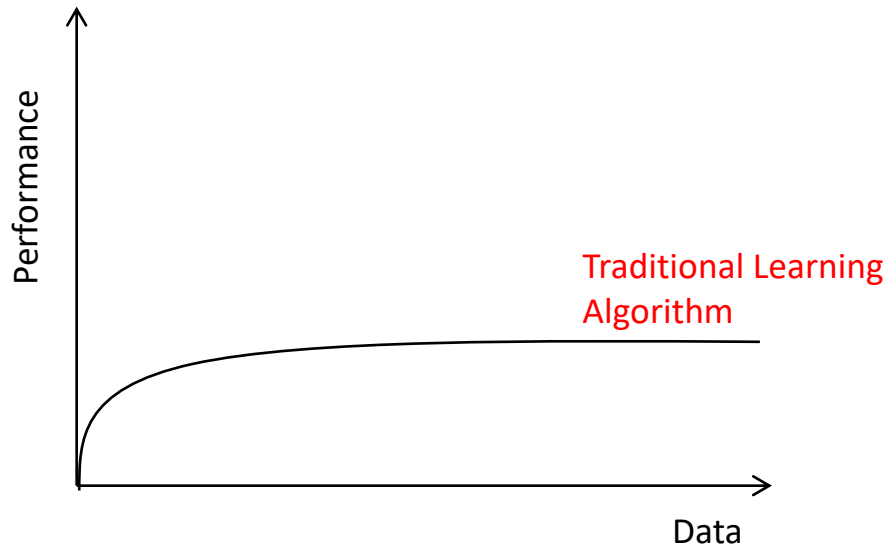
Original

Sharpen

Edge Detect

"Strong" Edge Detect

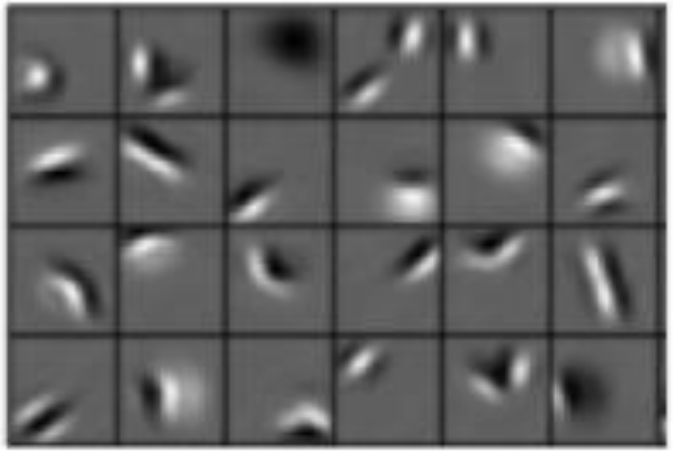
Performance of computer vision algorithms



- Where next? Better classifiers? building better features?
- Difficult to hand-engineer features -> What about learning them?

Features Hierarchy

Low level features



Edges, dark spots

Mid level features

Eyes, ears, nose

High level features

Facial structure

Learning Visual Features

Learning Feature Hierarchy

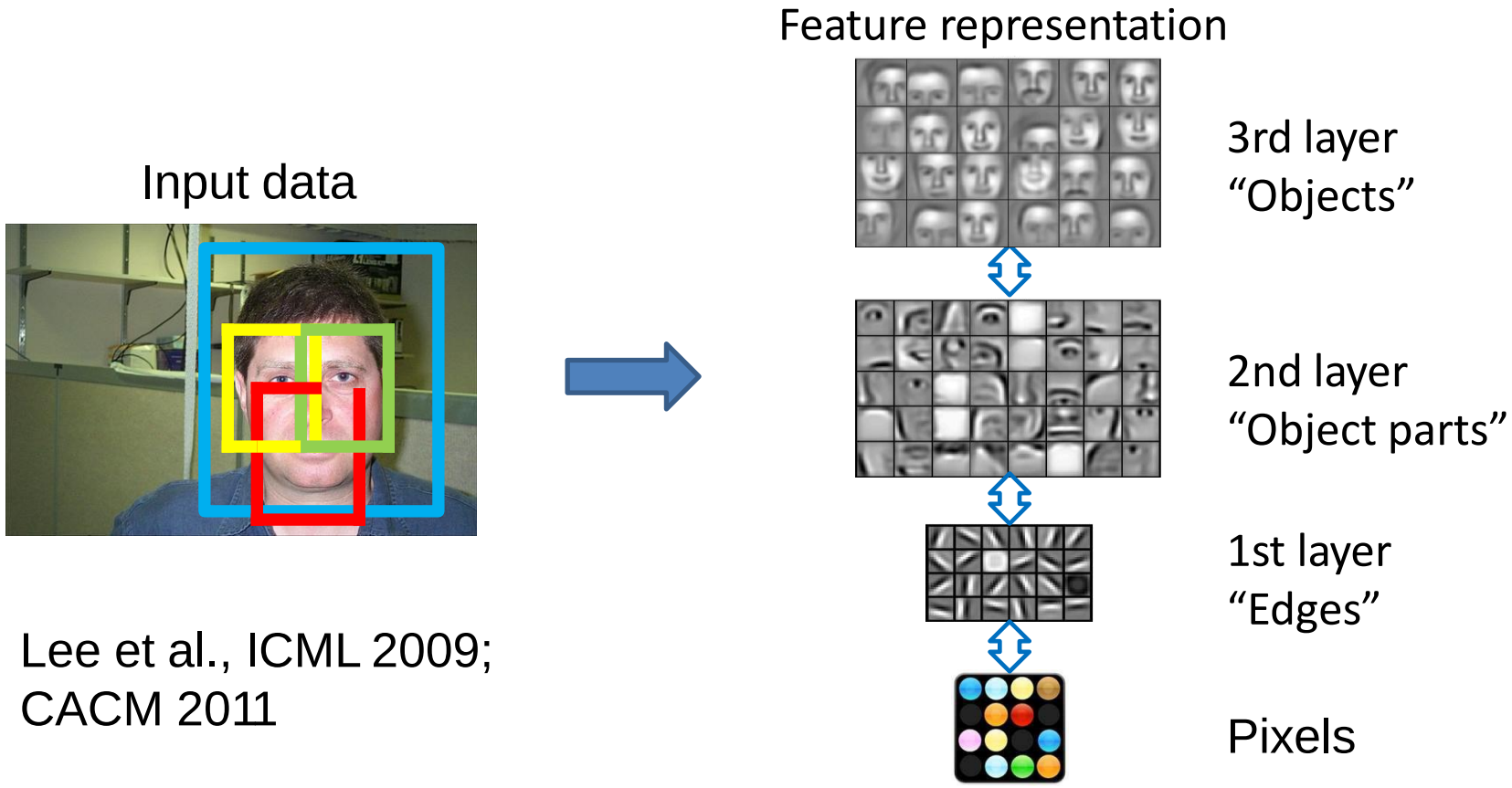
- Learn hierarchy
- All the way from pixels -> classifier
- One layer extracts features from output of previous layer

- **Train all layers jointly**

Slide: R. Fergus

Learning Feature Hierarchy

1. Learn **useful higher-level features** from images



Lee et al., ICML 2009;
CACM 2011

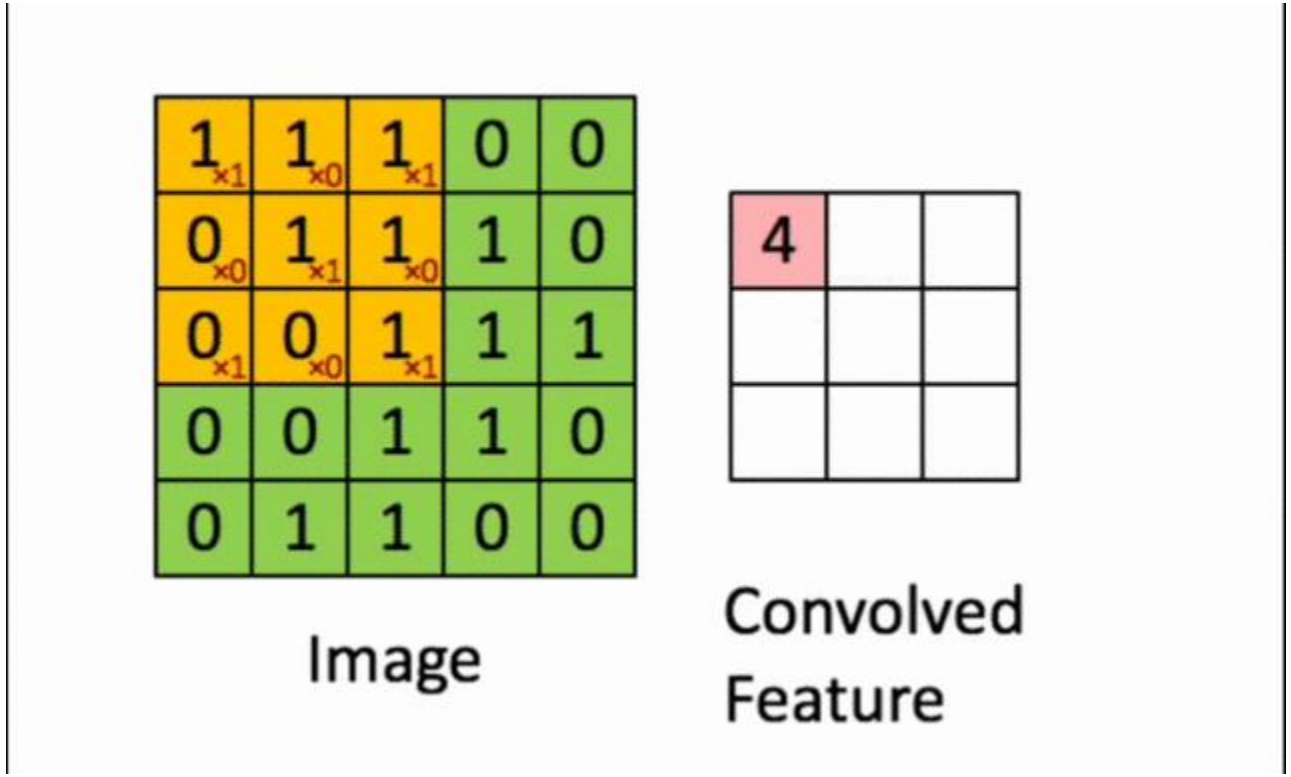
2. **Fill in representation gap** in recognition

Approaches to learning

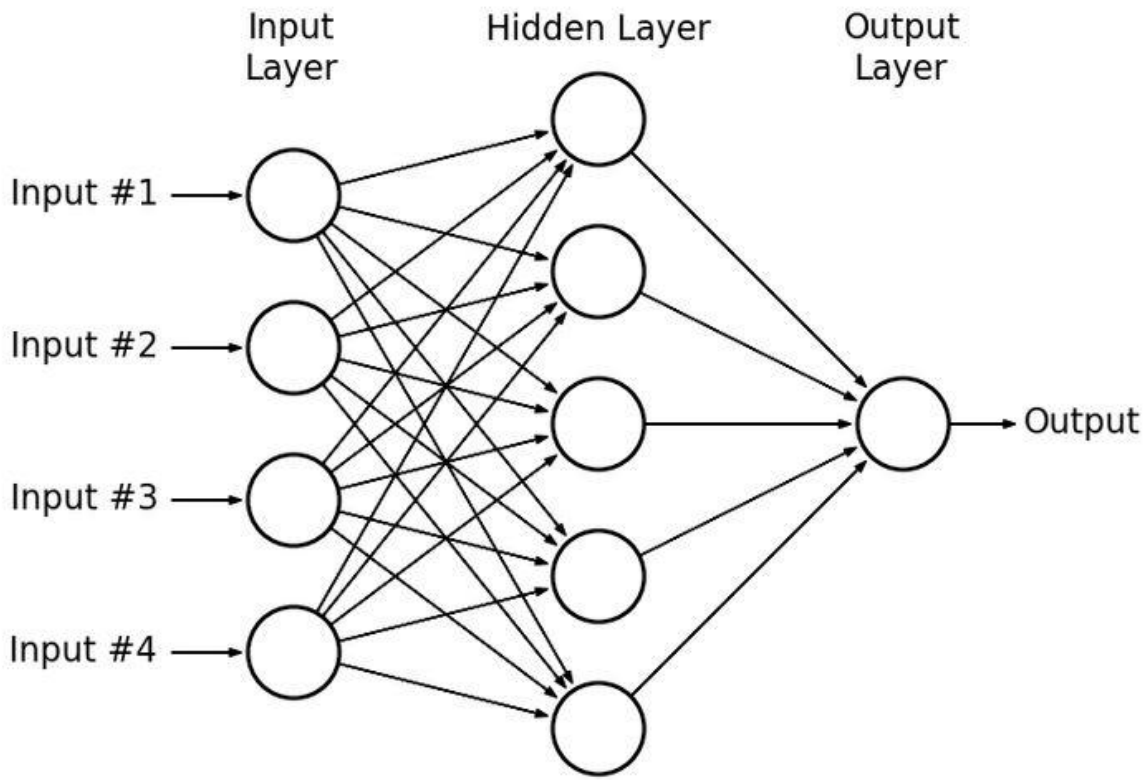
- Supervised Learning
  - End-to-end learning of deep architectures (e.g., deep neural networks) with back-propagation
  - Works well when the amounts of labels is large
  - Structure of the model is important (e.g. convolutional structure)

- Unsupervised Learning
  - Learn statistical structure or dependencies of the data from unlabeled data
  - Layer-wise training
  - Useful when the amount of labels is not large

How we detect features - image convolution



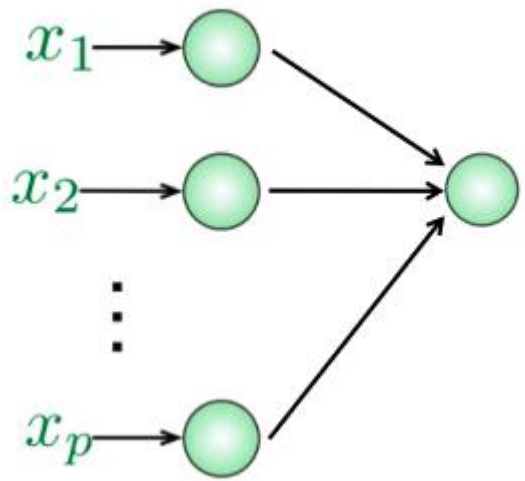
How we learn end to end? Neural Networks



Limitations of Fully Connected NNs for object detection

Input:

- 2D image
- Vector of pixel values

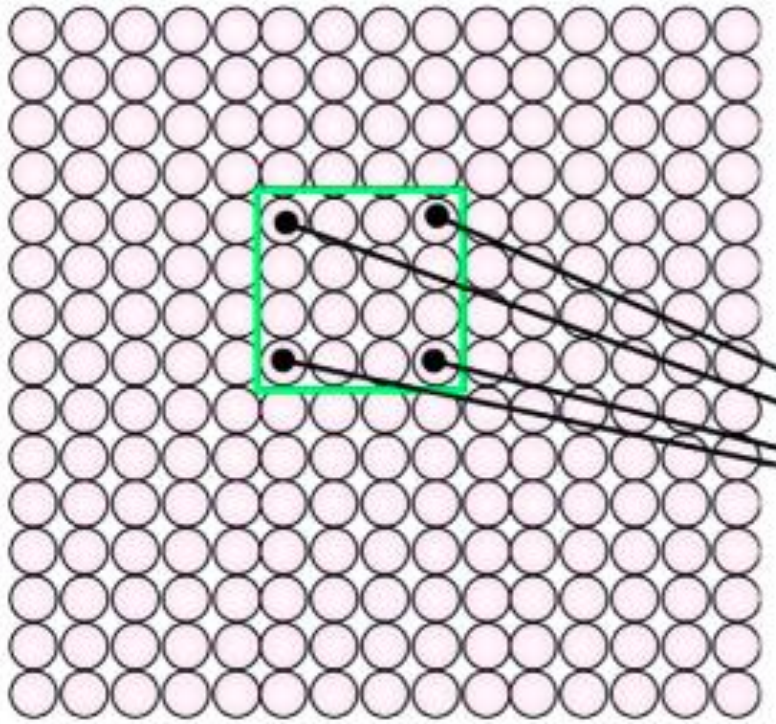


Fully Connected:

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

Using spatial structure

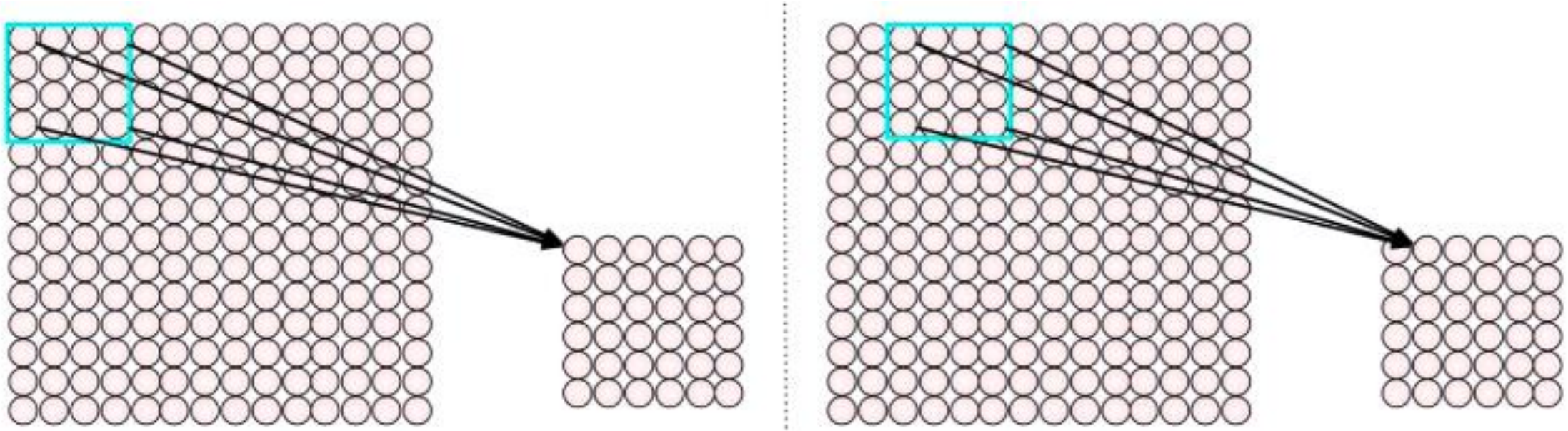
Input: 2D image.
Array of pixel values



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

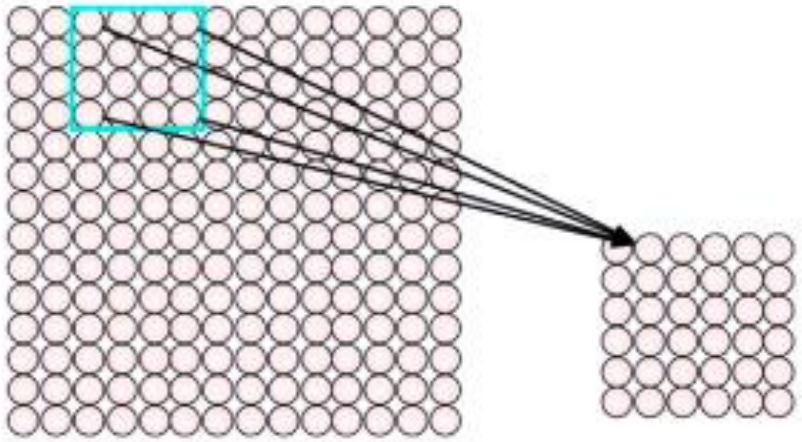
Neuron connected to region of input. Only "sees" these values.

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?*

Learn the weights through convolution & back-propagation



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

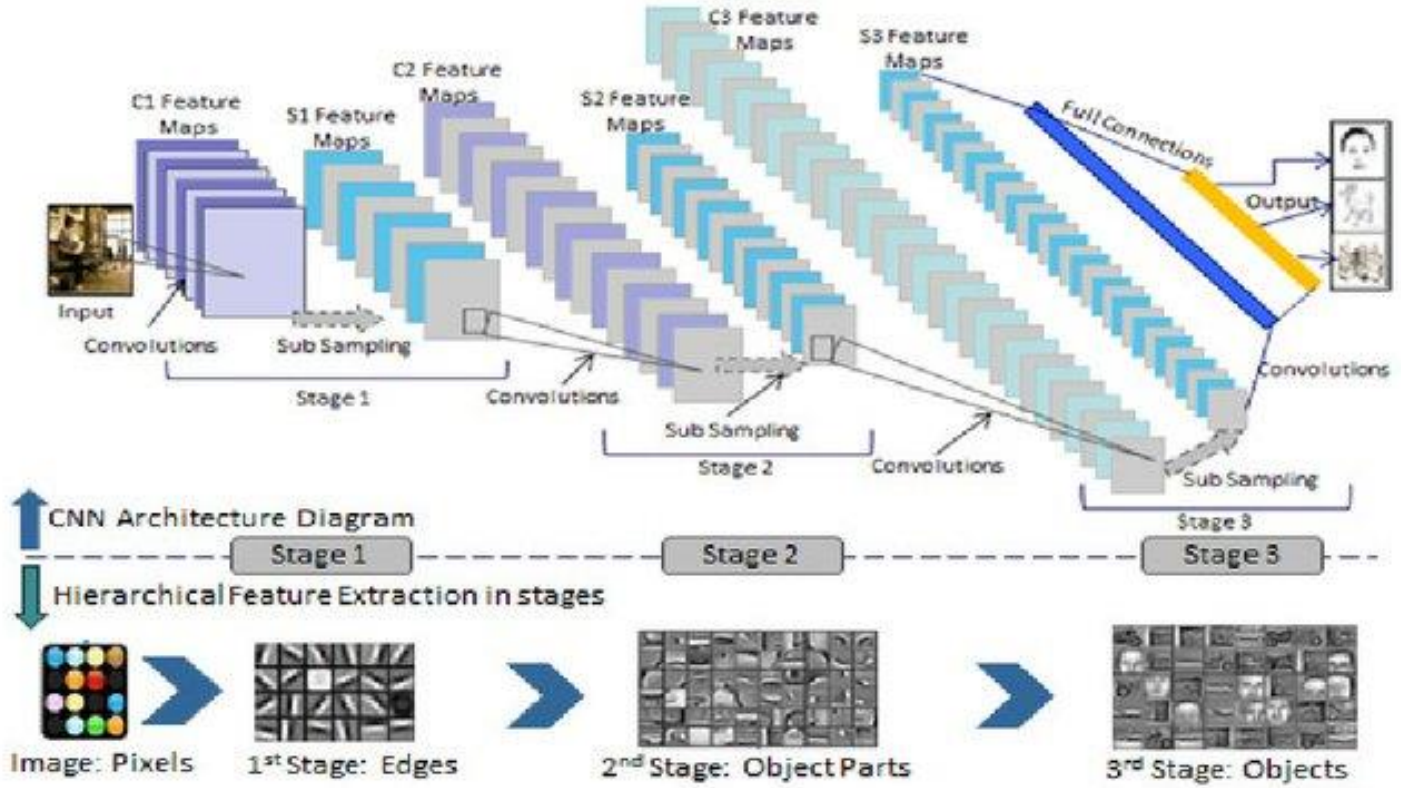
This "patchy" operation is **convolution**

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

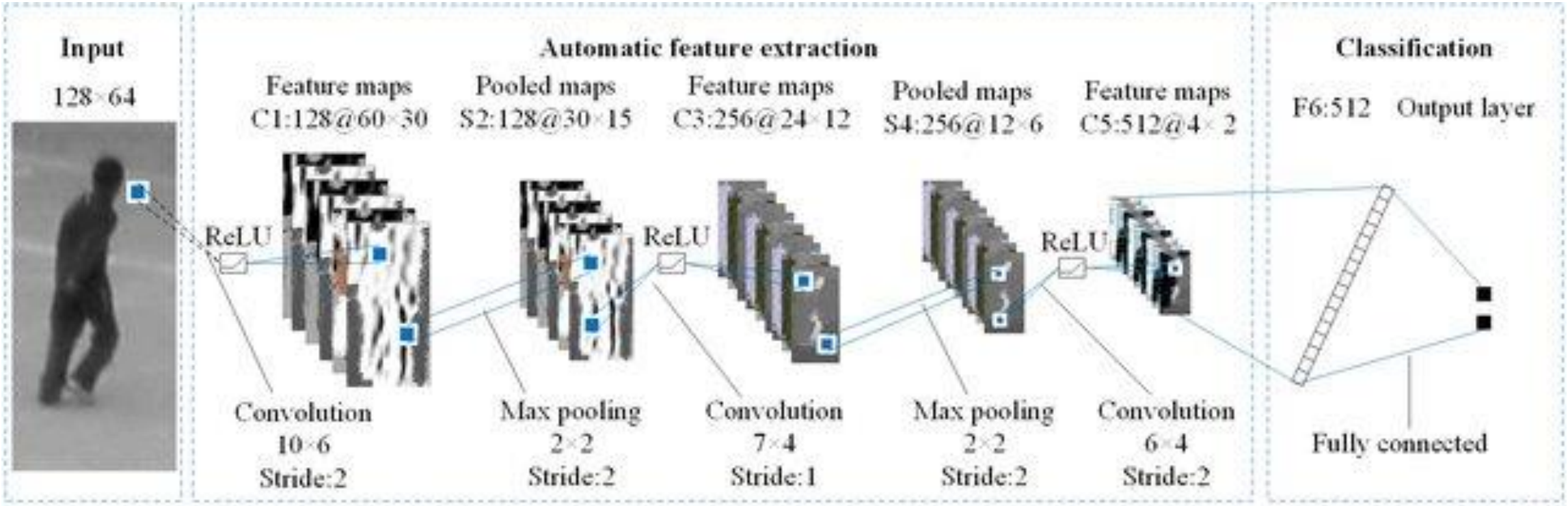
2) Use **multiple filters** to extract different features

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

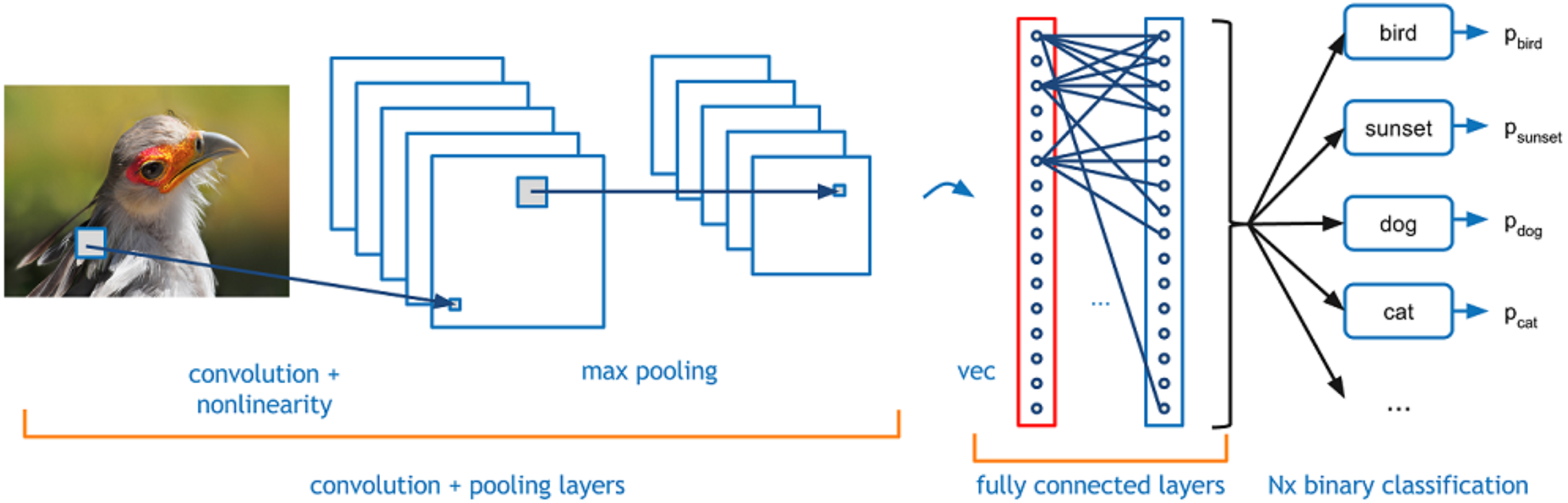
CNN and Features Hierarchy



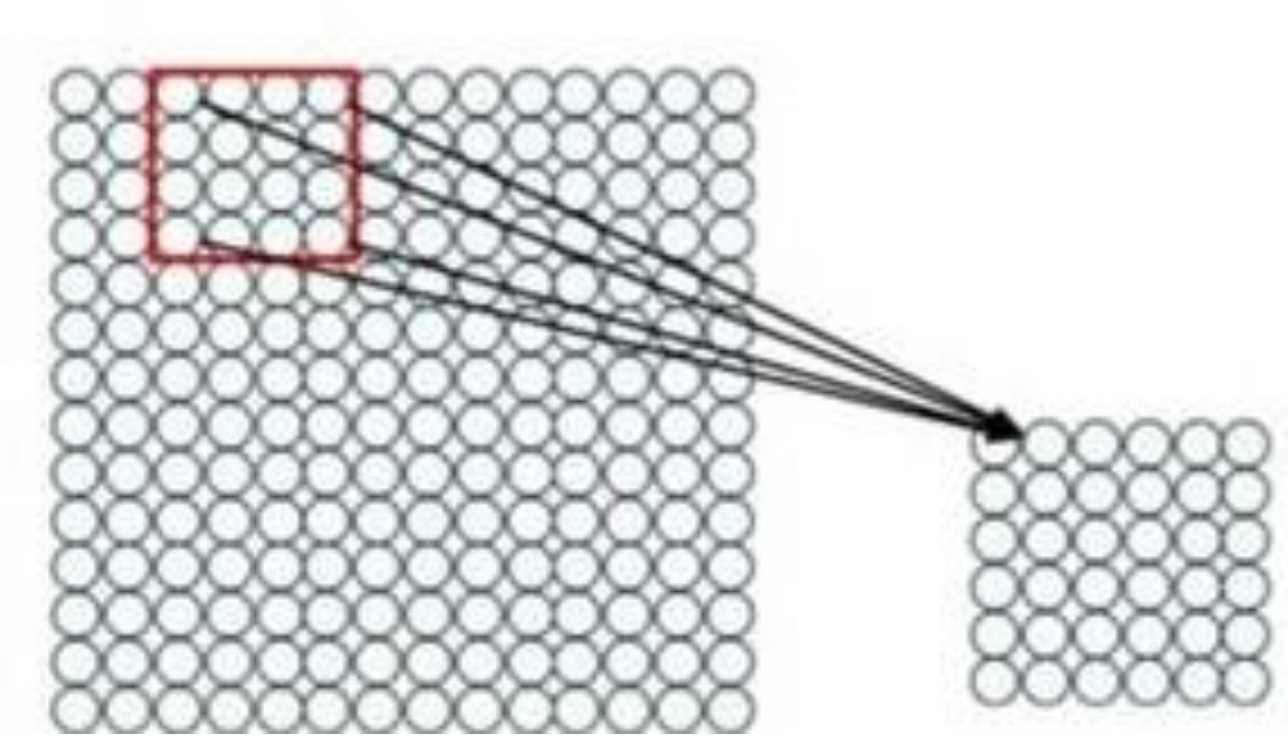
Automatic Feature extraction and Classification



CNNs for classification



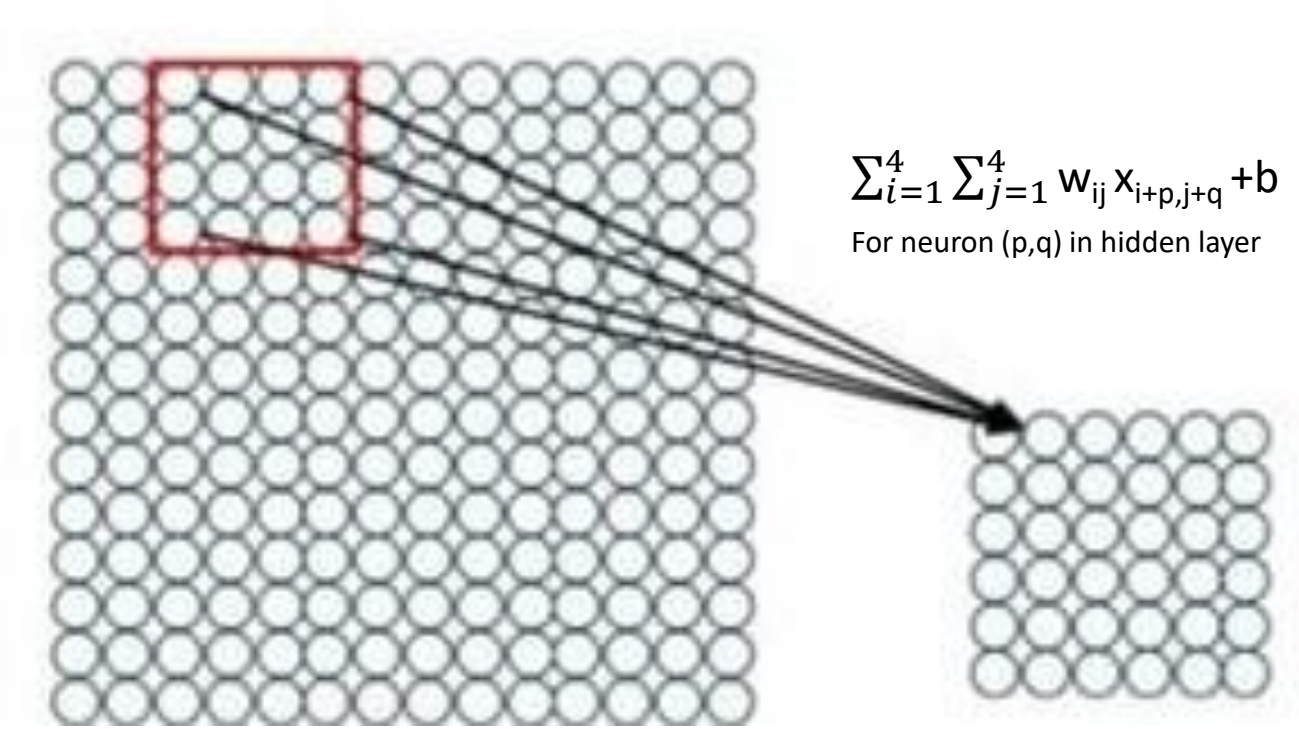
Convolutional Layers: Local Connectivity



- **For a neuron in hidden layer**:
- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity

4x4 filter: matrix of weights w_{ij}



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

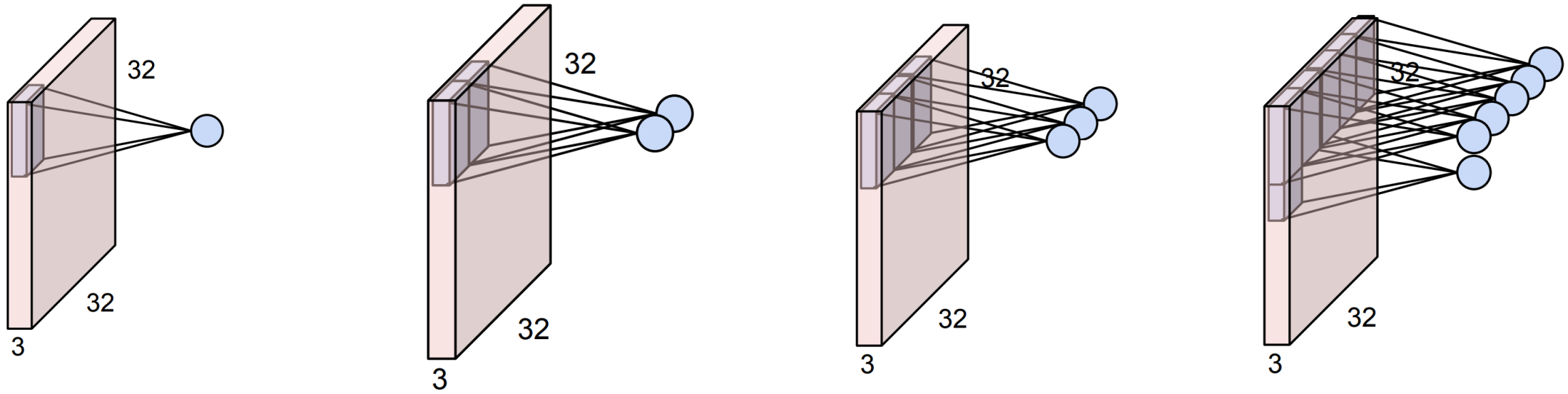
For neuron (p,q) in hidden layer

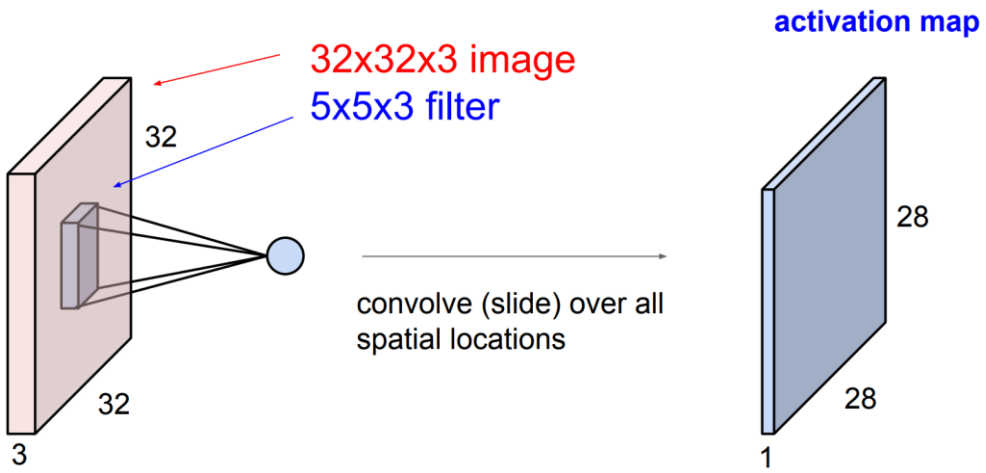
For a neuron in hidden layer:

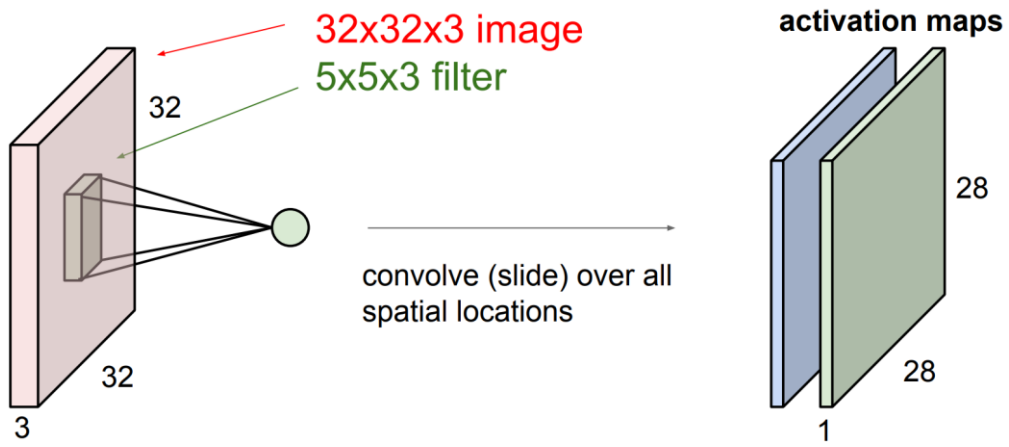
- Take inputs from patch
- Compute weighted sum
- Apply bias

1) Applying window for weights
2) Computing linear combinations
3) Activating with non-linear function

Convolutional layer

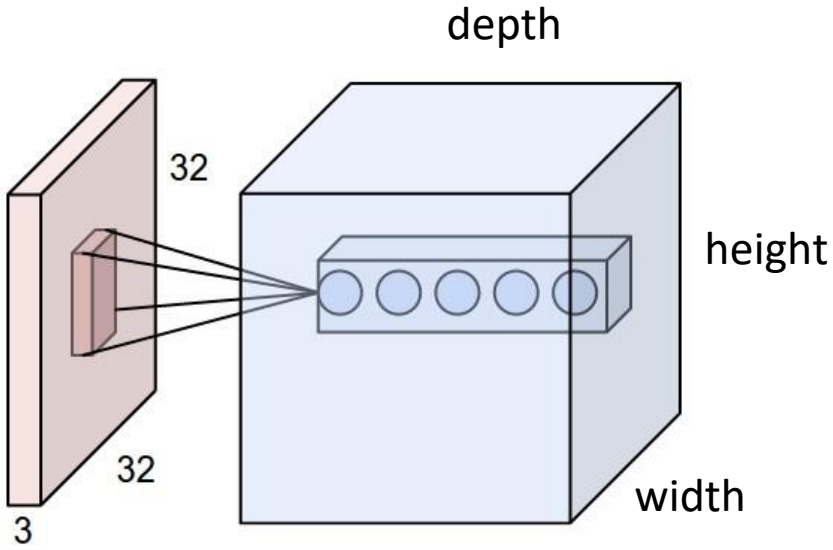






Convolutional layer – two filters

CNNs: Spatial arrangement of output volume



Layer Dimensions:

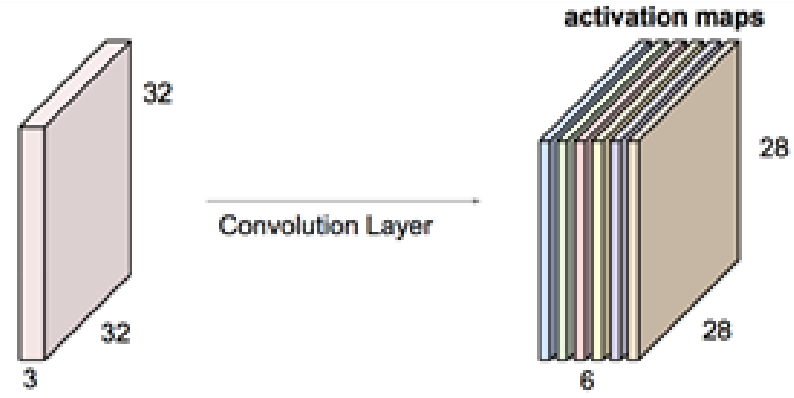
h *x* w *x* d

h, w are spatial dimensions

d(depth) = number of filters

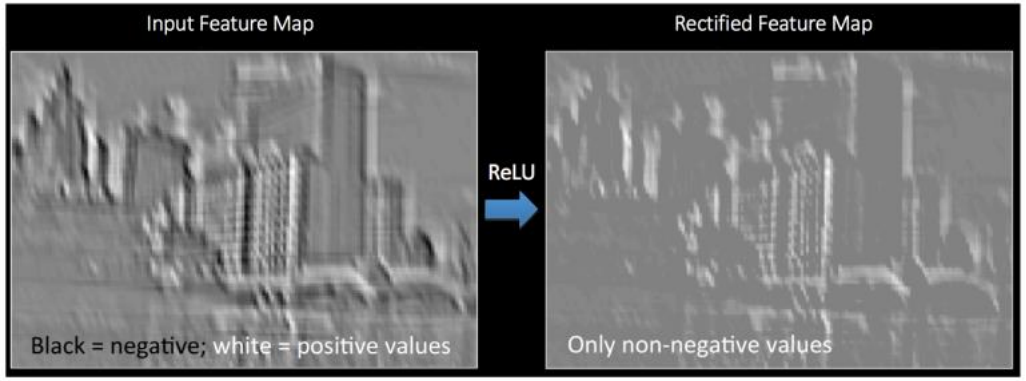
Stride: Filter step size

Receptive field: Locations in input image that a node is path connected to

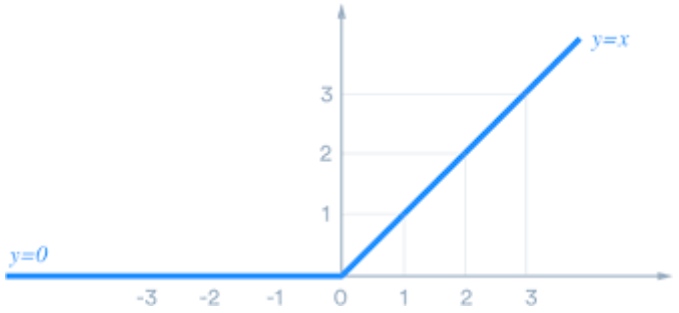


Introducing non-linearity - ReLU

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.

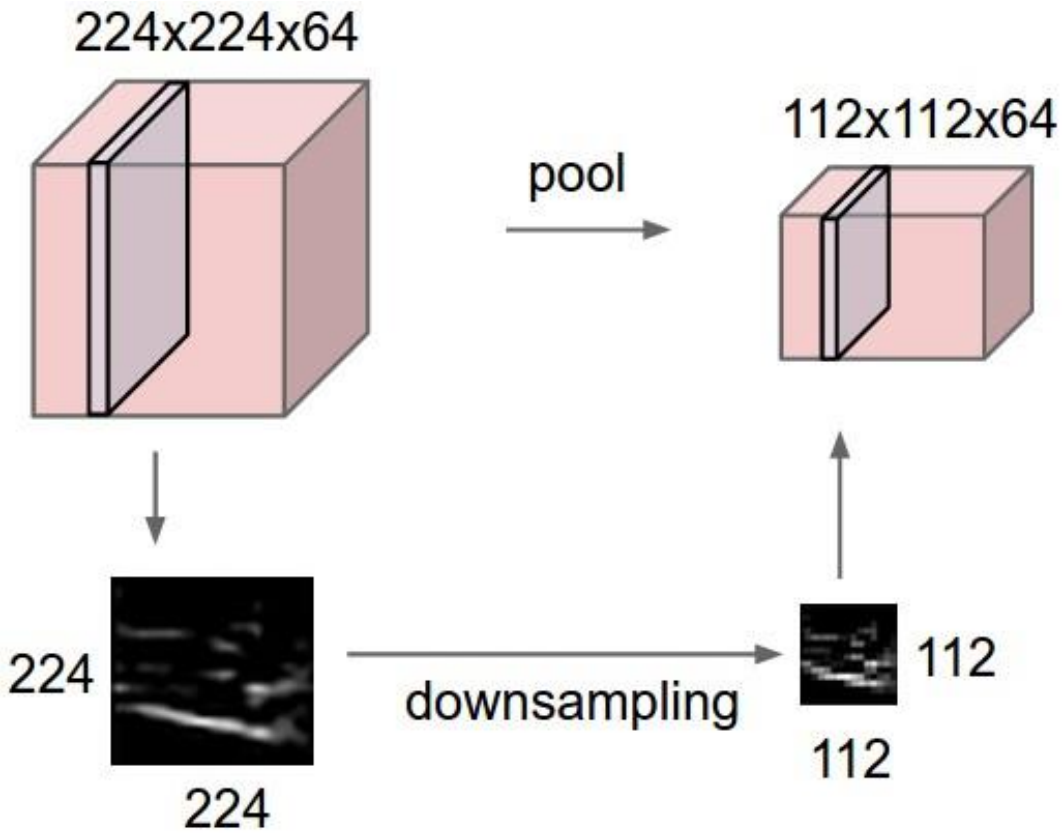


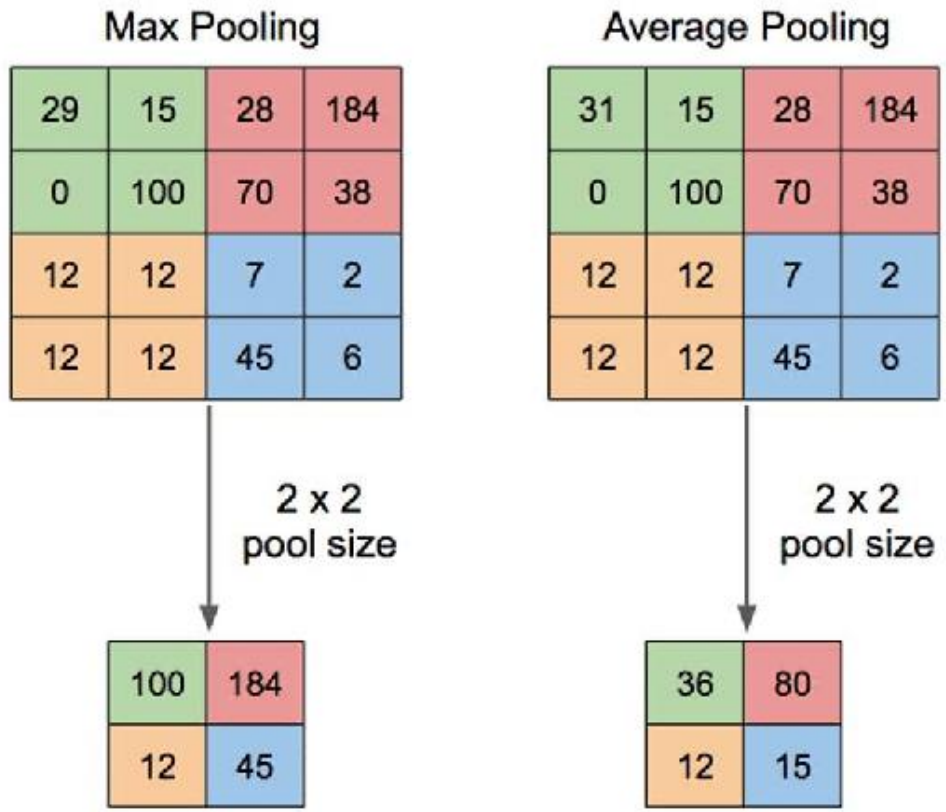
Rectified Linear Unit (ReLU)



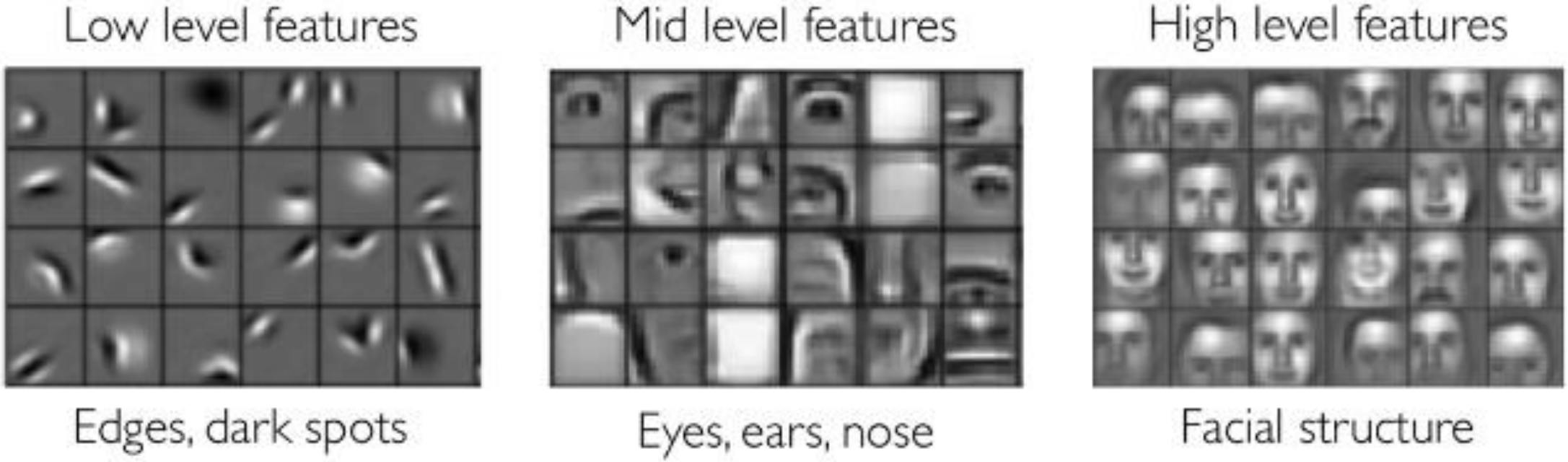
$g(z) = max(0,z)$

Pooling


224x224x64
pool
112x112x64
224
224
downsampling
112
112


Max Pooling
29 15 28 184
0 100 70 38
12 12 7 2
12 12 45 6
2 x 2
pool size
100 184
12 45
Average Pooling
31 15 28 184
0 100 70 38
12 12 7 2
12 12 45 6
2 x 2
pool size
36 80
12 15

Learning Features Hierarchy in CNNs

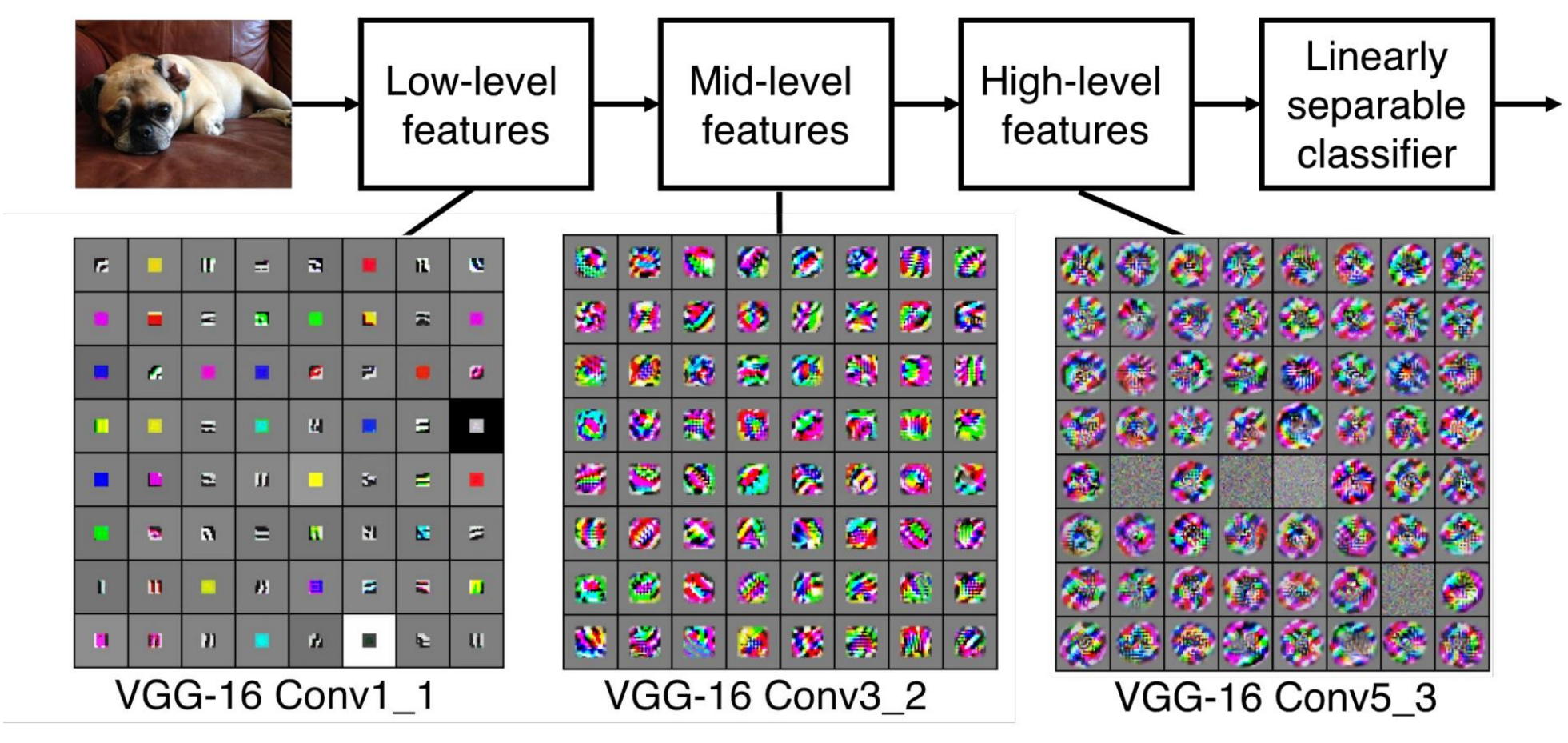


Conv Layer 1

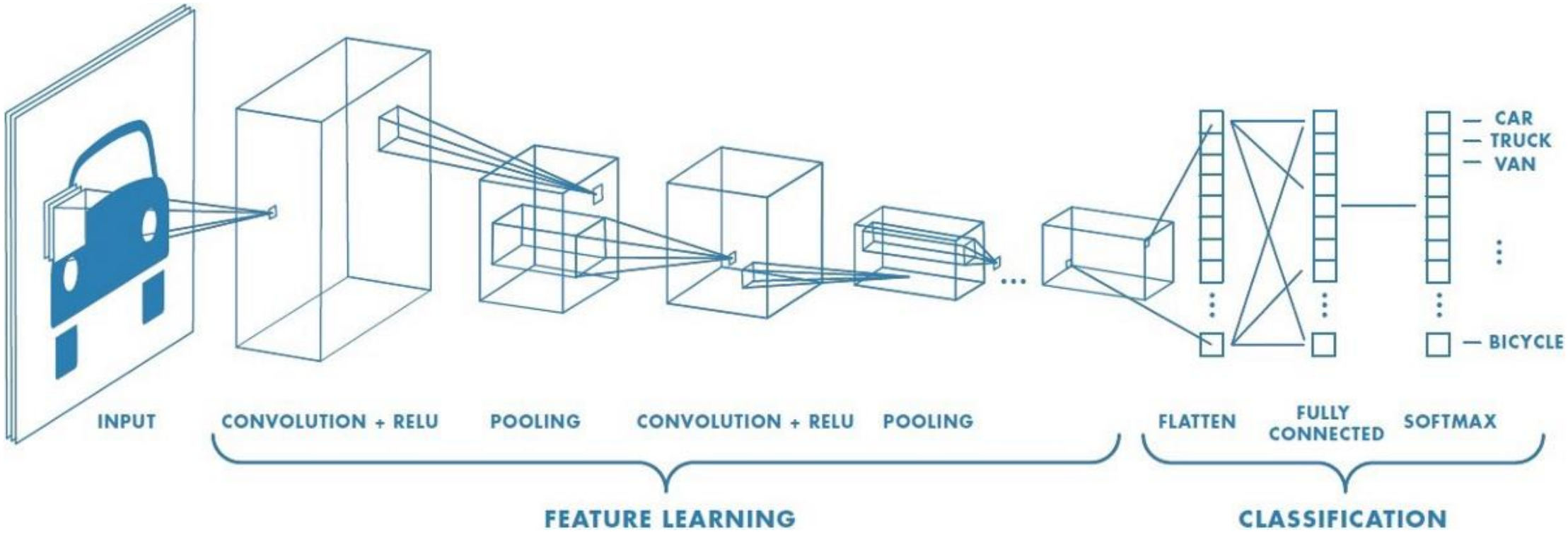
Conv Layer 2

Conv Layer 3

Learning Feature Hierarchies



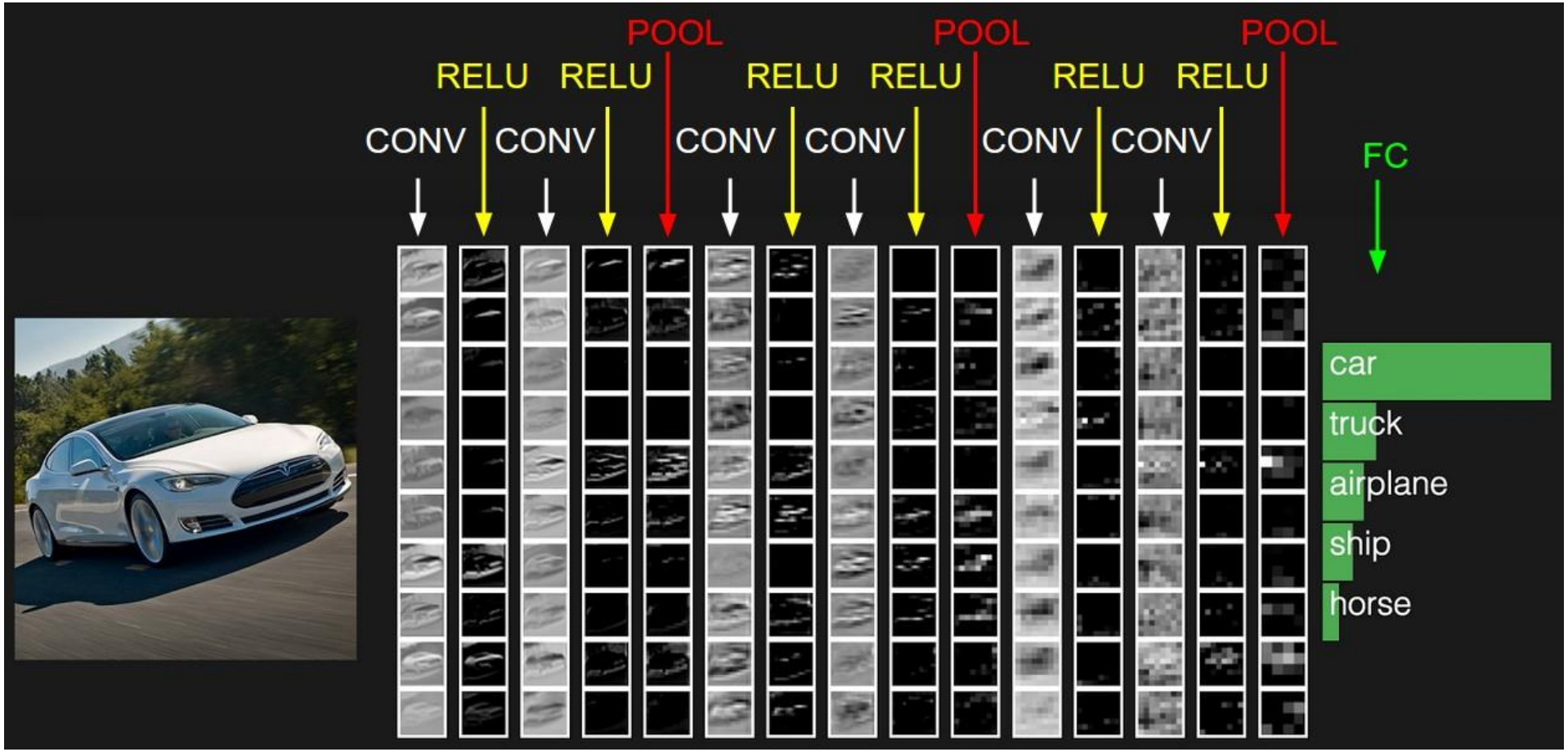
CNNs for Classification: Feature Learning & Class Probabilities



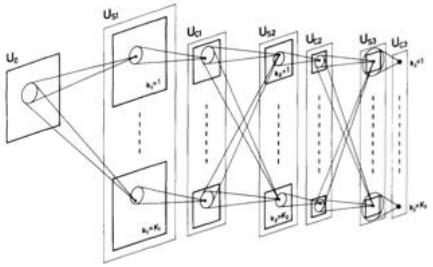
Convolution: Apply features to generate feature maps
Non-linearity: Often use ReLU
Pooling: Downsampling operation on each feature map

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

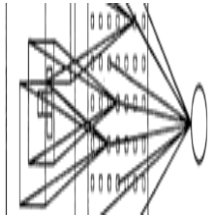
CNN – Example car classification



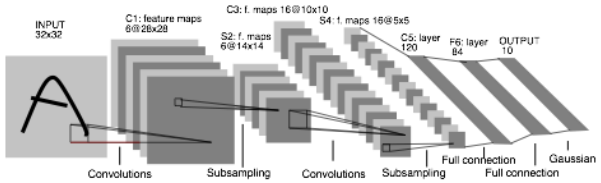
History of ConvNets



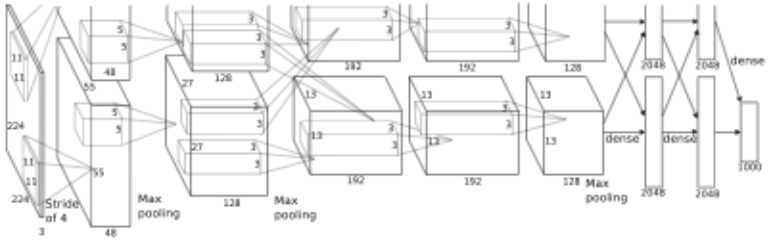
Fukushima 1980
Neocognitron



Rumelhart, Hinton, Williams 1986
“T” versus “C” problem



LeCun et al. 1989-1998
Hand-written digit reading



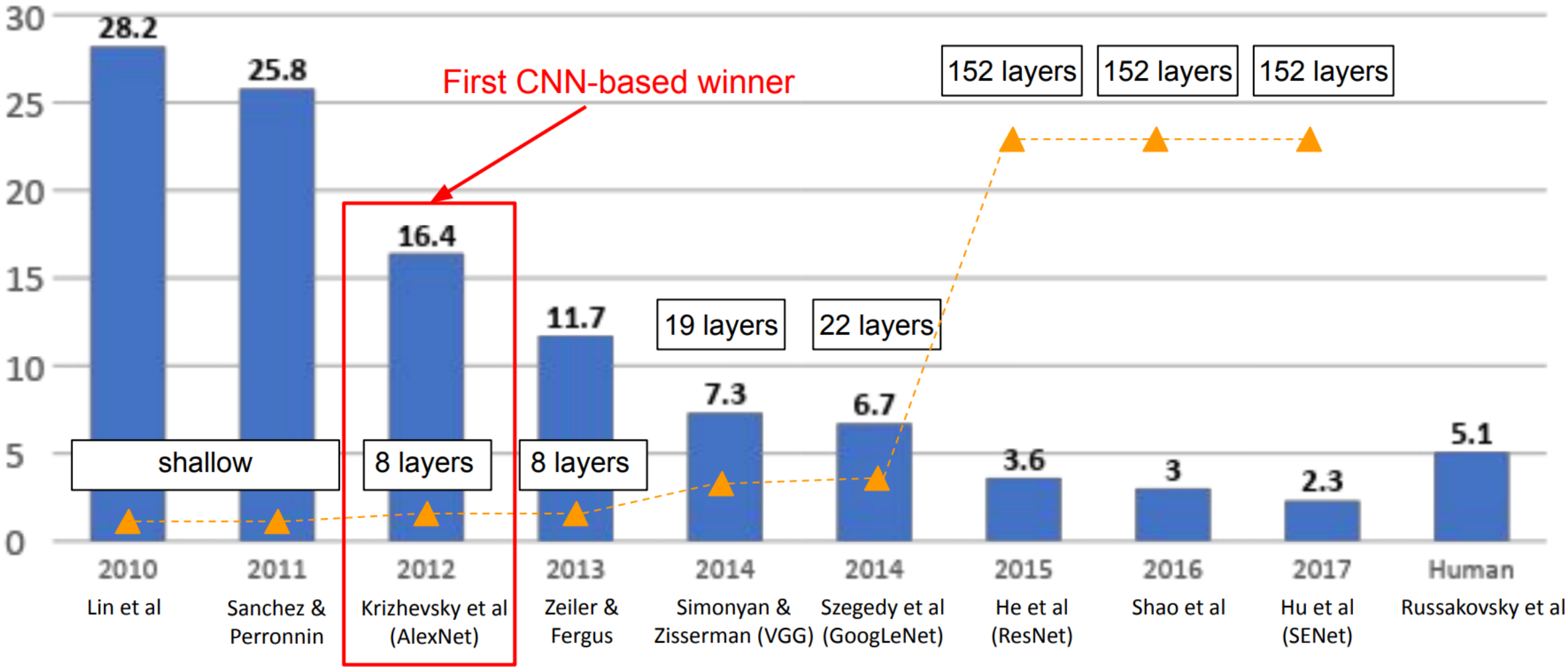
Krizhevksy, Sutskever, Hinton 2012
ImageNet classification breakthrough
“SuperVision” CNN

slide: Girshick

CNN Architectures

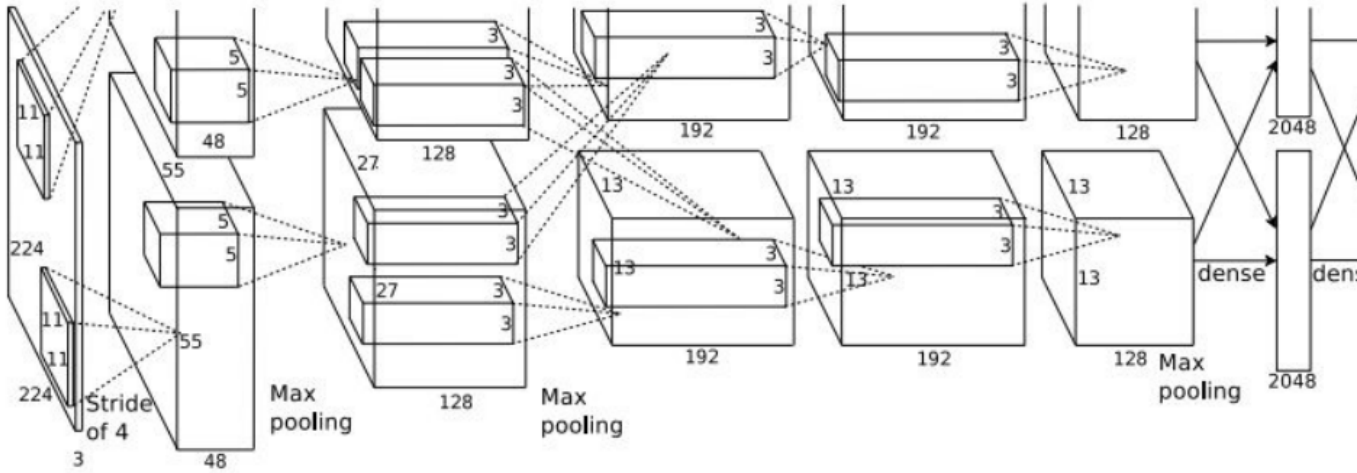
Best known	• AlexNet • VGG • GooLeNet • ResNet
Efficient nets	• MobileNets • EfficientNet • SuffleNet
Also..	• SENet • WideResNet • DenseNet • NasNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



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: VGGNet

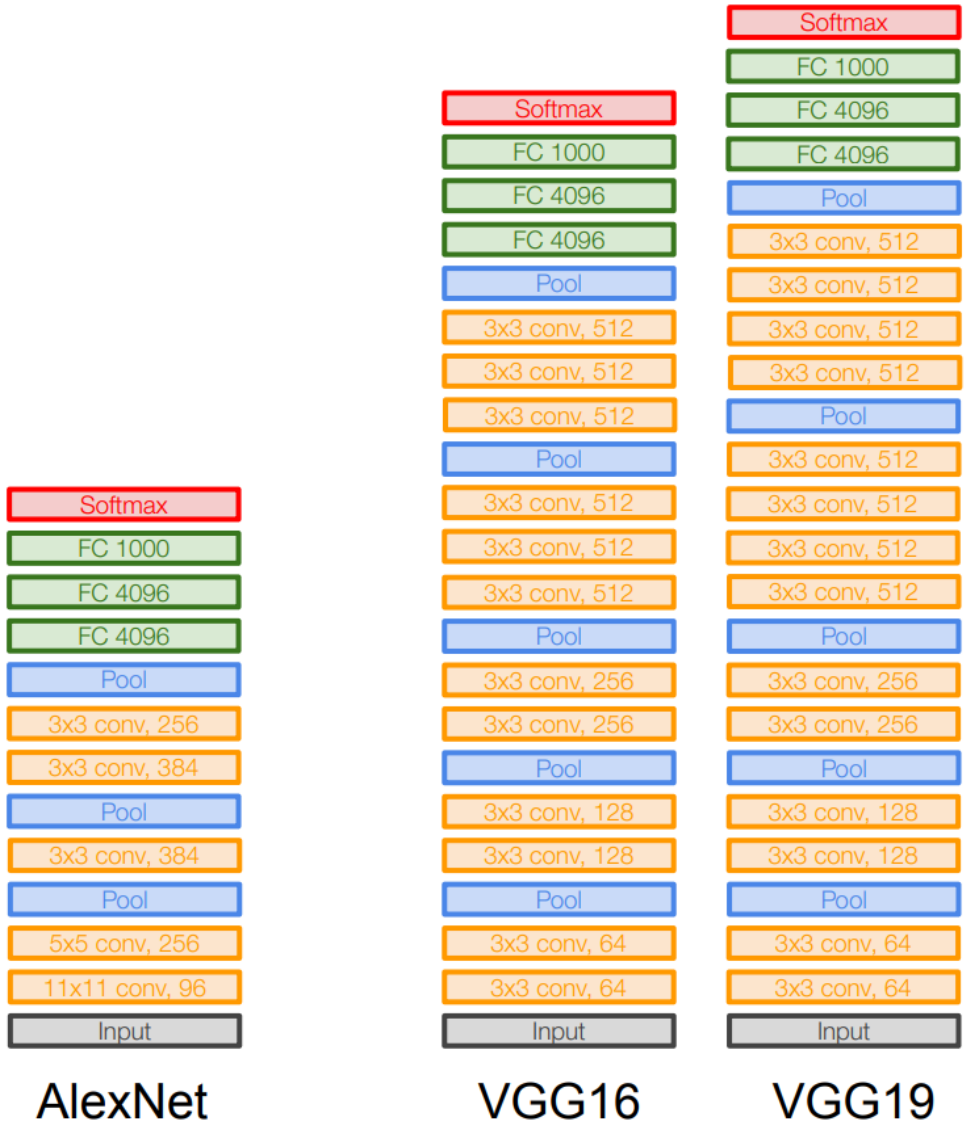
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

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

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

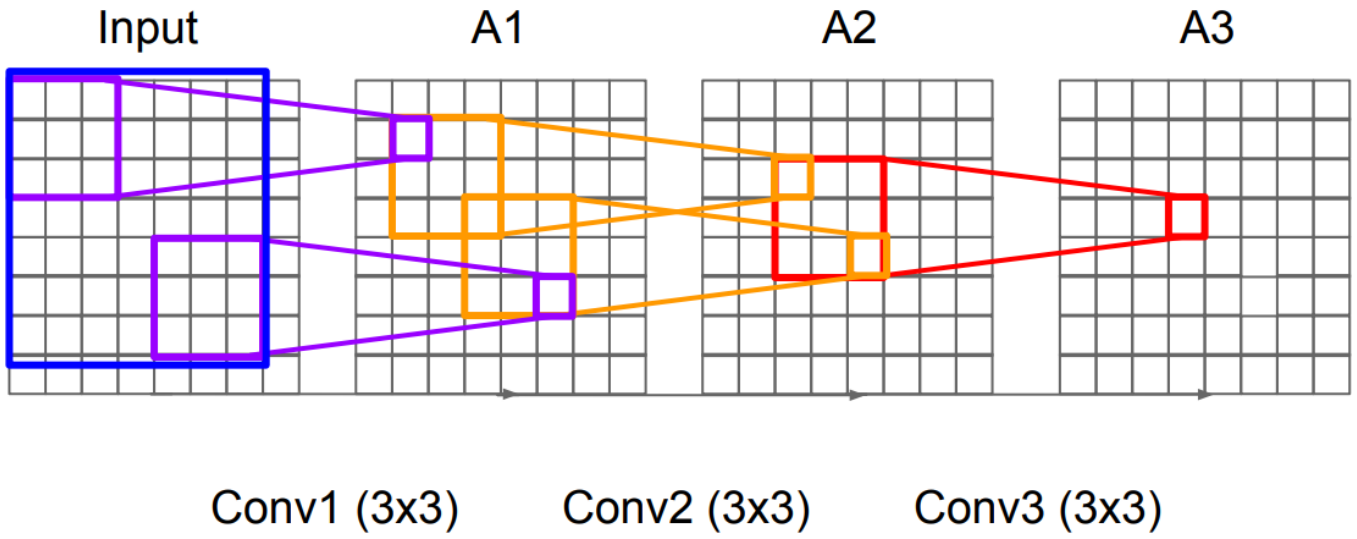
11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



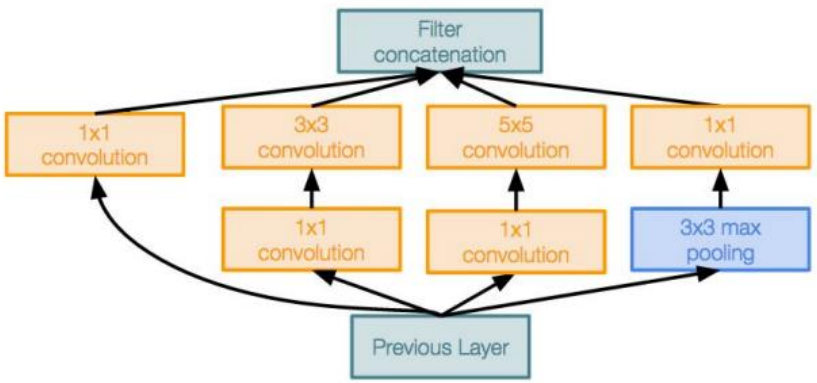
- The two have the same effective receptive field (7x7)
- A single 7x7 filter has parameters proportional to 49
- A triple 3x3 stack has parameters proportional to 3x(3x3) = 27

Case Study: GoogLeNet

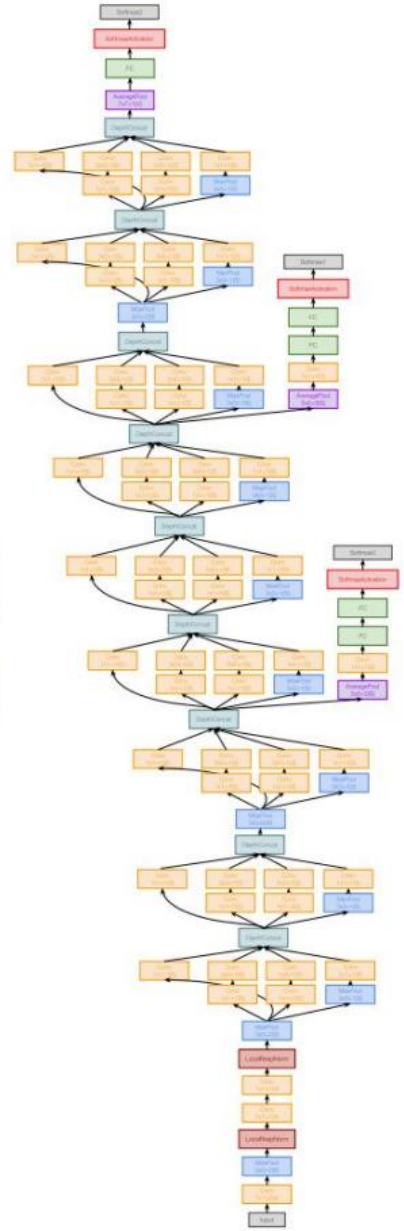
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers



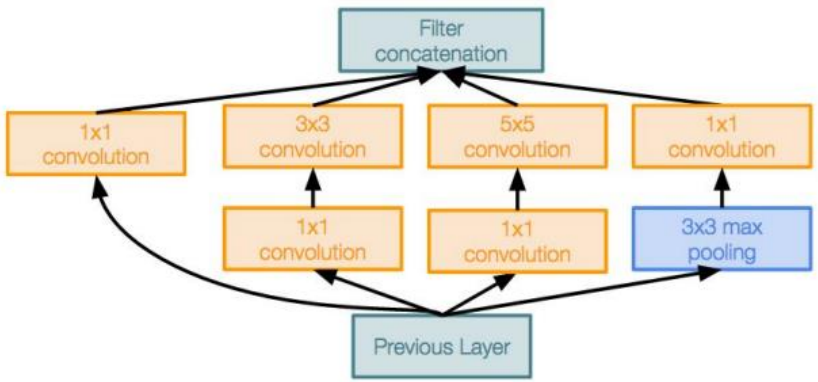
Inception module



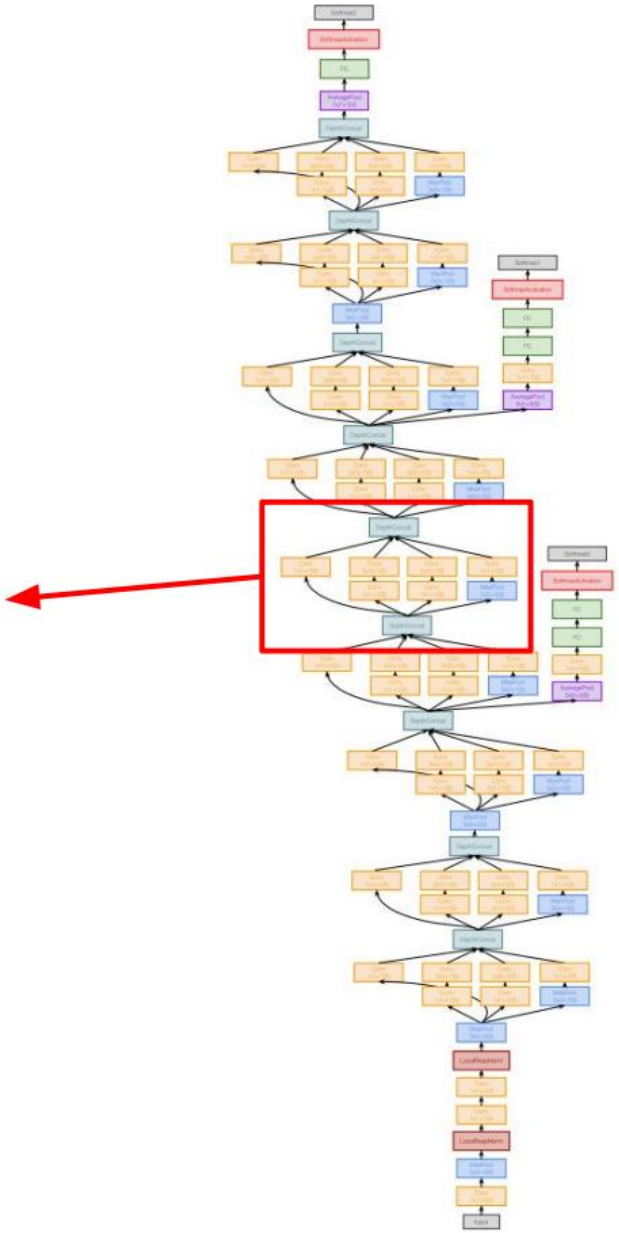
Case Study: GoogLeNet

[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

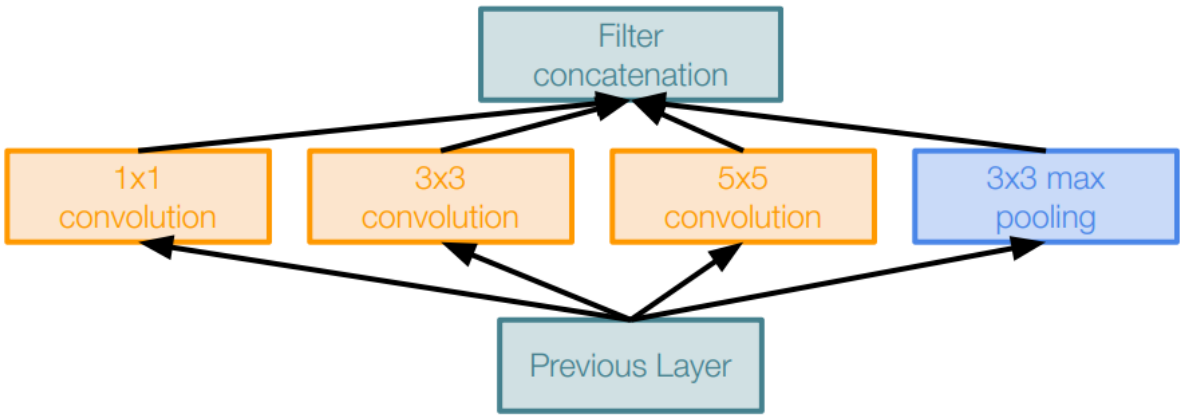


Inception module

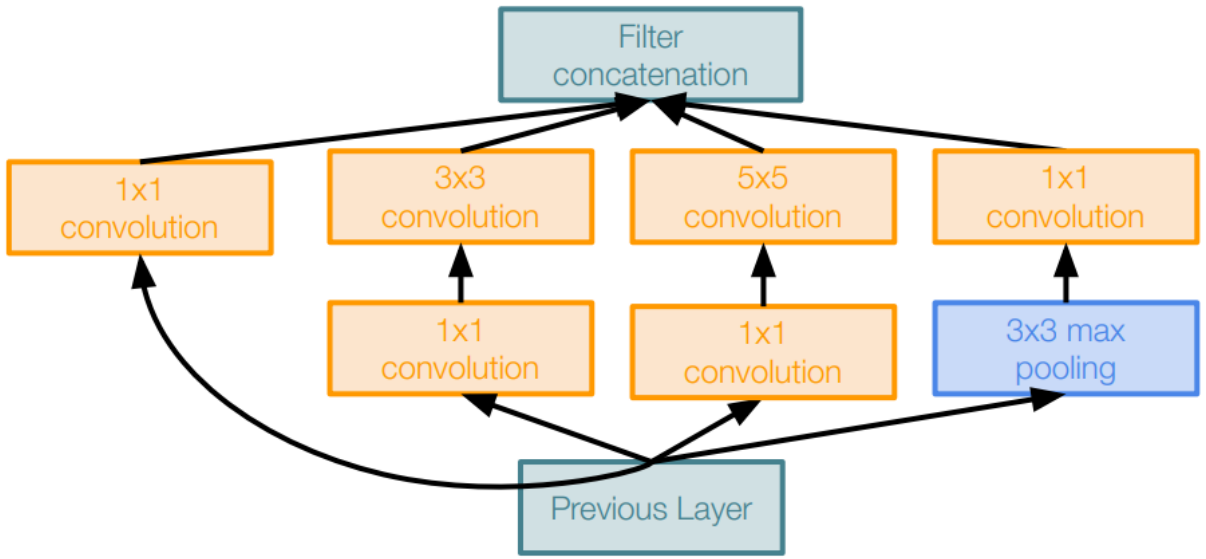


Case Study: GoogLeNet

[Szegedy et al., 2014]

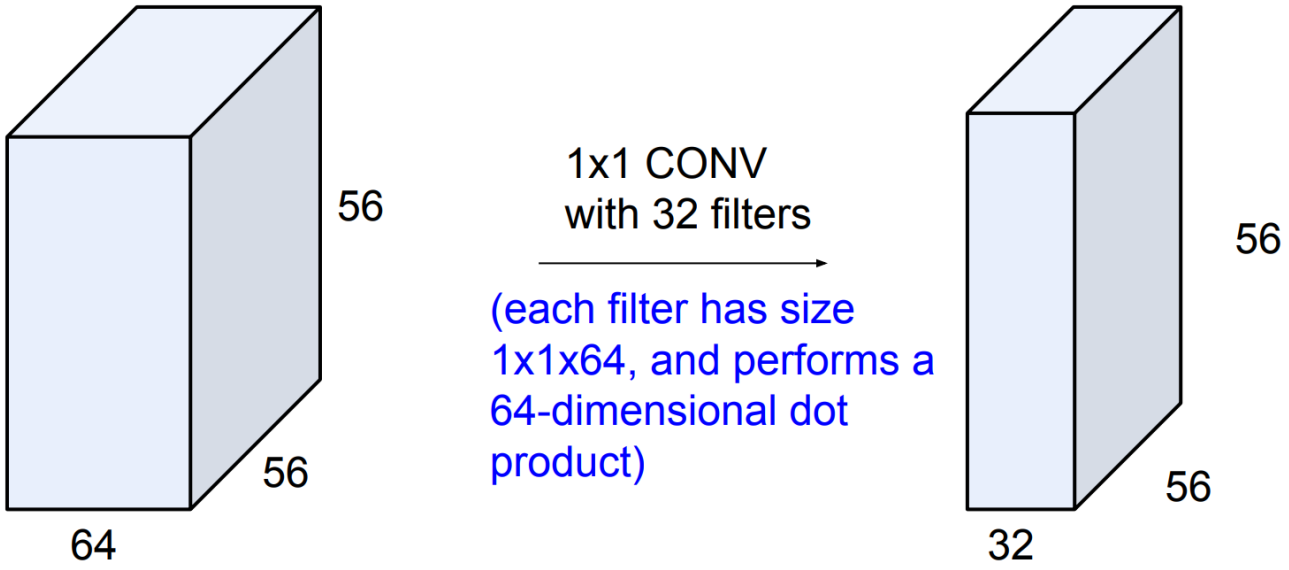


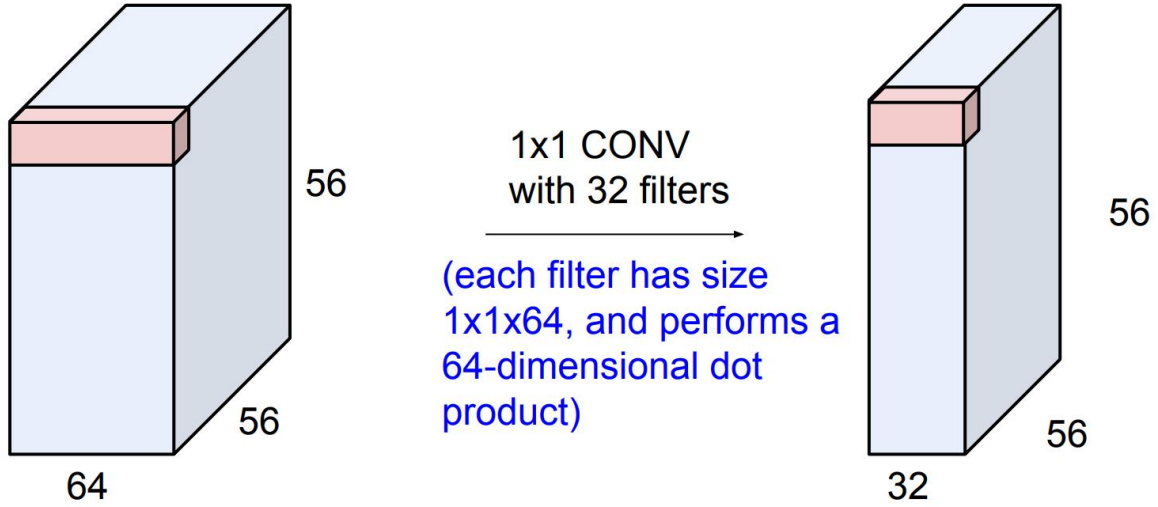
Naive Inception module



Inception module with dimension reduction

1 x 1 Convolutions

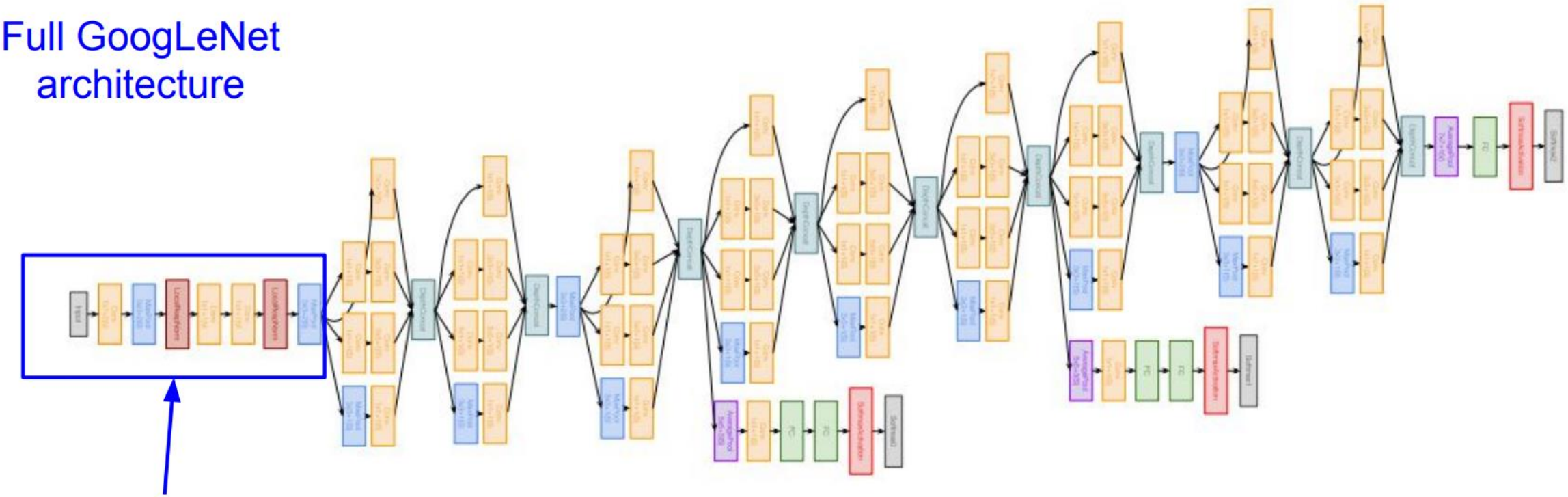




Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

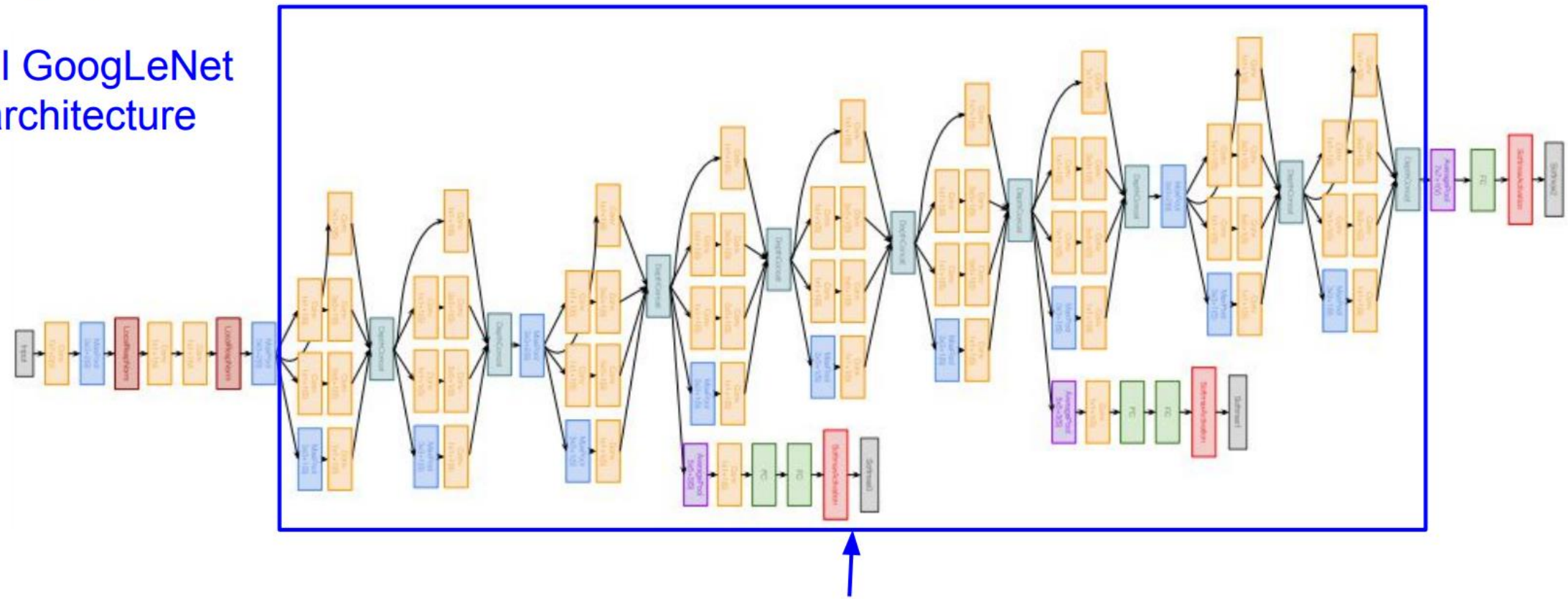


Stem Network: Conv-Pool- 2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

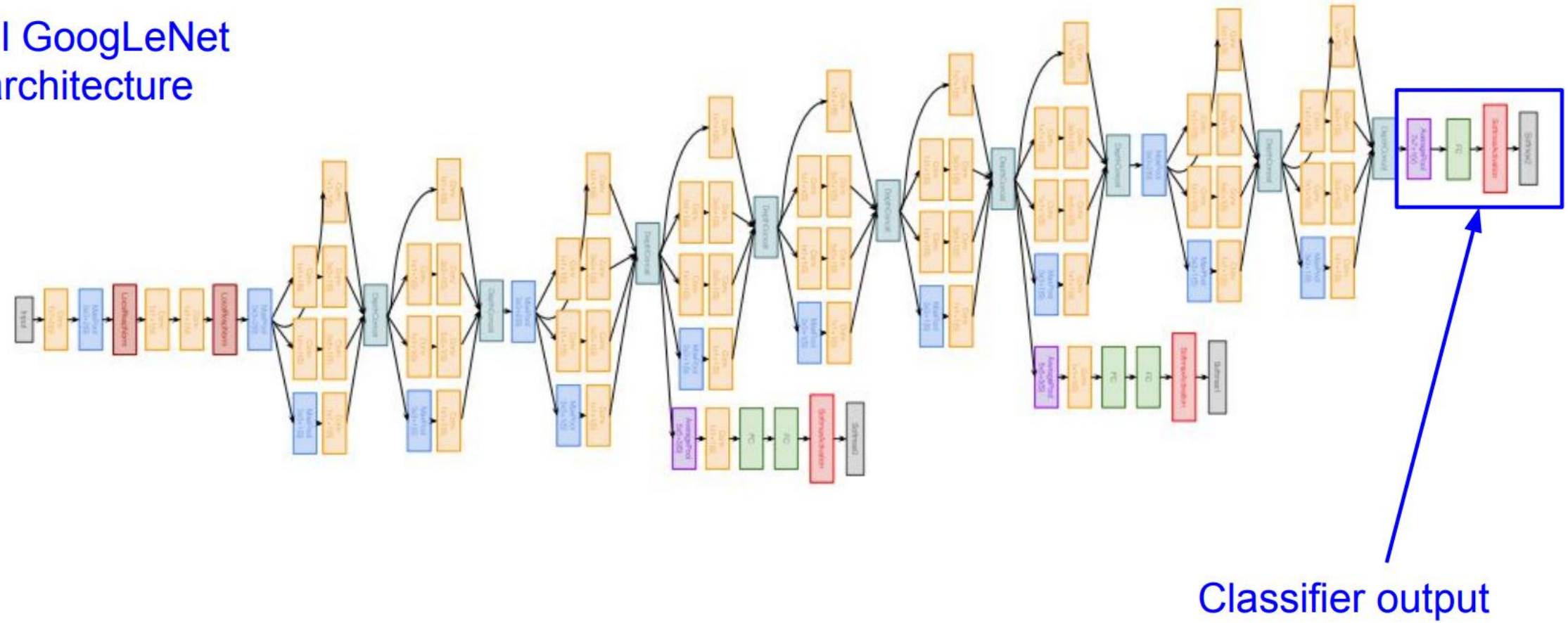


Stacked Inception Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

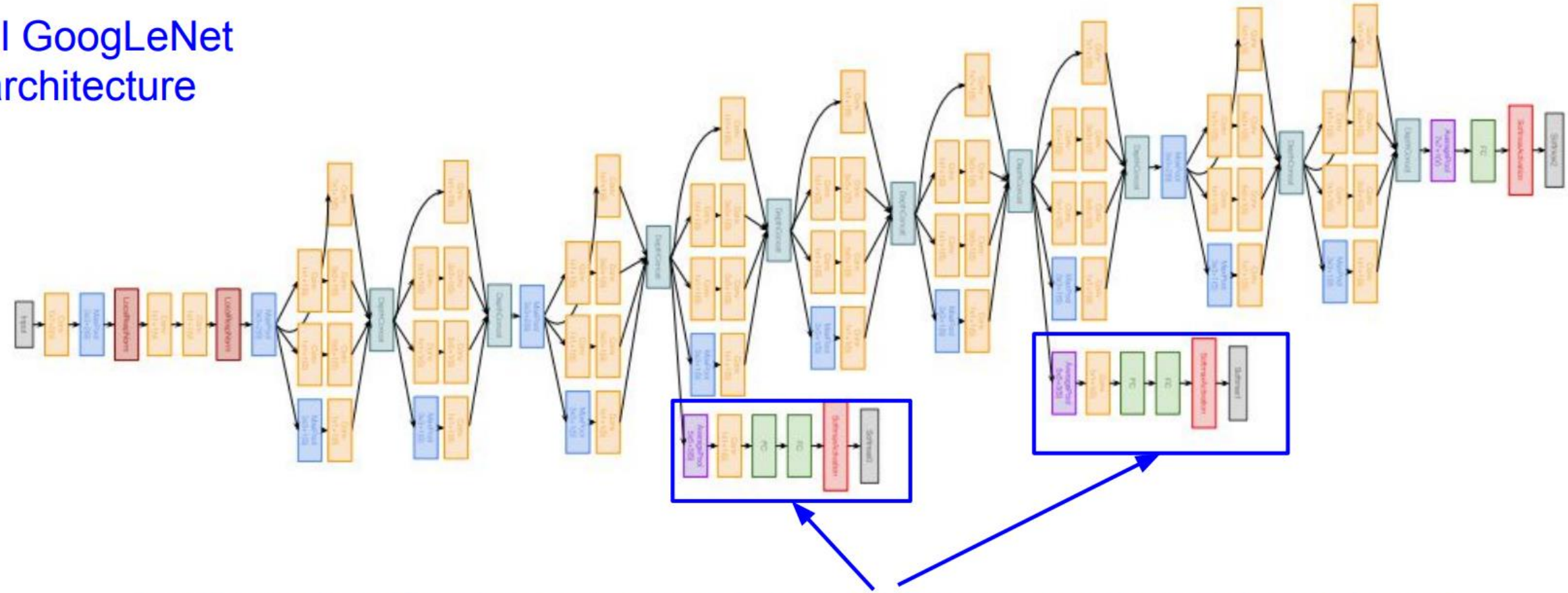


Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



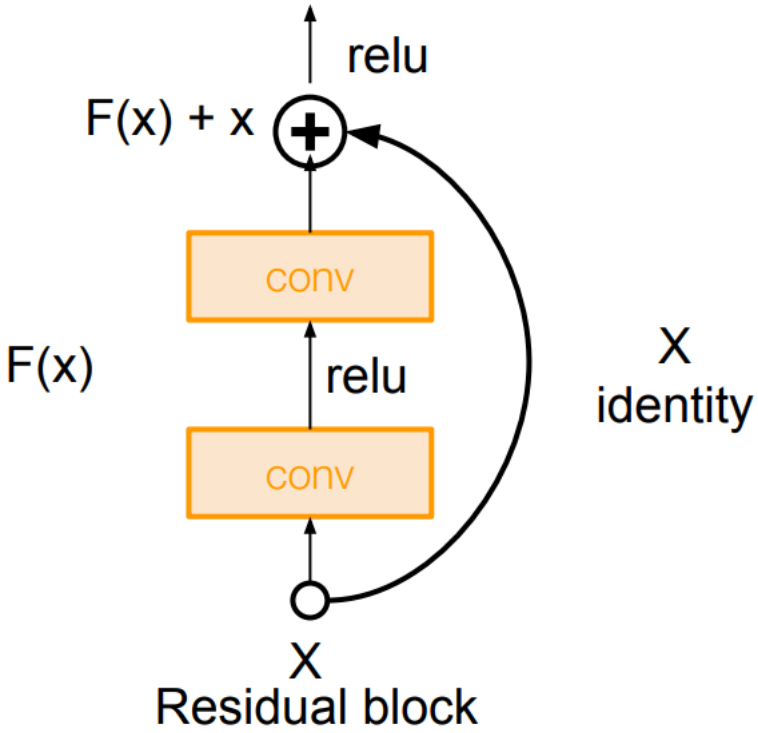
Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

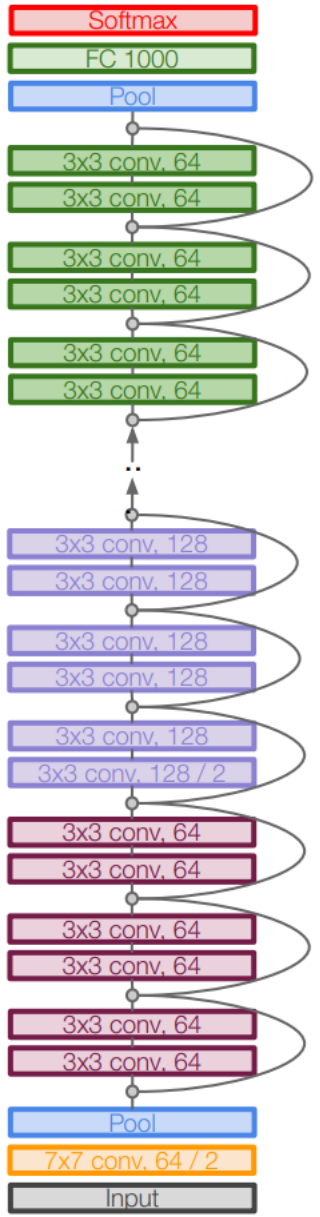
Case Study: ResNet

[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!

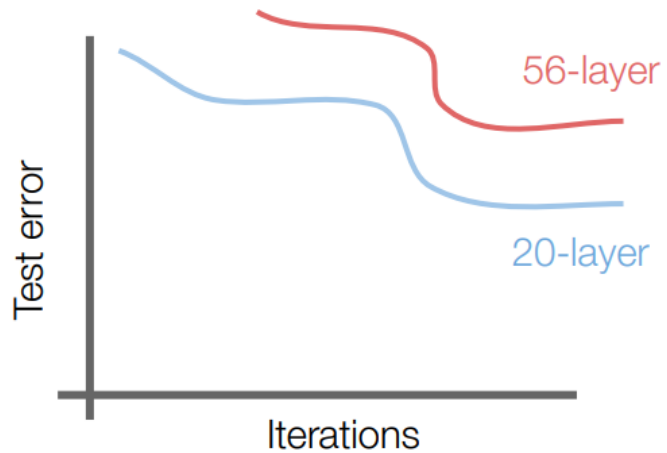


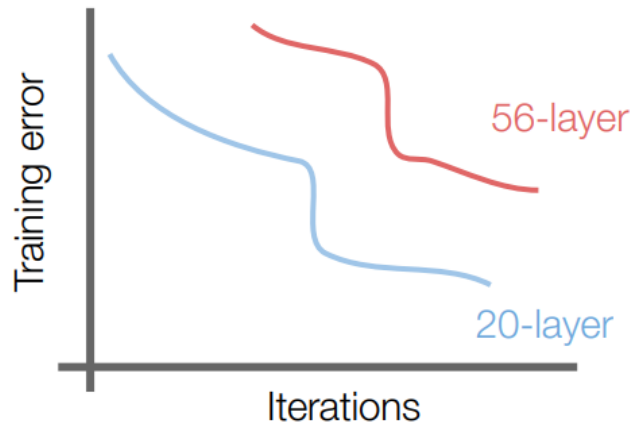


Case Study: ResNet

[He et al., 2015]

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





Case Study: ResNet

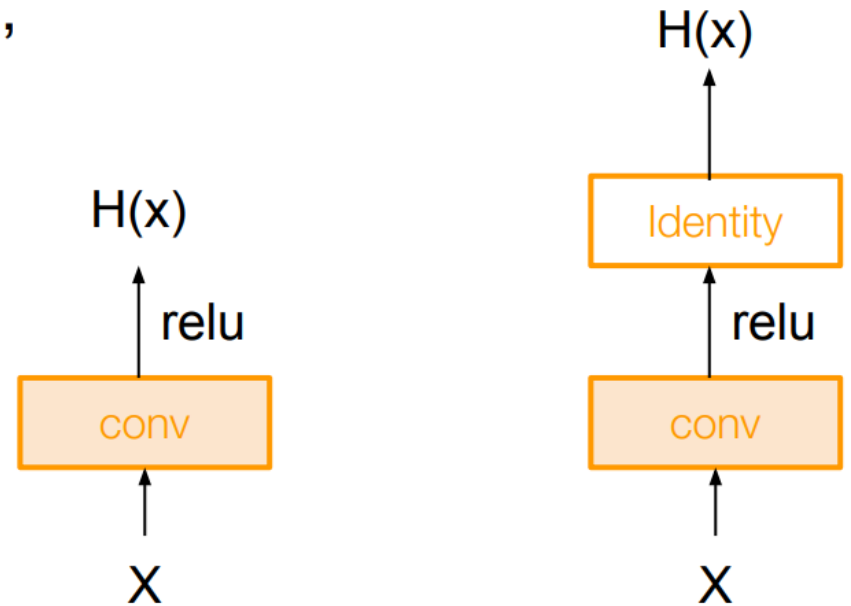
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

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

What should the deeper model learn to be at least as good as the shallower model?

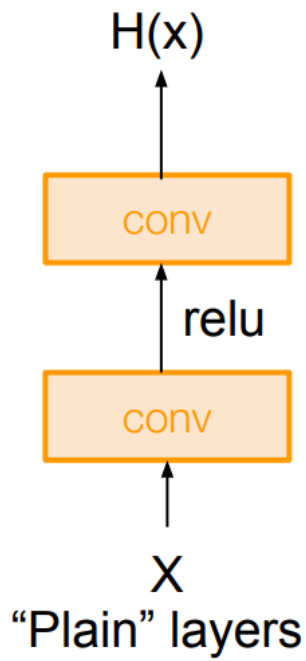
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

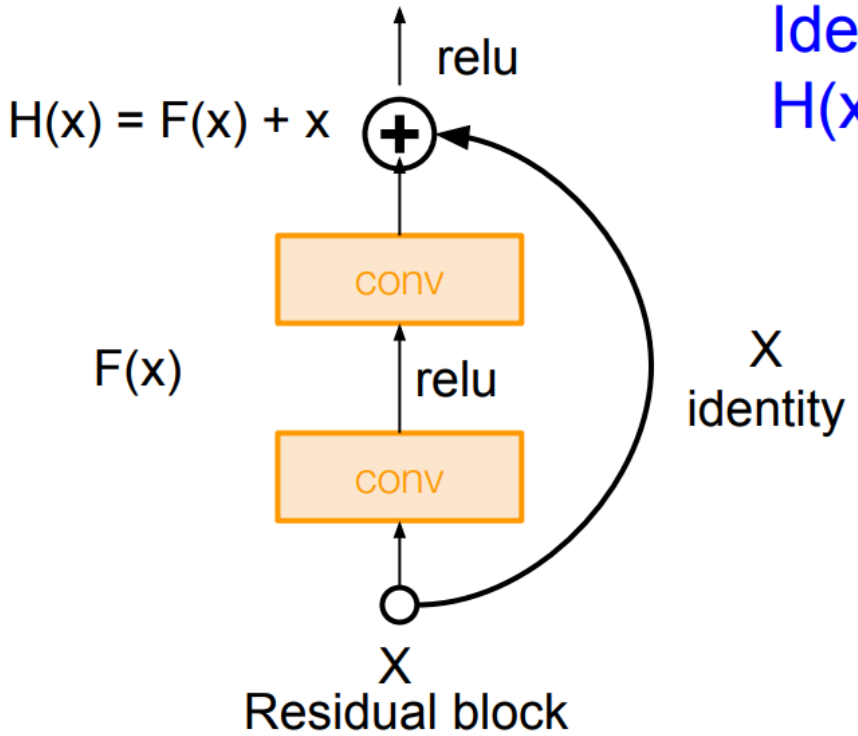


Case Study: ResNet

[He et al., 2015]

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





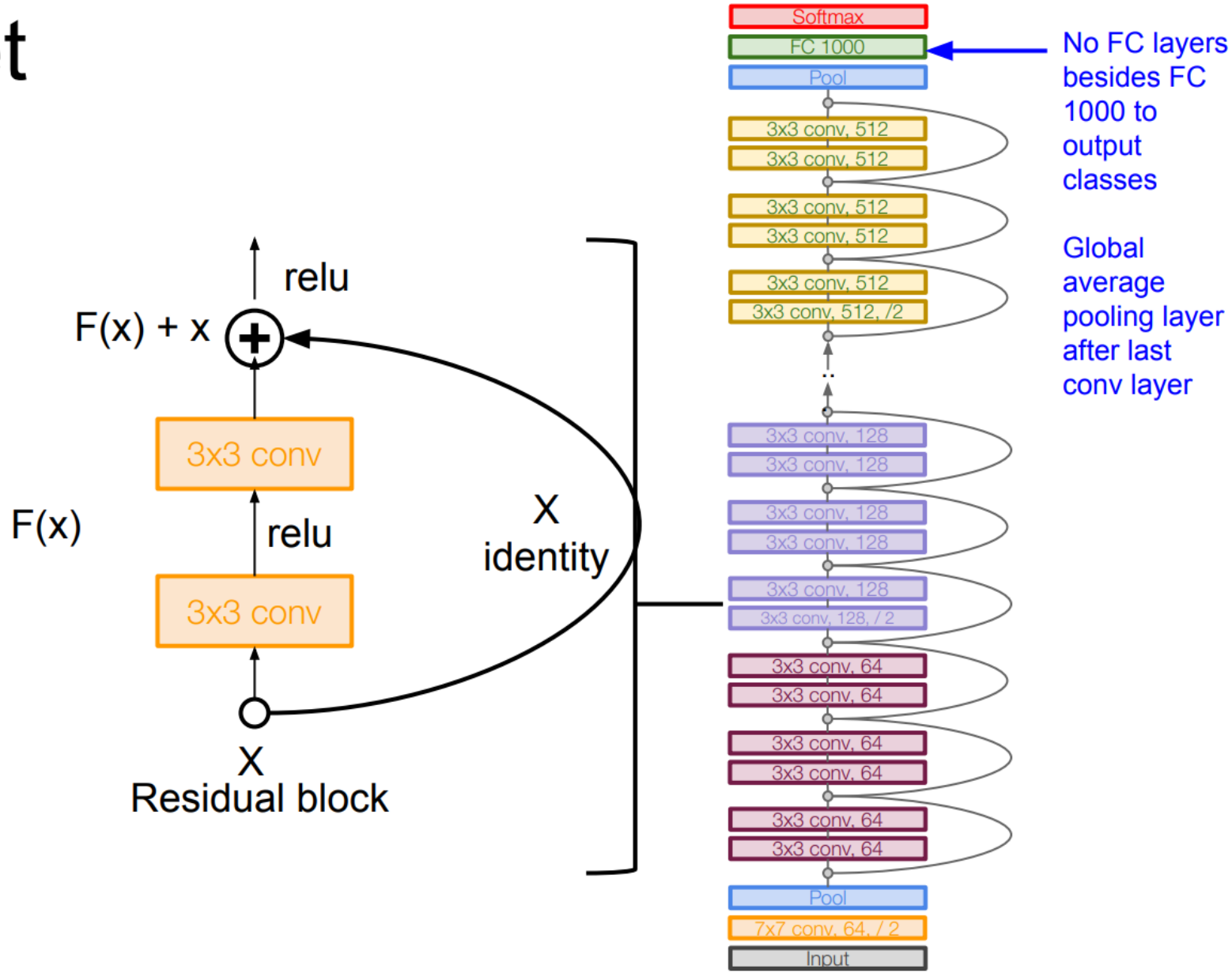
Identity mapping:
H(x) = x if F(x) = 0

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)



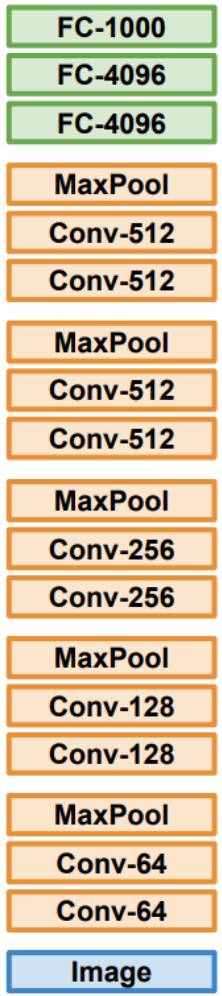
Efficient networks...

https://openai.com/blog/ai-and-efficiency/

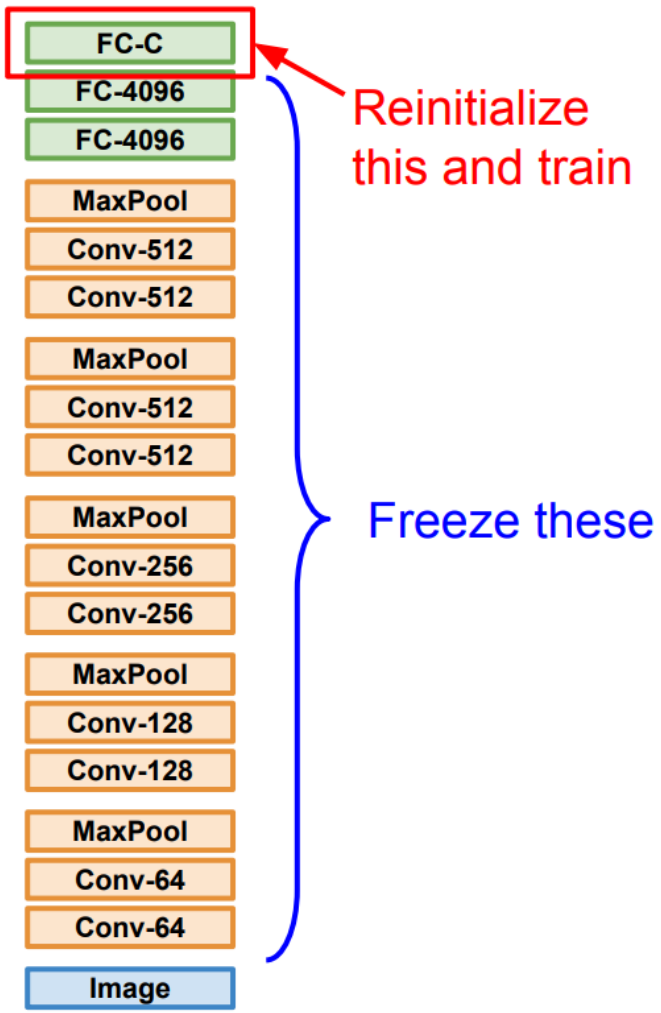
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

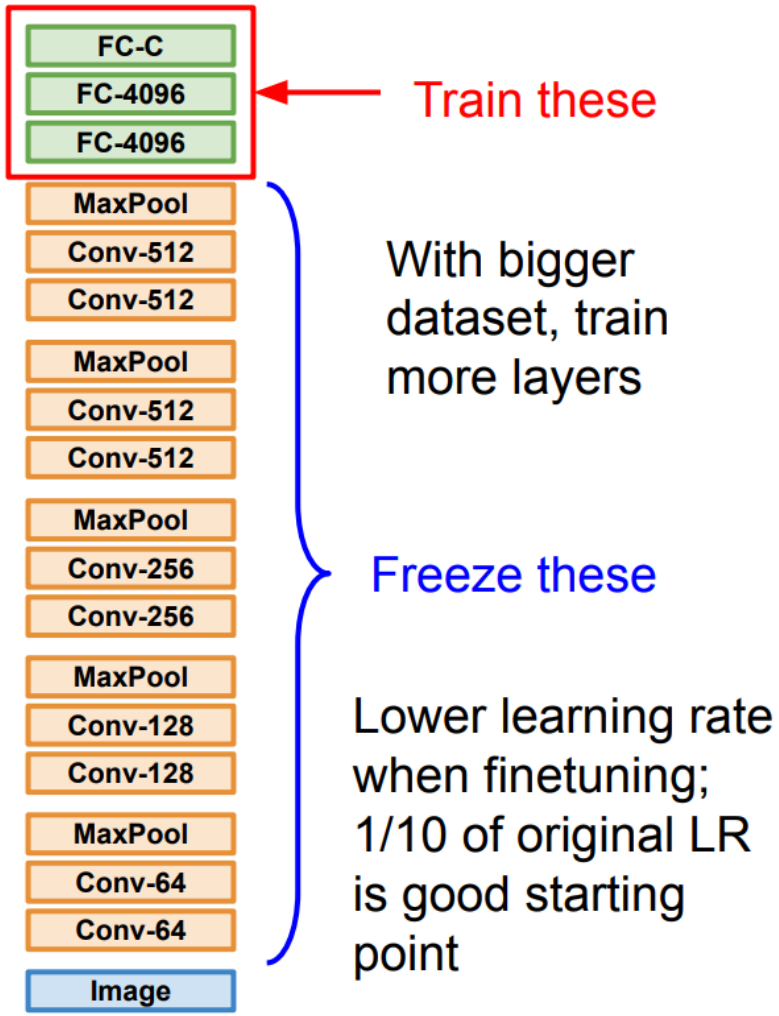
1. Train on Imagenet



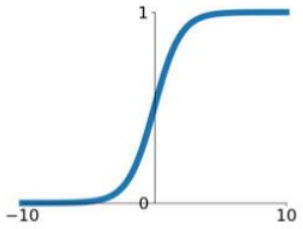
2. Small Dataset (C classes)



3. Bigger dataset



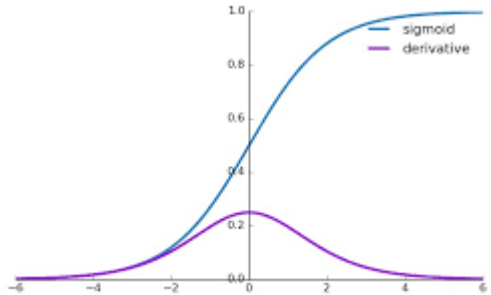
Deep Networks – Tricks of the trade – activation function

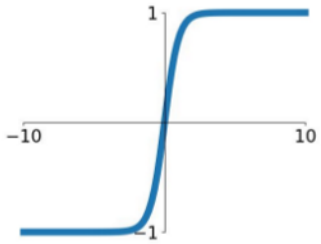


Sigmoid

sigmoid(x) = 1/ (1+e^{-x})

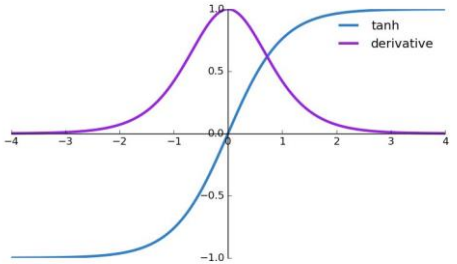
- Squashes numbers to range [0,1]
- Historically popular
- Saturated neurons “kill” the gradients
- Sigmoid outputs are not zero-centered – gradients in same direction
- Expensive to compute

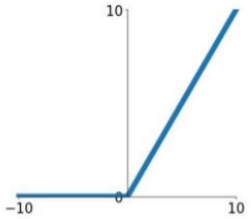




tanh(x) = e^{x} – e^{-x}/ e^{x} –+e^{-x}

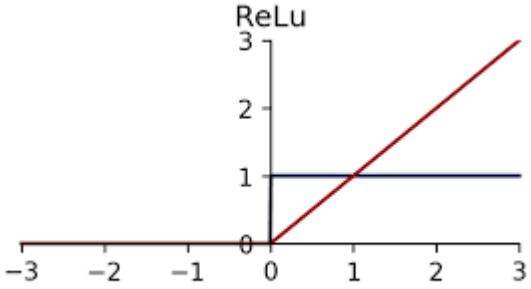
- Squashes numbers to range [1,-1]
- Saturated neurons “kill” the gradients
- Zero centered





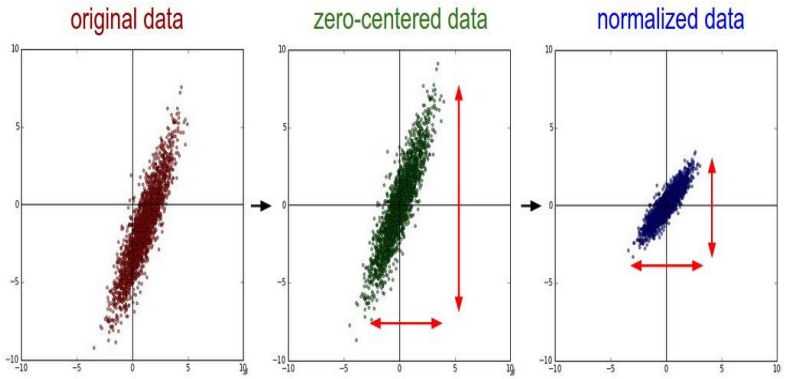
ReLU (Rectified Linear Unit)
relu(x) = max (0,x)

- Does not saturate (in + region)
-Converges much faster in practice (6x)
- Computationally efficient
- Not zero-centered

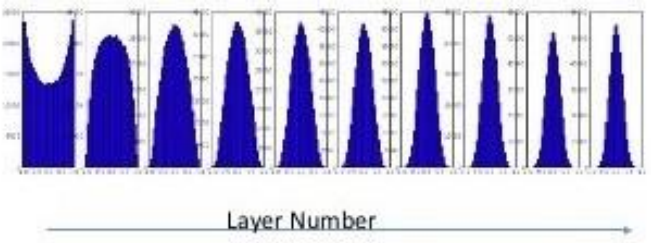


Deep Networks – Tricks of the trade

Data Initialisation

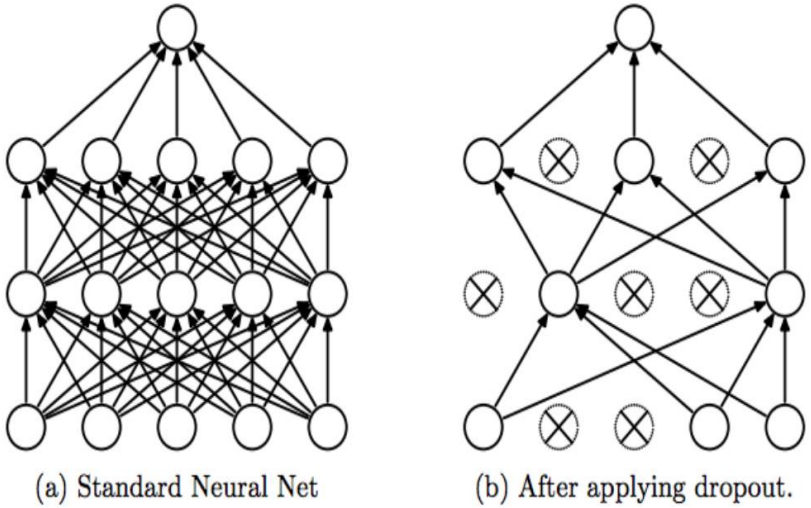


Weight Initialisation



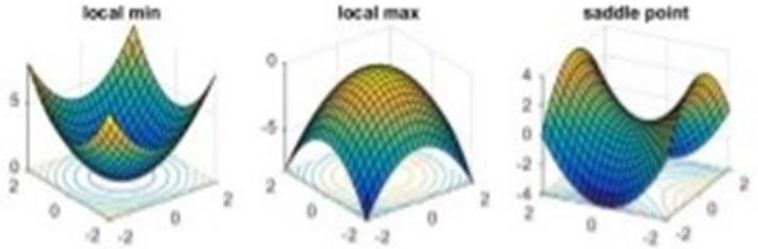
Depends on
- number of connections to hidden unit
- activation function of units

Regularisation



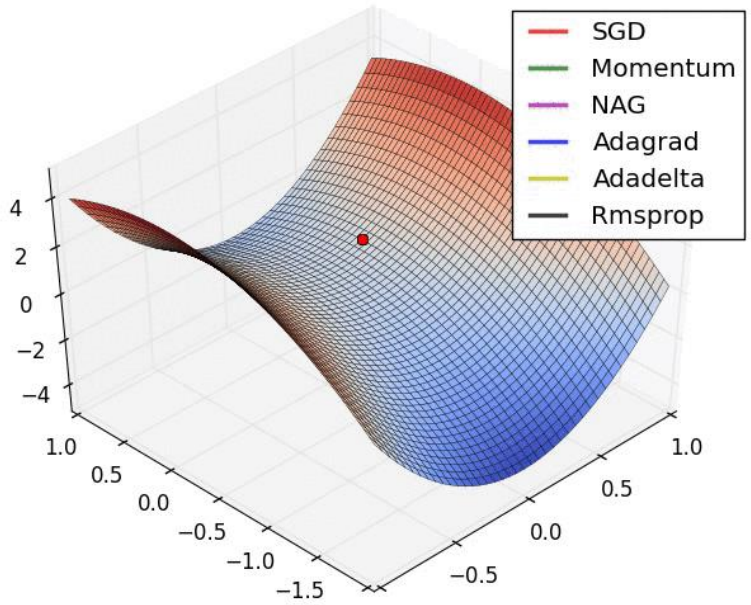
(a) Standard Neural Net (b) After applying dropout.

Escaping areas of low gradient



Local minima dominate in low Dimensions, but **saddle points dominate in high dimensions.**

Deep Networks – Tricks of the trade



Deep Networks – Main Takeaways

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

CNN Key Concepts

- Architecture
  - Convolution
  - Activation function as ReLU (rectified linear unit)
  - Sparsely connected (vs. fully connected MLP multi-layer perceptron)
  - Going deep – replicating feature responses
  - Pooling multi-responses (max-pooling, mean-pooling)
- Model training
  - Feedforward: Compute & propagate feature representations
  - Backpropagation: Compute & propagate gradients of error functions
  - Optimisation: Gradient descent and stochastic gradient descent (SGD)
  - Dropout (randomised repetition to minimise overfit)
  - Data augmentation (minimise overfit)
  - Mini-batches per epoch (random sampling a training set)
  - Many epochs (re-visiting the same training set in different ways)
- Going deep

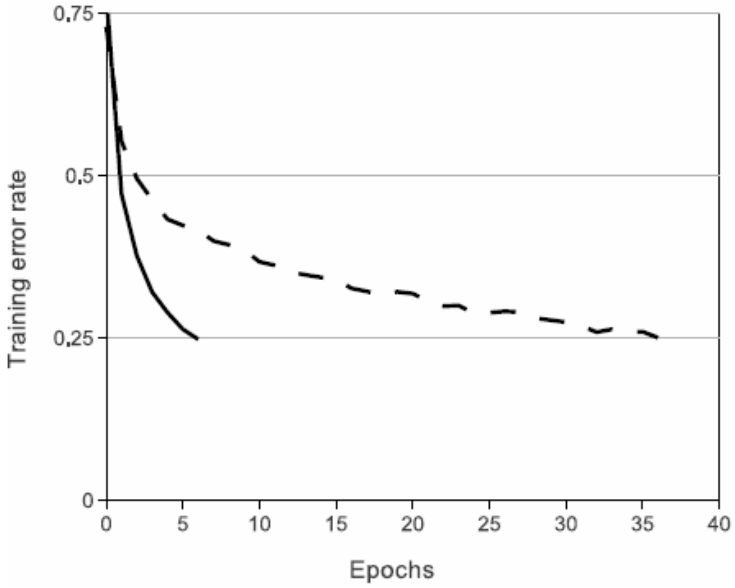
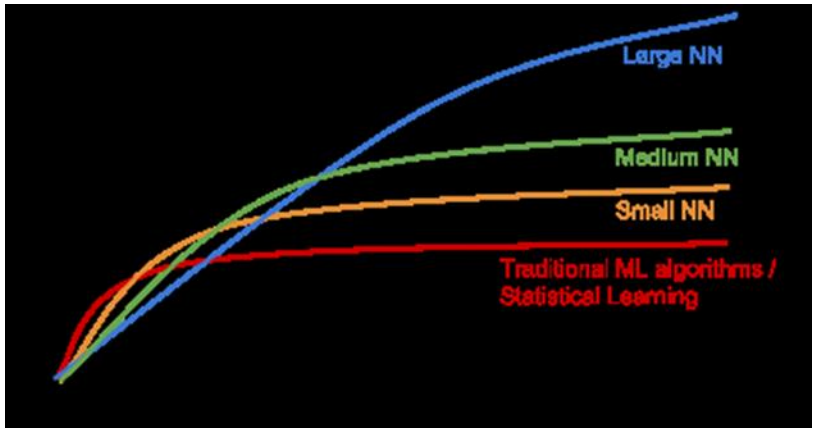


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons



Main CNN Architectures

AlexNet showed that you can use CNNs to train Computer Vision models.
ZFNet, **VGG** shows that bigger networks work better
GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to Efficient networks:

- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet**

Neural Architecture Search can now automate architecture design