Instituto Federal de Educação, Ciência e Tecnologia do Ceará
PPGCC

Introduction to Convolutional Neural Networks

Computer Vision

Prof. Dr. Pedro Pedrosa Rebouças Filho

pedrosarf@ifce.edu.br

professorpedrosa.com

Index

1. Introduction
2. Applications
3. Classical Approach
4. Data-Driven Approach
5. Regularization
6. Convolutional Networks
7. Modern Architectures
8. Comparative Analysis
9. Transfer Learning
10. Bibliography

Introduction

Deep Learning Revolution

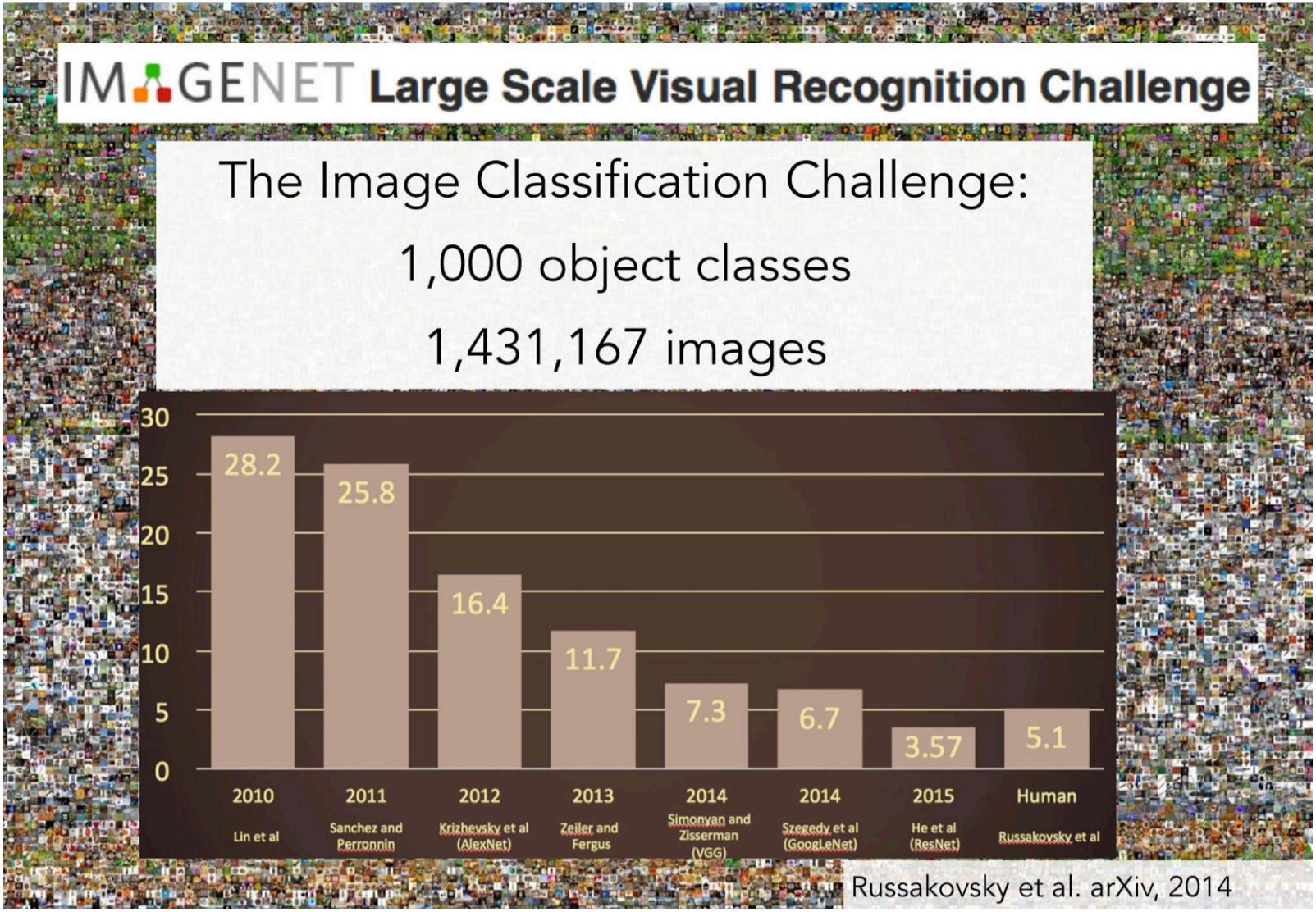
IMAGENET

22K categories and 14M images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

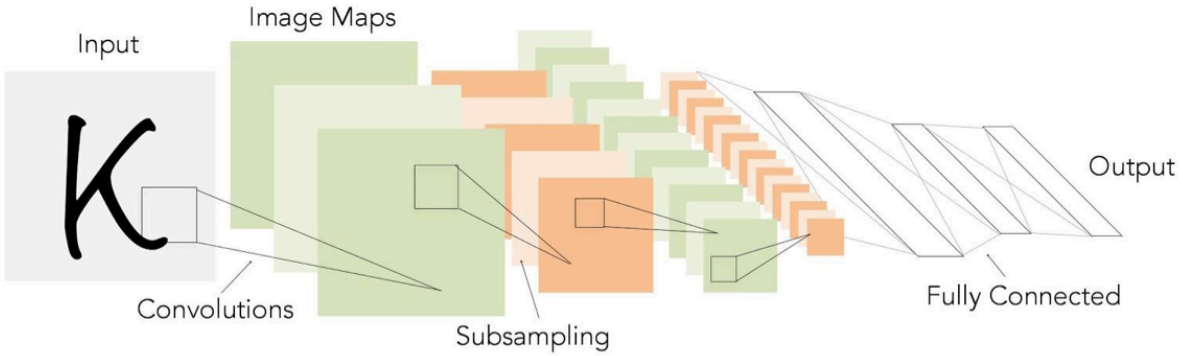
Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

Deep Learning Revolution



CNNs were not invented overnight

1998
LeCun et al.



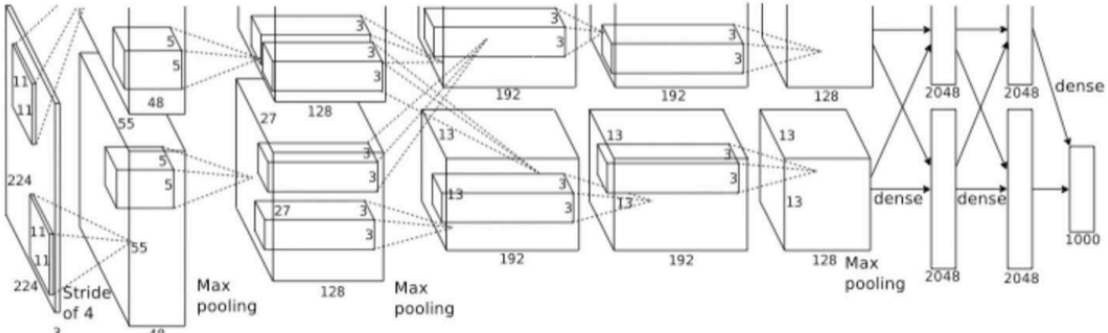
# of transistors

10^6

# of pixels used in training

10^7 NIST

2012
Krizhevsky et al.



# of transistors

10^9

GPUs

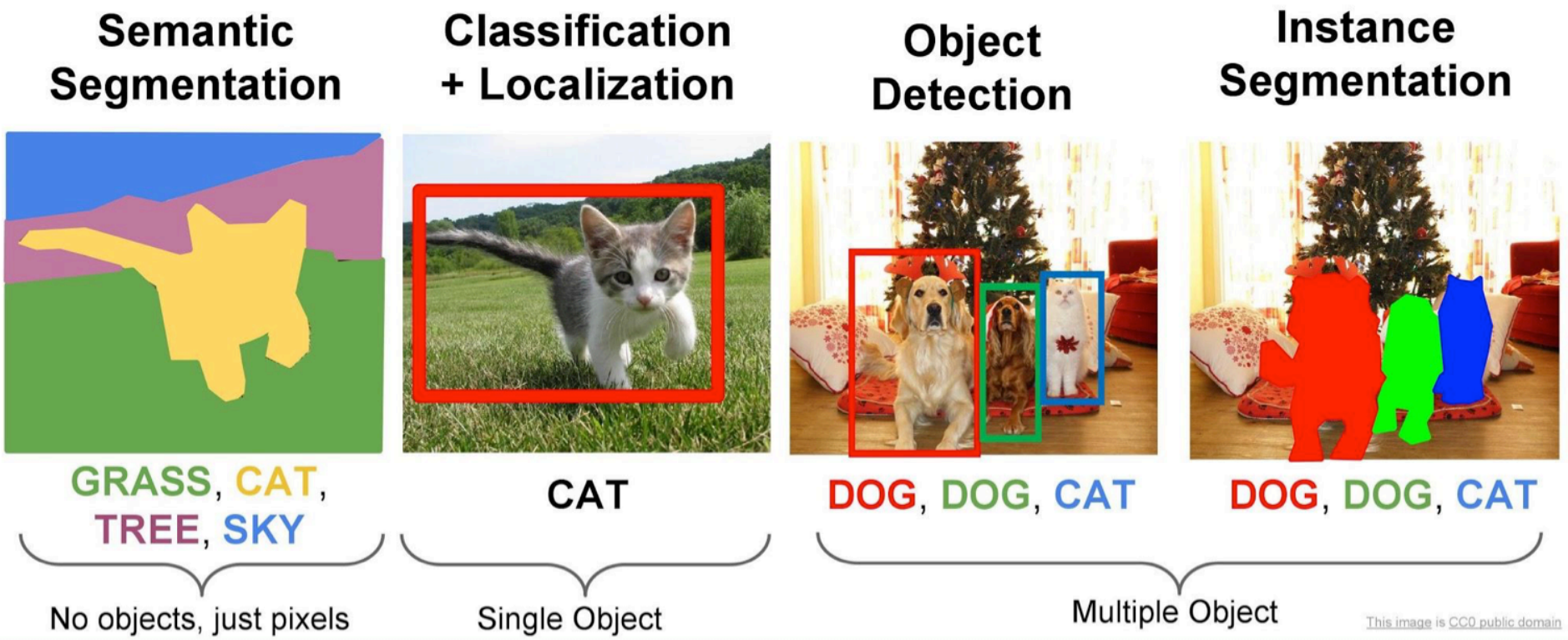
# of pixels used in training

10^{14} IMAGENET

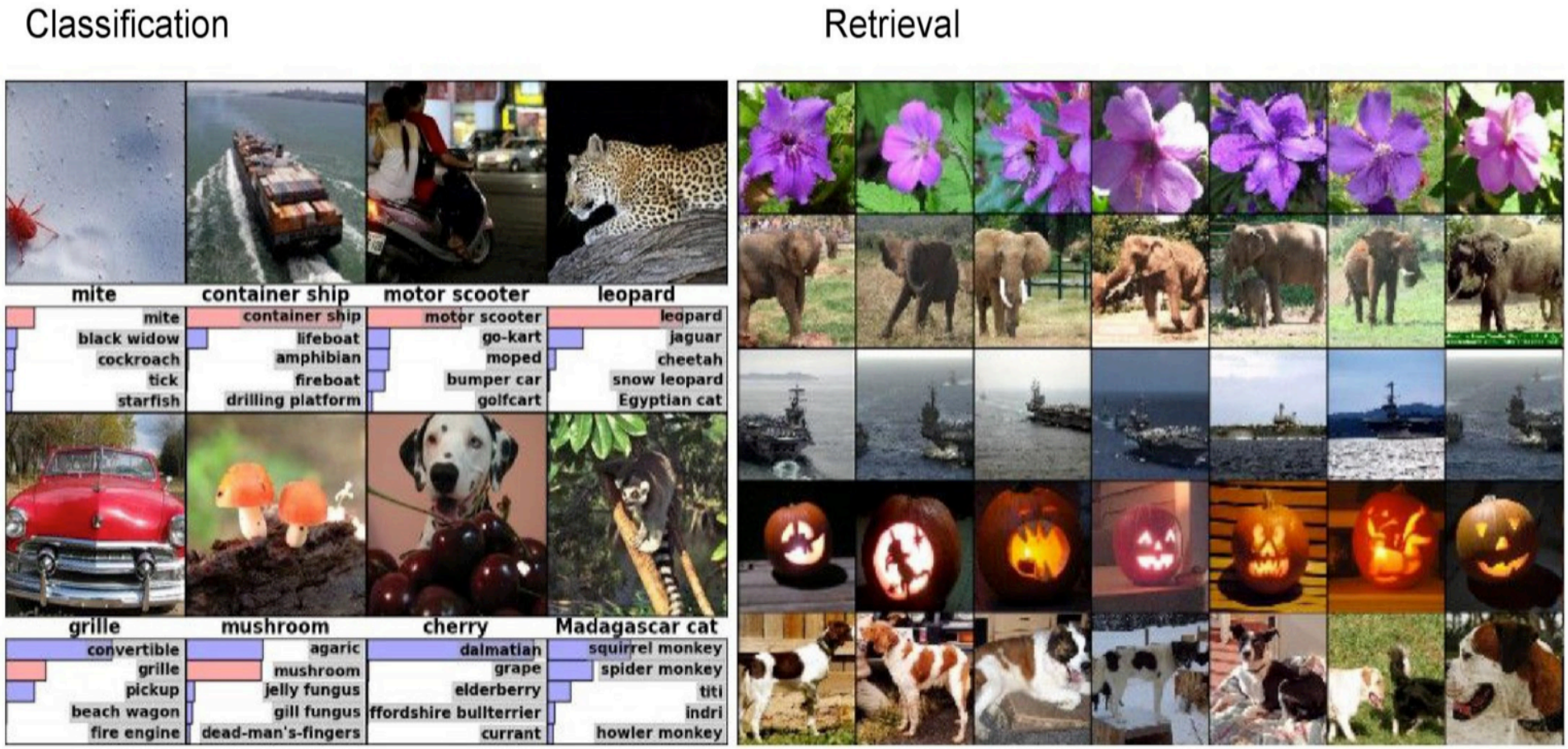
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Applications

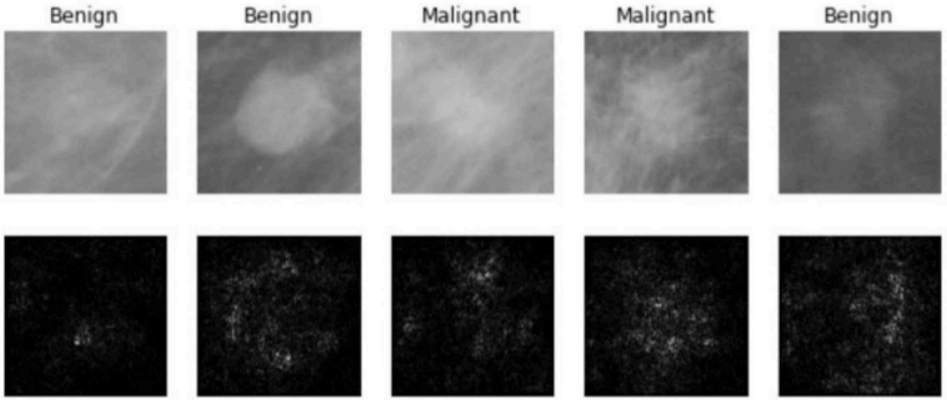
Applications



Applications



Applications - Classification



[Levy et al. 2016]

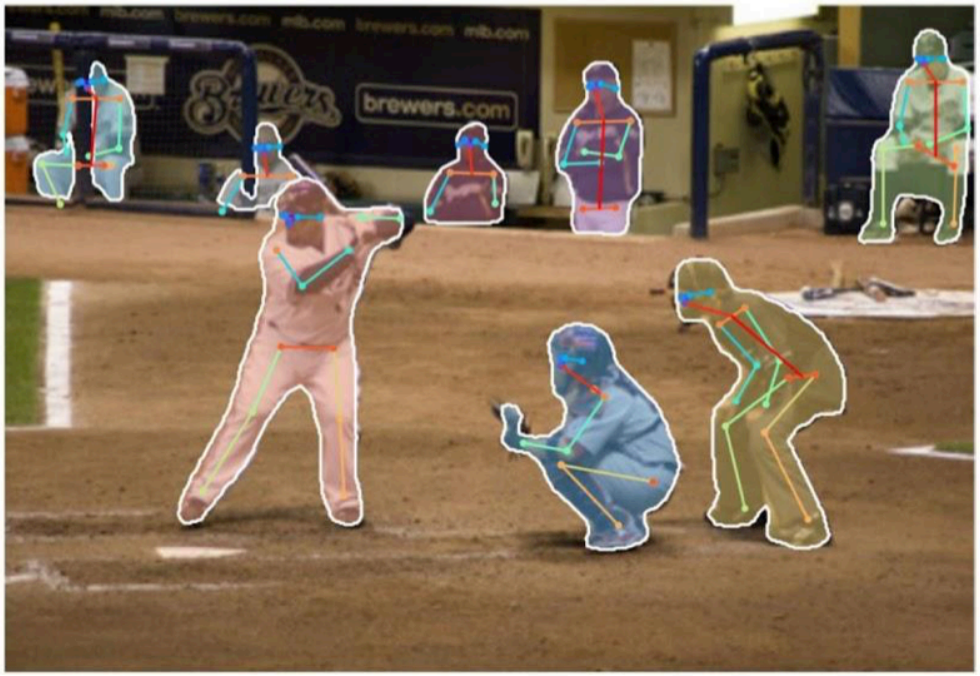
[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

Applications - (Tesla)

Applications - Instance Segmentation

Applications



He et al, “Mask R-CNN”, arXiv 2017
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.
Reproduced with permission.

Applications - Image Captioning

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

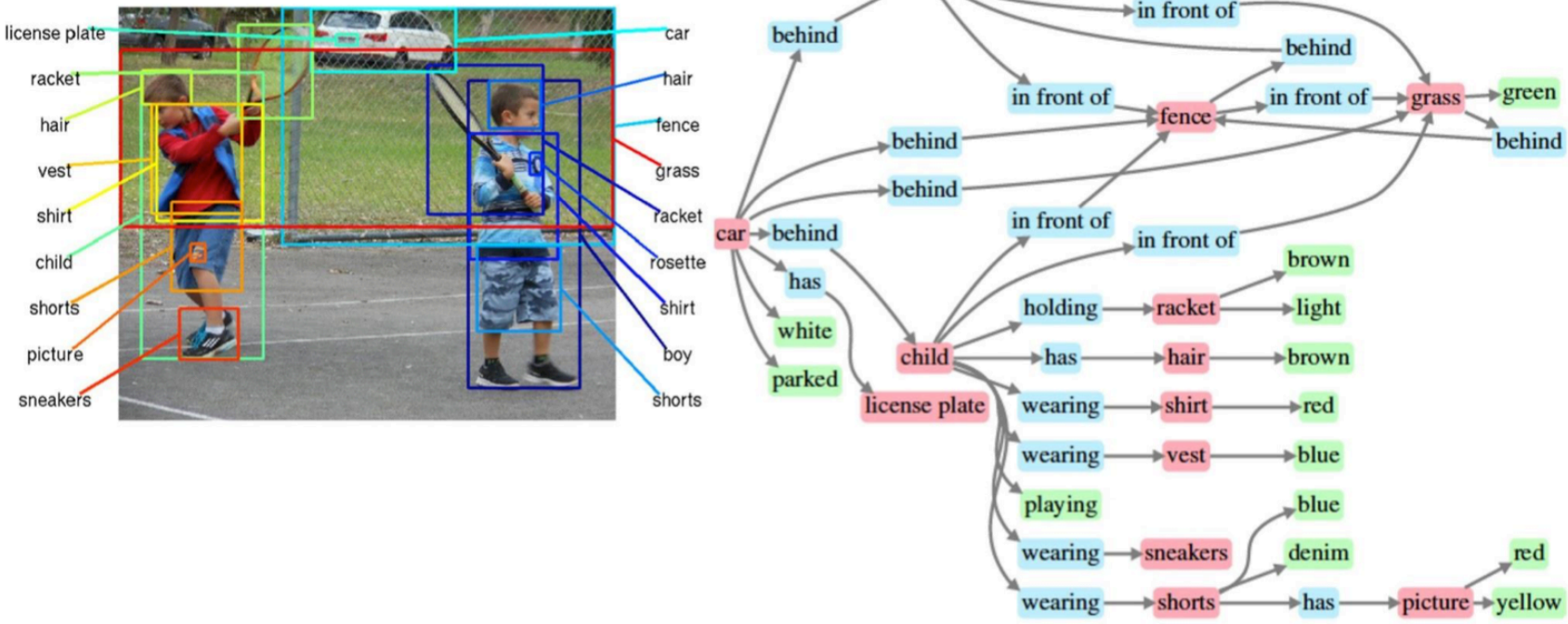
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

Applications - Scene Graphs

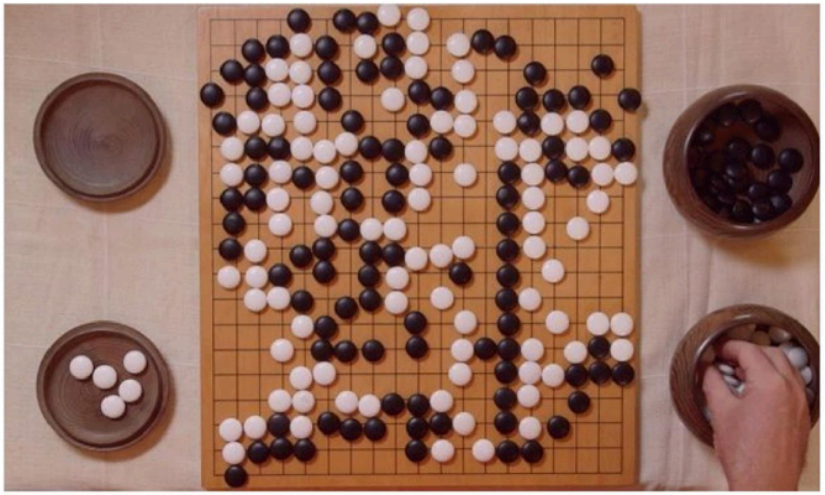


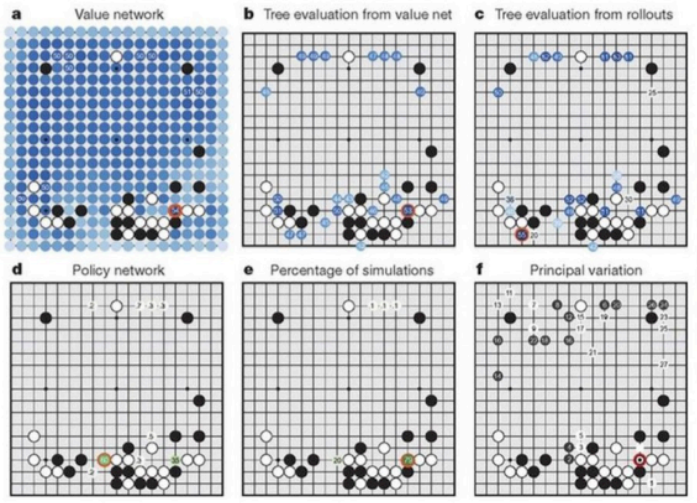
Johnson *et al.*, "Image Retrieval using Scene Graphs", CVPR 2015

Applications - Games

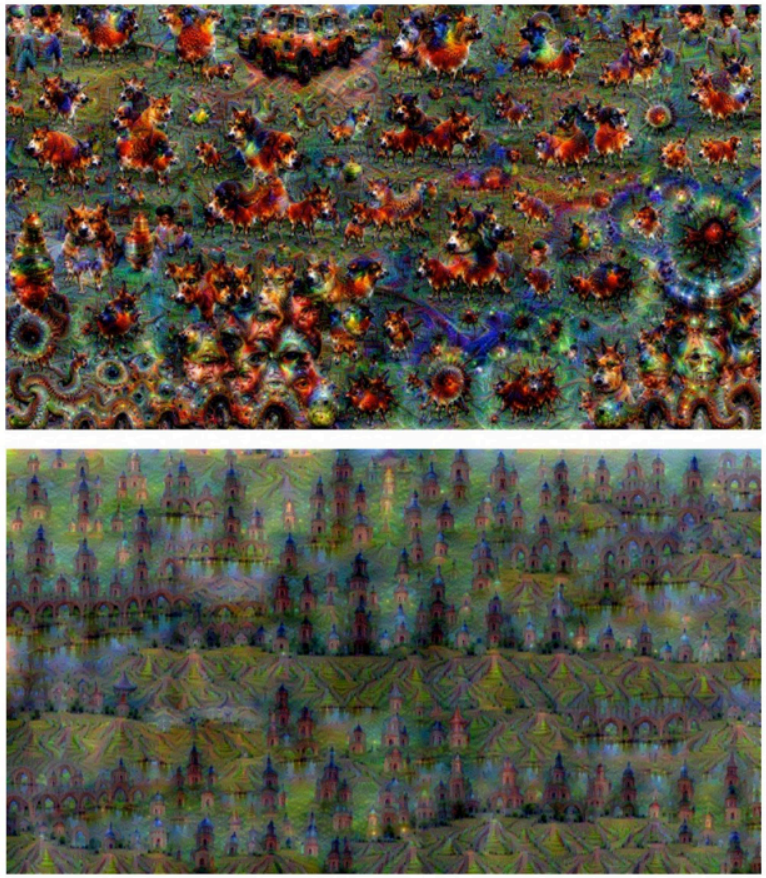
Google DeepMind

OpenAI





Applications - Deep Style



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Classical Approach

How computer perception is done?

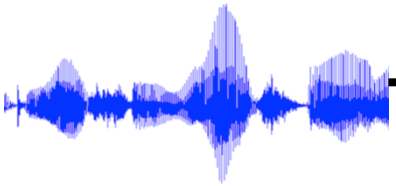
Object detection

Image

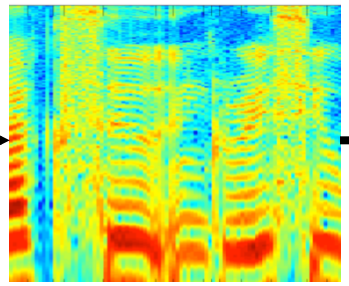
Low-level vision features

Recognition

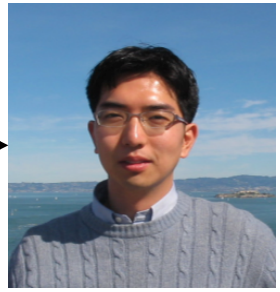
Audio classification



Audio



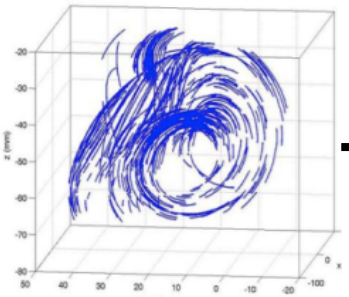
Low-level audio features



Speaker identification

Helicopter control

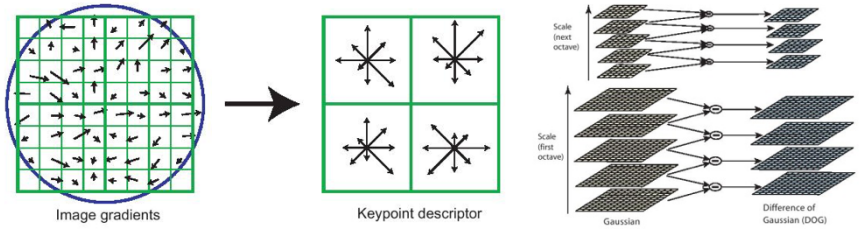
Helicopter



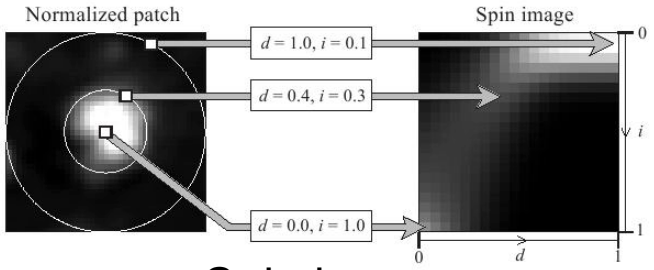
Low-level state features

Action

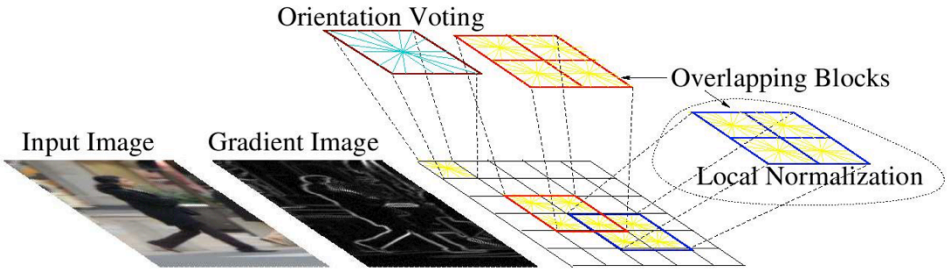
Computer Vision Features



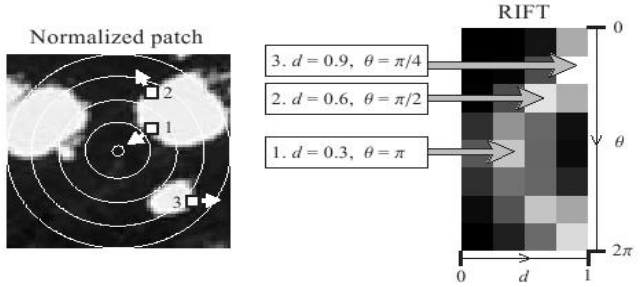
SIFT



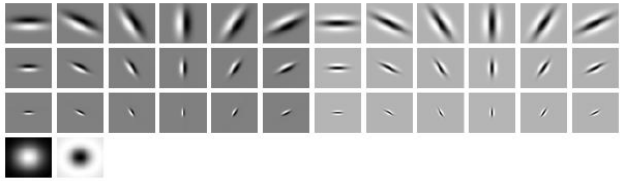
Spin image



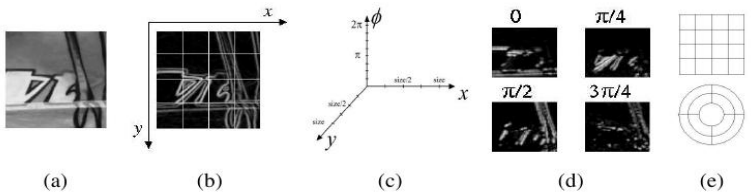
HoG



RIFT

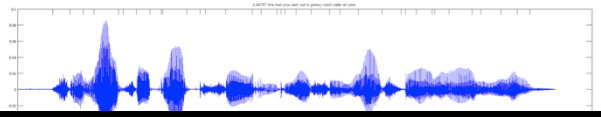


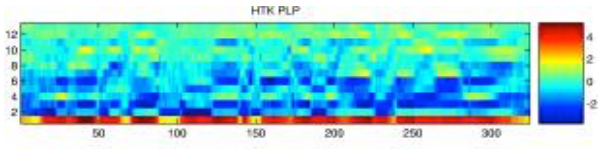
Textons



GLOH

Audio Features



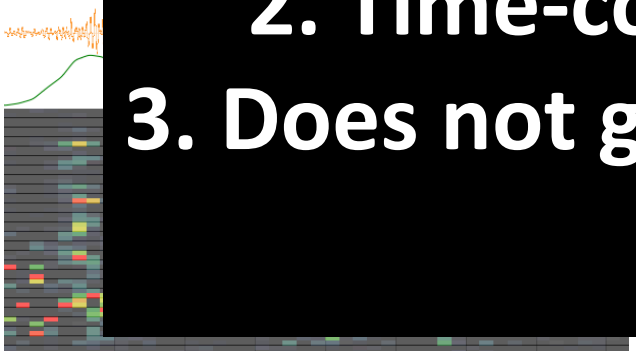


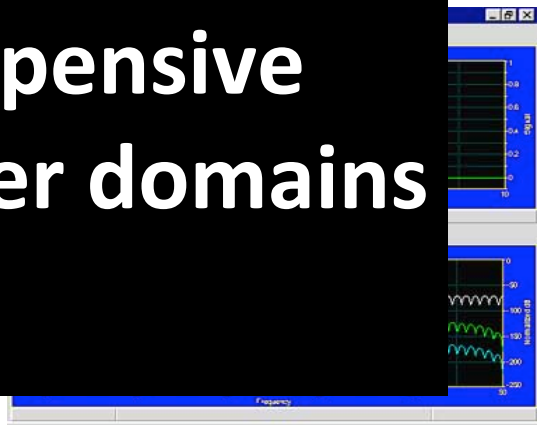
Problems of hand-tuned features

1. Needs expert knowledge

2. Time-consuming and expensive

3. Does not generalize to other domains





Flux

ZCR

Rolloff

Data-Driven Approach

Data-Driven Approach!

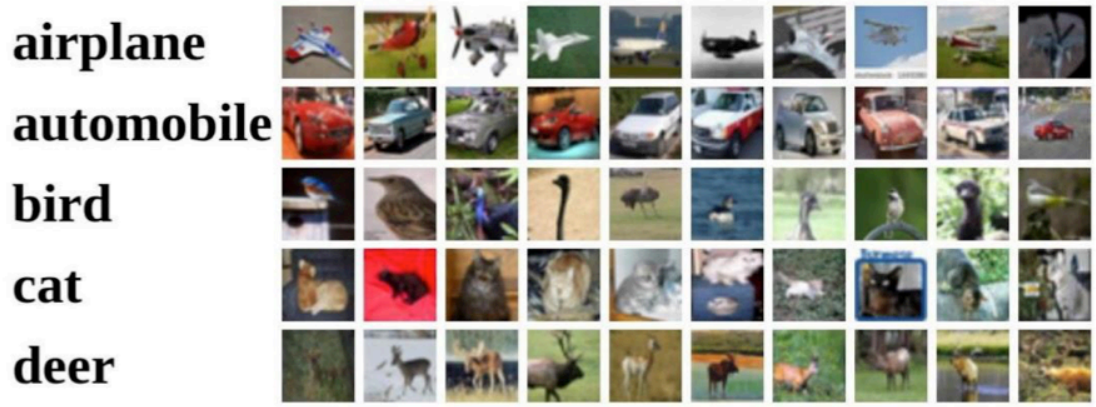
Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

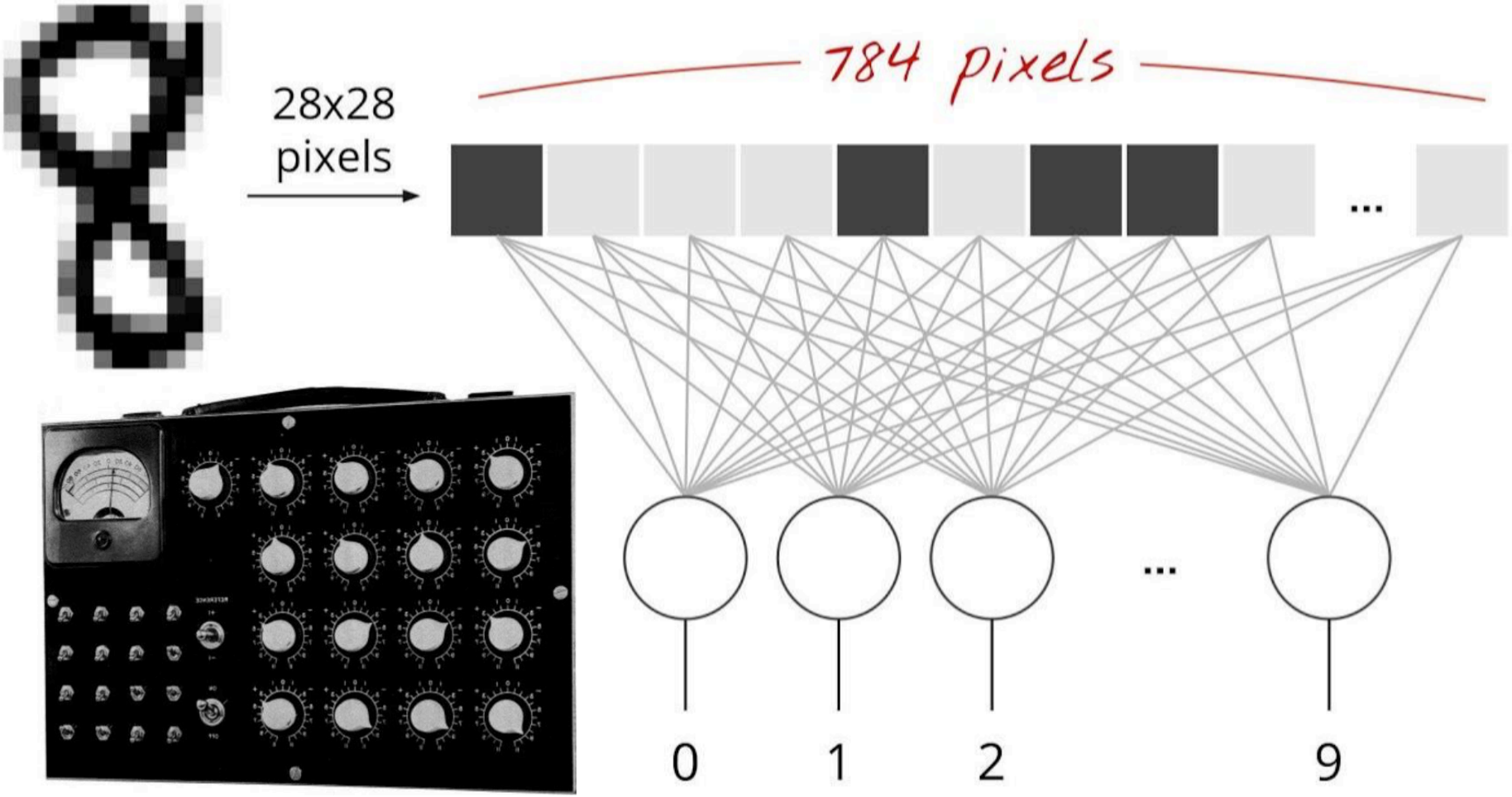
```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

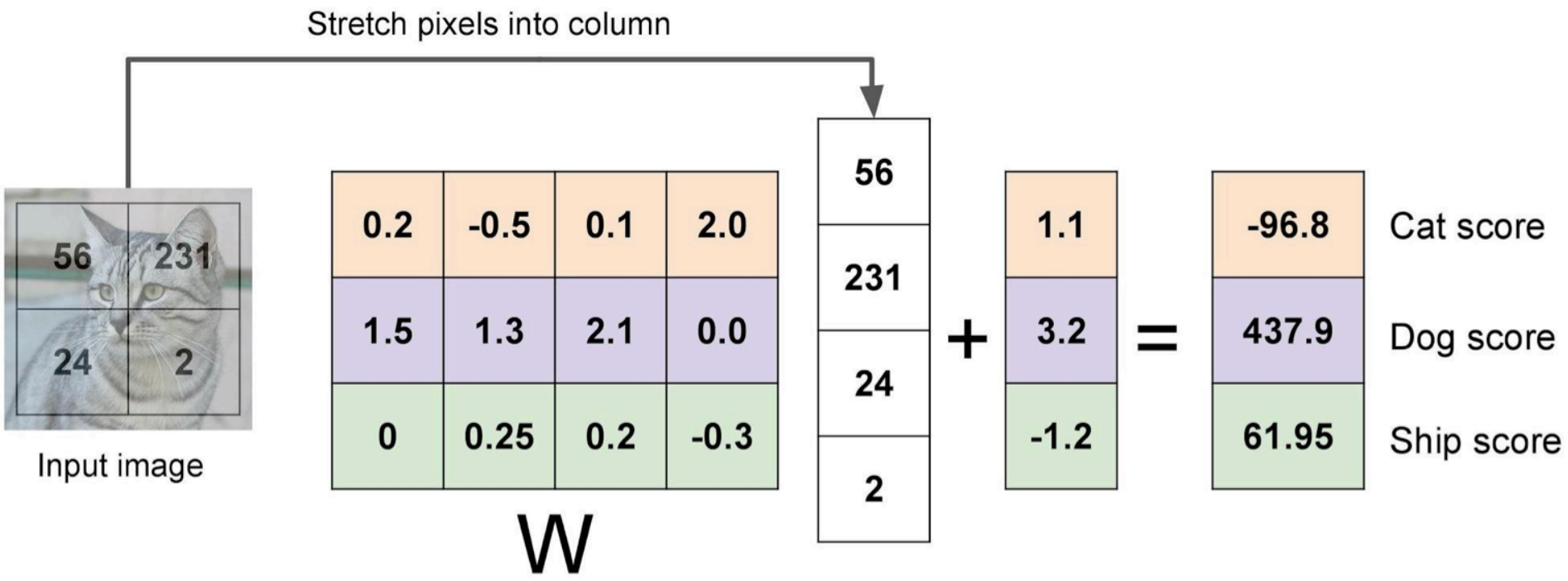
Example training set



Linear Classification

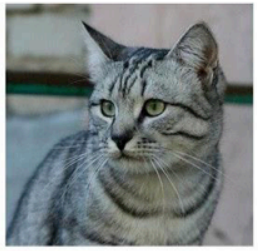


Example w/ 4 pixels and 3 classes



Cross-entropy loss

Softmax Classifier (Multinomial Logistic Regression)

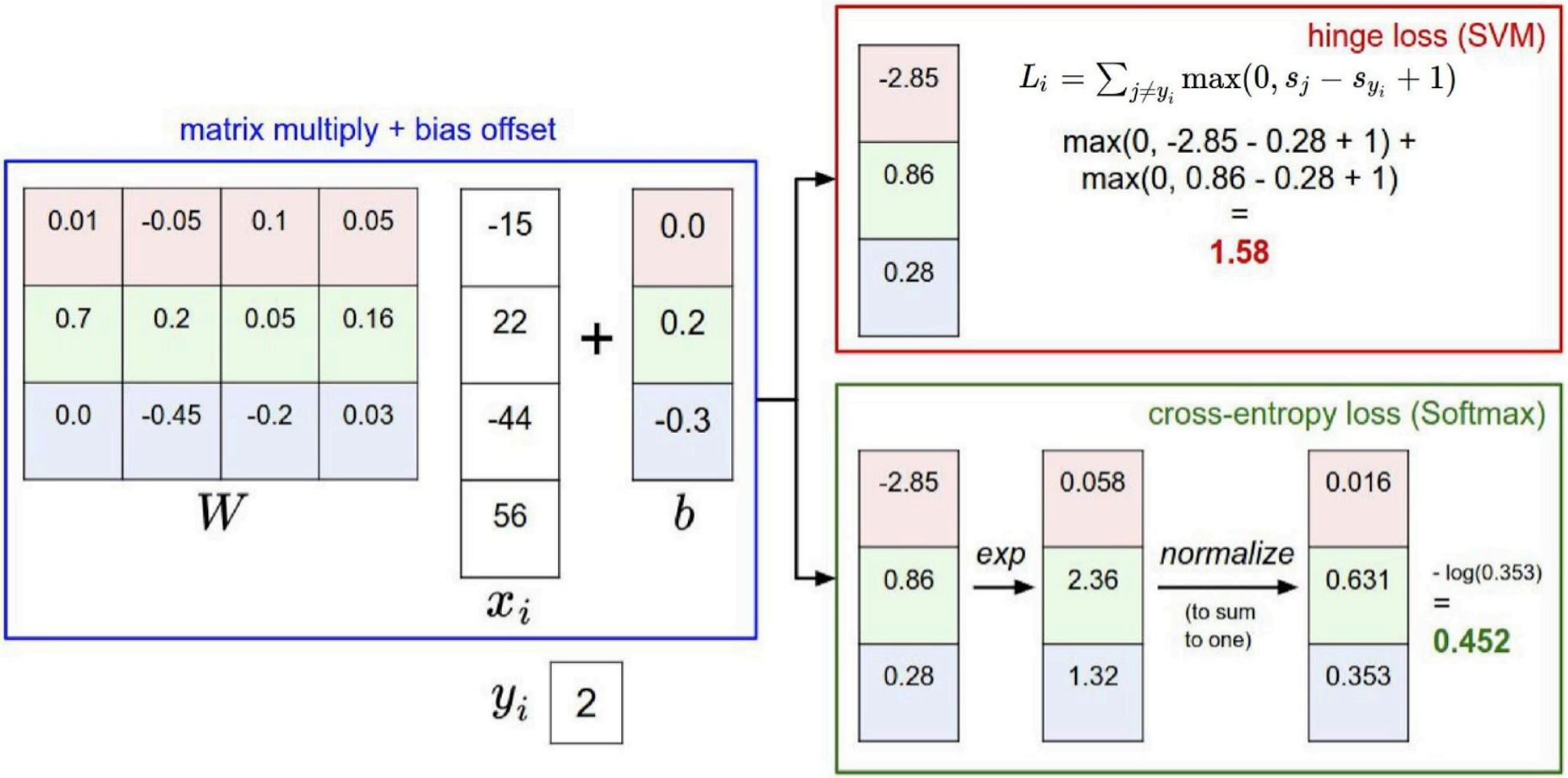


$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**	→	**24.5**	→	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	→	164.0	→	0.87	
frog	-1.7	→	0.18	→	0.00	

Loss


matrix multiply + bias offset
0.01 -0.05 0.1 0.05
0.7 0.2 0.05 0.16
0.0 -0.45 -0.2 0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
hinge loss (SVM)
-2.85
0.86
0.28
L_i = ∑_{j≠y_i} max(0, s_j − s_{y_i} + 1)
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum
to one)
0.016
0.631
0.353
- log(0.353)
=
0.452

Optimization

Follow the slope w/ Back Propagation

Backpropagation: a simple example

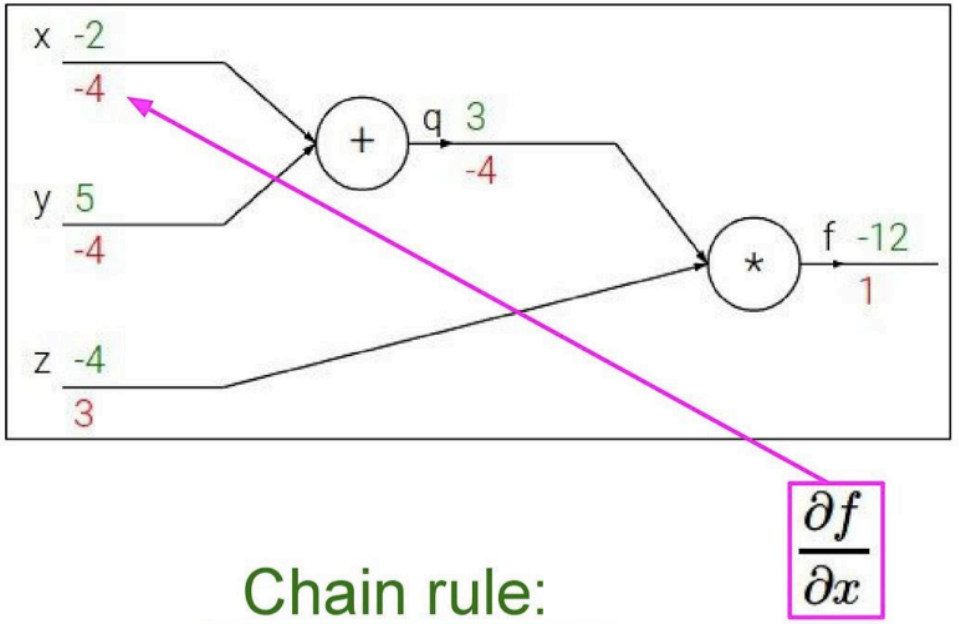
$$f(x, y, z) = (x + y)z$$

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

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

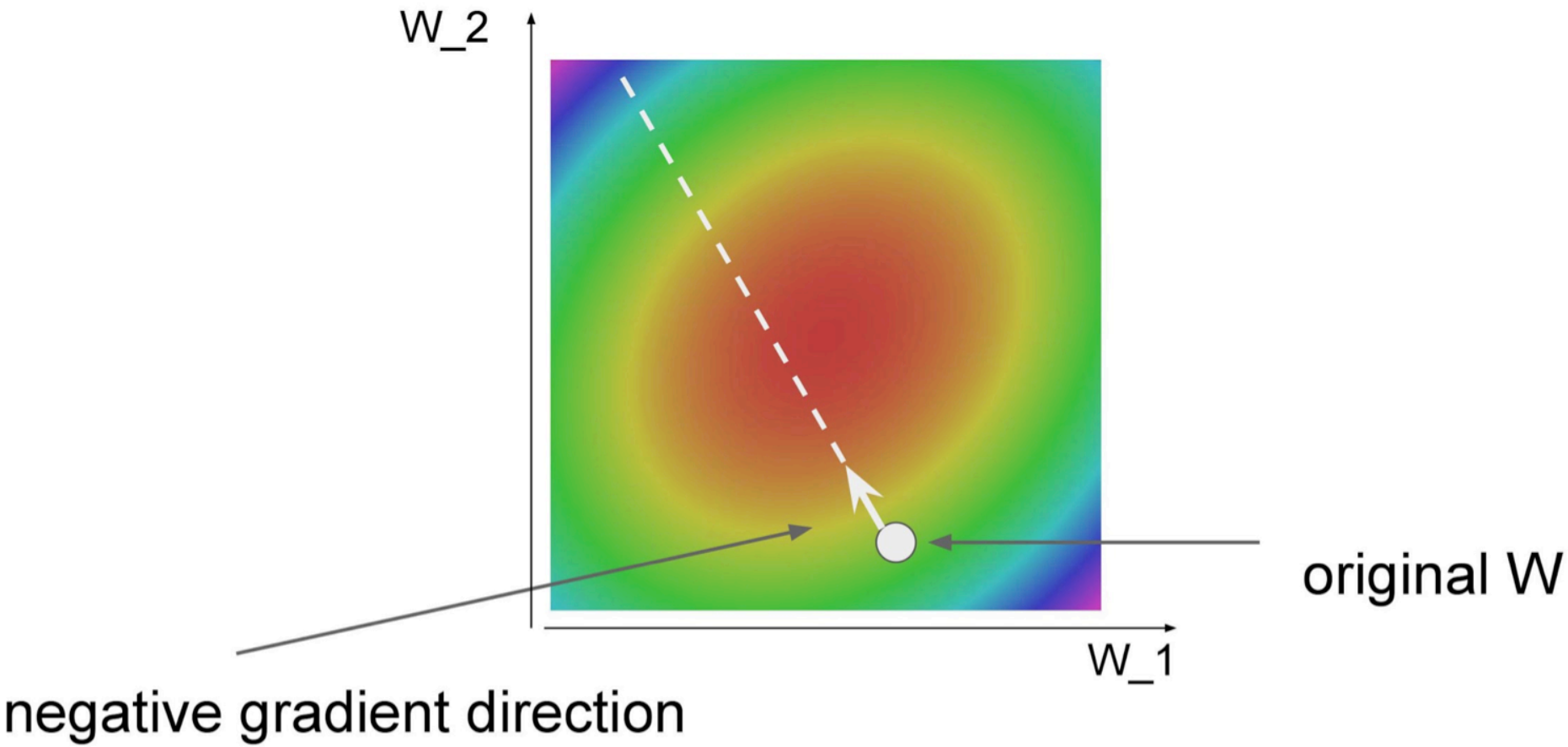
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

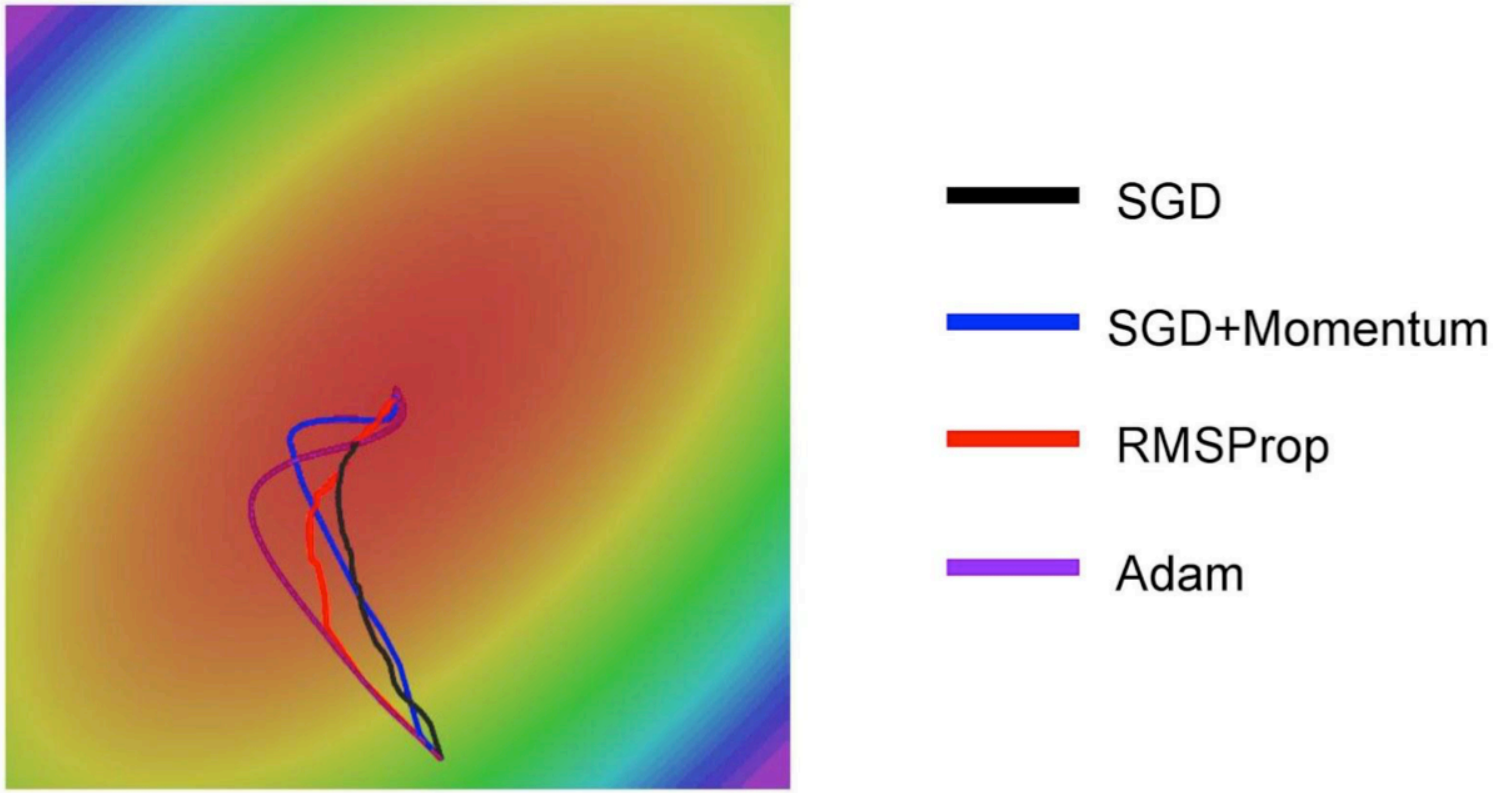


Chain rule:

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


W_2
W_1
original W
negative gradient direction

Modern Optimization Techniques



Adam Optimizer

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

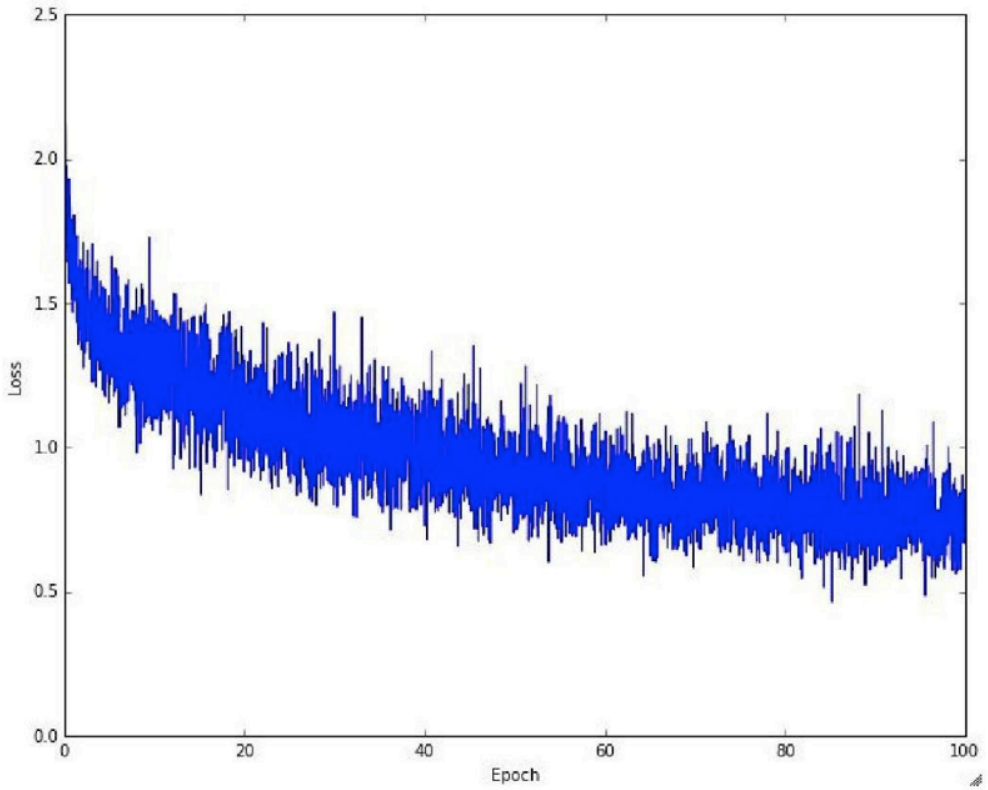
Bias correction for the fact that first and second moment estimates start at zero

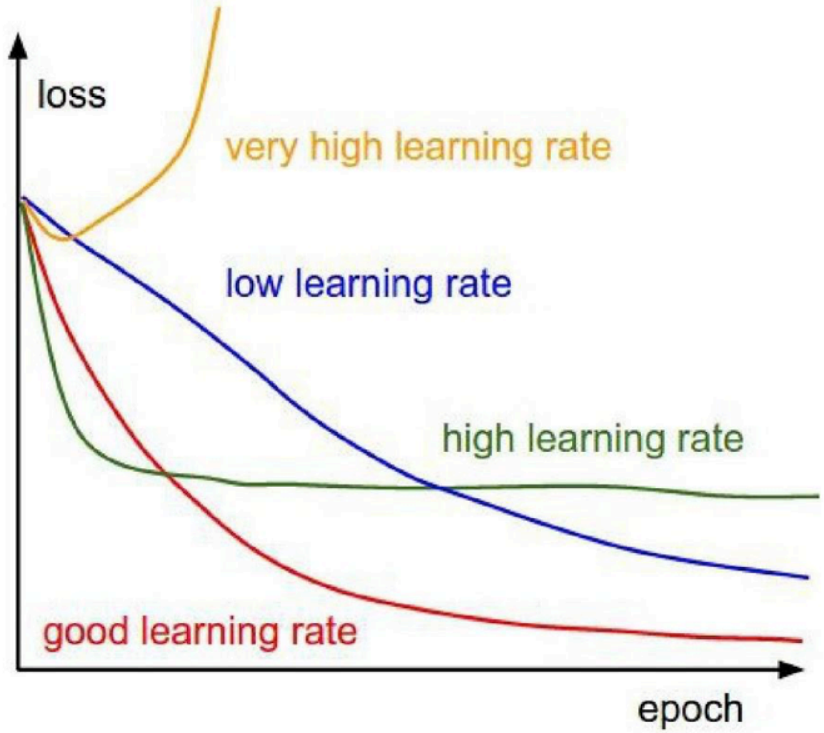
Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

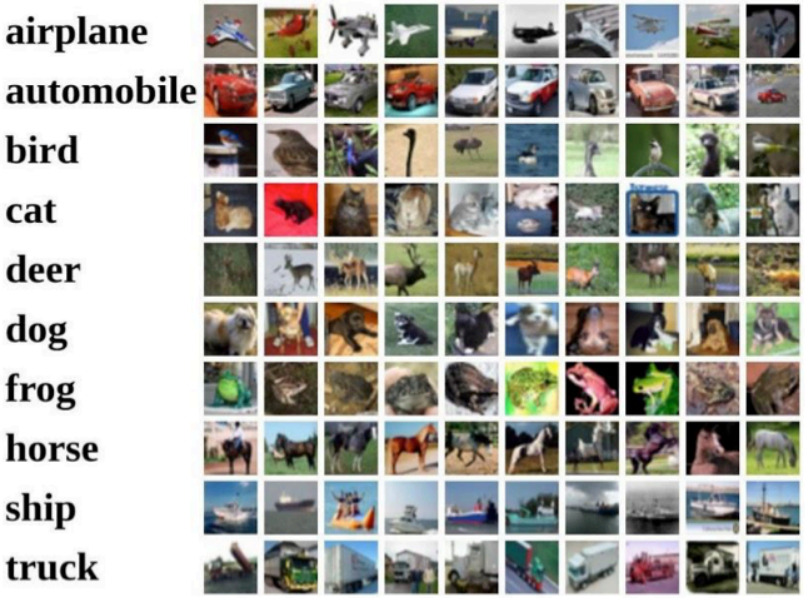
Learning rate vs. loss curve

Monitor and visualize the loss curve





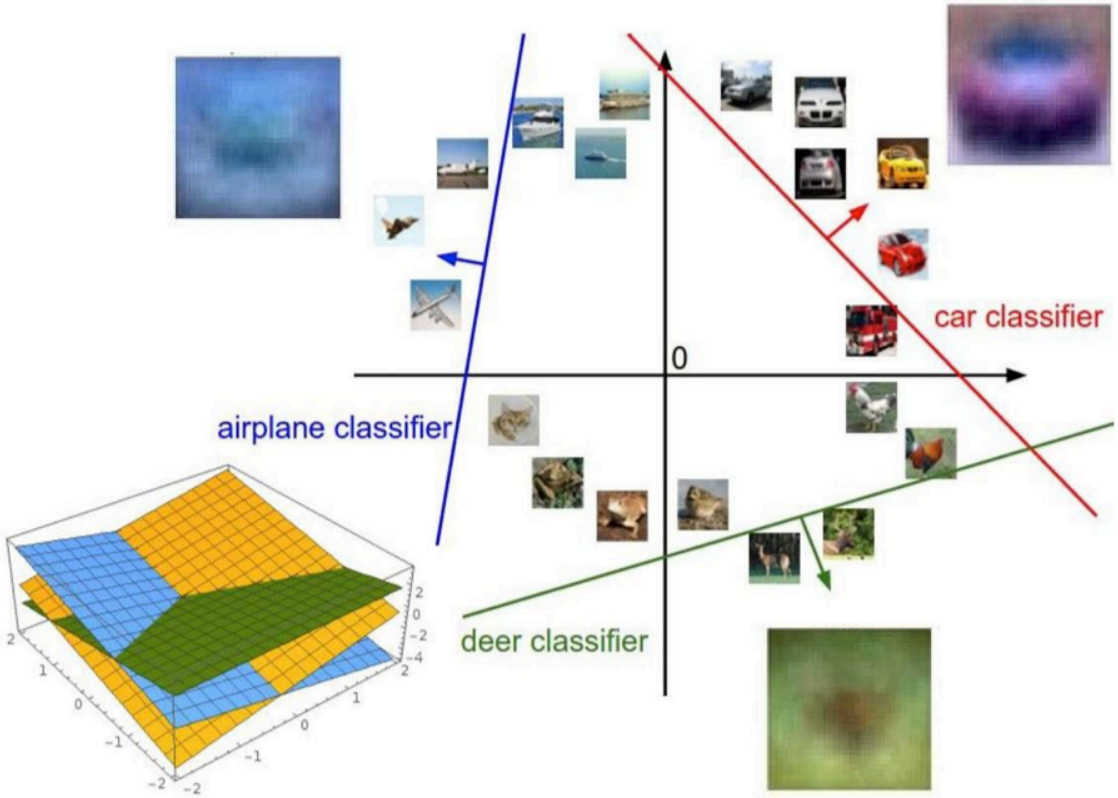
Interpreting a Linear Classifier



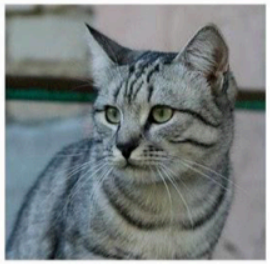
$$f(x,W) = Wx + b$$

Example trained weights of a linear classifier trained on CIFAR-10:

Interpreting a Linear Classifier

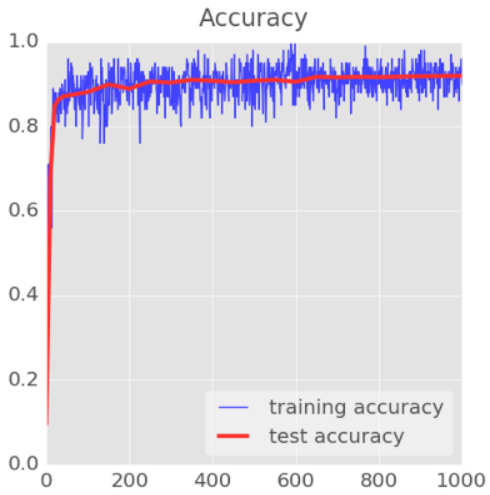


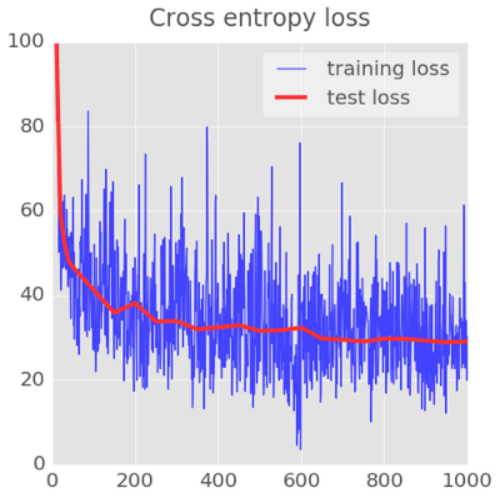
$$f(x,W) = Wx + b$$

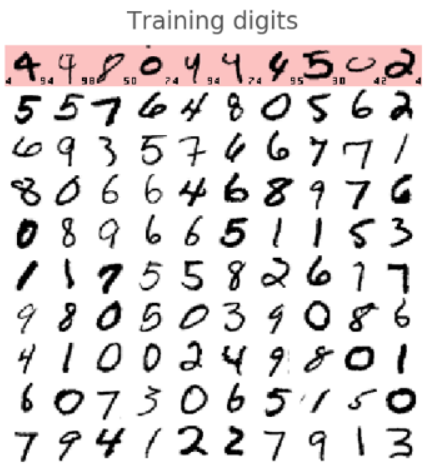


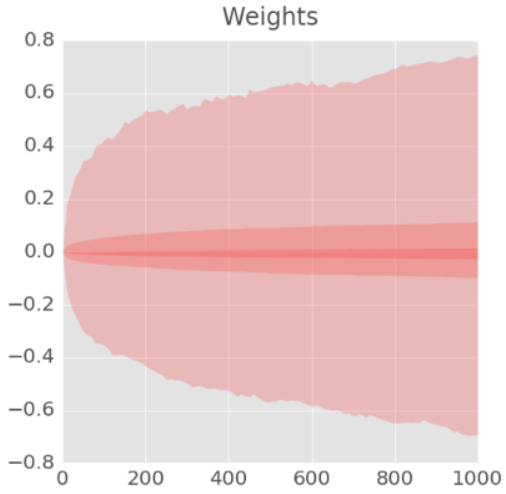
Array of **32x32x3** numbers
(3072 numbers total)

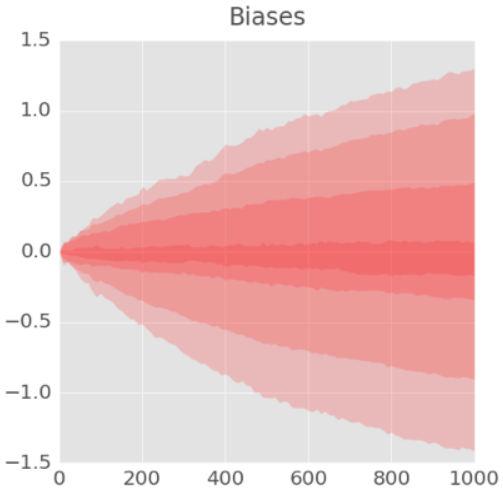
Linear Classification over MNIST









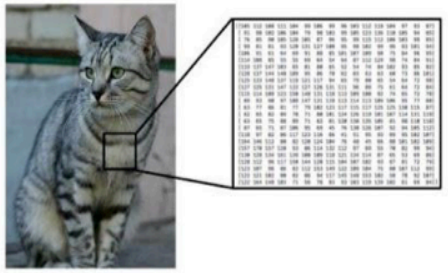


92 %

MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

Challenges of recognition

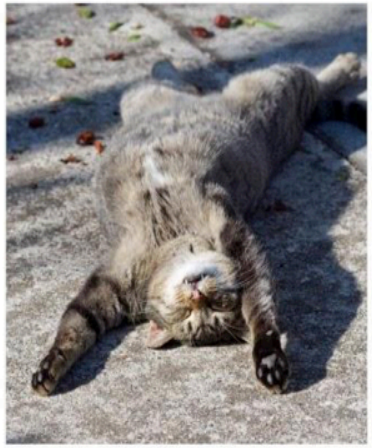
Viewpoint



Illumination

This image is CC0 1.0 public domain

Deformation



This image by Umberto Salvagnin is licensed under CC-BY 2.0

Occlusion

This image by jonsson is licensed under CC-BY 2.0

Clutter

This image is CC0 1.0 public domain

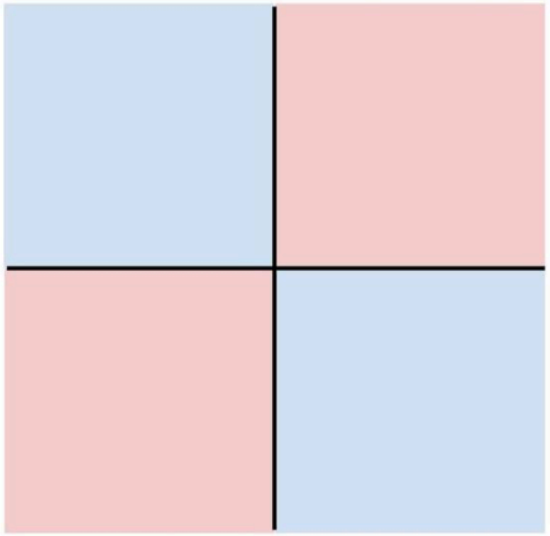
Intraclass Variation

This image is CC0 1.0 public domain

Hard Cases for a linear classifier

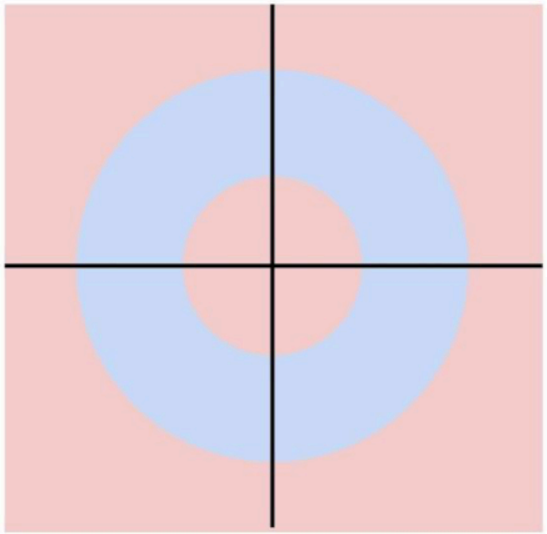
Class 1:
number of pixels > 0 odd

Class 2:
number of pixels > 0 even



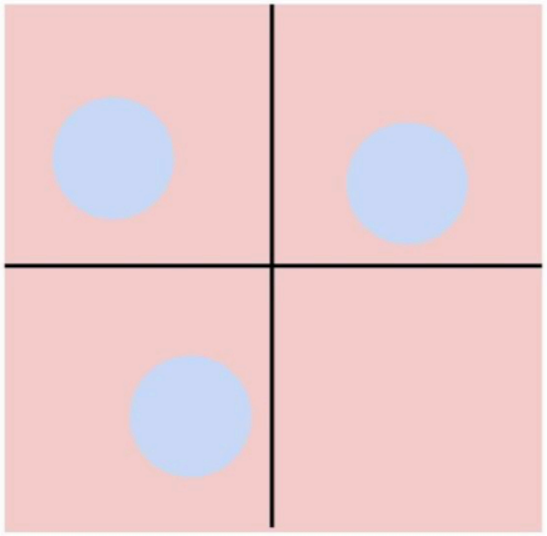
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else

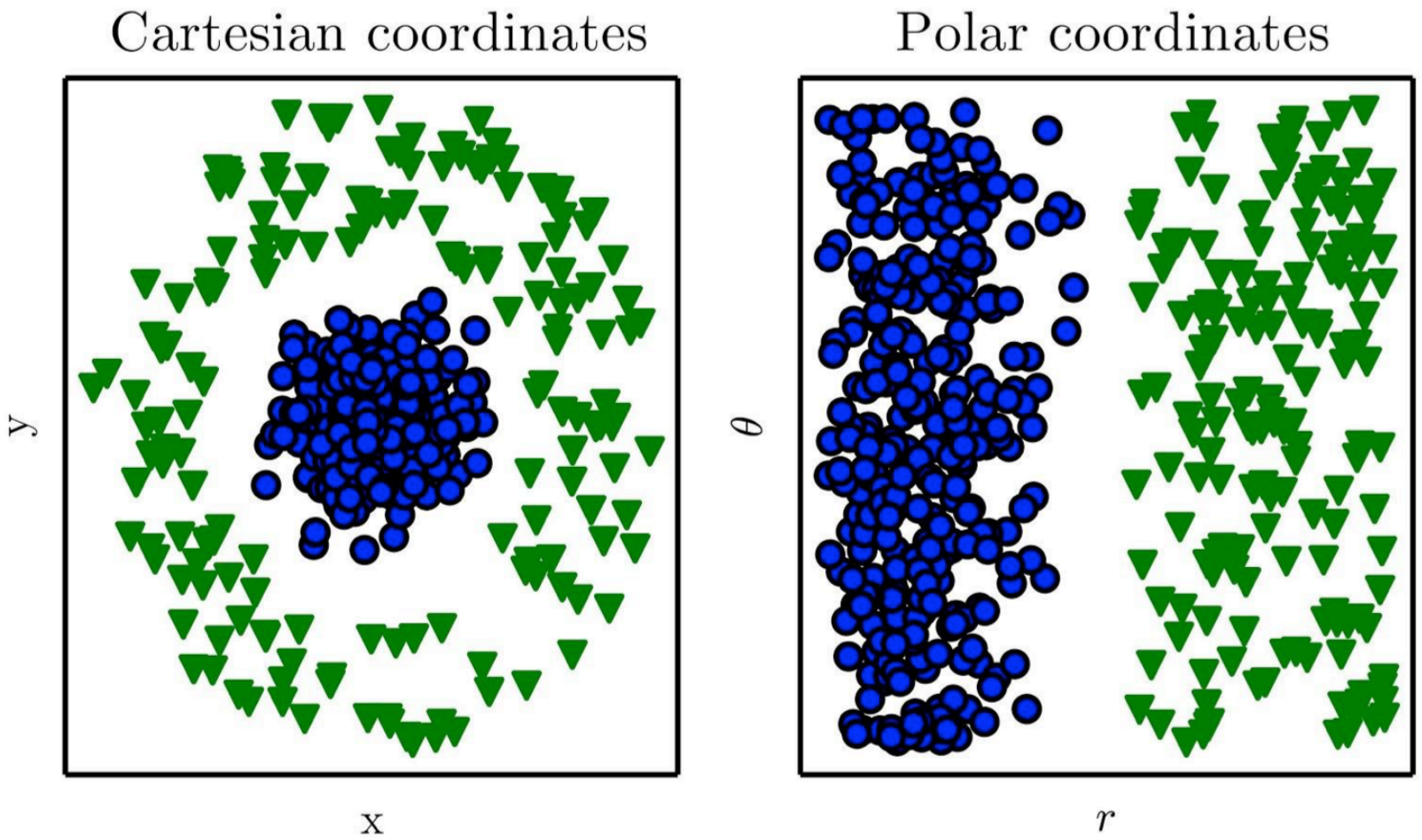


Class 1:
Three modes

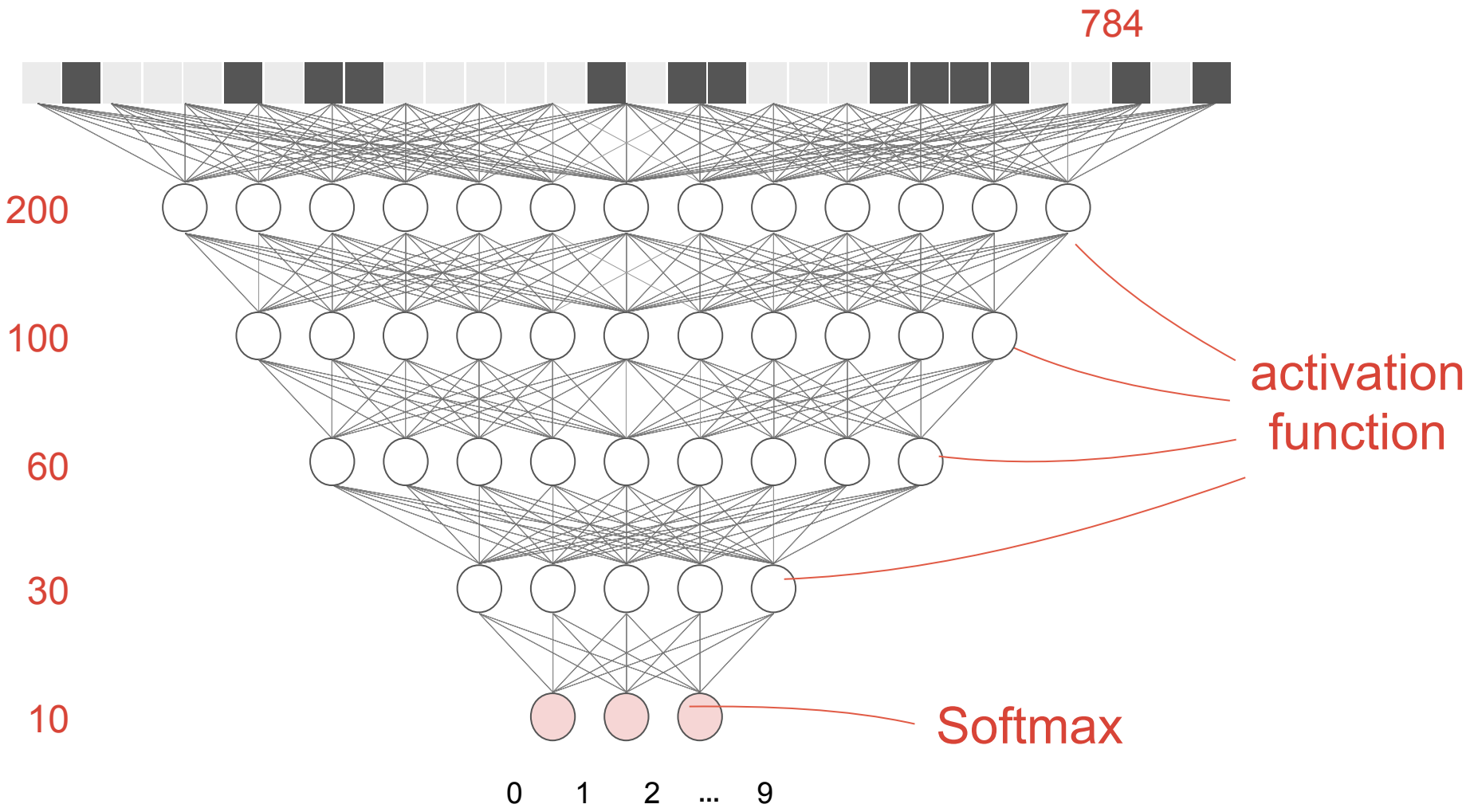
Class 2:
Everything else



Representation Matters



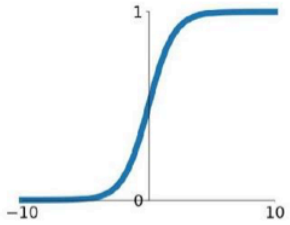
Going deeper w/ multilayer perceptron



Activation Functions

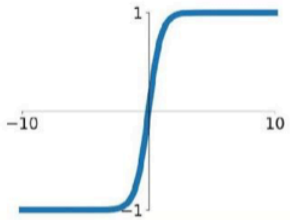
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



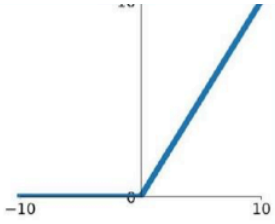
tanh

$$\tanh(x)$$



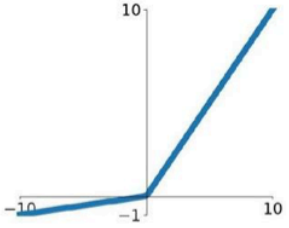
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

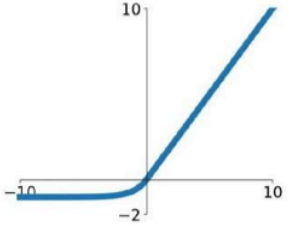


Maxout

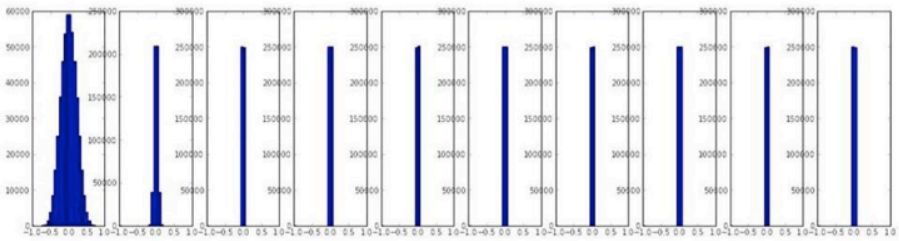
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

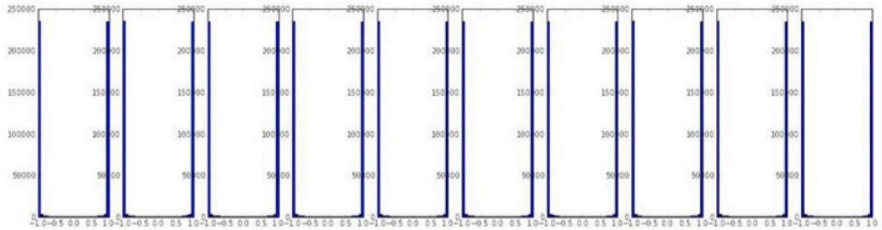
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



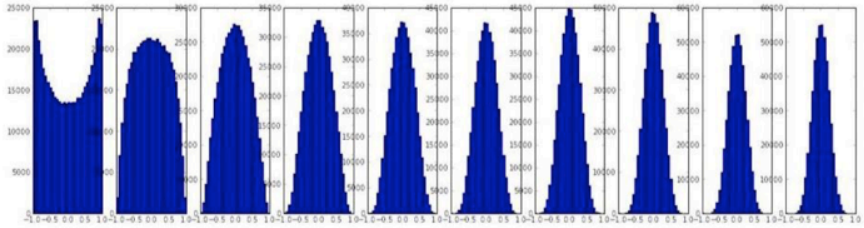
Weight Initialization



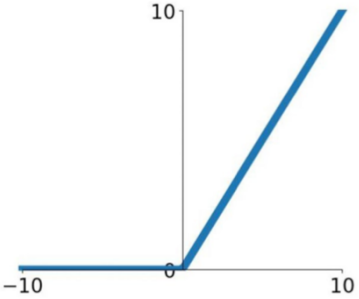
Initialization too small:
Activations go to zero, gradients also zero,
No learning



Initialization too big:
Activations saturate (for tanh),
Gradients zero, no learning



Initialization just right:
Nice distribution of activations at all layers,
Learning proceeds nicely




```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

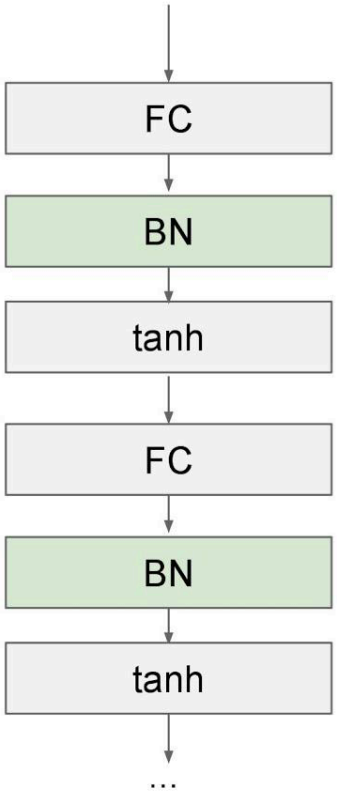
He et al., 2015
(note additional /2)

Batch Normalization

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations? just make them so."



Normalize:

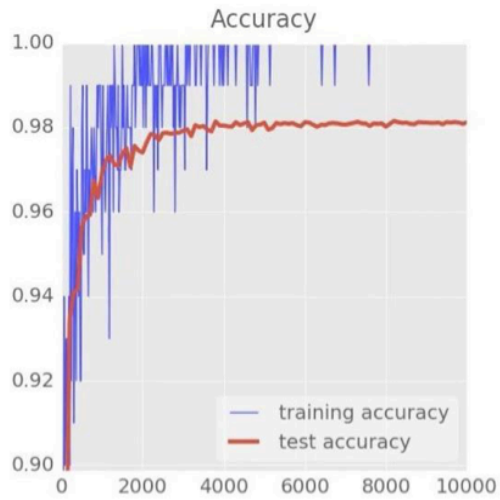
$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

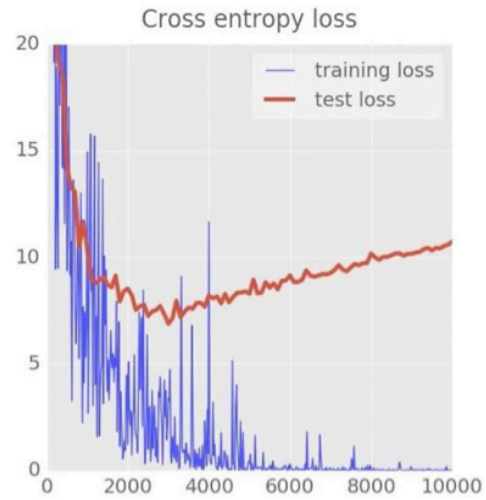
And then allow the network to squash the range if it wants to:

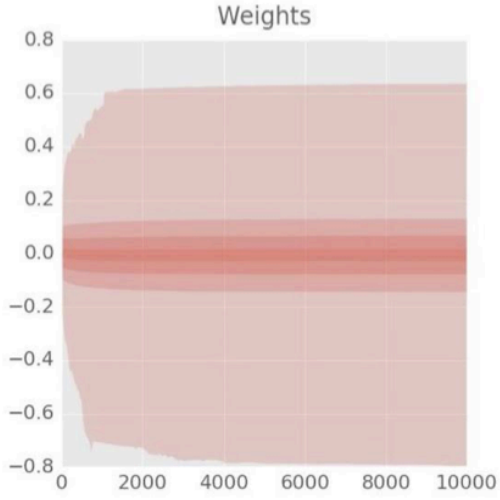
$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

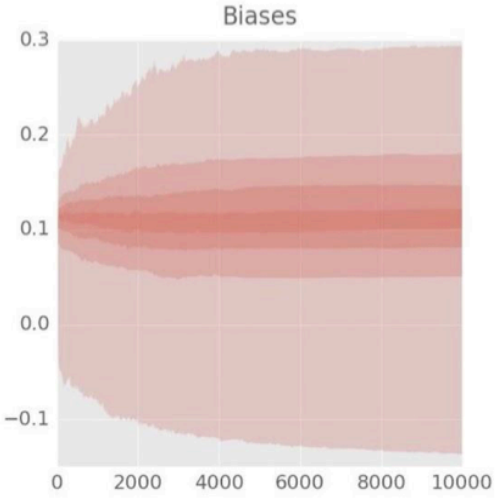
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization

MNIST handwritten digit database



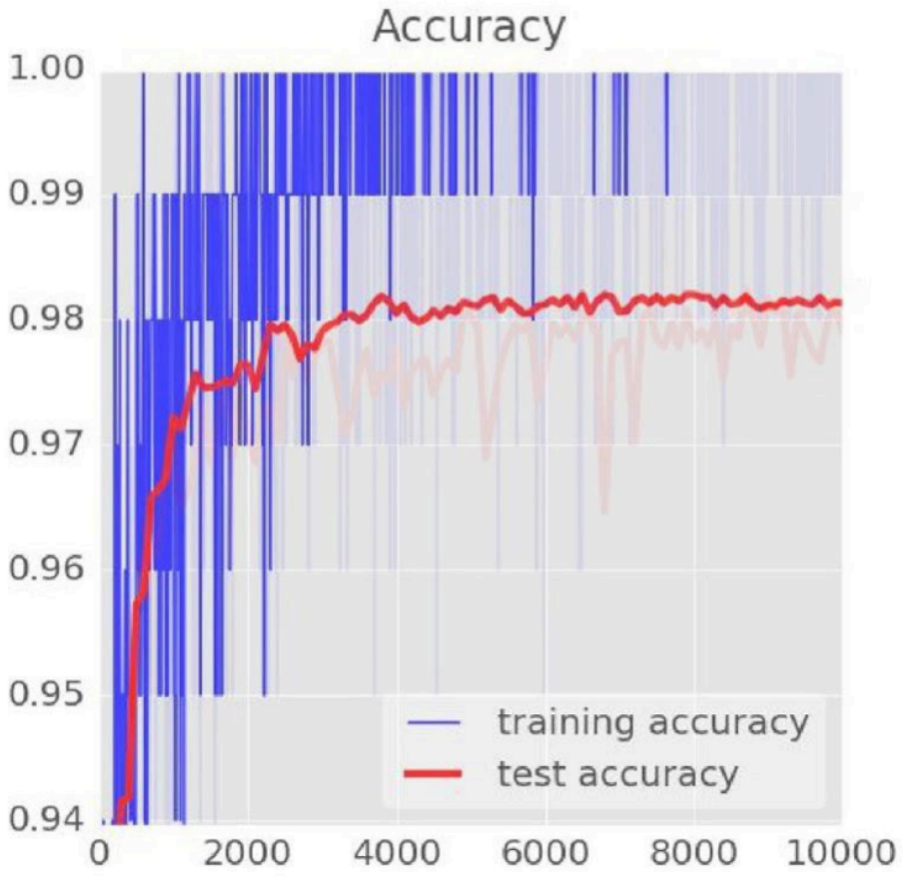


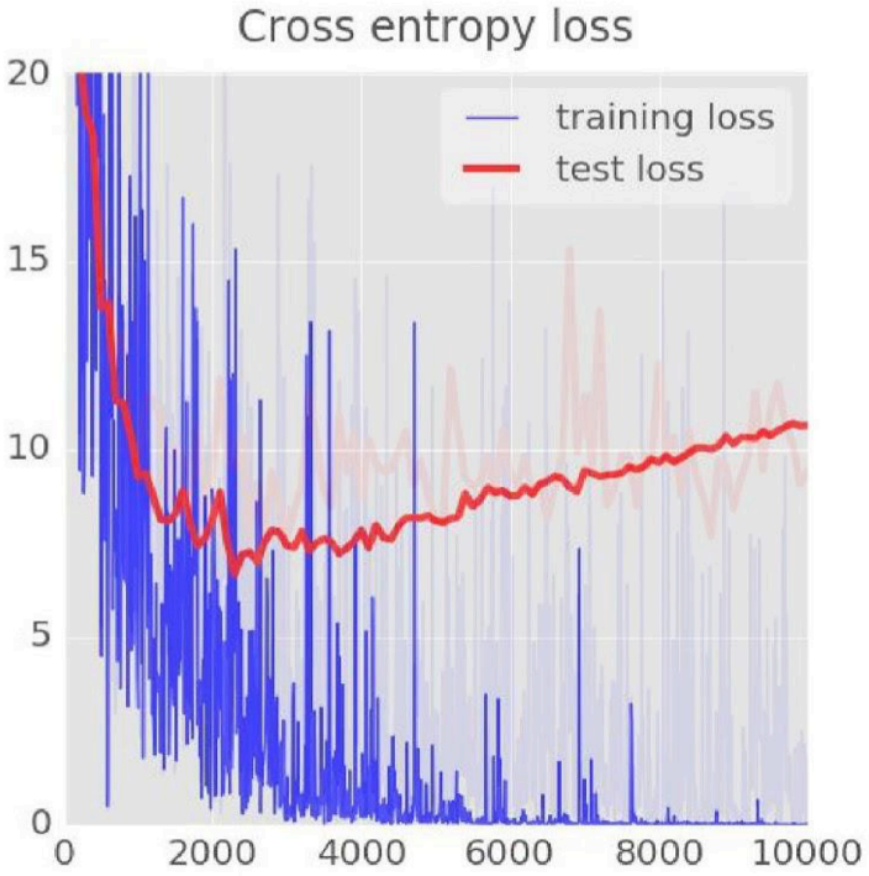




MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

Overfitting



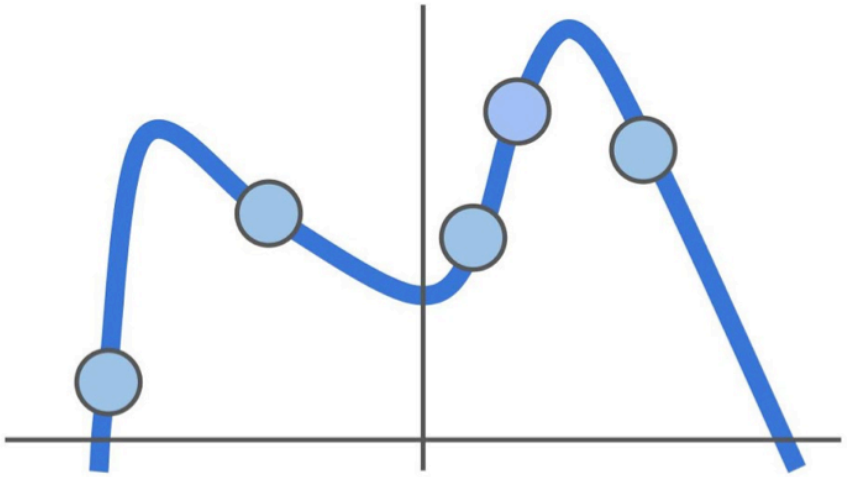


Regularization

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

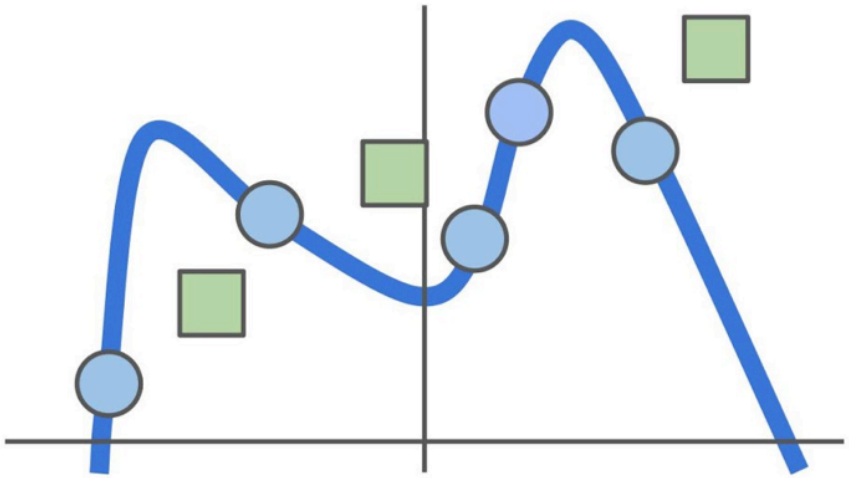
Data loss: Model predictions should match training data



Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

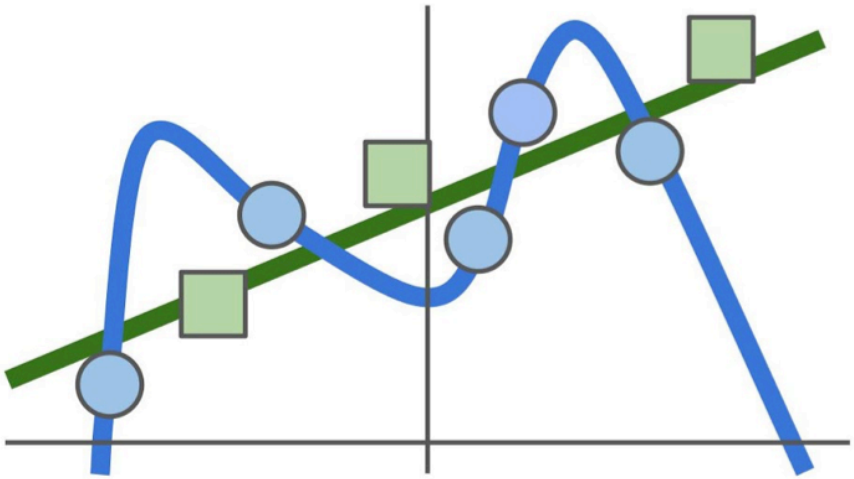
Data loss: Model predictions should match training data



Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

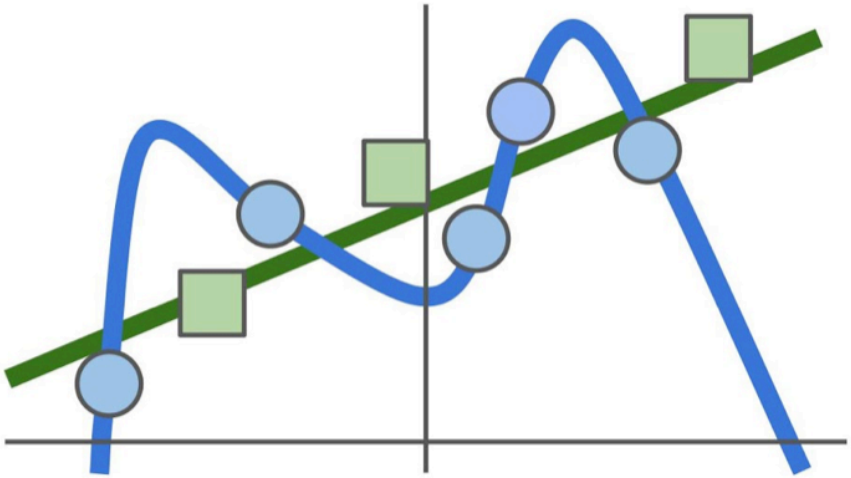


Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data

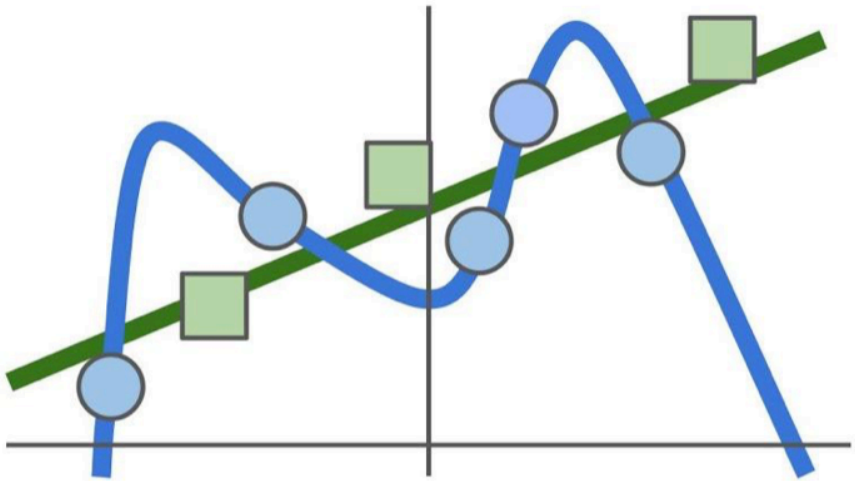


Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

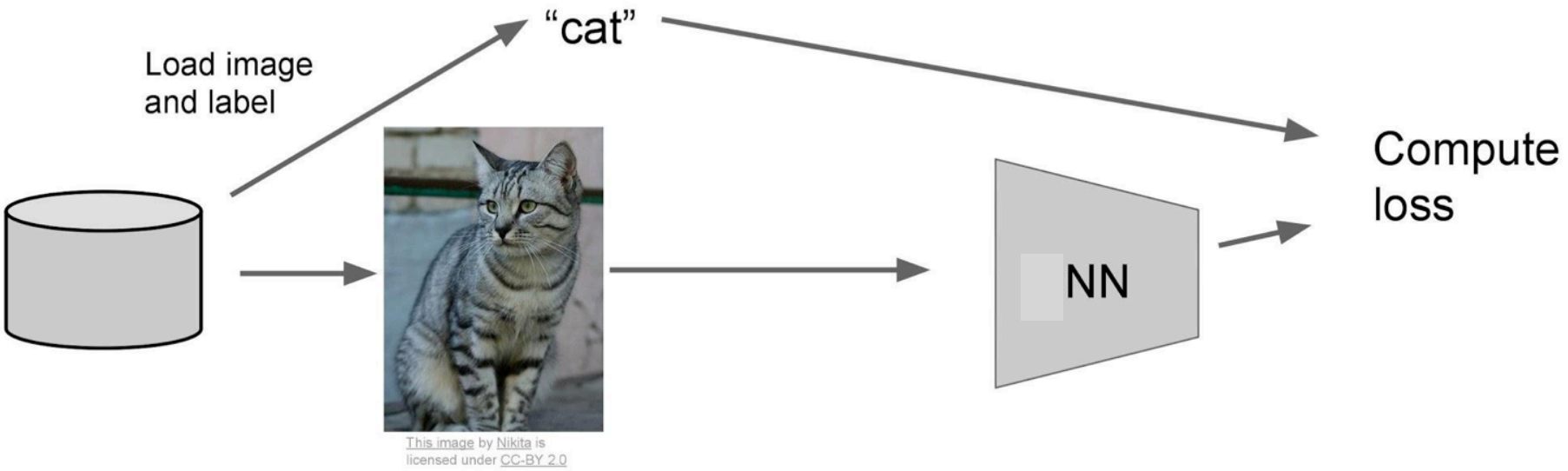
Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data

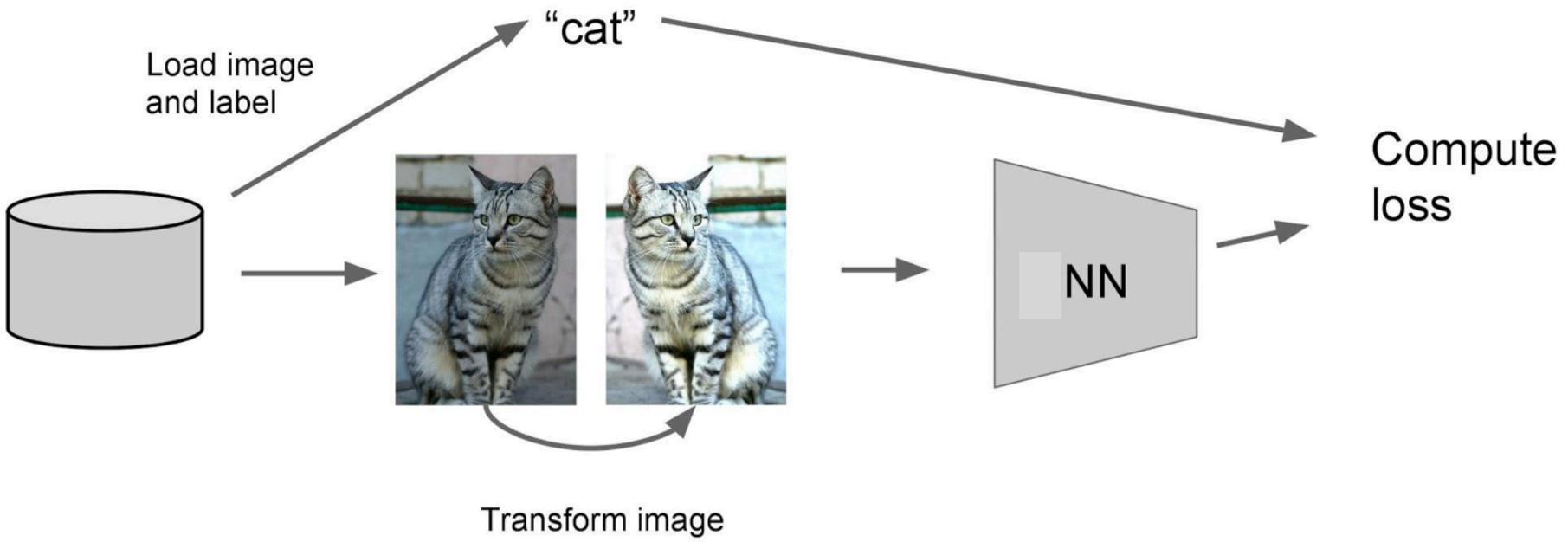


Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

Regularization: Data Augmentation



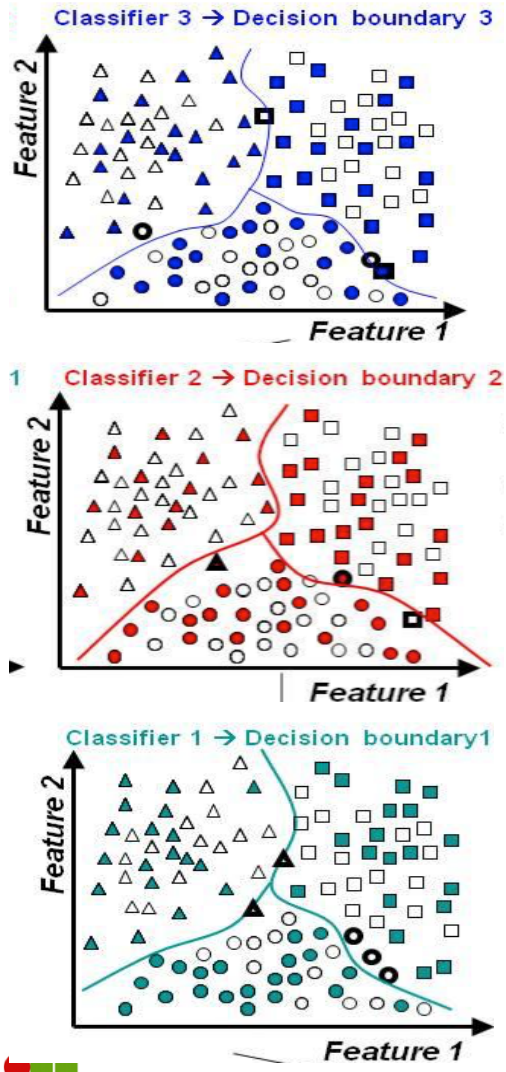
Regularization: Data Augmentation

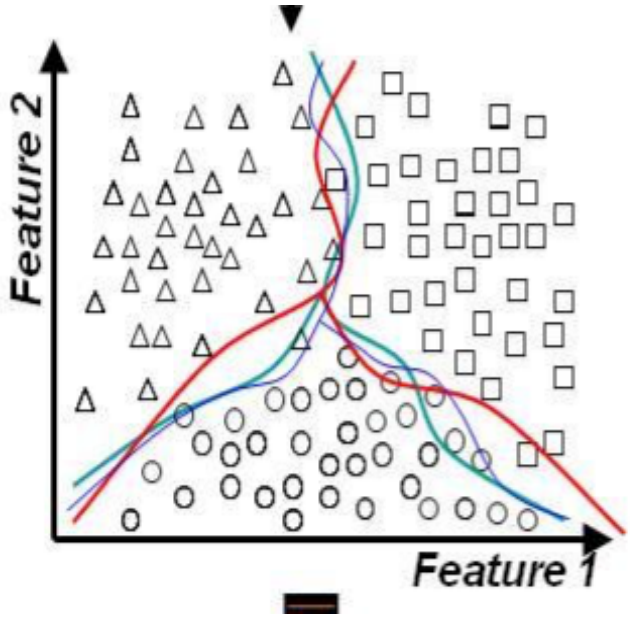


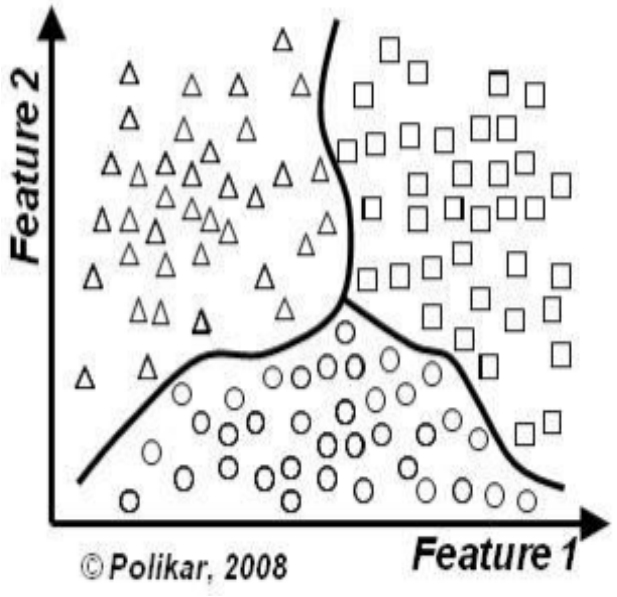
Random mix/combinations of:

- Translation
- Rotation
- Stretching
- Shearing
- Lens distortions
- Get creative

Regularization: Bagging Models

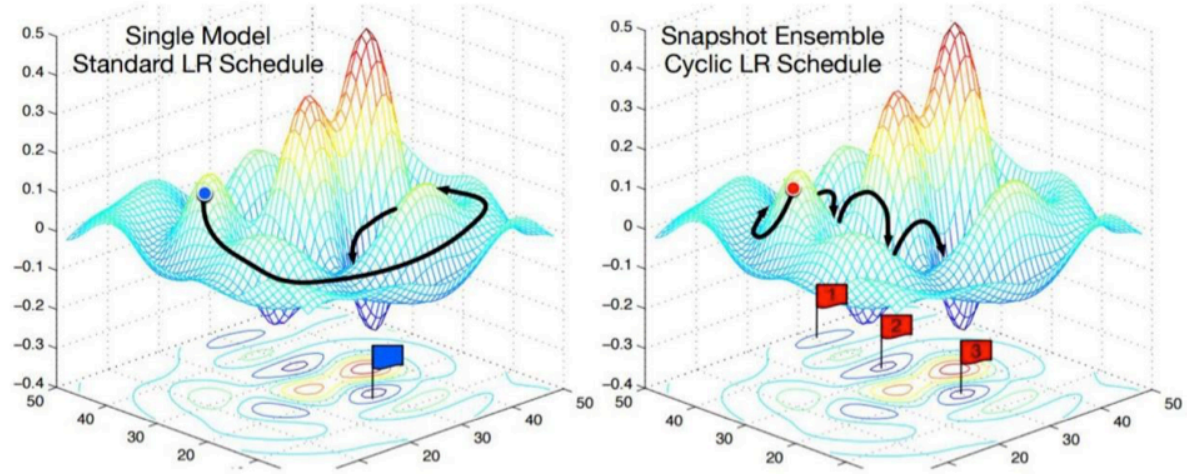


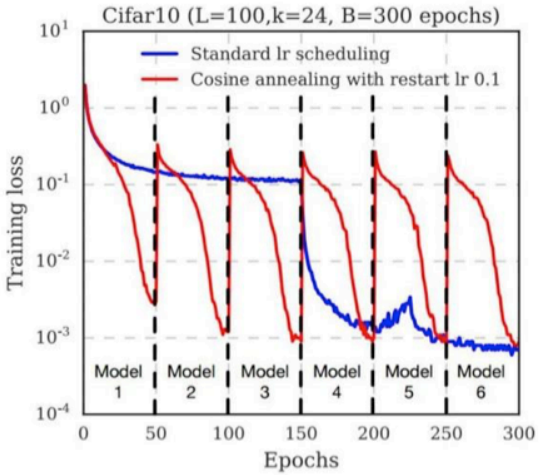




Regularization: Model Ensembles

Instead of training independent models, use multiple snapshots of a single model during training!



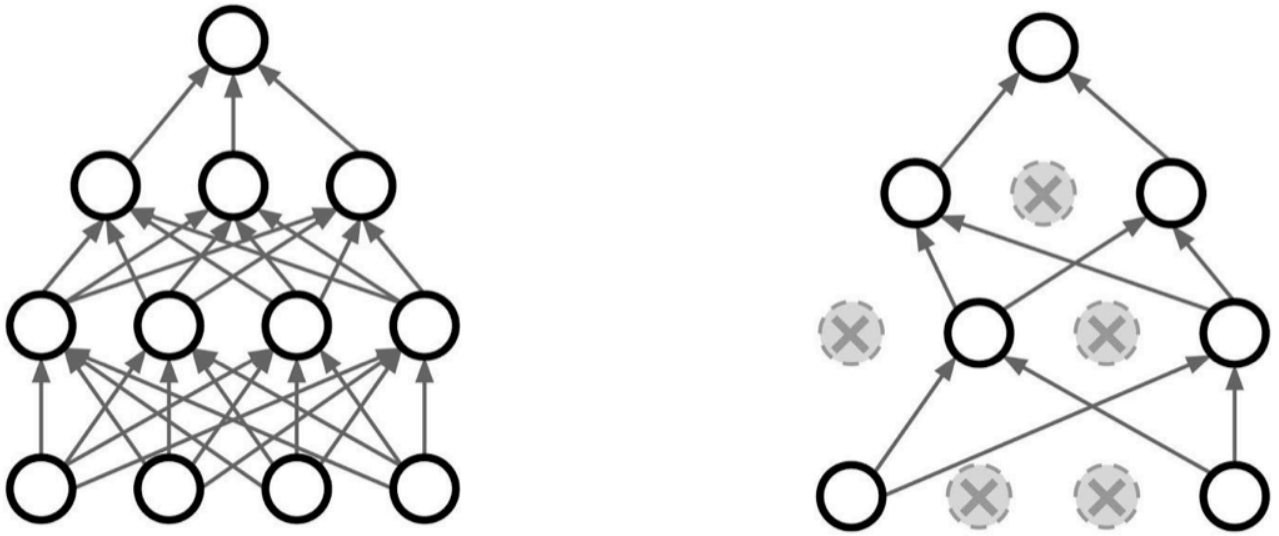


Cyclic learning rate schedules can make this work even better!

Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Regularization: Dropout

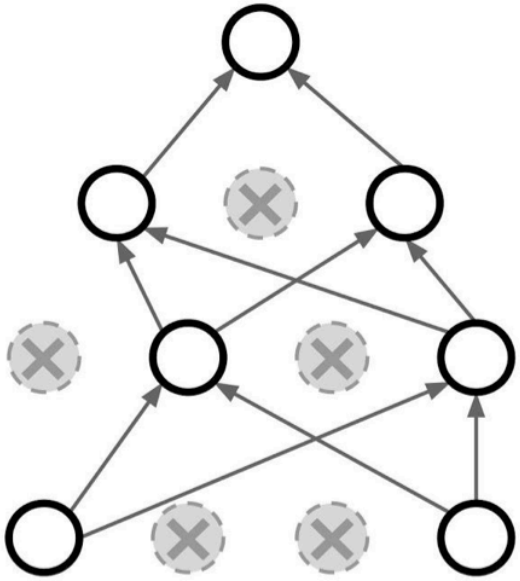
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

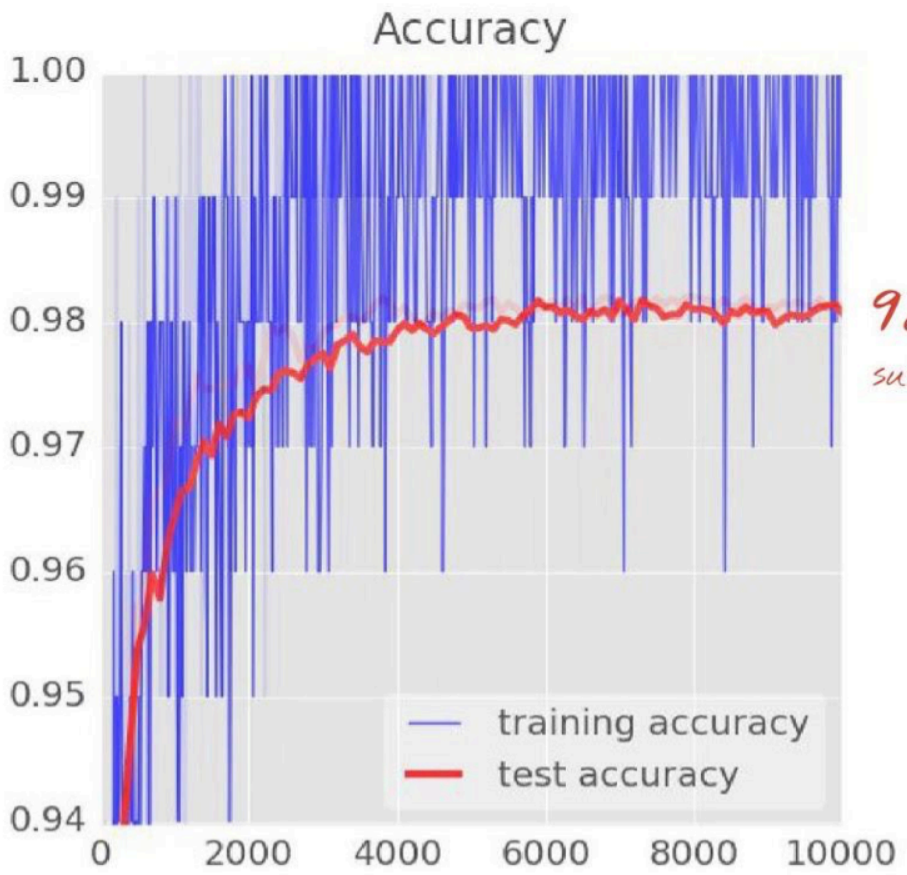
Regularization: Dropout

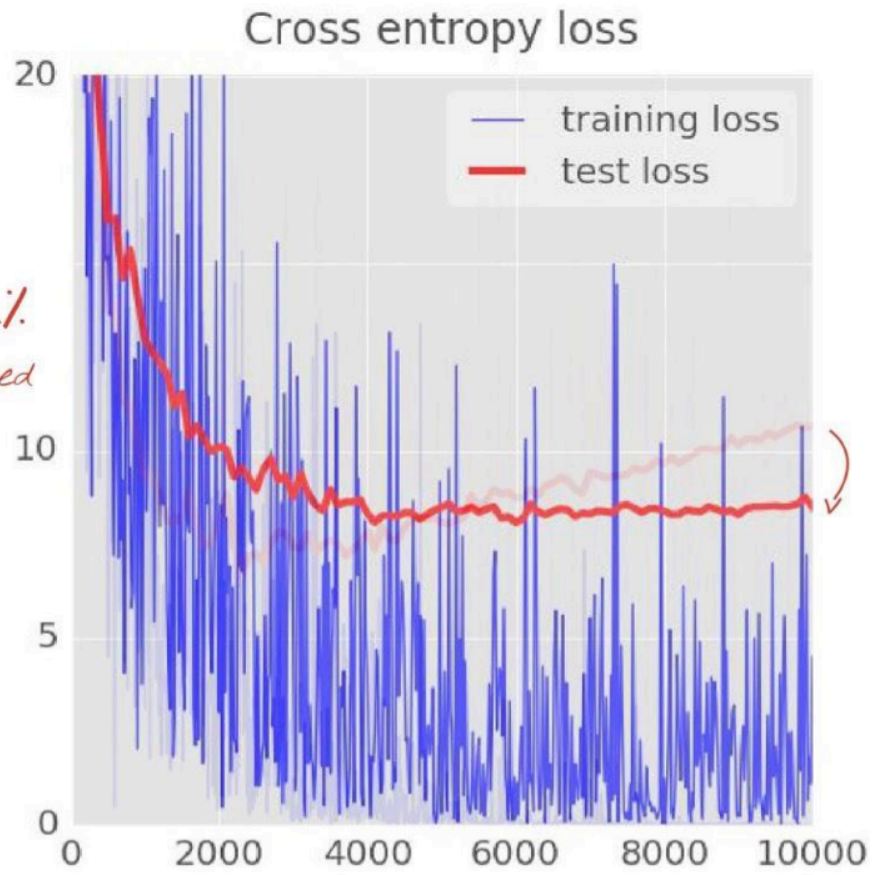
How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

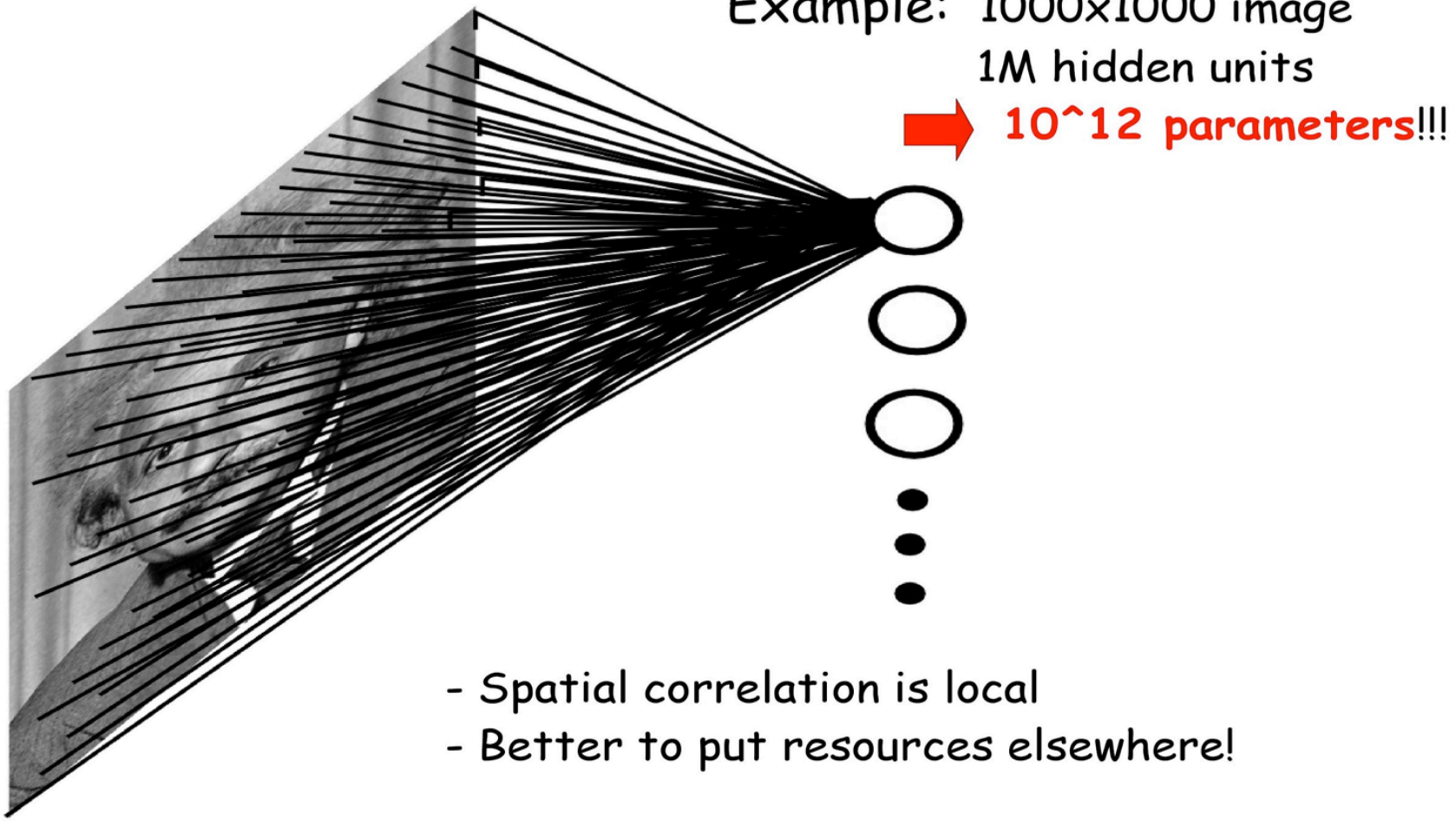
MLP with Dropout



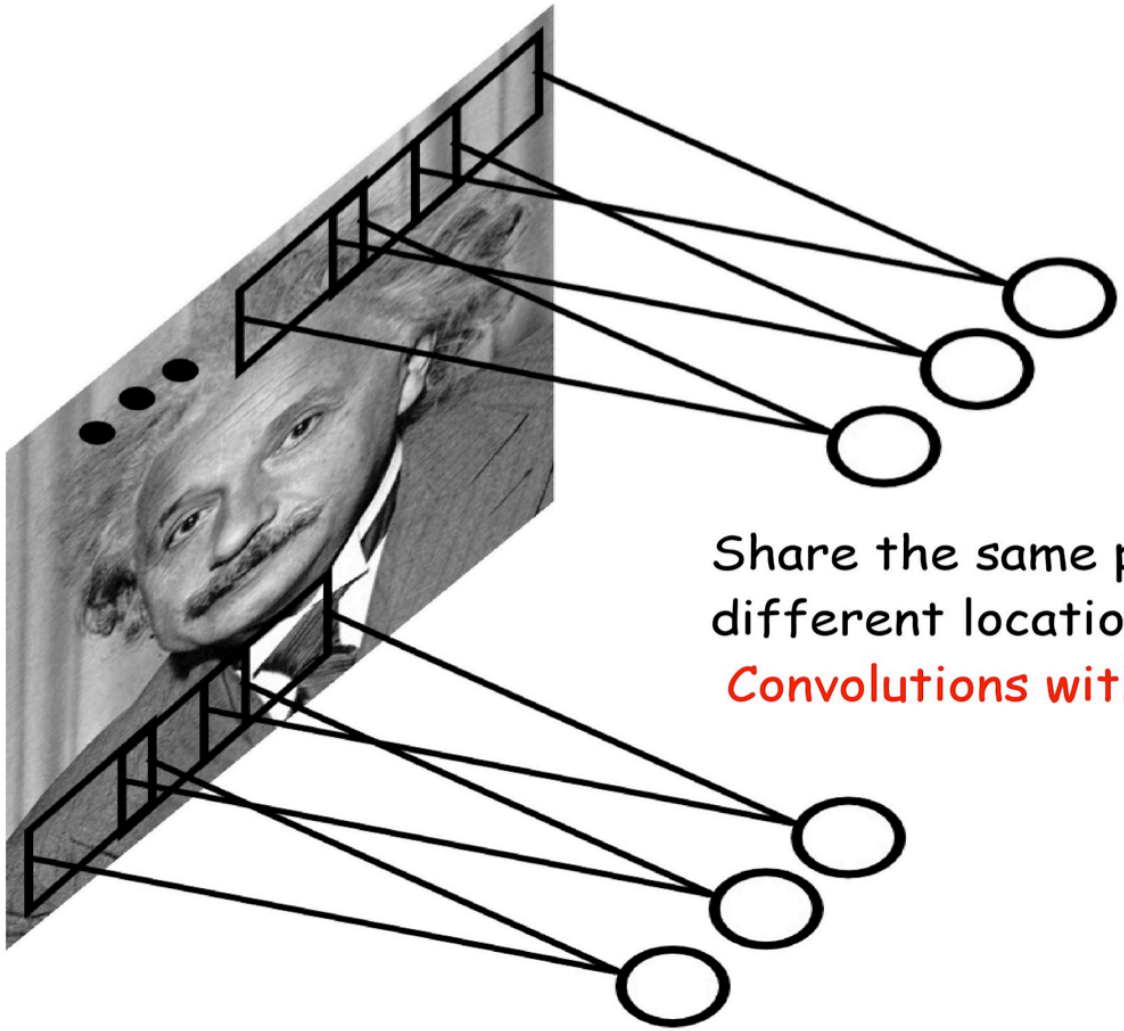


Convolutional Neural Networks

Kinds of Connectivity



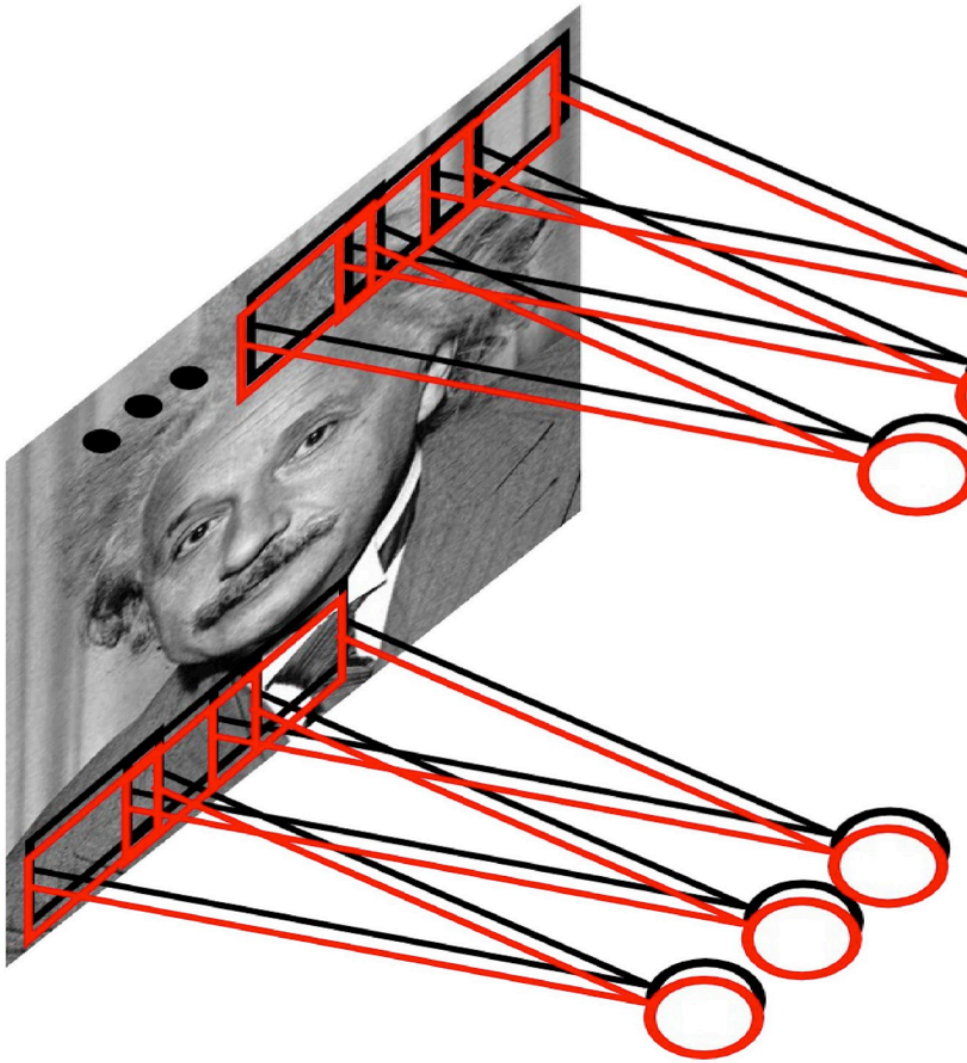
Convolutional Net



Share the same parameters across different locations:

Convolutions with learned kernels

Convolutional Net



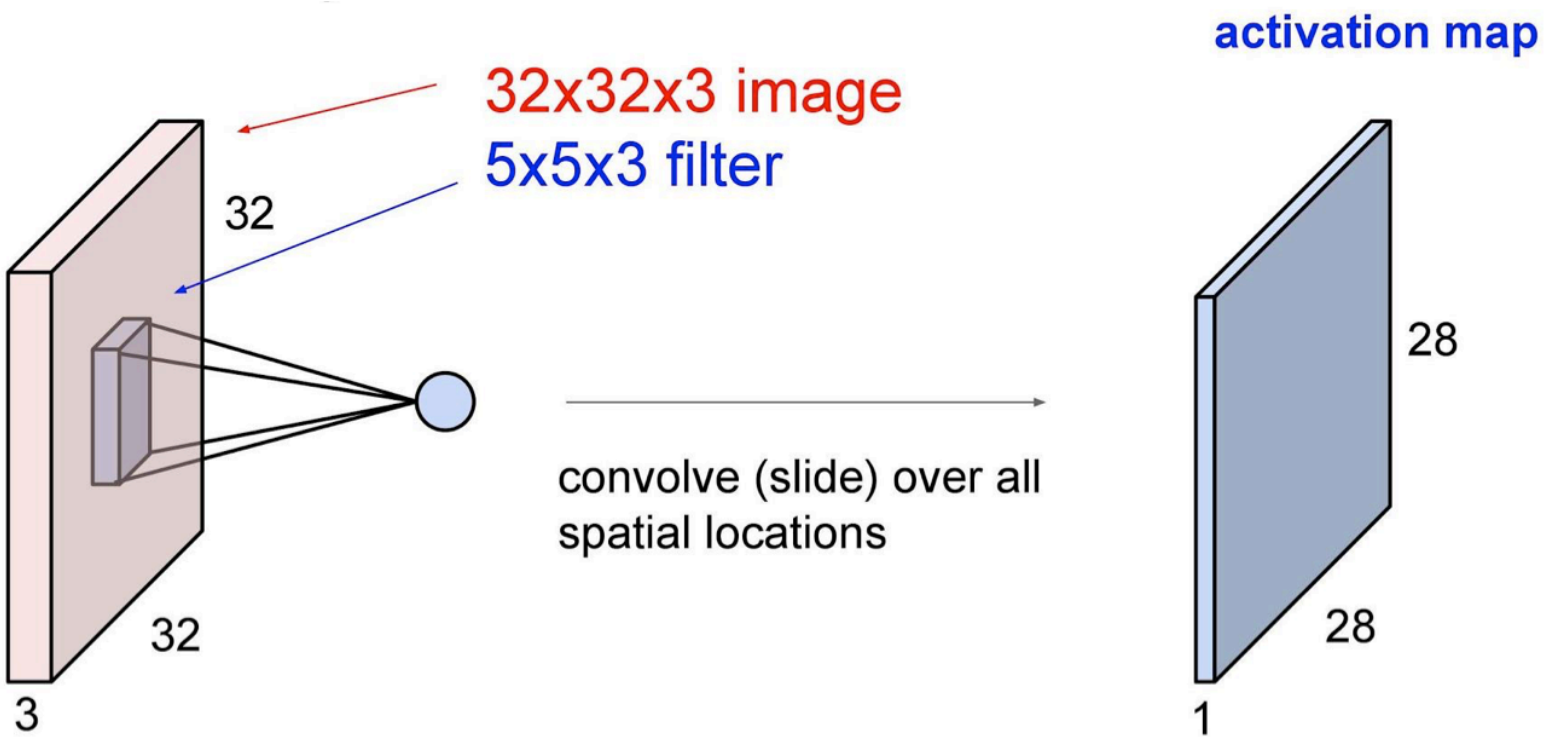
Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Feature Map

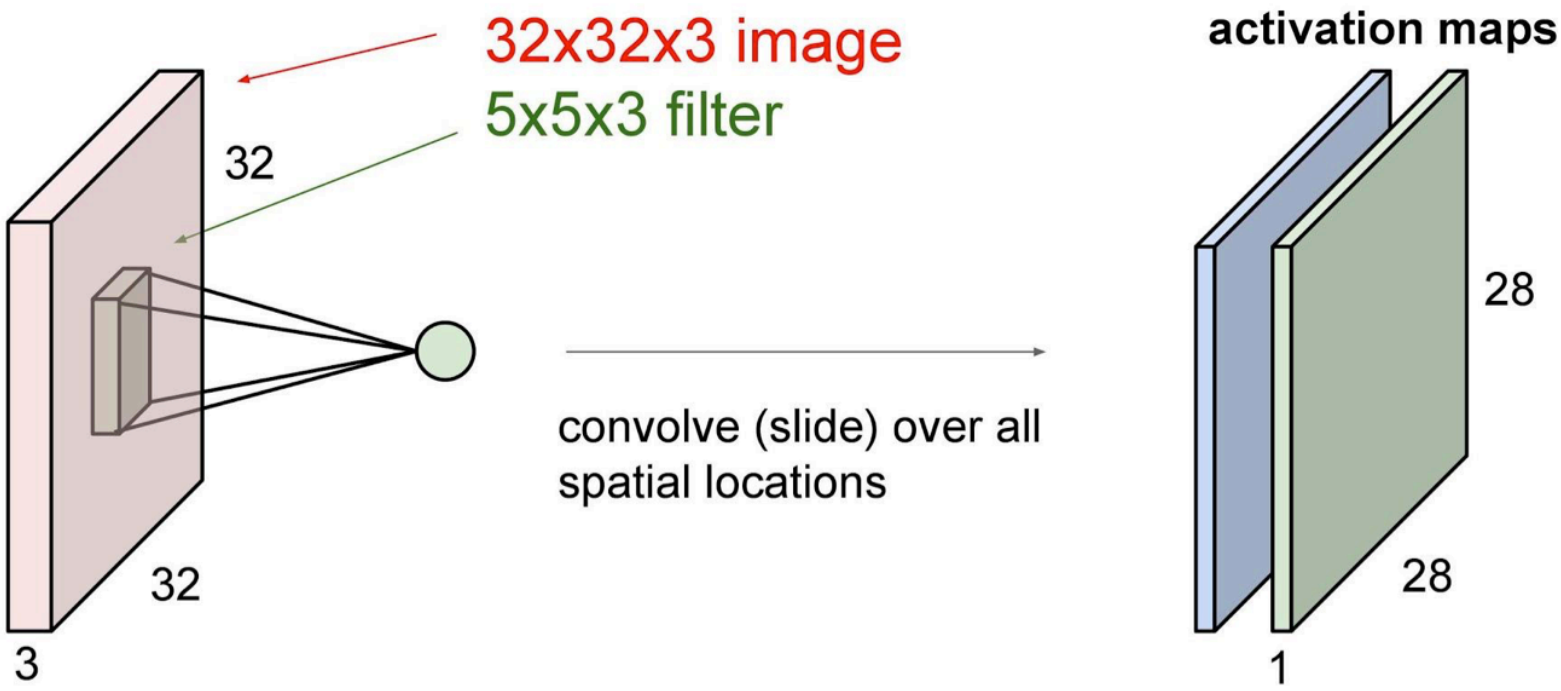
Input

Convolution Layer



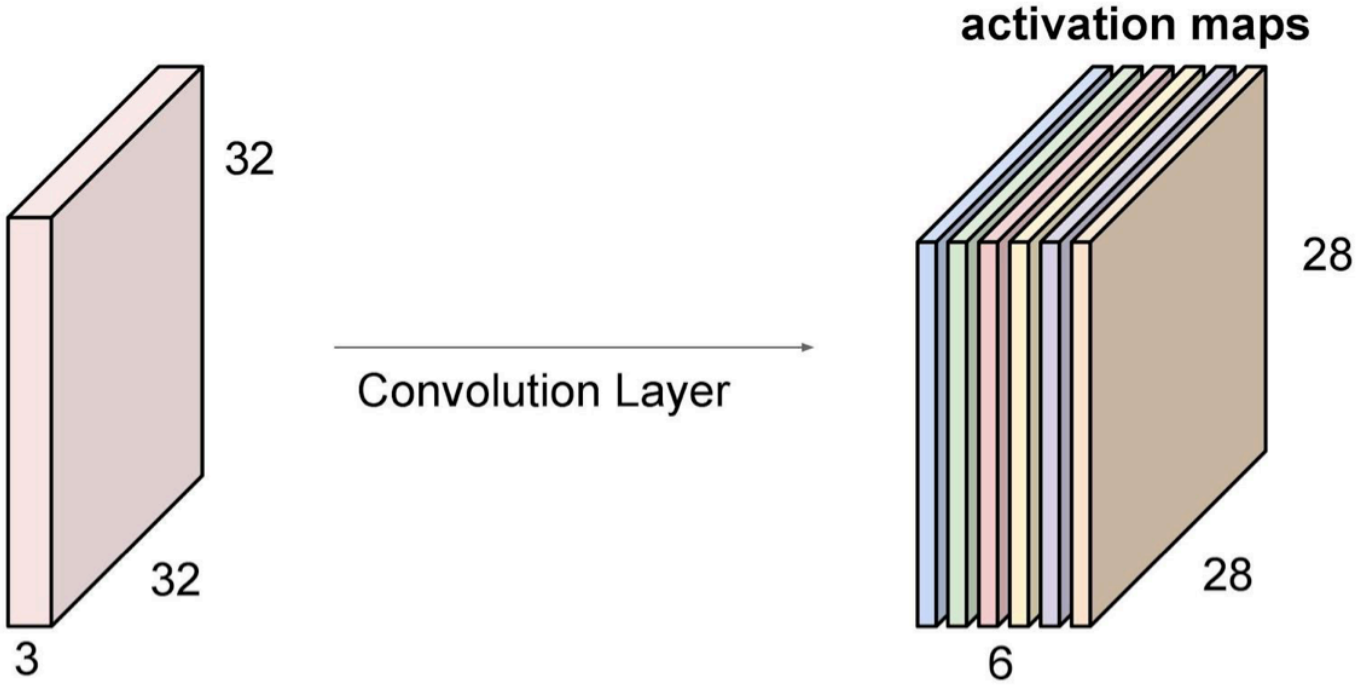
Convolution Layer

consider a second, green filter



Convolution Layer

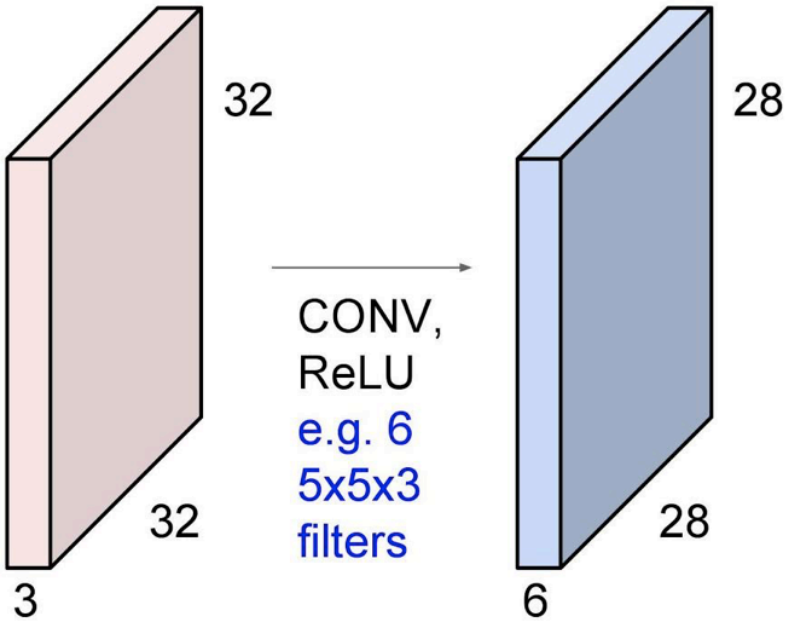
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

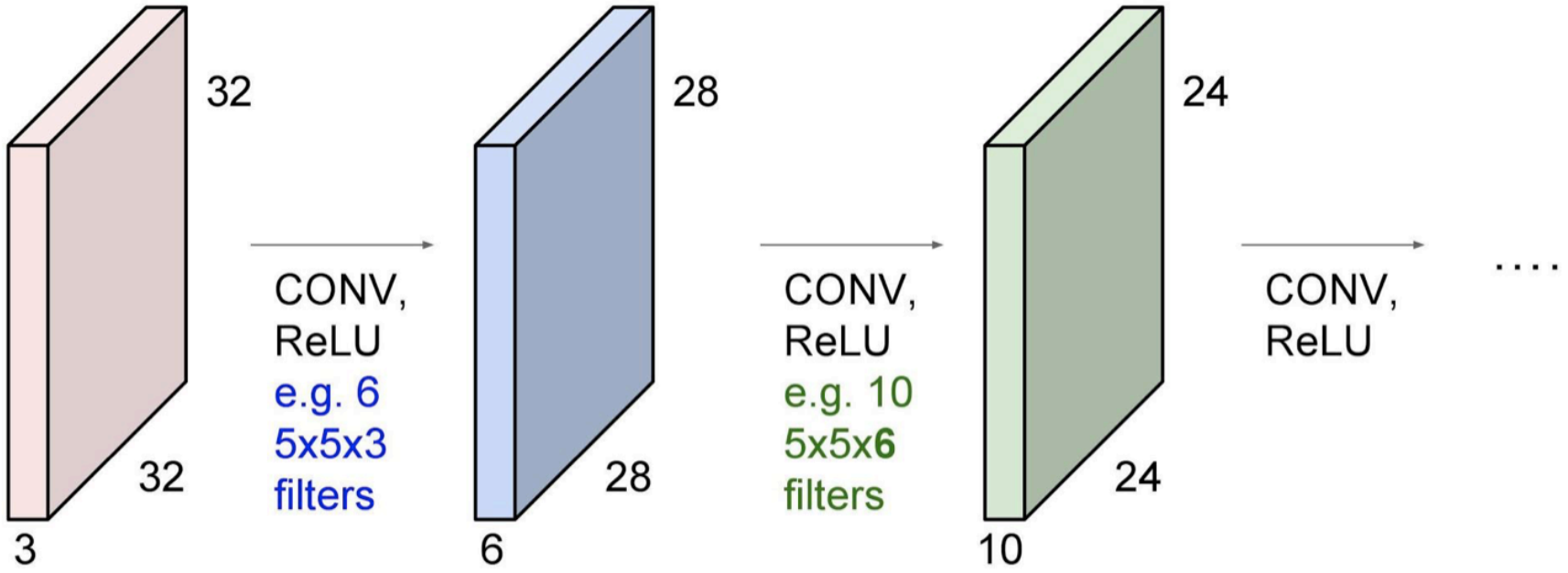
ConvNet / CNN

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

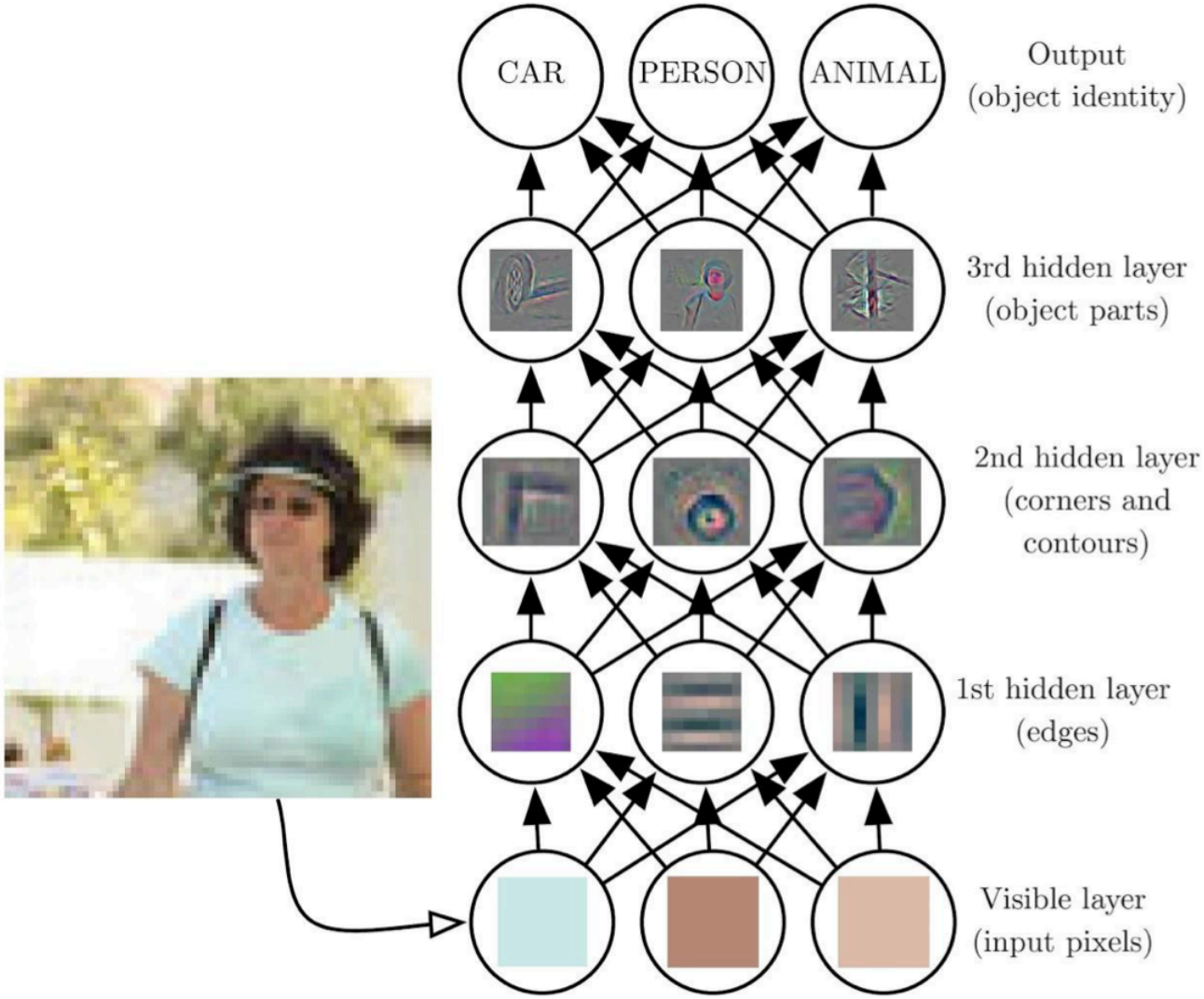


ConvNet / CNN

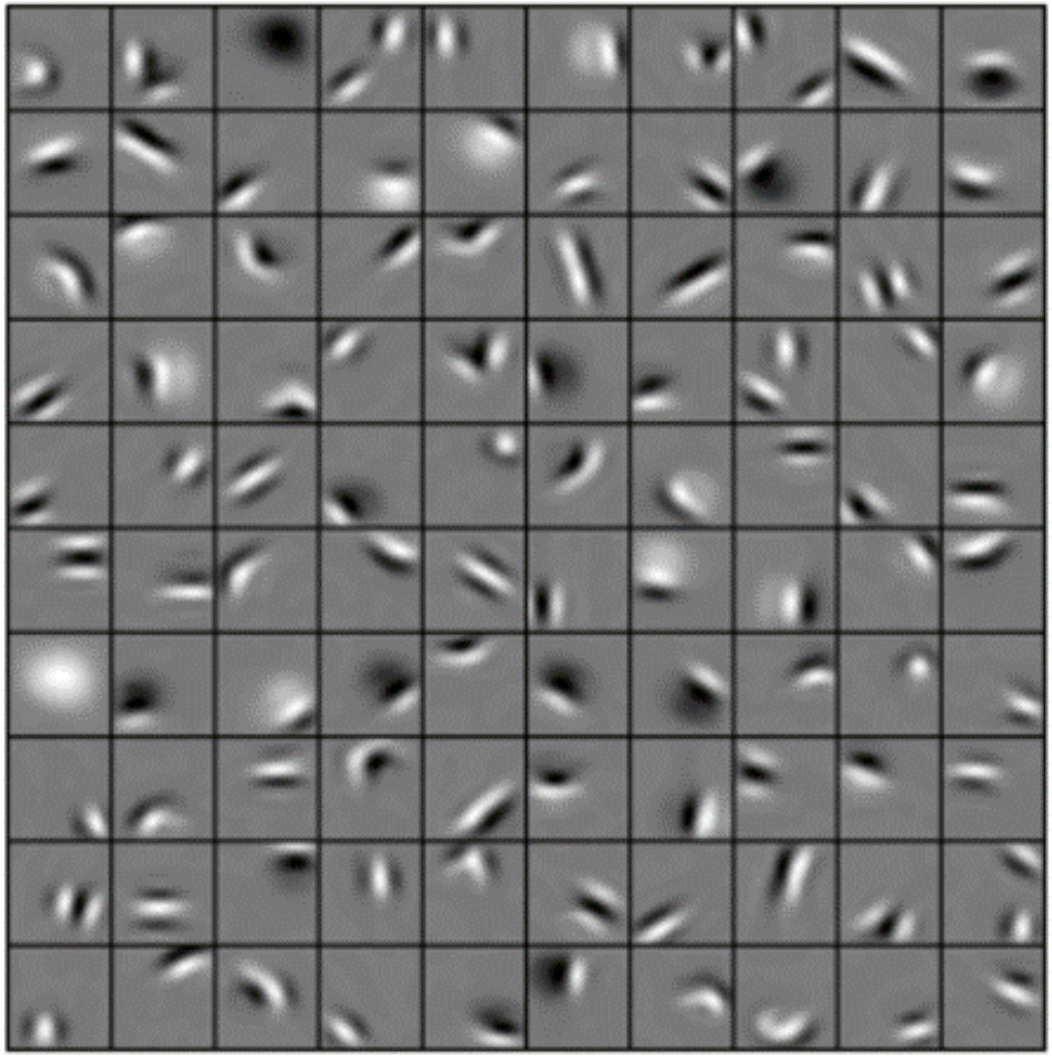
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

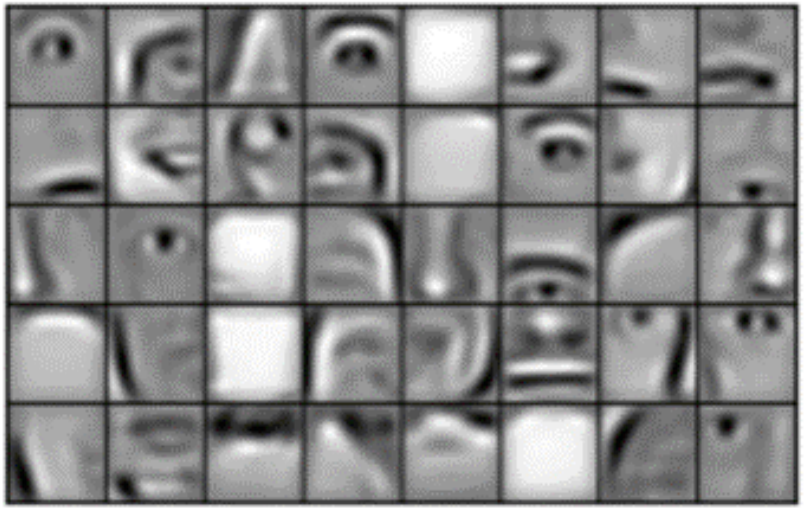


Depth: Repeated Composition

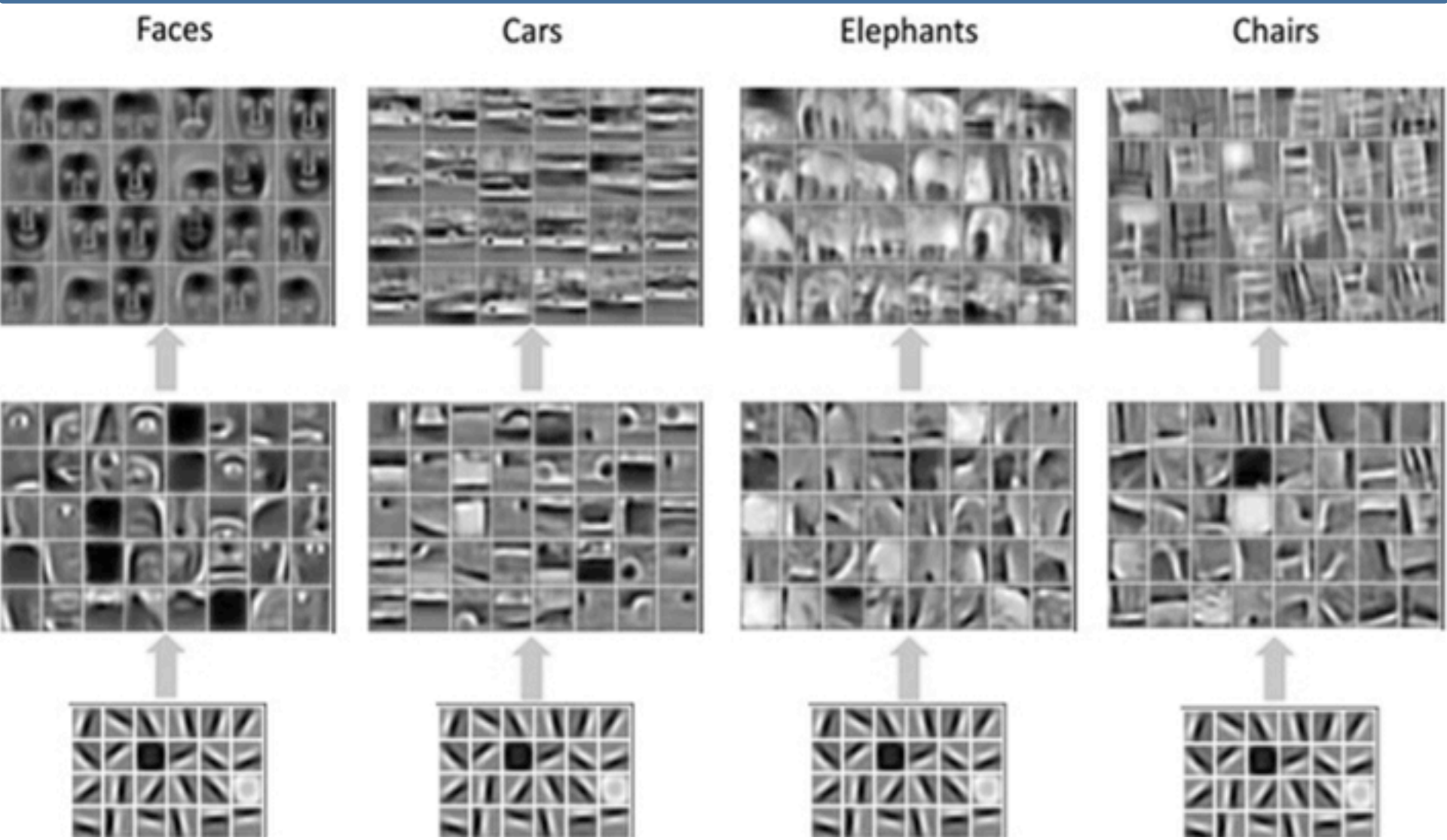


Hierarchical Features





Hierarchical Features



Biologically inspired?

A bit of history:

Hubel & Wiesel,

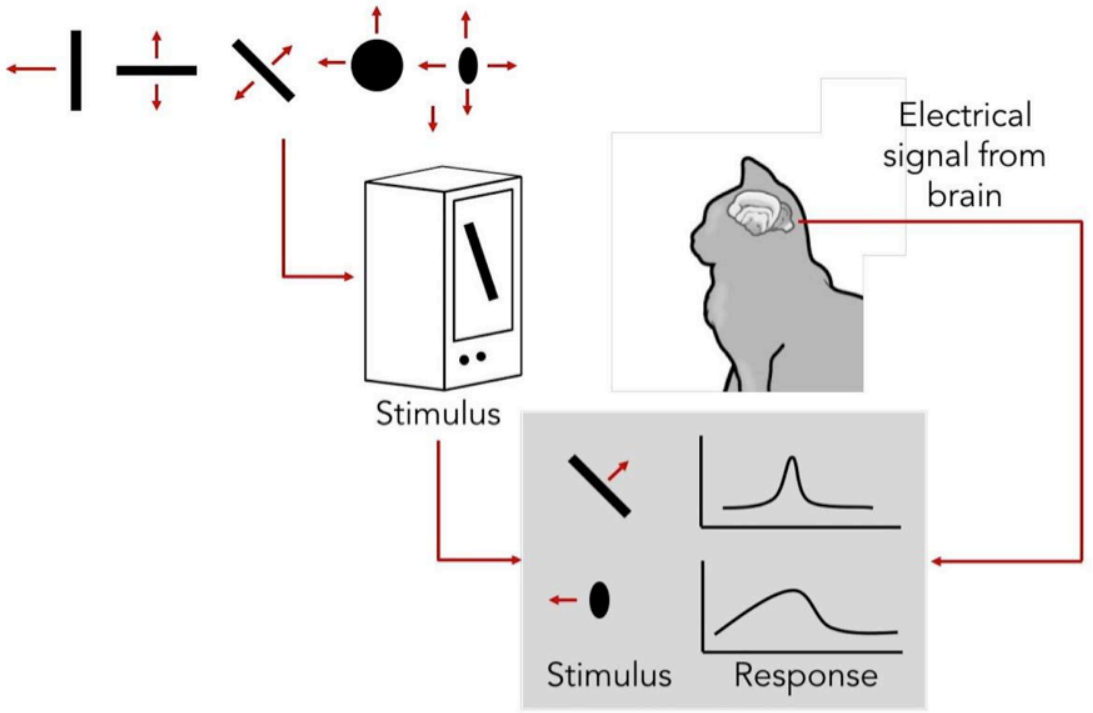
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

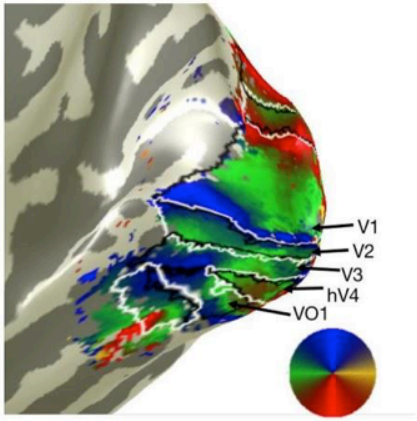
1968...

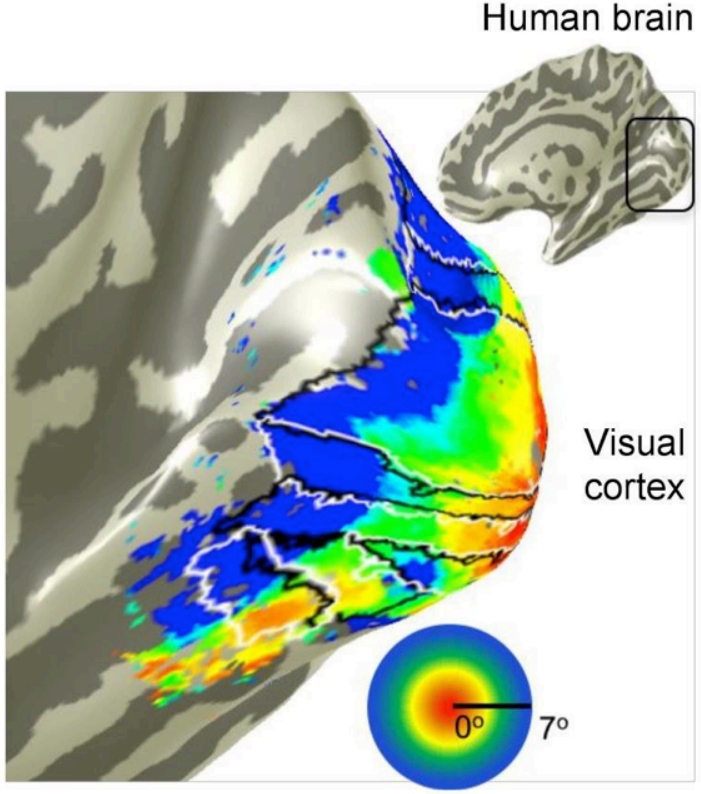


Biologically inspired?

A bit of history

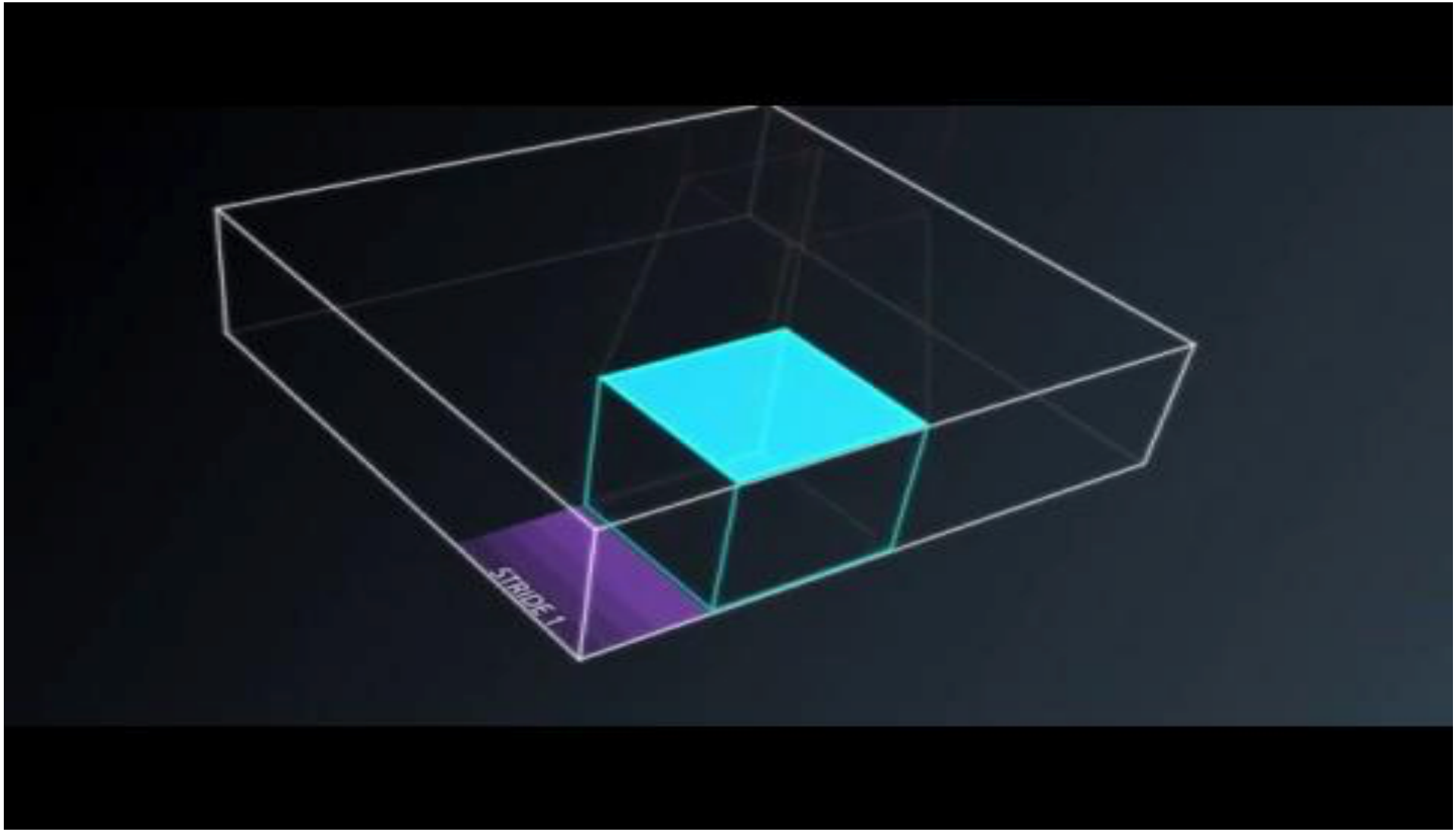
Topographical mapping in the cortex:
nearby cells in cortex represent
nearby regions in the visual field





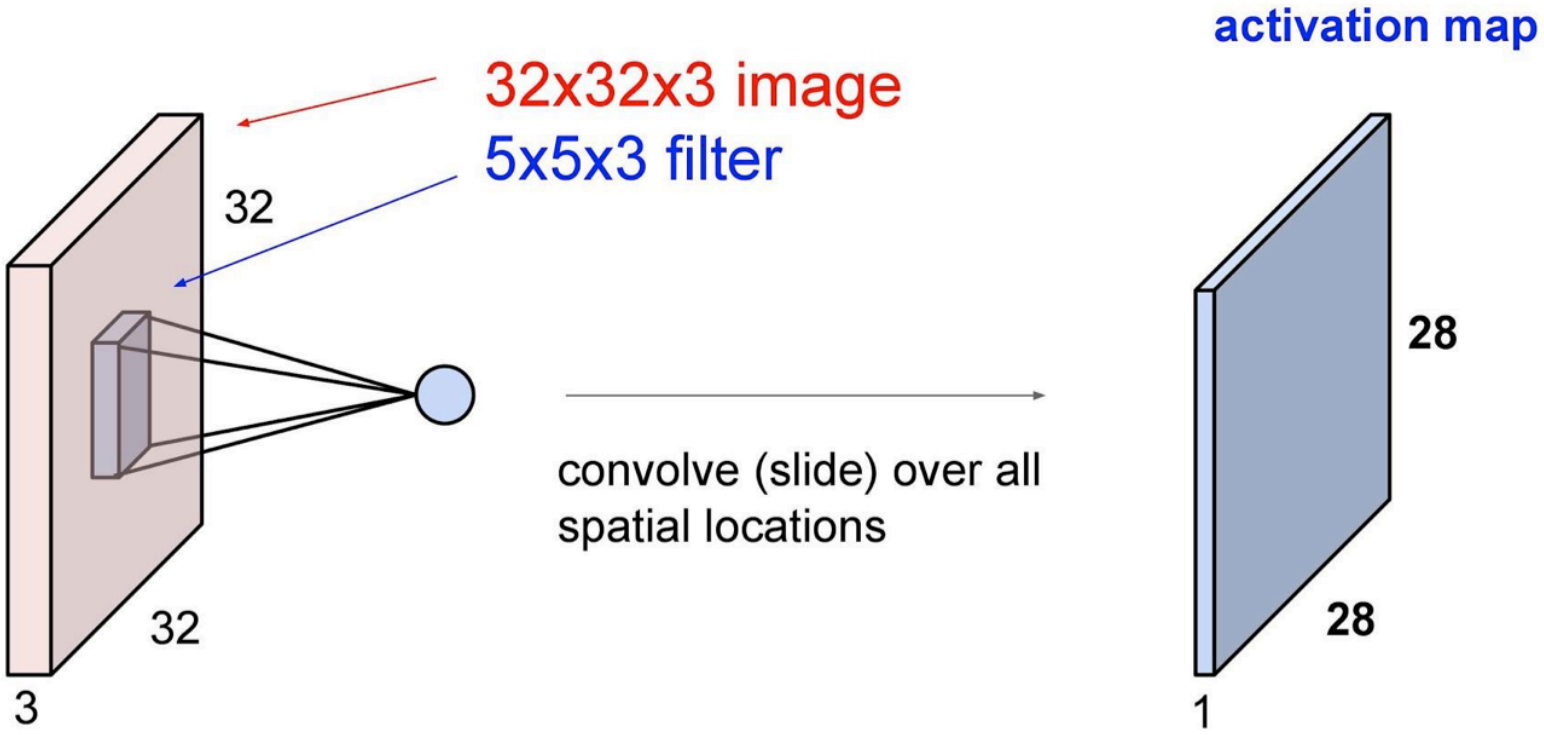
Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

ConvNets By Google



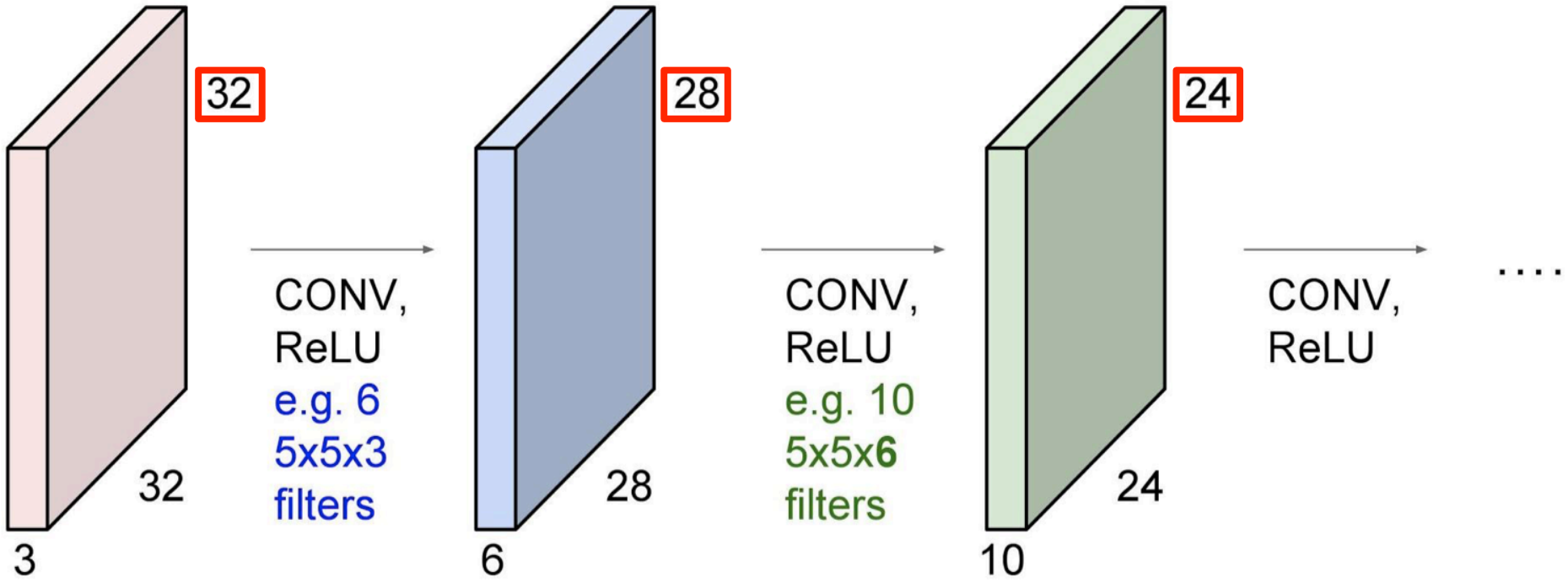
ConvNets

A closer look at spatial dimensions:

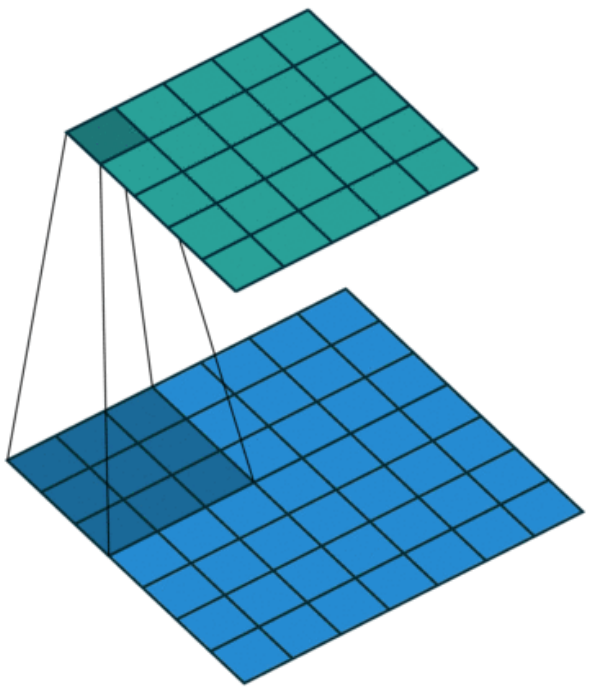


ConvNet / CNN

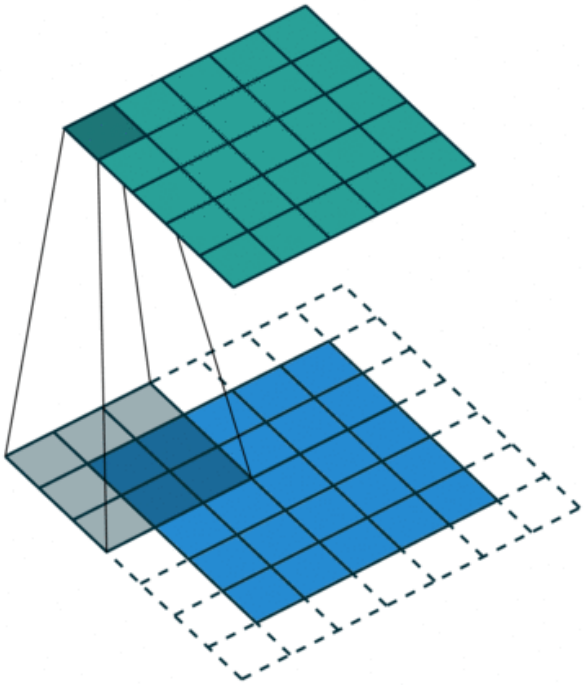
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



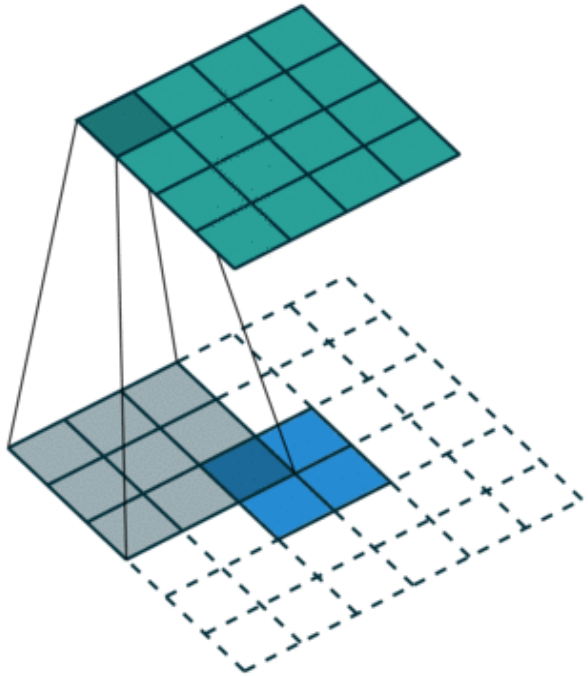
Padding



VALID

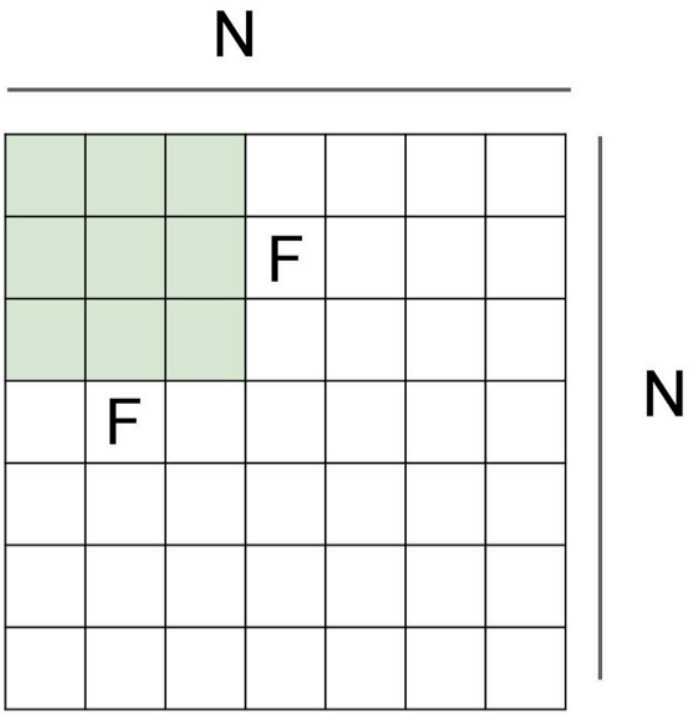


SAME



CUSTOM

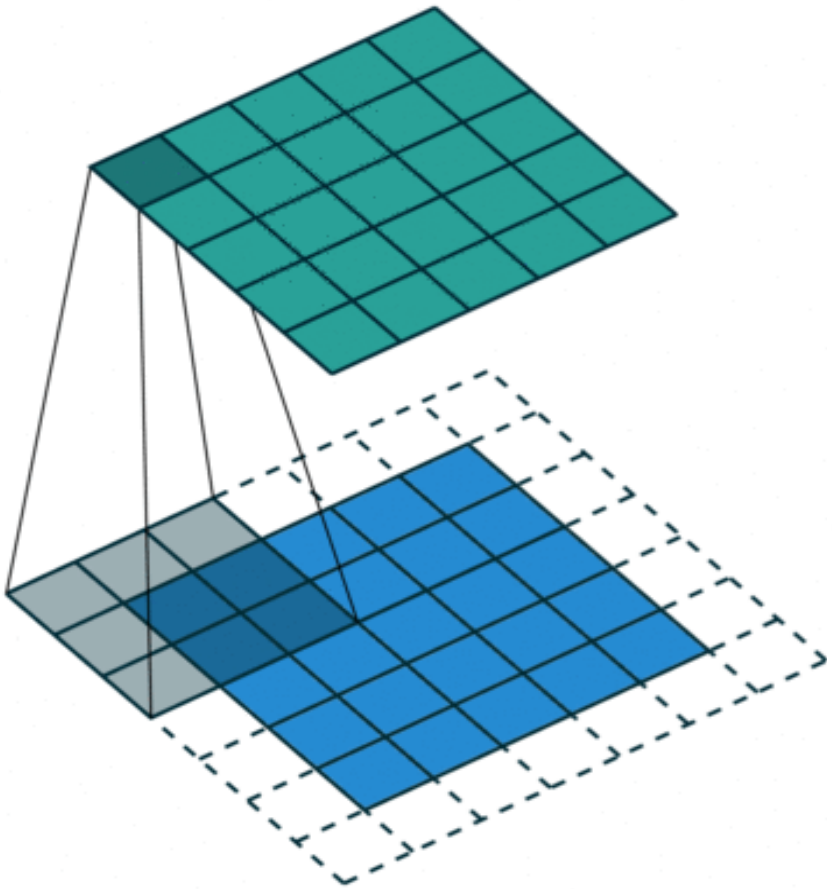
Stride



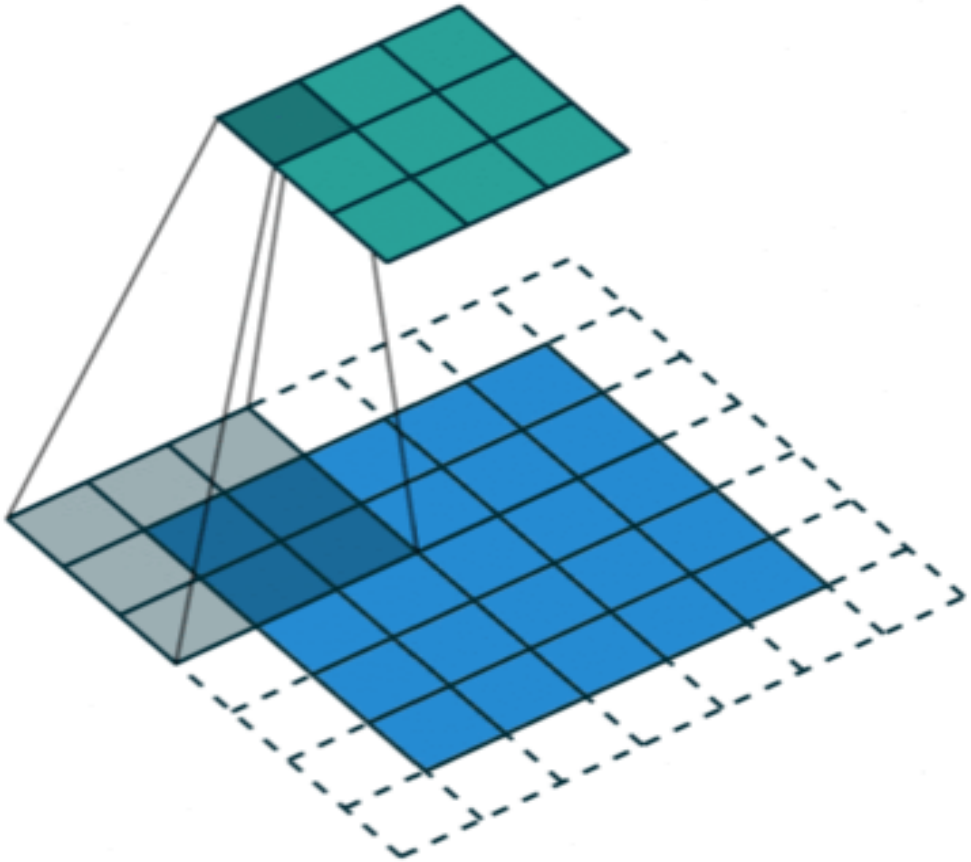
Output size:
(N - F) / stride + 1

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 :\

Stride

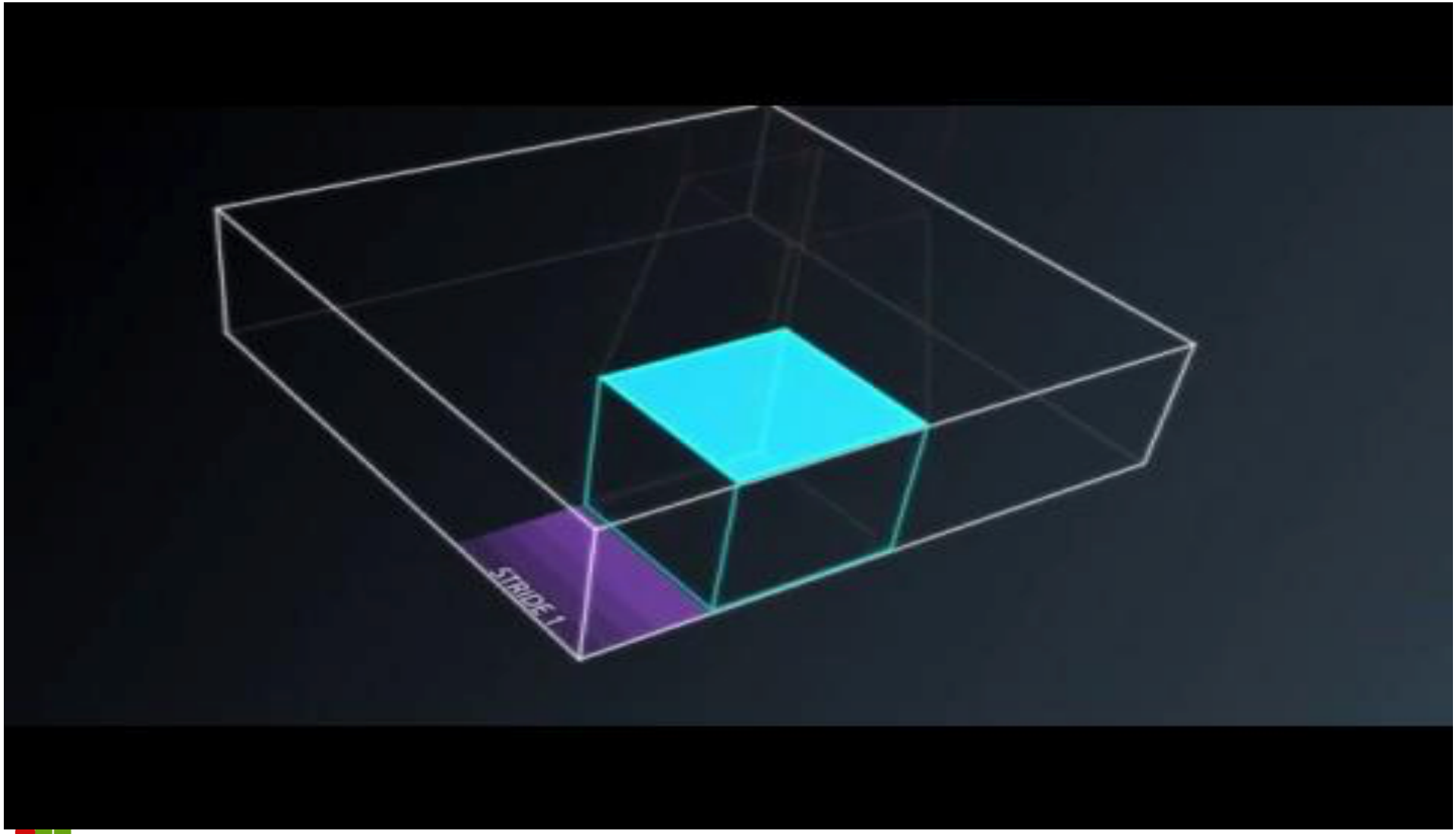


STRIDE 1



STRIDE 2

ConvNets By Google

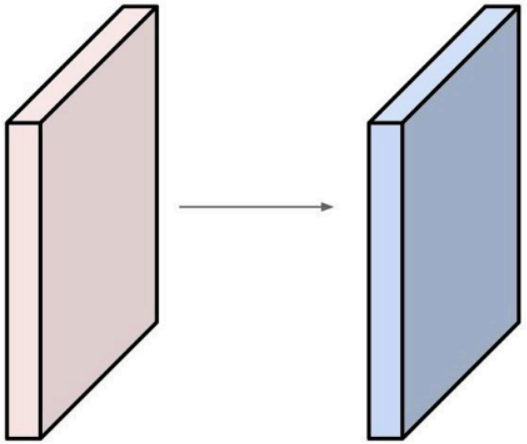


Example:

Examples time:

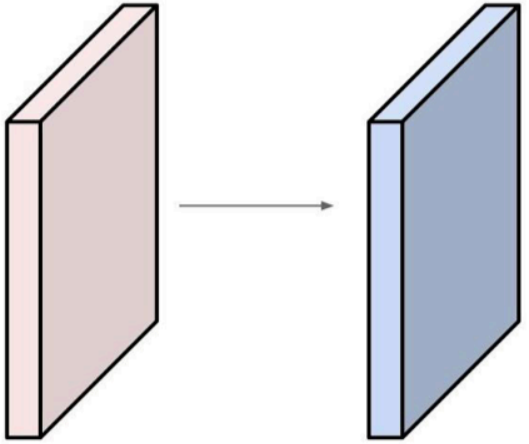
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Example:

Examples time:

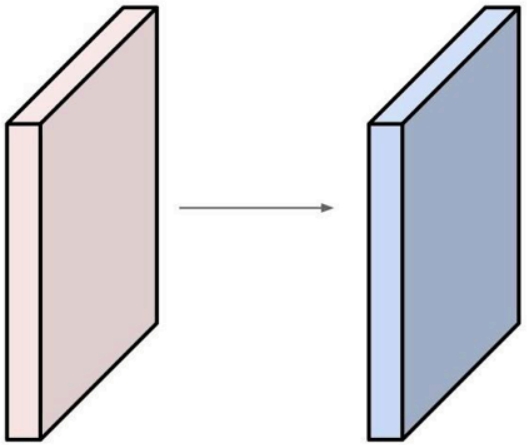


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

Example:

Examples time:

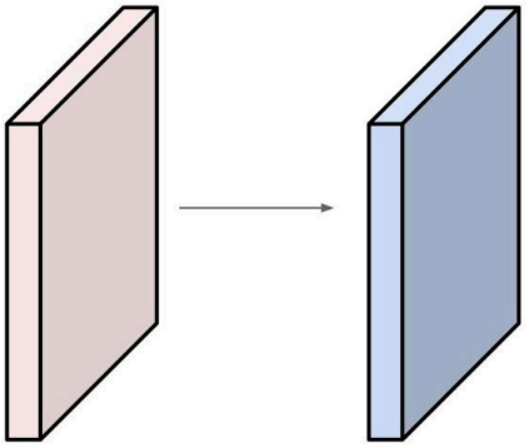


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Example:

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**

ConvNet Summary

Summary. To summarize, the Conv Layer:

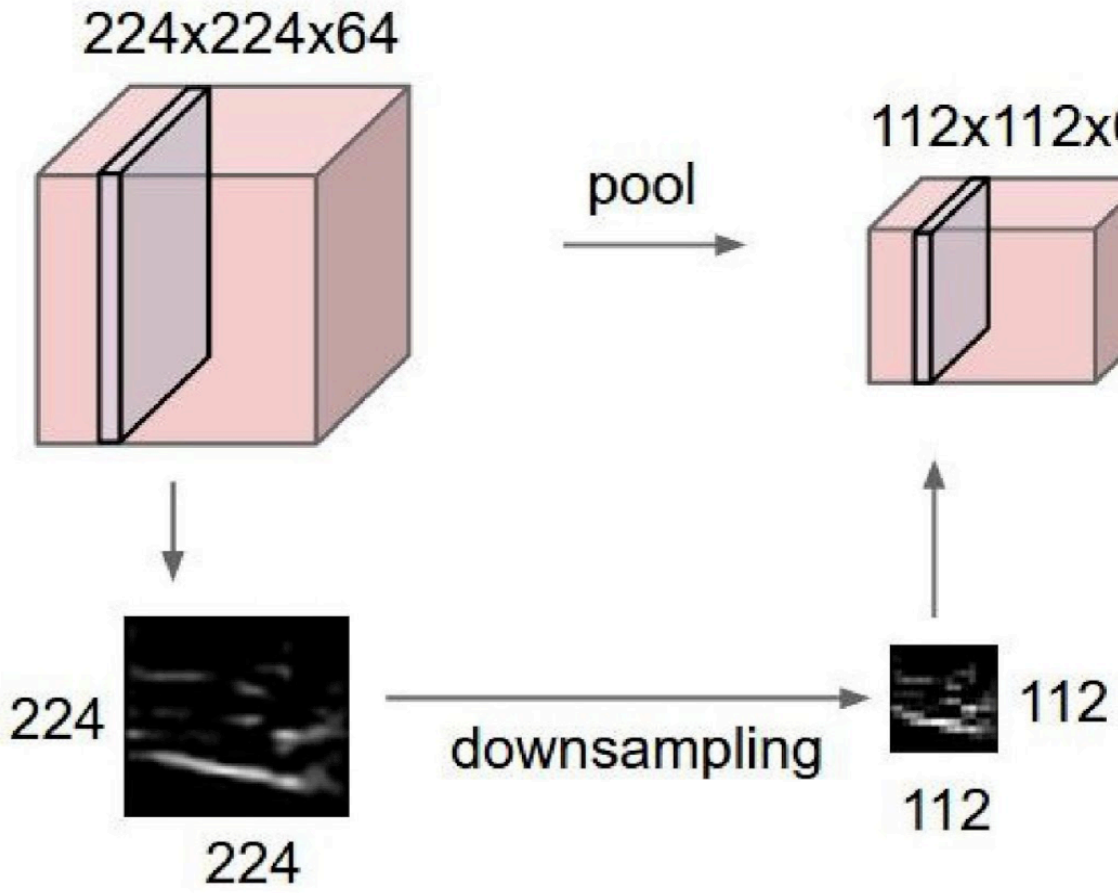
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (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.

Common settings:

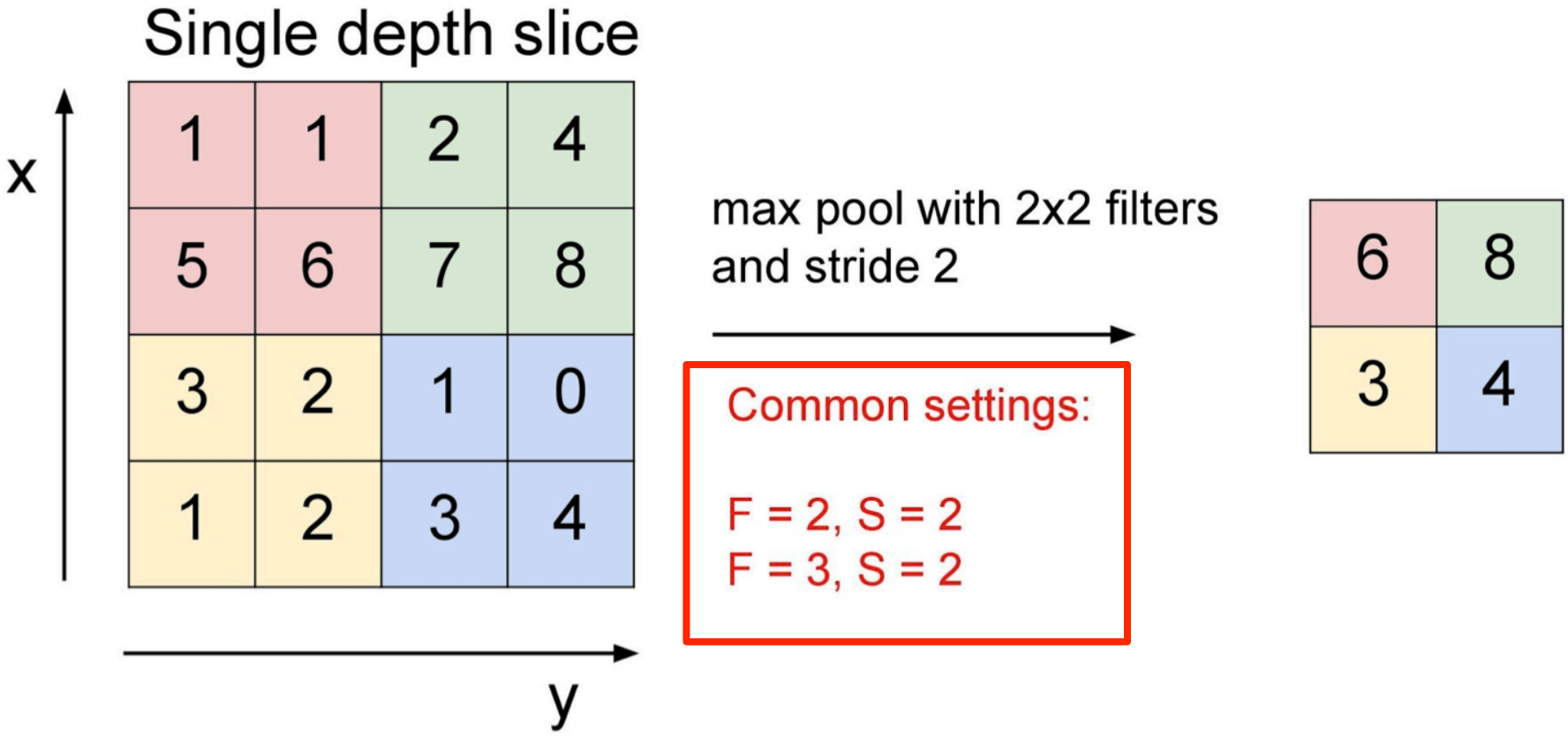
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Pooling layer

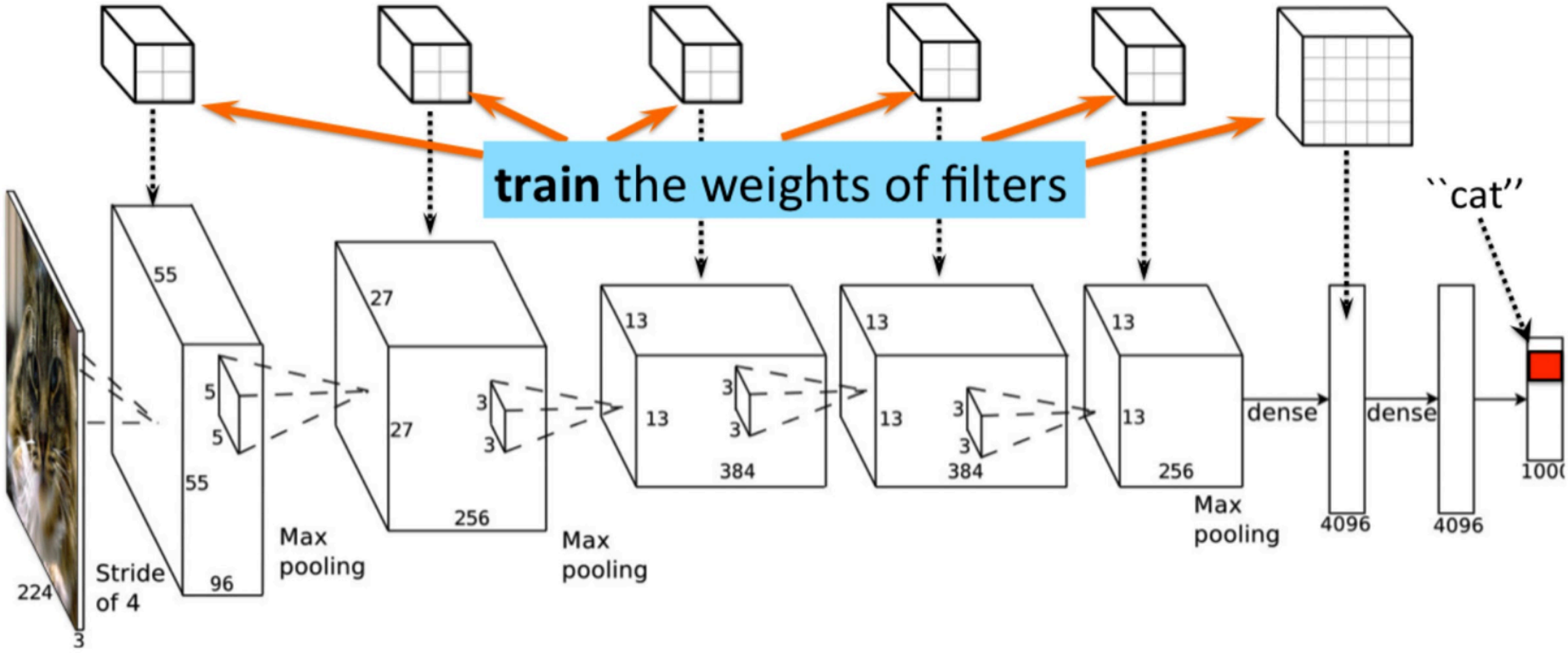
- makes the representations smaller and more manageable
- operates over each activation map independently:



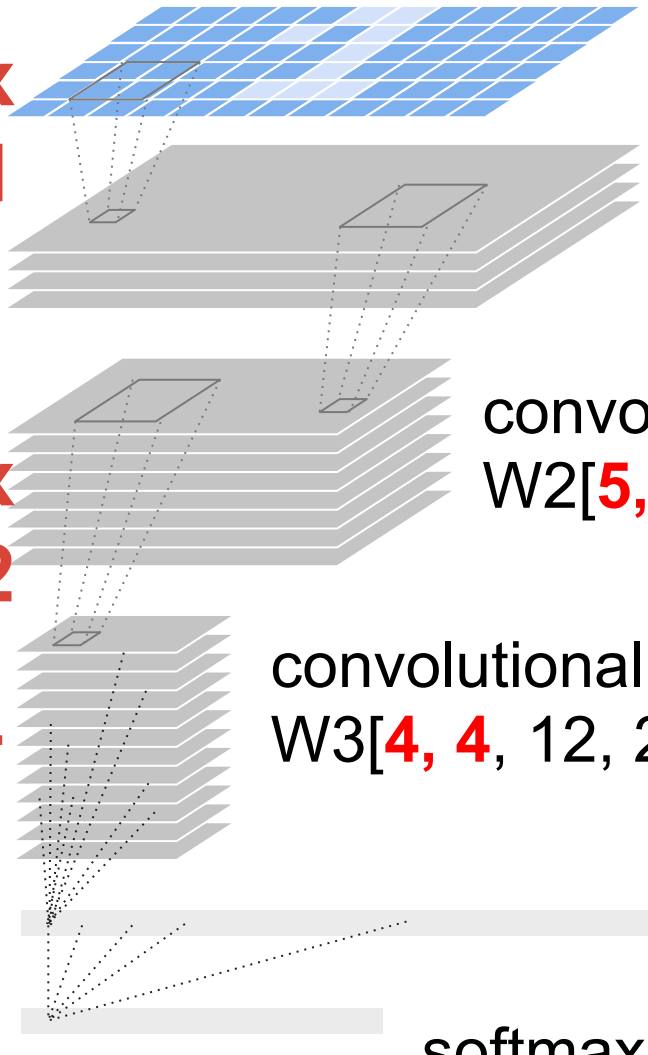
MAX Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4
Common settings:
F = 2, S = 2
F = 3, S = 2

Overview


train the weights of filters
224
3
55
55
96
5
5
Stride of 4
Max pooling
27
27
256
3
3
Max pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
3
3
Max pooling
dense
4096
dense
4096
``cat''

CNN in Action - MNIST Dataset



convolutional layer, **6 channels**
W1[**6, 6**, 1, 6] **stride 1**

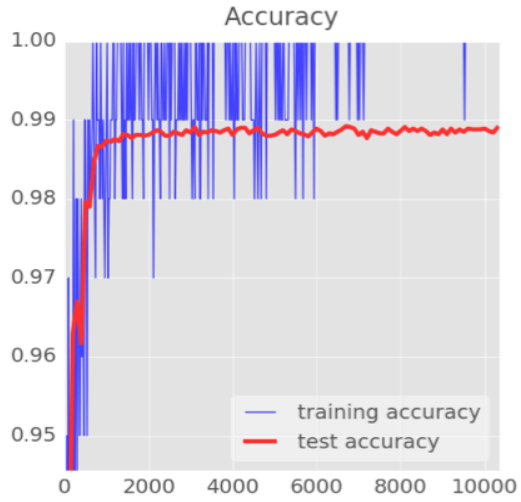
convolutional layer, **12 channels**
W2[**5, 5**, 6, 12] **stride 2**

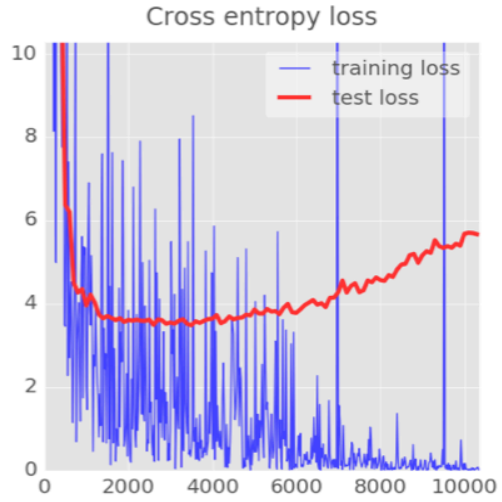
convolutional layer, **24 channels**
W3[**4, 4**, 12, 24] **stride 2**

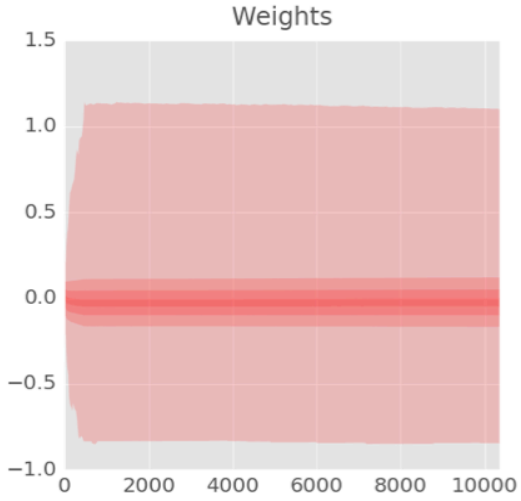
fully connected layer
W4[7x7x24, **200**]

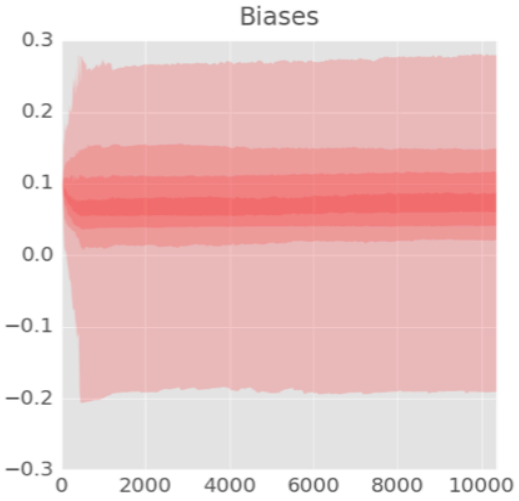
softmax readout layer W5[200, **10**]

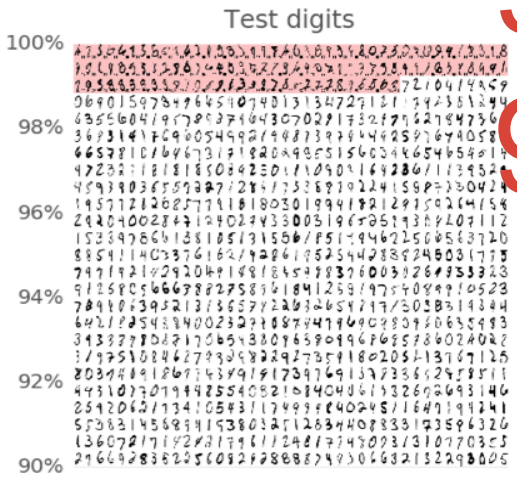
CNN in Action - MNIST Dataset







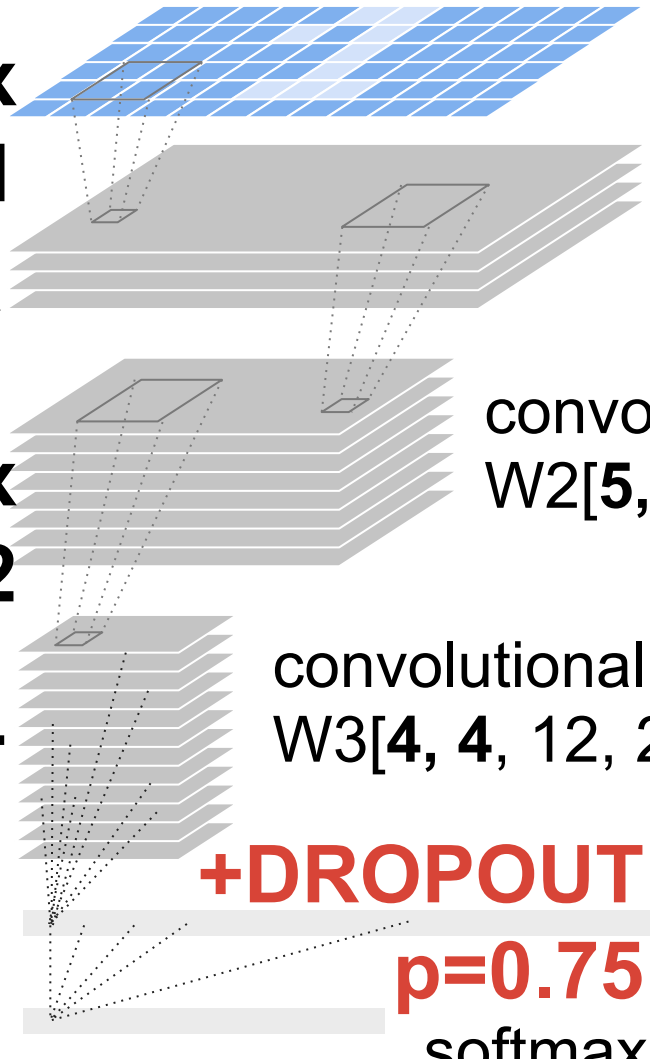




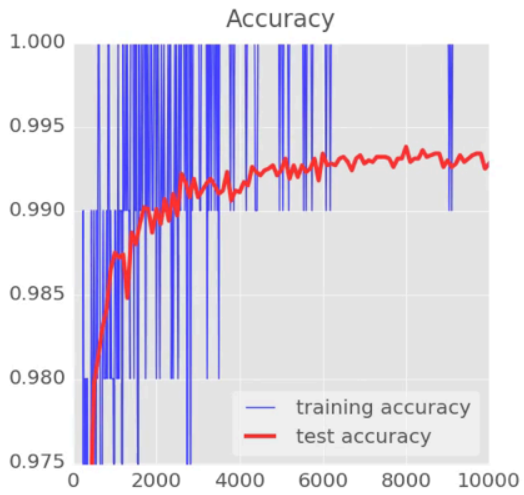
98.9%

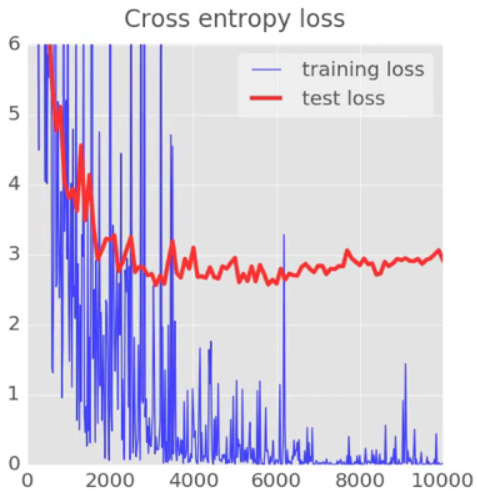
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

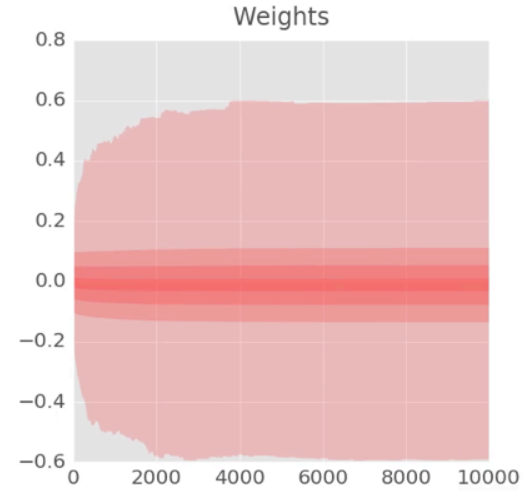
CNN in Action - MNIST Dataset

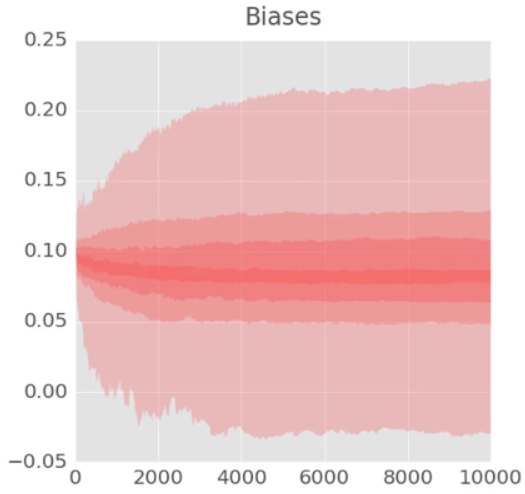


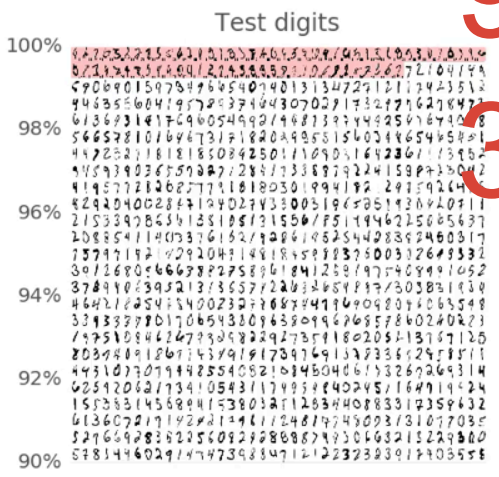
CNN in Action - MNIST Dataset







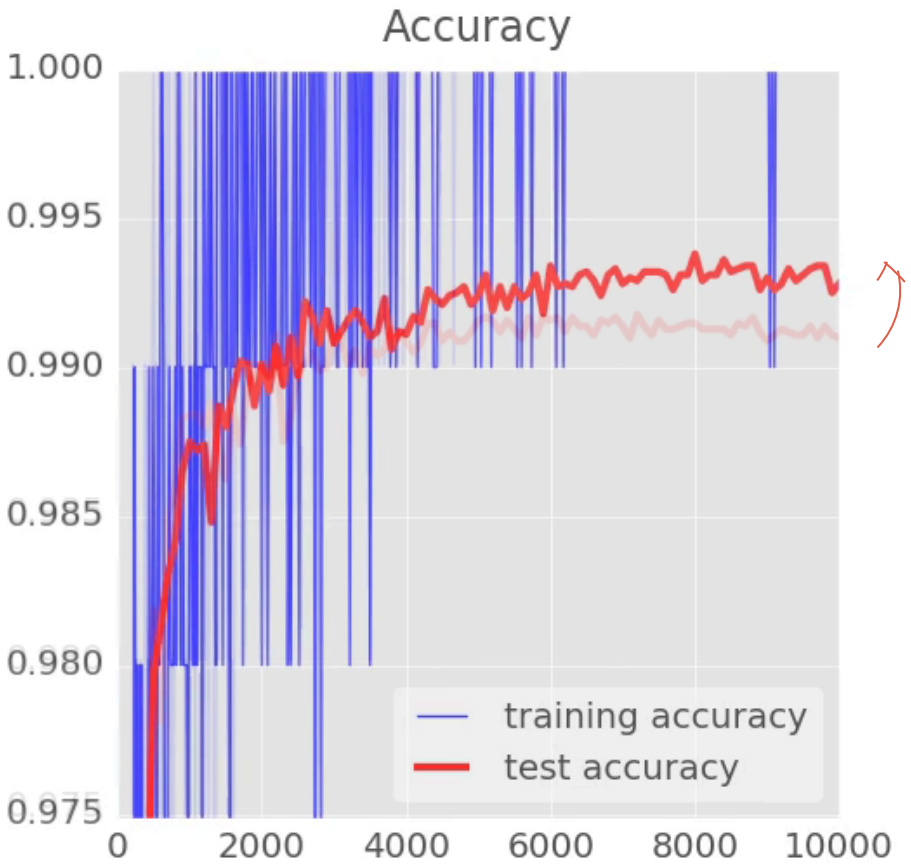


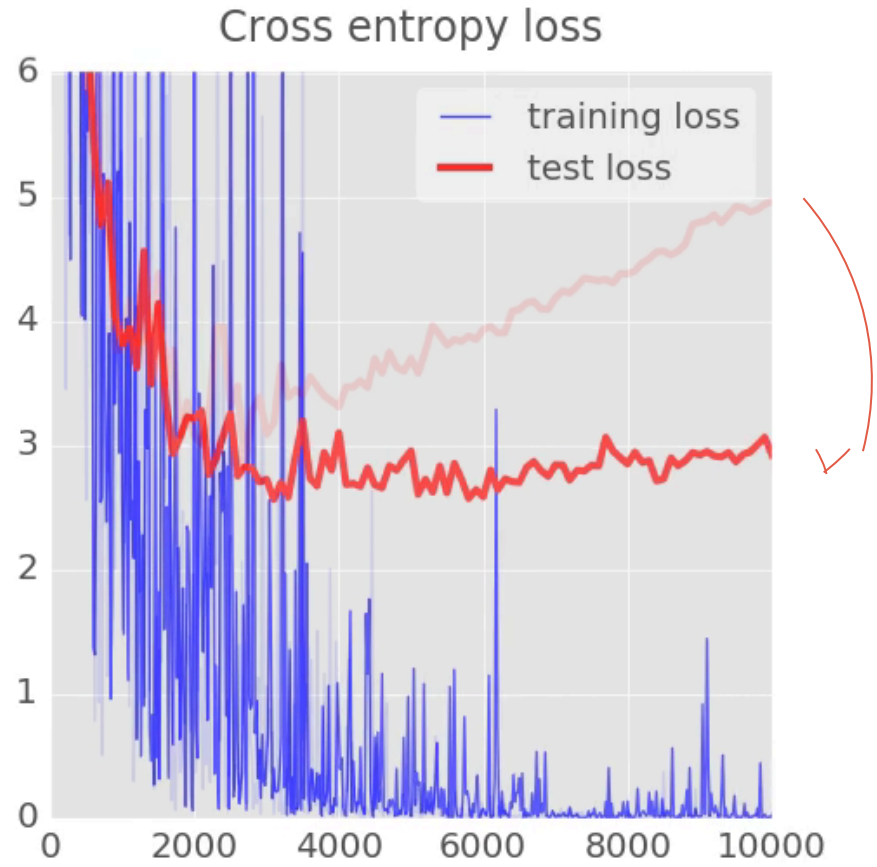


99.3%

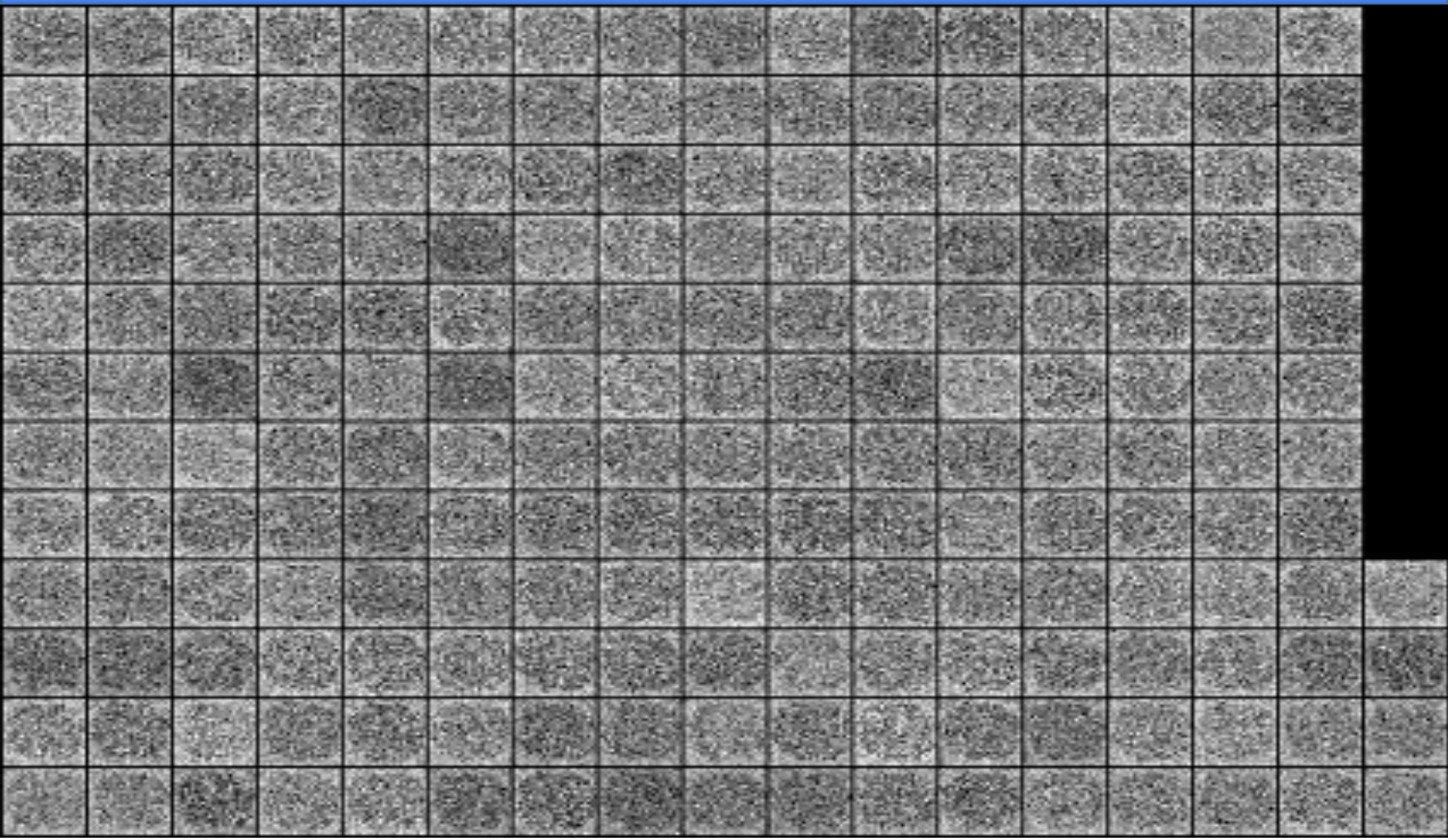
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

CNN in Action - MNIST Dataset





Weight learning



Modern Architectures

Le Net 5

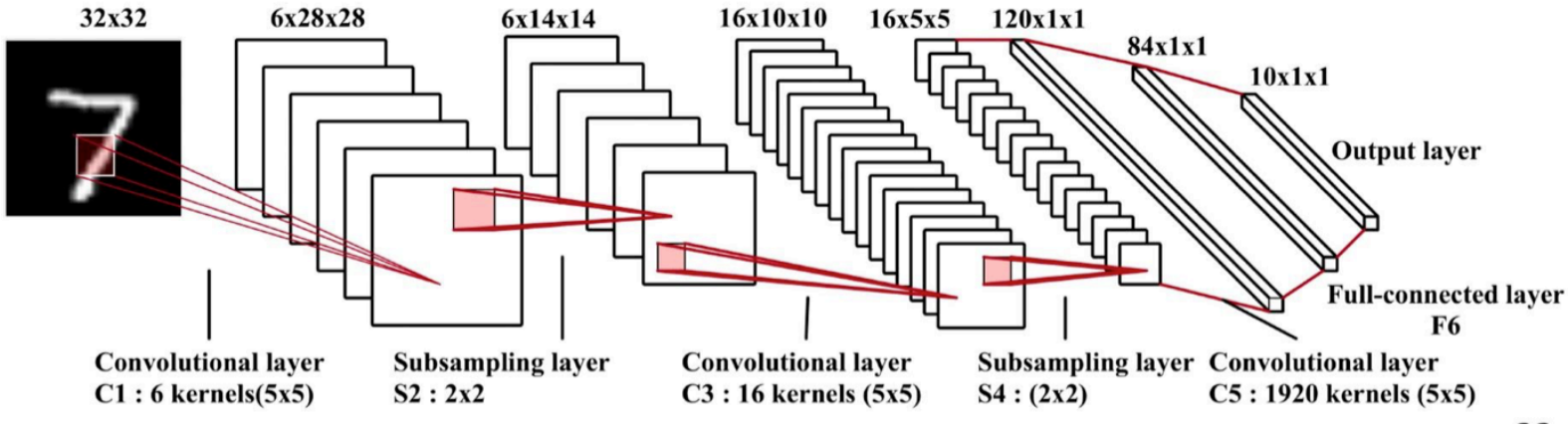
Main ideas:

- Local → global processing
- Retain coarse posn info

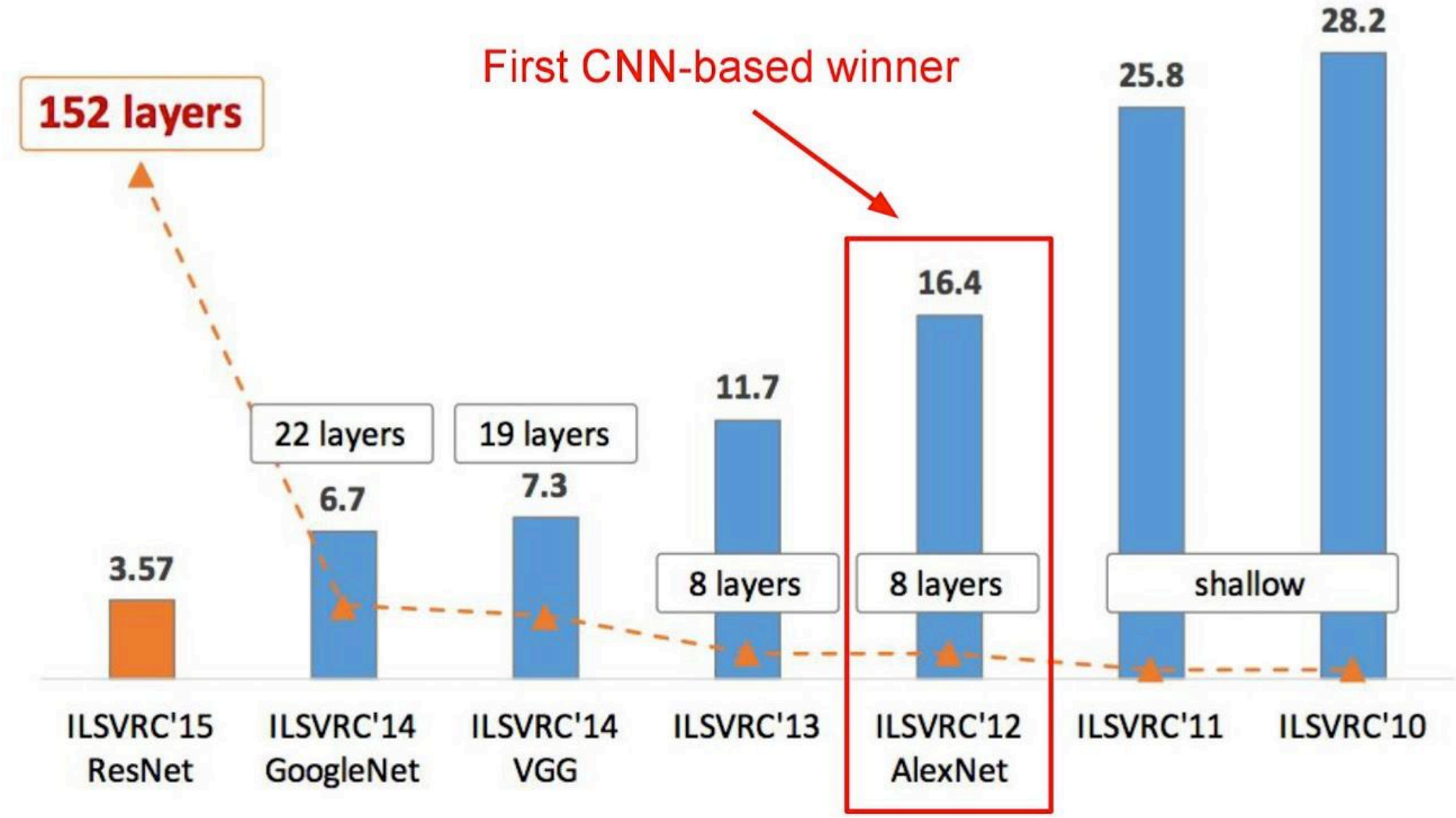
Main technique: weight sharing – units arranged in feature maps

Connections: 1256 units, 64,660 cxns, 9760 free parameters

Results: 0.14% (train), 5.0% (test)



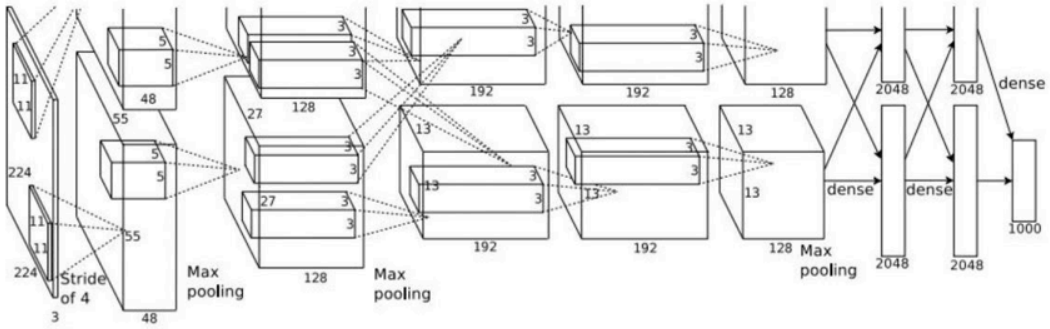
ImageNet (ILSVRC) winners



AlexNet

Case Study: AlexNet

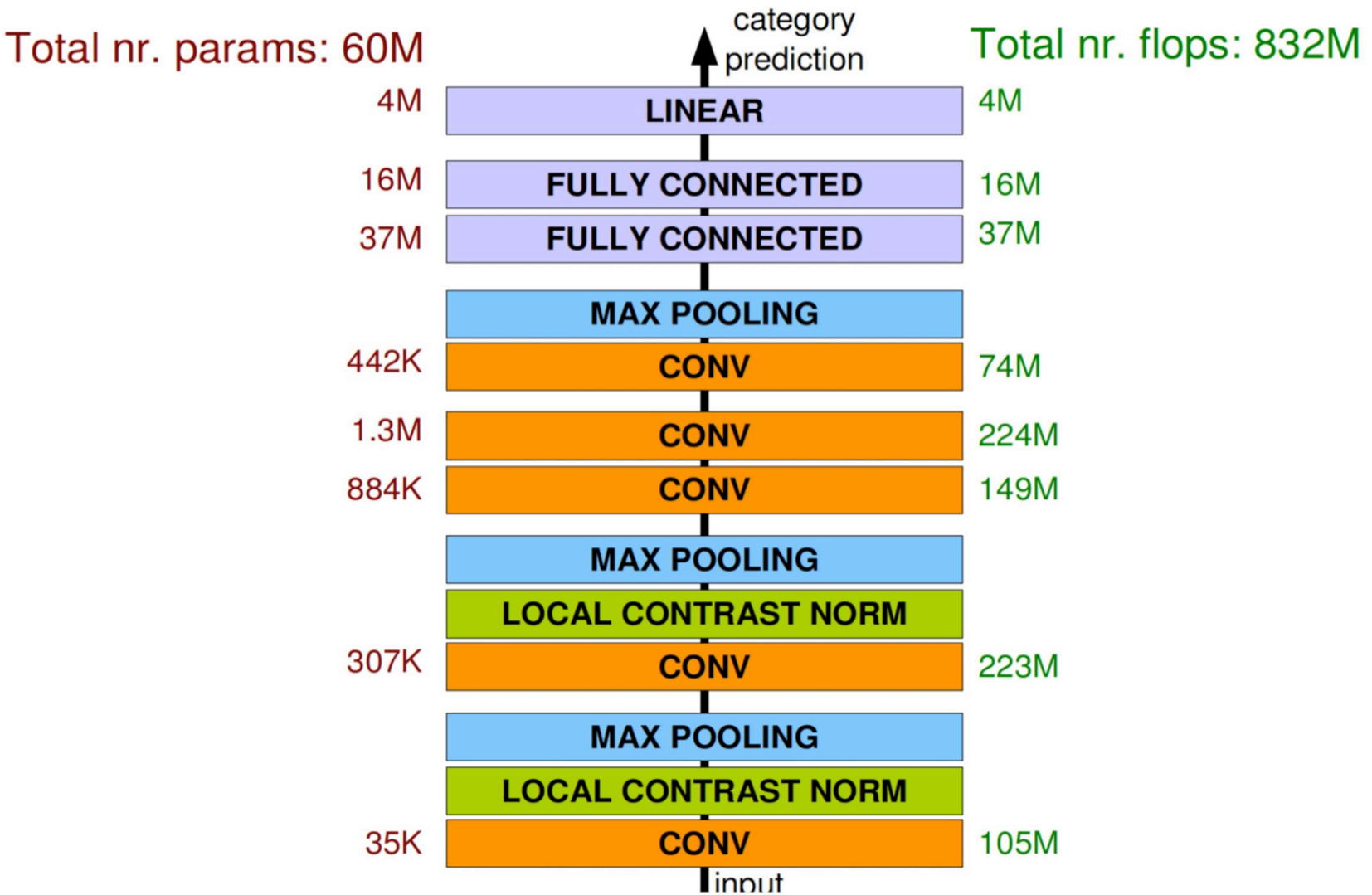
[Krizhevsky et al. 2012]



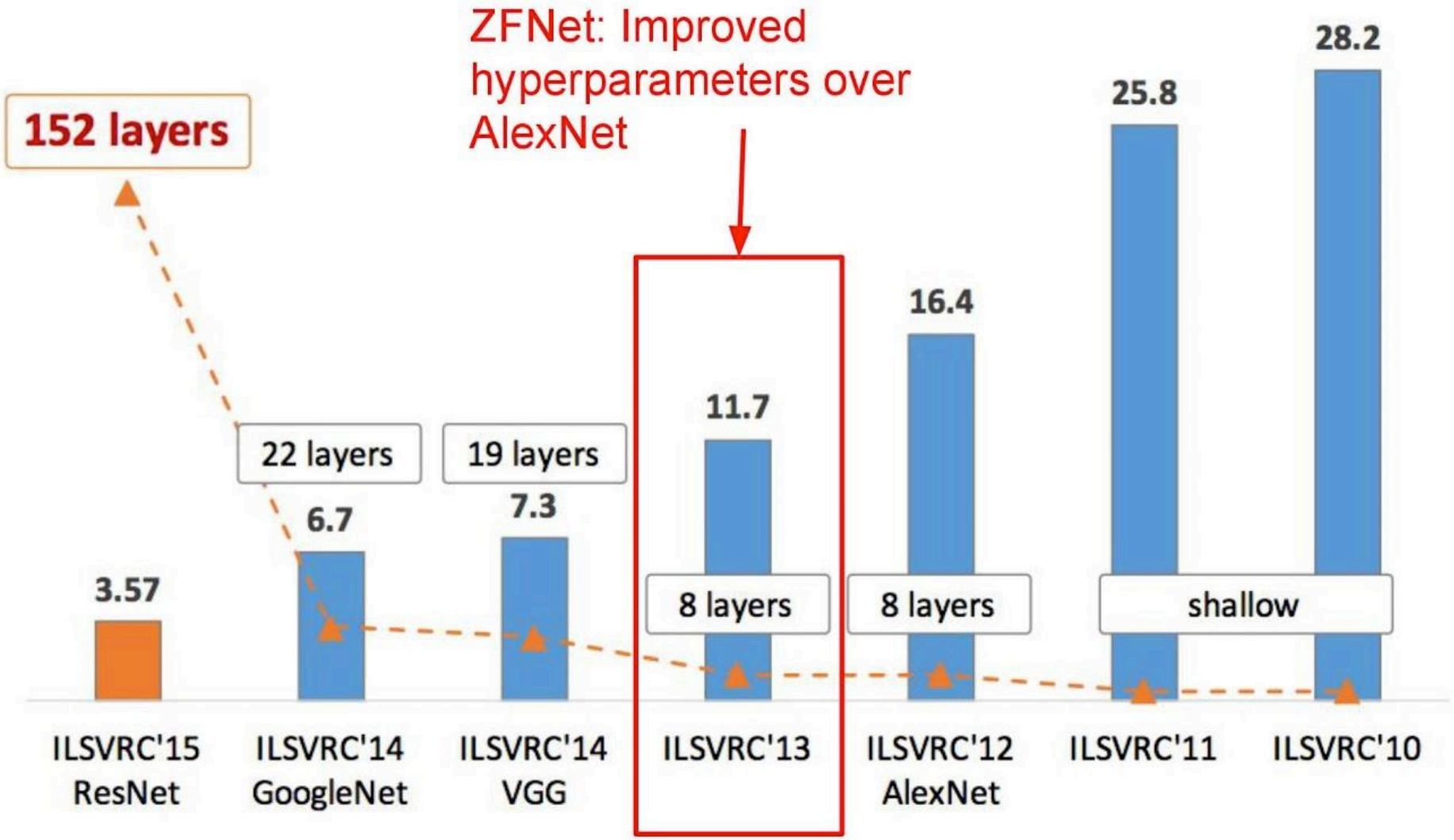
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

AlexNet


Total nr. params: 60M
Total nr. flops: 832M
category prediction
4M
LINEAR
4M
16M
FULLY CONNECTED
16M
37M
FULLY CONNECTED
37M
MAX POOLING
442K
CONV
74M
1.3M
CONV
224M
884K
CONV
149M
MAX POOLING
LOCAL CONTRAST NORM
307K
CONV
223M
MAX POOLING
LOCAL CONTRAST NORM
35K
CONV
105M
input

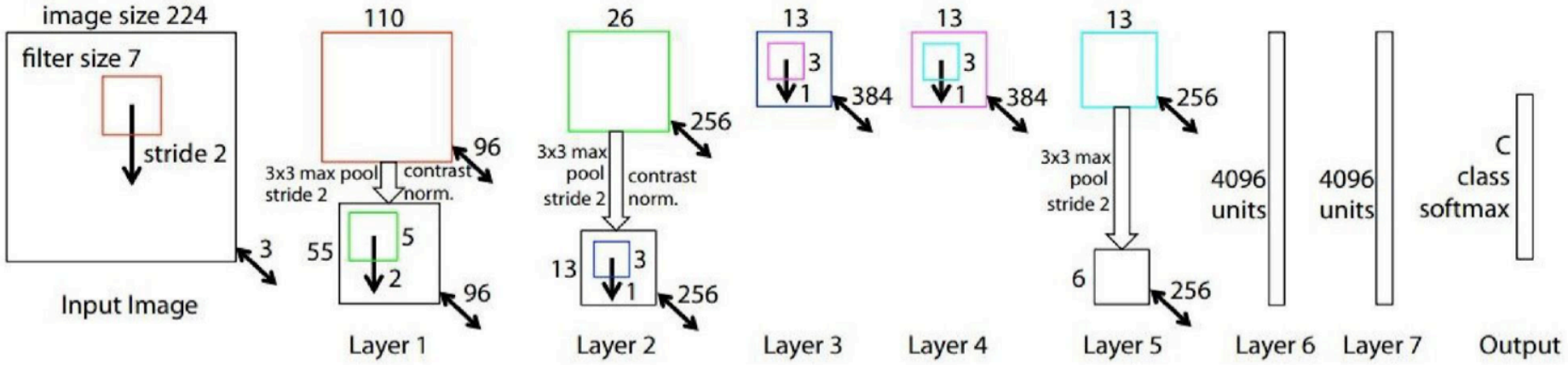
ImageNet (ILSVRC) winners



ZFNet

ZFNet

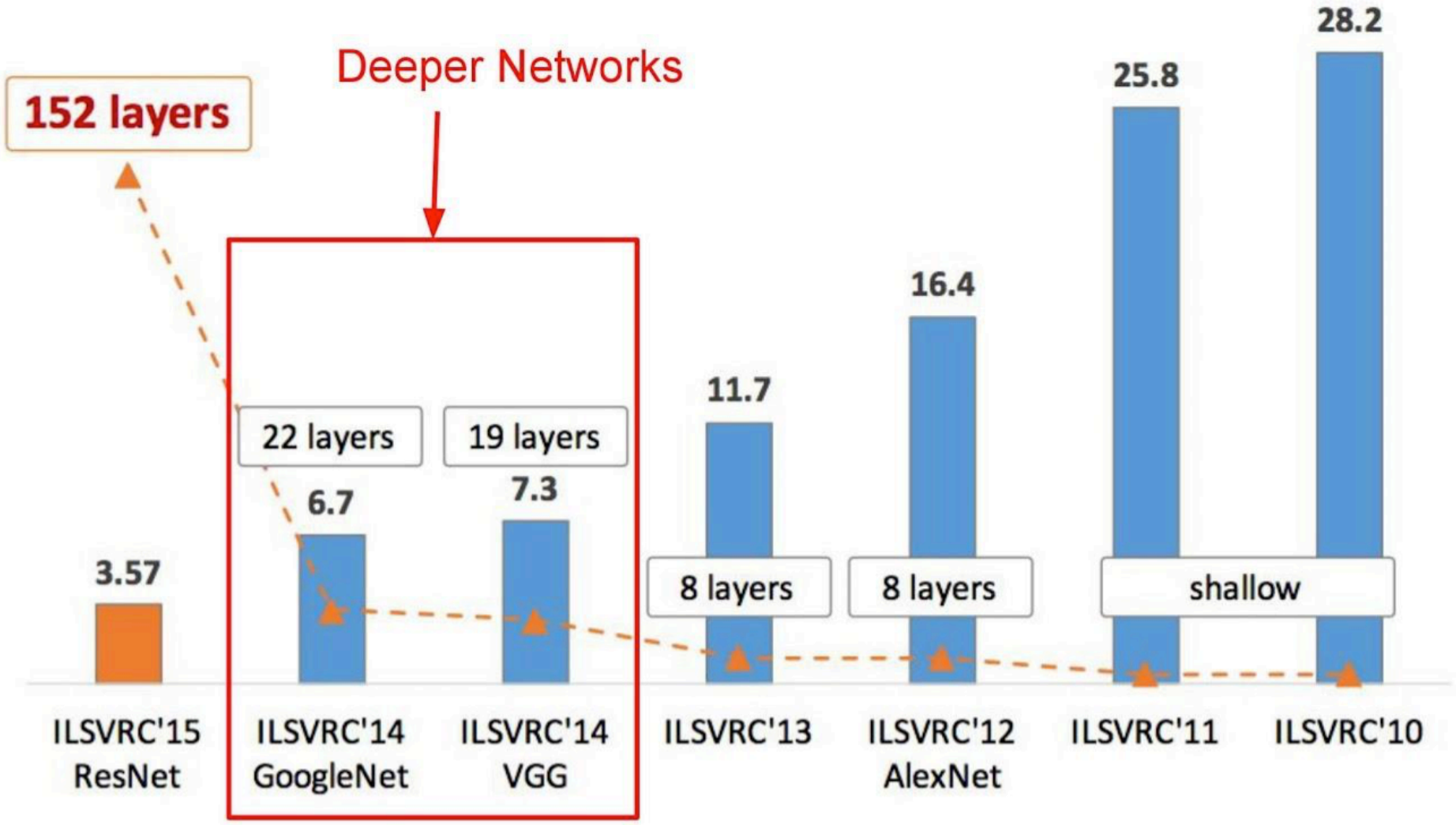
[Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

ImageNet (ILSVRC) winners



VGGNet

Case Study: VGGNet

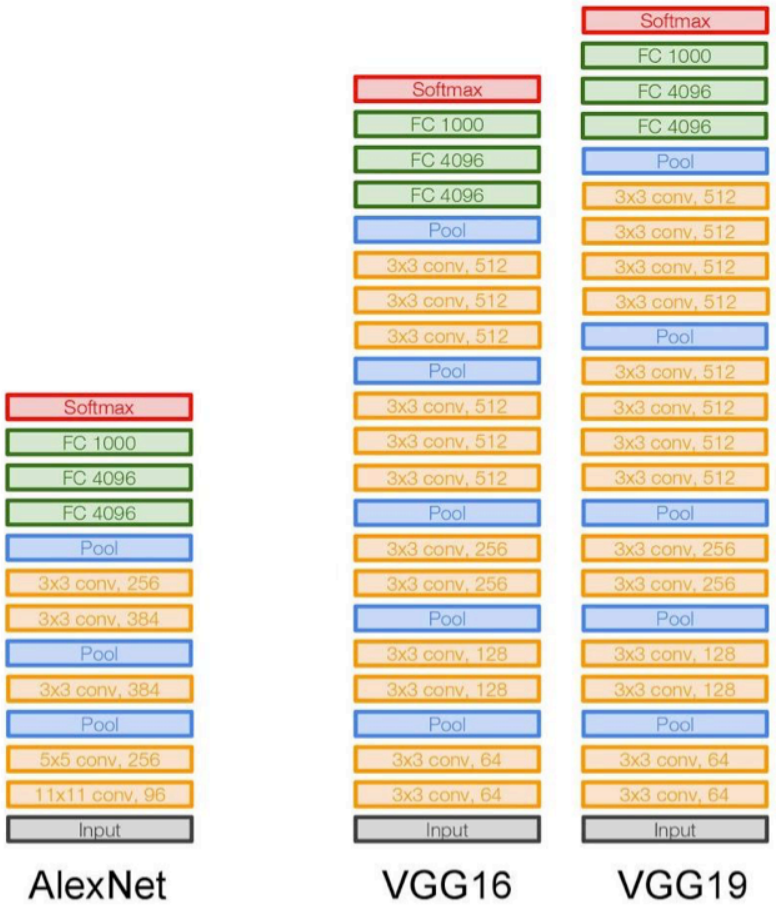
[Simonyan and Zisserman, 2014]

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

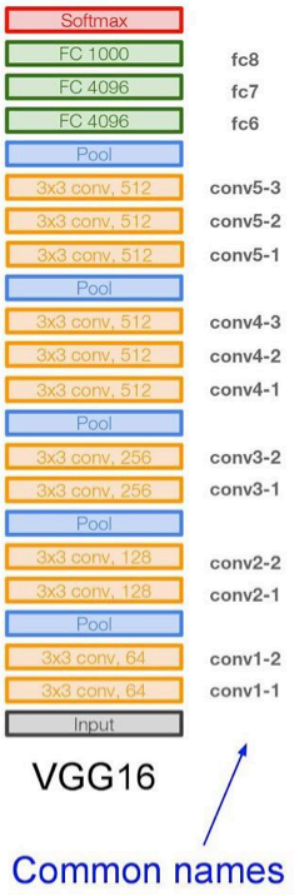


VGGNet

(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 ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



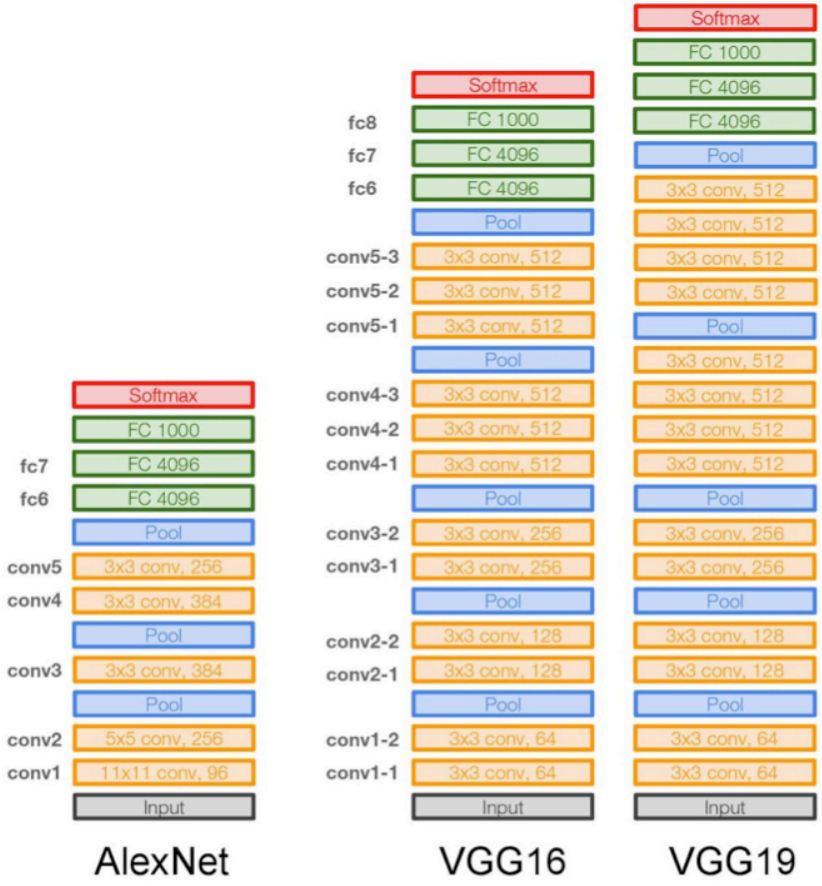
VGGNet

Case Study: VGGNet

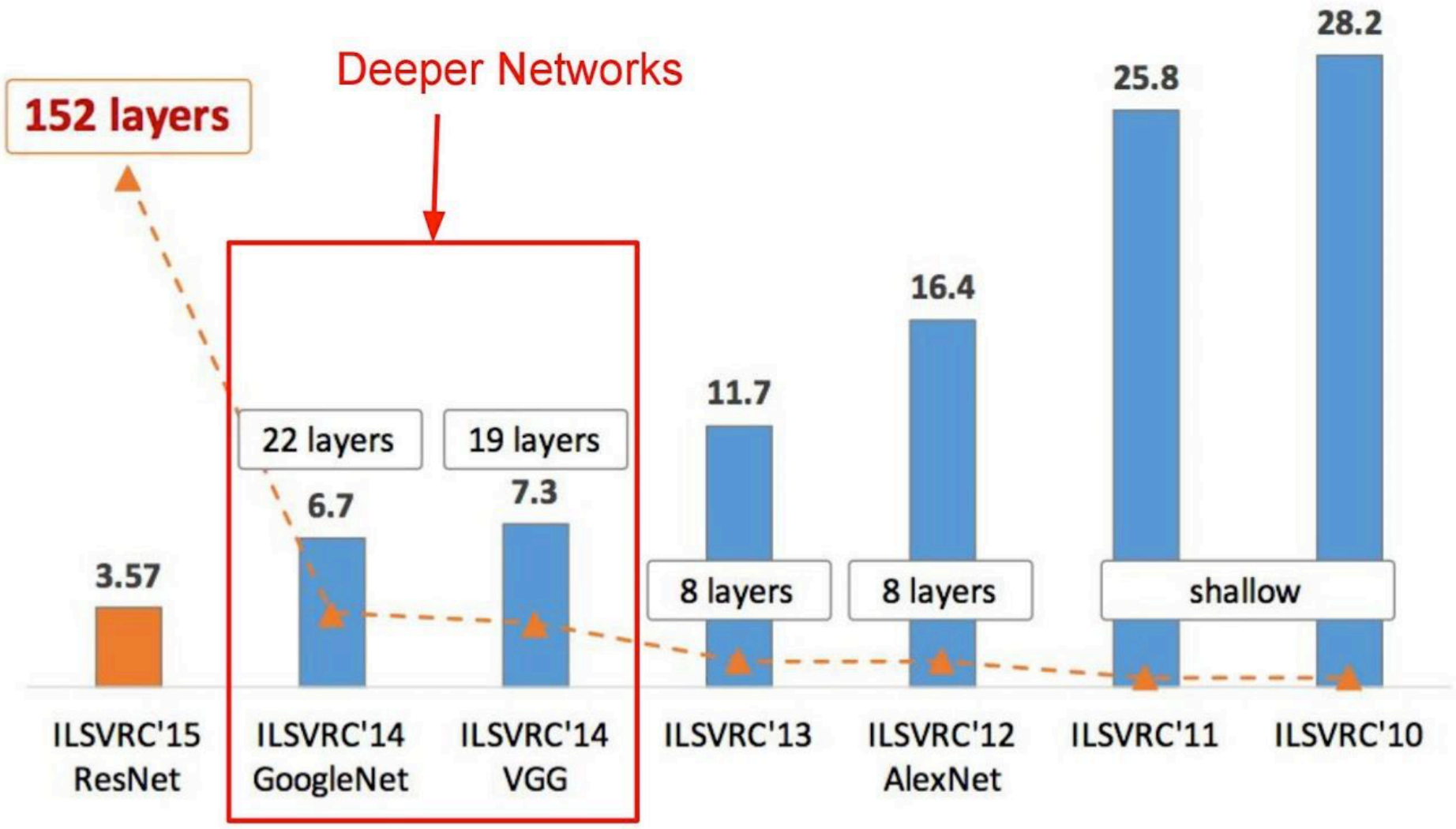
[Simonyan and Zisserman, 2014]

Details:

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



ImageNet (ILSVRC) winners



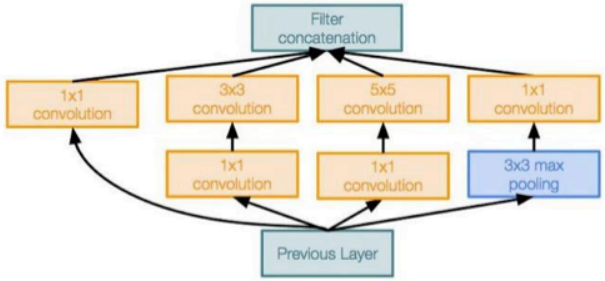
GoogLeNet

Case Study: GoogLeNet

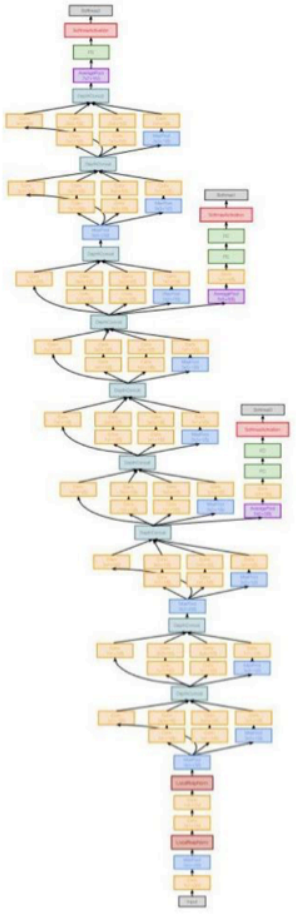
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

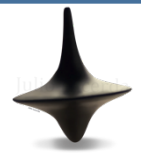
- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)



Inception module

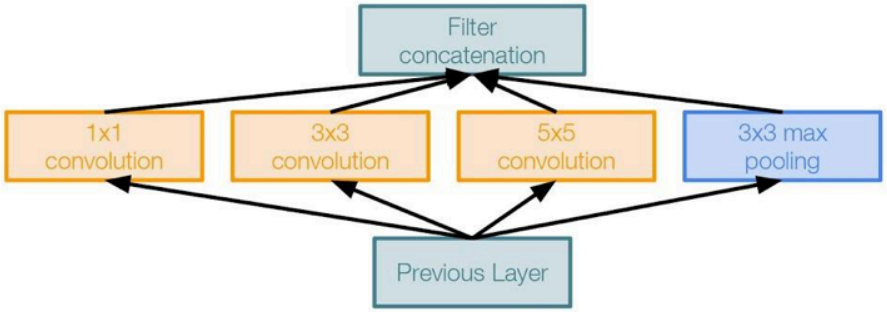


Inception



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

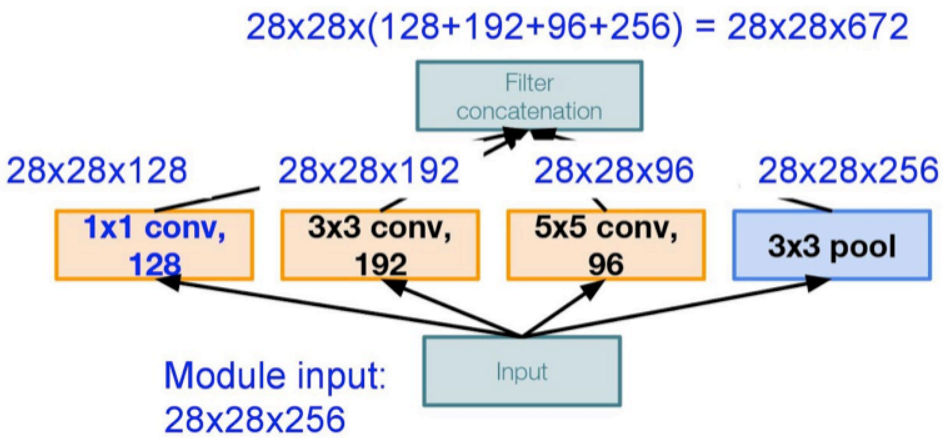
Inception

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

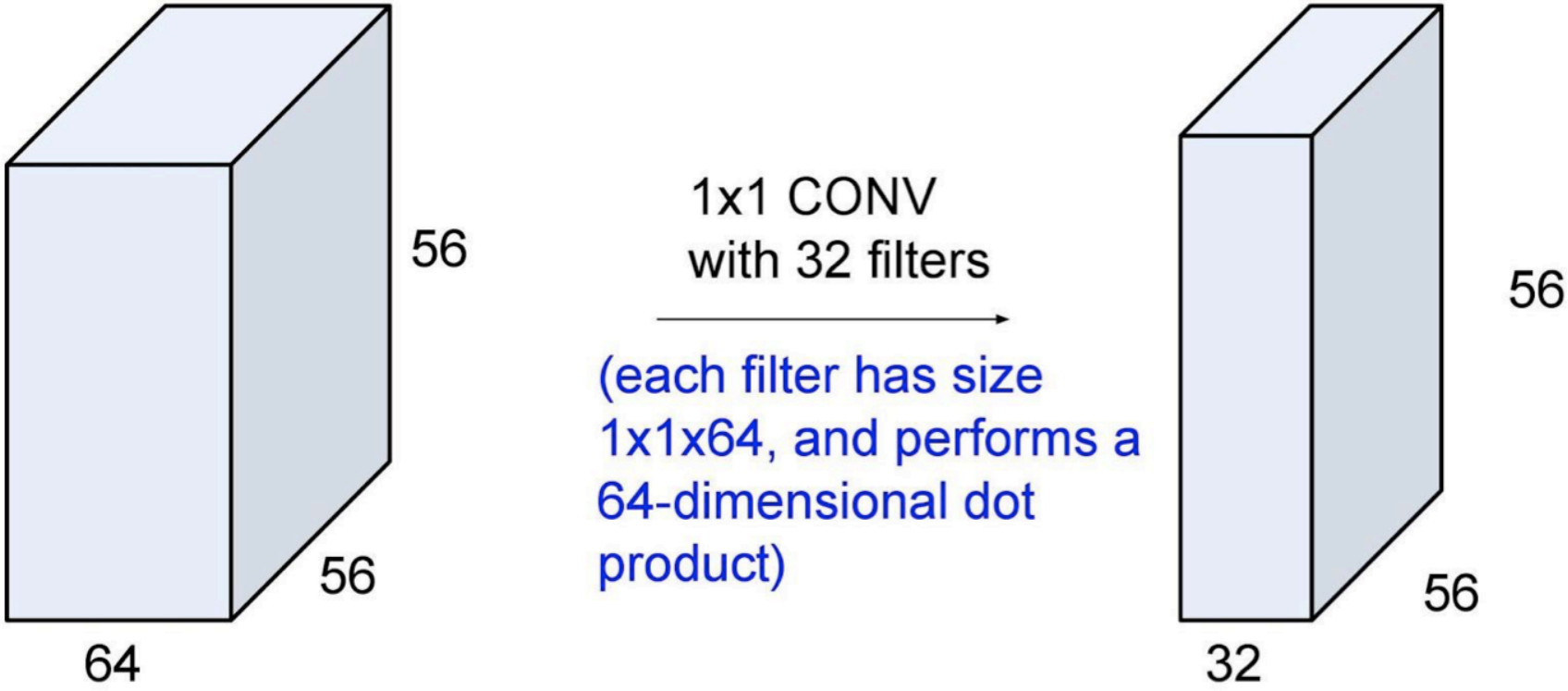
Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

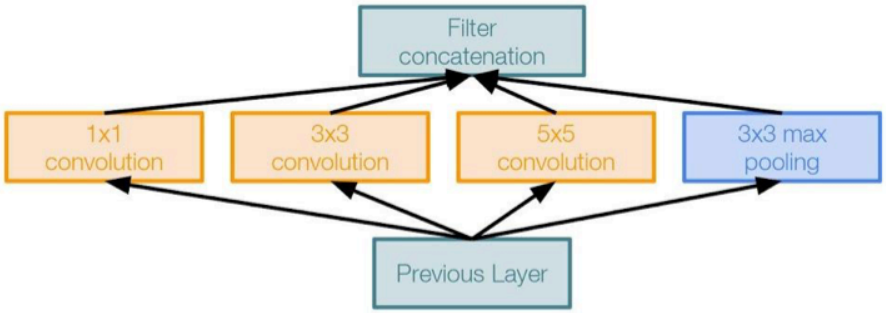
1x1 convolutions



Efficient Inection

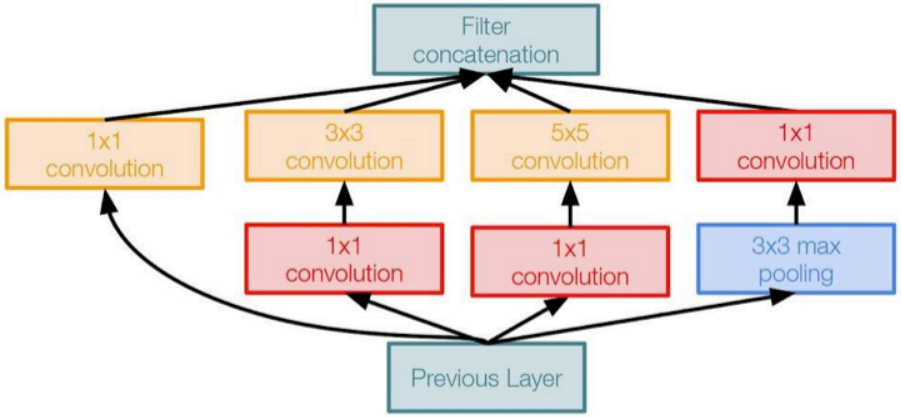
Case Study: GoogLeNet

[Szegedy et al., 2014]



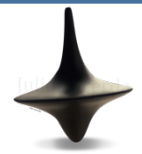
Naive Inception module

1x1 conv "bottleneck" layers



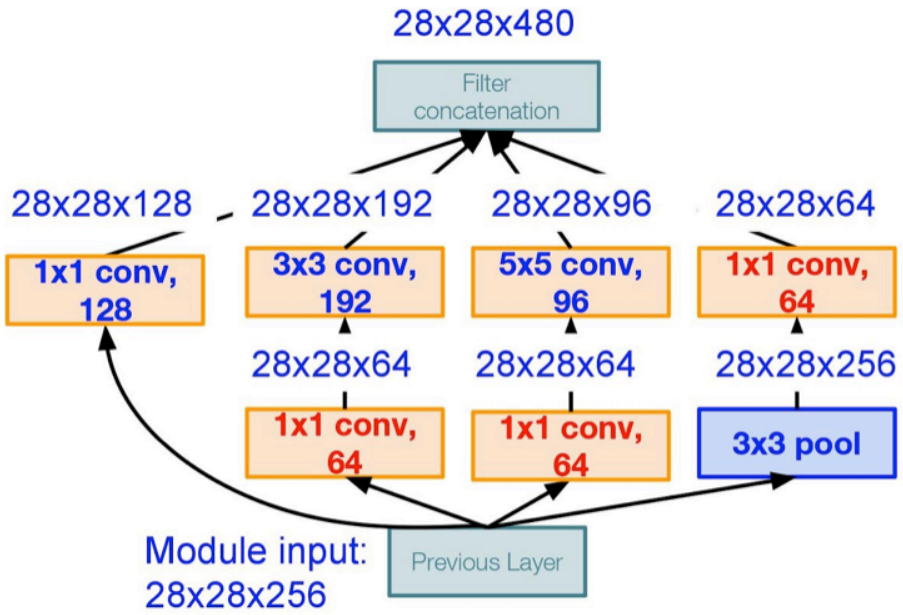
Inception module with dimension reduction

Inception



Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

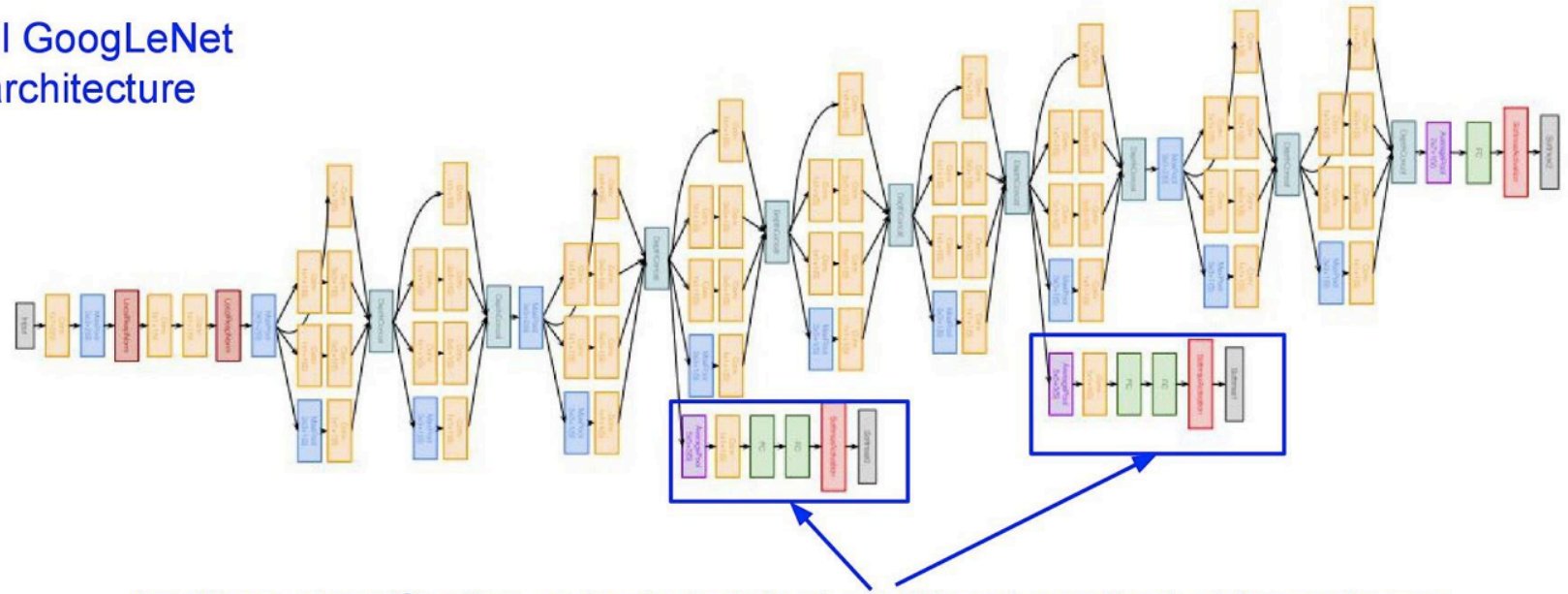
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

Challenge of Deep Networks

Case Study: GoogLeNet

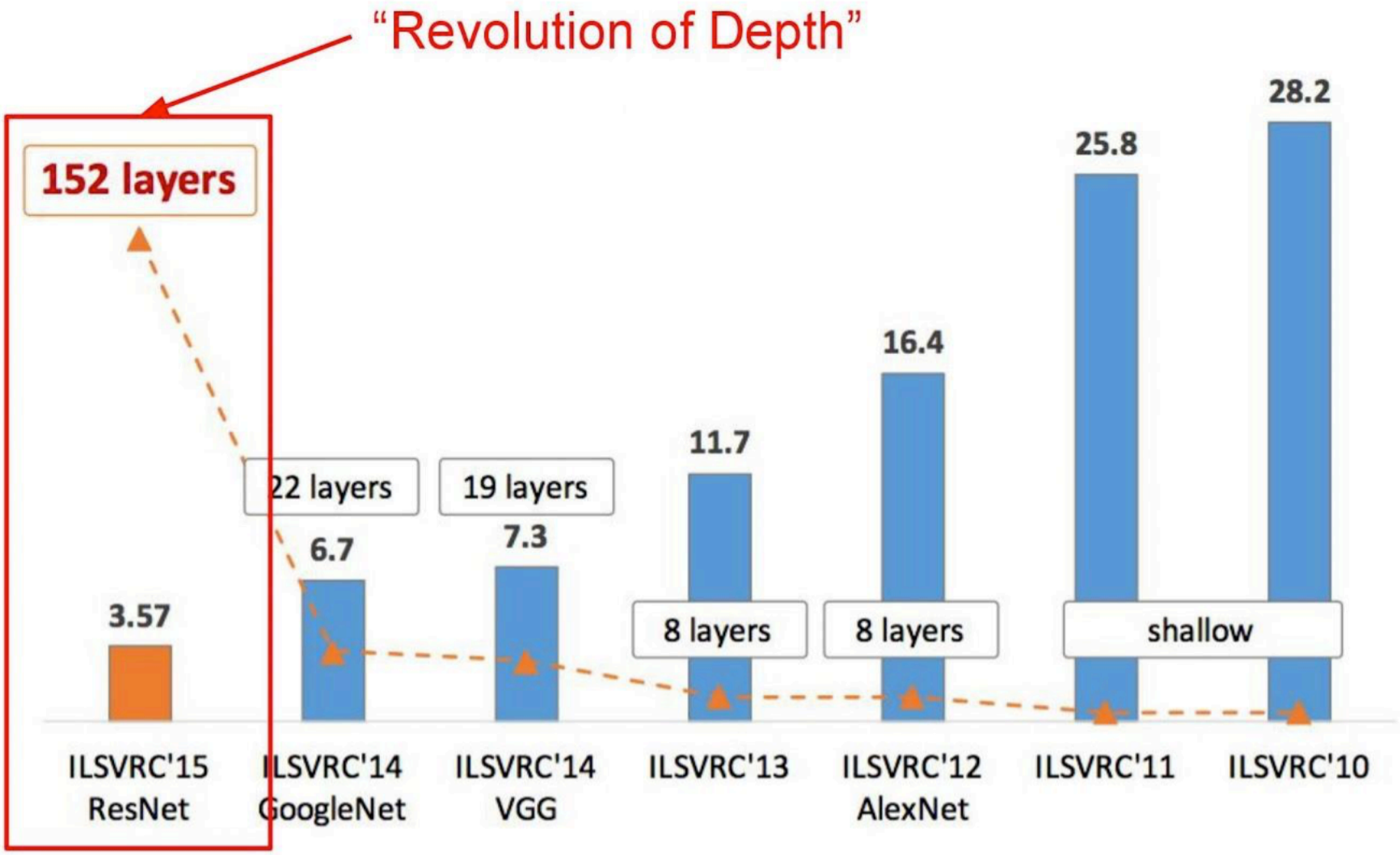
[Szegedy et al., 2014]

Full GoogLeNet architecture



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

ImageNet (ILSVRC) winners



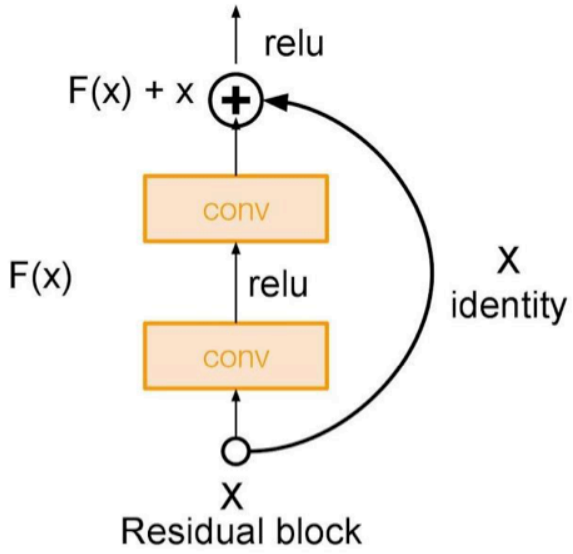
Residual Networks

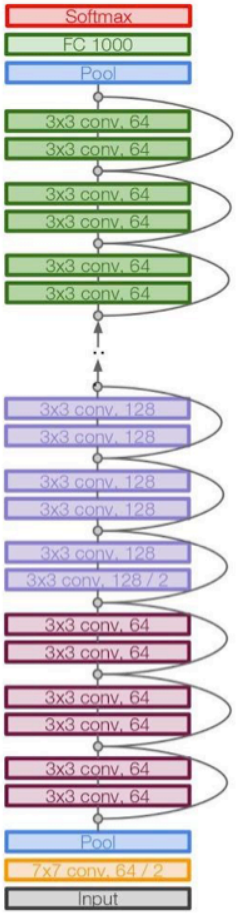
Case Study: ResNet

[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!



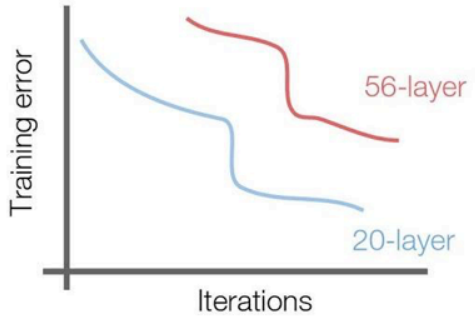


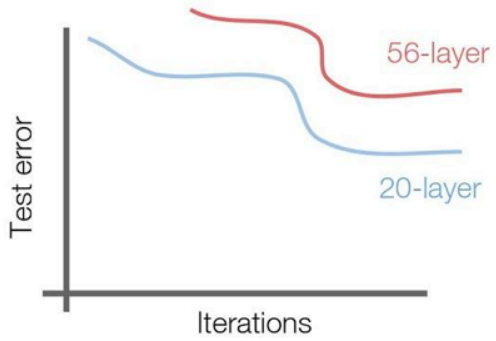
Residual Networks

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!

Residual Networks

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.

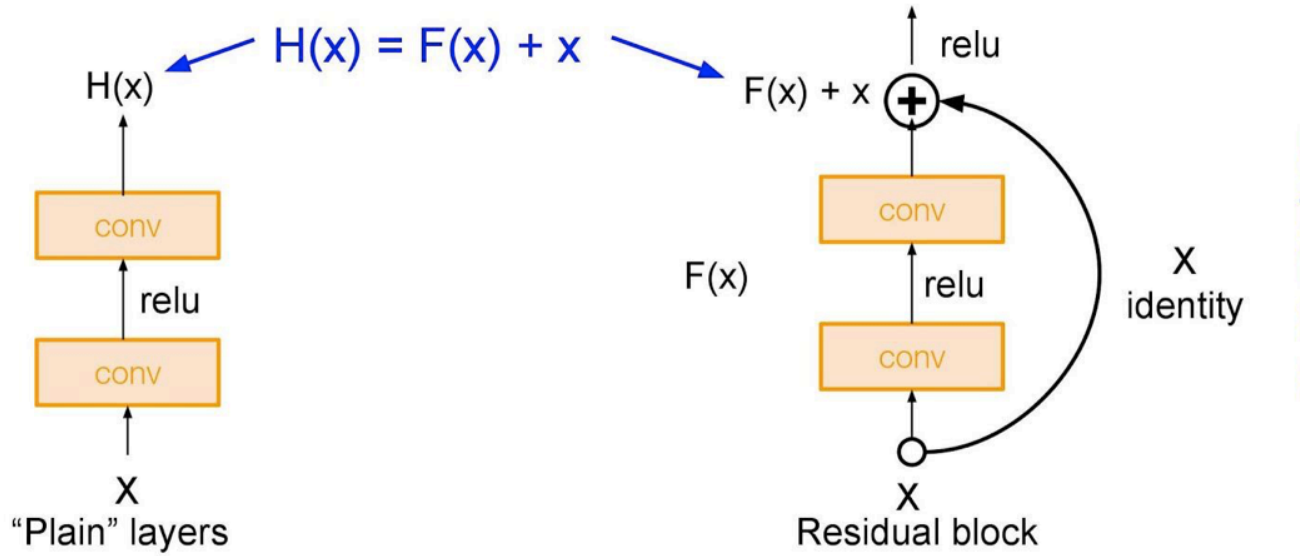
Residual Networks

Case Study: ResNet

[He et al., 2015]

Inspired by LSTM gating units
- Combats vanishing gradients
- Simple form of auto-regression

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



Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

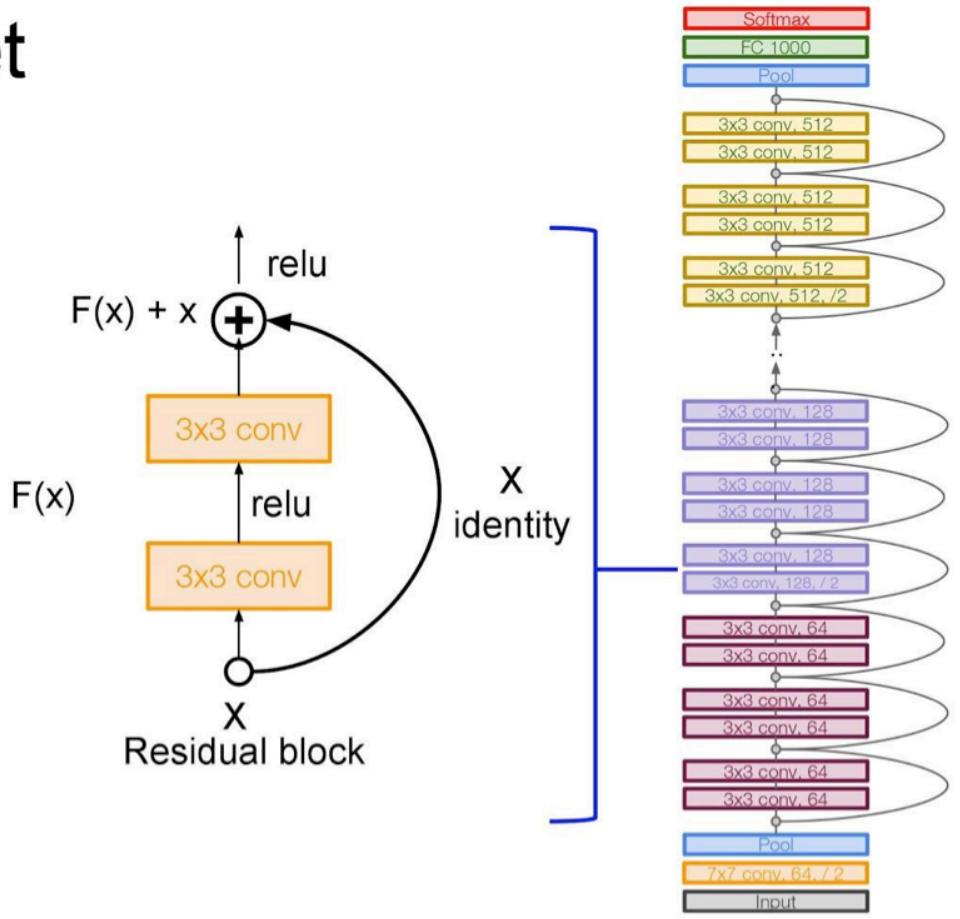
Residual Networks

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers



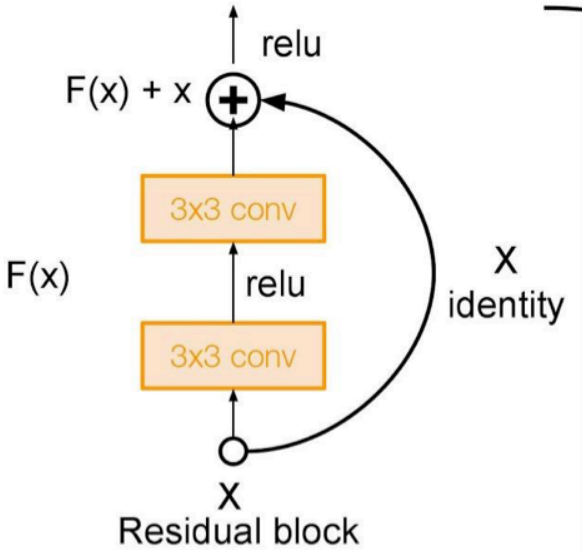
Residual Networks

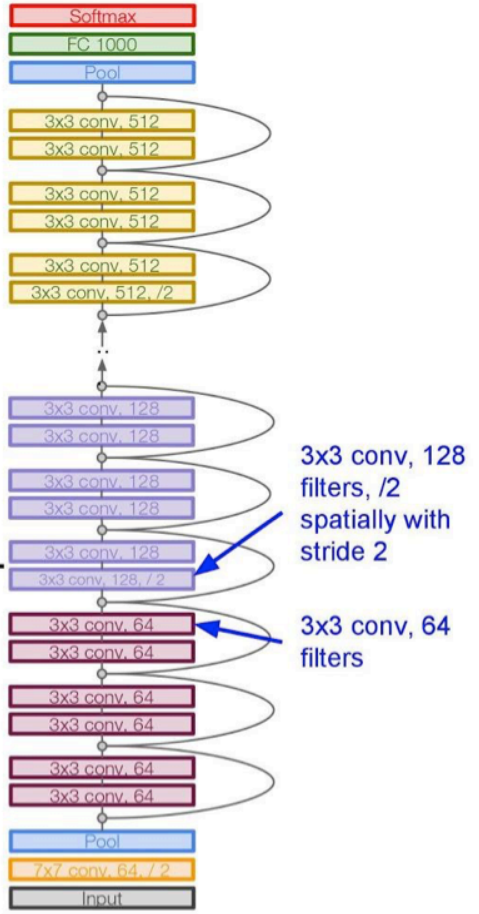
Case Study: ResNet

[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)





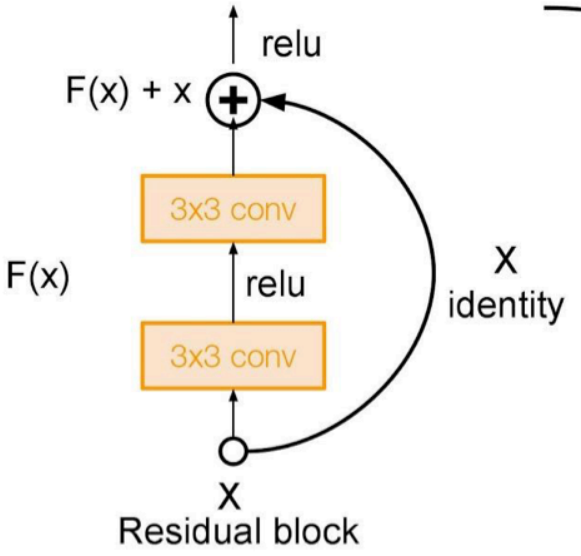
Residual Networks

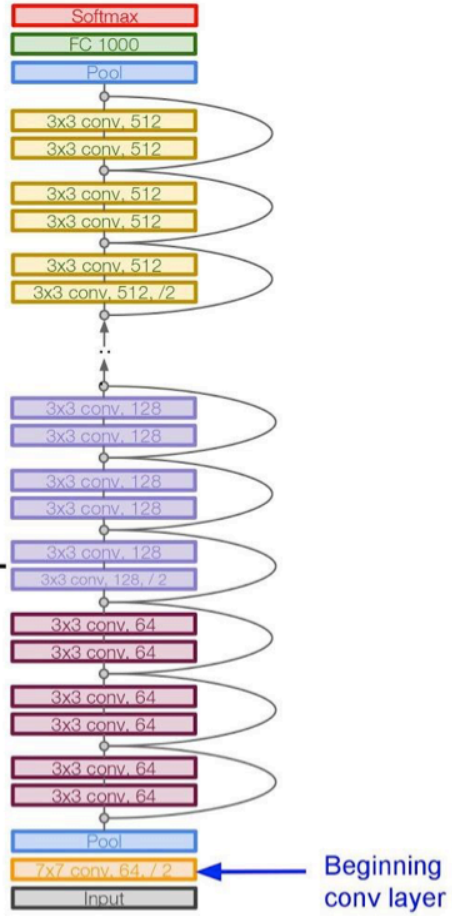
Case Study: ResNet

[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





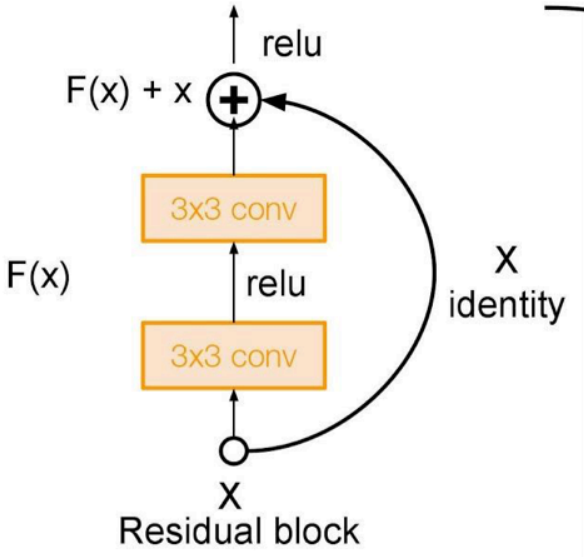
Residual Networks

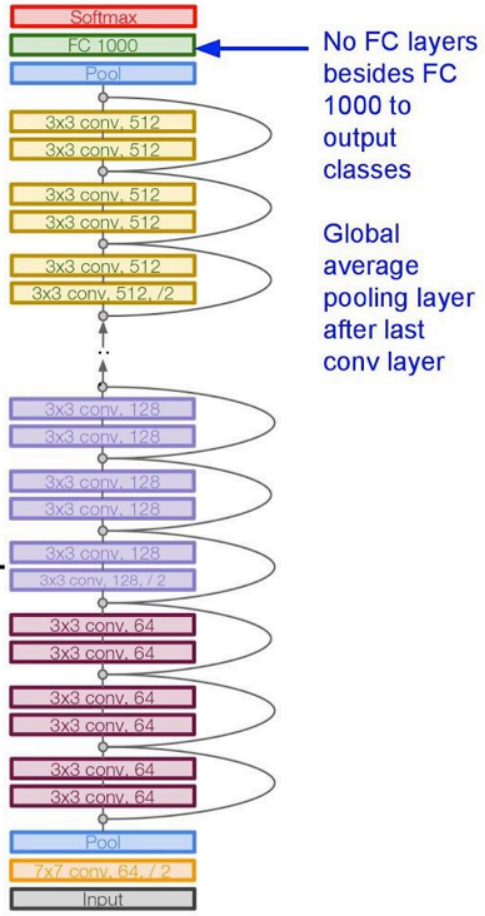
Case Study: ResNet

[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
- No FC layers at the end (only FC 1000 to output classes)





Residual Networks

Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

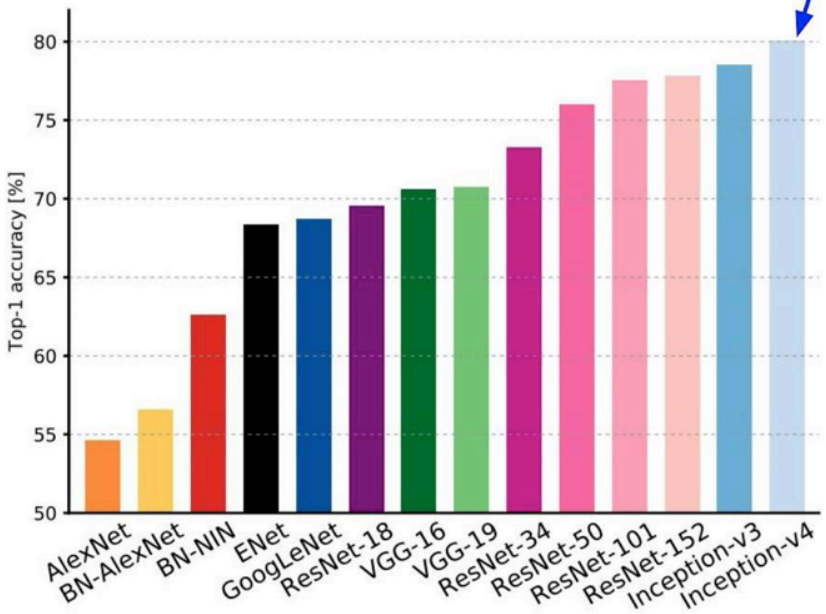
ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

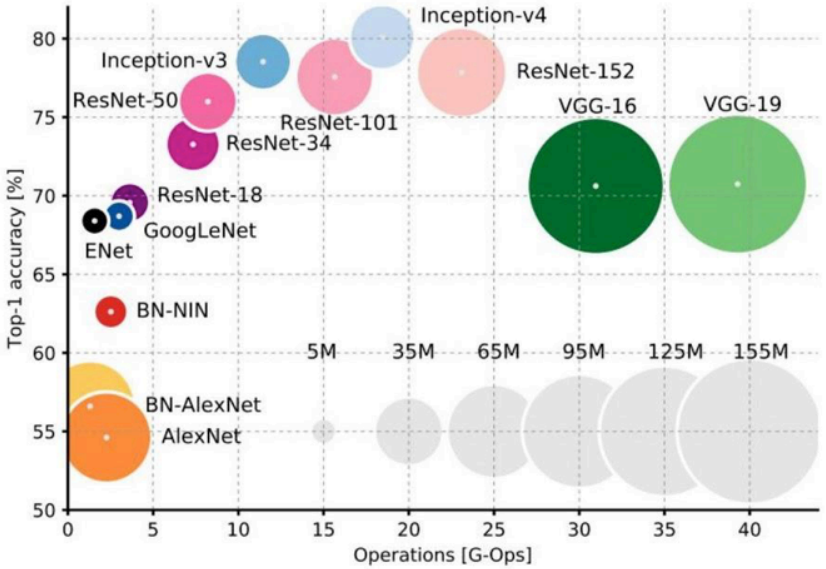
Comparative Analysis

CNNs Benchmark

Comparing complexity…

Inception-v4: Resnet + Inception!

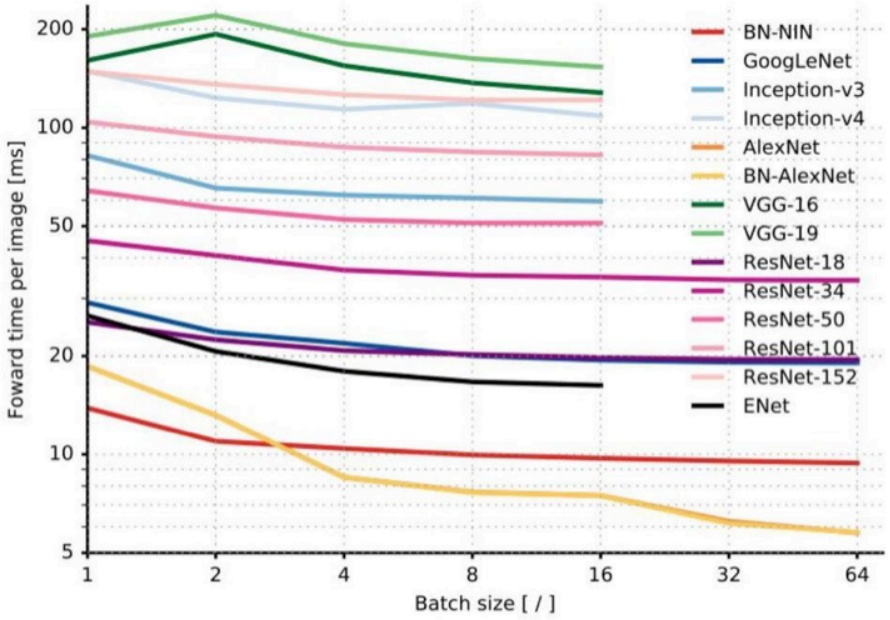


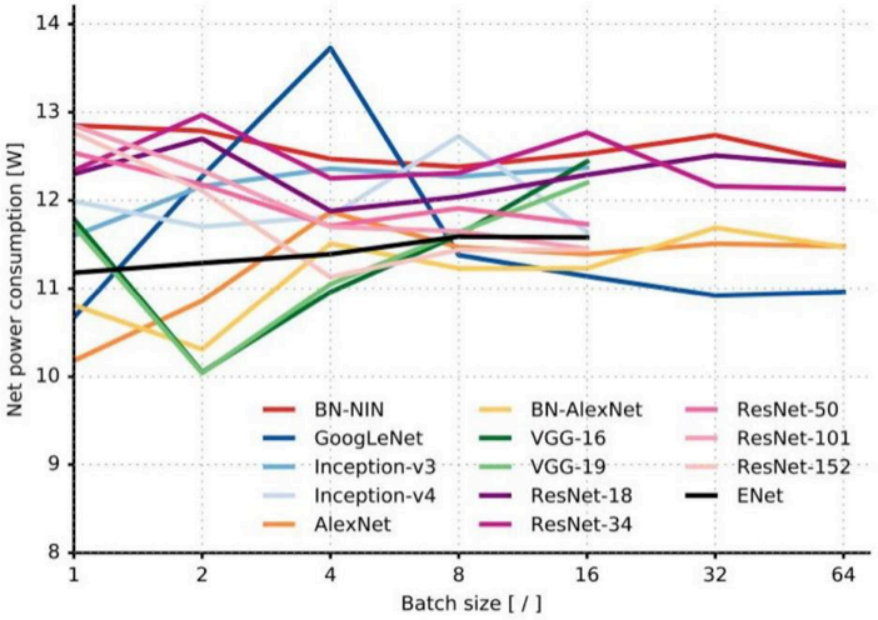


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

CNNs Benchmark

Forward pass time and power consumption





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

Transfer Learning

Transfer Learning

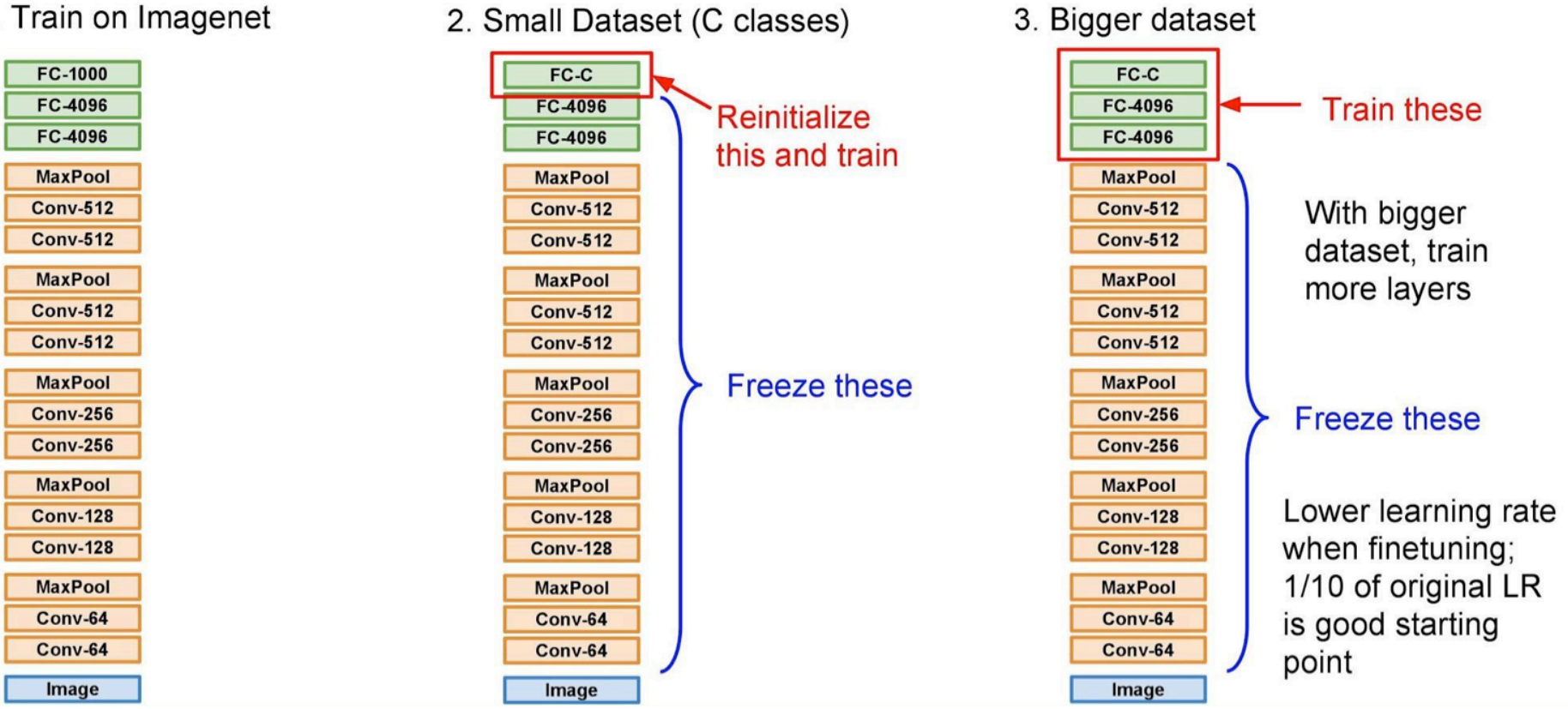
"You need a lot of a data if you want to train/use CNNs"

Transfer Learning

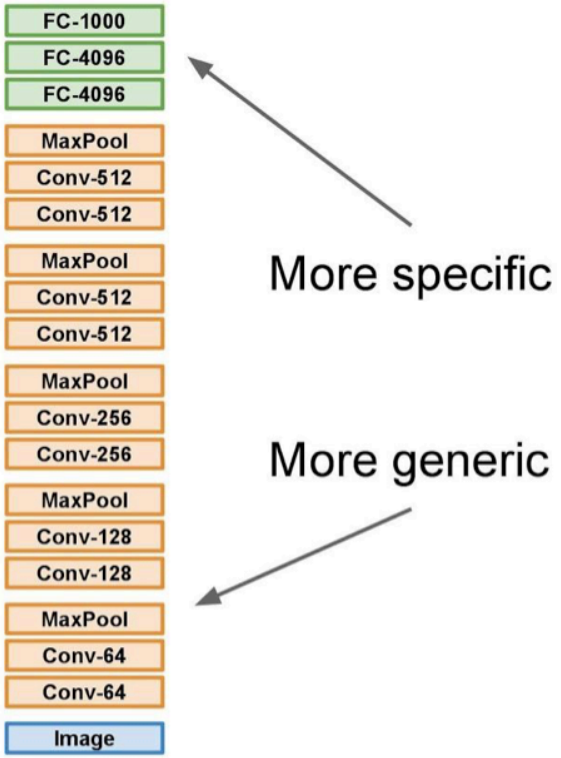
Transfer Learning

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



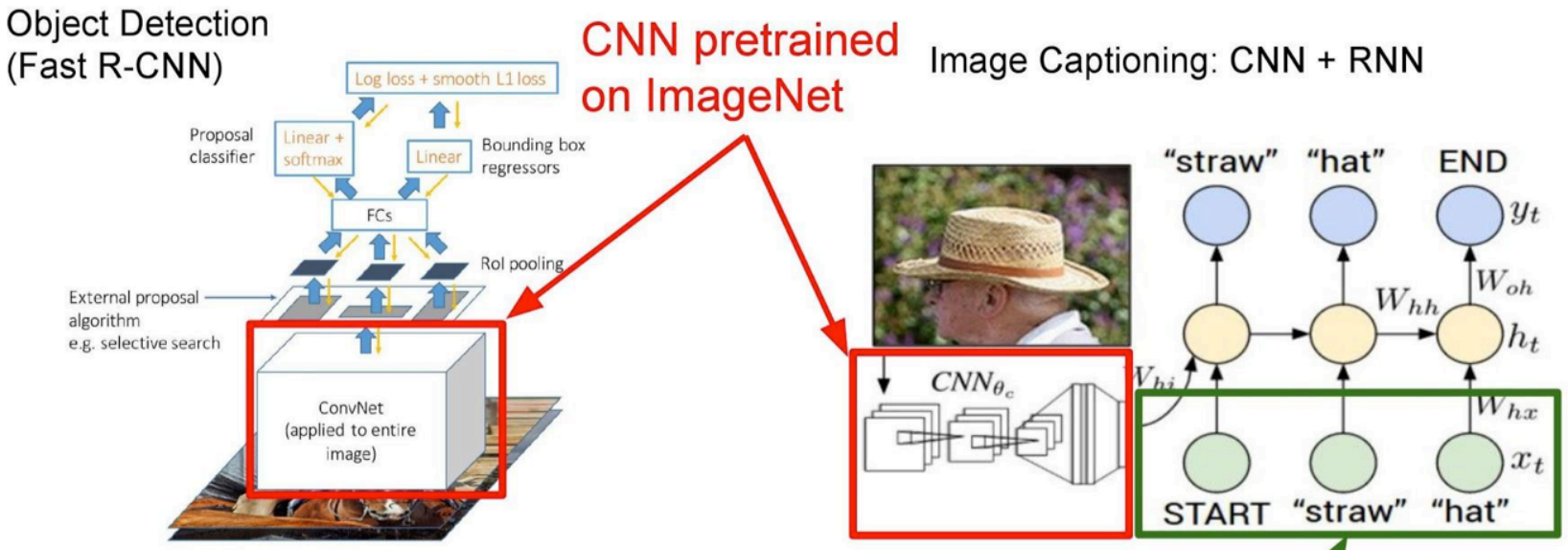
Transfer Learning



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer Learning

Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Bibliography

Bibliography

Deep Learning textbook: Deep Learning (Adaptive Computation and Machine Learning series)

Authors: Ian Goodfellow, Yoshua Bengio and Aaron Courville

Hardcover: 800 pages

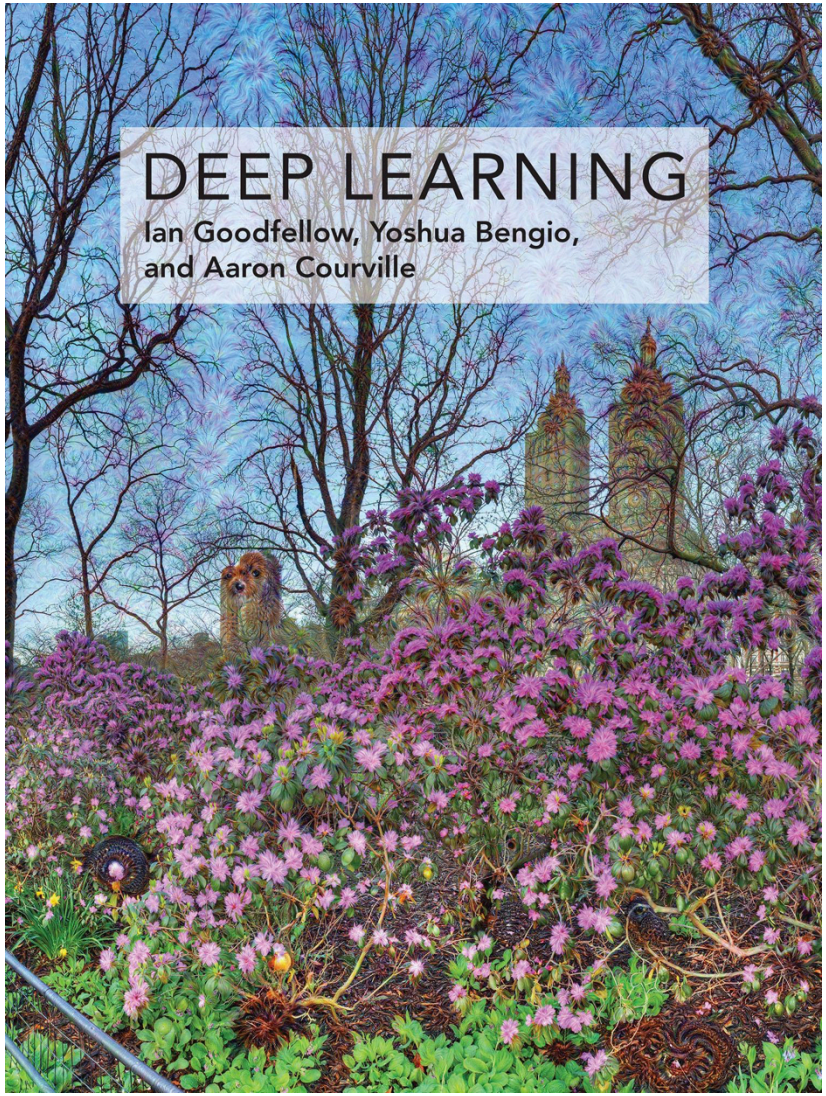
Publisher: The MIT Press

(November 18, 2016)

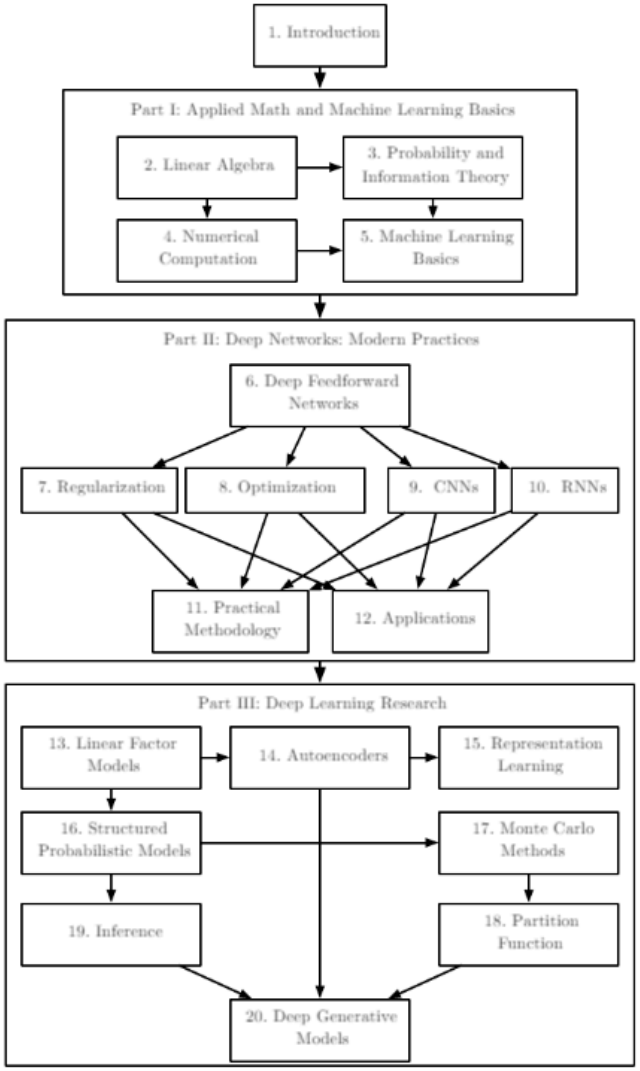
Language: English

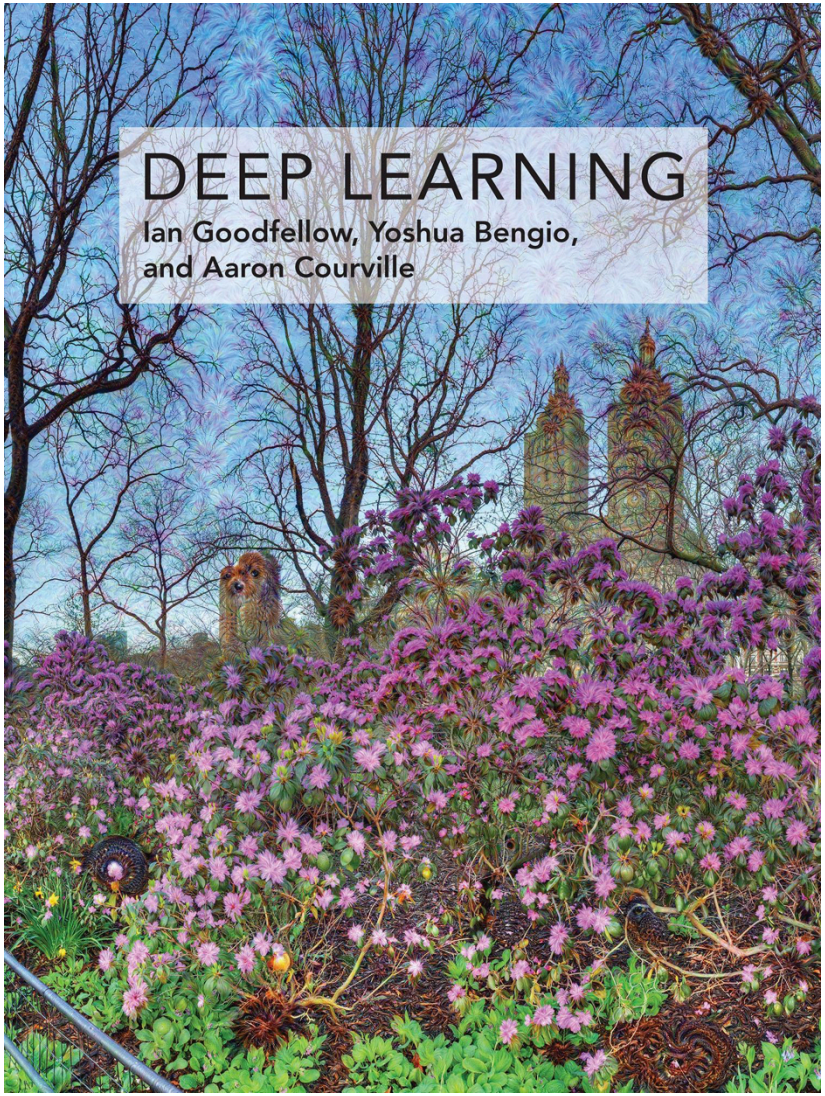
Link:

http://www.deeplearningbook.org

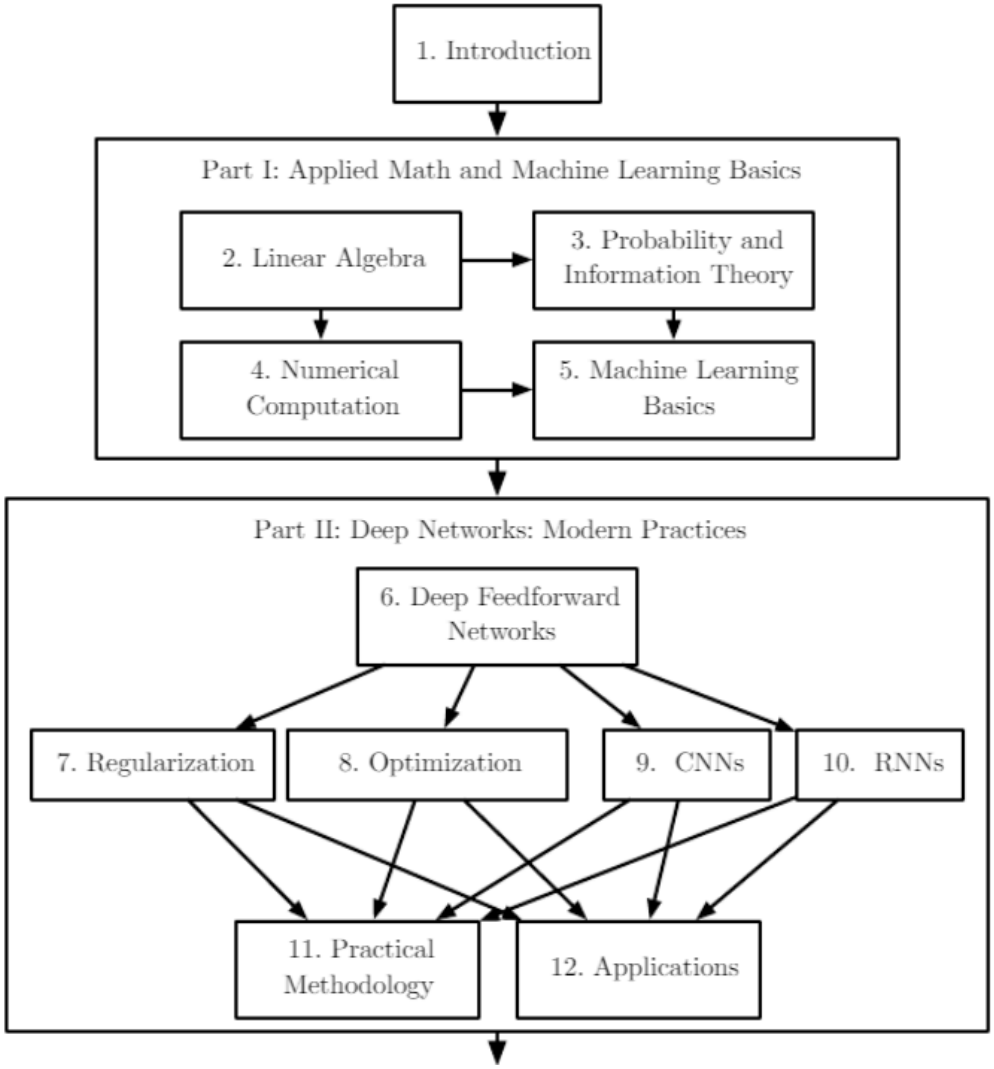


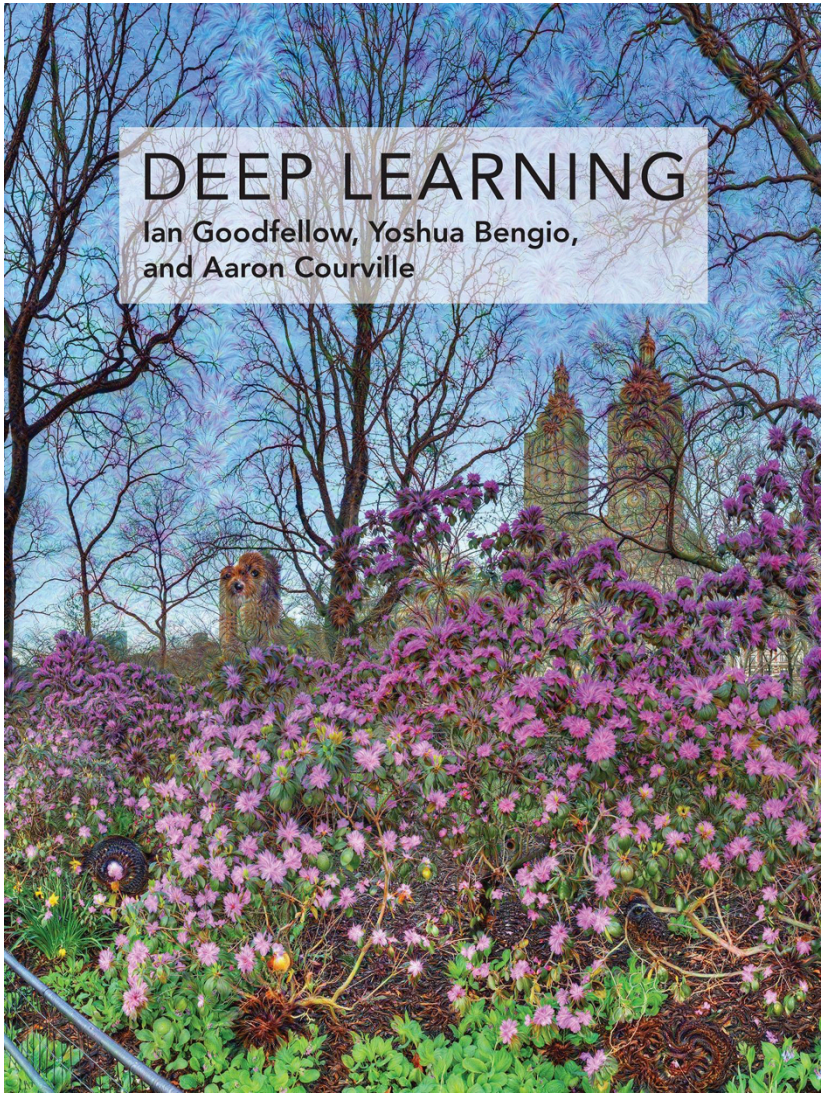
Bibliography



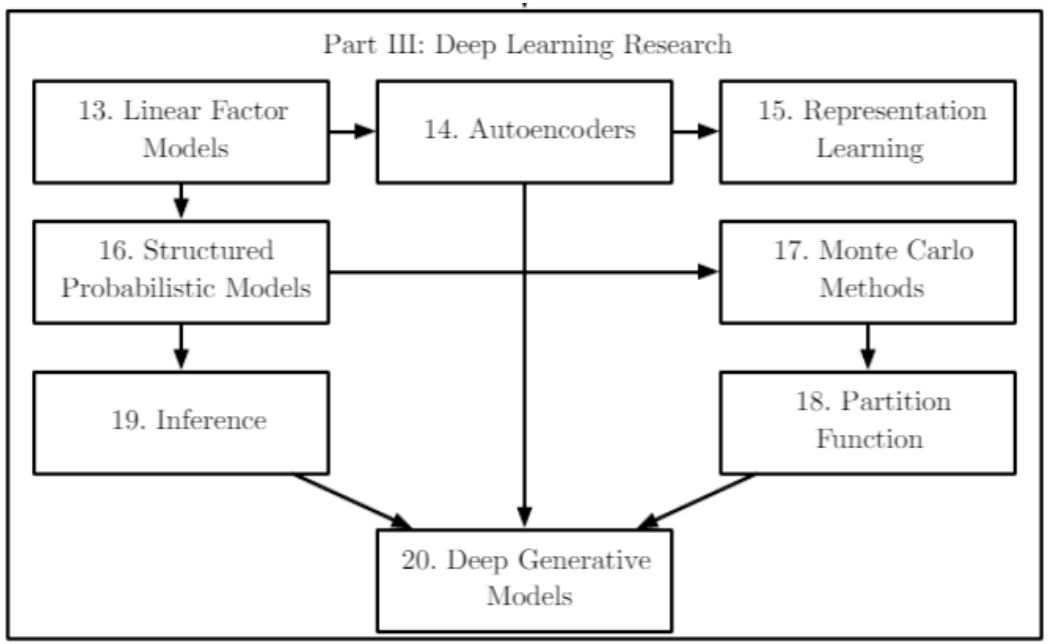


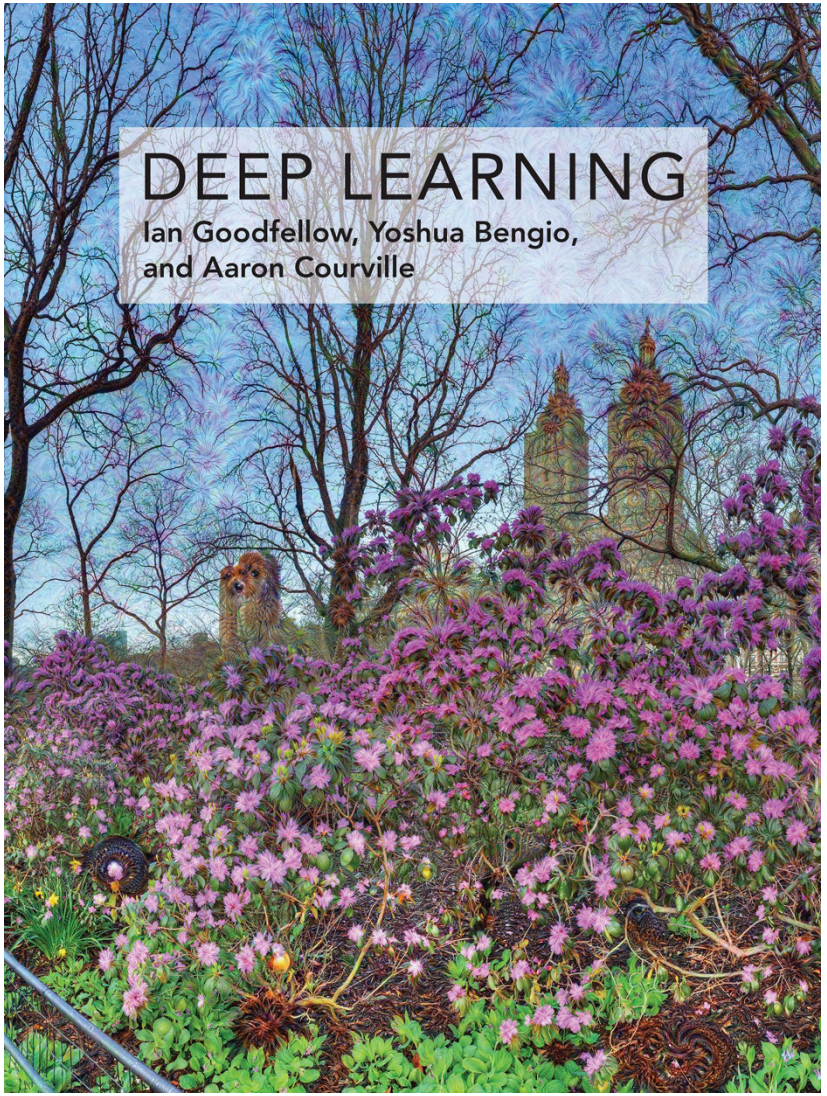
Bibliography





Bibliography





Bibliography

Written by three experts in the field, *Deep Learning* is the only comprehensive book on the subject. It provides much-needed broad perspective and mathematical preliminaries for software engineers and students entering the field, and serves as a reference for authorities.

(Elon Musk, cochair of OpenAI; cofounder and CEO of Tesla and SpaceX)

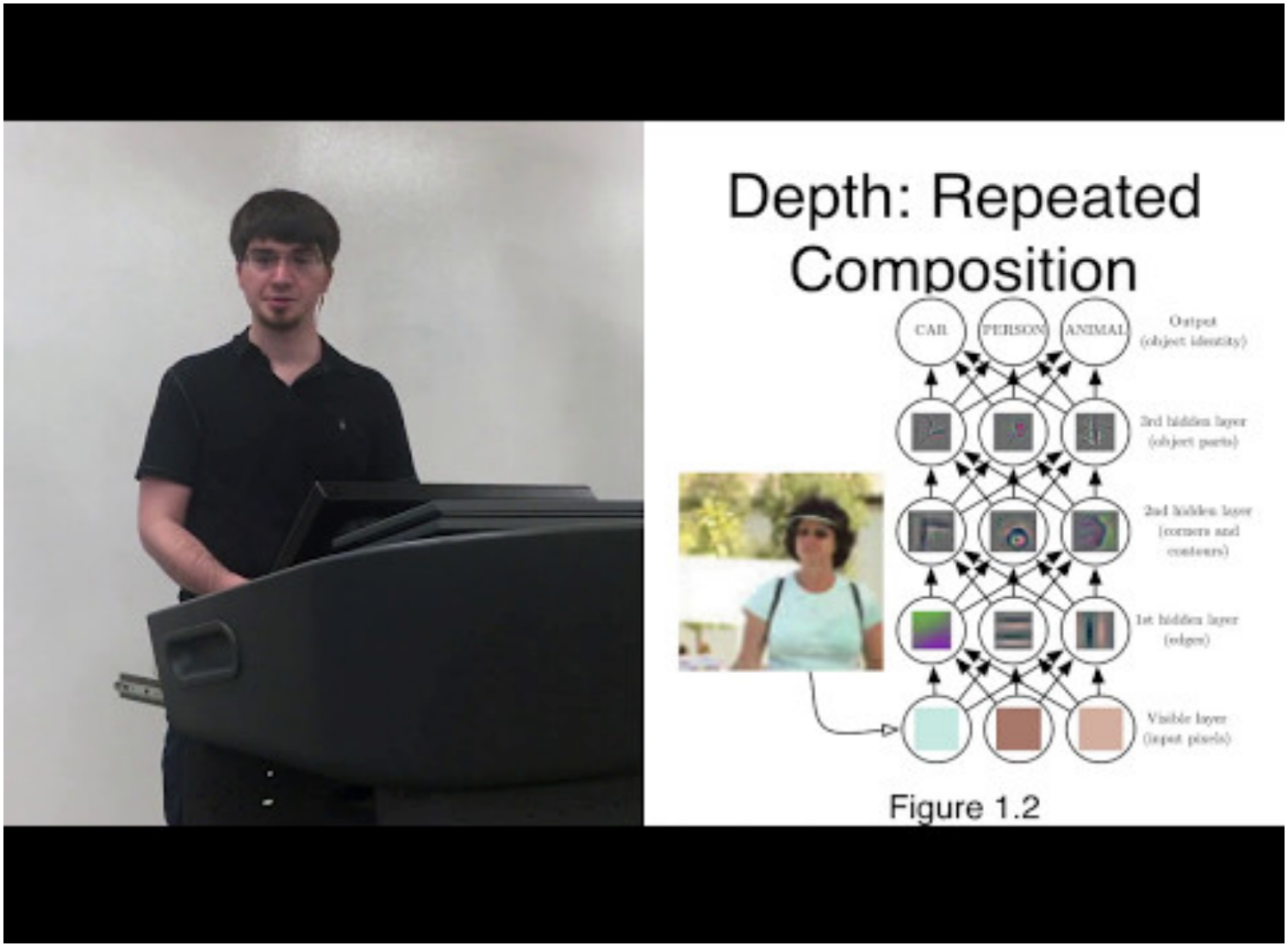
This is the definitive textbook on deep learning. Written by major contributors to the field, it is clear, comprehensive, and authoritative. If you want to know where deep learning came from, what it is good for, and where it is going, read this book.

(Geoffrey Hinton FRS, Emeritus Professor, University of Toronto; Distinguished Research Scientist, Google)

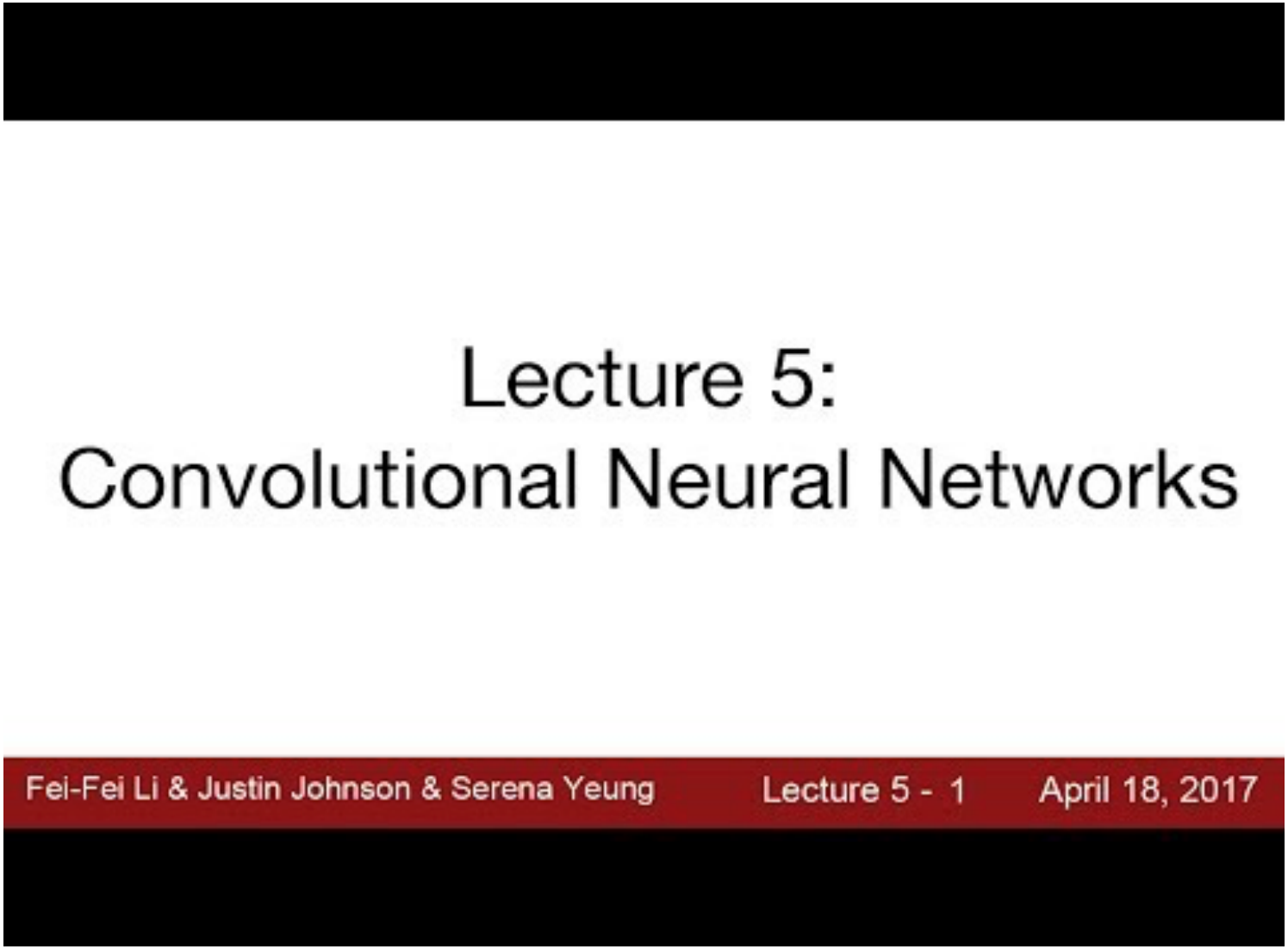
Deep learning has taken the world of technology by storm since the beginning of the decade. There was a need for a textbook for students, practitioners, and instructors that includes basic concepts, practical aspects, and advanced research topics. This is the first comprehensive textbook on the subject, written by some of the most innovative and prolific researchers in the field. This will be a reference for years to come.

(Yann LeCun, Director of AI Research, Facebook; Silver Professor of Computer Science, Data Science, and Neuroscience, New York University)

Bibliography



Bibliography! Thanks!



Bibliography

Thank you!