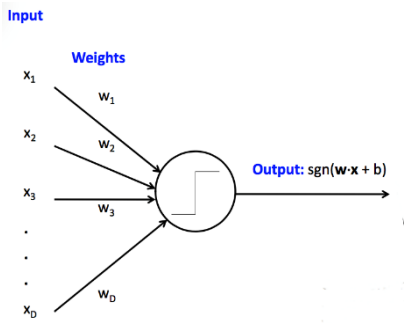
Deep Learning

Applied Machine Learning
Derek Hoiem

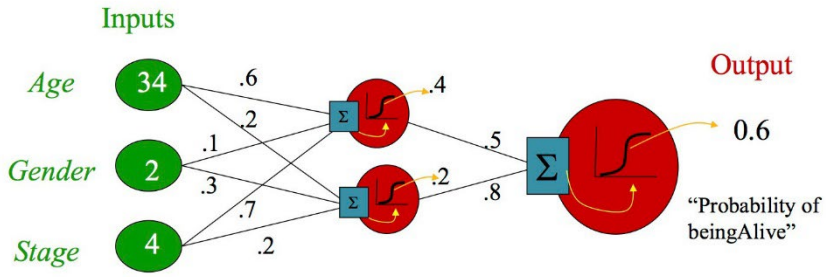
Dall-E

Last class

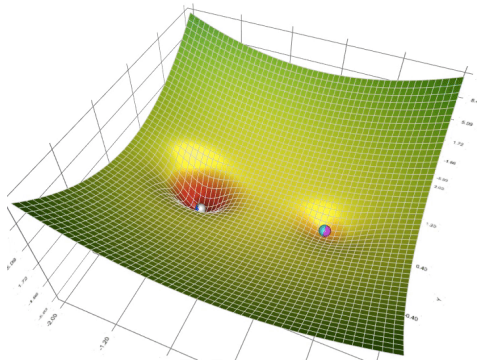
- Perceptrons are linear prediction models



- MLPs are non-linear prediction models, composed of multiple linear layers with non-linear activations



- MLPs can model more complex functions, but are harder to optimize



- Optimization is by stochastic gradient descent

Figure source

Another application example: mapping position/rays to color

- L2 loss
- ReLU MLP with 4 layers and 256 channels (nodes per layer)
- Sigmoid activation on output
- 256 frequency positional encoding

Fourier Features (Tancik et al. 2020) NeRF (Mildenhall et al. 2020)

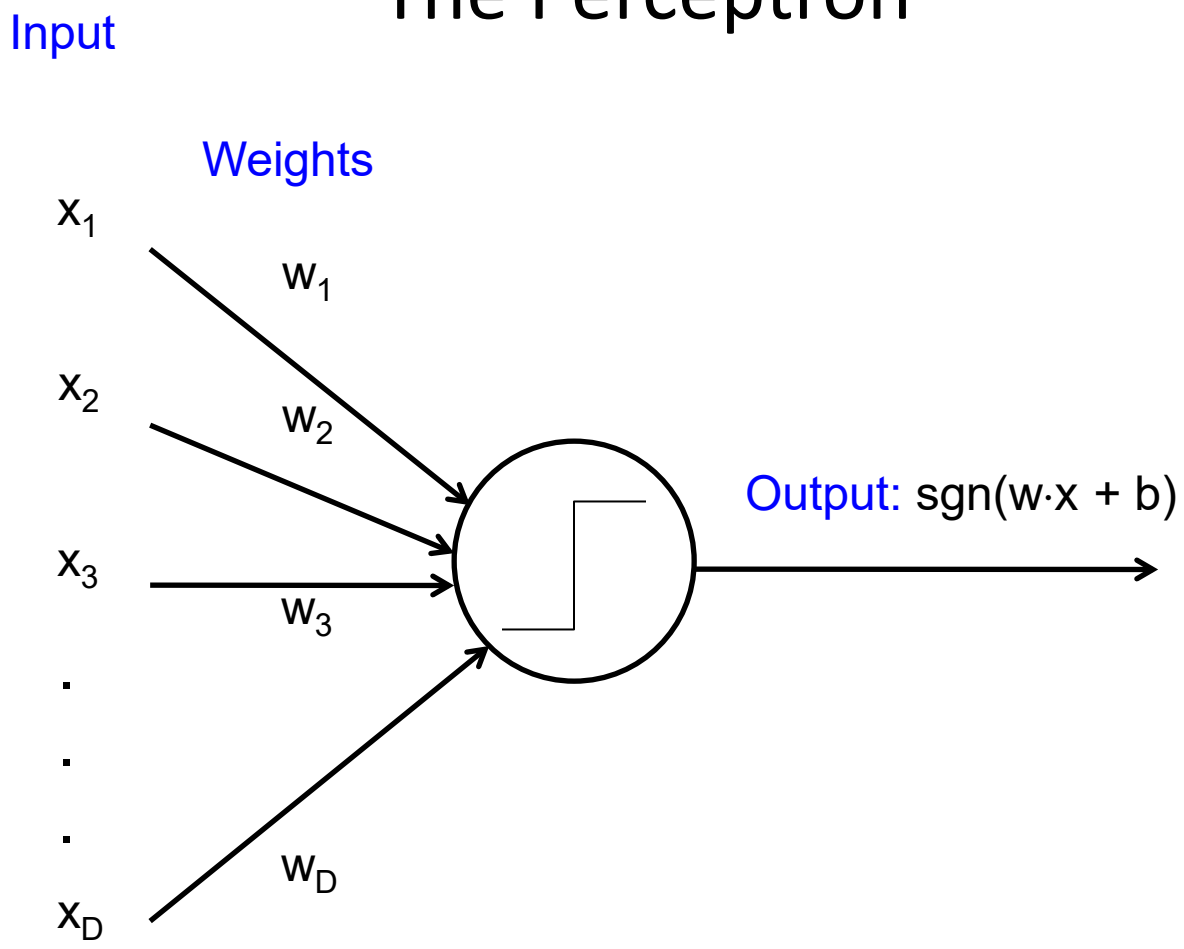
HW 2

https://docs.google.com/document/d/13vTEGx3fdfc4rtcF86xoUyXm6eHmuvys5ZkMbcEgK4s/edit

Today’s Lecture

- The story of how deep learning became so important
- Optimization
- Residual Networks

The Perceptron



Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408.

1958 New York Times…

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

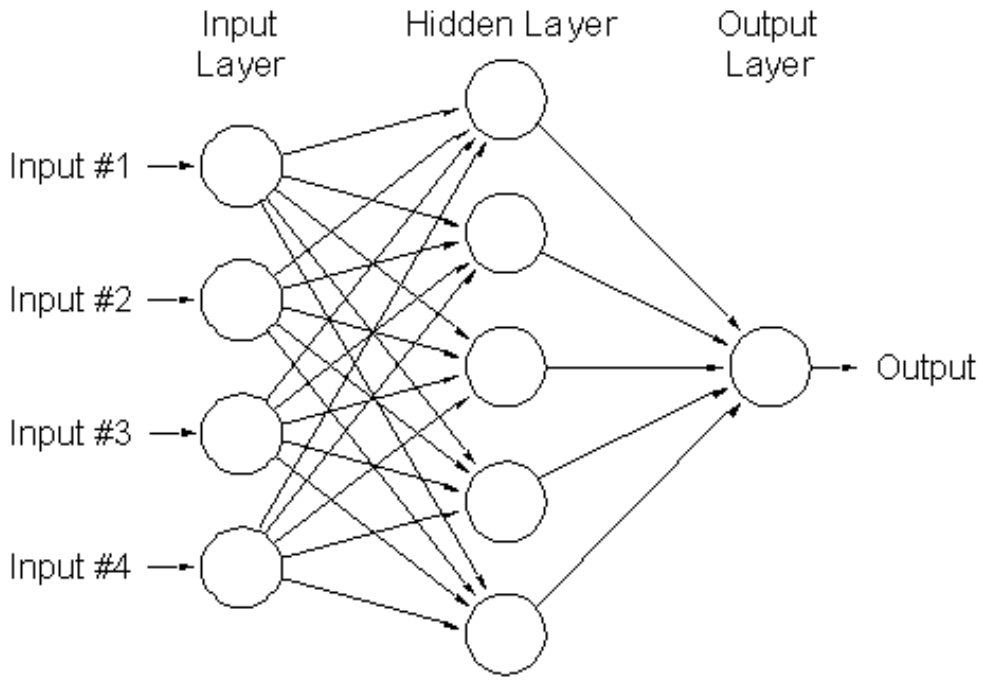
Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

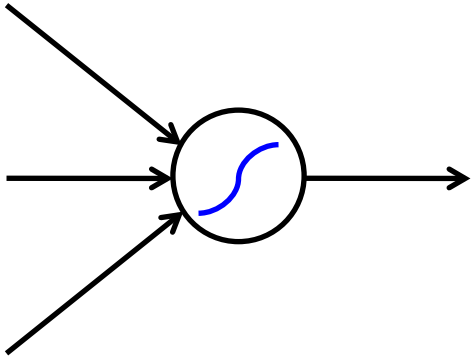
Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Two-layer neural network



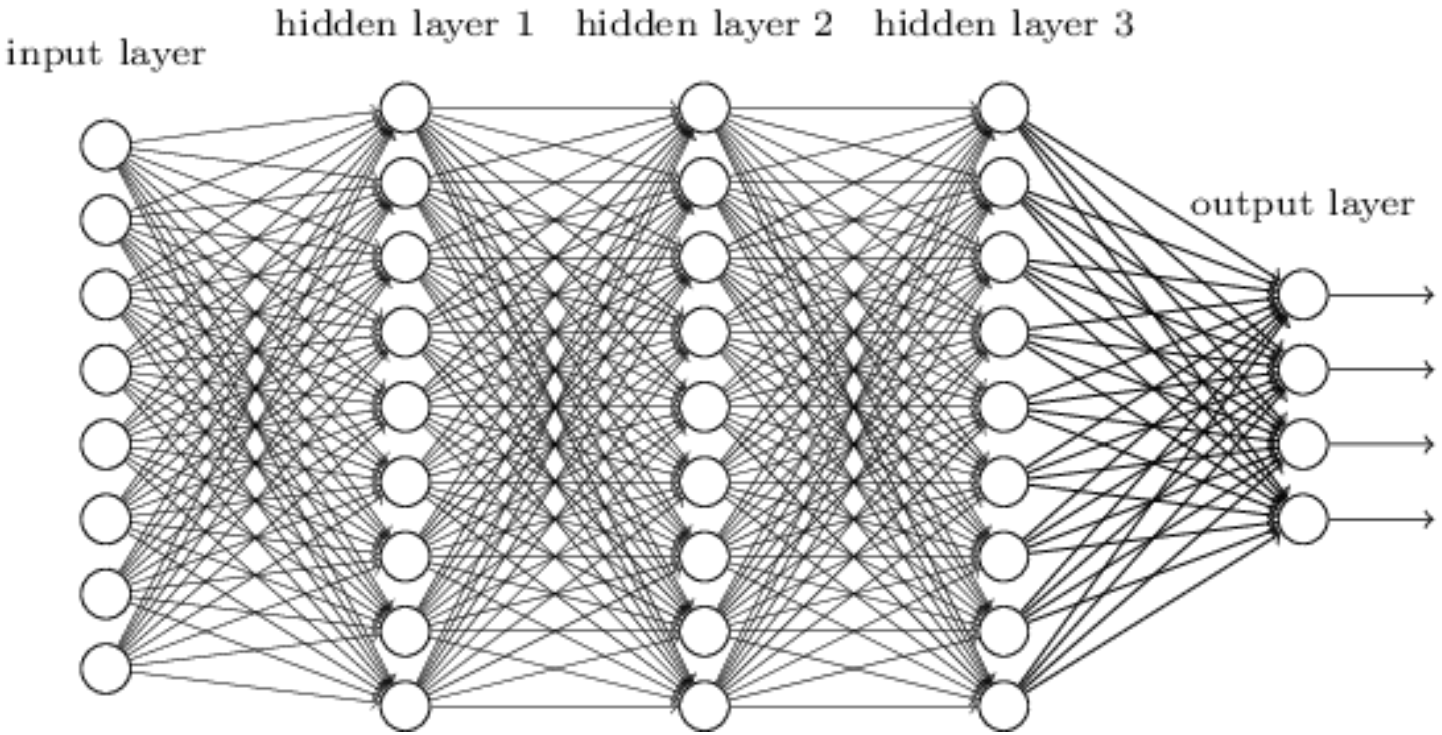
- Can learn nonlinear functions provided each perceptron has a differentiable nonlinearity



Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$

Slide: Lazebnik

Multi-layer neural network



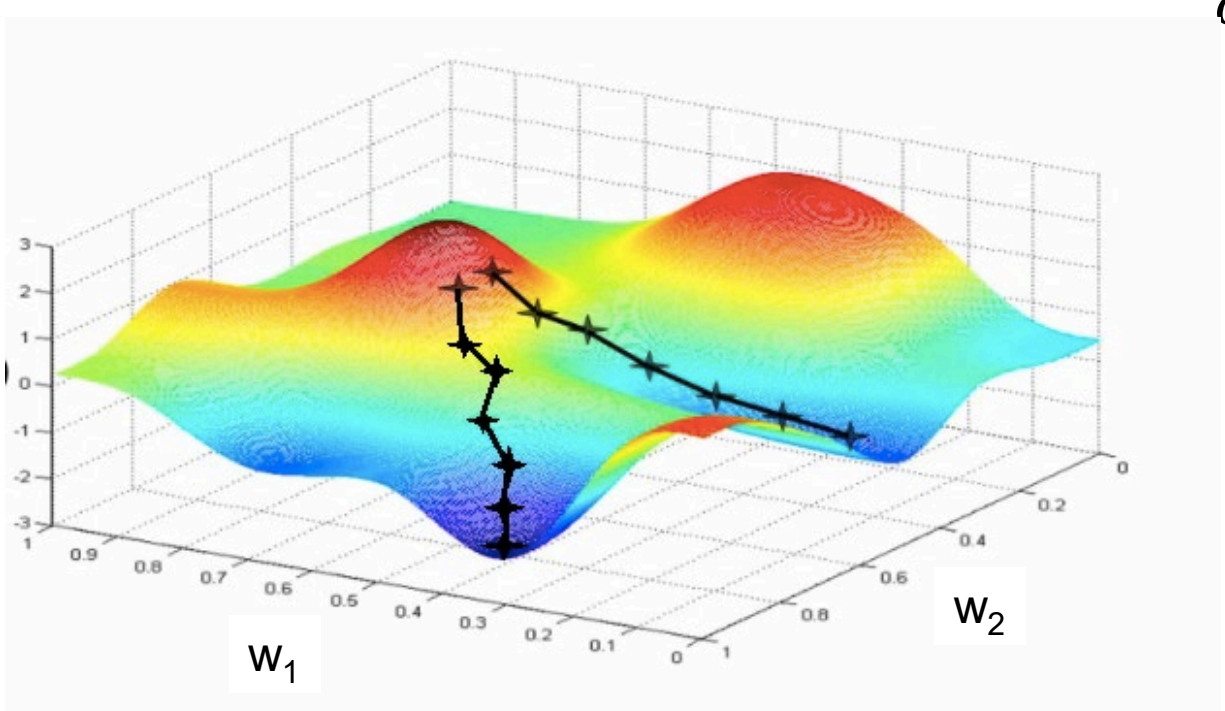
Slide: Lazebnik

Training of multi-layer networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f_{\mathbf{w}}(\mathbf{x}_i) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



Slide: Lazebnik

Training of multi-layer networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f_{\mathbf{w}}(\mathbf{x}_i) \right)^2$$

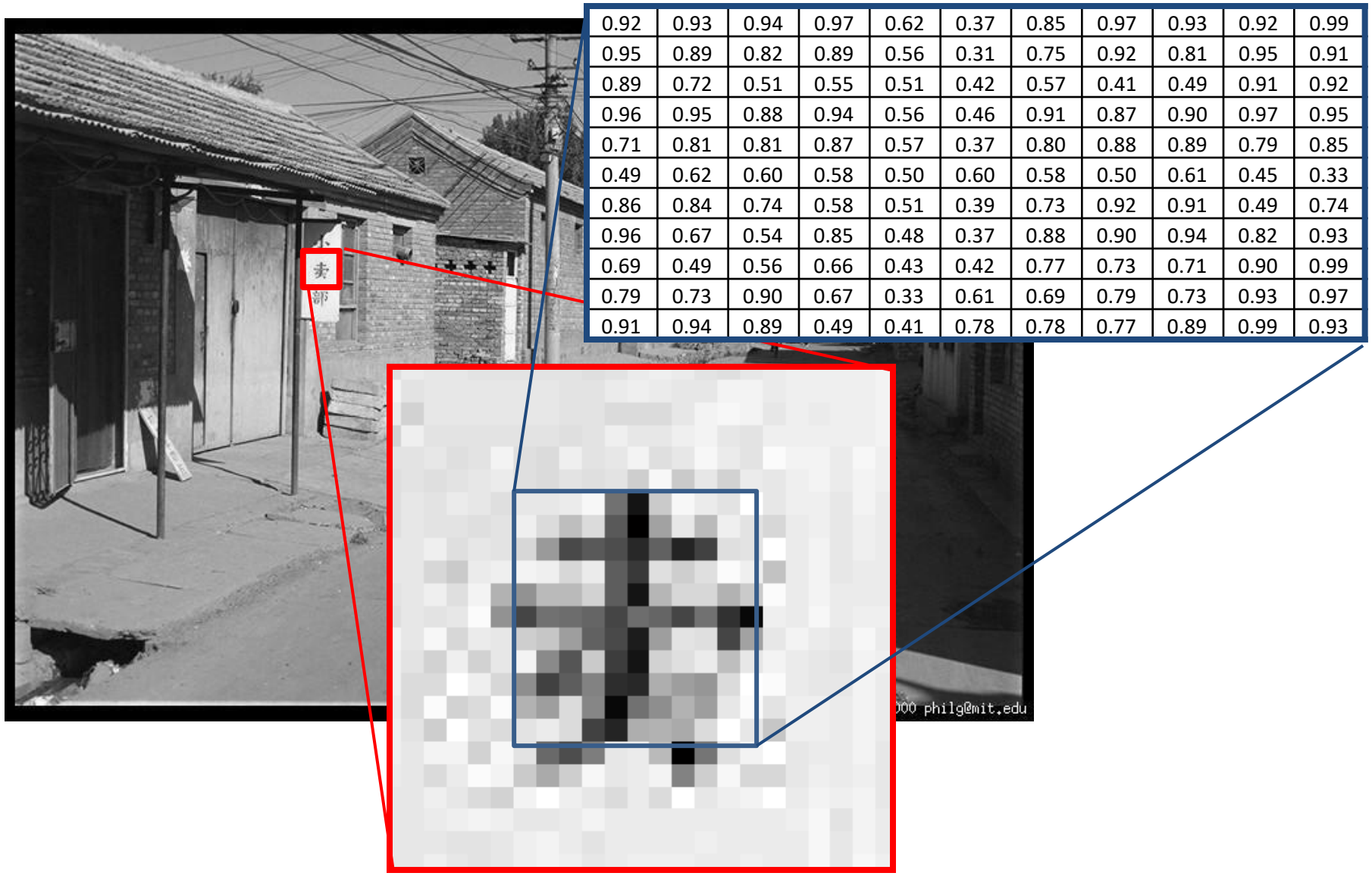
- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. a small batch of examples at a time, cycle through training examples in random order in multiple epochs

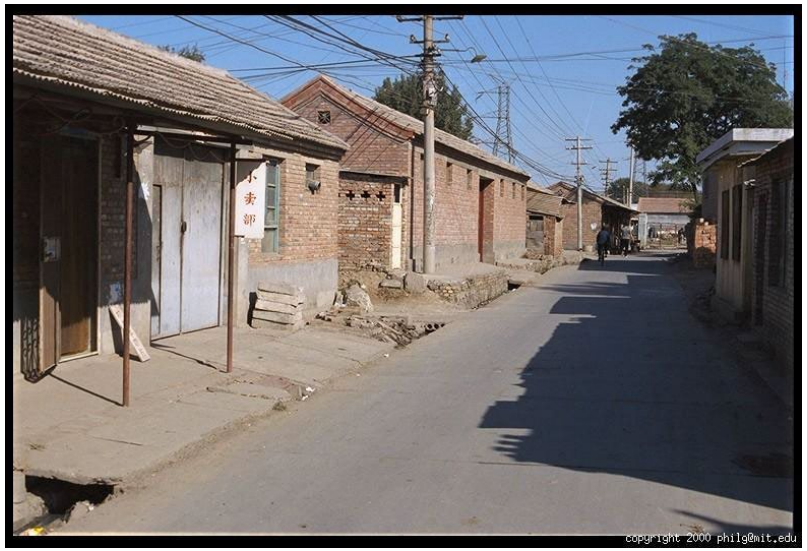
Slide: Lazebnik

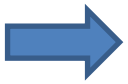
MLPs on Images

The raster image (pixel matrix)



Color Image


copyright 2000 philg@mit.edu

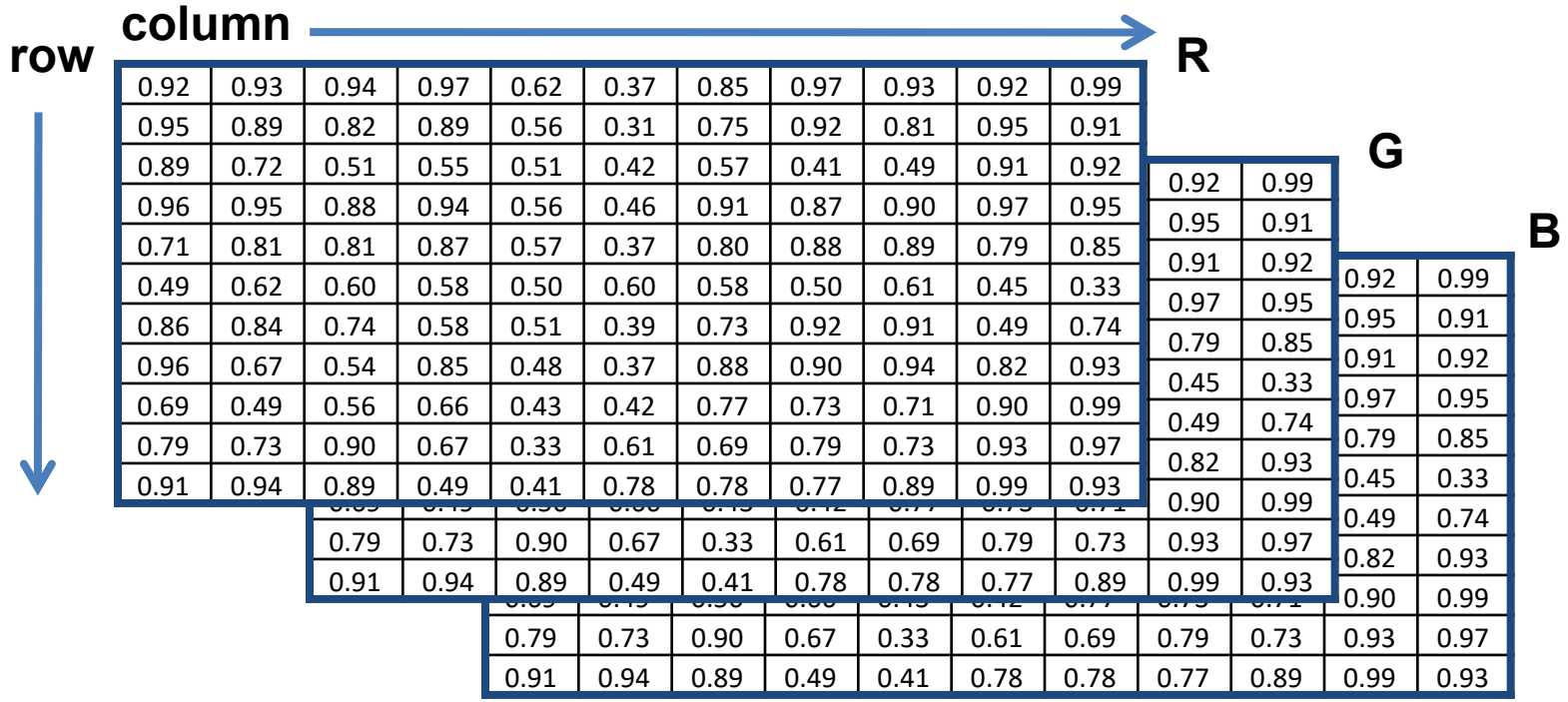



R
G
B
copyright 2000 philg@mit.edu

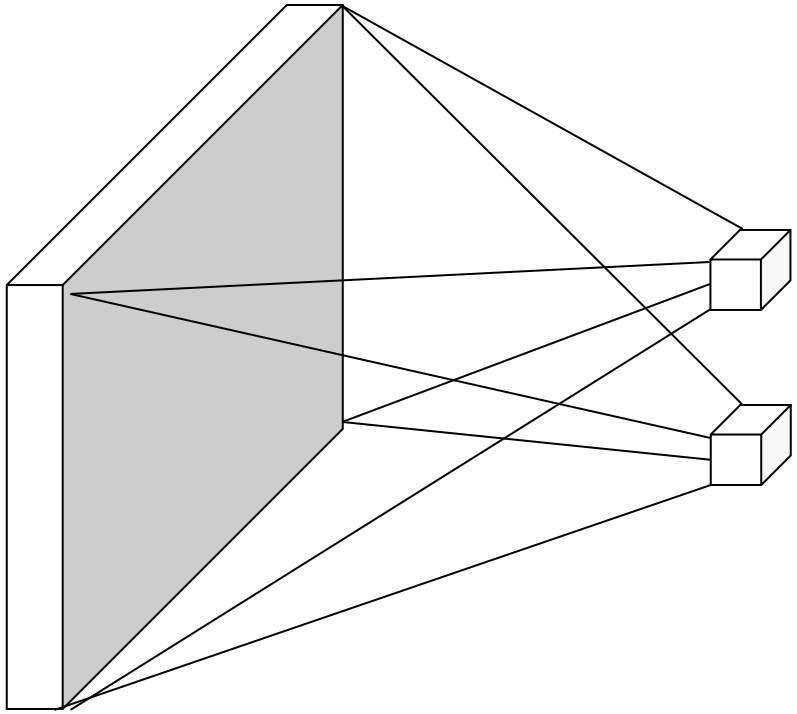
Images in Python

```
im = cv2.imread(filename)                    # read image
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB) # order channels as RGB
im = im / 255                                # values range from 0 to 1
```

- RGB image `im` is a H x W x 3 matrix (numpy.ndarray)
- `im[0,0,0]` = top-left pixel value in R-channel
- `im[y, x, c]` = y+1 pixels down, x+1 pixels to right in the c^{th} channel
- `im[H-1, W-1, 2]` = bottom-right pixel in B-channel

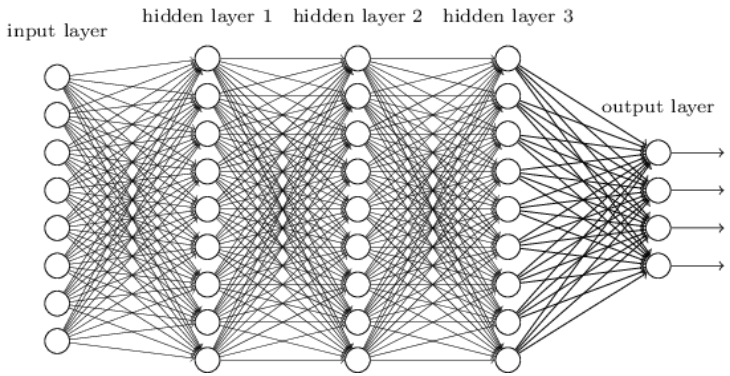


From fully connected to convolutional networks



image

Fully connected layer



You could treat the image like a vector of values and add fully connected layers (which is what we do in HW 1 and 2)

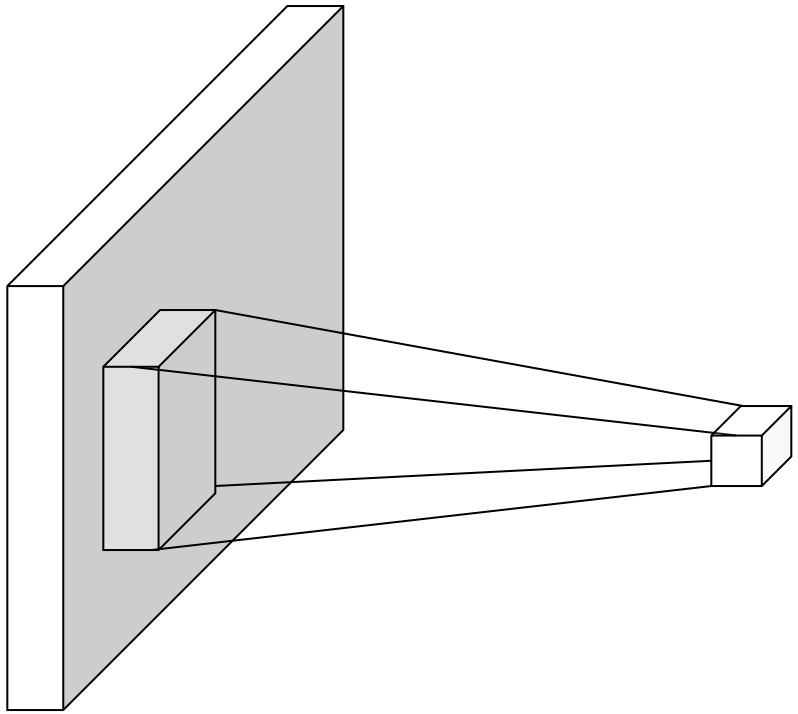
But this doesn't take advantage of the 2D structure of images

Slide: Lazebnik

Image filtering

- Image filtering: compute function of local neighborhood at each position
  - Enhance images
    - Denoise, resize, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching

From fully connected to convolutional networks

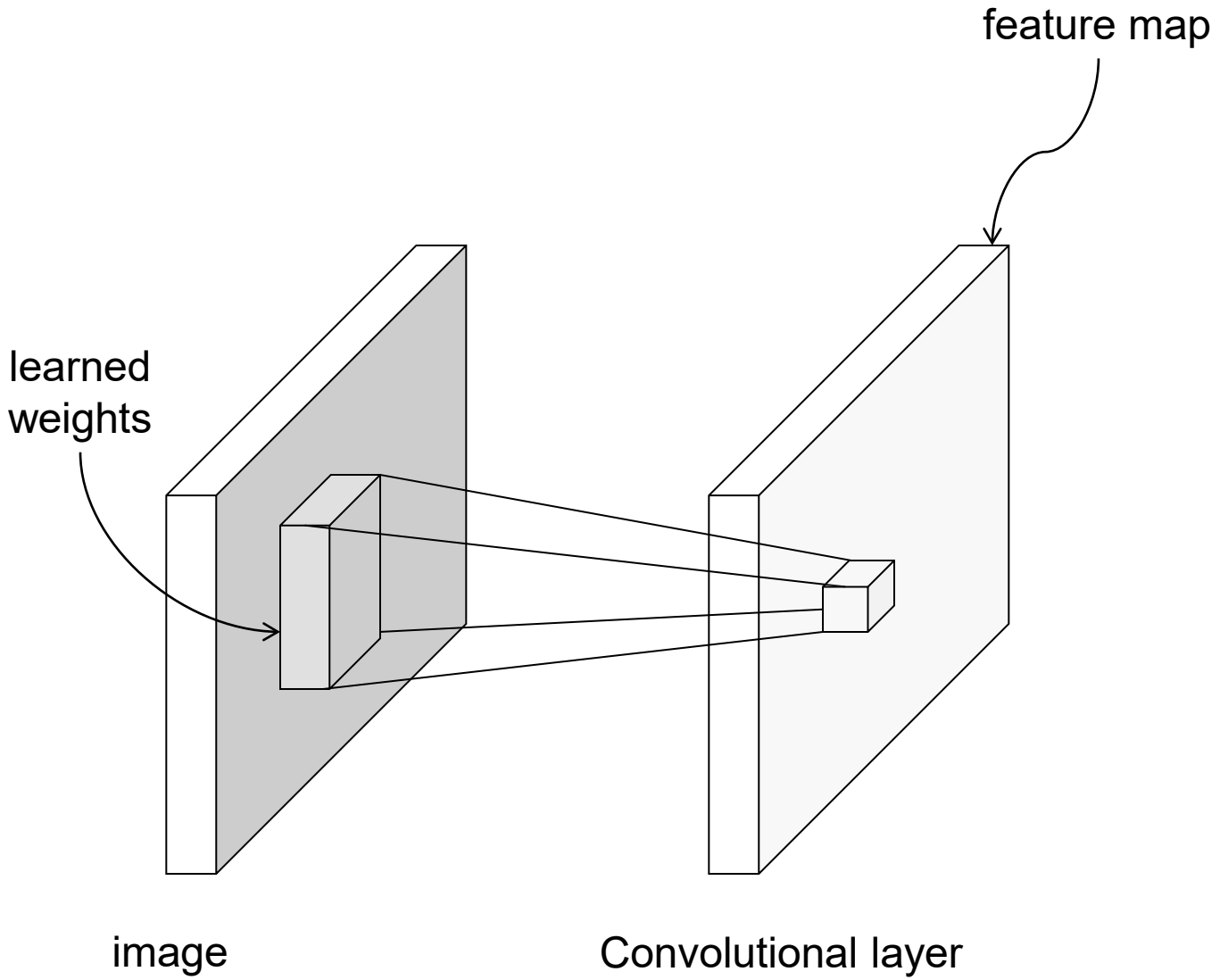


image

Convolutional layer

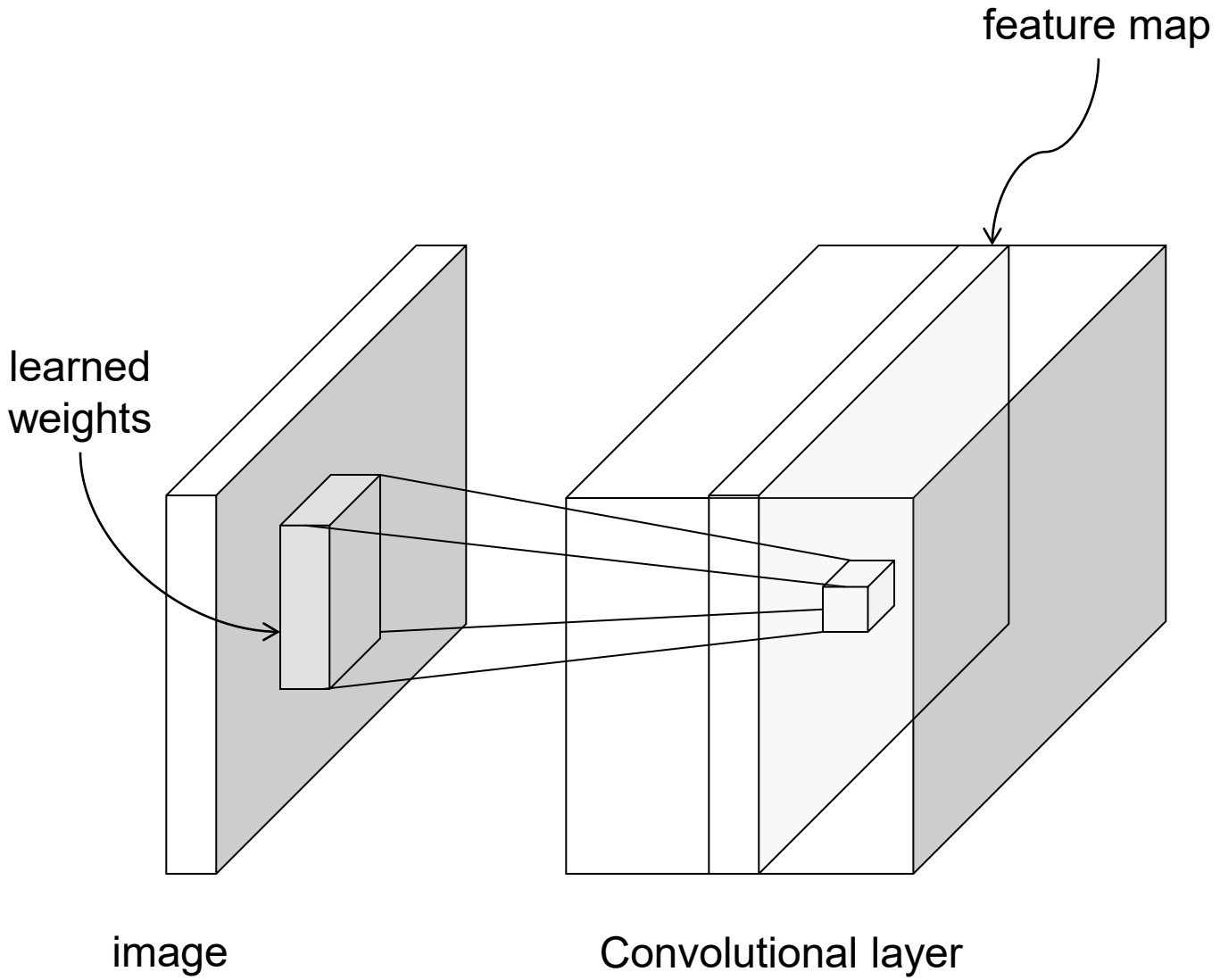
Slide: Lazebnik

From fully connected to convolutional networks



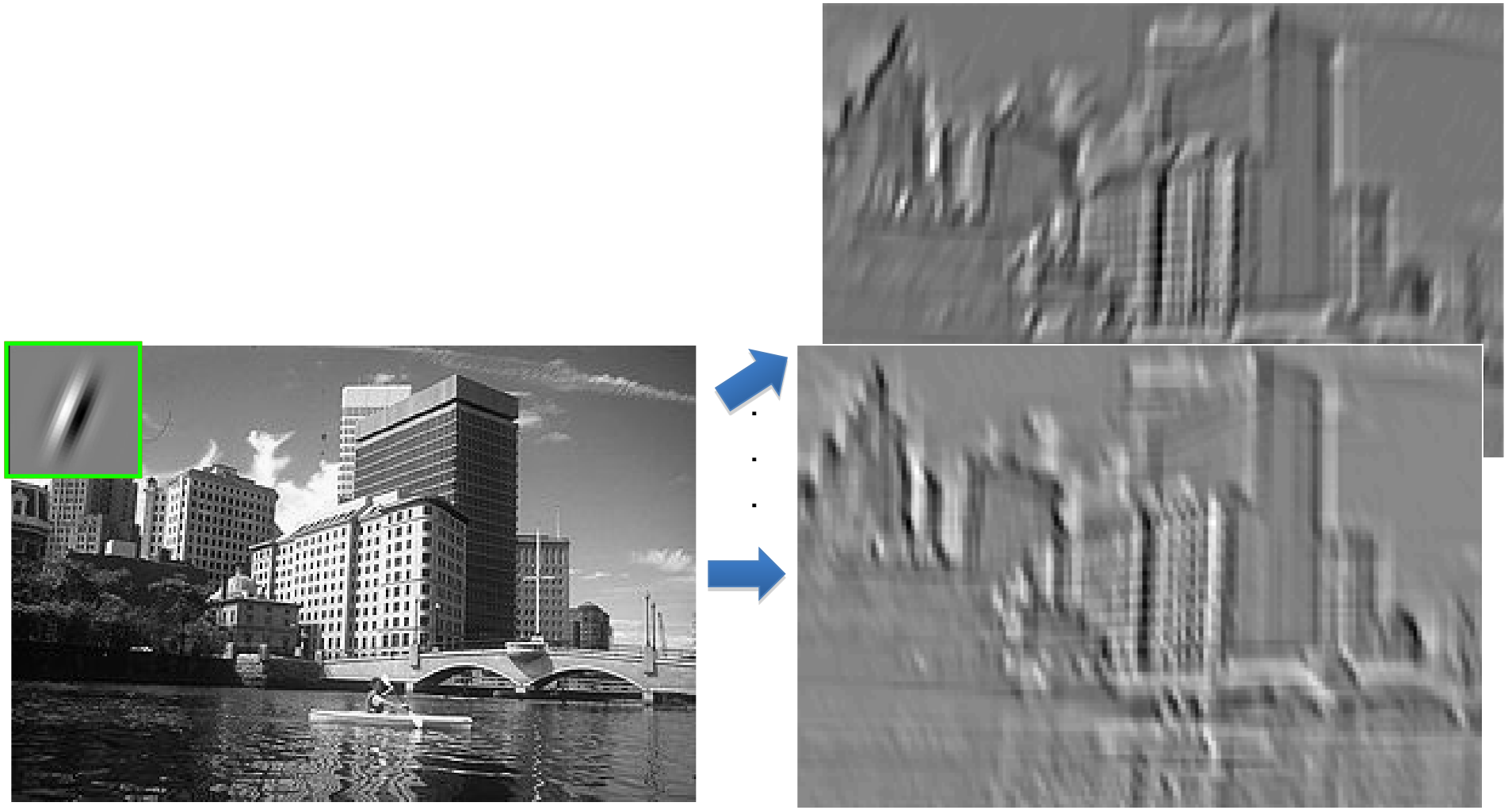
Slide: Lazebnik

From fully connected to convolutional networks



Slide: Lazebnik

Convolution as feature extraction

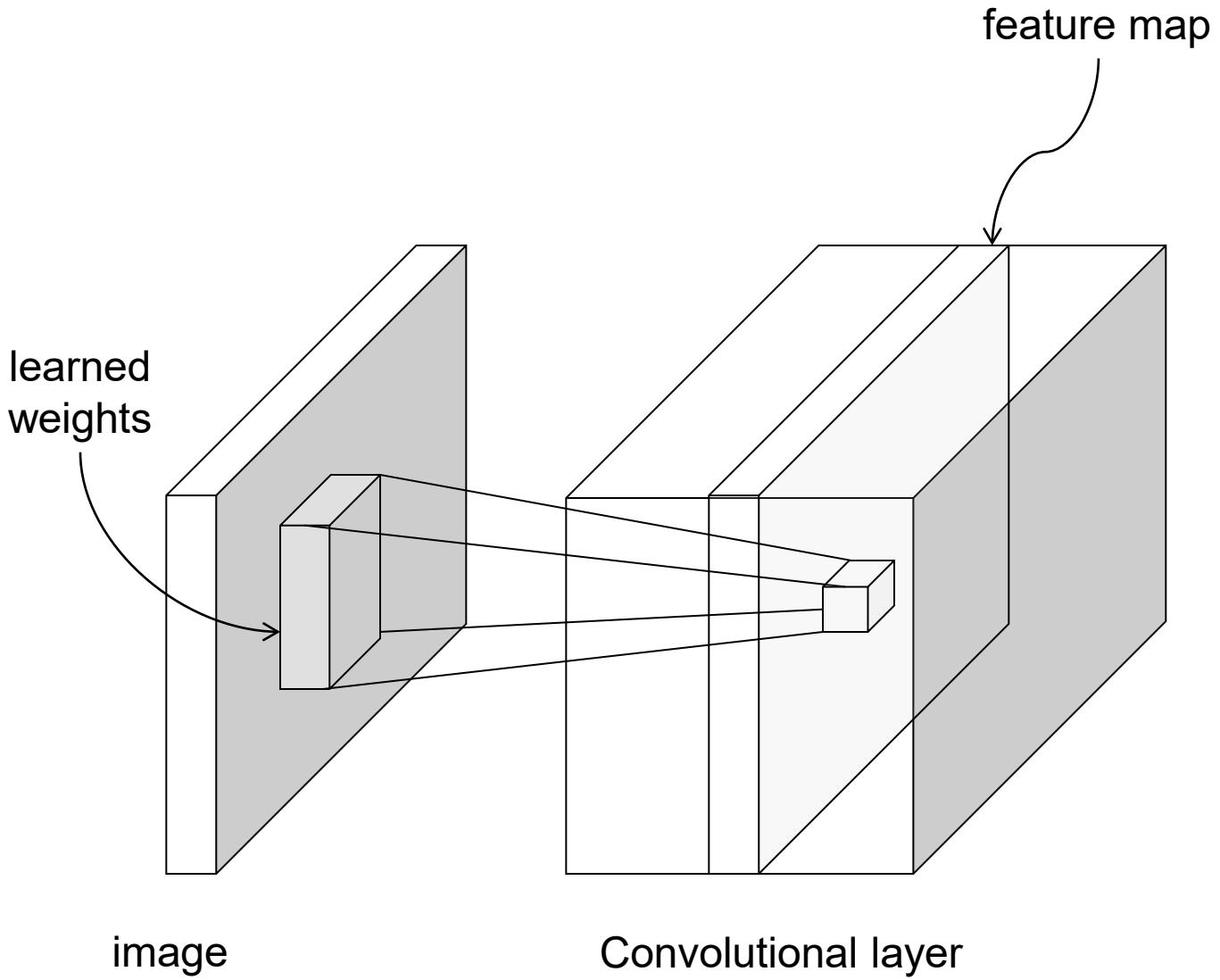


Input

Feature Map

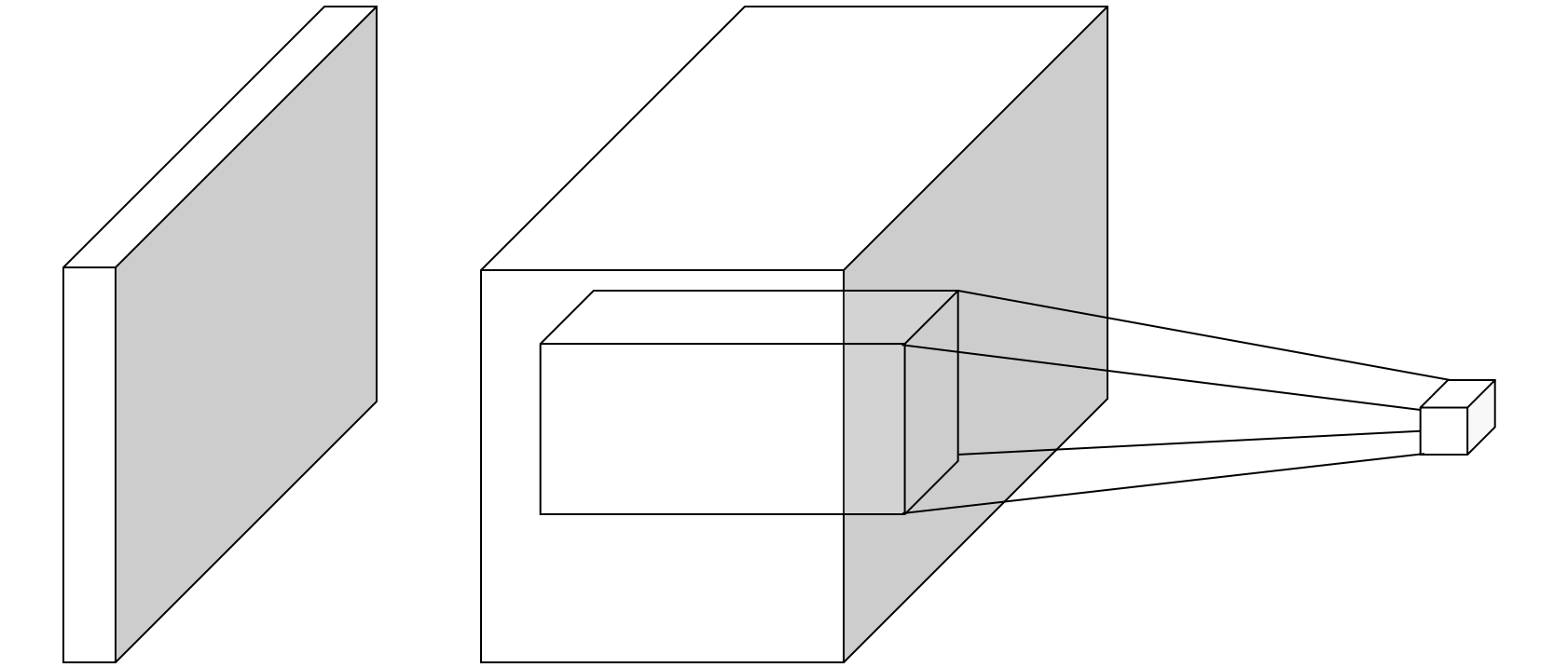
Slide: Lazebnik

From fully connected to convolutional networks



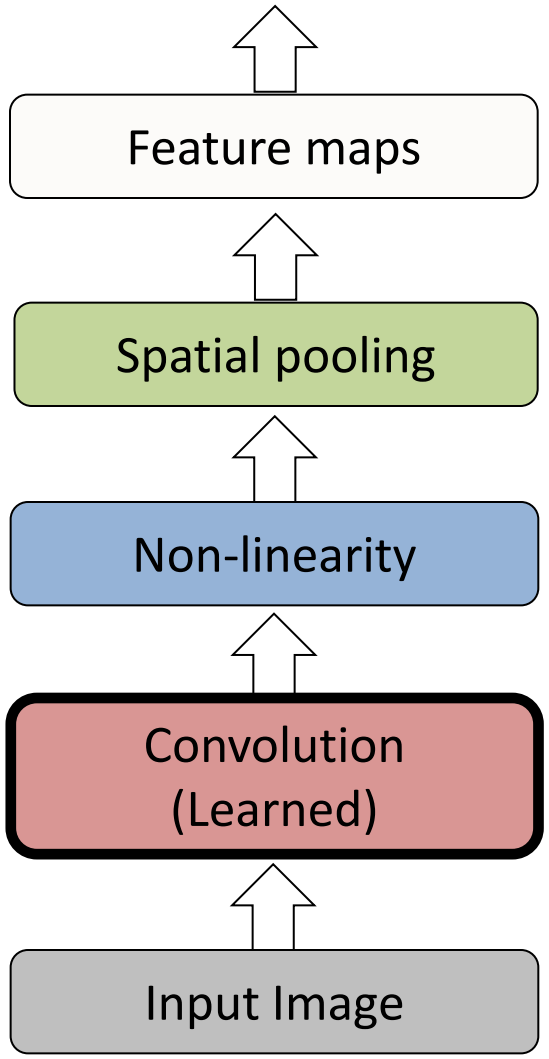
Slide: Lazebnik

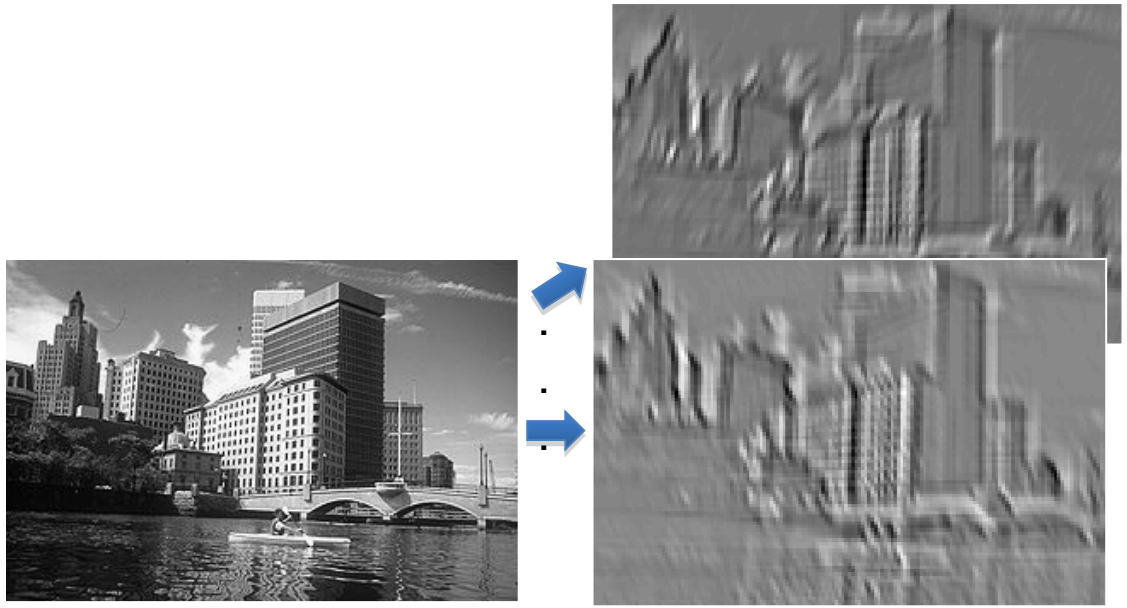
From fully connected to convolutional networks



Slide: Lazebnik

Key operations in a CNN





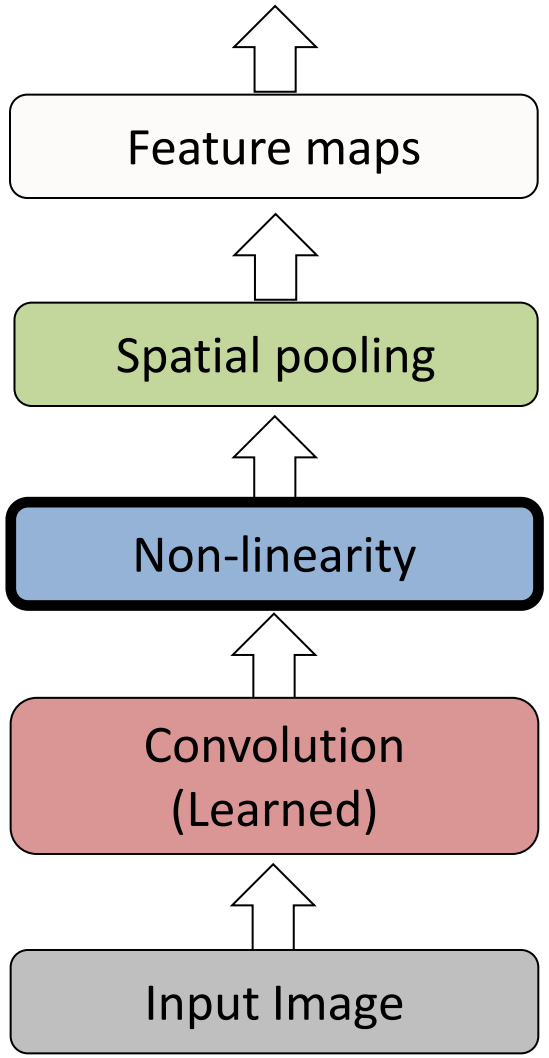
Input

Feature Map

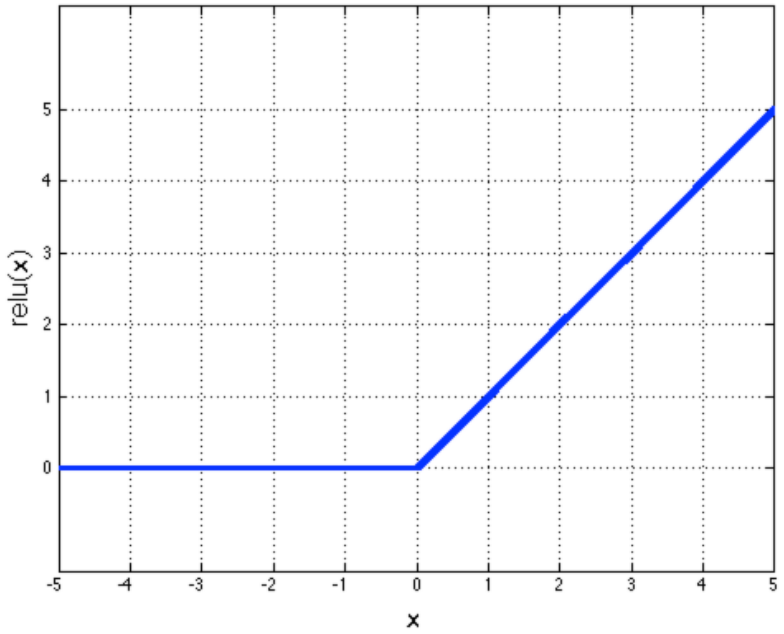
Source: R. Fergus, Y. LeCun

Slide: Lazebnik

Key operations



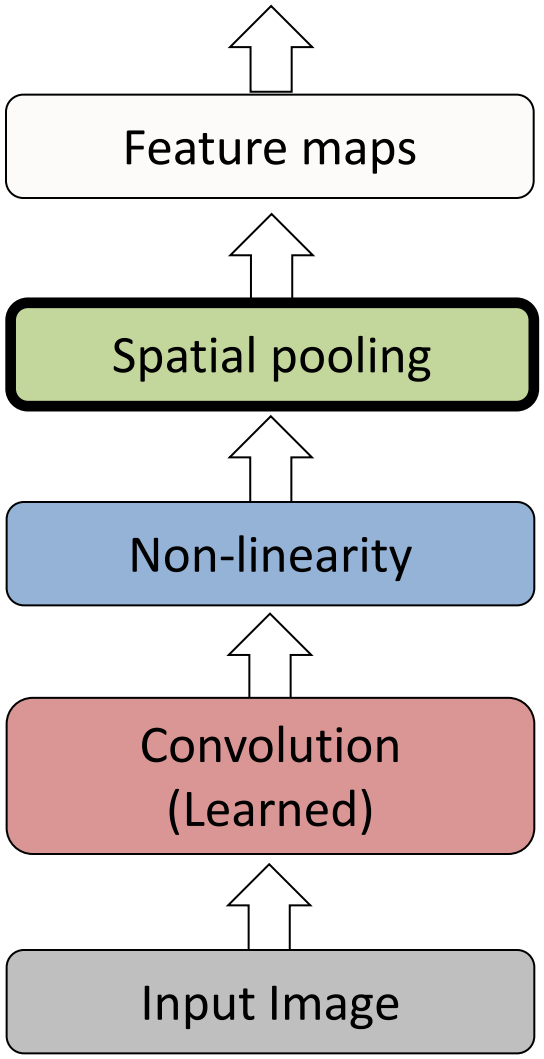
Rectified Linear Unit (ReLU)

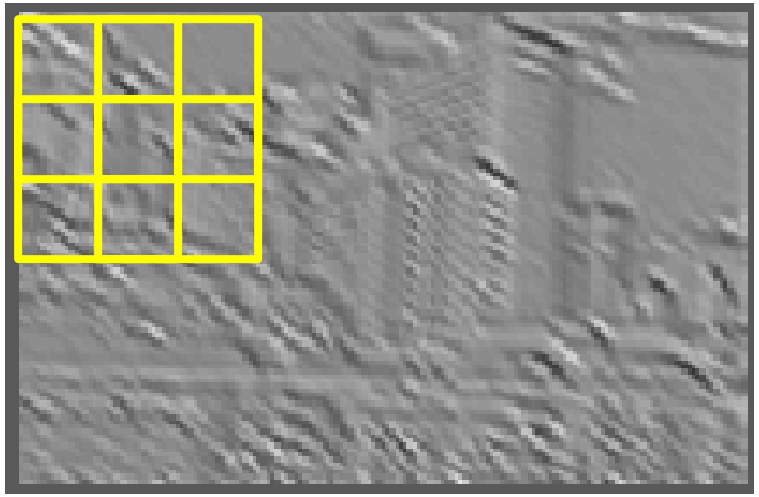


Source: R. Fergus, Y. LeCun

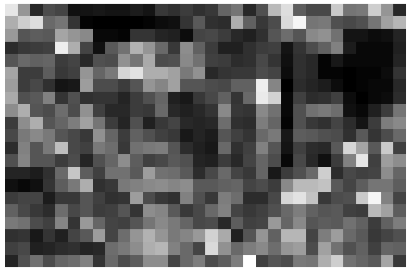
Slide: Lazebnik

Key operations





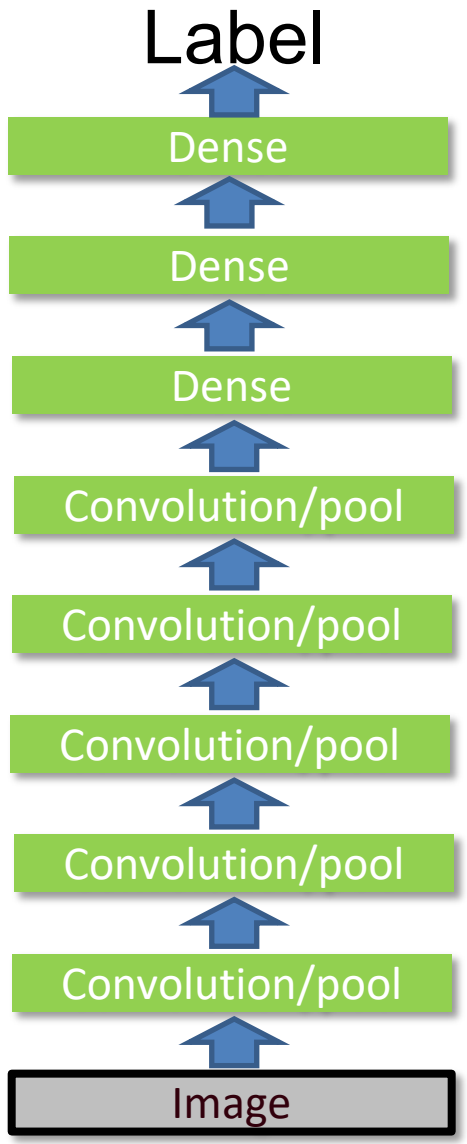
Max



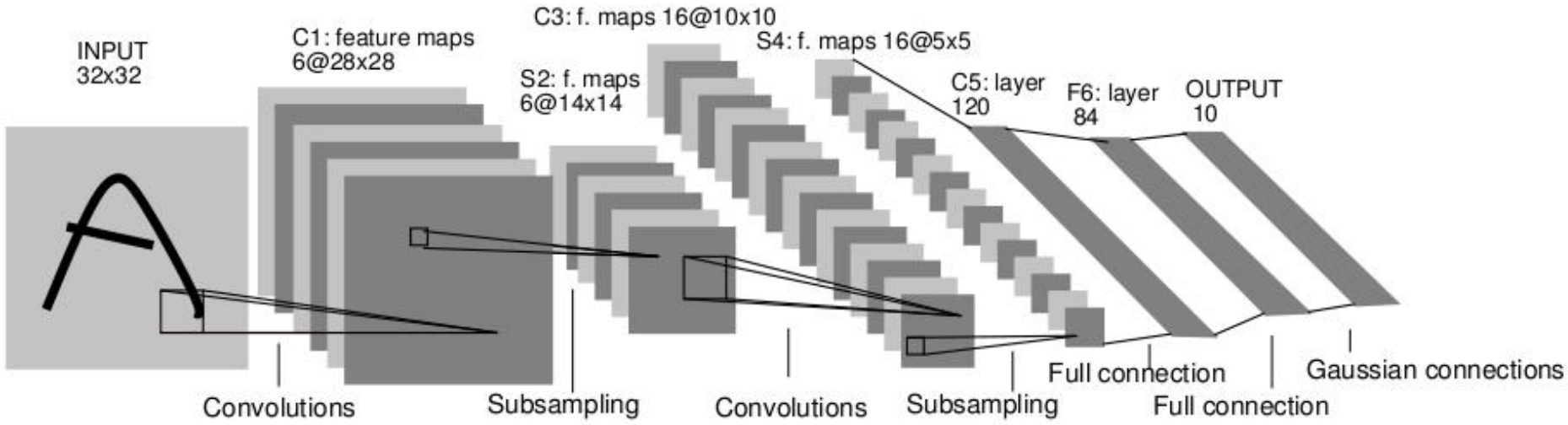
Source: R. Fergus, Y. LeCun

Slide: Lazebnik

Key idea: learn features and classifier that work well together (“end-to-end training”)



LeNet-5 for character/digit recognition



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

Fast forward to the arrival of big visual data…

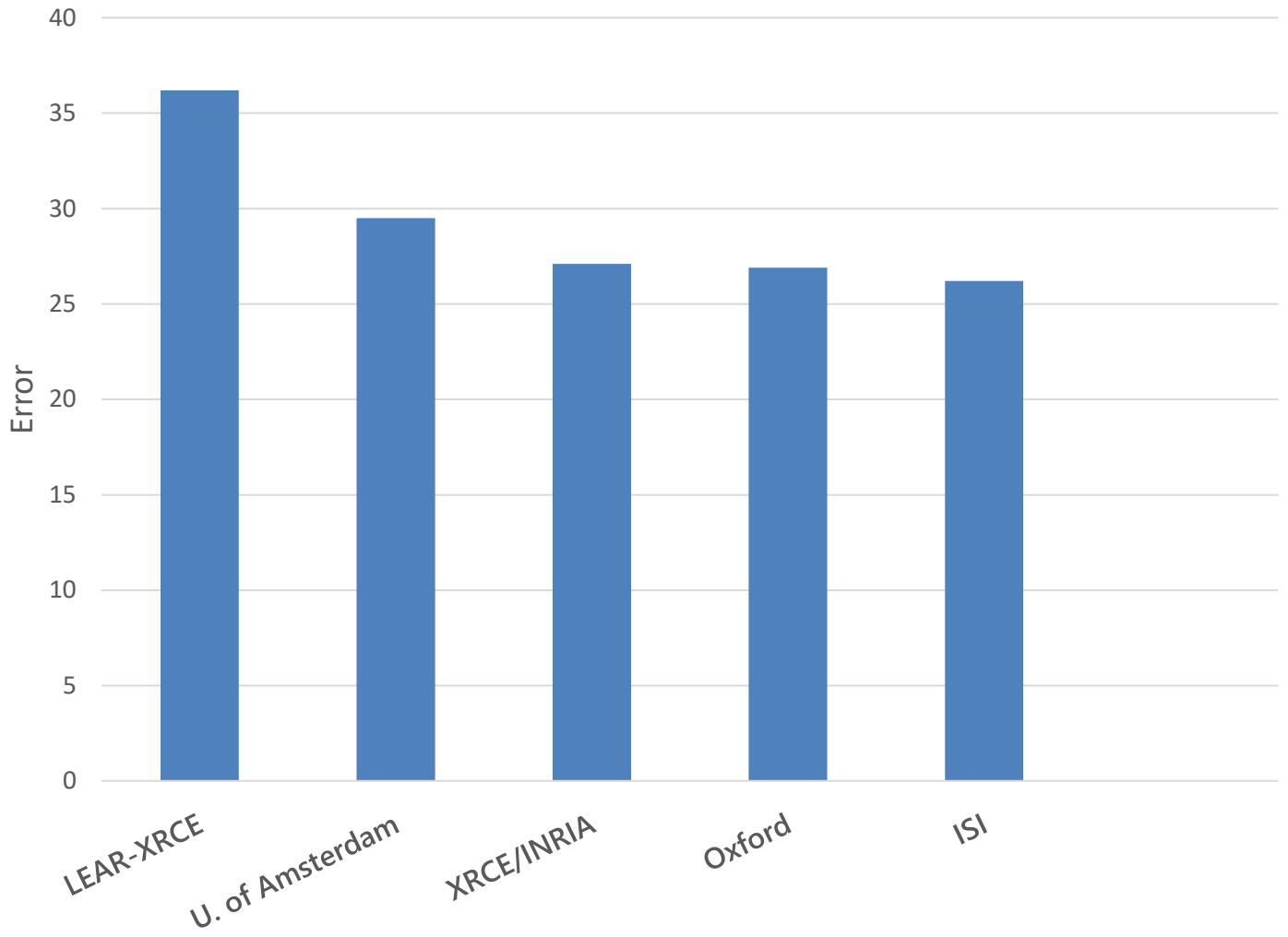
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes

www.image-net.org/challenges/LSVRC/

Slide: Lazebnik

2012 ImageNet 1K

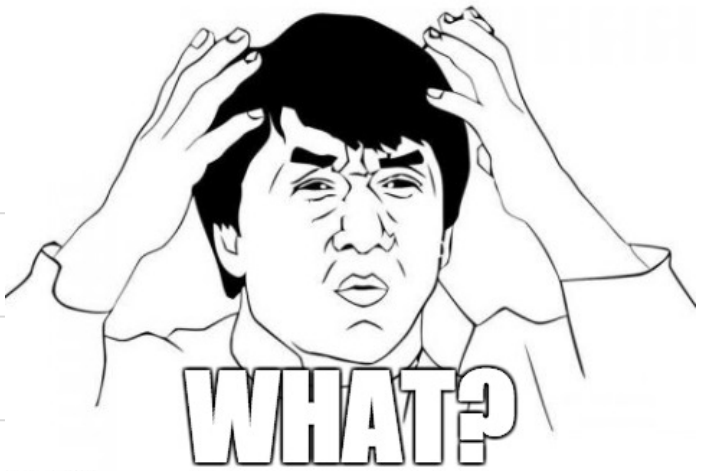
(Fall 2012)

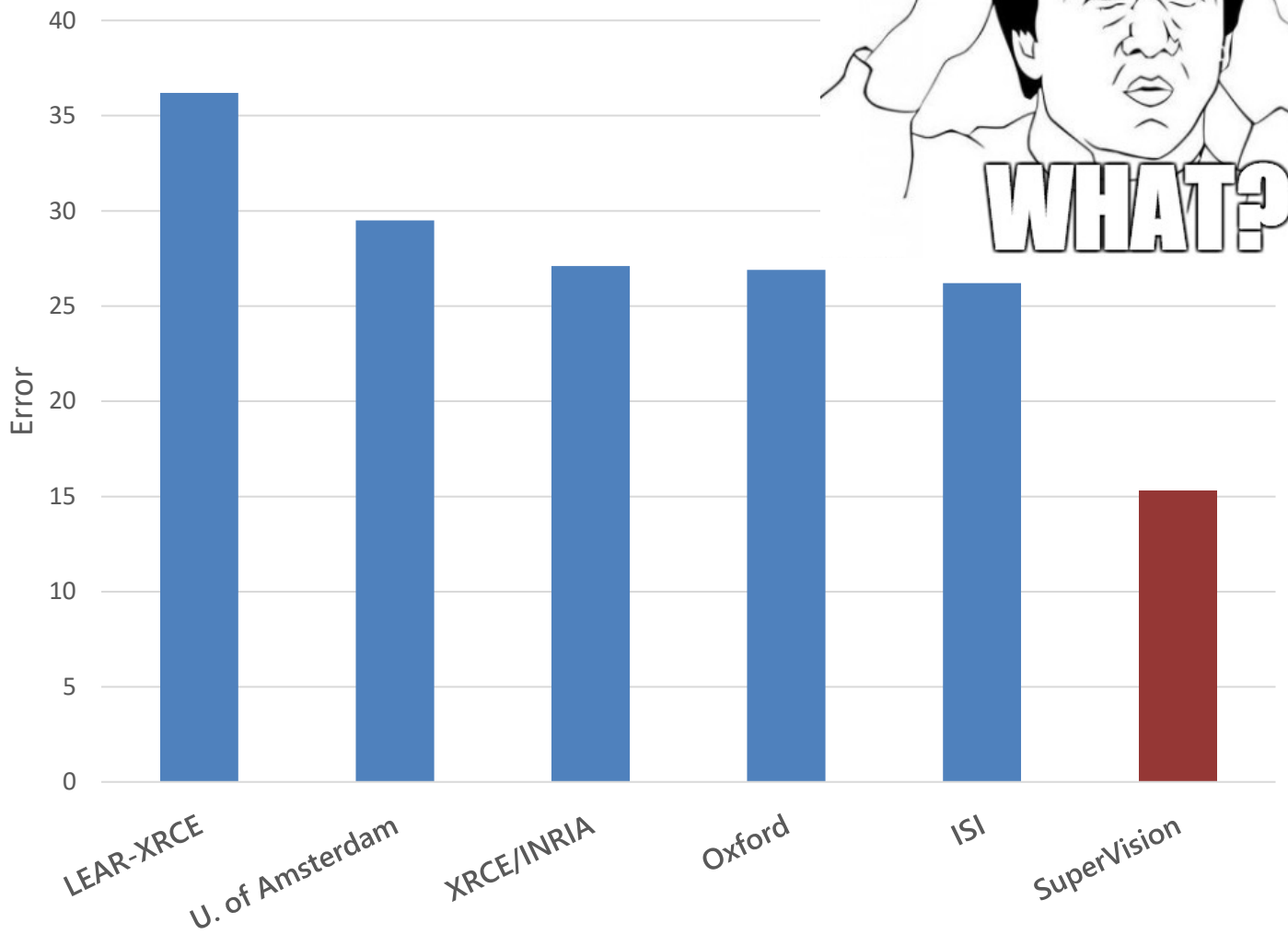


Slide: Jia-bin Huang

2012 ImageNet 1K

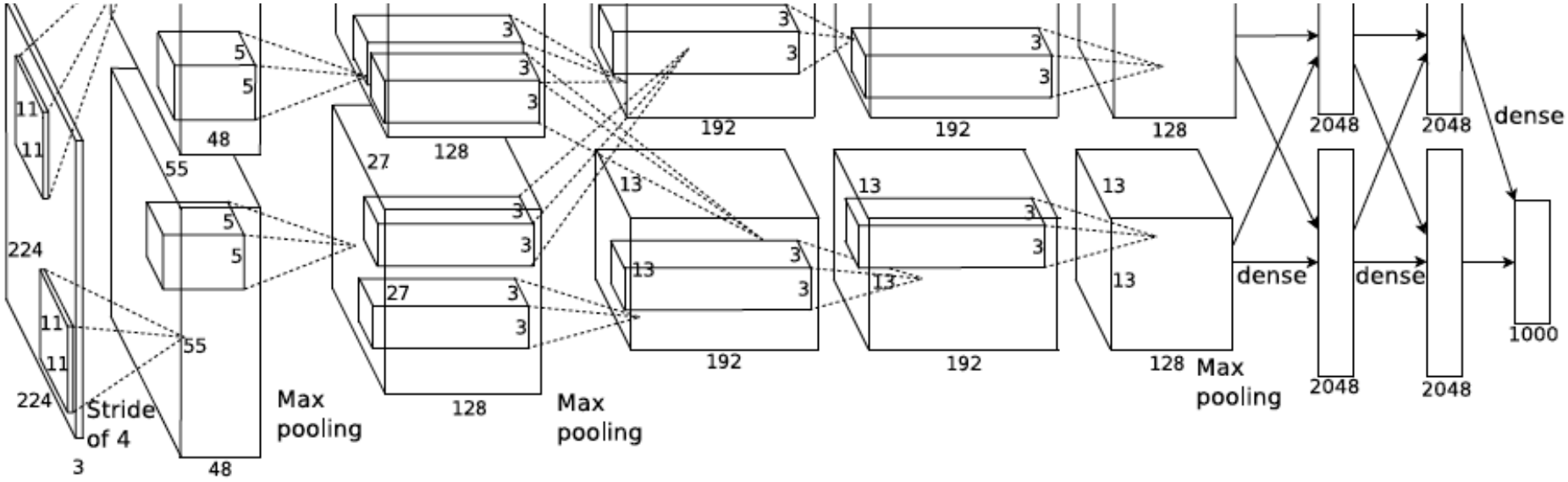
(Fall 2012)





Slide: Jia-bin Huang

AlexNet: ILSVRC 2012 winner



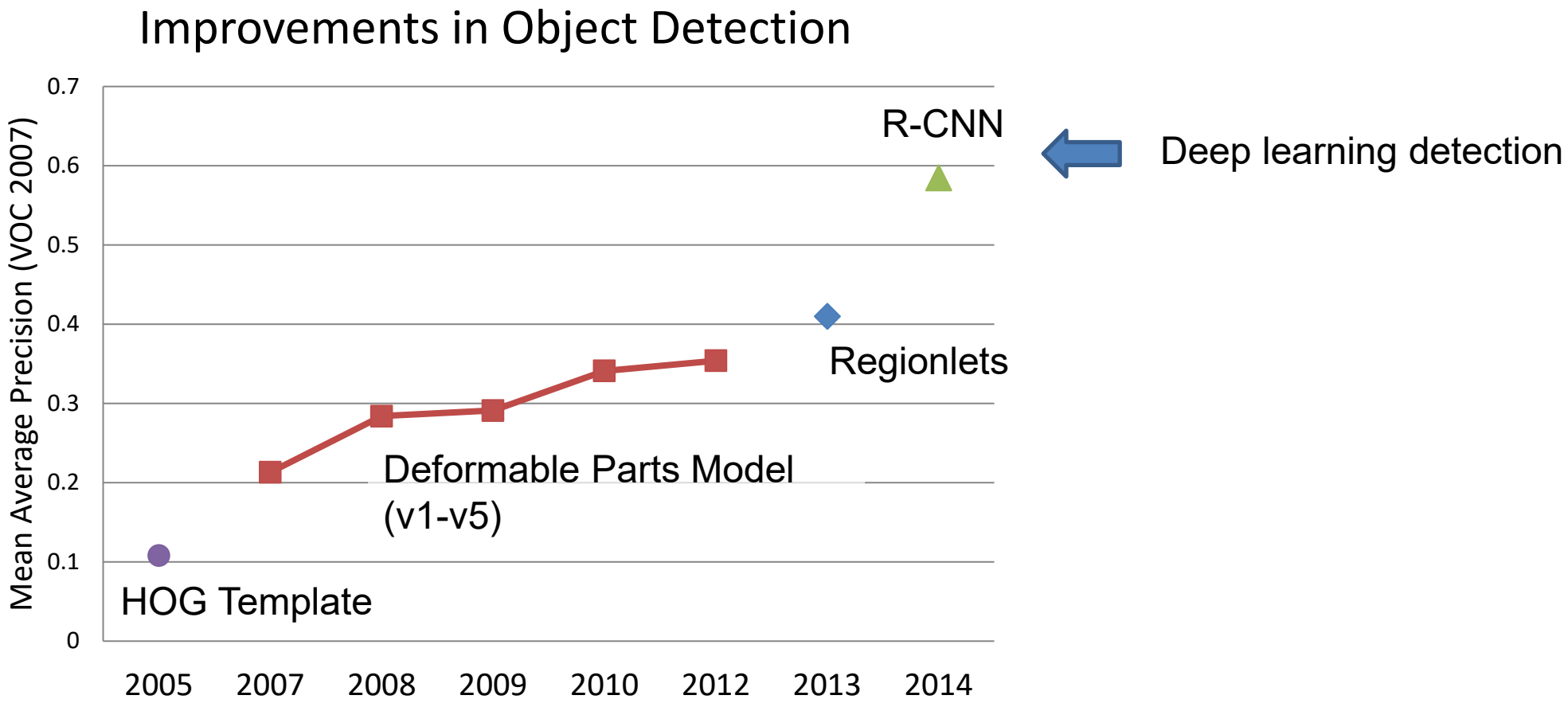
- Similar framework to LeNet but:
  - Max pooling, **ReLU nonlinearity**
  - **More data** and **bigger model** (7 hidden layers, 650K units, 60M params)
  - GPU implementation (**50x speedup** over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

What enabled the breakthrough?

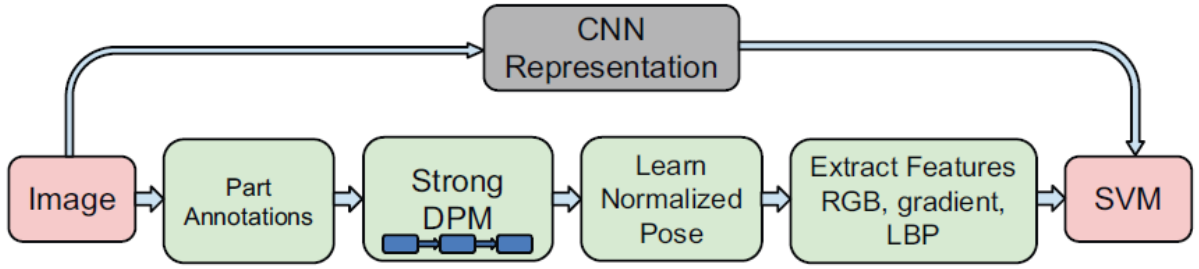
1. ReLU activation enabled large models to be optimized
2. ImageNet provided diverse and massive annotation to take advantage of the models
3. GPU processing made the optimization practicable

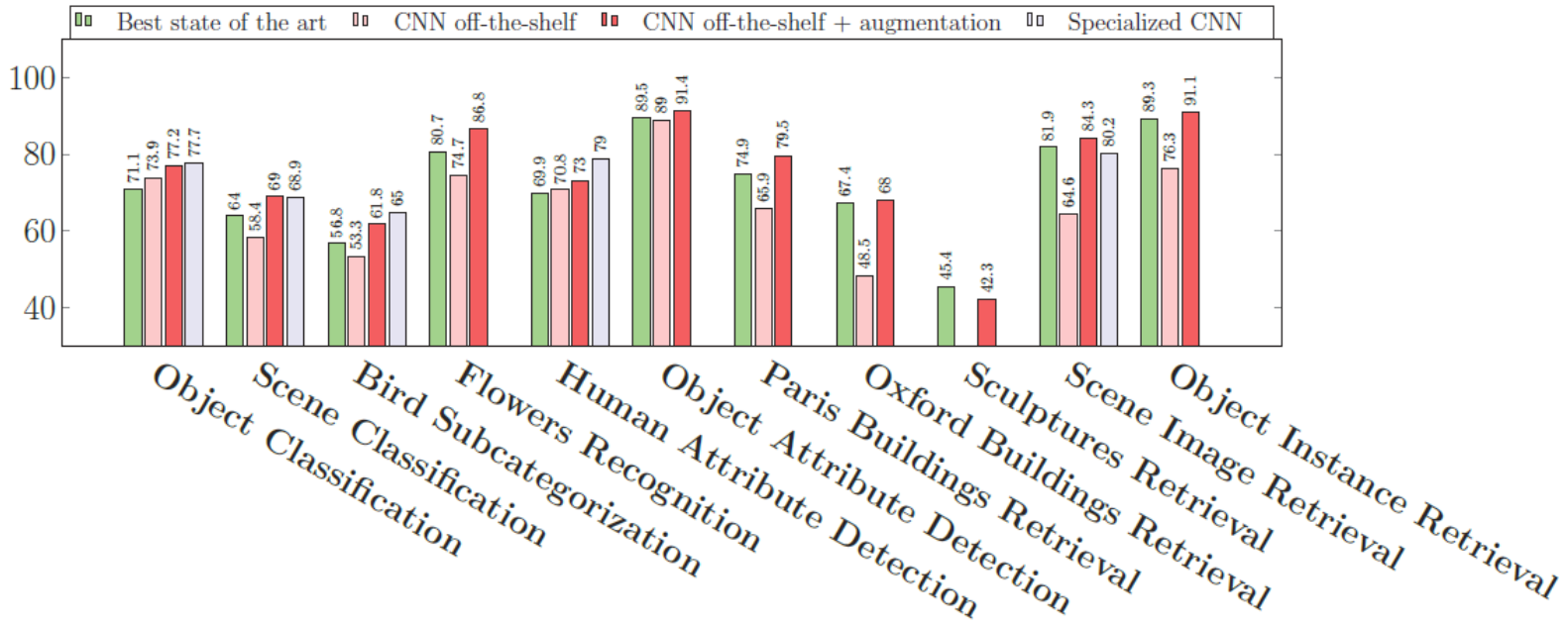
R-CNN demonstrates major detection improvement by pre-training on ImageNet and fine-tuning on PASCAL



HOG: Dalal-Triggs 2005 DPM: Felzenszwalb et al. 2008-2012 Regionlets: Wang et al. 2013 R-CNN: Girshick et al. 2014

“CNN Features off-the-shelf: an Astounding Baseline for Recognition”





Razavian et al. CVPR 2014

2 Minute Stretch Break

How it felt to be an object recognition researcher

https://youtu.be/XCtuZ-fDL2E?t=140

Recall
Q1: Why were neural networks less effective than other approaches for many years?
Q2: What were the three most important ingredients to the breakthrough?

A1:
- Deep networks couldn't be optimized
- Needed bigger datasets
  - Datasets were created to be sufficiently big for existing algorithms

A2:
- ReLU activation (easier optimization)
- Big dataset
- GPU processing

Optimization: check out this great site

Great site by Lili Jiang

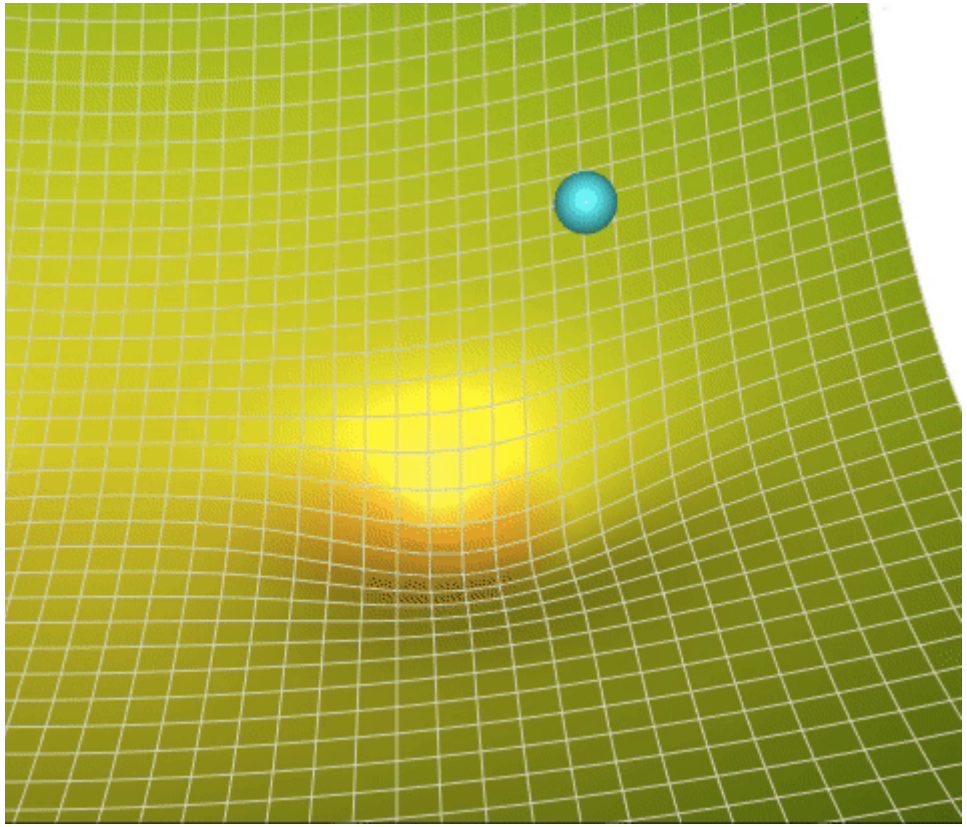
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Gradient of loss wrt weights

Basic SGD:

$$\Delta w_t = -\eta g_t(w_t)$$

$$w_{t+1} = w_t + \Delta w_t$$



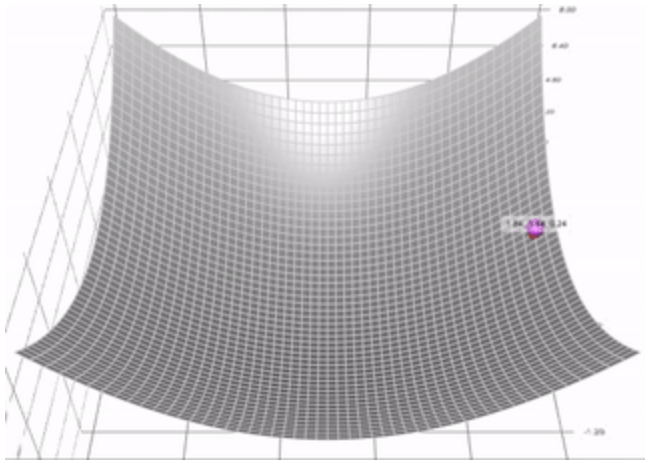
SGD + Momentum

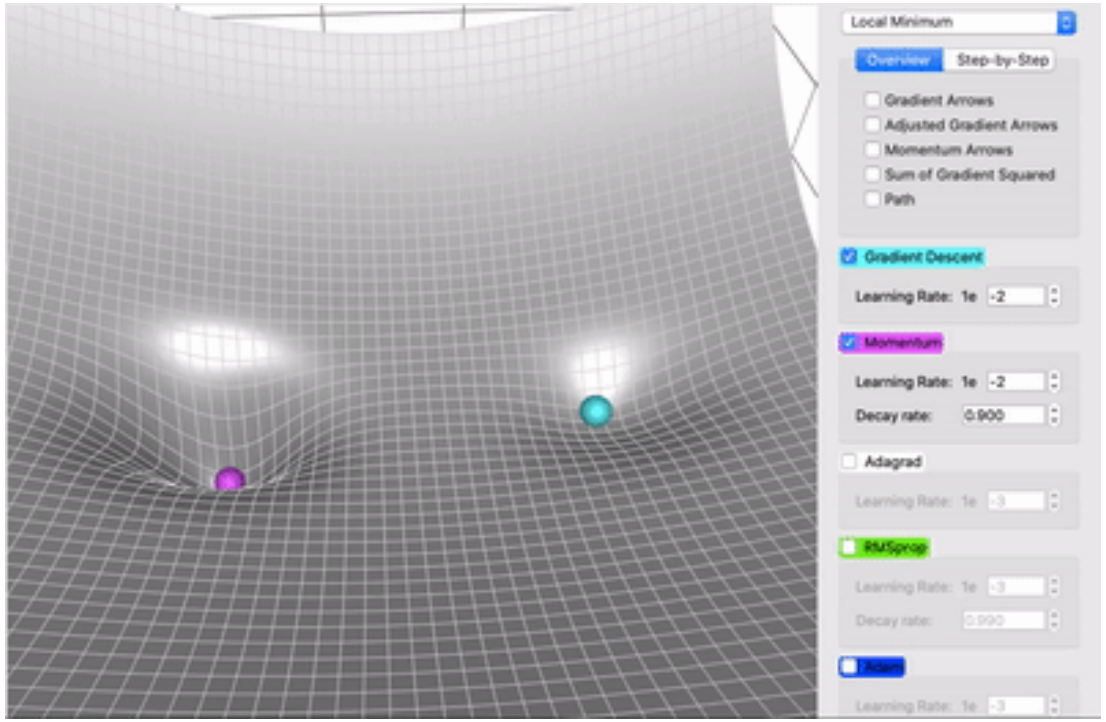
SGD + Momentum:

$$m(w_t) = \beta \cdot m(w_t) + g_t(w_t)$$

$$\Delta w_t = -\eta \cdot m(w_t)$$

$$w_{t+1} = w_t + \Delta w_t$$





Momentum (magenta) converges faster and carries the ball through a local minimum

Figure source

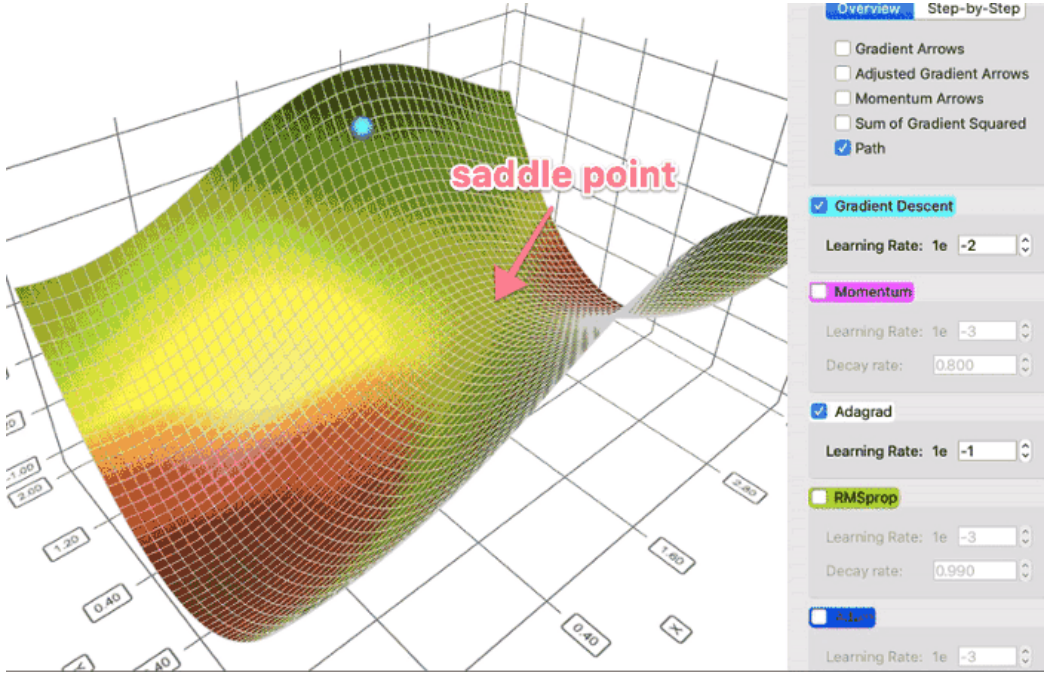
AdaGrad: Adaptive Gradient

AdaGrad:

$g_{mag}(w_t) = g_{mag}(w_{t-1}) + g_t(w_t)^2$

$\Delta w_t = -\eta g_t(w_t) / \sqrt{g_{mag}(w_t)}$ (normalize by path length of all previous updates)

$w_{t+1} = w_t + \Delta w_t$



AdaGrad (white) avoids moving in only one weight direction, and can lead to smoother convergence

Figure source

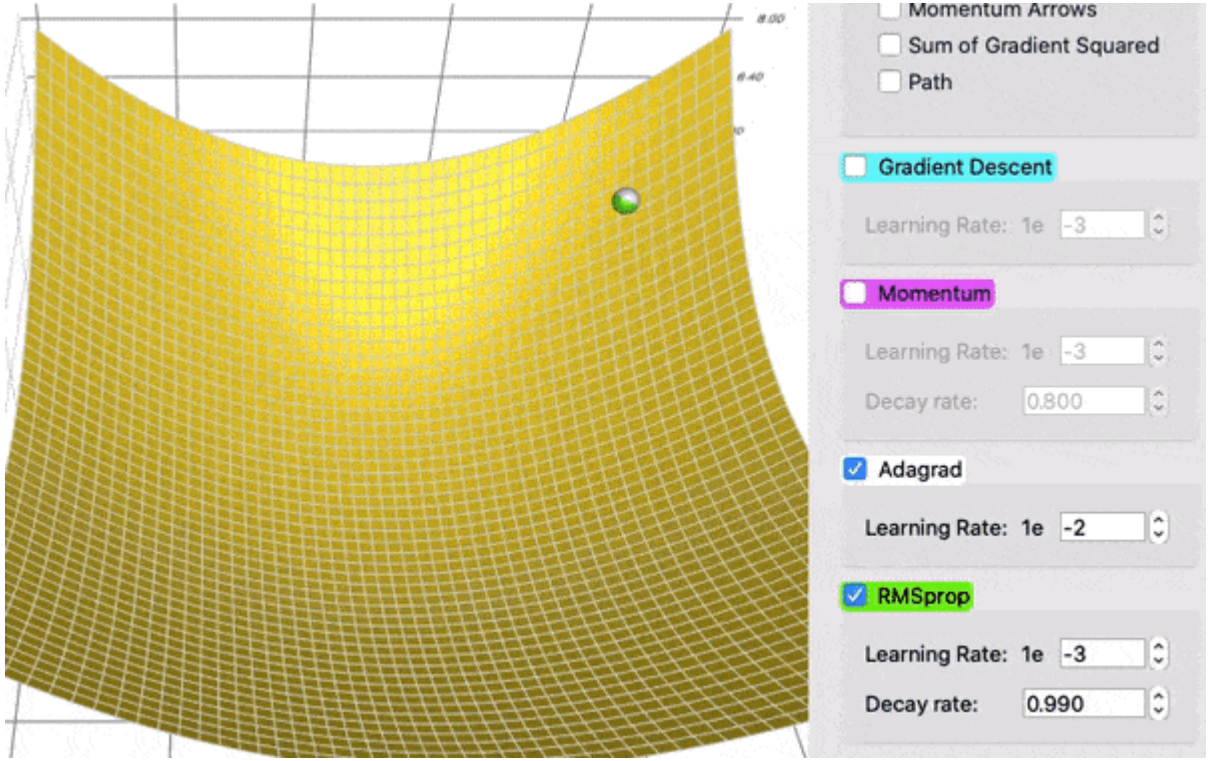
RMSProp: Root Mean Squared Propagation

RMSProp:

$g_{mag}(w_t) = \epsilon \cdot g_{mag}(w_{t-1}) + (1-\epsilon) \cdot g_t(w_t)^2$ (introducing decay rate turns this into moving avg)

$\Delta w_t = -\eta g_t(w_t) / \sqrt{g_{mag}(w_t)}$ (normalize by path length of all previous updates)

$w_{t+1} = w_t + \Delta w_t$



RMSProp (green) moves faster than AdaGrad (white)

Figure source

Adam: Adaptive Moment Estimation

Adam:

$m(w_t) = \beta \cdot m(w_t) + (1 - \beta) \cdot g_t(w_t)$ [momentum, $\beta = 0.9$]

$g_{mag}(w_t) = \epsilon \cdot g_{mag}(w_{t-1}) + (1 - \epsilon) \cdot g_t(w_t)^2$ [RMSProp, $\epsilon = 0.999$]

$\Delta w_t = -\eta \cdot m(w_t) / \sqrt{g_{mag}(w_t)}$

$w_{t+1} = w_t + \Delta w_t$

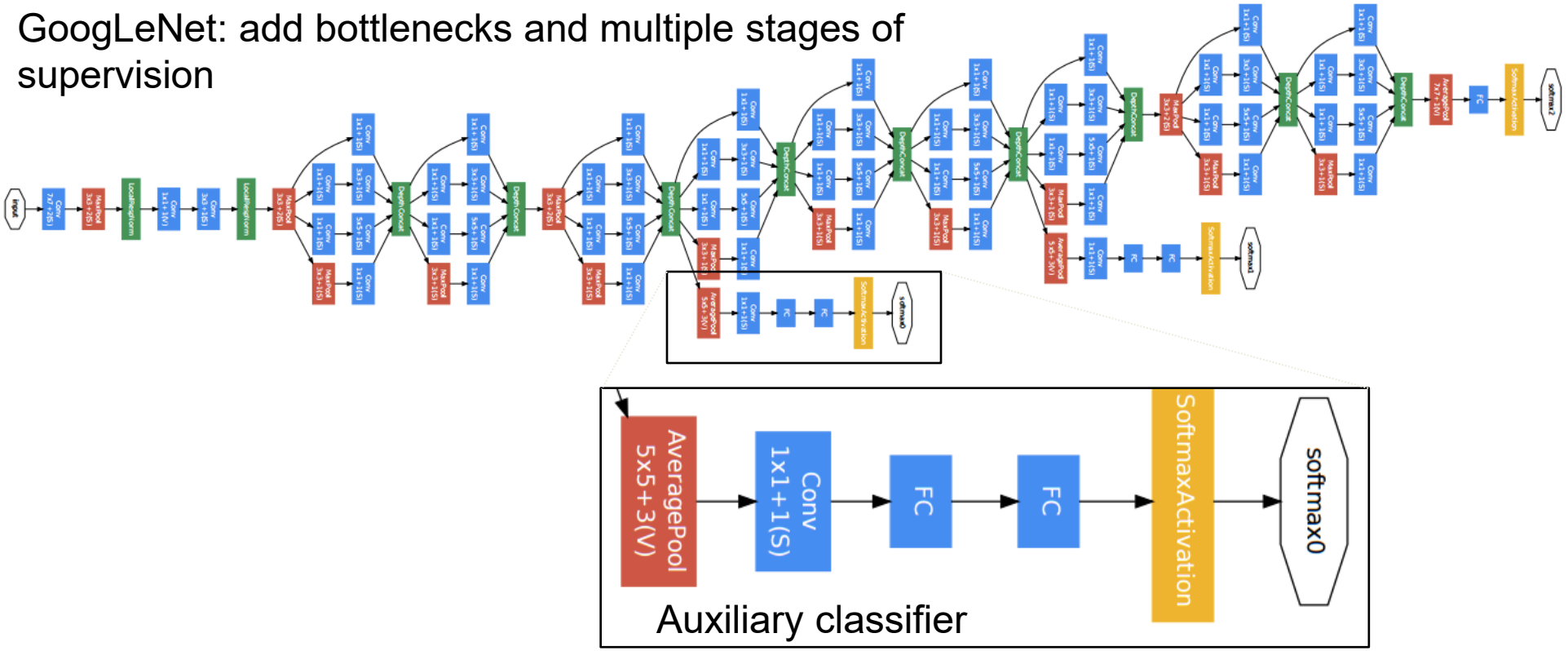
Videos

Adam is widely used and easier to tune than SGD + momentum

Figure source

Even with ReLU and Adam optimization, it was hard to get very deep networks to work well

GoogLeNet: add bottlenecks and multiple stages of supervision

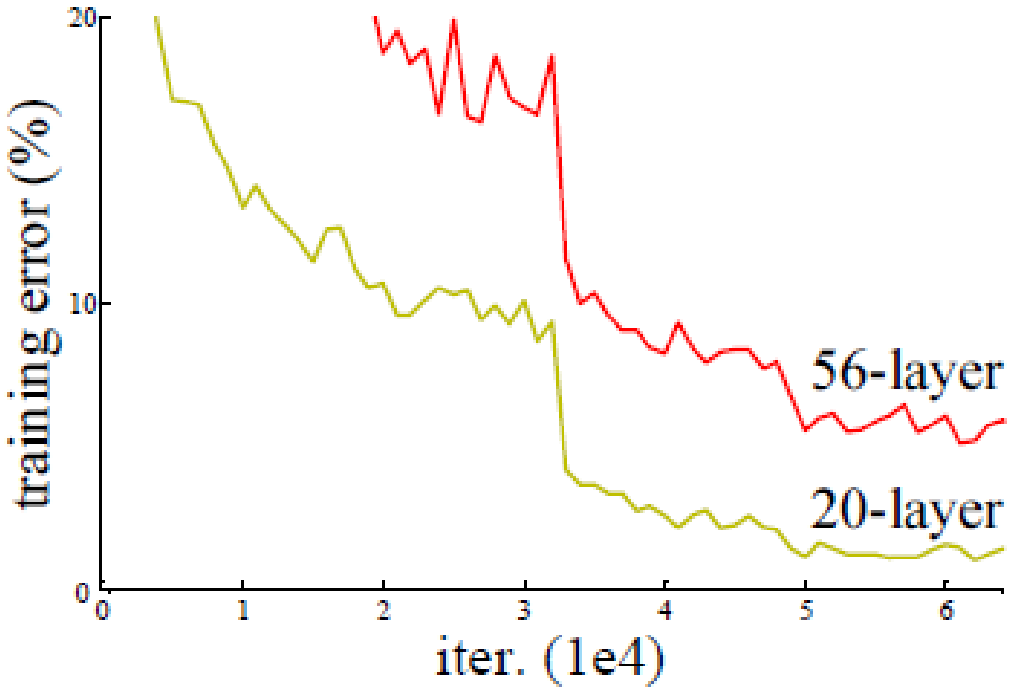


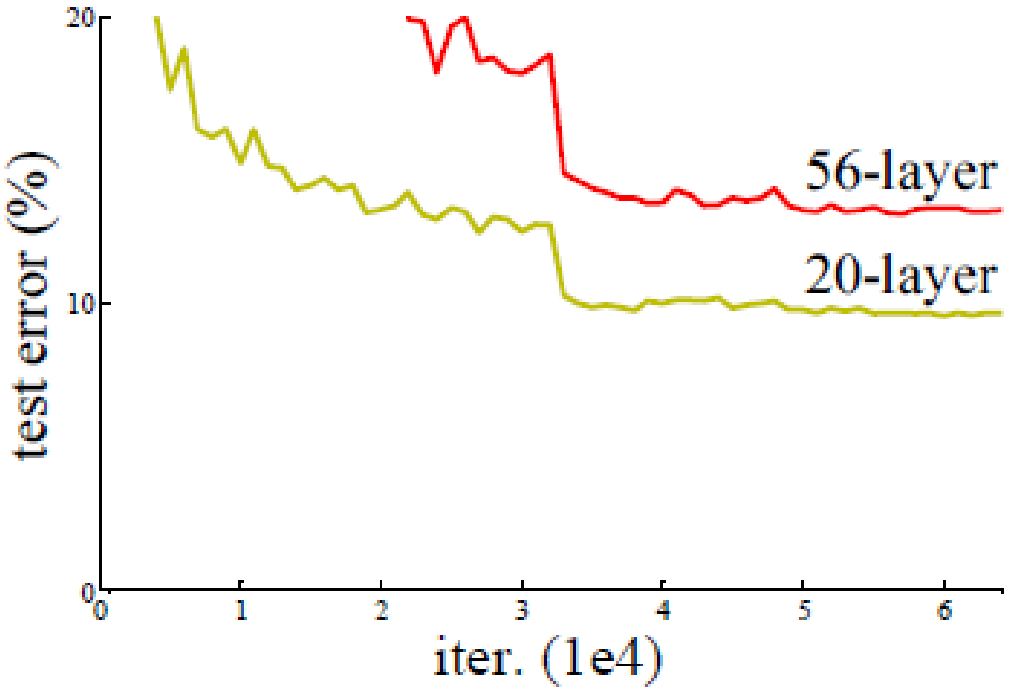
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

What was the problem?

- Were deeper networks overfitting the training data?
- Or was the problem just that we couldn’t optimize them?
- How could we answer this question?

Look at the training error!





With deeper networks, the training error goes up!?!

Fig: He et al. 2016

Very deep networks, vanishing gradients, and information propagation

Vanishing gradients

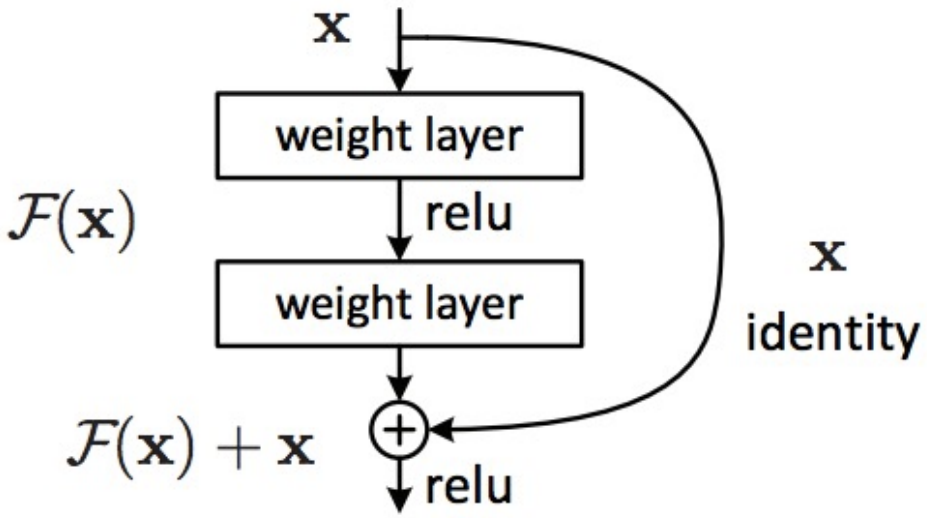
- Early weights have a long path to reach output
- Any zeros along that path kill the gradient
- Early layers cannot be optimized
- Multiple stages of supervision can help, but it's complicated and time-consuming

Information propagation

- Without residual connections, networks need to continually maintain and add to information represented in previous layers

ResNet: the residual module

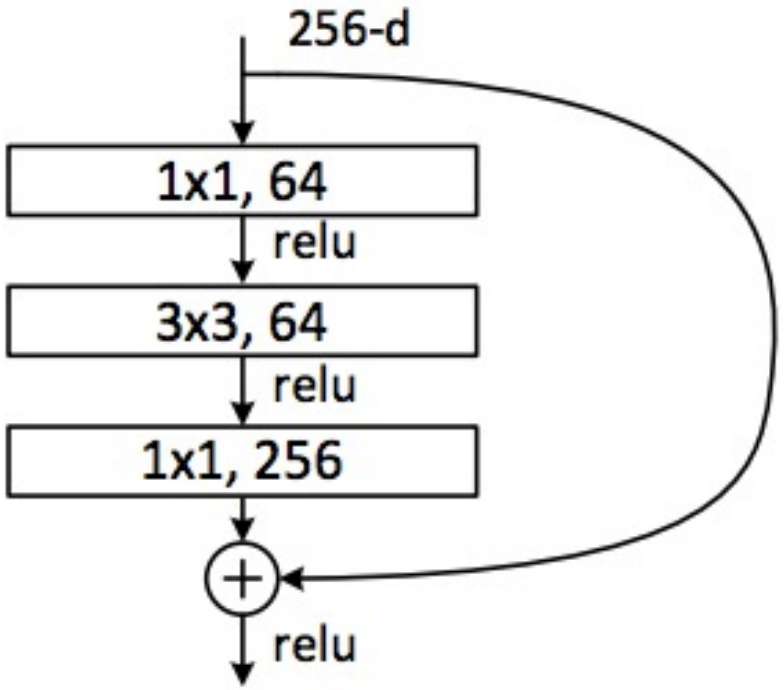
- Use *skip* or *shortcut* connections around 2-3 layer MLPs
- Gradients can flow quickly back through skip connections
- Each module needs only add information to the previous layers



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper), 150K+ citations

ResNet: Residual Bottleneck Module

Used in 50+ layer networks

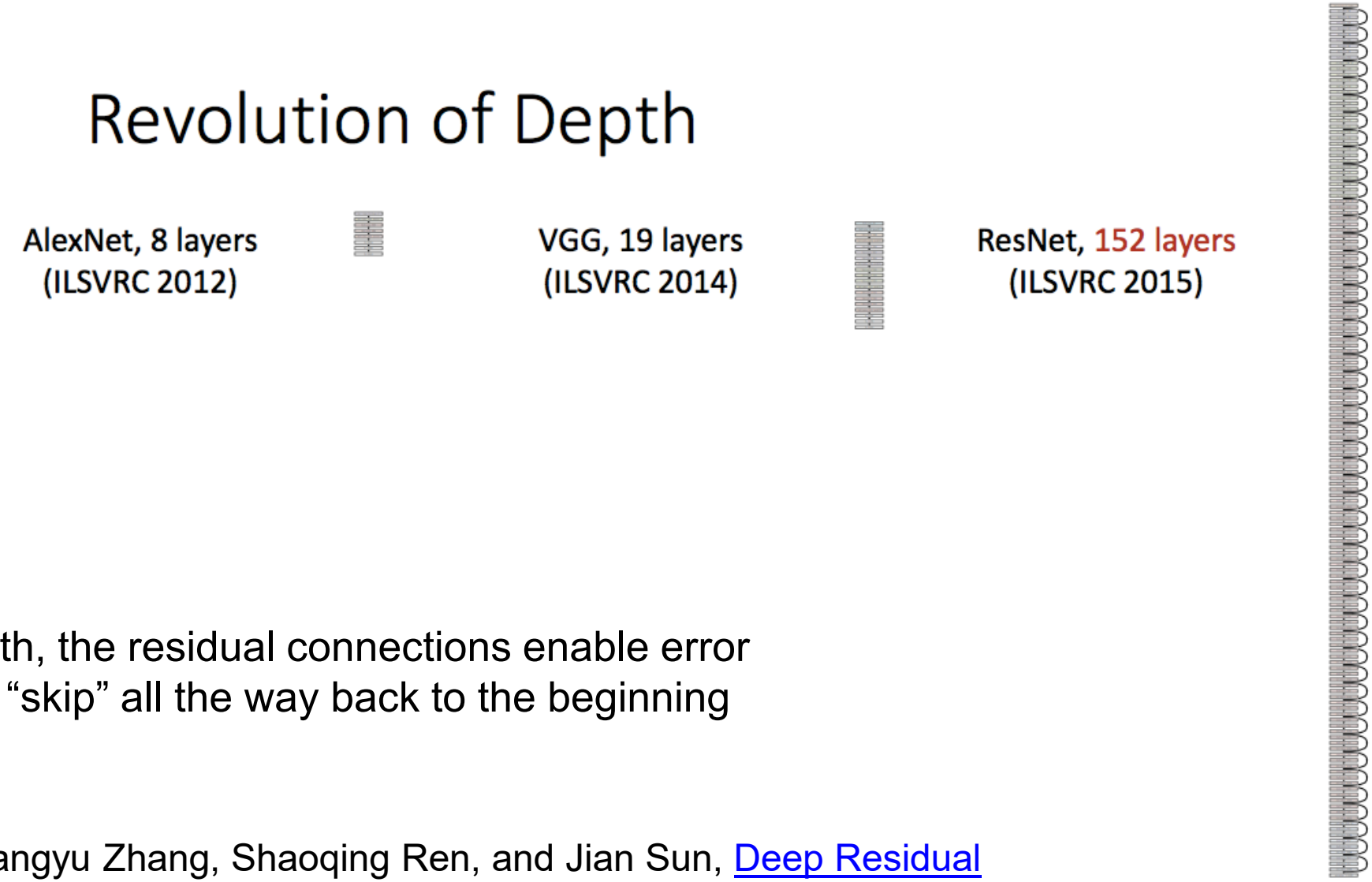


- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Slide: Lazebnik

ResNet: going real deep



Despite depth, the residual connections enable error gradients to “skip” all the way back to the beginning

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

Example code: ResBlock

```
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample):
        super().__init__()
        if downsample:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1)
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=2),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.shortcut = nn.Sequential()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, input):
        shortcut = self.shortcut(input)
        input = nn.ReLU()(self.bn1(self.conv1(input)))
        input = nn.ReLU()(self.bn2(self.conv2(input)))
        input = input + shortcut
        return nn.ReLU()(input)
```

Example code: ResNet-18 architecture for ImageNet

```
class Network(nn.Module):
    def __init__(self, num_classes=37):
        super().__init__()
        resblock = ResBlock
        self.layer0 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.layer1 = nn.Sequential(
            resblock(64, 64, downsample=False),
            resblock(64, 64, downsample=False)
        )
        self.layer2 = nn.Sequential(
            resblock(64, 128, downsample=True),
            resblock(128, 128, downsample=False)
        )
        self.layer3 = nn.Sequential(
            resblock(128, 256, downsample=True),
            resblock(256, 256, downsample=False)
        )
        self.layer4 = nn.Sequential(
            resblock(256, 512, downsample=True),
            resblock(512, 512, downsample=False)
        )
        self.gap = torch.nn.AdaptiveAvgPool2d(1)
        self.fc = torch.nn.Linear(512, num_classes)
```

```
    def forward(self, input):
        input = self.layer0(input)
        input = self.layer1(input)
        input = self.layer2(input)
        input = self.layer3(input)
        input = self.layer4(input)
        input = self.gap(input)
        input = torch.flatten(input, 1)
        input = self.fc(input)

        return input
```

ResNet Architectures and Results

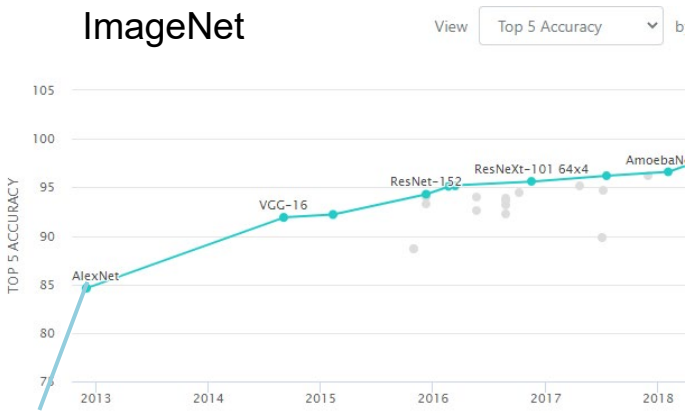
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

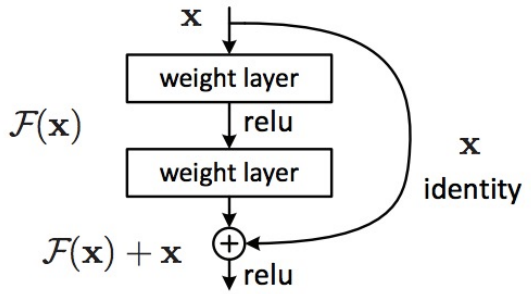
method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

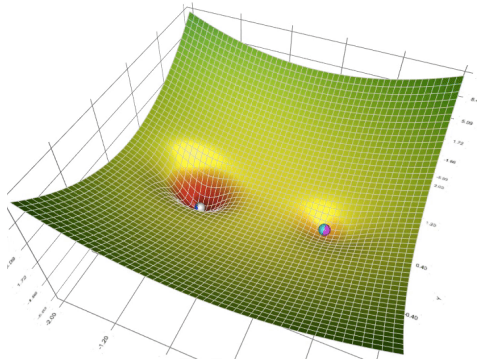
Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

What to remember

- Deep networks provide huge gains in performance
  - Large capacity, optimizable models
  - Learn from new large datasets
- ReLU and skip connections simplify optimization
- SGD is still often used in practice, but Adam is most widely used







Next consolidation and review

- How to represent and measure data
- Q & A