10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

CNNs

+

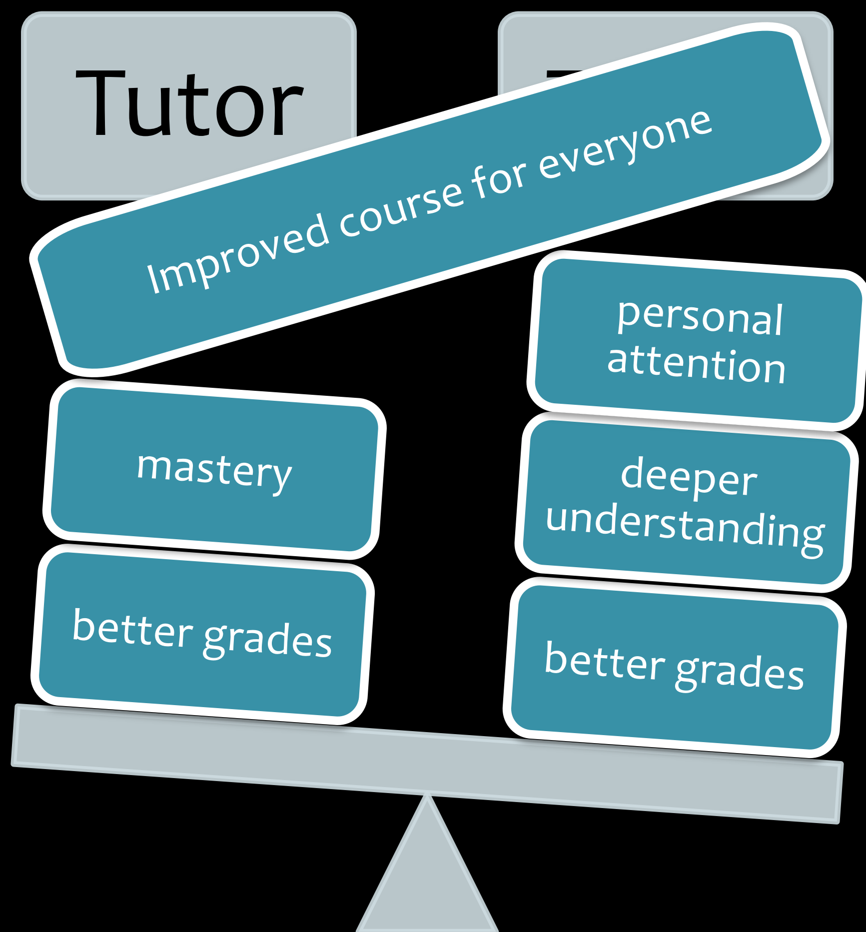
PAC Learning

Matt Gormley
Lecture 14
Mar. 14, 2022

Reminders

- Homework 5: Neural Networks
  - Out: Sun, Feb 27
  - Due: Fri, Mar 18 at 11:59pm

Peer Tutoring



Dynamic Programming

Question:

Have you studied dynamic programming in a previous course?

A. Yes

B. No

Answer:

Question:

What is the difference between memoization and dynamic programming, when applied to a recursive function f(**x**)?

A. **memoization** computes a function recursively without storing intermediate results, whereas **dynamic programming** stores intermediate results

B. **memoization** stores function values as they are encountered top-down, whereas **dynamic programming** stores function values as they are encountered bottom-up

C. **memoization** stores only the output of a tertiary function g(**x**), whereas **dynamic programming** stores the outputs of f(**x**) directly

D. **memoization** typically increases computational complexity of an algorithm while decreasing space complexity, whereas **dynamic programming** typically decreases computational complexity and increases space complexity

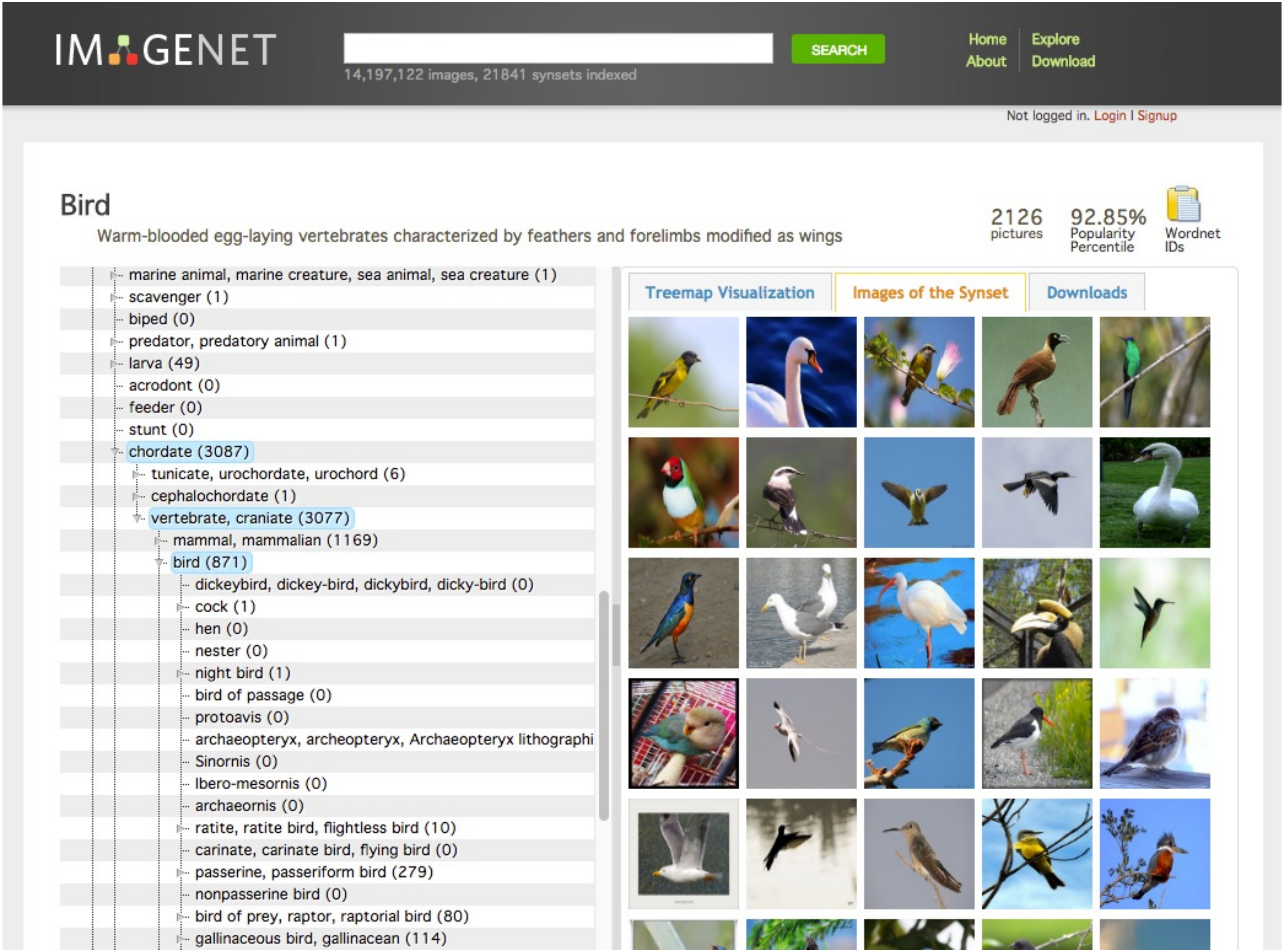
E. **memoization** memorizes a function, whereas **dynamic programming** has a programmer generate code for the function on-the-fly (i.e. I answered "Yes" to previous question)

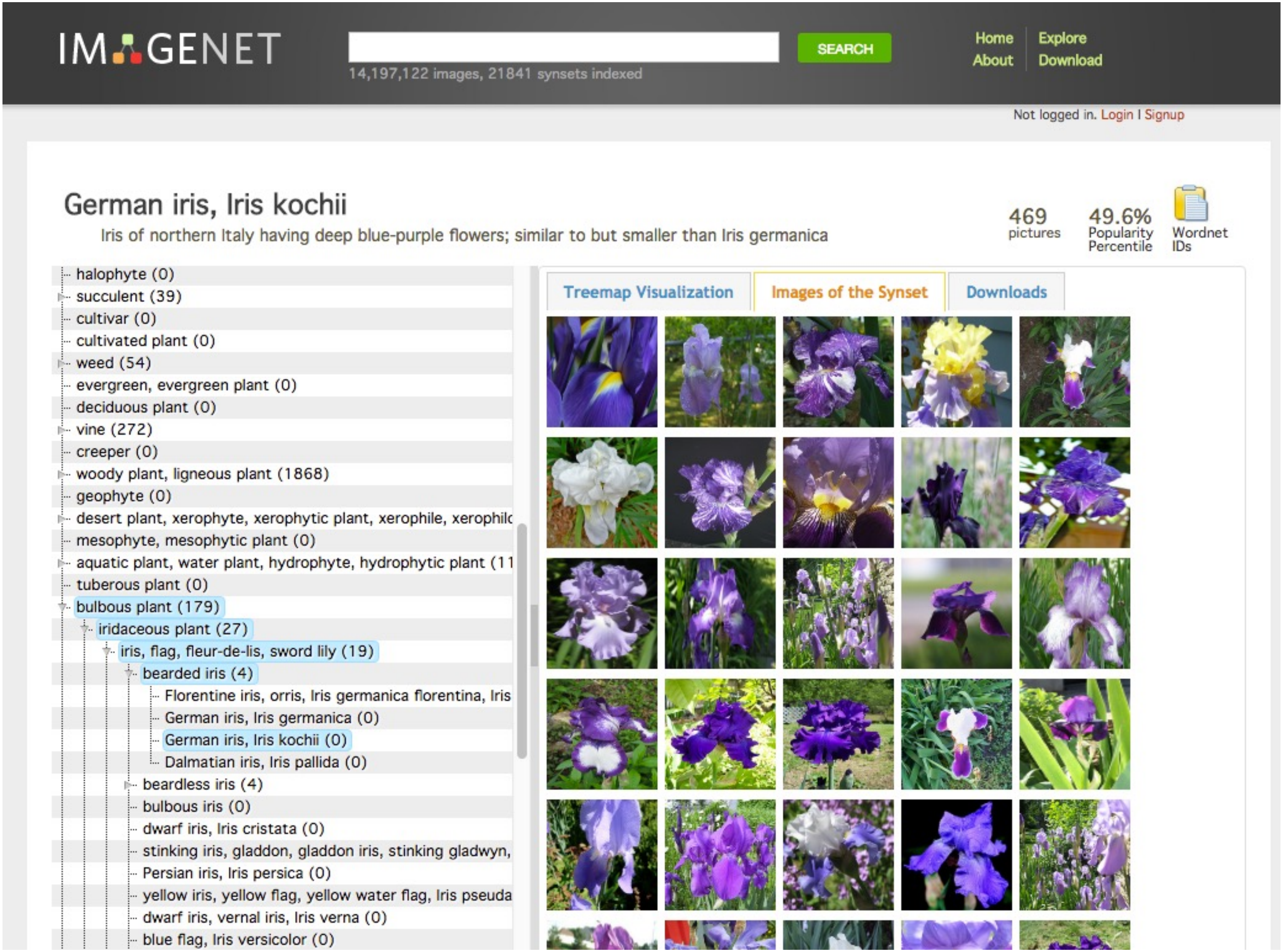
Answer:

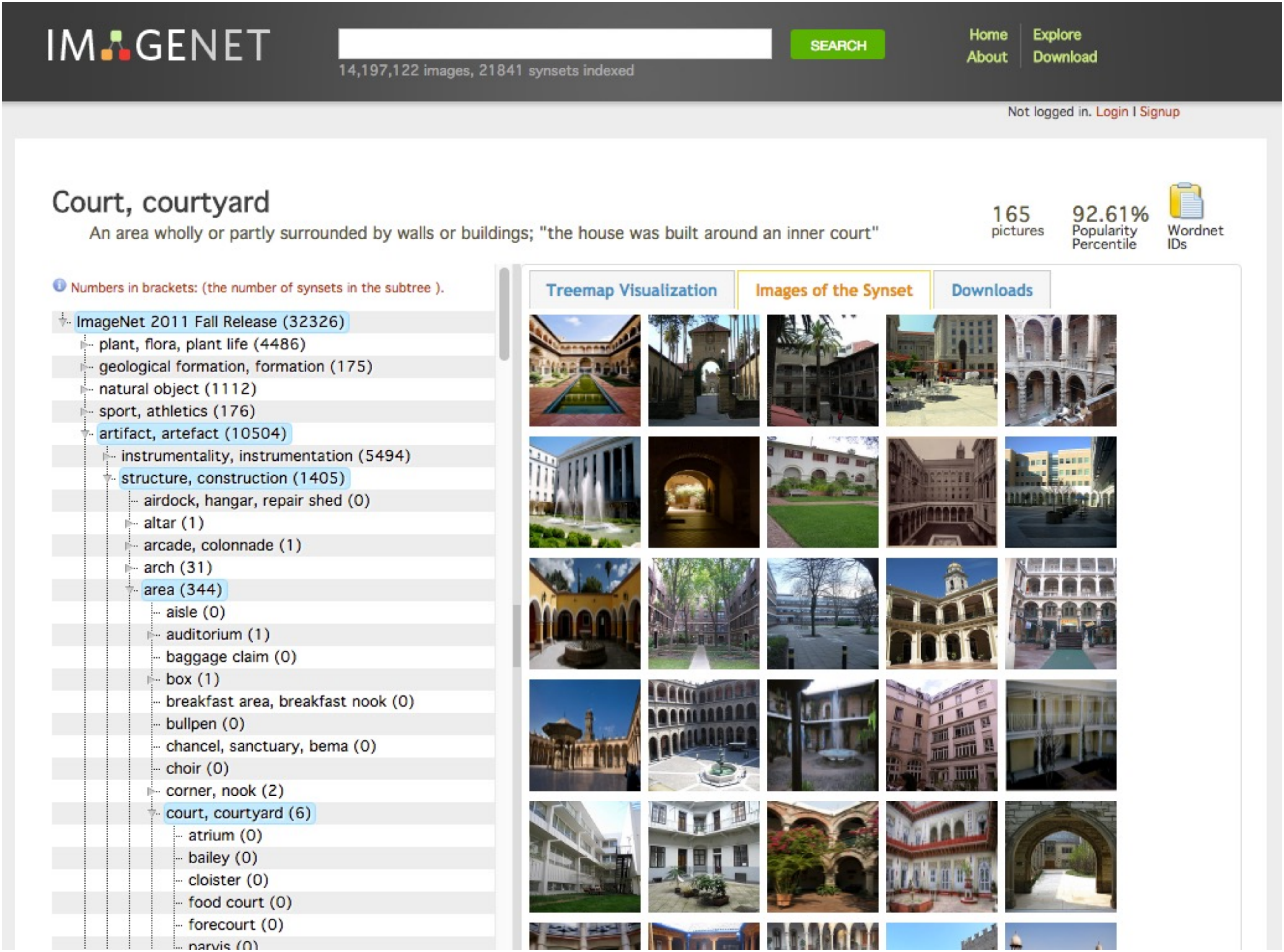
BACKGROUND: COMPUTER VISION

Example: Image Classification

- ImageNet LSVRC-2011 contest:
  - **Dataset:** 1.2 million labeled images, 1000 classes
  - **Task:** Given a new image, label it with the correct class
  - **Multiclass** classification problem
- Examples from http://image-net.org/


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings
2126 pictures
92.85% Popularity Percentile
Wordnet IDs
marine animal, marine creature, sea animal, sea creature (1)
scavenger (1)
biped (0)
predator, predatory animal (1)
larva (49)
acrodont (0)
feeder (0)
stunt (0)
chordate (3087)
tunicate, urochordate, urochord (6)
cephalochordate (1)
vertebrate, craniate (3077)
mammal, mammalian (1169)
bird (871)
dickeybird, dickey-bird, dickybird, dicky-bird (0)
cock (1)
hen (0)
nester (0)
night bird (1)
bird of passage (0)
protoavis (0)
archaeopteryx, archeopteryx, Archaeopteryx lithographi
Sinornis (0)
Ibero-mesornis (0)
archaeornis (0)
ratite, ratite bird, flightless bird (10)
carinate, carinate bird, flying bird (0)
passerine, passeriform bird (279)
nonpasserine bird (0)
bird of prey, raptor, raptorial bird (80)
gallinaceous bird, gallinacean (114)
Treemap Visualization
Images of the Synset
Downloads


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica
469 pictures
49.6% Popularity Percentile
Wordnet IDs
halophyte (0)
succulent (39)
cultivar (0)
cultivated plant (0)
weed (54)
evergreen, evergreen plant (0)
deciduous plant (0)
vine (272)
creeper (0)
woody plant, ligneous plant (1868)
geophyte (0)
desert plant, xerophyte, xerophytic plant, xerophile, xerophilc
mesophyte, mesophytic plant (0)
aquatic plant, water plant, hydrophyte, hydrophytic plant (11
tuberous plant (0)
bulbous plant (179)
iridaceous plant (27)
iris, flag, fleur-de-lis, sword lily (19)
bearded iris (4)
Florentine iris, orris, Iris germanica florentina, Iris
German iris, Iris germanica (0)
German iris, Iris kochii (0)
Dalmatian iris, Iris pallida (0)
beardless iris (4)
bulbous iris (0)
dwarf iris, Iris cristata (0)
stinking iris, gladdon, gladdon iris, stinking gladwyn,
Persian iris, Iris persica (0)
yellow iris, yellow flag, yellow water flag, Iris pseuda
dwarf iris, vernal iris, Iris verna (0)
blue flag, Iris versicolor (0)
Treemap Visualization
Images of the Synset
Downloads

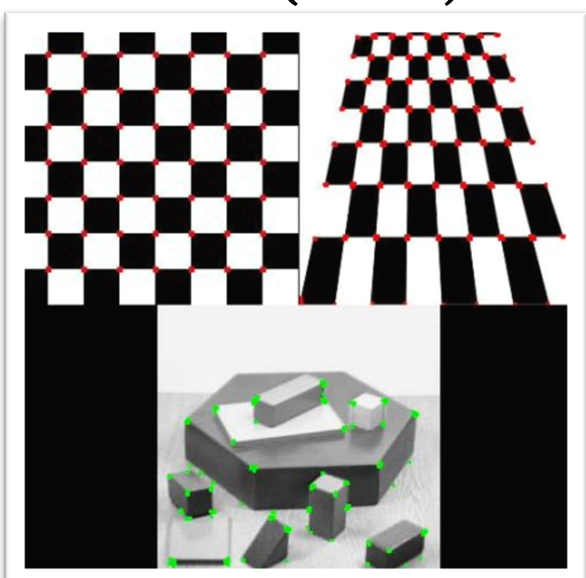


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
Court, courtyard
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"
165 pictures
92.61% Popularity Percentile
Wordnet IDs
Numbers in brackets: (the number of synsets in the subtree).
ImageNet 2011 Fall Release (32326)
plant, flora, plant life (4486)
geological formation, formation (175)
natural object (1112)
sport, athletics (176)
artifact, artefact (10504)
instrumentality, instrumentation (5494)
structure, construction (1405)
airdock, hangar, repair shed (0)
altar (1)
arcade, colonnade (1)
arch (31)
area (344)
aisle (0)
auditorium (1)
baggage claim (0)
box (1)
breakfast area, breakfast nook (0)
bullpen (0)
chancel, sanctuary, bema (0)
choir (0)
corner, nook (2)
court, courtyard (6)
atrium (0)
bailey (0)
cloister (0)
food court (0)
forecourt (0)
Treemap Visualization
Images of the Synset
Downloads

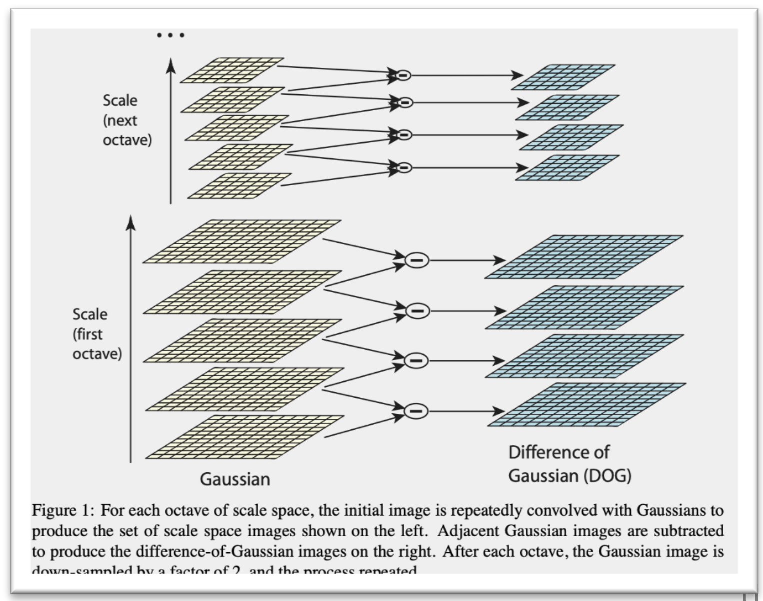
Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)



Scale Invariant Feature Transform (SIFT)



Figures from http://opencv.org

Figure from Lowe (1999) and Lowe (2004)

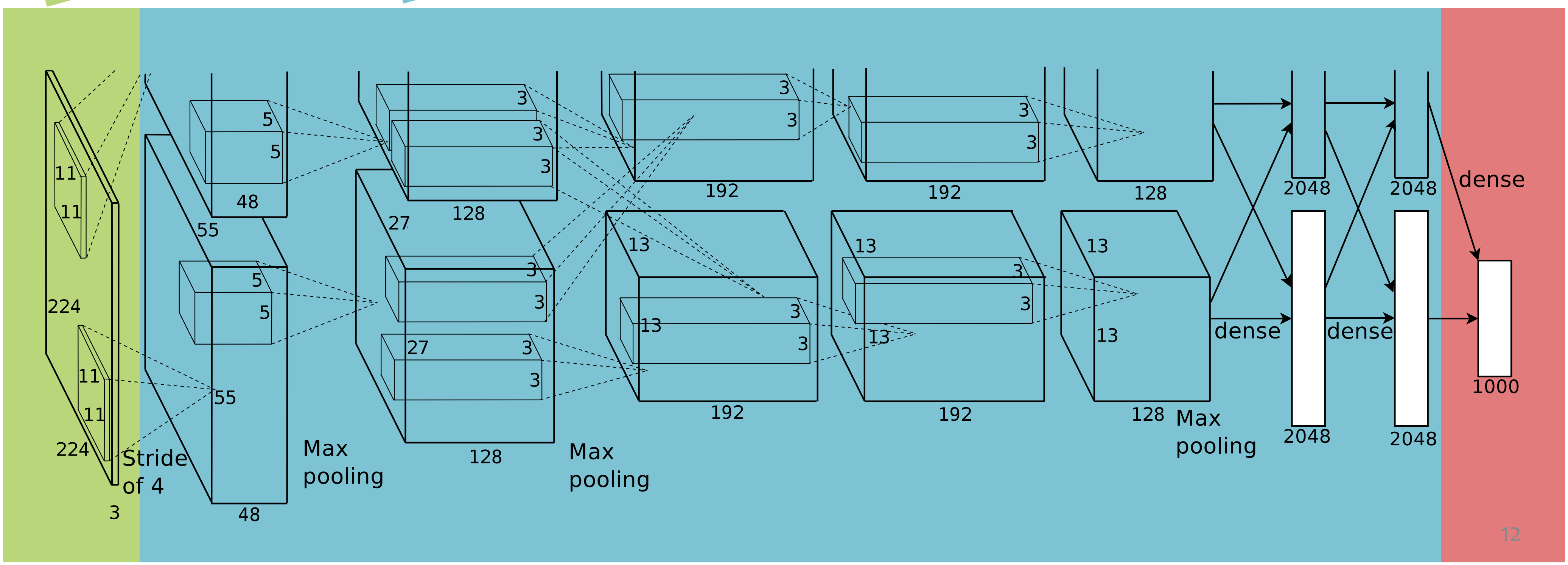
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

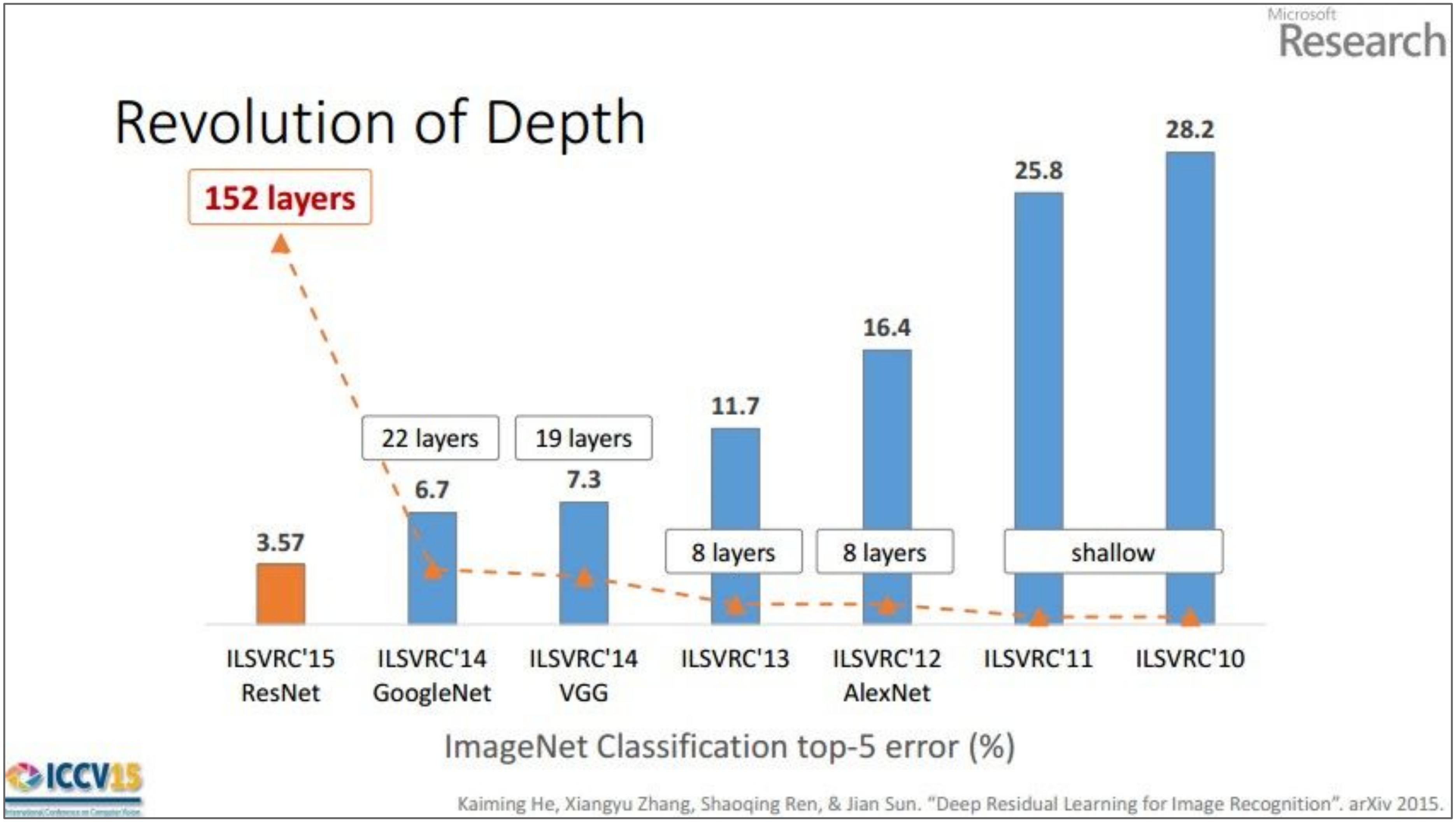
Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



CNNs for Image Recognition



Slide from Kaiming He

Backpropagation and Deep Learning

Convolutional neural networks (CNNs) and **recurrent neural networks** (RNNs) are simply fancy computation graphs (aka. hypotheses or decision functions).

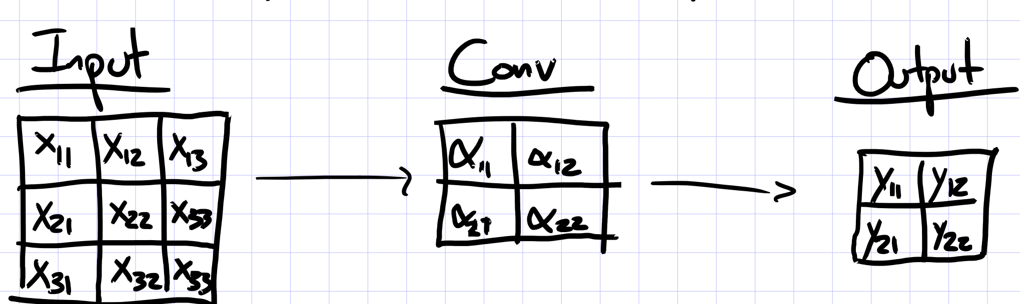
Our recipe also applies to these models and (again) relies on the **backpropagation algorithm** to compute the necessary gradients.

CONVOLUTION

What's a convolution?

- Basic idea:
  - Pick a 3x3 matrix F of weights
  - Slide this over an image and compute the "inner product" (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation
- Key point:
  - Different convolutions extract different types of low-level "features" from an image
  - All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel



$$y_{11} = \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0$$
$$y_{12} = \alpha_{11} x_{12} + \alpha_{12} x_{13} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0$$
$$y_{21} = \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0$$
$$y_{22} = \alpha_{11} x_{22} + \alpha_{12} x_{23} + \alpha_{21} x_{32} + \alpha_{22} x_{33} + \alpha_0$$

Slide adapted from William Cohen

Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

Convolved Image

1	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	0	0	0	0

Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

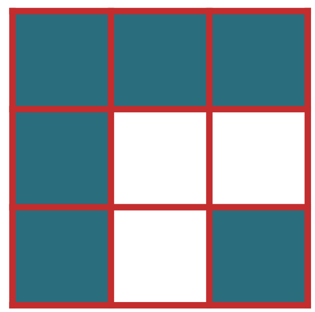
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

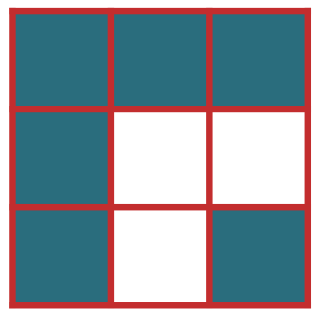
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

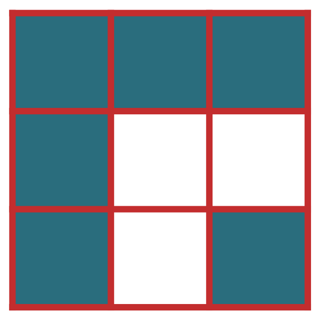
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

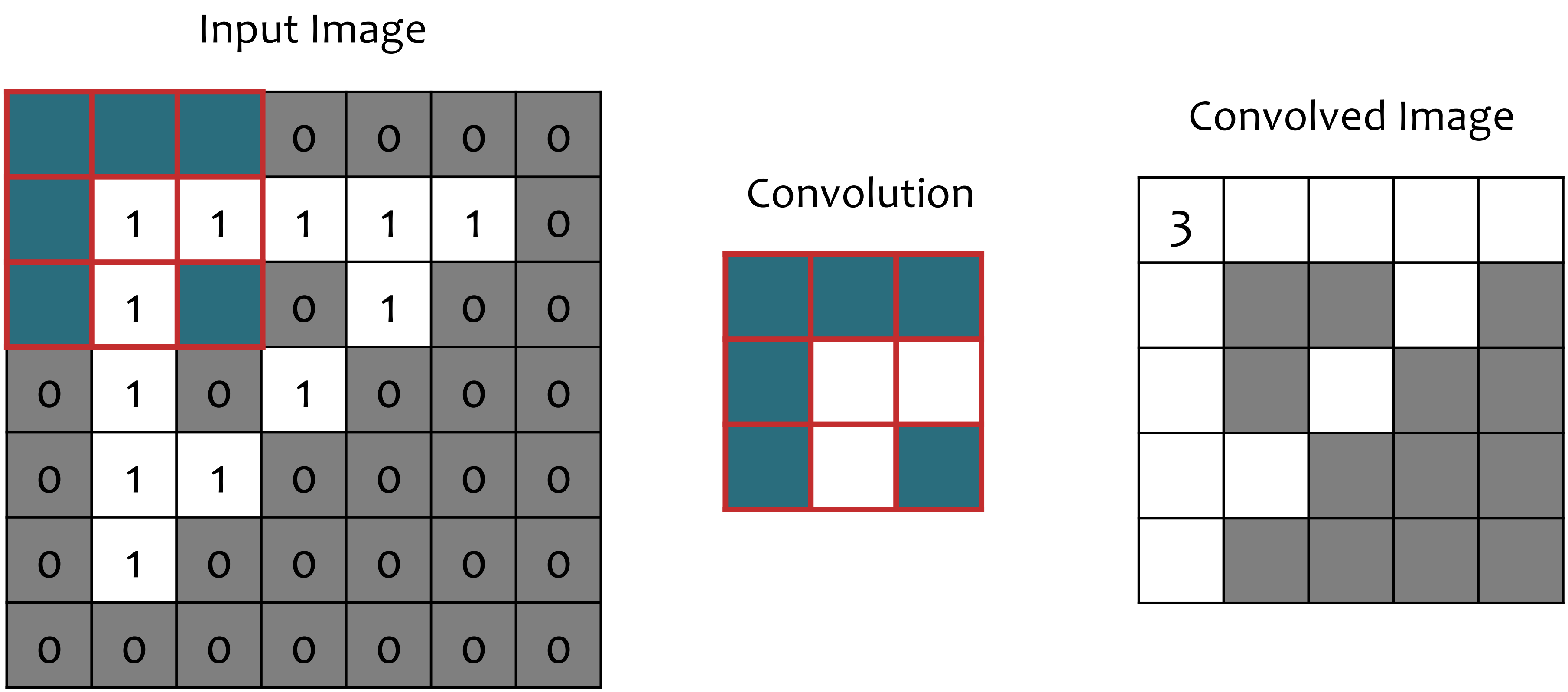


Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

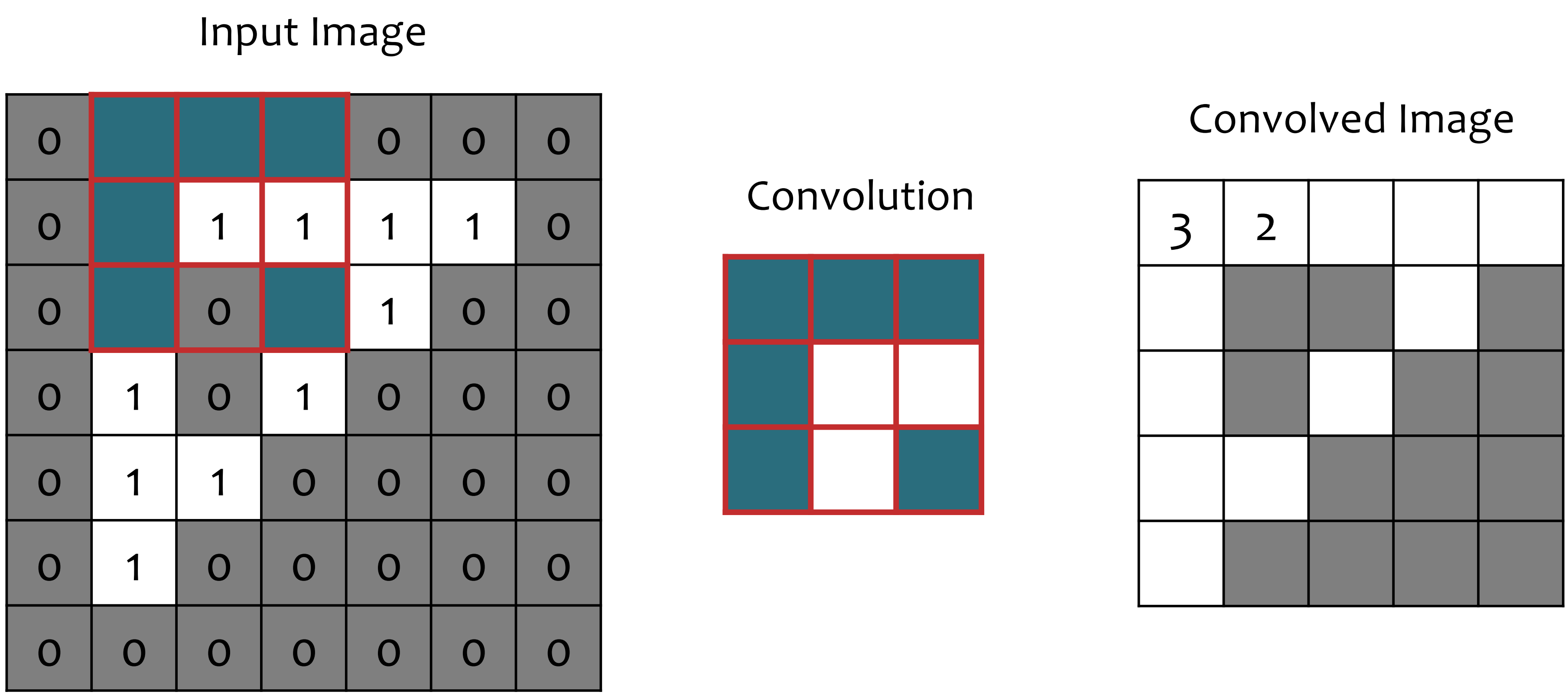
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



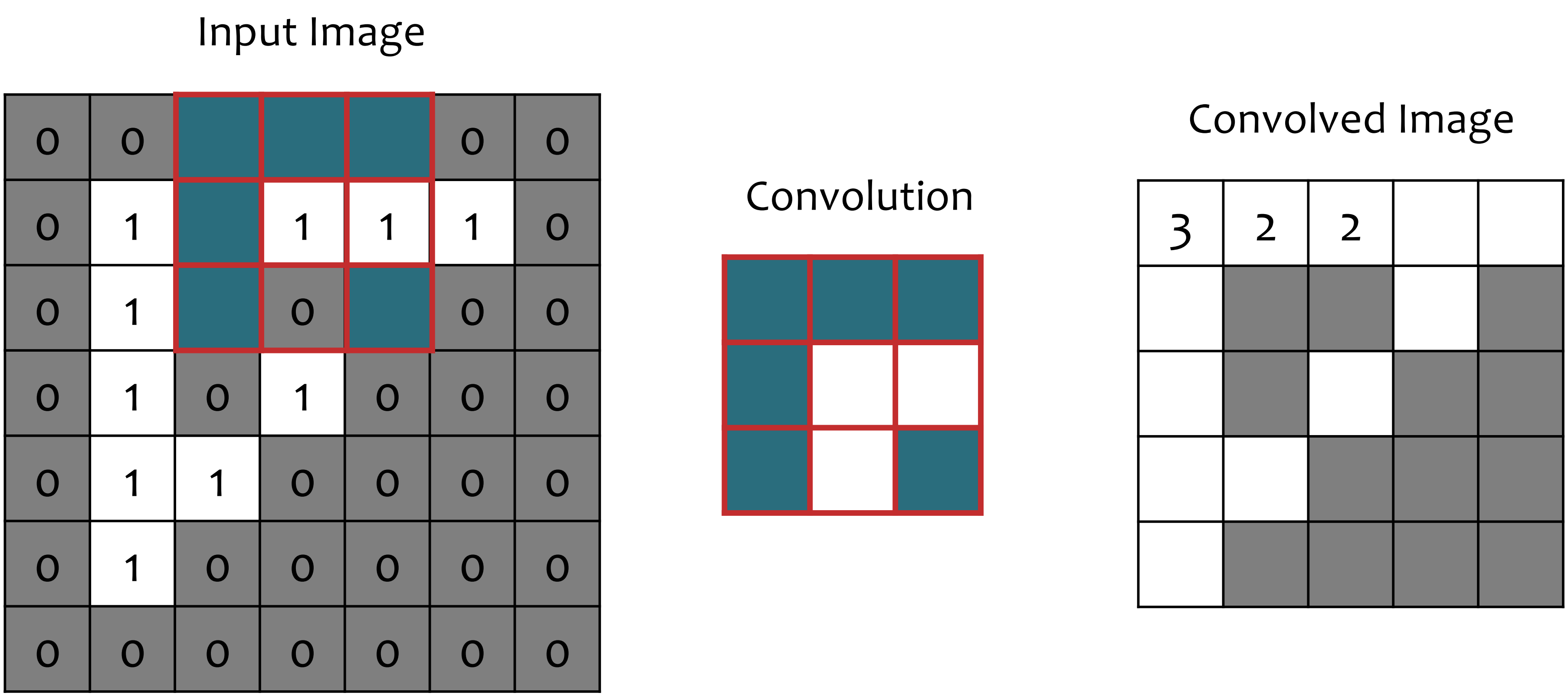
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



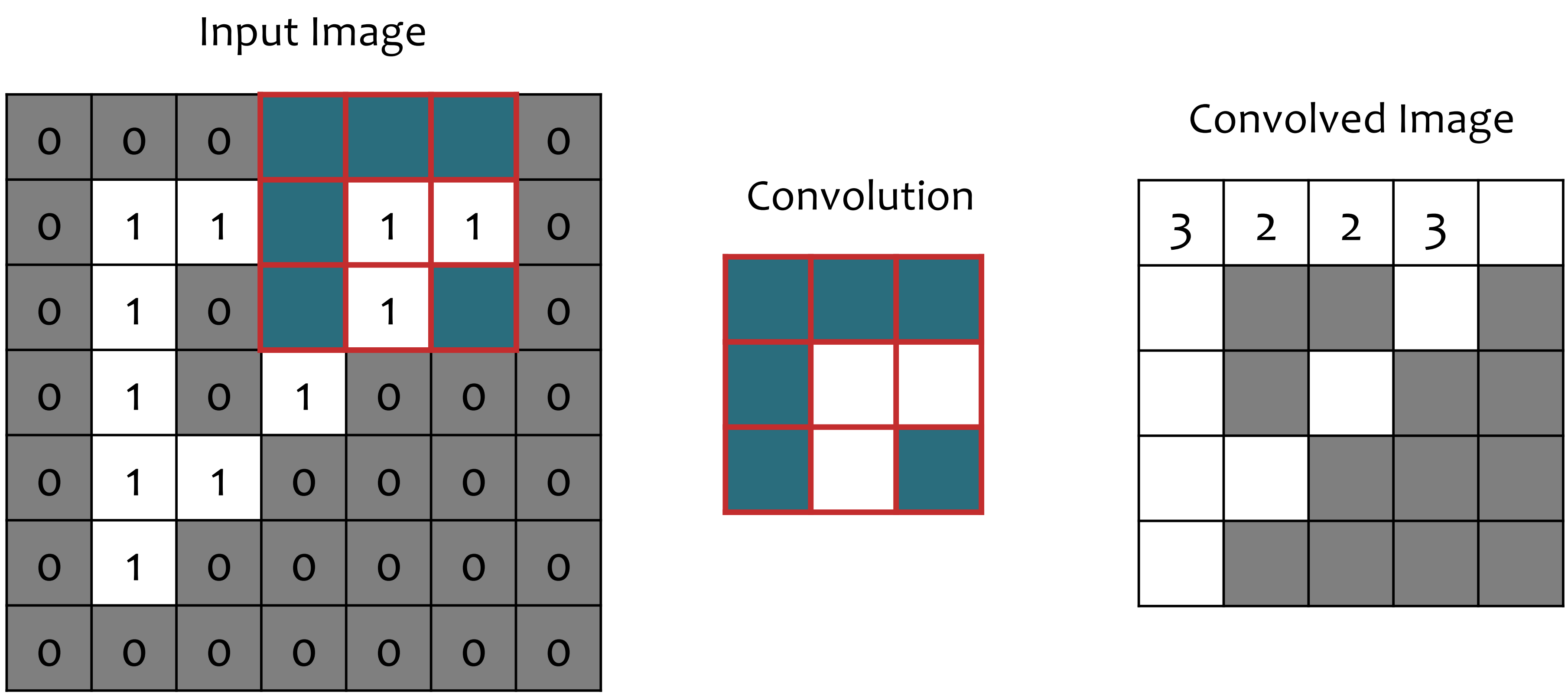
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



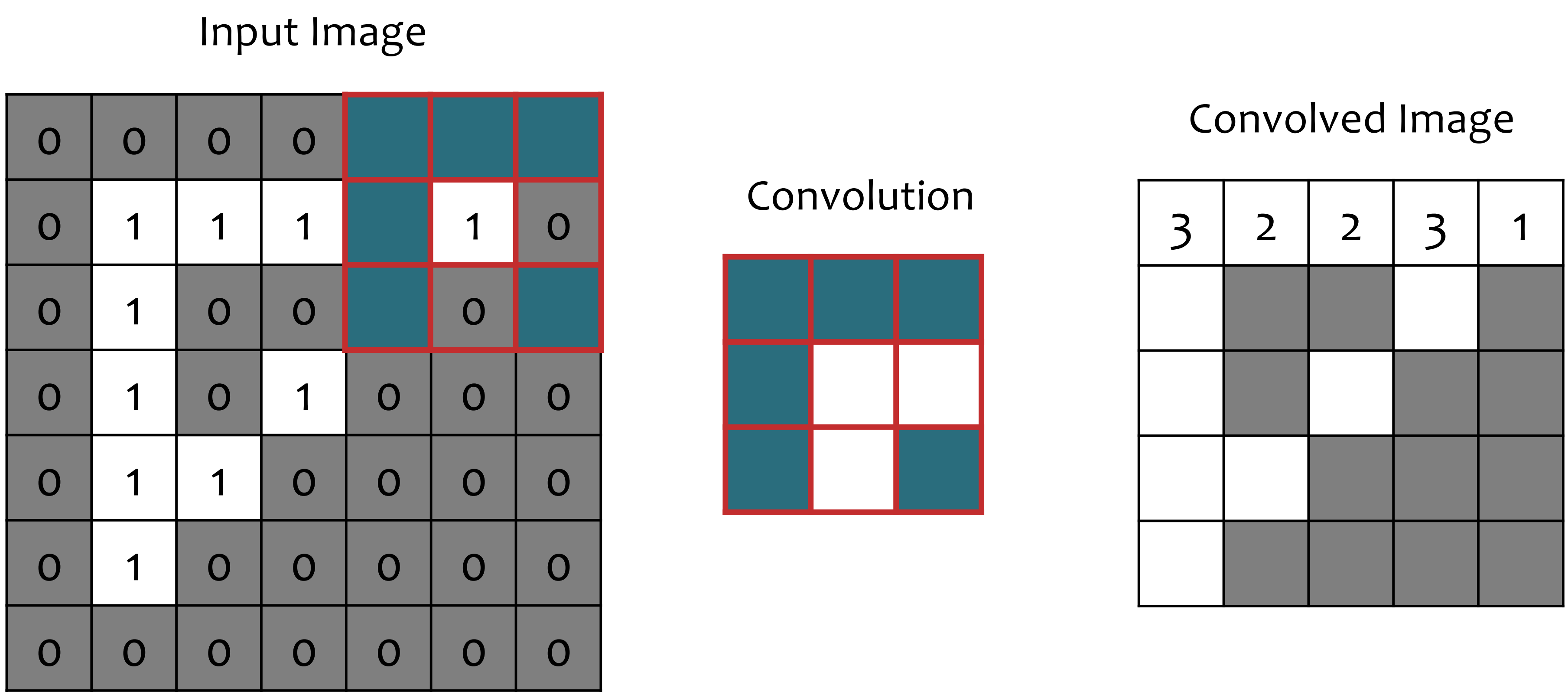
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



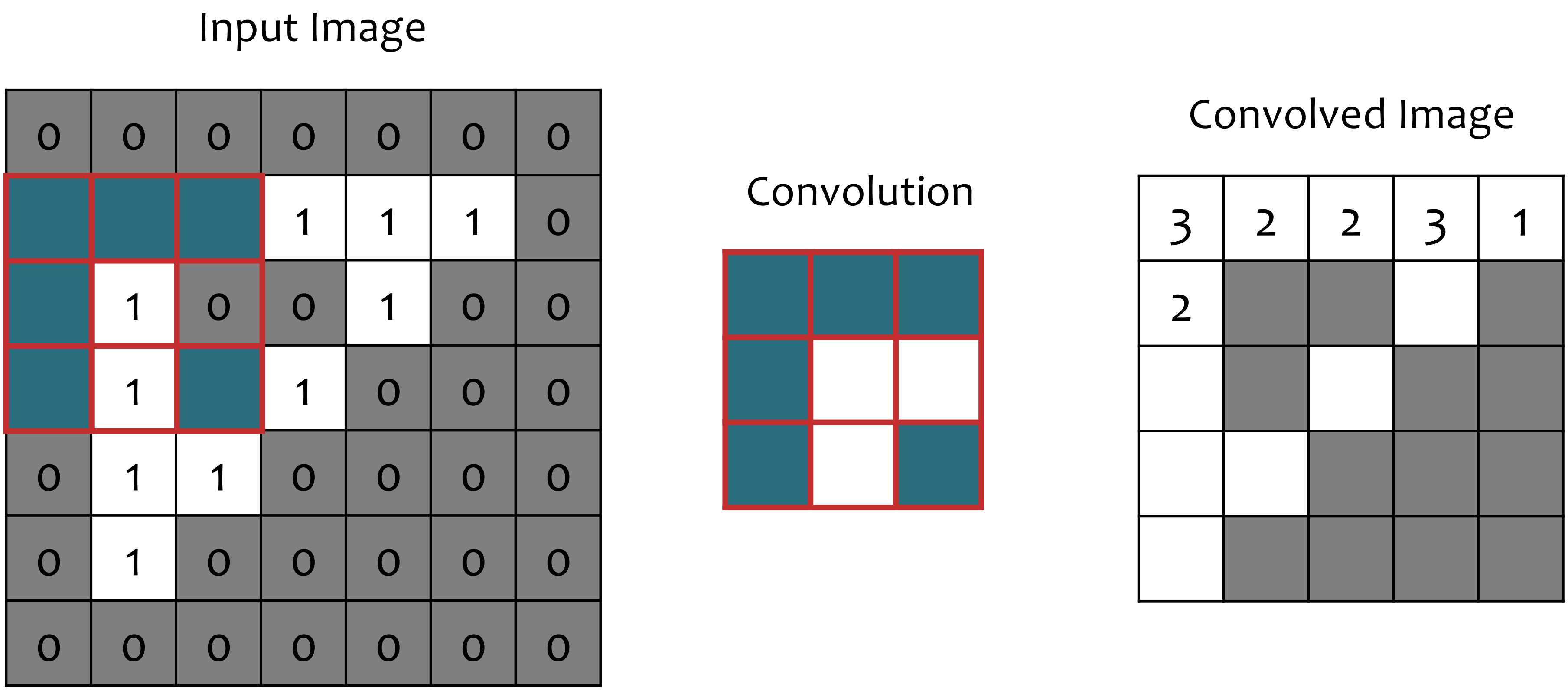
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



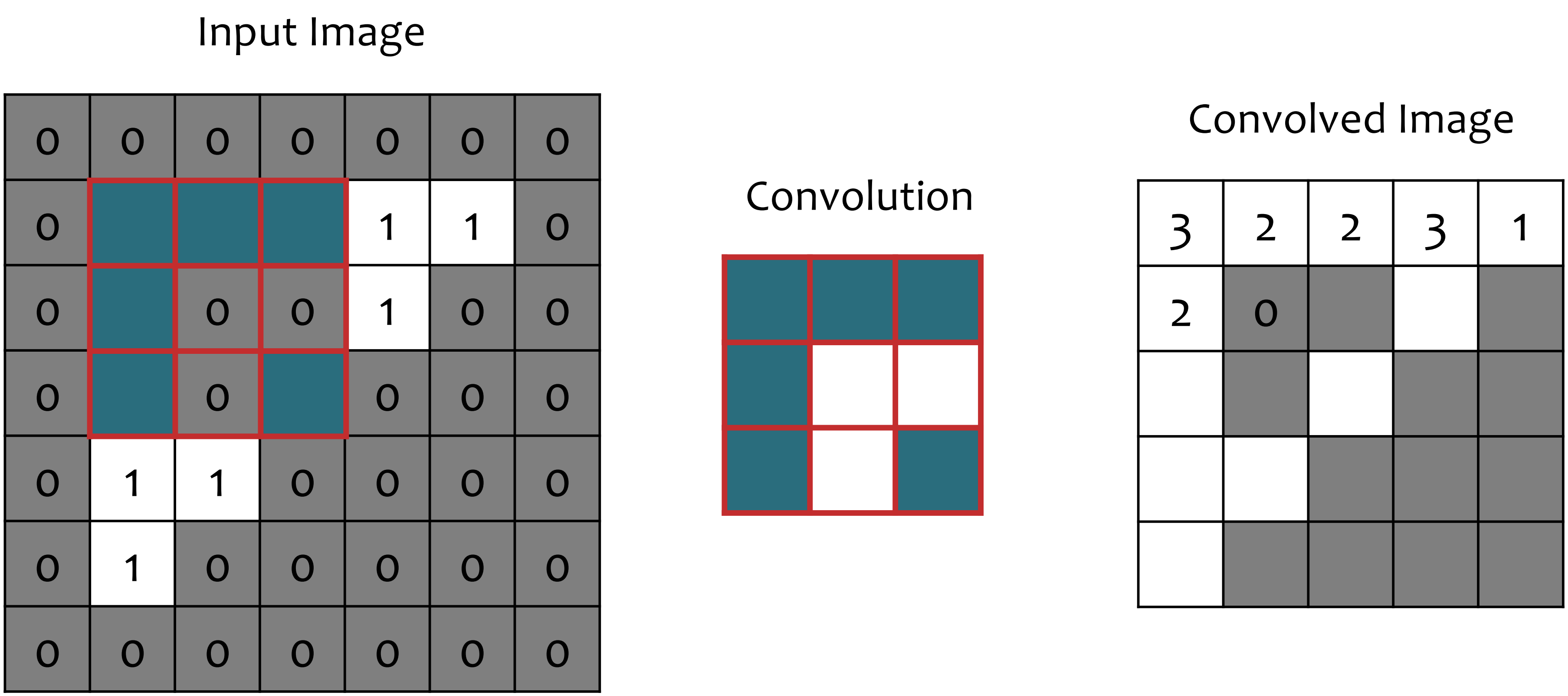
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.



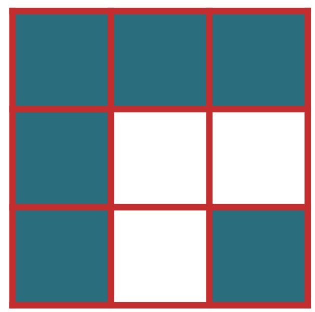
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Identity Convolution

0	0	0
0	1	0
0	0	0

Convolved Image

1	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	0	0	0	0

Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Blurring Convolution

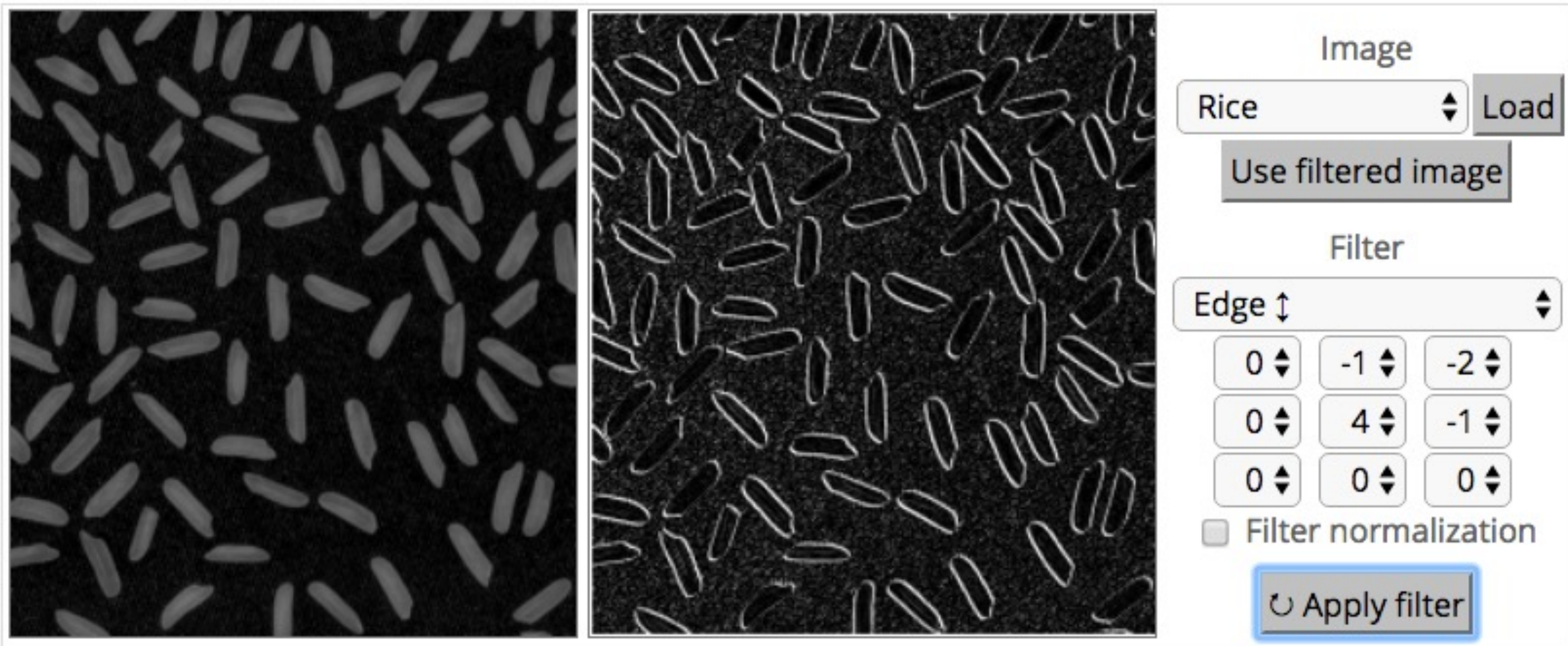
.1	.1	.1
.1	.2	.1
.1	.1	.1

Convolved Image

.4	.5	.5	.5	.4
.4	.2	.3	.6	.3
.5	.4	.4	.2	.1
.5	.6	.2	.1	0
.4	.3	.1	0	0

What's a convolution?

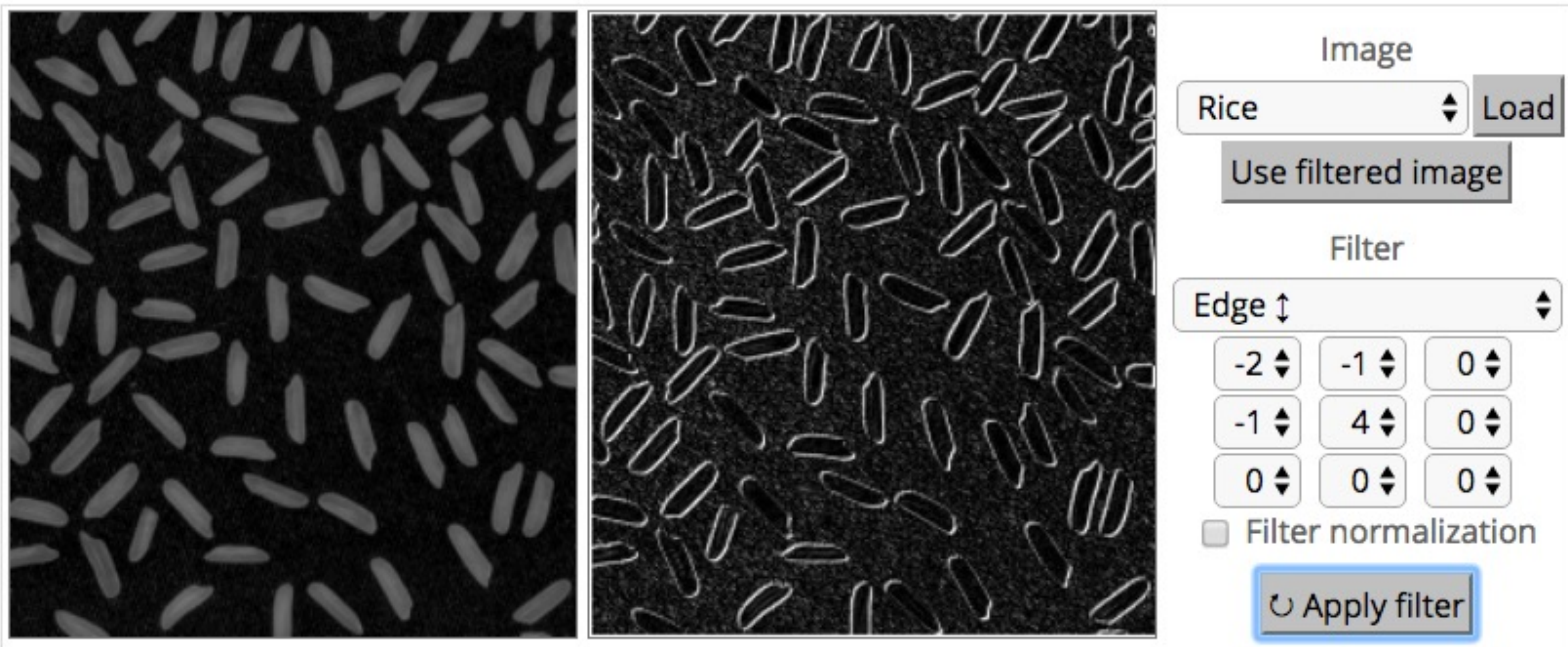
http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo



Slide from William Cohen

What’s a convolution?

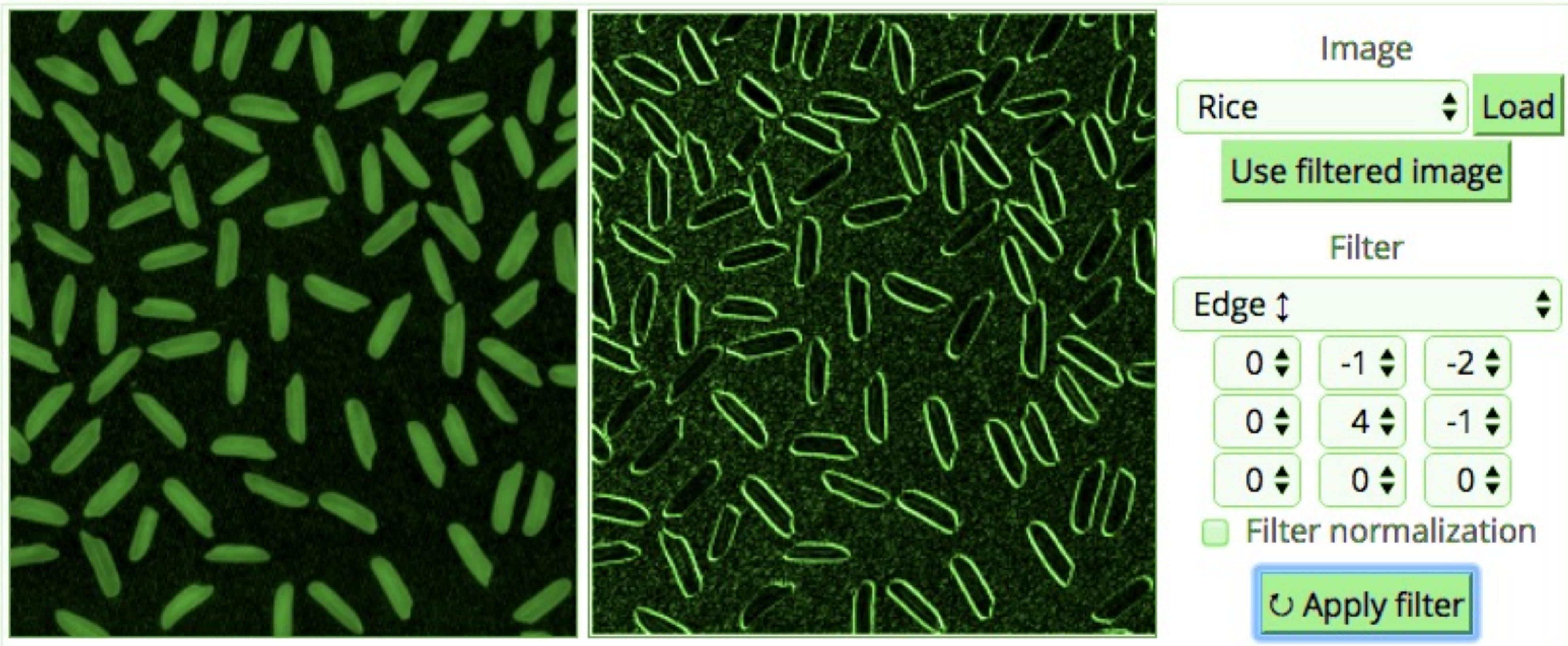
http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo



Slide from William Cohen

What's a convolution?

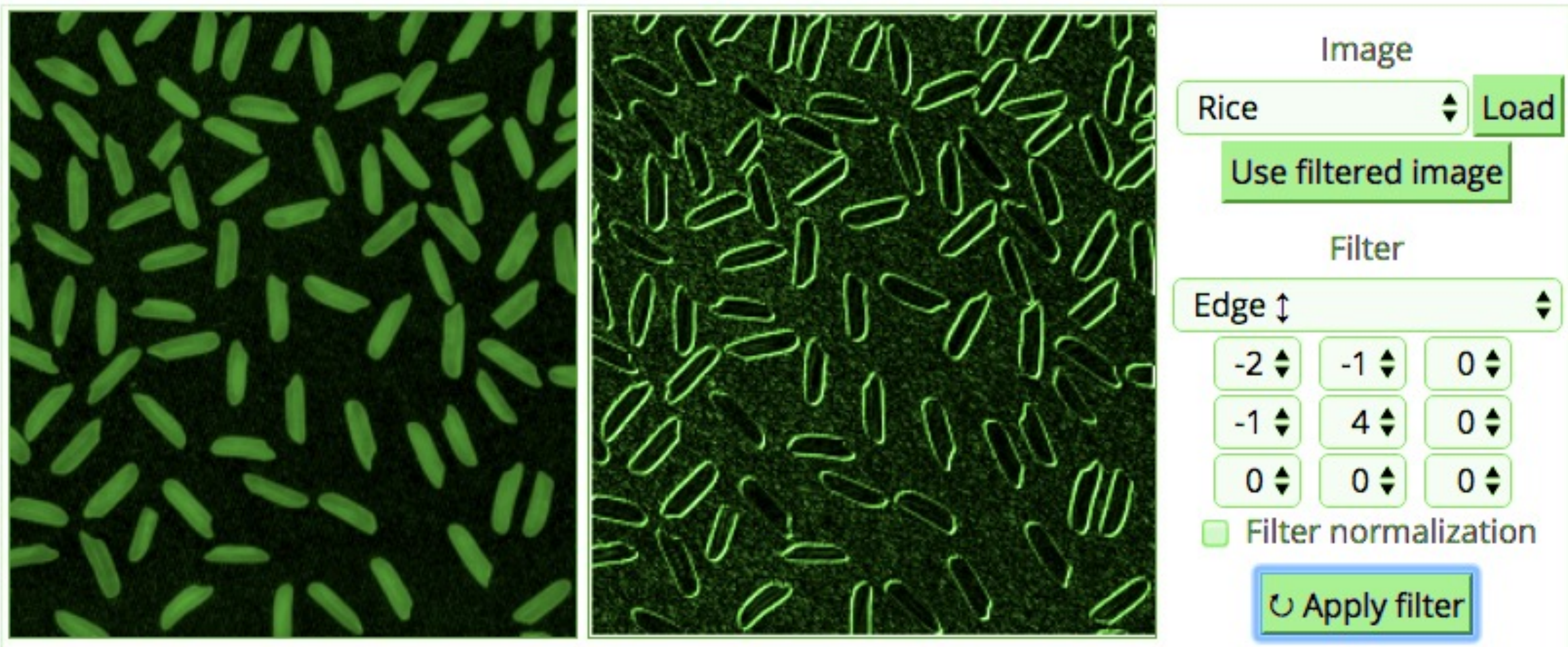
http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo



Slide from William Cohen

What's a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo



Slide from William Cohen

What’s a convolution?

- Basic idea:
  - Pick a 3x3 matrix F of weights
  - Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation
- Key point:
  - Different convolutions extract different types of low-level “features” from an image
  - All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}}_{\text{Input}} \longrightarrow \underbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix}}_{\text{Conv}} \longrightarrow \underbrace{\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}}_{\text{Output}}$$

$$y_{11} = \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0$$

$$y_{12} = \alpha_{11} x_{12} + \alpha_{12} x_{13} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0$$

$$y_{21} = \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0$$

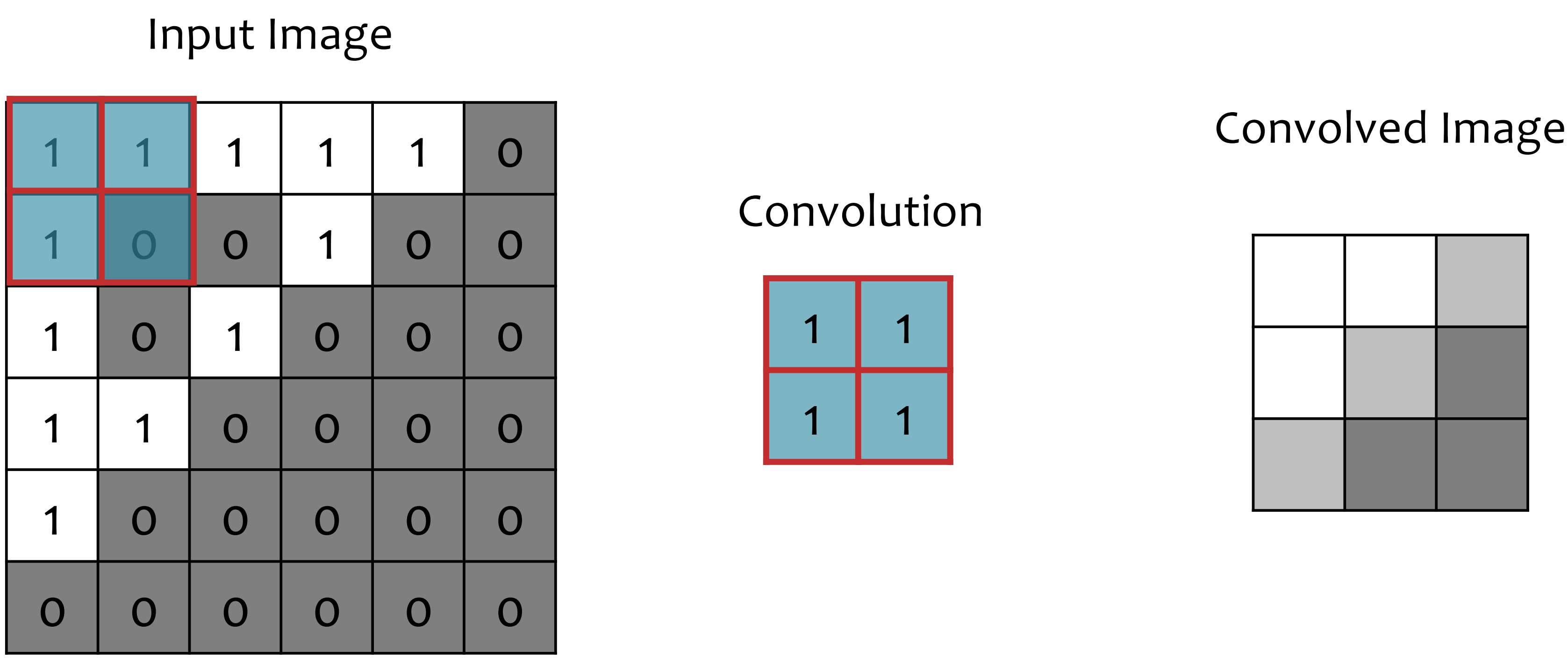
$$y_{22} = \alpha_{11} x_{22} + \alpha_{12} x_{23} + \alpha_{21} x_{32} + \alpha_{22} x_{33} + \alpha_0$$

Slide adapted from William Cohen

DOWNSAMPLING

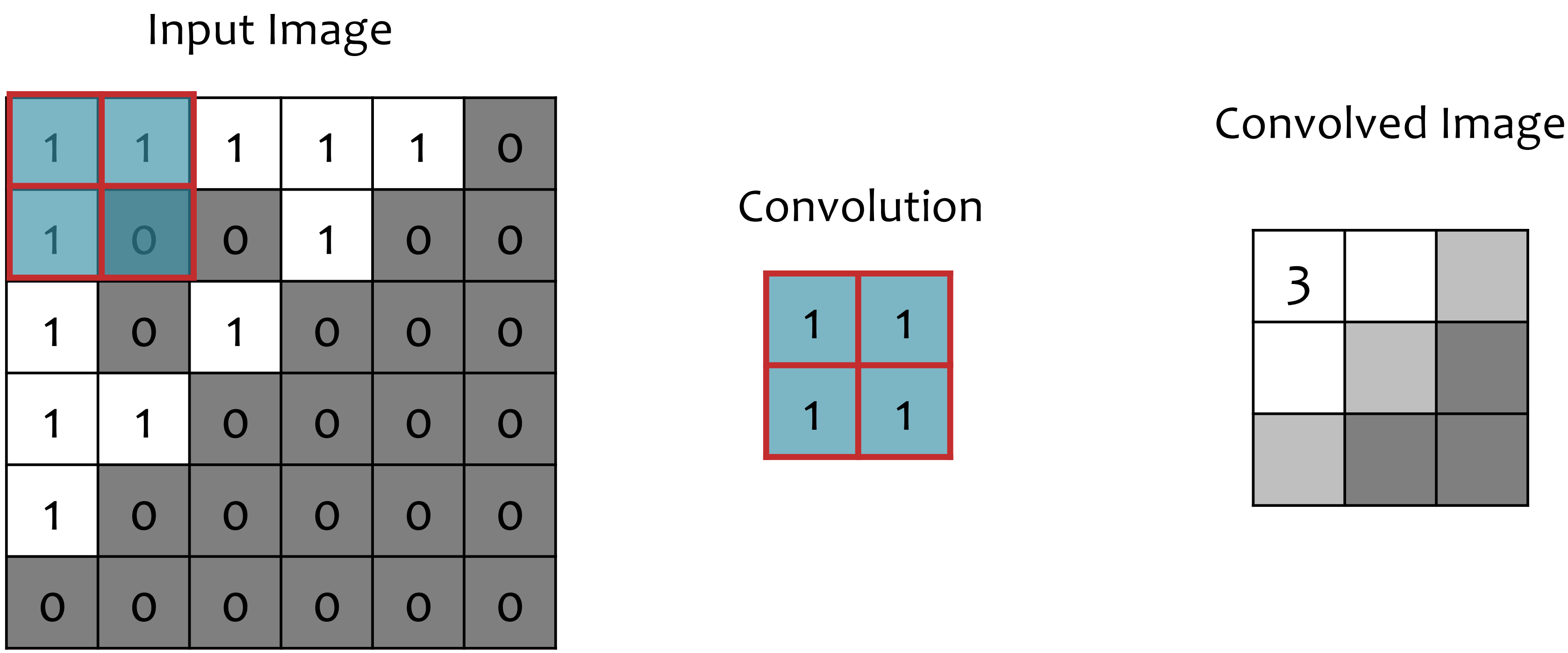
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



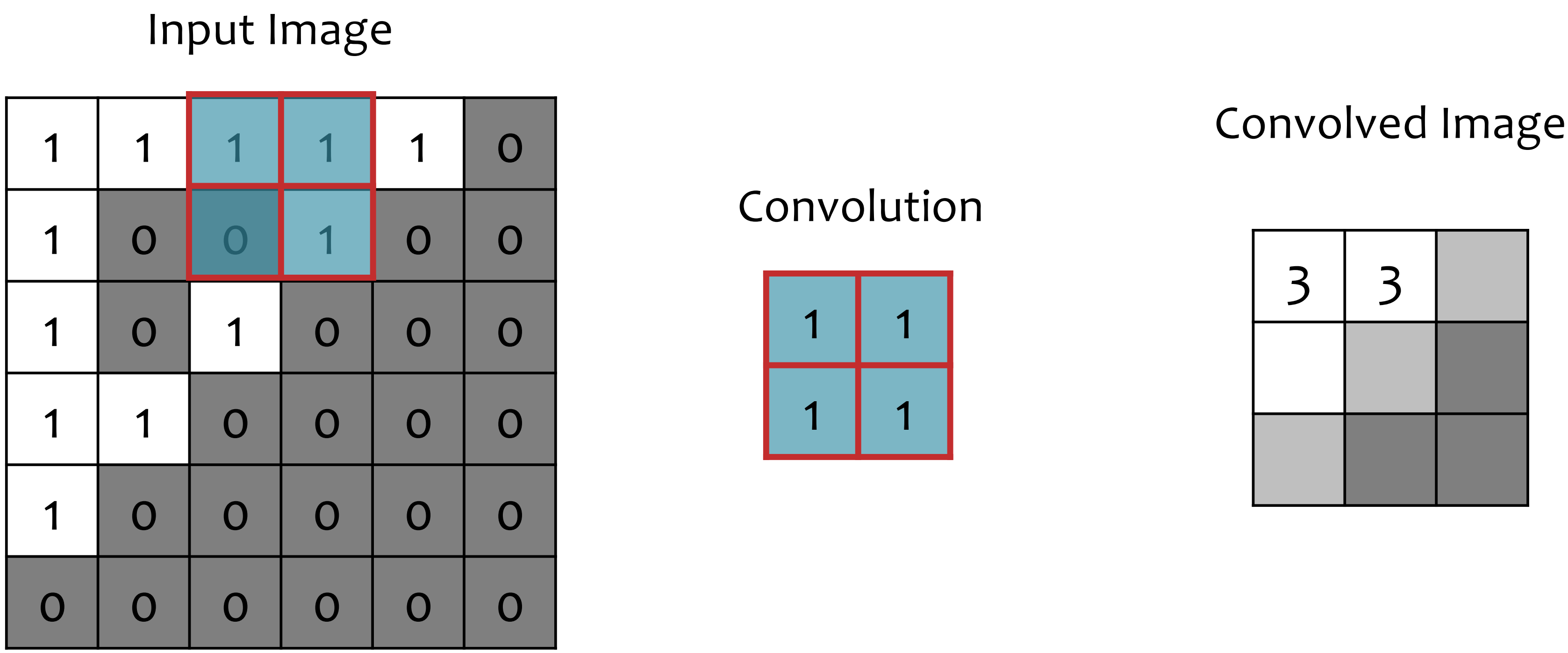
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



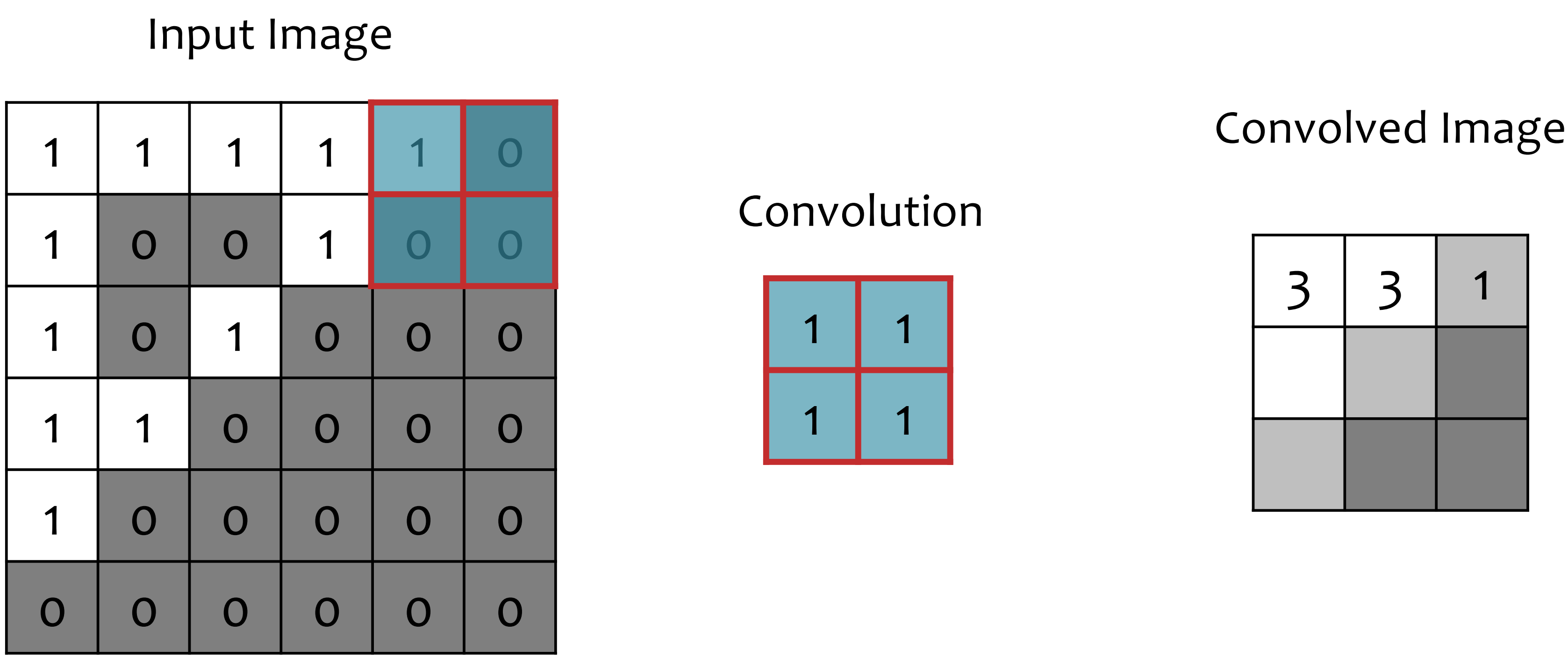
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



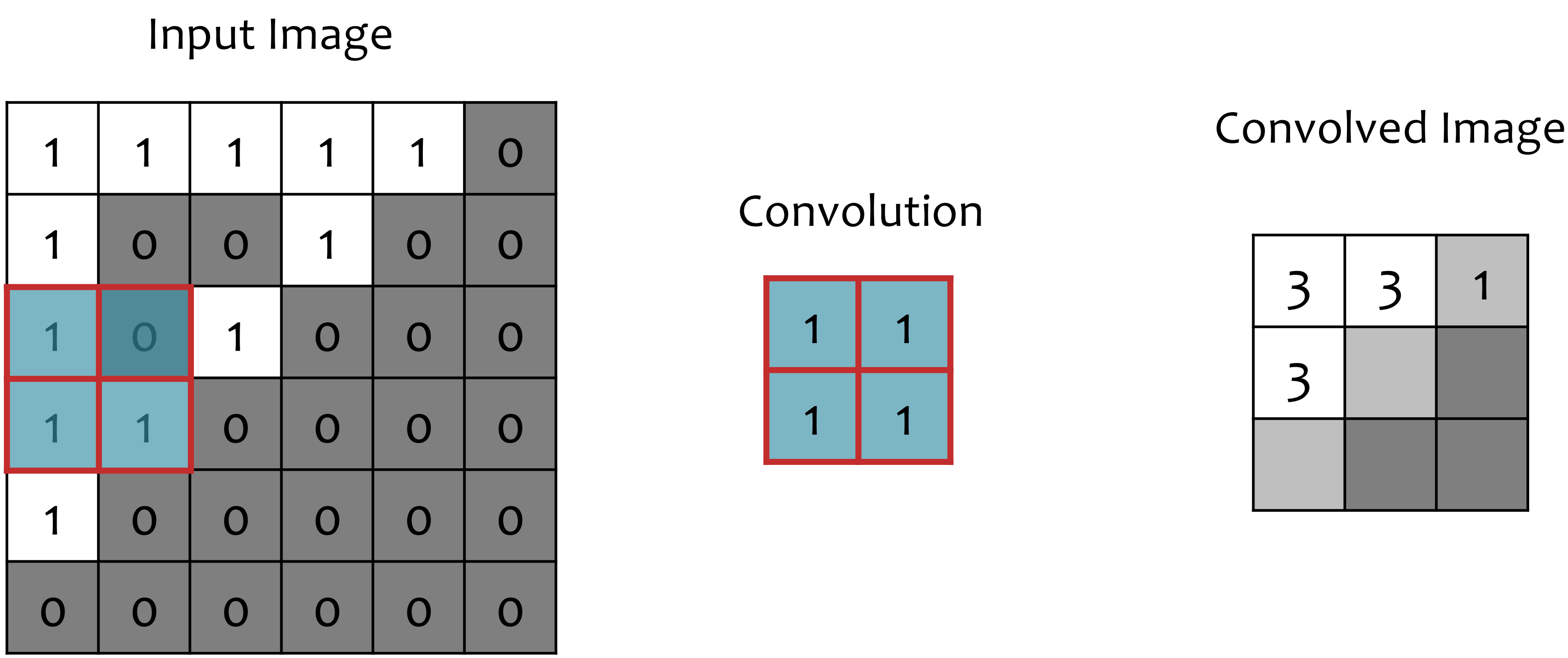
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



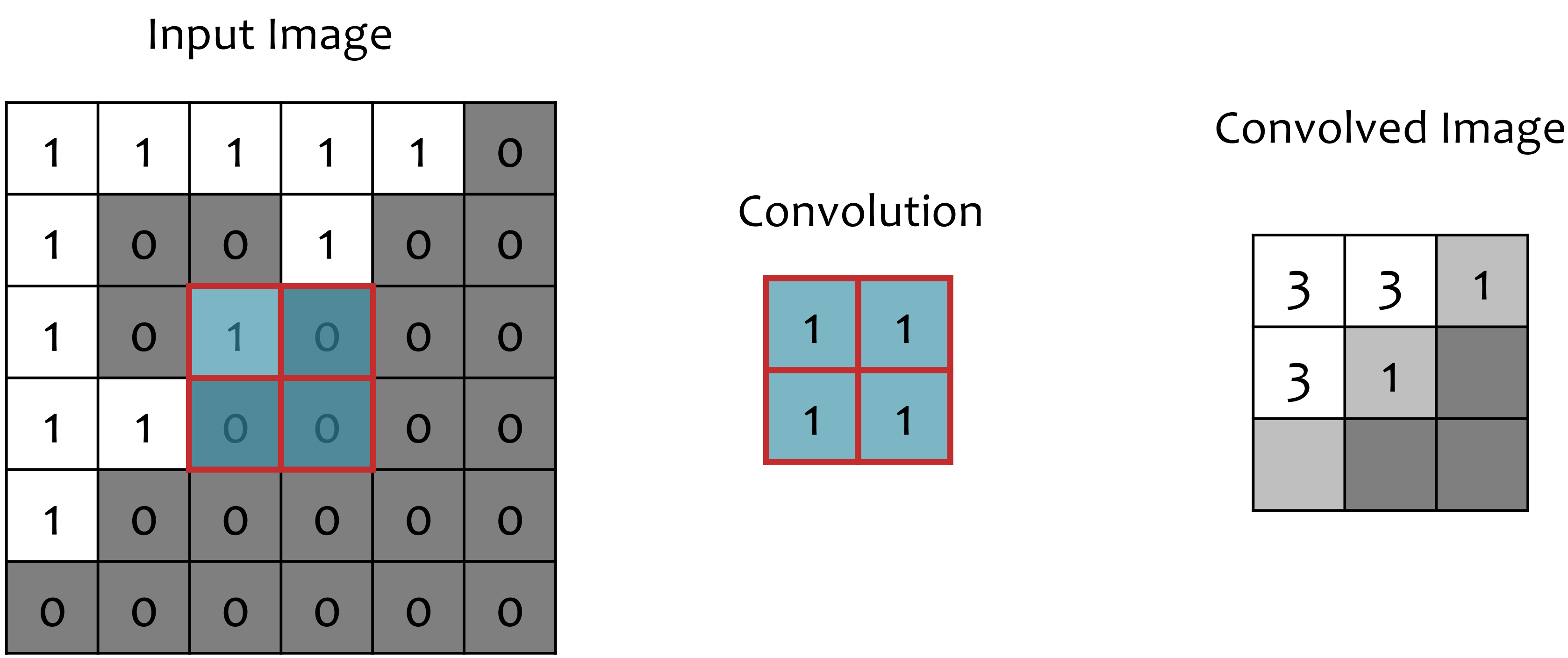
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



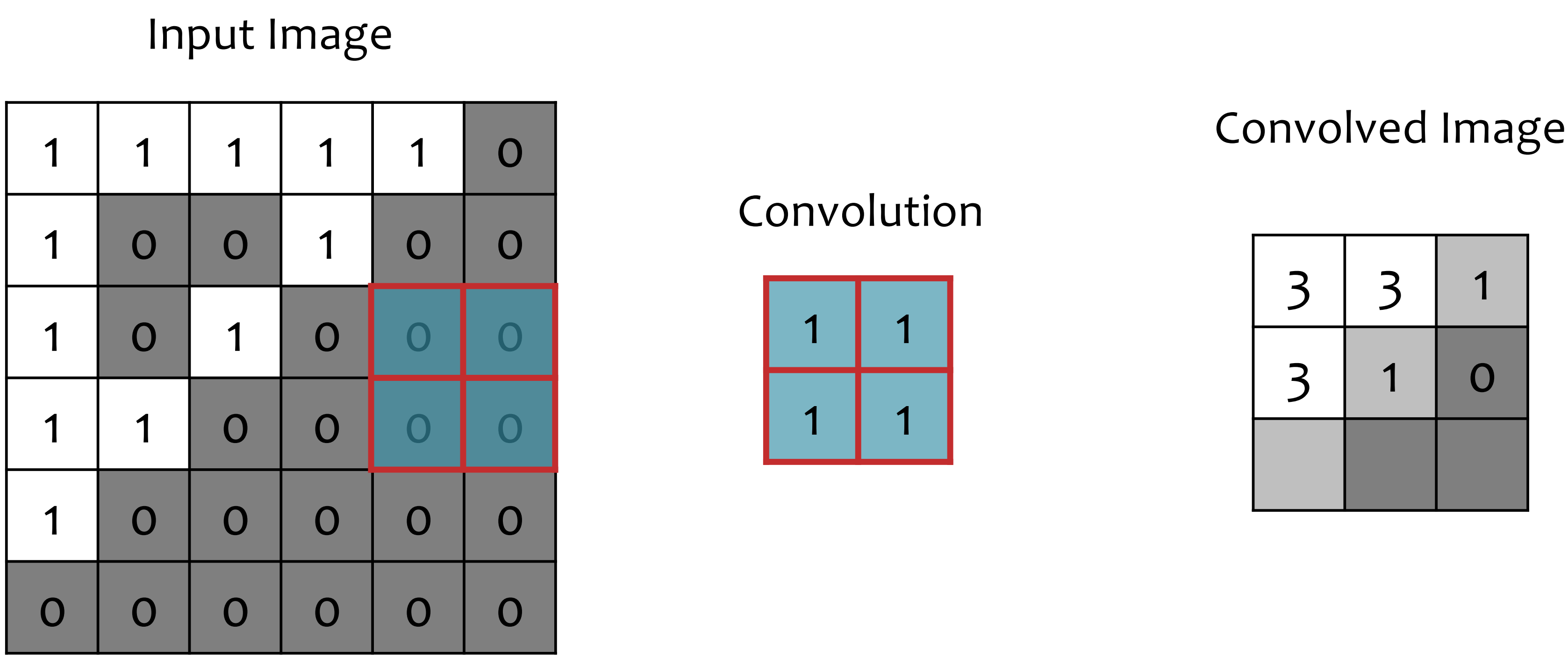
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



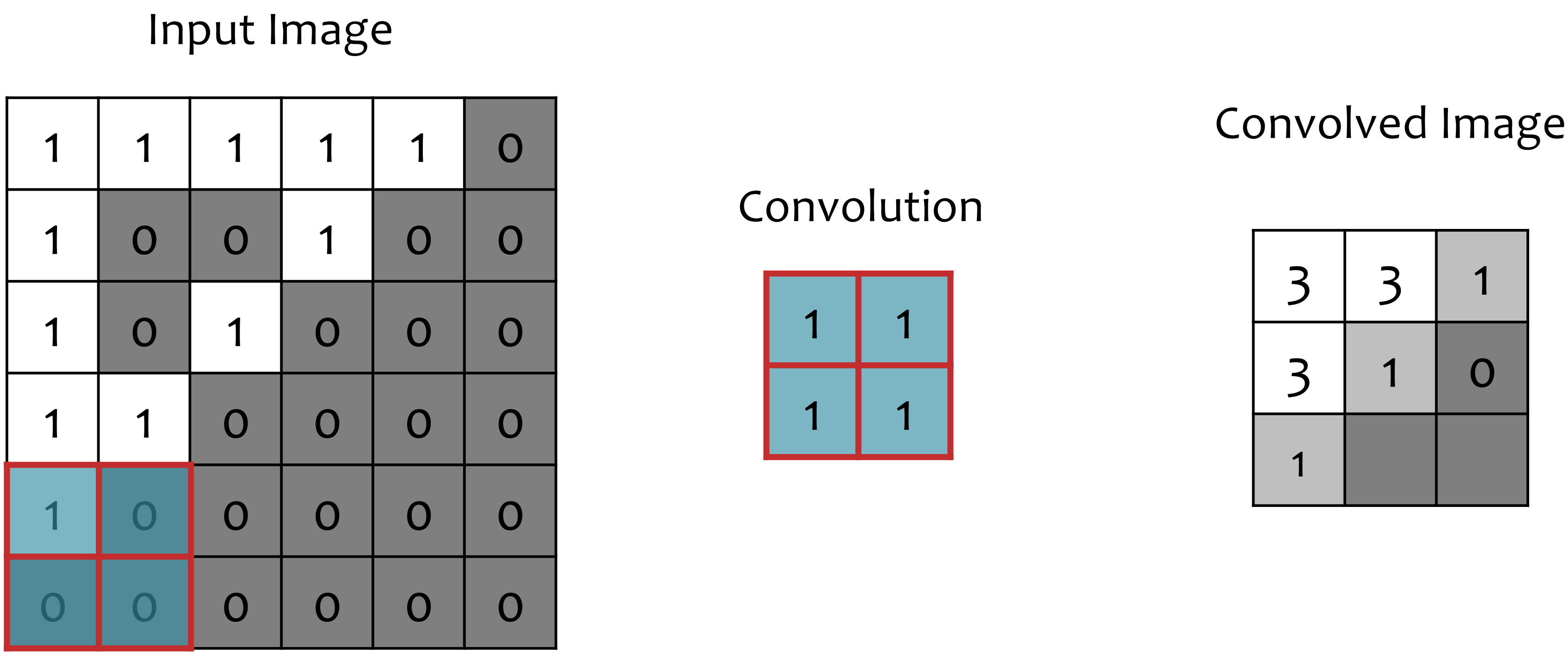
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



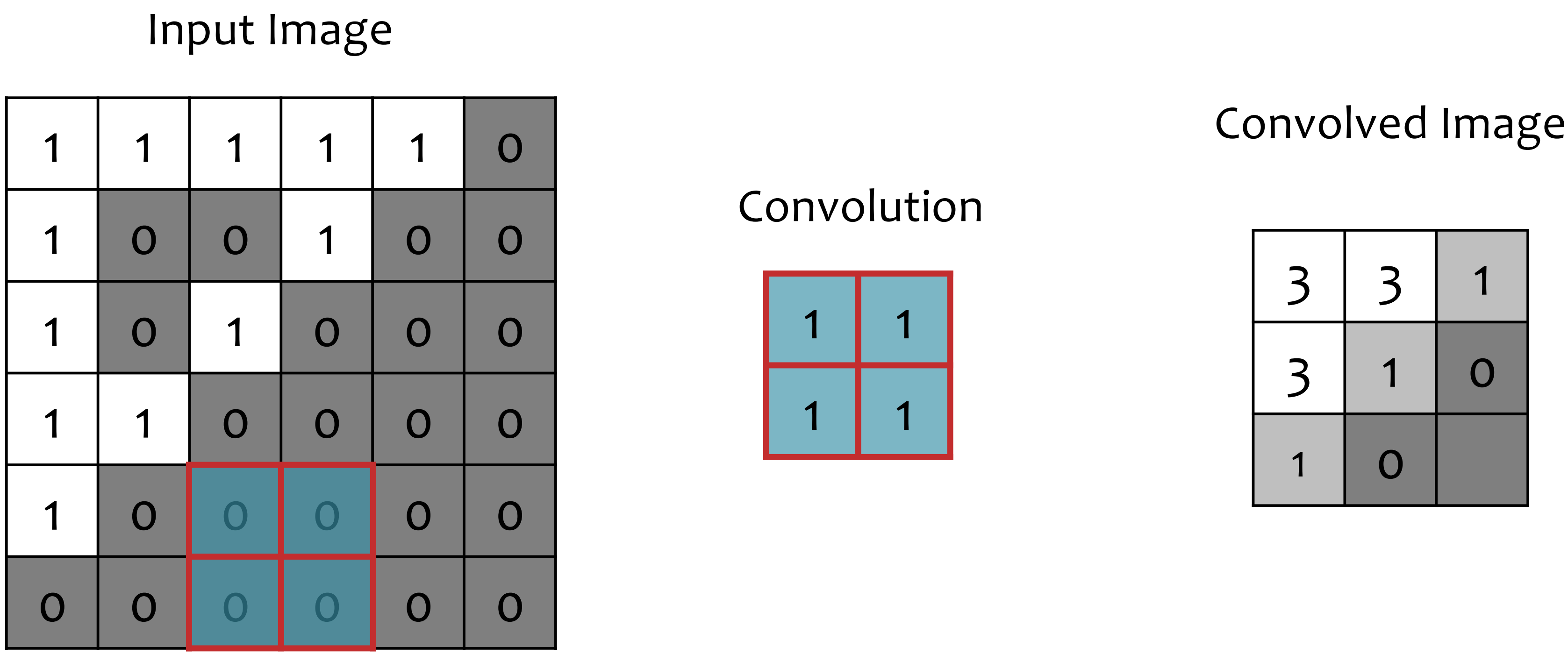
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	0
1	0	0

Downsampling by Averaging

- Downsampling by averaging is a special case of convolution where the weights are fixed to a uniform distribution
- The example below uses a stride of 2

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1/4	1/4
1/4	1/4

Convolved Image

3/4	3/4	1/4
3/4	1/4	0
1/4	0	0

Max-Pooling

- Max-pooling is another form of downsampling
- Instead of averaging, we take the max value within the same range as the equivalently-sized convolution
- The example below uses a stride of 2

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Max-pooling

$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j}$	$x_{i+1,j+1}$

Max-Pooled Image

1	1	1
1	1	0
1	0	0

$$y_{ij} = \max(x_{ij}, \\ x_{i,j+1}, \\ x_{i+1,j}, \\ x_{i+1,j+1})$$

CONVOLUTIONAL NEURAL NETS

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for Machine Learning

1.

- Convolutional Neural Networks (CNNs) provide another form of **decision function**
- Let’s see what they look like…

2. Choose each of these:
   - Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Convolutional Layer

CNN key idea:
Treat convolution matrix as parameters and learn them!

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Learned Convolution

θ_{11}	θ_{12}	θ_{13}
θ_{21}	θ_{22}	θ_{23}
θ_{31}	θ_{32}	θ_{33}

Convolved Image

.4	.5	.5	.5	.4
.4	.2	.3	.6	.3
.5	.4	.4	.2	.1
.5	.6	.2	.1	0
.4	.3	.1	0	0

Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

Architecture #1: LeNet-5

PROC. OF THE IEEE, NOVEMBER 1998 7

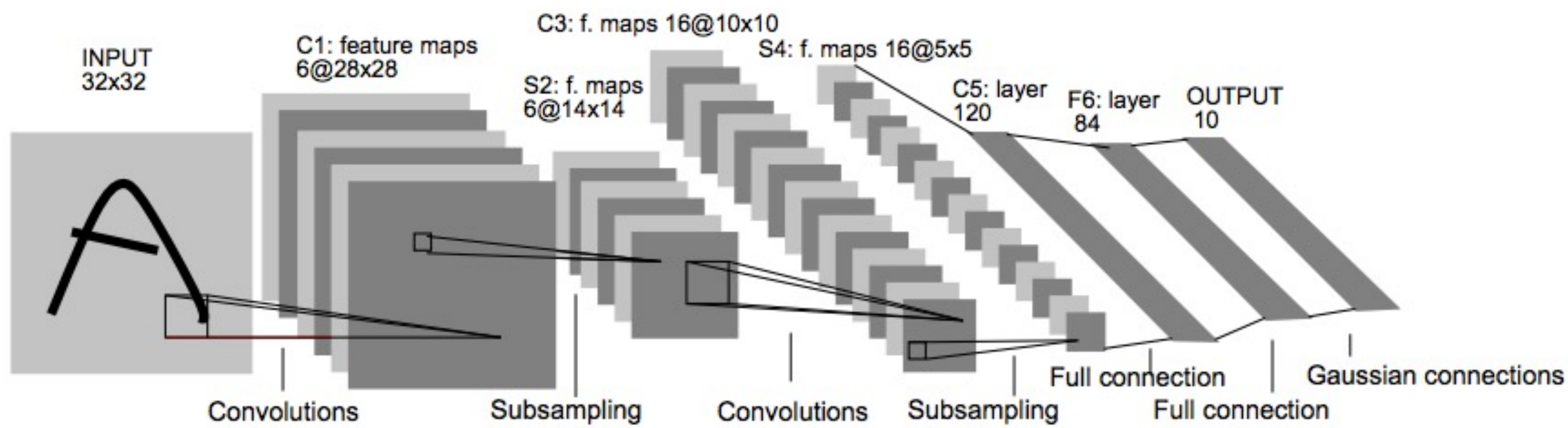


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

TRAINING CNNS

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

3. Define goal:

2. Choose each of th

   – Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   – Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

- Q: Now that we have the CNN as a decision function, how do we compute the gradient?
- A: Backpropagation of course!

opposite the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

SGD for CNNs

SGD for CNNs

Ex: Architecture: Given $\vec{x}, y^*$

$J = \ell(y, y^*)$

$y = \text{softmax}(z^{(5)})$

$z^{(5)} = \text{linear}(z^{(4)}, W)$

$z^{(4)} = \text{relu}(z^{(3)})$

$z^{(3)} = \text{conv}(z^{(2)}, \beta)$

$z^{(2)} = \text{max-pool}(z^{(1)})$

$z^{(1)} = \text{conv}(\vec{x}, \alpha)$

Parameters $\vec{\Theta} = [\alpha, \beta, W]$

SGD:

1. Init $\vec{\Theta}$
2. While not converged:

   Sample $i \in \{1, \ldots, N\}$

   Forward: $y = h_\Theta(\vec{x}^{(i)})$, $J_i(\Theta) = \ell(y, y^*)$

   Backward: $\nabla_{\vec{\Theta}} J_i(\Theta) = \ldots$

   Update: $\vec{\Theta} \leftarrow \vec{\Theta} - \lambda \nabla_{\vec{\Theta}} J_i(\Theta)$

LAYERS OF A CNN

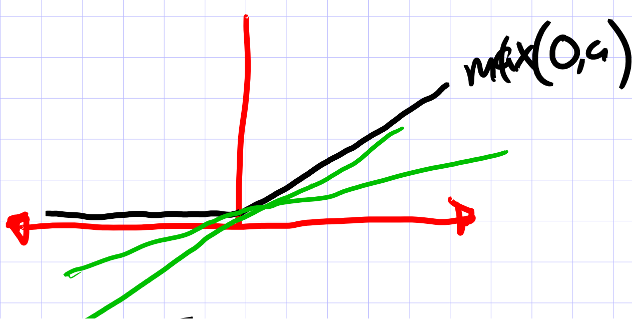
ReLU Layer

ReLU Layer Input: $\vec{x} \in \mathbb{R}^K$ Output: $\vec{y} \in \mathbb{R}^K$

Forward:

$\vec{y} = \sigma(\vec{x})$ ← element-wise

$\sigma(a) = \max(0,a)$



Backward:

$$\frac{dJ}{dx_i} = \frac{dJ}{dy_i}\frac{dy_i}{dx_i}$$

subderivative

where $\frac{dy_i}{dx_i} = \begin{cases} 1 & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$

Softmax Layer

Softmax Layer

Input: $\vec{x} \in \mathbb{R}^K$ Output: $\vec{y} \in \mathbb{R}^K$

Forward:

$$y_i = \frac{\exp(x_i)}{\sum_{k=1}^{K} \exp(x_k)}$$

Backward:

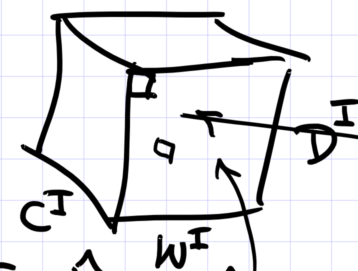
$$\frac{dJ}{dx_j} = \sum_{i=1}^{K} \frac{dJ}{dy_i} \frac{dy_i}{dx_j}$$

where $\frac{dy_i}{dx_j} = \begin{cases} y_i(1-y_i) & \text{if } i=j \\ -y_i y_j & \text{otherwise} \end{cases}$

Fully-Connected Layer

$\boxed{\text{Fully Connected Layer}}$ (w/ tensor input)

- Suppose input is a 3D Tensor: $X =$



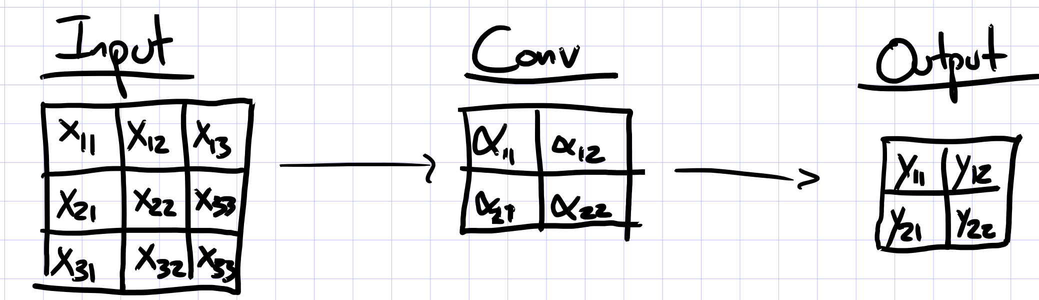
- stretch out into a long vector. $\hat{X} = [\hat{x}_1, \ldots, \hat{x}_{(C \times H \times W)}]$
- then standard linear layer:

$$y = \alpha^T \hat{x} + \alpha_0 \quad \text{where } \alpha \in \mathbb{R}^{A \times B}$$

$$|\hat{x}| = A, \ |y| = B$$

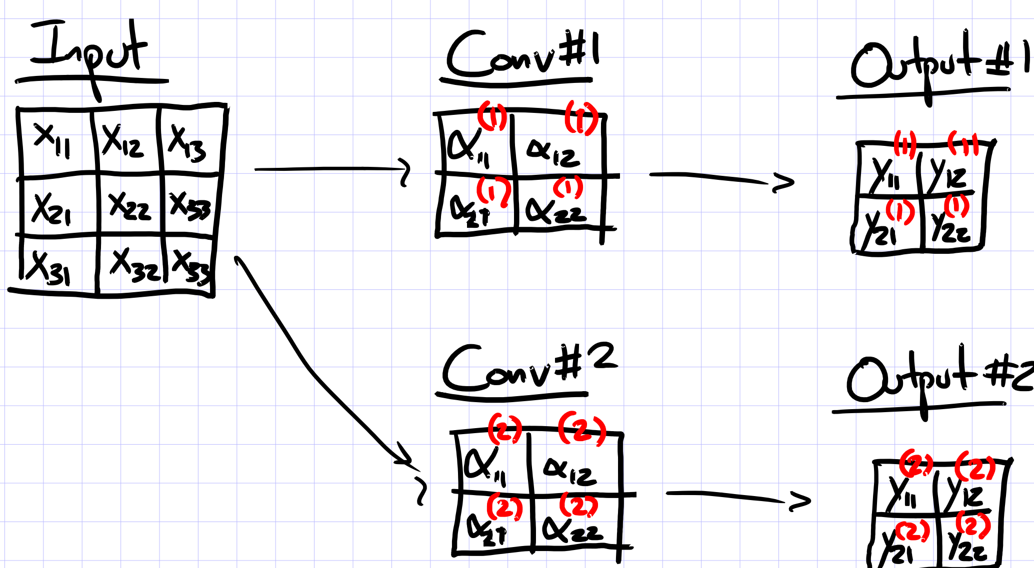
Convolutional Layer

Ex: 1 input channel, 1 output channel



$$y_{11} = \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_0$$
$$y_{12} = \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_0$$
$$y_{21} = \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_0$$
$$y_{22} = \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0$$

Ex: 1 input channel, 2 output channels

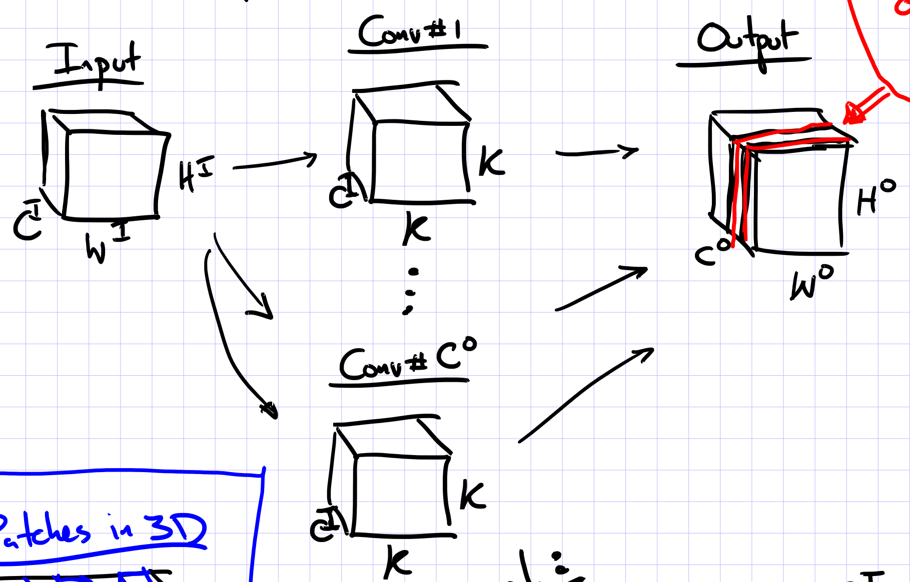


$$y_{11}^{(1)} = \alpha_{11}^{(1)}x_{11} + \alpha_{12}^{(1)}x_{12} + \alpha_{21}^{(1)}x_{21} + \alpha_{22}^{(1)}x_{22} + \alpha_0^{(1)}$$
$$y_{12}^{(1)} = \cdots$$
$$y_{21}^{(1)} = \cdots$$
$$y_{22}^{(1)} = \alpha_{11}^{(1)}x_{22} + \alpha_{12}^{(1)}x_{23} + \alpha_{21}^{(1)}x_{32} + \alpha_{22}^{(1)}x_{33} + \alpha_0^{(1)}$$

$$y_{11}^{(2)} = \alpha_{11}^{(2)}x_{11} + \alpha_{12}^{(2)}x_{12} + \alpha_{21}^{(2)}x_{21} + \alpha_{22}^{(2)}x_{22} + \alpha_0^{(2)}$$
$$y_{12}^{(2)} = \cdots$$
$$y_{21}^{(2)} = \cdots$$
$$y_{22}^{(2)} = \alpha_{11}^{(2)}x_{22} + \alpha_{12}^{(2)}x_{23} + \alpha_{21}^{(2)}x_{32} + \alpha_{22}^{(2)}x_{33} + \alpha_0^{(2)}$$

Convolutional Layer

Ex: C^I input channels, C^I output channels

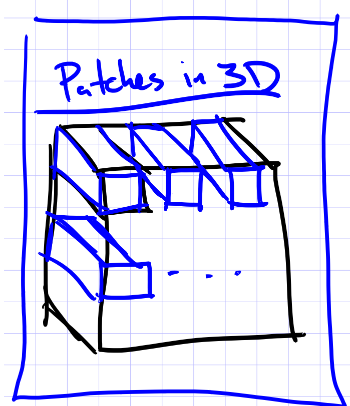


jth slice is output from jth convolution matrix

$$H^O = \lfloor (H^I + 2p - K)/s + 1 \rfloor$$

$$W^O = \lfloor (W^I + 2p - K)/s + 1 \rfloor$$

where p = # pixels of padding on input
K = size of conv. matrix
s = stride length



Forward:

$$y_{ij}^{(k)} = \alpha_0^{(k)} + \sum_{c=1}^{C^I} \sum_{q=1}^{K} \sum_{r=1}^{K} \alpha_{qr}^{(k)} x_{mn}^{(c)} \quad \text{where } \begin{array}{l} m = s(i-1)+q \\ n = s(j-1)+r \end{array}$$

Backward:

$$\frac{dJ}{d\alpha_0^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{d\alpha_0^{(k)}}$$

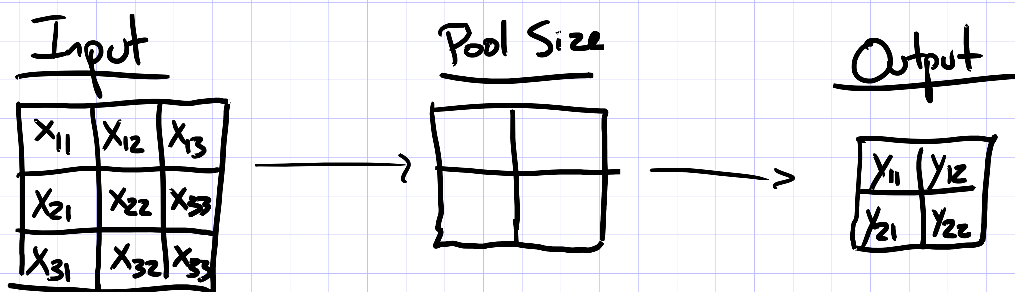
$$\frac{dJ}{d\alpha_{qr}^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{d\alpha_{qr}^{(k)}}$$

$$\frac{dJ}{dx_{mn}^{(c)}} = \sum_i \sum_j \sum_k \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{dx_{mn}^{(c)}}$$

just some calculus

Max-Pooling Layer

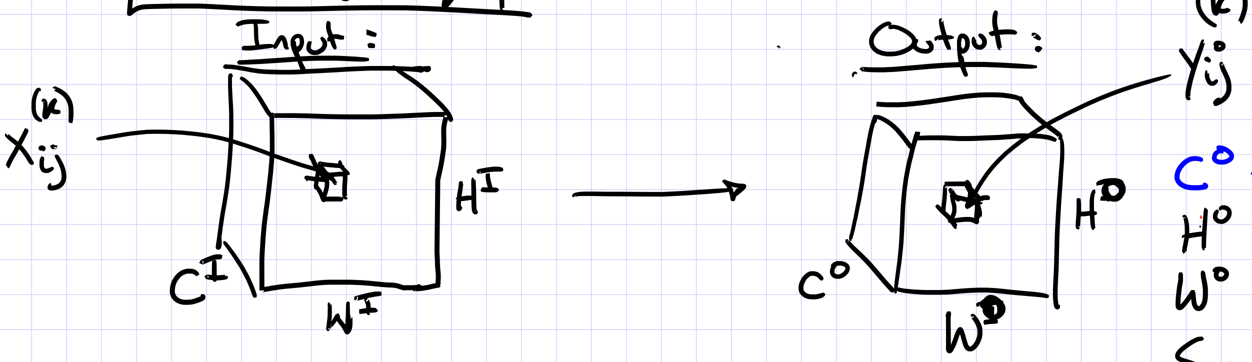
Ex: 1 input channel, 1 output channel, stride of 1



$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$
$$y_{12} = \max(x_{12}, x_{13}, x_{22}, x_{23})$$
$$y_{21} = \max(x_{21}, x_{22}, x_{31}, x_{32})$$
$$y_{22} = \max(x_{22}, x_{23}, x_{32}, x_{33})$$

Max-Pooling Layer

Max-Pooling Layer



$C^O = C^I$

$H^O = \ldots$ } same as conv. layer

$W^O = \ldots$ }

Same hyperparams too.

Forward:

$$y_{ij}^{(k)} = \max_{\substack{q \in \{1,\ldots,K\} \\ r \in \{1,\ldots,K\}}} x_{mn}^{(k)} \quad \text{where} \quad \begin{aligned} m &= s(i-1)+q \\ n &= s(j-1)+r \end{aligned}$$

Backward:

$$\frac{dJ}{dx_{mn}^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{dx_{mn}^{(k)}}$$

$\boxed{\text{Subderivatives}}$

- Max() is not differentiable, but subdifferentiable.
- There are a set of derivatives and we can just choose one for SGD.

$y = \max(a, b)$

$$\Rightarrow \frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da} \quad \text{where} \quad \frac{dy}{da} = \begin{cases} 1 & \text{if } a > b \\ 0 & \text{otherwise} \end{cases}$$

Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

Architecture #1: LeNet-5

PROC. OF THE IEEE, NOVEMBER 1998 7

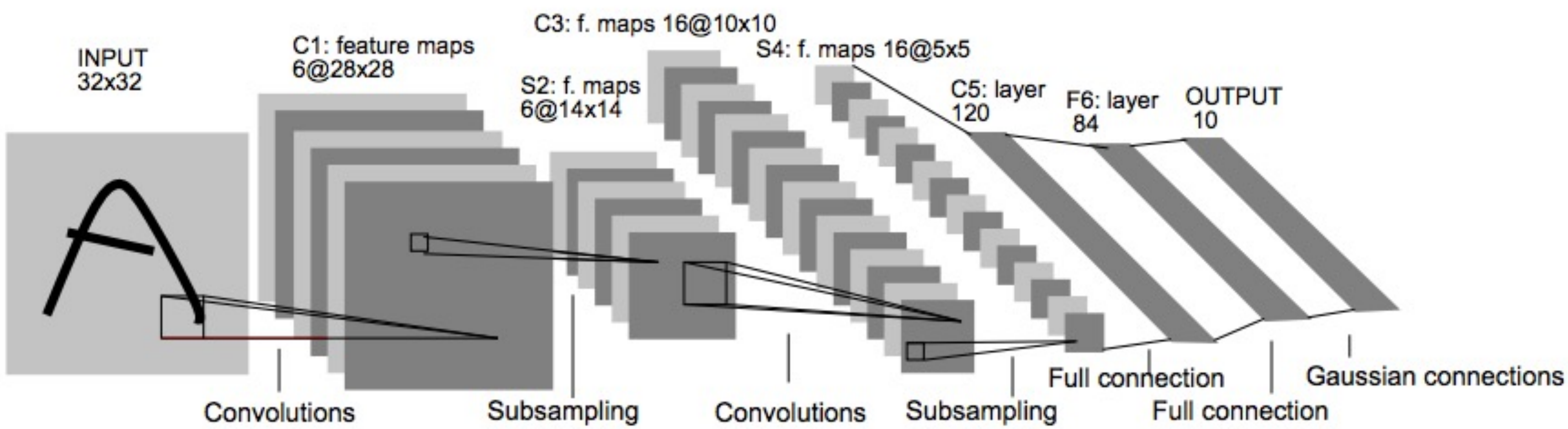


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

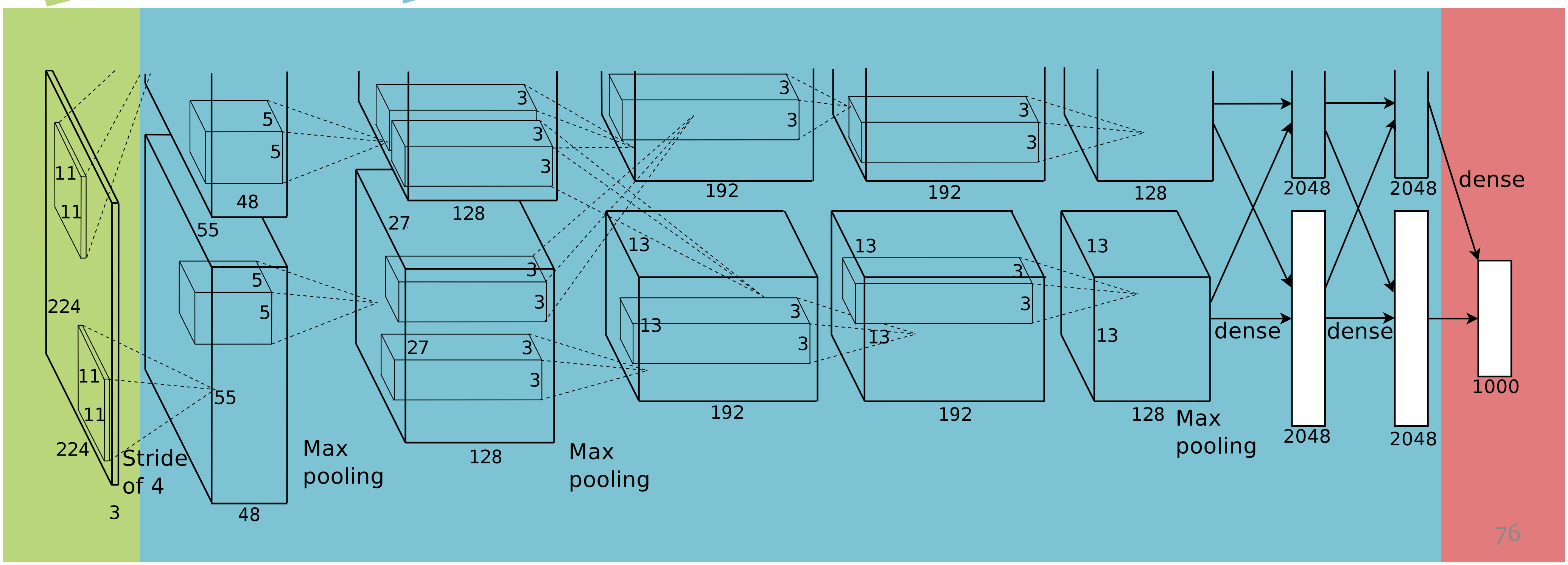
Architecture #2: AlexNet

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

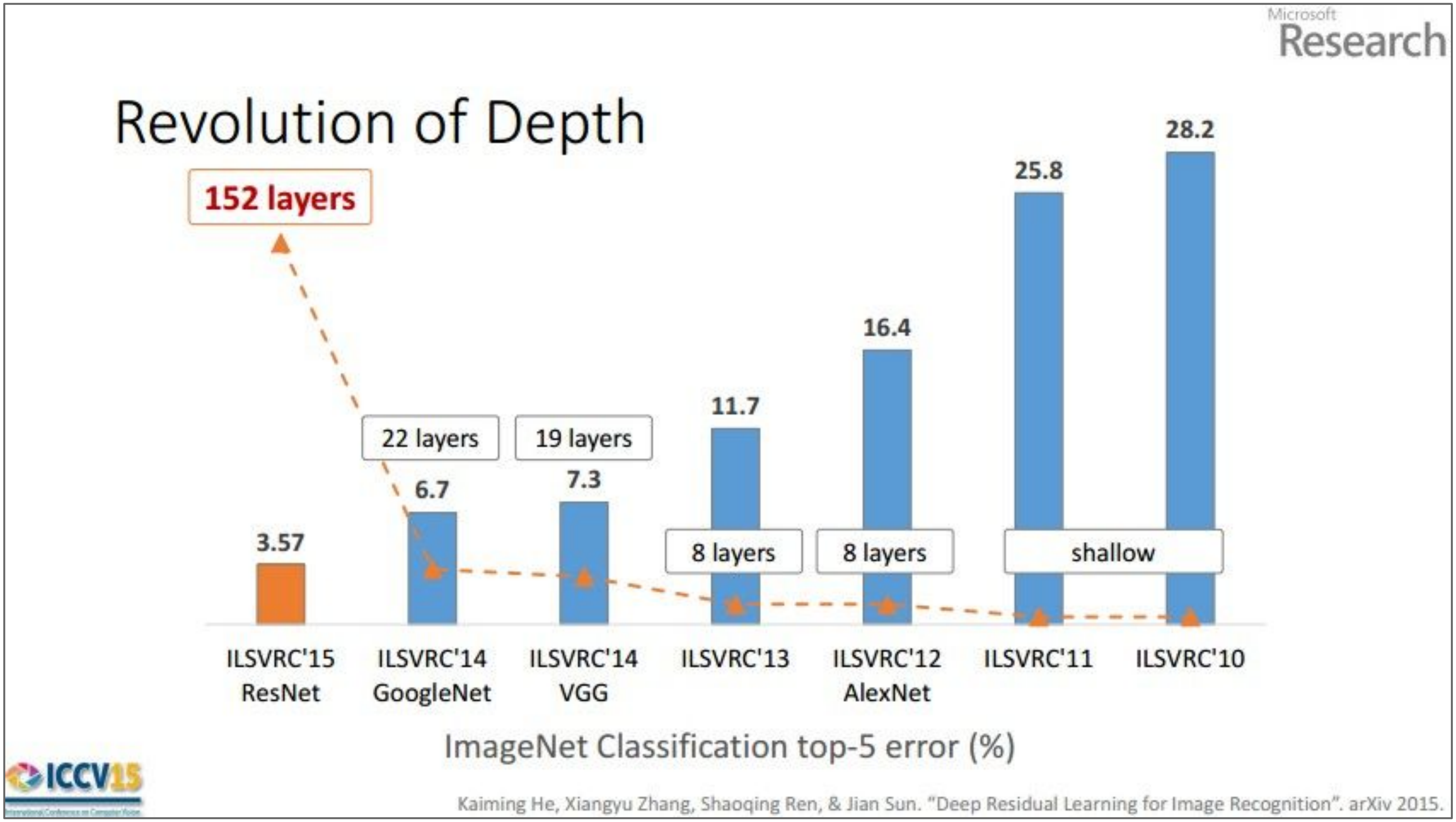
Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



CNNs for Image Recognition

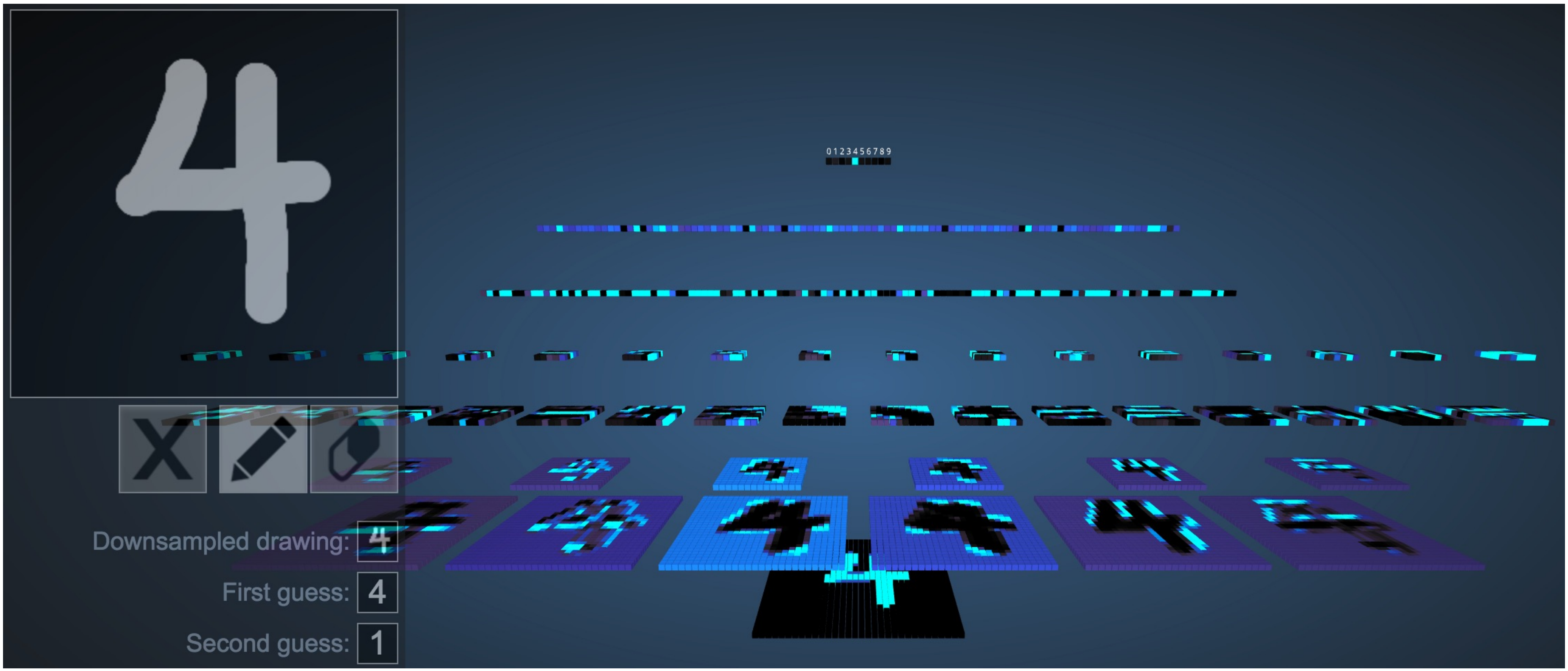


Slide from Kaiming He

CNN VISUALIZATIONS

3D Visualization of CNN

http://scs.ryerson.ca/~aharley/vis/conv/



Convolution of a Color Image

- Color images consist of 3 floats per pixel for RGB (red, green blue) color values
- Convolution must also be 3-dimensional

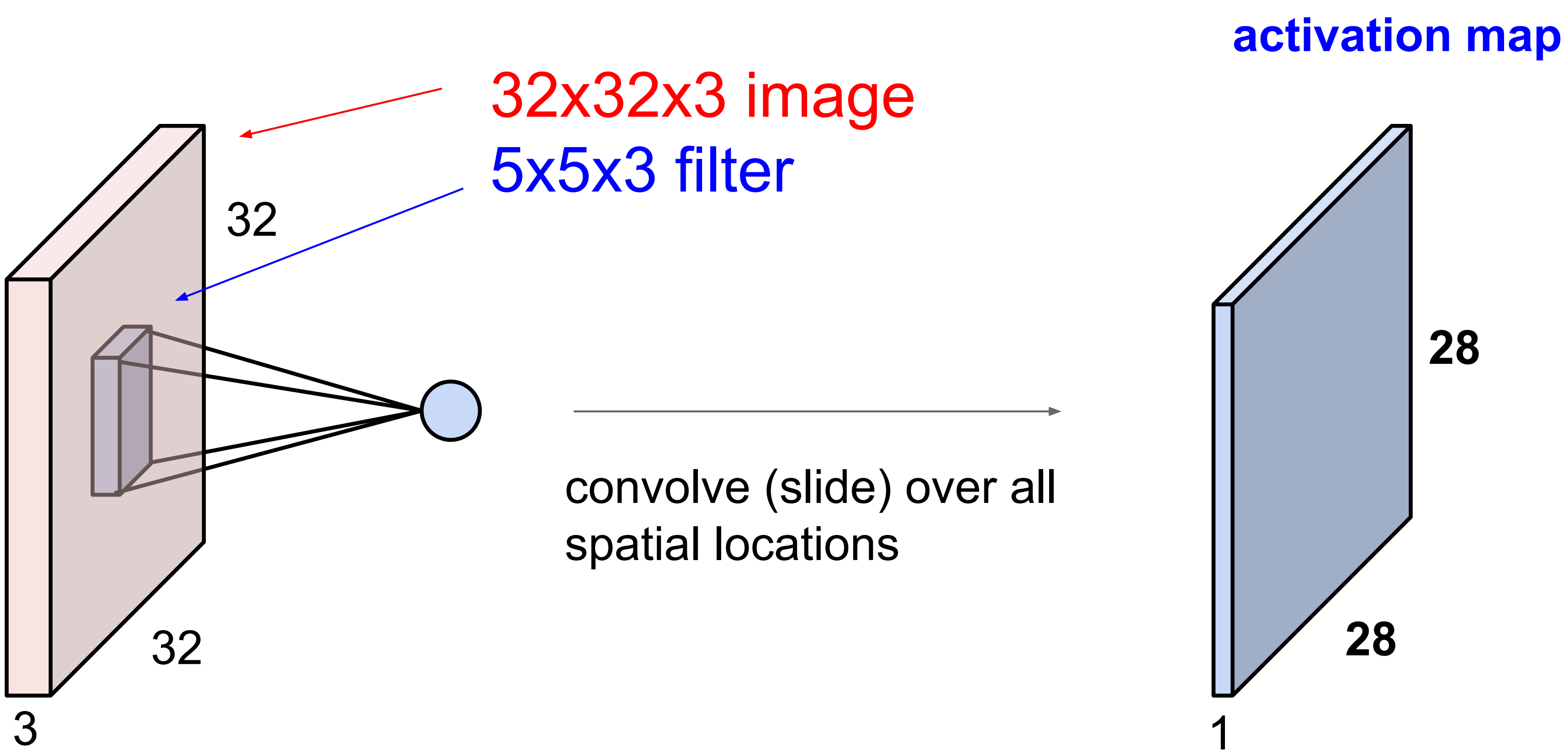


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

Animation of 3D Convolution

http://cs231n.github.io/convolutional-networks/

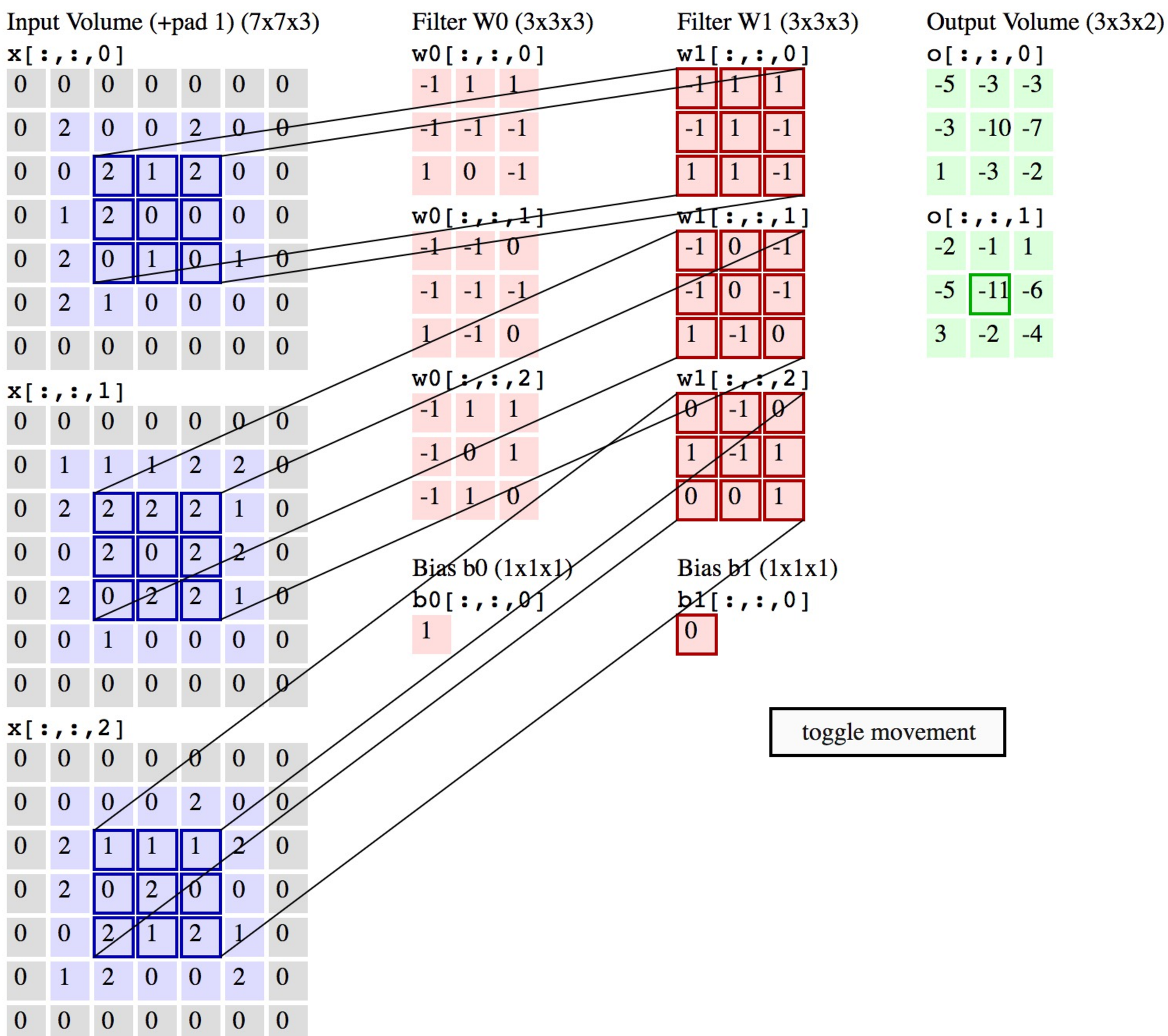


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

MNIST Digit Recognition with CNNs (in your browser)

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

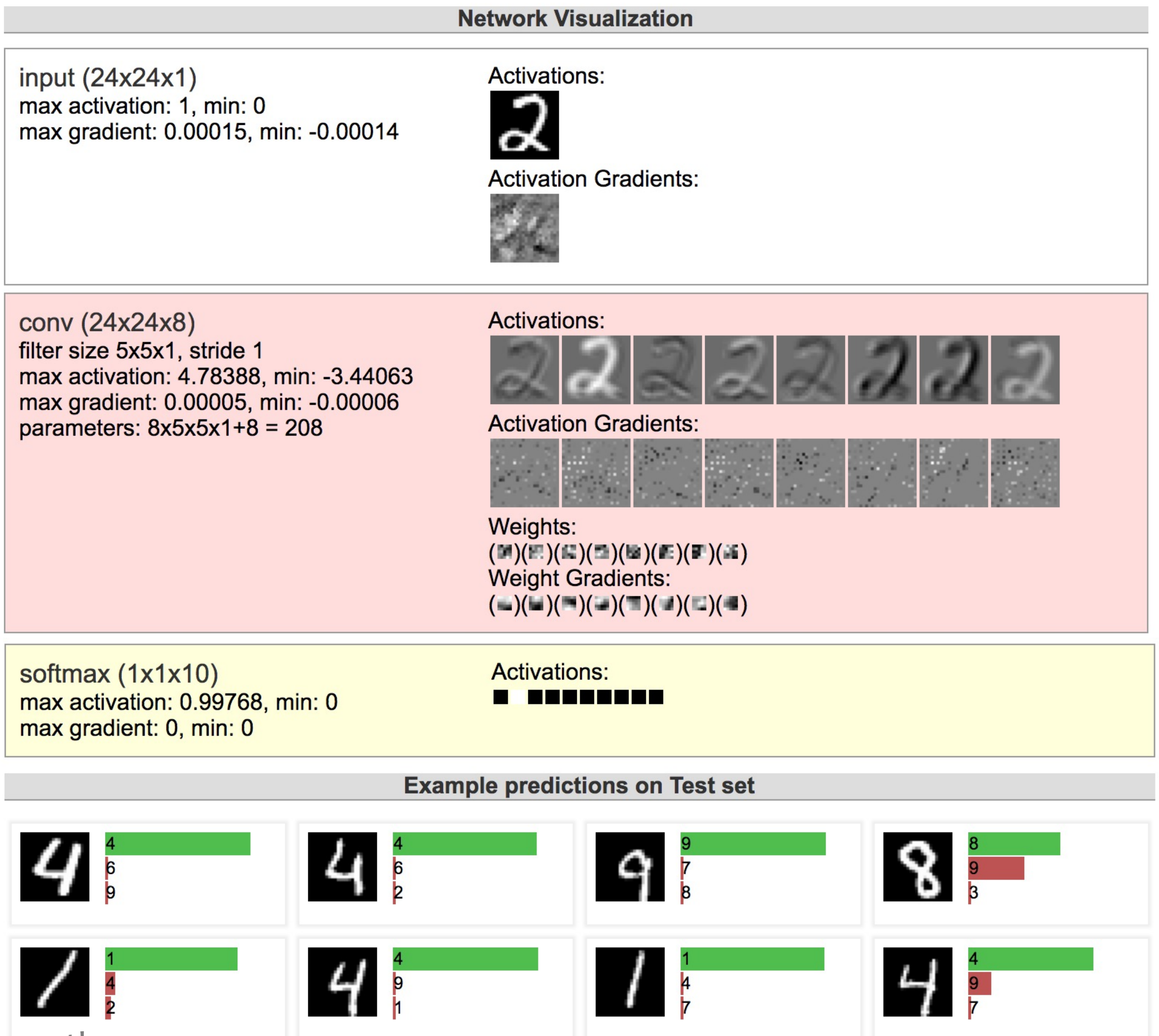


Figure from Andrej Karpathy

CNN Summary

CNNs

– Are used for all aspects of **computer vision**, and have won numerous pattern recognition competitions
– Able learn **interpretable features** at different levels of abstraction
– Typically, consist of **convolution** layers, **pooling** layers, **nonlinearities**, and **fully connected** layers

Other Resources:

– Readings on course website
– Andrej Karpathy, CS231n Notes
http://cs231n.github.io/convolutional-networks/

Deep Learning Objectives

You should be able to…

- Implement the common layers found in Convolutional Neural Networks (CNNs) such as linear layers, convolution layers, max-pooling layers, and rectified linear units (ReLU)
- Explain how the shared parameters of a convolutional layer could learn to detect spatial patterns in an image
- Describe the backpropagation algorithm for a CNN
- Identify the parameter sharing used in a basic recurrent neural network, e.g. an Elman network
- Apply a recurrent neural network to model sequence data
- Differentiate between an RNN and an RNN-LM

ML Big Picture

Learning Paradigms:

What data is available and when? What form of prediction?

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Problem Formulation:

What is the structure of our output prediction?

boolean	Binary Classification
categorical	Multiclass Classification
ordinal	Ordinal Classification
real	Regression
ordering	Ranking
multiple discrete	Structured Prediction
multiple continuous	(e.g. dynamical systems)
both discrete & cont.	(e.g. mixed graphical models)

Application Areas

Key challenges?

NLP, Speech, Computer Vision, Robotics, Medicine, Search

Theoretical Foundations:

What principles guide learning?

- ❑ probabilistic
- ❑ information theoretic
- ❑ evolutionary search
- ❑ ML as optimization

Facets of Building ML Systems:

How to build systems that are robust, efficient, adaptive, effective?

1. *Data prep*
2. *Model selection*
3. *Training (optimization / search)*
4. *Hyperparameter tuning on validation data*
5. *(Blind) Assessment on test data*

Big Ideas in ML:

Which are the ideas driving development of the field?

- *inductive bias*
- *generalization / overfitting*
- *bias-variance decomposition*
- *generative vs. discriminative*
- *deep nets, graphical models*
- *PAC learning*
- *distant rewards*

LEARNING THEORY

PAC(-MAN) Learning

For some hypothesis $h \in \mathcal{H}$:

1. True Error

$$R(h)$$

2. Training Error

$$\hat{R}(h)$$

Question 2:
What is the expected number of PAC-MAN levels Matt will complete before a **Game-Over**?

A. 1-10
B. 11-20
C. 21-30

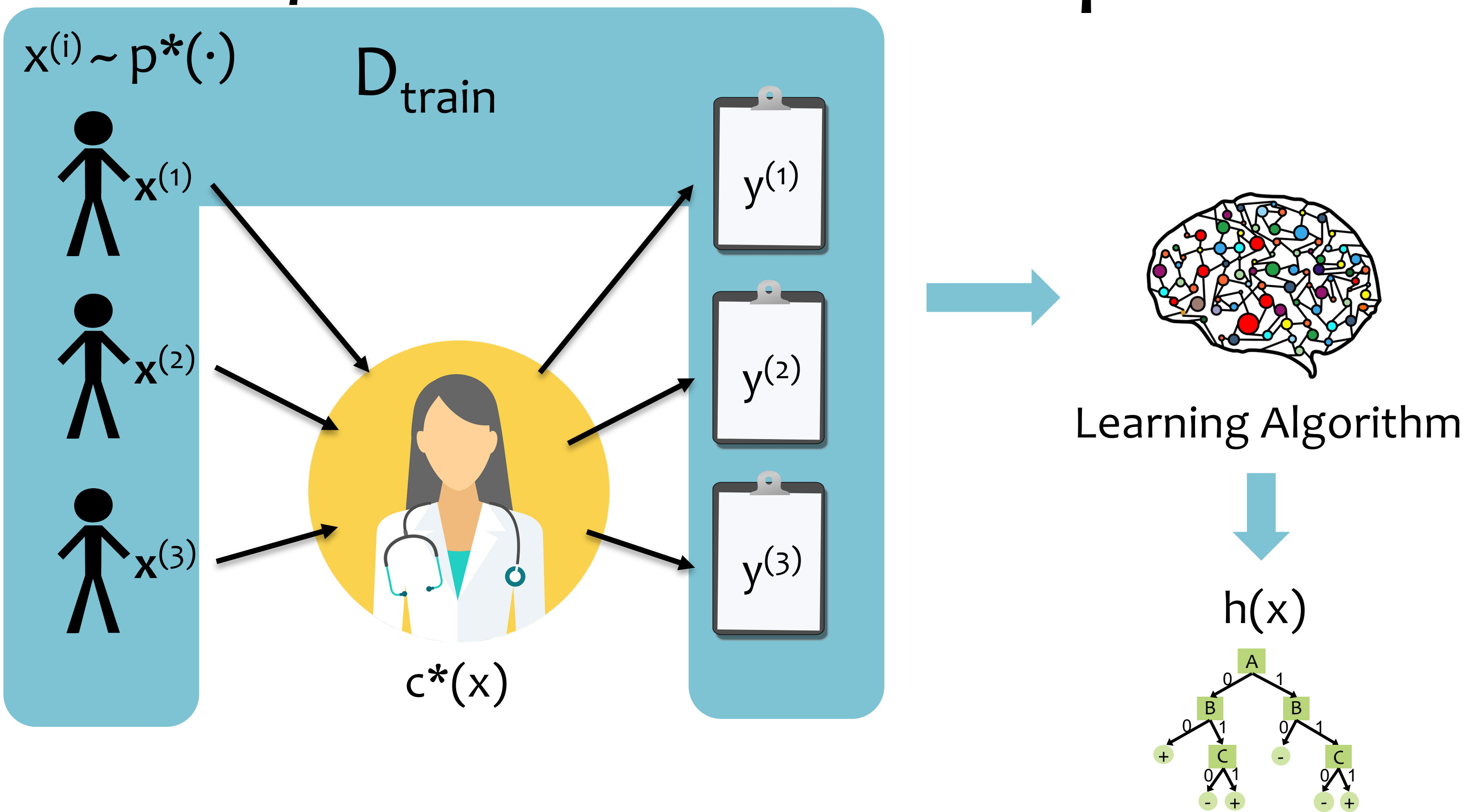
Questions for today (and next lecture)

1. Given a classifier with **zero training error**, what can we say about **true error** (aka. generalization error)?
   (Sample Complexity, Realizable Case)

2. Given a classifier with **low training error**, what can we say about **true error** (aka. generalization error)?
   (Sample Complexity, Agnostic Case)

3. Is there a **theoretical justification for regularization** to avoid overfitting?
   (Structural Risk Minimization)

PAC/SLT Model for Supervised ML



PAC/SLT Model for Supervised ML

- **Problem Setting**
  - Set of possible inputs, $\mathbf{x} \in \mathcal{X}$ (all possible patients)
  - Set of possible outputs, $y \in \mathcal{Y}$ (all possible diagnoses)
  - Distribution over instances, $p^*(\cdot)$
  - Exists an unknown target function, $c^* : \mathcal{X} \rightarrow \mathcal{Y}$ (the doctor's brain)
  - Set, $\mathcal{H}$, of candidate hypothesis functions, $h : \mathcal{X} \rightarrow \mathcal{Y}$ (all possible decision trees)
- **Learner is given** N training examples $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$ where $x^{(i)} \sim p^*(\cdot)$ and $y^{(i)} = c^*(\mathbf{x}^{(i)})$ (history of patients and their diagnoses)
- **Learner produces** a hypothesis function, $\hat{y} = h(x)$, that best approximates unknown target function $y = c^*(x)$ on the training data

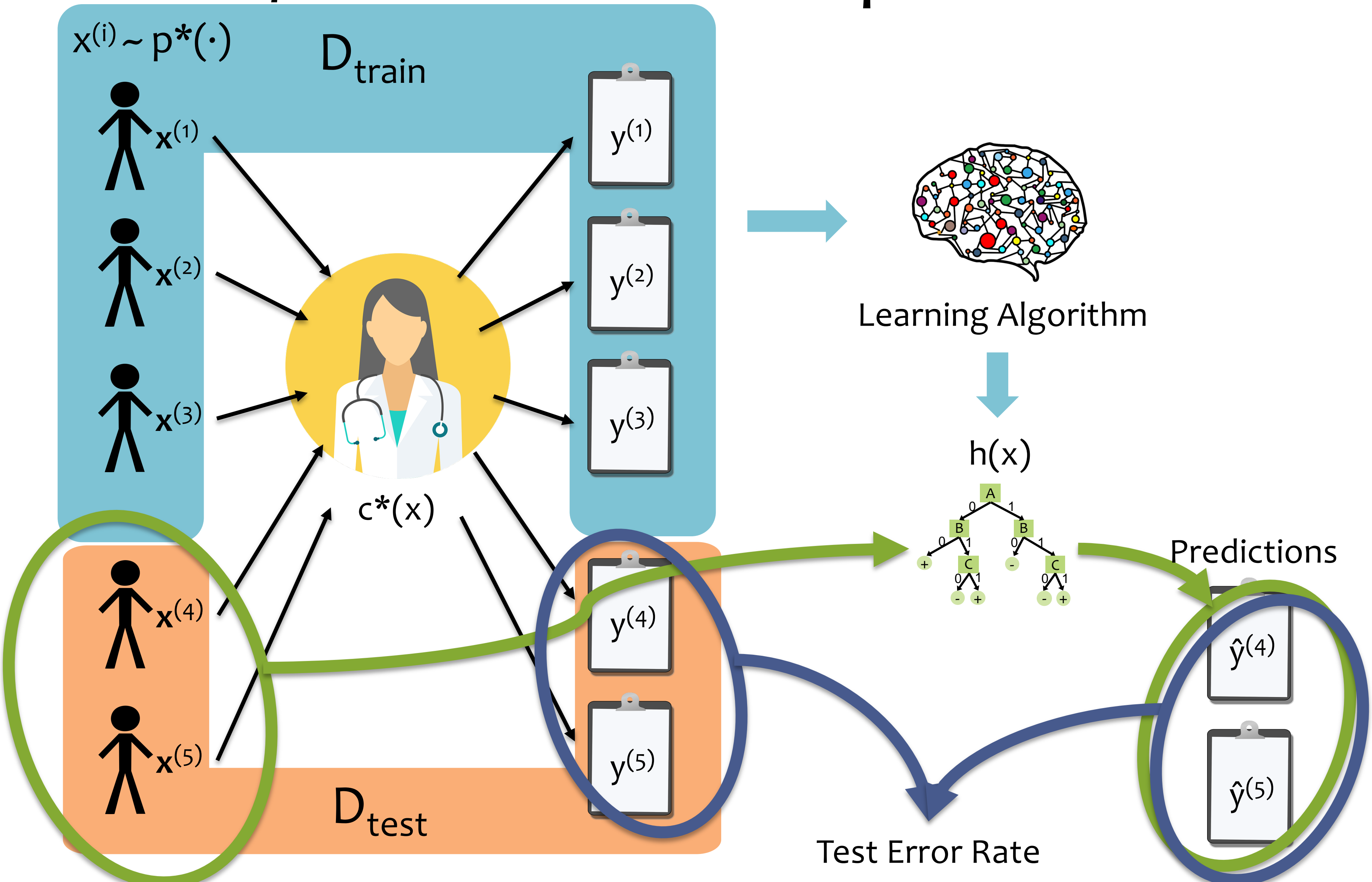
PAC/SLT Model for Supervised ML

- **Problem Setting**
  - Set of possible inputs, $\mathbf{x} \in \mathcal{X}$ (all possible patients)
  - Set of possible outputs, $y \in \mathcal{Y}$ (all possible diagnoses)
  - Distribution over instances, $p^*(\cdot)$
  - Exists an unknown target function, $c^* : \mathcal{X} \rightarrow \mathcal{Y}$
    (the doctor's brain
  - Set, $\mathcal{H}$, of candida
    (all possible decisi
- **Learner is given** N t
  D = $\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)},$
  where $x^{(i)} \sim p^*(\cdot)$ an
  (history of patients
- **Learner produces** a
  best approximates
  the training data

Two important settings we'll consider:

1. **Classification:** the possible outputs are **discrete**
2. **Regression:** the possible outputs are **real-valued**

PAC/SLT Model for Supervised ML



Two Types of Error

1. True Error (aka. **expected risk**)

$$R(h) = P_{\mathbf{x} \sim p^*(\mathbf{x})}(c^*(\mathbf{x}) \neq h(\mathbf{x}))$$

This quantity is always **unknown**

2. Train Error (aka. **empirical risk**)

$$\begin{aligned}\hat{R}(h) &= P_{\mathbf{x} \sim \mathcal{S}}(c^*(\mathbf{x}) \neq h(\mathbf{x})) \\ &= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(c^*(\mathbf{x}^{(i)}) \neq h(\mathbf{x}^{(i)})) \\ &= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y^{(i)} \neq h(\mathbf{x}^{(i)}))\end{aligned}$$

We can **measure** this on the training data

where $\mathcal{S} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})\}_{i=1}^{N}$ is the training data set, and $\mathbf{x} \sim \mathcal{S}$ denotes that $\mathbf{x}$ is sampled from the empirical distribution.

PAC / SLT Model

We've also referred to this as the "Function Approximation View"

1. Generate instances from *unknown* distribution p^*

$$\mathbf{x}^{(i)} \sim p^*(\mathbf{x}), \forall i \tag{1}$$

2. Oracle labels each instance with *unknown* function c^*

$$y^{(i)} = c^*(\mathbf{x}^{(i)}), \forall i \tag{2}$$

3. Learning algorithm chooses hypothesis $h \in \mathcal{H}$ with low(est) training error, $\hat{R}(h)$

$$\hat{h} = \underset{h}{\operatorname{argmin}}\, \hat{R}(h) \tag{3}$$

4. Goal: Choose an h with low generalization error $R(h)$

Three Hypotheses of Interest

The **true function** c^* is the one we are trying to learn and that labeled the training data:

$$y^{(i)} = c^*(\mathbf{x}^{(i)}), \; \forall i \tag{1}$$

The **expected risk minimizer** has lowest true error:

$$h^* = \operatorname*{argmin}_{h \in \mathcal{H}} R(h)$$

Question:
True or False:
h* and c* are always equal.

The **empirical risk minimizer** has lowest training error:

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \hat{R}(h) \tag{3}$$

PAC LEARNING

Probably Approximately Correct (PAC) Learning

Whiteboard:

- PAC Criterion
- Meaning of “Probably Approximately Correct”
- Def: PAC Learner
- Sample Complexity
- Consistent Learner
- Realizable vs. Agnostic Cases
- Finite vs. Infinite Hypothesis Spaces

SAMPLE COMPLEXITY RESULTS

Sample Complexity Results

Definition 0.1. The **sample complexity** of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e. close to 1).

Four Cases we care about…

We'll start with the finite case…

	Realizable	Agnostic
Finite $\|\mathcal{H}\|$		
Infinite $\|\mathcal{H}\|$		

Probably Approximately Correct (PAC) Learning

Whiteboard:

- Theorem 1: Realizable Case, Finite |H|
- Proof of Theorem 1

Sample Complexity Results

Definition 0.1. The **sample complexity** of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e. close to 1).

Four Cases we care about…

	Realizable	Agnostic
Finite $\lvert\mathcal{H}\rvert$	**Thm. 1** $N \geq \frac{1}{\epsilon}\left[\log(\lvert\mathcal{H}\rvert) + \log(\frac{1}{\delta})\right]$ labeled examples are sufficient so that with probability $(1-\delta)$ all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$.	
Infinite $\lvert\mathcal{H}\rvert$		

Example: Conjunctions

Question:

Suppose H = class of conjunctions over **x** in $\{0,1\}^M$

Example hypotheses:

$h(\mathbf{x}) = x_1 (1-x_3) x_5$

$h(\mathbf{x}) = x_1 (1-x_2) x_4 (1-x_5)$

If M = 10, ε = 0.1, δ = 0.01, how many examples suffice according to Theorem 1?

Answer:

A. 10*(2*ln(10)+ln(100)) ≈ 92
B. 10*(3*ln(10)+ln(100)) ≈ 116
C. 10*(10*ln(2)+ln(100)) ≈ 116
D. 10*(10*ln(3)+ln(100)) ≈ 156
E. 100*(2*ln(10)+ln(10)) ≈ 691
F. 100*(3*ln(10)+ln(10)) ≈ 922
G. 100*(10*ln(2)+ln(10)) ≈ 924
H. 100*(10*ln(3)+ln(10)) ≈ 1329

Thm. 1 $N \geq \frac{1}{\epsilon}\left[\log(|\mathcal{H}|) + \log(\frac{1}{\delta})\right]$ labeled examples are sufficient so that with probability $(1-\delta)$ all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$.