10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Convolutional Neural Networks (CNNs) + Recurrent Neural Networks (RNNs)

Matt Gormley & Geoff Gordon

Lecture 17

Oct. 27, 2025

Reminders

- **Homework 6: Learning Theory & Generative Models**
  - Out: Mon, Oct 27
  - Due: Sat, Nov 01 at 11:59pm

THE BIG PICTURE

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*

Classification and Regression: The Big Picture

Recipe for Machine Learning

1. Given data $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N$
2. (a) Choose a decision function $h_{\boldsymbol{\theta}}(\mathbf{x}) = \cdots$ (parameterized by $\boldsymbol{\theta}$)
   (b) Choose an objective function $J_{\mathcal{D}}(\boldsymbol{\theta}) = \cdots$ (relies on data)
3. Learn by choosing parameters that optimize the objective $J_{\mathcal{D}}(\boldsymbol{\theta})$

$$\hat{\boldsymbol{\theta}} \approx \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J_{\mathcal{D}}(\boldsymbol{\theta})$$

4. Predict on new test example $\mathbf{x}_{\text{new}}$ using $h_{\boldsymbol{\theta}}(\cdot)$

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}_{\text{new}})$$

Decision Functions

- Perceptron: $h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T\mathbf{x})$
- Linear Regression: $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x}$
- Discriminative Models: $h_{\boldsymbol{\theta}}(\mathbf{x}) = \underset{y}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$
  - Logistic Regression: $p_{\boldsymbol{\theta}}(y = 1 \mid \mathbf{x}) = \sigma(\boldsymbol{\theta}^T\mathbf{x})$
  - Neural Net (classification): $p_{\boldsymbol{\theta}}(y = 1 \mid \mathbf{x}) = \sigma((\mathbf{W}^{(2)})^T\sigma((\mathbf{W}^{(1)})^T\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$
- Generative Models: $h_{\boldsymbol{\theta}}(\mathbf{x}) = \underset{y}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(\mathbf{x}, y)$
  - Naive Bayes: $p_{\boldsymbol{\theta}}(\mathbf{x}, y) = p_{\boldsymbol{\theta}}(y)\prod_{m=1}^{M} p_{\boldsymbol{\theta}}(x_m \mid y)$

Optimization Method

- Gradient Descent: $\boldsymbol{\theta} \to \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
- SGD: $\boldsymbol{\theta} \to \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}J^{(i)}(\boldsymbol{\theta})$
  for $i \sim$ Uniform$(1, \ldots, N)$
  where $J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$
- mini-batch SGD
- closed form
  1. compute partial derivatives
  2. set equal to zero and solve

Objective Function

- MLE: $J(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log p(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
- MCLE: $J(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log p(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)})$
- L2 Regularized: $J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_2^2$
  (same as Gaussian prior $p(\boldsymbol{\theta})$ over parameters)
- L1 Regularized: $J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_1$
  (same as Laplace prior $p(\boldsymbol{\theta})$ over parameters)

Backpropagation and Deep Learning

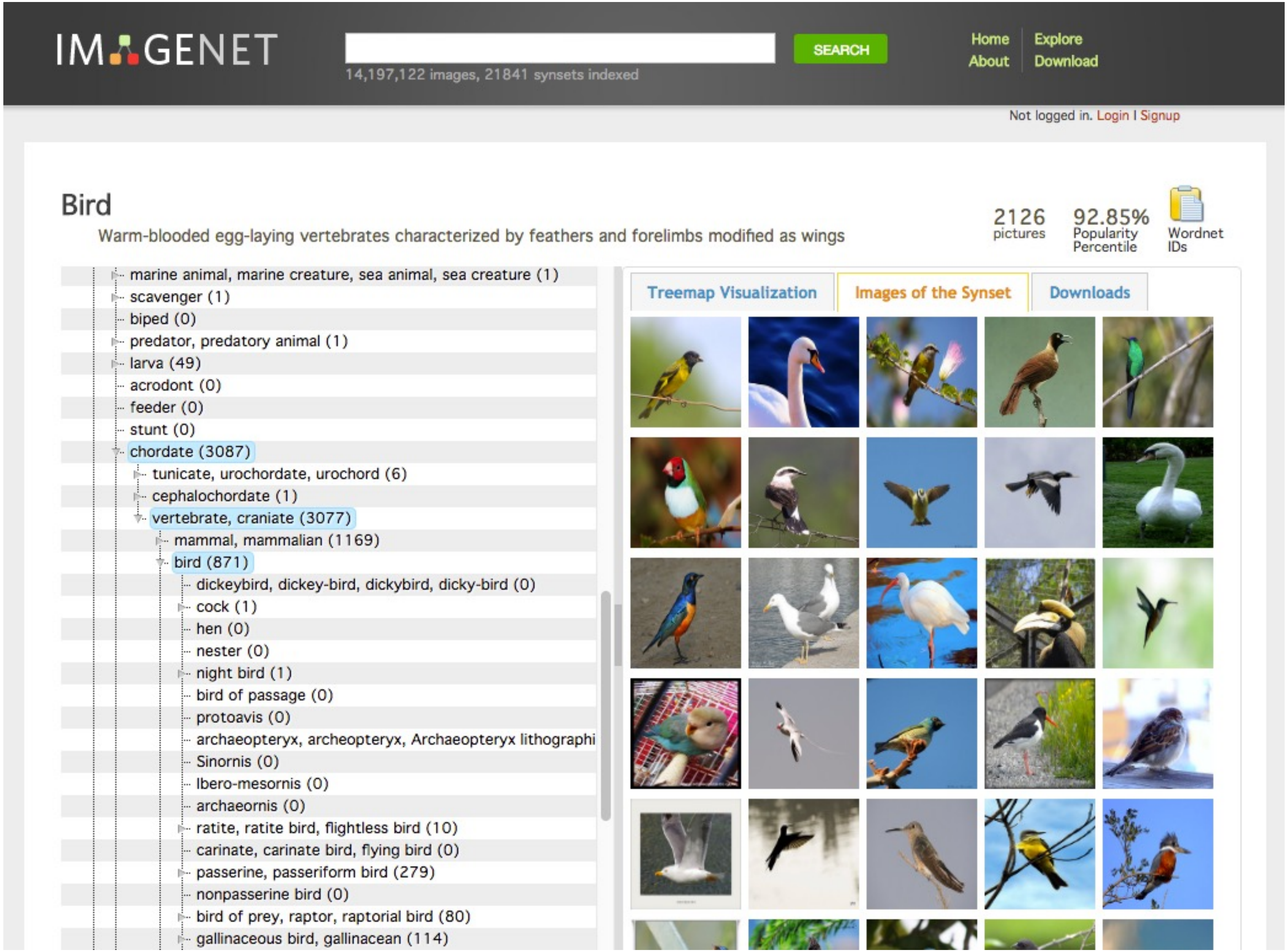
Convolutional neural networks (CNNs) and **recurrent neural networks** (RNNs) are simply fancy computation graphs (aka. hypotheses or decision functions).

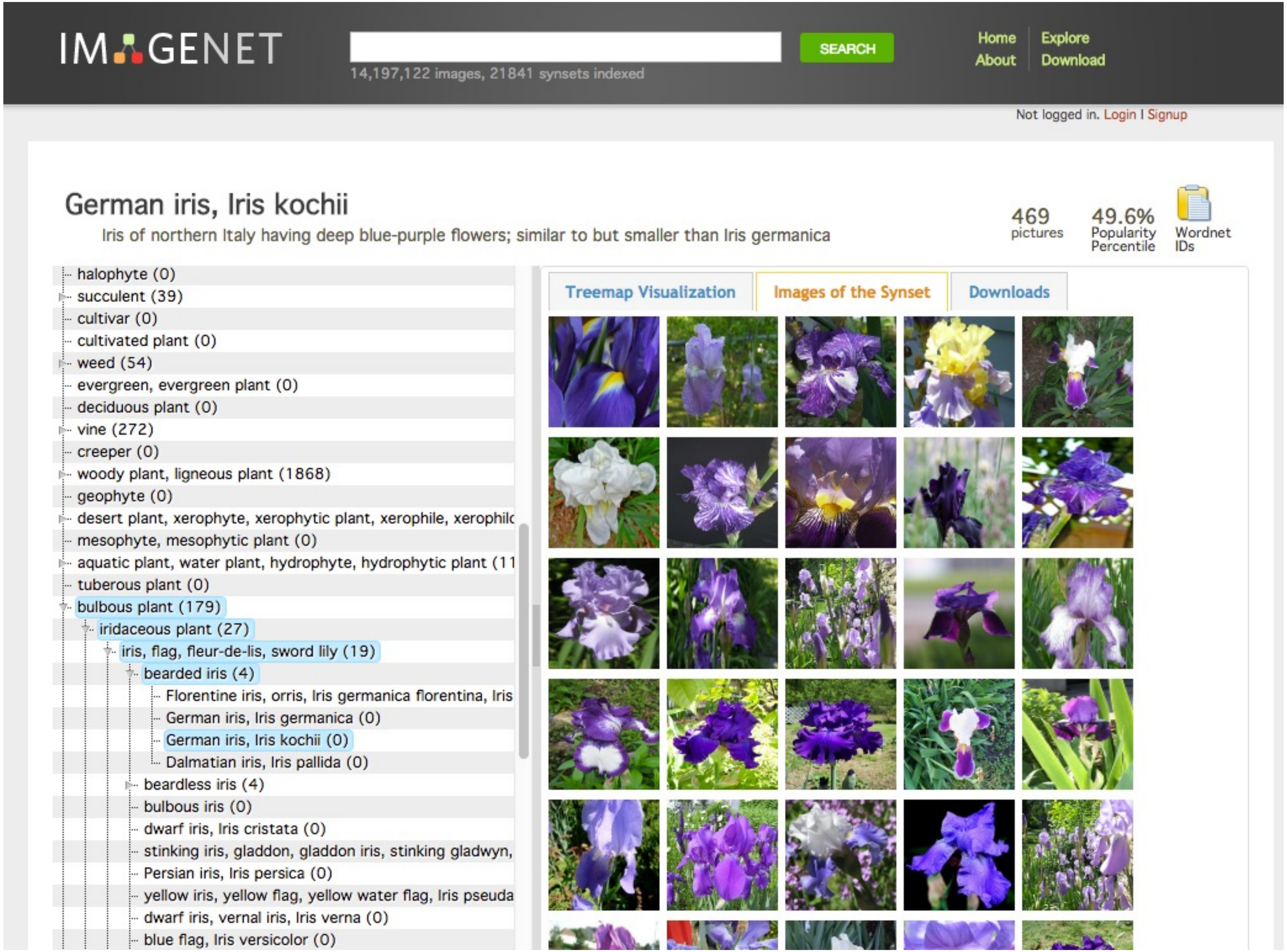
Our recipe also applies to these models and (again) relies on the **backpropagation algorithm** to compute the necessary gradients.

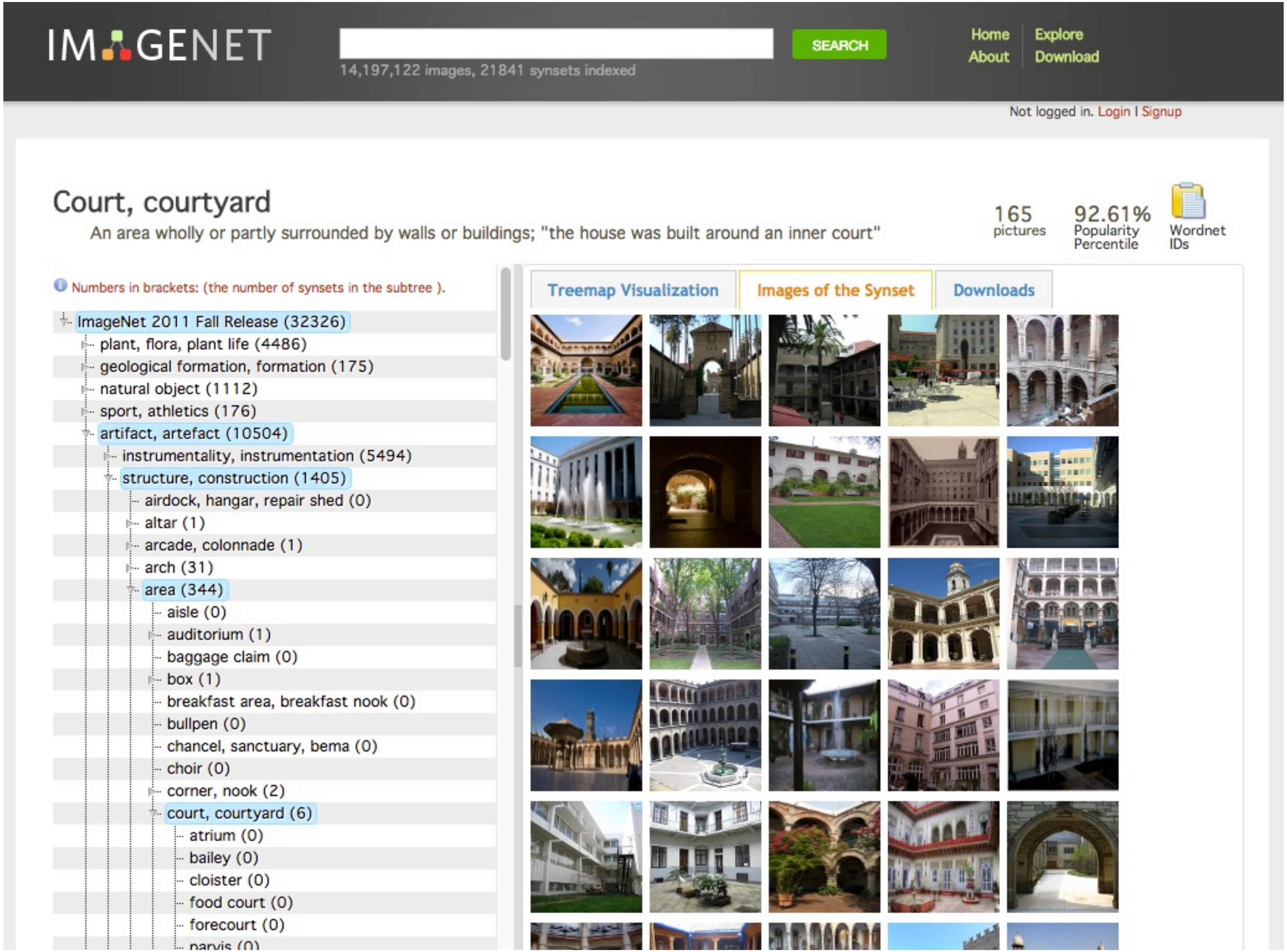
BACKGROUND: COMPUTER VISION

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

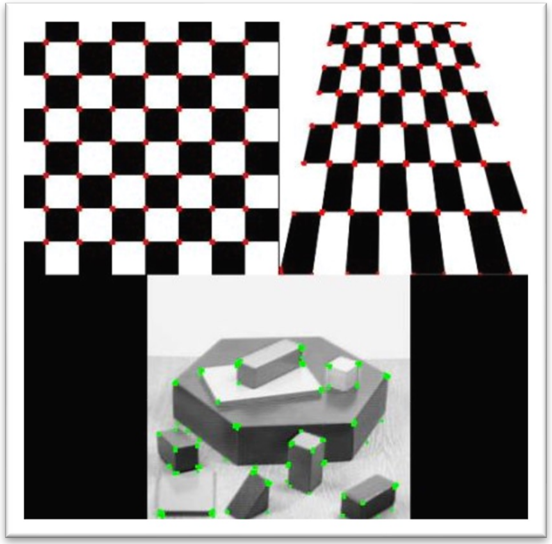

IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica
469
pictures
49.6%
Popularity Percentile
Wordnet IDs
halophyte (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 | 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)



Figures from http://opencv.org

Scale Invariant Feature Transform (SIFT)

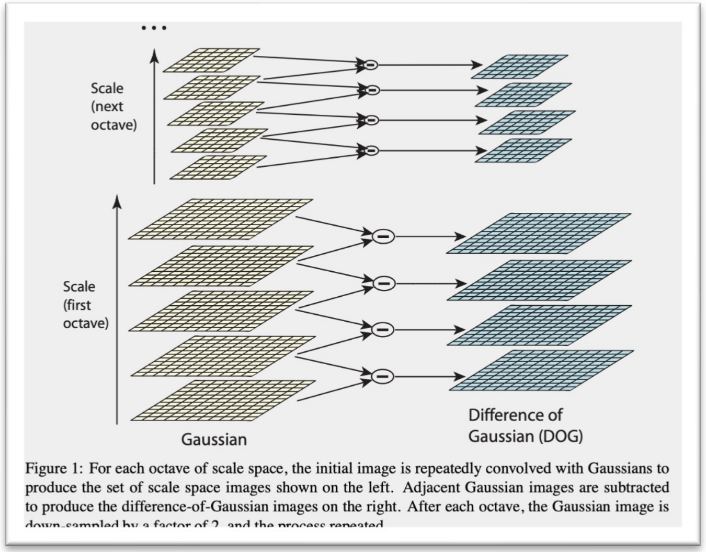


Figure from Lowe (1999) and Lowe (2004)

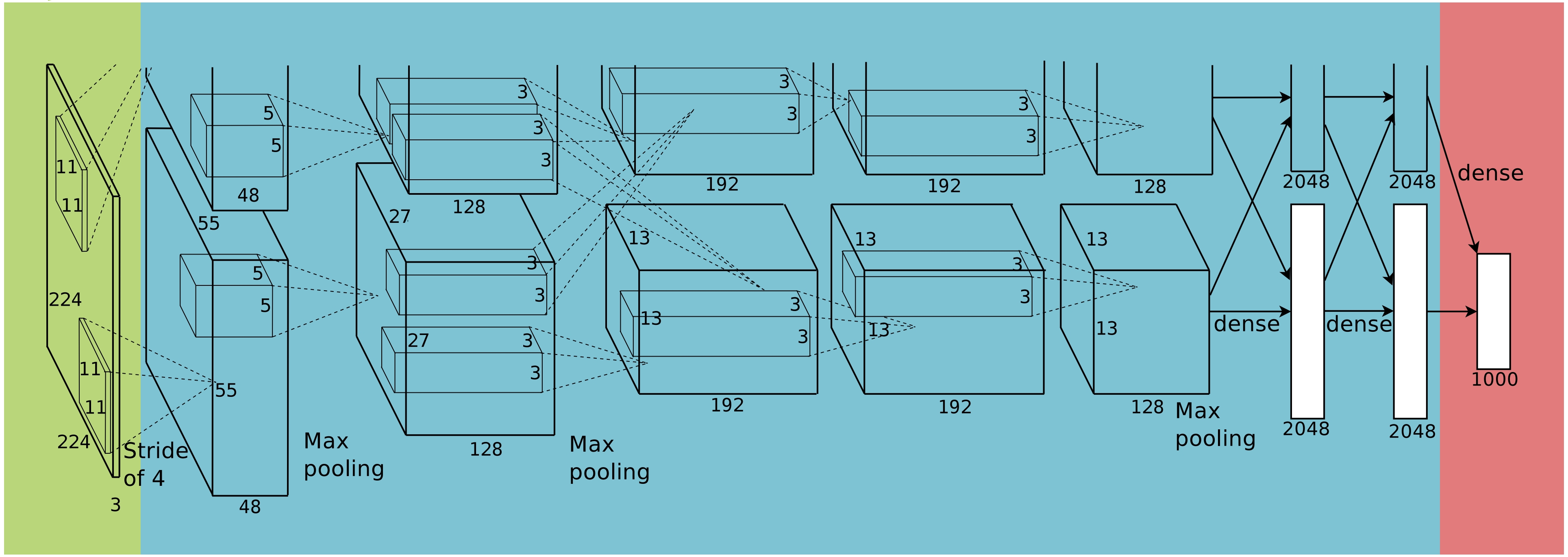
Example: Image Classification

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

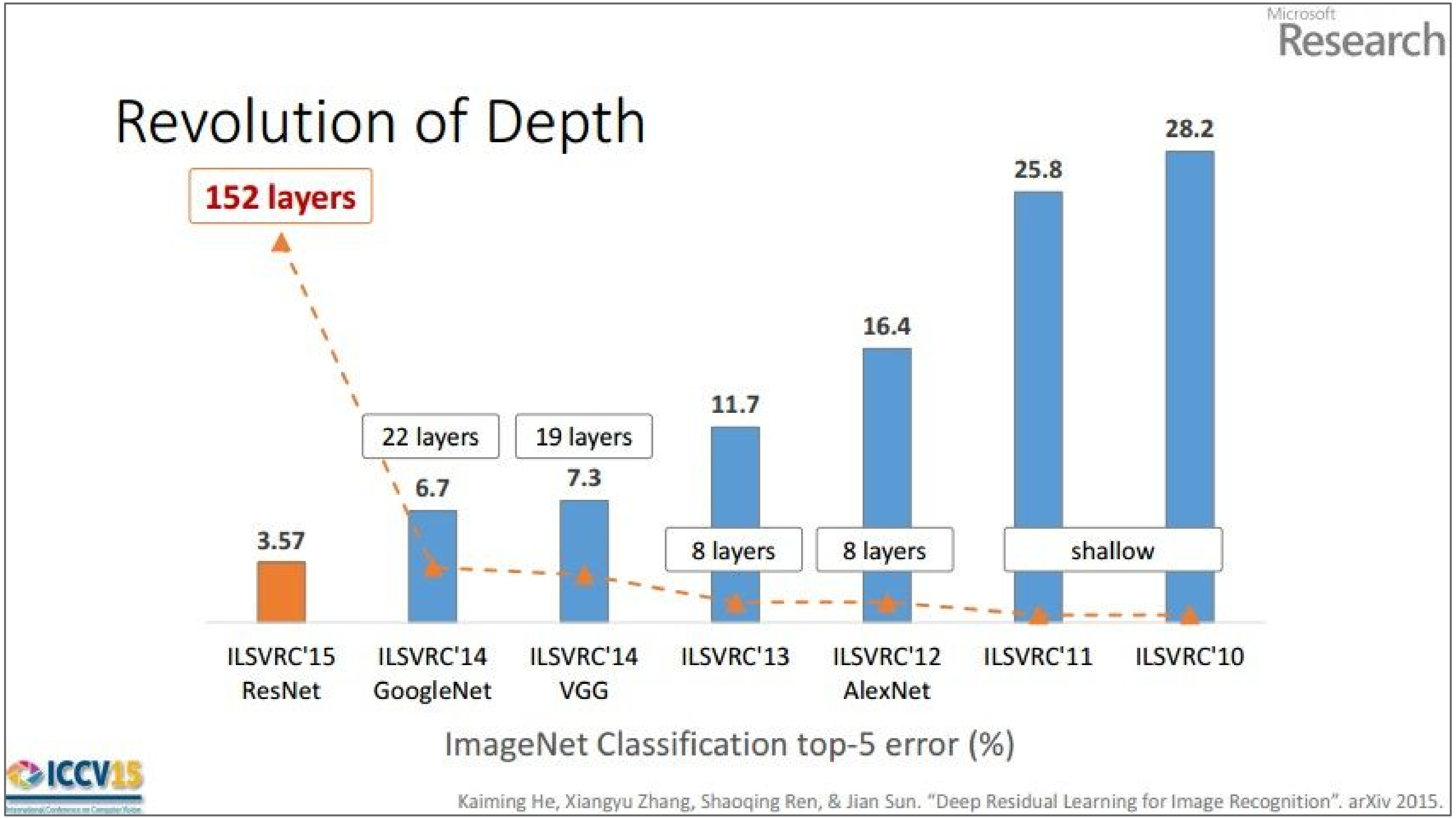
Input image (pixels)

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

1000-way softmax



CNNs for Image Recognition



Slide from Kaiming He

Feed-forward Neural Networks for Computer Vision

Feed-forward Neural Networks for Computer Vision

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

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

2. Choose each of these:
   - Decision function

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x})$$

   - Loss function

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

4. Train with SGD:

(take small steps opposite the gradient)

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

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

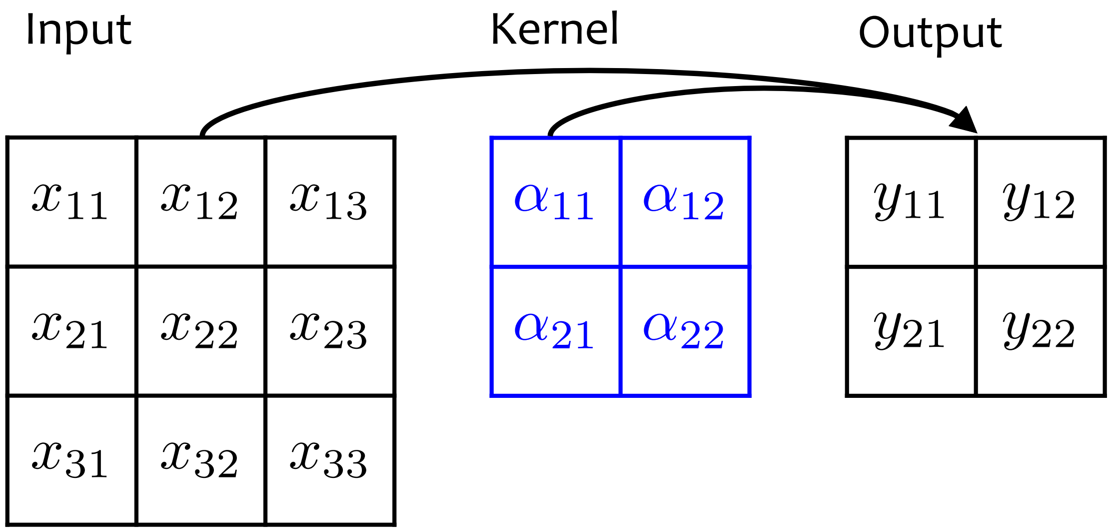
.4	.5	.5	.5	.4
.4	.2	.3	.6	.3
.5	.4	.4	.2	.1
.5	.6	.2	.1	0
.4	.3	.1	0	0

CONVOLUTION

2D Convolution

- Basic idea:
  - Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
  - 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

Example: 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

2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

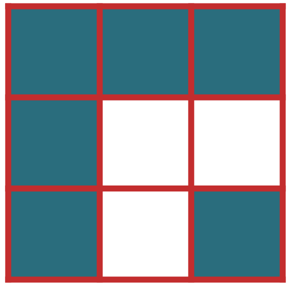
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

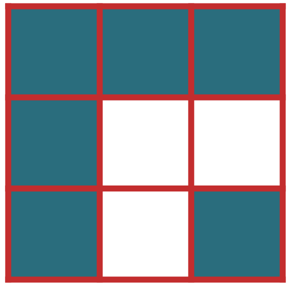
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

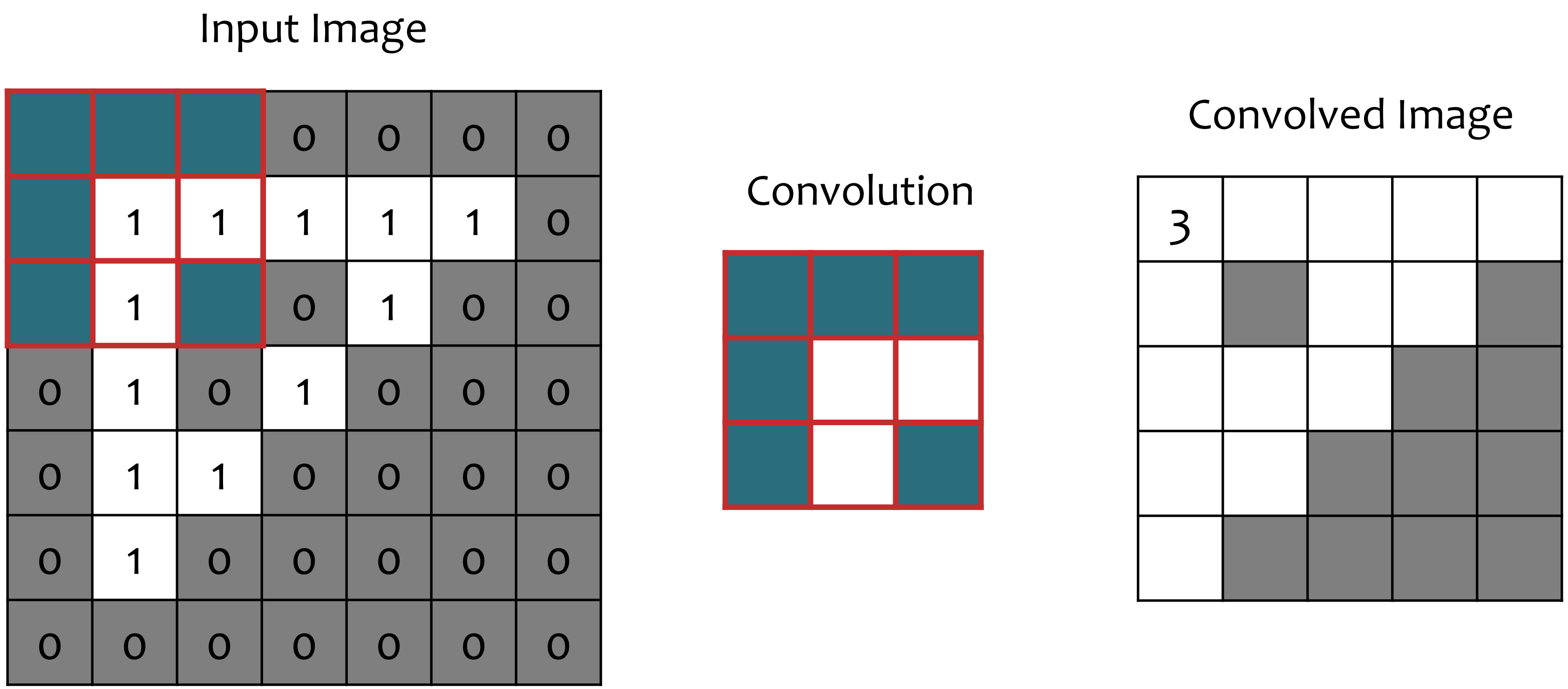


Convolved Image

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

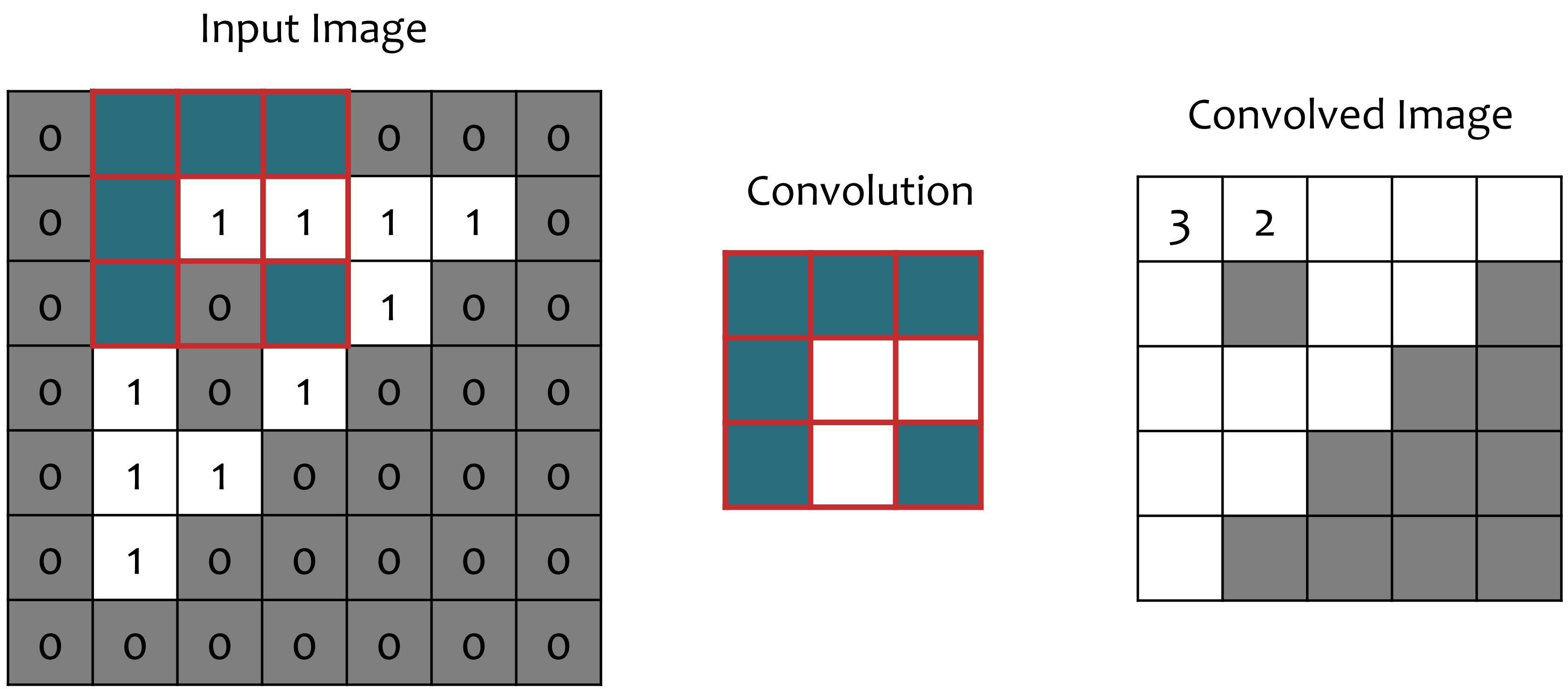
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



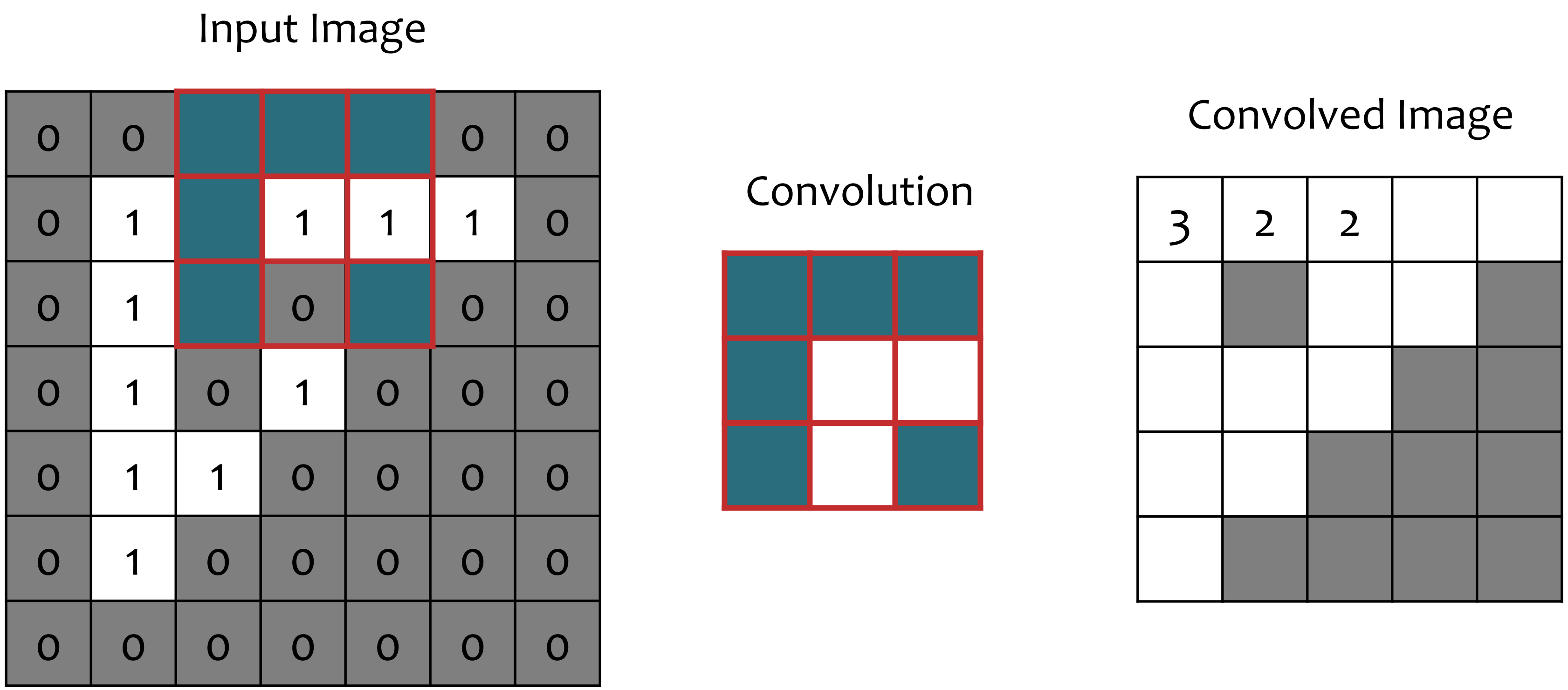
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



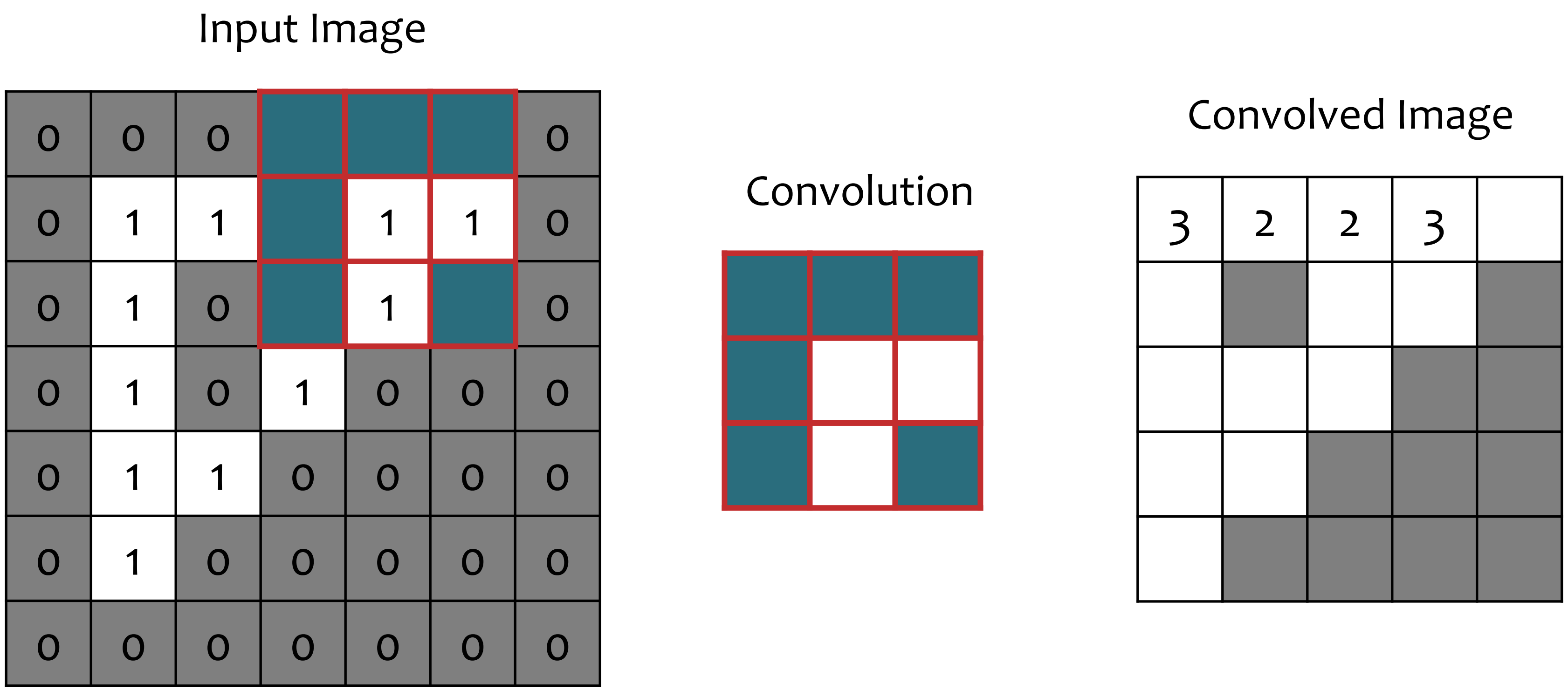
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



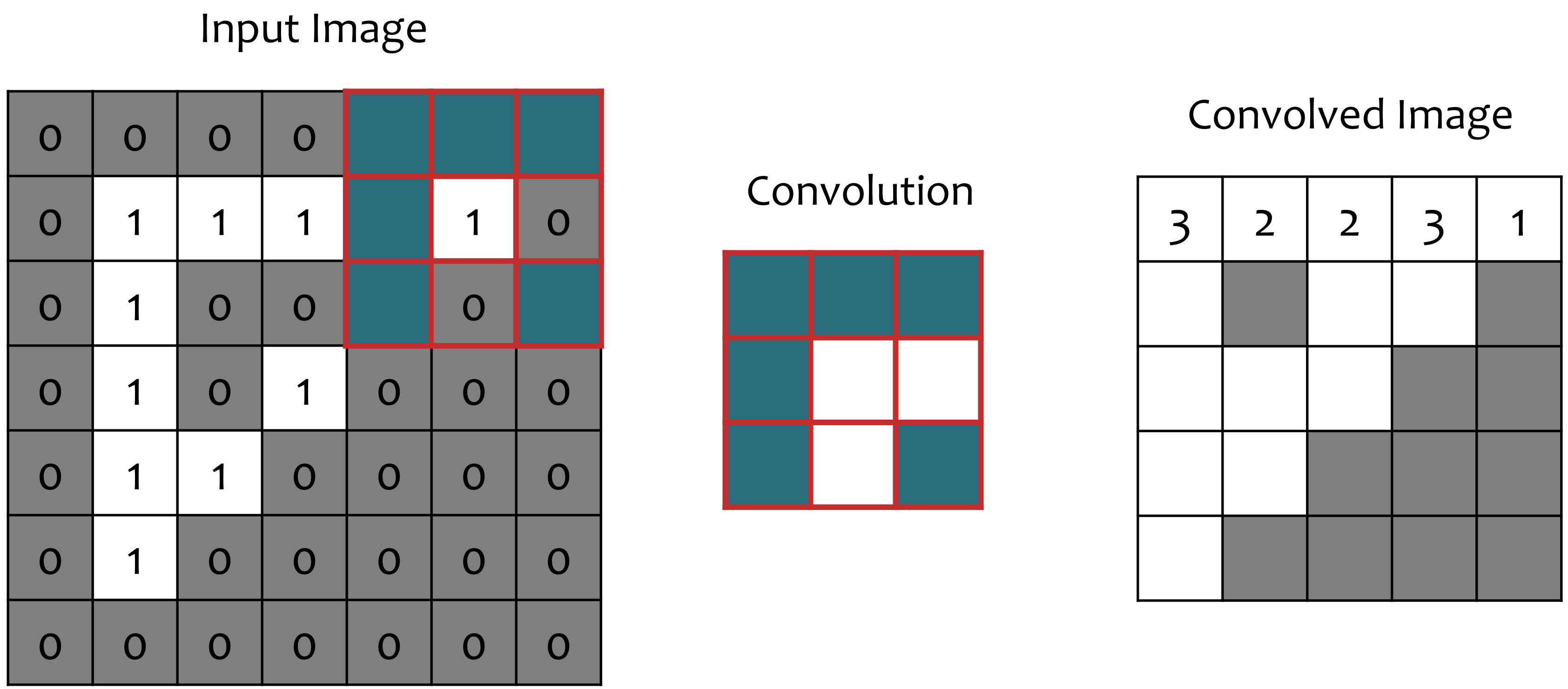
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



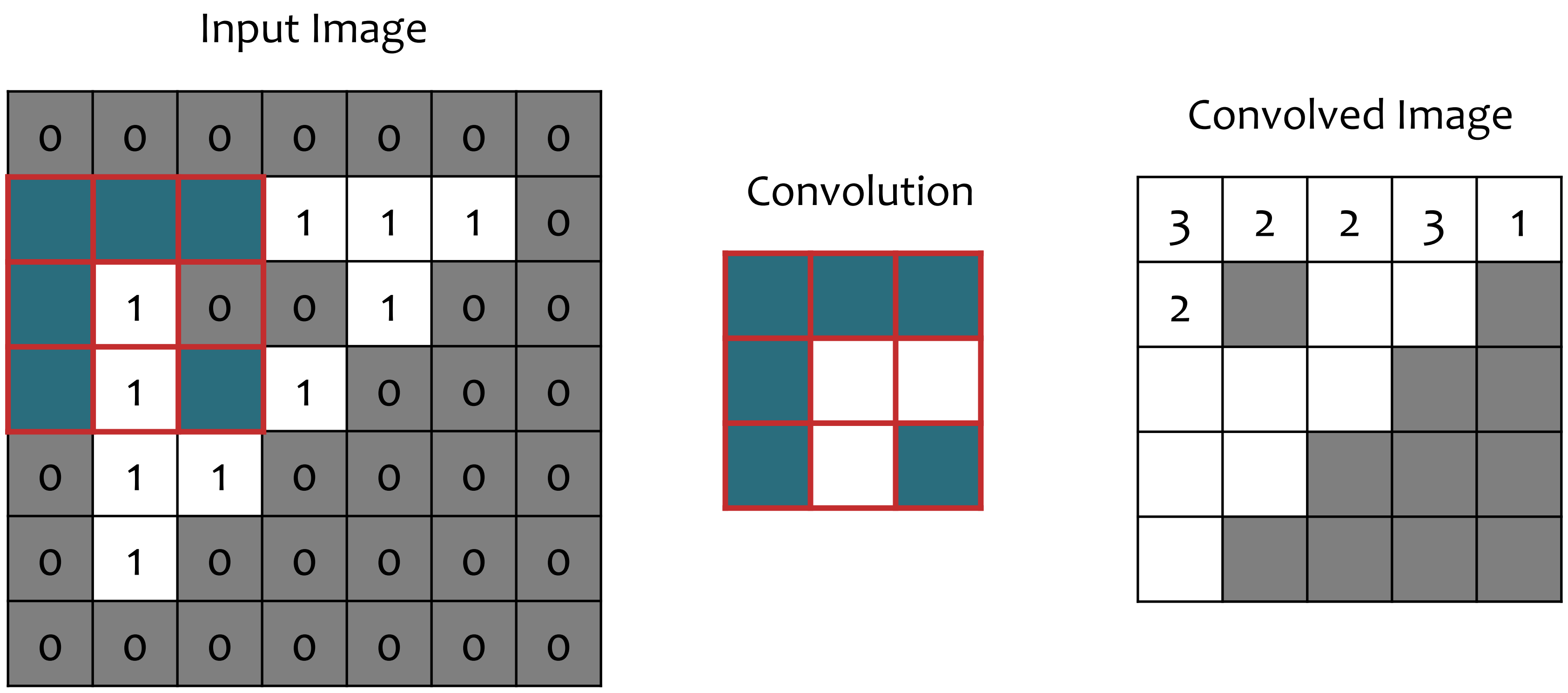
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



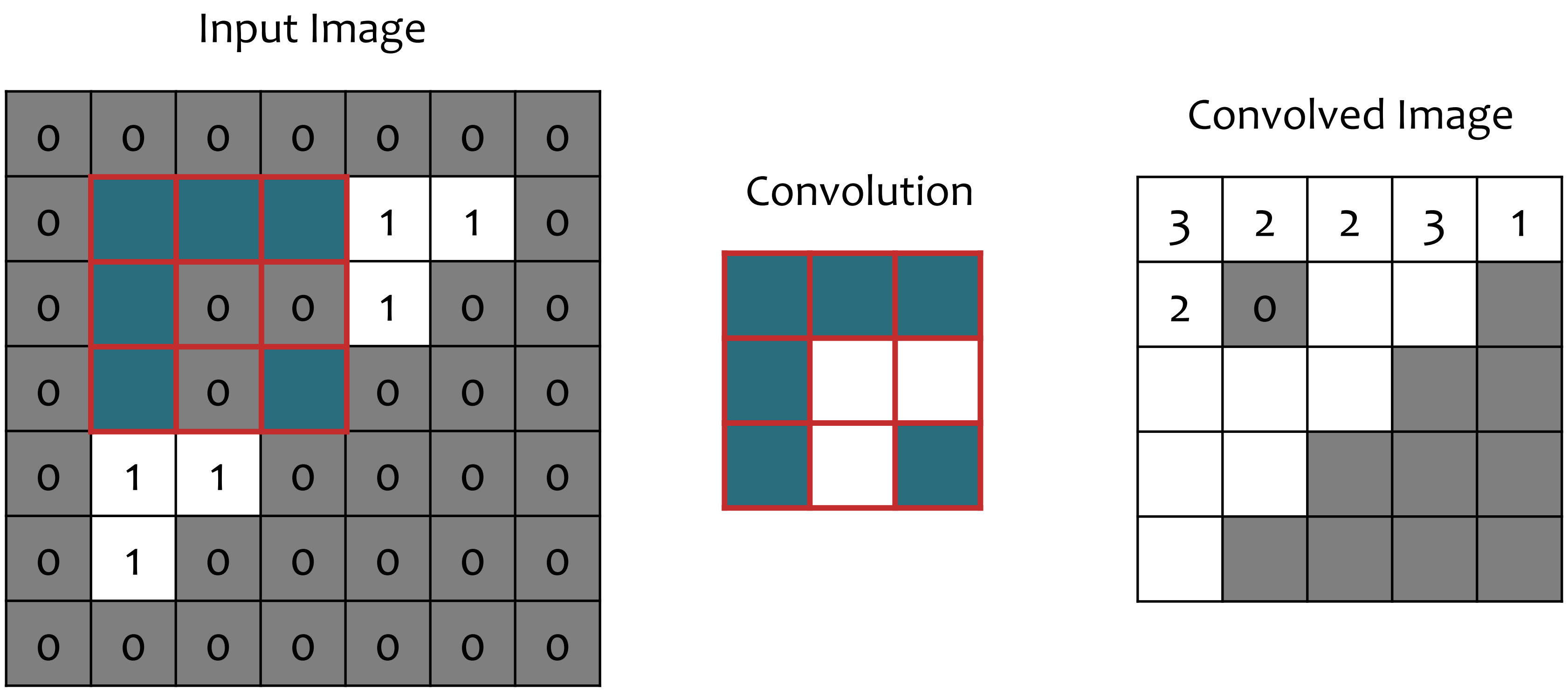
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



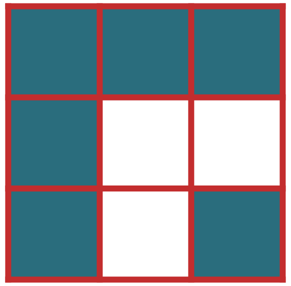
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

PADDING

Padding

Suppose you want to preserve the size of the original input image in your convolved image.

You can accomplish this by padding your input image with zeros.

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

Padding

Suppose you want to preserve the size of the original input image in your convolved image.

You can accomplish this by padding your input image with zeros.

Input Image

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

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

Kernels for Image Processing

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

Input Image

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

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

Kernels for Image Processing

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

Input Image

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

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

Kernels for Image Processing

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

Input Image

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

Vertical Edge Detector

-1	0	1
-1	0	1
-1	0	1

Convolved Image

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

Kernels for Image Processing

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

Input Image

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

Horizontal Edge Detector

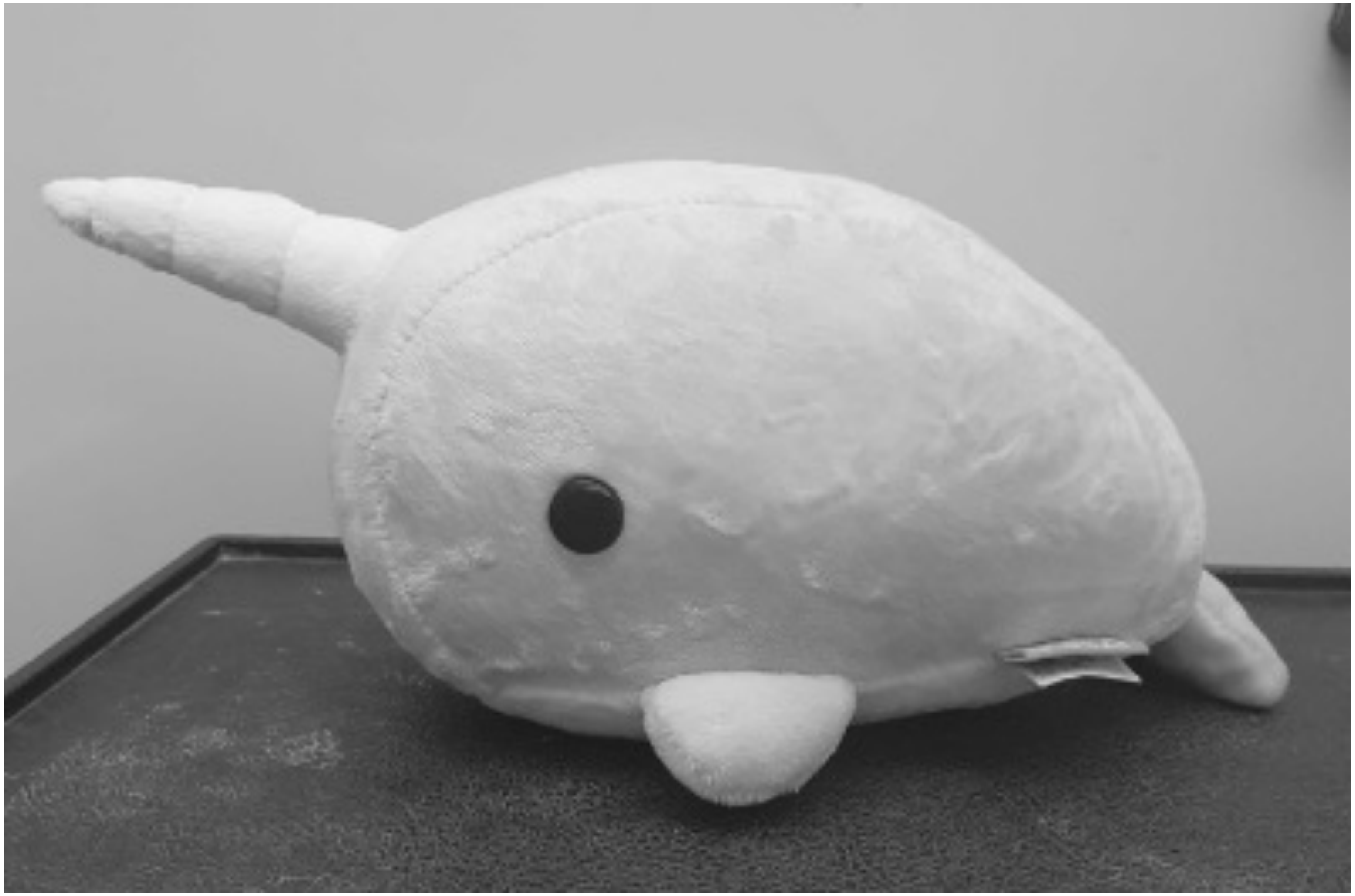
-1	-1	-1
0	0	0
1	1	1

Convolved Image

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

Convolution Examples

Original Image



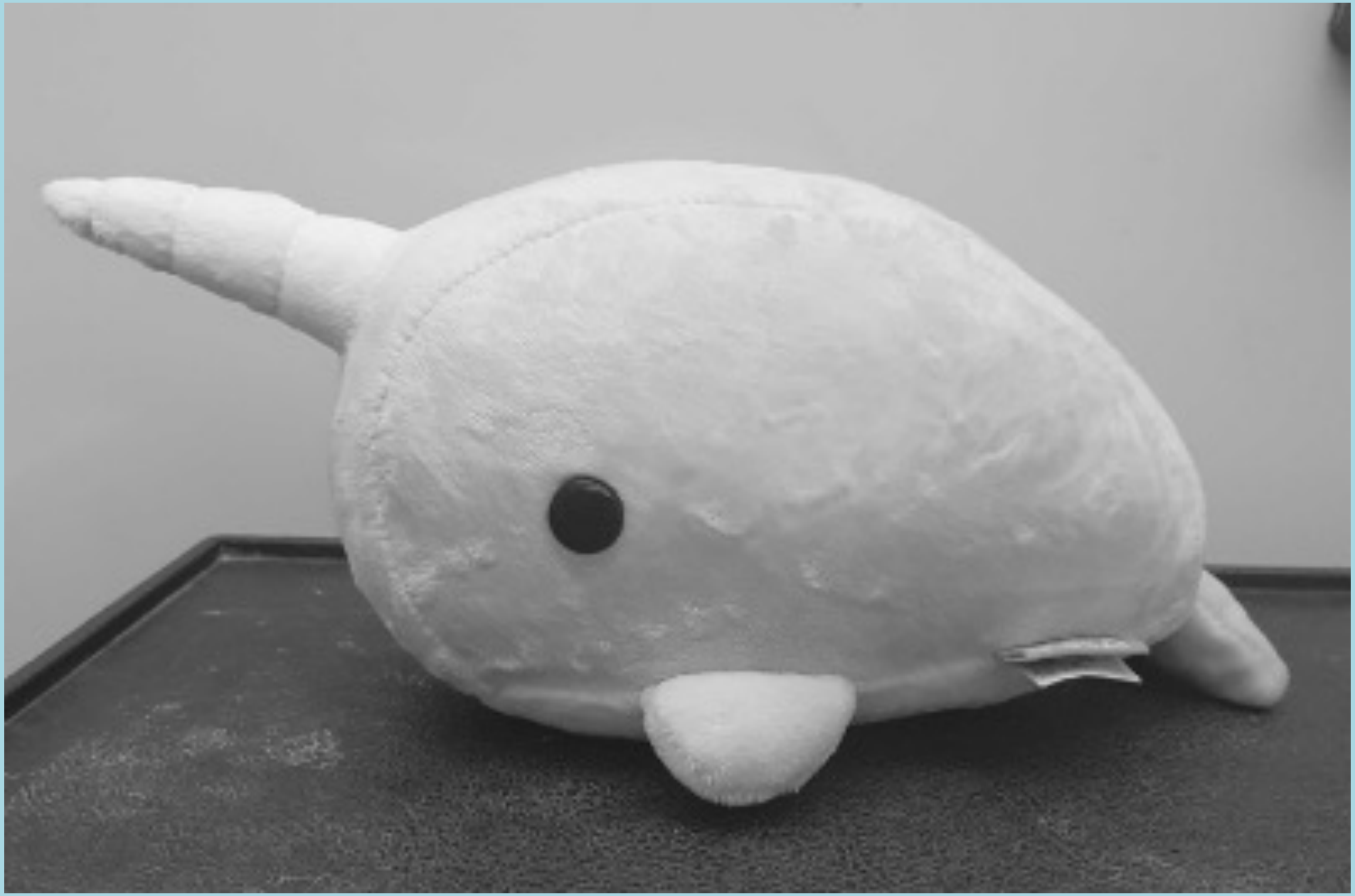
Convolution Examples

Poll Question 1:

What effect do you think the following filter will have on an image?

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

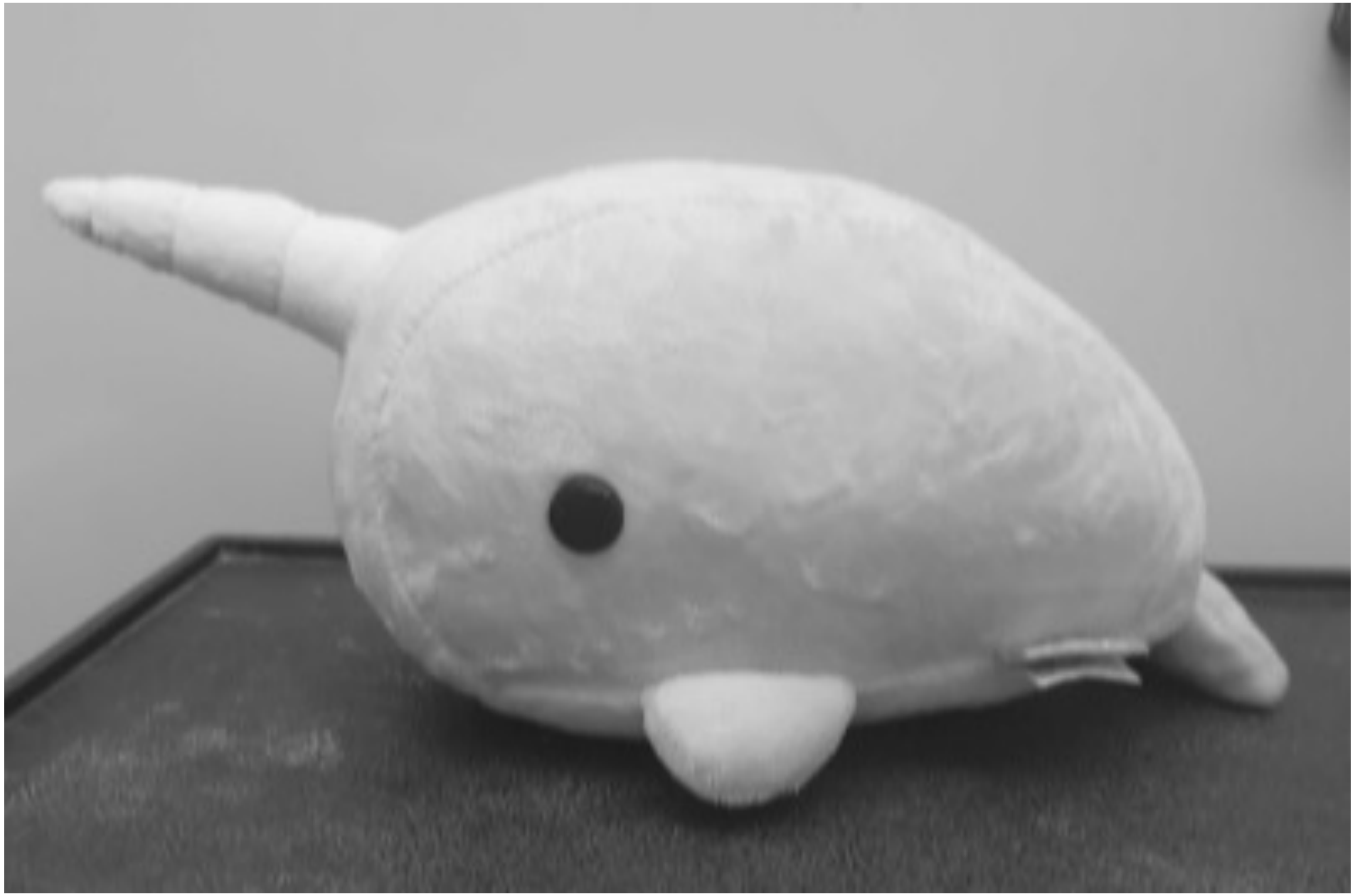
A. Sharpen the image
B. Blur the image
C. Shift the image left
D. Rotate the image clockwise
E. Detect edges
F. Nothing (TOXIC)



Convolution Examples

Gaussian
Blur

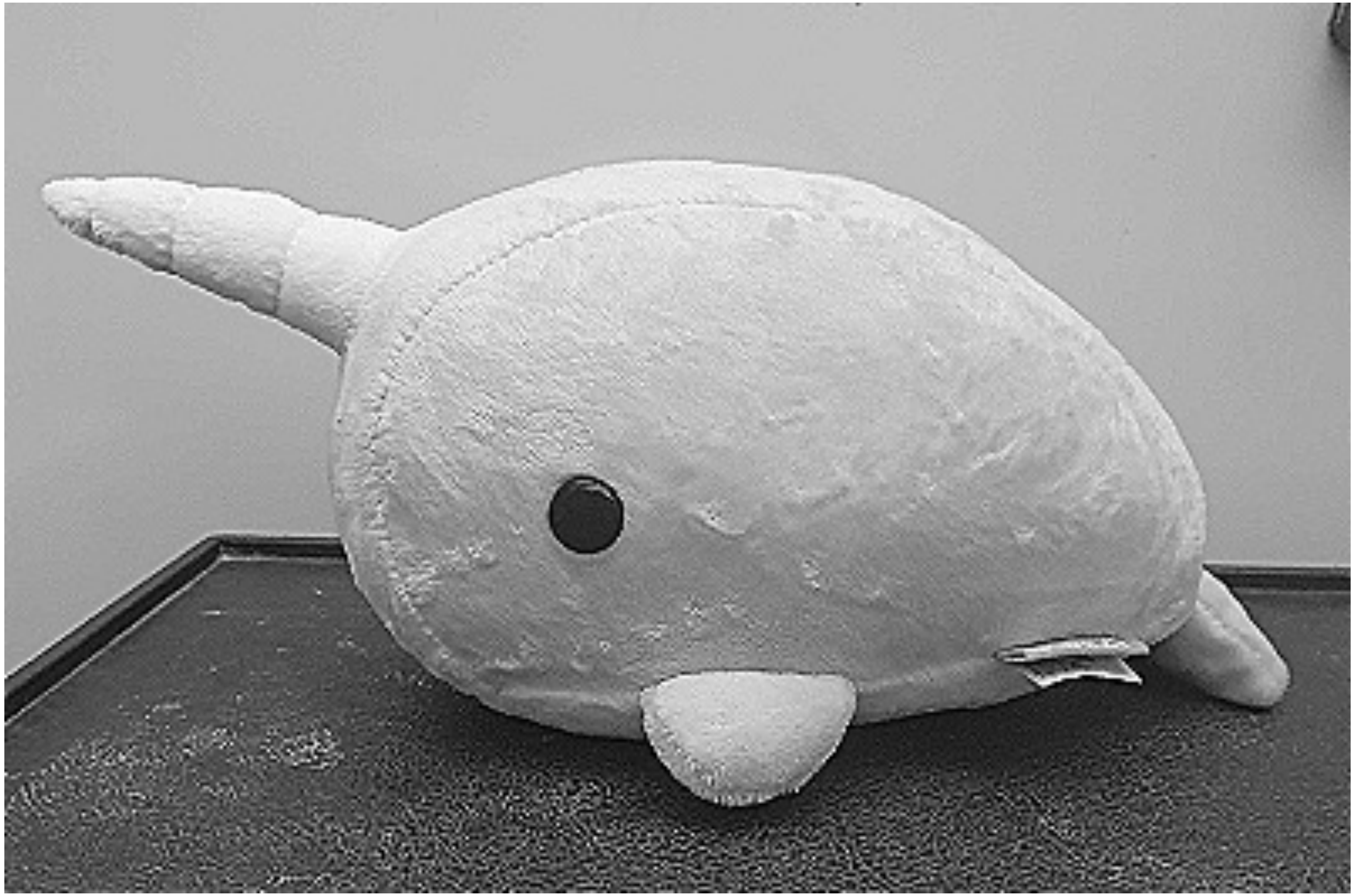
.01	.04	.06	.04	.01
.04	.19	.25	.19	.04
.06	.25	.37	.25	.06
.04	.19	.25	.19	.04
.01	.04	.06	.04	.01



Convolution Examples

Sharpening
Kernel

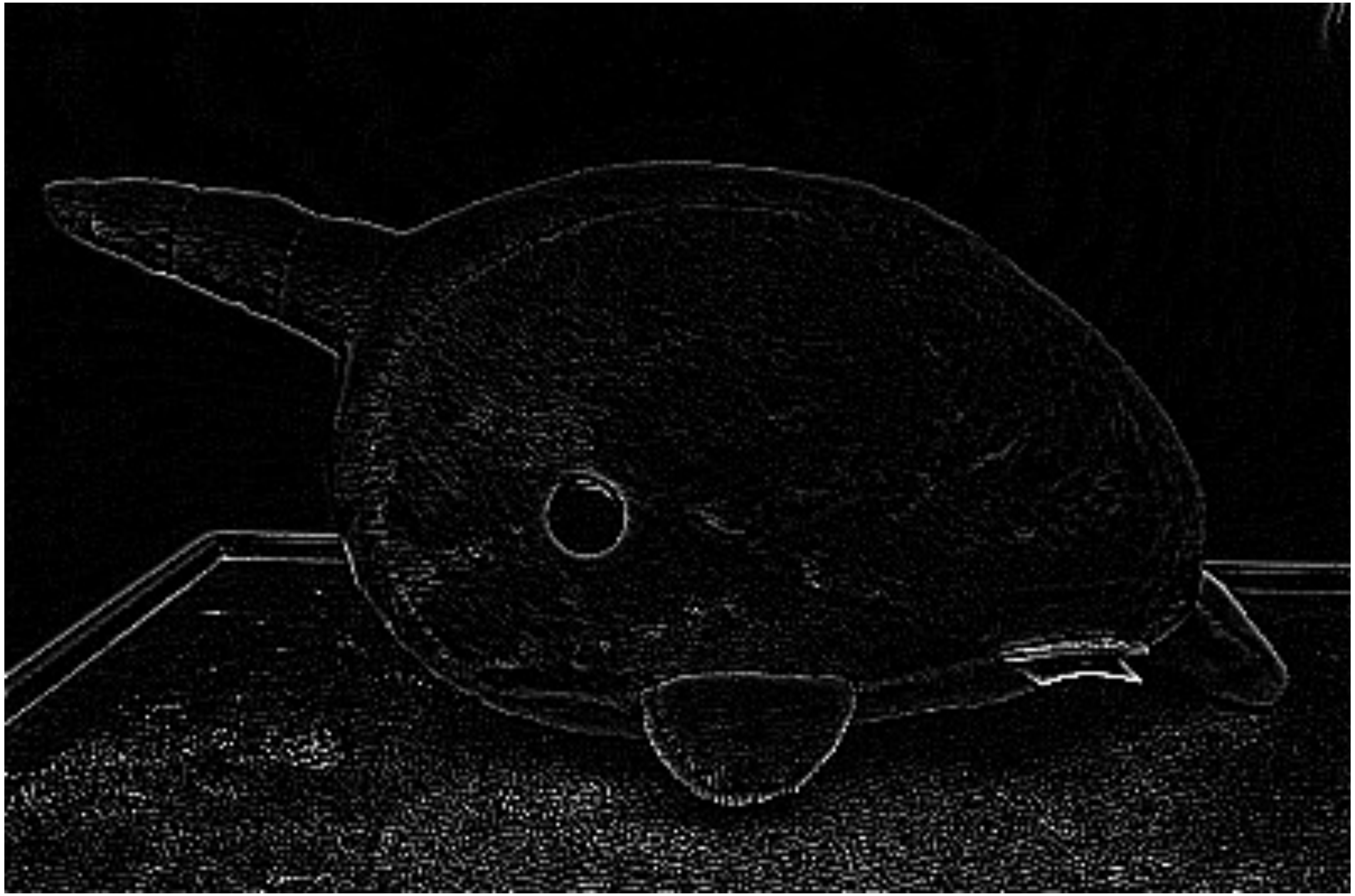
0	-1	0
-1	5	-1
0	-1	0



Convolution Examples

Edge Detector

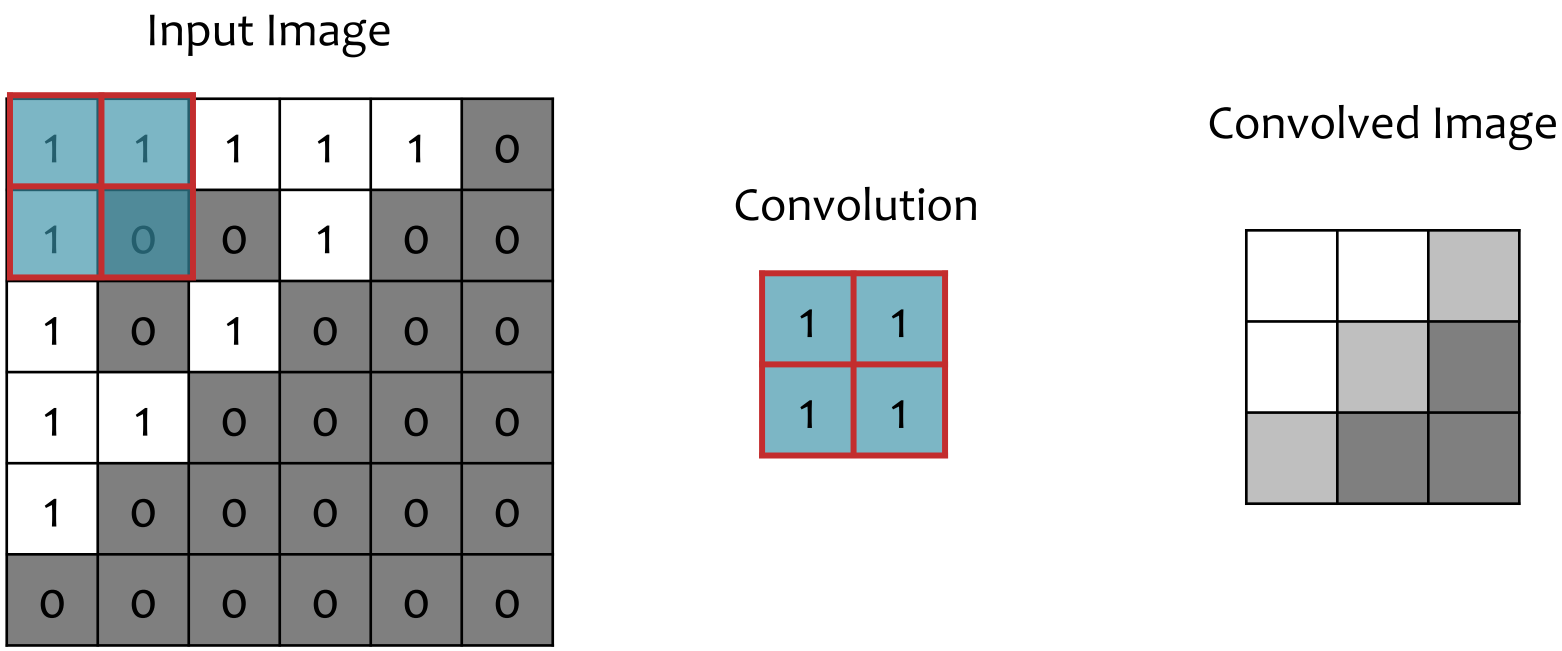
-1	-1	-1
-1	8	-1
-1	-1	-1



STRIDE AND DOWNSAMPLING

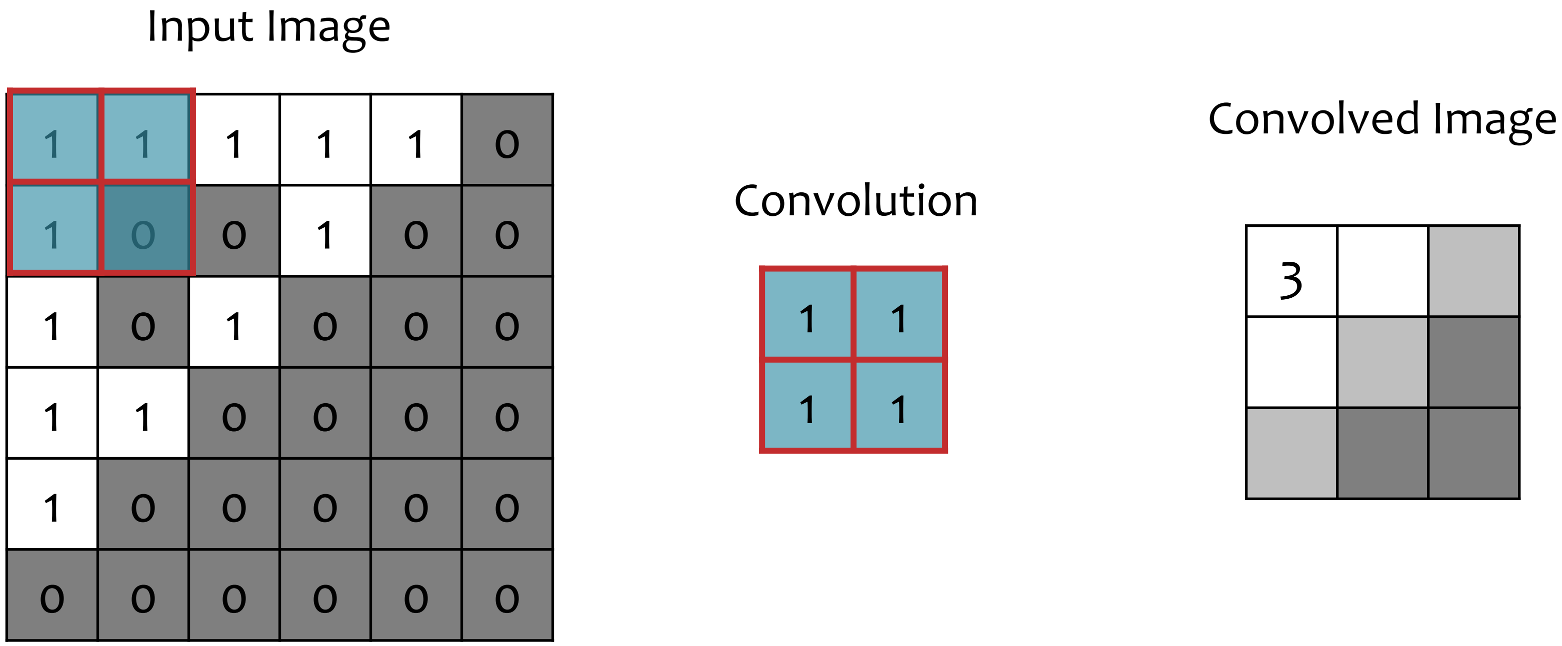
Stride and Downsampling

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



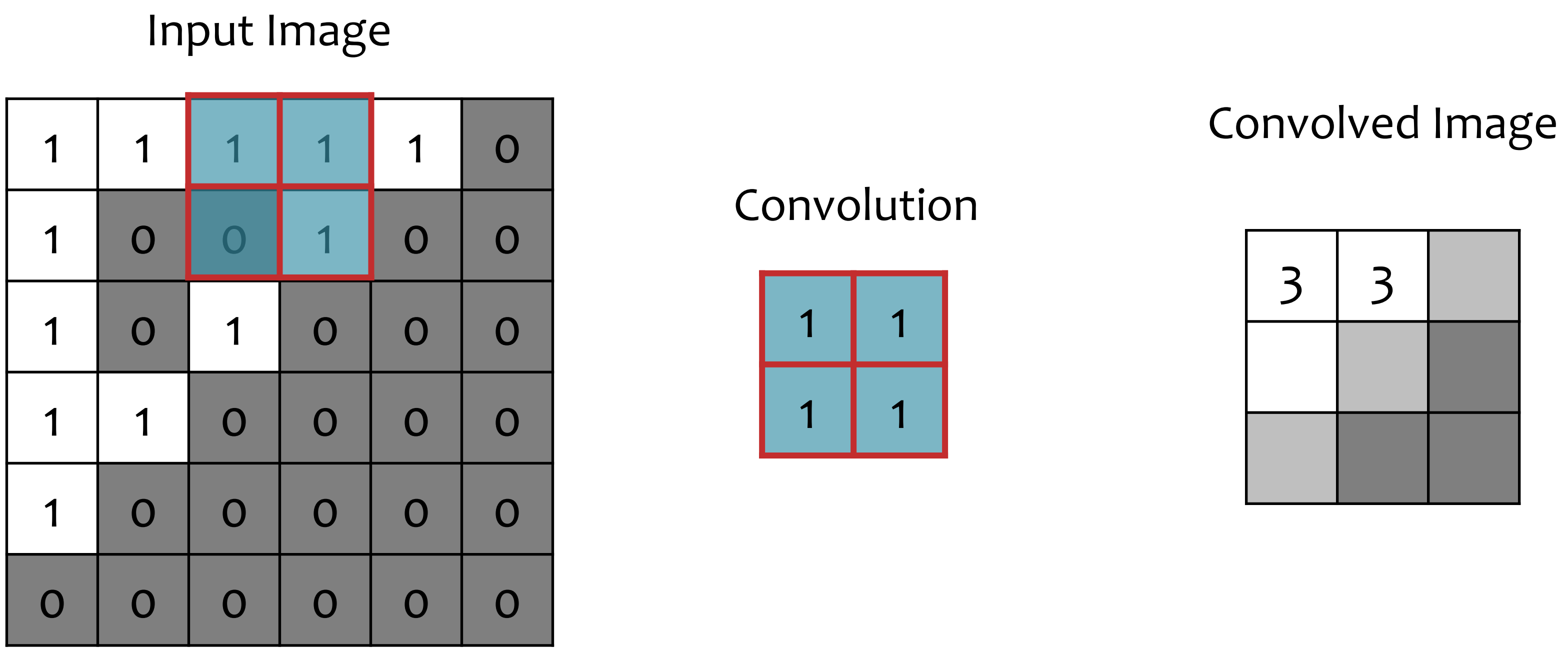
Stride and Downsampling

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



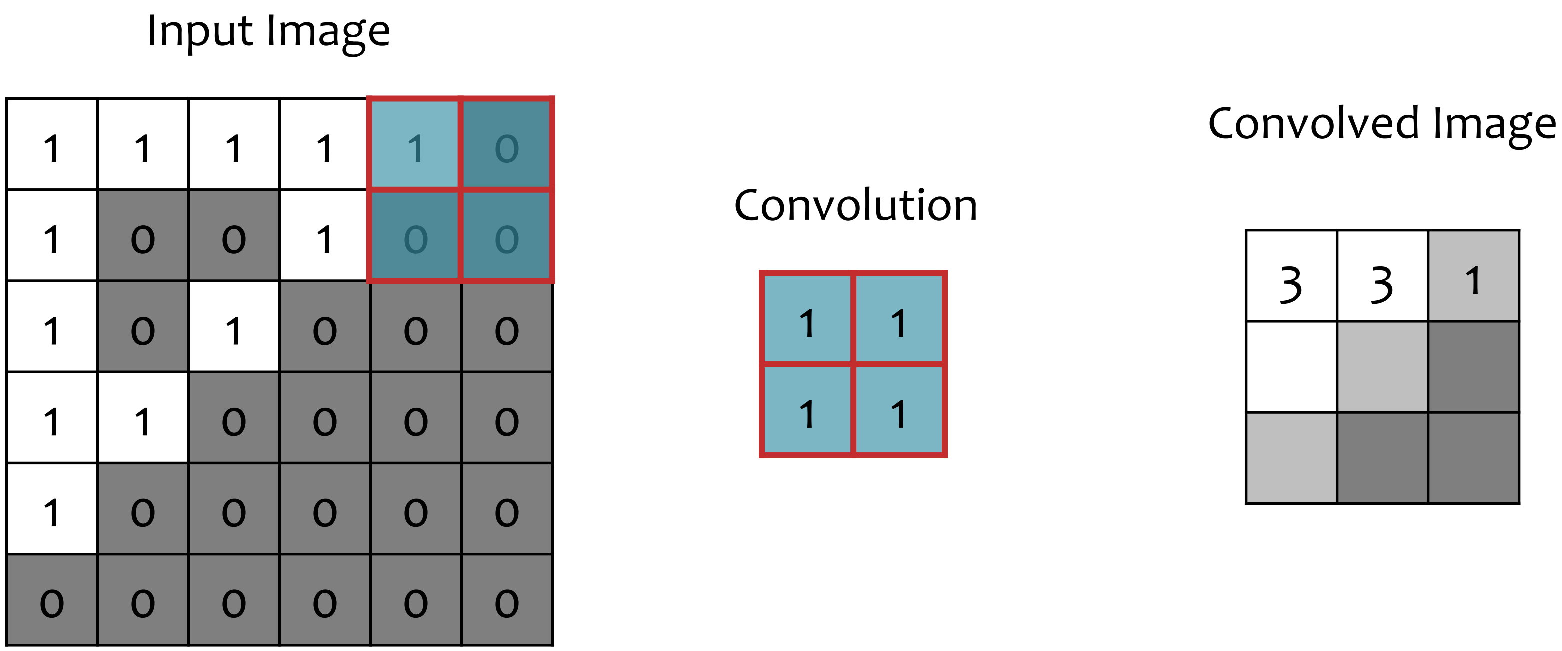
Stride and Downsampling

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



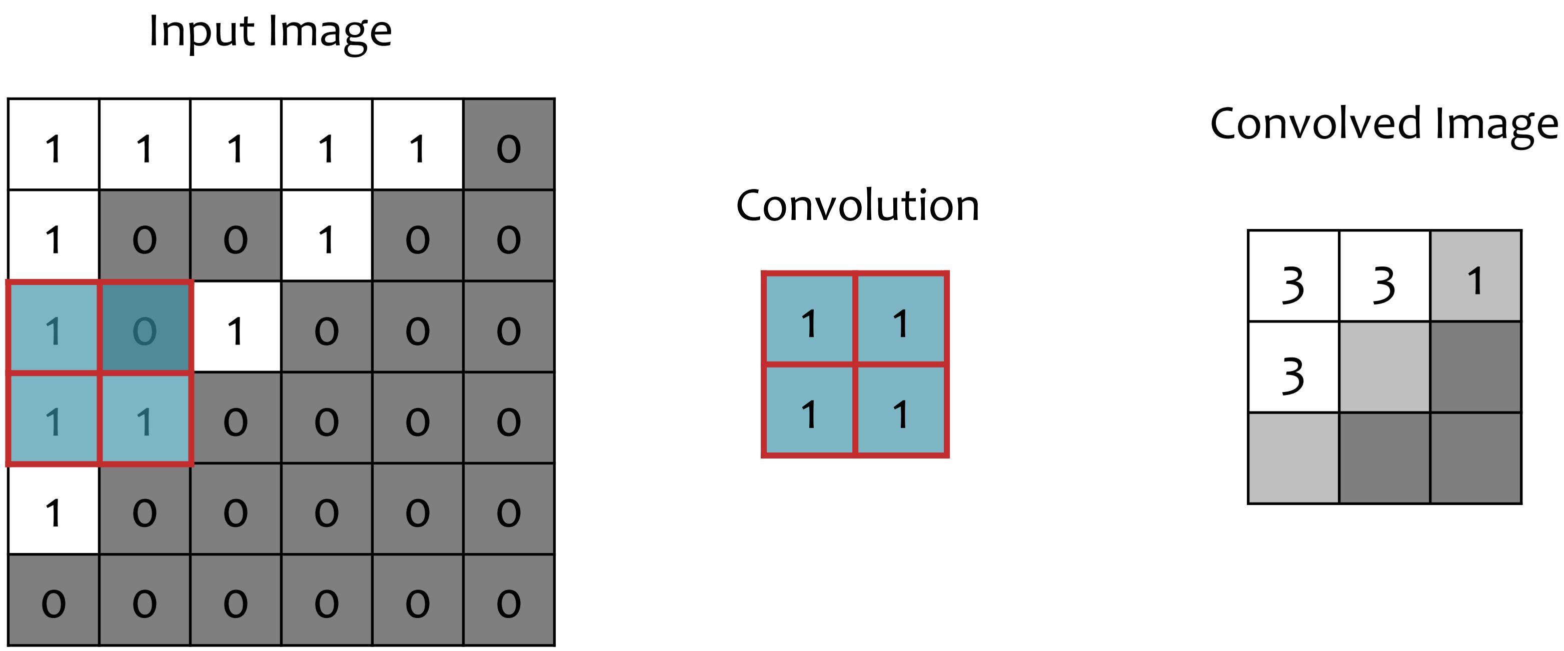
Stride and Downsampling

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



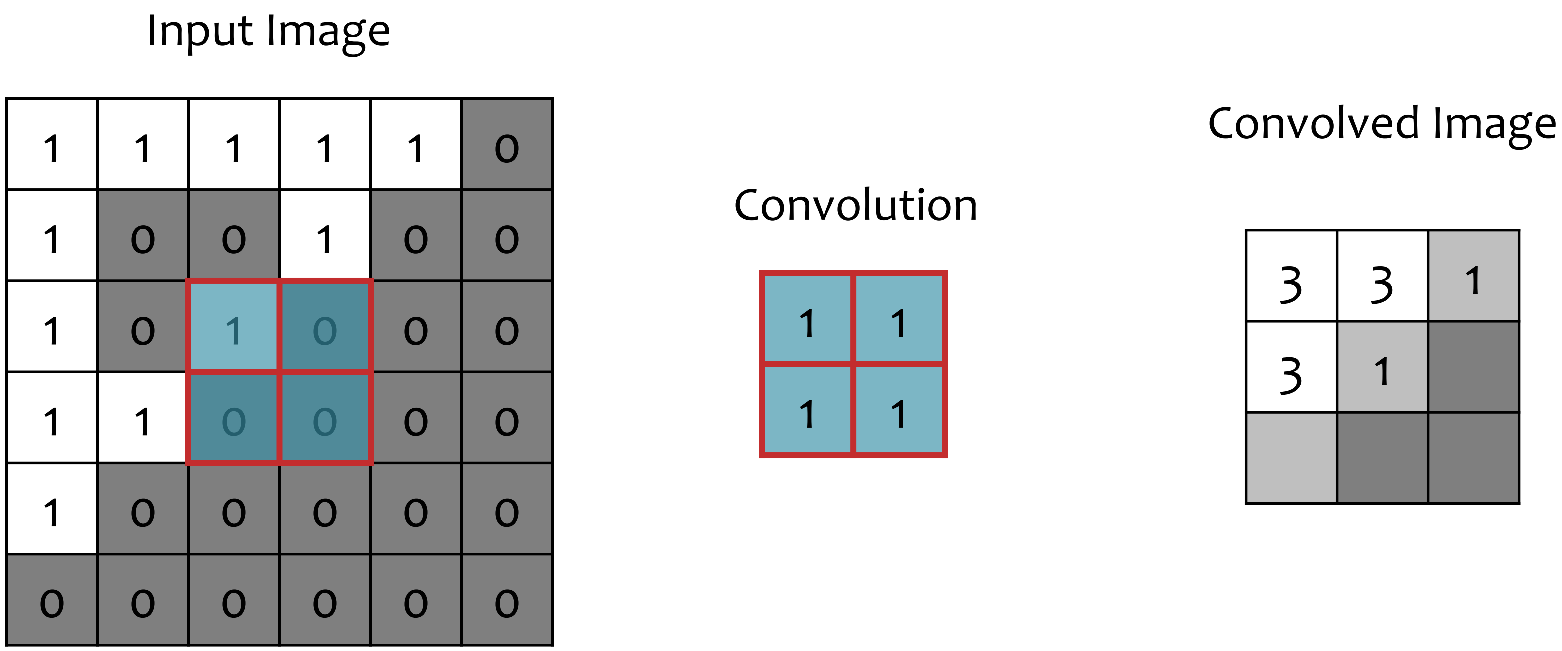
Stride and Downsampling

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



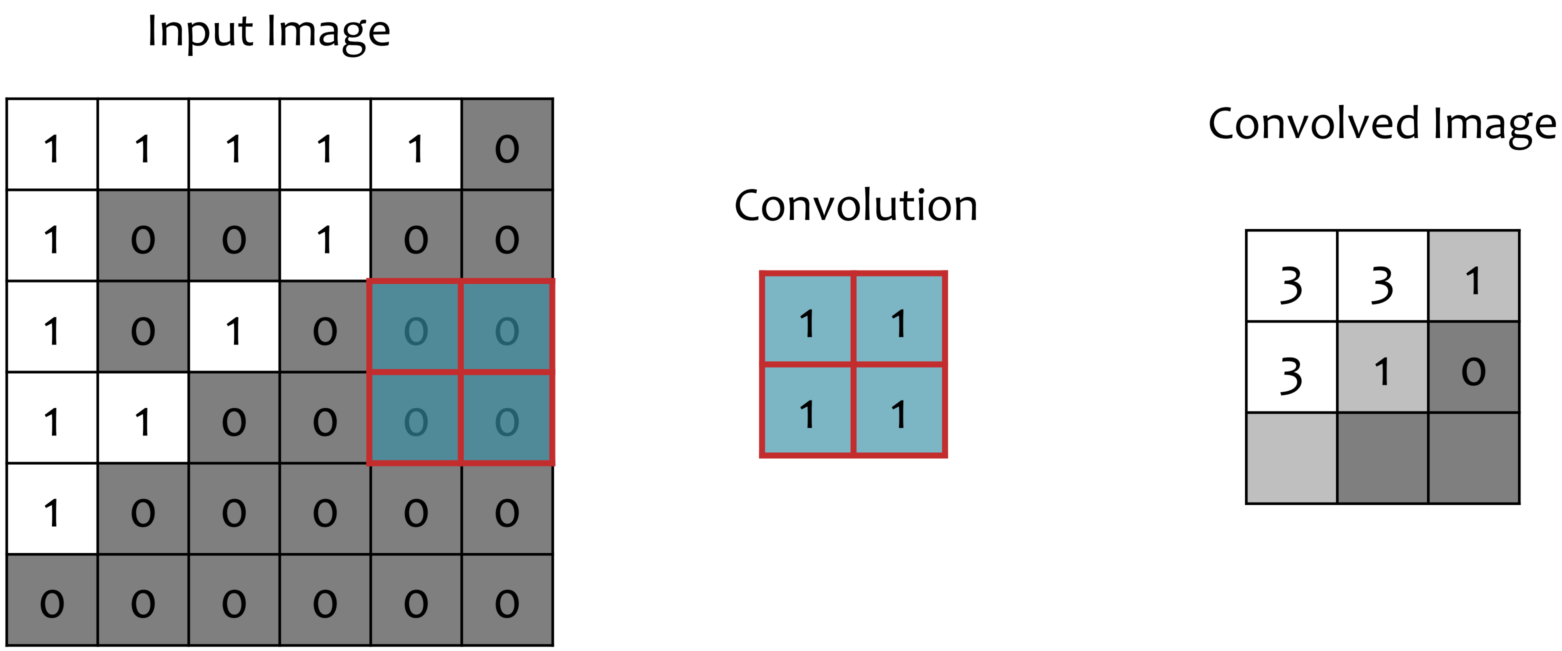
Stride and Downsampling

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



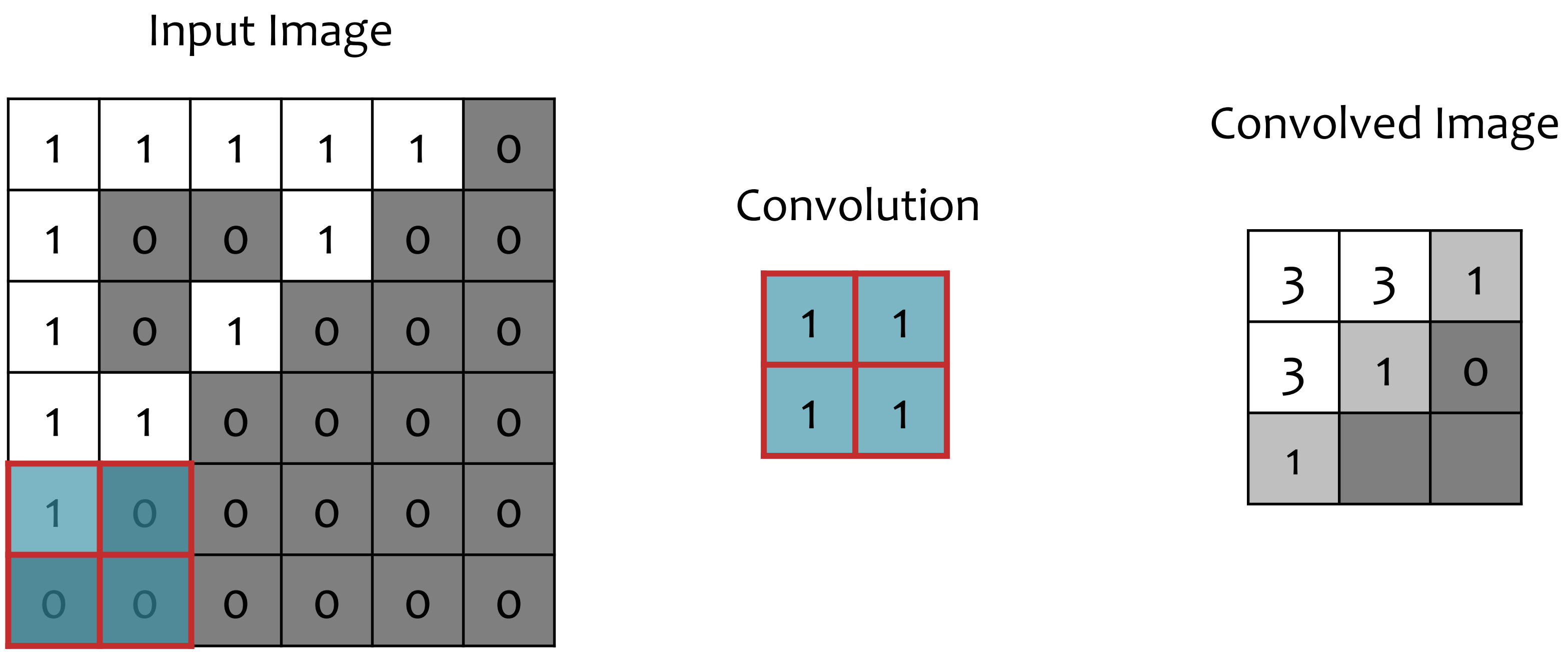
Stride and Downsampling

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



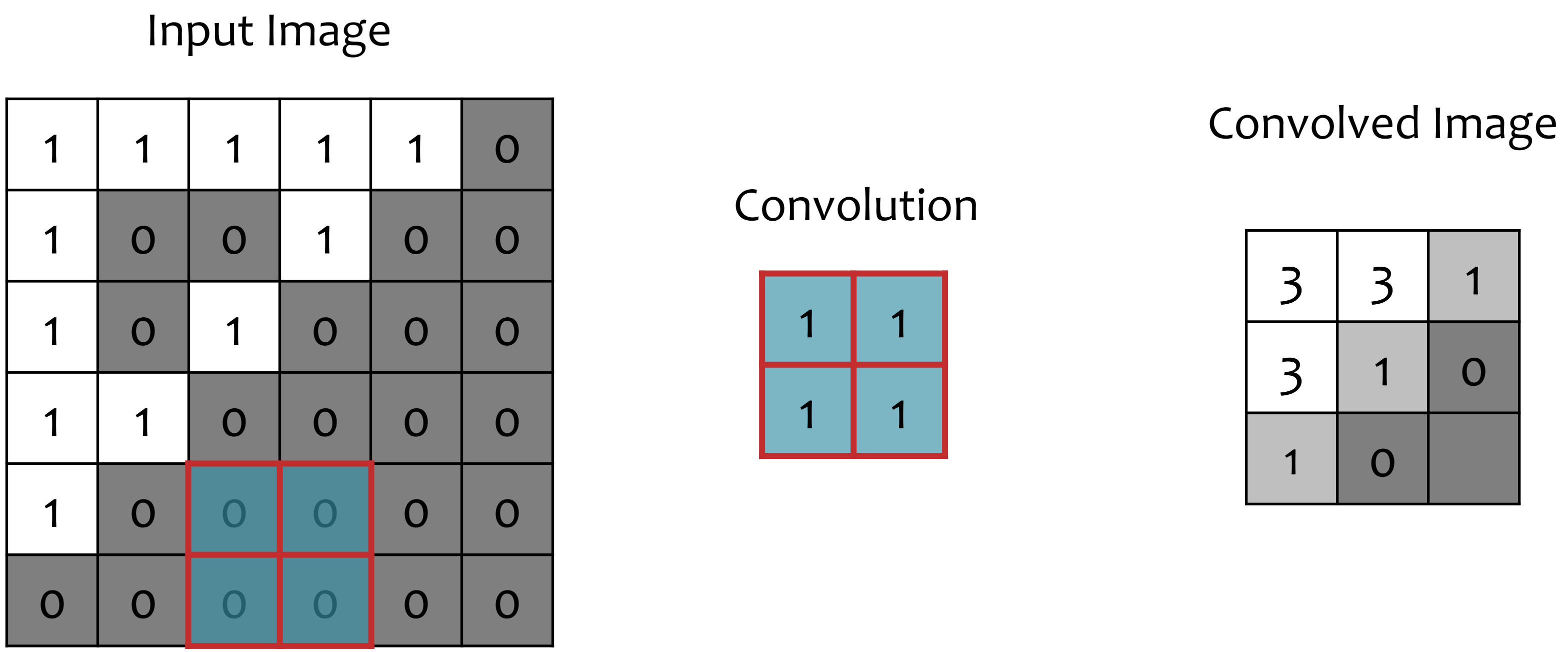
Stride and Downsampling

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



Stride and Downsampling

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



Stride and 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

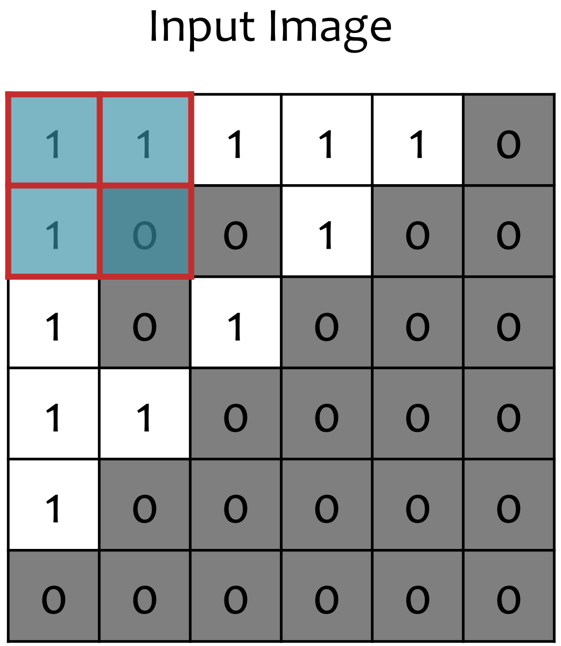
1/4	1/4
1/4	1/4

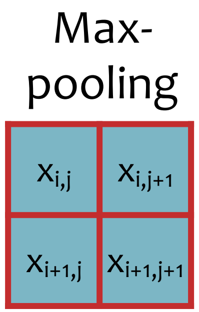
Convolved Image

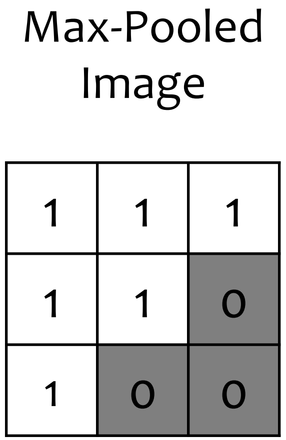
3/4	3/4	1/4
3/4	1/4	0
1/4	0	0

Max-Pooling

- Max-pooling with a stride > 1 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







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

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)

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

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of t

– Decision functio

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

– Loss function

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

3. Define goal:

N … $\boldsymbol{y}_i)$

opp site the gradient)

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

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

SGD for CNNs

Example: Simple CNN Architecture

Given $\mathbf{x}, \mathbf{y}^*$ and parameters $\boldsymbol{\theta} = [\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{W}]$

$$
\begin{aligned}
J &= \ell(\mathbf{y}, \mathbf{y}^*) \\
\mathbf{y} &= \text{softmax}(\mathbf{z}^{(5)}) \\
\mathbf{z}^{(5)} &= \text{linear}(\mathbf{z}^{(4)}, \mathbf{W}) \\
\mathbf{z}^{(4)} &= \text{relu}(\mathbf{z}^{(3)}) \\
\mathbf{z}^{(3)} &= \text{conv}(\mathbf{z}^{(2)}, \boldsymbol{\beta}) \\
\mathbf{z}^{(2)} &= \text{max-pool}(\mathbf{z}^{(1)}) \\
\mathbf{z}^{(1)} &= \text{conv}(\mathbf{x}, \boldsymbol{\alpha})
\end{aligned}
$$

Algorithm 1 Stochastic Gradient Descent (SGD)

```
Initialize θ
while not converged do
    Sample i ∈ {1, ..., N}
    Forward: y = h_θ(x^(i)),
                 J(θ) = ℓ(y, y^(i))
    Backward: Compute ∇_θ J(θ)
    Update: θ ← θ − η∇_θ J(θ)
```

LAYERS OF A CNN

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

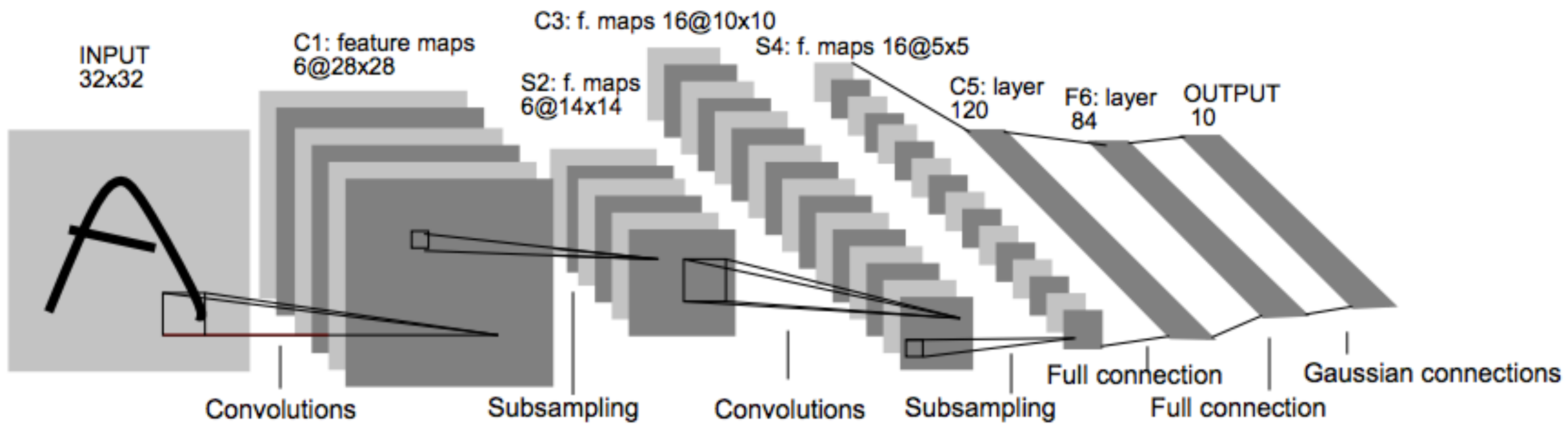


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

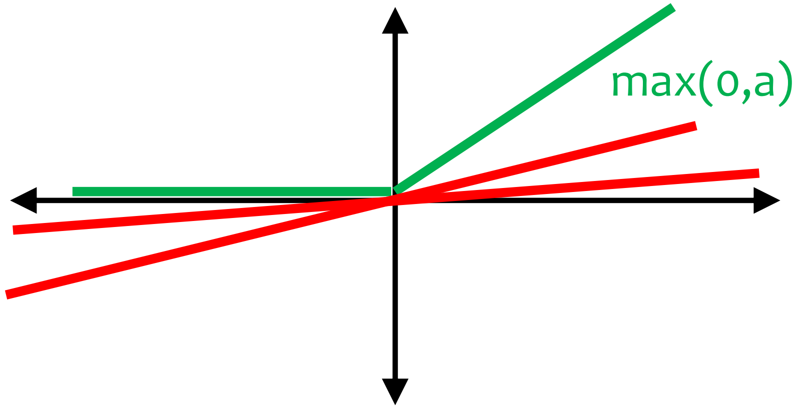
ReLU Layer

Output: $\mathbf{y} \in \mathbb{R}^K$

Forward:

$$\mathbf{y} = \sigma(\mathbf{x}), \text{ element-wise}$$
$$\sigma(a) = \max(0, a)$$

Input: $\mathbf{x} \in \mathbb{R}^K$



Input: $\frac{\partial J}{\partial \mathbf{y}} \in \mathbb{R}^K$

Backward: for each j,

$$\frac{\partial J}{\partial x_j} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial x_j}$$

where

subderivative

$$\frac{\partial y_j}{\partial x_j} = \begin{cases} 1 & \text{if } x_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

Output: $\frac{\partial J}{\partial \mathbf{x}} \in \mathbb{R}^K$

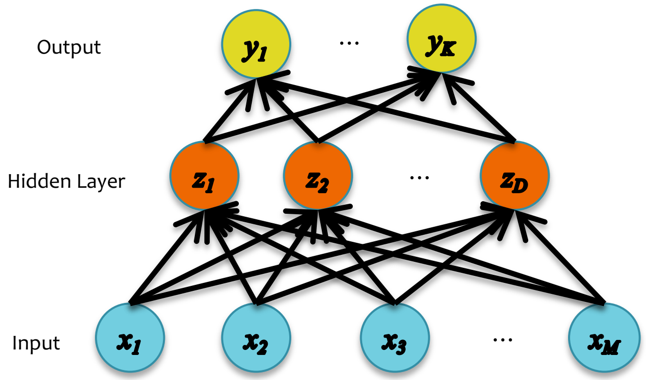
Softmax Layer

Output: $\mathbf{y} \in \mathbb{R}^K$

Forward: for each i,

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

Input: $\mathbf{x} \in \mathbb{R}^K$



Input: $\frac{\partial J}{\partial \mathbf{y}} \in \mathbb{R}^K$

Backward: for each j,

$$\frac{\partial J}{\partial x_j} = \sum_{i=1}^{K} \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial x_j}$$

where

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

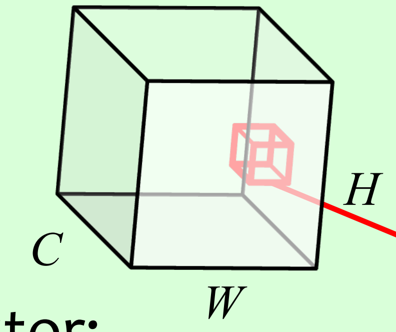
Output: $\frac{\partial J}{\partial \mathbf{x}} \in \mathbb{R}^K$

Fully-Connected Layer (3D input)

Forward:

1. suppose input is a 3D tensor:

$$\mathbf{x} =$$



2. flatten out tensor into a vector:

$$\hat{\mathbf{x}} = [\hat{x}_1, \ldots, \hat{x}_{(C \times H \times W)}] \quad \text{where } \hat{x}_{(H \times W \times i + W \times j + k)} = x_{i,j,k}$$

3. then push that vector through a standard linear layer:

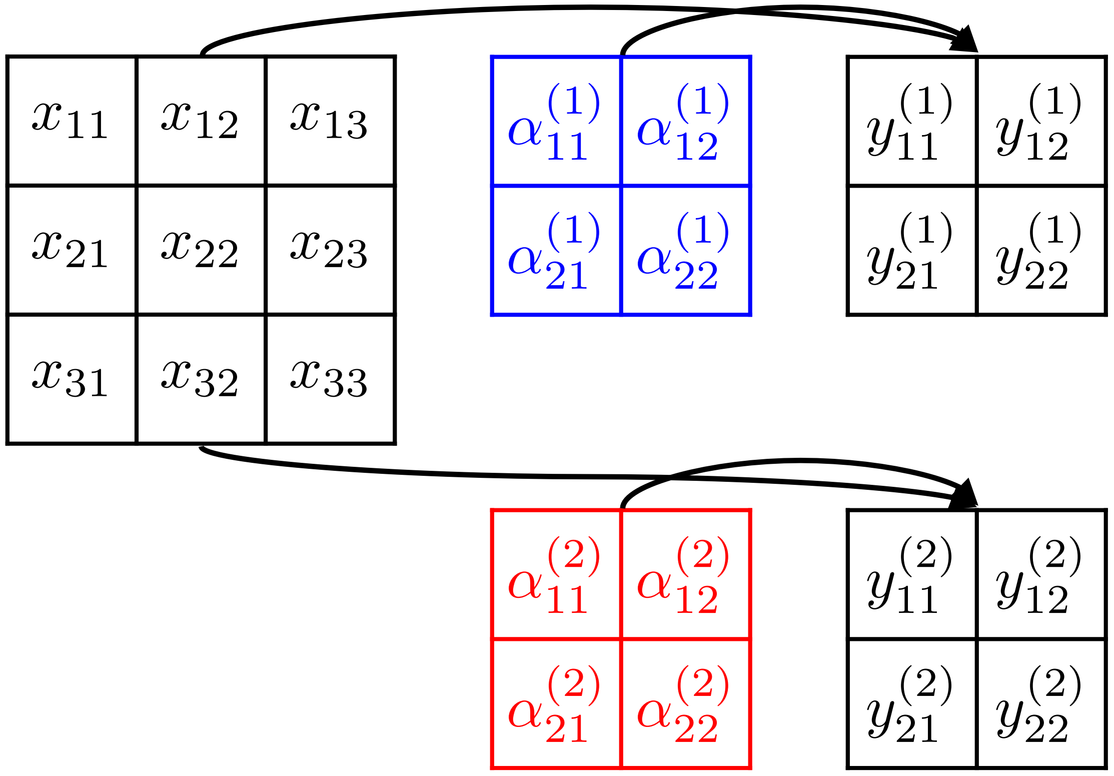
$$\mathbf{y} = \boldsymbol{\alpha}^T \hat{\mathbf{x}} + \boldsymbol{\alpha}_0 \quad \text{where } \boldsymbol{\alpha} \in \mathbb{R}^{A \times B}, \quad \boldsymbol{\alpha}_0 \in \mathbb{R}^B$$

$$|\hat{\mathbf{x}}| \in \mathbb{R}^A, \quad |\mathbf{y}| \in \mathbb{R}^B$$

2D Convolution

Example: 1 input channel, 2 output channels

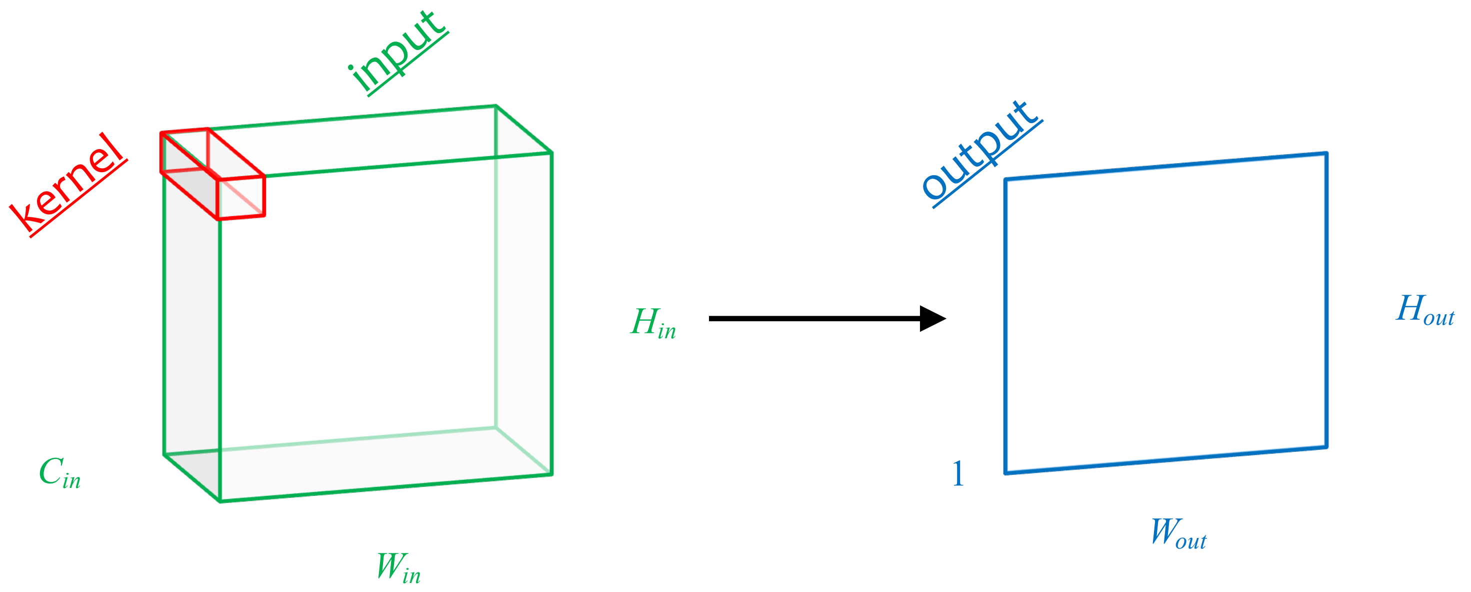
Input Kernel Output



$$y_{11}^{(1)} = \alpha_{11}^{(1)} x_{11} + \alpha_{12}^{(1)} x_{12} + \alpha_{21}^{(1)} x_{21} + \alpha_{22}^{(1)} x_{22} + \alpha_{0}^{(1)}$$
$$y_{12}^{(1)} = \alpha_{11}^{(1)} x_{12} + \alpha_{12}^{(1)} x_{13} + \alpha_{21}^{(1)} x_{22} + \alpha_{22}^{(1)} x_{23} + \alpha_{0}^{(1)}$$
$$y_{21}^{(1)} = \alpha_{11}^{(1)} x_{21} + \alpha_{12}^{(1)} x_{22} + \alpha_{21}^{(1)} x_{31} + \alpha_{22}^{(1)} x_{32} + \alpha_{0}^{(1)}$$
$$y_{22}^{(1)} = \alpha_{11}^{(1)} x_{22} + \alpha_{12}^{(1)} x_{23} + \alpha_{21}^{(1)} x_{32} + \alpha_{22}^{(1)} x_{33} + \alpha_{0}^{(1)}$$

$$y_{11}^{(2)} = \alpha_{11}^{(2)} x_{11} + \alpha_{12}^{(2)} x_{12} + \alpha_{21}^{(2)} x_{21} + \alpha_{22}^{(2)} x_{22} + \alpha_{0}^{(2)}$$
$$y_{12}^{(2)} = \alpha_{11}^{(2)} x_{12} + \alpha_{12}^{(2)} x_{13} + \alpha_{21}^{(2)} x_{22} + \alpha_{22}^{(2)} x_{23} + \alpha_{0}^{(2)}$$
$$y_{21}^{(2)} = \alpha_{11}^{(2)} x_{21} + \alpha_{12}^{(2)} x_{22} + \alpha_{21}^{(2)} x_{31} + \alpha_{22}^{(2)} x_{32} + \alpha_{0}^{(2)}$$
$$y_{22}^{(2)} = \alpha_{11}^{(2)} x_{22} + \alpha_{12}^{(2)} x_{23} + \alpha_{21}^{(2)} x_{32} + \alpha_{22}^{(2)} x_{33} + \alpha_{0}^{(2)}$$

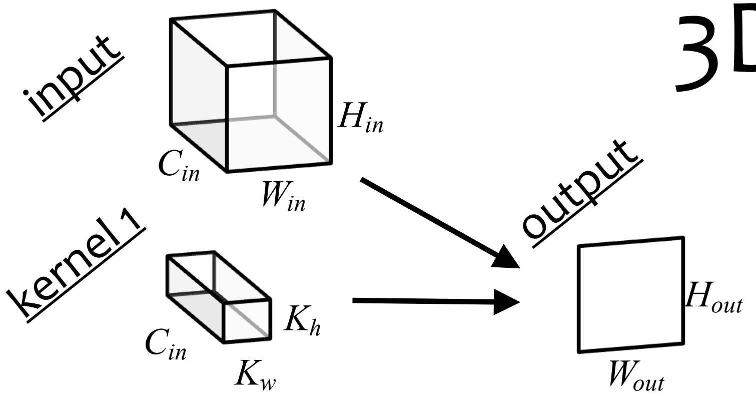
Convolution of a Color Image

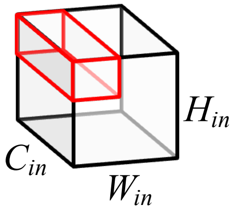


- Color images consist of 3 floats per pixel for RGB (red, green blue) color values
- The kernel must also be 3-dimensional

- input = 3x64x64
- kernel = 3x5x5
- output = 1x64x64 (assuming padding)

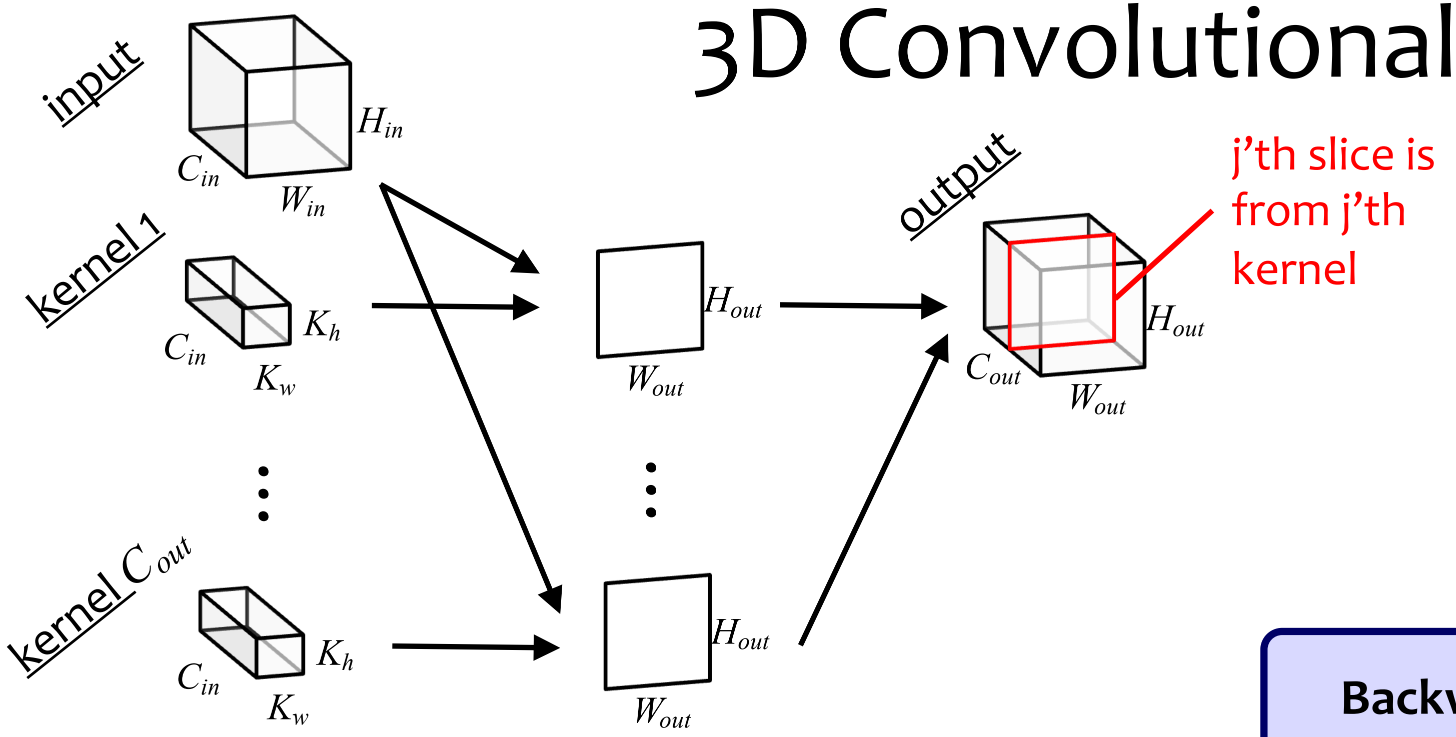
3D Convolutional Layer



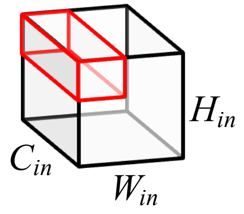


- Color images consist of 3 floats per pixel for RGB (red, green blue) color values
- Convolution must also be 3-dimensional

3D Convolutional Layer



<u>Convolution in 3D</u>



Forward:

$$y^{(c')}_{h',w'} = \beta^{(c')} + \sum_{c=1}^{C_{\text{in}}} \sum_{m=1}^{K_{\text{h}}} \sum_{n=1}^{K_{\text{w}}} x^{(c)}_{h'+ms,w'+ns} \cdot \alpha^{(c',c)}_{m,n}$$

Backward:

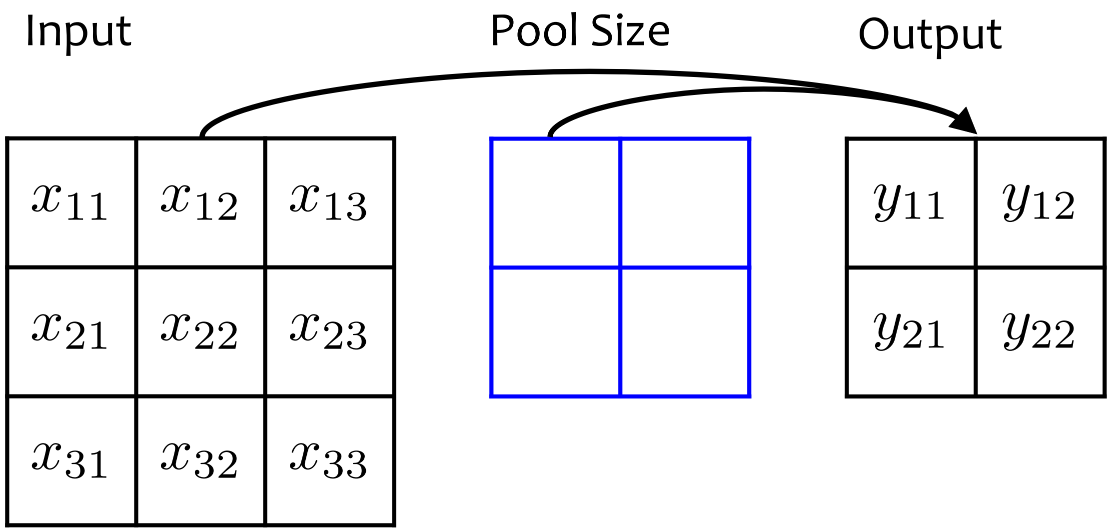
$$\frac{\partial J}{\partial \alpha^{(c',c)}_{m,n}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}} \cdot x^{(c)}_{h'+ms,w'+ns}$$

$$\frac{\partial J}{\partial \beta^{(c')}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}}$$

$s \in \mathbb{Z}$ (stride)

Max-Pooling Layer

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



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

3D Max-Pooling Layer

Output: $\mathbf{y} \in \mathbb{R}^{C \times H_{\text{out}} \times W_{\text{out}}}$

Forward:

$$y_{ij}^{(c)} = \max_{q \in \{1,\ldots,K_h\},\, r \in \{1,\ldots,K_w\}} x_{mn}^{(c)}$$

where

$$m = s(i-1) + q$$
$$n = s(j-1) + r$$

Input:

$\mathbf{x} \in \mathbb{R}^{C \times H_{\text{in}} \times W_{\text{in}}}$

K_h, K_w (kernel size)

s (stride)

Input: $\frac{\partial J}{\partial \mathbf{y}} \in \mathbb{R}^{C \times H_{\text{out}} \times W_{\text{out}}}$

Backward:

$$\frac{\partial J}{\partial x_{mn}^{(c)}} = \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{(c)}} \frac{\partial y_{ij}^{(c)}}{\partial x_{mn}^{(c)}}$$

Output: $\frac{\partial J}{\partial \mathbf{x}} \in \mathbb{R}^{C \times H_{\text{in}} \times W_{\text{in}}}$

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

$$y = \max(a, b)$$

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

CNN ARCHITECTURES

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

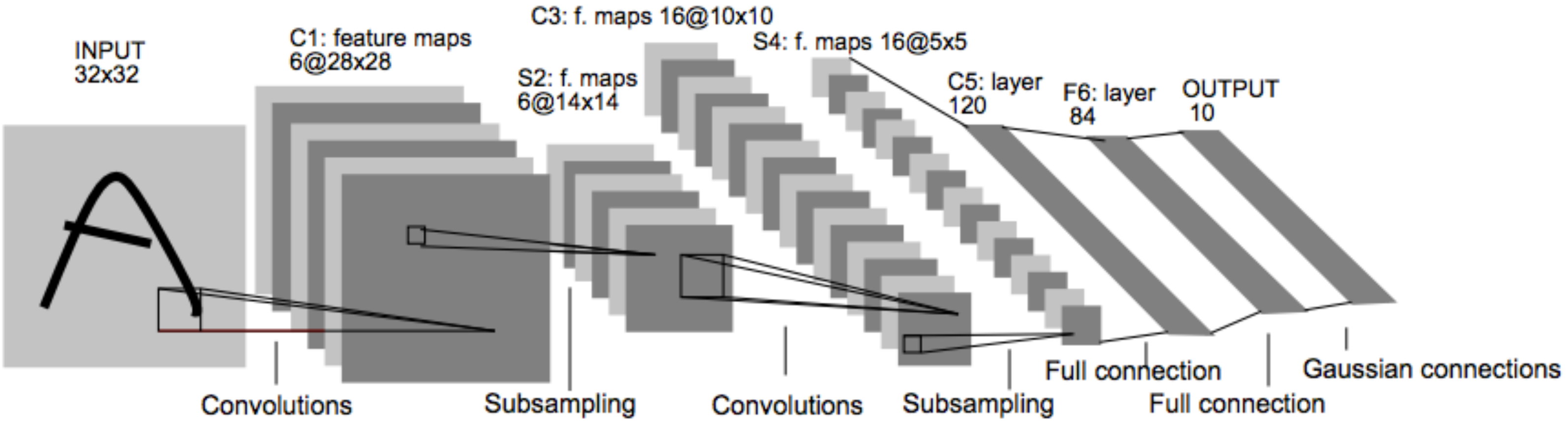


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

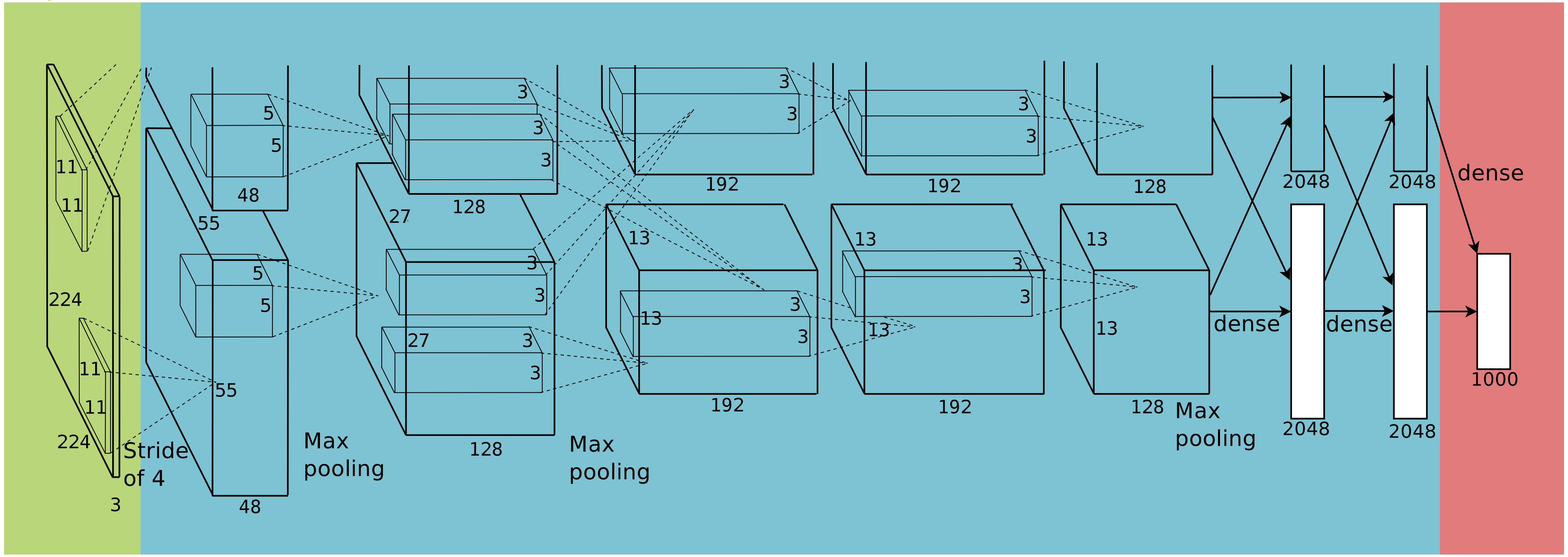
Architecture #2: AlexNet

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

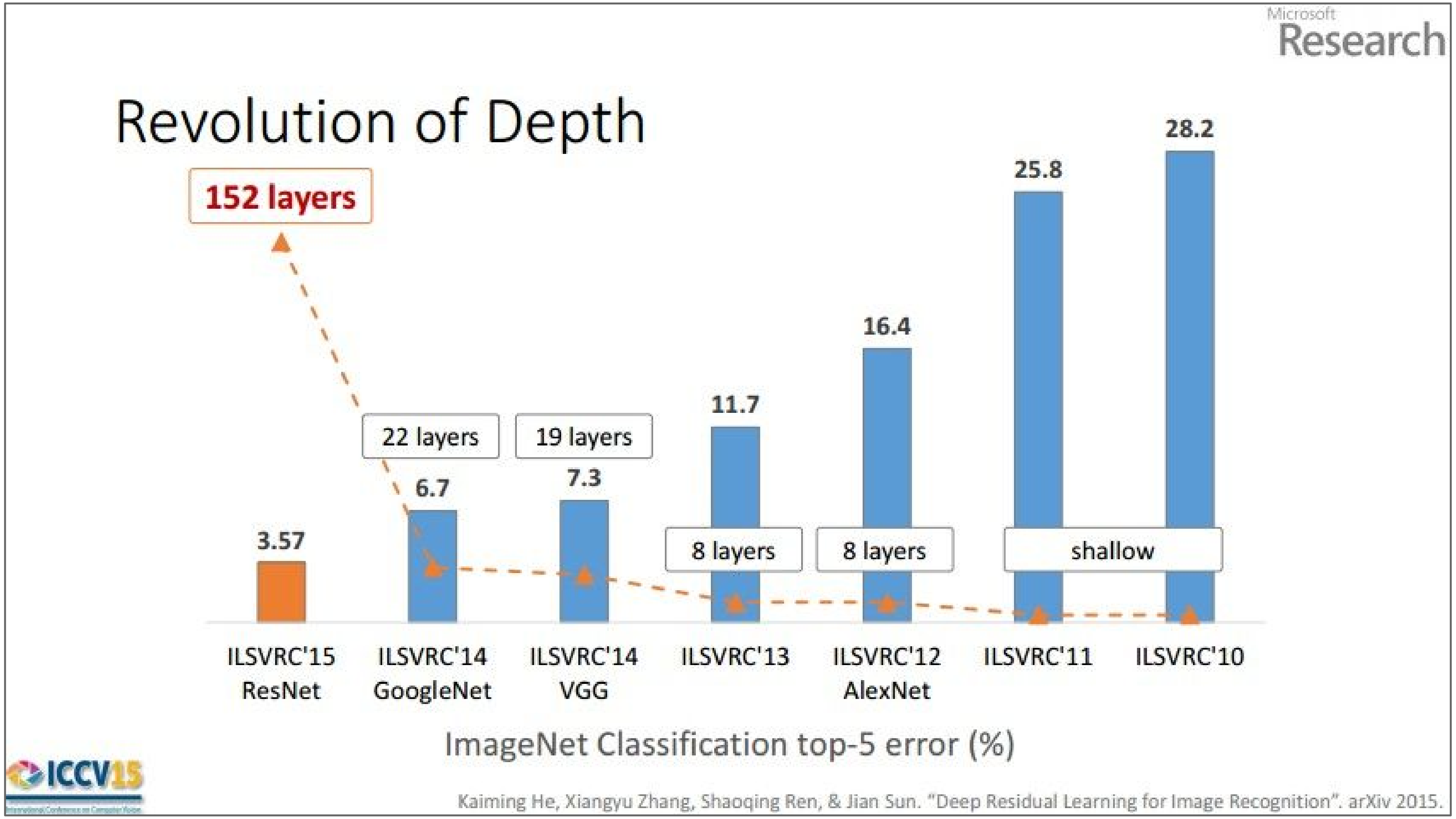
Input image (pixels)

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

1000-way softmax



CNNs for Image Recognition



Slide from Kaiming He

Convolutional Neural Network (CNN)

Typical Architectures

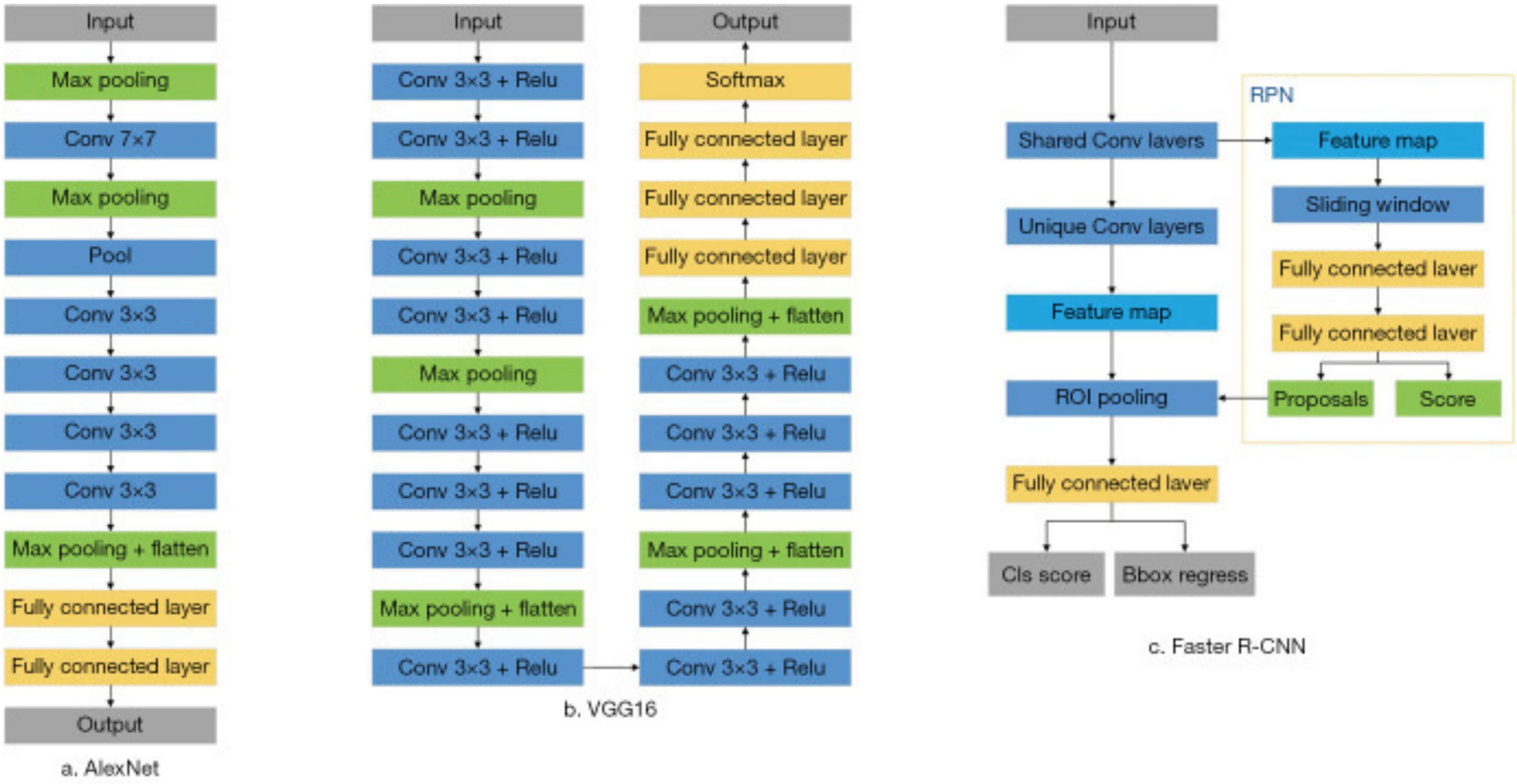


Figure from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7327346/

Convolutional Neural Network (CNN)

Typical Architectures

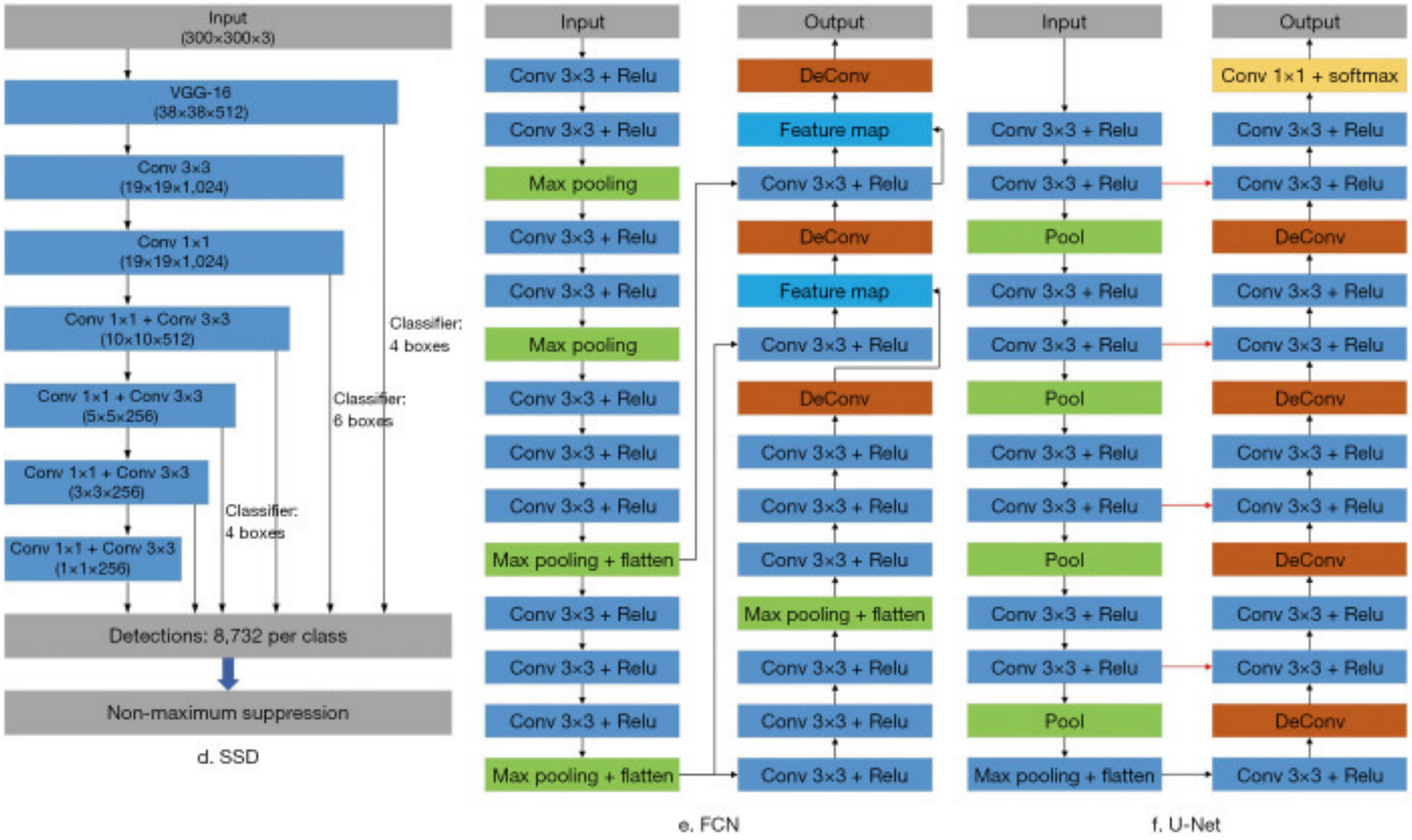


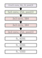
Figure from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7327346/

Convolutional Neural Network (CNN)

Typical Architectures

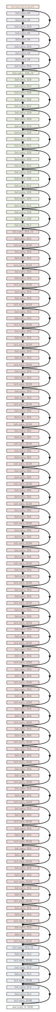
Microsoft Research

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)



ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Location-specific Parameters

Poll Question 2:

Why do many layers used in computer vision *not have* location specific parameters?

Answer:

Convolutional Layer

For a convolutional layer, how do we pick the kernel size (aka. the size of the convolution)?

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

2x2 Convolution

θ_{11}	θ_{12}
θ_{21}	θ_{22}

3x3 Convolution

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

4x4 Convolution

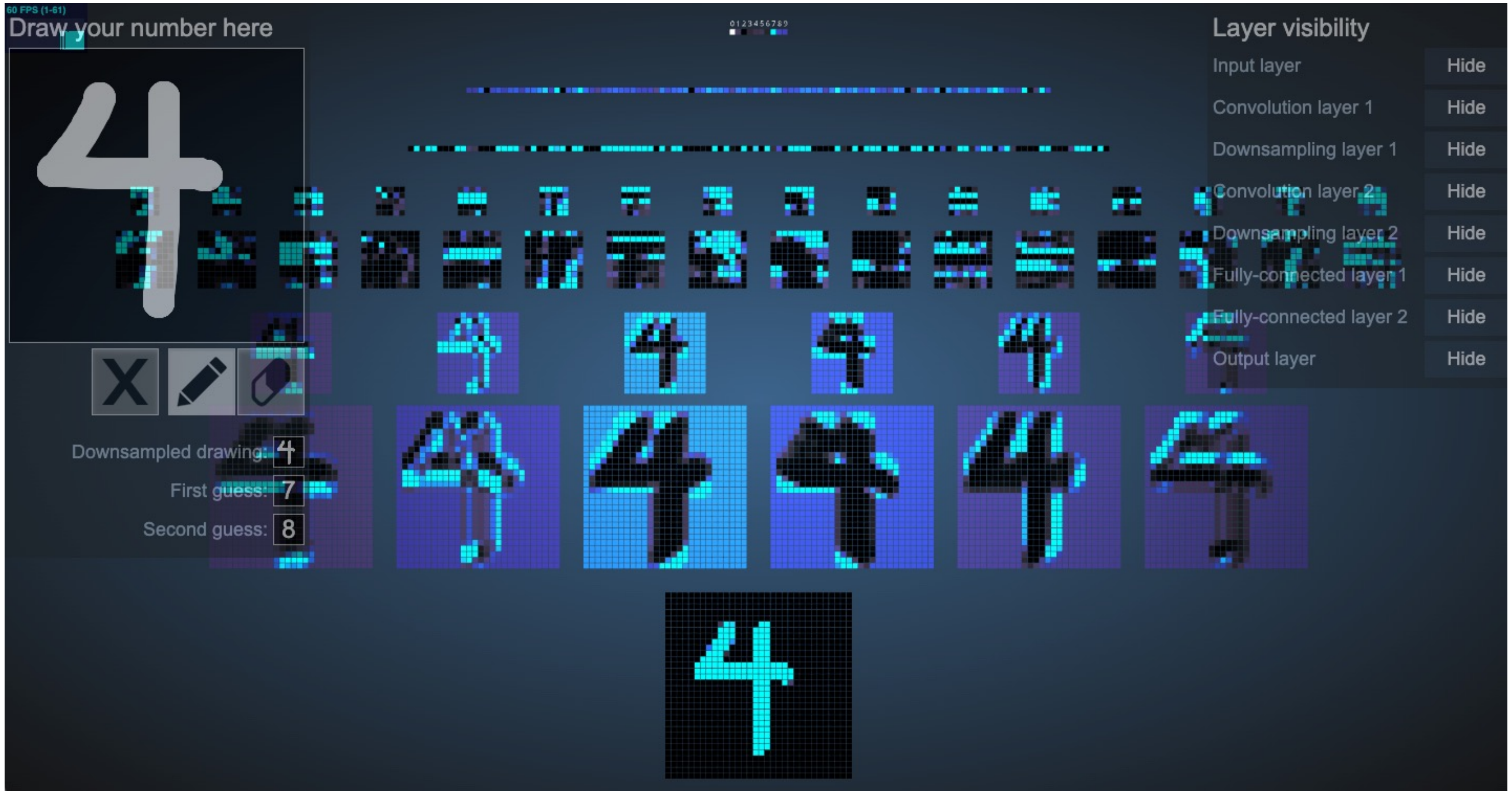
θ_{11}	θ_{12}	θ_{13}	θ_{14}
θ_{21}	θ_{22}	θ_{23}	θ_{24}
θ_{31}	θ_{32}	θ_{33}	θ_{34}
θ_{41}	θ_{42}	θ_{43}	θ_{44}

- A small kernel can only see a very small part of the image, but is fast to compute
- A large kernel can see more of the image, but at the expense of speed

CNN VISUALIZATIONS

Visualization of CNN

https://adamharley.com/nn_vis/cnn/2d.html



MNIST Digit Recognition with CNNs (in your browser)

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

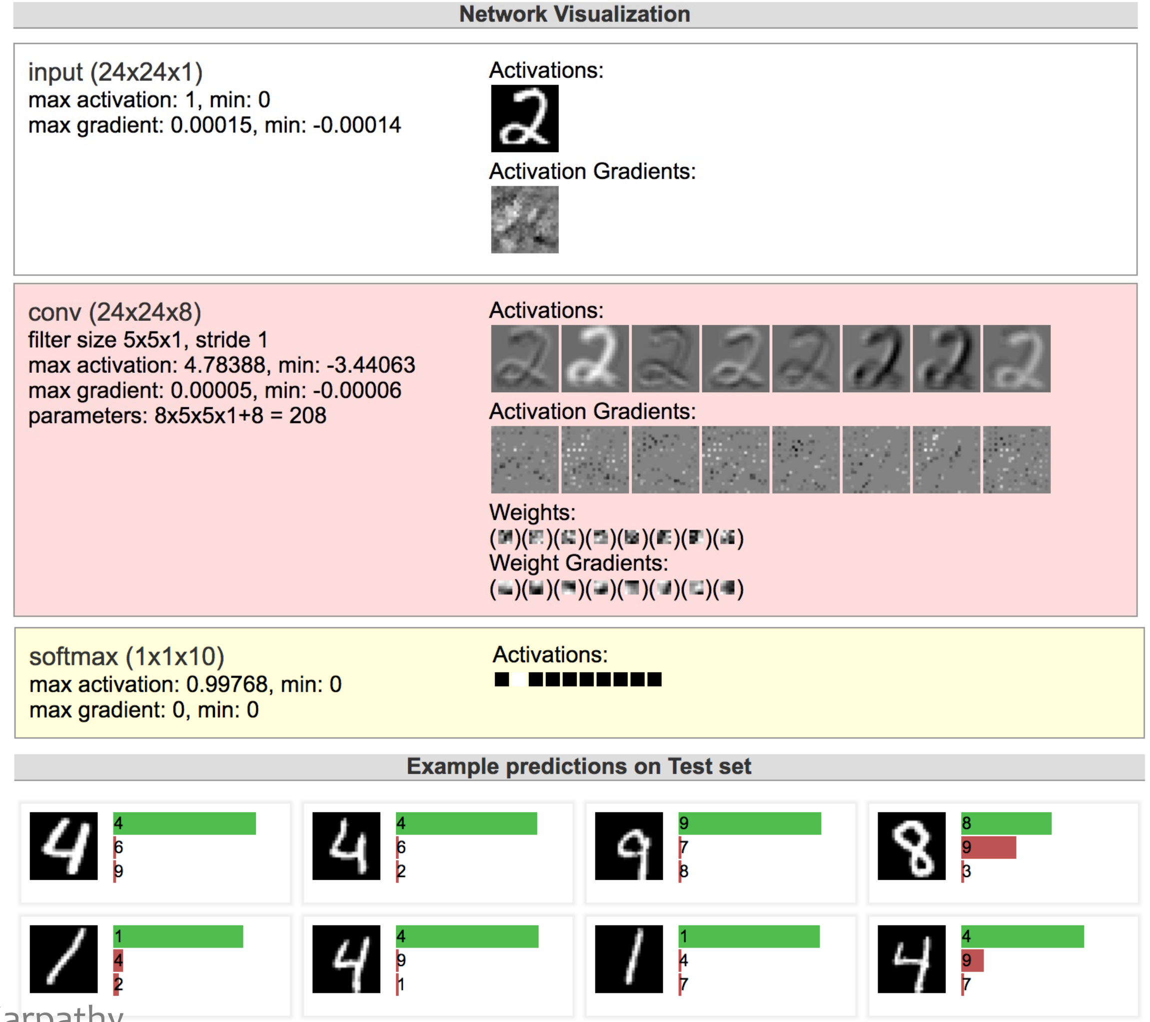


Figure from Andrej Karpathy

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

WORD EMBEDDINGS

Word Embeddings

Key Idea:

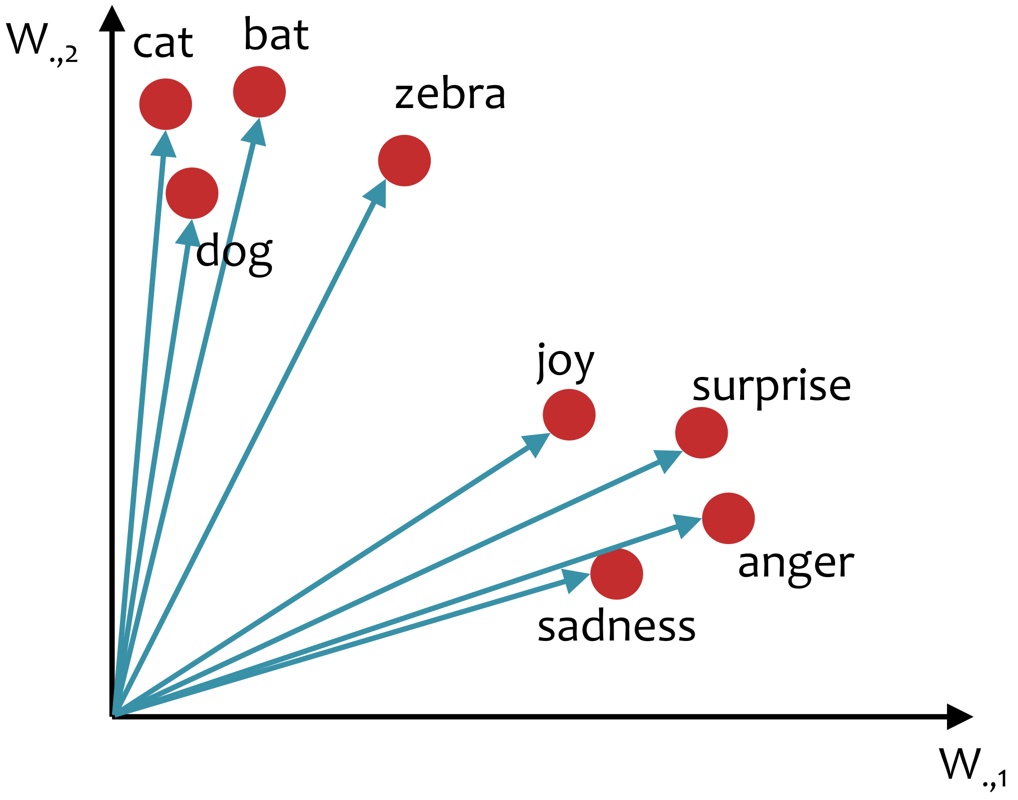
- represent each word in your vocabulary as a vector
- store as a V x D matrix where:
  V = number of words in vocab.
  D = vector's dimension

Modeling:

- define a model in which the vectors are parameters
- each copy of the word uses the same parameter vector
- train model so that similar words have high cosine similarity

W

anger	W_{11}	W_{12}
bat	W_{21}	W_{22}
cat	W_{31}	W_{32}
dog	W_{41}	W_{42}
joy	W_{51}	W_{52}
sadness	W_{61}	W_{62}
surprise	W_{71}	W_{72}
zebra	W_{81}	W_{82}



Word Embeddings

Key Idea:

- represent each word in your vocabulary as a vector
- store as a V x D matrix where:
  V = number of words in vocab.
  D = vector's dimension

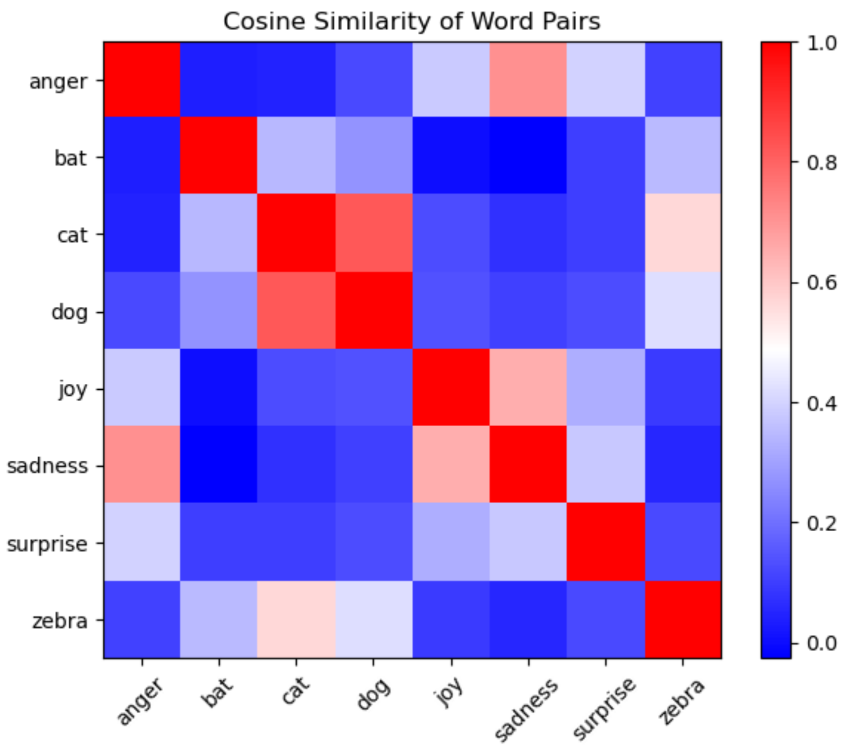
Modeling:

- define a model in which the vectors are parameters
- each copy of the word uses the same parameter vector
- train model so that similar words have high cosine similarity

W

aardvark	-2.3	0.0	-2.8	…	-4.5
anger	-2.8	-0.9	-1.7	…	-4.3
bat	-4.5	-1.3	0.6	…	-1.7
cat	3.5	-2.0	-2.3	…	-0.4
…				…	
joy	3.0	-0.6	-0.6	…	4.9
…				…	
zebra	-4.7	-4.2	-4.5	…	4.3

in a real use case, the typical embedding dimension is in the hundreds, e.g. D = 300

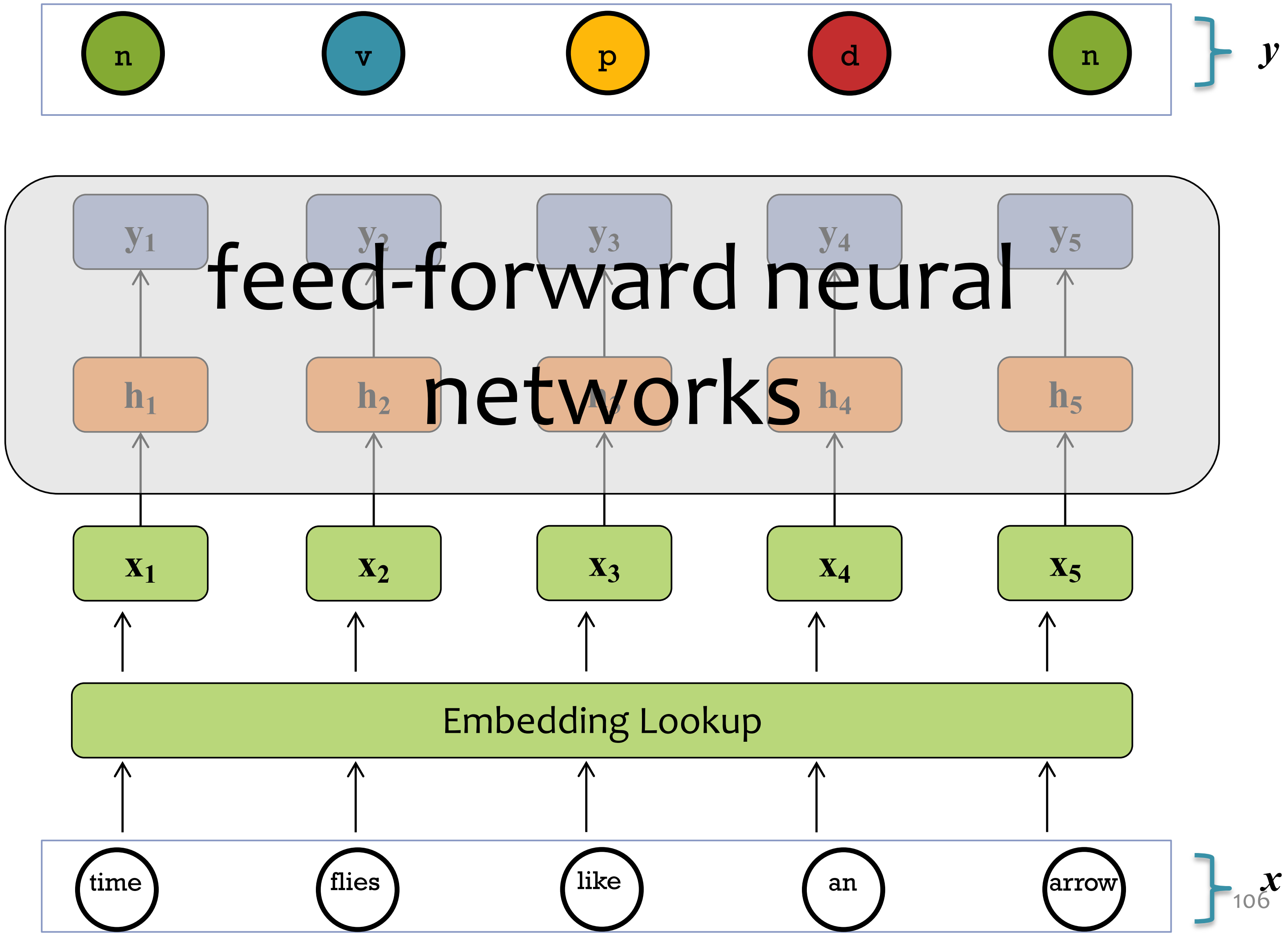


we can't visualize 300 dimensional vectors, but we can inspect their pairwise cosine similarities

Word Embeddings

In all the models we're about to consider (neural networks, RNNs, Transformers) that work with sentences…

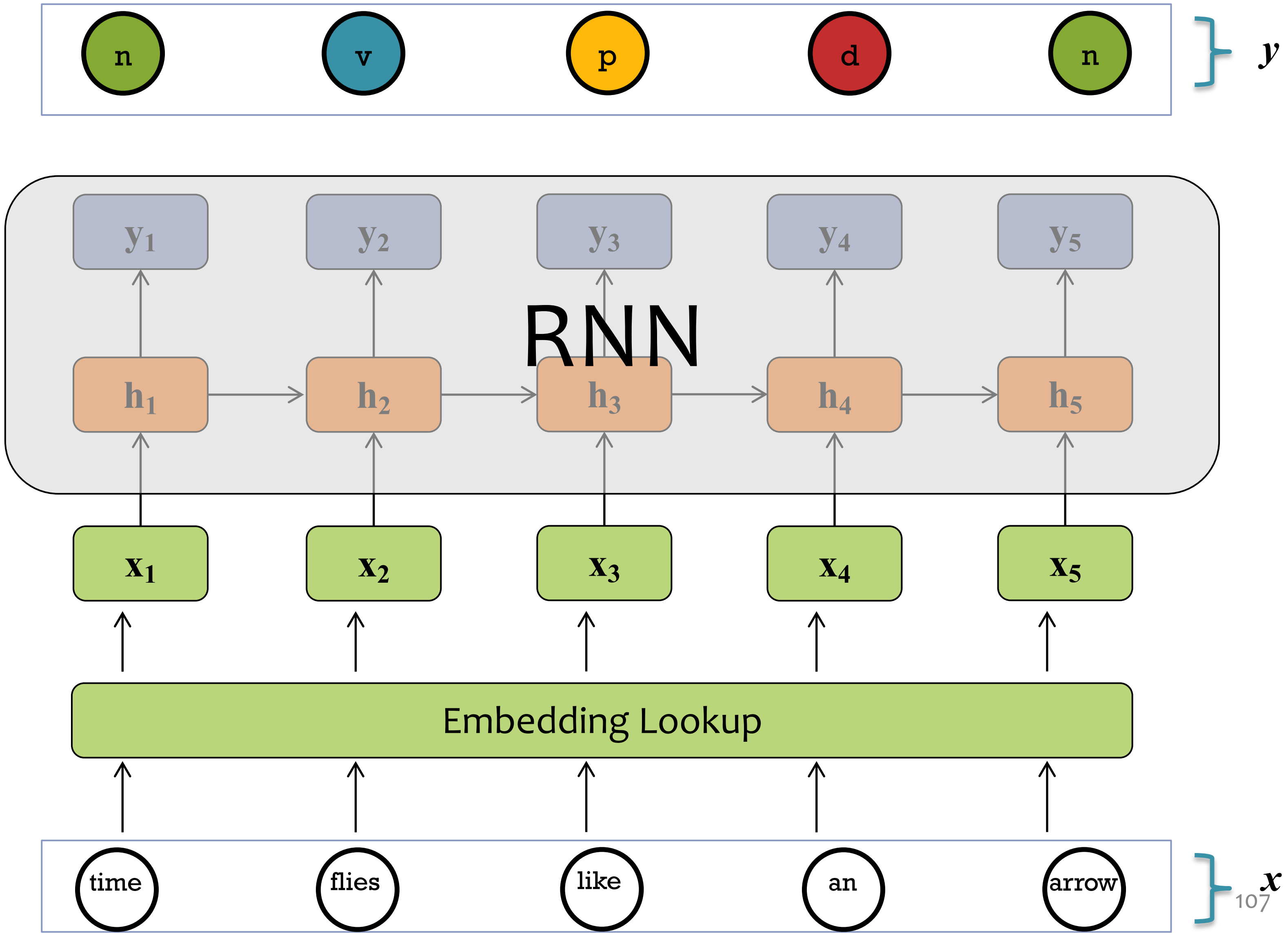
…the first step is always to look up the t'th word's embedding vector parameters and use said vector for the value of $\mathbf{x}_t$



Word Embeddings

In all the models we're about to consider (neural networks, RNNs, Transformers) that work with sentences…

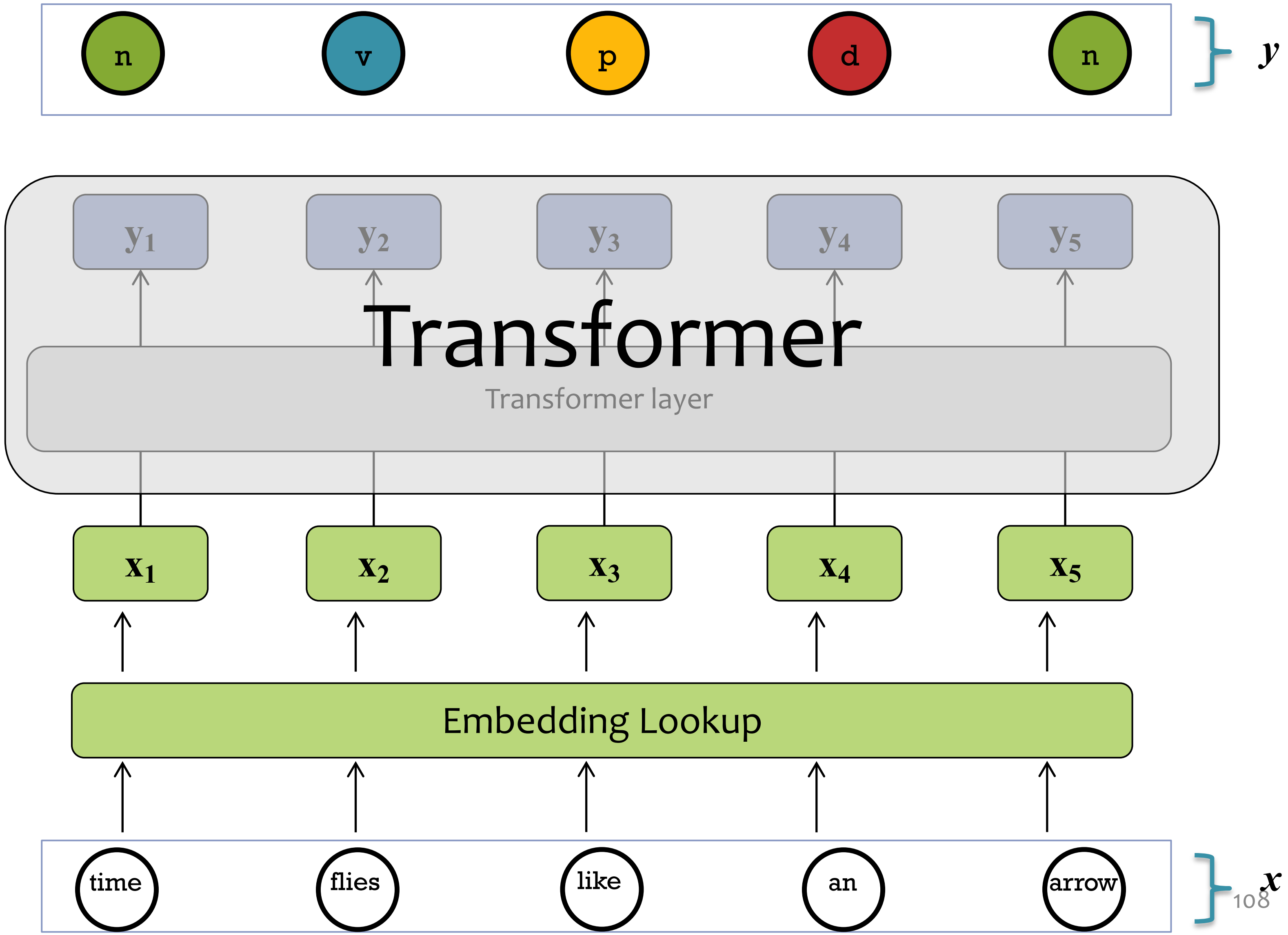
…the first step is always to look up the t'th word's embedding vector parameters and use said vector for the value of $\mathbf{x}_t$



Word Embeddings

In all the models we're about to consider (neural networks, RNNs, Transformers) that work with sentences…

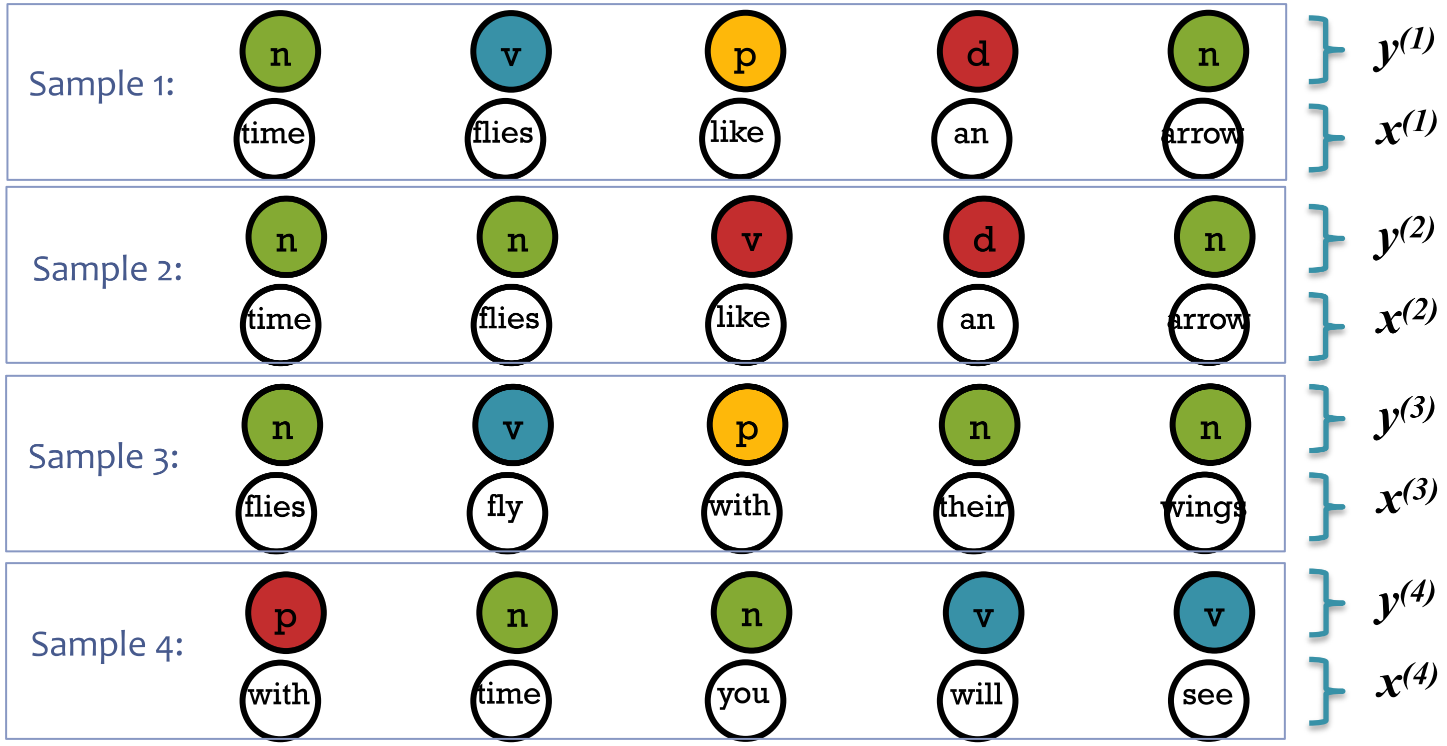
…the first step is always to look up the t'th word's embedding vector parameters and use said vector for the value of $\mathbf{x}_t$



SEQUENCE TAGGING

Dataset for Supervised Part-of-Speech (POS) Tagging

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



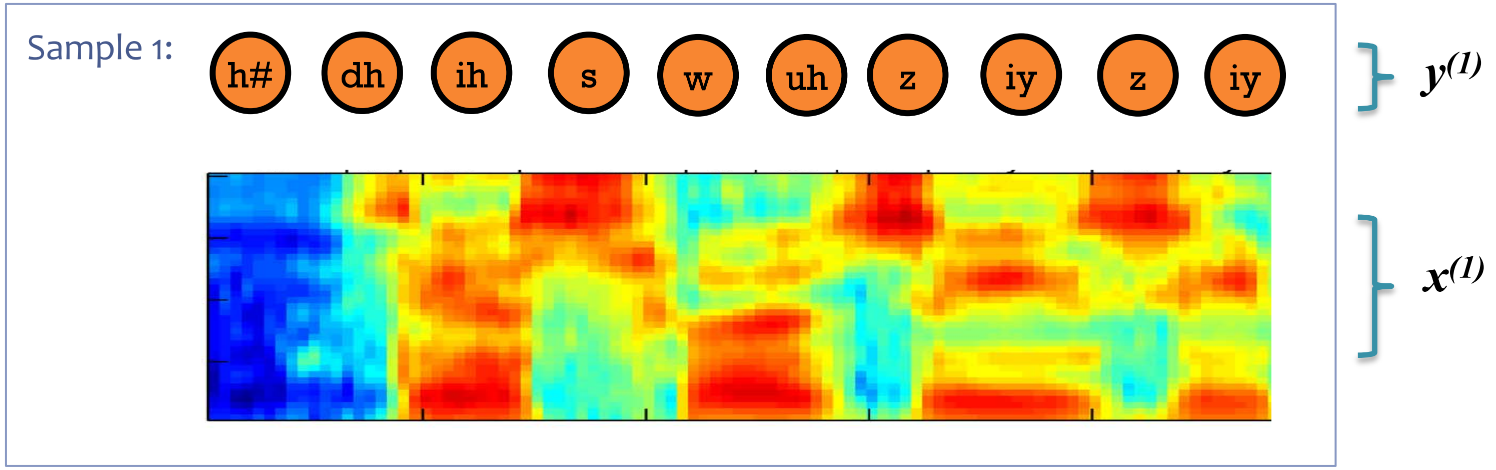
Dataset for Supervised Handwriting Recognition

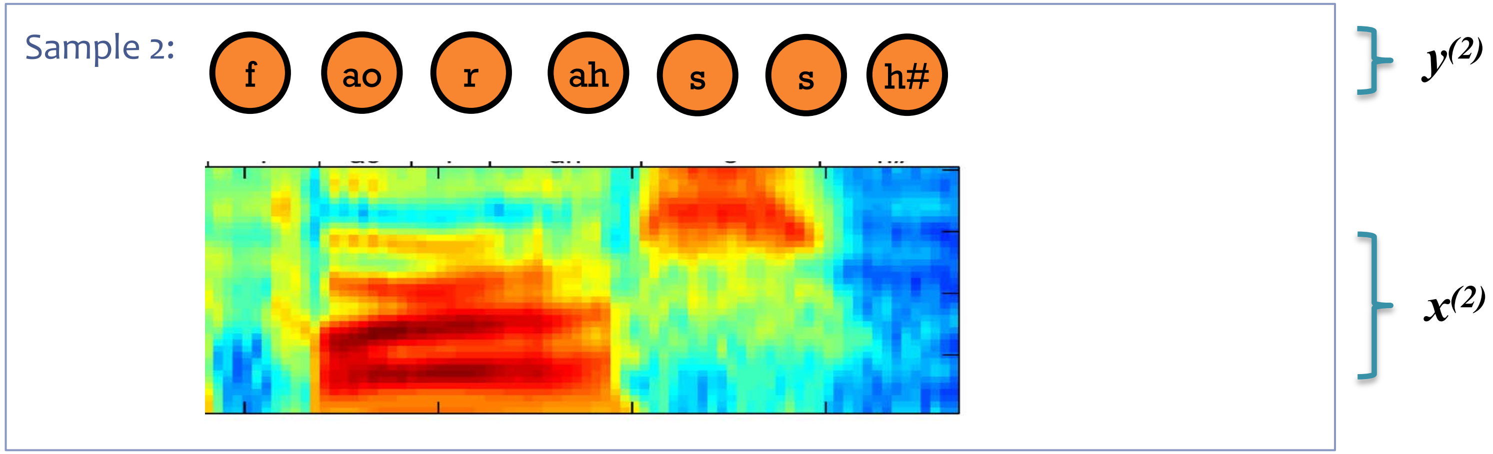
Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

Figures from (Chatzis & Demiris, 2013)

Dataset for Supervised Phoneme (Speech) Recognition

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

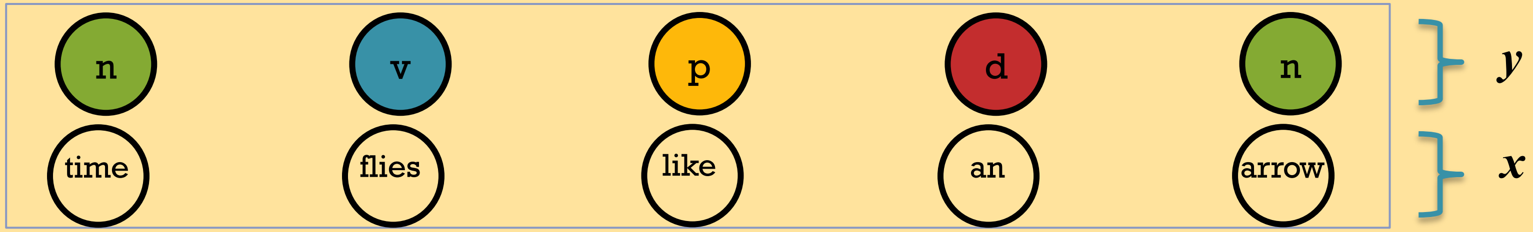




Figures from (Jansen & Niyogi, 2013)

Time Series Data

Poll Question 3: How could we apply the neural networks we've seen so far (which expect **fixed size input/output**) to a prediction task with **variable length input and output**?



Answer:

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

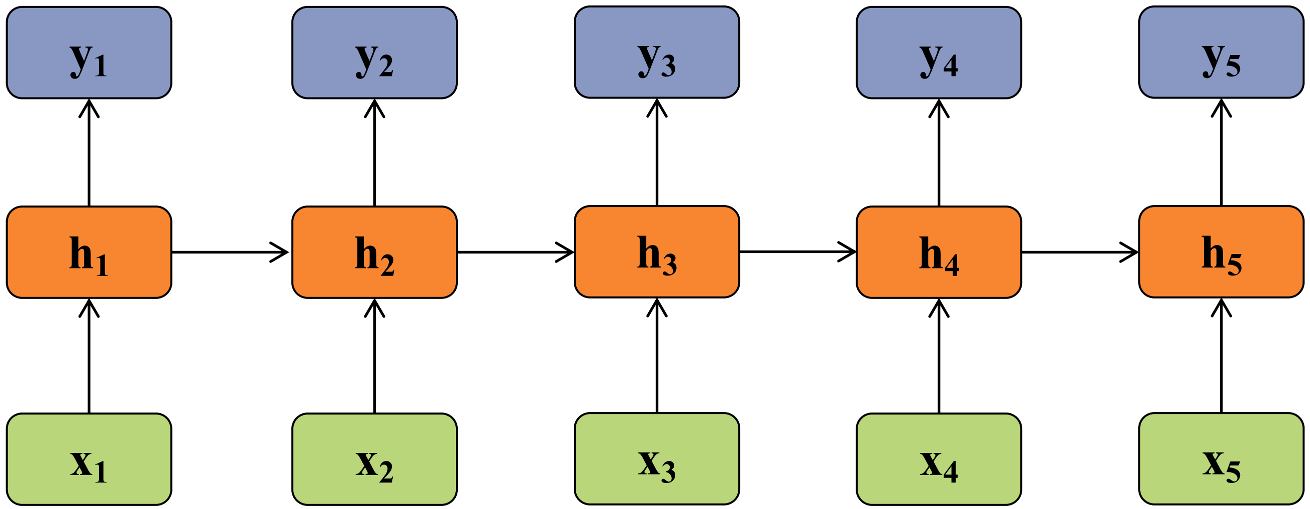
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

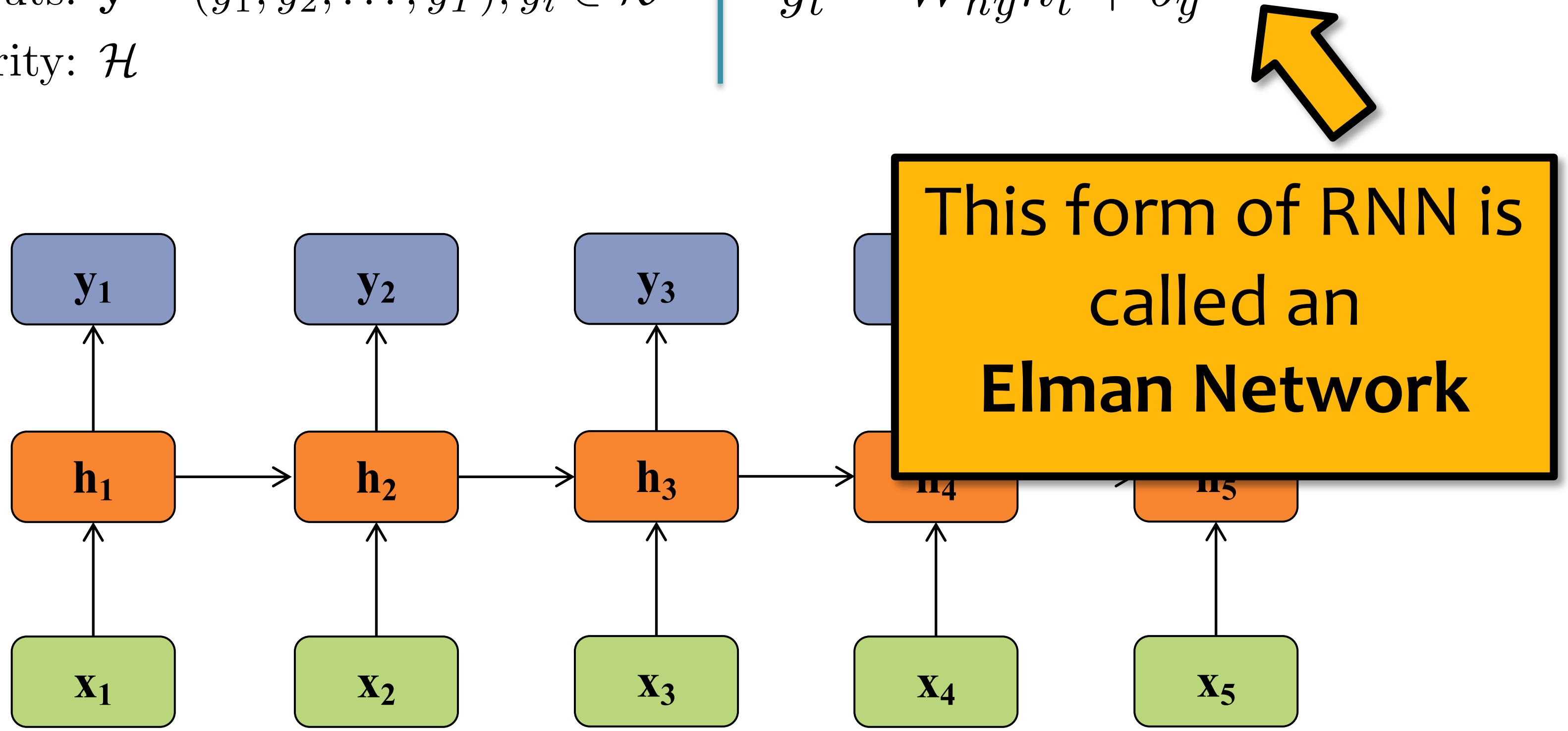
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



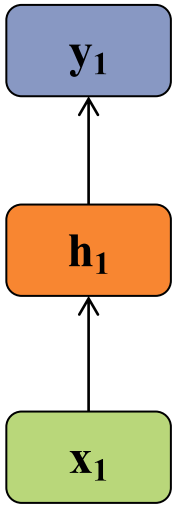
Recurrent Neural Networks (RNNs)

$$\begin{aligned} \text{inputs: } & \mathbf{x} = (x_1, x_2, \dots, x_T), x_i \in \mathcal{R}^I \\ \text{hidden units: } & \mathbf{h} = (h_1, h_2, \dots, h_T), h_i \in \mathcal{R}^J \\ \text{outputs: } & \mathbf{y} = (y_1, y_2, \dots, y_T), y_i \in \mathcal{R}^K \\ \text{nonlinearity: } & \mathcal{H} \end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



- If T=1, then we have a standard feed-forward neural net with one hidden layer, which requires **fixed size inputs/outputs**
- By contrast, an RNN can handle arbitrary length inputs/outputs because T can vary from example to example
- The key idea is that we reuse the same parameters at every timestep, always building off of the previous hidden state

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.

2. Choose each of these:

– Decision function

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

4. Train with SGD:

(take small steps opposite the gradient)

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

- Recurrent Neural Networks (RNNs) provide another form of **decision function**
- An RNN is just another differential function

- We'll just need a method of computing the gradient efficiently
- Let's use Backpropagation Through Time...

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

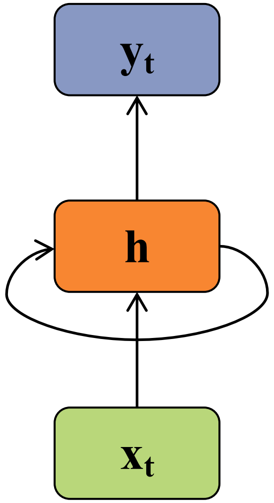
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

$$y_t = W_{hy} h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

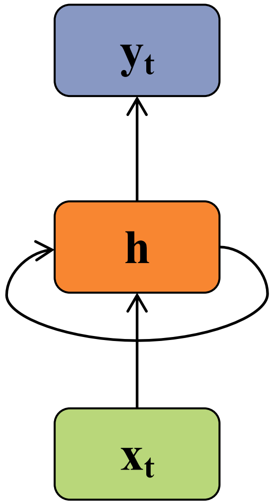
nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$

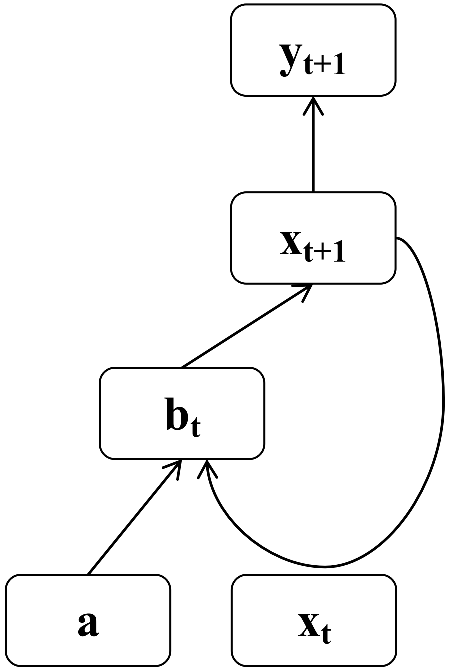
- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.



Background: Backprop through time

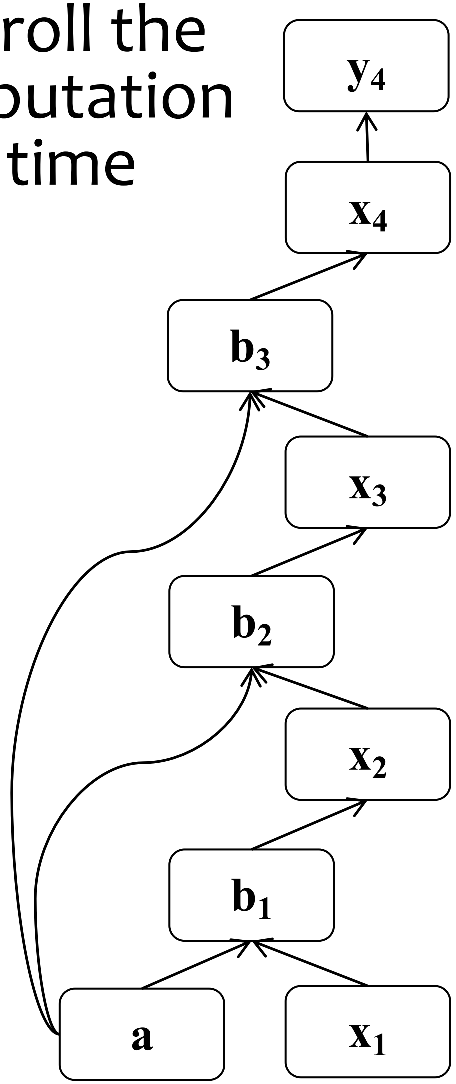
(Robinson & Fallside, 1987)
(Werbos, 1988)
(Mozer, 1995)

Recurrent neural network:



BPTT:

1. Unroll the computation over time



2. Run backprop through the resulting feed-forward network

Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

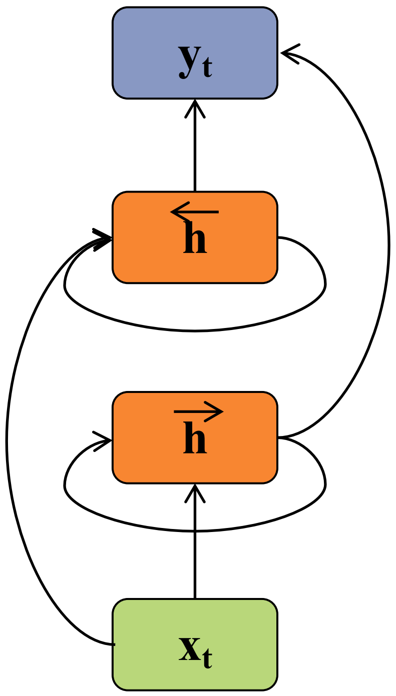
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

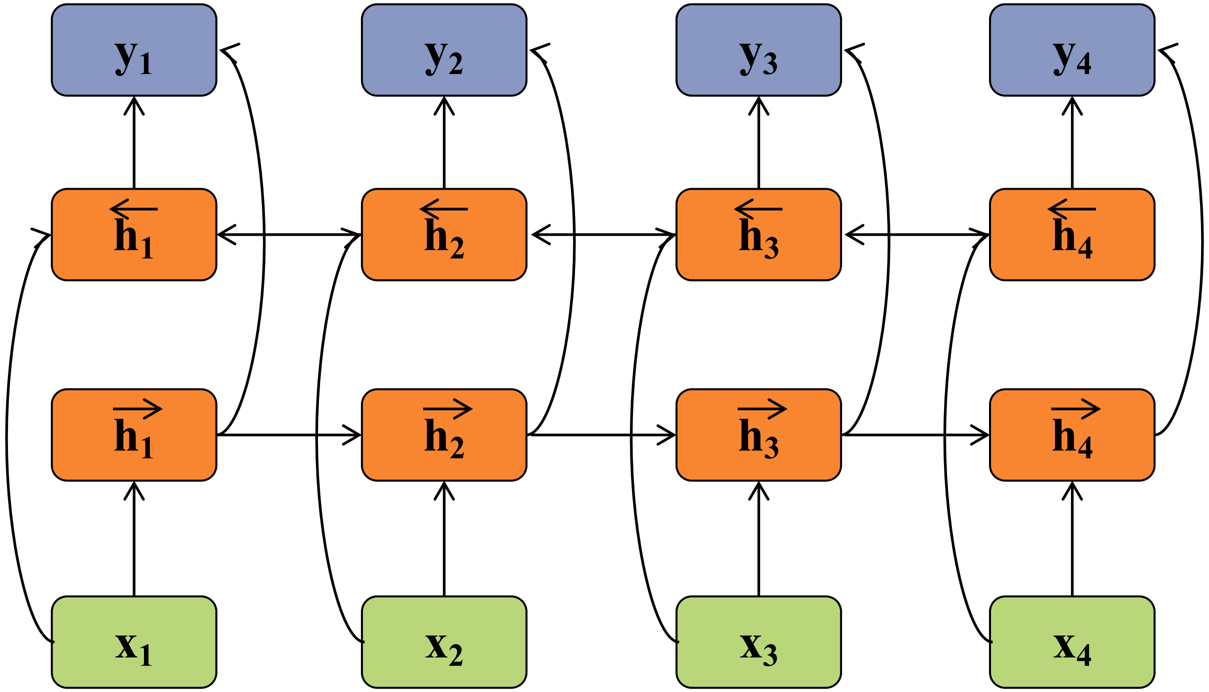
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Deep RNNs

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$

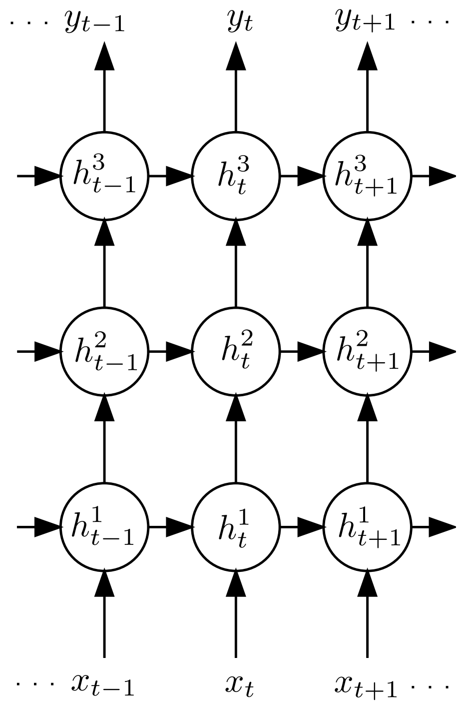


Figure from (Graves et al., 2013)

Deep Bidirectional RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

- Notice that the upper level hidden units have input from **two previous layers** (i.e. wider input)
- Likewise for the output layer

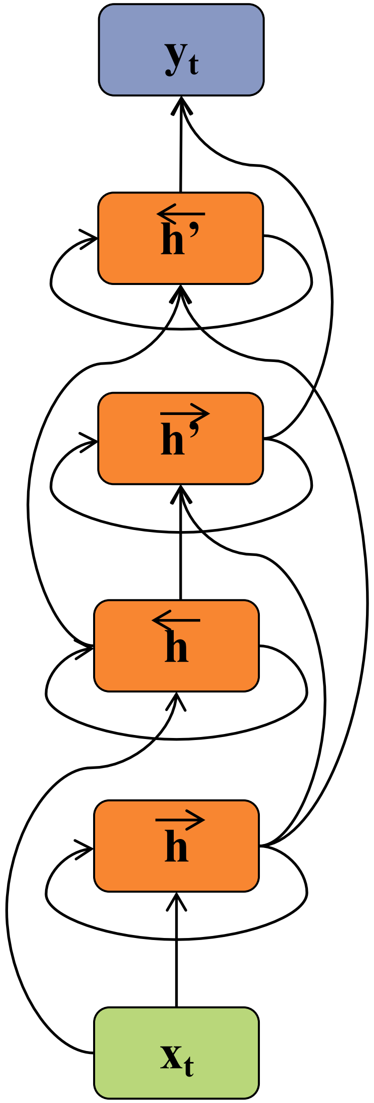


Figure from (Graves et al., 2013)

RNN / LSTM RESULTS

Dataset for Supervised Named Entity Recognition (NER)

- **Goal:** label the spans of persons, locations, organizations, times, etc. (aka. entities)
- **Data Representation:** to cast as a sequence tagging problem, we use Begin-Inside-Outside (BIO) tagging
- BIO tags distinguish between adjacent entities of the same type

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

Sample 1:	B-PER	I-PER	O	B-LOC	I-LOC		$y^{(1)}$
	Tenzing	Norgay	climbed	Mount	Everest		$x^{(1)}$
Sample 2:	B-PER	O	B-LOC	I-LOC			$y^{(2)}$
	Obama	visits	Paris	France			$x^{(2)}$
Sample 3:	B-PER	I-PER	B-ORG	I-ORG	O	O	$y^{(3)}$
	Steve	Jobs’	Apple	Inc.	changed	tech	$x^{(3)}$
Sample 4:	B-LOC	B-LOC	O	O			$y^{(4)}$
	Spain	Italy	win	medals			$x^{(4)}$

LSTM Empirical Results

- CoNLL-2003 is the most prominent dataset for NER
- F1 – higher is better
- blue dots are methods that use an LSTM
- an LSTM is the primary model behind the state-of-the-art (*ACE + document-context*)

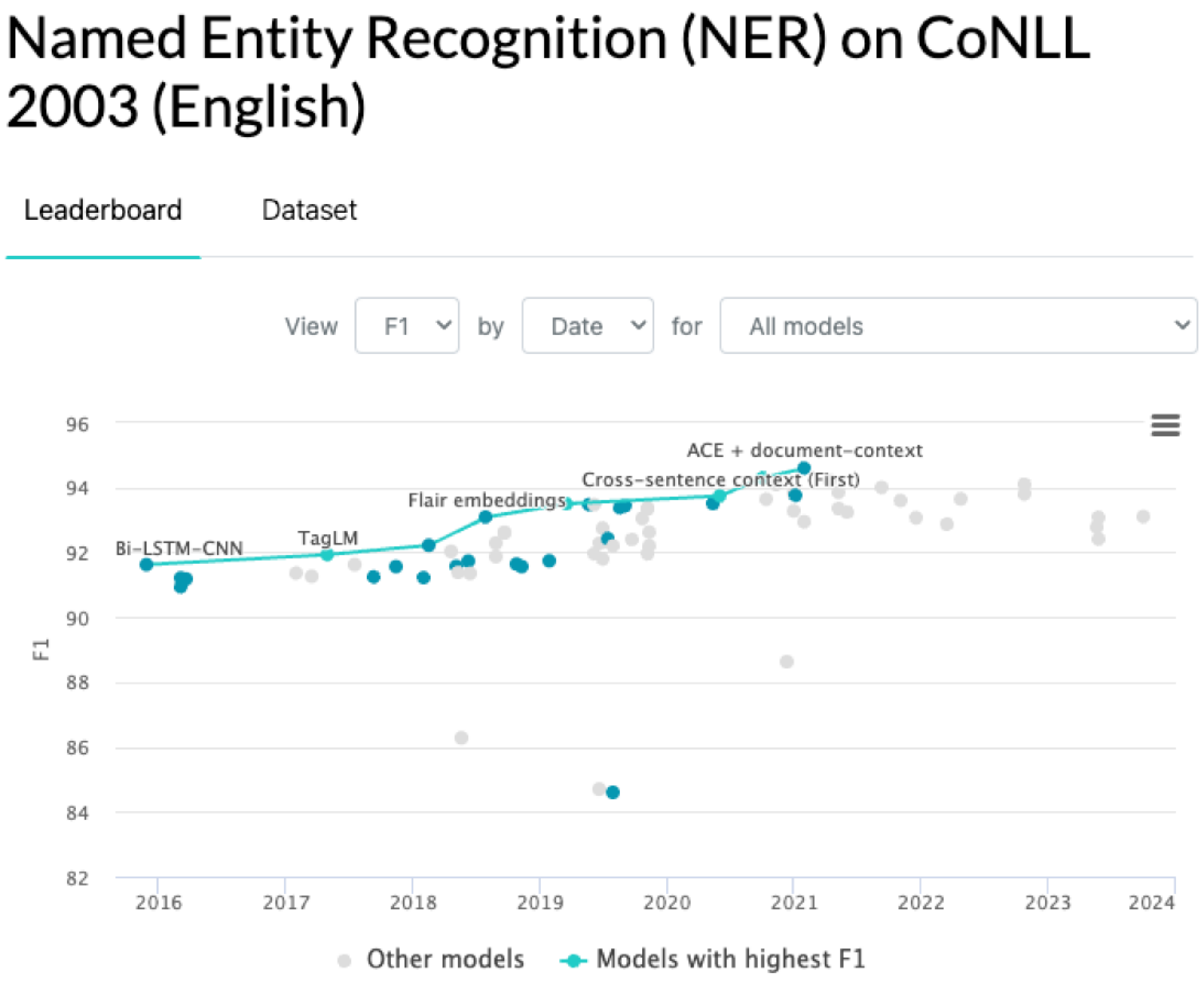


Figure from https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003

CNN & RNN 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