Lecture 11: Recognition

Instructor: Roni Sengupta
ULA: Andrea Dunn, William Li, Liujie Zheng

Course Website: Scan Me!
What is “Recognition”?

Next few slides adapted from Li, Fergus, & Torralba’s excellent short course on category and object recognition
What is “Recognition”?

• Verification: is that a lamp?
What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
What is “Recognition”?  
- Verification: is that a lamp?  
- Detection: where are the people?  
- Identification: is that Potala Palace?  
- Object categorization
What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization
- Scene and context categorization
What is “Recognition”? 
- Verification: is that a lamp? 
- Detection: where are the people? 
- Identification: is that Potala Palace? 
- Object categorization 
- Scene and context categorization 
- Activity / Event Recognition 

what are these people doing?
Recognition: What type of output?

- **Image classification**: person, sheep, dog
- **Object detection**: bounding boxes around objects
- **Semantic segmentation**: colored regions of objects
- **Instance segmentation**: individual objects with unique colors

Diagram examples include:
- A man herding sheep and a dog, with labels showing classification.
- The same scene with bounding boxes highlighting objects.
- Semantic segmentation showing different colors for sheep, dog, and man.
- Instance segmentation with each object in a different color.
Image Classification: a core task in computer vision

• Assume given set of discrete labels, e.g.
  \{cat, dog, cow, apple, tomato, truck, ...\}

\[
f(\text{apple}) = \text{“apple”} \\
f(\text{tomato}) = \text{“tomato”} \\
f(\text{cow}) = \text{“cow”}
\]
What Matters in Recognition?

• Data
  • More is always better (as long as it is good data)
  • Annotation is the hard part

• Representation
  • Low level: SIFT, HoG, GIST, edges
  • Mid level: Bag of words, sliding window, deformable model
  • High level: Contextual dependence
  • Deep learned features

• Learning Techniques
  • E.g. choice of classifier or inference method
What Matters in Recognition?

• Data
  • More is always better (as long as it is good data)
  • Annotation is the hard part

• Representation
  • Low level: SIFT, HoG, GIST, edges
  • Mid level: Bag of words, sliding window, deformable model
  • High level: Contextual dependence
  • Deep learned features

• Learning Techniques
  • E.g. choice of classifier or inference method
Data Sets

• PASCAL VOC
  • *Not* Crowdsourced, bounding boxes, 20 categories

• ImageNet
  • Huge, *Crowdsourced*, Hierarchical, *Iconic* objects

• SUN Scene Database, Places
  • *Not* Crowdsourced, 397 (or 720) scene categories

• LabelMe (Overlaps with SUN)
  • Sort of Crowdsourced, Segmentations, Open ended

• SUN *Attribute* database (Overlaps with SUN)
  • Crowdsourced, 102 attributes for every scene

• OpenSurfaces
  • Crowdsourced, materials

• **Microsoft COCO**
  • Crowdsourced, large-scale objects

... and many more https://paperswithcode.com/datasets?task=image-classification
Large Scale Visual Recognition Challenge (ILSVRC)

20 object classes | 22,591 images
1000 object classes | 1,431,167 images

Dalmatian

What Matters in Recognition?

• Data
  • More is always better (as long as it is good data)
  • Annotation is the hard part

• Representation
  • Low level: SIFT, HoG, GIST, edges
  • Mid level: Bag of words, sliding window, deformable model
  • High level: Contextual dependence
  • Deep learned features

• Learning Techniques
  • E.g. choice of classifier or inference method
History of recognition: Appearance-based models


History of recognition: Features and classifiers

Appearance manifolds + neural network

Support vector machines

Neural network

Statistics of feature responses, probabilistic classifier

Rectangle features, boosting

Osuna, Freund, Girosi (1997)

Rowley, Baluja, Kanade (1998)

Sung & Poggio (1994)

Schneiderman & Kanade (1998)

Viola & Jones (2001)
History of recognition: Deformable templates

Pictorial structures revisited

Discriminatively trained deformable part-based models

Felzenszwalb & Huttenlocher (2000)

Felzenszwalb et al. (2008)
What Matters in Recognition?

• Data
  • More is always better (as long as it is good data)
  • Annotation is the hard part

• Representation
  • Low level: SIFT, HoG, GIST, edges
  • Mid level: Bag of words, sliding window, deformable model
  • High level: Contextual dependence
  • Deep learned features

• Learning Techniques
  • E.g. choice of classifier or inference method
Training & Testing a Classifier

Training

Training Images

Training Labels

Image Features

Training

Learned Classifier

Dataset: ETH-80, by B. Leibe  Slide credit: D. Hoiem, L. Lazebnik
Training & Testing a Classifier

Training

Training Images

Training Labels

Image Features

Training

Learned Classifier

Testing

Test Image

Image Features

Learned Classifier

Prediction

Dataset: ETH-80, by B. Leibe  Slide credit: D. Hoiem, L. Lazebnik
Classifiers

• Nearest Neighbor
• kNN ("k-Nearest Neighbors")
• Linear Classifier
• Neural Network
• Deep Neural Network
• ...

Linear Classifiers
Score functions

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/
Parametric Approach

\[ f(x, W) \]

10 numbers, indicating class scores

[32x32x3] = 3072
array of numbers 0...1
(3072 numbers total)
Parametric Approach: Linear Classifier

\[ f(x, W) = Wx \]

[32x32x3] = 3072
array of numbers 0...1

10x1 10x3072

3072x1

10 numbers, indicating class scores

parameters, or “weights”

Slide adapted from Andrej Karpathy and Fei-Fei Li [http://vision.stanford.edu/teaching/cs231n/]
Parametric Approach: Linear Classifier

\[ f(x, W) = WX + b \]

10x1

10x3072

3072x1

(+b)

10x1

10 numbers, indicating class scores

parameters, or “weights”

[32x32x3] = 3072 array of numbers 0...1

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/
Linear Classifier

\[ f(x_i, W, b) = Wx_i + b \]

define a **score function**

data (image)

class scores

“weights”

“bias vector”

“parameters”

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/
Interpretation: Algebraic
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)
Parameters define a hyperplane for each class:

\[ f(x_i, W, b) = Wx_i + b \]

We can think of each class score as defining a distribution that is proportional to distance from the corresponding hyperplane.
Simpler example: binary classification

• Two classes (e.g., “cat” and “not cat”)
  • AKA “positive” and “negative” classes
Simpler example: binary classification

- Find linear function (*hyperplane*) to separate positive and negative examples

\[
x_i \text{ positive} : \quad x_i \cdot w + b \geq 0 \\
x_i \text{ negative} : \quad x_i \cdot w + b < 0
\]

Which hyperplane is best? We need a **loss function** to decide
Linear classification

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
</tr>
</tbody>
</table>

Output scores

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. (optimization)
Loss functions

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x)</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>f(x)</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>f(x)</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

A **loss function** tells how good our current classifier is. Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^N$$

Where $x_i$ is image and $y_i$ is (integer) label.

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$
Softmax classifier

- Interpret Scores as unnormalized log probabilities of classes
  
  Squashes values into probabilities ranging from 0 to 1

Example with three classes:

\[
[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]
\]
# Softmax classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

<table>
<thead>
<tr>
<th></th>
<th>0.2</th>
<th>-0.5</th>
<th>0.1</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.5</td>
<td>1.3</td>
<td>2.1</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.25</td>
<td>0.2</td>
<td>-0.3</td>
</tr>
<tr>
<td>$W$</td>
<td>56</td>
<td>231</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>$x_i$</td>
<td>1.1</td>
<td>3.2</td>
<td>-1.2</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f(x_i; W, b)$</td>
<td>-96.8</td>
<td>437.9</td>
<td>61.95</td>
<td></td>
</tr>
</tbody>
</table>

Softmax “probabilities”

- Cat score: $\frac{e^{f_{c}}}{\sum_{j} e^{f_{j}}}$ = 0.06
- Dog score: $\frac{e^{f_{d}}}{\sum_{j} e^{f_{j}}}$ = 0.82
- Ship score: $\frac{e^{f_{s}}}{\sum_{j} e^{f_{j}}}$ = 0.12
Cross-entropy loss

\[ f(s)_i = \frac{e^{s_i}}{\sum_j e^{s_j}} \quad CE = - \sum_i t_i \log(f(s)_i) \]

\( f(x_i, W) = Wx_i \) (score function)
Summary

• Have score function and loss function
  • Currently, score function is based on linear classifier
  • Next, will generalize to convolutional neural networks
• Find $W$ and $b$ to minimize loss

$$L = \frac{1}{N} \sum_{i} -\log \left( \frac{e^{f_{yi}}}{\sum_{j} e^{f_{j}}} \right) + \lambda \sum_{k} \sum_{l} W_{k,l}^2$$

Average of cross-entropy loss over all training examples

Regularization term
(Deep) Neural Networks
History of recognition: Neural networks

- Perceptrons: Rosenblatt (1958)
- Minsky & Papert (1969)
- LeNet-5: LeCun et al. (1998)
- AlexNet: Krizhevsky et al. (2012)

- Rosenblatt (1958): Perceptrons
- Minsky & Papert (1969)
- Rumelhart, Hinton & Williams (1986)
- LeCun et al. (1998)
- Krizhevsky et al. (2012)
Neural networks

(Before) Linear score function: \[ f = Wx \]
Neural networks

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network

\[
f = W_2 \max(0, W_1 x)
\]

Non-linear Activation Function
(many other choices exist)
Neural networks

(Before) Linear score function:
\[ f = Wx \]

(Now) 2-layer Neural Network
\[ f = W_2 \max(0, W_1x) \]

![Diagram of a 2-layer neural network with dimensions and operations explained.](image)
Neural networks

(Before) Linear score function: \[ f = Wx \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

- Total number of weights to learn:
  \[ 3,072 \times 100 + 100 \times 10 = 308,200 \]
Neural networks

(Before) Linear score function: \[ f = Wx \]

(Now) 2-layer Neural Network or 3-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]

also called “Multi-Layer Perceptrons” (MLPs)
Neural networks

• Very coarse generalization of neural networks:
  • Linear functions chained together and separated by non-linearities (activation functions), e.g. “max”

  \[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]

• Why separate linear functions with non-linear functions?
• Very roughly inspired by real neurons
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[
\begin{align*}
    x & & x \geq 0 \\
    \alpha(e^x - 1) & & x < 0
\end{align*}
\]
Neural network architecture

• Computation graph for a 2-layer neural network
Deep networks typically have many layers and potentially millions of parameters.
Optimizing parameters with gradient descent

• How do we find the best $\mathbf{W}$ and $\mathbf{b}$ parameters?

• In general: gradient descent
  1. Start with a guess of a good $\mathbf{W}$ and $\mathbf{b}$ (or randomly initialize them)
  2. Compute the loss function for this initial guess and the gradient of the loss function
  3. Step some distance in the negative gradient direction (direction of steepest descent)
  4. Repeat steps 2 & 3

• Note: efficiently performing step 2 for deep networks is called backpropagation

The Learning Cycle: Forward Propagation

Using the current weights of the model calculate the hidden layer neurons, then calculate the output p.
The Learning Cycle: Backward Propagation

Want $p$ to be larger...

$(Y - p)$ is positive

How do we change our weights?
Gradient descent: walk in the direction opposite gradient

- **Q**: How far?
- **A**: Step size: *learning rate*
- Too big: will miss the minimum
- Too small: slow convergence
What Matters in Recognition?

• Data
  • More is always better (as long as it is good data)
  • Annotation is the hard part

• Representation
  • Low level: SIFT, HoG, GIST, edges
  • Mid level: Bag of words, sliding window, deformable model
  • High level: Contextual dependence
  • Deep learned features

• Learning Techniques
  • E.g. choice of classifier or inference method

Combine!
Convolutional Neural Networks (CNNs)
Convolutional neural networks

A bit of history:
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]
Fast-forward to today: ConvNets are everywhere
f = linear classifier

Feature Extraction

10 numbers giving scores for classes

training

10 numbers giving scores for classes

training
Last layer of most CNNs is a linear classifier

This piece is just a linear classifier

Input Pixels → Perform everything with a big neural network, trained end-to-end → Ans

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable
Visualizing AlexNet in 2D with t-SNE

(c) DeCAF$_1$  
(d) DeCAF$_6$

(2D visualization using t-SNE)  

Convolutional neural networks

- Layer types:
  - Convolutional layer
  - Pooling layer
  - Fully-connected layer
AlexNet: An Early Example

Convolution Layer

32x32x3 image -> preserve spatial structure

32 height
32 width
3 depth
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution (Recap)

- Same as cross-correlation, except that the kernel is “flipped” (horizontally and vertically)

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

- Convolution is **commutative** and **associative**
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

Number of weights: $5 \times 5 \times 3 + 1 = 76$
(vs. 3072 for a fully-connected layer)
(+1 for bias)

Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”
Convolutions Layer

32x32x3 image
5x5x3 filter $w$

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5*5*3 = 75$-dimensional dot product + bias)

$w^T x + b$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
consider a second, green filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

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

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

(total number of parameters: 6 x (75 + 1) = 456)
Padding & Stride in CNN

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size \((N, C_{in}, H, W)\) and output \((N, C_{out}, H_{out}, W_{out})\) can be precisely described as:

\[
\text{out}(N_i, C_{out,j}) = \text{bias}(C_{out,j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out,j}, k) \ast \text{input}(N_i, k)
\]

where \(\ast\) is the valid 2D cross-correlation operator, \(N\) is a batch size, \(C\) denotes a number of channels, \(H\) is a height of input planes in pixels, and \(W\) is width in pixels.

\[
H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor
\]

\[
W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor
\]

\(\text{padding}=1\), \(\text{stride}=2\)
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \]
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
AlexNet: An Early Example

How Else to Shrink the Model Size?

Pooling Layer:
- Max Pooling
- Other pooling options like average pooling are also used
Convolutional Networks

Learnable 3x3 Convolutional Kernels

Input Image (Grayscale) → 2D Conv. → Conv. Feature Maps → 2D Max Pooling → Pooled Feature Maps

80 x 120 x 1 → 80 x 120 x 4 → 40 x 60 x 4
AlexNet: An Early Example

Convolutional Networks

Learnable FC Layer Weight Matrix

\[ W \in \mathbb{R}^{d \times C} \]

- \( d \) - feature dimensionality (4800 in this example)
- \( C \) - number of classes

Pooled Feature Maps

2D Conv.

40 x 60 x 4

40 x 60 x 2

Flatten

1 x 4800

Slide Credits: Gedas Bertasius
Convolutional Networks

Learnable FC Layer Weight Matrix

\[ W \in \mathbb{R}^{d \times C} \]

- \( d \) - feature dimensionality (4800 in this example)
- \( C \) - number of classes

Pooled Feature Maps

2D Conv. \( \rightarrow \) \( 40 \times 60 \times 2 \)

Flatten \( \rightarrow \) \( 1 \times 4800 \)

FC Layer + Softmax \( \rightarrow \) \( 1 \times C \)

Cat: 0.01
Dog: 0.03
Penguin: 0.91

Slide Credits: Gedas Bertasius
Where Models Learn Features of an Image

Low-level features
Lines, oriented edges

Mid-level features
Combine edges: curves, shapes

High-level features
Combine shapes: objects, scenes

Predictor
Process features and predict output
Preview

Low-level features → Mid-level features → High-level features → Linearily separable classifier

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3
AlexNet (2012)

6M parameters in total

Input image (RGB)  CONV1  CONV2  CONV3  CONV4  CONV5  FC6  FC7  FC8

Image input  5 Convolution layers  3 Fully-connected layers

Input: 224 x 224
Output: 1,000-D vector (probabilities over 1,000 ImageNet categories)

“AlexNet”
[Krizhevsky et al. NIPS 2012]

“GoogLeNet”
[Szegedy et al. CVPR 2015]

“VGG Net”
[Simonyan & Zisserman, ICLR 2015]

“ResNet”
[He et al. CVPR 2016]
Big picture

• A convolutional neural network can be thought of as a function from images to class scores
  • With millions of adjustable weights...
  • ... leading to a very non-linear mapping from images to features / class scores.
  • We will set these weights based on classification accuracy on training data...
  • ... and hopefully our network will generalize to new images at test time
Data is key—enter ImageNet

- ImageNet (and the ImageNet Large-Scale Visual Recognition Challenge, aka ILSVRC) has been key to training deep learning methods

- **ILSVRC**: 1,000 object categories, each with ~700-1300 training images. Test set has 100 images per categories (100,000 total).

- Standard ILSVRC error metric: top-5 error
  - if the correct answer for a given test image is in the top 5 categories, your answer is judged to be correct
Performance improvements on ILSVRC

• ImageNet Large-Scale Visual Recognition Challenge
• Held from 2011-2017
• 1000 categories, 1000 training images per category
• Test performance on held-out test set of images

AlexNet

Deep learning era

Pre-deep learning era

ImageNet competition results

Year

0.0 0.1 0.2 0.3 0.4 0.5
Error rate
Closer look at Convolution (Extra Slides)
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7

7 x 7 input (spatially) assume 3 x 3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)  
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7×7 input (spatially) assume 3×3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:

\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):

- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = 2.33 \)
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)
\[(N - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- e.g. input 7x7
- 3x3 filter, applied with **stride 1**
- **pad with 1 pixel** border => what is the output?

**7x7 output!**

Output filter size: \((N + 2\times pad - F)/S + 1\)
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E.g. input $7 \times 7$

3x3 filter, applied with **stride 1**

Pad with **1 pixel** border => what is the output?

**7x7 output!**

In general, common to see CONV layers with

stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

E.g. $F = 3$ => zero pad with 1

$F = 5$ => zero pad with 2

$F = 7$ => zero pad with 3
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Output volume size:
\((32+2 \times 2-5)/1+1 = 32\) spatially, so
\(32 \times 32 \times 10\)
Examples time:

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

Number of parameters in this layer?
Examples time:

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

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params ( +1 for bias)
=> 76*10 = 760
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Convolutional layer—properties

- Small number of parameters to learn compared to a fully connected layer
- Preserves spatial structure—output of a convolutional layer is shaped like an image
- **Translation equivariant**: passing a translated image through a convolutional layer is (almost) equivalent to translating the convolution output (but be careful of image boundaries)
Self-study
ConvNetJS CIFAR-10 demo

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Train a neural network for classification on CIFAR10 dataset in google colab

Google Colab Page
Where to Look for More Information

- Explore existing computer vision and machine learning frameworks
  - https://pytorch.org/
  - https://www.tensorflow.org/
  - https://keras.io/
  - https://opencv.org/

- Watch more in-depth lecture series
  - The Ancient Secrets of Computer Vision - Joseph Redmon
  - Deep Learning Specialization - Andrew Ng

- Checkout other online courses and guides
  - https://ai.google/education/
  - https://www.udacity.com/course/deep-learning-pytorch--ud188
Slide Credits

• **CS5670, Introduction to Computer Vision**, Cornell Tech, by Noah Snavely.

• **CS 543 Computer Vision**, by Stevlana Lazebnik, UIUC.

• **EECS 442 Computer Vision**, by Justin Johnson & David Fouhey, U Michigan.