Lecture: Convolutional Neural Networks

Juan Carlos Niebles and Ranjay Krishna
Stanford Vision and Learning Lab
What should you take away from this class?

<table>
<thead>
<tr>
<th>Pixels</th>
<th>Segments</th>
<th>Images</th>
<th>Videos</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutions</td>
<td>Resizing</td>
<td>Recognition</td>
<td>Motion</td>
<td>Neural networks</td>
</tr>
<tr>
<td>Edges</td>
<td>Segmentation</td>
<td>Detection</td>
<td>Tracking</td>
<td>Convolutional neural networks</td>
</tr>
<tr>
<td>Descriptors</td>
<td>Clustering</td>
<td>Ontology</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin
Recall: image classification pipeline

Training Images

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Image Features

Training Labels

Training

Learned Classifier

Test Image

Image Features

Learned Classifier

Prediction
Let's change the features by adding another layer
Recall: 2 or 3-layer network

- linear classifier:
  \[ y = W x \]

- 2-layer network:
  \[ y = W_2 \max(0, W_1 x) \]

- 3-layer network:
  \[ y = W_3 \max(0, W_2 \max(0, W_1 x)) \]
Today's agenda

- Backprop in neural networks
  - Convolutional neural networks
  - Architecture design
  - Exam and brief class overview
  - En fin
Recall: Gradient Descent Pseudocode

for i in {0,...,num_epochs}:
    for x, y in data:
        ŷ = f(x, W)
        L = \sum_i L_i (y_i, \hat{y}_i)
        \frac{dL}{dW} = ??
        W := W - \alpha \frac{dL}{dW}
Backprop using calculus

\[ \text{Loss} = L(\hat{y}, y) \]
\[ = L(f(x, W), y) \]
\[ = L(W_3 \max(0, W_2 \max(0, W_1 x)), y) \]

\[ \frac{dL}{dW} = \nabla_W L(W_3 \max(0, W_2 \max(0, W_1 x)), y) \]

which can get messy really fast. And these are small 3-layer neural networks.

But remember, we learned about the chain rule?

\[ \frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW} \]
Generalizing the chain rule for neural networks

• Let's start with an easy 1D example:

\[ a = wx \]
\[ \hat{y} = \max(0, a) \]

\[ \frac{dL}{da} = \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{da} \]

\[ \frac{dL}{dw_1} = \frac{dL}{da} \cdot \frac{da}{dw_1} \]

At the end, we want to calculate \( \frac{dL}{dw} \)
Generalizing the chain rule for neural networks

Let's assume $x = 2$, $w_1 = 3$

- Let's assume the loss is 1

\[
a = wx
\]
\[
\hat{y} = \max(0, a)
\]

\[
\frac{dL}{da} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{da}
\]

\[
\frac{dL}{dw_1} = \frac{dL}{da} \frac{da}{dw_1}
\]

\[
\frac{dL}{dx} = \frac{dL}{da} \frac{da}{dx}
\]
Let's try another harder example

\[ a = w + x \]
\[ \hat{y} = e^a \]

\[
\frac{dL}{da} = \frac{dL}{d\hat{y}} \quad \frac{d\hat{y}}{da} \\
\frac{dL}{dw_1} = \frac{dL}{da} \quad \frac{dL}{dw_1} \\
\frac{dL}{dx} = \frac{dL}{da} \quad \frac{dL}{dx} 
\]
General rules for calculating gradients

- Addition: gradient distributor

\[
\frac{dL}{dw} = \frac{dL}{dy} \quad \frac{dL}{dy} = 1 \\
\frac{dL}{dx} = \frac{dL}{dy} \quad \frac{dy}{dx} = 1 \\
\frac{dy}{dw} = 1 \\
\frac{dL}{w} = \frac{dL}{y} \\
\]

\[
y = x + w
\]
General rules for calculating gradients

- Multiplication: gradient swap multiplier

\[
\frac{dL}{dw} = x \frac{dL}{dy} \\
\frac{dL}{dx} = w \frac{dL}{dy} \\
y = x \times w \\
\frac{dL}{dy} \\
\frac{dy}{dx} = w \\
\frac{dy}{dw} = x
\]
General rules for calculating gradients

- ReLU or max(0, …): gradient multiplexer

\[
\frac{dL}{dw} = \begin{cases} 
\frac{dL}{dy} & \text{if } w \geq x \\ 
0 & \text{otherwise}
\end{cases}
\]

\[
\frac{dL}{dx} = \begin{cases} 
\frac{dL}{dy} & \text{if } x \geq w \\ 
0 & \text{otherwise}
\end{cases}
\]

\[y = \max(x, w)\]
General rules for calculating gradients

- Exponentiation: gradient output multiplier

\[
\frac{dL}{dx} = y \frac{dL}{dy} \\
\frac{dL}{dy} \\
\frac{dy}{dx} = e^x
\]
Backprop with gradients

• Remember that the dimensions of a variable and its gradients have to be the same.

• Dimensions of $x$ and $\frac{dL}{dx}$ are the same

\[
\frac{dL}{dw} = \frac{dy}{dw} \frac{dL}{dy}
\]

\[
\frac{dL}{dx} = \frac{dy}{dx} \frac{dL}{dy}
\]

$y = f(x, W)$
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$

\[
\frac{dL}{dw} = \frac{dL}{dy} \frac{dy}{dw}
\]

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

\[
\frac{dL}{dy}
\]

\[
\frac{dL}{dx} = \frac{dL}{dy} \frac{dy}{dx}
\]
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{dL}{dy} \in \mathbb{R}^N$
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{dL}{dy} \in \mathbb{R}^N$

So, $W \in \mathbb{R}^{N \times M}$, then $\frac{dL}{dw} \in \mathbb{R}^{N \times M}$
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{dL}{dy} \in \mathbb{R}^N$
- So, $W \in \mathbb{R}^{N \times M}$, then $\frac{dL}{dw} \in \mathbb{R}^{N \times M}$
- So, what are the dimensions of $\frac{dy}{dx}$ and $\frac{dy}{dw}$?
Backprop with gradients

• If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$

• Similarly, if $y \in \mathbb{R}^N$, then $\frac{dL}{dy} \in \mathbb{R}^N$

• So, $W \in \mathbb{R}^{N \times M}$, then $\frac{dL}{dw} \in \mathbb{R}^{N \times M}$

• So, what are the dimensions of $\frac{dy}{dx}$ and $\frac{dy}{dw}$?

$$\frac{dy}{dx} \in \mathbb{R}^{M \times N}$$

$$\frac{dy}{dx} = \begin{bmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_N}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{dy_1}{dx_M} & \cdots & \frac{dy_N}{dx_M} \end{bmatrix}$$
Backprop with gradients

• If \( x \in \mathbb{R}^M \), then \( \frac{dL}{dx} \in \mathbb{R}^M \)

• Similarly, if \( y \in \mathbb{R}^N \), then \( \frac{dL}{dy} \in \mathbb{R}^N \)

• So, \( W \in \mathbb{R}^{N \times M} \), then \( \frac{dL}{dw} \in \mathbb{R}^{N \times M} \)

• So, what are the dimensions of \( \frac{dy}{dx} \) and \( \frac{dy}{dw} \)?

\[
\begin{align*}
W & \quad \frac{dL}{dw} = \frac{dy}{dw} \frac{dL}{dy} \\
\frac{dL}{dx} & = \frac{dy}{dx} \frac{dL}{dy}
\end{align*}
\]

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

\[
\frac{dy}{dw} \in \mathbb{R}^{(N \times M) \times N}
\]

\[
\frac{dy}{dx} = \begin{bmatrix}
\frac{dy_1}{dw_1} & \cdots & \frac{dy_N}{dw_1} \\
\vdots & \ddots & \vdots \\
\frac{dy_1}{dw_{NM}} & \cdots & \frac{dy_N}{dw_{NM}}
\end{bmatrix}
\]
Wrap up: Backprop allows us to build arbitrarily large Neural Networks and gradients only need to know local input and output information to calculate gradients.
Today's agenda

• Backprop in neural networks
• Convolutional neural networks
• Architecture design
• Exam and brief class overview
• En fin
Recall: we can featurize images into a vector

<table>
<thead>
<tr>
<th>Raw pixels</th>
<th>Raw pixels + (x,y)</th>
<th>PCA</th>
<th>LDA</th>
<th>BoW</th>
<th>BoW + spatial pyramids</th>
</tr>
</thead>
</table>

But these representations lose spatial information
Bag of words had the same problem
Solution: pyramids

Locally orderless representation at several levels of spatial resolution
Can we design a neural network to retain spatial information?

• YES

• We have already designed convolutions to be spatially invariant.
Recall: well designed convolutions give important features

Just like the weights in the neural network, we can learn the weights of a convolution filter as well

\[
\begin{array}{ccc}
0 & -\frac{1}{2} & 0 \\
0 & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
\end{array}
\]

\* 

= 

\[
\text{image}
\]
The convolution layer

- Traditionally we used only 1 2D filter at a time.
- And operated only on black-and-white images
Calculating the size of activation after applying convolution:

- Stride $S = 1$, Padding $P = 2$, Spatial extend $F = 3$ for a 3x3 kernel
- Given input size $(W_1 = 32, H_1 = 32)$, $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- So, $W_2 = \frac{W_1 - F + 2P}{S} + 1 = \frac{32 - 3 + 2}{1} + 1 = 32$
The convolution layer

- We can apply 3 filters to operate over all three color channels
The convolution layer

• Let's represent this new kernel as a tensor as well

32x32x3  3x3 kernel  32x32x1
The convolution layer

- We can apply $K = 64$ multiple kernels in parallel to learn different kinds of features

32x32x3  3x3 kernel 64 filters  32x32x64

$K = 64$ multiple kernels in parallel to learn different kinds of features.
The convolution layer

- How many weights would a linear layer need to convert the input to the following feature?
- How many weights would a convolution filter need?

\[
\begin{align*}
\text{32x32x3} & \rightarrow \text{linear layer} & \rightarrow \text{3x3x3x64 Conv layer} & \rightarrow \text{32x32x64}
\end{align*}
\]
Stacking Convolutions

A convolutional neural network architecture

Note that the non-linear ReLU layers between conv layers are not shown
Today's conv networks have pool layers

Pooling layers reduce the spatial dimensions by 2

$3 \times 3 \times 3 \times 64$

Conv

$32 \times 32 \times 64$

ReLU

$32 \times 32 \times 64$

MaxPool

$16 \times 16 \times 64$
Today's conv networks have pool layers

Pooling layers reduce the spatial dimensions by 2

32x32x3 → 3x3x3x64Conv → 32x32x64 → ReLU → 32x32x64 → MaxPool → 16x16x64

Pooling reduces the spatial dimensions by 2.
Similar to Filter example #2: Image Segmentation from lecture #2

- Image segmentation based on a simple threshold:

\[
g[n, m] = \begin{cases} 
255, & f[n, m] > 100 \\
0, & \text{otherwise.}
\end{cases}
\]
Today's agenda

• Backprop in neural networks
• Convolutional neural networks
• Architecture design
• Exam and brief class overview
• En fin
Convolutional neural network

Conv Block ReLU Conv Block ReLU Pool Conv Block ReLU Conv Block ReLU Pool Conv Block ReLU Conv Block ReLU Pool

32×32×3 Input

10×1 Output
Frank Rosenblatt, ~1957: Perceptron

implemented in circuits / electronics in hardware

no loss function
no backprop
Widrow and Hoff, ~1960: Adaline/Madaline

• Introduced the idea of stacking layers of perceptrons

• no loss function

• no backprop
Fukushima 1980 – NeoCognitron

- First network architecture
- Took inspiration from Hubel and Wiesel's Neuroscience research and designed two types of neurons:
  - simple cells: contains weights that need to be learned
  - complex cells: pools activations from previous layers as a way of modeling the hierarchical nature of our visual system
Rumelhart et al., 1986: back-propagation

- Chain rule and update rule

\[
\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}
\]
LeCun '97 – LeNet architecture

• First real use case: Yann LeCun's digit recognizer
• Used by postoffice to recognize digits
• 70% test accuracy on MNIST (dataset of digits)
• They do average pooling instead of max pooling
First major breakthrough with deep learning

• Mohamed et al. *Acoustic Modeling using Deep Belief Networks*, 2010

• Dahl et al. *Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition*, 2012
First major deep learning breakthrough in Computer Vision

AlexNet - 2012

- Reduced error rate in the ImageNet Challenge in half
- very similar overall structure to LeNet architecture, only larger
- needed a GPU to train in a reasonable amount of time
- Incorporated Neurocognitron's pooling layers
ImageNet Challenge error rates over time

AlexNet

ZF-Net

2010 25.8
2011 28.2
2012 16.4
2013 11.7
2014 7.3
2014 6.7
2015 3.6
2016 3
2017 2.3
Human 5.1

shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers

Lin et al
Sanchez & Perronnin
Krizhevsky et al (AlexNet)
Zeiler & Fergus
Simonyan & Zisserman (VGG)
Szegedy et al (GoogLeNet)
He et al (ResNet)
Shao et al
Hu et al (SENet)
Russakovsky et al
Zeiler 2013 – ZF-Net

- Changes from convolution sizes of 11 to 7.
- Changes strides from 4 to 2
- Used more filters per layer
ImageNet Challenge error rates over time

- **2010**: 28.2
  - Lin et al
- **2011**: 25.8
  - Sanchez & Perronnin
- **2012**: 16.4
  - Krizhevsky et al (AlexNet)
- **2013**: 11.7
  - Zeiler & Fergus
- **2014**: 7.3
  - Simonyan & Zisserman (VGG)
- **2014**: 6.7
  - Szegedy et al (GoogLeNet)
- **2015**: 3.6
  - He et al (ResNet)
- **2016**: 3
  - Shao et al
- **2017**: 2.3
  - Hu et al (SENet)
- **Human**: 5.1

- **VGG-Net**
  - 19 layers
  - 22 layers

- **152 layers**

Stanford University
05-Dec-2019
Simmoyan 2014 – VGG-Net

• Deeper than previous layers
  – 16 layers instead of 8
• They used 3x3 conv filters instead of 7x7 used by AlexNet. Why?
ImageNet Challenge error rates over time

- **2010**: 28.2
- **2011**: 25.8
- **2012**: 16.4
- **2013**: 11.7
- **2014**: 7.3
- **2014**: 6.7
- **2015**: 3.6
- **2016**: 3
- **2017**: 2.3
- **Human**: 5.1

*Authors:* Lin et al, Sanchez & Perronnin, Krizhevsky et al (AlexNet), Zeiler & Fergus, Simonyan & Zisserman (VGG), Szegedy et al (GoogLeNet), He et al (ResNet), Shao et al, Hu et al (SENet), Russakovsky et al

*Note:* GoogLeNet is highlighted with 19 layers, 22 layers, and 152 layers.
Szegedy et al. 2014 – GoogLeNet

- 22 layers
- Efficient “Inception” module
- No fully connected layers
- Only 5 million parameters!
  - 12x less than AlexNet
• Multiple output branches to make sure gradients reached the earlier layers.
• As gradients are propagated backwards, they might explore or vanish as we are multiple partial gradients together. Having multiple branches fixes the vanishing gradient problem.
Inception module – hand designed local network topology that can be thought of as a network within a network and stacked on top of each other.
ImageNet Challenge error rates over time

- **2010**: Lin et al., shallow
- **2011**: Sanchez & Perronnin
- **2012**: Krizhevsky et al. (AlexNet), 8 layers
- **2013**: Zeiler & Fergus, 8 layers
- **2014**: Simonyan & Zisserman (VGG), 19 layers
- **2014**: Szegedy et al. (GoogLeNet), 22 layers
- **2015**: He et al. (ResNet), 152 layers
- **2016**: Shao et al.
- **2017**: Hu et al. (SENNet)
- **Human**: 5.1

**ResNet**

- 152 layers
- 152 layers
- 152 layers
ResNet depth in comparison to VGG
He 2016 - ResNet

- Learning residuals instead of feature transformations
- Also prevented the vanishing gradient problem
But there are open problems with all these architectures

• Let's consider on one specific problem.
• Remember from lecture 2:
  – we derived convolutions to mimic properties of our own visual system
  – One of those properties was shift invariance.

\[ f[n - n_0, m - m_0] \xrightarrow{S} g[n - n_0, m - m_0] \]

• But are these architectures shift invariant systems?
Recent paper found that all these architectures are not shift-invariant

- black line is AlexNet and VGG on ImageNet and CIFAR datasets.
- They proposed a small fix that improves all these models (in blue)

Zhang, Making Convolutions Shift-Invariant Again, ICML 2019
Can anyone guess which component is not shift invariant?

• All the models we spoke of are composed of three building blocks:
  – Convolution layers
  – Pooling Layers
  – ReLU activations
Convolutions were designed to be shift-invariant

\[ f[n - n_0, m - m_0] \xrightarrow{s} g[n - n_0, m - m_0] \]
Let's test ReLU with an example

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3</td>
<td>19</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>1</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>17</td>
<td>-5</td>
<td>4</td>
</tr>
<tr>
<td>-2</td>
<td>-5</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

ReLU

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3</td>
<td>19</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>17</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Let's test Pooling

Typically all models use filter size of 2 stride 2

MaxPool

L2 loss: $\sqrt{3}$
They proposed a new pooling filter: BlurPool

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Blur**

<table>
<thead>
<tr>
<th>0.67</th>
<th>0.67</th>
<th>0.17</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.67</td>
<td>0.67</td>
<td>0.17</td>
<td>0</td>
</tr>
</tbody>
</table>

**MaxPool**

| 0.67 | 0.17 |

Zhang, Making Convolutions Shift–Invariant Again, ICML 2019
Aside

• For an in-depth explanation on aliasing and how it's related to discrete signals (for ex, images), check out this link:
  • [http://www.rctn.org/bruno/npb261/aliasing.pdf](http://www.rctn.org/bruno/npb261/aliasing.pdf)

• This doesn't completely fix the problem as these models are still no shift-invariant. So, we still get unpredictable behavior. This is one of the many open problems left.
Today's agenda

• Backprop in neural networks
• Convolutional neural networks
• Architecture design
• Exam and class brief overview
• En fin
Exam details

• Monday, December 10
• 3:30 to 6:30pm
• Location: Hewlitt teaching center, room 200
• Practice exam available online on piazza.

• Make-up exam for those with conflicts
  – You should have received an email about this.
Class overview

• 80% assignments
• 20% final exam
• +10% extra credit

• Don’t be worried about the final.
• Optimal strategy:
  – Go over all the materials we covered in class.
  – Read through the entire final first and work on the problems you find easy first.
Exam overview

• 100 points in total
• 15 points for multiple choice
• 20 points for true false
• 25 points for filters, edges, corners, RANSAC, Hough transform
• 15 points for segmentation and seam carving
• 15 points for recognition and detection
• 10 points for video and deep learning
The goal of computer vision

- To bridge the gap between pixels and “meaning”

What we see

What a computer sees

Source: S. Narasimhan
Convolution and Cross-correlation

**Convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function

\[(f * h)[m,n] = \sum_{k,l} f[k,l] h[m-k,n-l]\]

**Cross-correlation** compares the similarity of two sets of data

\[(f \star g)[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] \cdot g[i-m, j-n]\]

Convolution with Impulse function

\[f[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \delta_2[n - k, m - l]\]
Characterizing edges

• An edge is a place of rapid change in the image intensity function
Finite differences: example

- Which one is the gradient in the x-direction? How about y-direction?
Designing an edge detector

• Criteria for an “optimal” edge detector:
  – **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  – **Good localization:** the edges detected must be as close as possible to the true edges
  – **Single response:** the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge
Canny edge detector

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  - Assures minimal response
- Use hysteresis and connectivity analysis to detect edges
Detecting lines using Hough transform
RANSAC: Pros and Cons

• **Pros:**
  – General method suited for a wide range of model fitting problems
  – Easy to implement and easy to calculate its failure rate

• **Cons:**
  – Only handles a moderate percentage of outliers without cost blowing up
  – Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)

• A voting strategy, The Hough transform, can handle high percentage of outliers
Requirements for keypoint localization

• Region extraction needs to be repeatable and accurate
  – Invariant to translation, rotation, scale changes
  – Robust or covariant to out-of-plane (≈affine) transformations
  – Robust to lighting variations, noise, blur, quantization

• Locality: Features are local, therefore robust to occlusion and clutter.

• Quantity: We need a sufficient number of regions to cover the object.

• Distinctiveness: The regions should contain “interesting” structure.

• Efficiency: Close to real-time performance.
Harris corner detector and second moment matrix

• First, let’s consider an axis-aligned corner:

\[
M = \begin{bmatrix}
\sum I_x^2 & \sum I_x I_y \\
\sum I_x I_y & \sum I_y^2
\end{bmatrix} = \begin{bmatrix}
\lambda_1 & 0 \\
0 & \lambda_2
\end{bmatrix}
\]

\[
\begin{align*}
\mathbf{u} & = \begin{bmatrix}
\mu_x \\
\mu_y
\end{bmatrix} \\
\mathbf{v} & = \begin{bmatrix}
\nu_x \\
\nu_y
\end{bmatrix}
\end{align*}
\]

\[
\mathbf{u}^T M \mathbf{v} = \lambda_1 \mathbf{u}^T \mathbf{v} + \lambda_2 \mathbf{v}^T \mathbf{u}
\]

\[
\begin{align*}
\lambda_1 & = (\mathbf{u}^T \mathbf{v})^2 \\
\lambda_2 & = \mathbf{u}^T M \mathbf{v}
\end{align*}
\]
Harris Detector: Properties

• Translation invariance
• Rotation invariance
• Scale invariance?

Corner

All points will be classified as **edges**!

**Not invariant to image scale!**
Scale Invariant Detectors

- **Harris–Laplacian**¹
  *Find local maximum of:*
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale

- **SIFT (Lowe)**²
  *Find local maximum of:*
  - Difference of Gaussians in space and scale

---

SIFT descriptor formation

• Using precise gradient locations is fragile. We’d like to allow some “slop” in the image, and still produce a very similar descriptor.
• Create array of orientation histograms (a 4x4 array is shown).
• Put the rotated gradients into their local orientation histograms:
  – A gradient’s contribution is divided among the nearby histograms based on distance. If it’s halfway between two histogram locations, it gives a half contribution to both.
  – Also, scale down gradient contributions for gradients far from the center.
• The SIFT authors found that best results were with 8 orientation bins per histogram.
Difference between HoG and SIFT

• HoG is usually used to describe entire images. SIFT is used for key point matching
• SIFT histograms are oriented towards the dominant gradient. HoG is not.
• HoG gradients are normalized using neighborhood bins.
• SIFT descriptors use varying scales to compute multiple descriptors.
Seam Carving

- Assume \( m \times n \rightarrow m \times n' \), \( n' < n \) (summarization)

- Basic Idea: remove unimportant pixels from the image
  - Unimportant = pixels with less “energy”

\[
E_1(I) = |\frac{\partial}{\partial x} I| + |\frac{\partial}{\partial y} I|.
\]

- Intuition for gradient-based energy:
  - Preserve strong contours
  - Human vision more sensitive to edges – so try remove content from smoother areas
  - Simple enough for producing some nice results
Gestalt Factors

- These factors make intuitive sense, but are very difficult to translate into algorithms.

Image source: Forsyth & Ponce
Conclusions: Agglomerative Clustering

Good
• Simple to implement, widespread application.
• Clusters have adaptive shapes.
• Provides a hierarchy of clusters.
• No need to specify number of clusters in advance.

Bad
• May have imbalanced clusters.
• Still have to choose number of clusters or threshold.
• Does not scale well. Runtime of $O(n^3)$.
• Can get stuck at a local optima.
K-means clustering

1. Initialize Cluster Centers
2. Assign Points to Clusters
3. Re-compute Means
Repeat (2) and (3)

• Java demo:
  http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Illustration Source: wikipedia
Feature Space

- Depending on what we choose as the feature space, we can group pixels in different ways.

- Grouping pixels based on color similarity

- Feature space: color value (3D)

Slide credit: Kristen Grauman
Feature Space

- Depending on what we choose as the feature space, we can group pixels in different ways.

- Grouping pixels based on texture similarity

- Feature space: filter bank responses (e.g., 24D)

Slide credit: Kristen Grauman
Segmentation as Clustering

• Depending on what we choose as the *feature space*, we can group pixels in different ways.

• Grouping pixels based on *intensity*+*position* similarity

⇒ Way to encode both *similarity* and *proximity*.

Slide credit: Kristen Grauman
Mean-Shift Clustering

• Cluster: all data points in the attraction basin of a mode

• Attraction basin: the region for which all trajectories lead to the same mode

Slide by Y. Ukrainitz & B. Sarel
Summary Mean-Shift

• **Pros**
  – General, application-independent tool
  – Model-free, does not assume any prior shape (spherical, elliptical, etc.) on data clusters
  – Just a single parameter (window size $h$)
    • $h$ has a physical meaning (unlike k-means)
  – Finds variable number of modes
  – Robust to outliers

• **Cons**
  – Output depends on window size
  – Window size (bandwidth) selection is not trivial
  – Computationally (relatively) expensive (~2s/image)
  – Does not scale well with dimension of feature space
The machine learning framework

\[ y = f(x) \]

• **Training:** given a *training set* of labeled examples \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), estimate the prediction function \( f \) by minimizing the prediction error on the training set

• **Testing:** apply \( f \) to a never before seen *test example* \( x \) and output the predicted value \( y = f(x) \)
Nearest Neighbor Classifier

• Assign label of nearest training data point to each test data point

Source: N. Goyal
Bias versus variance trade off

![Graph showing the relationship between error, model complexity, bias, and variance.](image-url)
Curse of dimensionality

• Assume 5000 points uniformly distributed in the unit hypercube and we want to apply 5-NN. Suppose our query point is at the origin.
  – In 1-dimension, we must go a distance of $5/5000=0.001$ on the average to capture 5 nearest neighbors.
  – In 2 dimensions, we must go $\sqrt{0.001}$ to get a square that contains 0.001 of the volume.
  – In d dimensions, we must go $(0.001)^{1/d}$.
Fischer's Linear Discriminant Analysis

Slide inspired by N. Vasconcelos
Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”
Bag of words + pyramids

Locally orderless representation at several levels of spatial resolution
Naïve Bayes

• Classify image using histograms of occurrences on visual words:

\[ x = \begin{array}{ccccccccccc}
\end{array} \]

\[ x_i \]

• if only present/absence of a word is taken into account:

• Naïve Bayes assumes that visual words are conditionally independent given object class

Csurka Bray, Dance & Fan, 2004
Detecting a person with their parts

- For example, a person can be modelled as having a head, left arm, right arm, etc.
- All parts can be modelled relative to the global person detector
Fine-Grained Recognition

Key: Find the right features.
Estimating optical flow

- Given two subsequent frames, estimate the apparent motion field $u(x,y), v(x,y)$ between them

- **Key assumptions**
  - **Brightness constancy:** projection of the same point looks the same in every frame
  - **Small motion:** points do not move very far
  - **Spatial coherence:** points move like their neighbors

Source: Silvio Savarese 12/5/19
Lucas Kande optical flow

- Optimal \((u, v)\) satisfies Lucas–Kanade equation

\[
\begin{bmatrix}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
= - \begin{bmatrix}
\sum I_x I_t \\
\sum I_y I_t
\end{bmatrix}
\]

\(A^T A\)

When is This Solvable?

- \(A^T A\) should be invertible
- \(A^T A\) should not be too small due to noise
  - eigenvalues \(\lambda_1\) and \(\lambda_2\) of \(A^T A\) should not be too small
- \(A^T A\) should be well-conditioned
  - \(\lambda_1/\lambda_2\) should not be too large (\(\lambda_1 = \text{larger eigenvalue}\))

Does this remind anything to you?

Source: Silvio Savarese
Tracking

Feature point tracking

Slide credit: Yonsei Univ.
Today's agenda

• Backprop in neural networks
• Convolutional neural networks
• Architecture design
• Exam and brief class overview
• En fin
What should you take away from this class?

• A broad understanding of computer vision as a field.

• Learning to use common software packages: github, jupyter, numpy, scipy.

• Converting ideas into mathematical equations.

• Converting mathematical equations into code.

• Learning to communicate ideas and algorithms in formal writing.
What should you take away from this class?

<table>
<thead>
<tr>
<th>Pixels</th>
<th>Segments</th>
<th>Images</th>
<th>Videos</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutions</td>
<td>Resizing</td>
<td>Recognition</td>
<td>Motion</td>
<td>Neural networks</td>
</tr>
<tr>
<td>Edges</td>
<td>Segmentation</td>
<td>Detection</td>
<td>Tracking</td>
<td>Convolutional</td>
</tr>
<tr>
<td>Descriptors</td>
<td>Clustering</td>
<td>Ontology</td>
<td></td>
<td>neural networks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Feedback – it matters a lot

• Feel free to email me with any personal feedback.
  – I am always trying to improve.

• Fill out anonymous class evaluations on axess.stanford.edu
Feedback – it matters a lot!

• Assignments
  – Which assignments were too easy? Too hard?
• Topics
  – Which topics did you enjoy the most and would recommend we keep and which ones did we not cover that you would like to include?
• Class notes and extra credit
  – Did you find them to be educational?
• Lectures
  – Were they engaging? How can we improve them?
How to stay involved in Computer Vision?

• Join us to do research!!
  – We expect 20 hours of research a week
  – Positions are open for credit starting winter quarter (if you are undergrad student)
  – Positions are open both for credit and assistantship starting winter quarter (if you are a graduate student)
  – Paid summer research internship opportunities are also available

• Juan Carlos focuses on visual recognition and understanding of human actions and activities, objects, scenes, and events.
  – Check out his one hour talk on his research here: http://tv.vera.com.uy/video/55276
How to stay involved in Computer Vision?

- **Ranjay** focuses on building Intelligence systems that grow their visual knowledge by interacting with and learning directly from people.
Where to go from here?

Still lots of open questions!

- Machine Learning (CS 229) (to learn the fundamentals of ML)
- 3D Computer Vision (CS 231A) (Silvio is on sabbatical this year)
- Convolutional Neural Networks (CS 231n) (I will be teaching it in the spring with Fei-Fei and Danfei)
- Mobile Computer Vision (CS 231M) (not offered every year)
- Representation Learning (CS 331B) (not offered every year)
- Advanced Computer Vision (CS 331A) (not offered every year)
CS 131 Computer Vision: Foundations and Applications

Juan Carlos Niebles and Ranjay Krishna
Sasha Harrison, Maxime Voisin, Brent Yi, Boxiao Pan
Stanford Vision and Learning Lab