Overall goals for the course

• Introduce fundamental concepts in computer vision

• Enable one or all of several such outcomes
  – Pursue higher studies in computer vision
  – Join industry to do cutting-edge work in computer vision
  – Gain appreciation of modern computer vision technologies

• Engage in discussions and interaction

• This is a great time to study computer vision!
Course details

• Class webpage:
  – http://cseweb.ucsd.edu/~mkchandraker/classes/CSE152A/Fall2022/

• TAs
  – Meng Song, Mallikarjun Swamy, Rishi Chandrasekaran, Vishal Vinod: {mes050, mswamy, r3chandr, vvinod}@ucsd.edu

• Tutors
  – Nick Chua, Navya Sharma, Ang Li: {nchua, n1sharma, a3li}@ucsd.edu

• Discussion section: M 3-3:50pm

• Office hours posted on course calendar

• Piazza: https://piazza.com/ucsd/fall2022/cse152a/
Recap
Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Represent images by histograms of “visual words”
4. Compare test image to each training image using normalized dot product between their histograms
K-Nearest Neighbors classification

• For a new point, find the k closest points from training data
• Labels of the k points “vote” to classify

If query lands here, the 5 NN consist of 3 negatives and 2 positives, so we classify it as negative.

Black = negative
Red = positive

Source: D. Lowe
Bayesian estimation

- Calculate $P(R | \text{skin})$ from labeled training images
- MAP estimation:
  - Prior $P(\text{skin})$ could be the proportion of skin pixels in training set
  - Use Bayes rule to get posterior
- ML estimation:
  - Uniform prior: $P(\text{skin}) = P(\sim \text{skin}) = 0.5$
  - Maximizing the posterior is equivalent to maximizing the likelihood

$$P(\text{skin}) = 0.75$$
$$P(\sim \text{skin}) = 0.25$$
Learning a Hierarchy of Feature Extractors

- Hierarchical and expressive feature representations
- Trained end-to-end, rather than hand-crafted for each task
- Remarkable in transferring knowledge across tasks
Neuron: Linear Perceptron

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

\[
\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)
\]

- If the activation is:
  - Positive, output +1
  - Negative, output -1
Two-layer perceptron network

\[
\begin{align*}
\begin{bmatrix}
    f_1' \\
    f_2' \\
    f_3'
\end{bmatrix} &= \begin{bmatrix}
    w_{11} & w_{21} & w_{31} \\
    w_{12} & w_{22} & w_{32} \\
    w_{13} & w_{23} & w_{33}
\end{bmatrix}
\begin{bmatrix}
    f_1 \\
    f_2 \\
    f_3
\end{bmatrix} > 0?
\end{align*}
\]

[w1, w2, w3]^T \text{ max}(Wf, 0)

CSE 152A, FA22: Manmohan Chandraker

Slide credit: Pieter Abeel and Dan Klein
Two-layer perceptron network

\[
\begin{bmatrix}
    f_1' \\
    f_2' \\
    f_3'
\end{bmatrix} = \begin{bmatrix}
    w_{11} & w_{21} & w_{31} \\
    w_{12} & w_{22} & w_{32} \\
    w_{13} & w_{23} & w_{33}
\end{bmatrix} \begin{bmatrix}
    f_1 \\
    f_2 \\
    f_3
\end{bmatrix} > 0?
\]

\[ [w_1, w_2, w_3]^T \max(Wf, 0) \]

CSE 152A, FA22: Manmohan Chandraker

Slide credit: Pieter Abeel and Dan Klein
Neural networks

Linear score function: $f = Wx$

2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$

3-layer Neural Network

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

Non-linearity
Activation functions

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

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

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

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

**ELU**
\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Learning w

- Training examples: pairs of (data, label)
  $$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

- Objective: minimize a misclassification loss
  $$\sum_{i=1}^{m} (y_i - f(x_i))^2$$

- Procedure:
  - Gradient descent or hill climbing
What is a Convolution?

- Weighted moving sum
- Convolving an image with a filter produces a feature map
k-D spatial filters

- A k-dimensional input can be convolved with a k-dimensional filter
- Compute dot product at each location for overlapping “volume”
- Output will be a 2-dimensional feature map
From fully connected to convolutional networks
From fully connected to convolutional networks

learned weights

image

Convolutional layer

feature map

CSE 152A, FA22: Manmohan Chandraker

Slide: Lazebnik
From fully connected to convolutional networks

CSE 152A, FA22: Manmohan Chandraker
From fully connected to convolutional networks
Learnable filters

- Several, handcrafted filters in computer vision
  - Canny, Sobel, Gaussian blur, smoothing, low-level segmentation, morphological filters, Gabor filters
- Are they optimal for recognition?
- Can we learn them from our data?
- Are they going resemble the handcrafted filters?
Filters over the whole image

Assume the image is 30x30x3.

1 filter every pixel (stride = 1)

How many parameters in total?

24 filters along the x axis
24 filters along the y axis
Depth of 5
$7 \times 7 \times 3$ parameters per filter

$423K$ parameters in total
Weight sharing

Insight: Images have similar features at various spatial locations!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring

Assume the image is $30 \times 30 \times 3$.
1 column of filters common across the image.
How many parameters in total?

Depth of 5
$\times 7 \times 7 \times 3$ parameters per filter
735 parameters in total
When weight sharing is not good

- When images are registered and each pixel has a particular significance
  - E.g. after face alignment specific pixels hold specific types of inputs, like eyes, nose, etc.
- In these cases maybe better every spatial filter to have different parameters
  - Network learns particular weights for particular image locations [Taigman2014]
Convolutional layer is differentiable

- **Activation**

  \[ a_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot \theta_{ij} \]

- Essentially a dot product, similar to linear layer

  \[ a_{rc} \sim x_{\text{region}}^T \cdot \theta \]

- **Gradient w.r.t. the parameters**

  \[ \frac{\partial a_{rc}}{\partial \theta_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r-i,c-j} \]
Pooling operations

• Aggregate multiple values into a single value

Single depth slice

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Pooling operations

• Aggregate multiple values into a single value

• Invariance to small transformations
  • Keep only most important information for next layer

• Reduces the size of the next layer
  • Fewer parameters, faster computations

• Observe larger receptive field in next layer
  • Hierarchically extract more abstract features
Receptive fields in CNNs

• The area in the input image “seen” by a unit in a CNN
• Units in deeper layers will have wider receptive fields
Max Pooling

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $i_{\text{max}}, j_{\text{max}} = \arg \max_{i,j \in \Omega(r,c)} x_{ij} \rightarrow a_{rc} = x[i_{\text{max}}, j_{\text{max}}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & \text{if } i = i_{\text{max}}, j = j_{\text{max}} \\ 0, & \text{otherwise} \end{cases}$
- The preferred choice of pooling
1 x 1 convolutions

1 x 1 convolution layers also possible, equivalent to a dot product.

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Key operations in a CNN

- Feature maps
- Spatial pooling
- Non-linearity
- Convolution (Learned)
- Input Image

Source: R. Fergus, Y. LeCun
Key operations in a CNN

- Input Image
- Convolution (Learned)
- Non-linearity
- Spatial pooling
- Feature maps

Rectified Linear Unit (ReLU)

Source: R. Fergus, Y. LeCun
Key operations in a CNN

1. Input Image
2. Convolution (Learned)
3. Non-linearity
4. Spatial pooling
5. Feature maps

Source: R. Fergus, Y. LeCun
Convolutional Neural Networks
Optimization in Neural Networks
A 3-layer network for digit recognition

- Each grayscale image is of size 28x28.
- 60,000 training images and 10,000 test images
- 10 possible labels (0,1,2,3,4,5,6,7,8,9)

Input normalized to a value between 0 and 1.

Example outputs:
6 -> [0000001000]'}
The network tries to approximate the function $y(x)$ and its output is $a$.

We use a quadratic cost function, or MSE, or “L2-loss”.

CSE 152A, FA22: Manmohan Chandraker
Gradient descent

\[ C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \]

parameters to compute

# of input samples

\[ \Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \]

Small changes in parameters to leads to small changes in output

\[ \nabla C \equiv \left( \frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \]

Gradient vector!

\[ \Delta v = -\eta \nabla C \]

Change the parameter using learning rate (positive) and gradient vector!

\[ v \rightarrow v' = v - \eta \nabla C \]

Update rule!
**Stochastic gradient descent**

Cost function is a sum over all the training samples:

\[
C(w, b) = \sum_x C_x(w, b), \text{ where } C_x(w, b) = \frac{1}{2}\|y(x) - a\|^2
\]

Gradient from entire training set:

\[
\nabla C = \frac{1}{n} \sum_x \nabla C_x
\]

Update rules for each parameter:

\[
w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}
\]

\[
b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}
\]

Parameters to compute vs. # of input samples:

Usually, \( n \) is very large.
Stochastic gradient descent

Gradient from entire training set:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

- For large training data, gradient computation takes a long time
  - Leads to “slow learning”

- Instead, consider a mini-batch with $m$ samples
- If sample size is large enough, properties approximate the dataset

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$
Stochastic gradient descent

What if the loss function has a local minima or saddle point?

Zero gradient, gradient descent gets stuck
Stochastic gradient descent

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction

Loss function has high condition number: ratio of largest to smallest singular value of the Hessian matrix is large
Stochastic gradient descent

Our gradients come from minibatches so they can be noisy!

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) \]

\[ \nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) \]
Stochastic gradient descent

Momentum update:

Velocity

actual step

Gradient

SGD

\[ x_{t+1} = x_t - \alpha \nabla f(x_t) \]

SGD+Momentum

\[ v_{t+1} = \rho v_t + \nabla f(x_t) \]
\[ x_{t+1} = x_t - \alpha v_{t+1} \]

Build up velocity as a running mean of gradients.
Layer to layer relationship

\[ a_j^l = \sigma(z_j^l) \]

\[ z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \]

\[ a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \]

- \( b_j^l \) is the bias term in the jth neuron in the lth layer.
- \( a_j^l \) is the activation in the jth neuron in the lth layer.
- \( z_j^l \) is the weighted input to the jth neuron in the lth layer.
Cost and gradient computation

\[ C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 \]

- The goal of the backpropagation algorithm is to compute the gradients \( \frac{\partial C}{\partial w} \) and \( \frac{\partial C}{\partial b} \) of the cost function \( C \) with respect to each and every weight and bias parameters. Note that backpropagation is only used to compute the gradients.

\[ C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 \]

- Stochastic gradient descent is the training algorithm.

CSE 152A, FA22: Manmohan Chandraker
Chain rule of differentiation

- In order to differentiate a function $z = f(g(x))$ w.r.t $x$, we can do the following:

Let $y = g(x)$, $z = f(y)$, $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and $f$ maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

This is all you need to know to get the gradients in a neural network!
Backpropagation

• In order to differentiate a function $z = f(g(x))$ w.r.t $x$, we can do the following:

Let $y = g(x)$, $z = f(y)$, $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and $f$ maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

This is all you need to know to get the gradients in a neural network!

Backpropagation: application of chain rule in certain order, taking advantage of forward propagation to efficiently compute gradients.
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2 \), \( y = 5 \), \( z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\( f(x, y, z) = (x + y)z \)

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
\begin{align*}
q &= x + y \\
\frac{\partial q}{\partial x} &= 1, \quad \frac{\partial q}{\partial y} = 1
\end{align*}
\]

\[
\begin{align*}
f &= qz \\
\frac{\partial f}{\partial q} &= z, \quad \frac{\partial f}{\partial z} = q
\end{align*}
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation example

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \; y = 5, \; z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \; \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \; \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \; \frac{\partial f}{\partial y}, \; \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} \]
Patterns in backpropagation

**add** gate: gradient distributor
**max** gate: gradient router
**mul** gate: gradient switcher

![Diagram of a neural network](image-url)
Convolutional layer is differentiable

- Activation function
  \[ a_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot \theta_{ij} \]

- Essentially a dot product, similar to linear layer
  \[ a_{rc} \sim x_{region}^T \cdot \theta \]

- Gradient w.r.t. the parameters
  \[ \frac{\partial a_{rc}}{\partial \theta_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r-i,c-j} \]