Outline

Basics of computer images

Image processing

Image features

Object recognition
Outline

Basics of computer images

Image processing

Image features

Object recognition
### Computer images

<table>
<thead>
<tr>
<th>What you see</th>
<th>What computer sees</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image_url" alt="Image" /></td>
<td><img src="matrix_url" alt="Matrix" /></td>
</tr>
</tbody>
</table>

- Images represented as matrices of intensity values (already seen this in the digit classification problem)
• For color images, image just contains three intensity matrices, one for red, green, and blue channels

\[ \text{image} = \text{red channel} + \text{green channel} + \text{blue channel} \]

• Almost all the techniques presented here could be applied to color images by applying them to their individual channels, but it’s more common to just use grayscale images

• When we write mathematical forms, we’ll think of image pixels being real-valued entries, even though they are typically e.g. 8 bit quantities in the images themselves
This lecture

• This lecture will be about some basic methods that practitioners use to “understand” images on a computer

• It will also be a (very brief) introduction to a tool used for computer vision applications: OpenCV

OpenCV

http://opencv.org
OpenCV basics

- Open/save/display an image

```python
import numpy
import cv2

img = cv2.imread('camera.png', cv2.IMREAD_GRAYSCALE)
img = cv2.imwrite('camera2.png', img)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Images are stored as normal numpy arrays

```python
img[10:15, 10:15] # returns 5x5 numpy array of uint8
```
• Display video from camera

```python
import numpy
import cv2

cap = cv2.VideoCapture(0)
while(True):
    ret, frame = cap.read()
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) > 0:
        break

cap.release()
cv2.destroyAllWindows()
```
Outline

Basics of computer images

Image processing

Image features

Object recognition
Color to grayscale

- Because we typically work in grayscale, one of the more basic image processing options to convert images from color to grayscale

- A simple approach, averaging

  \[ Y = \frac{R + G + B}{3} \]

- However, there are alternatives

  Lightness: \( Y = \frac{\max\{R, G, B\} + \min\{R, G, B\}}{2} \)

  Luminosity: \( Y = 0.21R + 0.72G + 0.07B \)

  (where exact constants are subject to change, these ones and following images are from GIMP documentation)
• Average, lightness, luminosity for sunflower

Original  Average  Lightness  Luminosity

• OpenCV command

```python
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

(uses luminosity transformation but with slightly different weights)
Convolutions

• Convolutions are one of the most basic image operations, and form the foundation for many more involved approaches

• Written e.g. as the following

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}
\ast Y
\]

• The operation “slides” the left matrix over the entire image \( Y \in \mathbb{R}^{m \times n} \), sums the product of corresponding entries for each position
\[
\begin{array}{ccc}
H_{11} & H_{12} & H_{13} \\
H_{21} & H_{22} & H_{23} \\
H_{31} & H_{32} & H_{33} \\
\end{array}
\times
\begin{array}{ccccc}
Y_{11} & Y_{12} & Y_{13} & Y_{14} & Y_{15} \\
Y_{21} & Y_{22} & Y_{23} & Y_{24} & Y_{25} \\
Y_{31} & Y_{32} & Y_{33} & Y_{34} & Y_{35} \\
Y_{41} & Y_{42} & Y_{43} & Y_{44} & Y_{45} \\
Y_{51} & Y_{52} & Y_{53} & Y_{54} & Y_{55} \\
\end{array}
=
\begin{array}{ccc}
Z_{11} & Z_{12} & Z_{13} \\
Z_{21} & Z_{22} & Z_{23} \\
Z_{31} & Z_{32} & Z_{33} \\
\end{array}
\]
\[
\begin{array}{ccc}
H_{11} & H_{12} & H_{13} \\
H_{21} & H_{22} & H_{23} \\
H_{31} & H_{32} & H_{33} \\
\end{array} \quad \times \quad \begin{array}{ccccc}
Y_{11} & Y_{12} & Y_{13} & Y_{14} & Y_{15} \\
Y_{21} & Y_{22} & Y_{23} & Y_{24} & Y_{25} \\
Y_{31} & Y_{32} & Y_{33} & Y_{34} & Y_{35} \\
Y_{41} & Y_{42} & Y_{43} & Y_{44} & Y_{45} \\
Y_{51} & Y_{52} & Y_{53} & Y_{54} & Y_{55} \\
\end{array} = \begin{array}{ccc}
Z_{11} & Z_{12} & Z_{13} \\
Z_{21} & Z_{22} & Z_{23} \\
Z_{31} & Z_{32} & Z_{33} \\
\end{array}
\]

\[
Z_{11} = H_{11}Y_{11} + H_{12}Y_{12} + \ldots
\]
\[
\begin{array}{ccc}
H_{11} & H_{12} & H_{13} \\
H_{21} & H_{22} & H_{23} \\
H_{31} & H_{32} & H_{33} \\
\end{array}
\quad \quad
\begin{array}{cccc}
Y_{11} & Y_{12} & Y_{13} & Y_{14} & Y_{15} \\
Y_{21} & Y_{22} & Y_{23} & Y_{24} & Y_{25} \\
Y_{31} & Y_{32} & Y_{33} & Y_{34} & Y_{35} \\
Y_{41} & Y_{42} & Y_{43} & Y_{44} & Y_{45} \\
Y_{51} & Y_{52} & Y_{53} & Y_{54} & Y_{55} \\
\end{array}
\quad =
\begin{array}{ccc}
Z_{11} & Z_{12} & Z_{13} \\
Z_{21} & Z_{22} & Z_{23} \\
Z_{31} & Z_{32} & Z_{33} \\
\end{array}
\]

\[
Z_{12} = H_{11}Y_{12} + H_{12}Y_{13} + \ldots
\]
Typically, separate operations for image “borders” so that convolved image is the same size as original

Usually use odd-sized convolutions, so that “center” pixels correspond
• Example: average pixels (blur)

\[ H = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \]

• Example: Gaussian blur

\[ H = \begin{bmatrix} G_\sigma(-1, -1) & G_\sigma(-1, 0) & G_\sigma(-1, 1) \\ G_\sigma(0, -1) & G_\sigma(0, 0) & G_\sigma(0, 1) \\ G_\sigma(1, -1) & G_\sigma(1, 0) & G_\sigma(1, 1) \end{bmatrix} \]

where

\[ G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{x^2 + y^2}{2\sigma^2} \right\} \]
Example: blur

Original image

5x5 averaging
Example: Gaussian blur

Original image

7x7 Gaussian blur
As you would expect, OpenCV has built-in methods for general convolution, blur and Gaussian blur.

```python
H = np.ones((5,5))/25.0
blurred1 = cv2.filter2D(gray, cv2.CV_8U, H);
blurred2 = cv2.blur(img,(5,5))
gaussian_blurred = cv2.GaussianBlur(img,(7,7),0))
```
Aside: Fourier transform

- Fourier transform expresses image in terms of “frequencies”

Original image

Log frequency magnitudes
• A very nice property: convolution in image space is equivalent to (elementwise) multiplication in frequency space

• Leads to procedure: apply FFT to convert image and filter to frequency space, multiply, then apply inverse FFT to convert to image space

• When filter is large, can be a big benefit, but for small filters, better to just do convolution manually
Image gradients and edge detection

• “Edges” in images represent one of the more interesting properties we could extract

• Informally, we expect an edge to occur when there is a sudden change in image intensity
• If we think of images as two-dimensional functions $f(x, y)$, then we can think about the gradients of this function

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$$

• In practice, since images are discrete, we could approximate these just using differences, i.e. to compute all $x$ and $y$ gradients respectively, we could use the convolutions

$$H_x = \begin{bmatrix} -1 & 1 \end{bmatrix}, \quad H_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$
• In practice, this simple difference is too noisy, a better alternative is to first (Gaussian) blur the image, then take differences

• We can do both of these using a single convolution: Sobel filters

3x3: \( H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \)

\( H_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -2 & -3 & 0 & 3 & 2 \\ -3 & -5 & 0 & 5 & 3 \\ -2 & -3 & 0 & 3 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} \), \( H_y = \begin{bmatrix} -1 & -2 & -3 & -2 & -1 \\ -2 & -3 & -5 & -3 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 3 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix} \)
Magnitude of gradient

$$\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$
- OpenCV calls:

```python
dfdx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
dfdy = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
```
Canny edge detector

- Extremely popular edge detection algorithm, compute image gradients and then
  
  1. Non-maximal suppression: only keep “ridges” of the gradient magnitude
  
  2. Ignore edges that are too small
  
  3. Ignore edges that never reach high enough magnitude

```python
dges = cv2.Canny(img, 100, 200)
```
Outline

Basics of computer images

Image processing

Image features

Object recognition
Image features

• In computer vision “features” correspond to patches of the image that are “easy to identify”

Can you find these patches in the image?
• What makes a “good” feature?

• One possible answer: if we shift the window for an image patch in any direction, it should look different from the original patch.
• Mathematically, let’s consider our image as a function again, and look at all pixels inside some window $W$

• Define the function

$$E_W(\Delta x, \Delta y) = \sum_{x,y \in W} (f(x, y) - f(x + \Delta x, y + \Delta y))^2 \quad (1)$$

• If $E_W(\Delta x, \Delta y)$ is big for small displacements $\Delta x, \Delta y$ in any direction, the window is a good feature
For small displacements

\[ f(x + \Delta x, y + \Delta y) \approx f(x, y) + \frac{\partial f(x, y)}{\partial x} \Delta x + \frac{\partial f(x, y)}{\partial y} \Delta y \]

so

\[ E_W(\Delta x, \Delta y) = \sum_{x,y \in W} (f(x, y) - f(x + \Delta x, y + \Delta y))^2 \]

\[ \approx \sum_{x,y \in W} \left( \frac{\partial f(x, y)}{\partial x} \Delta x + \frac{\partial f(x, y)}{\partial y} \Delta y \right)^2 \]

\[ = [\Delta x \quad \Delta y] M_W \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \]

where

\[ M_W = \sum_{x,y \in W} \begin{bmatrix} \left( \frac{\partial f(x, y)}{\partial x} \right)^2 & \frac{\partial f(x, y)}{\partial x} \cdot \frac{\partial f(x, y)}{\partial y} \\
\frac{\partial f(x, y)}{\partial x} \cdot \frac{\partial f(x, y)}{\partial y} & \left( \frac{\partial f(x, y)}{\partial y} \right)^2 \end{bmatrix} \]
• Again, want to find patches \( W \) where

\[
E_W(\Delta x, \Delta y) \approx \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M_W \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\]

is big for inputs in any direction \( \Delta x, \Delta y \)

• Look at eigenvalues \( \lambda_1, \lambda_2 \) of \( M_W \); if they are both large, displacement in any direction will cause increase in \( E \)

  – Harris corner detector: good feature if 
    \[
    \det(M_W) - k(\text{trace}(M_W))^2
    \]
    is above some threshold

  – Shi-Tomasi corner detector: good feature if \( \min(\lambda_1, \lambda_2) \) is above some threshold
• In OpenCV:

```
corners = cv2.goodFeaturesToTrack(gray, 50, 0.01, 10)
```
SIFT features (very briefly)

- Downside of Harris/Shi-Tomasi features is that they are not invariant to rescaling.

- Scale Invariant Feature Transform, (Lowe 2004): compute (something similar to) these features, but at multiple image scales.

- Also computes feature descriptors, a 128-dimensional vector describing histogram of image gradients at feature points/scales.
Tracking features over time

- One of the most common tasks in images/video is to track how features move across multiple images.

- Consider image (video) now also as function of time \( f(x, y, t) \).

- For some point \( x, y \) (e.g., an image features) find displacement \( \Delta x, \Delta y \) such that

\[
f(x, y, t) = f(x + \Delta x, y + \Delta y, t + \Delta t)
\]
• By first order expansion,

\[
f(x + \Delta x, y + \Delta y, t + \Delta t) \approx f(x, y, z) + \frac{\partial f(x, y, t)}{\partial x} \Delta x + \frac{\partial f(x, y, t)}{\partial y} \Delta y + \frac{\partial f(x, y, t)}{\partial t} \Delta t
\]

so we want to find \(\Delta x, \Delta y\) such that

\[
\frac{\partial f(x, y, t)}{\partial x} \Delta x + \frac{\partial f(x, y, t)}{\partial y} \Delta y + \frac{\partial f(x, y, t)}{\partial t} \Delta t = 0
\]

• Written a bit more compactly

\[
\frac{\partial f(x, y, t)}{\partial x} u + \frac{\partial f(x, y, t)}{\partial y} v = -\frac{\partial f(x, y, t)}{\partial t}
\]

where \(u = \Delta x/\Delta t, \ v = \Delta y/\Delta t\)

• One equation and two unknowns
Lucas-Kanade optical flow

- Assume that flow field is the same over small patch $W$ (say, centered around the feature) in an image

- Find $u, v$ that solve optimization problem

$$\min_{u,v} \sum_{x,y \in W} \left( \frac{\partial f(x,y,t)}{\partial x} u + \frac{\partial f(x,y,t)}{\partial y} v + \frac{\partial f(x,y,t)}{\partial t} \right)^2$$

- This is a least-squares problem, can be solved using a matrix inverse

- In fact, quadratic term in $u, v$ is the exact same $M_W$ matrix as in feature detection: good features to track also give “good” least-squares problems
• Some tricks are needed to make this robust:
  
  – Compute multiple optimal flow fields at different scalings of the image (track larger differences)
  
  – Track features/points across multiple frames
  
  – Add/remove new features as necessary

• As usual, OpenCV has a fast method for doing this

  corners = cv2.calcOpticalFlowPyrLK(img1, img2, points)
Outline

Basics of computer images

Image processing

Image features

Object recognition
Example: face detection

- How can we detect all the faces in an image?

- Seems less impressive than 10 years ago (our phones all do this now), but here we'll see how it’s done
• Conceptually, this is a simple machine learning task
  
  – Collect many images of faces and non-faces, scaled to some small but reasonable size (24x24 is common)

  – Use machine learning to learn a classifier for face/not-face

  – Given new image, create new images at many different scales, look at all 24x24 windows and classify each as face/not-face, add non-maximal suppression

• What could possibly go wrong?
• We need extremely high accuracy (very low false positive rate), computed very quickly

• Linear classifier? (like you used for digit classification)
  – Fast, but 90% accuracy isn't going to cut it

• SIFT features and descriptors
  – Very accurate, but much too slow
Viola-Jones object detection

• (Viola and Jones, CVPR 2001)

Rapid Object Detection using a Boosted Cascade of Simple Features

Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

• “Just” a set of good features and ways to speed up ML algorithms for face detection (9,000+ citations)

• ... like SIFT was “just” a set of good images features and descriptions (27,000+ citations)
Haar features

- A type of feature that is good at capturing relevant characteristics and easy to compute

Edge Features: 

Line Features: 

Rectangle Features: 

- These features take the sum of all pixels in white area, minus sum of all pixels in black area
• Quickly compute Haar features (at any image scale) through integral image

\[
\bar{f}(x, y) = \sum_{x' \leq x} \sum_{y' \leq y} f(x', y')
\]

• Then

\[
\sum_{x_1 \leq x' \leq x_2} \sum_{y_1 \leq y' \leq y_2} f(x', y') = \bar{f}(x_2, y_2) + \bar{f}(x_1, y_1) - \bar{f}(x_1, y_2) - \bar{f}(x_2, y_1)
\]

• Upshot: any Haar feature at any scale can be computed in a constant number of operations
Fast classification

- Still have 180,000+ Haar features for each 24x24 region (most of these are uninteresting)

- Use boosted decision stumps to extract 5,000 most relevant features

- Use “cascade” of classifiers that can quickly reject many regions using far fewer features

Figures: (Viola and Jones, 2001)
Take home points

• Computer vision studies how computers can “understand” the content of images

• The “general” vision problem is extremely challenging, but many “simple” operations can reveal a surprisingly large amount of information about images

• Lots of these items have been implemented already, a good idea to use existing software package