UMSL
Mathematics & Computer Science

24. Perception

24.2 “Early” Image-Processing Operations

- Three useful “low-level” or “early” image processing operations:
  - Edge detection
  - Texture analysis
  - Computation of optical flow

- These operations are “low-level” because
  - they are local in nature, i.e. they can be carried out in one part of the image without regard for anything more than a few pixels away
  - they lack knowledge about the object present in the scene
- Low level operations are good candidates for implementation in parallel hardware
  - either an “eye” or a graphics processing unit (GPU)
- “Mid-level” and “High-level” operations
  - Segmenting an image into regions can be considered a “mid-level” operation
  - Object/face detection/recognition can be considered a “high-level” operations

24.2.1 Edge Detection

- Edges are straight lines are curves in the image across which
  - there is “significant” change in image brightness
- Why detect edges?
  - To produce a more abstract representation of an input image (which is often messy/noisy)
  - Example applications: self-driving cars, medical image segmentation
  - Direct application of edge detection is fading out (because of deep learning)
    - but DL uses edge detection in principle (through various filters in CNN)

24.2.1 The Principle of Edge Detection

- Consider the profile of image brightness along a small one-dimensional line
- Edges correspond to the locations in images where the brightness undergoes a sharp change

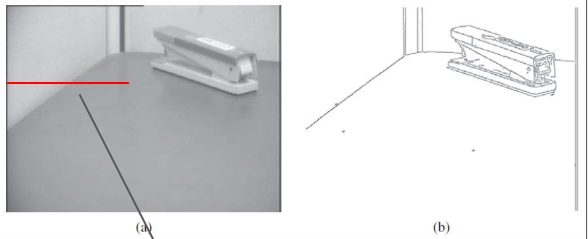
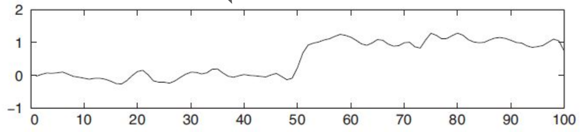
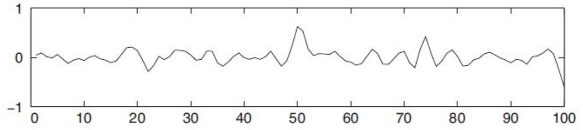


Figure 24.7 (a) Photograph of a stapler. (b) Edges computed from (a).

Principle: Build a profile of image brightness $I(x)$ along a dimension, and differentiate the image to look for places where the magnitude of the derivative $I'(x)$ is large.

Intensity profile $I(x)$

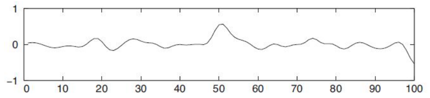




$I'(x)$

- There is a peak at around x = 50
- There is also another peak at x = 75 (noise)

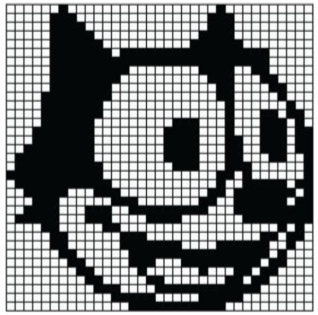
How can we remove/minimize/smooth such noise?

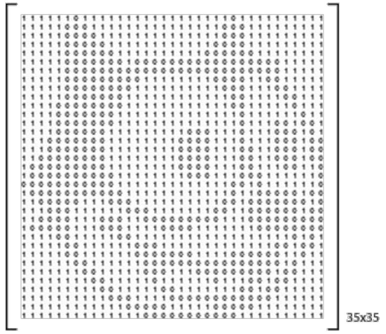


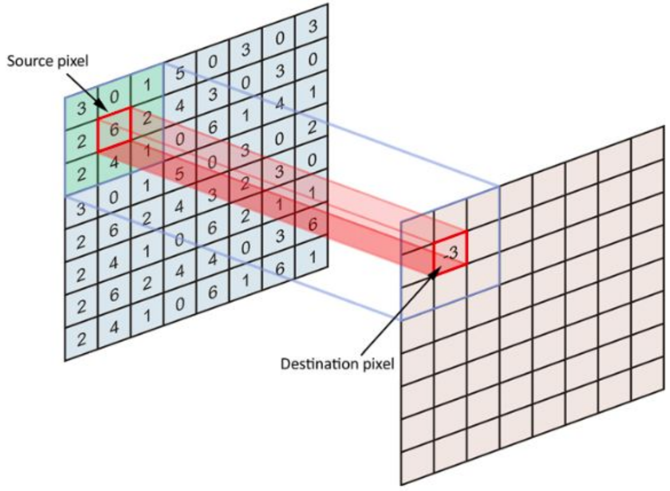
24.2.1 How to Smooth an Image?

Idea: Assign each pixel the average of its neighbors

1. How to calculate average for corner values?
2. Will the size of the output image decrease?
3. How many neighbors should we consider?



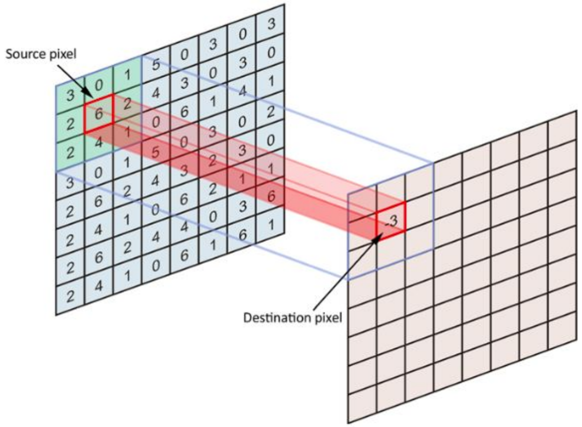


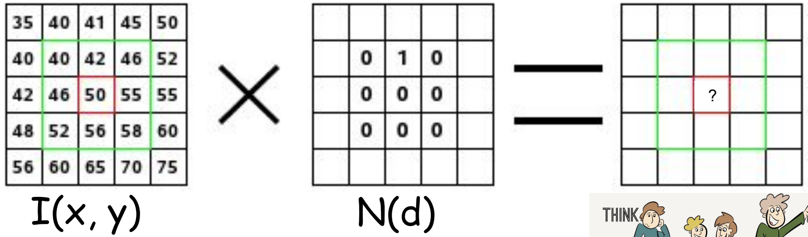


24.2.1 How Many Neighbors Should We Consider?

Answer: Weighted average - such that we weight the nearest pixels most, then gradually decrease the weight for more distant pixels.

This weight matrix is known as **Gaussian filter**.





35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

I(x, y)

×

	0	1	0	
	0	0	0	
	0	0	0	

N(d)

=

		?		

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Example weight matrix

The application of Gaussian filter replaces the intensity $I(x_0,y_0)$ with the sum, over all (x,y) pixels, of $I(x,y)\,N(d)$ where d is the distance from (x_0,y_0) to (x,y). Such a weighted sum is also known an **Convolution**.

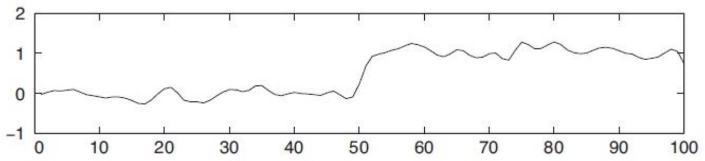
24.2.1 Smoothing using Convolution - 1D & 2D

- A function h is the convolution of two functions f and g (denoted by f * g) if we have

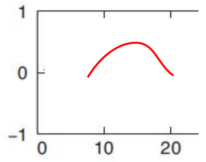
$$h(x) = (f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u)$$

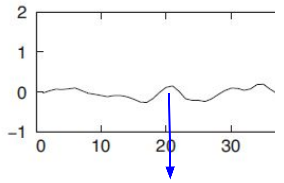
- Usually f(x) is the image matrix and g(x) is the Gaussian filter. Instead of running u from -inf to +inf we usually run it from -3 to +3.

f(x)



g(x)

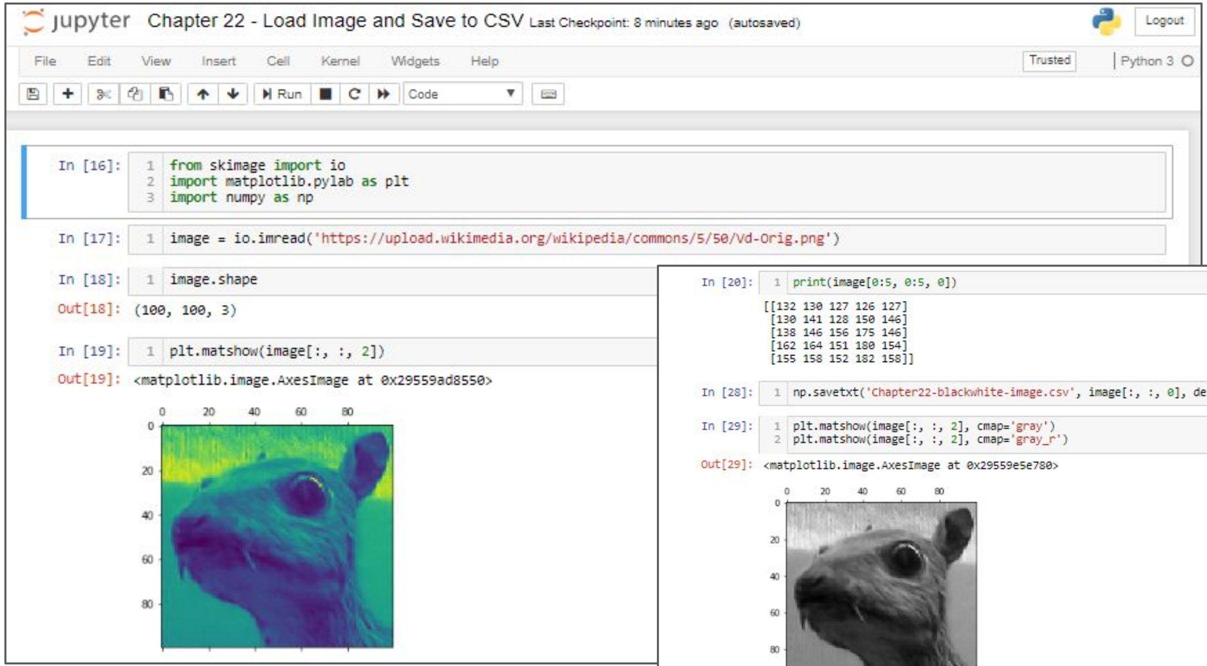




decreased

$$h(x,y) = (f * g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)\, g(x-u, y-v)$$

24.2.1 Smoothing using Convolution - 2D



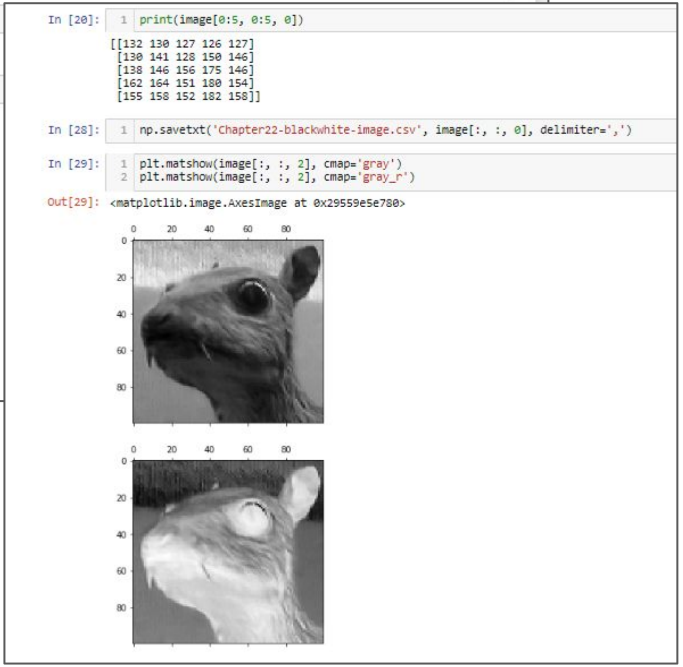
Chapter 22 - Load Image and Save to CSV Last Checkpoint: 8 minutes ago (autosaved)

```
In [16]: from skimage import io
         import matplotlib.pylab as plt
         import numpy as np

In [17]: image = io.imread('https://upload.wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png')

In [18]: image.shape
Out[18]: (100, 100, 3)

In [19]: plt.matshow(image[:, :, 2])
Out[19]: <matplotlib.image.AxesImage at 0x29559ad8550>
```

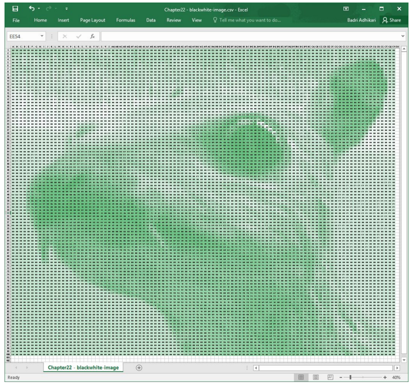


```
In [20]: print(image[0:5, 0:5, 0])
         [[132 130 127 126 127]
          [130 141 128 150 146]
          [138 146 156 175 146]
          [162 164 151 180 154]
          [155 158 152 182 158]]

In [28]: np.savetxt('Chapter22-blackwhite-image.csv', image[:, :, 0], delimiter=',')

In [29]: plt.matshow(image[:, :, 2], cmap='gray')
         plt.matshow(image[:, :, 2], cmap='gray_r')
Out[29]: <matplotlib.image.AxesImage at 0x29559e5e780>
```

Using Excel to apply Gaussian Filter on a Image



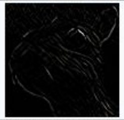
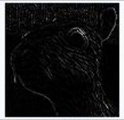
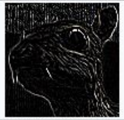
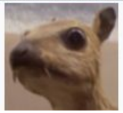
1. Load an image
2. Select only one channel
3. Print the channel to a CSV file

Github location
https://github.com/badriadhikari/UMSL-CS-4300-5300

24.2.1 How Detect Edges?

Theorem: For any functions f and g, the derivative of the convolution, (f*g)′ is equal to the convolution with the derivative f * (g′).

- Instead of smoothing the image and then differentiating
  - Convolve the image with the derivative of the smoothing function

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Homework

1. Code a Python subroutine for convolution (say, `convolve`). The subroutine will accept two 2D arrays as input and return the convolution of the two arrays.
2. Load the input image (`channel2.csv` file) into a Python 2D array (say, `image`).

   For each of the 9 kernels (see image/table on the side)

   a. create another 2D array (say, `identity`)
   b. apply the convolution operation two times, and
   c. visualize the output image.
3. Submit (a) your code (b) all output images

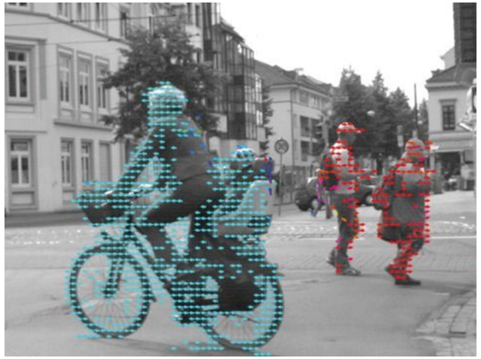
   The names of images should be the name of the operation - for example, the output image after applying 'sharpen' kernel should be '`sharpen.png`'.

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Github location: https://github.com/badriadhikari/UMSL-CS-4300-5300

24.2.3 Optical Flow



- Optical flow is the apparent motion in an image
  - When object in a video is moving or when camera is moving relative to an object
- It describes the direction and speed of motion of features in the image
  - Example, if a car is moving in a video, its optical flow can be measured in pixels per second
- Optical flow encodes useful information about the scene structure
  - For example, in a video of scenery taken from a moving bus, distant objects have slower apparent motion than close objects; the rate of apparent motion can tell us about distance
- https://www.youtube.com/watch?v=qbVx4dzWQHI

24.2.3 Optical Flow Enables us to Recognize Actions

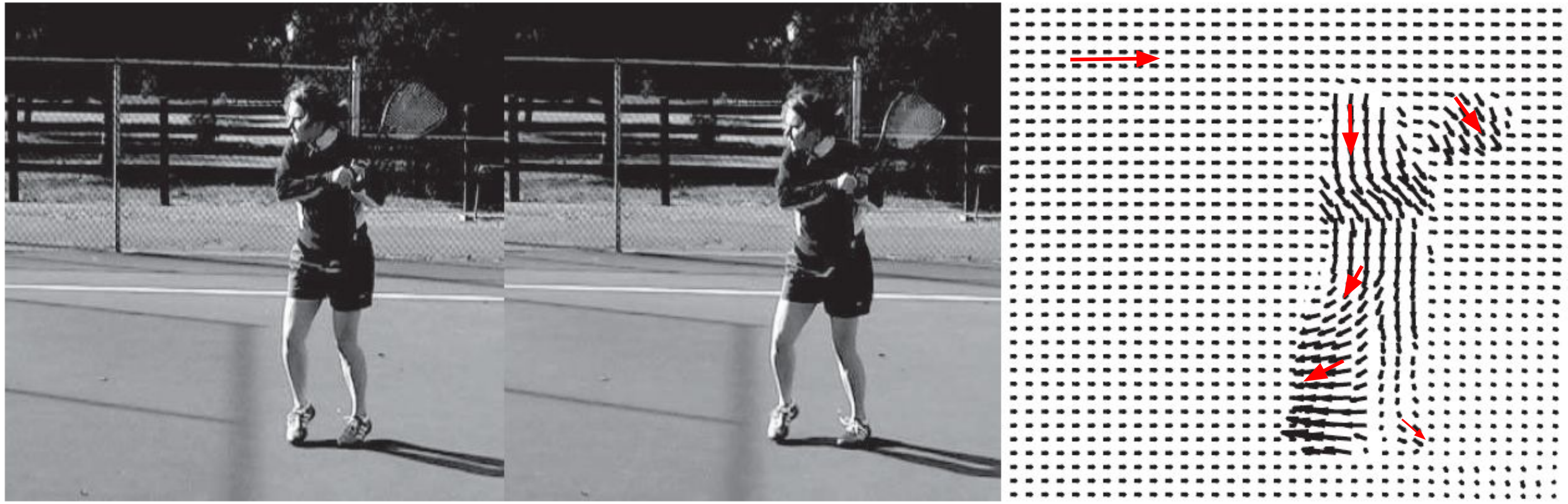
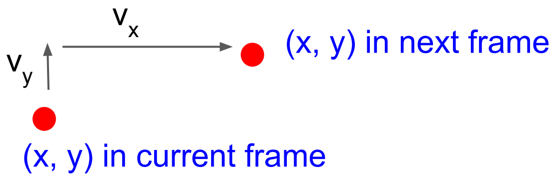


Figure 24.10 Two frames of a video sequence. On the right is the optical flow field corresponding to the displacement from one frame to the other. Note how the movement of the tennis racket and the front leg is captured by the directions of the arrows. (Courtesy of Thomas Brox.)

24.2.3 Representing Optical

- For any point (x, y) in a given image, the optical flow can be described by **v_x(x, y)** in the x direction, and **v_y(x, y)** in the y direction
- To measure optical flow (i.e. v_x and v_y) we need to find corresponding points between one time frame and the next



Current Frame Next Frame

- How to find the corresponding points between one time frame and the next? In other words, how to superimpose the current image's (x, y) and the next image's (x, y)?

How to find the corresponding points between one time frame and the next?

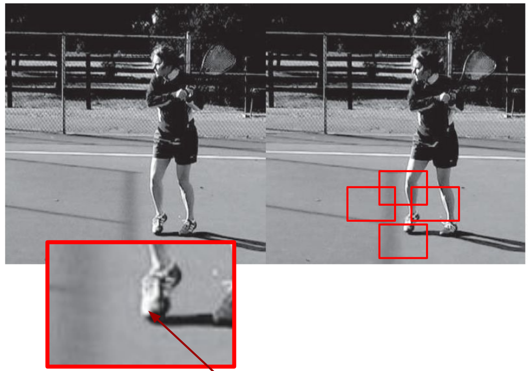
- Image patches around corresponding points have similar intensity patterns
    - This idea can be used to find most similar (matching) patches
- Consider a block of pixels centered at pixel p, (x_0, y_0), at time t
    - Step 1: We compare this block of pixels with many pixel blocks centered at various candidate pixels $(x_0 + D_x, y_0 + D_y)$ at time $t + D_t$

      $(x_0 + D_x, y_0 + D_y)$ is the center of a candidate block at time D_t
    - Step 2: Calculate Sum of Squared Differences (SSD) of intensities for each of the many blocks

$$\mathrm{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x,y,t) - I(x + D_x, y + D_y, t + D_t))^2$$

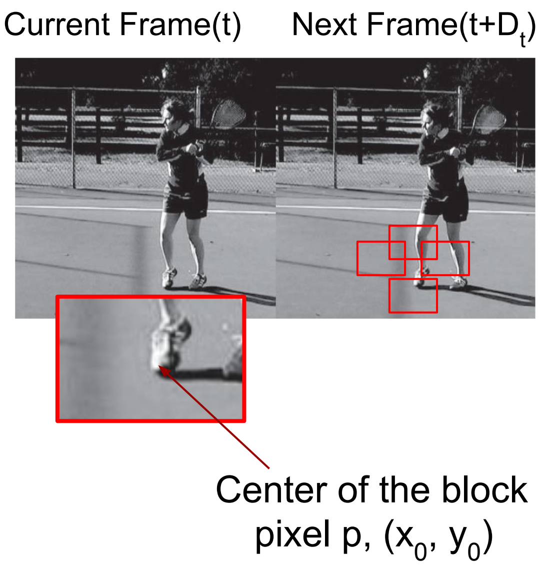
      (x, y) ranges over pixels in the block centered at (x_0, y_0)
    - Step 3: Find the (D_x, D_y) that minimizes SSD

Current Frame(t) Next Frame($t+D_t$)



24.2.3 How to Calculate Optical Flow?

- (x_0, y_0) is the pixel in the current frame (at time = t)
  - for which we would like to compute the optical flow
- (D_x, D_y) gives how much the pixel has moved in the next frame
- (x_0+D_x, y_0+D_y) is the location of the pixel in new frame (D_t)
- The optical flow at (x_0, y_0) is then
  - $(v_x, v_y) = (D_x/D_t, D_y/D_t)$
  - The unit of optical flow is pixels/sec



24.2.4 Segmentation of Images

- Segmentation is the process of breaking an image into regions of similar pixels (based on brightness, color, and texture)
- There are two approaches to segmentation
    - 1. Detect the boundaries of the regions and then compute regions
    - 2. Directly detect the regions themselves (more powerful)

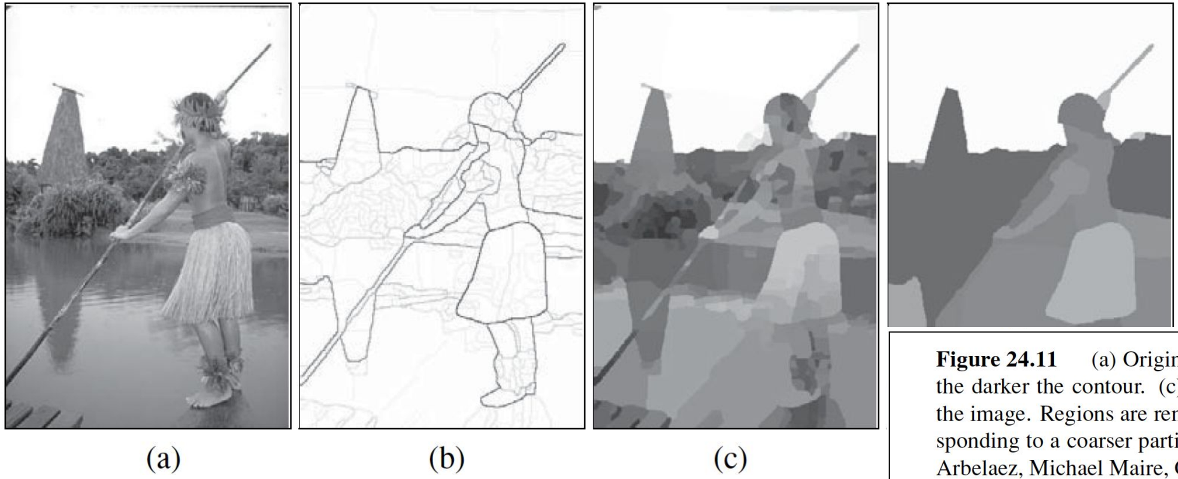
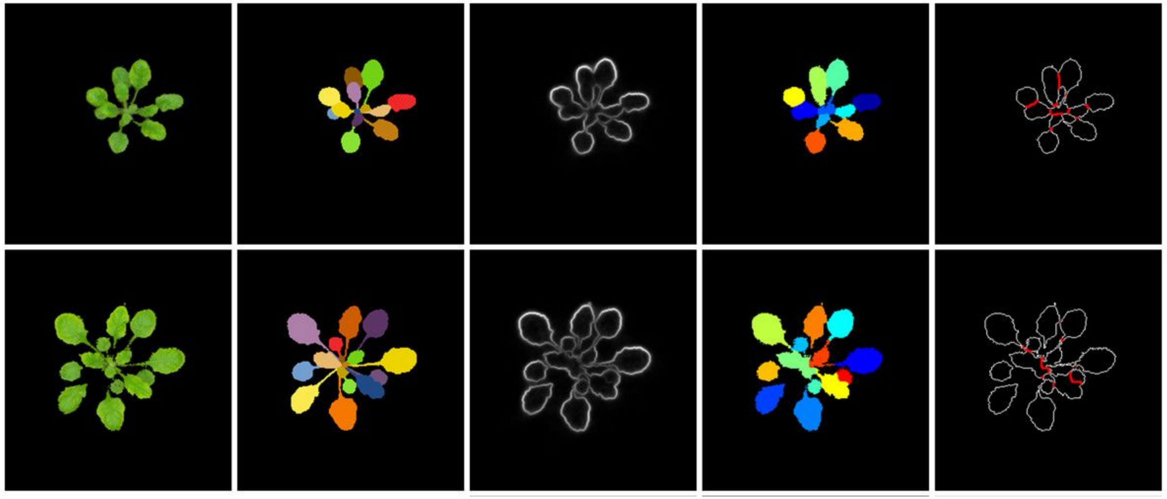
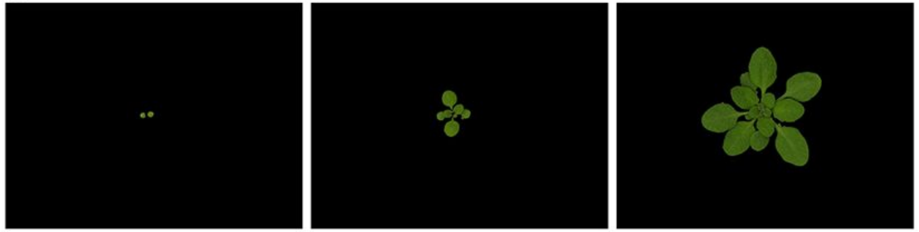


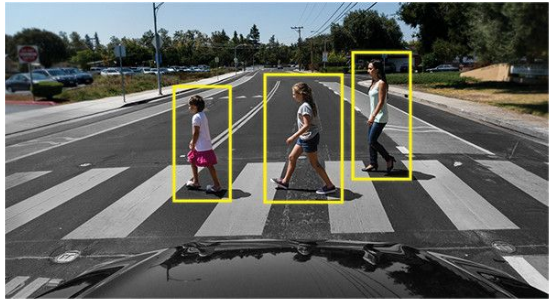
Figure 24.11 (a) Original image. (b) Boundary contours, where the higher the P_b value, the darker the contour. (c) Segmentation into regions, corresponding to a fine partition of the image. Regions are rendered in their mean colors. (d) Segmentation into regions, corresponding to a coarser partition of the image, resulting in fewer regions. (Courtesy of Pablo Arbelaez, Michael Maire, Charles Fowlkes, and Jitendra Malik)

Applications of Image Segmentation





Counting the number of leaves in plants is easier than..



.. detecting pedestrians in an image

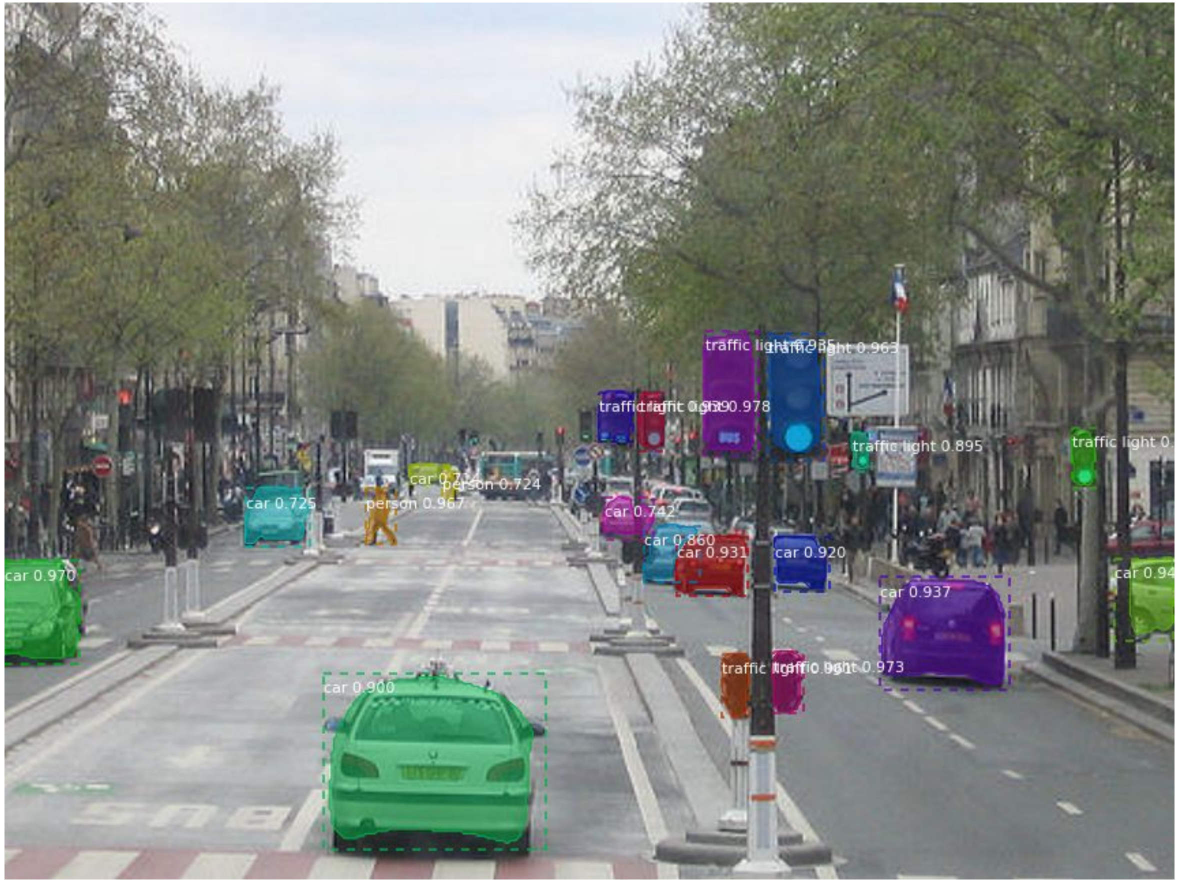
How do we define segments?

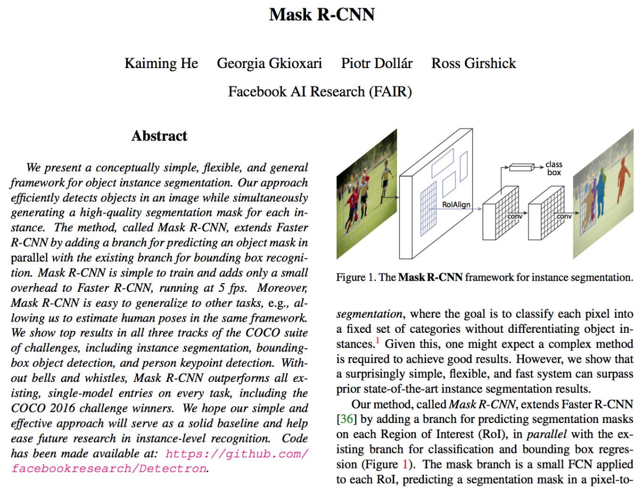
Is a definition based on color, brightness, and texture sufficient to detect pedestrians?

24.2.4 Segmentation of Images

- Segmentation based purely on low-level, local attributes such as brightness and color cannot be expected to deliver the final correct boundaries of all the objects in the scene
- To reliably find object boundaries we need high-level knowledge of the likely kinds of objects in the scene
- Representing this knowledge is a topic of active research

Mask R-CNN (2018) - Facebook AI Research





Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick

Facebook AI Research (FAIR)

Abstract

We present a conceptually simple, flexible, and general framework for object instance segmentation. Our approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. The method, called Mask R-CNN, extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps. Moreover, Mask R-CNN is easy to generalize to other tasks, e.g., allowing us to estimate human poses in the same framework. We show top results in all three tracks of the COCO suite of challenges, including instance segmentation, bounding-box object detection, and person keypoint detection. Without bells and whistles, Mask R-CNN outperforms all existing, single-model entries on every task, including the COCO 2016 challenge winners. We hope our simple and effective approach will serve as a solid baseline and help ease future research in instance-level recognition. Code has been made available at: `https://github.com/facebookresearch/Detectron`.

Figure 1. The **Mask R-CNN** framework for instance segmentation.

segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.[1] Given this, one might expect a complex method is required to achieve good results. However, we show that a surprisingly simple, flexible, and fast system can surpass prior state-of-the-art instance segmentation results.

Our method, called *Mask R-CNN*, extends Faster R-CNN [36] by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in *parallel* with the existing branch for classification and bounding box regression (Figure 1). The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-

https://github.com/matterport/Mask_RCNN

https://arxiv.org/abs/1703.06870

Summary

- Edge detection and computation of optical flow are examples of low-level image processing operation
- Image segmentation is an example of a mid-level image processing operation
- Convolution is an operation (f*g) with huge potential and usefulness
    - For example, we can blur, edge detect, etc. using various kernels (filters)