Computer Vision
for HCI

Image Segmentation
and
Template Matching

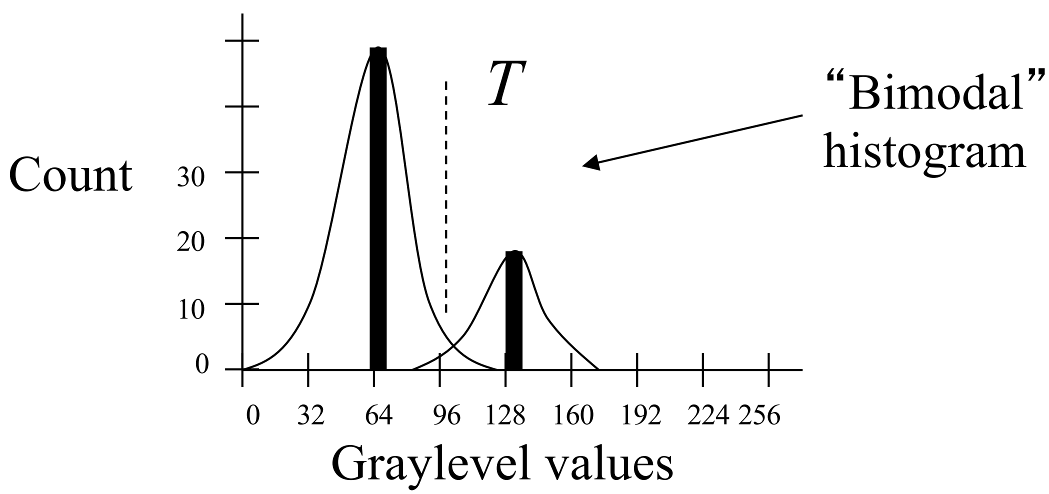
Image Segmentation

- Goal: Partition an image into distinct regions containing each pixels with similar attributes
- Use "*discontinuity*" or "*similarity*" approach
  - *Discontinuity:* segment into regions based on discontinuity (gradient or edge detection)
  - *Similarity:* Merge similar regions (clustering, region growing etc.)
- Topics
  - Simple Segmentation
  - Segmentation by Clustering
  - Superpixel Segmentation

Goal

Recap: Otsu's Simple Segmentation

- Distribution of graylevels can be used to determine binary threshold
- <u>Histogram</u> graphs number of pixels in the image with a particular graylevel, as a function of the possible graylevels
  - Find peaks and set threshold between peaks



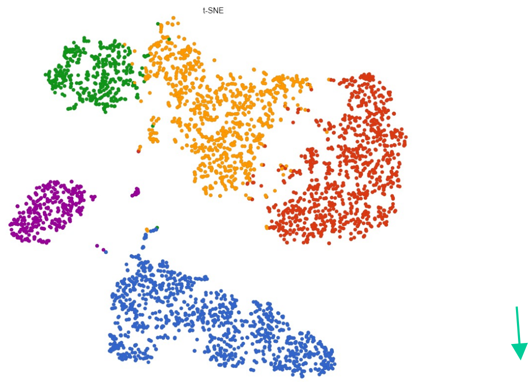
Otsu's Method

- "A threshold selection method from graylevel histograms", IEEE Trans on Sys., Man, and Cyb., Vol 9, No 1, pp 62-66, 1979.
  - Basic idea: threshold is chosen such that the division in the histogram yields the largest reduction in standard deviation of the pixel intensities (black, white)
  - Matlab: graythresh()

Image Segmentation by Clustering

Identify groups of pixels that “go together”

Each “point” is a pixel in
color space (3-D: RGB)



- K-Means
- Mean-Shift Clustering

K-Means

K-means Clustering

- Each "point" is a 3-D vector of color (RGB)
- Initialization:
  - Choose k cluster centers (how pick k?)
- Repeat:
  - **Assignment step:**
    - For every point, find its closest center
  - **Update step:**
    - Update every center as the mean of its assigned points
- Until:
  - The maximum number of iterations is reached, or
  - No changes during the assignment step, or
  - The average distortion per point drops very little

K-means: Initialization

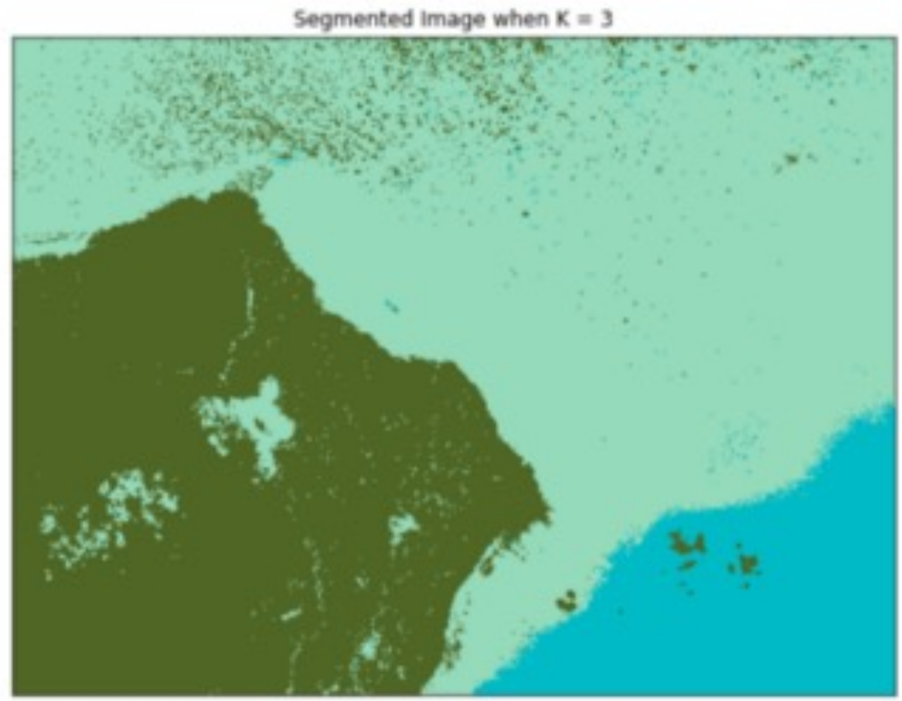
- K-means is *extremely sensitive* to initialization
- Bad initialization can lead to
  - Poor convergence speed
  - Poor overall clustering
- How to initialize?
  - Randomly from data
  - Try to find K "spread-out" points
- Try multiple initializations and pick best result
  - Minimize total "distortion" (sum of distances of points from their cluster centers)

Cluster center Data

$$J(\mu,r) = \sum_{n=1}^{N} \sum_{k=1}^{K} \delta_{nk} ||x_n - \mu_k||^2$$

Whether x_j is assigned to μ_i

Example



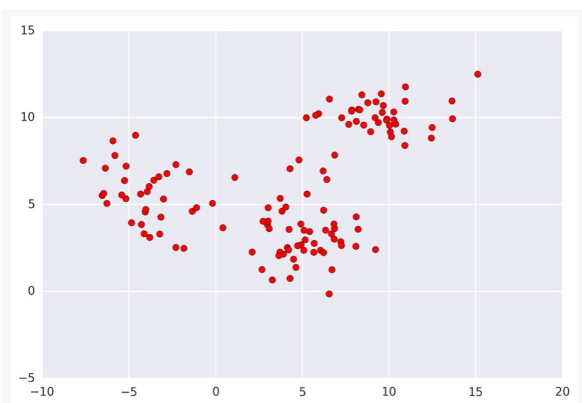
Mean-Shift

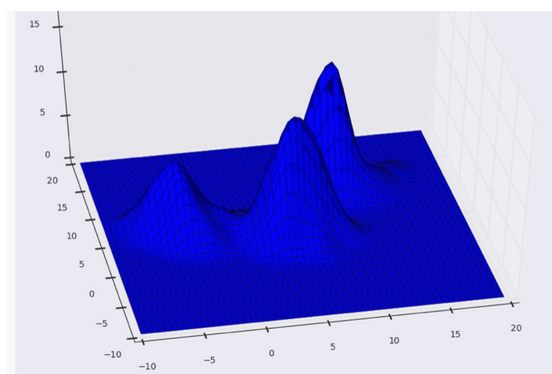
Mean-Shift segmentation

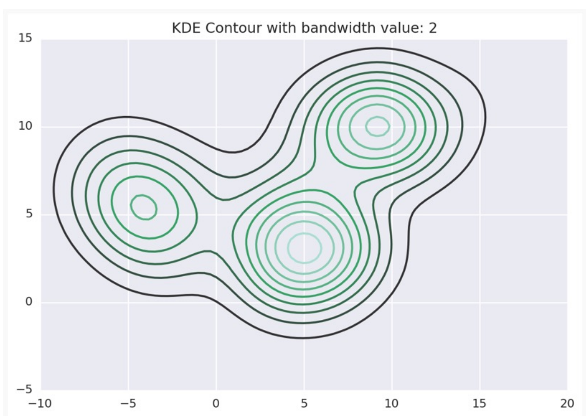
- Recall Mean-Shift tracking lecture…
- Used here for unsupervised clustering
  - Unlike K-means, do not need initial ‘K’
- Assigns the data points to clusters iteratively by shifting points towards the local modes
  - Mode: The highest density of data points in the region, in the context of the Mean-Shift

Mean-Shift clustering

- **Relate to "Kernel Density Estimation"**
  - Imagine the data sampled from a probability distribution
  - Estimate the underlying distribution (also called the probability density function) for a set of data
    - Place kernel on each point (think weighing function)
    - Add all the individual kernels generates a probability surface (e.g., density function)

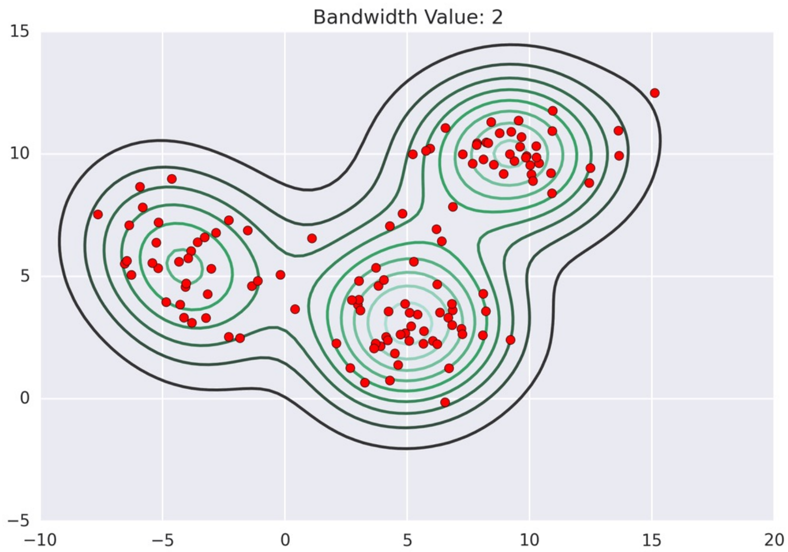


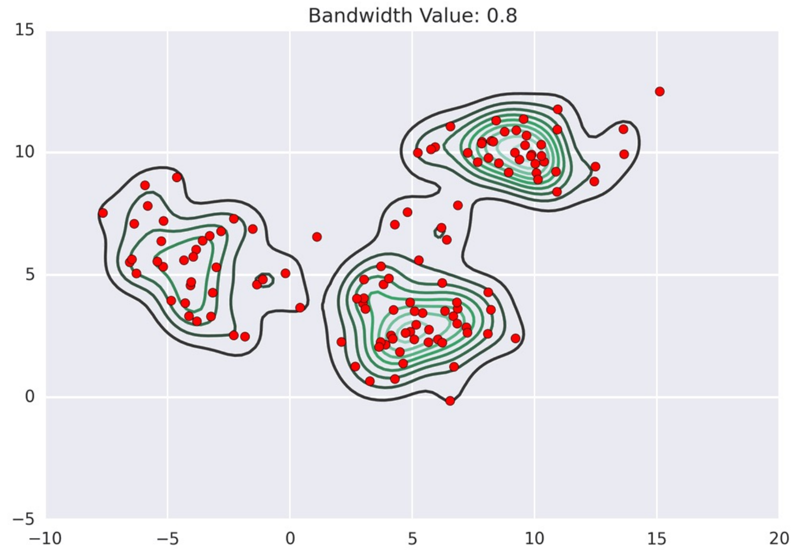




Mean-Shift Clustering

- Idea:
  - Make points climb up the hill to the nearest peak on the density surface
  - *Iteratively* shift each point uphill until it reaches a peak





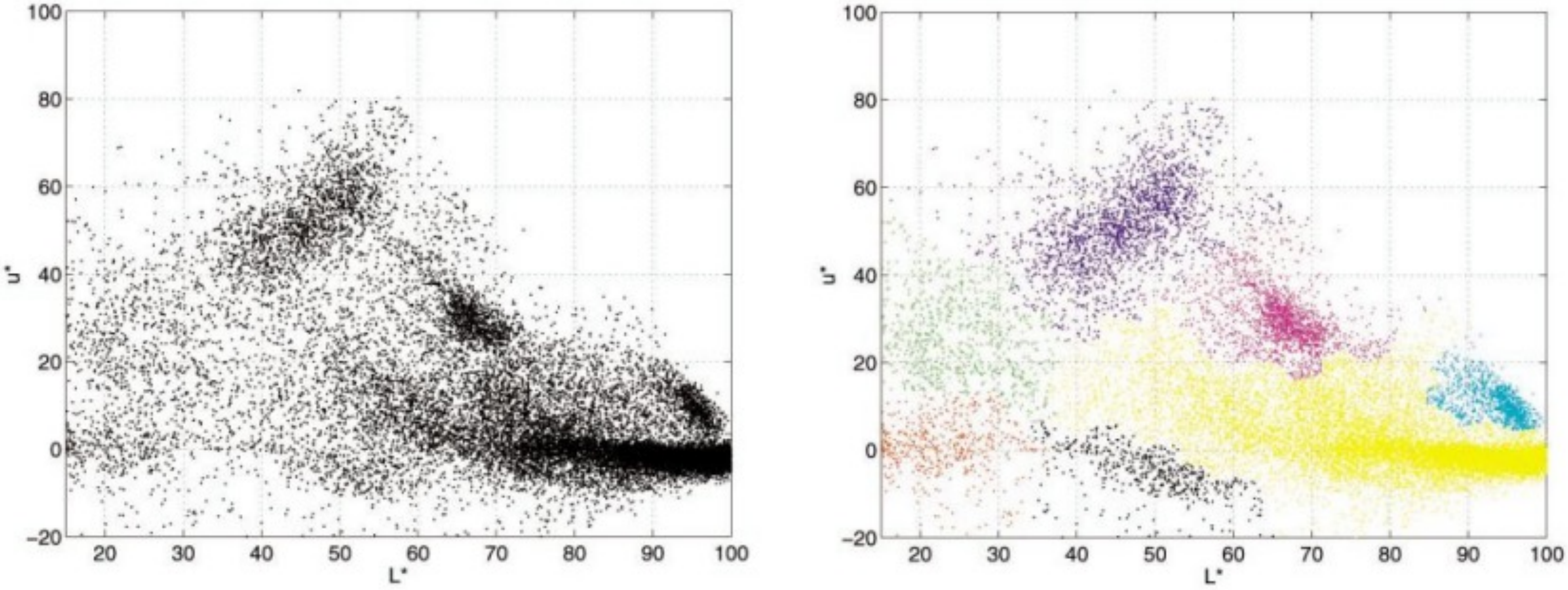
Mean-Shift Algorithm

- Define x
  - Color only: $[R, G, B]$
  - Spatial and Color: [*x-loc, y-loc, R, G, B*]
- For each datapoint x, find the neighboring points $N(x)$ of x, given Kernel function/window K
- For each x, calculate the ***mean shift*** $m(x)$:

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

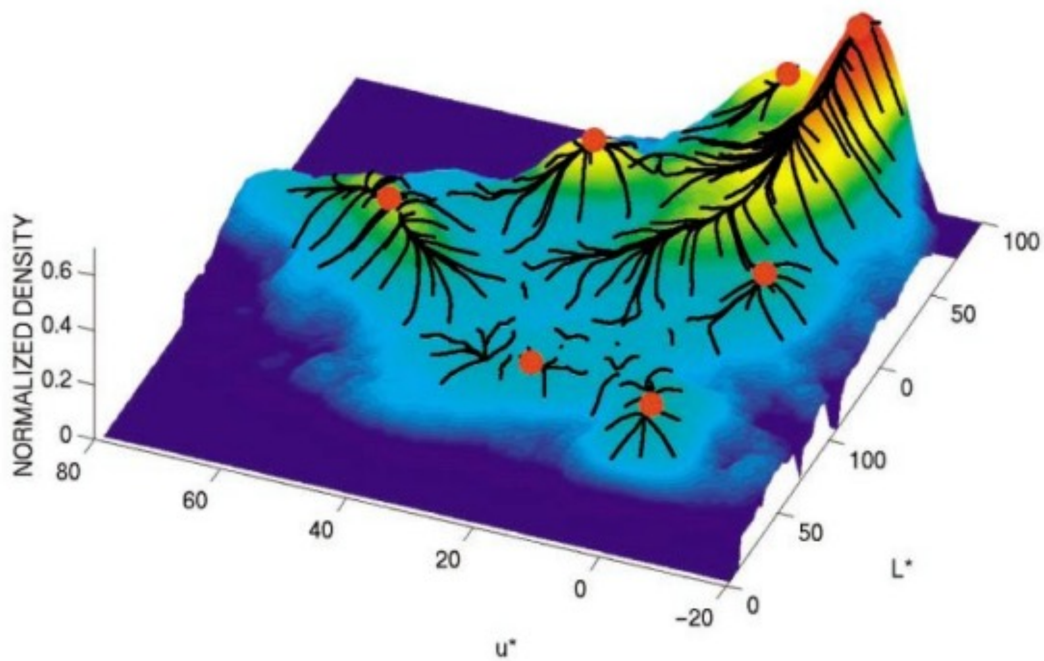
- Then update each x with $x \leftarrow m(x)$
- Repeat *n times* or until the points stabilize

More details:
https://spin.atomicobject.com/2015/05/26/mean-shift-clustering/

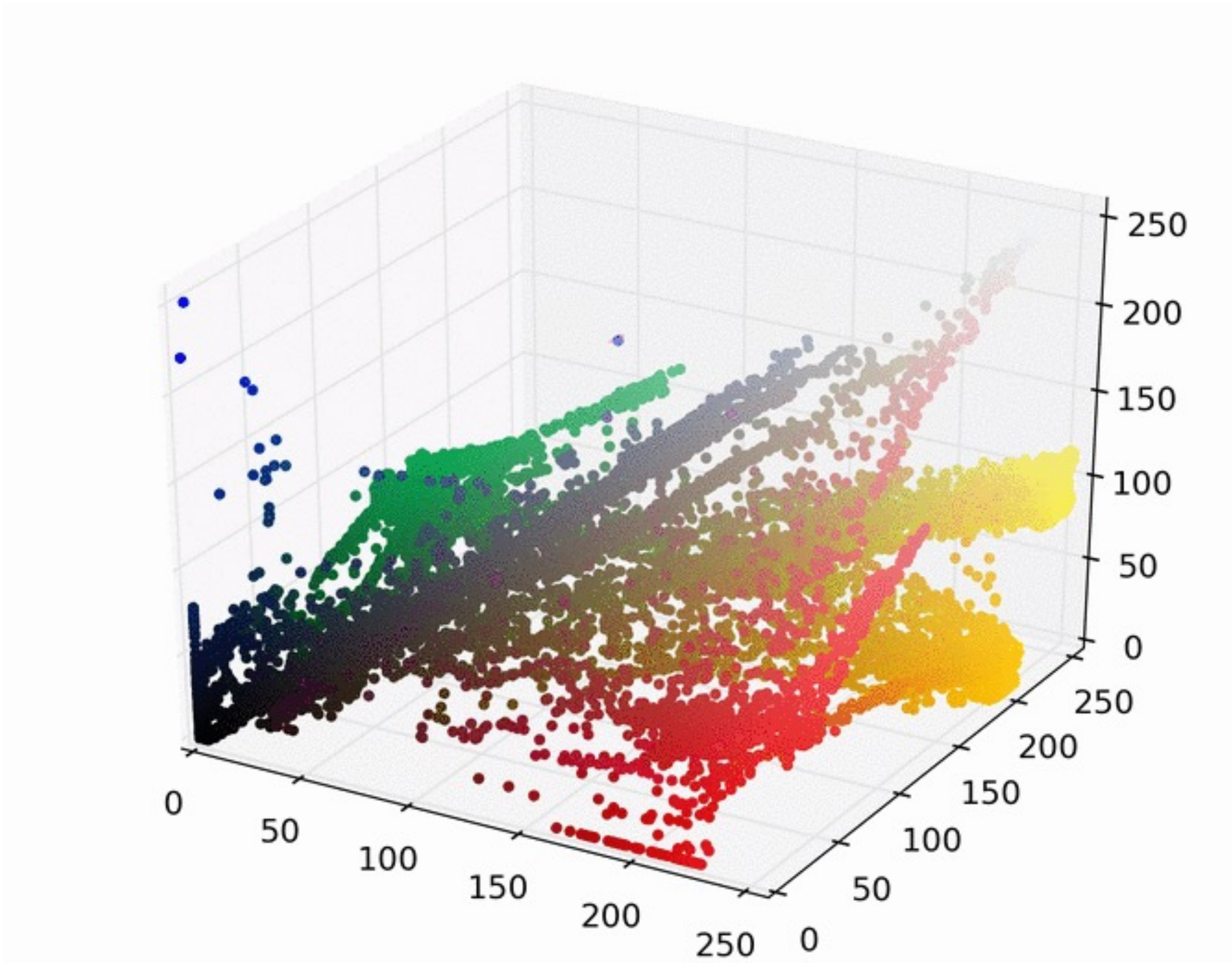

100
80
60
40
20
0
−20
u*
20
30
40
50
60
70
80
90
100
L*
100
80
60
40
20
0
−20
u*
20
30
40
50
60
70
80
90
100
L*

(a)

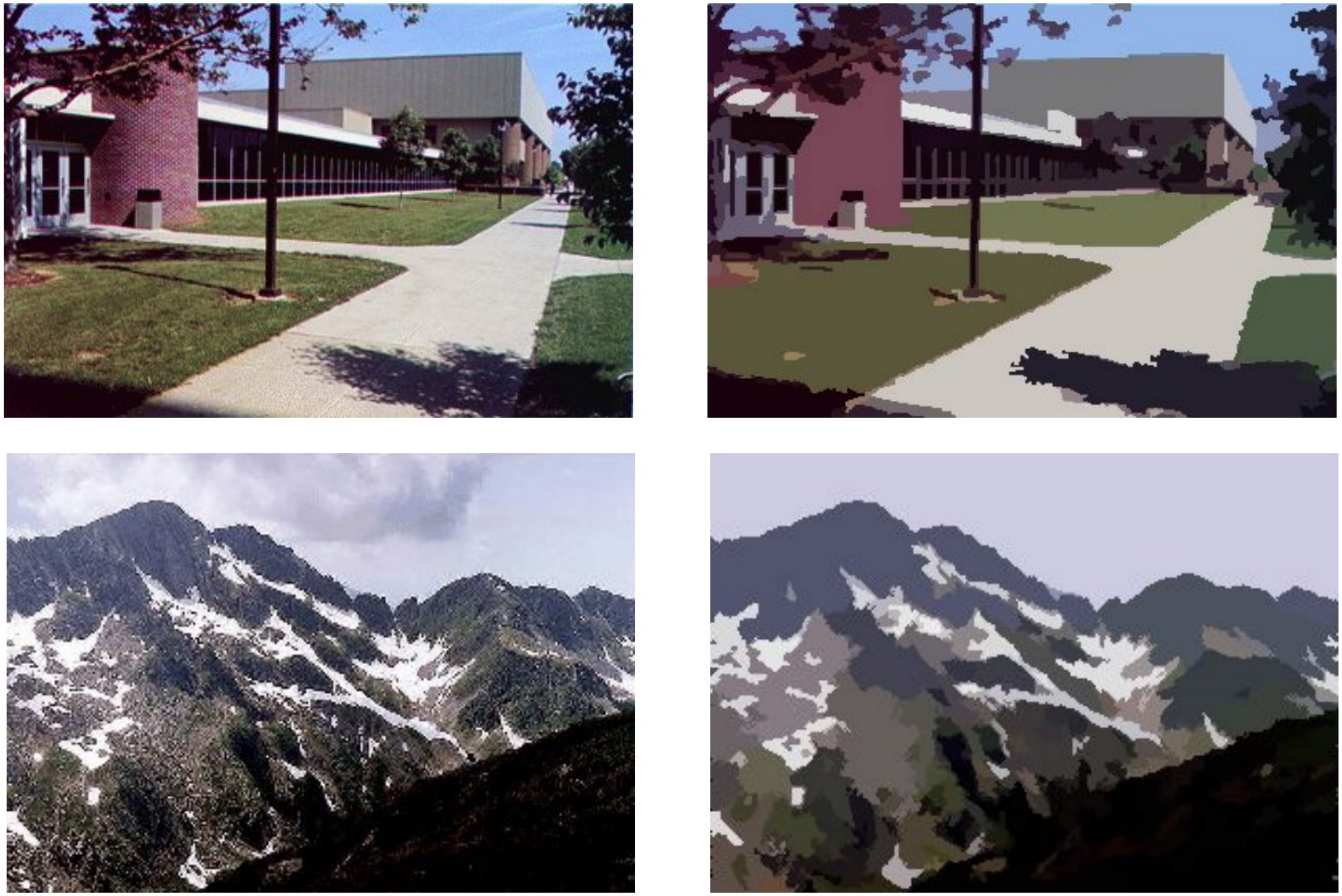
(b)


NORMALIZED DENSITY
0.6
0.4
0.2
0
80
60
40
20
0
−20
u*
100
50
0
100
50
0
L*

RGB Visualization



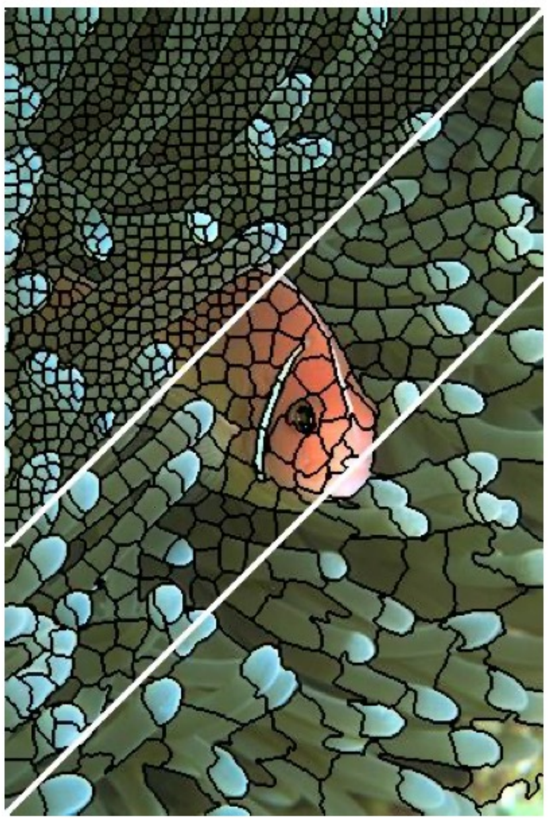
Examples

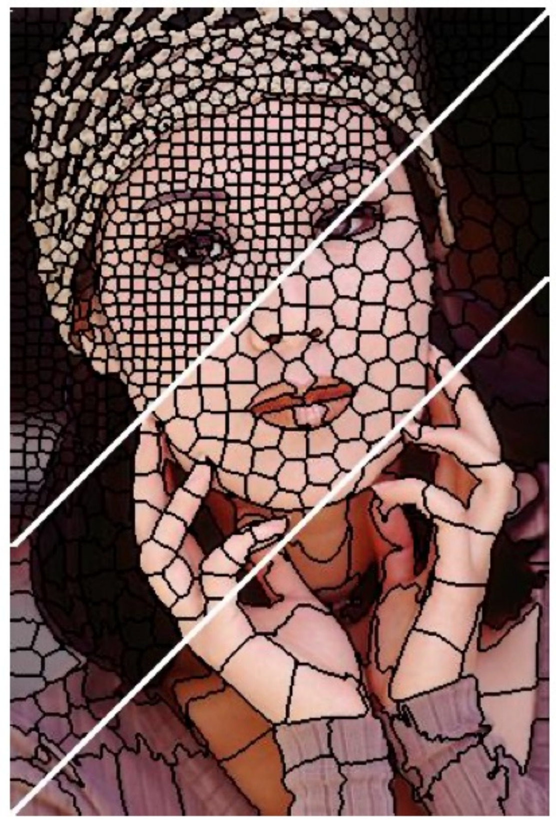


Superpixels

Superpixel Segmentation

- "Superpixels" capture local visual redundancy in the image
  - **SLIC** Superpixel algorithm





SLIC Superpixel Segmentation

- Generated by clustering pixels based on:
  - Color similarity, and
  - Spatial proximity in the image
- Employ 5-D vector per pixel: [lab, x, y]
  - lab = pixel color vector in CIELAB color space
    - (l): intensity, (a, b): color
    - Perceptual color space with Euclidean distance properties
  - x, y = pixel position (or row, col)

SLIC Superpixel Algorithm

- Initially choose K = number of desired superpixels
- Divide image into regular "grid" steps S (for the K)

Algorithm 1 Efficient superpixel segmentation

1: Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.
2: Perturb cluster centers in an $n \times n$ neighborhood, to the lowest gradient position.
(to the pixel with smallest lab gradient magnitude in 3x3 region)
3: **repeat**
4:     **for** each cluster center C_k **do**
5:         Assign the best matching pixels from a $2S \times 2S$ square neighborhood around the cluster center according to the distance measure (see next slide)
6:     **end for**
7:     Compute new cluster centers and residual error E $\{L1$ distance between previous centers and recomputed centers$\}$
8: **until** $E \leq$ threshold

Notation

N	Number of pixels in the input image
K	Number of Superpixels used to segment the input image
N/K	Approximate size of each superpixel
$\boldsymbol{S = \sqrt{N/K}}$	For roughly equally sized superpixels there would be a superpixel centre at every grid interval S

SLIC Superpixel Segmentation

- Distance function between 2 pixels:

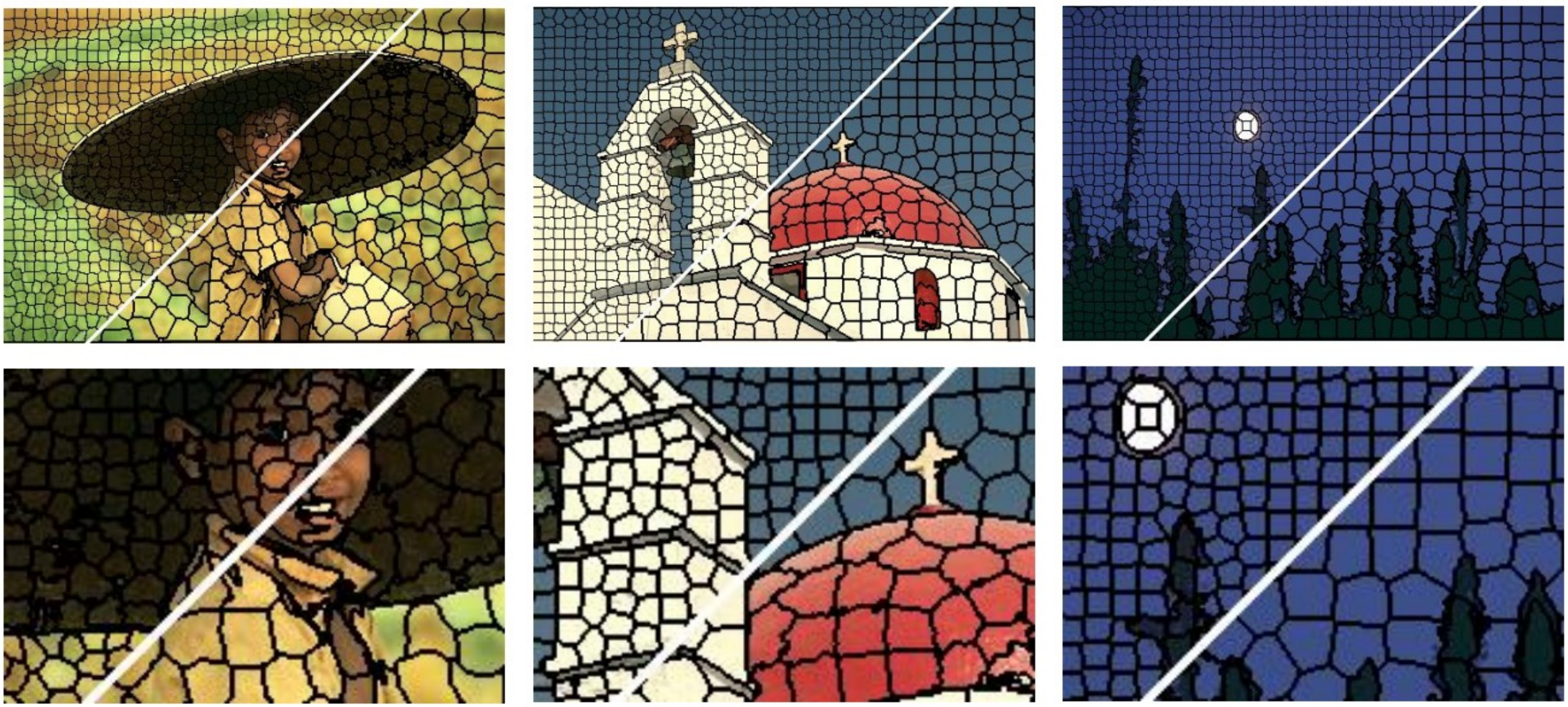
$$D_s = d_{lab} + \frac{m}{S} d_{x,y}$$

d_{lab} : lab distance (Euclidean) between the 2 pixels.

$\frac{1}{S} d_{x,y}$: Euclidean spatial distance, normalized by grid interval S.

m: **compactness** control of a super pixel. Large values make it more compact.

Superpixels: More Examples



Approx. 300 to 100 superpixels

Template Matching

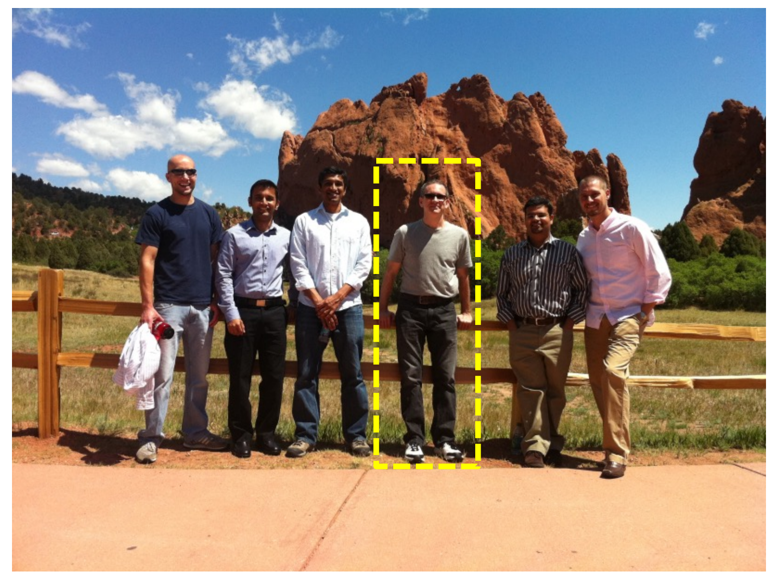
Template Matching Intro

- Want to find areas of a search image that are similar to given template image *T*

Template Image *T*

Search Image

Best Matching Patch in Search Image



General Approaches

- Template-Based:
  - Utilize raw template (pixels) and find best matching patches in search image
    - Sum-of-absolute differences (SAD)
    - Sum-of-squared differences (SSD)
    - Normalized cross-correlation (NCC)

1) Sum-of-Absolute Differences (SAD)

- Compute **absolute differences of pixel intensities** of template T and image patch P extracted from search image (note that P is same size as template T)

$$SAD(P,T) = \sum_{R,G,B} \sum_{x,y} \left| P(x,y) - T(x,y) \right|$$

- Compute SAD for all unique patch locations within the search image
- Keep patch with minimum SAD or patches with SAD less than given threshold

SAD Example

Template Image *T*

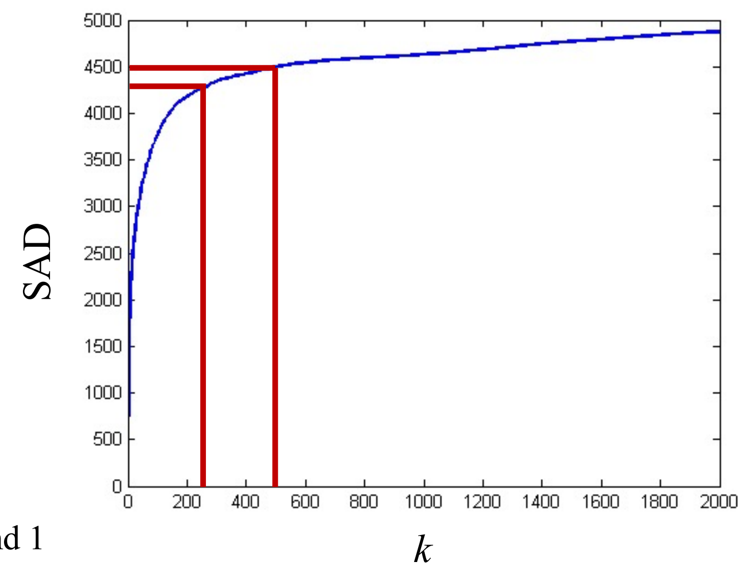
Search Image

<u>Negative</u> SAD,
Origin is in center of patch

SAD Example

k^{th} best matching patch

$k = 1$ $k = 10$ $k = 50$ $k = 100$ $k = 150$ $k = 200$ $k = 250$ $k = 500$ $k = 1000$



Similar???

*Pixel values scaled between 0 and 1

2) Sum-of-Squared Differences (SSD)

- Similar to SAD, but replace absolute differences with **squared differences**

$$SSD(P,T) = \sum_{R,G,B} \sum_{x,y} \left(P(x,y) - T(x,y)\right)^2$$

- Compute SSD for all unique patches within the search image
- Keep patch with minimum SSD

SSD Example

Template Image T

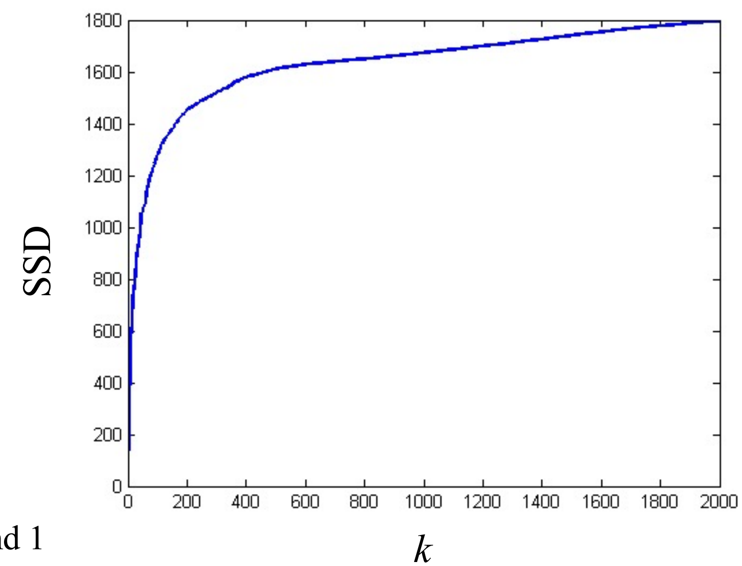
Search Image

<u>Negative</u> SSD,
Origin is in center of patch

SSD Example

k^{th} best matching patch

$k = 1$ $k = 10$ $k = 50$ $k = 100$ $k = 150$ $k = 200$ $k = 250$ $k = 500$ $k = 1000$



*Pixel values scaled between 0 and 1

Illumination Changes

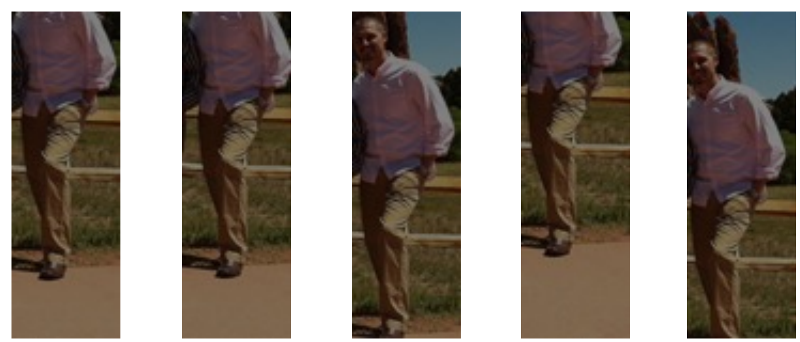
- SAD and SSD can work well if the template and search images have the same brightness
  - Problem: images can have varying illumination conditions

Template Image T

Search Image

k^{th} best matching patch using SSD

$k = 1$ $k = 10$ $k = 50$ $k = 100$



$k = 150$ $k = 200$ $k = 250$ $k = 500$ $k = 1000$

3) Normalized Cross-Correlation
(NCC)

- Normalize images to remove variations from illumination conditions

Mean of pixel values in patch
(each color computed independently)

$$NCC(P,T) = \sum_{R,G,B} \frac{1}{n-1} \sum_{x,y} \frac{\left(P(x,y) - \overline{P}\right) \cdot \left(T(x,y) - \overline{T}\right)}{\sigma_P \sigma_T}$$

constant; can just be calculated once.

Standard deviation of pixel values in patch
(each color computed independently)

Note: larger values of NCC better!

The maximum value is 1 when two 1-channel signals are exactly the same:

NCC Example

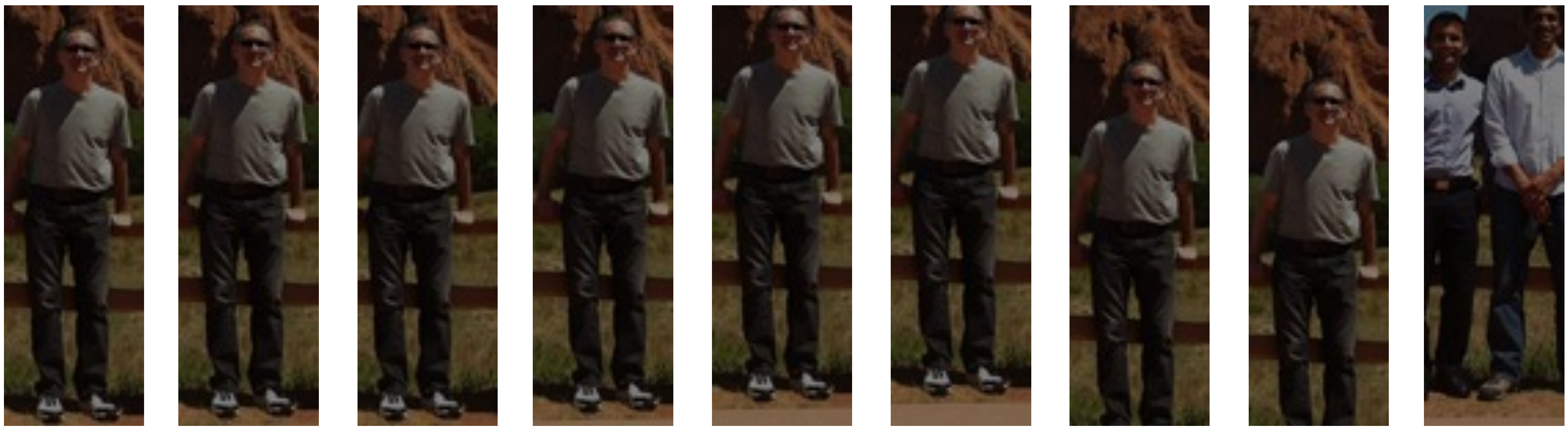
Template Image *T*

Search Image

NCC,
Origin is in center of patch

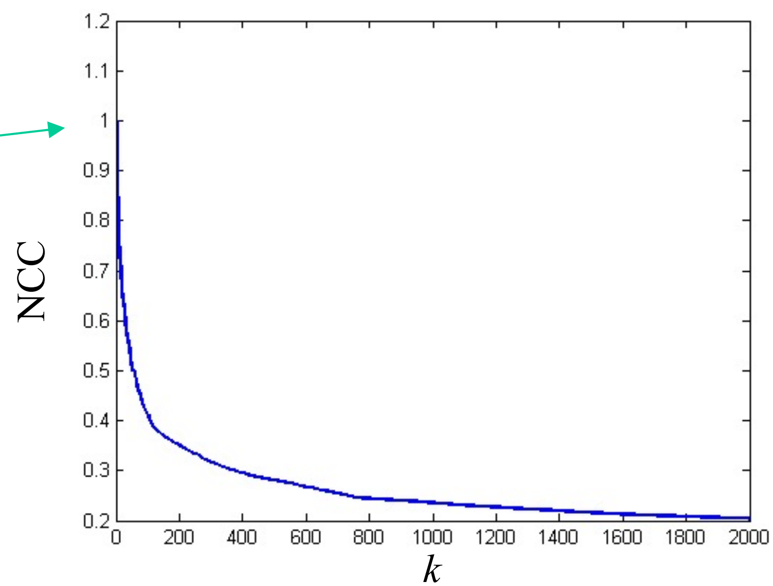
NCC Example

k^{th} best matching patch using **color** images



$k = 1$ $k = 10$ $k = 50$ $k = 100$ $k = 150$ $k = 200$ $k = 250$ $k = 500$ $k = 1000$

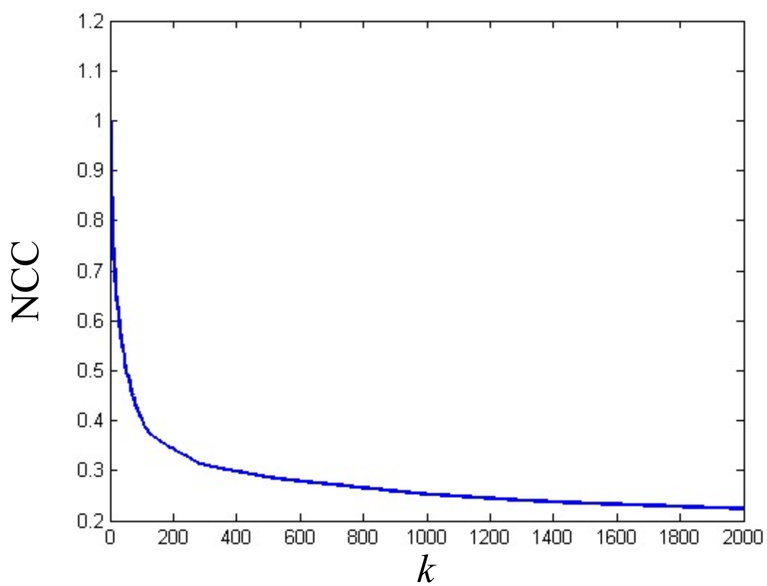
Divided NCC by 3 (for RGB)



NCC Example

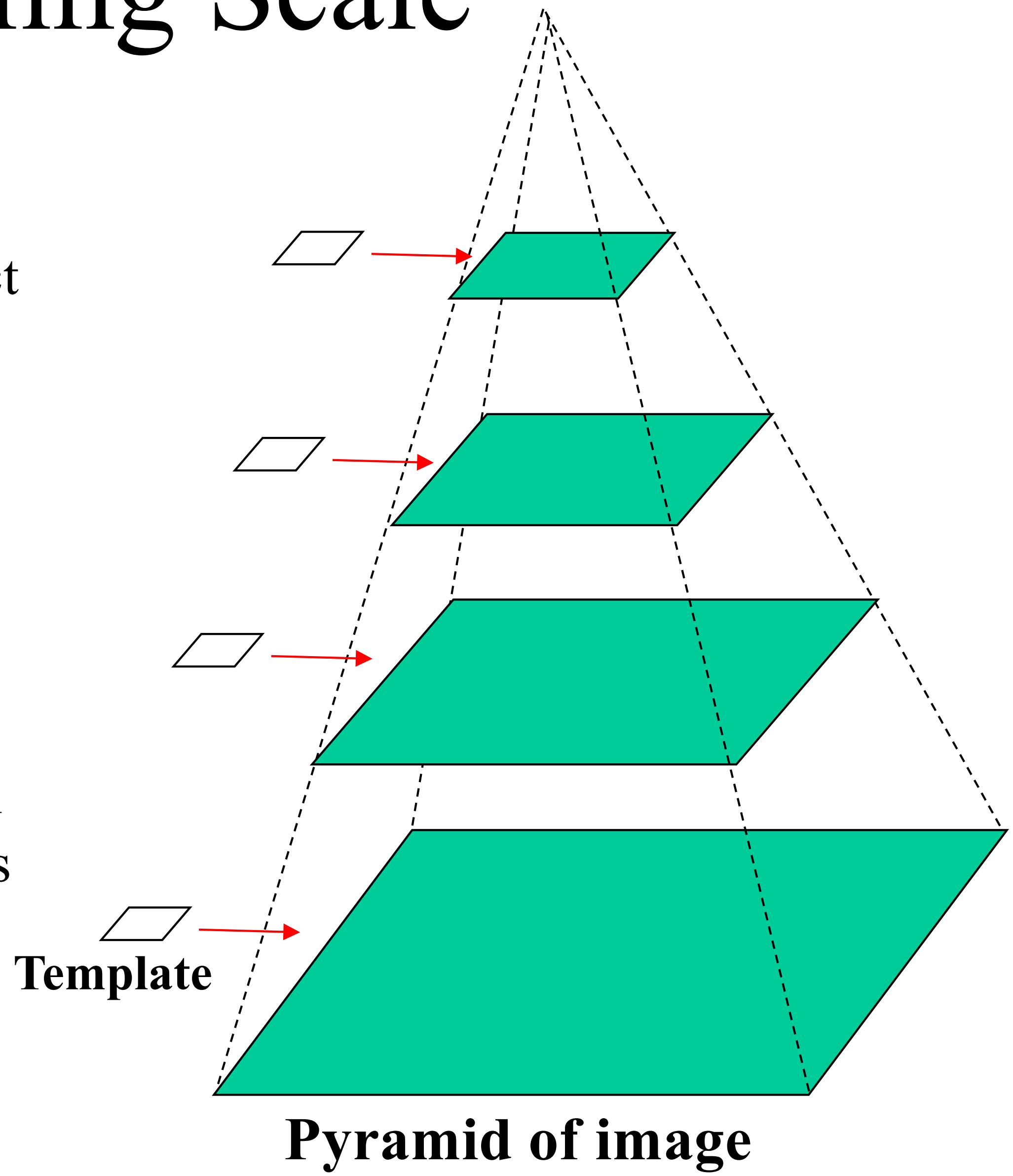
k^{th} best matching patch using **grayscale** images

$k = 1$ $k = 10$ $k = 50$ $k = 100$ $k = 150$ $k = 200$ $k = 250$ $k = 500$ $k = 1000$



Handling Scale

- Construct fixed-size template of the smallest size you want to detect
- Scan through image pyramid
  - Detects larger scales of object higher in the pyramid
- Efficient scanning method, with less pixels to examine overall
  - Instead of repeatedly scaling template and scanning original full-sized image multiple times



Summary

- K-Means
  - Choose cluster centers and label every pixel based on its nearest neighbor
  - Minimize total distortion
  - Sensitive to initialization
- Mean-Shift
  - Iteratively shifts data towards peaks
- Superpixel Segmentation
  - Clustering small pixel regions based on color similarity and proximity
- Template Matching
  - SAD, SSD, NCC