intel AI Academy

CONTOURS, SEGMENTATION, AND MATCHING

LEGAL NOTICES AND DISCLAIMERS

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2018, Intel Corporation. All rights reserved.

CONTOURS

CONTOURS

A contour is an assembled collection of edges that (hopefully) represent one object in the image.

- Can be used to automatically segment the different objects present in an image.

PREPROCESSING FOR CV2.FINDCONTOURS

Can use canny edges

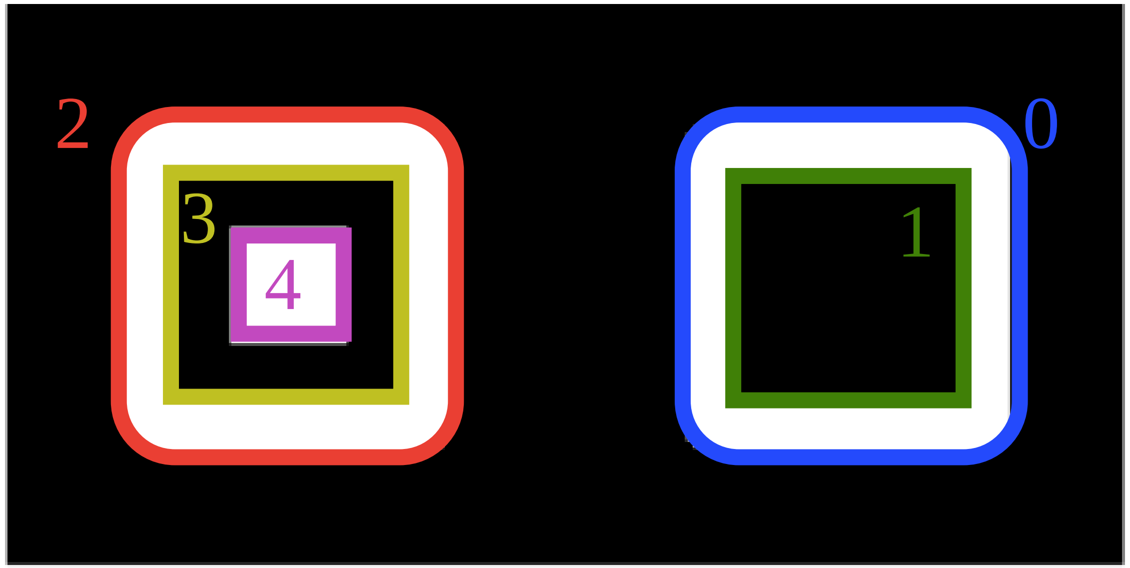
- Images created by cvCanny

Can use thresholded image where edges are boundaries between white and black

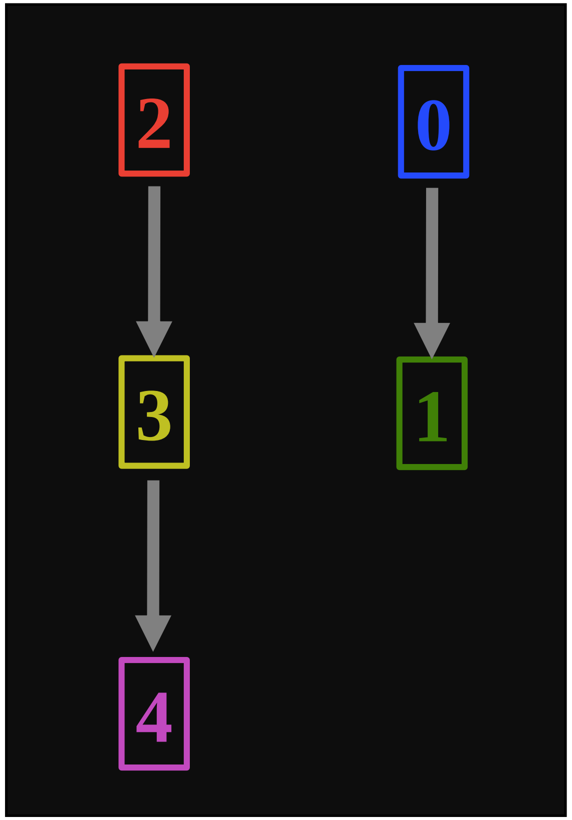
- Images created by cvThreshold or cvAdaptiveThreshold

Call cv2.findContours on result of Canny or thresholding.

CONTOUR HIERARCHY



Parent – Child Hierarchy

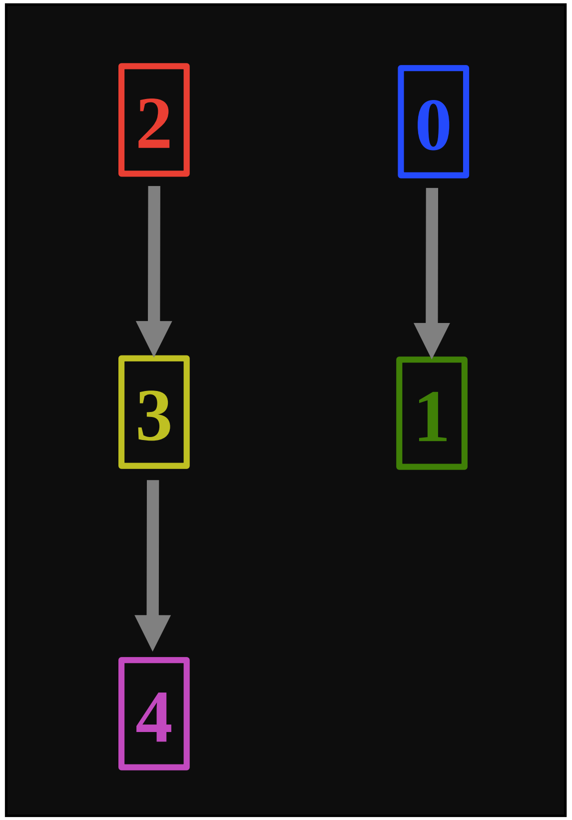


CONTOUR HIERARCHY

cvContour can represent data as a contour tree.

0 and 2 are the contours represented at the root, or parent, nodes.

Other contours are represented as children of 0 or 2.



CVFINDCONTOURS

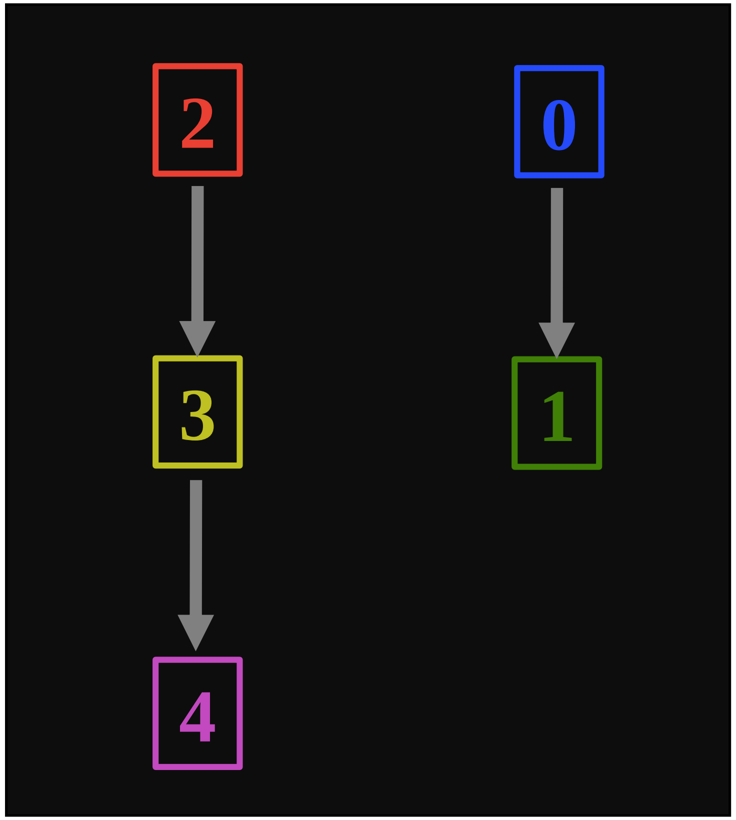
Input needs to be 8-bit binary image.

Make a copy of the image before using cvFindContours

- The original image will be written over by cvFindContours

OpenCV represents the contour hierarchy as an array:

[next, previous, first_child, parent]



METHOD VARIABLE: ARGUMENTS TO FIND CONTOURS

Method clarifies *how* the contour is being computed.

CV_CHAIN_CODE

- Output = sequence of vertices (Freeman chain code)

CV_CHAIN_APPROX_NONE

- All chain code points = contour points

CV_CHAIN_APPROX_SIMPLE

- Endpoints of horizontal, vertical, diagonal segments

CV_CHAIN_APPROX_TC89_L1

- Teh_Chin chain algorithm

CV_LINK_RUNS

- Links horizontal segments of 1s
- Limited to CV_RETR_LIST

MODE VARIABLE

Mode clarifies *what* contours should be found and desired format for return value.

Finds contours and uses horizontal and vertical links to link the found contours.

Mode options:

- CV_RETR_EXTERNAL
- CV_RETR_LIST
- CV_ RETR_CCOMP
- CV_RETR_TREE

CV_RETR_LIST

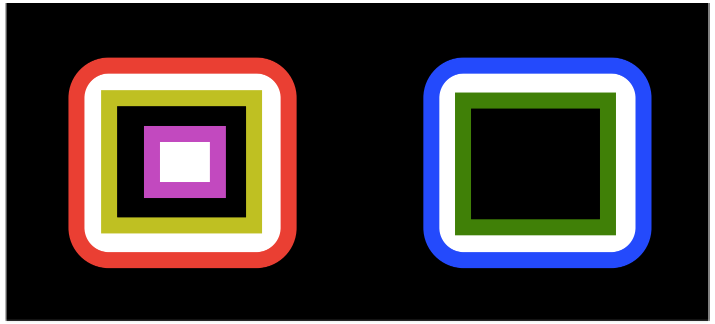
All contours are put into a list

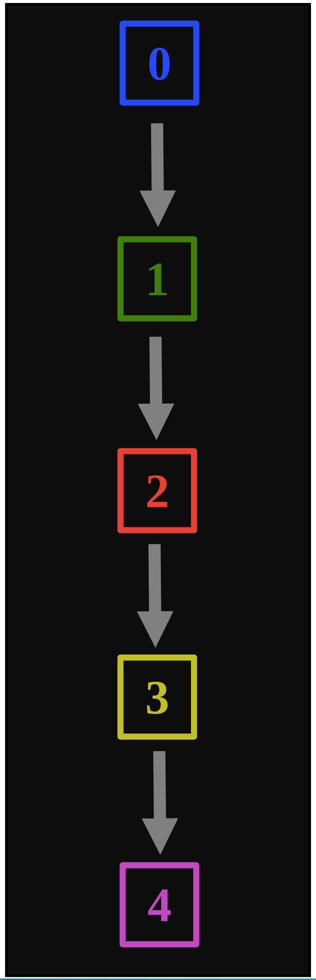
This list does not have parent – child relationships

Uses horizontal links

All contours are on same level (no actual hierarchy)

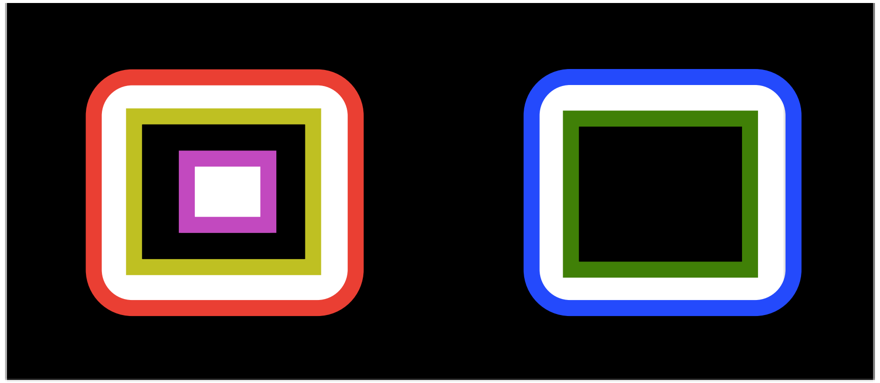
	next	prev	child	parent
0	1	-1	-1	-1
1	2	0	-1	-1
2	3	1	-1	-1
3	4	2	-1	-1
4	-1	3	-1	-1



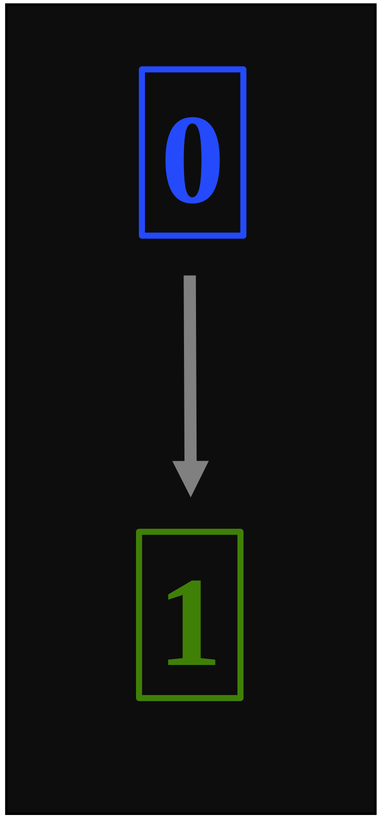


CV_RETR_EXTERNAL

Extreme outer contours



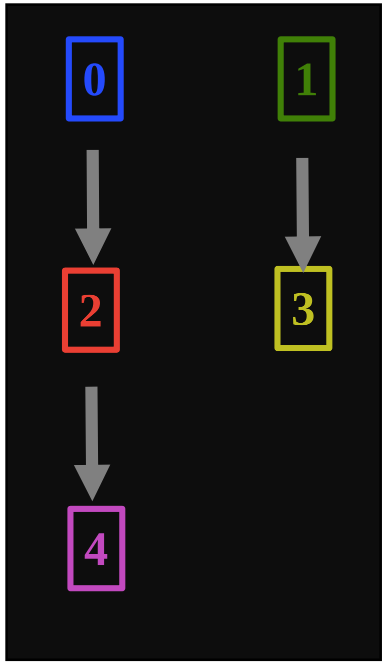
	next	prev	child	parent
0	1	-1	-1	-1
1	-1	0	-1	-1



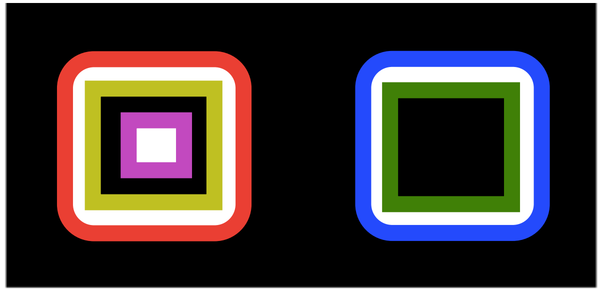
CV_RETR_CCOMP

All contours are used to generate a 2-level hierarchy

- Top-level hierarchy = external
- Bottom-level = internal
- Horizontal and vertical linkers



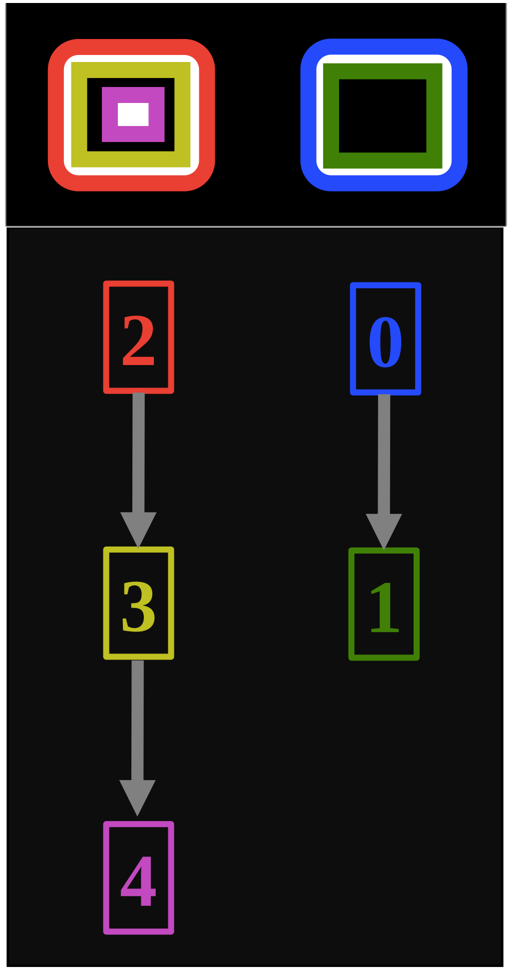
	next	prev	child	parent
0	1	-1	-1	-1
1	3	0	2	-1
2	-1	-1	-1	1
3	-1	1	4	-1
4	-1	-1	-1	3



CV_RETR_TREE

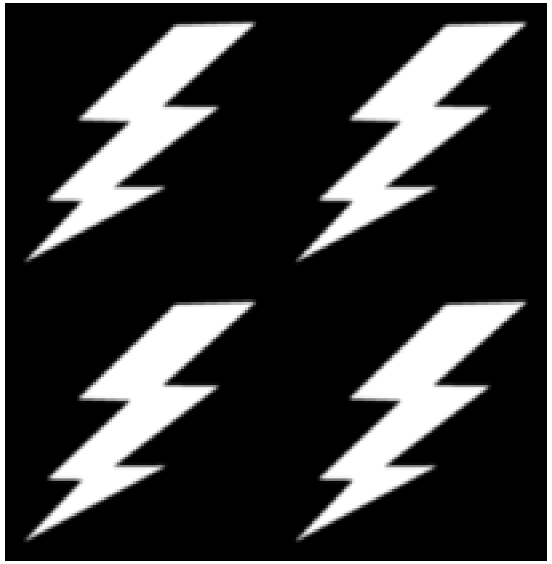
All contours are used to generate a full hierarchy

	next	prev	child	parent
0	2	-1	1	-1
1	-1	-1	-1	0
2	-1	0	3	-1
3	-1	-1	4	2
4	-1	-1	-1	3

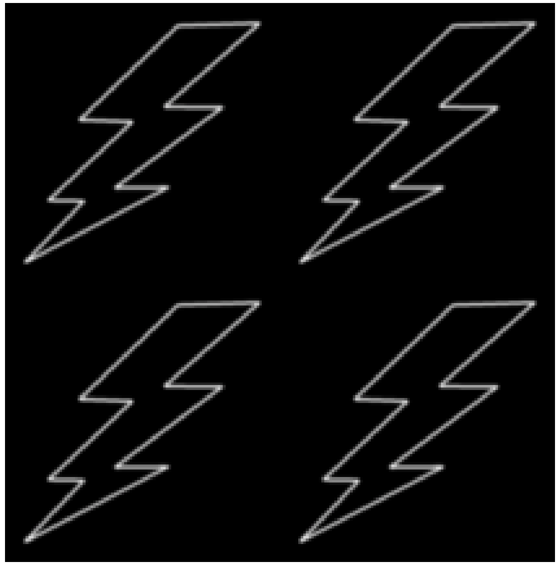


DRAWING CONTOURS

Lets look at a new image and draw contours with cv2.drawContours



Original



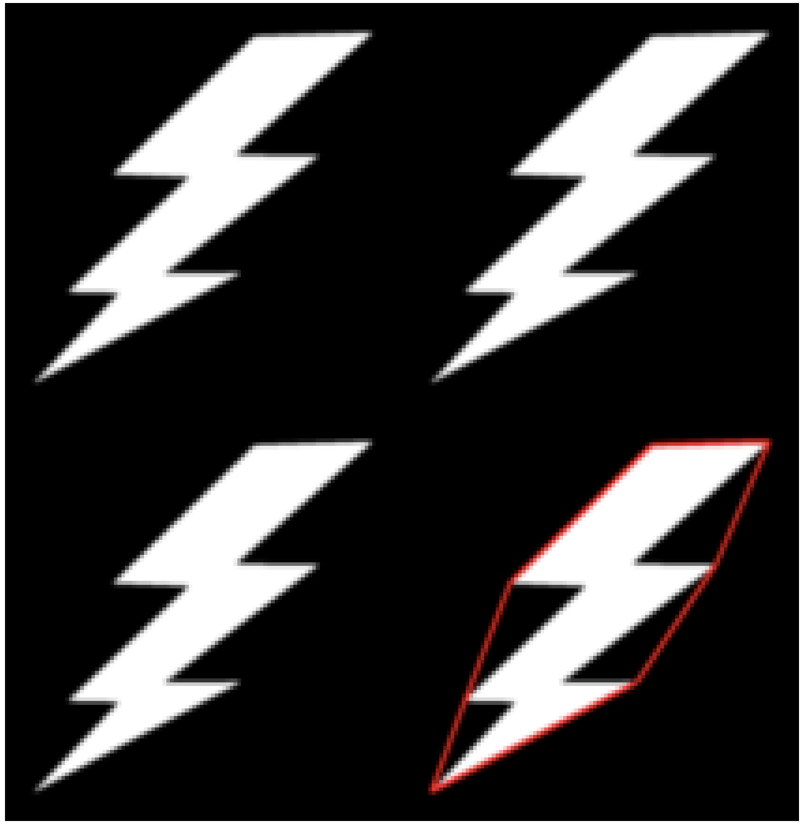
Edges

Edges used to compute contours

HULL BORDERS

Compute a convex hull

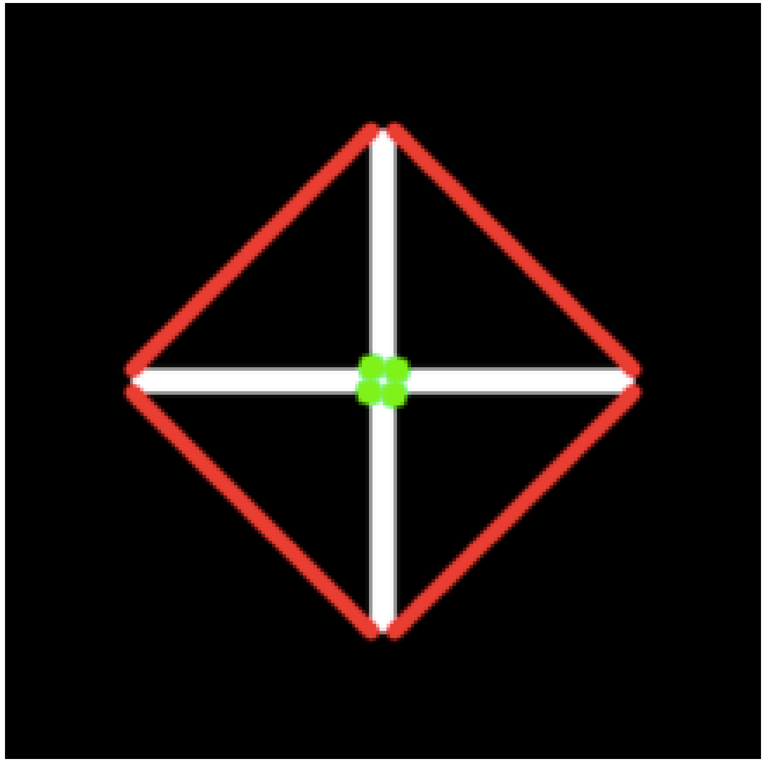
    cv2.convexHull()



HULL BORDERS

Compute convexity defects

cv2.convexityDefects



Checks contours to see if it is convex.

LENGTH AND AREA

Length

- cv2.ContourPerimeter

  Returns length of a contour

You can summarize length and area as a bounding structure

- Boxes
- Circles
- Ellipses

BOUNDING BOXES

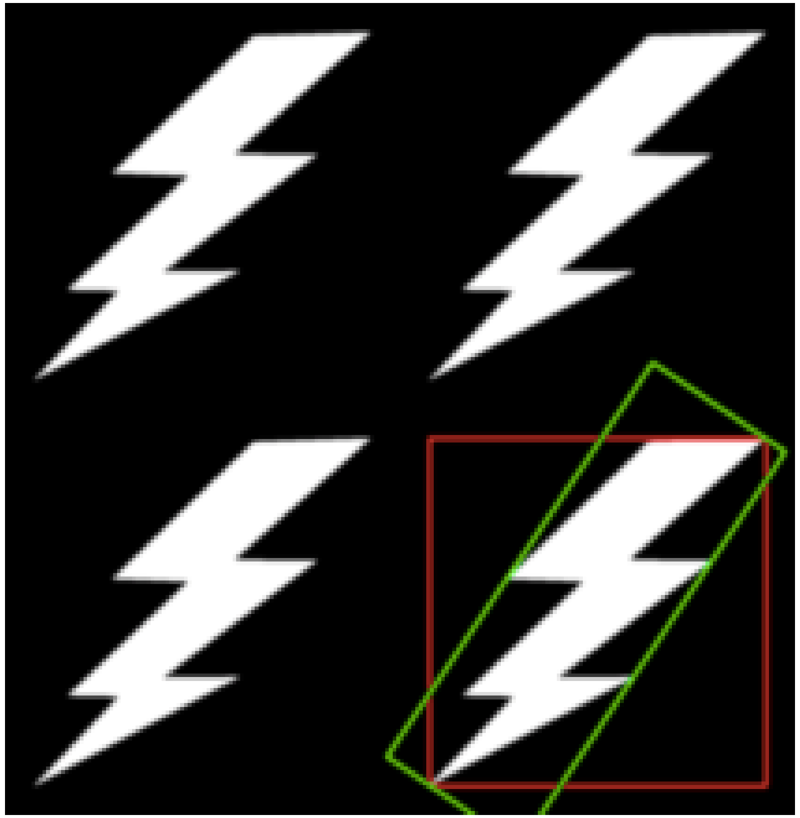
cv2.BoundingRect()

- Returns rectangle that bounds the contour
- Red box in image

cv2.MinAreaRec2()

- Returns smallest rectangular bound
- Green box in image

    Note green box is rotated for best fit



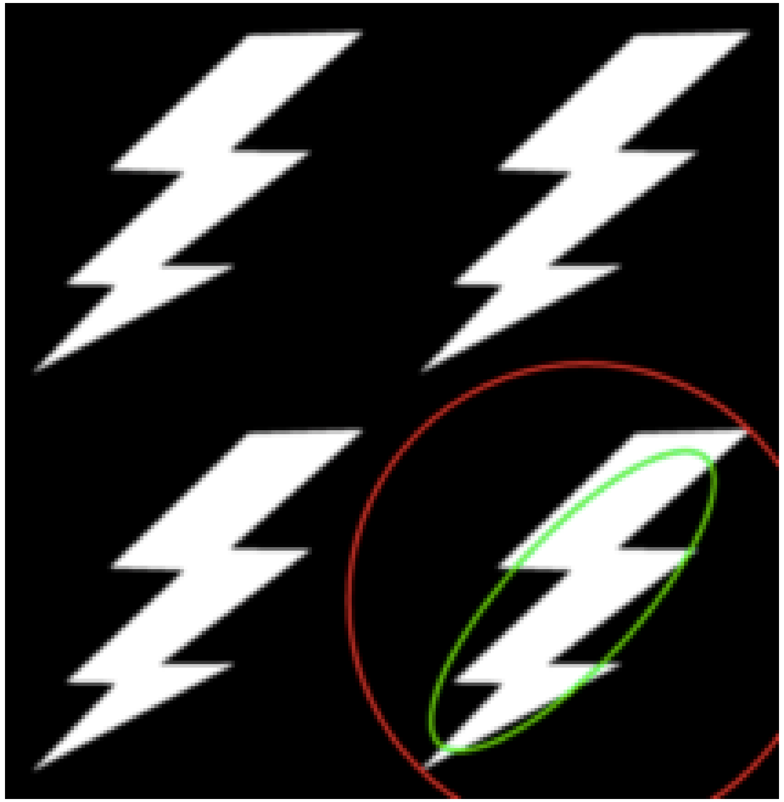
CIRCLES AND ELLIPSES

cv2.MinEnclosingCirlce()

- Similar to cv2.BoundingRect() in that it bounds the contour, but not a best fit
- Needs radius as input
- Red circle in image

cv2.FitEllipse2()

- Allows us to get a best fit bound
- Green circle image



GEOMETRIC TOOLKITS/CHECKS

cv2.maxRect()

- Two input rectangles are used to compute a new rectangle

cv2.BoxPoints()

- Computers corner points of cvBox2D structure

cv2.PointPolygonTest()

- Test: is point in polygon?

CONTOUR STATISTICS

$$x, y, w, h = cv2.boundingRect(cont)$$

$$area = cv2.contourArea(cont)$$

$$hull = cv2.convexHull(cont)$$

$$hull\_area = cv2.contourArea(hull)$$

$$Aspect\ Ration = \frac{Width}{Height}$$

$$Extent = \frac{Object\ Area}{Bounding\ Rectangle\ Area}$$

$$Solidity = \frac{Contour\ Area}{Convex\ Hull\ Area}$$

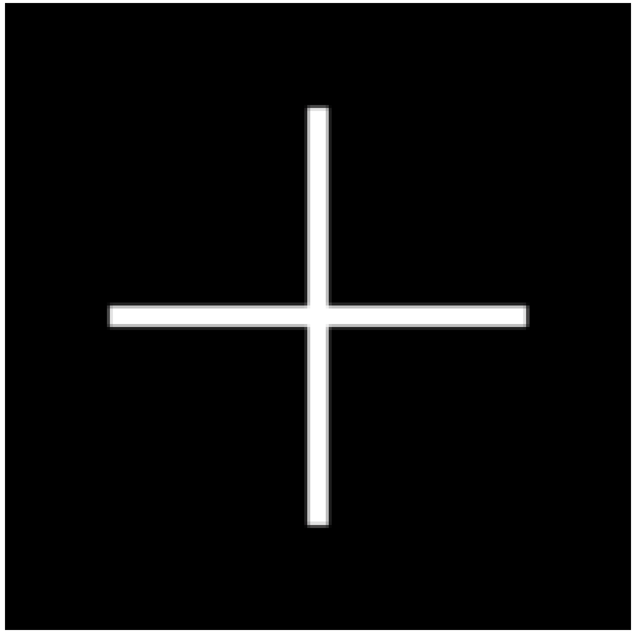
$$Equivalent\ Diameter = \sqrt{\frac{4\ x\ Contour\ Area}{\pi}}$$

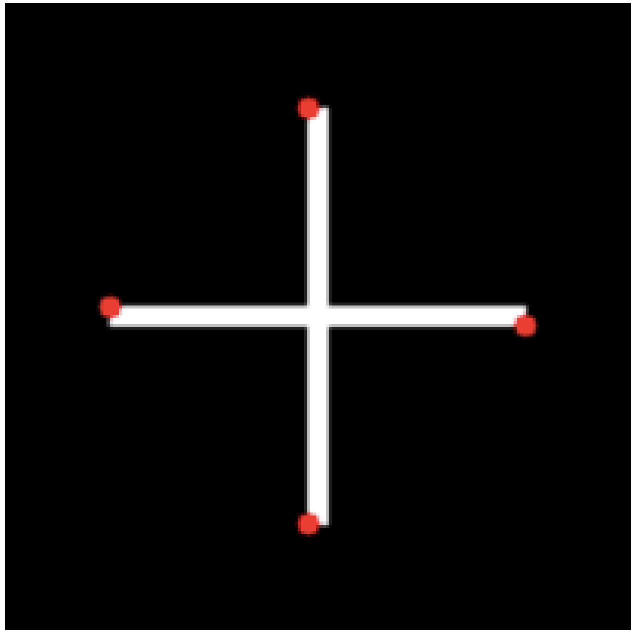
$$Orientation\ Angle = cv2.fitEllipse(cnt)$$

FINDING EXTREME POINTS: MIN/MAX

Min/Max is computed with NumPy.

Extreme points of the cross are min/max left/right and up/down.





SEGMENTATION

FOREGROUND/BACKGROUND SEPARATION

Separate foreground from background

Send foreground on for further analysis

- Cars in security camera
- Skin within an image (reduces complexity of finding faces)

Superpixels

- Groups of pixels in same object/type of object

METHODS: IMAGE PYRAMIDS

Collection of images where each subsequent image is a downsampling of the previous.

Gaussian pyramid

- Used to downsample
- Removes even rows and columns
- Will give you higher resolution

Laplacian pyramid

- Used to upsample
- Works with lower-resolution images and allows for faster computations

GAUSSIAN PYRAMID

Use cvPyrDown() with a 5x5 Guassian kernel

- Start with G_0 (original image)
- Convolve G_0 with Gaussian kernel
- G_i is ¼ size of G_0 because we removed even-numbered rows and columns from G_0
- Repeat for all desired G_{i+1}

LAPLACIAN KERNEL

Use cvPyrUp()

Notice the relationship between Gausian and Laplacian

$$Lap_i = Gausi - UP(Gaus_{i+1}) \otimes \mathrm{Gaus}_{5x5}$$

In OpenCV we see this as:

$$Lap_i = Gaus_i - PyrUp(Gaus_{i+1})$$

Remember, we saw this in Week 3!

METHODS TO SEGMENT AN IMAGE

cvPyrSegmentation ()

- Generates an image pyramid
- Associates pyramid layers to parent-child relationships
- Map pixels between G_{i+1} and G_i
- Perform segmentation on lower-resolution images
- Note: Start image must be divisible by 2 for each layer of the pyramid

MEAN SHIFT CLUSTERING OVER COLOR

pyrMeanShiftFiltering()

Peak of color-spatial distribution over space

Data dimensions are:

- Spatial (x, y)
- Color (blue, green, red)

Parameters are spatialRadius, colorRadius, max_level, and CvTermCriteria.

WATERSHED

Used to separate objects when you do not have a separate background image.

Input is a grayscale image or a smoothed color image.

Can cause oversegmentation or undersegmentation

- If you make too many basins or too few

Lines converted to peaks; uniform regions to catchment basins.

WATERSHED METHOD

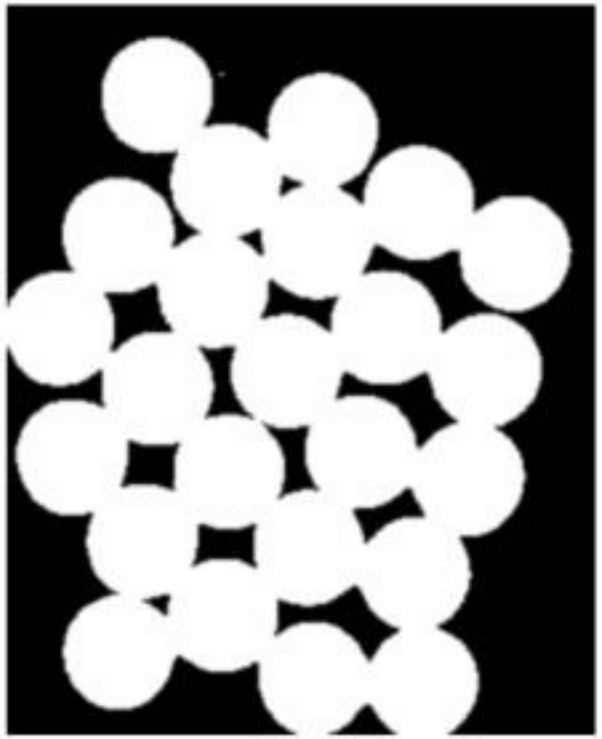
Take the gradient of intensity image.

Marker points, or known areas of region of interest, are defined by user.

Basins get connected to marker points to create segments.

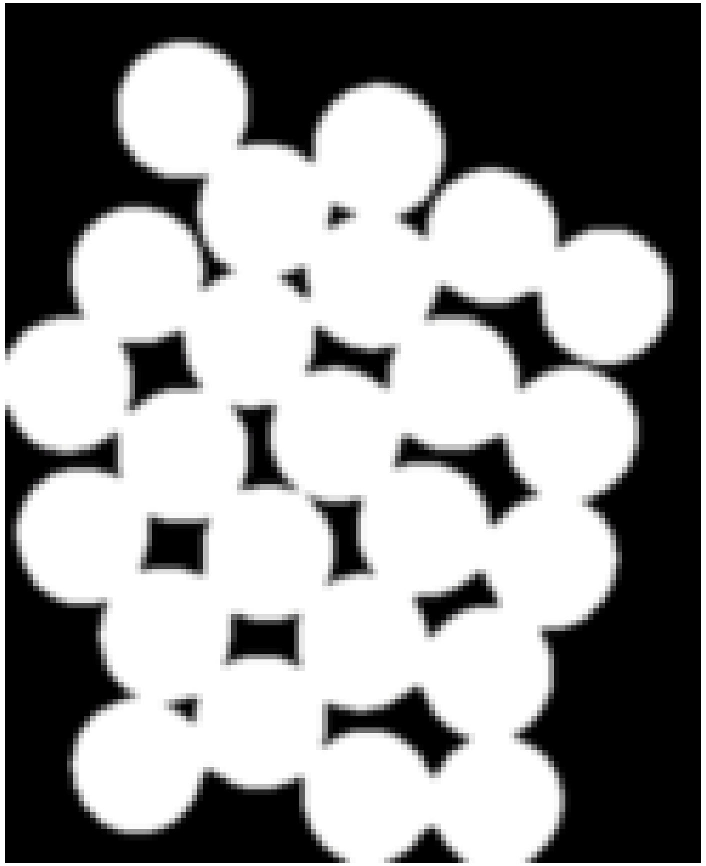
WATERSHED: EXAMPLE

Step 1: Threshold a color image



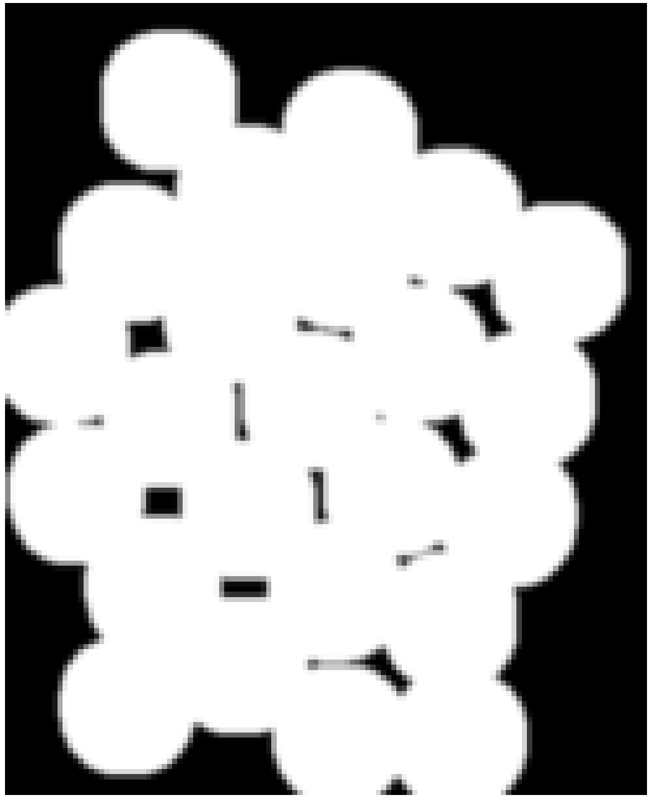
WATERSHED: EXAMPLE

Step 2: Remove noise with cv2.morphologyEx



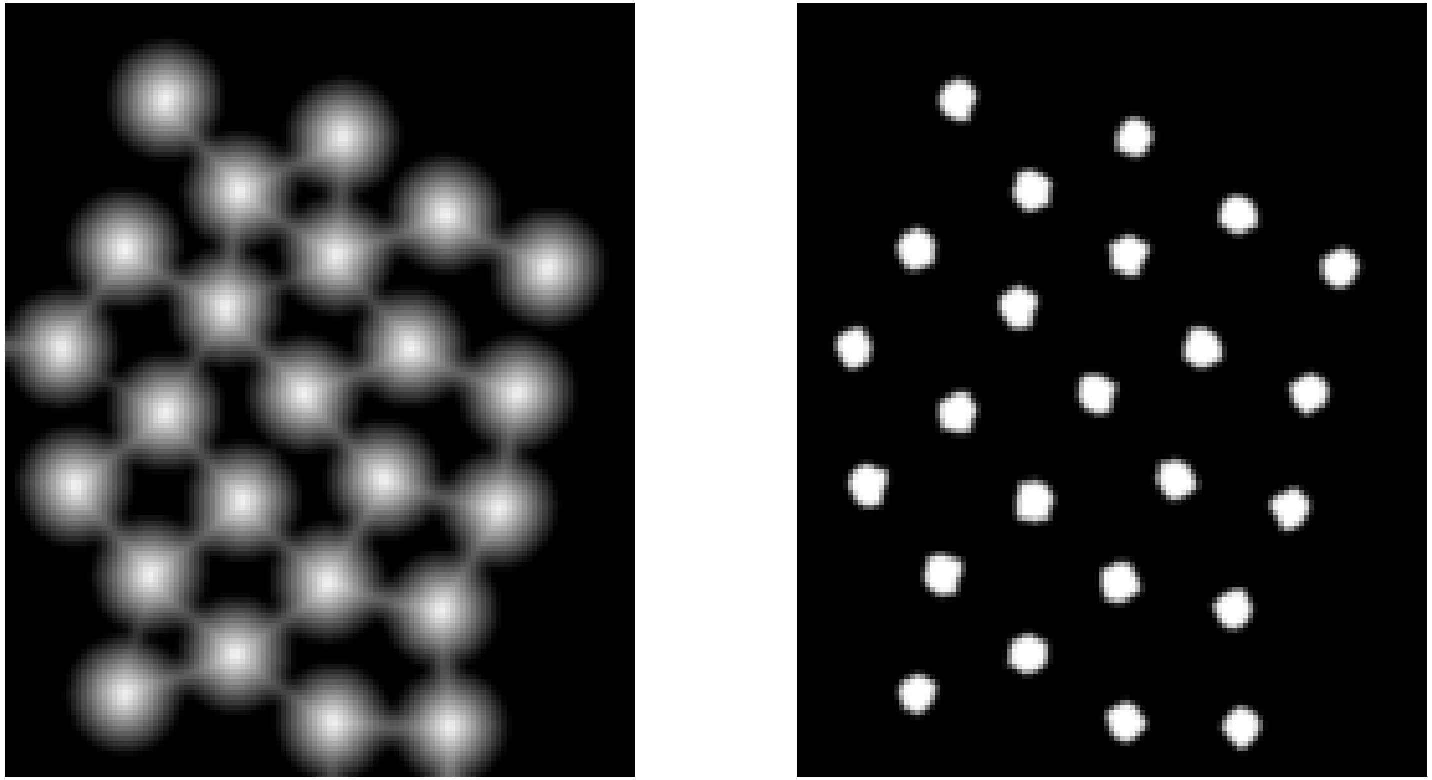
WATERSHED: EXAMPLE

Step 3: Sure background area with cv2.dilate()



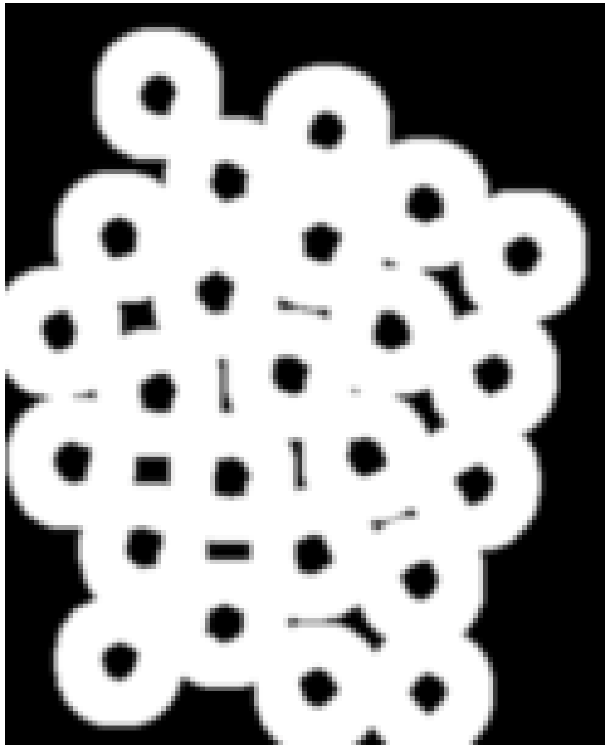
WATERSHED: EXAMPLE

Step 4: Find sure foreground area with cv2.distanceTransform() and cv2.threshold()



WATERSHED: EXAMPLE

Step 5: Find unknown region with cv2.subtract()



GRABCUT: GRAPH CUTTING TECHNIQUES

Pixel-based energy function

Basic binary segmentation algorithm to identify object

Iteratively recalculates the region statistics (Gaussians) to perform segmentation

GRABCUT PROCEDURE

(1) Assign background and unknown

Unknown becomes potential foreground

(2) Create five probabilistic clusters (each) of foreground and background

(3) Assign every background pixel to a cluster of background

Do the same for foreground

(4) Compute cluster statistics for all clusters

GRABCUT PROCEDURE (CONTINUED)

(5) Create a graph where:

Pixels are linked to their neighbors and to two special nodes

- Nodes represent foreground and background

The weights of the links are related to neighbor similarity and class probability

GRABCUT PROCEDURE (CONTINUED)

(6) Apply a classic computer science graph theory algorithm *min-cut* to the graph

The net result cuts the connections from each pixel to one of the two special nodes.

In turn, this gives us a new assignment for foreground/background to the pixels.

(7) Until convergence, we loop back to Step 4 and repeat

Classifications remain similar enough from one round to the next.

MATCHING

CONTOUR MOMENTS: MATCHING CONTOURS

Compare statistical *moments* of contours:

- In statistics, first moment is the mean, second moment is the variance, and so on.

Moments are used to compare two contours (outlines) for similarity.

Another technique: Compute rough contour characteristic computed by summation of all pixels over contour boundary:

$$m_{p,q} = \sum_{i=1}^{n} I(x,y)x^p y^q$$

p: x-order q: y-order

Normalized moments are better for comparing similar shapes of different sizes.

STATISTICAL MOMENTS

In classical statistics, we use mean, variance, skew, and kurtosis.

cvMoments()

- cvGetCentralMoment()
  - invariant with respect to translation
- cvGetNormalizedMoment()
  - invariant with respect to scale
- cvGetHuMoments()
  - invariant with respect to rotation

NORMALIZED MOMENTS

Central moments normalized with respect to a function intensity.

First need to calculate *central moment*:

$$\mu_{p,q} = \sum_{x,y} I(x,y)(x - x_{avg})^p (y - y_{avg})^q$$

Then you can calculate *normalized moments*:

$$\eta_{p,q} = \frac{\mu_{p,q}}{m_{00}^{(p+q)/2+1}}$$

HU INVARIANT MOMENTS

Linear combinations of centralized moments.

Use central moments to get invariant functions

- Invariant to scale, rotation, and reflection

cvGetHuMoments()

$$h_1 = \eta_{20} + \eta_{02}$$
$$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$h_6 = (\eta_{20} - \eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

REGION MATCHING

TEMPLATE MATCHING OVERVIEW

Use cvMatchTemplate() to find one image within another.

Uses the actual image of matching object instead of a histogram

- This is called an image patch

Input image

- 8-bit, floating point plane, or color image

Output

- Single-channel byte or floating point image

TEMPLATE MATCHING METHODS

Squared difference

CV_TM_SQDIFF_NORMED

$$R_{sqdiff}(x,y) = \frac{R_{sqdiff}(x,y)}{Z(x,y)}$$

Correlation

CV_TM_CCORR_NORMED

$$R_{ccorr}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)}$$

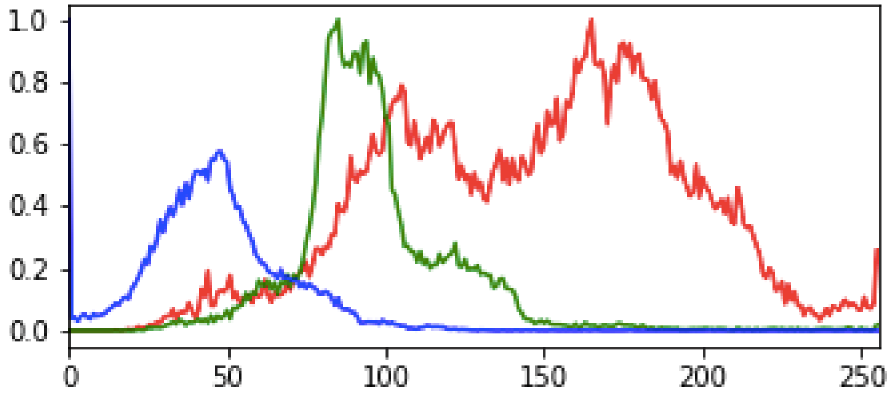
Correlation coefficient

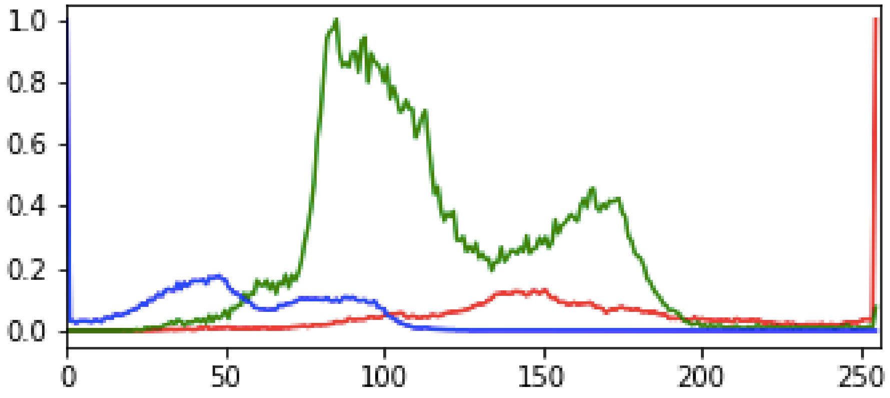
CV_TM_CCOEFF_NORMED

$$R_{ccoeff}(x,y) = \frac{R_{ccoeff}(x,y)}{Z(x,y)}$$

HISTOGRAM BACKPROJECTION

Uses a histogram model to match pixels (or patches of pixels) to approach object recognition.





HISTOGRAM BACKPROJECTION

Better thought of as segmentation to foreground ROIs used for further analysis.

If you have a histogram of the ROI, we can use that to find the object in another image.

Use histogram from one image to match to histogram profile from another image

- Normalize to pixel number
- Find matching object, or ROI

cvCalcBackProjectPatch()

HISTOGRAM BACKPROJECTION

Given a color, we want to know the probability of foreground/background.

Approximate probability that a particular color belongs to a particular object

$$p(object|color) = \frac{p(color|object) * p(object)}{p(color)} \sim p(color|object)$$

You can also make a ratio histogram where you de-emphasize colors not in the ROI.

BACKPROJECTION METHOD

1. Calculate ROI histogram
2. Normalize histogram
3. Apply cv2.calcBackProject
4. Convolve with circular disc
5. Duplicate threshold