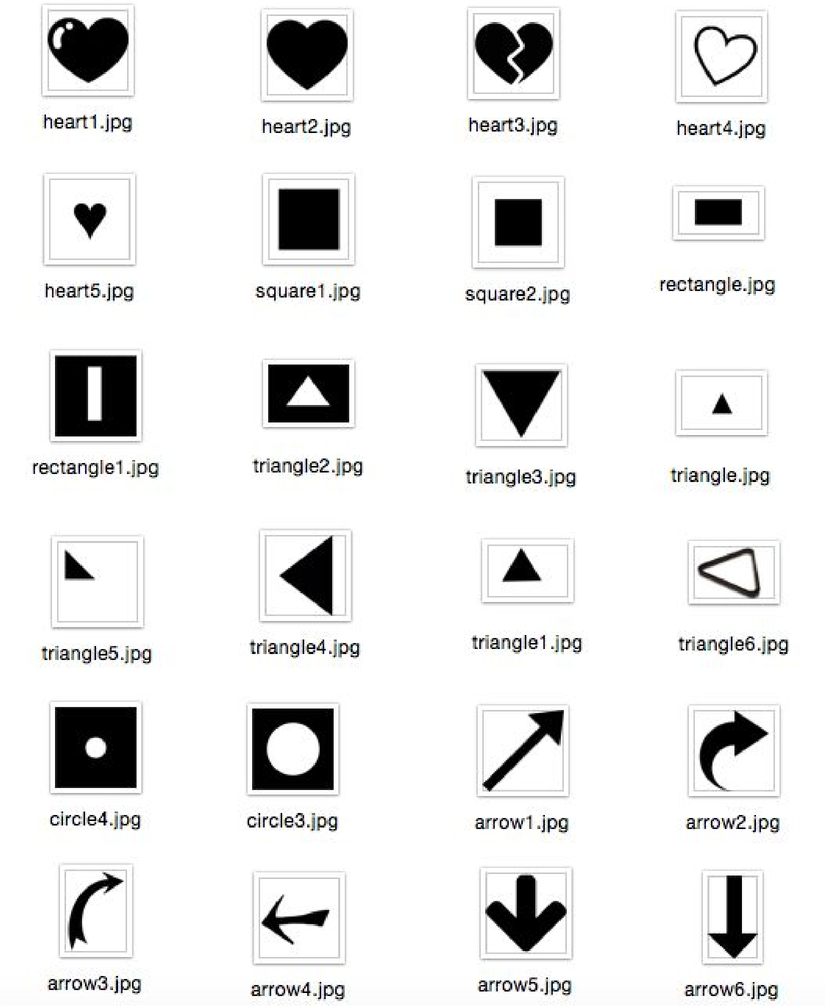
Image processing and categorization

Xinyi Li & Yian Mo & Zhihe Shen

Dataset

- **Step 1**: Google basic shapes
- **Step 2:** Convert images to 0-1 matrices in Matlab
- **Step 3:** Use matrices as test data in python file



Matlab code to convert images

```
clear all
image = imread('/Users/xinyi_li/Desktop/images/circle.jpg');
newimg = im2bw(image);          // convert matrix to binary
newmatrix = 1 - newimg;         // exchange 0 & 1’s in matrix
text = fopen('circle.txt','wt');
for i = 1:size(newmatrix,1)     //write new matrix to text file
fprintf(text,’%d’,newmatrix(i,:));
fprintf(text,'\n');
end
fclose(text);
```

Basic steps

- **Preprocessing and Standardizing**
  - Convert background
  - De-noise
  - Rotate
  - Border
  - Scale
  - Center

- **Categorize**
  - Symmetry: Determine if the image is symmetric along vertical and horizontal midlines
  - Convex: Determine if the image is convex or non-convex
  - Area: Divide images into >50% or <50% of area of the frame

- **Calculate distance between 2 matrices and return the closest k matrices**

Covert background

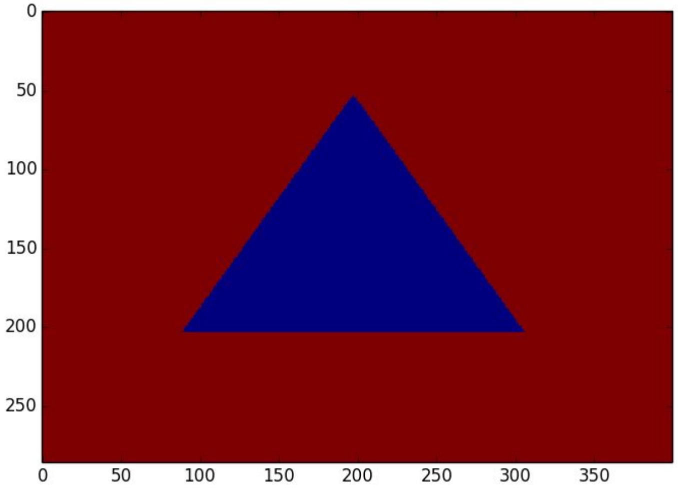
- We want all the shapes filled in black(1) and use white(0) as background

- Change images with black background into white background

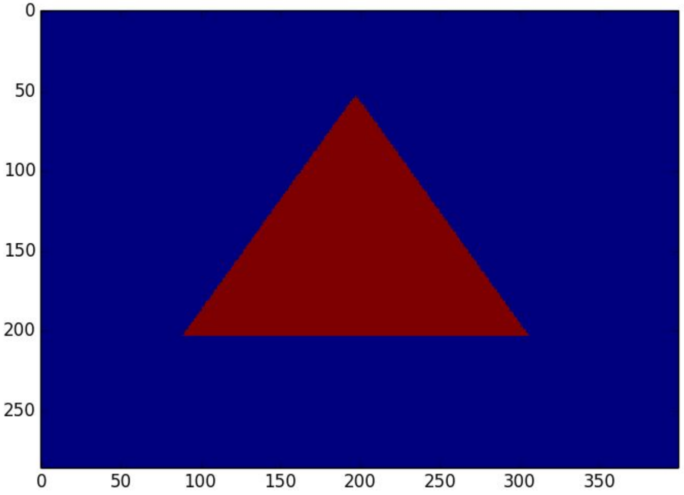
- How to detect an image's background color?
  - Scan all the boundary pixels of the frame, put them into a list
  - If >80% of pixels in the list are 1, we determine the image has black background
  - Then we loop through entire image, change all 0's to 1's , 1's to 0's

Convert Background

Black background
white shape



White background
black shape



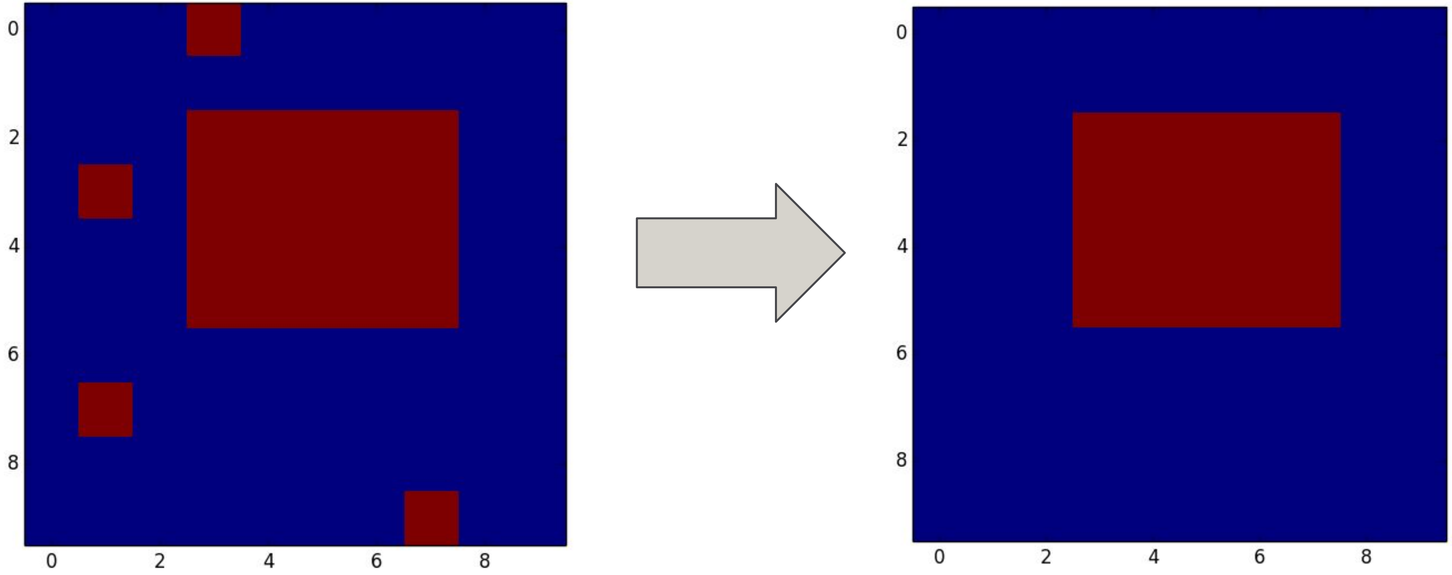
De-noise

- Change “noises” black pixels to white pixels

- What standard do we use?
  - For each black pixel: check the values of up, down, left, right four neighbor pixels
  - If all four neighbors equal to 0, we determine the black pixel to be “noise”
  - Change the pixel to 0

- Why not look at all eight neighbors? (Consider diagonal neighbors)
  - If we use a 3*3 square to loop thru matrix, we may encounter indivisible row length and column lengths
  - Will increase run time

De-noise

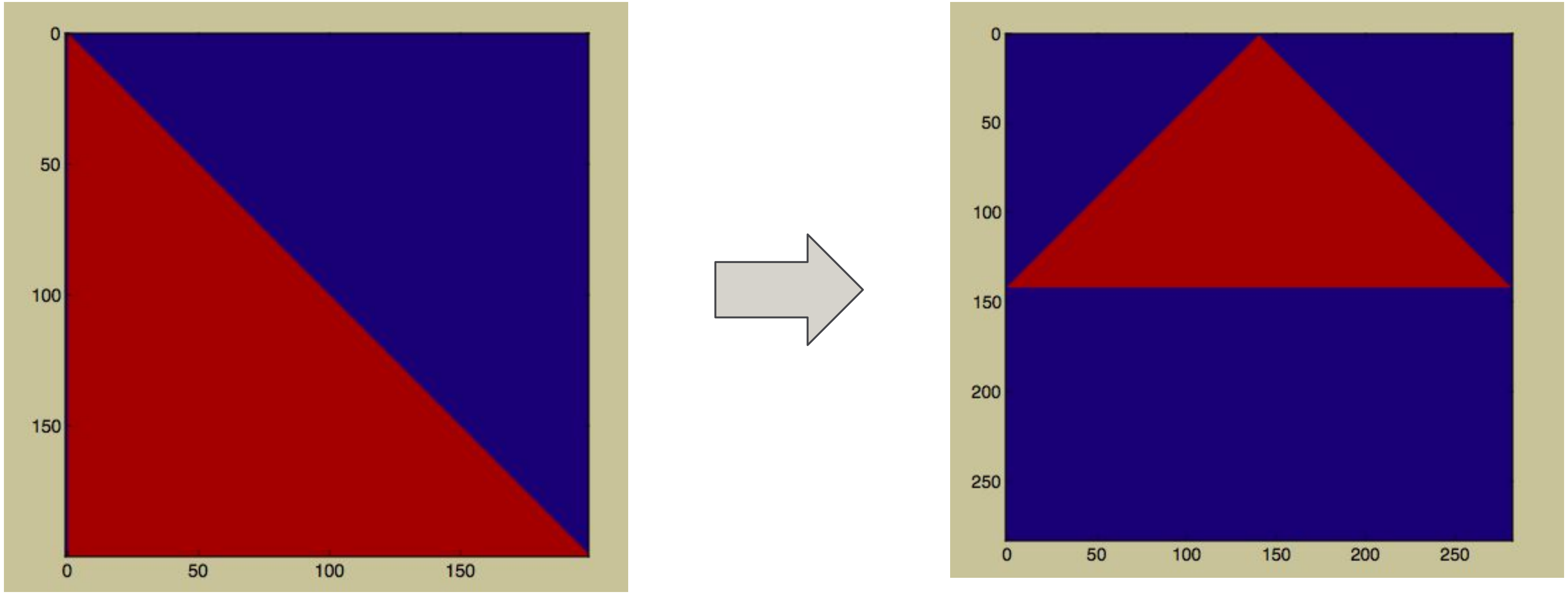


Rotate

- Library: **scipy.ndimage.interpolation.rotate(*input*, *angle*)**

- How to find the degree of rotation? / How to find the line best fit the image?
  - Tried: Linear regression… (not linear, not precise)
  - Final solution: The line connecting the top leftmost and the bottom rightmost pixels
  - Get the slope and calculate arctan of the slope, which is the degree between the line and the x-axis and then get the degree we want to rotate

Rotate



Border

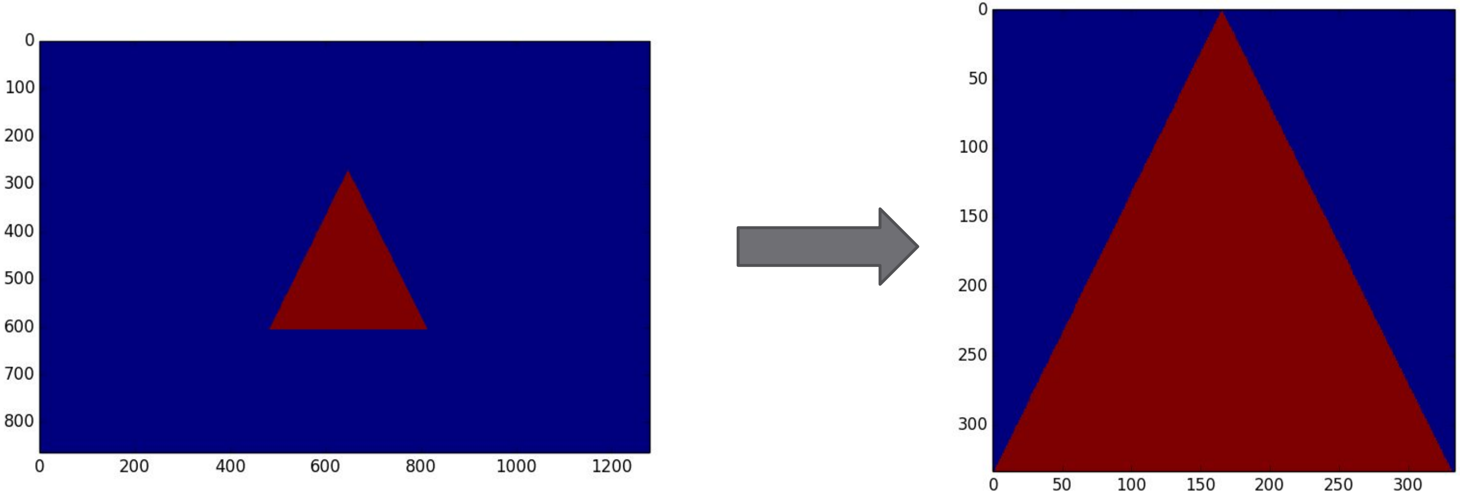
- Detect the borders of the shape

- Cut the shape into a smaller square matrix that fits in the entire shape

- Use the longest edge as the length of the sides

- How do we detect the edges?
  - Scan each row of matrix, document the location of first and last black pixel
  - Scan each column of matrix, document the location of first and last black pixel

Border

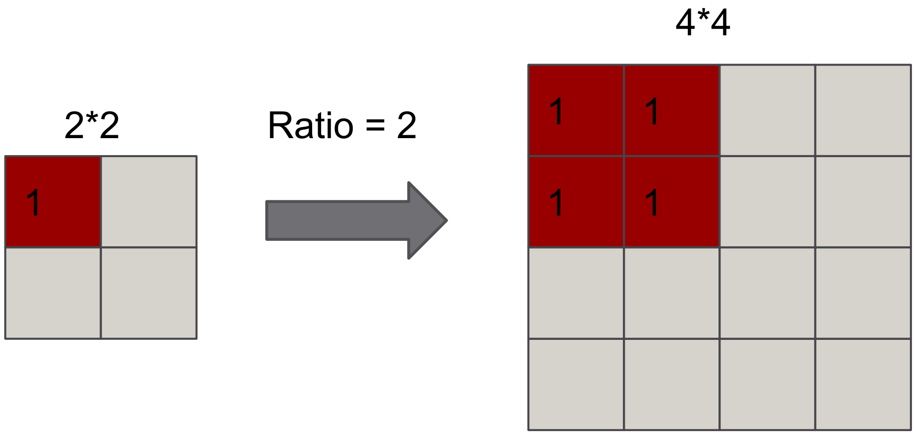

0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
0
50
100
150
200
250
300

Scale

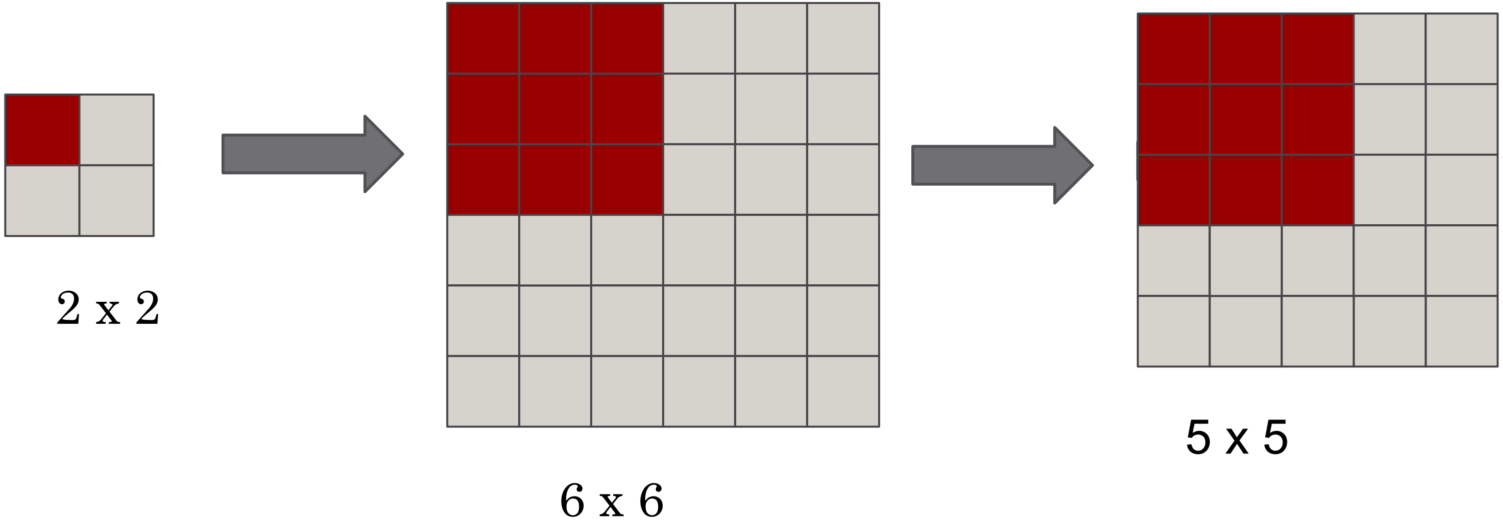
- We want to convert each image into 50*50 for comparison
    - Enlarge
    - Shrink

Scale - enlarge

- Calculate ratio = largelength/smalllength
- Append pixels at index [i/ratio][j/ratio] from original matrix to the new matrix
- 1 x 1 pixel in the original image => ratio x ratio pixel in the after image
    - Same value



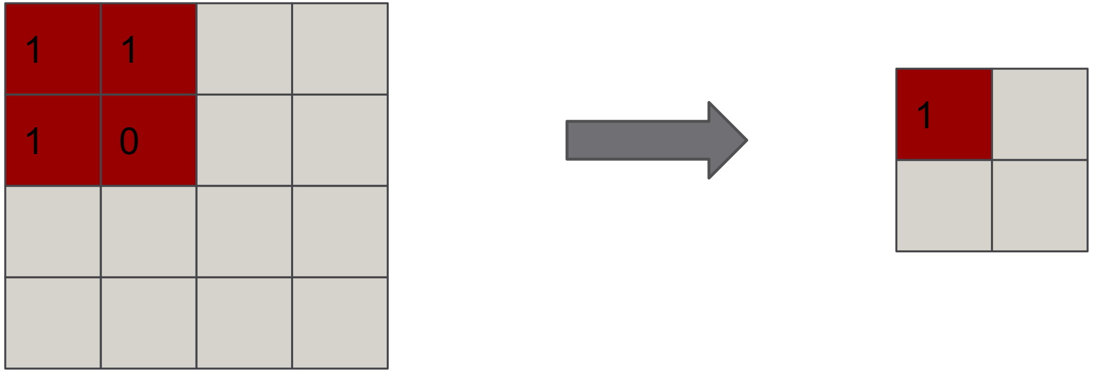
What if we don't have perfect ratio(not divisible)?



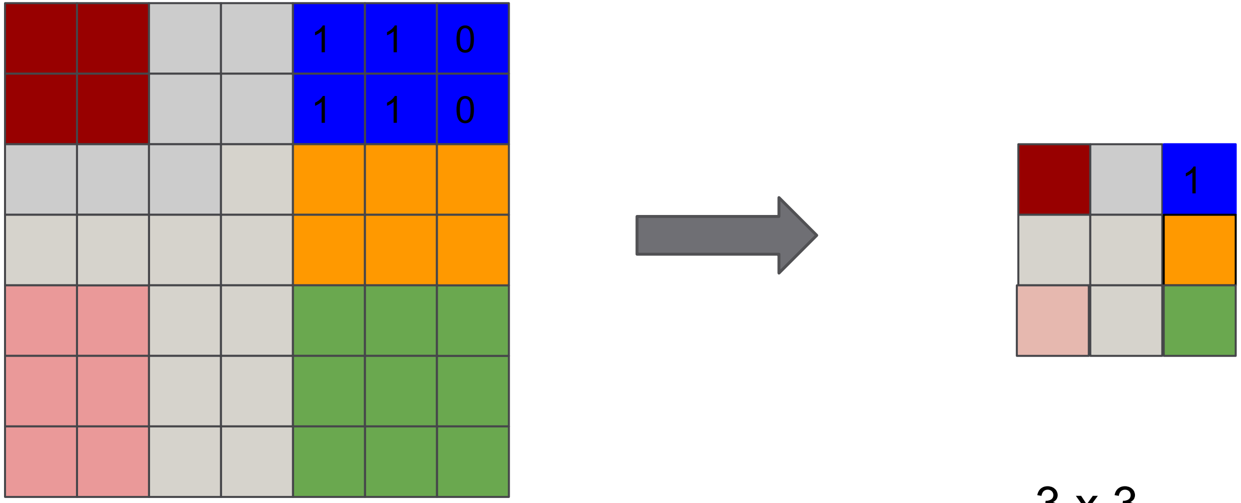
- When larger length(5) can't be divided by the smaller length(2),
- we have ratio = largerlength/smalllength + 1, e.g ratio = 5/2+1 = 3
- We then enlarge the small matrix to a matrix of smalllength*ratio e.g 6*6
- Then we cut a 5*5 matrix from the 6*6 matrix

Scale - shrink

- Find the ratio = largelength / smalllength
- Each ratio x ratio matrix in the original image => 1 x 1 matrix in the after image
  - Find majority in the 2 x 2 matrix, 0 or 1



What if we don’t have perfect ratio(not divisible)?



7 x 7

3 x 3

Ratio = 7/3 = 2

Center

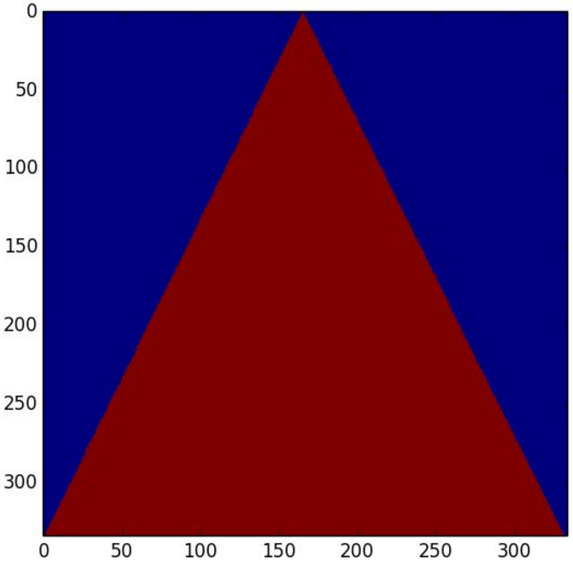
- After rescaling the image to 50*50, shape may not be in the center of the frame

- We use center function to detect the difference between the frame center and shape center, then move the shape to the center of frame

- How to do the moving?
  - Calculate location of center of the shape by : Sum of row(col) indexes of all black pixels / # of black pixels
  - Calculate index difference between center of shape and center of matrix
  - Create a new 50*50 matrix with all 0's inside
  - For all pixels at index [m][n] in original matrix that are black, change pixels at index [m+rowdiff][n+coldiff] in new matrix to 1

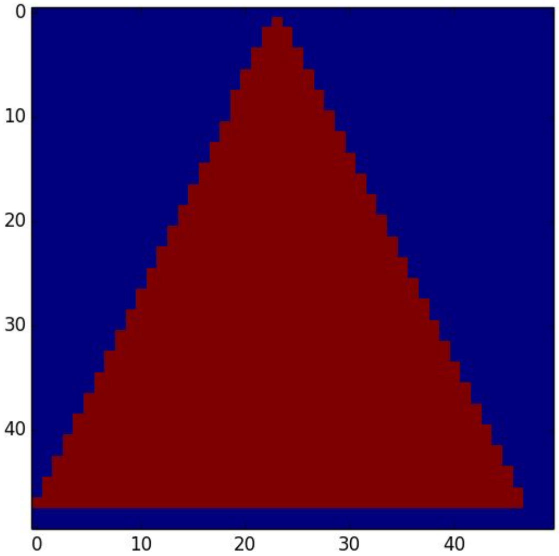
Scale - shrink

350*350



Shrink

50*50



Center

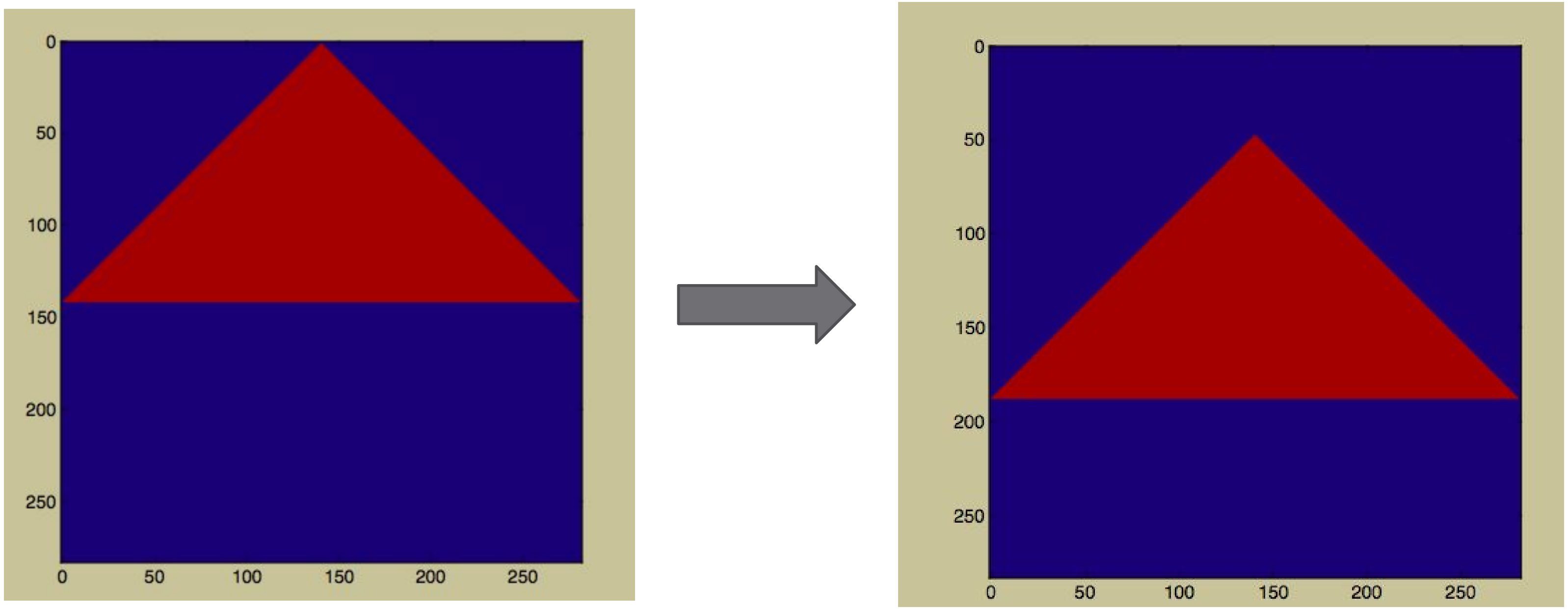
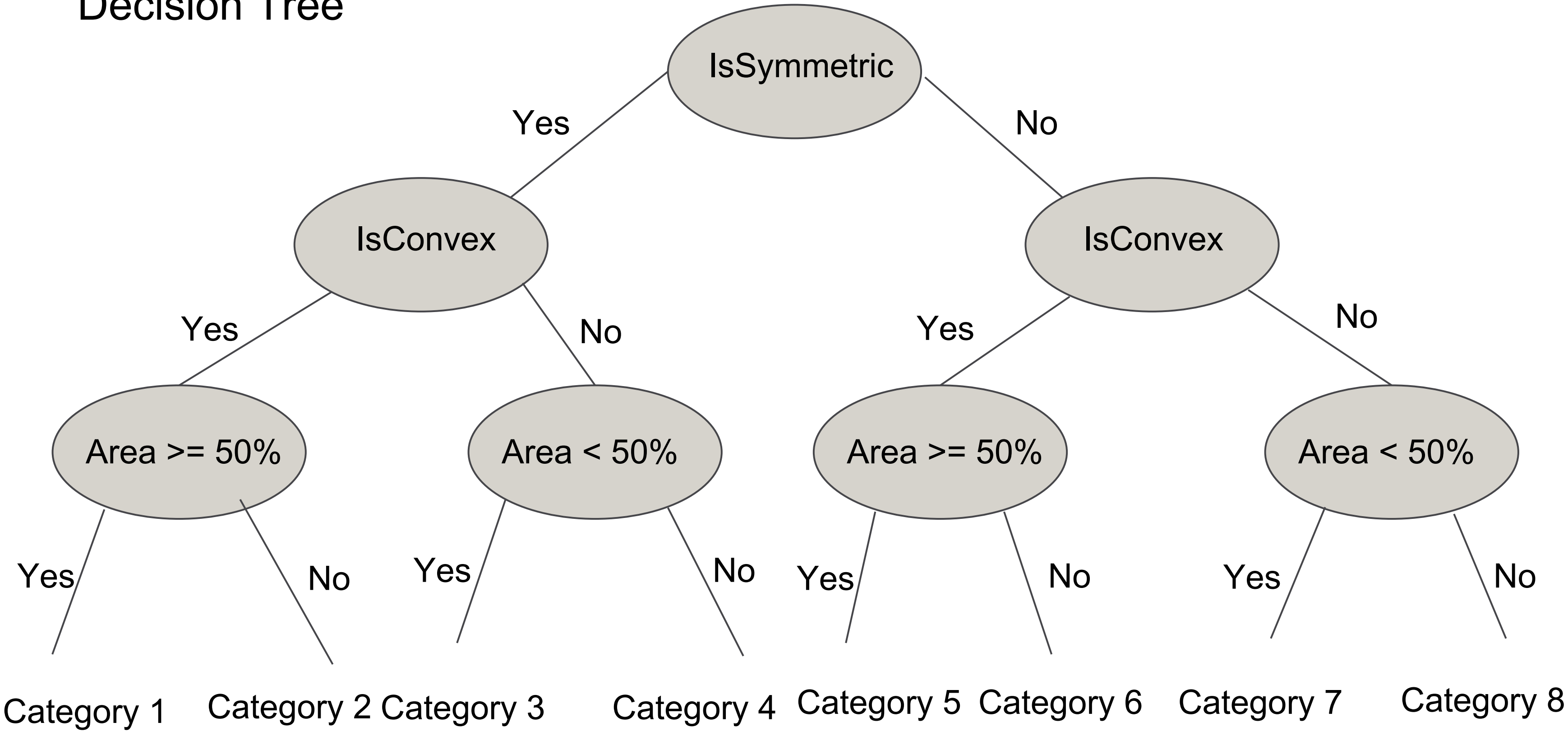

0
50
100
150
200
250
0
50
100
150
200
250
0
50
100
150
200
250
0
50
100
150
200
250

Image Categorization

- IsSymmetric
- IsConvex
- Area

The above three functions will give us 8 categories in total to minimize the search for closest matrices later.

Decision Tree



IsSymmetric

Yes

No

IsConvex

IsConvex

Yes

No

Yes

No

Area >= 50%

Area < 50%

Area >= 50%

Area < 50%

Yes

No

Yes

No

Yes

No

Yes

No

Category 1

Category 2

Category 3

Category 4

Category 5

Category 6

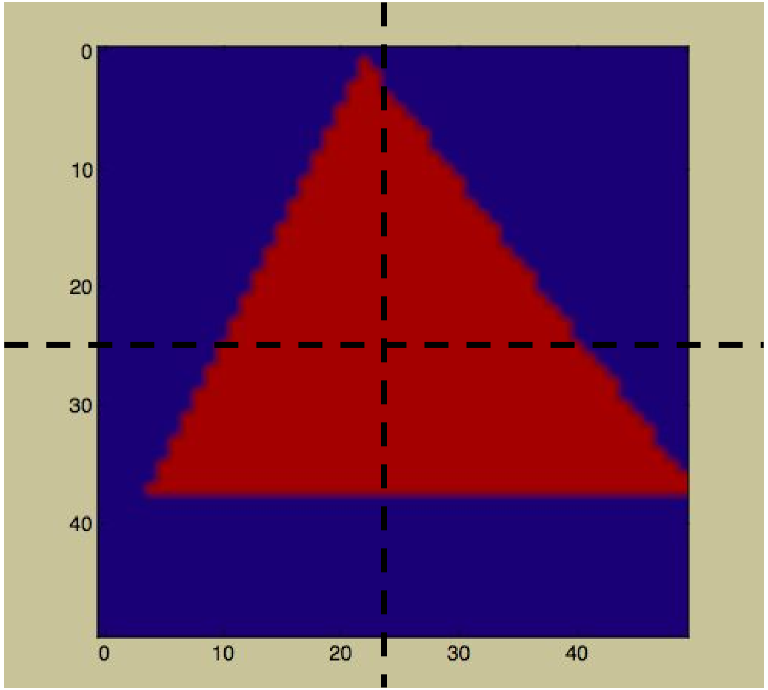
Category 7

Category 8

Symmetry

- Determine if the matrix is symmetric along horizontal and vertical midlines
- Separate matrix to top and bottom, left and right halves along midlines
- Compare pixels at the same location in top and bottom (left and right) parts, if they are the same value, count +1
- Loop through pixels, then calculate correct percentage = count / # of pixels in half matrix
- If percentage >=80% along either horizontal or vertical midline, we determine the shape is symmetric.

Symmetry

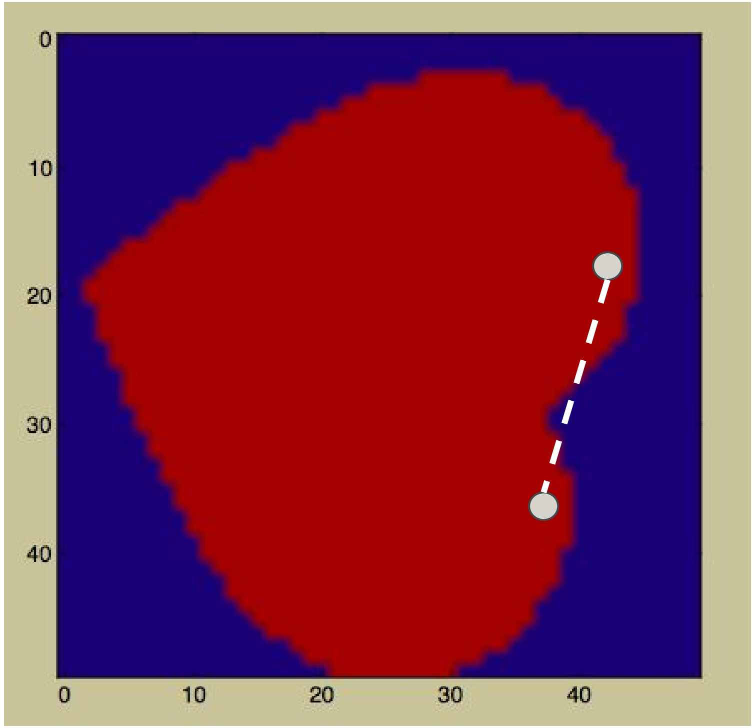


For example, this triangle is neither symmetric along horizontal nor vertical midlines in our definition.

Convexity

- Determine if the image is convex or not
- Original idea: Check the midpoints of pairs of vertices of the shape to see if they are inside the shape (loop through edges)
- Final idea: randomly pick pairs of pixels with value 1 and iterate through the connecting lines. Check points on the connecting line to see if they are inside the shape
- Corner case: nonzero denominator when calculating the slope; iterate x value from small to large

Convexity



For example, there are points on the connecting line between the two points that are not in the shape, so this image is not convex.

Area

- Determine if the area of the shape is >=50% or < 50% of the area of the matrix

- Loop through matrix, return # of black pixels

- Calculate percentage = # of black pixels / total # of pixels

Image Comparison

- ➢ Distance function: calculate the squared error pixel by pixel
- ➢ Tried Nearest Neighbor Algorithm (KD-tree)
- ➢ Final idea: use sort to get the closest k

Run Time Complexity

- Loop folder 1, preprocess and categorize each matrix
- Loop folder 2, for each image, preprocess and categorize it and search through same category to find the closest k matrices

- Preprocess: O(N^2)
  - Convertbackground: O(N^2)
  - De-noise: O(N^2)
  - Rotate: O(N^2)
  - Border: O(N^2)
  - Scale: O(N^2) for enlarge or O(N^4) for shrink
  - Center: O(N^2)
- Category: O(N^2)
- Run: O(N^4)

Thoughts on improvement

- Reducing runtime
  - Combine several steps of preprocessing when we loop through rows & cols
  - Using KD-tree (O(logn)) vs. sorted (O(nlogn)) for image comparison

- Be more considerate about preprocessing
  - De-noise: consider diagonal neighbors to increase accuracy
  - Fill in shapes that are empty in the middle