Segmenting and Analyzing Concrete Crack Images

Segment the Crack Images

There is no right answer for how to best segment and analyze the images in the concrete crack dataset. Your approach will also depend on your goals. For example, accurately segmenting both large and small cracks is challenging. Choose one of the goals listed below and try segmenting the cracks with that goal in mind. Recall the approaches to segmenting cracks in Course 2 of this specialization that you can use as a starting point.

Use the Image Batch Processor App to check your results and refine your approach.

Choose a goal:

1. Segment small and large cracks accurately to determine their area
2. Identify images with large cracks that should be prioritized for repair.

Our Segmentation and Analysis function

Stop! Don't read until you try segmenting the cracks.

Below you can see a detailed description of a function we used for goal 1. Later in this course you'll learn more about analyzing the crack images.

...

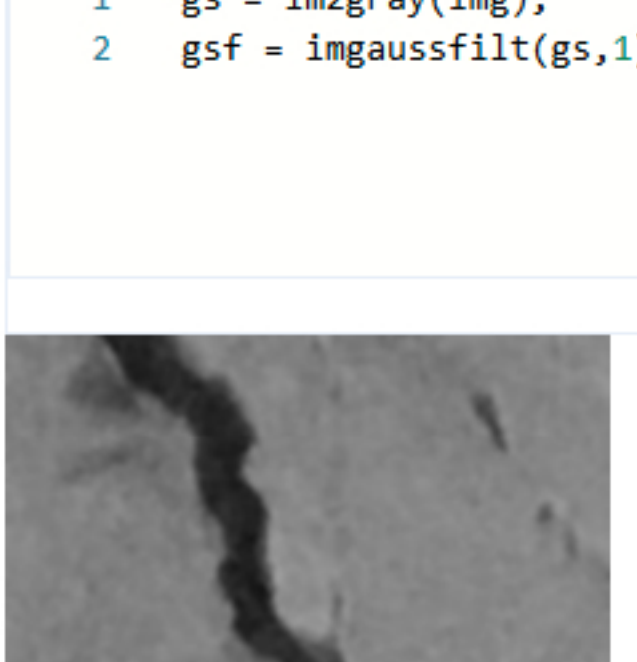
The following example function was our approach to goal 1 above. It is made to be compatible with the Image Batch Processor App.

```
1   function results = crackAnalysis(img)
2   %crackAnalysis segments and analyzes an image from the concrete crack dataset.
3   %
4   %  results = crackAnalysis(img) preprocesses, segments, and analyzes image img.
5   %  results, is returned as a structure array with 4 fields:
6   %  BW, the final black and white mask.
7   %  NumRegions, the number of individual regions in the mask.
8   %  Area, the total area in number of pixels of all regions.
9   %  MaxWidth, the maximum width in number of pixels of all regions.
10  %--------------------------------------------------
11  
12  %Preprocessing
13  %  Convert the input image to grayscale and apply a Gaussian filter to
14  %  smooth rough concrete textures.
15  gs = im2gray(img);
16  gsf = imgaussfilt(gs,1);
17  
18  %Segmentation
19  %  Apply multi-level thresholding to distinguish dark cracks from
20  %  textured concrete. Then morphologically close the mask,
21  %  and finally filter out all regions below a cutoff size.
22  thresh = multithresh(gsf,2);
23  levels = imquantize(gsf,thresh);
24  bw = levels>1;
25  bw = ~bw;
26  bw = imclose(bw,strel("disk",4,0));
27  bw = bwpropfilt(bw,"Area",[50,inf]);
28  
29  %Region Analysis
30  %  Calculate the area of all regions in the mask, then
31  %  compute the distance from each foreground "crack" pixels to the
32  %  nearest background pixel.
33  rp = regionprops("table",bw,"Area");
34  distToEdge = bwdist(~bw);
35  maxWidth = 2 * max(distToEdge,[],"all");
36  
37  %Results
38  %  Append the results to the returned structure array variable.
39  results.BW = bw;
40  results.NumRegions = height(rp);
```

The remainder of this reading explains how the function works.

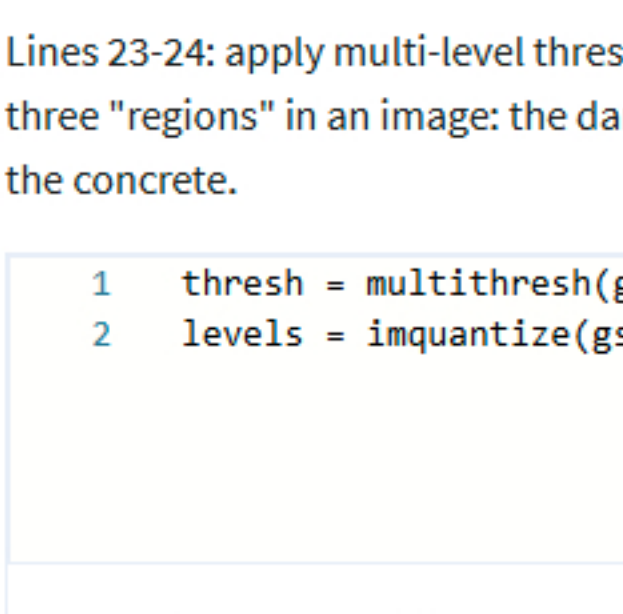
Preprocessing steps

Line 1: the initial input image, `img`.



Lines 16-17: convert the image to grayscale and apply a Gaussian filter.

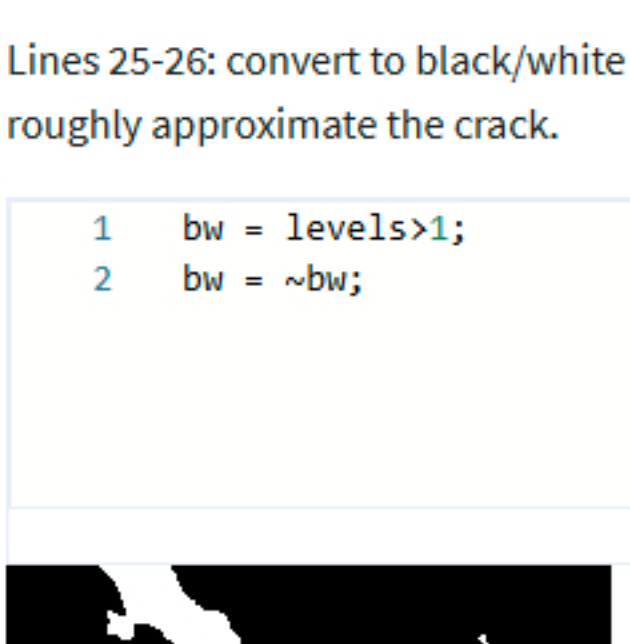
```
gs = im2gray(img);
gsf = imgaussfilt(gs,1);
```



Segmentation steps

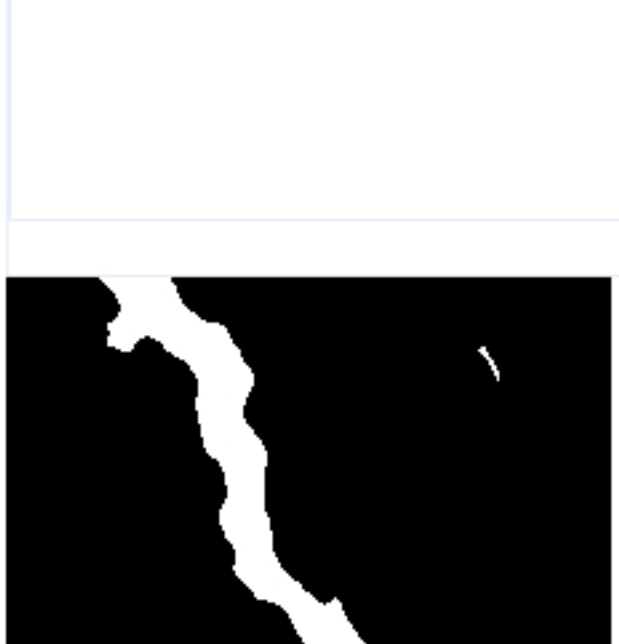
Lines 23-24: apply multi-level thresholding. This method works well for this dataset because there typically are three "regions" in an image: the dark-intensity crack, the light-intensity concrete, and medium-intensity texture of the concrete.

```
thresh = multithresh(gsf,2);
levels = imquantize(gsf,thresh);
```



Lines 25-26: convert to black/white image and invert mask. Here we keep only the 3rd (darkest) level, which should roughly approximate the crack.

```
bw = levels>1;
bw = ~bw;
```

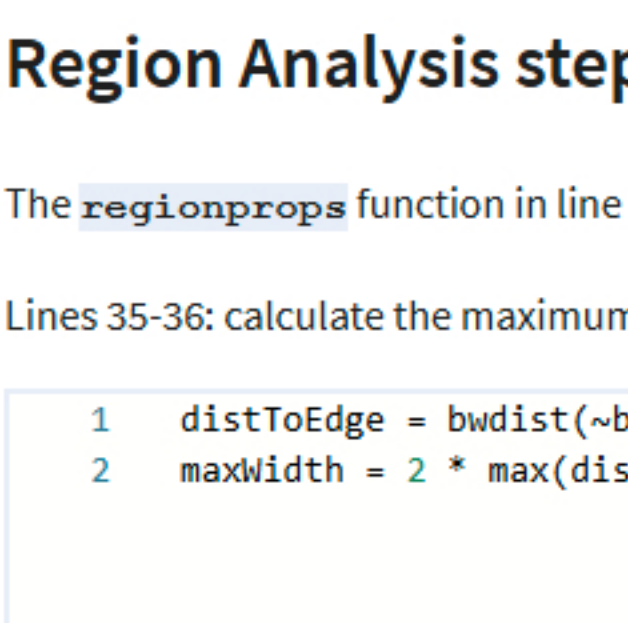


Line 27: morphologically close the mask with a disk of radius 4.

```
bw = imclose(bw,strel("disk",4,0));
```

Line 28: remove regions smaller than a cutoff area.

```
bw = bwpropfilt(bw,"Area",[50,inf]);
```



This is the final black/white mask exported to `results` in line 40. It is also used for the following region analysis.

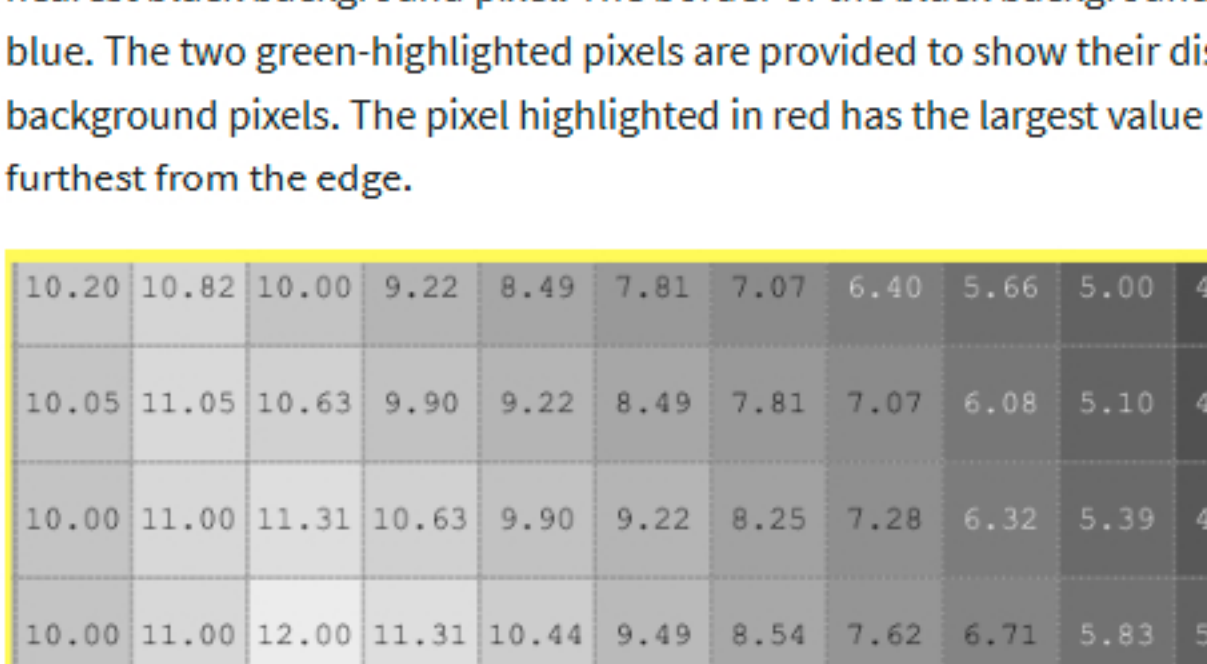
Region Analysis steps

The `regionprops` function in line 34 finds the number of regions and area of every region in the above mask.

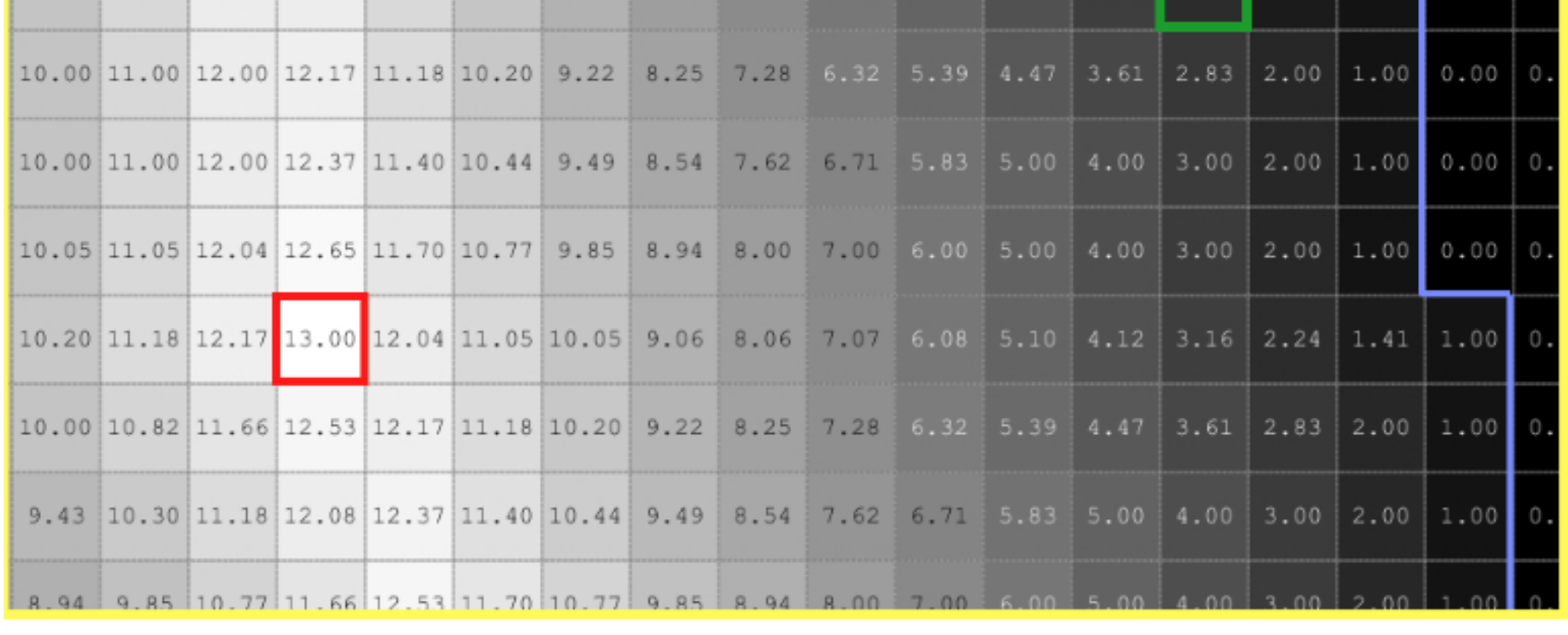
Lines 35-36: calculate the maximum width of the crack.

```
distToEdge = bwdist(~bw);
maxWidth = 2 * max(distToEdge,[],"all");
```

The function `bwdist` takes the inverted mask as input and calculates the distance of every foreground white pixel to the nearest background black pixel (i.e. the edge of the region or crack). This result is visualized in grayscale in the left image below. The distance to the nearest background pixel is represented in this image by pixel intensity, with pixels furthest from the background appearing brighter and pixels neighboring the background appearing darker. Background pixels (which have a distance of 0) remain black. For comparison the lower right image is the final mask that is also shown above.



The image below shows a zoomed in version of on the highlighted region above. Each pixel has a distance to the nearest black background pixel. The border of the black background pixels (the edge of the crack) is highlighted in blue. The two green-highlighted pixels are provided to show their distance (white arrow) to the nearest black background pixels. The pixel highlighted in red has the largest value of 13 for the entire image, meaning that it is the furthest from the edge.



This value of 13 is then doubled in line 36 to calculate the maximum width of the segmented crack.