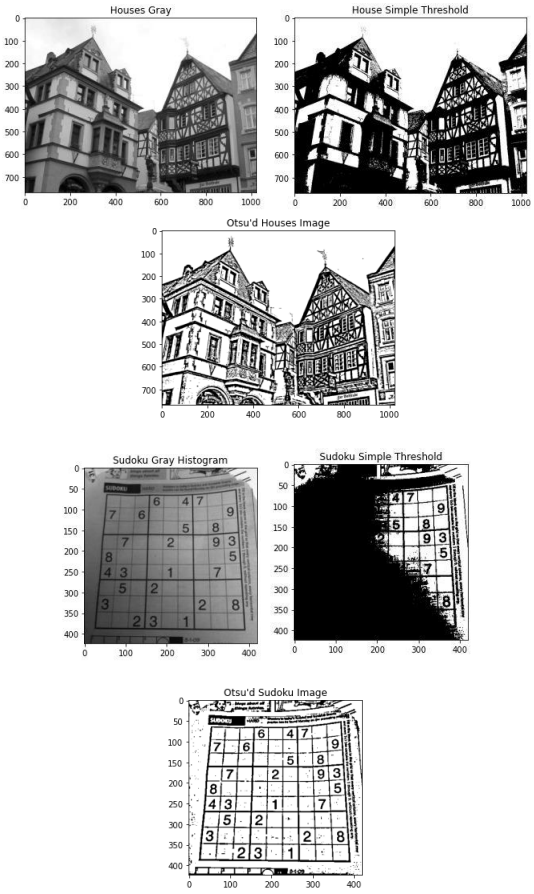


ELEC 474

Lab 1 Histograms

Histograms and Adaptive Thresholding

Kevin Huang and Michael Greenspan

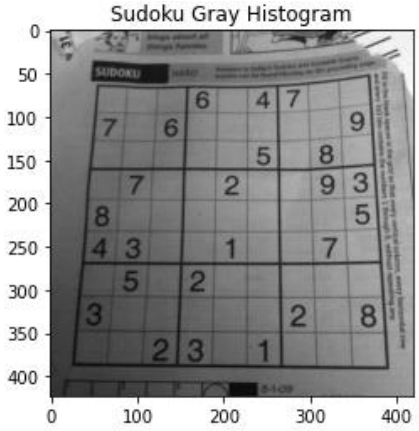
copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

Contents

copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

1. Histograms and Simple Thresholding

A histogram is one of the simplest ways to represent and process an image. It is based only on the image pixel values, and is independent of the spatial distribution of the pixels. For this portion of the lab you will perform global thresholding on both the “houses.jpg” and “sudoku.jpg” images. Load these in as grayscale images, as we will be looking first only at the single channel intensities.

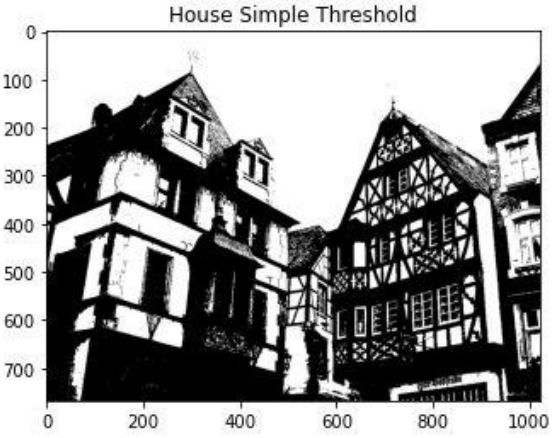


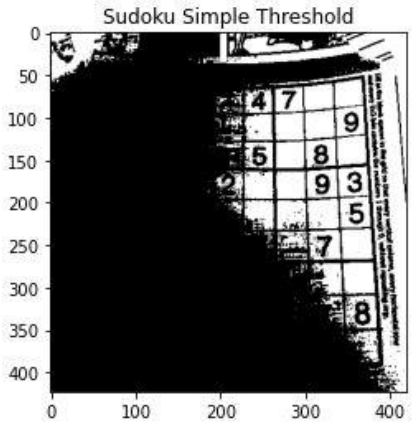
1.1 Global Binary Threshold

Write a method to perform simple global thresholding on these images. Keep in mind that the intensity values are between 0 and 255. You should:

- Create an empty image;
- Set an arbitrary threshold (e.g. 127);
- Perform pixel access and manipulation to segment the image into a binary image of either black (0) or white (255) pixels, based on the selected threshold.

These are what the output images will look like, for a threshold value of 127:





You can see that simple thresholding is able to divide the images into darker and lighter elements, but in the Sudoku image, the lighting gradient across the image made the left side of the image darker. To handle this we must implement Adaptive Thresholding.

copyright 2020 RCVLab, Queen’s University, Kingston, Ontario, Canada

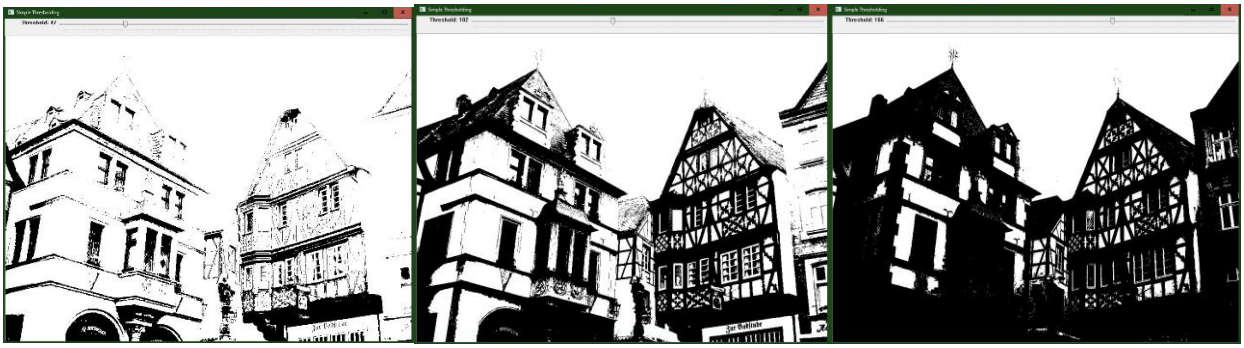
1.2 Global Binary Threshold, Trackbar Input

Before we get to Adaptive Thresholding, let's first implement a slider to allow the user to interactively set the global binary threshold value. To accomplish this, you will use OpenCV's **createTrackbar()** method. **createTrackbar()** will allow you to use the slider's value in order to modify you image's global threshold. It is important to note that you must use OpenCV's **cv2.imshow()** instead of the plot's **imshow()** function which shows images in Jupyter Lab's cells. The use of **cv2.imshow()** will allow the modification of images interactively in their own separate window, as well as added GUI functionality.

Important things to know:

- Use **cv2.namedWindow()** to set a window to place both your image and slider;
- Set your **createTrackbar()** threshold range from 0 to 255 (intensity values);
- Your image should be updated after every slider interaction;
- Include **cv2.waitKey(0)** and **cv2.destroyAllWindows()** at the end of your code to prevent Jupyter Lab from crashing.

Your output should look like this:



These images have their global binary threshold value set by their trackbar for three different threshold settings.

copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

2. Adaptive Thresholding

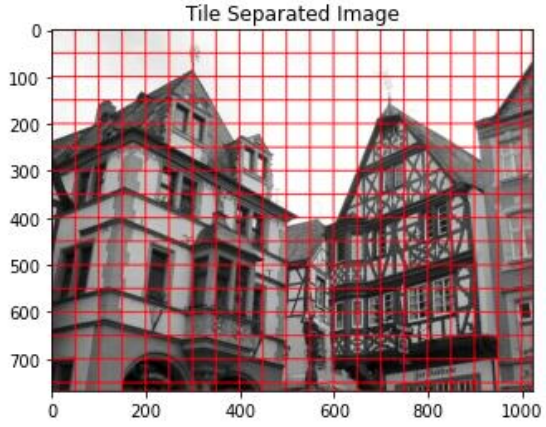
To account for variations in image intensity values at different regions of the image, which can be caused for example by variable lighting conditions, adaptive thresholding can be implemented. Here, instead of using a single global threshold value for the entire image, you will calculate multiple distinct threshold values for distinct small regions (e.g. "tiles") of the image. Each tile will have its own calculated threshold applied to it, and this will provide better results for images with varying intensity values.

2.1 Tiles

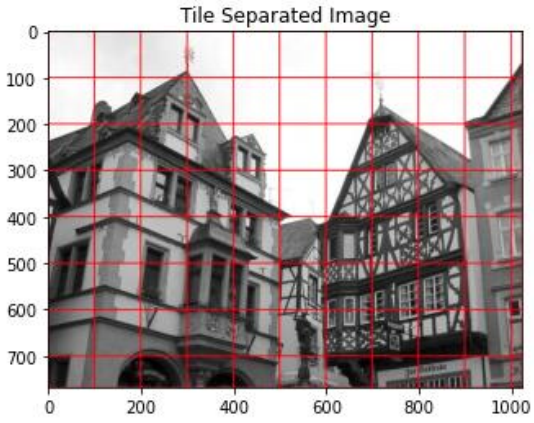
The first important thing you should do is split your image into "tiles". As a first step, implement a drawing function that provides a visual representation of the tile concept. Start with a big tile size (e.g. 100x100) and implement the following :

- Loop through your image based on the tile size and the window size;
- Use OpenCV's **rectangle()** function to draw rectangles in the correct positions. OpenCV's rectangles are drawn based on two specified points, i.e. the top-left and bottom-right points;
- You should account for out of bounds regions, which are areas where parts of your tiles are past the image bounds. This will happen when your tile size is not an integer factor of your image size. You need to keep track of how many columns and/or rows pixels that your window passes, and account for this difference when retrieving image pixels that are within the image bounds.

The implementation of your sliding window drawing function should look like this, for two different tile sizes:



tile size = 50x50



tile size = 100x100

Notice that the tiles are drawn for the smaller partial regions at both image bounds, i.e. at the far right column and bottom row.

copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

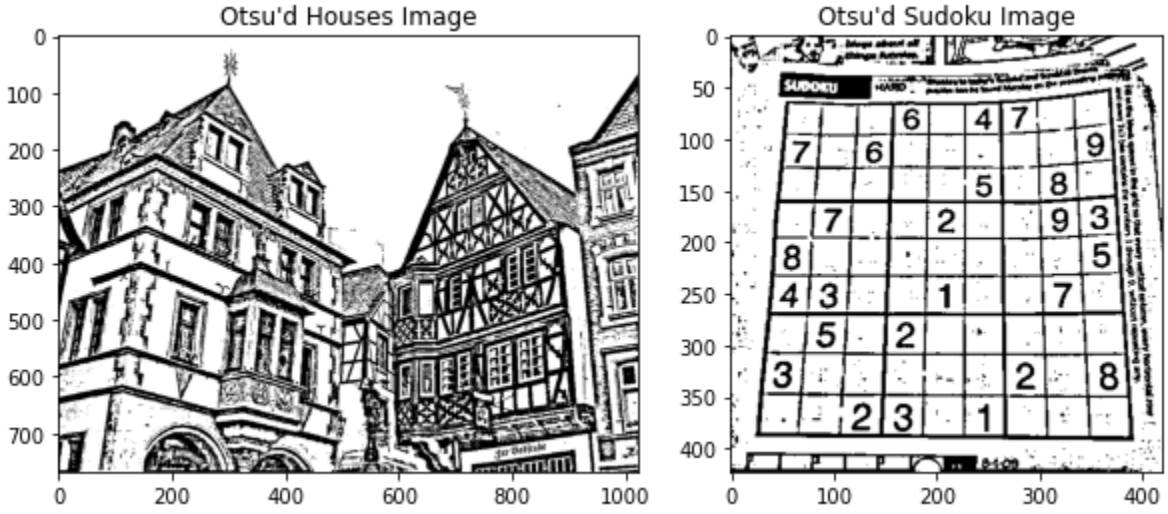
2.2 Otsu Threshold

Next, rather than arbitrarily selecting a binary threshold value as above, implement your own version of Otsu's method to calculate a distinct optimal threshold value for each tile.

For Otsu's method, here are some important things to keep in mind:

- Numpy provides some useful histogram functions to access the data;
- Implement Otsu's method as a separate Python method, that can be applied independently to each tile;
- You must calculate the within-class and between-class variances for all possible threshold values that lie within [0, 255];
- For each image tile, perform Otsu's method to determine a threshold for that region and apply the calculated threshold to that region;
- Repeat this for all possible tiles.

Your output on "houses.jpg" and "sudoku.jpg" should now look like this:



copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

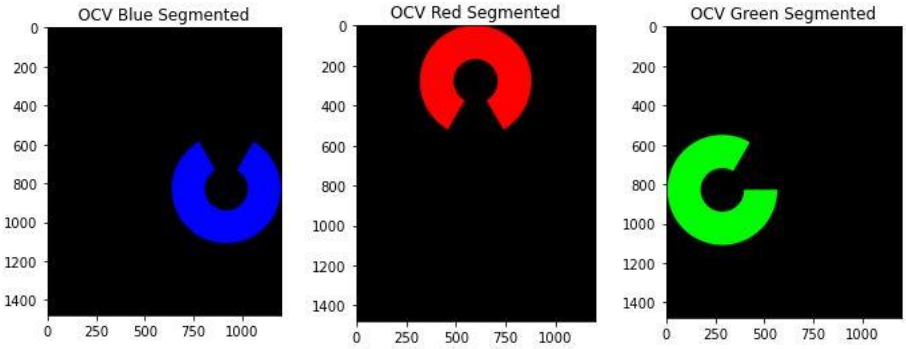
2.3 Thresholding RGB images

For this section, you will apply thesholding to the three color channels of the OpenCV logo. You should:

- Separate the RGB image to color channels and make sure they are all one channel images holding the RGB image's color intensities;
- Perform thresholding on each color's intensities and segment each color channel into a separate image. For threshold selection for each channel, you can either apply the Otsu method that you developed above, or you can choose your threshold some other way (e.g. through manual selection).

You should initially have:

Your output should be:



copyright 2020 RCVLab, Queen's University, Kingston, Ontario, Canada

3. Deliverables

The deliverables for this lab should include:

- .ipynb file that includes code that performs:
  - 1.1 Global Binary Threshold;
  - 1.2 Global Binary Threshold, Trackbar Input: Execute this on both “houses.jpg” and “sudoku.jpg” (3 threshold images each);
  - 2.1 Tiles: Applied to “houses.jpg”;
  - 2.2 Otsu Threshold: Implement and apply Otsu thresholding on both “houses.jpg” and “sudoku.jpg”;
  - 2.3 Thesholding RGB Images: Apply thresholding (Otsu, or other) on the RGB color channels of “OCVLogo.png”, and display their thresholded images.

Your delivered code will be run in Jupyter Lab to test for for functionality. The marking rubric is as follows:

Section	mark
1.1 Global Binary Threshold	1
1.2 Global Binary Threshold, Trackbar Input	1
2.1 Tiles	1
2.2 Otsu Threshold	1
2.3 Thesholding RGB Images	1
Total:	5

copyright 2020 RCVLab, Queen’s University, Kingston, Ontario, Canada