CSE 152 : Introduction to Computer Vision, Spring 2018 – Assignment 3

Instructor: Ben Ochoa

Assignment Published On: Wednesday, April 25, 2018

Due On: Wednesday, May 9, 2018, 11:59 PM

Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains both math and programming problems.
- All solutions must be written in this notebook
- For the Math problems you may use Markdown/LATEX or you can work it out on paper and upload the scanned copy after merging with the .ipynb PDF. Remember to show work and describe your solution.
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.

Problem 1: Binarization [10 points]

Write a Python function to implement Otsu's method described in class. This algorithm should automatically determine an intensity level to threshold an image to segment out the foreground from the background. The output of your function should be a binary image that is black (pixel values = 0) for all background pixels and white (pixel values = 1) for all foreground pixels. Apply this function to the image can_pix.png and turn in the output image in your report.

Notes:

- Load in an image and convert it to grayscale.
- You can use np.histogram or matplotlib.pyplot.hist to create a histogram of pixel intensities as a first step.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

def otsu(grey_img):
    binary_img = np.copy(grey_img)
    h, bins, patches = plt.hist(grey_img)   # Histogram of px intensitie
s

    threshold = 0

    # Set appropriate threshold using Otsu's method
    # Your code here
    #

    ####

    # Set pixel values in the binary_img based on the threshold
    # Your code here
    #
    return binary_img
```

Problem 2: Connected Components [40 points]

2.1 Connected Regions [15 pts]

(a) Write Python code to implement the connected component labeling algorithm discussed in class, assuming 8-connectedness. Your function should take as input a binary image (computed using your algorithm from question 1) and output a 2D matrix of the same size where each connected region is marked with a distinct positive number (e.g. 1, 2, 3). On the image can pix.png, display an image of detected connected components where connected region is mapped to a distinct color using imshow. You may need to handle recursive calls carefully to avoid system crash by using sys.setrecursionLimit(1000).

(b) How many components do you think are in the image coins pix.jpg? What does your connected component algorithm give as output for this image? Include your output on this image in your report. It may help to reduce the size of the image before running the connected components algorithm but after converting the image to a binary image.

2.2 Take your own images [5 pts]

For this part, you will be using images you take with your own camera. Choose three objects of different shapes, preferably with non-shiny surfaces. Possibilities include paper/cardboard cut-outs, bottle tops, pencils, Lego pieces, potatoes, etc. Take 3 pictures of these objects individually with a solid background. The object should be clearly distinguishable from the background (e.g. bright object & dark background). Include these three images in your report as in Figure 1. In addition, show similar output images as in Problem 2.1 for each of your new images (there is only 1 connected component in this case).

Figure 1: Sample images.

2.3 Image moments and rectification [10 pts]

Write three functions which compute the moments, central moments, and normalized moments of a marked region. Each function should take as input a 2D matrix (output of part 2.2) and 3 numbers j, k, and d. The output should be the (j, k) moment $M_{j,k}$, central moment $\mu_{j,k}$, normalized moment $m_{j,k}$ of the region marked with positive number d in the input matrix.

Using these functions, on each of the three images, draw the centroid of each object. Also compute the eigenvectors of the centralized second moment matrix and draw the two eigenvectors on the centroid. This should indicate the orientation of each object, see Figure 2 for the results on the example images. Turn in the outputs for your own images.

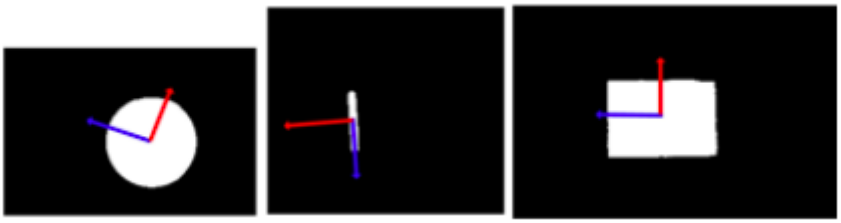

Figure 2: Sample images with principle directions overlaid.

2.4 Image alignment [10 pts]

We have seen that the orientation computed from Problem 2.3 can be used to roughly align the orientation of the region (i.e. in-plane rotation). Write a function to rotate the region around its centroid so that the eigenvector corresponding to the largest eigenvalue (i.e. the blue vector in Figure 3) will be horizontal (aligned with $[1, 0]^T$). This might look like in Figure 3. Your function should take as input a 2D matrix (output of Problem 2.1) and output a matrix of the same size in which all marked regions are aligned. Turn in the aligned outputs for your images. Note: After finding the rotation matrix R to rotate the largest eigenvector to $[1, 0]^T$, we rotate all points (x, y) belonging to that region using the following transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{bmatrix} x - \hat{x} \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$$

where $[\hat{x}, \hat{y}]^T$ T are the centroid coordinates of the region. For simplicity, just ignore the cases when part of the aligned region falls outside of the image border or is overlapped with other regions. You can avoid these issues when capturing your images (e.g. put your objects a bit far apart). Finally, note that the rotation matrix can be created trivially from the eigenvectors.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import sys

sys.setrecursionlimit(100000)

def connected_component(img):
    # your code here
    return cc_img

def moment(cc_img, j, k, d):
    # your code here
    return M_jk

def central_moment(cc_img, j, k, d):
    # your code here
    return mu_jk

def norm_moment(cc_img, j, k, d):
    # your code here
    return m_jk

def plot_img_moments(img):
    cc_img = connected_component(img)
    # your code here

def rotate_region(cc_img, centroid, eigenvectors):
    aligned_img = np.copy(cc_img)
    # your code here
    return aligned_img
```

Problem 3: Filtering [15 points]

In this problem we will play with convolution filters. Filters, when convolved with an image, will respond strongest on locations of an image that look like the filter when it is rotated by 180 degrees. This allows us to use filters as object templates in order to identify specific objects within an image. In the case of this assignment, we will be finding cars within an image by convolving a car template onto that image. Although this is not a very good way to do object detection, this problem will show you some of the steps necessary to create a good object detector. The goal of this problem will be to teach some pre-processing steps to make vision algorithms be successful and some strengths and weaknesses of filters. Each problem will ask you to analyze and explain your results. If you do not provide an explanation of why or why not something happened, then you will not get full credit.

3.1 Warmup - Mickey Detection [5 pts]

First you will convolve a filter to a synthetic image. The filter or template is filter.jpg and the synthetic image is toy.png. These files are available on the course webpage. You will first want to modify the filter image and original slightly. We will do so by subtracting the mean of the image intensities from the image i.e. $I_t \rightarrow I - means(\mathbf{vec}(I))$, where I_t is the transformed image and I is the original image.

To convolve the filter image with the toy example, use scipy.ndimage.convolve. The output of the convolution will create an intensity image. In the original image (not the image with its mean subtracted), draw a bounding box of the same size as the filter image around the top 3 intensity value locations in the convolved image. Provide both the intensity map and bounding box images in the report.

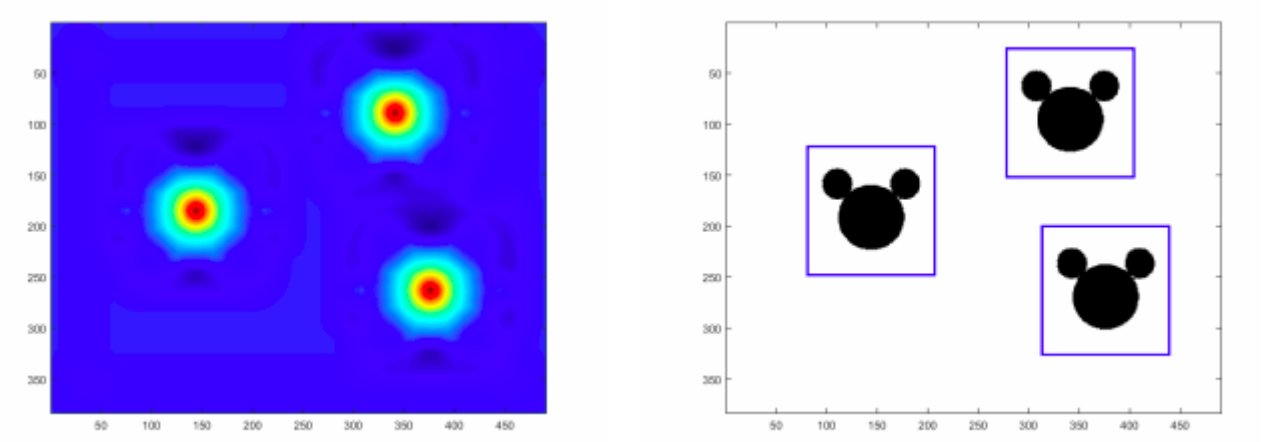

Figure 4: Example outputs for the synthetic example. Left - Intensity map. Right - Bounding boxes

The outputs should look like the above figure. Describe how well you think this technique will work on more realistic images. Do you foresee any problems for this algorithm on more realistic images?

3.2 Detection quality [4 pts]

We have now created an algorithm that produces a bounding box around a detected object. However, we have no way to know if the bounding box is good or bad. In the example images shown above, the bounding boxes look reasonable, but not perfect. Given a ground truth bounding box (g) and a predicted bounding box (p), a commonly used measurement for bounding box quality is:

$$r = \frac{p \cap g}{p \cup g}$$

More intuitively, this is the number of overlapping pixels between the bounding boxes divided by the total number of unique pixels of the two bounding boxes combined. Assuming that all bounding boxes will be axis-aligned rectangles, implement this error function and try it on the toy example in the previous section. Choose 3 different ground truth bounding box sizes around one of the Mickey 3 silhouettes. In general, if the overlap is 50% or more, you may consider that the detection did a good job.

3.3 Car Detection [6 pts]

Now that you have created an algorithm for matching templates and a function to determine the quality of the match, it is time to try some more realistic images. The file, cartemplate.jpg , will be the filter to convolve on each of the 5 other car images **(car1.jpg, car2.jpg, car3.jpg)**. Each image will have an associated text files that contains two x, y coordinates (one pair per line). These coordinates will be the ground truth bounding box for each image. For each car image, provide the following:

1. A heat map image
2. The provided ground truth bounding box drawn on the original image (green)
3. The detected bounding box drawn on the original image (blue)
4. The overlap between the two bounding boxes drawn on the original image (purple)
5. The bounding box overlap percent r × 100%

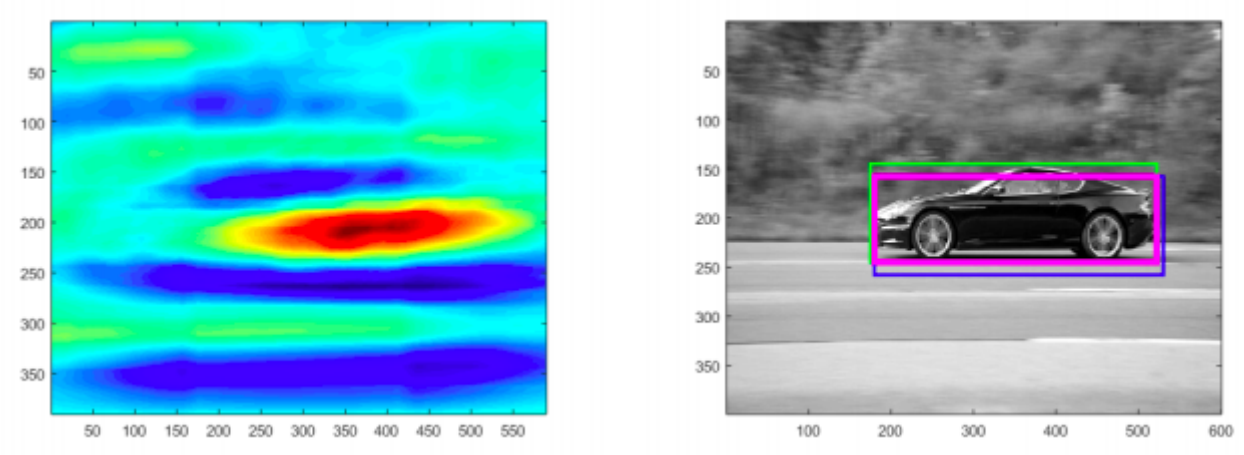

Figure 4: Example outputs for the car example. Left - Intensity map. Right - Bounding boxes

Here are some helpful hints to increase the overlap percentage:

- Rescaling the car template to various sizes (for **car1.jpg**)
- Horizontally flipping the car template (for **car2.jpg**)
- A combination of the first two hints (for **car3.jpg**)
- Gaussian blurring might be useful in all cases

An example output is shown for **car1.jpg** in Figure 5. It may not be possible to achieve 50% overlap on all the images. Your analysis of the images will be worth as much as achieving a high overlap percentage.

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.ndimage.convolve as conv

def detection_quality(predicted_bbox, true_bbox):
    # your code here
    return r
```