Fall 2016 - CS 477/577 - Introduction to Computer Vision

Assignment Five

Due: 11:59pm (*) Tuesday, November 1.

(*) There is grace until 8am the next morning, as the instructor will not grade assignment before then. However, once the instructor starts grading assignments, no more assignments will be accepted.

Weight: 6 points

This assignment must be done individually

General instructions

Unless there is a good reason others, you should use Matlab for this assignment. **If you want to use any other language, you must discuss this with the instructor at lease one week before the due date**. If permission is granted, you will have to sort out any image handling and numerical library support on your own or as a group. (The IVILAB has support for C/C++ , with example files geared towards this course that can be made available on request).

You need to create a PDF document that tells the story of the assignment, copying into it output, code snippets, and images that are displayed when the program runs. Even if the question does not remind you to put the resulting image into the PDF, if it is flagged with ($), you should do so. I should not need to run the program to verify that you attempted the question. See

http://kobus.ca/teaching/assignment-instructions.pdf

for more details about doing a good write-up. While it takes work, it is well worth getting better and more efficient at this. A substantive part of each assignment grade is reserved for exposition.

Assignment specification

This assignment has two parts, one which is required for both undergrads and grads, and one more which is required for grads. The grad student question further breaks down to choosing one of two options. If you would like to do both of them, modest extra credit is available. Finally, you can part E from assignment 3 for modest extra credit if you have not already done it.

To simplify things, you should hard code the file names in the version of your program that you hand in. You can assume that if the grader needs to run your code, they will do so in a directory that has the files linked from this page.

Part A (Required for both undergrads and grads)

To keep things simple, we will focus on black and white (gray level) images This means that an image is basically a matrix. Thus both in Matlab and in C it likely easiest to get the image into a matrix, work on the matrix, and then convert the matrix back to an image for display. (Or simply take advantage of the fact that in Matlab `imshow` usually does the correct thing, but you still may need to scale and/or shift the image pixel values to get a nicely displayed image).

The input image for this assignment is http://kobus.ca/teaching/cs477/data/climber.tiff

1. Write a program to show the magnitude of the gradient of the input image using convolution to implement finite differences. You will want to scale the brightness of your result so that it looks reasonable. Your program should display your result, which should also be put into your writeup **($)**.

   You can use built-in / library routine(s) for convolution in Matlab / C. Or you may prefer to implement convolution yourself to help make sure you really understand it. (A naive implementation in Matlab will run much slower than the built in function--you should understand why). However, you should **not** use the Matlab routine `imgradient`, as is obscures what is going on a bit too much for pedagogy.

2. Now find a threshold for the gradient magnitude to determine real edges. Make a new image which is white if the gradient magnitude exceeds that threshold, and black (zero) otherwise. Likely you will find that there does not exist a perfect threshold which identifies exactly all points on all significant edges. Don't spend too much time tweaking the threshold, but do find a reasonable value. Put the result into your writeup **($)**.

3. Now use convolution to smooth the image with a Gaussian mask with sigma set at 2 pixels. First create the mask using relatively low level operations (either loops and the Gaussian formula, or perhaps using functions like `meshgrid` and `mvnpdf`). The point is to make sure you understand filter creation, including judging a reasonable size and scaling it so that it sums to one. Provide a surface plot of your filter **($)** . Then smooth your image using a convolution routine which does not assume anything about the mask (e.g., conv2). In particular, please do not use `imgaussfilt`() as it hides too much of what is going on. Put the result into your writeup **($).**

4. Now we will do blurring followed by edge detection. Apply the procedure from (1) and (2) to find edges in the blurred image that you computed in (3). Thus, for the image computed in (3), do edge detection as in (1) and find a threshold as in (2) and output a binary image based on that threshold (as in (2)). Put the result into your writeup **($)**.

5. Now we will combine blurring and edge detection into one filter. In particular, convolve the image with an appropriate single mask to implement **both** smoothing by a sigma set to 4 pixels and finite difference operation in the X direction. This filter is the composition of two filters. Do the same for the Y direction. Combine the two results, and put the new gradient image into your writeup **($)**. Verify for yourself (i.e., this is simply a check---don't hand anything in) that this gives roughly the same result as doing the convolutions separately with the appropriate masks, which is like doing (4) with a sigma of 4 instead of 2.

6. A function is said to be separable (in x and y), if f(x,y)=g(x)h(y), If this is the case, then convolution with f(x,y) can be implemented as a 1D convolution by g(x) followed by a 1D convolution by h(y). Re-implement the smoothing in (3) using this approach. Verify for yourself that this gives roughly the same result as before.

   > *Matlab users: This is just a matter of calling conv2 with different parameters.*
   > *KJB library users: See x_convolve_matrix and y_convolve_matrix.*

   Do you expect any speed difference? Why? Do you see any speed difference? (You may want to put the convolution in a loop, or use a bigger sigma and image to test for time differences). You might consider a different separable function other than the Gaussian smoother in case Matlab is too smart about your input. Put your description of what you found into your writeup ($).

(Required for grad students only).

Choose between B1 and B2 below. If you do both of them, you will be eligible for modest extra credit.

Part B1 (edge detection)

Implement the gradient based edge detector described in class. You will need to implement the maximum suppression part, as well as edge following to get edge pieces. Determine some way of visualizing the connectivity that you find (color?) **($).** Notice that the edges found in the previous sometimes follow contours, but the existence of a contour (that some edge points chain together) is all in **your** mind. Now we are computing a representation of contours, and it would be nice if that could be visualized. In your writeup, tell the TA the story of your edge detection work, how your edge detector performs with various choices for thresholds, sigmas, etc., providing some images with informative captions. **($)**

Part B2 (OCR)

The following file has some rendered text
http://kobus.ca/teaching/cs477/data/lorem_ipsum.tiff

Create templates for some letters (say 5 of them) by cutting out small images containing just that letter. Recall that it is usually a good idea to shift such a filter so that the response to uniform image sections is zero (i.e., the filter sums to zero). Use correlation to find areas of high filter response to each of the filters. You will need to set a threshold for when the likelihood of misidentification is too great. Also, if you have a hit at a given threshold for two letters, you will naturally choose the one with highest response. If you do not have any such conflicts, perhaps add a few letters to keep it interesting. You will want to visualize the results of your OCR system. In particular, you want to make it clear which letters are found, and what they are identified as, perhaps with some color coded boxes overlaid on the original images. Again, you will have made some choices (e.g., a threshold), and the TA would love to know what you found, partly through nice pictures **($).** Finally, a nice touch would be a confusion matrix **($).**

> *In a confusion matrix the rows and columns both correspond to the items you are trying to classify (e.g., letters). So, if you have 5 letters, you will have 5 rows and 5 columns. The order of correspondence should be the same. Each element of the matrix counts how many times the row letter was classified as a column letter. So, a perfect classifier only has non-zero elements on the diagonal. If we scale the matrix properly, we have an estimate of how likely a letter (say "Q" will be called any of the possible letters, and tells us about how "confused" the classifier is for each item. This is very good for understanding in detail the kinds of mistakes your classifier is making.*

Part C (Optional)

If you were keen on doing HW3 part E, but did not quite have time, it will be considered for grade as part of this assignment.

What to Hand In

As usual, the main deliverable will be PDF document that tells the story of your assignment as described above. Ideally the grader can focus on that document, simply checking that the code exists, and seems up to the task of producing the figures and results in the document. But you need hand in your code as well as follows.

If you are working in Matlab (recommended): You should provide a Matlab program named hw5.m, as well any additional dot m files if you choose to break up the problem into multiple files. Do not package up the files into a tar or zip file–this messes up the D2L conventions.

If you are working in C/C++ or any other compiled language you need to discuss this with the instructor at least one week before the due date: You should provide a Makefile that builds a program named hw5, as well as the code. The grader will type:

```
make    ./hw5
```

You can also hand in hw4-pre-compiled which is an executable pre-built version that can be consulted if there are problems with make. However, the grader has limited time to figure out what is broken with your build. In general a C/C++ solution will require nonstandard libraries, and you should discuss with the instructor how they can be provided as part of your submission, or assumed to exist on the system that is used for testing.

If you are working in any other interpreted/scripting language (again you need to discuss this with the instructor at least one week before the due date): Hand in a script named hw4 and any supporting files. The grader will type:

```
./hw5
```