Assignment 1

CS5370: Deep Learning for Vision
IIT-Hyderabad
Jul-Nov 2019

Max Marks: 35
Due: 20th Aug 2019 11:59 pm

Instructions

- Please use Google Classroom to upload your submission by the deadline mentioned above. Your submission should comprise of a single ZIP file, named `<Your_Roll_No>_Assign1`, with all your solutions, including code.
- For late submissions, 10% is deducted for each day (including weekend) late after an assignment is due. Note that each student begins the course with 7 grace days for late submission of assignments. Late submissions will automatically use your grace days balance, if you have any left. You can see your balance on the CS5370 Marks and Grace Days document under the course Google drive.
- You have to use PYTHON for the programming questions.
- Please read the department plagiarism policy. Do not engage in any form of cheating - strict penalties will be imposed for both givers and takers. Please talk to instructor or TA if you have concerns.

1 Exercises (35 marks)

1. **Cross-correlation** *(4 + 2 + 3 + 3 = 12 marks)*: In class, we covered cross-correlation, in which a template image is multiplied with sections of a larger image to measure how similar each section is to the template. Normalized cross-correlation is a small refinement to this process. More specifically, before multiplying the template with each small section of the image, the section is scaled and offset so it has zero mean and variance of 1. This increases accuracy by penalizing image sections which have high intensity but do not match the pattern of the template.

   (a) Write your own code to perform normalized cross-correlation (NCC) given an image and a template. Test this using your own face image, matching it in any group photo of yours (simply take a group photo of yours, crop out your face and use it as a template). In the result, describe why the peak occurs where it does. Include the pictures in your report (to be submitted with the assignment).

   (b) Now, use the provided image (`u2cuba.jpg`) and template (`trailer.png`) to test your code. Explain the straight-line artifacts you observe in the cross-correlation (*Hint:* look at the template and the original image).

   (c) Now, perform cross-correlation using the larger template (`trailerSlightlyBigger.png`). Note that the larger template does not exactly match the image. Describe your results, and why they

are different from part (b). What does this tell you about the limitations of cross-correlation for identifying objects in real-world photos? (If required, scale the image intensity in the correlation image so that it covers the full range. If you do this, please remember that the color white will not denote the same value in the two output cross-correlation images.)

(d) Above, we saw that cross-correlation can be fragile. One way to make it less fragile is to perform cross-correlation using many templates to cover the different ways an object may appear in an image. Suppose we wish to search for N_R possible rotations of an object at N_S possible sizes. Assume the image is size $n \times n$ and the template is roughly size $m \times m$. How many mathematical operations will the entire search require? Here, were looking for a Big-O Notation estimate. In other words, you may neglect constant factors, such as the effects of image edge padding and smaller terms.

2. **Convolution** *(5 marks)*: Write your own Python function that implements the discrete 2D convolution operator, i.e., given an intensity image and a filter mask (a matrix with coefficients), your function should convolve the source image with the filter mask and return the resulting output image. You may assume that the filter mask is a square matrix with odd size (e.g. 3 x 3, 5 x 5, 7 x 7). You will need to decide on a sensible strategy for dealing with the image borders. An example skeleton for your function is given as follows.

```
def my_conv2(im_in, kernel):
...
return (im_in*kernel)
```

To execute your operator, you should call your implemented function as:

```
>> my_conv2(inputimage, mask)
```

Please note: your implementation will be slow, depending on the input image size and kernel size. So you may want to test your code with smaller, thumbnail images during debugging.

3. **Edge Detection** *($4 + 2 = 6$ marks)*:

(a) Using your implementation of the convolution operator, try out and compare the following edge filters on the provided `clown.tif` image. The filters are described in the lecture slides.

- Sobel edge detector: size 3×3, vertical G_x and horizontal G_y. Compute the approximate magnitude $|G|$ of the filter responses: $|G| = |G_x| + |G_y|$.
- Laplacian for edge detection: size 3×3.

Show the original image and all result images (labeled with the used filter kernel and filter kernel size). Compare the results and give a qualitative comment.

(b) Compare your results with the 2-D convolution function `scipy.signal.convolve2d` from the SCIPY package. Compare the outputs and speed issues that you observed between your code and the inbuilt function.

4. **Hybrid Images** *(8 marks)*: This exercise is based on the paper on this topic in SIGGRAPH 2006 by Oliva, Torralba, and Schyns. A hybrid image is the sum of a low-pass filtered version of one image (removes high-frequency components, i.e. removes edges) and a high-pass filtered version of a second image (the complement of the previous filter). The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high-frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances. Here is an example (Figure 1):

If you're having trouble seeing the multiple interpretations of the hybrid image, a useful way to visualize the effect is by progressively downsampling the hybrid image as is done here (Figure 2).

Figure 1: Example of a hybrid image

Figure 2: Visualizing a hybrid image at different resolutions

We provide you with 5 pairs of aligned images which can be merged reasonably well into hybrid images. The hybrid images will differ depending on which image you assign as `image1` (which will provide the low frequencies) and which image you assign as `image2` (which will provide the high frequencies). You need to submit your code to generate a hybrid image, as well as any one interesting hybrid image example generated using your code. (You can use other images too, but you'll need to align the images in a photo editor such as Photoshop).
There is a free parameter, which can be tuned for each image pair, which controls how much high frequency to remove from the first image and how much low frequency to leave in the second image. This is called the *cutoff frequency*. The cutoff frequency can be controlled by changing the standard deviation of the Gaussian filter used in constructing the hybrid images. You will want to tune this for every image pair to get the best results.
NOTE: You may want to scale the images or translate its intensity (by adding a fixed quantity to every pixel) to get better visualizations where required.

5. **Decoding Photoshop** *(4 marks)***:** Consider any effect in Adobe Photoshop (or GIMP, or any equivalent software) that was NOT discussed in class, and explain what filter you think it is using underneath the hood. Substantiate your answer using example images in your report.

2 Practice Exercises

NO SUBMISSION REQUIRED; PLEASE USE THIS FOR PRACTICE AND LEARNING.
Please follow this link to guide you through the setup process for Python-OpenCV.

1. **Basics of Python-OpenCV-I:** Suppose you are given a 100×100 matrix A representing a grayscale image. Write a few lines of code to do each of the following. Try to avoid using loops.

   (a) Plot all the intensities in A, sorted in decreasing value. (Note, in this case, we don't care about the 2-D structure of A, we only want to sort all the intensities in one list.)

   (b) Display a histogram of A's intensities with 20 bins.

(c) Create and display a new color image the same size as A, but with 3 channels to represent R,G and B values. Set the values to be bright red (i.e., $R = 255$) wherever the intensity in A is greater than a threshold t, and black everywhere else.

(d) Generate a new image, which is the same as A, but with A's mean intensity value subtracted from each pixel (without loops).

2. **Basics of Python-OpenCV-II:** Write functions to do each of the following to an input grayscale image, and then write a script that loads an image, applies each of your functions to the input image, and displays the output results. Label each subplot with title. (Sample images have been supplied with the assignment. Please submit answers using at least one of those images.)

(a) Map a grayscale image to its "negative image", in which the lightest values appear dark and vice versa.

(b) Map the image to its "mirror image", i.e., flipping it left to right.

(c) Swap the red and green color channels of the input color image.

(d) Add or subtract a random value between $[0, 255]$ to every pixel in a grayscale image, then clip the resulting image to have a minimum value of 0 and a maximum value of 255.

3. **Linear Filters:** In class, we introduced 2D discrete space convolution. Consider an input image $I[i, j]$ and a filter $F[i, j]$. The 2D convolution $F * I$ is defined as

$$(F * I)[i, j] = \sum_{k,l} I[i - k, j - l]F[k, l] \tag{1}$$

(a) Convolve the following I and F (using pen and paper). Assume we use zero-padding where necessary.

$$I = \begin{bmatrix} 2 & 0 & 1 \\ 1 & -1 & 2 \end{bmatrix} F = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \tag{2}$$

Please DO NOT write programs. It will also be helpful for answering question (d).

(b) Note that the F given in Equation 2 is separable, that is, it can be written as a product of two filters: $F = F_1F_2$. Find F_1 and F_2. Then, compute $(F_1 * I)$ and $F_2 * (F_1 * I)$, i.e., first perform 1D convolution on each column, followed by another 1D convolution on each row. (Please DO NOT write programs. Do it by hand.)

(c) Prove that for any separable filter $F = F_1F_2$
$F * I = F_2 * (F_1 * I)$
Hint: Expand Equation 1 directly.

(d) Carefully count the exact number of multiplications (multiplications only, including those multiplications due to zero-padding) involved in part (a) and part (b). which one of these requires fewer operations? You may find the computation steps you wrote down for (a) and (b) helpful here.

(e) Consider a more general case: I is an $M_1 \times N_1$ image , and F is an $M_2 \times N_2$ separable filter.

i. How many multiplications do you need to do a direct 2D convolution?

ii. How many multiplications do you need to do 1D convolution on rows and columns?
Hint: For (i) and (ii), we are asking for two functions of M_1, N_1, M_2 and N_2 here, no approximations.

iii. Use Big-O notation to argue which one is more efficient in general: direct 2D convolution or two successive 1D convolutions?

4. **Canny Edge Detector:** Suppose the Canny edge detector successfully detects an edge. The detected edge (shown as the red horizontal line in Figure 4a) is then rotated by θ, where the relationship between a point on the original edge (x, y) and a point on the rotated edge (x', y') is defined as

$$x' = xcos\theta \ ; \ y' = ysin\theta$$

(a) Will the rotated edge be detected using the same Canny edge detector? Provide either a mathematical proof or a counter example. *Hint:* The detection of an edge by the Canny edge detector depends only on the magnitude of its derivative. The derivative at point (x,y) is determined by its components along the x and y directions. Think about how these magnitudes have changed because of the rotation.

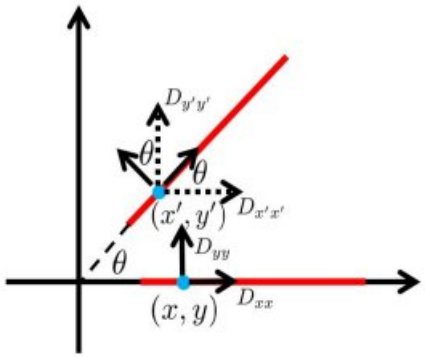

Figure 3: Canny Edge Detector

(b) After running the Canny edge detector on an image, you notice that long edges are broken into short segments separated by gaps. In addition, some spurious edges appear. For each of the two thresholds (low and high) used in hysteresis thresholding, state how you would adjust the threshold (up and down) to address both problems. Assume that a setting exists for the two thresholds that produce the desired result.