P0: Alohomora!

Sarthak Mehta
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA 01609
Email: smehta1@wpi.edu

I. Introduction

In this assignment, the part 1 consists of 3 subparts. For part1 the assignment is to learn about convolution. In which the implementation is done with three different methods. In Part 1.1 convolution is done by scipy.signal.convolve2d. Part 1.2 convolution was done by multiplying a kernel/filter with the input image. And 1.3 is done using pytorch. In all these three cases the image used was Fig 1.

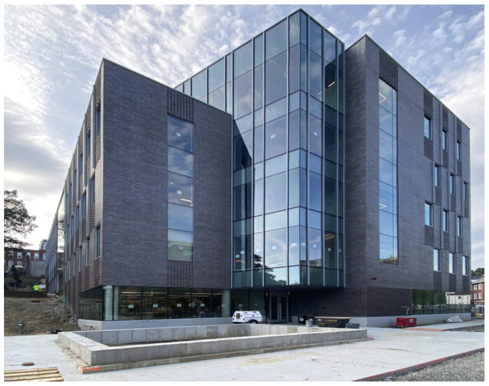

Fig. 1. The input image used for convolution in all the three subparts.

A. Assignment Part-1

1) Part 1.1 Keep Calm And Use scipy.signal.convolve2d: This section used a function named signal.convolve2d from library scipy. The given function takes two inputs. The first input is an image (Fig 1) and the second input is a kernel/filter according to which the convolution will happen. For the assignment, a specific kernel was given.

A kernel is a 3x3 matrix, which is multiplied by the image to bring out the specific features needed for image processing, mapping, etc. The kernel used in this case is called Sobel, which is typically used for edge detection. But the kernel in this case cannot be multiplied directly, if you want to extract horizontal edges then we flip the kernel matrix horizontally. If we want to extract vertical edges then we flip the kernel vertically. In this case, we want the edges both ways so we flip the kernel in both directions.

Sobel kernel/filter: [[1, 0, -1] [2, 0, -2] [1, 0, -1]]

In this, the output printed was the data-type of the image. And the Elapsed Time (s) taken for the function to run its course.
The Elapsed time, in this case, is: **0.024282217025756836**. The output is shown below in **Fig 2**.

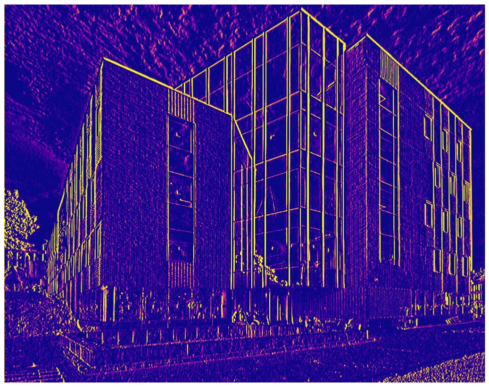

Fig. 2. Output image for part 1.1

2) Part 1.2 Slow And Steady: Applying Convolution With For-Loops: In this section final result was the same but the process followed was different which is instead of using a direct function, we are using a ”for” loop. But in this case, the iteration is done at every pixel by imposing the kernel on a particular pixel and multiplying the corresponding elements, and taking a summation of the individual product. This method was also implemented using Sobel Kernel. In this case, also we used the flipped kernel. However, the output image has different dimensions than the input because of which padding is used. Padding is a process in which we add zeros as extra elements so that the final results don't change the dimensions of the output.
Padding: pad_img = np.pad(img, pad_width=1, mode='constant', constant_values=0)

In this, the output printed was the data-type of the image. And the Elapsed Time (s) taken for the function to run its course.
The Elapsed time, in this case, is: **4.9731810092926025**.
The output is shown in **Fig 3**

3) Part 1.3 Let's Torch It: In this subpart python library, PyTorch is used. The conditions given for this case are that the conv2d layer must be used with a hard-coded kernel. And kernel should be flipped the same as in the above two cases. But the catch in this case was the torch.nn.conv2d layer doesn't take numpy2d as the input because of which matrices are converted into tensor. However, converting into tensor reduces the dimensions of the input matrix and image. Hence,

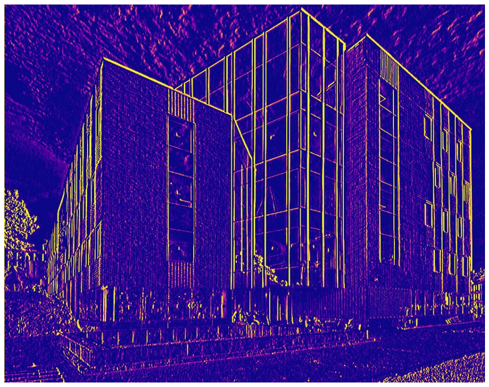

Fig. 3. Output image for part 1.2

unsqueeze function is used two time. To solve the same. e.g tensor_img = tensor_img.unsqueeze(0).unsqueeze(0).float()

In this, the output printed was the data-type of the image. And the Elapsed Time (s) taken for the function to run its course.
The Elapsed time, in this case, is: **0.014664173126220703**.
The output v/s Ground truth is shown in **Fig 4**

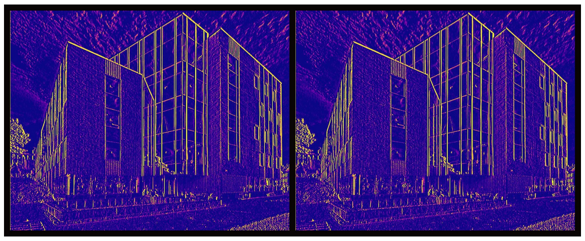

Fig. 4. Output for 1.3 v/s Groundtruth

Parts	Elapsed Time (s)
Part 1.1	0.024282217025756836
Part 1.2	4.9731810092926025
Part 1.3	0.014664173126220703

TABLE I
ELAPSED TIMES FOR PART 1

4) Comparison between Elapsed Times: All the method gives the same final result but the process followed in all the cases are different. In case 1.1 we have directly used a function scipy.signal, which in the backend uses a c compiler. Which runs faster compared to the for loop i.e part 1.2 In the case of 1.2, the program computes by going to each pixel and multiplying with corresponding elements for the convolution. This makes this one a tedious process and hence takes a lot more time compared to the other two. In the case of 1.3 the Pytorch uses GPU directly, which makes the use of hardware efficient. Hence, the with the lowest runtime.

B. Assignment Part-2

In this part of the assignment, the purpose was to learn how to segment images or classes based on their colors. The input is 7 RGB images containing 4 distinct classes of objects. The segmentation of color is done by using Gaussian models that represent probability distribution of each class.

1) Training Code: First we make a list of all the paths of images. In this we would be needing 3 functions, namely masking, extract_rgb and mean_cov_rgb.

Using masking, we are asking the user to select a particular region of interest. This is done to select the region from which we extract the rgb values for the particular area. Masking turns all the other pixel values 0 expect the selected area.

After this the masking image is passed onto extract_rgb function to extract all the RGB pixel values along with the original image. After which a 'bitwise and' operator is used between the original image and mask image and all the pixels that have mask¿0 are only passed further. This values are retuned in the form of list.

The values passed are after that used to find the mean and co-variance of that particular masked image. This mean and covariance values are then stored into a separate json file for also accessible to the segmentation part.

2) Segmentation: Loading the image which is then passed into gaussian distribution function. In the gaussian distribution function firstly the image is converted into float64. And the image is reshaped into pixel vector which contains the pixel values of the image. Each of this pixel value is then passed into probability density function using multivariate gaussina from scipy.stats. The threshold taken in this part for probability¿1e-6 for masking. Then this mask is applied onto the image and output is given. Output is shown Fig5 below.

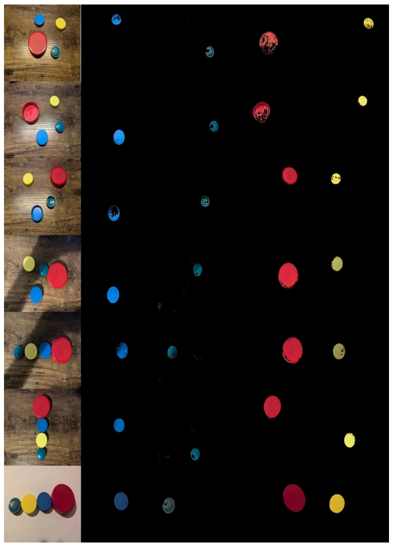

Fig. 5. 5x7 grid for segmentation output