DV1614: Basic edge detection using Python

August 31, 2020

1 Introduction to edge detection

An image consists of pixels, where each pixel is typically represented by three values corresponding to three channels: red, green, and blue. Detecting edges in an image is a frequently employed effect (see Figure 1) in many applications of image processing, machine vision, computer vision, etc.

(a) Input image (b) After edge detection

Figure 1: Performing edge detection on an image

There are multiple ways to detect edges in an image. For edge detection, the input image is typically converted to grayscale. An 8-bit grayscale image represents a pixel using one value (between 0 and 255). The output image can then be obtained by applying a simple technique on all the pixels of the input image. Figure 2 illustrates this technique to detect edges in an image. The *mask* is used to show large differences in pixel values. If a pixel's neighboring pixels differ strongly in intensity, its updated value will appear white, otherwise black.

1.1 Masks that can be used for detecting edges

Mathematical formulae can be used to derive a mask of arbitrary size using which edge detection can be performed on the input image. However, to keep

things simple, we will use two predefined 3x3 masks, either of which can be used to finish this task.

$$\text{Mask } 1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Mask } 2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

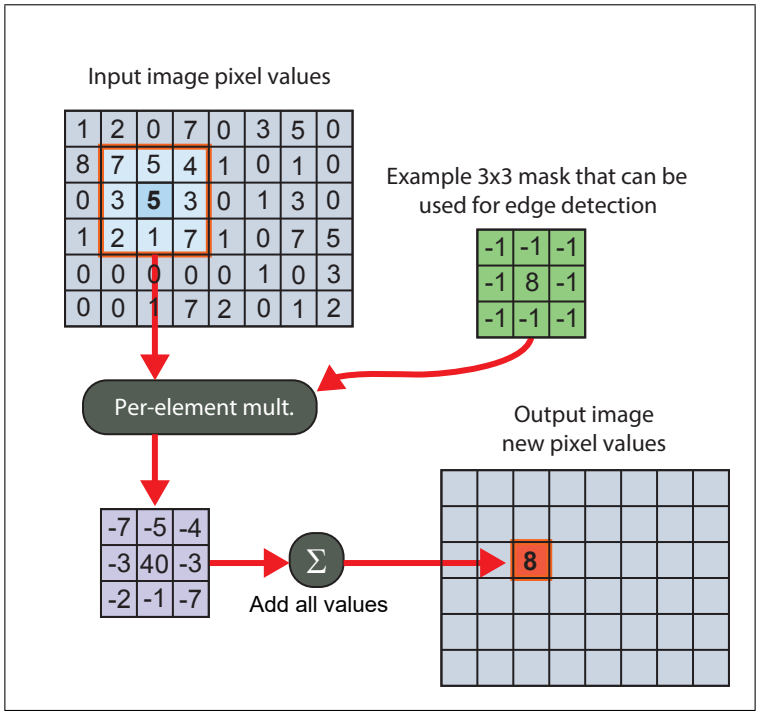

Figure 2: Illustration of edge detection using an example mask. Source: http://cg.ivd.kit.edu

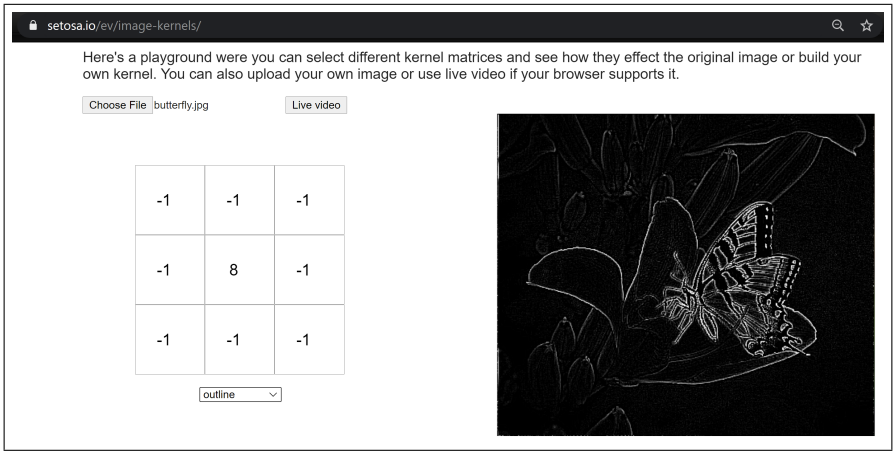

Figure 3: Performing edge detection using the image in Figure 1 and Mask 2. Source: https://setosa.io/ev/image-kernels/

2 Tasks to complete

This section describes the tasks constituting the edge detection program. In order to complete this assignment, you need to complete the tasks 3–5 as per the following instructions.

In the `edgedetect_assgn.py` file:

1. Read the input image file and convert it to 8-bit grayscale format. In this format, each pixel is represented using an integer in the range [0, 256).

2. Add dummy pixels with a value of 0 along the image boundary, so that the 3x3 mask does not fall outside the actual image.

   The above tasks are already implemented in the function `read_colorimg()` in the `helper_functions.py` file given to you. The function returns a list of lists representing the pixel values. Let us name this 2D list `pixel_values`.

3. Using list comprehension, create a list of lists to store the updated pixel information. Let us name this `new_pixel_values`. The length of the list is `numb_rows` and the length of each sublist is `numb_colns`. Initialize all the values to 0.

4. Represent the values comprising the 3x3 image mask using a tuple of tuples. Let us name this 2D tuple `mask`. You can use the values in Mask 1 or Mask 2.

5. Use `mask` to calculate a new value for every pixel in the image, excluding the dummy pixels. Since `pixel_values` is a 2D data structure, use a

nested `for` loop to iterate through every pixel. For each pixel, the following are the tasks that you have to implement:

(a) Call a function `get_slice_2d_list()` (to be implemented by you), which takes as input the `pixel_values`, the position of the pixel being calculated, and returns the 3x3 patch of surrounding pixels as a list of lists. For example, when updating the pixel shown in Figure 2, calling this function from within the nested `for` loop should return `neighbor_pixels` = [[7, 5, 4], [3, 5, 3], [2, 1, 7]].

Use list slicing on `pixel_values` to extract the required neighboring pixels. Use list comprehension to create the 2D list `neighbor_pixels`.

(b) Call a function `flatten()` (to be implemented by you), which takes as input a 2D list or a 2D tuple, and returns a 1D list. For example, when this function is given [[7, 5, 4], [3, 5, 3], [2, 1, 7]] as input, it should return [7, 5, 4, 3, 5, 3, 2, 1, 7]. When it is given ((-1, -1, -1), (-1, 8, -1), (-1, -1, -1)), it returns [-1, -1, -1, -1, 8, -1, -1, -1, -1]. Using this function, flatten `neighbor_pixels` and `mask`.

Use list comprehension in the `flatten()` function to create a 1D list from the 2D list.

(c) Apply the 1D list containing the mask values on the 1D list containing the neighborhood pixels using the `map()` function and a lambda. This lambda should multiply the corresponding elements of the two flattened lists. Your implementation should look as follows:

```
map (lambda ..., list1, list2)
```

Note: This may not be the most efficient way to perform element-wise multiplication of the 3x3 patch of pixels and the mask. However, it is important to learn how to use the `map()` function.

(d) Use the output of `map()` function and sum all the multiplication results to obtain the new value for the pixel. See the Σ operation in Figure 2.

(e) Set the new value by appropriately indexing `new_pixel_values`. Remember that the 2D list `new_pixel_values` does not has dummy pixels.

6. Verify your result against the correct result (computed using the `scipy` package). If your computations are correct, "True result" will be printed on the console. Otherwise, it will output "False result".

7. View the original image and the detected edges.

The last two steps have been implemented in the functions `verify_result()` and `view_images()` in the `helper_functions.py` file.

3 Final note

Note that to be able to use the `helper_functions.py` file, you need to install the following Python packages: `matplotlib, scikit-image, scipy, numpy`. After successful installation, when you run the code provided to you without making any changes, it should look like this:

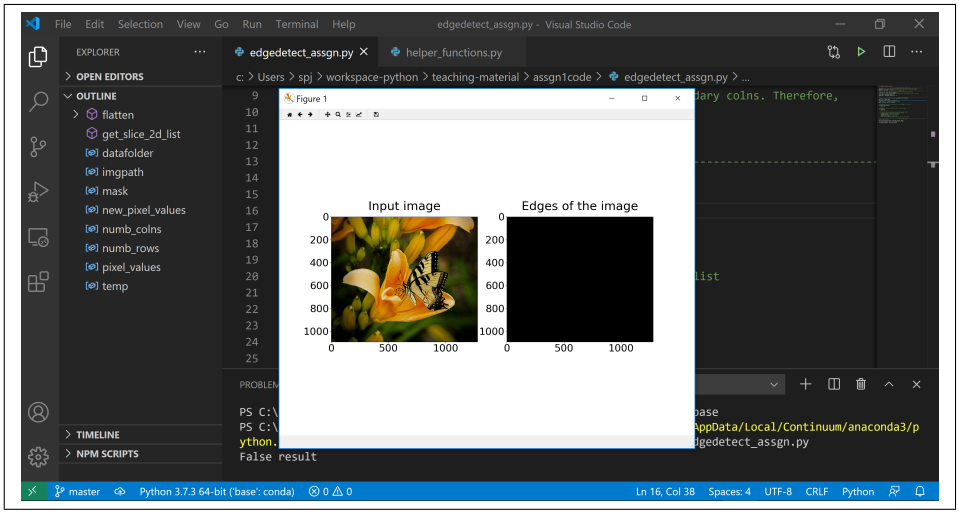

Figure 4: Screenshot of the output when the provided code is executed

You do not need the `temp` variable that currently exists in the code. You can delete it when writing your code. After successfully finishing your tasks, you will be able to see the edges of the input image in the window. Also, the console will print a "True result" message.

If you have any questions, please send a mail to sai.prashanth.josyula@bth.se. All the best!