EEP 596 Computer Vision

HW #2

The purpose of this homework is to continue to explore basic image processing and computer vision concepts using Python and associated libraries. You are free to work with fellow students but should turn in your own work. Specifically, you should be able to explain anything in your assignment if asked.

You should turn in your assignment as “Assignment2.py” and “Report.pdf” in Gradescope. You should develop code with the skeleton provided. You could use print() and cv2.imsave() to help you to generate outputs for the report, but should be commented while turning in.

1. **Floodfill.** Implement the stack-based floodfill algorithm as a Python function named floodfill. The function should take as input a seed and return an image after performing the stack-based floodfill algorithm. Perform the algorithm on the 'ant_outline.png' image and fill the image with green color.

2. **Convolution for Gaussian smoothing.** Write a function with a double-for loop to convolve a grayscale image with the 2D separable Gaussian kernel defined by the 1D kernel, 0.25 * [1 2 1]. Implement this as two 1D convolutions with a vertical and horizontal 3x1 kernel. Apply the function to the image `cat_eye.jpg`. Repeat the convolution **5** times in succession to generate increasingly blurrier outputs. Store the blurred images sequentially in a list and return. You are supposed to write the convolve operation with only add and multiply operations without calling existing functions, e.g., np.convolve. Notice that the returned images should have the same shape as the original image. Use zero padding.

3. **Convolution for differentiation along the vertical direction.** Repeat the previous problem with a 2D Gaussian derivative, defined along the horizontal direction by the 1D smoothing kernel, 0.25 * [1 2 1], and along the vertical direction by the 1D differentiating kernel, 0.5 * [1 0 -1]. In this case, do **not** repeat the convolution 5 times in succession. Rather, apply the convolution to each of the 5 smoothed images from the previous problem. Store the images in a list and return. You are supposed to write the convolve operation with only add and multiply operations without calling existing functions, e.g., np.convolve. Notice that the returned images should have the same shape as the original image. Use zero padding, and be sure to flip the derivative kernel (because it’s convolution).

4. **Differentiation along another direction along the horizontal direction.** Repeat the previous problem with the transpose of the Gaussian derivative, i.e., defined along the vertical direction by the 1D smoothing kernel, 0.25 * [1 2 1], and along the horizontal

direction by the 1D differentiating kernel, 0.5 * [1 0 -1]. Store the images in a list and return. You are supposed to write the convolve operation with only add and multiply operations without calling existing functions as np.convolve. Notice that the returned images should have the same shape with the original image. Use zero padding, and be sure to flip the derivative kernel (because it's convolution).

5. **Gradient magnitude.** Now combine the results by computing the actual magnitude of the gradient (using square root) based on both the horizontal and vertical derivative. You can use the generated images from 3 and 4. Repeat this for each of the 5 smoothed images. Store the gradient magnitude of the 5 images in the list and return.

6. **Built-in convolution.** Now implement Gaussian smoothing using `scipy.signal.convolve2d`. Repeat question 3 with this function. Notice that the returned images should have the same shape as the original image. The difference between the corresponding outputs of question 3 and question 6 should be within 5. Use zero padding.

7. **Repeated box filtering.** Write a function repeatedly convolving a 1D box filter [1,1,1] with itself. The function should take the number of repetitions (i.e., number of convolutions) as input and return the convolved filter. (For example, zero returns the original filter, one performs one convolution, etc.) For each convolution step, it should return a shape of (N+M-1), where M and N are the lengths of two arrays you compute convolution with. You can use the built-in convolution function.