**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 can use print() and cv2.imsave() to help you to generate outputs for the report, but these 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 ant’s face with red color. (Why would you not want to apply floodfill to a JPEG image?)  
   *Details: 1) Load image as single-channel uint8. 2) Use RGB=(255,0,0) for filling the face.*
2. **Convolution for Gaussian smoothing.** Write a function 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 should write the convolve operation with only add and multiply operations and a double for-loop over the pixels, without calling existing functions (e.g., np.convolve).   
   *Details: 1) Load image as single-channel uint8. 2) Returned images should have the same shape as the original image. 3) Returned images should also be single-channel uint8 (round to nearest unsigned byte – be sure to clamp to avoid wraparound). 3) Use zero padding.*
3. **Convolution for differentiation along the vertical direction.**  Repeat the previous problem with a 2D Gaussian vertical 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]. (This is the Sobel kernel.) 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 should write the convolve operation with only add and multiply operations and a double for-loop over the pixels, without calling existing functions (e.g., np.convolve).   
   *Details: 1) Load image as single-channel uint8. 2) Returned images should have the same shape as the original image. 3) Returned images should also be single-channel uint8. 4) Because computation involves negative values, you will need to use signed ints (or floats) for computation. 5) To convert back to unit8, use the following formula: pout = clamp(2 \* pin + 127) (the scale factor of 2 increases visibility, adding an offset of 127 preserves negative values after conversion, and clamping avoids wraparound). 6) Use zero padding. 7) 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. As before, you should write the convolve operation with only add and multiply operations and a double for-loop over the pixels, without calling existing functions (e.g., np.convolve).  
   *Details: Same as previous.*
5. **Gradient magnitude.** Write a function to compute the gradient magnitude of an image. Instead of Euclidean norm (which involves square root), use the Manhattan formula, abs(gx) + abs(gy), where gx and gy are the horizontal and vertical derivatives, respectively. Do not use the generated images from 3 and 4 above. (Why not? Because these have already been scaled to fit within uint8, so clamping may corrupt some values.) Rather, for each pixel, immediately compute the absolute value before storing back as uint8. Repeat this for each of the 5 smoothed images. Store the gradient magnitude of the 5 images in the list and return.  
   *Details: 1) Load image as single-channel uint8. 2) Returned images should have the same shape as the original image. 3) Returned images should also be single-channel uint8. 4) Because computation involves negative values, you will need to use signed ints (or floats) for computation, but you should immediately compute abs() to avoid having to store the result as int (or float). 5) To convert back to unit8, use the following formula: pout = clamp(4 \* pin), where pin = abs(gx) + abs(gy) (the scale factor of 4 increases visibility, and clamping avoids wraparound). 6) Use zero padding. 7) Be sure to flip the derivative kernel (because it’s convolution).*
6. **Built-in convolution.** Now implement Gaussian smoothing using `scipy.signal.convolve2d`. Repeat question 3 with this function. Note that the returned images should have the same shape as the original image.   
   *Details: 1) Use zero padding. 2) The corresponding outputs of question 3 and question 6 should be the same; to allow for some discrepancy, we will use a small threshold when comparing.*
7. **Repeated box filtering.** Write a function to repeatedly convolve a 1D box filter [1,1,1] with itself. (Ignore the scaling factor.) 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, the function should return a shape of (N+M-1), where M and N are the lengths of two arrays you compute convolution with. Write the for-loop yourself; do not use the built-in convolution function. Plot the result for 5 convolutions, to show that it visually approximates a Gaussian.  
   *Details: 1) Be sure to use ints (or floats), in case values go beyond 255.*