Lab 4: Edges and Features

(10% of final score, due by Mar. 12 11:59 PM)

In the following tasks, you can use your input images to test your codes or pick up images from skimage.data() module: [https://scikit-image.org/docs/dev/api/skimage.data.html]

Upload your results (scanned handwritings, image files, WORD or PDF documents) and *.*py* files to the folder of **Lab 4** under **Assessments** > **Assignment** in the D2L system. Name the *.*py* files according to the series numbers of the questions. The *.*py* files should follow the PEP-8 Style Guide for Python Code. [https://www.python.org/dev/peps/pep-0008/]

1. Convolution

   Use Python functions {scipy.signal.convolve2d} or {scipy.signal.convolve}. Here are the links to the function documentation.
   [https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html]

   [https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html]

   a. Convolve the 2D image I with the 2D kernel G, both given below. To properly handle the borders, extend the input by replicating the values, and set the output size to be the same as the input.

   $$I = \begin{bmatrix} 5 & 4 & 0 & 3 \\ 6 & 2 & 1 & 8 \\ 7 & 9 & 4 & 2 \\ 8 & 3 & 6 & 1 \end{bmatrix} \qquad G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

   b. Repeat the computation of the same problem using the separable version of the kernel. First convolve with the horizontal $\frac{1}{4}[1 \quad 2 \quad 1]$, then with the vertical $\frac{1}{4}[1 \quad 2 \quad 1]$. Change the order by convolving the vertical first then the horizontal. Are the results the same? Why?

   c. Compare results from (a) and (b). Are they the same? Why?

2. The first derivative. Compute the gradient of an image in Python according to the following pseudocode. Use {scipy.ndimage.sobel()}to get $G_x(x, y)$ and $G_y(x, y)$. Or define your own derivative operator such as

   $$Sobel_x = gauss_{0.5}(y) \circledast \dot{g}auss_{0.5}(x) = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \circledast \frac{1}{2}[1 \quad 0 \quad -1] = \frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & 1 \end{bmatrix}$$

   $$Sobel_y = gauss_{0.5}(x) \circledast \dot{g}auss_{0.5}(y) = \frac{1}{4}[1 \quad 2 \quad 1] \circledast \frac{1}{2}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

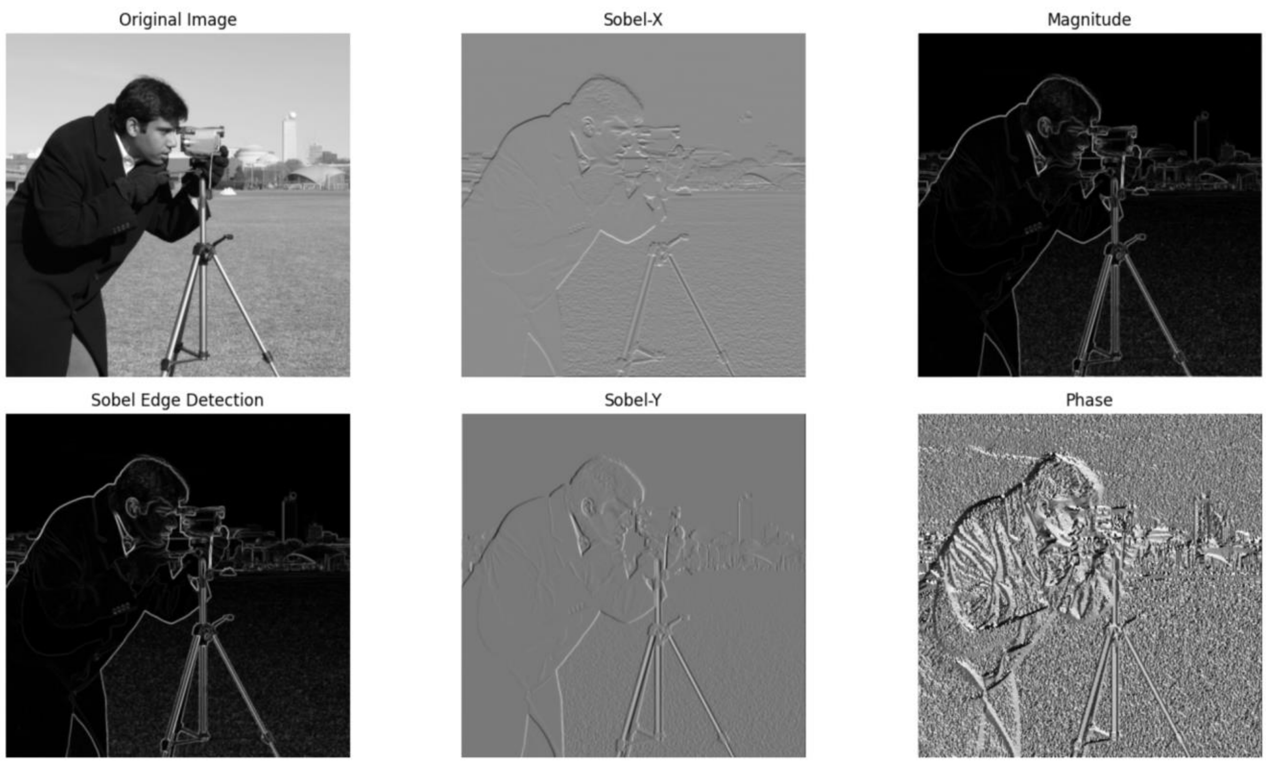
   And calculate $G_x(x, y)$ and $G_y(x, y)$ by {scipy.signal.convolve}.

ALGORITHM 5.9 Compute the gradient of an image

ComputeImageGradient(I, σ)

$gauss$ = CreateGaussianKernel(σ)
$gauss\text{-}deriv$ = CreateGaussianDerivativeKernel(σ)
G_x = ConvolveSeparable(I, $gauss\text{-}deriv$, $gauss$)
G_y = ConvolveSeparable(I, $gauss$, $gauss\text{-}deriv$)
for $(x, y) \in I$ **do**
  $G_{mag} = |G_x(x, y)| + |G_y(x, y)|$
  $G_{phase} = \text{Atan2}(G_y(x, y), G_x(x, y))$
return G_{mag}, G_{phase}

Output the partial derivatives of the image in the *x* and *y* directions, the magnitude and phase of the gradient. An example of the outputs would look like this:



3. The second derivative convolution can detect blobs, which are bright on dark or dark on bright regions in an image. In the following link, blobs are detected using three algorithms: Laplacian of Gaussian, Difference of Gaussian, and Determinant of Hessian. The image used in this case is the Hubble Extreme Deep Field. Each bright dot in the image is a star or a galaxy. [https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_blob.html]

   Download Python source code: < plot_blob.py > from the link and run it. Answer the following questions:

   a. How the three approaches work? Which approach is the most accurate and why?

   b. Which approach is the fastest and why? (Import a module {timeit()} to measure execution time. Print the execution time for each approach. Check which approach uses the shortest time and answer why it is shortest? Here is the link: [https://docs.python.org/3/library/timeit.html] and the function timeit.default_timer())

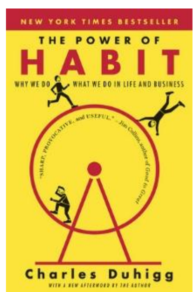
4. Compare Harris and Shi-Tomasi corner measures with the same image data.camera(). Use functions corner_harris(), corner_shi_tomasi(), corner_subpix(), corner_peaks(). Examples shown in the following link:

   [https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_corner.html]

   In the output image, mark the corners. Compare the outputs of the two corner-detection and explain the difference.

5. Feature detection and matching.

   a. Download the following image < book.jpg > from the folder Lab4.
   b. Use {skimage.transform.AffineTransform} to conduct transformation of translation, rotation, scale, and shear. Save the transformed image as < transformed.jpg >.
   c. Use three methods: CENSURE, ORB, and SIFT to extract descriptors in the two images and mark the descriptors in the two images.
   d. Use a matching method: *e.g.* { skimage.feature.match_descriptors } or {cv2.BFMatcher.knnMatch } or { cv2.FlannBasedMatcher.knnMatch } to match the descriptors in the two images; show the matching in the two images.



   e. Compare the three detection methods, which one works the best? Why? (compare at least two performance metrics, *e.g*. *The Putative Match Ratio = # of putative matches / # of features* and the execution time)
   f. Download two images < IMG_1.jpg> and <IMG_2.jpg> from the folder for feature detection and matching. Use the best detection method you identify in step (e).

Here are some examples for reference:

[https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_censure.html]

[https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_orb.html]

[https://scikit-image.org/docs/0.19.x/auto_examples/features_detection/plot_sift.html]

[https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html]