Lab 1: Image Basics and Simple Transformations

(10% of final score, due by Jan. 29 11:59 PM)

In the following tasks, you can use your input images to test your codes. Upload your results and *.*py* files to the folder of **Lab 1** under **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. Search the Web for job openings in *Computer Vision*. List three jobs that you have interests in, along with the qualifications needed to apply. Summarize in your words instead of copying them here.
2. Read Pillow Image Module: https://pillow.readthedocs.io/en/stable/reference/Image.html

   Complete the following operations:

   a. Read a RGB image, convert it to a grey level image, and save it as a PNG file.
   b. Resize the grey level image to half width and half height and show the resized image on screen.
3. Plotting Numpy arrays as images. Set up a 4×5 (rows by columns) array and plot the array as a gray-level 8-bit image, using matplotlib.pyplot [https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html]
4. Suppose an image has 640 columns and 480 rows and is stored in row-major order. Convert the coordinates $(x, y) = (38, 52)$, $(592, 241)$, and $(33, 0)$ to 1D indices. Conversely, convert the following 1D indices to (x, y) coordinates: $i = 8092$, 24061, and 38190.
5. Given the following 8-bit grayscale image,

$$\begin{bmatrix} 176 & 94 & 201 & 219 \\ 37 & 161 & 16 & 88 \\ 71 & 129 & 177 & 81 \\ 41 & 198 & 107 & 19 \end{bmatrix}$$

   a. Perform the following operations by hand: flip, flop, invert, and rotate clockwise by 90 degrees. Write down the new arrays.
   b. Follow the pseudocode provided below to implement the same operations in Python for a grayscale image: flip, flop, invert, and rotate clockwise by 90 degrees.

      Here is the pseudocode for reference:

ALGORITHM 3.1 Flip an image by reflecting about a horizontal axis

FLIPIMAGE (I)

Input: image I of size $width \times height$
Output: upside-down image I'

$I' \leftarrow$ ALLOCATEIMAGE($width$, $height$) ➤ Allocate memory for output image.
for $(x, y) \in I$ **do** ➤ For each pixel in input image,
    $I'(x, height - 1 - y) \leftarrow I(x, y)$ set corresponding pixel in output image.
return I' ➤ Return output image.

ALGORITHM 3.2 Flop an image by reflecting about a vertical axis

FLOPIMAGE (I)

Input: image I of size $width \times height$
Output: mirror-reversed image I'

$I' \leftarrow$ ALLOCATEIMAGE($width$, $height$) ➤ Allocate memory for output image.
for $(x, y) \in I$ **do** ➤ For each pixel in input image,
    $I'(width - 1 - x, y) \leftarrow I(x, y)$ set corresponding pixel in output image.
return I' ➤ Return output image.

ALGORITHM 3.3 Rotate an image by a multiple of 90 degrees

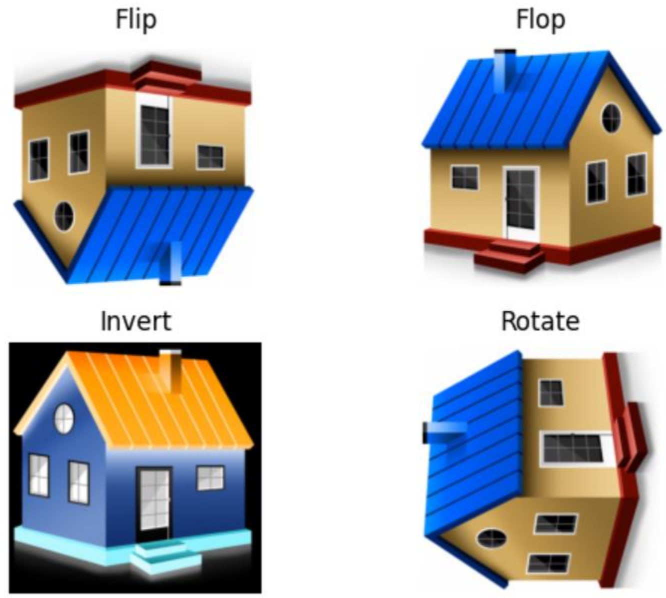
ROTATEIMAGEBYMULTIPLEOF90DEGREES(I, m)

Input: image I of size $width \times height$, signed integer m indicating the number of 90-degree turns
Output: image I' of size $new\text{-}width \times new\text{-}height$, which is I rotated by $90m$ degrees clockwise

case mod(m,2) **of**
    0: $new\text{-}width \leftarrow width$, $new\text{-}height \leftarrow height$ ➤ Image dimensions remain the same.
    1: $new\text{-}width \leftarrow height$, $new\text{-}height \leftarrow width$ ➤ Image dimensions are swapped.
$I' \leftarrow$ ALLOCATEIMAGE($new\text{-}width$, $new\text{-}height$)
for $(x', y') \in I'$ **do**
    case mod(m,4) **of**
        0: $I'(x', y') \leftarrow I(x', y')$ ➤ no rotation
        1: $I'(x', y') \leftarrow I(y', height - 1 - x')$ ➤ 90 degrees clockwise
        2: $I'(x', y') \leftarrow I(width - 1 - x', height - 1 - y')$ ➤ 180 degrees
        3: $I'(x', y') \leftarrow I(width - 1 - y', x')$ ➤ 90 degrees counterclockwise
return I'

c. Modify the algorithm to implement the same operations for a RGB color image, and plot the four outputs (the flip one, the flop one, the invert one, and the rotate one) in a 2×2 subplot array, using Python **Matplotlib** Module. [https://matplotlib.org/tutorials/introductory/sample_plots.html]

The output would look like this (you can use your own color image):



6. Given the following 4-bit image:

$$\begin{bmatrix} 5 & 8 & 3 & 7 \\ 1 & 3 & 3 & 9 \\ 6 & 8 & 2 & 7 \\ 4 & 1 & 0 & 9 \end{bmatrix}$$

   a. Compute the histogram, normalized histogram, and cumulative normalized histogram for the image by hand and provide calculation process.

   b. Follow the pseudocode provided below to implement histogram equalization in Python and run the code on a grayscale image.

   Here is the pseudocode for reference:

ALGORITHM 3.9 Perform histogram equalization on an image

```
HISTOGRAMEQUALIZE(I)

Input:  grayscale image I
Output: histogram-equalized grayscale image I' with increased contrast
h̄ ← COMPUTENORMALIZEDHISTOGRAM(I)
c̄ ← RUNNINGSUM(h̄)
for (x, y) ∈ I do
    I'(x, y) ← ROUND(255 * c̄[I(x, y)])
return I'

RUNNINGSUM(a)

Input:  1D array a of length values
Output: 1D running sum s of array
s[0] ← a[0]
for k ← 1 to length − 1 do
    s[k] ← s[k − 1] + a[k]
return s
```

c. Search Python scikit-image module or openCV module to find a function that implements histogram equalization. Compare your result with their results pixel-by-pixel. Output the comparison in a black-and-white image: black means no difference while white means different.

7. Given the following 3 consecutive frames,

$$I1 = \begin{bmatrix} 18 & 168 & 94 & 67 \\ 120 & 97 & 78 & 198 \\ 83 & 70 & 208 & 17 \\ 238 & 208 & 189 & 68 \end{bmatrix} \quad I2 = \begin{bmatrix} 21 & 168 & 92 & 71 \\ 122 & 71 & 191 & 227 \\ 83 & 212 & 16 & 187 \\ 240 & 216 & 188 & 68 \end{bmatrix} \quad I3 = \begin{bmatrix} 20 & 171 & 92 & 70 \\ 76 & 193 & 39 & 255 \\ 209 & 20 & 20 & 194 \\ 241 & 210 & 190 & 73 \end{bmatrix}$$

a. Compute the double difference and the triple difference between the 3 frames by hand, using the threshold $\tau = 40$;

b. Follow the pseudocode provided below to implement the double difference and the triple difference in Python and run the code on three consecutive frames to detect a moving object. Here are the pseudocodes for reference.

ALGORITHM 3.13 Compute the double difference between three consecutive image frames

FRAMEDIFFERENCEDOUBLE $(I_{t-1}, I_t, I_{t+1}, \tau)$

Input: successive images I_{t-1}, I_t, and I_{t+1}, and threshold τ
Output: binary image indicating the moving regions

for $(x, y) \in I_t$ **do**
  $d_1 \leftarrow |I_{t-1}(x, y) - I_t(x, y)|$
  $d_2 \leftarrow |I_{t+1}(x, y) - I_t(x, y)|$
  $I'(x, y) \leftarrow 1$ **if** $d_1 > \tau$ AND $d_2 > \tau$ **else** 0
return I'

ALGORITHM 3.14 Compute the triple difference between three consecutive image frames

FRAMEDIFFERENCETRIPLE $(I_{t-1}, I_t, I_{t+1}, \tau)$

Input: successive images I_{t-1}, I_t, and I_{t+1}, and threshold τ
Output: binary image indicating the moving regions

for $(x, y) \in I_t$ **do**
  $d_1 \leftarrow |I_{t-1}(x, y) - I_t(x, y)|$
  $d_2 \leftarrow |I_{t+1}(x, y) - I_t(x, y)|$
  $d_3 \leftarrow |I_{t+1}(x, y) - I_{t-1}(x, y)|$
  $I'(x, y) \leftarrow 1$ **if** $d_1 + d_2 - d_3 > \tau$ **else** 0
return I'

8. Given two images with masks:

$$I_A = \begin{bmatrix} 132 & 231 & 227 \\ 237 & 105 & 238 \\ 193 & 59 & 128 \end{bmatrix}, \quad M_A = \begin{bmatrix} 255 & 255 & 0 \\ 255 & 255 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I_B = \begin{bmatrix} 43 & 79 & 116 \\ 56 & 246 & 184 \\ 36 & 119 & 162 \end{bmatrix}, \quad M_B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 255 & 255 \\ 0 & 255 & 255 \end{bmatrix}$$

Compute by hand with the three Porter-Duff operators: I_A OVER I_B, I_A IN I_B, and I_A ATOP I_B. Show only the values of the valid pixels; indicate invalid pixels using an X.

9. Given the following image

$$I = \begin{bmatrix} 232 & 177 & 82 & 7 \\ 241 & 18 & 152 & 140 \\ 156 & 221 & 67 & 3 \end{bmatrix}$$

a. Use bilinear interpolation to compute the value at (0.1, 0.7), (1.2, 0.5), (1.3, 1.6), and (2.8, 1.7) by hand and provide the calculation process.

b. Follow the pseudocode provided below to implement the bilinear interpolation in Python and run the code on the given 4×3 image.

Here is the pseudocode for reference:

ALGORITHM 3.16 Perform bilinear interpolation on an image at a point

InterpolateBilinear(I, x, y)

Input: image, floating-point coordinates (x, y)
Output: weighted average of gray levels of nearest four pixels

$x_0 \leftarrow$ Floor (x)
$y_0 \leftarrow$ Floor (y)
$\alpha_x \leftarrow x - x_0$
$\alpha_y \leftarrow y - y_0$
$\overline{\alpha}_x \leftarrow 1 - \alpha_x$
$\overline{\alpha}_y \leftarrow 1 - \alpha_y$
return $\overline{\alpha}_x * \overline{\alpha}_y * I(x_0, y_0) + \alpha_x * \overline{\alpha}_y * I(x_0+1, y_0) + \overline{\alpha}_x * \alpha_y * I(x_0, y_0+1) + \alpha_x * \alpha_y * I(x_0+1, y_0+1)$