ENGR 13300
Python Project Resource – Matrix and Basic Image Handling

This document contains highlights from the Image Processing Tutorial by Professor Edward Delp. The slides are posted in Brightspace. The links are posted in Brightspace and here:

- Part 1: https://mediaspace.itap.purdue.edu/media/VIP+Image+Processing+Tutorial+-+Part+1/0_i8465vjq
- Part 2:
  - https://mediaspace.itap.purdue.edu/media/VIP+Image+Processing+Tutorial+-+Part+2.1/0_5nxsrtum
  - https://mediaspace.itap.purdue.edu/media/VIP+Image+Processing+Tutorial+-+Part+2.2/0_xwgyff1t

Digital Images:

Before any processing can be performed on an image, a digital representation of it must be obtained. Digital images can be represented by a matrix of numbers. Each element of the matrix is called a picture element or pixel.

In the below black and white image, the image is represented by a matrix of 12 pixels wide by 16 pixels high. In each pixel is a value representing the amount of energy of the pixel. Pixel values will range from 0 to 255, with 0 being black and 255 being white.

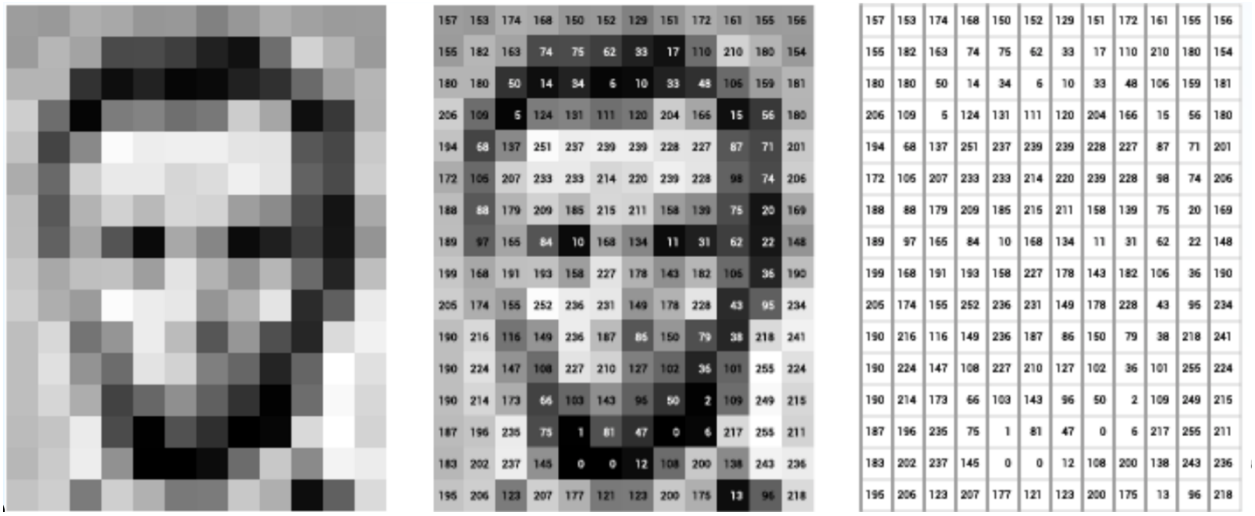


Figure 1. Digital Image (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 2)

A digital image (such as a `.png` file) is a matrix of numbers with a few additional numbers added to the beginning.

Color Images:

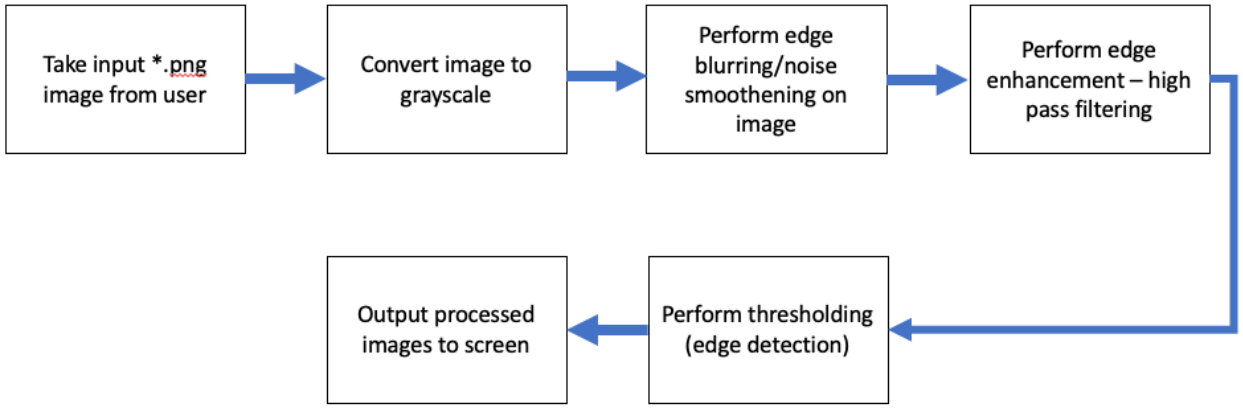
The Human Visual System has rods and cones which are sensitive to light and color. There are three types of cones which are sensitive to a particular color range of light: red, green, and blue. Any color is a

weighted sum of red, green, and blue light. However, there is more than one way to represent the colors, which are known as color spaces. Examples of color spaces are the Red, Green, Blue (RGB) and YUV (luminance/chrominance) color spaces. In contrast to black and white (B&W) images, a color image is represented by multiple matrices, one for each of the components of the color space. For RGB images, there are three matrices: Red, Green, and Blue. Like B&W images, each location in the matrix represents one pixel. When all three matrices are combined, they generate a color in one pixel.

Edge Detection

Image analysis techniques to complete edge detection consist of both pointwise and window operations. For the pointwise operations, you do the same operation to every pixel like converting the image from RGB to grayscale and some thresholding. There are also window operations in which an operation is performed on a small window or square region of size NxN (N is some number of pixels) of the image. These are often filtering operations which can blur, sharpen, and find the edges of an image. The system diagram of the edge detection algorithm to be implemented in this project is shown in Figure 2 below.

Figure 2. Systems Diagram of Edge Detection:



Color to Greyscale Images:

In this project we'll be using grayscale images because they are easier to process. These are also referred to as luminance, Y component, or Y images. Gray scale images typically have 8 bits/pixel which allows for 256 shades of gray to be represented.

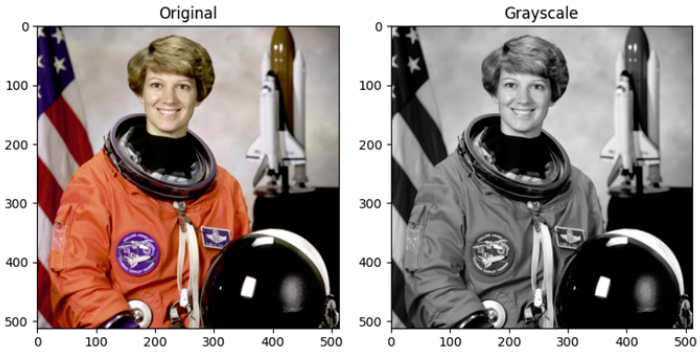


Figure 2. Grayscale Conversion Example of 500x500 pixel image (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 11)

It is important to note that not all color scales contribute equally to an image's luminance or brightness when converting RGB images to grayscale. Green contributes the most to the brightness, where blue contributes the least. Two international standards have been developed for converting color images to grayscale images:

- ITU-R Recommendation BT.709: `Y = 0.2126R + 0.7152G + 0.0722B`
- ITU-R Recommendation BT.601: `Y = 0.299R + 0.587G + 0.114B`

In the above equations Y is the output matrix image, R is the red matrix portion of the image, G is the green matrix portion of the image, and B is the blue matrix portion of the image. Research these standards to determine which method is most appropriate for your project, and why.

Edge Blurring/Noise Smoothing:

Image smoothing (or edge blurring) is completed on images to reduce the noise in the images, which is important to do before the edge enhancement operation. A noisy image can negatively affect the quality of the edge detection algorithm.

The smoothing operation is a window operation. To implement a window operation, you extract a small NxN region from the image, perform the operation on the pixels in the window, and then replace the value of the center pixel of the NxN region with the result. Thus, for the example in Figure 3, a 3x3 region, or Window, is selected centered around the pixel located in row 3, column 3.

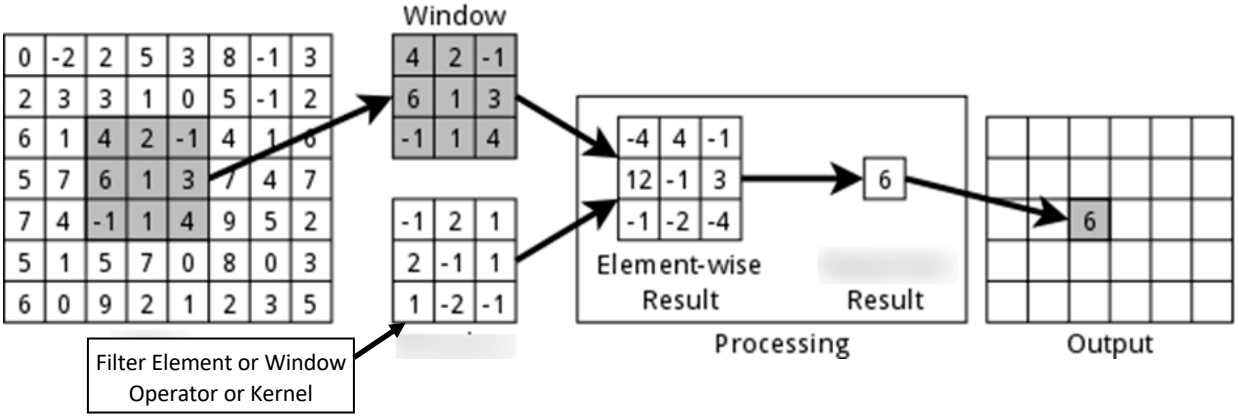


Figure 3. 3x3 Window Operation Example (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 22)

For this example, each pixel from the Window is multiplied with the corresponding value of the window operator (also known as filter element or kernel), as shown in the "Element-wise Result". The values of the element-wise result are then summed, and the pixel of the output image is replaced with that value.

The smoothing (or blurring) operation is done by averaging or weighted filters. An example of an averaging filter using a 3x3 kernel is, where each pixel of the output is computed as the average of the surrounding pixels:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Note, that all weights must sum to one for the smoothing filters, hence the fraction term at the front of the kernel. This is so the kernel sum of the filter values is multiplied by the inverse of the sum. For the Gaussian filter, more weight is given to the central pixels. Below is a 5x5 Gaussian kernel, with Sigma = 1.056.

$$\frac{1}{256}\cdot\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Here is an example of the image of Grace Hopper, an American computer scientist and United States Navy rear admiral, which has been smoothed with a simple averaging filter:

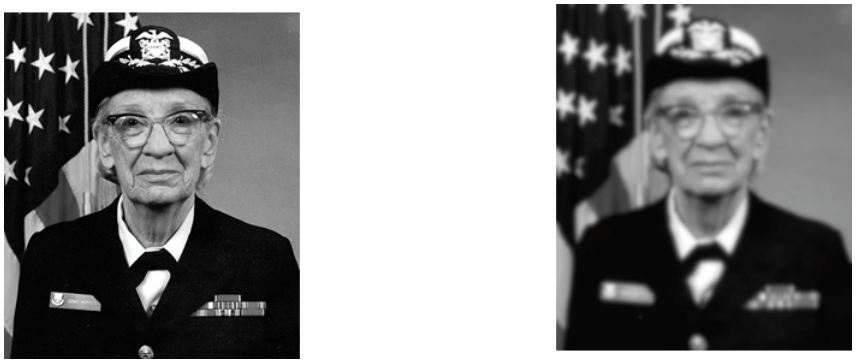


Figure 4. Image Smoothing Example of Grace Hopper (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 23)

Edge Enhancement:

An edge in an image is an abrupt change in the intensity (pixel) values, or intensity gradient, that is perceived as a boundary or change. The edges can be located by taking the derivative of the intensity values across the image, then finding the points where it is a maximum. For example, if we were to plot the pixel values of the image on the left of Figure 5 in the direction of the red line, we would get the plot in the center. Then, if we take a derivative of the center plot, we get to plot on the right.

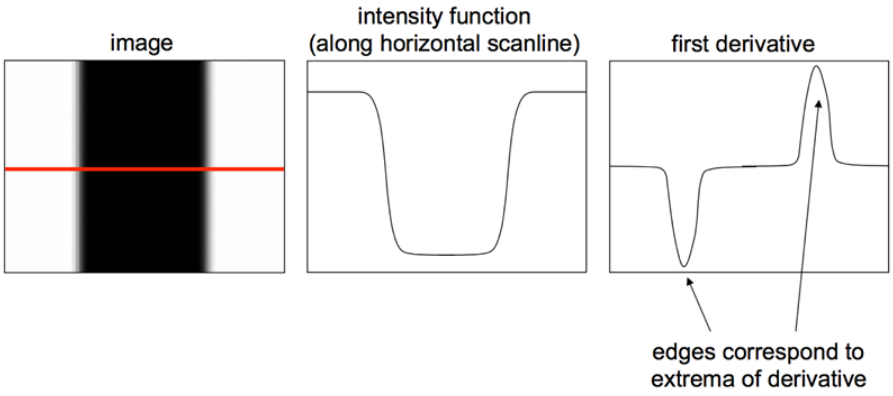


Figure 5. Edges in Images (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 26, also http://stanford.edu/)

The Sobel operator is a discrete differentiation operator that can be used to locate the edges of an image by finding the maximums of the gradients in both the x and y directions separately, then computing the magnitude of the vector. Thus, for an image $f(x, y)$, we want to find the partial derivatives, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$, where the magnitude of the gradient is

$$G = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial xy}\right)^2}$$

This is done by performing window operations using the following kernels:

X – Direction

-1	0	1
-2	0	2
-1	0	1

Y – Direction

-1	-2	-1
0	0	0
1	2	1

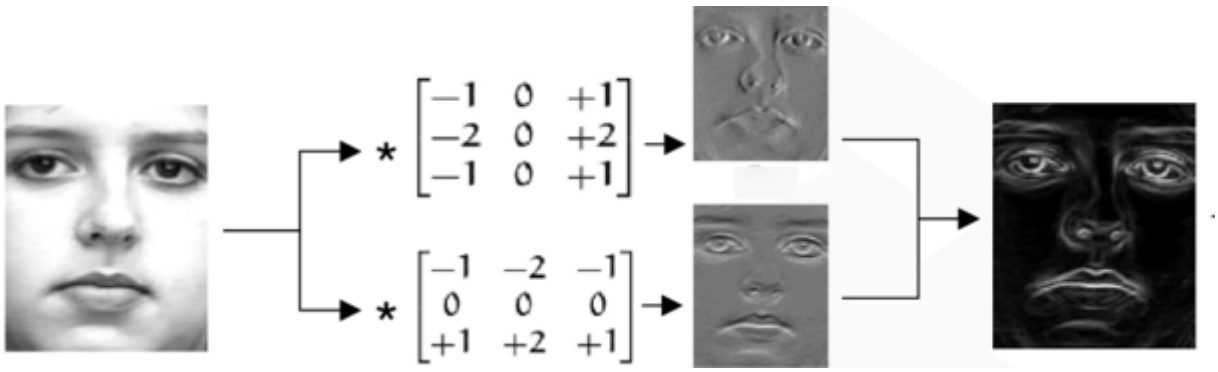


Figure 6. Sobel Edge Enhancement (Source: Image Processing and Analysis Tutorial – Part II, Edward J. Delp, 8/29/2020, slide 33)

It is important to be cognizant of the data type of the images, particularly when doing the derivatives. If the data type is "uint8" (unsigned integer), negative values will not be properly stored, and thus the results will not be correct.

Thresholding

The final step is to further define the edges by only displaying those above a certain threshold value as 255 (white) and below a certain value as black (0). You may invert the white and black values if you wish. Experiment with different algorithms for determining the threshold value to use for your project. You will need to provide evidence for your design decision in your project report.

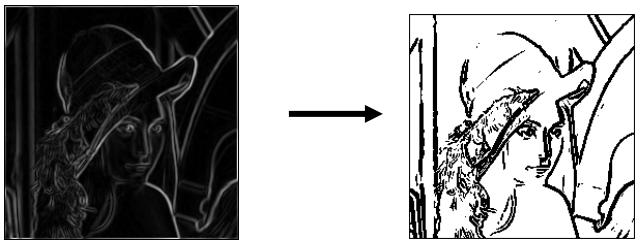
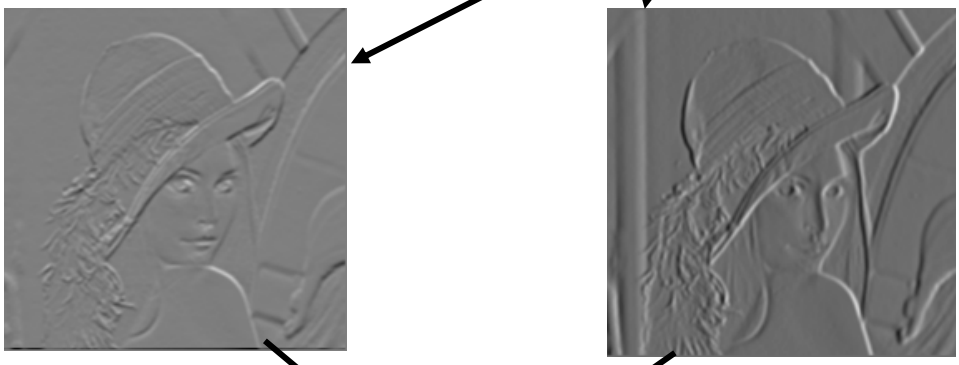


Figure 7. Thresholding Example

Here is an example of the entire edge detection process for the Lena/Lenna png image:

Original **Grayscale** **Smoothed (Gaussian)**

Sobel – x direction **Sobel – y direction**



Magnitude of Gradient

Threshold

Working with Matrices in Python: https://www.programiz.com/python-programming/matrix

Useful Libraries:

- matplotlib
- numpy

Python does not have a built-in type for matrices, but a matrix can be expressed as a list of a list.

Example: 3 x 4 Matrix

4 columns

$$\begin{bmatrix} 127 & 126 & 80 & 60 \\ 120 & 110 & 65 & 50 \\ 100 & 80 & 50 & 30 \end{bmatrix}$$

3 rows

In Python:

```
A = [ [127, 126, 80, 60],
      [120, 110, 65, 50],
      [100, 80, 50, 30] ]
```

To transpose this matrix, we would need to use nested loops:

```
A_t = [[0,0,0,0],
       [0,0,0,0],
       [0,0,0,0]]
# iterate through rows
for i in range(len(A)):
   # iterate through columns
   for j in range(len(A[0])):
       A_t[j][i] = A[i][j]
```

However, it is much easier to perform matrix operations using NumPy N-dimensional array objects:

```
import numpy as np
A = np.array([[127,126,80,60],[ 120,110,65,50],[ 100,80,50,30]])

#To access rows
print(f'A[0] = {A[0]}') # First row
print(f'A[2] = {A[2]}') # Third row

# To access columns
print(f'A[:,0] = {A[:,0]}') # First column
print(f'A[:,3] = {A[:,3]}') # Fourth column


# Transpose using numpy
A_t = A.transpose()
print(A_t)
```

Will output:

```
A[0] =  [127 126 80 60]
A[2] =  [100 80 50 30]
A[:,0] =  [127 120 100]
```

```
A[:,3] =  [60 50 30]
[[127 120 100]
 [126 110  80]
 [ 80  65  50]
 [ 60  50  30]]
```

Reading Image files:

In order to work with images, the image has to be first read by your program as a matrix of Blue Green and Red pixel values. To read an image located in the same directory as your program is located, you may use the following syntax,

1. With matplotlib:

```
import matplotlib.pyplot as plt
image = plt.imread('Sample.jpg')
```

Showing Image files:

In order to display image files to the user,

1. With matplotlib: The following snippet of code will display the image in Spyder console:

```
import matplotlib.pyplot as plt
#Read and Process image file
plt.imshow(image)

#Read and Process image file using "grey" color map
plt.imshow(image, cmap='gray')
```

Writing Image files:

To write an image back in the same directory as your program is located, you may use the following syntax,

1. With matplotlib:

```
import matplotlib.pyplot as plt
#Read and Process image file
plt.imsave('Sample_copy.jpg',image)

#Read and Process image file using "grey" color map
plt.imsave('Sample_copy.jpg', image, cmap='gray')
```

Important Notes:

- For reading or writing images located in different folder, you will have to use the complete path to the image (ex. `'C:\...\Sample.jpg'`)

- The image arrays are stored as an array of pixel data, where each (x,y) coordinate has three values associated to brightness of Red, Blue and Green (RBG) colors. The brightness range goes from 0 being dark, to 255 being bright, and values have to strictly be integers. **However, Matplotlib rescales the 8 bit data from each channel to floating point data between 0.0 and 1.0.**
- Although different libraries share similar syntax and datatypes, and thus gives the freedom to use functions of different libraries interchangeably, use caution as image standards used may differ. For instance,
    - matplotlib stores images in Numpy Array, whereas other tools do not. As noted above, it also rescales the 8 bit values to floating point data.
    - matplotlib stores each pixel value as RGB, while OpenCV, a popular image processing library, stores pixels as BGR.

Sample programs:

1. Read an image, display its dimension and show the image

```
import matplotlib.pyplot as plt

image = plt.imread('sample.jpg')
print(f'Dimensions of the image is {image.shape}')
plt.imshow(image)
```