Image Processing

OGUNLEYE OLAOLUWA

Matrix Theory, Clarkson University, NY

13th December, 2021

List of Figures

Contents

Abstract

This project report tends to summarize and explain the relationship between Image Processing (a vibrant field of computer science) and Matrix Theory (a core foundation of mathematics).

Chapter 1

INTRODUCTION

"A visual image in the hand of an artist is merely a tool to trigger a mental image."

Roy H Williams

"Images attempt to capture scientific thought. They represent pyhsical manifestation of the thought process."

Peter Fraser

1.1 Background

Digital Image Processing means processing digital image by means of a digital computer. We can also say that it is a use of computer algorithms, in order to get enhanced image or extract some useful information.

Image processing mainly include the following steps:

- Importing the image via image acquisition tools;

- Analysing and manipulating the image;
- Output in which result can be altered image or a report which is based on analysing that image.

1.1.1 What is an image?

An image is defined as a two-dimensional function, $F(x, y)$, where x and y are spatial co-ordinates, and the amplitude of F at any pair of coordinates (x, y) is called the intensity of that image at that point. When x, y, and amplitude values of F are finite, we call it a digital image.

In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns. Digital Image is composed of a finite number of elements, each of which elements have a particular value at a particular location. These elements are referred to as picture elements, image elements, and pixels. A Pixel is most widely used to denote the elements of a Digital Image.

1.1.2 Types of an image

- **BINARY IMAGE** – The binary image as its name suggests, contain only two pixel elements i.e 0&1, where 0 refers to black and 1 refers to white. This image is also known as *Monochrome*.
- **8 bit COLOR FORMAT** – It is the most famous image format.It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0

stands for Black, and 255 stands for white, and 127 stands for gray.

- **16 bit COLOR FORMAT** – It is a color image format. It has $65,536$ different colors in it. It is also known as *High Color Format*. In this format the distribution of color is not as same as Grayscale image.

Remark 1.1.1. A 16 bit format is actually divided into three further formats which are Red, Green and Blue. The famous RGB format.

Chapter 2

Image as a Matrix

As we know, images are represented in rows and columns we have the following syntax in which images are represented:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

Figure 2.1: Image as Matrix

The right side of this equation is digital image by definition. Every element of this matrix is called image element, picture element, or pixel.

2.1 Digital Image Representation In MATLAB:

$$f = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix}$$

Figure 2.2: Image in MATLAB

In MATLAB the start index is from 1 instead of 0. Therefore, $f(1,1) = f(0,0)$. henceforth the two representation of image are identical, except for the shift in origin. In MATLAB, matrices are stored in a variable i.e A, X, input_image, and so on. The variables must be a letter just like in other programming languages.

2.2 Phases of Image Processing

- **ACQUISITION** – It could be as simple as being given an image which is in digital form. The main work involves:
  - Scaling
  - Color conversion(RGB to Gray or vice-versa)
- **IMAGE ENHANCEMENT** – It is amongst the simplest and most appealing in areas of Image Processing it is also used to extract some hidden details from an image and is subjective.
- **IMAGE RESTORATION** – It is the operation of taking a corrupt/noisy image and estimating the clean, original image. Corruption may come in many forms such

as motion blur, noise and camera mis-focus. It involves the use of mathematical and probabilistic model.

- **IMAGE COMPRESSION** - is a type of data compression applied to digital images, to reduce their cost for storage or transmission. Algorithms may take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data. It mainly deals with image size or resolution.

- **IMAGE SEGMENTATION** - include an intersection of two fields - digital image processing and computer vision, and involves the process of partitioning an image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. It includes partitioning an image into its constituent parts or objects.

- **OBJECT DETECTION AND RECOGNITION** - is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. It is a process that assigns a label to an object based on its descriptor.

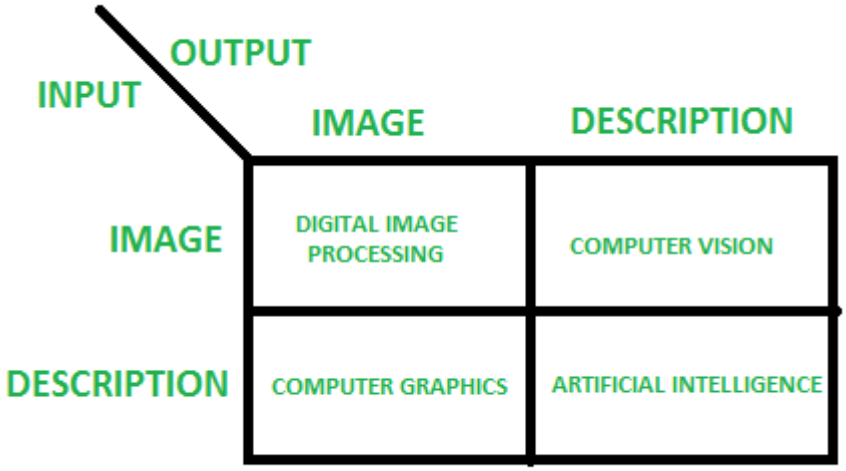


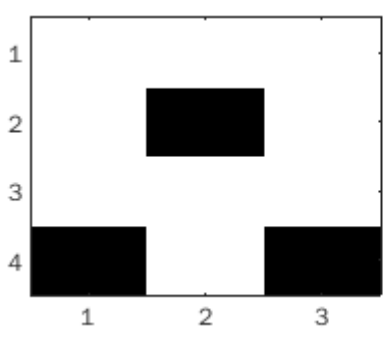
Figure 2.3: Overlapping Fields

2.3 Matrix representations:

- A grayscale image is a 2-dimensional array of numbers. An 8-bit image has entries between 0 and 255.
- The value 255 represents a white color, and the value 0 represents a black color.
- Lower numbers translate to darker pixels, while higher numbers translate to lighter pixels.
- For an image that has $m \times n$ pixels (i.e., picture elements), we represent that image using a matrix of size $m \times n$. The entries of the matrix indicate the pixel value of the corresponding part of the image.
- Color images can be stored in a similar fashion to a grayscale image. Instead of one number $(0 - 255)$ per pixel, one stores three numbers per pixel – these three numbers

denote the "amount" of red, "amount" of green, and "amount" of blue in each pixel. These three numbers can be used to depict a wide range of colors.

Example 2.3.1. Write the matrix that represents the following image:

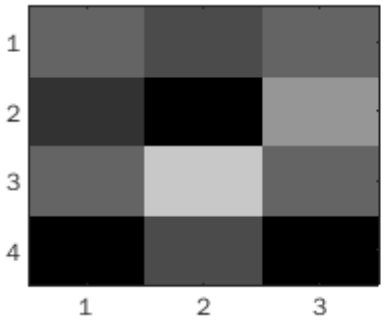



```
A =

   255   255   255
   255     0   255
   255   255   255
     0   255     0
```

Figure 2.4: Example 1

Example 2.3.2. Write the matrix that represents the following image:




```
A =

   100    75   100
    50     0   150
   100   200   100
     0    75     0
```

Figure 2.5: Example 2

2.4 Image transformations:

Once images are represented as matrices, we can describe many transformations of those images using basic matrix operations!

- **Matrix Sum:**
  - Adding the image in Example 2.3.1 and 2.3.2:

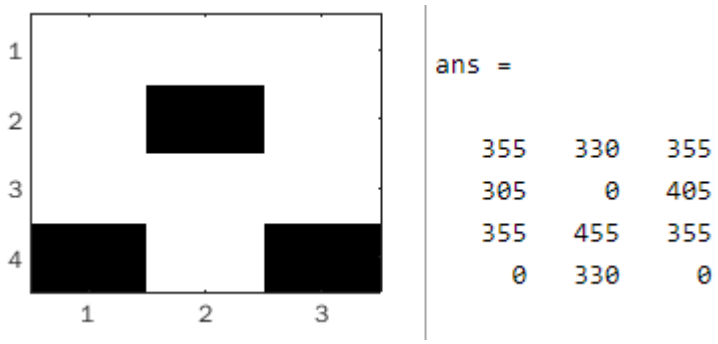


Figure 2.6: Sum of Images I

We observe that entries greater that 255 are regarded as white.

  - Subtracting image in Example 2.3.1 from 2.3.2:

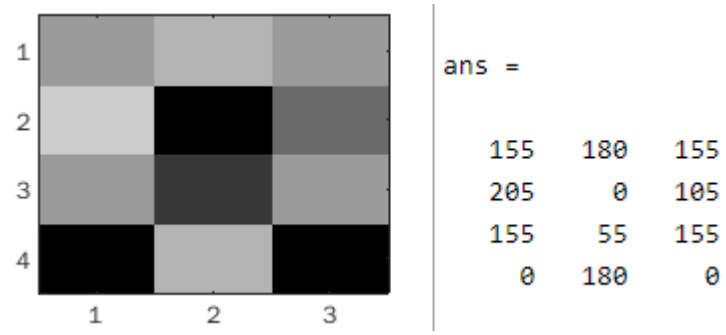


Figure 2.7: Sum of Images II

– Subtracting image in Example 2.3.2 from 2.3.1:

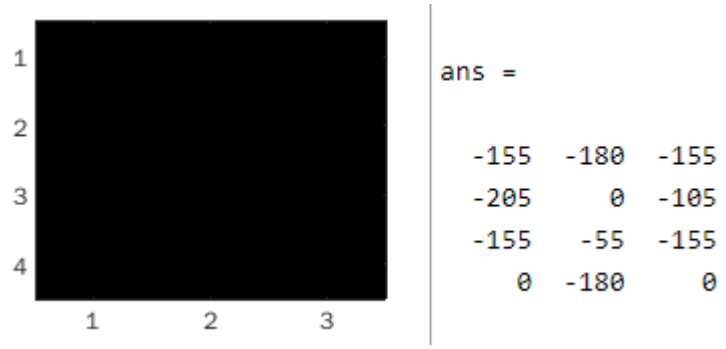


Figure 2.8: Sum of Images III

We observe that entries lesser than 0 are regarded as black.

- **Scalar Multiplication:** We can multiply image matrix by scalar to get a "new" image, and this process forms the basis of **image blurring**.

  For instance, $\frac{1}{5}A$ where is from Example 2.3.2 gives:

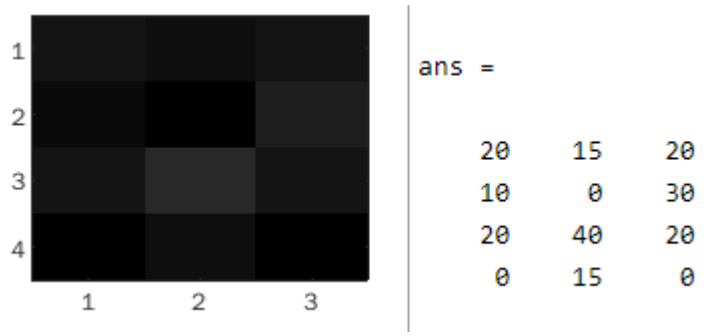


Figure 2.9: Sum of Images IV

Here, we observe different shades of black.

Remark 2.4.1. When mathematical operations on images result in entries that range outside of the integer values 0 through 255, image visualization tools treat them as 255 if the value exceeds 255, and as 0 if the value is below 0

Chapter 3

Manipulating images

3.1 Reflecting/rotating images

Let us consider how we might reflect an image over the x or y-axis. Here's an example!

Figure 3.1: Normal me

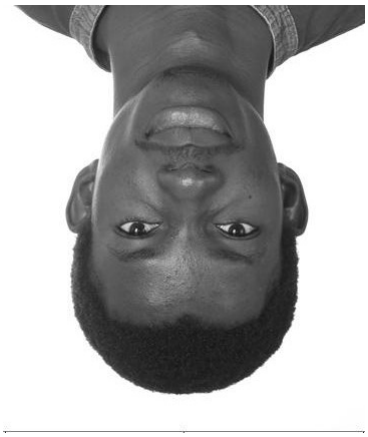
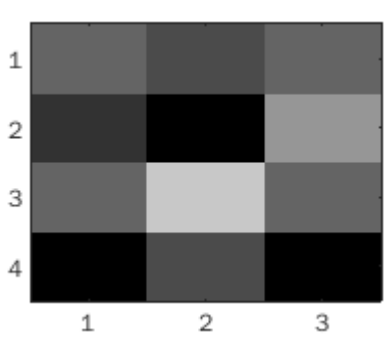


Figure 3.2: Reflecting myself on x-axis

Figure 3.3: Reflecting myself on y-axis

We consider the representation of reflections using matrices multiplication.

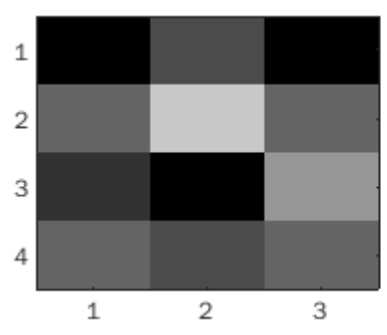
Let's us consider the matrix representation of image in Example 2:




```
A =

   100    75   100
    50     0   150
   100   200   100
     0    75     0
```

Reflecting this image about the x-axis i.e an horizontal flip, produces:

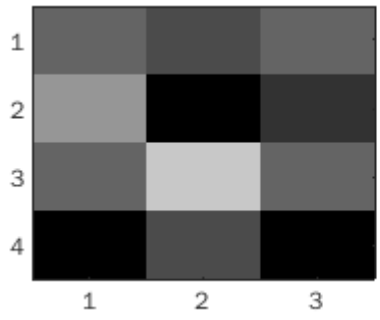



```
ans =

     0    75     0
   100   200   100
    50     0   150
   100    75   100
```

Figure 3.4: Reflecting about x-axis

Reflecting this image about the y-axis i.e an horizontal flip, produces:




```
ans =

   100    75   100
   150     0    50
   100   200   100
     0    75     0
```

Figure 3.5: Reflecting about y-axis

3.2 Reflecting images using matrix-matrix multiplication

Theoretically, or rather more mathematically, the reflection the above matrix A in Example 2 along the x-axis and y-axis is the matrix multiplication of a special kind of **permutation matrix P** with A from the right and left respectively.

We recall the definition of permutation matrix:

Definition 3.2.1. A permutation matrix **P** is a square binary matrix that has exactly one entry of 1 in each row and each column and $0's$ elsewhere.

Furthermore, each of such matrix, say P, represents a permutation of m elements and, when used to multiply another matrix, say A, results in permuting the *rows* (when pre-multiplying, to form PA) or *columns* (when post-multiplying, to form AP) of the matrix A.

This permutation matrix P is crucial in obtaining reflection. In particular, for a given $m \times n$ matrix image A, P is defined as:

- P is an $m \times m$ "reverse" identity matrix while reflecting about the x-axis. In Example 2,

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{x-axis Reflection} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 100 & 75 & 100 \\ 50 & 0 & 150 \\ 100 & 200 & 100 \\ 0 & 75 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 75 & 0 \\ 100 & 200 & 100 \\ 50 & 0 & 150 \\ 100 & 75 & 100 \end{bmatrix}$$

- P is an $n \times n$ "reverse" identity matrix while reflecting about the y-axis

  In Example 2,

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\text{y-axis Reflection} = \begin{bmatrix} 100 & 75 & 100 \\ 50 & 0 & 150 \\ 100 & 200 & 100 \\ 0 & 75 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 100 & 75 & 100 \\ 150 & 0 & 50 \\ 100 & 200 & 100 \\ 0 & 75 & 0 \end{bmatrix}$$

Chapter 4

Takeaway!

4.1 Looking Ahead!

I am personally thrilled at the infinite wealth of research that exists in the application of matrix theory to Image Processing, Computer Vision and Machine Learning. Seeing how matrix norms approximation form the foundation of the convergence of many fixed point iterations and recursion in computer science, as in Markov Decision Process and Unsupervised Learning, is absolutely beautiful and this at it's core is a basis of my most recent admiration of mathematics and computer science.

I appreciate the knowledge gained during this class, thanks Professor!

4.2 Code in MATLAB & Python

The codes are attached in the following pages.

Mathlab Codes:

```
% Example 1
A = [255 255 255;
255 0 255;
255 255 255;
0 255 0];
imagesc(A,[0 255]); colormap(gray);
% Example 2
B = [100 75 100;
50 0 150;
100 200 100;
0 75 0];
imagesc(B,[0 255]); colormap(gray);
A-B;
imagesc(A-B,[0 255]); colormap(gray);
(1/5)*B;
imagesc((1/5)*B,[0 255]); colormap(gray);
% horizontal flip
P= [0 0 0 1;
   0 0 1 0;
   0 1 0 0;
   1 0 0 0
];
P*B;
imagesc(P*B,[0 255]); colormap(gray);
% Vertical Flip
P= [0 0 1;
   0 1 0;
   1 0 0;
];
B*P
imagesc(B*P,[0 255]); colormap(gray);
% Example 1
I = [150 255 255;
255 0 255;
255 255 255;
0 255 0
   ]

imagesc(I,[0 255]); colormap(gray);
% Image Sum
A = [0 0; 255 0]
imagesc(A,[0 255]);
colormap(gray);
B =[255 0; 0 255]
imagesc(B,[0 255]); colormap(gray);
imagesc(A+B,[0 255]); colormap(gray);
imagesc(0.2*B,[0 255]); colormap(gray);
```

```
% Reflectors
C = [255 255 255;
255 0 255;
255 255 255;
0 255 0
   ]
P = [0 0 0 1;
   0 0 1 0;
   0 1 0 0;
   1 0 0 0]
P * C           % x-axis
P = [ 0 0 1;
    0 1 0;
    1 0 0;
   ]
C * P           % y- axis
```

Python Codes:

```
import numpy as np
import copy
from PIL import Image


# get image matrix A
imgGray = Image.open('initPhoto.jpg').convert('L')
imgGray.show()

imgArray = np.array(imgGray)


# Generate P x-axis    427 * 427, PA
P1=[[0 for  i in range(427)] for i in range(427)]

for i in range(427):
    P1[i][426-i] = 1
P1 = np.array(P1)
```

```
    # reflect = PA
xReflectArr = np.matmul(P1, imgArray)

# show reflected image
xReflectPhoto = Image.fromarray(xReflectArr)
xReflectPhoto.show()



# generate P y-axis     361 * 361
P2=[[0 for  i in range(361)] for i in range(361)]

for i in range(361):
    P2[i][360-i] = 1
P2 = np.array(P2)

    # reflect = AP
yReflectArr = np.matmul(imgArray, P2)

# show reflected image
yReflectPhoto = Image.fromarray(yReflectArr)
yReflectPhoto.show()


# Over and out!
```

4.3 References

[1] https://www.geeksforgeeks.org/digital-image-processing-basics/.

[2] https://asaibab.math.ncsu.edu/Module2ImageProcessing.pdf.

[3] https://en.wikipedia.org/wiki/Permutation_matrix.

[4] https://en.wikipedia.org/wiki/Image_restoration

[5] https://en.wikipedia.org/wiki/Image_compression

[6] https://medium.com/swlh/applications-of-linear-algebra-in-image-filters-part-i-operations-aeb64f236845