Nick Kamper and Eric Henderson

CSSE463 – Image Recognition

Lab 3

# Matrix Multiplication

Typing in A \* B results in the matrix [19 22; 43 50].  This is because the \* operator (without a period before it) performs matrix multiplication.  On the other hand, A .\* B results in the matrix [5 12; 21 32] because it is performing element-wise multiplication.  Prefixing an operator (such as **\*** or **/** or **^**) with a period will cause it to perform an element-wise operation instead of the matrix version.

# Gaussian Approximation

gfilter = [0.01, 0.02, 0.04, 0.02, 0.01; 0.02, 0.04, 0.08, 0.04, 0.02; 0.04, 0.08, 0.16, 0.08, 0.04; 0.02, 0.04, 0.08, 0.04, 0.02; 0.01, 0.02, 0.04, 0.02, 0.01];

While it is still somewhat similar to a normal curve, it does not place as much emphasis on the center point (0.16 versus 0.6187) and is a more gentle normal curve. We created it by taking the array [0.1, 0.2, 0.4, 0.2, 0.1] (which seemed similar enough to a normal curve and added up to 1) and multiplying it by its transpose to obtain a matrix. The resulting matrix added up to 1 without requiring any adjustment. If this guessed filter were used to blur an image, it would blur it to a greater degree than the real gaussian filter.

# Our Sobel Edgel Detector

We implemented a rudimentary Sobel edgel detector using the 3x3 Sobel filters given on the lecture slides:

Figure – Original Image

We chose a test image that was relatively ‘edgy’, making it easy to determine if the detector was working properly.

First, we applied the horizontal and vertical Sobel filters, giving us horizontal and vertical edges in the image.

We then added the two resulting matrices together, giving us pseudo-bidirectional edges.

We took the gradient, giving us magnitudes and directions. Finally, we used the magnitude data to filter out low-magnitude edges from the directions image, giving us a final image.

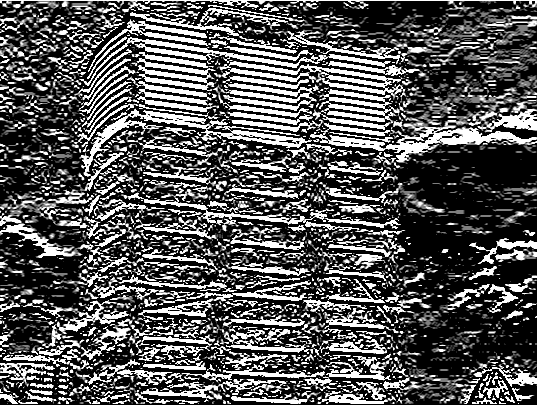
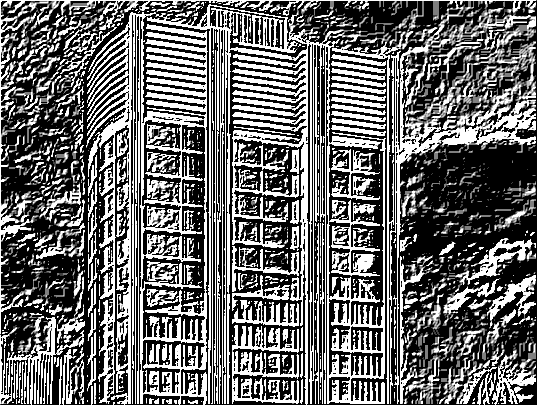
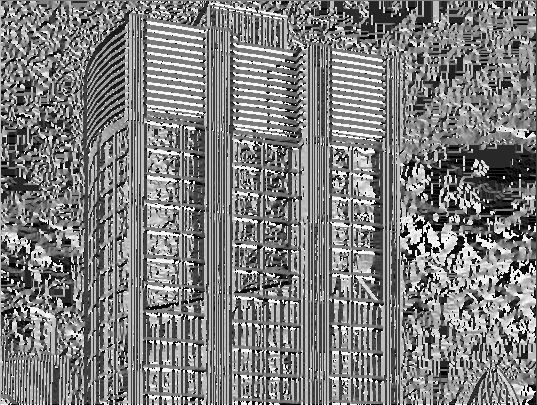


Figure 7 – Filtered Directional Magnitudes

Figure 6 – Gradient Magnitudes

Figure 5 – Gradient Directions

Figure 4 – Bidirectional Edges

Figure – Vertical Edges

Figure – Horizontal Edges

MATLAB Quick Reference

# Syntax

## Suppressing Output

To suppress output of a function call in MATLAB, end the statement with a semicolon. This is especially useful when opening an image or other large file!

# Arrays and Matrices

## Definition

To define an array, or a 1D matrix, one can use the simple [1 2 3 4] notation, like such:

>> A = [1 2 3 4]

A =

1 2 3 4

Multi-dimensional matrices can be defined two ways: similar to the 1D matrix, which is useful for 2D matrices, or piecemeal, useful for 3 or more dimensions. A 2D matrix can be defined like such:

>> B = [1 2; 3 4]

B =

1 2

3 4

Higher dimension arrays can be defined like such:

>> C(:, :, 1) = [1 2; 3 4];

>> C(:, :, 2) = [5 6; 7 8];

>> C

C(:,:,1) =

1 2

3 4

C(:,:,2) =

5 6

7 8

## Subscripting

Reminder: MATLAB is one-indexed!

As you may have noticed in defining a multi-dimensional matrix, you can access the rows, columns, depths, and more, of a matrix using slicing. If you specify a colon, it will include all possible values for that particular entry – just like how B(:, 2) will give the values for the second column.

## Flexible Subscripting

In addition to slicing, you can also specify an array of indices to get an array of the values for those indices. For example:

>> A = [5 6 3 4];

>> A([1 2])

ans =

5 6

## Operations on Matrices

You can add and divide matrices, just as you would expect. For instance,

>> A = [1 2 3];

>> B = [4 5 6];

>> C = A + B

C =

5 7 9

Multiplication is a bit tricky. MATLAB allows you to do scalar multiplication, as you would expect:

>> D = 3\*A

D =

3 6 9

If you try to multiple two matrices using the \* operator, it will perform matrix multiplication. If you want to perform element-wise multiplication, use the .\* operator.

## Finding

To find a set of values in a matrix that satisfy a certain predicate, use the find(predicate) command (e.g., find(a == 3) would give you the indices of a where that element is equal to 3). You can combine this with flexible subscripting to set the values (e.g., a(find(a < 0.5)) = 0 would be a roundabout way to round all values below 0.5 to 0).

# Image Handling

## Opening and Saving Images

To open an image in MATLAB, you can use the imread function, providing it the path to the image:

>> img = imread(‘images/some\_image.jpg’);

>>

Likewise, you can save an image using imsave:

>> imsave(img, ‘images/another\_image.png’);

## Manipulating Images

Images loaded by imread are actually represented in MATLAB as a three-dimensional matrix, where img(:,:,1) is the red channel, img(:,:,2) is the green channel, and img(:,:,3) is the blue channel.

## Converting between color spaces

To convert from RGB to HSV color spaces, use MATLAB’s rgb2hsv(). To convert from HSV to RGB, use MATLAB’s hsv2rgb().

# Functions and Scripts

## Functions in MATLAB

### Defining Functions

Functions in MATLAB can be defined using the following syntax:

function [a, b, c] = functionName(param1, param2)

where a, b, c are return values and param1 and param2 are parameters to the function.

### Returning Values

MATLAB doesn’t have an explicit “return” statement for returning values. Instead, you will assign values to the return value names you defined in the function definition and the last values assigned when the function exits will be the return values.

### Saving Functions

MATLAB doesn’t allow for multiple exposed functions to be in the same file. Each file should contain one exposed method. However, it may contain as many private helper functions as you want. The file should be named “function.m” where function is the method you want exposed.

## Scripts in MATLAB

A script in MATLAB is simply a file containing MATLAB commands and operators. These files can be executed in MATLAB by using the filename without extension. These scripts should be saved with “script.m” filename where script is whatever name you want for the script.