Image Manipulation Using Matrix Techniques[1]

Names	Student ID	Professor	TA	Recitation
Will Farmer	101446930	Mimi Dai	Amrik Sen	608
Jeffrey Milhorn	100556107	Kevin Manley	Ed Yasutake	605
Patrick Harrington	100411000	Mimi Dai	Ash Same	618

Friday, March 22

[1] Report LaTeXSource Code is attached.

Contents

List of Figures

List of Equations

Introduction

Since images stored on computers are simply matrices where each element represents a pixel, matrix methods learned in class can be used to modify images. The purpose of this project was to apply matrix manipulations on given image files, shown below as Figure 1a and Figure 1b.

(a) Photo 1

(b) Photo 2

Figure 1: Provided Images

1 Reading Image Files & Grayscale Conversion

Colored images have an interesting, although problematic property; they do not readily lend themselves to matrix manipulation because in order to get color images, seperate values are used to represent each primary color, which are then mixed together for the final color. For example, in Figure 2, the block represents very simple a 2×2 pixel image.

This very simple image can be represented as either a trio of primary color matrices where each entry in each primary color matrix coresponds to the same pixel:

$$\underbrace{\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}}_{\text{Red Matrix}}, \underbrace{\begin{bmatrix}0 & 0\\ 1 & 1\end{bmatrix}}_{\text{Blue Matrix}}, \underbrace{\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}}_{\text{Green Matrix}}$$

A single matrix may be used, with each entry being a submatrix, wherein each element in the submatrix corresponds to a primary color.

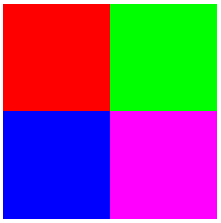

Figure 2: A simple RGB image

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

Using one of the given images, the splitting of color channels gives the following set of images shown in Figure 3.

Figure 3: A given image split into its three primary color channels

While it is possible to manipulate color images, it would be far simpler to manipulate *grayscale* images, where only the final intensity is concerned. To do this, each color is considered independently for its intensity alone as shown in Figure 4, where it may be scaled, and then added together to produce a final black-and-white image, which is a matrix where each entry is a single value. Note how the third panel representing the blue color channel is darker – this implies that blue is a less intense color in the image.

Since each primary color is freely editable, it is simple to scale the intensity of each before mixing; in our report, we used 30% of the red channel, 59% of the green channel and 11% of the blue channel. The final outputs for both given images can be seen in Figure 5. Note how the final output is lighter than any of the individual color channels.

Figure 4: A given image split into its three primary color channels, but only intensity of each color is shown.

2 Horizontal Shifting

Now that we are working in grayscale, it is far more straightforward to manipulate aspects of the image, such as its horizontal position. Since we are dealing with a normal matrix, transforming the positions of columns requires only that we multiply the image matrix by a transformation identity matrix.

As discussed in the lab instructions, to shift an image horizontally without losing information requires the use of a transformation matrix as shown below.

$$\underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}}_{\text{Identity Matrix}} \Longrightarrow \underbrace{\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}}_{\text{Transformation Matrix}}$$

$$\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix} \cdot \begin{bmatrix}a & b & c\\ d & e & f\\ g & h & i\end{bmatrix} = \underbrace{\begin{bmatrix}c & a & b\\ f & d & e\\ i & g & h\end{bmatrix}}_{\text{the horizontally shifted matrix}}$$

3 Vertical Shifting

Very similar to the horizontal position change, the vertical position change merely requires the transformation matrix to be shifted row-wise as opposed to column-wise.

$$\underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}}_{\text{Identity Matrix}} \Longrightarrow \underbrace{\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}}_{\text{Transformation Matrix}}$$

$$\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix} \cdot \begin{bmatrix}a & b & c\\ d & e & f\\ g & h & i\end{bmatrix} = \underbrace{\begin{bmatrix}c & a & b\\ f & d & e\\ i & g & h\end{bmatrix}}_{\text{the horizontally shifted matrix}}$$

(a) Photo 1 - Grayscale

(b) Photo 2 - Grayscale

Figure 5: Grayscale Images

4 Inversion

5 Code

The entire codebase for the project follows, and is available for download here.

5.1 Python

The Python code to generate the images is included below.

```
1  #!/usr/bin/env python
2  '''
3  APPM 2360 Differential Equations Project Two
4    |-Will Farmer
5    |-Jeffrey Milhorn
6    |-Patrick Harrington
7
8  This code takes the two given images and performs several
9  mathematical operations on them using matrix methods.
10 '''
11
```

(a) Photo 1 Horizontal Shift

(b) Photo 2 - Horizontal Shift

Figure 6: Horizontally Shifted Images

```
12 import sys                          # Import system library
13 import scipy.misc                   # Import image processing libraries
14 import numpy                        # Import matrix libraries
15 import matplotlib.pyplot as plt     # Import plotting libraries
16 import pp                           # Library for Parallel Processing
17
18 jobServer = pp.Server() # Create a new jobserver
19 jobs      = []          # List of jobs to complete
20
21 def main():
22     # Open images for manipulation
23     print('Opening Images')
24     image1 = scipy.misc.imread('../img/photo1.jpg')
25     image2 = scipy.misc.imread('../img/photo2.jpg')
26
27     # Run manipulations on both images
28     print('Generating Manipulations')
29     manipulate(image1, '1')
30     manipulate(image2, '2')
31
32     # Visualize Determinants of DST Matrix
```

(a) Photo 1 Horizontal Shift

(b) Photo 2 - Horizontal Shift

Figure 7: Horizontally Shifted Images

```
    print('Generating Determinant Graph')
    visualize_s()

    # Compress images using DST
    print('Compressing Images')
    jobs.append(
            jobServer.submit(compression,
                             (image1, '1', 0.5),
                             (create_grayscale, dst, create_S),
                             ('numpy', 'scipy.misc'))
            ) # Add a new job to compress our first image
    jobs.append(
            jobServer.submit(compression,
                             (image2, '2', 0.5),
                             (create_grayscale, dst, create_S),
                             ('numpy', 'scipy.misc'))
            ) # Add a new job to compress our second image

    # Analyze Compression Effectiveness
    print('Generating Compression Effectiveness')
    comp_effect(image1, image2)
```

(a) Photo 1 - Vertical and Horizontal Shift

(b) Photo 2 - Vertital and Horizontal Shift

Figure 8: Vertically Shifted Images

```
54
55      # Create Picture Grid
56      print('Generating Picture Grid')
57      mass_pics(image1, '1')
58      mass_pics(image2, '2')
59
60      for job in jobs:
61          job() # Evaulate all current jobs
62
63  def manipulate(image, name):
64      '''
65      Manipulate images as directed
66      1) Create grayscale image
67      2) Produce horizontal shifts
68      3) Produce Vertical/Horizontal Shifts
69      4) Flip image vertically
70      '''
71      # Create grayscale
72      g = create_grayscale(image.copy())
73      scipy.misc.imsave('../img/gray%s.png' %name, g)
74
```

```
75      # Shift Horizontally
76      hs = shift_hort(g)
77      scipy.misc.imsave('../img/hsg%s.png' %name, hs)
78
79      # Shift Hort/Vert
80      hs = shift_hort(g)
81      vhs = shift_vert(hs.copy())
82      scipy.misc.imsave('../img/vhsg%s.png' %name, vhs)
83
84      # Flip
85      flipped = flip(g)
86      scipy.misc.imsave('../img/flip%s.png' %name, flipped)
87
88  def flip(image):
89      '''
90      flips an image
91      '''
92      t  = numpy.transpose(image) # creates a transpose
93      il = numpy.identity(len(image)).tolist()  # Creates a matchting identity
94      for row in il: # Reverses the identity matrix
95          row.reverse()
96      i       = numpy.array(il) # Turns it into a formal array
97      flipped = numpy.transpose(numpy.dot(t, i))
98      return flipped # Returns transpose of t.i
99
100 def shift_hort(image):
101     '''
102     Shift an image horizontally
103     1) Create rolled identity matrix:
104         | 0 0 1 |
105         | 1 0 0 |
106         | 0 1 0 |
107     2) Dot with image
108     '''
109     i       = numpy.roll(numpy.identity(len(image[0])),
110                          240, axis=0) # Create rolled idm
111     shifted = numpy.dot(image, i) # dot with image
112     return shifted
113
114 def shift_vert(image):
115     '''
116     Shift an image horizontally
117     1) Create rolled identity matrix:
118         | 0 0 1 |
119         | 1 0 0 |
120         | 0 1 0 |
121     2) Dot with image
122     '''
123     i       = numpy.roll(numpy.identity(len(image)),
124                          100, axis=0) # create rolled idm
125     shifted = numpy.dot(i, image) # dot with image
126     return shifted
127
128 def create_grayscale(image):
```

```
129     '''
130     Creates grayscale image from given matrix
131     1) Create ratio matrix
132     2) Dot with image
133     '''
134     ratio = numpy.array([30., 59., 11.])
135     return numpy.dot(image.astype(numpy.float), ratio)
136 
137 def shift_hort_color(image):
138     '''
139     Shift a color image horizontally
140     1) Create identity matrix that looks as such:
141         | 0 0 1 |
142         | 1 0 0 |
143         | 0 1 0 |
144     2) Dot it with image matrix
145     3) Return Transpose
146     '''
147     # Create an identity matrix and roll the rows
148     i       = numpy.roll(
149             numpy.identity(
150                 len(image[0]))
151             , 240, axis=0)
152     shifted = numpy.dot(i, image) # Dot with image
153     return numpy.transpose(shifted) # Return transpose
154 
155 def compression(image, name, p):
156     '''
157     Compress the image using DST
158     '''
159     g = create_grayscale(image.copy()) # Create grayscale image matrix copy
160     t = dst(g)  # Acquire DST matrix of image
161     (row_size, column_size) = numpy.shape(t) # Size of t
162     for row in range(row_size):
163         for col in range(column_size):
164             if (row + col + 2) > (2 * p * column_size):
165                 t[row][col] = 0 # if the data is above a set line, delete it
166     scipy.misc.imsave('../img/comp%s.png' %name, dst(t))
167 
168 def dst(image):
169     '''
170     If given a grayscale image array, use the DST formula
171     and return the result
172     Uses this method:
173         image = X
174         DST   = S
175         Y = S.(X.S)
176     '''
177     rows    = numpy.dot(image, create_S(len(image[0])))
178     columns = numpy.dot(create_S(len(image)), rows)
179     return columns
180 
181 def create_S(n):
182     '''
```

```
183     Discrete Sine Transform
184     1) Initialize variables
185     2) For each row and column, create an entry
186     '''
187     new_array = []  # What we will be filling
188     size      = n
189     for row in range(size):
190         new_row = []    # New row for every row
191         for col in range(size):
192             S = ((numpy.sqrt(2.0 / size)) * # our equation
193                  (numpy.sin((numpy.pi * ((row + 1) - (1.0/2.0)) *
194                      ((col + 1) - (1.0/2.0)))/(size))))
195             new_row.append(S) # Append entry to row list
196         new_array.append(new_row) # append row to array
197     return_array = numpy.array(new_array)
198     return return_array
199
200 def mass_pics(image, name):
201     '''
202     Create a lot of compressed Pictures
203     '''
204     answer = raw_input('Create .gif Images? (y/n) ')
205     if answer == 'n':
206         return None # It takes a while, so it's optional
207     domain = numpy.arange(0, 1.01, 0.01) # Range of p vals
208     for p in domain:
209         jobs.append(
210                 jobServer.submit(compression,
211                     (image, 'array_%s_%f' %(name, p), p),
212                     (create_grayscale, dst, create_S),
213                     ('numpy', 'scipy.misc'))
214                 ) # For each value of p, add a new compression job
215
216 def visualize_s():
217     '''
218     DST
219     Visualize the discrete sine transform equation implemented below.
220     Uses matplotlib to create graph
221     '''
222     nrange   = numpy.arange(1, 33, 1) # Create values range [1,32] stepsize 1
223     det_plot = plt.figure() # New matplotlib class instance for a figure
224     det_axes = det_plot.add_axes([0.1, 0.1, 0.8, 0.8]) # Add axes to figure
225     yrange   = [] # Create an empty y range (we'll be adding to this)
226     for number in nrange:
227         array = create_S(number)    # Get a new array with size n
228         yrange.append(numpy.linalg.det(array)) # append determinant to yrange
229     det_axes.plot(nrange, yrange, label='Set of determinants') # Create line
230     det_axes.plot(nrange, nrange*0, 'k:')   # Also create line at y=0
231     det_axes.legend(loc=4) # Place legend
232     plt.xlabel('Size of Discrete Sine Transform Matrix') # Label X
233     plt.ylabel('Determinant of Matrix') # Label Y
234     plt.title('Size of Matrix vs. its Determinant') # Title
235     plt.savefig('../img/dst_dets.png') # Save as a png
236
```

```
237 def comp_effect(image1, image2):
238     '''
239     Analyzes compression effectiveness
240     If the image already exists, it will not run this
241     '''
242     try:
243         open('../img/bitcount.png', 'r')
244         open('../img/bitrat.png', 'r')
245         print(' |-> Graphs already created, skipping.\
246                 (Delete existing graphs to recreate)')
247         # If it already exists, don't create it. (It takes a while)
248     except IOError:
249         g1 = create_grayscale(image1.copy()) # Create grayscale from copy of 1
250         g2 = create_grayscale(image2.copy()) # Create grayscale from copy of 2
251 
252         domain1 = numpy.arange(0.0, 1.01, 0.01) # Range of p values
253         domain2 = numpy.arange(0.0, 1.01, 0.01) # Range of p values
254 
255         # Parallelize System and generate range
256         count_y1, rat_y1 = jobServer.submit(get_yrange,
257                         (domain1, g1),
258                         (dst, clear_vals, create_S),
259                         ('numpy', 'scipy.misc'))()
260         count_y2, rat_y2 = jobServer.submit(get_yrange,
261                         (domain2, g2),
262                         (dst, clear_vals, create_S),
263                         ('numpy', 'scipy.misc'))()
264 
265         count_plot = plt.figure() # New class instance for a figure
266         count_axes = count_plot.add_axes([0.1, 0.1, 0.8, 0.8]) # Add axes
267         count_axes.plot(domain1, count_y1, label='Image 1')
268         count_axes.plot(domain2, count_y2, label='Image 2')
269         count_axes.legend(loc=4)
270         plt.xlabel("Value of p")
271         plt.ylabel("Number of Non-Zero Bytes")
272         plt.title("Compression Effectiveness")
273         plt.savefig("../img/bitcount.png")
274 
275         ratio_plot = plt.figure() # New class instance for a figure
276         ratio_axes = ratio_plot.add_axes([0.1, 0.1, 0.8, 0.8]) # Add axes
277         ratio_axes.plot(domain1, rat_y1, label='Image 1')
278         ratio_axes.plot(domain2, rat_y2, label='Image 2')
279         ratio_axes.legend(loc=4)
280         plt.xlabel("Value of p")
281         plt.ylabel("Ratio of Non-Zero Bytes to Total Bytes")
282         plt.title("Compression Effectiveness")
283         plt.savefig("../img/bitrat.png")
284 
285 def get_yrange(domain, g):
286     bit_count = [] # Range for image
287     bit_ratio = []
288     for p in domain:
289         t = dst(g.copy()) # Transform 1
290         initial_count = float(numpy.count_nonzero(t))
```

```
291         clear_vals(t, p) # Strip of high-freq data
292         final_count = float(numpy.count_nonzero(t))
293         bit_count.append(final_count) # Append number of non-zero entries
294         bit_ratio.append(final_count / initial_count)
295     return bit_count, bit_ratio
296 
297 def clear_vals(transform, p):
298     '''
299     Takes image and deletes high frequency
300     '''
301     (row_size, column_size) = numpy.shape(transform) # Size of t
302     for row in range(row_size):
303         for col in range(column_size):
304             if (row + col + 2) > (2 * p * column_size):
305                 transform[row][col] = 0 # if the data is above line, delete it
306     return transform
307 
308 if __name__ == '__main__':
309     sys.exit(main())
```

5.2 MATLAB Code

Some MATLAB Code was also made that features equivalent functionality

Grayscale

```
function gray_image=grayscale(image)
% This is a function to take an image in jpg form and put it into grayscale

% This reads in the image
image_matrix=imread(image);

% get the dimensions
[rows,columns,~]=size(image_matrix);

% preallocate
gray_image = zeros(rows,columns);
for a=1:rows;
    for b=1:columns;
            gray_image(a,b)=0.3*image_matrix(a,b,1)...
                +0.59*image_matrix(a,b,2)...
                +0.11*image_matrix(a,b,3);
    end
end
imwrite(uint8(gray_image),'name.jpg')

end
```

Horizontal Shifting

```
function [hshifted_image] = hshift(image)

% c is the number of cols we want to shift by
c = 240;

% read in the image and make it a nice little matrix
image_matrix=double(imread(image));

% get the dimensions of the matrix
[rows, cols] = size(image_matrix);

% get the largest dimension for the identity matrix
n = max(rows, cols);

% Preallocate for the id matrix:
T = zeros(n,n);

% generate a generic identity matrix
id = eye(n);

%fill in the first c cols of T with the last c cols of id
T(:,1:c)=id(:,n-(c-1):n);
%fill in the rest of T with the first part of id
```

```
T(:,c+1:n) = id(:,1:n-c);

hshifted_image=uint8(image_matrix*T);

imwrite(hshifted_image,'hshifted.jpg');
```

Vertical Shifting

```
function [vshifted_image] = vshift(image)

% r is the number of rows we want to shift by
r = 100;

% read in the image and make it a nice little matrix
image_matrix=double(imread(image));

% get the dimensions of the matrix
[rows, cols] = size(image_matrix);

% get the largest dimension for the identity matrix
n = min(rows, cols);

% Preallocate for the id matrix:
T = zeros(n,n);

% generate a generic identity matrix
id = eye(n);

%fill in the first c cols of T with the last c cols of id
T(1:r,:)=id(n-(r-1):n,:);
%fill in the rest of T with the first part of id
T(r+1:n,:) = id(1:n-r,:);

vshifted_image=uint8(T*image_matrix);

imwrite(vshifted_image,'vshifted.jpg');
```