

2018 Fall Advance Digital Image Processing Homework #2-2

EE 245765

Advisor: 電子所 高立人

106368002 張昌祺 Justin, Chang-Qi Zhang

justin840727@gmail.com

Due Date: 13:00pm, Oct 9 2018

Problem 2 Zooming and Shrinking (C/C++)

- a. Zooming the image with ratio 2:1 raw-column replication. Compare the output with lena512.raw. (Figure, 10%; Discussion, 5%)

Ans

In Figure 1, it is very clear to describe how row-col-replication works to achieve rooming image. Scale step, we multiply row index an column index with scale factor (2 in this case). Row and column replication are simply duplicate the row i and column j to row $i + 1$ and column $j + 1$.

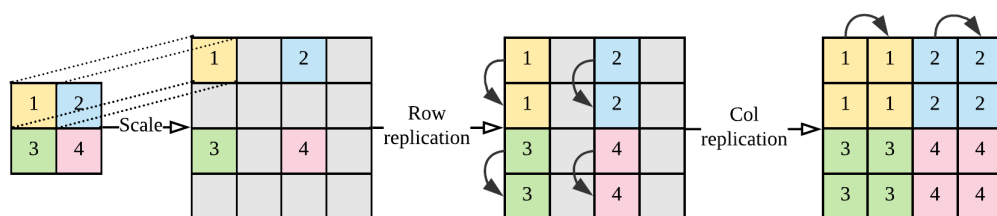


Figure 1: Concept of row-col replication.

Figure 2 shows the original Lena 512 image(result_img/lena_512.png) and the result(result_img/2-a zooming lena row-col replication.png) of row-col replication from Lena 256 image. Then you can see there is checkerboard effect on row-col replication result.



(a) Lena 512 original.



(b) Row-col replication from Lena 256.

Figure 2: Lena 512 and Lena 256 Row-col replication.

I calculated MSE and PSNR between Lena 512 and col-row replication. The running result as Figure 3. The data is loss a lot here. The typical PSNR value for video compression are between 30 to 50 dB.

```
> ./hw2_2_rooming_shrinking
Hw2.2.a
MSE: 24.3996, PSNR: -13.8738 db
```

Figure 3: MSE and PSNR result.

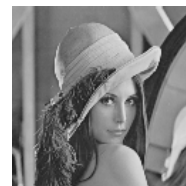
- b. Shrinking the image with ratio 1:2 row-column deletion. Check your result with or without blurring (using Xnview) your input image before shrinking. (Figure, 10%; Discussion, 5%)

Ans

Raw-column deletion is a simple method to shrinking image. The difference between Figure 4 and Figure 5 is that Figure 4 direct compute row-col deletion and Figure 5 use gaussian blur first then compute row-col deletion. Row-col deletion is a sampling method so it is very easy to get the aliasing effect, if the input image is a high detail image. For solving aliasing effect we can make the image blur before we apply row-col deletion.

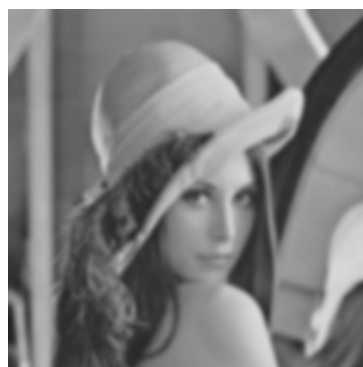


(a) Lena 256 original.

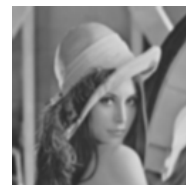


(b) Row-col deletion from Lena 256.

Figure 4: Results of Lena 256 Row-col deletion.



(a) Lena 256 gaussian blur.



(b) Row-col deletion from Lena 256 gaussian blur.

Figure 5: Results of Lena 256 gaussian blur Row-col deletion.

- c. Zooming the image with ratio 2.3 using both nearest-neighboring and bilinear interpolation. Discuss the difference in the output images. (Figure, 10%; Discussion, 5%)

Ans

On results of those two method, we can see in result of *nearest neighboring* got obvious checkerboard effect but it does not happen on result of *bilinear interpolation*.



(a) Zooming with nearest-neighboring



(b) Zooming with bilinear interpolation

Figure 6: Zooming results for *nearest-neighboring* and *bilinear interpolation*

Source code for Problem 2

hw2_2-rooming-shrinking.hpp

```
1 #include <iostream>
2 #include <opencv2/opencv.hpp>
3 #include <opencv2/highgui/highgui.hpp>
4
5 const std::string SAVE_IMG_FOLDER = "../result_img/";
6
7 void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int height);
8 void showImage(std::string win_name, cv::Mat &show_img);
9 void rowColReplication(cv::Mat &src_img, cv::Mat &dst_img);
10 void rowColDeletion(cv::Mat &src_img, cv::Mat &dst_img);
11 void gaussianBlur(cv::Mat &src_img, cv::Mat &dst_img, int kernel_size);
12 void nearestNeighboring(cv::Mat &src_img, cv::Mat &dst_img);
13 void bilinearInterpolation(cv::Mat &src_img, cv::Mat &dst_img);
14 void saveImage(cv::Mat &img, std::string prefix);
15 double getMSE(cv::Mat &src, cv::Mat &target);
16 double getPSNR(double mse, int num_bits);
```

hw2_2-rooming-shrinking.cpp

```
1 #include "hw2_2-rooming-shrinking.hpp"
2
3 void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int height)
4 {
5     std::FILE* f = std::fopen(file_path.c_str(), "rb");
6     // std::vector<char> buf(width*height); // char is trivially copyable
7     unsigned char buf[width][height];
8     std::fread(&buf[0], sizeof buf[0], width*height, f);
9     for (int i = 0; i < dst_img.rows; i++)
```

```

10 {
11     for (int j = 0; j < dst_img.cols; j++)
12     {
13         dst_img.at<char>(i, j) = buf[i][j];
14     }
15 }
16 std::fclose(f);
17 }
18
19 void showImage(std::string win_name, cv::Mat &show_img)
20 {
21     static int win_move_x = 50;
22     static int win_move_y = 50;
23     cv::namedWindow(win_name, 0);
24     cv::resizeWindow(win_name, show_img.cols, show_img.rows);
25     cv::moveWindow(win_name, win_move_x, win_move_y);
26     cv::imshow(win_name, show_img); //display Image
27     win_move_x += show_img.cols;
28     if (win_move_x > 1920-256)
29     {
30         win_move_x = 50;
31         win_move_y += (show_img.rows+35);
32     }
33 }
34
35 void rowColReplication(cv::Mat &src_img, cv::Mat &dst_img)
36 {
37     int scale = dst_img.cols / src_img.cols;
38     for (int i = 0; i < src_img.rows; i++)
39     {
40         for (int j = 0; j < src_img.cols; j++)
41         {
42             dst_img.at<char>(scale*i, scale*j) = src_img.at<char>(i, j);
43         }
44     }
45     for (int i = 0; i < dst_img.cols; i = i + 2)
46     {
47         for (int j = 1; j < scale; j++)
48         {
49             dst_img.col(i).copyTo(dst_img.col(i+j));
50         }
51     }
52     for (int i = 0; i < dst_img.rows; i = i + 2)
53     {
54         for (int j = 1; j < scale; j++)
55         {
56             dst_img.row(i).copyTo(dst_img.row(i+j));
57         }
58     }
59 }
60
61 void gaussianBlur(cv::Mat &src_img, cv::Mat &dst_img, int kernel_size)
62 {
63     for (int i=1; i<kernel_size; i=i+2)
64     {
65         GaussianBlur( src_img, dst_img, cv::Size( i, i ), 0, 0 );
66     }
67 }
68
69 void rowColDeletion(cv::Mat &src_img, cv::Mat &dst_img)

```

```

70 {
71     int scale = src_img.cols / dst_img.cols;
72     for (int i = 0; i < dst_img.rows; i++)
73     {
74         for (int j = 0; j < dst_img.cols; j++)
75         {
76             dst_img.at<char>(i, j) = src_img.at<char>(i*scale, j*scale);
77         }
78     }
79 }
80
81 void nearestNeighboring(cv::Mat &src_img, cv::Mat &dst_img)
82 {
83     cv::Mat mat_status(dst_img.rows, dst_img.cols, CV_8UC1, cv::Scalar(0));
84     double scale = (double)dst_img.cols / (double)src_img.cols;
85     for (int i = 0; i < src_img.rows; i++)
86     {
87         for (int j = 0; j < src_img.cols; j++)
88         {
89             int x = scale*j;
90             int y = scale*i;
91             if (mat_status.at<char>(y, x) == 0)
92             {
93                 mat_status.at<char>(y, x) = 1;
94                 std::array<int, 2> index{ {y, x} };
95                 dst_img.at<char>(y, x) = src_img.at<char>(i, j);
96             }
97         }
98     }
99     // find nearest point and fill in data to images
100     int search_margin = scale + 1;
101     for (int i = 0; i < mat_status.rows; i++)
102     {
103         for (int j = 0; j < mat_status.cols; j++)
104         {
105             if (mat_status.at<char>(i, j) != 1)
106             {
107                 int min_x = std::max(0, j - search_margin);
108                 int min_y = std::max(0, i - search_margin);
109                 int max_x = std::min(mat_status.cols - 1, j + search_margin);
110                 int max_y = std::min(mat_status.rows - 1, i + search_margin);
111                 int index[2];
112                 double nearest_dis = 2*search_margin;
113                 for (int k = min_y; k <= max_y; k++)
114                 {
115                     for (int l = min_x; l <= max_x; l++)
116                     {
117                         if (mat_status.at<char>(k, l) == 1)
118                         {
119                             double dis = sqrt(pow(k-i, 2) + pow(l-j, 2));
120                             if (dis < nearest_dis)
121                             {
122                                 nearest_dis = dis;
123                                 index[0] = k;
124                                 index[1] = l;
125                             }
126                         }
127                     }
128                 }
129                 dst_img.at<char>(i, j) = dst_img.at<char>(index[0], index[1]);

```

```

130     }
131   }
132 }
133 }
134
135 void bilinearInterpolation(cv::Mat &src_img, cv::Mat &dst_img)
136 {
137   cv::Mat mat_status(dst_img.rows, dst_img.cols, CV_8UC1, cv::Scalar(0));
138   double scale = (double)dst_img.cols / (double)src_img.cols;
139   for (int i = 0; i < src_img.rows; i++)
140   {
141     for (int j = 0; j < src_img.cols; j++)
142     {
143       int x = scale*j;
144       int y = scale*i;
145       if (mat_status.at<char>(y, x) == 0)
146       {
147         mat_status.at<char>(y, x) = 1;
148         std::array<int, 2> index{ {y, x} };
149         dst_img.at<char>(y, x) = src_img.at<char>(i, j);
150         // fill margin
151         if (j == src_img.cols - 1)
152         {
153           mat_status.at<char>(y, dst_img.cols - 1) = 1;
154           dst_img.at<char>(y, dst_img.cols - 1) = src_img.at<char>(i, j);
155         }
156         if (i == src_img.rows - 1)
157         {
158           mat_status.at<char>(dst_img.rows - 1, x) = 1;
159           dst_img.at<char>(dst_img.rows - 1, x) = src_img.at<char>(i, j);
160         }
161         if (i == src_img.rows - 1 && j == src_img.cols - 1)
162         {
163           mat_status.at<char>(dst_img.rows - 1, dst_img.cols - 1) = 1;
164           dst_img.at<char>(dst_img.rows - 1, dst_img.cols - 1) = src_img.at<char>(
165             i, j);
166         }
167       }
168     }
169     std::vector<int> pixel_list_x;
170     std::vector<int> pixel_list_y;
171     for (int i = 0; i < mat_status.cols; i++)
172     {
173       if (mat_status.at<char>(0, i) == 1)
174       {
175         pixel_list_x.push_back(i);
176       }
177     }
178     for (int i = 0; i < mat_status.rows; i++)
179     {
180       if (mat_status.at<char>(i, 0) == 1)
181       {
182         pixel_list_y.push_back(i);
183       }
184     }
185
186     // linear fill column
187     for (int &y : pixel_list_y)
188     {

```

```

189     int x_past = -1;
190     for (int &x : pixel_list_x)
191     {
192         if (x_past == -1)
193         {
194             x_past = x;
195         }
196         else
197         {
198             int dx = x - x_past;
199             unsigned char value_past = dst_img.at<char>(y, x_past);
200             unsigned char value = dst_img.at<char>(y, x);
201             double dv = (value - value_past)/(double)dx;
202             for (int i=1; i < dx; i++)
203             {
204                 dst_img.at<char>(y, x_past + i) = value_past + dv*i;
205             }
206             x_past = x;
207         }
208     }
209 }
210 // linear fill row
211 for (int x = 0; x < mat_status.cols; x++)
212 {
213     int y_past = -1;
214     for (int &y : pixel_list_y)
215     {
216         if (y_past == -1)
217         {
218             y_past = y;
219         }
220         else
221         {
222             int dy = y - y_past;
223             unsigned char value_past = dst_img.at<char>(y_past, x);
224             unsigned char value = dst_img.at<char>(y, x);
225             double dv = (value - value_past)/(double)dy;
226             for (int i = 1; i < dy; i++)
227             {
228                 dst_img.at<char>(y_past + i, x) = value_past + dv*i;
229             }
230             y_past = y;
231         }
232     }
233 }
234 }
235
236 void saveImage(cv::Mat &img, std::string prefix)
237 {
238     std::string save_file = SAVE_IMG_FOLDER + prefix + ".png";
239     cv::imwrite(save_file, img);
240 }
241
242 double getMSE(cv::Mat &src, cv::Mat &target)
243 {
244     double mse = 0;
245     for (int i = 0; i < src.rows; i++)
246     {
247         for (int j = 0; j < src.cols; j++)
248         {

```

```

249     unsigned char src_value = src.at<char>(i, j);
250     unsigned char target_value = target.at<char>(i, j);
251     mse += pow(src_value - target_value, 2);
252 }
253 }
254 return mse/(src.rows * src.cols);
255 }
256
257 double getPSNR(double mse, int num_bits)
258 {
259     char max_i = 0xff >> (8 - num_bits);
260     return 10 * log10(pow(max_i, 2) / mse);
261 }
262
263 int main(int argc, char **argv)
264 {
265     cv::Mat lena_256_src(256, 256, CV_8UC1);
266     loadRawFile(lena_256_src, "../images/lena_256.raw", 256, 256);
267
268     // HW2.a
269     cv::Mat row_col_rep(512, 512, CV_8UC1);
270     rowColReplication(lena_256_src, row_col_rep);
271     cv::Mat lena_512_src(512, 512, CV_8UC1);
272     loadRawFile(lena_512_src, "../images/lena_512.raw", 512, 512);
273     double mse = getMSE(lena_512_src, row_col_rep);
274     double psnr = getPSNR(mse, 8);
275     std::cout << "Hw2.2.a" << std::endl;
276     std::cout << "MSE: " << mse << ", PSNR: " << psnr << " db" << std::endl;
277     // HW2.b
278     cv::Mat row_col_del(128, 128, CV_8UC1);
279     rowColDeletion(lena_256_src, row_col_del);
280     cv::Mat lena_256_blur(256, 256, CV_8UC1);
281     gaussianBlur(lena_256_src, lena_256_blur, 10);
282     cv::Mat row_col_blur_del(128, 128, CV_8UC1);
283     rowColDeletion(lena_256_blur, row_col_blur_del);
284     // HW2.c
285     double zooming_ratio = 2.3;
286     cv::Mat nearest_neighboring(256*zooming_ratio, 256*zooming_ratio, CV_8UC1, cv::
        Scalar(0));
287     nearestNeighboring(lena_256_src, nearest_neighboring);
288     cv::Mat bilinear_interpolation(256*zooming_ratio, 256*zooming_ratio, CV_8UC1,
        cv::Scalar(0));
289     bilinearInterpolation(lena_256_src, bilinear_interpolation);
290
291     // Show results
292     showImage("lena 256 src", lena_256_src);
293     showImage("lena 512 src", lena_512_src);
294     showImage("lena row-col replication", row_col_rep);
295     showImage("lena row-col deletion", row_col_del);
296     showImage("lena blur", lena_256_blur);
297     showImage("lena blur deletion", row_col_blur_del);
298     showImage("lena nearest neighboring", nearest_neighboring);
299     showImage("lena bilinear interpolation", bilinear_interpolation);
300
301     // Save results
302     saveImage(lena_512_src, "lena_512");
303     saveImage(row_col_rep, "2-a zooming lena row-col replication");
304     saveImage(row_col_del, "2-b-1 shrinking lena row-col deletion");
305     saveImage(lena_256_blur, "2-b-2 lena 256 blur");
306     saveImage(row_col_blur_del, "2-b-3 lena 256 blur row-col deletion");

```



```
307 saveImage(nearest_neighboring, "2-c-1 zooming lena nearest neighboring");
308 saveImage(bilinear_interpolation, "2-c-2 zooming lena bilinear interpolation");
309 cv::waitKey(0);
310 return 0;
311 }
```

Problem 3 Isopreference test (C/C++)

Experiment the isopreference test on lena_256.raw and baboon_256.raw images with your programs developed in Problems 1 & 2. Do your experiments and observations match the textbook description? Discuss it. (Discussion, 20%)

Ans

In textbook, it just mention about the isopreference with different gray-level resolution. There is no any section which discussion the relationship between rooming, shrinking and gray-level resolution. The experiments in this section is try to zooming and shrinking the high detail(Baboon) and low detail(Lena) image in different gray-level resolution.