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 $j+1$.

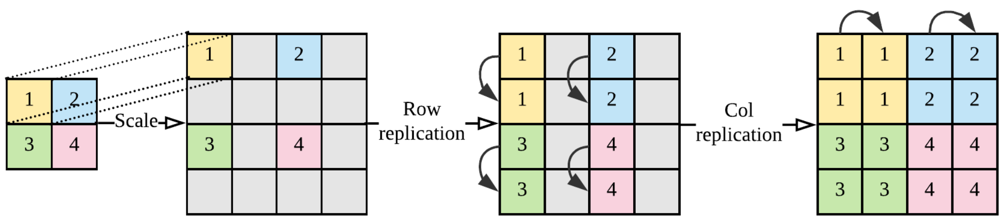

Figure 1: Concept of row-col replication.

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.

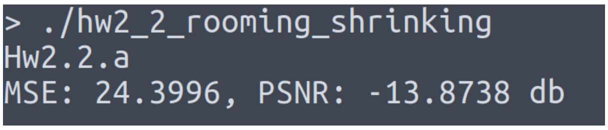

Figure 3: MSE and PSNR result.

b. Shrinking the image with ratio 1:2 raw-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 rosw-col deletion and Figure 5 use gaussian blur first then compute ros-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

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>

const std::string SAVE_IMG_FOLDER = "../result_img/";

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int height);
void showImage(std::string win_name, cv::Mat &show_img);
void rowColReplication(cv::Mat &src_img, cv::Mat &dst_img);
void rowColDeletion(cv::Mat &src_img, cv::Mat &dst_img);
void gaussionBlur(cv::Mat &src_img, cv::Mat &dst_img, int kernel_size);
void nearestNeighboring(cv::Mat &src_img, cv::Mat &dst_img);
void bilinearInterpolation(cv::Mat &src_img, cv::Mat &dst_img);
void saveImage(cv::Mat &img, std::string prefix);
double getMSE(cv::Mat &src, cv::Mat &target);
double getPSNR(double mse, int num_bits);
```

hw2_2_rooming_shrinking.cpp

```
#include "hw2_2_rooming_shrinking.hpp"

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int height)
{
  std::FILE* f = std::fopen(file_path.c_str(), "rb");
  // std::vector<char> buf(width*height);     // char is trivally copyable
  unsigned char buf[width][height];
  std::fread(&buf[0], sizeof buf[0], width*height, f);
  for (int i = 0; i < dst_img.rows; i++)
```

```
  {
    for (int j = 0; j < dst_img.cols; j++)
    {
      dst_img.at<char>(i, j) = buf[i][j];
    }
  }
  std::fclose(f);
}

void showImage(std::string win_name, cv::Mat &show_img)
{
  static int win_move_x = 50;
  static int win_move_y = 50;
  cv::namedWindow(win_name, 0);
  cv::resizeWindow(win_name, show_img.cols, show_img.rows);
  cv::moveWindow(win_name, win_move_x, win_move_y);
  cv::imshow(win_name, show_img); //display Image
  win_move_x +=  show_img.cols;
  if (win_move_x > 1920-256)
  {
    win_move_x = 50;
    win_move_y += (show_img.rows+35);
  }
}

void rowColReplication(cv::Mat &src_img, cv::Mat &dst_img)
{
  int scale = dst_img.cols / src_img.cols;
  for (int i = 0; i < src_img.rows; i++)
  {
    for (int j = 0; j < src_img.cols; j++)
    {
      dst_img.at<char>(scale*i, scale*j) =  src_img.at<char>(i, j);
    }
  }
  for (int i = 0; i < dst_img.cols; i = i + 2)
  {
    for (int j = 1; j < scale; j++)
    {
      dst_img.col(i).copyTo(dst_img.col(i+j));
    }
  }
  for (int i = 0; i < dst_img.rows; i = i + 2)
  {
    for (int j = 1; j < scale; j++)
    {
      dst_img.row(i).copyTo(dst_img.row(i+j));
    }
  }
}

void gaussionBlur(cv::Mat &src_img, cv::Mat &dst_img, int kernel_size)
{
  for (int i=1; i<kernel_size; i=i+2)
  {
    GaussianBlur( src_img, dst_img, cv::Size( i, i ), 0, 0 );
  }
}

void rowColDeletion(cv::Mat &src_img, cv::Mat &dst_img)
```

```
{
  int scale =  src_img.cols / dst_img.cols;
  for (int i = 0; i < dst_img.rows; i++)
  {
    for (int j = 0; j < dst_img.cols; j++)
    {
       dst_img.at<char>(i, j) = src_img.at<char>(i*scale, j*scale);
    }
  }
}

void nearestNeighboring(cv::Mat &src_img, cv::Mat &dst_img)
{
  cv::Mat mat_status(dst_img.rows,dst_img.cols, CV_8UC1, cv::Scalar(0));
  double scale = (double)dst_img.cols / (double)src_img.cols;
  for (int i = 0; i < src_img.rows; i++)
  {
    for (int j = 0; j < src_img.cols; j++)
    {
      int x = scale*j;
      int y = scale*i;
      if (mat_status.at<char>(y, x) == 0)
      {
        mat_status.at<char>(y, x) = 1;
        std::array<int, 2> index{ {y, x} };
        dst_img.at<char>(y, x) =  src_img.at<char>(i, j);
      }
    }
  }
  // find nearest point and fill in data to images
  int search_margin = scale + 1;
  for (int i = 0; i < mat_status.rows; i++)
  {
    for (int j = 0; j < mat_status.cols; j++)
    {
      if  (mat_status.at<char>(i, j) != 1)
      {
        int min_x = std::max(0, j - search_margin);
        int min_y = std::max(0, i - search_margin);
        int max_x = std::min(mat_status.cols - 1, j + search_margin);
        int max_y = std::min(mat_status.rows - 1, i + search_margin);
        int index[2];
        double nearest_dis = 2*search_margin;
        for (int k = min_y; k <= max_y; k++)
        {
          for (int l = min_x; l <= max_x; l++)
          {
            if (mat_status.at<char>(k, l) == 1)
            {
              double dis = sqrt(pow(k-i, 2) + pow(l-j, 2));
              if (dis < nearest_dis)
              {
                nearest_dis = dis;
                index[0] = k;
                index[1] = l;
              }
            }
          }
        }
        dst_img.at<char>(i, j) = dst_img.at<char>(index[0], index[1]);
```

```
      }
    }
  }
}

void bilinearInterpolation(cv::Mat &src_img, cv::Mat &dst_img)
{
  cv::Mat mat_status(dst_img.rows,dst_img.cols, CV_8UC1, cv::Scalar(0));
  double scale = (double)dst_img.cols / (double)src_img.cols;
  for (int i = 0; i < src_img.rows; i++)
  {
    for (int j = 0; j < src_img.cols; j++)
    {
      int x = scale*j;
      int y = scale*i;
      if (mat_status.at<char>(y, x) == 0)
      {
        mat_status.at<char>(y, x) = 1;
        std::array<int, 2> index{ {y, x} };
        dst_img.at<char>(y, x) =  src_img.at<char>(i, j);
        // fill margin
        if(j ==  src_img.cols -1)
        {
          mat_status.at<char>(y, dst_img.cols -1) = 1;
          dst_img.at<char>(y, dst_img.cols -1) = src_img.at<char>(i, j);
        }
        if(i ==  src_img.rows -1)
        {
          mat_status.at<char>(dst_img.rows -1, x) = 1;
          dst_img.at<char>(dst_img.rows -1, x) = src_img.at<char>(i, j);
        }
        if(i ==  src_img.rows -1 && j ==  src_img.cols -1)
        {
          mat_status.at<char>(dst_img.rows -1, dst_img.cols -1) = 1;
          dst_img.at<char>(dst_img.rows -1,  dst_img.cols -1) = src_img.at<char>(i, j);
        }
      }
    }
  }
  std::vector<int> pixel_list_x;
  std::vector<int> pixel_list_y;
  for (int i = 0; i < mat_status.cols; i++)
  {
    if (mat_status.at<char>(0, i) == 1)
    {
      pixel_list_x.push_back(i);
    }
  }
  for (int i = 0; i < mat_status.rows; i++)
  {
    if (mat_status.at<char>(i, 0) == 1)
    {
      pixel_list_y.push_back(i);
    }
  }

  // linear fill column
  for (int &y : pixel_list_y)
  {
```

```
    int x_past = -1;
    for (int &x : pixel_list_x)
    {
      if (x_past == -1)
      {
        x_past = x;
      }
      else
      {
        int dx = x - x_past;
        unsigned char value_past = dst_img.at<char>(y, x_past);
        unsigned char value = dst_img.at<char>(y, x);
        double dv = (value - value_past)/(double)dx;
        for (int i=1; i < dx; i++)
        {
          dst_img.at<char>(y, x_past + i) = value_past + dv*i;
        }
        x_past = x;
      }
    }
  }
  // linear fill row
  for (int x = 0; x < mat_status.cols; x++)
  {
    int y_past = -1;
    for (int &y : pixel_list_y)
    {
      if (y_past == -1)
      {
        y_past = y;
      }
      else
      {
        int dy = y - y_past;
        unsigned char value_past = dst_img.at<char>(y_past, x);
        unsigned char value = dst_img.at<char>(y, x);
        double dv = (value - value_past)/(double)dy;
        for (int i = 1; i < dy; i++)
        {
          dst_img.at<char>(y_past + i, x) = value_past + dv*i;
        }
        y_past = y;
      }
    }
  }
}

void saveImage(cv::Mat &img, std::string prefix)
{
  std::string save_file = SAVE_IMG_FOLDER + prefix + ".png";
  cv::imwrite(save_file, img);
}

double getMSE(cv::Mat &src, cv::Mat &target)
{
  double mse = 0;
  for (int i = 0; i < src.rows; i++)
  {
    for (int j = 0; j < src.cols; j++)
    {
```

```
      unsigned char src_value = src.at<char>(i, j);
      unsigned char target_value = target.at<char>(i, j);
      mse += pow(src_value - target_value, 2);
    }
  }
  return mse/(src.rows * src.cols);
}

double getPSNR(double mse, int num_bits)
{
  char max_i = 0xff >> (8 - num_bits);
  return 10 * log10(pow(max_i, 2) / mse);
}

int main(int argc, char **argv)
{
  cv::Mat lena_256_src(256, 256, CV_8UC1);
  if (argc > 1)
  {
    cv::imread(argv[1], CV_LOAD_IMAGE_COLOR).copyTo(lena_256_src);
  }
  else
  {
    loadRawFile(lena_256_src, "../images/lena_256.raw", 256, 256);
  }
  // HW2.a
  cv::Mat row_col_rep(512, 512, CV_8UC1);
  rowColReplication(lena_256_src, row_col_rep);
  cv::Mat lena_512_src(512, 512, CV_8UC1);
  loadRawFile(lena_512_src, "../images/lena_512.raw", 512, 512);
  double mse = getMSE(lena_512_src, row_col_rep);
  double psnr = getPSNR(mse, 8);
  std::cout << "Hw2.2.a" << std::endl;
  std::cout << "MSE: " << mse << ", PSNR: " << psnr << " db" << std::endl;
  // HW2.b
  cv::Mat row_col_del(128, 128, CV_8UC1);
  rowColDeletion(lena_256_src, row_col_del);
  cv::Mat lena_256_blur(256, 256, CV_8UC1);
  gaussionBlur(lena_256_src, lena_256_blur, 10);
  cv::Mat row_col_blur_del(128, 128, CV_8UC1);
  rowColDeletion(lena_256_blur, row_col_blur_del);
  // HW2.c
  double zooming_ratio = 2.3;
  cv::Mat nearest_neighboring(256*zooming_ratio, 256*zooming_ratio, CV_8UC1, cv::Scalar(0));
  nearestNeighboring(lena_256_src, nearest_neighboring);
  cv::Mat bilinear_interpolation(256*zooming_ratio, 256*zooming_ratio, CV_8UC1, cv::Scalar(0));
  bilinearInterpolation(lena_256_src, bilinear_interpolation);

  // Show results
  showImage("lena 256 src", lena_256_src);
  showImage("lena 512 src", lena_512_src);
  showImage("lena row-col replication", row_col_rep);
  showImage("lena row-col deletion", row_col_del);
  showImage("lena blur", lena_256_blur);
  showImage("lena blur deletion", row_col_blur_del);
  showImage("lena nearest neighboring", nearest_neighboring);
  showImage("lena bilinear interpolation", bilinear_interpolation);

```

```
    // Save results
    saveImage(lena_512_src, "lena_512");
    saveImage(row_col_rep, "2-a zooming lena row-col replication");
    saveImage(row_col_del, "2-b-1 shrinking lena row-col deletion");
    saveImage(lena_256_blur, "2-b-2 lena 256 blur");
    saveImage(row_col_blur_del, "2-b-3 lena 256 blur row-col deletion");
    saveImage(nearest_neighboring, "2-c-1 zooming lena nearest neighboring");
    saveImage(bilinear_interpolation, "2-c-2 zooming lena bilinear interpolation");
    cv::waitKey(0);
    return 0;
}
```

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.

The experiment result shows that if we rooming with high detail image, the checkerboard effect is stronger than low detail image. For shrinking, the high detail image will cause aliasing but it will not obvious or not happen on low detail image.