2018 Fall Advance Digital Image Processing Homework #2-1

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 2 2018

Problem 1 Gray-level resolution with C++

a. Using C/C++ to quantize the gray-level resolution of lena_256.raw and baboon_256.raw from 8 bits to 1 bit. Show the results of these quantize images and explain the difference between each result image. (Figure, 15%; Discussion, 10%)

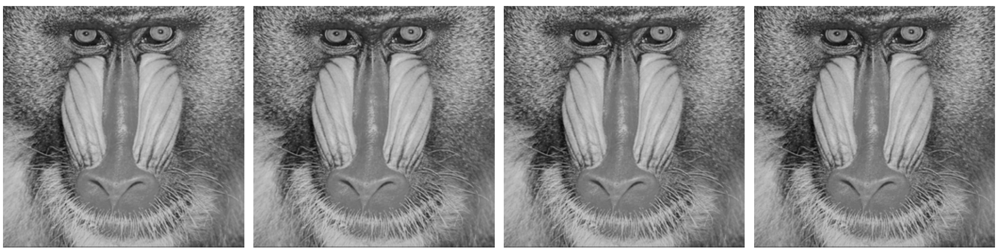
Ans

Firstly we take a look the result images which are generated by my program for both Lena and baboon gray-level resolution from 8 bits to 1 bit.

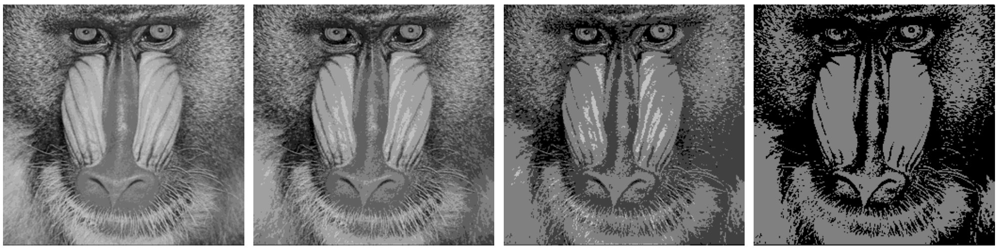
(a) Lena 8 bit. (b) Lena 7 bits. (c) Lena 6 bits. (d) Lena 5 bits.

(e) Lena 4 bits. (f) Lena 3 bits. (g) Lena 2 bits. (h) Lena 1 bits.

Figure 1: lena_256.raw gray-level resolution from 8 bits to 1 bit.



(a) Baboon 8 bit. (b) Baboon 7 bits. (c) Baboon 6 bits. (d) Baboon 5 bits.



(e) Lena 4 bit. (f) Lena 3 bits. (g) Lena 2 bits. (h) Lena 1 bits.

Figure 2: baboon_256.raw gray-level resolution from 8 bits to 1 bit.

In this section, we compare **False Contouring** between Lena (Figure 1) and Baboon (Figure 2) images. In Lena's case, when the gray-level resolution down to 3 bits. The figure shows obvious False contouring. In Baboon case, the false contouring effect happens in 2 bits gray-level resolution. Then we know false contouring might happen in different bit number in different detail images. For low detail image like Lena, we need to represent the image with more bits than high detail baboon image. The results for this problem is matching the Isopreference Curve theory.

b. Calculate the corresponding with MSE (Mean Square Error, study yourself) and PSNR value. (Discussion, 10%)

Ans

For calculate the MES (Mean square error) for the images, we use Equation (1) [1].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(I_i - \hat{I}_i)^2 \tag{1}$$

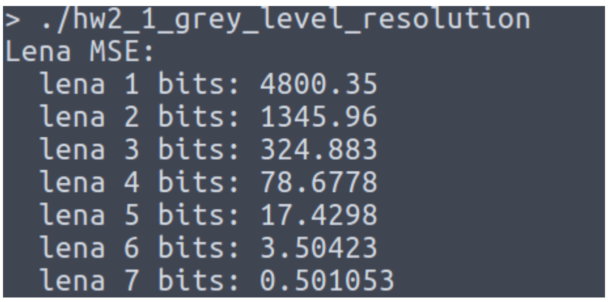
The total number of pixels **n** is define as images $width \times height$. And we sum up the square of pixel difference between original image I_i and resample image $\hat{I}_i$.

The equation for PSNR (Peak signal-to-noise ratio) value shows in Equation (2) below.

$$PSNR = 10 \cdot \log_{10}(\frac{MAX_I^2}{MSE}) \tag{2}$$

Here, the MAX_I is the maximux pixel value of the image. For example, In 8 bits case, the $MAX_I = 2^8 - 1 = 255$. MSE is same value that is calculated by Equation (1) [2].

The execute results for MSE and PSNR of different gray-level resolution are show on Figure 3.



```
> ./hw2_1_grey_level_resolution
Lena MSE:
  lena 1 bits: 4800.35
  lena 2 bits: 1345.96
  lena 3 bits: 324.883
  lena 4 bits: 78.6778
  lena 5 bits: 17.4298
  lena 6 bits: 3.50423
  lena 7 bits: 0.501053
```

```
  lena 8 bits: 0
Lena PSNR:
  lena 1 bits: -36.8127 db
  lena 2 bits: -21.7479 db
  lena 3 bits: -8.21531 db
  lena 4 bits: 4.5633 db
  lena 5 bits: 17.4143 db
  lena 6 bits: 30.5409 db
  lena 7 bits: 45.0772 db
  lena 8 bits: inf db
Baboon MSE:
  baboon 1 bits: 5382.14
  baboon 2 bits: 1497.8
  baboon 3 bits: 338.664
  baboon 4 bits: 77.857
  baboon 5 bits: 17.4878
  baboon 6 bits: 3.52115
  baboon 7 bits: 0.503754
  baboon 8 bits: 0
Baboon PSNR:
  lena 1 bits: -37.3096 db
  lena 2 bits: -22.2121 db
  lena 3 bits: -8.39573 db
  lena 4 bits: 4.60885 db
  lena 5 bits: 17.3999 db
  lena 6 bits: 30.52 db
  lena 7 bits: 45.0539 db
  lena 8 bits: inf db
```

Figure 3: MSE and PSNR results for different gray-level resolution.

The result shows that if we represent the image with more bits. The MSE will be lower and the PSNR value is higher. When there is no error between two images ($MSE = 0$), the PSNR will become infinity high.

Source code for Problem 1

Steps for build and execute the code

```
# build
cd hw2_1_gray_level_resolution/
mkdir build && cd build
cmake ..
make
# execute code
./hw2_1_gray_level_resolution
```

hw2_1_gray_level_resolution.hpp

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

void loadRawFile(cv::Mat &dst_img, std::string file_path, int width, int height);
cv::Mat getQuantizeImage(cv::Mat &src, int num_bit);
void showImage(std::string win_name, cv::Mat &show_img);
void showAllImages(std::vector<cv::Mat> &list, std::string prefix);
double getMSE(cv::Mat &src, cv::Mat &target);
double getPSNR(double mse, int num_bits);
void saveAllImage(std::vector<cv::Mat> &list,
                  std::string save_folder,
                  std::string prefix);
```

hw2_1_gray_level_resolution.cpp

```cpp
#include "hw2_1_grey_level_resolution.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);
}

cv::Mat getQuantizeImage(cv::Mat &src, int num_bit)
{
  cv::Mat img_out(src.rows, src.cols, CV_8UC1);
  char mask = 0xff << (8-num_bit);
  for (int i = 0; i < src.rows; i++)
  {
    for (int j = 0; j < src.cols; j++)
    {
      img_out.at<char>(i, j) = src.at<char>(i, j) & mask;
    }
  }
  return img_out;
}

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;
```

```cpp
        win_move_y += (show_img.rows+35);
    }
}

void showAllImages(std::vector<cv::Mat> &list, std::string prefix)
{
    for (int i = 0; i < list.size(); i++)
    {
        showImage(prefix + " " + std::to_string(i + 1) + " bits", list[i]);
    }
}

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++)
        {
            mse += pow(src.at<char>(i, j) - target.at<char>(i, j), 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);
}

void saveAllImage(std::vector<cv::Mat> &list,
                  std::string save_folder,
                  std::string prefix)
{
    for (int i = 0; i < list.size(); i++)
    {
        std::string save_file = save_folder + prefix + " " +
                                std::to_string(i + 1) + " bits.png";
        cv::imwrite(save_file, list[i]);
    }
}

int main(int argc, char **argv)
{
    cv::Mat lena_src(256, 256, CV_8UC1);
    cv::Mat baboon_src(256, 256, CV_8UC1);
    loadRawFile(lena_src, "../images/lena_256.raw", 256, 256);
    loadRawFile(baboon_src, "../images/baboon_256.raw", 256, 256);
    std::vector<cv::Mat> lena_result_list;
    std::vector<cv::Mat> baboon_result_list;
    // get quantize data from 1 bit to 8 bits
    for (int i = 1; i <= 8; i++)
    {
        lena_result_list.push_back(getQuantizeImage(lena_src, i));
        baboon_result_list.push_back(getQuantizeImage(baboon_src, i));
    }
```

```cpp
    // calculate MSE and PSNR
    std::vector<double> lena_mse_list;
    std::vector<double> lena_psnr_list;
    std::vector<double> baboon_mse_list;
    std::vector<double> baboon_psnr_list;
    for (int i = 0; i < 8; i++)
    {
        double lena_mse = getMSE(lena_src, lena_result_list[i]);
        double baboon_mse = getMSE(baboon_src, baboon_result_list[i]);
        lena_mse_list.push_back(lena_mse);
        baboon_mse_list.push_back(baboon_mse);
        lena_psnr_list.push_back(getPSNR(lena_mse, i+1));
        baboon_psnr_list.push_back(getPSNR(baboon_mse, i+1));
    }
    std::cout << "Lena MSE:" << std::endl;
    for (int i = 0; i < 8; i++)
    {
        std::cout << "  lena " << i+1 << " bits: " << lena_mse_list[i] << std::endl;
    }
    std::cout << "Lena PSNR:" << std::endl;
    for (int i = 0; i < 8; i++)
    {
        std::cout << "  lena " << i+1 << " bits: " << lena_psnr_list[i] << " db" << std::endl;
    }
    std::cout << "Baboon MSE:" << std::endl;
    for (int i = 0; i < 8; i++)
    {
        std::cout << "  baboon " << i+1 << " bits: " << baboon_mse_list[i] << std::endl;
    }
    std::cout << "Baboon PSNR:" << std::endl;
    for (int i = 0; i < 8; i++)
    {
        std::cout << "  lena " << i+1 << " bits: " << baboon_psnr_list[i] << " db" << std::endl;
    }
    saveAllImage(lena_result_list, "../result_img_2_1/", "lena");
    saveAllImage(baboon_result_list, "../result_img_2_1/", "baboon");
    showAllImages(lena_result_list, "lena");
    showAllImages(baboon_result_list, "baboon");
    cv::waitKey(0);
    return 0;
}
```

References

[1] *Wikipedia.* Mean squared error*[online].*
Available from World Wide Web: (https://en.wikipedia.org/wiki/Mean_squared_error).

[2] *Wikipedia.* Peak signal-to-noise ratio*[online].*
Available from World Wide Web:
(https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio).