KAIST
EE535 Digital Image Processing
Assignment 0

Honi Selahaddin (20226089)

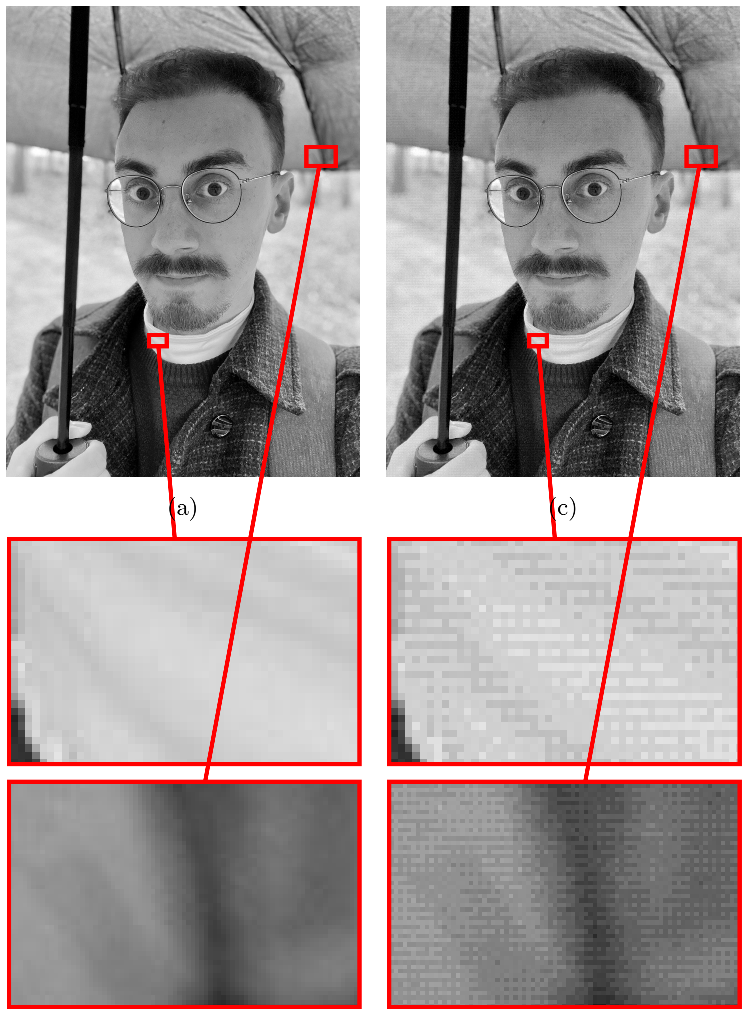
April, 2022

1 Dithering

Let a gray-scaled image I has $H \times W$ dimensions, where H and W represent spatial height and spatial width whose intensity values stored by 8-bits. If reducing the storage size of I is desired, decreasing spatial resolution or intensity resolution are two possible solutions. Changing H and W affects the size of I a lot; but, this reduction also drops meaningful information. On the other hand, we can compress the size of I while preserving most of the information by performing intensity quantization followed by dithering.

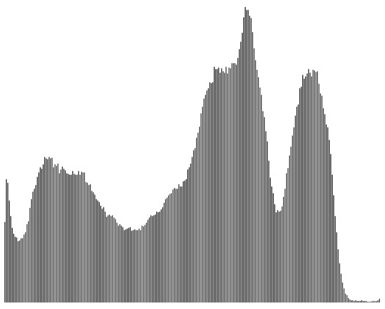
In this assignment, 8-bit input image I, shown in (a), quantized and dithered to 4-bit image (c), see Figure 1. Drop from $2^8 = 256$ gray-levels to $2^4 = 16$ gray-levels provides 50% size compression, roughly. It is not easy to discriminate the differences of the images by human-eye which is good. Therefore, two same regions in both input and dithered images are magnified in (b) and (d) parts, respectively. The regions belong to input image have smoother transitions compared to the regions belong to dithered image. The reason is straightforward: dithered image has only 16 gray-levels to cover the color palette; however, dithering is such an intelligent simulation of non-existing gray-levels that output image is still very close to input image. Dithered image tricks human-eye to see more than 16 gray-levels by replacing pixels with patterns. Moreover, number of gray-levels of input and dithered images can be computed via histogram as shown in Figure 2. These normalized histograms (a) and (b) have 256 and 16 color bins, respectively.

C++ code is available in Section 2.


(a) (c)

(b) (d)

Figure 1: Input and dithered images. (a) Gray-scaled input image, 256-levels (b) Magnified samples of input image (c) Dithered image, 16-levels (d) Magnified samples of dithered image


(a) Normalized histogram of input image, 256-levels

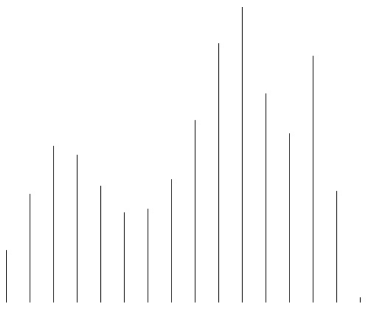

(b) Normalized histogram of dithered image, 16-levels

Figure 2: Image histograms.

2 Code

```
// @Author: Honi, Selahaddin (honis@kaist.ac.kr)
// @Date: 2022 April

#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/highgui.hpp>
#include <iostream>

using namespace std;
using namespace cv;

#define mmin(a, b) ((a) < (b) ? (a) : (b))

int dmat[4][4]={{0, 4, 2, 6}, {7, 12, 10, 14},{3, 7, 1, 5}, {11, 15, 9, 13}};

Mat getHistogram(Mat);

void main()
{
    // Read Image & Convert Gray-scale
    std::string image_path = samples::findFile("honi.jpg");
    Mat img_BGR = imread(image_path);
    Mat img;
    cvtColor(img_BGR, img, COLOR_BGR2GRAY);

    // Clone original image for initial dithered image
    Mat img_out = img.clone();

    // Pointers of original and output image data
    uchar* in  = img.data;  // input image with 256 levels
    uchar* out = img_out.data; // dithered image with 16 levels

    int i,j,m,n;
    int height = img.rows;
    int width = img.cols;

    for(i=0;i<height;i=i+4)
    for(j=0;j<width;j=j+4)
    {

        for(m=0;m<4;m++)
        for(n=0;n<4;n++)
        {
            double dithered = double(in[(i+m)*width+j+n]);
            out[(i+m)*width+j+n]= mmin(255, (floor((dithered/16)
                                       +((double)dmat[m][n]/16))*16));
        }

    }

    Mat hist = getHistogram(img);
    Mat histOut = getHistogram(img_out);

    // Write images
    imwrite("honi-gray.jpg", img);
    imwrite("honi-out.jpg", img_out);

    // Write histograms
    imwrite("honi-hist.jpg", hist);
    imwrite("honi-out-hist.jpg", histOut);

    // Release memory
    img.release();
    img_out.release();
}
```

```cpp
Mat getHistogram(Mat image)
{
    // Credits:  https://followtutorials.com/2013/01/intensity-histogram-using-
    // c-and-opencv-image-processing.html

    int histogram[256];

    // initialize all intensity values to 0
    for(int i = 0; i < 255; i++)
        histogram[i] = 0;

    // count each intensity values
    for(int y = 0; y < image.rows; y++)
        for(int x = 0; x < image.cols; x++)
            histogram[(int)image.at<uchar>(y,x)]++;

    // draw the histograms
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound((double) hist_w/256);

    Mat histImage(hist_h,
                hist_w,
                CV_8UC1,
                Scalar(255, 255, 255));

    // find the maximum intensity elem from histogram
    int max = histogram[0];
    for(int i = 1; i < 256; i++)
        if(max < histogram[i]) max = histogram[i];

    // normalize the histogram between 0 and histImage.rows
    for(int i = 0; i < 255; i++)
        histogram[i] =
        ((double)histogram[i]/max)*histImage.rows;

    // draw the intensity line for histogram
    for(int i = 0; i < 255; i++)
    {
        line(histImage, Point(bin_w*(i), hist_h),
            Point(bin_w*(i), hist_h - histogram[i]),
            Scalar(0,0,0), 1, 8, 0);
    }

    return histImage;
}
```