Image Processing

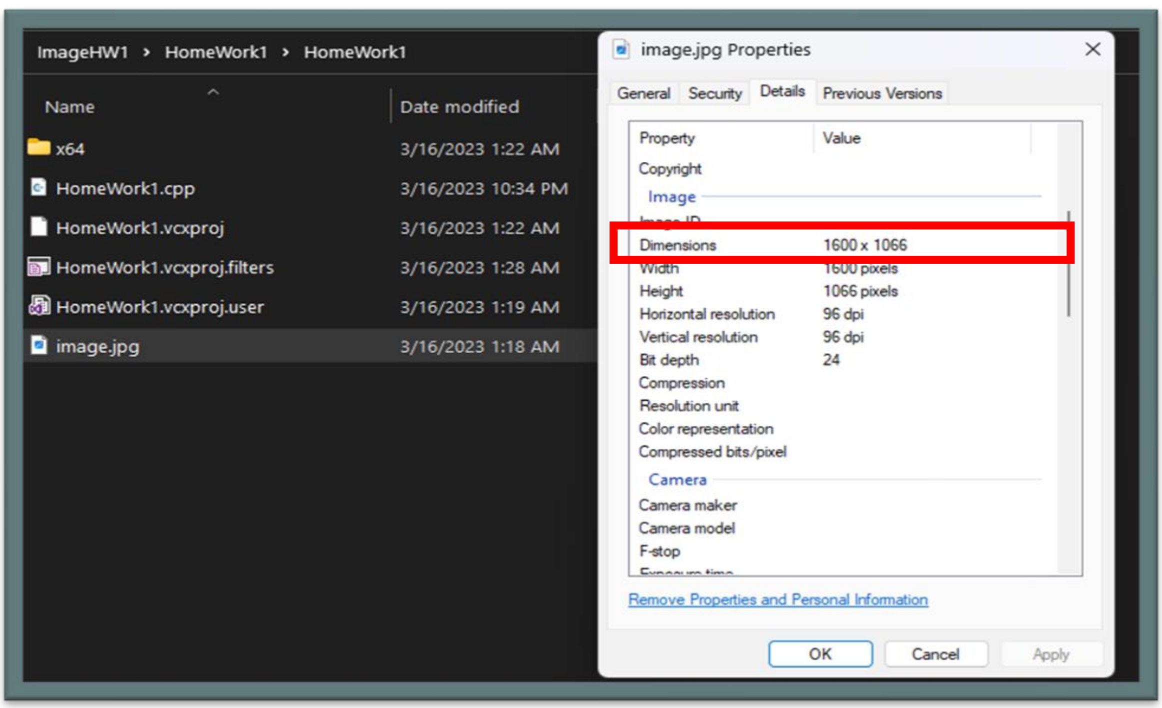
By Riham Muneer

Homework1 - report

Dr. Anas Toma

THE COLOR INPUT IMAGE

The original size of this image



THE GRAY-SCALE IMAGE

- Transfer the BRG image into g ray-scale image

  We can do that in two ways:

  - Read it as a grayscale image directly using "`imread`" method (which I used).

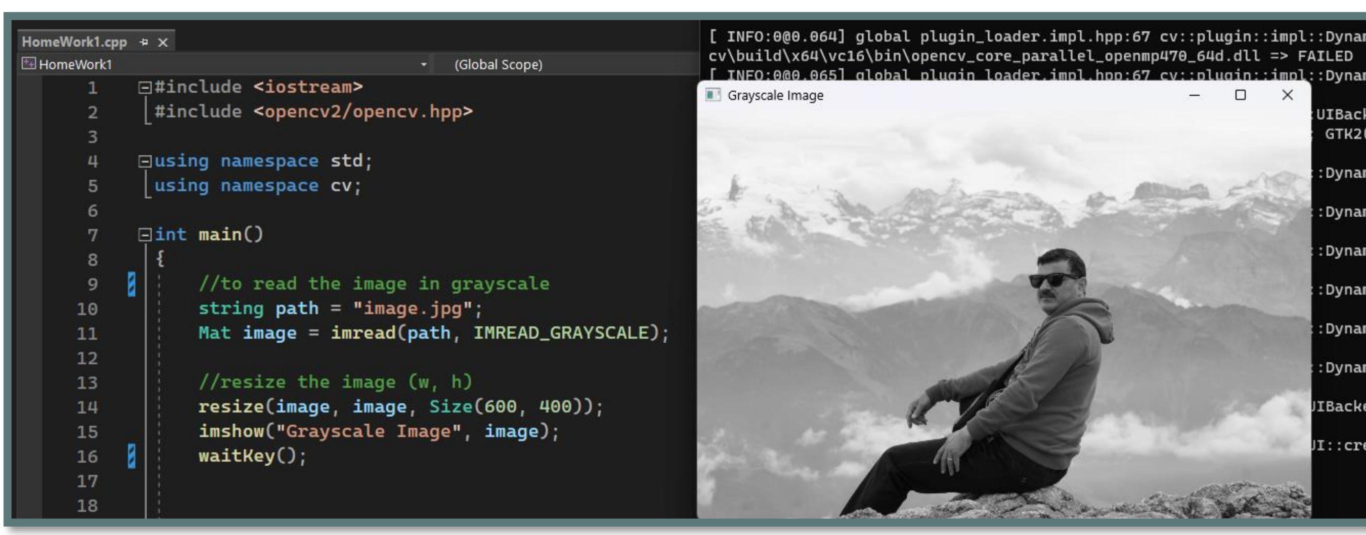
```
//to read the image in grayscale
string path = "image.jpg";
Mat image = imread(path, IMREAD_GRAYSCALE);
```

  - Read it as it is using "imread" method then transfer it into gray-scale using `cv :: cvtColor(source, destination, conversion code)` method.

- After that, I resized it into (600 X 400) pixels, and show it using "`imshow`" method.

```
//resize the image (w, h)
resize(image, image, Size(600, 400));
imshow("Grayscale Image", image);
waitKey();
```

- The result



ORIGINAL GRAY-SCALE IMAGE'S HISTOGRAM

Histogram: is a representation of frequency distribution of pixel intensities in the image.

- Use `calcHist (images, #images, #channels, mask, histogram, dimensions, histogram size, ranges)` to calculate the histogram of the gray-scale image.

```
//to make the histogram of the image
Mat histogram;
int histSize = 256; //maximum value
float channelRange[] = { 0.0, 256.0 };
const float* channelRangePTR = channelRange;

//images, number of images, channels, mask, hist, dims, histSize, ranges, uniform = true, accumulate = false
calcHist(&image, 1, 0, Mat(), histogram, 1, &histSize, &channelRangePTR);
```

  - **&image**: The gray-scale image
  - **1:** The number of images
  - **0:** The number of channels, (gray-scale image need only a single channel)
  - **Mat():** The mask to apply to the image (in our case we don't need a mask)
  - **histogram:** The resultant histogram
  - **1:** 1D histogram
  - **&histSize:** The number of bins on the x-axis
  - **&channelRangePTR:** The range of these bins

- Use normalize(input, output, alpha-the scaling factor, beta-the shifting factor, normalization type constant, desired output data type, mask) to normalize the histogram.

```
//to show the histogram
int hist_w = 600, hist_h = 400;
int bin_w = cvRound((double)hist_w / histSize);

//rows, columns, type, const scalar
// Use CV_8UC1, ..., CV_64FC4 to create 1-4 channel matrices, or CV_8UC(n), ..., CV_64FC(n)
Mat histImage(hist_h, hist_w, CV_8UC1, Scalar(0, 0, 0));

//normalize -> to scale and shift the pixel values of an image so that they fall within a specified range.
//input, output, alpha (the scaling factor), beta (the shifting factor), norm_type, dtype (the data type of the output), mask
normalize(histogram, histogram, 0, histImage.rows, NORM_MINMAX, -1, Mat());
```

  - **histogram:** the input histogram
  - **histogram:** the output histogram
  - **0:** the scaling factor
  - **histImage.rows:** the shifting factor
  - **NORM_MINMAX:** the type of the normalization
  - **-1:** it is negative so the output will have the same type as the input.
  - **Mat():** The mask

➢ Draw the histogram using line(inputOutput image, point1, point2, color, thickness, lineType, shift) method and show it

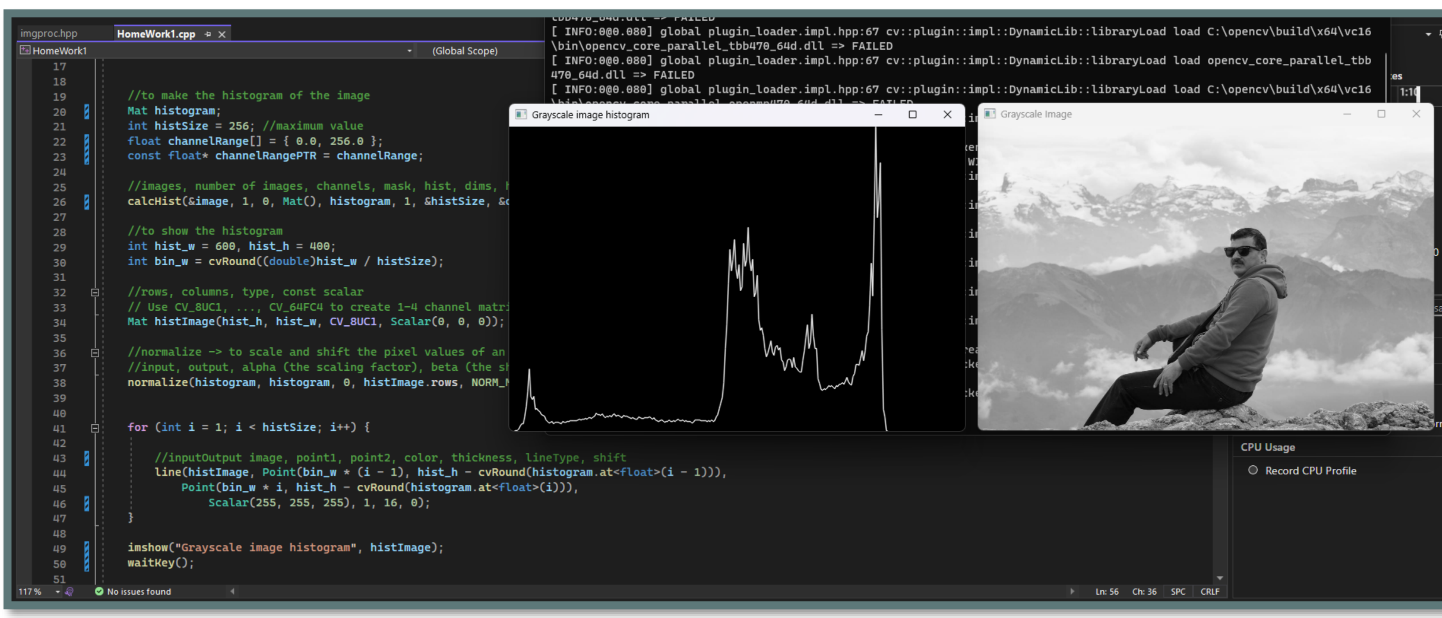
```
for (int i = 1; i < histSize; i++) {

    //inputOutput image, point1, point2, color, thickness, lineType, shift
    line(histImage, Point(bin_w * (i - 1), hist_h - cvRound(histogram.at<float>(i - 1))),
        Point(bin_w * i, hist_h - cvRound(histogram.at<float>(i))),
            Scalar(255, 255, 255), 1, 16, 0);
}

imshow("Grayscale image histogram", histImage);
waitKey();
```

- histImage: inputOutput image
- Point(x, y)
- Scalar(255, 255, 255): the color – white
- 1: the thickness of the line
- 16: the type of the line it can be -1, 4, 8, 16
- 0: shift

➢ The result



MODIFY THE BRIGHTNESS

Modifying the brightness with gamma: $s = c * r^{(1/G)}$

- Generate a random gamma value

```
//modify the brightness using a random gamma
double gamma = rand() / 25.0;
```

- Prepare a look up table with a palette of 1-256

```
Mat lookupTable(1, 256, CV_8U);
uchar* p = lookupTable.ptr();
for (int i = 0; i < 256; i++) {
    //saturate_cast to clamp the resulting value correctly
    //uchar range [0, 255]
    p[i] = saturate_cast<uchar>(pow(i / 255.0, gamma) * 255.0);
}
```

  - using "uchar" because it has a range {0, 255}
  - using saturate_cast method to clamp the resultant value correctly
  - we divide I by 255 then multiply it by 255 to scale it correctly

- Use the look up table with LUT(input image1, imput image2, output image) method and calculate the execution time of the process using a timer

```
//using look up table
Mat lookUpTableImage;
TickMeter timer1;
timer1.start();
LUT(image, lookupTable, lookUpTableImage);
timer1.stop();
```

- Modify the brightness for each pixel individually and calculate the execution time of the process using another timer

```
//modifying each pixel individually
TickMeter timer2;
Mat eachPixelImage(400, 600, CV_8UC1);
timer2.start();
for (int i = 0; i < 400; i++) {
    for (int j = 0; j < 600; j++) {
        eachPixelImage.at<uchar>(i, j) = saturate_cast<uchar>(pow(image.at<uchar>(i, j) / 255.0, gamma) * 255.0);
    }
}
timer2.stop();
```

- Show the difference between the execution time of the two methods and generate the histograms.

```
//show the resultant images
imshow("Look Up Table Image", lookUpTableImage);
imshow("Each Pixel Individually Image", eachPixelImage);
//waitKey();

//show the comparision result
cout << "Look Up Table method required time: " << timer1.getTimeMilli() << "ms" << endl
    << "Modifying Each Pixel Individually required time: " << timer2.getTimeMilli() << "ms" << endl;
//waitKey();



//draw  histograms for lookUpTableImage & eachPixelImage
MatND LUTHistogram, pixelHistogram;

calcHist(&lookUpTableImage, 1, 0, Mat(), LUTHistogram, 1, &histSize, &channelRangePTR);
calcHist(&eachPixelImage, 1, 0, Mat(), pixelHistogram, 1, &histSize, &channelRangePTR);


Mat LUTHistImage(hist_h, hist_w, CV_8UC1, Scalar(0, 0, 0)), pixelHistImage(hist_h, hist_w, CV_8UC1, Scalar(0, 0, 0));


normalize(LUTHistogram, LUTHistogram, 0, LUTHistImage.rows, NORM_MINMAX, -1, Mat());
normalize(pixelHistogram, pixelHistogram, 0, pixelHistImage.rows, NORM_MINMAX, -1, Mat());


for (int i = 1; i < histSize; i++) {

    line(LUTHistImage, Point(bin_w * (i - 1), hist_h - cvRound(LUTHistogram.at<float>(i - 1))),
        Point(bin_w * i, hist_h - cvRound(LUTHistogram.at<float>(i))),
        Scalar(255, 0, 0), 1, 16, 0);

    line(pixelHistImage, Point(bin_w * (i - 1), hist_h - cvRound(pixelHistogram.at<float>(i - 1))),
        Point(bin_w * i, hist_h - cvRound(pixelHistogram.at<float>(i))),
        Scalar(255, 0, 0), 1, 16, 0);
}

imshow("LUT image histogram", LUTHistImage);
imshow("Each pixel image histogram", pixelHistImage);
waitKey();
```

THE FINAL RESULT