Technical University of Cluj-Napoca
Faculty of Automation and Computer Science
Department of Computes Science

JPEG compression

Student: **Livia – Maria Mitrică**, gr. 30434

Supervisor: Robert VARGA

31/05/2021

Contents

1. Introduction

Today JPEG is very popular as a data format. But it is not really a data format. It is an compression method. There are many compression method and JPEG is one of them. It became a standard in 1992 and JPEG stands for *Joint Photographic Experts Group*. JPEG is a lossy compression method.

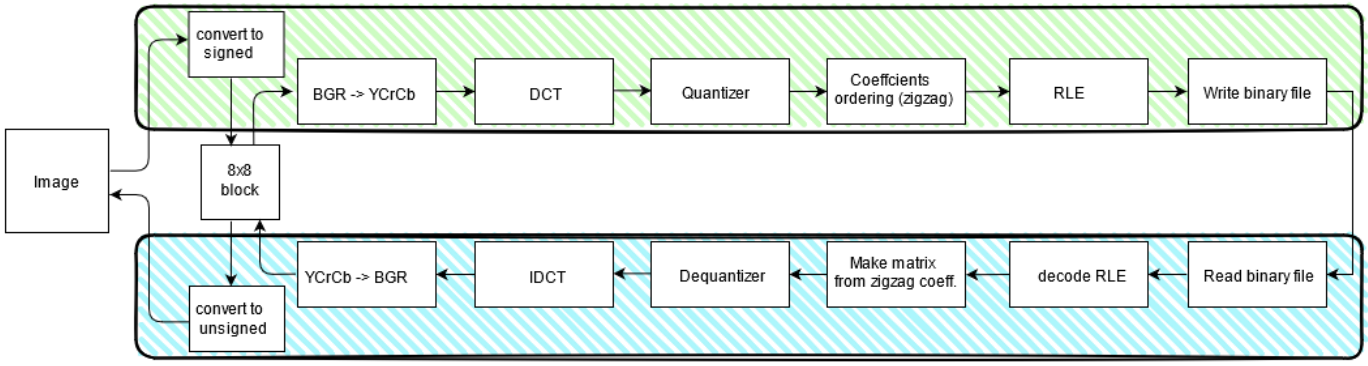
We perform such type of compression to reduce the size of the file without damaging its quality. By reducing the size we can store it in a huge amount which was not possible earlier. Reducing the size of images will also improve the efficiency of the system as it will give less load on it.

The aim of my project is to implement the JPEG compression algorithm on a color image. Save the compressed image as a binary file. Also implement the inverse function for decompression. Calculate the compression ratio for different images.

2. State of the art

The algorithms used for lossless image compression can be classified into four categories: entropy coding (Huffman and arithmetic coding), predictive coding (lossless JPEG,JPEG-LS, PNG, CALIC), transform coding (JPEG 2000, JPEG XR), and dictionary-basedcoding (LZW). All of the predictive and transform-based image compression techniquesuse an entropy coding strategy or Golomb coding as part of their compression procedure.

3. Proposed method



Algorithm steps:

1. Converting the image into signed. This is achieved by subtracting 128 from the initial image.
2. Split the image into 8x8 blocks. In case the width and height are not a multiple of 8 a padding is added.

3. Convert the image from BGR to YCrCb model. The operations that will be performed in the following steps need to be applied on each cahannel.
4. Apply DCT (Discrete cosine transform) given by the following formula:

$$X(k,l) = C(k)C(l)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)cos(\frac{(2m+1)k\pi}{2N})cos(\frac{(2n+1)l\pi}{2N})$$

$$k,l = 0,1...,N-1$$

其中

$$C(k) = C(l) = \begin{cases} \sqrt{\frac{1}{N}}, k,l = 0 \\ \sqrt{\frac{2}{N}}, else \end{cases}$$

The inverse of the transformation is the following:

$$\mathrm{x}(m,n) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} C(k)C(l)X(k,l)\cos(\frac{(2m+1)k\pi}{2N})\cos(\frac{(2n+1)l\pi}{2N})$$

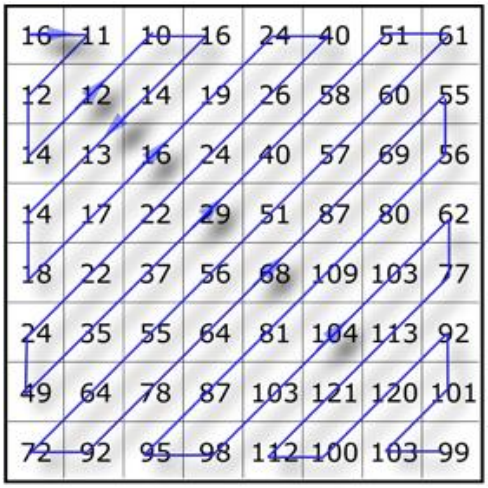
$$\text{其中，}\ \mathrm{C}(k) = C(l) = \begin{cases} \sqrt{\frac{1}{N}}, k = l = 0 \\ \sqrt{\frac{2}{N}}, else \end{cases}$$

Loosely speaking, the reason for this step is that the DCT tends to push most of the high intensity information (larger values) in the 8 x 8 block to the upper left-hand of C with the remaining values in C taking on relatively small values. The DCT is applied to each 8 x 8 block.

5. Multiply with the luminance quantization matrix. The matrix I used has the following values:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

6. Order the coefficients of the obtained matrix in zig zag manner:



7. Apply the run length encoding algorithm on the obtained matrix.
8. Save the results in a binary file.

As far as the reading and writing in the binary file is concerned, I decided to structure it the following way. Firstly I saved the original height and weight of the image, followed by the height and weight as a multiple of 8. Then follow the encodings: each encoding starts with the size of the vector followed by the actual pair of values, e.g. 4 5 1 0 5 which means 5 0 0 0 0 0.

4. Library/Functions used

The application relies on the OpenCV library. Apart from basic functions that include reading matrices and performing arithmetic operations on them I used a function to convert from BGR to YCrCb and viceversa.

The functions for the DCT and IDCT, zigzag coefficients and run length encoding I implemented myself.

5. Implementation details

The most difficult part were the DCT and IDCT functions, along with the reconstruction of the initial image from the read blocks.

The DCT matrix elements are computed using the formula inserted below. Each element of the result is computed as a sum of basis images. The basis images are actually the product of vertical and horizontal cosine.

```
Mat computeDCT(Mat1f src) {
    Mat1f dct(8, 8, CV_32FC1);

    float ck, cl, dct1, temp;
```

```
    for (int k = 0; k < src.rows; k++) {
            for (int l = 0; l < src.cols; l++) {
                    if (k == 0)
                            ck = 1.0 / sqrt(src.rows);
                    else
                            ck = sqrt(2.0) / sqrt(src.rows);
                    if (l == 0)
                            cl = 1.0 / sqrt(src.cols);
                    else
                            cl = sqrt(2.0) / sqrt(src.cols);

                    // sum will temporarily store the sum of
                    // cosine signals
                    temp = 0;
                    for (int m = 0; m < src.rows; m++) {
                            for (int n = 0; n < src.cols; n++) {
                                    dct1 = src(m,n) *
                                            cos((2 * m + 1) * k * PI / (2 * src.rows)) *
                                            cos((2 * n + 1) * l * PI / (2 * src.cols));
                                    temp = temp + dct1;
                            }
                    }
                    dct(k,l) = ck * cl * temp;
            }
    }

    return dct;
}
```

Similarly we compute the inverse of the transformation.

```
Mat computeIDCT(Mat1f src) {
    Mat1s idct(8, 8, CV_32S);

    double ck, cl;

    for (int m = 0;m < 8;m++)
    {
            for (int n = 0;n < 8;n++)
            {
                    double temp = 0.0;
                    for (int k = 0;k < 8;k++)
                    {
                            for (int l = 0;l < 8;l++)
                            {
                                    if (k == 0)
                                            ck = sqrt(1.0 / 8);
                                    else
                                            ck = sqrt(2.0 / 8);
                                    if (l == 0)
                                        cl = sqrt(1.0 / 8);
                                    else
                                            cl = sqrt(2.0 / 8);

                                    temp += ck * cl * src(k,l) *
                                                    cos((2 * m + 1) * k * PI / (2 * 8)) *
                                                    cos((2 * n + 1) * l * PI / (2 * 8));

                            }
                    }
                    idct(m,n) = around(temp);
            }
    }

    return idct;
}
```

Here I am reconstructing the initial image by successively adding the 8x8 blocks. The current row and column have been computed previously like :

```
if ( (col + 8) == W8 ) {
    row += 8;
    }

col = i % blocksCol * 8;
```

then each block is put to its correct position. We check if col+8 equals the width of the image so that we know when we reached the end of a line. When computing the column we first take into account the relative position of the block, but then we need to multiply it with 8 since each of the blocks we have already added has the number of columns equal to 8.

```
Mat_<Vec3s> putBlock(Mat_<Vec3s> src, Mat_<Vec3s> block, int row, int col) {

    for (int i = row; i < row + 8; i++) {
            for (int j = col; j < col + 8; j++) {
                    src(i, j) = (row == 0) ? (block(row, j % 8)) : (block(i % row, j % 8));
            }
    }

    return src;
}
```

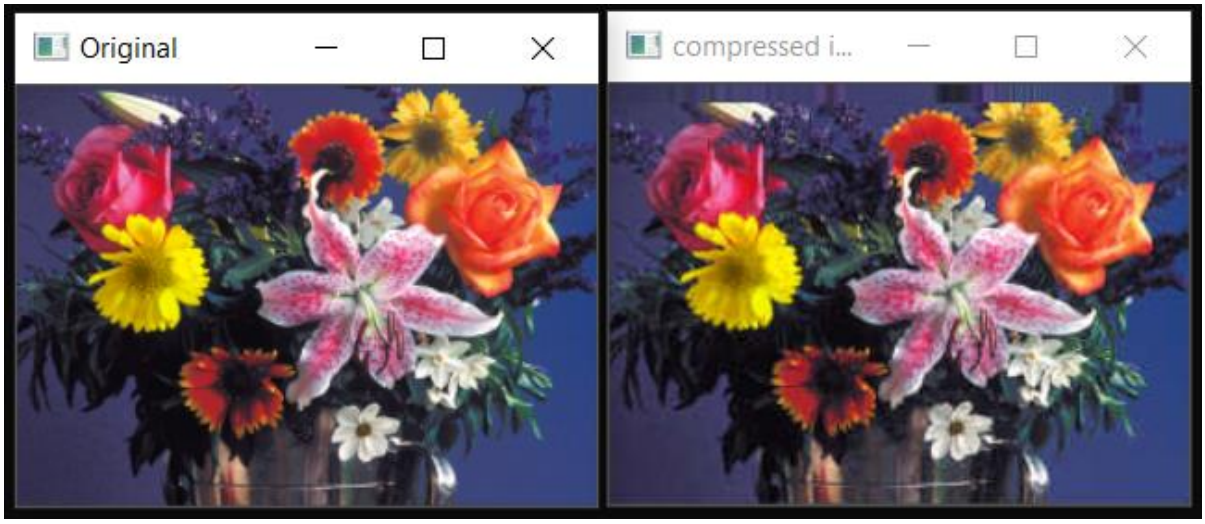
6. User manual

The user must run the application and after that a pop will open with the current folder. The user is required to select a picture. The picture is compressed, saved into a file and then decompressed and displayed on the screen. The jpeg image is also saved in the folder where the initial image is selected.

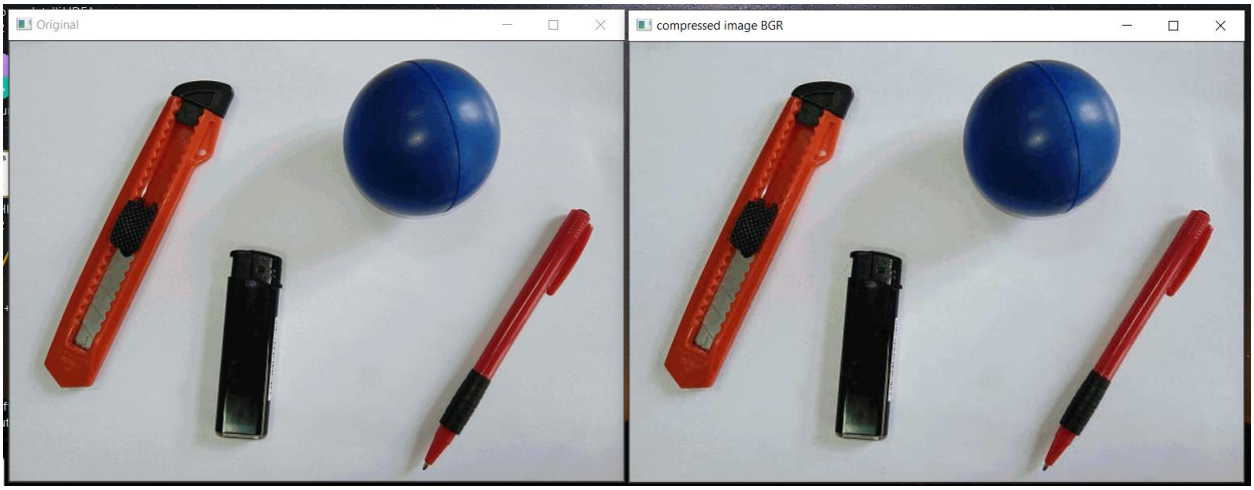
7. Experimental results

My dataset included approximately 10 colored images. I will attach some of the obtained results below:

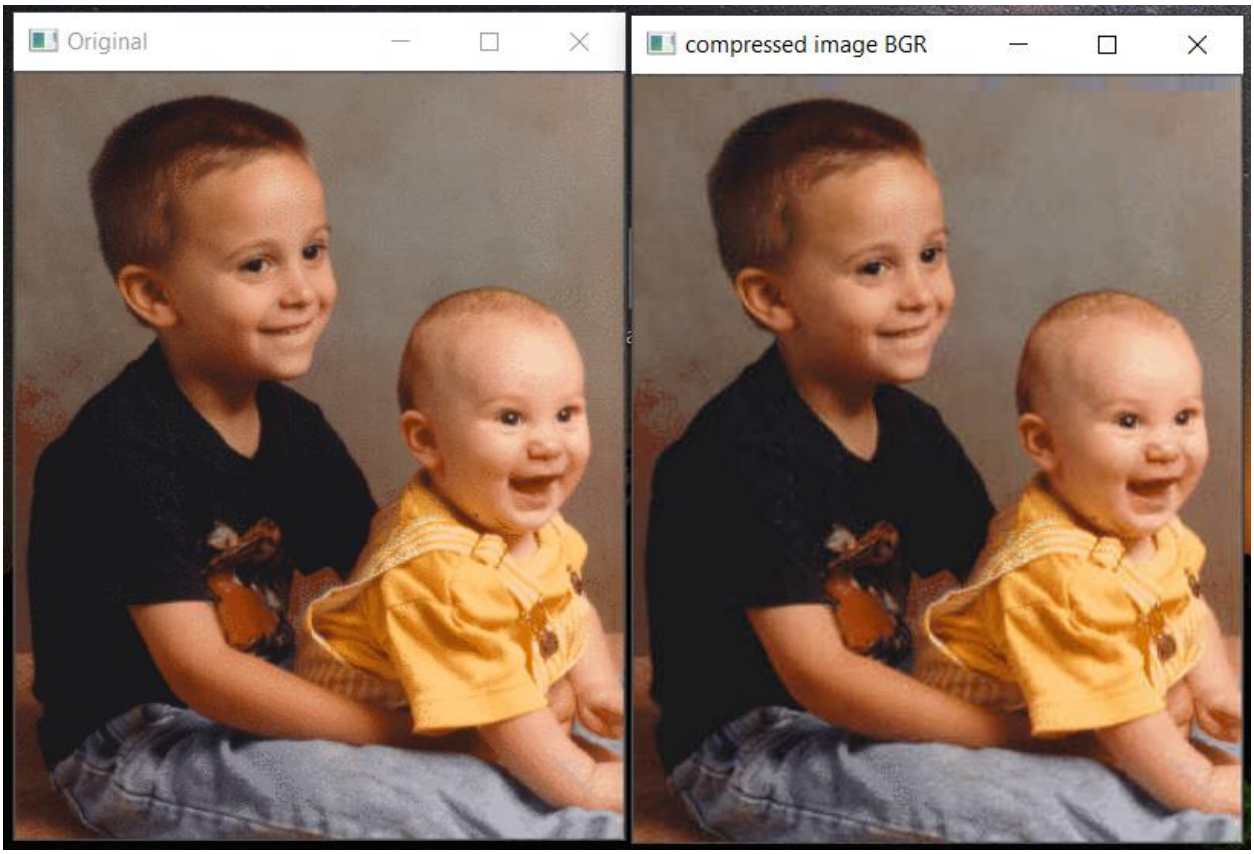
Flowers – compression ratio: 5.94



Objects – compression ratio: 7.22



Kids - compression ratio: 4.07



Compression ratio depends on how large the values in the quantization matrix are. In case of 10:1 ratio the compression is achievable without noticeable loss, but in cases of 100:1 the

artifacts become noticeable. In my case since the ratio is smaller than 1, the difference between images is not very noticeable.

For a high quality compression, the subsampling can be skipped and the quantization matrix can be selected that way, that the information loss is low. For high compression settings, the subsampling is turned on and the quantization matrix is selected to force most coefficients to 0. In that case, the image get clearly visible artifacts after decompression.

8. Conclusions

This application successfully compresses an image with a small loss, therefore the main objective of the project has been accomplished. However, in case of bigger images the operation takes longer to perform. One of the limitations is that the user cannot switch between different compression ratios.

One of the future developments would be increasing the speed of the execution and allowing different compression rates.

9. Bibliography

1. http://www.tutorialspoint.com/dip/Introduction_to_JPEG_compression.htm
2. http://www0.cs.ucl.ac.uk/teaching/GZ05/07-images.pdf
3. https://www.image-engineering.de/library/technotes/745-how-does-the-jpeg-compression-work
4. https://www.programmersought.com/article/38045304504/