Работа 2. Исследование каналов и JPEG-сжатия

автор: Лоев В.А.

дата: 3.03.2021
https://mysvn.ru/LOEV_V_A/loev_v_a/prj.labs/lab_2/

Задание

1. В качестве тестового использовать изображение data/cross_0256x0256.png
2. Сохранить тестовое изображение в формате JPEG с качеством 25%.
3. Используя cv::merge и cv::split сделать "мозаику" с визуализацией каналов для исходного тестового изображения и JPEG-версии тестового изображения

- левый верхний - трехканальное изображение
- левый нижний - монохромная (черно-зеленая) визуализация канала G
- правый верхний - монохромная (черно-красная) визуализация канала R
- правый нижний - монохромная (черно-синяя) визуализация канала B

4. Результы сохранить для вставки в отчет
5. Сделать мозаику из визуализации гистограммы для исходного тестового изображения и JPEG-версии тестового изображения, сохранить для вставки в отчет.

Результаты

Рис. 1. Тестовое изображение после сохранения в формате JPEG с качеством 25%

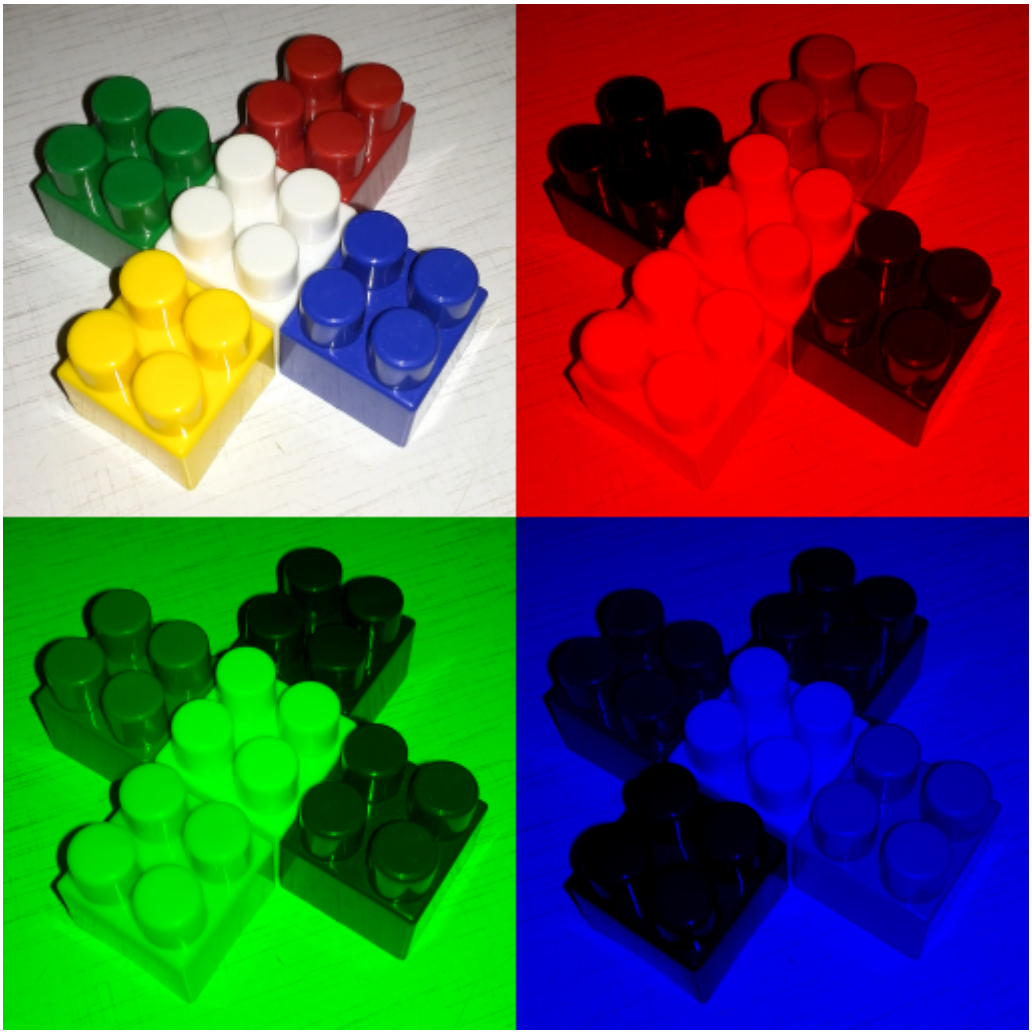

Рис. 2. Визуализация каналов исходного тестового изображения

Рис. 3. Визуализация каналов JPEG-версии тестового изображения

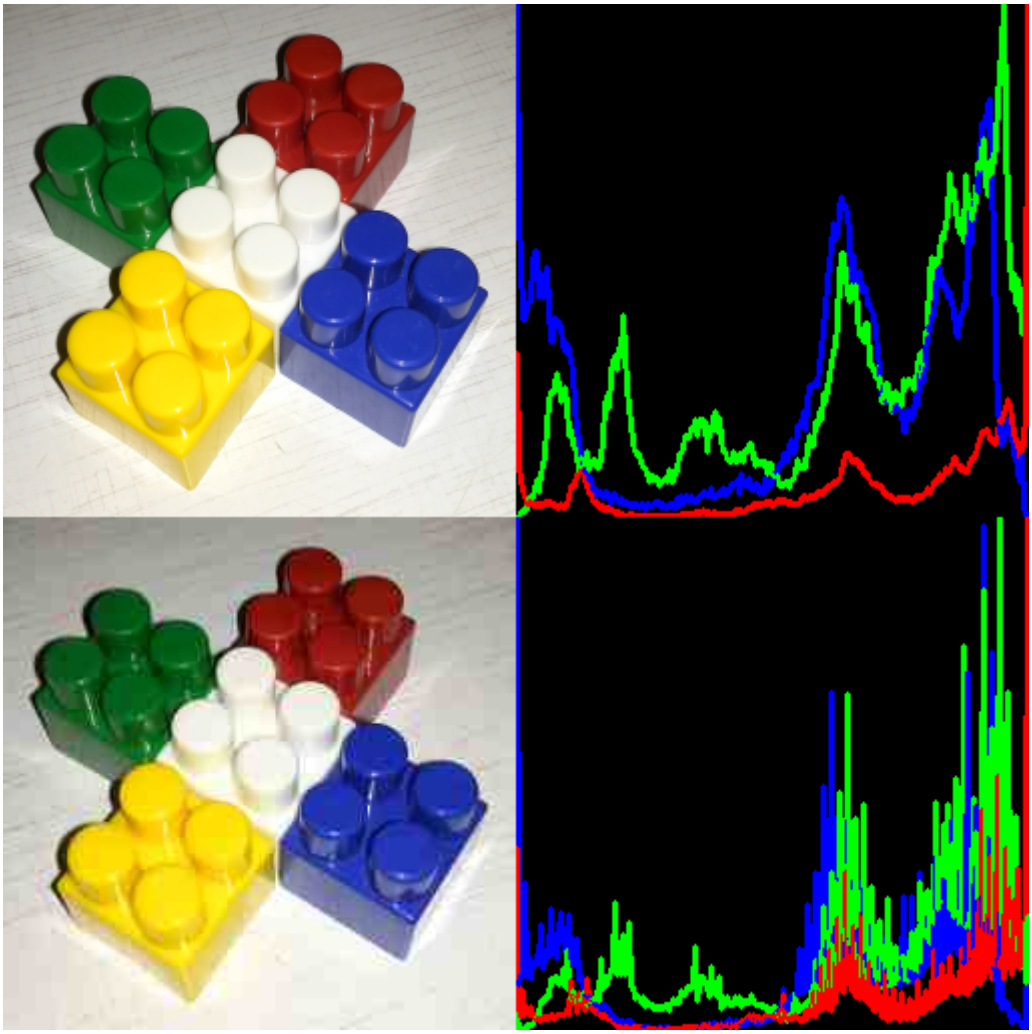

Рис. 3. Визуализация гистограм исходного и JPEG-версии тестового изображения

Текст программы

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

using namespace cv;
using namespace std;


Mat Colour_mosaic(Mat src) {

    //Split and merge src
    Mat splt[3], all_merged, r_merged, g_merged, b_merged;

    split(src, splt);

    Mat zero_channel = Mat::zeros(src.rows, src.cols, CV_8UC1);
    vector<Mat> all_channels = { splt[0], splt[1], splt[2] };
    vector<Mat> red_channel = { zero_channel, zero_channel, splt[2] };
    vector<Mat> green_channel{ zero_channel, splt[1], zero_channel };
    vector<Mat> blue_channel{ splt[0], zero_channel, zero_channel };

    merge(all_channels, all_merged);
    merge(red_channel, r_merged);
    merge(green_channel, g_merged);
    merge(blue_channel, b_merged);

    //make empty template
```

```
    Mat result(src.rows * 2, src.cols * 2, CV_8UC3);

    //fill channels one by one
    Rect2d rc = { 0, 0, 256, 256 };
    all_merged.copyTo(result(Rect(rc.x, rc.y, all_merged.cols,
all_merged.rows)));
    rc.x += rc.width;
    r_merged.copyTo(result(Rect(rc.x, rc.y, r_merged.cols, r_merged.rows)));
    rc.y += rc.height;
    b_merged.copyTo(result(Rect(rc.x, rc.y, b_merged.cols, b_merged.rows)));
    rc.x -= rc.width;
    g_merged.copyTo(result(Rect(rc.x, rc.y, g_merged.cols, g_merged.rows)));


    return result;
}

Mat Draw_histogram(Mat src) {

    //make empty template
    Mat res(src.rows, src.cols * 2, CV_8UC3);

    //split src image into channels
    vector<Mat> bgr_planes;
    split(src, bgr_planes);

    //define histogram parameters
    int histSize = 256;
    float range[] = { 0, 256 };
    const float* histRange = { range };
    bool uniform = true, accumulate = false;
    Mat b_hist, g_hist, r_hist;

    //calculate histograms
    calcHist(&bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange,
uniform, accumulate);
    calcHist(&bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange,
uniform, accumulate);
    calcHist(&bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange,
uniform, accumulate);

    //make image for histogram
    int hist_w = 256, hist_h = 256;
    Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));

    //normolize and draw histogram
    normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
    normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
    normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());

    for (int i = 1; i < histSize; i++)
    {
        line(histImage, Point((i - 1), hist_h - cvRound(b_hist.at<float>(i -
1))),
            Point((i), hist_h - cvRound(b_hist.at<float>(i))),
            Scalar(255, 0, 0), 2, 8, 0);
        line(histImage, Point((i - 1), hist_h - cvRound(g_hist.at<float>(i -
1))),
```

```
            Point((i), hist_h - cvRound(g_hist.at<float>(i))),
            Scalar(0, 255, 0), 2, 8, 0);
        line(histImage, Point((i - 1), hist_h - cvRound(r_hist.at<float>(i -
1))),
            Point((i), hist_h - cvRound(r_hist.at<float>(i))),
            Scalar(0, 0, 255), 2, 8, 0);
    }

    //concatenate src and histogram
    hconcat(src, histImage, res);
    return res;
}


int main() {

    //load source image
    string image_path = samples::findFile("cross_0256x0256.png");
    Mat original_img = imread(image_path, IMREAD_COLOR);
    imshow("original_img", original_img);

    //Save original_img in JPEG format
    vector<int> compression_params;
    compression_params.push_back(IMWRITE_JPEG_QUALITY);
    compression_params.push_back(25);
    imwrite("cross_0256x0256_025.jpg", original_img, compression_params);

    //load JPEG image
    string image_path_jpeg = samples::findFile("cross_0256x0256_025.jpg");
    Mat jpeg_img = imread(image_path_jpeg);
    imshow("JEPG_img", jpeg_img);

    //color mosaic for original image
    Mat orig_colors = Colour_mosaic(original_img);
    imwrite("cross_0256x0256_png_channels.png", orig_colors);
    imshow("colors_original", orig_colors);

    //color mosaic for jpeg image
    Mat jpeg_colors = Colour_mosaic(jpeg_img);
    imwrite("cross_0256x0256_jpg_channels.png", jpeg_colors);
    imshow("colors_jpeg", jpeg_colors);


    //draw histogram for each image
    Mat histImage_1 = Draw_histogram(original_img);
    Mat histImage_2 = Draw_histogram(jpeg_img);

    //concatenate histograms
    Mat hist_mosaic_res(512, 512, CV_8UC3);
    vconcat(histImage_1, histImage_2, hist_mosaic_res);
    imwrite("cross_0256x0256_hists.png", hist_mosaic_res);
    imshow("histograms", hist_mosaic_res);

    waitKey(0);
    return 0;
}
```