Работа 2. Исследование каналов и JPEG-сжатия

автор: Балаев А. А.
дата: 22.02.2022

url: https://github.com/BalaevAA/balaev_a_a.git

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. Результаты 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%

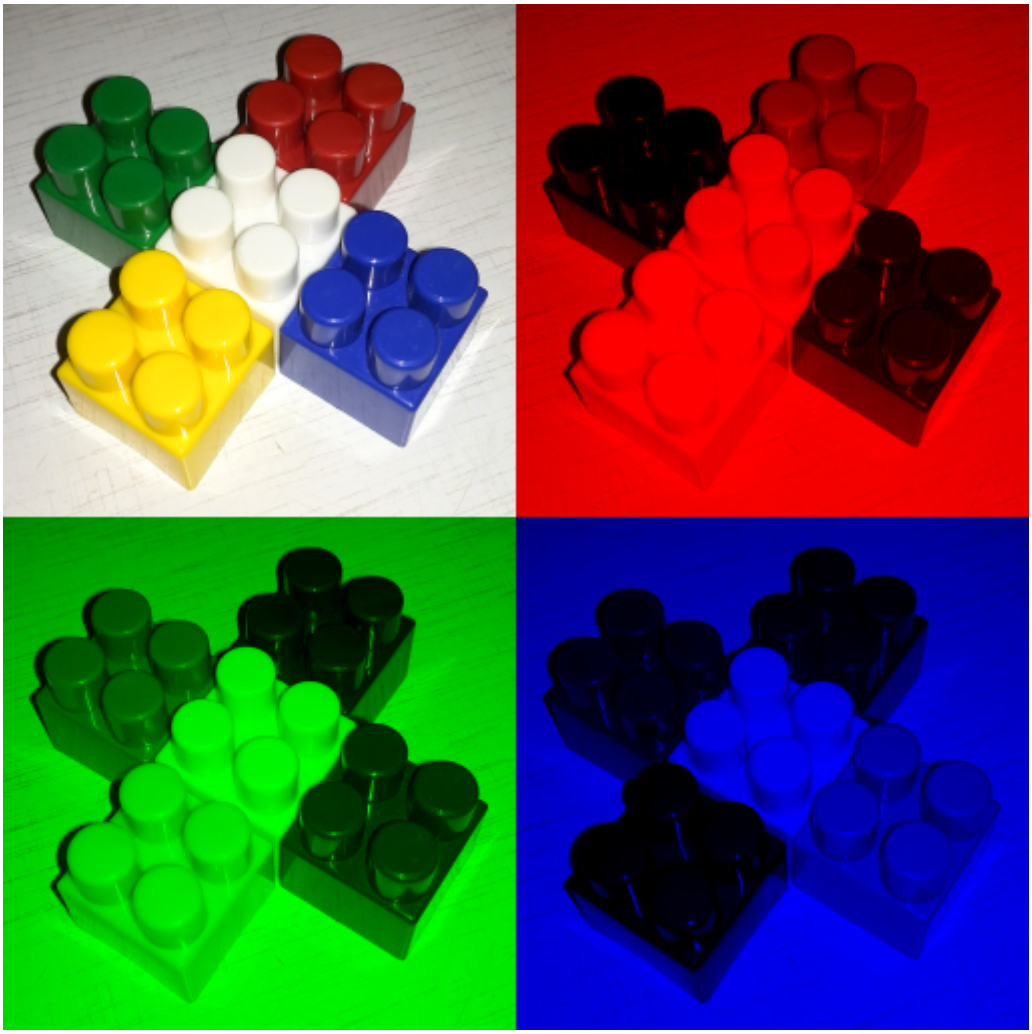

Рис. 2. Визуализация каналов исходного тестового изображения

Рис. 3. Визуализация каналов JPEG-версии тестового изображения

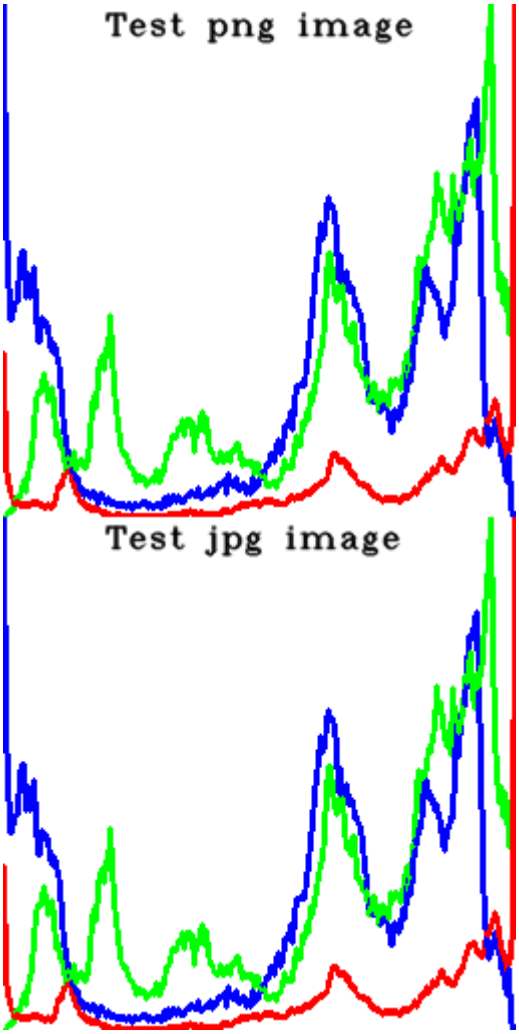

Рис. 3. Визуализация гистограмм исходного и JPEG-версии тестового изображения

Текст программы

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

cv::Mat makeMosaic(cv::Mat src) {
    cv::Mat rgbChannel[3];
    cv::Mat res_img(512, 512, CV_8UC3);
    cv::split(src, rgbChannel);
    cv::Mat zero_channel = cv::Mat::zeros(cv::Size(256, 256), CV_8UC1);

    cv::merge(std::vector<cv::Mat>{ rgbChannel[0], zero_channel, zero_channel },
rgbChannel[0]);
    cv::merge(std::vector<cv::Mat>{ zero_channel, rgbChannel[1], zero_channel },
rgbChannel[1]);
    cv::merge(std::vector<cv::Mat>{ zero_channel, zero_channel, rgbChannel[2] },
rgbChannel[2]);

    //cv::imshow("b", rgbChannel[0]);
    //cv::imshow("g", rgbChannel[1]);
    //cv::imshow("r", rgbChannel[2]);

    src.copyTo(res_img(cv::Rect(0, 0, 256, 256)));
    rgbChannel[0].copyTo(res_img(cv::Rect(256, 256, 256, 256)));
    rgbChannel[1].copyTo(res_img(cv::Rect(0, 256, 256, 256)));
    rgbChannel[2].copyTo(res_img(cv::Rect(256, 0, 256, 256)));
```

```
    return res_img;
}

cv::Mat makeHist(cv::Mat src) {
    std::vector<cv::Mat> bgr_planes;
    cv::split(src, bgr_planes);
    int histSize = 256;
    float range[] = { 0, 256 }; //the upper boundary is exclusive
    const float* histRange[] = { range };
    bool uniform = true, accumulate = false;
    cv::Mat b_hist, g_hist, r_hist;
    cv::calcHist(&bgr_planes[0], 1, 0, cv::Mat(), b_hist, 1, &histSize,
histRange, uniform, accumulate);
    cv::calcHist(&bgr_planes[1], 1, 0, cv::Mat(), g_hist, 1, &histSize,
histRange, uniform, accumulate);
    cv::calcHist(&bgr_planes[2], 1, 0, cv::Mat(), r_hist, 1, &histSize,
histRange, uniform, accumulate);
    int hist_w = 256, hist_h = 256;
    int bin_w = cvRound((double)hist_w / histSize);
    cv::Mat histImage(hist_h, hist_w, CV_8UC3, cv::Scalar(255, 255, 255));
    cv::normalize(b_hist, b_hist, 0, histImage.rows, cv::NORM_MINMAX, -1,
cv::Mat());
    cv::normalize(g_hist, g_hist, 0, histImage.rows, cv::NORM_MINMAX, -1,
cv::Mat());
    cv::normalize(r_hist, r_hist, 0, histImage.rows, cv::NORM_MINMAX, -1,
cv::Mat());
    for (int i = 1; i < histSize; i++)
    {
        line(histImage, cv::Point(bin_w * (i - 1), hist_h -
cvRound(b_hist.at<float>(i - 1))),
            cv::Point(bin_w * (i), hist_h - cvRound(b_hist.at<float>(i))),
            cv::Scalar(255, 0, 0), 2, 8, 0);
        line(histImage, cv::Point(bin_w * (i - 1), hist_h -
cvRound(g_hist.at<float>(i - 1))),
            cv::Point(bin_w * (i), hist_h - cvRound(g_hist.at<float>(i))),
            cv::Scalar(0, 255, 0), 2, 8, 0);
        line(histImage, cv::Point(bin_w * (i - 1), hist_h -
cvRound(r_hist.at<float>(i - 1))),
            cv::Point(bin_w * (i), hist_h - cvRound(r_hist.at<float>(i))),
            cv::Scalar(0, 0, 255), 2, 8, 0);
    }
    return histImage;
}

int main() {
    //1
    std::string img_path = "../../../data/cross_0256x0256.png";
    cv::Mat img = cv::imread(img_path);
    if (img.empty()) {
        std::cout << "Could not read the image: " << img_path << std::endl;
        return 1;
    }
    //cv::imshow("output", img);

    //2
    cv::imwrite("cross_0256x0256_025.jpeg", img, { cv::IMWRITE_JPEG_QUALITY , 25
});
```

```
    cv::Mat img_025 = cv::imread("cross_0256x0256_025.jpeg");
    //3
    cv::Mat rgbChannel[3];
    cv::Mat res_img = makeMosaic(img);
    cv::imwrite("cross_0256x0256_png_channels.png", res_img);

    cv::Mat rgbChannel_025[3];
    cv::Mat res_img_025 = makeMosaic(img_025);
    cv::imwrite("cross_0256x0256_jpg_channels.png", res_img_025);


    //5
    cv::Mat res_hists(512, 256, CV_8UC3);
    cv::Mat hist_img = makeHist(img);
    hist_img.copyTo(res_hists(cv::Rect(0, 0, 256, 256)));
    cv::putText(res_hists, "Test png image", cv::Point(50, 15),
        cv::FONT_HERSHEY_COMPLEX_SMALL, 0.8, cv::Scalar(0, 0, 0), 1,
cv::LINE_AA);
    cv::Mat hist_img_025 = makeHist(img);
    hist_img.copyTo(res_hists(cv::Rect(0, 256, 256, 256)));
    cv::putText(res_hists, "Test jpg image", cv::Point(50, 271),
        cv::FONT_HERSHEY_COMPLEX_SMALL, 0.8, cv::Scalar(0, 0, 0), 1,
cv::LINE_AA);


    cv::imwrite("cross_0256x0256_hists.png", res_hists);

    return 0;
}
```