# Работа 2. Исследование каналов и JPEG-сжатия

автор: Конев Т.В. дата: 2022-02-21T15:43:29

Github : https://github.com/timakon/KonevCV/tree/main/prj.labs/lab02

## 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%

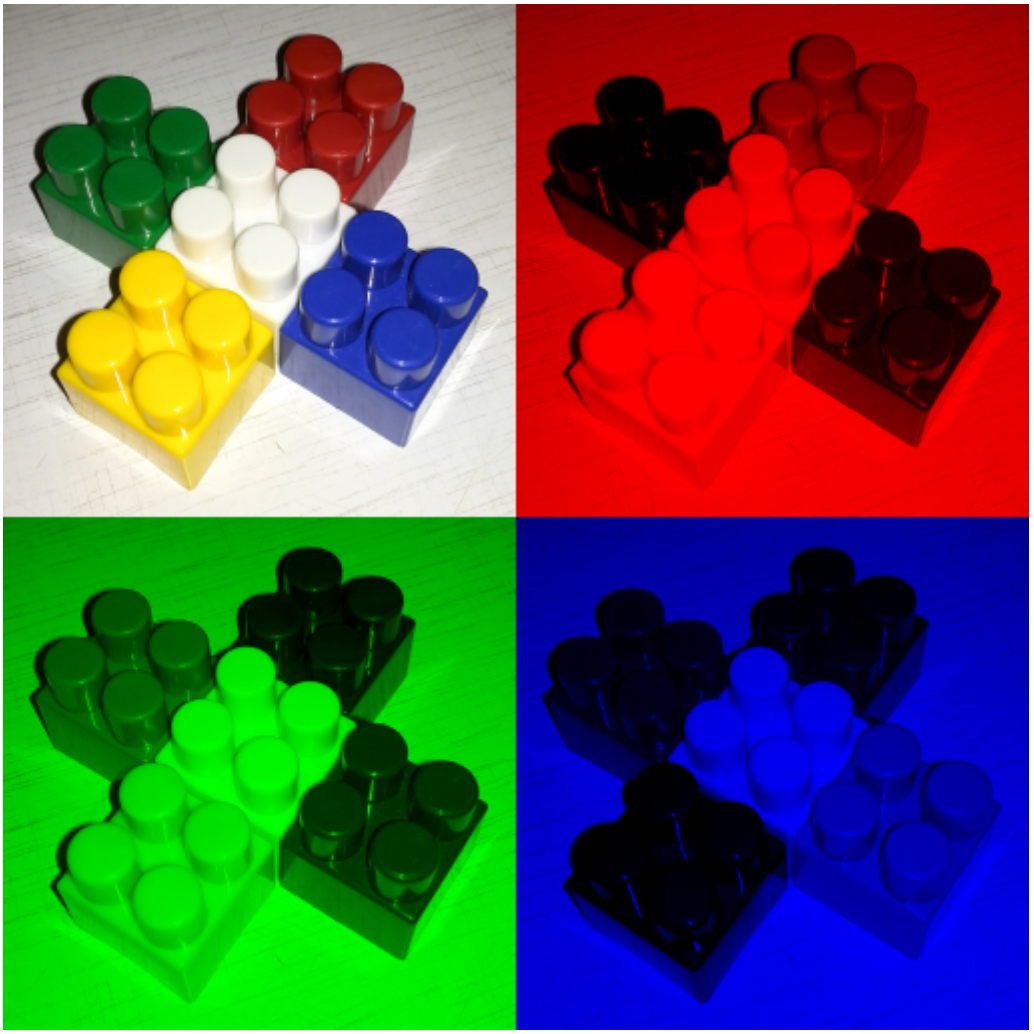

Рис. 2. Визуализация каналов исходного тестового изображения

Рис. 3. Визуализация каналов JPEG-версии тестового изображения

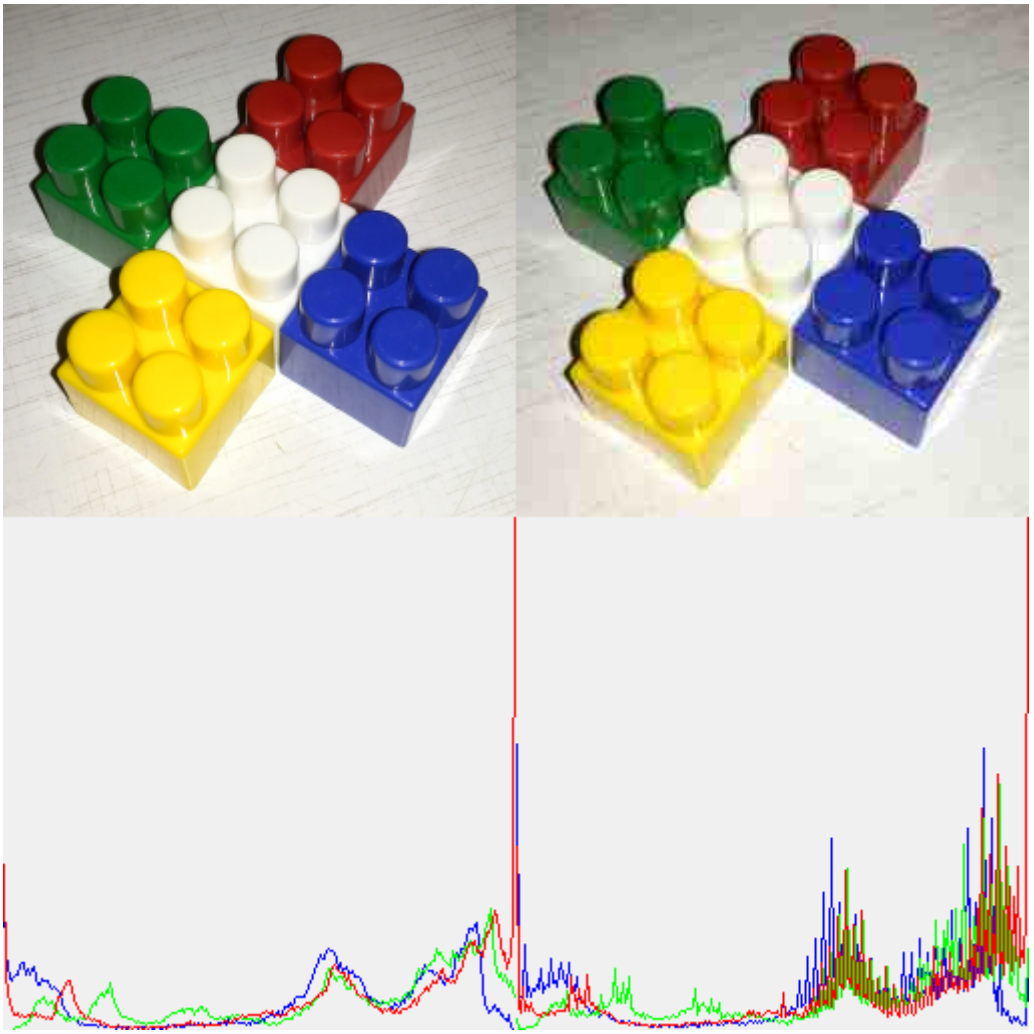

Рис. 3. Визуализация гистограм исходного и JPEG-версии тестового изображения

## Текст программы

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

void getBrightness(cv::Mat grayscale_image, int* pixel_array) {
	for (int i = 0; i < 256; i++) {
		pixel_array[i] = 0;
	}
	for (int i = 0; i < grayscale_image.rows; i++) {
		for (int j = 0; j < grayscale_image.cols; j++) {
			pixel_array[grayscale_image.at<uchar>(i, j)]++;
		}
	}
}

int main() {
	cv::Mat image_png = cv::imread("../../../data/cross_0256x0256.png");

	std::vector<int> quality;
	cv::imwrite("cross_0256x0256_025.jpg", image_png, { cv::IMWRITE_JPEG_QUALITY, 25 });

	cv::Mat channels_split[3];
	cv::split(image_png, channels_split);

	cv::Mat zero_matrix(256, 256, CV_8UC1);
	zero_matrix = 0;
```

```
    cv::Mat channels_merge_red[3] = { zero_matrix, zero_matrix, channels_split[2] };
    cv::Mat channels_merge_green[3] = { zero_matrix, channels_split[1], zero_matrix };
    cv::Mat channels_merge_blue[3] = { channels_split[0], zero_matrix, zero_matrix };

    cv::Mat red;
    cv::Mat green;
    cv::Mat blue;

    cv::merge(channels_merge_red, 3, red);
    cv::merge(channels_merge_green, 3, green);
    cv::merge(channels_merge_blue, 3, blue);

    cv::Mat mosaic_top;
    cv::Mat mosaic_bottom;
    cv::Mat mosaic;

    cv::hconcat(image_png, red, mosaic_top);
    cv::hconcat(green, blue, mosaic_bottom);
    cv::vconcat(mosaic_top, mosaic_bottom, mosaic);

    cv::imwrite("cross_0256x0256_png_channels.png", mosaic);

    cv::imwrite("image_jpg.jpg", image_png, { cv::IMWRITE_JPEG_QUALITY, 100 });

cv::Mat histogram_png(256, 256, CV_8UC3, cv::Scalar(240, 240, 240));
int blue_pixels_count[256];
int green_pixels_count[256];
int red_pixels_count[256];

getBrightness(channels_split[0], blue_pixels_count);
getBrightness(channels_split[1], green_pixels_count);
getBrightness(channels_split[2], red_pixels_count);

    cv::Mat image_jpg = cv::imread("./image_jpg.jpg");

    cv::split(image_jpg, channels_split);

    cv::Mat channels_merge_red_jpg[3] = { zero_matrix, zero_matrix, channels_split[2] };
    cv::Mat channels_merge_green_jpg[3] = { zero_matrix, channels_split[1], zero_matrix };
    cv::Mat channels_merge_blue_jpg[3] = { channels_split[0], zero_matrix, zero_matrix };

    cv::merge(channels_merge_red_jpg, 3, red);
    cv::merge(channels_merge_green_jpg, 3, green);
    cv::merge(channels_merge_blue_jpg, 3, blue);

    cv::hconcat(image_png, red, mosaic_top);
    cv::hconcat(green, blue, mosaic_bottom);
    cv::vconcat(mosaic_top, mosaic_bottom, mosaic);

    cv::imwrite("cross_0256x0256_jpg_channels.png", mosaic);

int bin_w = 1; //height scaling factor

int max = 0;
for (int i = 0; i < 256; i++) {
    if (max < blue_pixels_count[i]) {
        max = blue_pixels_count[i];
    }
}
for (int i = 0; i < 256; i++) {
    if (max < green_pixels_count[i]) {
```

```
        max = green_pixels_count[i];
    }
}
for (int i = 0; i < 256; i++) {
    if (max < red_pixels_count[i]) {
        max = red_pixels_count[i];
    }
}

std::cout << max;

for (int i = 0; i < 256; i++) {
    blue_pixels_count[i] = ((double)blue_pixels_count[i] / max) * histogram_png.rows;
}
for (int i = 0; i < 256; i++) {
    green_pixels_count[i] = ((double)green_pixels_count[i] / max) * histogram_png.rows;
}
for (int i = 0; i < 256; i++) {
    red_pixels_count[i] = ((double)red_pixels_count[i] / max) * histogram_png.rows;
}

for (int i = 0; i < 255; i++)
{
    cv::line(histogram_png, cv::Point(bin_w * (i), histogram_png.rows - blue_pixels_count[i]),
        cv::Point(bin_w * (i + 1), histogram_png.rows - blue_pixels_count[i + 1]),
        cv::Scalar(255, 0, 0), 1, 8, 0);
}
for (int i = 0; i < 255; i++)
{
    cv::line(histogram_png, cv::Point(bin_w * (i), histogram_png.rows - green_pixels_count[i]),
        cv::Point(bin_w * (i + 1), histogram_png.rows - green_pixels_count[i + 1]),
        cv::Scalar(0, 255, 0), 1, 8, 0);
}
for (int i = 0; i < 255; i++)
{
    cv::line(histogram_png, cv::Point(bin_w * (i), histogram_png.rows - red_pixels_count[i]),
        cv::Point(bin_w * (i + 1), histogram_png.rows - red_pixels_count[i + 1]),
        cv::Scalar(0, 0, 255), 1, 8, 0);
}

cv::Mat histogram_jpeg(256, 256, CV_8UC3, cv::Scalar(240, 240, 240));
getBrightness(channels_split[0], blue_pixels_count);
getBrightness(channels_split[1], green_pixels_count);
getBrightness(channels_split[2], red_pixels_count);

max = 0;
for (int i = 0; i < 256; i++) {
    if (max < blue_pixels_count[i]) {
        max = blue_pixels_count[i];
    }
}
for (int i = 0; i < 256; i++) {
    if (max < green_pixels_count[i]) {
        max = green_pixels_count[i];
    }
}
for (int i = 0; i < 256; i++) {
    if (max < red_pixels_count[i]) {
        max = red_pixels_count[i];
    }
}
```

```
    for (int i = 0; i < 256; i++) {
        blue_pixels_count[i] = ((double)blue_pixels_count[i] / max) * histogram_jpeg.rows;
    }
    for (int i = 0; i < 256; i++) {
        green_pixels_count[i] = ((double)green_pixels_count[i] / max) * histogram_jpeg.rows;
    }
    for (int i = 0; i < 256; i++) {
        red_pixels_count[i] = ((double)red_pixels_count[i] / max) * histogram_jpeg.rows;
    }

    for (int i = 0; i < 255; i++)
    {
        cv::line(histogram_jpeg, cv::Point(bin_w * (i), histogram_jpeg.rows - blue_pixels_count[i])
            cv::Point(bin_w * (i + 1), histogram_jpeg.rows - blue_pixels_count[i + 1]),
            cv::Scalar(255, 0, 0), 1, 8, 0);
    }
    for (int i = 0; i < 255; i++)
    {
        cv::line(histogram_jpeg, cv::Point(bin_w * (i), histogram_jpeg.rows - green_pixels_count[i]
            cv::Point(bin_w * (i + 1), histogram_jpeg.rows - green_pixels_count[i + 1]),
            cv::Scalar(0, 255, 0), 1, 8, 0);
    }
    for (int i = 0; i < 255; i++)
    {
        cv::line(histogram_jpeg, cv::Point(bin_w * (i), histogram_jpeg.rows - red_pixels_count[i]),
            cv::Point(bin_w * (i + 1), histogram_jpeg.rows - red_pixels_count[i + 1]),
            cv::Scalar(0, 0, 255), 1, 8, 0);
    }

    cv::hconcat(image_png, image_jpg, mosaic_top);
    cv::hconcat(histogram_png, histogram_jpeg, mosaic_bottom);
    cv::vconcat(mosaic_top, mosaic_bottom, mosaic);
    cv::imwrite("cross_0256x0256_hists.png", mosaic);

        cv::waitKey(0);
}
```