Работа 2. Исследование каналов и JPEG-сжатия

автор: Кузнецов Д.А. дата: 2022-03-06T22:53:36

url: https://github.com/4bureck/imageProcessing_6sem/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%

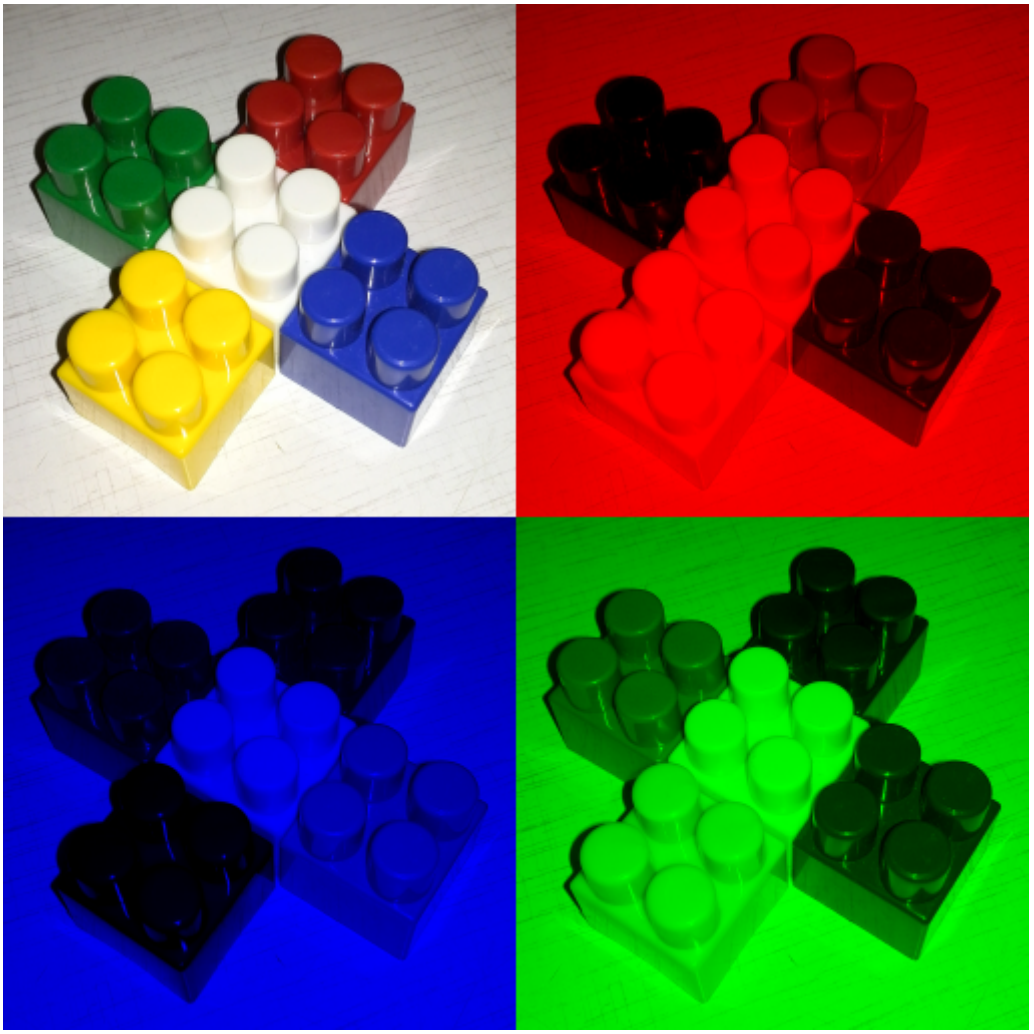

Рис. 2. Визуализация каналов исходного тестового изображения

Рис. 3. Визуализация каналов JPEG-версии тестового изображения

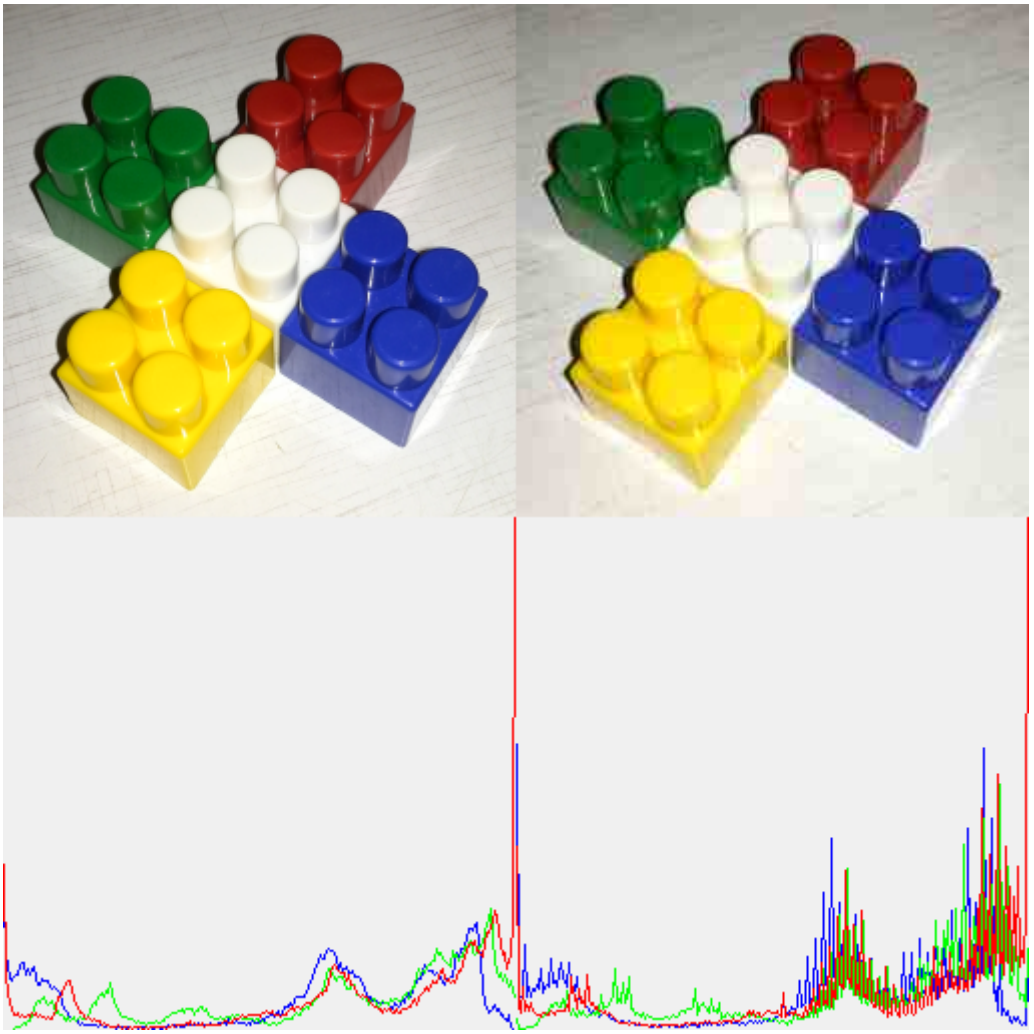

Рис. 4. Визуализация гистограм исходного и JPEG-версии тестового изображения

Текст программы

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

cv::Mat channelVizualization(cv::Mat image) {
        cv::Mat singleChannelArray[3];

        cv::split(image, singleChannelArray);

        cv::Mat zero_channel = singleChannelArray[0].clone();
        zero_channel = 0;

        cv::Mat matrixArray_B[3] = { singleChannelArray[0],  zero_channel, zero_channel };
        cv::Mat matrixArray_G[3] = { zero_channel, singleChannelArray[1], zero_channel };
        cv::Mat matrixArray_R[3] = { zero_channel, zero_channel, singleChannelArray[2] };

        cv::Mat blueImg, greenImg, redImg;
        cv::merge(matrixArray_B, 3, blueImg);

        cv::merge(matrixArray_G, 3, greenImg);

        cv::merge(matrixArray_R, 3, redImg);

        cv::Mat concatResoults1, concatResoults2;

        cv::vconcat(image, blueImg, concatResoults1);
        cv::vconcat(redImg, greenImg, concatResoults2);
```

```
	cv::Mat resoult;

	cv::hconcat(concatResoults1, concatResoults2, resoult);

	return resoult;
}

void getBrightness(cv::Mat grayscale_image, int* pixel_array) { //verni [256]
	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)]++;
		}
	}
}

cv::Mat createHistogram(cv::Mat image) {
	cv::Mat histogram(256, 256, CV_8UC3, cv::Scalar(240, 240, 240));
	int bluePixels[256], greenPixels[256], redPixels[256];

	cv::Mat singleChannelArray[3];
	cv::split(image, singleChannelArray);

	getBrightness(singleChannelArray[0], bluePixels);

	getBrightness(singleChannelArray[1], greenPixels);
	getBrightness(singleChannelArray[2], redPixels);

	//count max height of raw
	int max = 0;
	for (int i = 0; i < 256; i++) {
		if (max < bluePixels[i])
			max = bluePixels[i];
	}

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

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



	for (int i = 0; i < 256; i++) {
		bluePixels[i] = ((double)bluePixels[i] / max) * image.rows;
	}

	for (int i = 0; i < 256; i++) {
		greenPixels[i] = ((double)greenPixels[i] / max) * image.rows;
	}

	for (int i = 0; i < 256; i++) {
		redPixels[i] = ((double)redPixels[i] / max) * image.rows;
	}
```

```
        for (int i = 0; i < 255; i++){
                cv::line(histogram, cv::Point(i, histogram.rows - bluePixels[i]),
                        cv::Point(i + 1, histogram.rows - bluePixels[i + 1]),
                        cv::Scalar(255, 0, 0), 1, 8, 0);
        }

        for (int i = 0; i < 255; i++) {
                cv::line(histogram, cv::Point(i, histogram.rows - greenPixels[i]),
                        cv::Point(i + 1, histogram.rows - greenPixels[i + 1]),
                        cv::Scalar(0, 255, 0), 1, 8, 0);
        }

        for (int i = 0; i < 255; i++) {
                cv::line(histogram, cv::Point(i, histogram.rows - redPixels[i]),
                        cv::Point(i + 1, histogram.rows - redPixels[i + 1]),
                        cv::Scalar(0, 0, 255), 1, 8, 0);
        }


        return histogram;
}



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

        //vector of params for imwrite. The second one variant {cv::IMWRITE_JPEG_QUALITY, 25}
        std::vector<int> params = {};
        params.push_back(cv::IMWRITE_JPEG_QUALITY);
        params.push_back(25);

        cv::imwrite("Cross.jpeg", image_png, params);

        cv::Mat image_jpeg = cv::imread("D:/Sandbox/lab1_2/build.vs.2022/prj.labs/lab02/cross.jpeg"


        cv::Mat mosaica_png;
        mosaica_png = channelVizualization(image_png);
        cv::imshow("Channels.png", mosaica_png);
        cv::imwrite("Channels.png", mosaica_png);


        cv::Mat mosaica_jpeg;
        mosaica_jpeg = channelVizualization(image_jpeg);
        cv::imshow("Channels_jpeg.png", mosaica_jpeg);
        cv::imwrite("Channels_jpeg.png", mosaica_jpeg);


        cv::Mat histogram_png, histogram_jpeg;


        histogram_png = createHistogram(image_png);

        histogram_jpeg = createHistogram(image_jpeg);

        cv::Mat concatResoults1, concatResoults2;

        cv::vconcat(image_png, histogram_png, concatResoults1);
```

```
        cv::vconcat(image_jpeg, histogram_jpeg, concatResoults2);

        cv::Mat resoult;

        cv::hconcat(concatResoults1, concatResoults2, resoult);

        cv::imshow("Histograma", resoult);
        cv::imwrite("Histograma.png", resoult);

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