

# Отчёт по аугментатору

Ссылка на репозиторий: [https://github.com/valkon29/mipt2024f\\_konovalov\\_v\\_r](https://github.com/valkon29/mipt2024f_konovalov_v_r)

Автор: Коновалов Валентин

## Input/Output

Вход:

- набор изображений
- файл с разметкой, полученной с помощью [VGG Image Annotator](#). Для каждой картинки она представляет из себя набор точек, которые обозначают полигон вокруг штрих-кода. Помимо этого в атрибутах присутствует тип кода и метка валидности (за подробностями см. пример в репозитории).

Выход:

- набор изображений, полученных из входных путём некоторых искажений
- разметка в том же формате, что и на входе

## API

Исполняемый файл *main.py*, ему на вход подаются путь к папке с изображениями, путь к файлу и число, задающее кол-во изображений, которое необходимо получить из каждого исходного посредством аугментации:

```
python3 main.py images via_project_9Nov2024_20h28m_.json 2
```

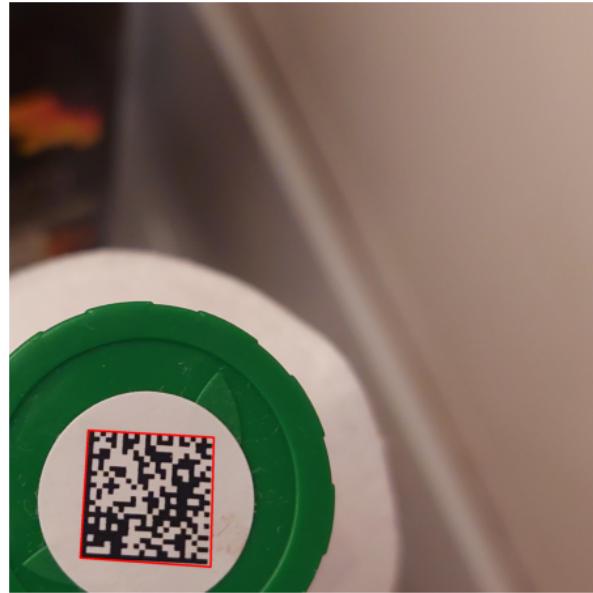
Полученные изображения будут сохранены в папку *augmented images*, а разметка в файл *augmented\_markup.json*. Формат выходной разметки совпадет со входным.

Сама операция искажения, применяемая к входным изображениям, захардкожена внутри файла *main.py*. В будущем планируется в какой-то степени перенести список искажений и их параметров в какой-нибудь конфиг.

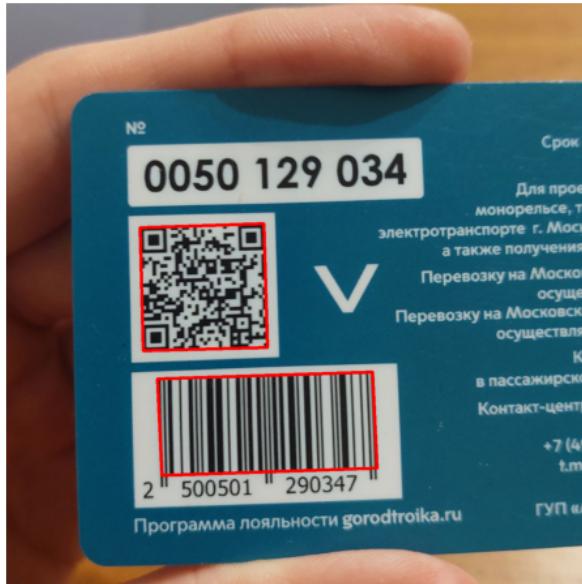
## Искажения

Все искажения изображений осуществляются посредством библиотеки [albumentations](#). Она поддерживает работу с ключевыми точками, масками и bounding\_box-ами, поэтому пока что её функционала вполне достаточно. Вот пример нескольких искажений, с визуализацией разметки, которые кажутся наиболее актуальными, и которые были опробованы на практике.

- **RandomCrop**: кадрирование изображение случайным образом



- **Rotate:** поворот на угол от 0 до 360 градусов;



- **Perspective:** углы изображения смещаются на случайные расстояния в заданных пределах, изображение деформируется соответствующим образом



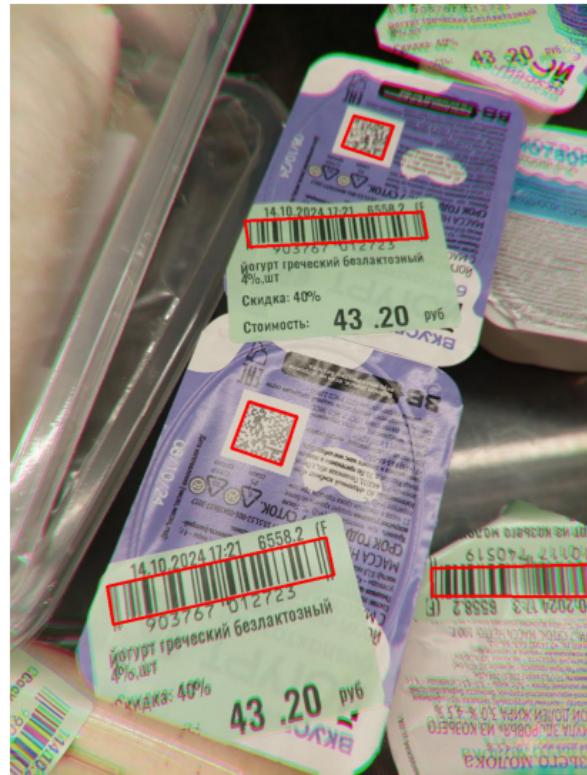
- **ISONoise:** симуляция шума, возникающего при высоких значениях ISO



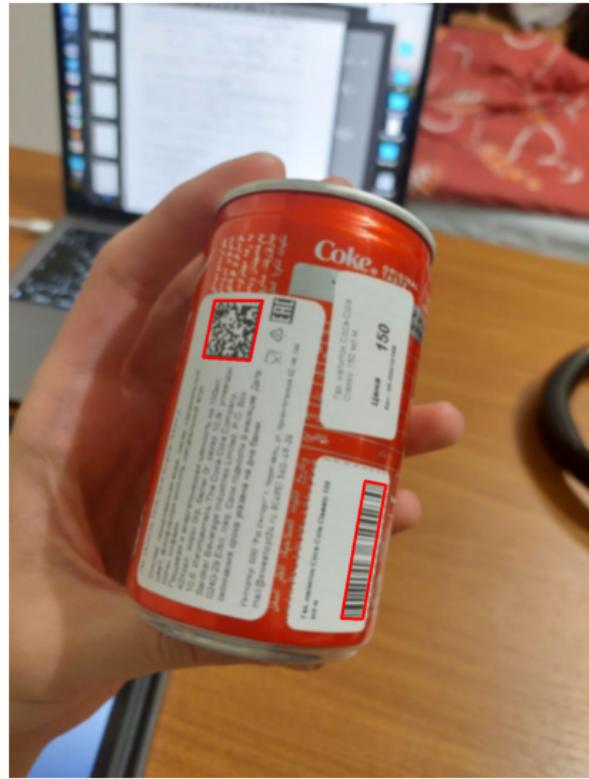
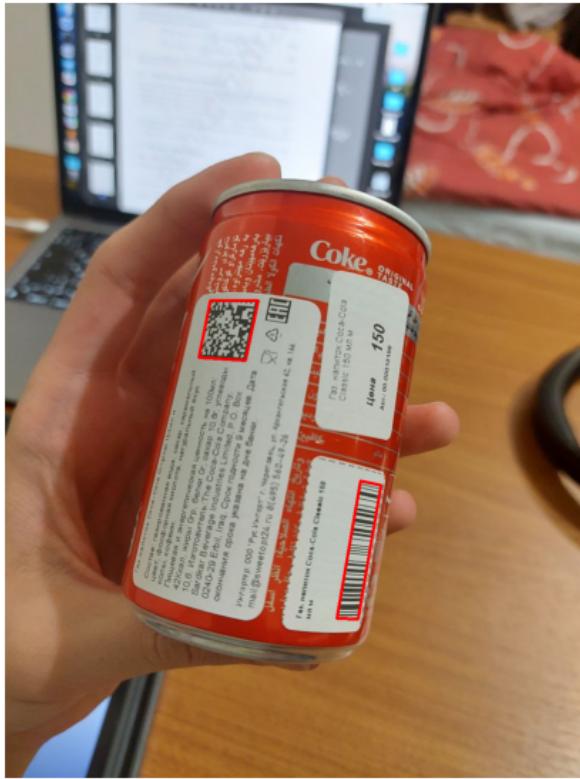
- **GaussNoise:** шум из нормального распределения, применяется отдельно для каждого пикселя и для каждого канала (хотя это можно настроить)



- **ChromaticAberration:** симуляция хроматических аберраций изображения



- **GaussianBlur:** свёртка с гауссовским ядром



- **MotionBlur:** свёртка с продольным ядром, направление "движения" при этом выбирается случайно



Кроме того, была создана [таблица](#) со всеми искажениями, которые потенциально нам могут пригодиться. В столбцах присутствуют оценка релевантности отдельных искажений для нашей задачи по 10-балльной шкале, а также местами их краткое описание и допустимые границы параметров. По многим из них пока есть сомнения, таблица заполнена не полностью.

Столбец *limits depend on image* для отдельных искажений даёт информацию о том, зависят ли допустимые границы искажения от конкретного фото. К примеру, из вышеупомянутых, границы Perspective или Rotate от изображения особо не зависят, в то время как у GaussianBlur и MotionBlur эта зависимость очень явная, чем качественнее было исходное фото, тем сильнее его потом можно размывать. И что делать с такими искажениями, на данный момент, скорее неясно.

## Некоторые детали

- На данный момент, если при искажении, хоть одна точка разметки кода выходит за границы полученного изображения, код считается не валидным и в разметке это отражается. В дальнейшем можно сделать умнее
- Можно задавать композицию из произвольного числа искажений, при этом задавая вероятности каждого из них. По умолчанию они применяются с вероятностью 0.5
- При всех искажениях, которые нарушают "прямоугольность" картинки, всё лишнее обрезалось. Из вышеупомянутых к таковым относятся Rotate и Perspective. Потенциально образовавшиеся области можно, например, заполнять зеркально отражённым содержанием исходной картинки

## ЗАВИСИМОСТИ

Исключительно питоновские библиотеки, которые можно установить, например, с помощью утилиты PyPI:

- OpenCV
- albumentations