

# Отчёт по аугментатору

---

Ссылка на репозиторий: [https://github.com/valkon29/mipt2024f\\_konovlov\\_v\\_r](https://github.com/valkon29/mipt2024f_konovlov_v_r)

Автор: Коновалов Валентин

## Input/Output

Вход:

- набор изображений
- файл с разметкой, полученной с помощью [VGG Image Annotator](#). Для каждой картинке 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.

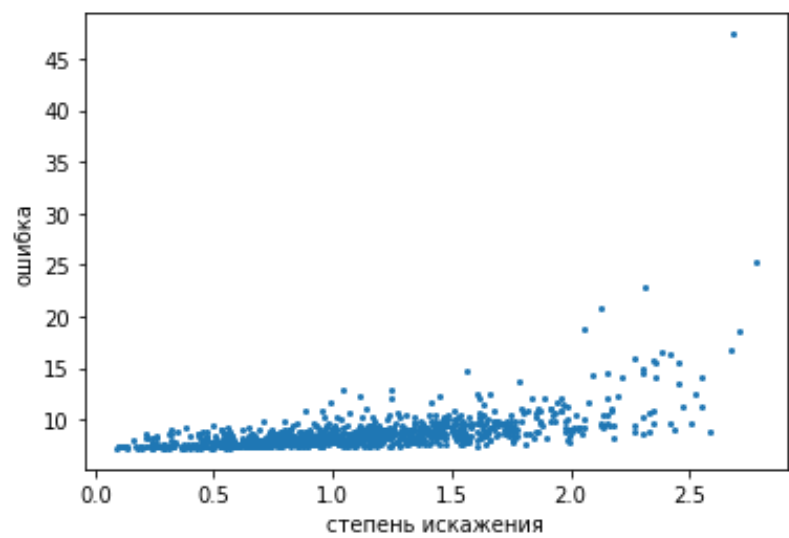
## Проективное искажение

Для исследования этих искажений на данном этапе были взяты синтетические изображения. Они деформировались в результате проективного преобразования, а затем путём обратного преобразования приводились к исходному прямоугольному виду.

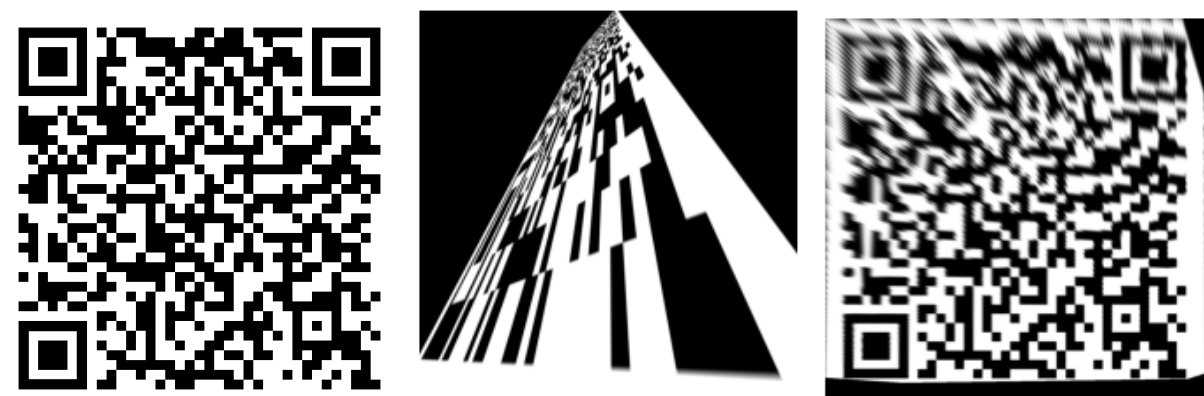
В качестве меры деформированности бралась сумма отклонений углов от 90 градусов, так как синтетические штрих-коды исходно прямоугольные. Но ниже значения углов представлены в радианах. За метрику близости между исходным кодом и изображением, полученным в результате обратного преобразования, была взята L2-норма разницы в grayscale (только по области штрих-кода).

Соответственно процедура искажения многократно запускалась на одном изображении, считалась мера искажения и близость итогового изображения к исходному.

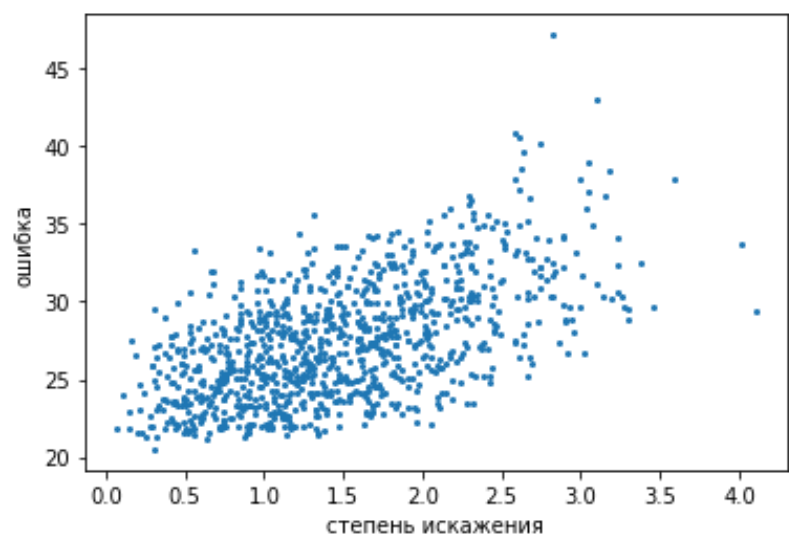
На qr-коде:



Пример сильного искажения:



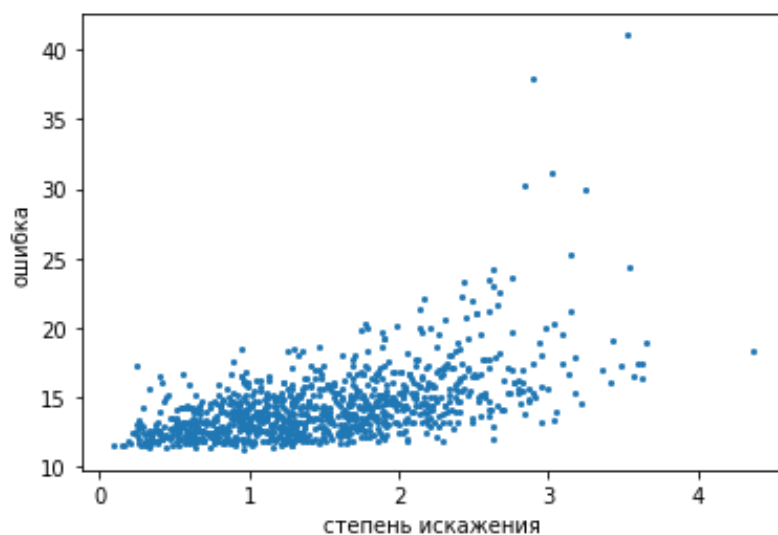
На штрих-коде:



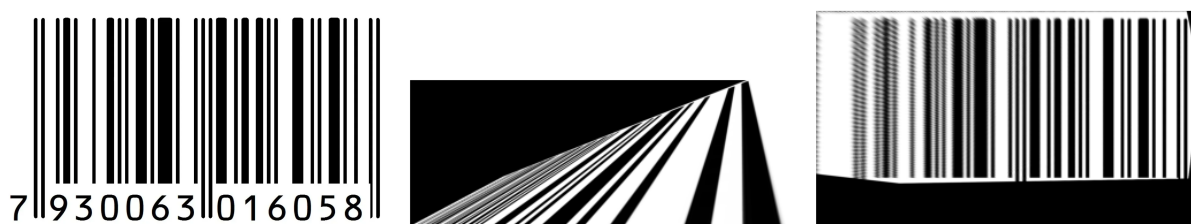
Пример сильного искажения:



На штрих-коде попроще:



Пример сильного искажения:

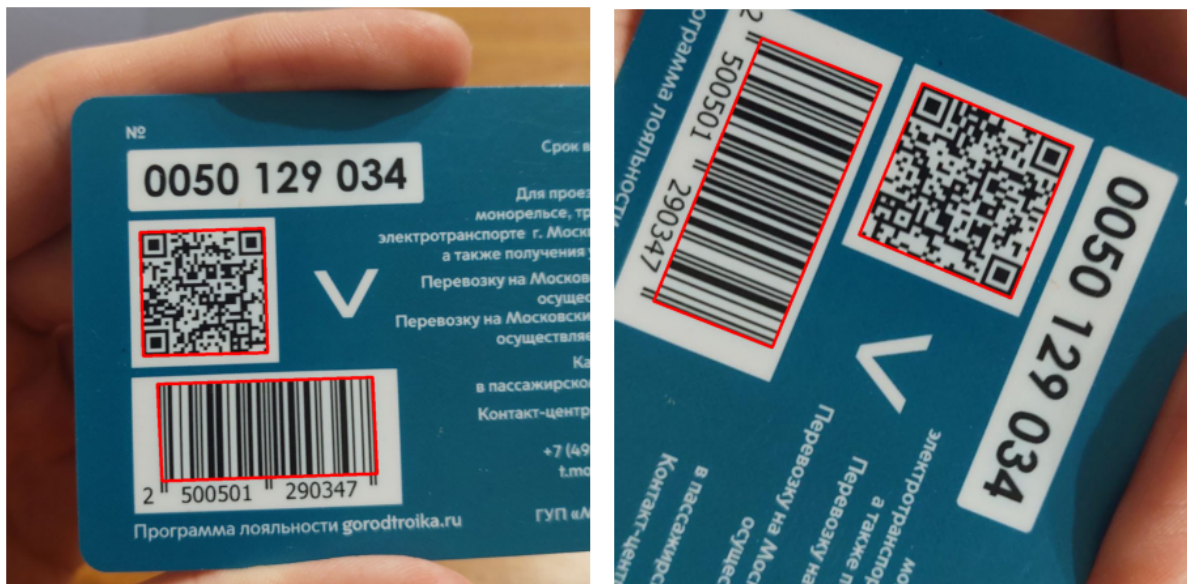


Исходя из полученных данных можно сделать вывод, что неадекватные деформации начинаются примерно с 2.0 радиан.

## Искажения из alumentations

Здесь приведены некоторые наиболее актуальные искажения посредством библиотеки [alumentations](#). Она поддерживает работу с ключевыми точками, масками и bounding\_box-ами.

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

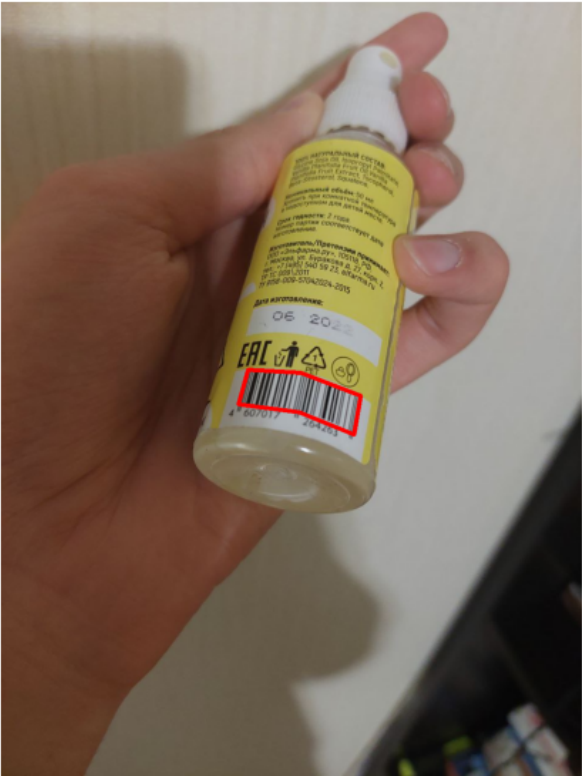


- **ISONoise**: симуляция шума, возникающего при высоких значениях ISO

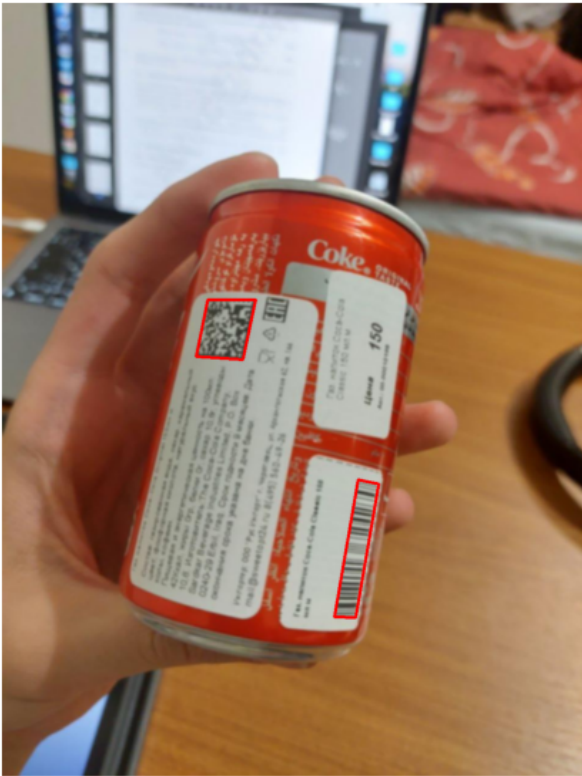
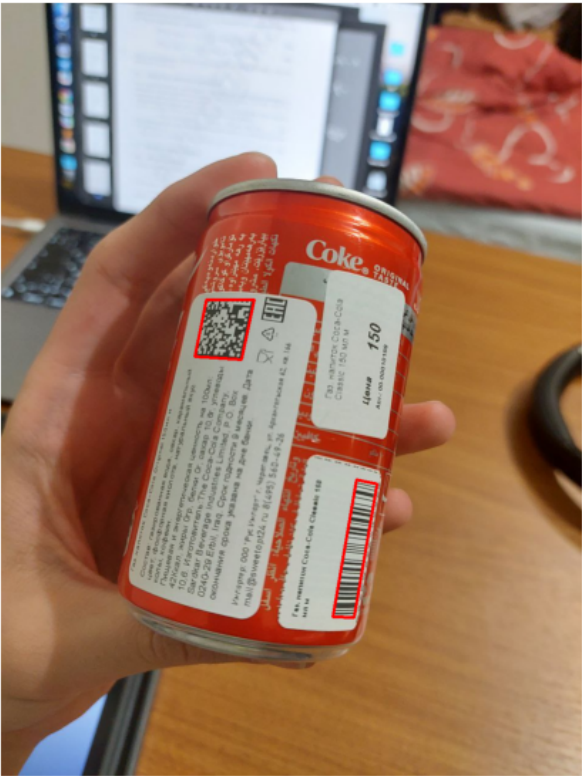


- **GaussNoise**: шум из нормального распределения, применяется отдельно для каждого пикселя и для каждого канала (хотя это можно настроить)





- **GaussianBlur**: свёртка с гауссовским ядром



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-бальной 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