

Точная локализация бар-кода

Точная локализация бар-кодов. Необходимо проработать алгоритм, который выделяет на изображении правильно ориентированную границу бар-кода, при этом на изображении он должен быть только один (то есть после грубой локализации баркодов).

P.S: Правильная ориентация определяется двумя условиями:

1. Бар-код находится слева от границы при обходе контура.
2. Первая точка границы - левая нижняя точка, если бы бар-код был бы эталонным.

API

- **Входные данные:** Карта Instance segmentation. То есть, отображение вида $R \times R \rightarrow N + \{0\}$ в виде двумерного массива, где каждый элемент обозначает принадлежность соответствующего пикселя к *i-ому* бар-коду + исходная картинка.
- **Выходные данные:** Массив, вида, который обозначает границы:

```
List[Tuple(uint, uint)]
```

```
python main.py -i /path/to/image -o /path/to/output
```

Точная локализация

Задача, на текущий момент, может решаться двумя способами:

1. Использовать результаты сегментации, найти положительно ориентированный контур выделенного бар-кода.
2. Использовать алгоритмический подход, основанный на различных фильтрах (аналогично алгоритму [Canny](#)).

Так как первый способ требует готового сегментатора, в этом отчете сосредоточимся на втором.

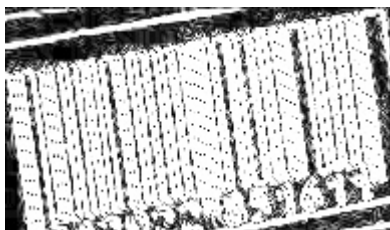
Алгоритм 1

Рассмотрим работу алгоритма на следующем примере:



Алгоритм состоит из нескольких шагов:

1. Найти на изображении "градиенты" каждого пикселя, с помощью фильтра [Собеля](#), предварительно сделав изображение черно-белым



2. Заблюрить изображение фильтром Гаусса, чтобы сгладить области с большими перепадами градиентов, а затем применить фильтр thresh, который считает все пиксели белыми, если их яркость превышает заданный порог (в алгоритме = 210).



3. Как видно, на изображении заметны "дырки", которые прямоугольным ядром, дозаполняются.



4. Затем, выделяется замкнутый контур, наибольший по площади, тем самым получается результирующая картинка.



Для метрики, в качестве очень базовой оценки, используется Intersection over Union (IoU).

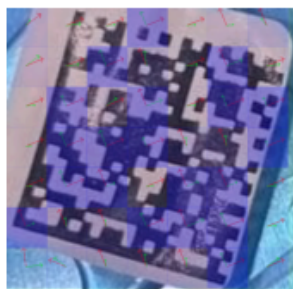
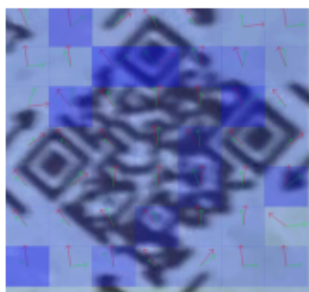
Алгоритм 2

Алгоритм основывается на HOG (Histogram of Oriented Gradients). Он работает за счет анализа распределения направленных градиентов (изменений интенсивности пикселей) в различных частях изображения.

Пока алгоритм работает только для QR-ов. Модификация для бар кодов должна быть похожей.

1. Предобработка, входное изображение преобразуется в градации серого.

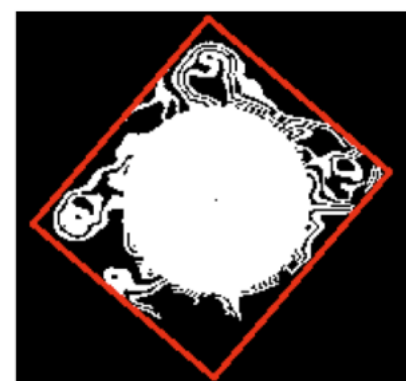
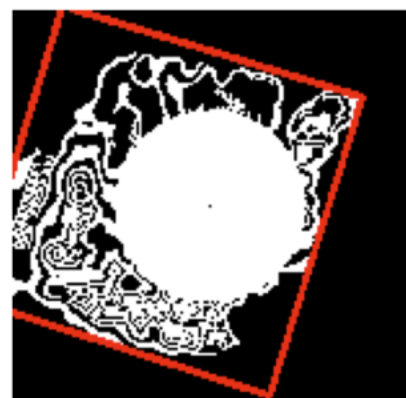
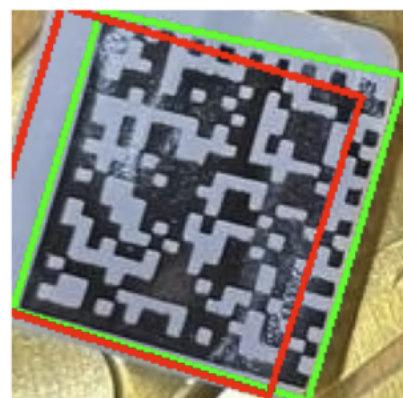
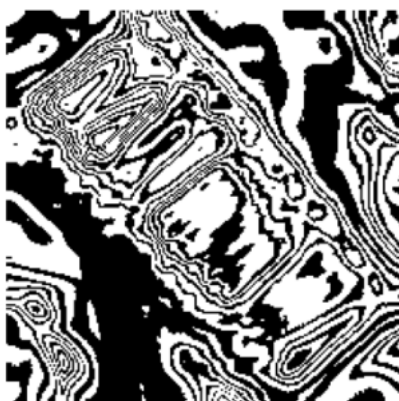
2. Построение гистограммы направленных градиентов, изображение делится на небольшие ячейки (tiles). Для каждой ячейки создается гистограмма направлений градиентов. Каждое направление (например, 0°, 45°, 90° и т.д.) получает значение, пропорциональное величине градиента в соответствующем направлении.
3. В каждом тайле выделяются два доминантных направления (то есть те два направления, которые в гистограмме имеют наибольшие значения). Затем считается вероятность того, что данный тайл принадлежит qr-коду по формуле:
$$p = \left(1 - \frac{|\theta_1 - \theta_2|}{\pi} \right) \cdot \frac{2 \cdot \min(\theta_1, \theta_2)}{\theta_1 + \theta_2}$$
 где первое слагаемое отвечает за близость разницы углов к 90 градусам, а второе отвечает за то, что в гистограмме значения близки.
4. Выбираются тайлы с значениями вероятности превышающими порог, а затем тайлы, вероятность которых чуть меньше, также добавляются, если их соседи включены
5. Строим выпуклую оболочку получившегося множества, получаем границу.

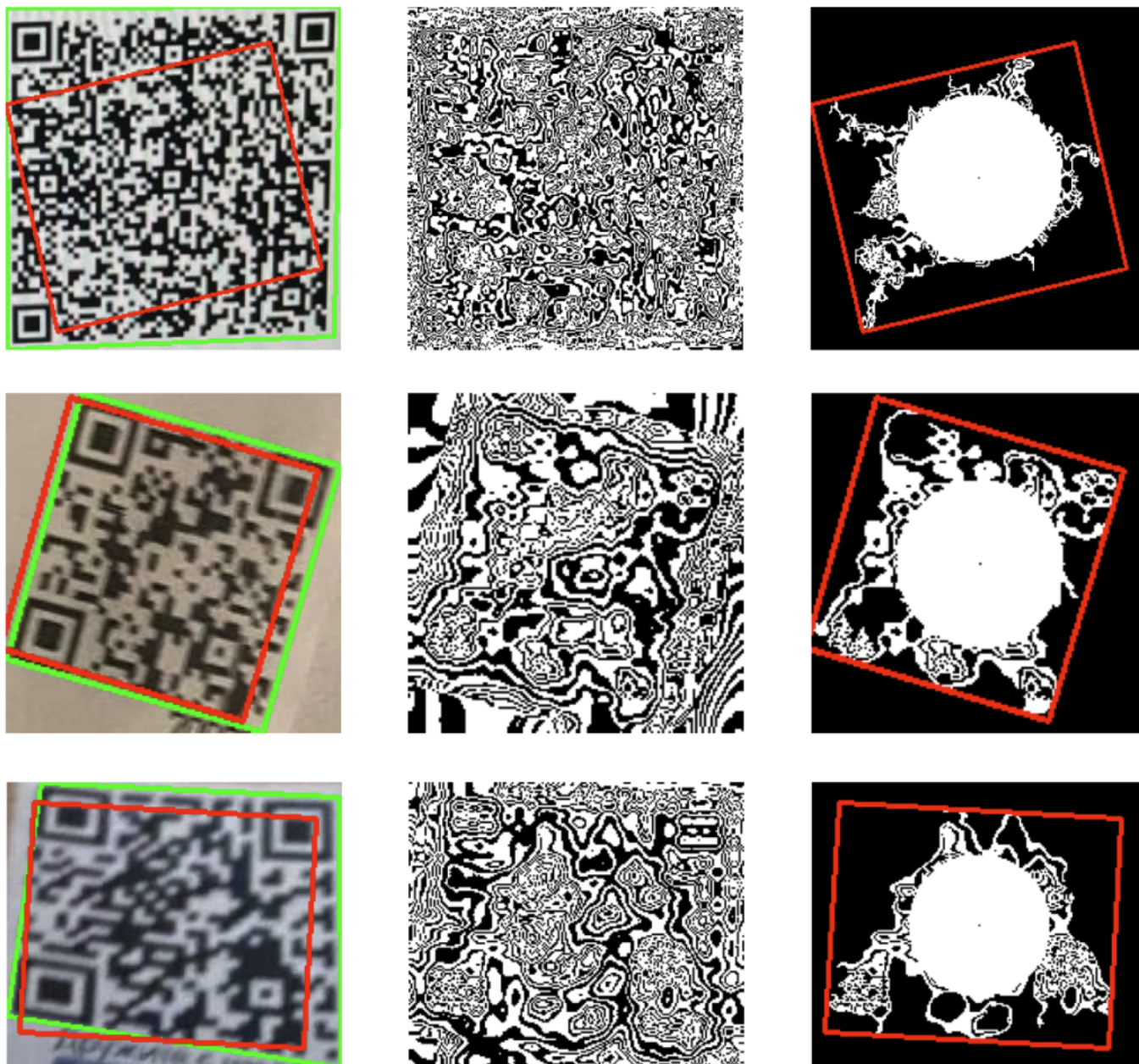


Алгоритм 3

1. Для каждого пикселя картинки, считаем дисперсию в окне некоторого размера, причем данный пиксель является его центром.
2. Выделяем, с помощью некоторого те пикксели, дисперсия окна которых больше некоторого порога. Это будут пиксели, которые с большой вероятностью принадлежат бар-коду.
3. В центре картинки выделяем область в центре, в виде круга, которую мы будем считать принадлежащей бар-коду.
4. Из полученного круга, с помощью bfs, ищем все пиксели, с достаточно высокой дисперсией, которые лежат в одной области связности с изначальным кругом.
5. Обрамляем выделенную область в четырехугольник.

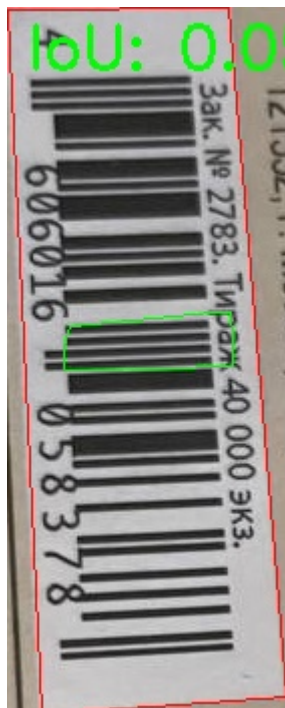
Таким алгоритмом получается добиться в среднем $\text{IoU} = 0.659$.



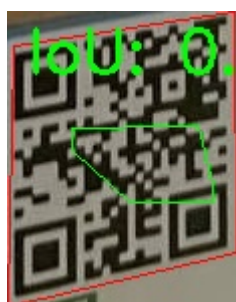


Текущие проблемы

1. Не очень понятно, как корректно выбирать пороги и параметры всех использующих фильтров. Например, на плохо контрастирующей картинке, по-хорошему нужно подбирать порог, исходя из значения максимального градиента. Или, при процессе "дозаполнения всех дырок" в 3 пункте алгоритма, алгоритм плохо показывает себя на вертикально повернутых штрихкодах

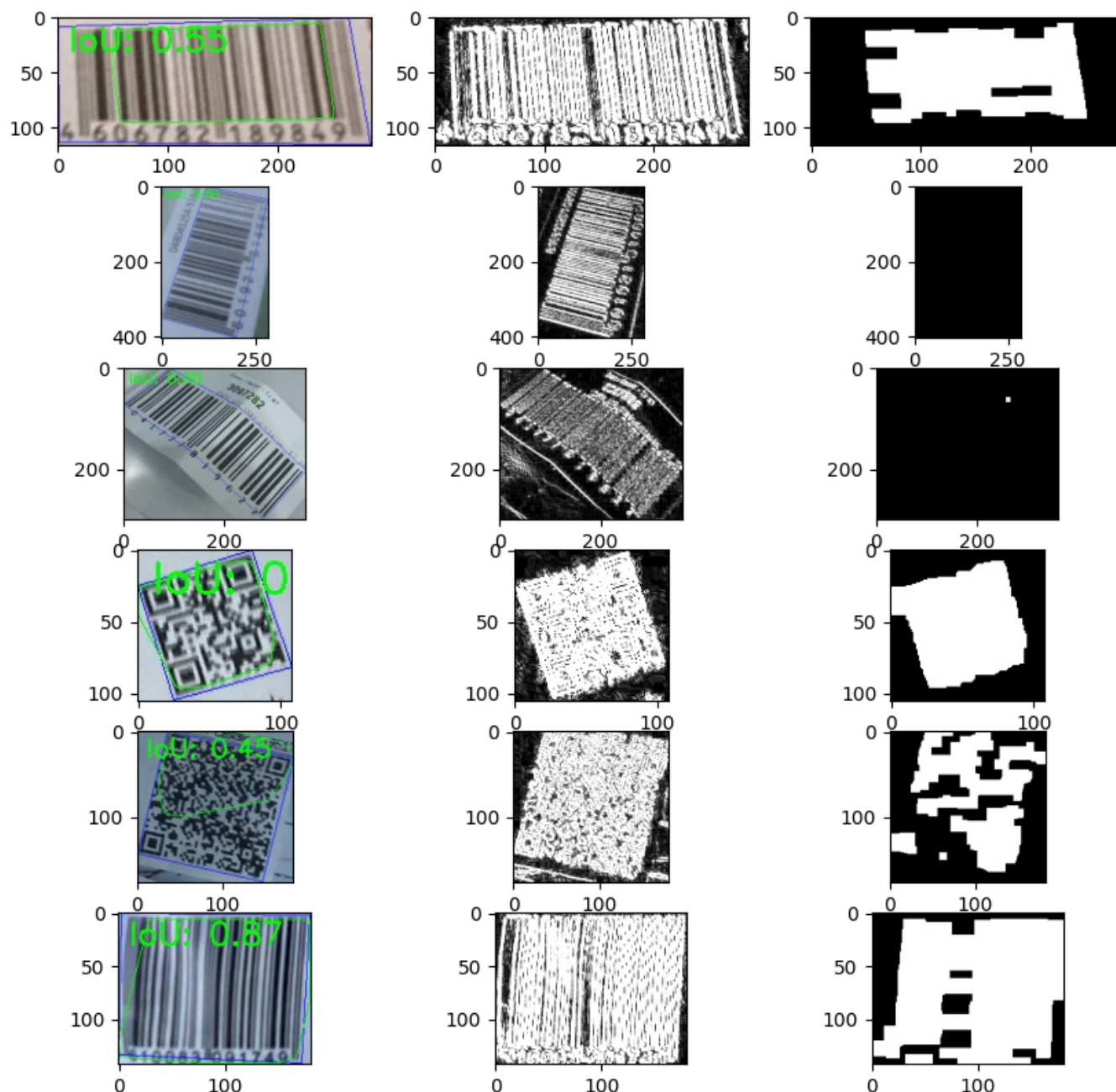


2. Плохо работает на QR-кодах, нужно думать либо о модификации алгоритма, либо придумывать что-то другое.



Текущие результаты

Результаты по всему обучающему датасету можно посмотреть, запустив `test.py`. Тут представлена лишь маленькая часть выборки:



Инструкция по запуску

Установка зависимостей

Установить все необходимые пакеты можно следующей командой:

```
pip install -r requirements.txt
```

Запуск скрипта

Скрипт возвращает картинку, с выделенной границей

```
python main.py -i /path/to/image -o /path/to/output
```

- -i, --image: Required. The path to the input image.
- -o, --output: Optional. The path to save the output image. Defaults to 'res.png' if not provided.

Запуск тестов

Скрипт, который по тестовому датасету выделяет границы на всех изображениях, при этом также возвращает промежуточные результаты работы фильтров.

```
python test.py -c annotations.json -d training_data -o output
```

- -c: Path to VGG annotator config
- -d: Path to corresponding training dataset
- -o: Path to output the images with contours of bar-codes

Состояние по разметке

На текущий момент размечено **119** картинок, содержащие **ean-13**, **qr**, **datamatrix** и некоторые более редкие виды одномерных и двумерных баркодов.