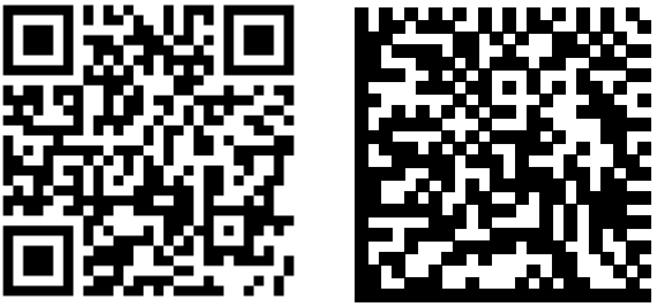
Система распознавания двумерных штрихкодов

Оглавление

1. Постановка задачи

Данная система предназначается для распознавания двух наиболее популярных типов двумерных кодов — QR-кодов и Data Matrix. Под распознаванием подразумевается определение точного положения кода и его данных по изображению.



QR-код (слева) и Data Matrix (справа)

Входные данные системы: изображение, содержащее двумерные коды.
Выходные данные системы: строки декодированных данных кодов и списки вершин ограничивающих их полигонов.

Задачу можно разделить на несколько этапов:

1. **Грубая локализация**
2. **Точная локализация**
3. **Нормализация**
4. **Декодирование**

Здесь рассматривается этап декодирования, а также соединение всех подзадач в единую систему. Задача декодирования состоит в определении данных кода по его нормализованному изображению.

Входные данные декодера: изображение одного нормализованного (без искажений) двумерного кода.
Выходные данные декодера: строка декодированных данных кода.

2. Реализация

2.1. Структура общей системы

Реализованная система представляет собой модуль python qr_dm_decoder с возможностью выбора локализатора. Кроме того реализован скрипт, используемый для отладки и подсчета статистики, позволяющий запускать систему на больших наборах данных с выбором декодера, локализатора и аугментации.

Набор входных данных описывается в специальном файле формата json (см. описание markup.json), содержащем названия изображений и соответствующую им истинную разметку (пример файла разметки здесь). Аналогично описываются результаты декодирования (см. описание result.json), содержащие разметки грубой и точной локализации, декодированную информацию, а также параметры аугментации.

Поддерживаются следующие виды аугментации:

- Поворот на случайный угол от 0 до 360,
- Перестановка цветовых каналов в случайном порядке,
- Обрезка изображения вокруг штрихкода,
- Поворот цвета на случайный угол.

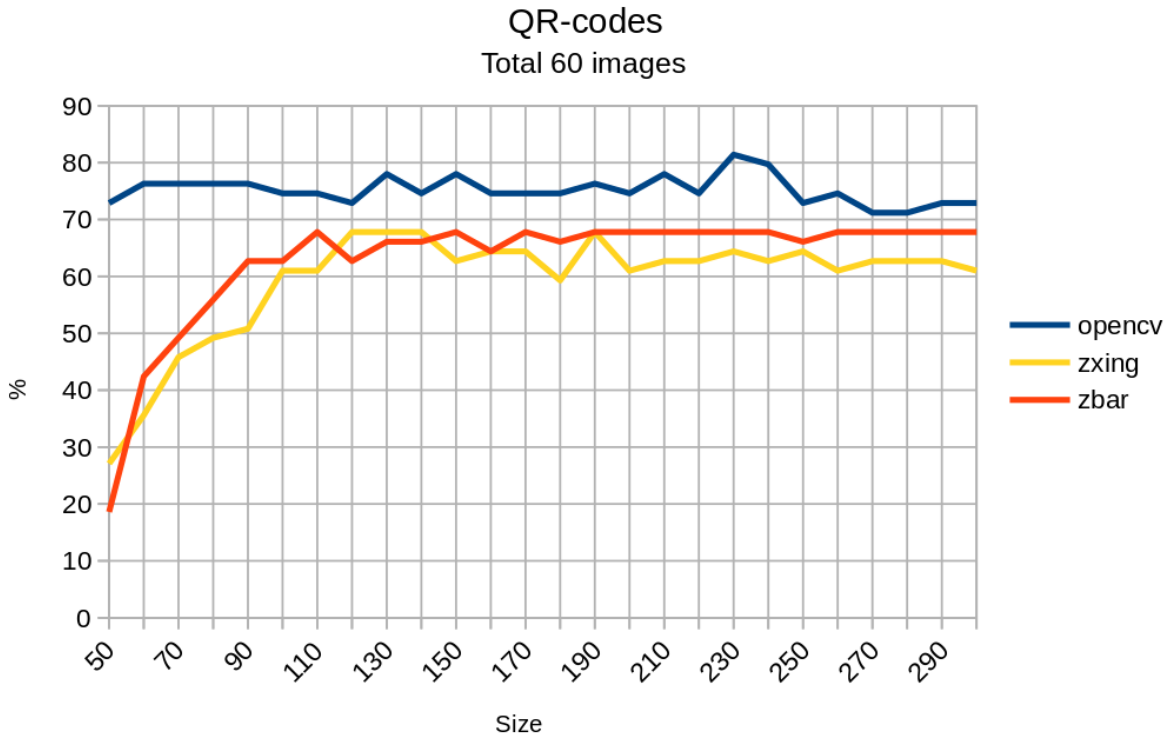
Функции аугментации здесь. Подробнее про использование системы см. ниже.

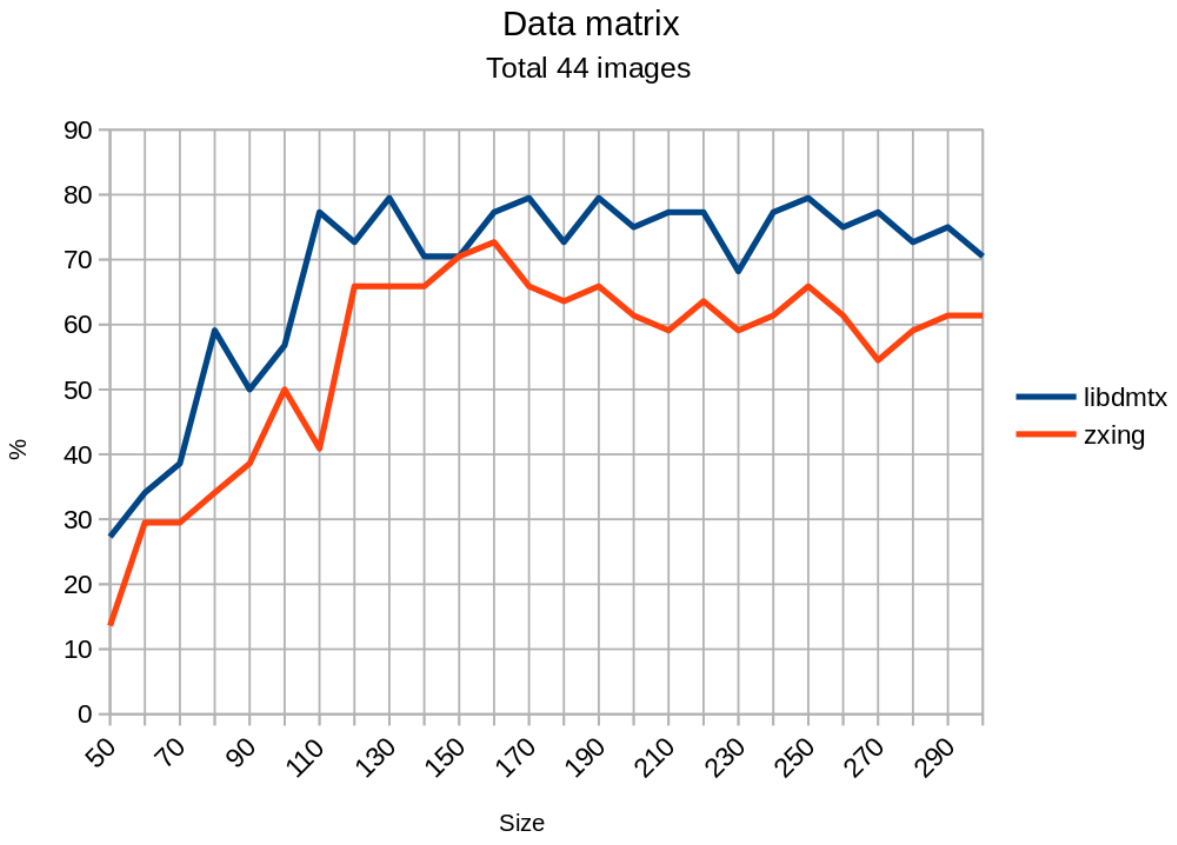
2.2. Выбор декодера

Существует множество готовых декодеров. Их основной недостаток — неустойчивость к искажениям кода, именно поэтому требуются этапы локализации и нормализации. Для данной реализации были выбраны следующие инструменты:

- OpenCV QRCodeDetector — декодер QR-кодов,
- ZBar — декодер QR-кодов и др.
- ZXing — универсальный декодер QR-кодов, Data Matrix и др.,
- Libdmtx — декодер Data Matrix.

Чтобы выбрать наиболее подходящие декодеры, были произведены оценки статистики распознаваемости для различных размеров входных изображений кодов, чтобы сразу определиться с оптимальным разрешением для декодирования (исходные изображения имеют высокое разрешение, размер сжатия указан по короткой стороне с сохранением пропорций, интерполяция кубическая):

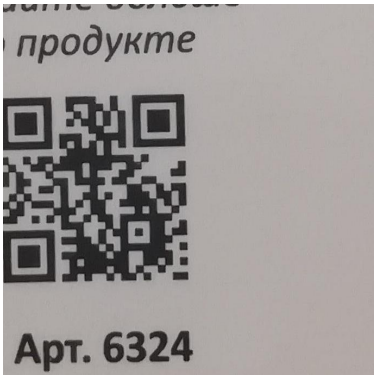

QR-codes
Total 60 images
%
Size
opencv
zxing
zbar


Data matrix
Total 44 images
%
Size
libdmtx
zxing

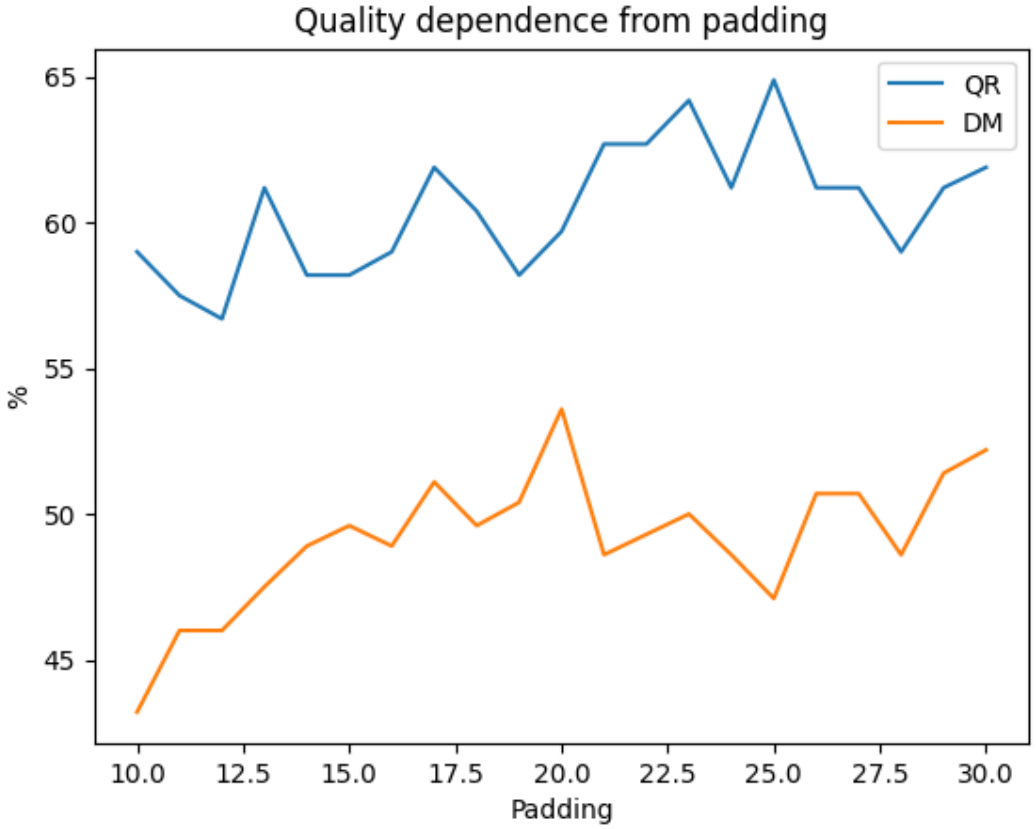
Из графиков наглядно видно, что лучший результат показывают декодеры ***opencv*** в случае QR-кодов и ***libdmtx*** в случае Data Matrix. Также можно заметить, ***opencv*** что значительно устойчивее к сжатию изображения. Таким образом, **оптимальный размер 120-130** пикселей. Для эксперимента использовался этот скрипт, поддерживающий выбор декодера, подробнее об этом ниже.

2.3. Выбор отступа

У декодера opencv была обнаружена проблема — падение с исключением в некоторых случаях, когда код обрезан или касается границ изображения. Например на таких:



Решение — добавление небольшого отступа вокруг кода. Кроме того выяснилось, что такой отступ положительно сказывается на качестве декодирования. Поэтому была произведена оценка оптимального отступа опытным путем:



Отступ указан в пикселях для изображения кода, приведенного к оптимальному размеру. То есть, если оптимальный размер, например, 130, то исходное изображение сжимается таким образом, что ширина самого кода становится равной 130 пикселям (короткая сторона) и вырезается с отступом (130+2*20=170).

3. Результаты

3.1. Примеры декодирования

3.2. Статистика

Проверка и отладка системы производится на отдельном наборе тестовых данных с помощью специального скрипта. Актуальная статистика следующая:

```
Localizer: 1
Weights: epoch=20-step=1218.ckpt
Augmentation seed: 0
================================================
 Total 248 images
------------------------------------------------
Decoder Type Decoded Total Percent Augmentation
opencv   QR    56    203   27.6%
libdmtx  DM    74    155   47.7%
opencv   QR    43    186   23.1%  rotate
libdmtx  DM    55    116   47.4%  rotate
opencv   QR    54    205   26.3%  mix_channels
libdmtx  DM    66    151   43.7%  mix_channels
opencv   QR    52    186   28.0%  crop
libdmtx  DM    67    150   44.7%  crop
opencv   QR    49    202   24.3%  rotate_color
libdmtx  DM    75    154   48.7%  rotate_color
================================================
```

```
Localizer: 2
Weights: best21042023I.pt
Augmentation seed: 0
================================================
 Total 248 images
------------------------------------------------
Decoder Type Decoded Total Percent Augmentation
opencv   QR    56    184   30.4%
libdmtx  DM    73    151   48.3%
opencv   QR    41    172   23.8%  rotate
libdmtx  DM    45    123   36.6%  rotate
opencv   QR    53    187   28.3%  mix_channels
libdmtx  DM    73    154   47.4%  mix_channels
opencv   QR    53    164   32.3%  crop
libdmtx  DM    61    140   43.6%  crop
opencv   QR    54    187   28.9%  rotate_color
libdmtx  DM    70    150   46.7%  rotate_color
================================================
```

3.3. Инструкции

Все необходимые зависимости указаны в файле requirements.txt. Веса для локализатора 2 (*best21042023l.pt*) есть в репозитории.

Запуск декодера с локализатором:
Аргументы:

- Изображение
- Локализатор: 1, 2
- Веса модели локализатора
- Дополнительно: выходное изображение

```
python3 decode.py <input image path> <localizer (1 or 2)>
<localizer checkpoint path> [output image path]
```

Пример:

```
python3 decode.py image.png 2 best21042023I.pt
```

Подсчет статистики распознаваемости:
Аргументы:

- Файл разметки
- Декодер QR: opencv, zbar, zxing
- Декодер DM: libdmtx, zxing
- Локализатор: 1, 2
- Веса модели локализатора
- Дополнительно: аугментация: rotate, mix_channels, crop, rotate_color

```
python3 decode_with_localizer.py <markup file> <QR decoder>
<DataMatrix decoder> <localizer> <localizer checkpoint>
[augmentation]
```

Пример:

```
python3 decode_with_localizer.py datast/markup.json opencv
```

```
libdmtx 2 best21042023I.pt
```

Пример использования модуля в коде:
Основная функция decode() принимает следующие аргументы:

- Изображение
- Локализатор: 1, 2
- Веса модели локализатора

```
import cv2 as cv
from qr_dm_decoder.decoder import decode

img = cv.imread('example_image.png')

decode_results = decode(img, '2', 'best21042023I.pt')
```

4. Ссылки

Репозиторий: https://github.com/egor79k/barcode-recognition
Тестовый датасет: https://github.com/ppadas/QR_codes_common
Формат разметки: https://github.com/egor79k/barcode-recognition/blob/master/docs/markup_format.pdf
OpenCV: https://opencv.org/
ZBar: https://github.com/mchehab/zbar
ZXing: https://github.com/zxing/zxing
Libdmtx: https://github.com/dmtx/libdmtx