Отчет по задачам генератор и синтезатор

https://github.com/IlyaYakovenko/mipt2024f_yakovenko_i_o/tree/main

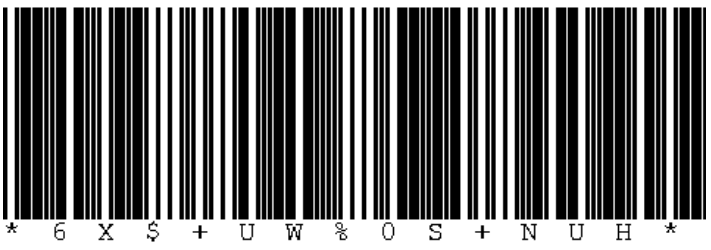
Яковенко Илья Б05-128
Редакция от 23.11.24

Передо мной стояла задача по созданию изображений баркодов и правильной разметки к ним. Моя задача была поделена на две части - Генератор и Синтезатор. Отчет также будет разделен на две части, для удобства отслеживания прогресса по обеим частям. Обе части написаны на языке Python3.

Генератор

Главная задача генератора - создание “эталонных” изображений баркодов и разметки для них. (эталонное изображение - синтетическая png картинка с изображением баркода без каких либо дефектов).

Пример эталонных изображений:



Сгенерированные изображения code39 и qr с рандомным текстом

Используется готовый генератор изображений кодов на основе “Barcode Writer in Pure PostScript”. На данный момент

мой генератор может создавать изображения и разметку следующих типов:

- code39
- ean8
- ean13
- ean128
- interleaved 2 of 5
- UPC-A
- UPC-E
- aztec code
- datamatrix
- maxicode
- pdf417
- qrcode

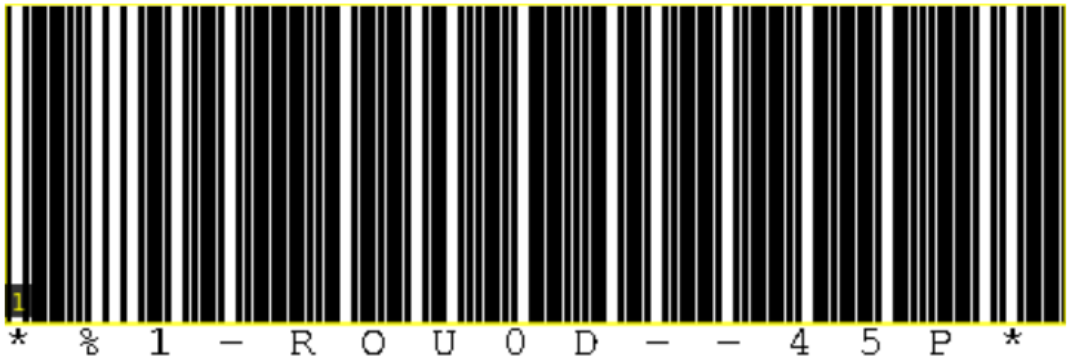
Однако расширение для остальных типов происходит достаточно недорого по времени и силам.

Также для каждого кода генерируется разметка в формате json файла, который можно просмотреть с помощью VGG Image Annotator.

Для двумерных кодов разметка генерируется по краям изображения и указывается тип кода.

	type
1	qr

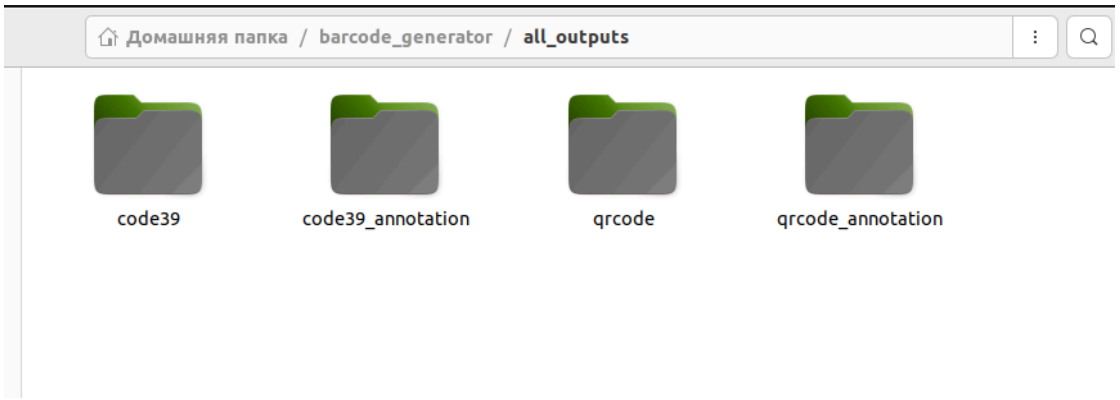
Для одномерных кодов с подписью разметка генерируется по краям значащей части, не затрагивая надпись и также указывается тип кода.



Чтобы сгенерировать такую разметку применяется алгоритм поиска конца линий, который проходит по всему изображению сверху вниз и регистрирует количество черных пикселей в строке. Как только количество меняется алгоритм запоминает пиксель на котором это произошло и отмечает его как нижнюю границу для рамки разметки.

В качестве входных данных программа принимает тип кода для генерации, данные и некоторые дополнительные параметры. Генератор может как принимать пользовательские данные для кодов так и создавать случайные данные. Для введенных данных реализована проверка валидности.

В качестве выхода программа создает отдельные директории для каждого типа кодов и его разметок.



API:

В файле generator.py реализован класс Generator предоставляющие следующие методы:

1)

```
def generate_barcode_data(self, barcode_type: str, count: int) -> list
```

Метод принимает тип кода и количество. Возвращает список длины count с валидными данными для данного типа кода.

2)

```
def generate_annotations(self, barcode_type, annotation_path, barcode_path, template_path):
```

Принимает тип кода, путь для сохранения разметки, путь до папки с изображениями и путь шаблона разметки.

Метод проходит по всем изображениям в папке с кодами и создает для них разметку в папке annotation_path

3)

```
def count_black_pixel_changes(self, image):
```

Метод принимает изображение кода и находит для него конец черных линий для задания нижней границы разметки.

4)

Также Generator содержит подкласс KnowledgeBase, который содержит словарь с типами кодов, которые программа может создать и метод

```
def validate_barcode(self, barcode_type: str,
data: str) -> bool:
```

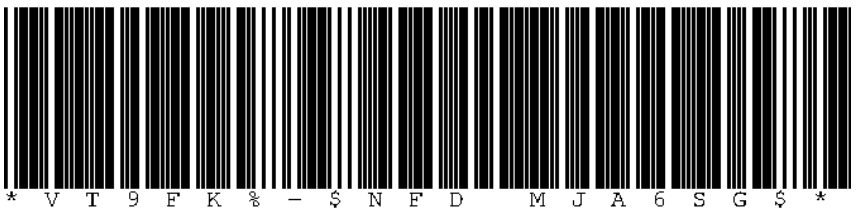
для проверки корректности пользовательских данных. Реализован с помощью сопоставления с регулярным выражением.

Синтезатор

Задача синтезатора - преобразовывать изображения, полученные от генератора, для того, чтобы они казались более реалистичными.

На данный момент реализованы следующие виды преобразований:

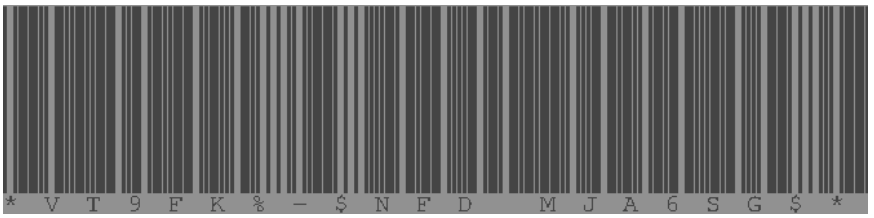
0) Оригинальное изображение



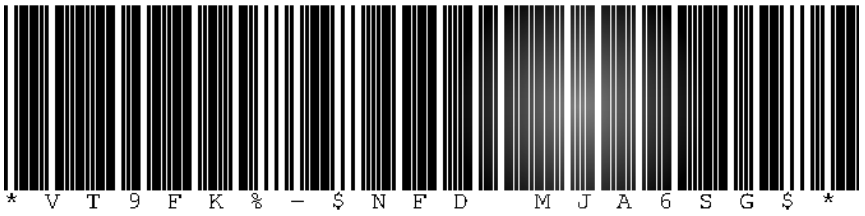
1) Размытие

2) Изменение уровня яркости

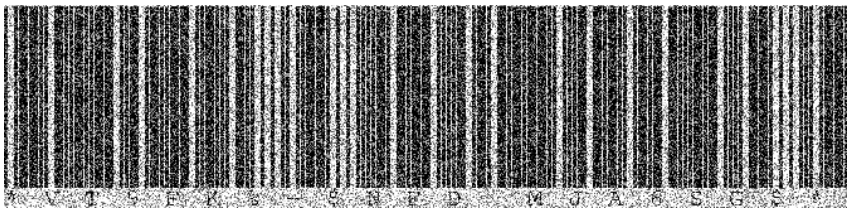
3) Изменение контрастности



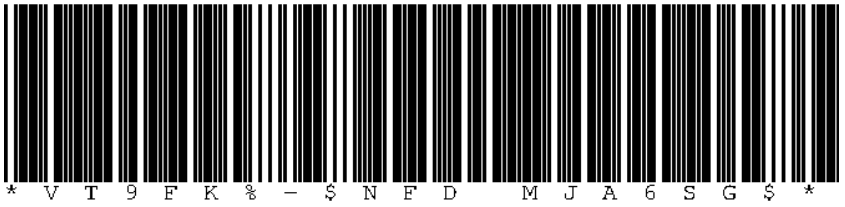
4) Добавление блика



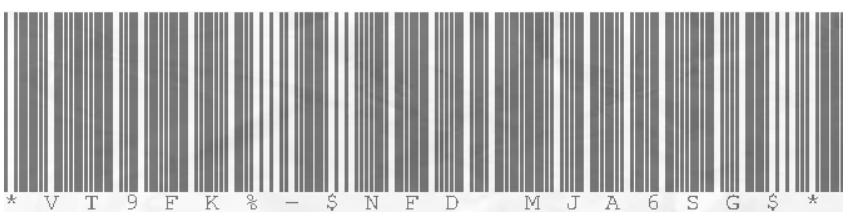
5) Зашумление



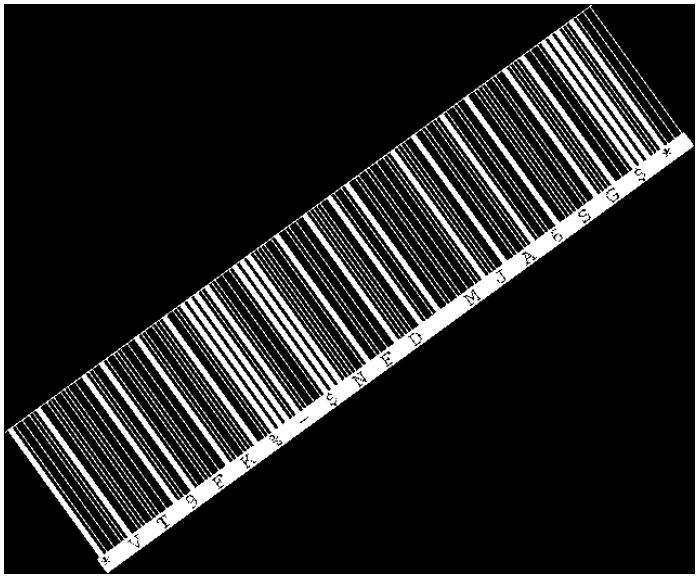
6) Изменение насыщенности



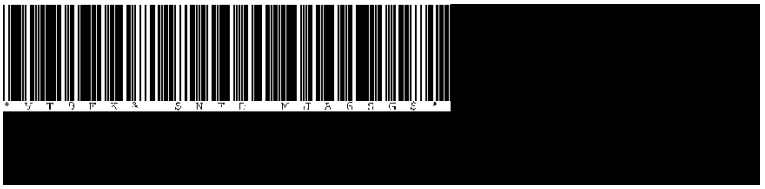
7) Добавление текстуры(в данном случае скомканная бумага)



8) Поворот

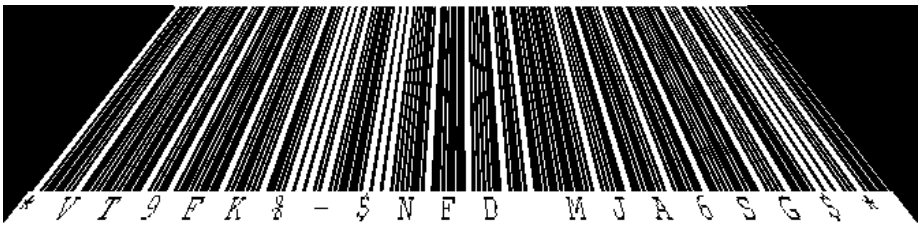


9) Изменение масштаба

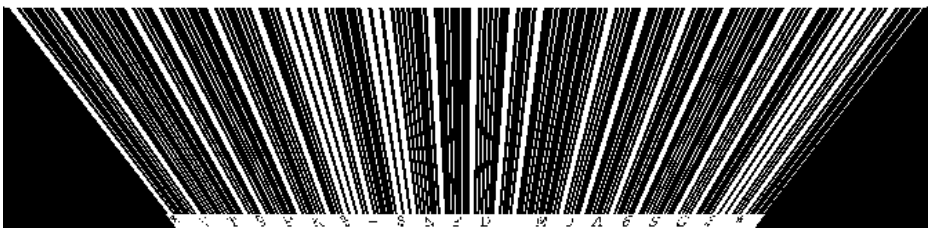


10) Перспективное искажение

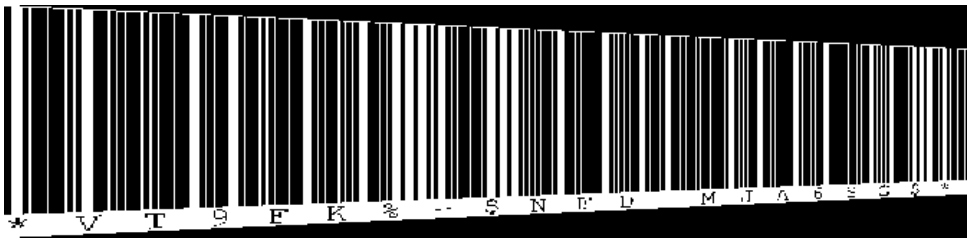
10.1) Вид снизу



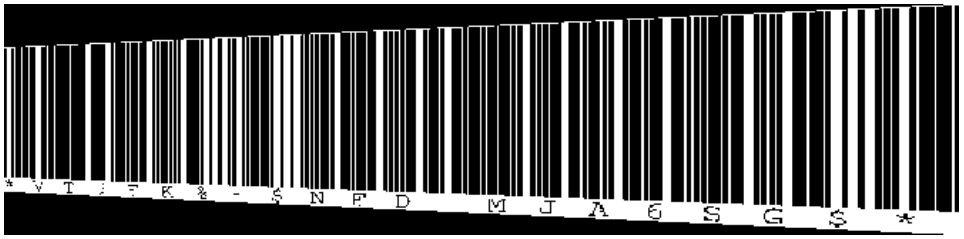
10.2) Вид сверху



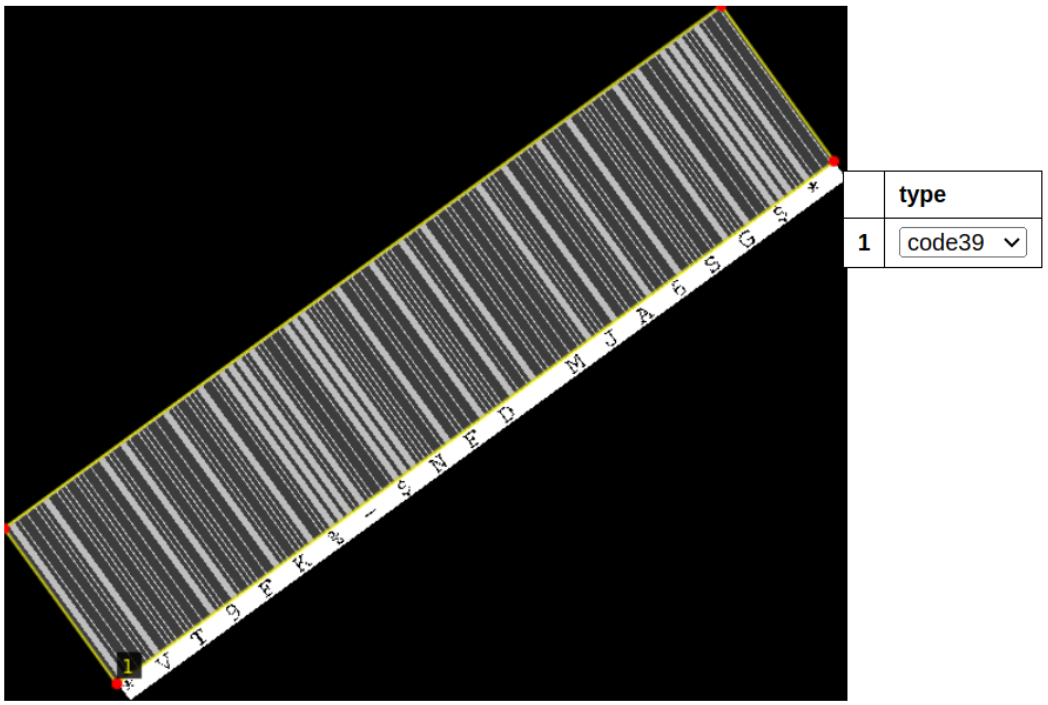
10.3) Вид слева



10.4) Вид справа

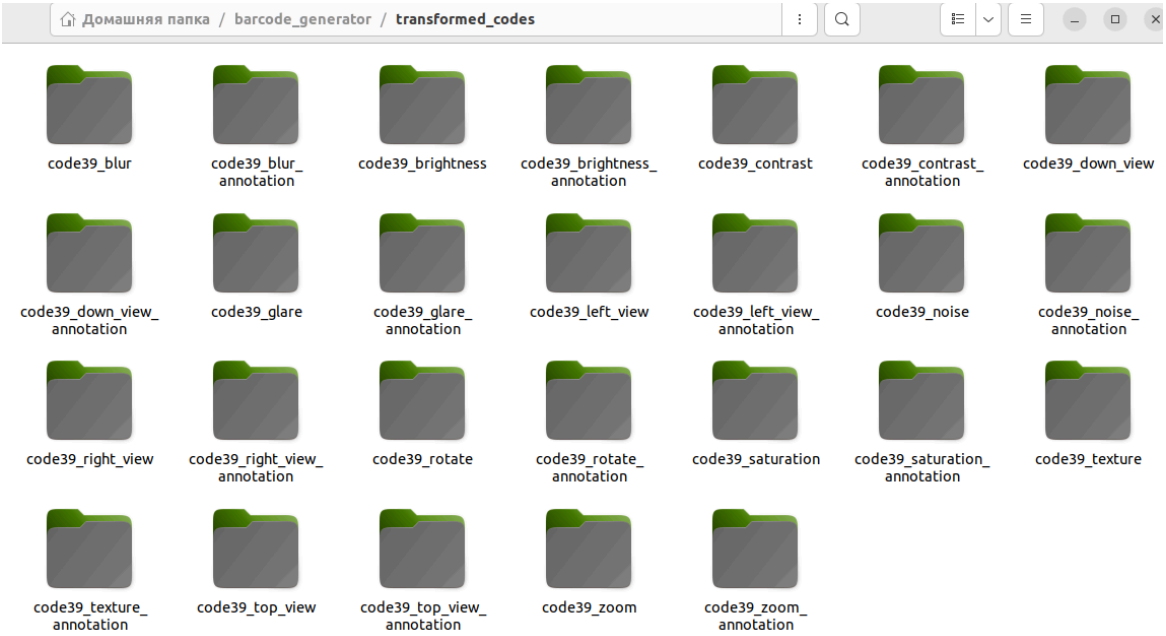


Для преобразований, меняющих положение кода или его перспективу также вслед за ним меняется и разметка. Например для поворота:



Этот эффект был получен благодаря умножению координат разметки на матрицу поворота. Для изменения масштаба или проективных искажений координаты также меняются после умножения на матрицу соответствующего преобразования.

В качестве входных данных программа принимает изображение и разметку. На выходе генерирует папку с соответствующим преобразованием и разметкой для него



API:

В файле synthesizer.py реализован класс Synthesizer предоставляющие методы, выполняющие соответствующие преобразования как изображения, так и разметки:

1)

```
def rotate(self, angle)
```

Принимает угол для поворота в градусах, поворот выполняется против часовой стрелки

2)

```
def change_perspective(self, src_points,
dst_points)
```

Метод, который выполняет все 4 вида перспективных преобразований. Принимает начальные и конечные точки углов изображения. Далее вычисляет матрицу, которая выполнит соответствующее преобразование и применяет её.

3)

```
def zoom(self)
```

Метод меняет масштаб.

4)

```
def add_noise(self, intensity=30)
```

Принимает интенсивность и соответствующе зашумляет изображение.

5)

```
def add_blur(self, radius=1)
```

Применяет фильтр Гаусса с данным радиусом для размытия.

6)

```
def add_texture(self, texture_path)
```

Добавляет текстуру к изображению, которую берет по данному пути

7)

```
def add_glare(self, intensity=0.5, radius=100,
position=None)
```

Добавляет на изображение в случайную или заданную позицию блик. Также принимает параметры для радиуса и интенсивности.

8-10)
Три функции с аналогичными сигнатурами

```
def adjust_brightness(self, factor=0.3)
def adjust_contrast(self, factor=0.3)
def adjust_saturation(self, factor=0.3)
```

Соответственно изменяют яркость, контрастность и насыщенность с заданным фактором. Фактор < 1 уменьшает параметр, > 1 увеличивает.

Все эти функции возвращают измененное изображение и вызывают функцию из пункта ниже

11)

```
def create_new_json(self, coords,
new_image_name, output_filename)
```

Принимает новые координаты разметки, имя измененного изображения кода и название выходного файла, создает новую разметку и возвращает путь до нее.

Дальнейшие шаги:

1) Создать удобный способ для запуска обоих частей для генератора это выбор кодов и данных, для синтезатора -

выбор того, какие преобразования применить к изображению и возможность их комбинировать (возможно графический интерфейс)

2) Добавление новых типов кодов и преобразований

Список зависимостей:

- numpy
- PIL.Image, PIL.ImageFilter, PIL.ImageDraw, PIL.ImageEnhance
- json
- math
- os
- random
- subprocess
- yaml
- shutil
- re
- string