МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ОБНАРУЖЕНИЯ И СЕГМЕНТАЦИИ QR-КОДОВ НА ИЗОБРАЖЕНИИ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Кирносова Кирилла Павловича

Научный руководитель
зав.каф., доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2025

1 Содержание работы

На сегодняшний день QR-коды являются одним из наиболее популярных видов двумерных штрихкодов. Они активно используются в самых различных сценариях: оплата, аутентификация, реклама и др. Классические методы распознавания QR-кодов достаточно хорошо выполняют поставленную задачу, тем не менее имеют свои ограничения: в сложных условиях, таких как, неравномерное освещение, искажение изображения при нетипичной перспективе, необычная стилизация кода, могут не справляться с задачей. Данную проблему можно решать с использованием нейронных сетей, решающих задачи компьютерного зрения.

Для решения данной проблемы предлагается использовать нейросетевые модели из семейства YOLO, способные решать задачи детекции и сегментации объектов на изображении. Ключевой особенностью данных моделей по сравнению с их альтернативами — высокая скорость обработки изображения, при сохранении достаточно высокой точности.

Цель работы: разработка приложения для извлечения и распознавания QR-кодов на изображениях из буфера обмена с использованием нейронных сетей.

Задачи:

1. Создание датасета: сбор необходимых данных или создание программы для генерации синтетических данных для обучения модели.
2. Обучение: обучение выбранной модели на полученных данных.
3. Тестирование модели: оценка качества работы обученной модели.
4. Разработка приложения: создание десктопного приложения.

Структура и объём работы. Бакалаврская работа состоит из введения, 2 основных глав, заключения, списка использованных источников и 3 приложений. Общий объем работы — 67 страниц, из них 41 страница – основное содержание, включая 16 рисунков и 5 таблицу, Flash-накопитель в качестве приложения, список использованных источников информации — 21 наименование.

Первая глава имеет название «Анализ существующих нейросетевых моделей и выбор подхода к решению задачи» и содержит информацию о существующих подходах к сканированию QR-кодов и возможных применениях нейронных сетей для решения данной задачи.

Вторая глава имеет название «Разработка приложения», данная глава со-

держит подробное описание процесса выполнения работы.

Выпускная квалификационная работа заканчивается заключением, списком использованных источников, а также приложениями с кодами А-В.

С учётом широкого распространения QR-кодов и их применения в критически важных задачах, возрастает универсального и удобного инструмента для их распознавания. Классические алгоритмы работают быстро и эффективно в большинстве случаев, однако теряют устойчивость при наличии визуальных искажений. Современные методы на основе нейронных сетей позволяют решать задачи обнаружения и сегментации объектов в таких сложных ситуациях. В связи с этим возникает задача обучения соответствующей модели и интеграции её в приложение, способное выполнять распознавание QR-кодов и предоставлять пользователю удобный интерфейс.

Все подходы к сканированию QR-кодов можно условно разделить на 3 группы:

- Классические — алгоритмы, не использующие машинное обучение, например, решения на основе ZBar, ZXing.
- Нейросетевые — алгоритмы, использующие методы машинного обучения для обнаружения и декодирования QR-кодов.
- Комбинированные — алгоритмы, использующие нейросетевые модели для обнаружения кодов и помощи в дальнейшей предобработке. Декодирование же происходит с помощью классических алгоритмов.

В данной работе мы рассмотрим последний вариант, так как классические алгоритмы на данный момент лучше себя показывают в вопросе непосредственного декодирования QR-кода и извлечения содержащейся в нём информации, благодаря своей более высокой точности. В тоже время, модели компьютерного зрения лучше справляются с поиском объектов в условиях геометрических искажений и неравномерной освещённости объектов.

Основные проблемы сканирования QR-кодов:

- Геометрические искажения: основными искажениями при фотографии являются: дисторсия, искажение перспективы и поворот.
- Неравномерная освещённость.
- Стилистически необычные QR-коды: сюда входят нестандартный выбор цветов, градиентная раскраска, нестандартные формы некоторых частей QR-кода.

— Частичное перекрытие QR-кода сторонними объектами.

В данной работе для решения проблемы перспективного искажения применяется метод сегментации объектов на изображении. Для обработки неравномерной освещённости и нестандартных цветов QR-кода применяется алгоритм бинаризации изображения.

Раздел «Обзор современных моделей сегментации объектов» посвящён различным архитектурам нейронных сетей, способным решать задачу сегментации объектов.

Для поиска QR-кода и выделения его границ на изображении необходимо решать задачу сегментации объектов (англ. instance segmentation). На сегодняшний день существует множество моделей, решающих данную задачу. Их условно можно разделить на две группы:

— Модели на основе свёрточных нейронных сетей (CNN) — сюда входят такие модели, как Mask R-CNN и YOLO
— Модели на основе архитектуры Transformer: SAM, SEEM, CO-DETR.

Модели на основе архитектуры Transformer, как правило, имеют многократно больше нейронов, как следствие более требовательны к вычислительной мощности устройства и, в среднем, более медленны в работе. Поэтому предпочтение отдано моделям на основе свёрточных нейронных сетей.

YOLO, как и R-CNN, от которой произошла в конечном итоге Mask R-CNN, изначально были созданы для решения задачи обнаружения объектов (англ. object detection), а именно выделения ограничивающих рамок. Существуют два основных подхода к решению этой задачи, основанных на использовании свёрточных нейронных сетей:

— Двухэтапные методы (англ. two-stage methods) — подход, разделённый на два этапа. На первом этапе селективным поиском или с помощью специального слоя нейронной сети выделяются регионы интереса (англ. regions of interest, RoI) — области, с высокой вероятностью содержащие внутри себя объекты. На втором этапе выбранные регионы рассматриваются классификатором для определения принадлежности исходным классам и регрессором, уточняющим местоположение ограничивающих рамок. К данным методам относится модели RCNN и все основанные на данном семействе модели.
— Одноэтапные методы (англ. one-stage methods) — подход, не использую-

щий отдельный алгоритм для генерации регионов, вместо этого предсказывая координаты определённого количества ограничивающих рамок с различными характеристиками, такими, как результаты классификации и степень уверенности и в дальнейшем корректируя местоположение рамок. К ним относятся модели YOLO, SSD.

В данной работе, учитывая важность скорости обработки изображения, предпочтение отдаётся одностадийной модели YOLO, которая обладает высокой производительностью при сохранении достаточной точности.

Раздел «Архитектура современных моделей YOLO» посвящён описанию внутреннего устройства последних версий моделей из семейства YOLO. По результатам сравнительного анализа последних версий модели (11-ой и 12-ой) было решено использовать модели 11-ой итерации, в частности, модель s.

К разрабатываемому в ходе работы приложению был предъявлен следующий перечень функциональных требований:

1. Работа в фоновом режиме — даже когда окно приложения закрыто.
2. Использование глобально активных горячих клавиш.
3. Использование уведомлений для оповещений пользователя о исполнении программы.
4. Сканирование изображений.
5. Сохранение истории предыдущих сканирований.
6. Вывод результатов сканирования в окне приложения.

Для обучения выбранной модели машинного обучения было решено использовать синтетические данные, так как они предоставляют возможность создавать более разнообразные и сбалансированные датасеты.

Раздел «Обучение и тестирование нейронной сети» посвящён описанию скрипта, с помощью которого производилась генерация синтетических данных, обучению модели и последующему её тестированию. В ходе работы для написания скрипта был использован язык программирования Python и сторонние библиотеки для данного языка, перечисленные в подразделе «Список используемых инструментов».

Алгоритм генерации, использованный для генерации данных состоит из следующих этапов:

1. Генерация кодируемой информации. Создаётся строчка из произвольных букв, содержащихся в ASCII-таблице. Длина строки выбирается равнове-

роятно из заданного диапазона. Учитывая, что в кодировке UTF-8, ASCII-символы занимают 1 байт информации, длина строки совпадает с её информационным размером в байтах.

2. Создание QR-кода на основе полученной на предыдущем этапе строки. Уровень коррекции выбирается случайным образом.
3. Создание стилизованной маски — на данном этапе полученному QR-коду придаётся форма: выбирается одна из трёх возможных форм пикселей и, независимо, одна из трёх форм поисковых узоров.
4. Окраска маски — данный этап отвечает за покраску полученной маски, которая помимо классической чёрно-белой может быть инвертированной, цветной, градиентно раскрашенной.
5. Преобразование данных в формат библиотеки Albumentations. Данный этап необходим для одновременной обработки как самого изображения, так и маски сегментации, ключевых точек и ограничивающей рамки при последующих аугментациях.
6. Первый блок аугментаций — применяется непосредственно к полученному ранее QR-коду.
7. Наложение QR-кода на случайное фоновое изображение.
8. Второй блок аугментаций — данный блок применяется после наложения QR-кода на фоновое изображение.
9. Финальная обработка — сюда входит перевод данных в формат YOLO, добавление экранного каше и масштабирование полученного изображения.

В ходе работы было проведено несколько экспериментов по обучению модели: по результатам экспериментов дорабатывался разработанный алгоритм генерации синтетических данных, изменялось количество используемых реальных данных, а также постепенно увеличивалось количество эпох и размер выборок для обучения. Детали трёх экспериментов представлены в таблице 1.

Таблица 1 – Параметры проведённых экспериментов по обучению выбранной модели

Номер эксперимента	Размер тренировочной выборки	Размер валидационной выборки	Количество эпох
Первое обучение	1000	160	80
Второе обучение	2500	500	120
Третье обучение	3000	1000	150

В каждом эксперименте после соответствующего обучения модели было проведено тестирование модели на тестовой выборке из 160 реальных изображений. Результаты представлены в таблице 2

Таблица 2 – Результаты сравнения моделей на тестовой выборке

Модель	mAP_{box} 50	mAP_{box} 50-95	mAP_{mask} 50	mAP_{mask} 50-95
Первый эксперимент	0.6264	0.4373	0.6275	0.4390
Второй эксперимент	0.8139	0.5578	0.8052	0.5436
Третий эксперимент	0.9350	0.6366	0.9223	0.6271

Дальнейшее тестирование последней полученной модели на более крупном наборе данных из 320 реальных изображений показало следующие результаты (см. таблицу 3):

Таблица 3 – Результаты тестирования третьей модели

Набор изображений	mAP_{box} 50	mAP_{box} 50-95	mAP_{mask} 50	mAP_{mask} 50-95
Большая выборка	0.8947	0.6201	0.8893	0.6093

На основе полученных результатов были сформулированы следующие выводы по обучению модели:

— mAP_{box} 50 = 0.8947 и mAP_{mask} 50 = 0.8893 при тестировании говорят о том, что модель уверенно определяет местоположение и форму объектов.

— mAP_{box} 50–95 = 0.6201 и mAP_{mask} 50–95 = 0.6093 указывают на устойчивую точность модели при разных значениях IoU, что особенно важно для практического применения.

Глава «Разработка приложения» посвящена описанию структуры разработанного приложения.

Ниже приведено описание структуры проекта и диаграмма взаимосвязей на рисунке 1.

— *main.py* отвечает за запуск приложения и инициализацию всех необходимых компонентов.
— Папка core содержит основные компоненты приложения, не относящиеся к графическому интерфейсу.
    1. *hotkey_listener.py* содержит класс *HotkeyListener* — класс, ответственный за обработку горячих клавиш.
    2. *notifier.py* содержит класс *Notifier* — класс, ответственный за отображение уведомлений о результатах работы программы пользователю.
    3. *scan_proccesor.py* содержит в себе класс *ScanProccesor*, ответственный непосредственно за обработку изображений, распознавание QR-кодов и последующую фильтрацию результатов.
— Папка *ui* содержит компоненты графического интерфейса приложения:
    1. *main_window.py* содержит реализацию основного окна приложения в виде класса *MainWindow*.
    2. *history_panel.py* содержит класс *HistoryPanel* — виджет, который находится в левой части окна приложения и отвечает за организацию истории сканирований.
    3. *display_widget.py* содержит класс *DisplayWidget*, который реализует функционал, связанный с просмотром найденных на изображении QR-кодов и отображением их содержимого.
    4. *tray_menu.py* содержит класс *TrayMenu*, который ответственен за взаимодействие с приложением, находящемся в системном трее (когда, например, окно приложения закрыто).
— Папка *utils* содержит в себе компонент *files.py*, внутри которого реализованы необходимые для работы с файловой системой функции.
— Папка *resources* содержит в себе иконку приложения *icon.png* и веса обученной модели YOLO, *best.pt*.
— Папка *history* содержит список изображений и полученных из этих изображений данных. Изображения находятся в подпапке *images*, данные — в *scan_data*.
— Файл *config.json* хранит в себе настройки приложения. На данный момент, это только настройка горячей клавиши для инициации сканирования.

— Файл *requirements.txt* содержит список зависимостей приложения.

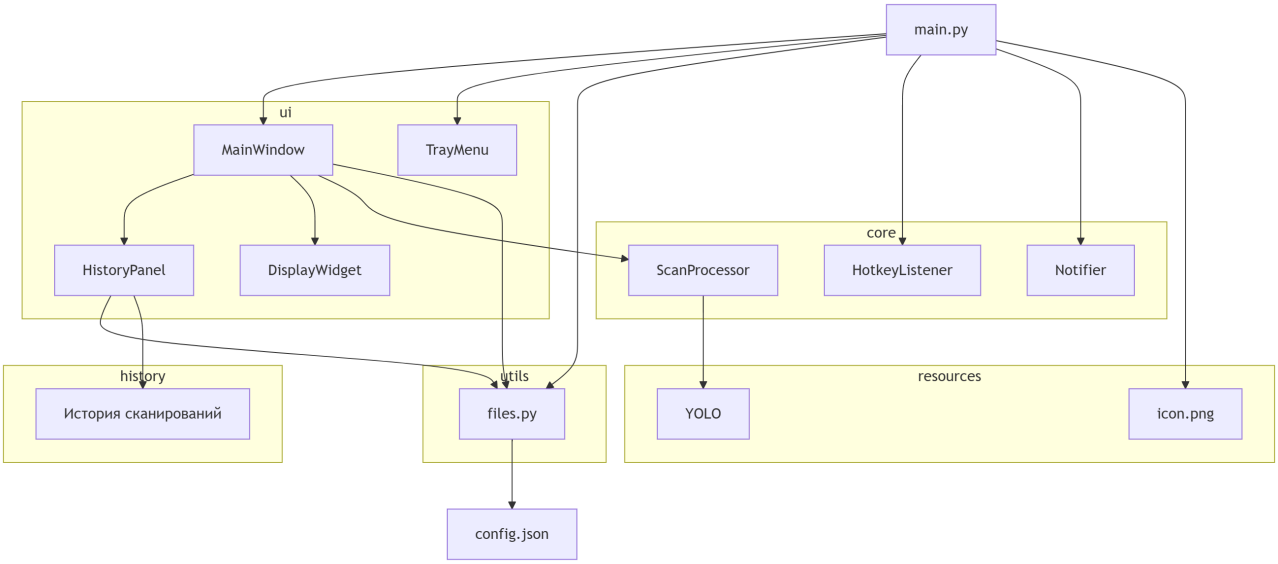

Рисунок 1 – Диаграмма взаимосвязей компонентов

Реализация класса *ScanProcessor* содержит 1 основной метод (*process()*) и 7 внутренних, выполняющих вспомогательную функцию. Метод *process()* принимает в качестве аргумента путь к изображению, которое необходимо отсканировать. Результатом метода является список найденных и отсканированных QR-кодов. Каждое вхождение этого списка представляет собой словарь, где поле *text* отвечает за содержимое QR-кода, а *poly* — за четырёхугольную рамку, ограничивающую данный QR-код.

Алгоритм состоит из нескольких вложенных циклов:

1. Внешний цикл перебирает коэффициент, с которым изображение будет масштабировано, прежде чем передано в модель для предсказания масок сегментации. За масштабирование и паддинг отвечает метод *_scale_and_pad_image()*. За запуск отвечает метод-обёртка *_polygon_iou*.
2. Далее в двух вложенных циклах перебираются ещё два параметра: количество итераций диляции, которое будет применено к маске сегментации (переменная *iterations*) и множитель масштаба (переменная *scale*), который определяет коэффициент увеличения (уменьшения) финального четырёхугольника, ограничивающего QR-код.

3. На основе параметров, перебираемых в двух внутренних циклах, на полученную из нейронной сети маску применяется диляция. Затем по маске строится контур, на основе которого, с помощью библиотеки QuadrilateralFitter строится аппроксимирующий маску четырёхугольник. За эти операции отвечает метод *_fit()*.
4. Полученный четырёхугольник масштабируется с заданным коэффициентом *scale*. Эта операция реализована в методе *expand_polygon()*.
5. С помощью средств библиотеки OpenCV производится обратное перспективное преобразование, которое отображает область, ограниченную полученным четырёхугольником в квадрат определённого размера (метод *_warp_to_rectangle()*).
6. Изображение преобразуется к оттенкам серого, за чем следует избавление от шумов с использованием Гауссовского размытия.
7. Полученное изображение бинаризуется с использованием метода Оцу [**?**] и передаётся в функцию-декодер в двух версиях: обычной и с инвертированными цветами (функция *decode()* из библиотеки pyzbar).
8. Результаты, полученные со всеми комбинациями параметром, конкатенируются и фильтруются на наличие неудачных сканирований и результатов-дубликатов. Фильтрацию осуществляет метод *_filter_results()*. Дубликаты определяется на основе метрики IoU, реализация которой для ограничивающих многоугольников находится в методе *_polygon_iou()*.

Было проведено тестирование полученного алгоритма. Для последующего сравнения и оценки в сравнение были добавлены алгоритмы считывания QR-кодов, реализованные в библиотеках pyzbar и OpenCV. Для тестирования были взяты 320 изображений, использованных ранее для тестирования обученной модели. Результаты представлены в таблице 4.

Таблица 4 – Результаты тестирования алгоритмов сканирования

Реализация алгоритма	Успешные сканирования (всего)	Средние затраты времени на одно изображение (мс)
pyzbar	118	13
OpenCV	178	16
ScanProcessor	250	953

По результатам измерений можно сделать вывод, что разработанный алгоритм выполняет поставленную задачу — он улучшает потенциал классических алгоритмов к сканированию QR-кодов. Тем не менее, он значительно уступает им в скорости, что является его главным недостатком.

Далее следует описание графического интерфейса разработанного приложения.

При закрытии основного окна приложение не заканчивает свою работу. Чтобы остановить приложение или снова открыть основное окно необходимо воспользоваться контекстным меню у иконки приложения в системном трее (двойное нажатие по иконке левой кнопкой мыши развернёт окно).

В левой части окна расположена панель истории (за неё отвечает класс *HistoryPanel*), на которой можно переключаться между сохранёнными сканированиями. Снизу панели расположена кнопка очистки истории, когда удаляет все сохранённые изображения и результаты.

В правой части расположен блок, отвечающий за просмотр изображения и результатов (этот блок реализован в классе *DisplayWidget*). С помощью него можно просматривать все успешно отсканированные на изображении QR-коды, а также содержимое каждого из них.

Для того, чтобы отсканировать изображение, находящееся в буфере обмена, необходимо нажать определённое сочетание клавиш (по умолчанию это комбинация клавиш *Ctrl*, *Shift* и *Q*), можно изменить в файле *config.json*). О результате сканирования пользователю придёт уведомление и в случае успеха, результат сохранится в панели истории.

Внешний вид графического интерфейса можно увидеть на рисунке 2.

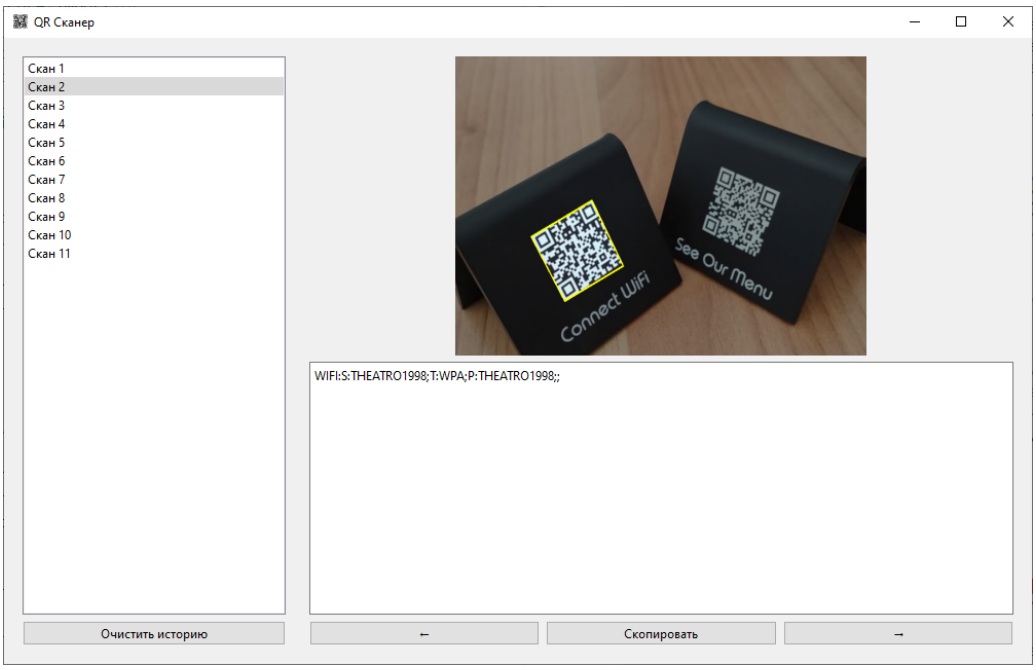

Рисунок 2 – Внешний вид окна приложения