МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

ГЛЫЗДОВ Станислав Юрьевич

# РАЗРАБОТКА СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ И ВЫБОРА АЛГОРИТМОВ ПРИ РЕШЕНИИ ЗАДАЧИ РАСПОЗНАВАНИЯ НОМЕРОВ АВТОМОБИЛЕЙ

Дипломная работа

Научный руководитель:
Кандидат физико-
математических наук, доцент
В.А. Образцов

Допущена к защите
«____»_____20___г.
Заведующий кафедрой информационных систем управления
доктор технических наук, доцент А.М. Недзьведь

Минск, 2025

# ОГЛАВЛЕНИЕ

# РЕФЕРАТ

Дипломная работа, 47 страниц, 8 рисунков, 3 таблицы, 4 формулы, 2 приложения, 12 источников.

**Ключевые слова:** КОМПЬЮТЕРНОЕ ЗРЕНИЕ, РАСПОЗНАВАНИЕ АВТОМОБИЛЬНЫХ НОМЕРОВ, НЕЙРОСЕТИ, СИСТЕМА ТЕСТИРОВАНИЯ.

**Объект исследования** – алгоритмы распознавания автомобильных номеров.
**Цель исследования** - рассмотрение задачи обнаружения номеров автомобилей, выделения их на изображении с последующим чтением, реализация нескольких решений этих задач, а также разработка системы для тестирования и выбора алгоритмов для решения этих задач.
**Методы исследования** - изучение соответствующей литературы, реализация нескольких решений задачи распознавания номеров автомобилей, анализ изученного материала.
**Результатом** является система для тестирования и выбора алгоритмов для решения задачи распознавания автомобильных номеров.
**Область применения результатов** - разработка эффективной системы распознавания автомобильных номеров.

# РЭФЕРАТ

Дыпломная праца, 47 старонак, 8 малюнкаў, 3 табліцы, 4 формулы 2 дадатка, 12 крыніц.

**Ключавыя словы:** КАМП'ЮТЭРНЫ ЗРОК, РАСПАЗНАВАННЕ АЎТАМАБІЛЬНЫХ НУМАРОЎ, НЕЙРАСЕЦІВЫ, СІСТЭМА ТЭСТАВАННЯ.

**Аб'ект даследавання -** алгарытмы распазнання аўтамабільных нумароў.
**Мэта даследавання -** разгляд задачы выяўлення нумароў аўтамабіляў, выдзялення іх на малюнку з наступным чытаннем, рэалізацыя некалькіх рашэнняў гэтых задач, а таксама распрацоўка сістэмы для тэставання і выбару ал-гарытмаў для вырашэння гэтых задач.
**Метады даследавання -** вывучэнне адпаведнай літаратуры, рэалізацыя некалькіх рашэнняў задачы распазнання нумароў аўтамабіляў, аналіз изученного матэрыялу.
**Вынікам** з'яўляецца сістэма для тэставання і выбару алгарытмаў для вырашэння задачы распазнання аўтамабільных нумароў.
**Вобласць ужывання вынікаў -** распрацоўка эфектыўнай сістэмы распазнання аўтамабільных нумароў.

# SUMMARY

Graduate work, 47 pages, 8 figures, 3 tables, 4 formulas, 2 appendix, 12 sources.

**Keywords:** COMPUTER VISION, LICENSE PLATE RECOGNITION, NEURAL NETWORKS, TESTING SYSTEM.

**The object of the study** is car license plate recognition algorithms.
**The purpose of the study** is to consider the problem of detecting car numbers, highlighting them in an image and then reading them, implementing several solutions to these problems, as well as developing a system for testing and selecting algorithms to solve these problems.
**Research methods** include the study of relevant literature, the implementation of several solutions to the problem of car license plate recognition, and the analysis of the studied material.
**The result** is a system for testing and selecting algorithms for solving the problem of car license plate recognition.
**Scope of the results** is development of an effective license plate recognition system.

# ВВЕДЕНИЕ

Задача распознавания автомобильных номеров никогда не теряет своей актуальности. Применений системам автоматического распознавания автомобильных номеров нашлось несчётное множество и далеко не всегда эта проблема решена полностью. Каждый отдельный случай, в силу специфики задачи, может потребовать абсолютно нового подхода к решению, из-за чего приходится создавать новую систему распознавания. И каждый новый случай требует тестирования для выявления соответствия предъявленным требованиям.

Одной из ключевых проблем в разработке систем распознавания номеров является выбор подходящих алгоритмов обработки изображений. Существующий спектр методов разнится от классических алгоритмов, таких как метод выделения контуров и бинаризация, до современных подходов, основанных на глубоком обучении. Каждый из этих методов имеет свои преимущества и недостатки, что требует тщательной оценки их эффективности в различных условиях.

Наличие эффективной системы для тестирования и выбора алгоритмов имеет важное значение, так как позволит оптимизировать вышеупомянутый процесс и повысить точность распознавания. Существующие решения часто требуют значительных временных и вычислительных ресурсов.

Система, разработанная в этой работе, решит или, как минимум, положит начало решению всех вышеупомянутых проблем.

Для этого необходимо сначала рассмотреть проблему ALPR изнутри, чтобы определить возможные проблемы, затем реализовать несколько подходов для её решения, чтобы иметь материал для тестов, который позволит наглядно показать принцип работы системы, после чего разработать систему для тестирования этих подходов.

# ГЛАВА 1
# АНАЛИТИЧЕСКИЙ ОБЗОР

В данной главе представлен краткий обзор существующих технологий и разработок в области распознавания номеров автомобилей. Сначала будет рассмотрена история направления, затем этапы развития технологий, начиная с первых попыток автоматизации распознавания, заканчивая современными методами, основанными на глубоком обучении. Далее будет рассмотрено текущее состояние проблемы, а также существующие решения и их применение в различных областях. Важное внимание будет уделено сложностям, с которыми сталкиваются разработчики при создании эффективных систем распознавания номеров.

## 1.1 История направления

Задача распознавания автомобильных появилась ещё в 20 веке и до сих пор является одной из ключевых задач компьютерного зрения. Сфера её применения включает автоматизацию транспортных систем, безопасность дорожного движения, контроль за соблюдением правил парковки и налоговой дисциплины. Несмотря на значительный прогресс, задача остается актуальной, особенно в условиях сложных погодных условий, низкого качества изображений и разнообразия шрифтов и форматов номеров в разных странах.

С задачей автоматического распознавания номерных знаков тесно связаны такие направления, как распознавание текста в сцене и чтение чисел в дикой природе (number reading in the wild), целью которых является обнаружение и считывание текста или чисел в естественных сценах. Несмотря на то, что задачу автоматического распознавания номерных знаков можно рассматривать как частный случай распознавания текста в сценах, эти задачи имеют ряд особенностей. В случае номерных знаков необходимо распознавать символы и цифры (при этом разнообразие шрифтов обычно невелико), и задача не предполагает использования семантической информации. Распознавание сцен, напротив, ориентировано на обработку текстов с высоким разнообразием шрифтов и часто включает использование лексических и семантических признаков, как, например, в работе [1]. Задача чтения чисел также не предполагает использование семантики, но её реализация проще, чем в случае номерных знаков, так как она не сталкивается с распространёнными путаницами между цифрами и буквами — такими как B и 8, D и 0, 1 и I, 5 и S.

«Точность технологии оптического распознавания символов (Optical Character Recognition, OCR) вносит существенный вклад в качество работы

LPR-системы в целом [2]». На ранних этапах задача распознавания автомобильных номеров решалась с использованием традиционных алгоритмов обработки изображений. Сначала проводили предварительную обработку изображения, затем отделяли номера от фона на основе анализа контуров и геометрических признаков, а в конце использовали алгоритмы OCR, основанные на эвристических правилах для распознавания символов. Но эти методы давали плохие результаты, сильно зависящие от качества входных данных, условий съёмки и не справлялись с перекрытыми, поврежденными или грязными номерами.

С развитием вычислительных мощностей и статистических методов началось использование машинного обучения. Применялись новые методы классификации, улучшенные алгоритмы сегментации, использующие морфологические операции и кластерный анализ, применялись первые нейронные сети: многослойные перцептроны (MLP) для распознавания символов. Данные подходы позволили не только повысить точность распознавания, но и улучшить устойчивость к изменениям освещения и фона, однако процесс подготовки данных и обучения оставался трудоемким.

С появлением глубокого обучения современные технологии начали развиваться в другом направлении, и в настоящее время во многих работах используются свёрточные нейросети из-за их высокой точности для обнаружения и распознавания объектов общего назначения [3,4]. Появилась возможность использовать генеративные модели для синтеза данных и увеличения объемов обучающей выборки. Фактически, глубокие сверточные нейросети стали ведущим методом машинного обучения, применяемым для определения транспортных средств и номерных знаков [5,6].

Современные методы обладают высокой точностью даже в сложных условиях (плохое освещение, частичная окклюзия). Однако остаются нерешенные проблемы, связанные с адаптацией моделей к разнообразным форматам номеров и снижением вычислительной нагрузки.

## 1.2 Состояние проблемы

В литературе встречаются различные подходы к задаче распознавания автомобильных номеров. Встречались как решения, основанные на нейросетевых методах, так и на более классических подходах. Хотелось бы выделить, что классические методы при решении этой задачи считаются более производительными, однако при этом необходимо подчеркнуть их ограниченность. В отличие от нейросетевого подхода классические методы недостаточно гибкие и могут показывать удовлетворительную точность только с теми типами номеров и в тех ситуациях, для которых они предназначены. В

противовес этому нейросети показывают хорошие результаты на любых типах номеров, однако получение высокой точности распознавания сопряжено с серьёзными затратами при обучении, ведь это достаточно затратный по ресурсам процесс.

В чём все источники сходятся во мнении, так это в том, что условно алгоритм распознавания можно разделить на следующие несколько основных частей (выбран наиболее общий из приведённых):

- Локализация номерного знака на изображении – нахождение и выделение автомобильного номера на изображении.
- Изменение размеров и ориентации - приведение размеров изображения и изменение его перспективы для лучшей читаемости.
- Нормализация – применение фильтров для приведения изображений к единому виду.
- Выделение символов на изображении.
- Синтаксический/геометрический анализ – учёт символов и их позиций для проверки правил, действующих в конкретных странах.

«Иногда процесс распознавания может завершиться сбоем, и обнаруженный номер может содержать ошибки. Некоторые из этих ошибок могут быть обнаружены с помощью синтаксического анализа распознанного номера. Если у нас есть регулярное выражение или правило для определения номерного знака конкретной страны, мы можем восстановить дефектные номерные знаки, используя это правило [7]».

## 1.3 Сложности решения задачи

При попытках решить задачу ALPR были выявлены различные проблемы, которые по большей части так или иначе связаны со спецификой практического применения этих систем. Вот перечень наиболее часто встречаемых:

- Низкое разрешение файла, как правило, из-за того, что фотокамера находится слишком далеко, но иногда из-за использования камеры низкого качества.
- Нечеткие изображения, особенно при движении.
- Плохое освещение и низкая контрастность из-за передержки, бликов или теней.
- Объект, закрывающий номерной знак или его часть, часто это буксировочный трос или грязь на номерном знаке.
- Изменение полосы движения транспортного средства в поле зрения камеры во время считывания номерного знака.

- Другой шрифт для персонализированных автомобильных номеров (в некоторых странах такие номера запрещены, что устраняет проблему).
- Методы обхода правил.
- Отсутствие координации между странами (два автомобиля из разных стран могут иметь одинаковый номер, но различный дизайн номерного знака).

Хотя некоторые из этих проблем могут быть исправлены с помощью программного обеспечения, решение этих проблем часто может быть достигнуто и другими путями. Увеличение высоты камеры может избежать проблем, связанных с тем, что объекты (например, другие транспортные средства) закрывают обзор, но это создает и усугубляет другие проблемы, такие как необходимость регулировки с учетом увеличенного наклона экрана. На некоторых автомобилях могут быть закрыты один или два знака номерного знака. Некоторые малогабаритные системы допускают некоторые ошибки в номерном знаке. При использовании для обеспечения доступа определенных транспортных средств к закрытой зоне может быть принято решение о допустимой частоте ошибок в один знак. Это объясняется тем, что вероятность того, что неавторизованный автомобиль будет иметь такой похожий номерной знак, довольно мала. Однако такой уровень неточности был бы неприемлем в большинстве применений системы ALPR.

## 1.4 Выводы

Задача распознавания автомобильных номеров имеет большую историю и затрагивает многие сферы жизни, о некоторых из которых мы даже не догадываемся.  И на протяжении всей истории существования этой задачи совершенствовались методы её решения. Классические методы постепенно сменяют нейросетевые. Какие-то факторы, мешающие решить задачу, устраняют программно, какие-то с помощью современных технологий. Но задача будет актуальна до тех пор, пока не будет создана единая система, которой не мешают никакие факторы, или единственные существующие из них не могут быть исключены ни каким образом.

# ГЛАВА 2
# ОБЗОР ЗАДАЧИ И ПОДХОДОВ К ЕЁ РЕШЕНИЮ

В этой главе будет представлен детальный обзор задачи распознавания номеров автомобилей и подходов к её решению. Сначала будет дана формулировка основной задачи, затем описаны ключевые этапы, которые необходимо пройти для успешного решения этой задачи. Для каждого из этапов будут представлены метрики, использованные при проверке качества моделей. Этот обзор поможет глубже понять ключевые аспекты задачи распознавания номеров автомобилей и выделить наиболее эффективные подходы для её решения.

## 2.1 Постановка задачи

Задача распознавания номеров автомобилей (ALPR) заключается в автоматическом извлечении и интерпретации информации с номерных знаков транспортных средств.

Это максимально общая постановка задачи. Надо заметить, что в ней не указана также возможная необходимость первоначального нахождения транспортного средства на изображении и последующего нахождения номера. Также замечу, что решение задачи будет сильно отличаться, если начать изменять место применения этого решения. Так, например, распознавание номеров на автоматических парковках, где такие факторы, как освещение номера, размытие номера, погодные условия, угол наклона камеры, перспектива почти всегда постоянны и не будут меняться со временем, а значит и учитываться будут по умолчанию, оставляет за программистом только решение проблемы грязных и частично закрытых номеров. В то время как распознавание номера на трассе в видеопотоке для камер, следящих, например, за превышением скорости может оказаться совсем нетривиальной задачей, ведь ко всем вышеперечисленным проблемам добавится ещё и то, что на изображениях может находиться множество автомобилей. Важно также заметить, что выбор задачи влияет не только на подход к решению, но и на то, какие данные нужно будет собрать для обучения.

Это всё говорит о том, что перед решением задачи важно определить практическое применение своего решения. Для лучшего понимания специфики задачи и получения материалов, которые позже будут использованы в тестировании, в рамках работы мною были реализованы 2 версии алгоритма для решения следующей более простой в реализации задачи, чем распознавание номеров в видеопотоке на трассе, но не такой простой, как распознавание на автоматизированной парковке.

Предположим, что существует программа для парковщиков с платных уличных парковок, которое распознаёт номера и по номеру автомобиля определяет, проведена ли оплата. Особенности этой задачи в том, что номера часто могут быть загрязнены, условия съёмки могут оказаться любыми, камера смартфона часто имеет небольшое разрешение, а также размер модели не может быть излишне велик.

## 2.2 Модели обнаружения

Для обнаружения автомобильного номера на изображении были использованы две достаточно известные модели, дообученные на данных, соответствующих специфике выбранной задачи.

Первой моделью является YOLO 11 версии. Это один из наиболее популярных алгоритмов для обнаружения объектов в изображениях и видео. Он был представлен в 2015 году и стал основополагающим в области глубокого обучения для задач компьютерного зрения. Основная идея YOLO заключается в том, что алгоритм рассматривает изображение как единое целое, а не разбивает его на участки, что позволяет значительно ускорить процесс обнаружения объектов.

Эта модель была выбрана за то, что её можно очень просто обучить, внедрить куда-либо и использовать после этого. Также у YOLO есть возможность выбора размера модели, что коррелирует с тем, что упоминалось в постановке задачи.

Второй моделью стала Faster R-CNN. Это достаточно эффективный алгоритм для обнаружения объектов, который был представлен в 2015 году. Он является развитием предыдущих моделей R-CNN и Fast R-CNN и сочетает в себе высокую точность и хорошую скорость работы.

В свою очередь эта модель была выбрана за свою высокую точность и ещё большую скорость обучения, чем имелась у YOLO, что позволило провести больше экспериментов с её параметрами.

Для оценки качества моделей были использованы следующие метрики:

- Average Precision (AP) - доля правильно предсказанных объектов среди всех предсказанных объектов.

$$\mathrm{AP} = \frac{TP}{TP + FP}$$

  - TP (True Positives) — количество истинно положительных предсказаний (правильно определенные объекты).
  - FP (False Positives) — количество ложноположительных предсказаний (объекты, которые были предсказаны, но не существуют).

- mAP50 — это средняя точность при пересечении (Intersection over Union, IoU) на уровне 0.5. Она вычисляется как среднее значение точности для всех классов при фиксированном уровне IoU = 0.5.
- mAP50-95 — это усредненная точность при различных уровнях IoU от 0.5 до 0.95 с шагом 0.05.

Эти метрики широко используются в исследованиях в области компьютерного зрения из-за того, что позволяют адекватно оценить работу модели и своевременно заметить признаки переобучения.

## 2.3 Нормализация изображений

Этап нормализации при распознавании достаточно важен, если при съёмке возникает множество факторов, мешающих распознаванию. Мною было реализовано два способа нормализации: без предобработки и с предобработкой.

Причиной отказа от предобработки в первом случае можно обосновать отсутствием необходимости в предобработке во многих случаях. Многие системы распознавания имеют собственные встроенные методы предобработки, которым может помешать использованный ранее алгоритм, но это не единственный сценарий. Также этот метод позволит сравнить качество распознавания с предобработкой и без неё.

Для предобработки я использовал следующую последовательность операций. Все они были выполнены с использованием библиотеки opencv.

- Первый шаг стандартен для всех подобных алгоритмов – перевод изображения в оттенки серого.
- Для размытия и уменьшения шума далее использовался гауссовский фильтр размера 5x5:

$$\begin{bmatrix} 1/243 & 4/243 & 6/243 & 4/243 & 1/243 \\ 4/243 & 16/243 & 24/243 & 16/243 & 4/243 \\ 6/243 & 24/243 & 36/243 & 24/243 & 6/243 \\ 4/243 & 16/243 & 24/243 & 16/243 & 4/243 \\ 1/243 & 4/243 & 6/243 & 4/243 & 1/243 \end{bmatrix}$$

- Далее использовался оператор Собеля для выделения границ:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, f = \sqrt{G_x^2 + G_y^2}$$

«Границы объектов на изображении в значительной степени уменьшают количество данных, которые необходимо обработать, и в то же время сохраняет важную информацию об объектах на изображении, их форму, размер, количество [8]».

- Следующим использовался метод Гауссовского усреднения для адаптивного порогового преобразования, что позволило ещё лучше отделить границы знака от фона. Вместе с ним используется пороговое преобразование для бинаризации изображения: все границы становятся белыми, а остальное изображение чёрным. Этот метод показал себя лучше, чем схожий алгоритм Кенни. Так как приходится работать с изображениями при различных погодных условиях, в особенности при различном освещении, алгоритм, способный лучше подстроиться под эти меняющиеся условия, был более предпочтителен.
- После этого была задействована функция нахождения контуров с параметрами на нахождение только внешних контуров, без учёта внутренних, и аппроксимацией точек, позволяющей сохранять только конечные точки, что позволило избежать работы с лишними точками.
- Наконец из всех найденных точек оставляем только 4 ближайшие к углам номерного знака и производим преобразование перспективы с изменением размера над получившейся областью, но у изначального изображения до всех операций изображения.

Как правило, входные изображения большего размера позволяют обнаруживать объекты меньшего размера, но увеличивают вычислительные затраты [9]. Проведённые тесты выявили, что оптимальный размер – 300x100 пикселей.

В итоге у нас получается изображение 300x100, на котором находится только номерной знак, имеющий правильную перспективу. Это изображение хорошо подходит для последующего распознавания информации на нём.

## 2.4 Распознавание текста на номерном знаке

Для одного из важнейших этапов моего решения – выделения информации из знака, были использованы готовые и проверенные средства.

Первым средством стала система OCR – Tesseract. Вторым – библиотека EasyOCR. Это открытые системы оптического распознавания символов, разработанные для распознавания текста на изображениях. Они поддерживает большое множество языков и могут распознавать даже кириллицу. Tesseract хорошо справляется с распознаванием текста на высококачественных изображениях, хотя качество может снижаться при наличии искажений или плохой освещенности. Также его легко встроить в свой проект при необходимости. Эта система отличается совместным использованием нейросетевых и классических подходов распознавания текста на изображениях. EasyOCR отличается простотой использования. Он использует модели глу-

бокого обучения, что позволяет достигать высокой точности распознавания текста при низких временных затратах.

## 2.5 Выводы

В этой главе была поставлена задача и описаны её основные характеристики и определены ключевые шаги алгоритма её решения. Далее были описаны средства, которые будут использованы для решения этой задачи.

# ГЛАВА 3
# ПРАКТИЧЕСКИЕ ИСПЫТАНИЯ

В этой главе будут представлены итоги решения поставленной ранее в Главе 2 задачи распознавания номеров автомобилей. Будет проведено сравнение каждой пары подходов, для чего будут применяться ранее описанные метрики. Оценка методов, не имеющих своих метрик, будет выполняться органолептически.

## 3.1 Итоги обнаружения

Как уже упоминалось ранее, для решения была выбрана модель, которая имела бы наименьший размер. Для этих целей была использована предобученная модель yolo11n, которая после обучения имеет размер 5,92 МБ. Обучение заняло около 100 эпох. В результате была получена модель, которая хорошо обнаруживает знаки даже на больших расстояниях или в потоке машин. Большим плюсом для нашей задачи является то, что объект с наибольшим показателем уверенности почти всегда (на тестовой выборке из 882 изображений не было обнаружено лишь 5 ближайших к камере и, соответственно, наиболее чётких знаков) является интересующий нас знак. Однако она не лишена недостатков. Во-первых, показатель уверенности сети остаётся довольно низким, вне зависимости от расстояния до номерного знака или качества изображения. Во-вторых, периодически сеть относит надписи на машине к номерным знакам. В-третьих, что связано с предыдущими двумя, такие надписи имеют довольно высокую степень уверенности, что не может не удручать.

В противовес YOLO была использована несколько большая, но всё ещё вписывающаяся в установленные задачей рамки модель Faster R-CNN. После обучения её размер составил 314 МБ. Процесс обучения также слегка отличалось. Оно заняло 300 эпох при существенно меньших временных затратах. Результатом же стала модель, не сильно отличающаяся от полученной ранее. Отличием будет разве что только то, что уверенность на номерных знаках на тестовых данных не была ниже 97%. Остальные проблемы всё так же остались, хотя нахождение лишних надписей и можно списать на особенности работы R-CNN сети.

Метрики оценки качества модели можно увидеть в таблице 3.1.

Таблица 3.1 - Метрики качества обнаружения

| Вид сети | AP | mAP50 | mAP50-95 |
|---|---|---|---|
| YOLO | 98.8 | 98.3 | 71.1 |
| Faster R-CNN | 60.7 | 94.6 | 66.8 |

Примеры работы алгоритмов обнаружения отображены на рисунках 3.1, 3.2, соответственно.


**Рисунок 3.1 – Пример обнаружения сетью YOLO**


**Рисунок 3.2 – Пример обнаружения сетью Faster R-CNN**

## 3.2 Итоги работы с нормализацией

В ходе работы над предложенным мной алгоритмом нормализации был построен алгоритм удовлетворительной точности. В большинстве ситуаций, с которыми можно столкнуться при распознавании, он успешно справляется с возложенной на него задачей или по крайней мере улучшает изображение, увеличивая его и меняя перспективу на более подходящую (а для OCR это очень важно).

Результаты можно увидеть на рисунке 3.3, где в верхней строке номера до предобработки, а в нижней после неё.


**Рисунок 3.3 – Пример использования алгоритма нормализации**

## 3.3 Итоги работы с OCR

По результатам проведённых мной тестов, OCR Tesseract и EasyOCR показали себя вполне достойно. Они прекрасно справляются с распознаванием чётко видных номерных знаков и могут заменять друг друга на знаках низкого качества. Часто получалось, что то, что не могла распознать одна система, могла распознать другая. Результаты проверки проще будет представить в виде таблицы 3.2.

Таблица 3.2 – Сравнение использованных OCR систем

| | Tesseract | EasyOCR |
|---|---|---|
| Распознавание текста в высоком качестве | + | + |
| Распознавание текста в низком качестве | +- | -+ |
| Распознавание текста на мелких изображениях | + | - |
| Распознавание наклонённого текста | -+ | + |
| Распознавание только строго заданных символов | + | + |
| Точность | 74.89 | 72.34 |

## 3.4 Выводы

Результаты полученных компонент алгоритма считаю удовлетворительными и подходящими для дальнейшей работы с ними. В следующей главе будет создана система, позволяющая тестировать алгоритмы, а также автоматически выбирать наиболее эффективные комбинации компонент. Так же система будет иметь средства для вычисления метрик алгоритмов и авто-

матического создания тестовых изображений с настраиваемыми параметрами при их нехватке. Полученные же результаты будут использованы в качестве объекта тестирования.

# ГЛАВА 4
# РАЗРАБОТКА СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ И ВЫБОРА АЛГОРИТМОВ

В этой главе будет описано приложение, разработанное на языке программирования Python (ПРИЛОЖЕНИЕ Б). Полученная система имеет два режима работы: система тестирования и система анализа и выбора алгоритмов. Далее оба этих режима будут тщательно рассмотрены, также будут приведены шаги, позволяющие любому разработчику использовать данную систему.

## 4.1 Система для тестирования алгоритмов

Система тестирования алгоритмов позволяет проверить работоспособность алгоритма и увидеть результат роботы каждой его составной части по отдельности. Это позволяет сравнить свой алгоритм с каким-либо другим или даже заменить одну конкретную часть алгоритма, чтобы увидеть, как изменится результат.

Программа обладает простым интерфейсом с областями выбора составных частей алгоритма. После нажатия кнопки «Test» открывается окно выбора, в котором можно выбрать абсолютно любое изображение со своего устройства. После выбора оно пройдёт все этапы распознавания и пользователю откроются: исходное изображение с обозначенным рамкой номером, вырезанный и прошедший нормализацию номер, результат чтения этого номера, замеры времени, затраченного на каждый из этапов.

После этого можно изменить любую из компонент алгоритма и по повторному нажатию кнопки «Test» выбранное ранее изображение будет обработано новой версией алгоритма. Также можно нажать на кнопку «Clear», и приложение вернётся в исходное состояние.

В строке меню есть меню «Mode», содержащее одну кнопку «Switch», по нажатию которой происходит смена режима. Эта кнопка позволяет менять режим с тестового на режим анализа и выбора и обратно.

На рисунке 4.1 приведены 2 пробных запуска системы в режиме тестов на одном из изображений из тестовой выборки. Необходимо отметить, что OCR EasyOCR испытывает затруднения при распознавании текста на мелких изображениях, OCR Tesseract работает на них лучше и наоборот на крупных.

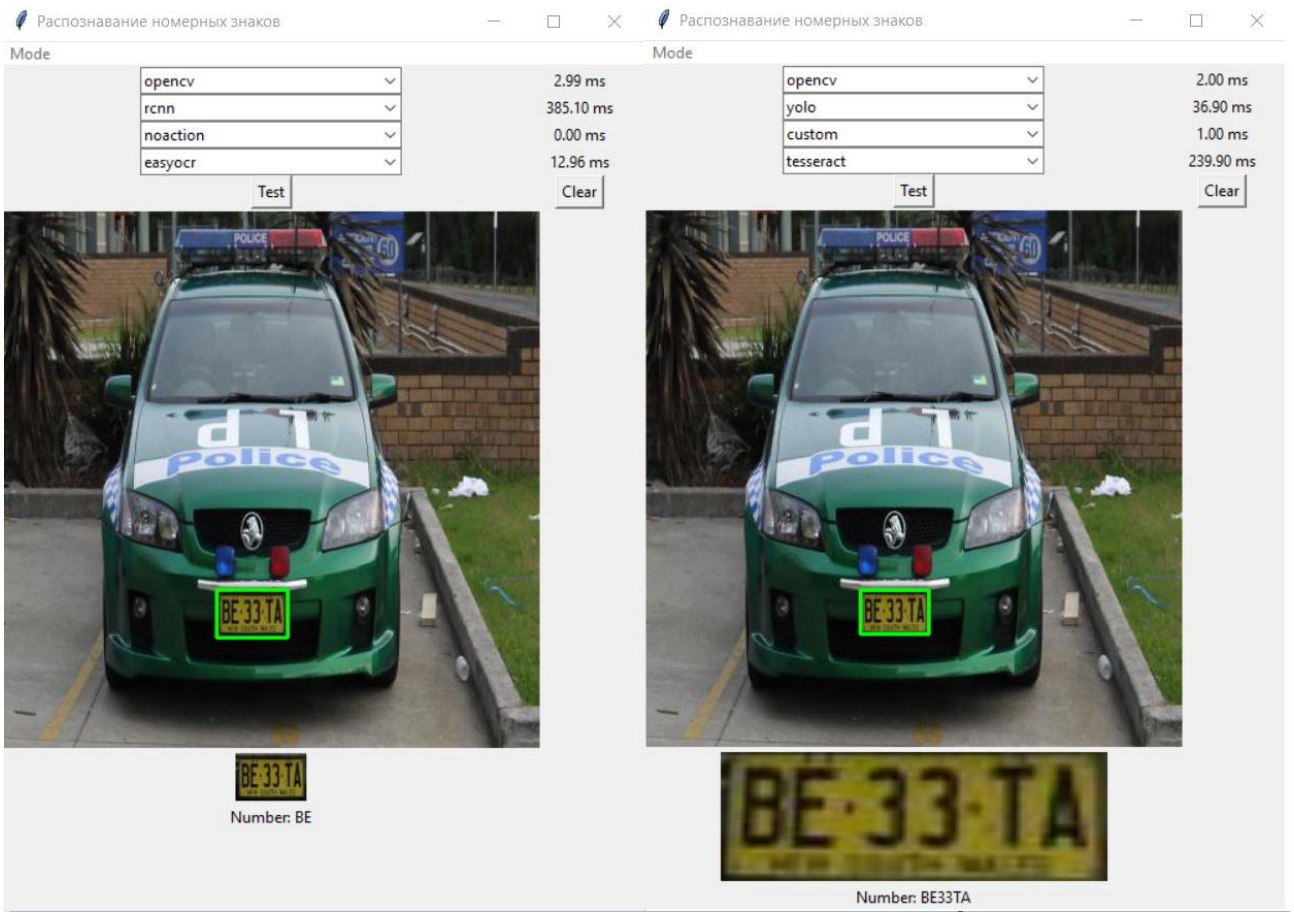


**Рисунок 4.1 – Примеры работы системы тестирования**

На рисунке 4.2 представлен исходный вид приложения (по умолчанию включен тестовый режим), являющийся довольно минималистичным и простым в использовании.

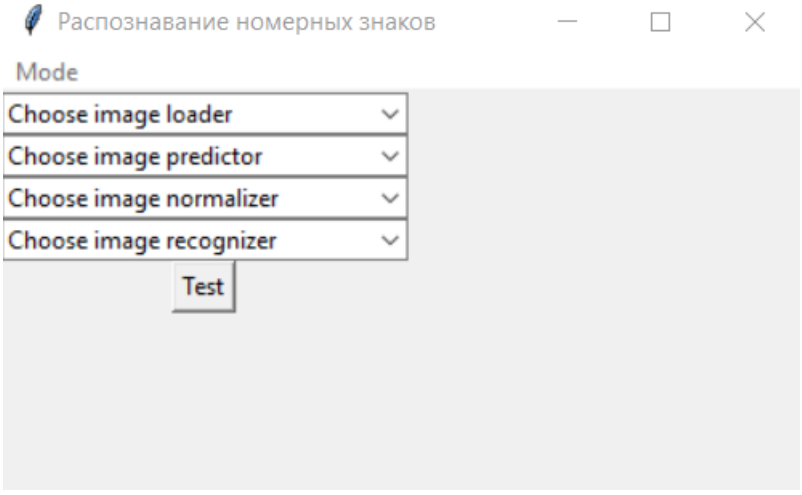


**Рисунок 4.2 – Система в исходном состоянии**

Таблица 4.1- Результаты тестирования через систему

| | YOLO | Faster R-CNN | С нормализацией | Без нормализации | Tesseract | EasyOCR |
|---|---|---|---|---|---|---|
| Точность на данных высокого качества | Высокая | Высокая | Высокая | - | Высокая | Высокая |
| Точность на данных низкого качества | Высокая | Высокая | Средняя | - | Средняя | Низкая |
| Точность на изображениях малого размера | Низкая | Средняя | Средняя | - | Средняя | Низкая |
| Размер | 5,92 МБ | 314 МБ | - | - | До 10 МБ | ≈50 МБ |
| Производительность | ≈150 мс | ≈150 мс | ≈1 мс | ≈0 мс | ≈200 мс | ≈50 мс |

Как можно заметить из рисунка 4.1, а также таблицы 4.1, в которой представлены усреднённые значения на всей выборке, составные части моего алгоритма показали схожую производительность, а это значит, что выбирать алгоритм следует исходя исключительно из характеристик моделей, таких как размер, точность на данных при интересующих нас погодных условиях и простота использования.

В моём случае лучшим выбором будет opencv+yolo+custom normalization+tesseract. Однако слабым местом в производительности этого алгоритма является загрузка изображения, которая занимает слишком много времени. Для исправления этого нужно найти более подходящую библиотеку для загрузки изображений.

Важно заметить, что этот результат получен тестированием алгоритмов вручную, что занимает время и может оказаться необъективным.

## 4.2 Система для анализа и выбора алгоритмов

Для того, чтобы выбор компонент был более обоснован, в моём приложении предусмотрен второй режим, перейти на который можно нажатием кнопки «Switch» в меню «Mode». Этот режим позволяет при нажатии кнопки «Test» выбрать директорию, в которой хранятся тестовые изображения. Далее эти изображения преобразуются в соответствии с выбранными опциями и помещаются в новое хранилище. После этого каждая компонента тестируется. Результатами тестирования являются некоторые метрики упомянутые в

Главе 2 и некоторые другие. Результат этого тестирования заносится в таблицы, позже эти таблицы можно будет просмотреть и определить наиболее подходящий алгоритм. Таким образом система позволяет не только выбрать наиболее эффективный для своей задачи алгоритм, но и протестировать его на большом объёме данных в различных условиях. Также система предоставляет свой собственный выбор лучшего алгоритма, однако он может не совпадать с нуждами разработчика, так как каждый из вычисляемых параметров алгоритма может оказаться более или менее важным для него в зависимости от ситуации.

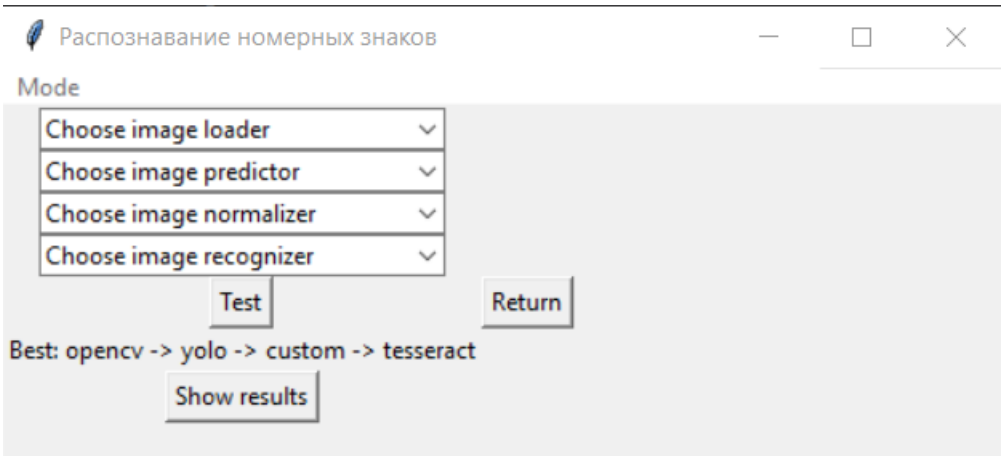


**Рисунок 4.3 – Примеры работы системы анализа**

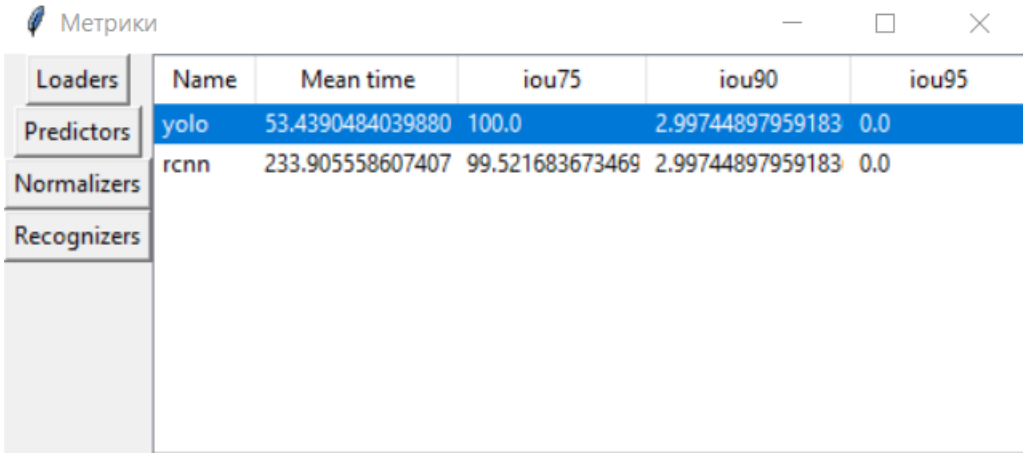


**Рисунок 4.4 – Окно метрик (выбраны метрики для средств обнаружения)**

На рисунке 4.3 изображён вид приложения после окончания процесса тестирования. Система позволяет повторно провести тестирование с теми же параметрами по нажатию кнопки «Test» (для случаев изменения производительности устройства или обстоятельств во время работы системы, повлиявших измерения), вернуться на экран выбора опций по нажатию кнопки «Return» и вывести окно метрик по нажатию кнопки «Show metrics». Окно вывода метрик изображено на рисунке 4.4, оно имеет две секции: в левой можно

увидеть кнопки, позволяющие выбрать, метрики каких алгоритмов просматривать, а в правой части находится таблица, в которой эти метрики выводятся.

## 4.3 Метрики, использованные для оценки качества алгоритмов

Для определения качества алгоритмов используются метрики. Однако для каждого набора данных значение метрик будет отличаться. В связи с этим моя система имеет встроенную подсистему подсчёта метрик. Каждая часть алгоритма подвергается вычислению своих собственных метрик. В этом пункте они будут рассмотрены.

### 4.3.1 Метрики загрузки

Загрузка – один из самых простых этапов алгоритма. В ней невозможно допустить ошибку или завершить её лучше/хуже, чем сможет другой алгоритм. Однако загрузка играет не последнюю роль в алгоритмах распознавания.

Основной характеристикой метода загрузки является скорость. Зачастую скорость загрузки может оказаться единственным фактором, замедляющим работу системы. Поэтому единственной метрикой классов загрузки является время.

После завершения загрузки не имеет значения, какой набор изображений передавать дальше по алгоритму (вследствие их идентичности), поэтому во избежание временных затрат на копирование далее используется полученный самым первым.

### 4.3.2 Метрики обнаружения

Обнаружение также играет не последнюю роль в алгоритме. Однако результаты обнаружения уже могут отличаться.

В качестве метрик для обнаружения в своей системе я использовал среднее время обнаружения для одного изображения, а также среднюю точность при пересечении (Intersection over Union, IoU) на уровнях 0.75, 0.9, 0.95.

$$\mathrm{IoU} = \frac{A \cap B}{A \cup B}$$

### 4.3.3 Метрики распознавания

Распознавание номера является ключевым элементом алгоритма. Даже после безошибочно проведённых предшествующих этапов, на этом этапе может быть допущена ошибка.

При оценке работы систем распознавания я оцениваю 3 параметра. Первым, конечно же является среднее время распознавания. Вторым оценивается частота абсолютно верно распознанных номеров, или Perfect Match Rate. Эта метрика оценивает долю изображений или строк, для которых результат работы OCR полностью совпадает с исходным текстом. Эту метрику можно взять, как основную для выбора лучшего алгоритма распознавания, однако бывают ситуации, когда необходимости в близкой к 100% точности нет. Также при очень близких показателях этой метрики может понадобиться более глубокий анализ. В таких случаях можно воспользоваться третьей метрикой. Character Error Rate (CER) измеряет частоту ошибок на уровне символов. Вычисляется по формуле:

$$\mathrm{CER} = \frac{\mathrm{Substitutions} + \mathrm{Insertions} + \mathrm{Deletions}}{\mathrm{Total\ Number\ of\ Characters\ in\ Reference}}$$

Здесь Substitutions — количество символов, неправильно распознанных (замененных). Insertions — количество добавленных несуществующих символов. Deletions — количество символов, пропущенных.

Эти метрики позволяют в полной мере оценить работу систем распознавания.

### 4.3.4 Метрики нормализации

Можно заметить, что при описании метрик был пропущен пункт, посвящённый нормализации. Это сделано не случайно. Так как нет метрик, которые могли бы оценить работу этого этапа, ведь сугубо индивидуален. Поэтому, кроме стандартной метрики среднего времени, затраченного на изображение, была придумана метрика читабельности, позволяющая оценить алгоритмы нормализации.

Эта метрика получается из метрики CER, используемой для OCR. Сначала на нормализованном множестве изображений подсчитывается CER для каждой из OCR. Затем вычисляется среднее значение. Это значение показывает средний процент нечитаемых для OCR символов после их нормализации. Чем меньше это значение, тем выше качество алгоритма. Я же вычитаю долю нечитаемых символов и получаю долю читаемых символов, или метрику читабельности.

## 4.4 Выбор алгоритма

Одним из важнейших этапов работы системы является выбор алгоритма. Этот этап не только важен, но и сложен, так как параметры, по которым можно судить об эффективности алгоритма, слишком слабо связаны, из-за чего создать метрику, учитывающую все нюансы и позволяющую численно выразить качество алгоритма, не представляется возможным. Однако подходящие метрики есть у компонент, из которых состоит алгоритм.

По метрикам составных частей алгоритма можно выбрать наиболее подходящие под конкретную задачу. На данный момент система способна выбирать алгоритм в двух режимах: самый быстрый, самый эффективный. Суть первого ясна из названия. Суть второго состоит в поиске решения без учёта времени (кроме загрузки, где время – единственная метрика качества). Для компонент обнаружения в первую очередь проверяется процент найденных номеров и затем уже качество их нахождения. Для нормализации учитывается читабельность. Для распознавания в первую очередь процент верно распознанных номеров, во вторую частоту ошибок на уровне символов. Все метрики описаны в пункте 4.3. Для более сложных вычислений существует окно метрик.

## 4.5 Подсистема для модификации изображений

Как уже упоминалось в Главе 4.2, система анализа и выбора предоставляет некоторые опции по модификации изображений для их лучшего соответствия задаче или проверки алгоритмов в различных условиях. На рисунке 4.5 представлен исходный вид системы с опциями для модификации.

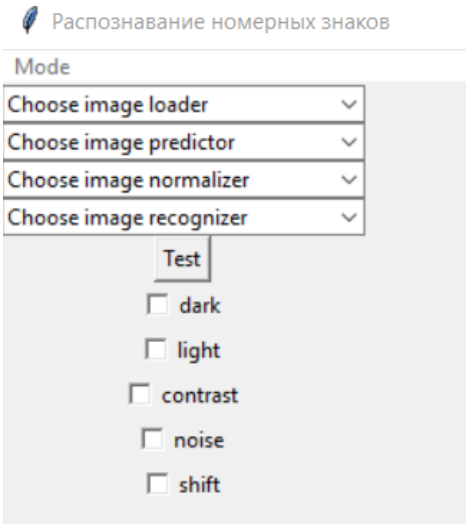


**Рисунок 4.5 – Система с опциями для модификации изображений**

Выбранные опции определяют, как будут изменяться изображения из выбранной пользователем директории. Приложение предоставляет следующие взаимодействия с изображениями:

- Уменьшение яркости изображения на 10%, 20%, 30%.
- Увеличение яркости изображения на 10%, 20%, 30%.
- Изменение контрастности изображения на -10%, -20%, -30%, 10%, 20%, 30%.
- Добавление шума величиной в 5%, 7.5%, 10% к изображению.
- Сдвиг цветов на -10%, -20%, -30%, 10%, 20%, 30%.

Одновременно может быть выбрано любое число опций. Подсистема возьмёт все изображения из выбранной пользователем директории, модифицирует их в соответствии с выбранными опциями и сохранит в собственную директорию. При этом каждая последующая опция применяется к результату, полученному после применения предыдущей. Это позволяет не только испытать алгоритм в других условиях, но и существенно увеличить выборку.

## 4.6 Работа с системой

Разработанная система обладает расширяемой архитектурой, что позволяет легко добавлять новые возможности, а также проста в использовании. Для использования сторонним пользователем система может быть преобразована в библиотеку.

Для взаимодействия с системой пользователю необходимо создать классы для загрузки изображений, обнаружения номеров, нормализации полученных результатов и распознавания номеров. Для этого были разработаны шаблонные классы (ПРИЛОЖЕНИЕ А), которые с помощью механизма наследования укажут, какие методы необходимы для корректной работы системы. Каждый метод дополнительно задокументирован, что позволяет узнать, какие параметры передаются методу и какие значения он должен вернуть.

После создания классов по шаблонам необходимо зарегистрировать их в системе. Для регистрации используются соответствующие методы, которые принимают список из пар вида (класс, имя). Здесь класс – объект класса, унаследованный от шаблона, имя – строковый идентификатор этого класса в системе. Причём класс не обязательно должен быть унаследован от шаблона, но обязательно должен иметь необходимые методы, возвращающие необходимые значения. Имя в свою очередь должно быть уникальным, по этому имени будет происходить выбор компоненты в выпадающем списке в окне системы.

## 4.7 Выводы

Разработанная система удобна и проста в использовании для тестирования различных частей алгоритмов и самих алгоритмов в целом, в частности были проведены тесты на алгоритмах, описанных в Главах 2, 3. Система позволяет пользователям легко сравнивать производительность алгоритмов на одном и том же наборе данных, что способствует более удобному выбору эффективного решения для конкретных задач.

# ЗАКЛЮЧЕНИЕ

В конце этой работы можно заключить, что созданная мной система показала себя достойно. Во время написания работы я смог побывать и на месте разработчика системы распознавания автомобильных номеров, и на месте тестировщика этой системы, и на месте разработчика системы анализа и выбора алгоритмов, и на месте её пользователя. Из полученного опыта могу заключить, что работа с системой была более удобна, чем работа с каждой отдельной компонентой. Так, например, если бы появилась необходимость проверить большее количество компонент или какая-либо из компонент появилась бы позже остальных, то, используя разработанную мной систему, всё тестирование было бы сведено к созданию простого класса по шаблону и регистрации его в программе, в то время как без системы подобная ситуация потребовала бы большого количества дополнительной рутинной работы.

В процессе разработки системы для тестирования и выбора алгоритмов распознавания номеров автомобилей была проведена тщательная оценка различных подходов и технологий, необходимых для распознавания автомобильных номеров. Результаты исследования показали, что выбор алгоритма должен основываться на нескольких ключевых факторах, включая точность распознавания, скорость обработки, устойчивость к различным условиям освещения и качеству изображений, возможность работы с разными языками и шрифтами, а также саму задачу с её особенностями.

Среди множества имеющихся средств для тестирования алгоритмов данную систему можно выделить своей возможностью комплексного тестирования, позволяющего быстро и удобно протестировать как конкретную компоненту алгоритма, так и весь остальной алгоритм. Также эта система имеет огромный потенциал и широкие возможности для расширения своего функционала, вплоть до перехода от тестирования алгоритмов распознавания автомобильных номеров к любым задачам распознавания. Каждая из описанных систем может быть развита во что-то большее и имеющее более широкую область применения.

В результате анализа, проведенного в рамках данной работы, была выявлена оптимальная комбинация разработанных в ходе исследования задачи алгоритмов, обеспечивающая высокую точность распознавания номеров автомобилей в заданных условиях. Это может значительно улучшить эффективность систем видеонаблюдения и автоматизации, применяемых в области дорожного движения, парковки и безопасности.

Дальнейшие исследования могут быть направлены на расширение функционала системы, её обобщение для работы с большим спектром систем. Также возможны исследования направленные на улучшение гибкости системы, а также на расширение системы для интеграции с другими компонентами, такими как базы данных для хранения и анализа информации о транспортных средствах, что позволило бы разделить систему и данные, на которых она была обучена, и проверить её в совершено других обстоятельствах. Разработка такой системы открывает новые перспективы для применения технологий распознавания в реальных условиях, что в свою очередь может способствовать повышению безопасности на дорогах и улучшению городской инфраструктуры.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Weinman, J. J. Scene text recognition using similarity and a lexicon with sparse belief propagation / J. J. Weinman, E. Learned-Miller, A. R. Hanson // IEEE Trans. on Pattern Anal. and Mach. Intell. – 2009. – Vol. 31, № 10. – P. 1733–1746.
2. Хайкин С. Нейронные сети: полный курс, 2-е издание; Пер. с англ. – М. Издательский дом «Вильямс», 2006.
3. Redmon, J. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon, S. Divvala, R. Girshick, A. Farhadi // Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2016. – P. 779–788.
4. Ren, S. Faster R-CNN: Towards real-time object detection with region proposal networks / S. Ren, K. He, R. Girshick, J. Sun // IEEE Trans. on Pattern Anal. and Mach. Intell. – 2017. – Vol. 39, № 6. – P. 1137–1149.
5. Li, H. Reading car license plates using deep convolutional neural networks and LSTMs / H. Li, C. Shen. – [Б. м.] : arXiv, 2016. – 10 с. – (Препринт / arXiv ; 1601.05610).
6. Laroca, R. A robust real-time automatic license plate recognition based on the YOLO detector / R. Laroca, E. Severo, L. A. Zanlorensi [и др.]. – [Б. м.] : arXiv, 2018. – 10 с. – (Препринт / arXiv ; 1802.09567).
7. Martinsky O. Algorithmic and mathematical principles of automatic number plate recognition systems / O. Martinsky BRNO : BRNO university of technology, 2007 – 83 p.
8. Свирин И., Ханин А. Некоторые аспекты автоматического распознавания автомобильных номеров // Алгоритмы безопасности, 2010, №3. С. 26-29.
9. Huang, J. Speed/accuracy trade-offs for modern convolutional object detectors / J. Huang, V. Rathod, C. Sun [и др.] // Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2017. – P. 3296–3297.
10. Zhang Y., Zhang C.: New Algorithm for Character Segmentation of License Plate, Intelligent Vehicles Symposium, IEEE, 2003
11. Li, H. Towards end-to-end car license plates detection and recognition with deep neural networks / H. Li, P. Wang, C. Shen. – [Б. м.] : arXiv, 2017. – 12 с. – (Препринт / arXiv ; 1709.08828).
12. Гончаров, А. И. Методы и средства цифровой обработки изображений : учеб. пособие / А. И. Гончаров. – М. : Радио и связь, 2019. – 352 с.

## ПРИЛОЖЕНИЕ А

# КЛАССЫ-ШАБЛОНЫ ДЛЯ РАБОТЫ С СИСТЕМОЙ

Шаблон класса для загрузки

```
class ImgLoader(ABC):
    @abstractmethod
    def load_imgs(self, img_paths: List[str]):
        """
            Метод для загрузки изображений из директории.
            :returns Список изображений размера 416x416.
            :param img_paths: Путь к изображениям с расширением: '.jpg', '.jpeg', '.png', '.bmp'.
        """
        pass

    @abstractmethod
    def load_img(self, img_path):
        """
            Метод для загрузки изображения.
            :returns Изображение размера 416x416.
            :param img_path: Путь к изображению с расширением: '.jpg', '.jpeg', '.png', '.bmp'.
        """
        pass
```

Шаблон класса для обнаружения

```
class ImgPredictor(ABC):
    @abstractmethod
    def predict(self, img):
        """
            Метод для обнаружения номера на изображении.
            :returns Список ограничивающих рамок в формате xyxy, список точностей предсказаний.
            :param img: Рабочее изображение.
        """
        pass
```

Шаблон класса для нормализации

```
class ImgNormalizer(ABC):
    @abstractmethod
    def normalize(self, image, boxes, scores):
        """
            Метод для приведения изображений к виду, удобному для чтения.
            :returns Преобразованное изображение.
            :param image: Рабочее изображение.
            :param boxes: Ограничивающие рамки в формате xyxy.
            :param scores: Список точностей предсказаний.
        """
        pass
```

Шаблон класса для распознавания

```
class ImgOCR(ABC):
    @abstractmethod
    def read(self, imgs):
        """
            Метод для чтения номера с изображений.
            :returns Список номеров.
            :param imgs: Список рабочих изображений.
        """
        pass
```

## ПРИЛОЖЕНИЕ Б

# ЛИСТИНГ КОДА СИСТЕМЫ

```
import os
import shutil
import time
import tkinter as tk
from tkinter import ttk, filedialog
from typing import List, Tuple

import cv2
import numpy as np
import pandas as pd
from PIL import Image, ImageTk

from Img_Loader import ImgLoader
from Img_Normalizer import ImgNormalizer
from Img_OCR import ImgOCR
from Img_Predictor import ImgPredictor


class AlgorithmTester:
    def __init__(self):
        self.__clear_button = None
        self.__master = tk.Tk()
        self.__master.title("Распознавание номерных знаков")
        self.__widgets = []
        self.currentImg = None
        self.currentImgPath = None

        self.__loaders = {}
        self.__predictors = {}
        self.__normalizers = {}
        self.__recognizers = {}
        self.__loader:ImgLoader = None
        self.__predictor:ImgPredictor = None
        self.__normalizer:ImgNormalizer = None
        self.__recognizer:ImgOCR = None
```

```
        self.__master.geometry("400x200+600+100")
        self.__menu_bar = tk.Menu(self.__master)
        mode_menu = tk.Menu(self.__menu_bar, tearoff=0)
        mode_menu.add_command(label="Switch mode", command=self.__switch_mode)
        self.__menu_bar.add_cascade(label="Mode", menu=mode_menu)
        self.__master.config(menu=self.__menu_bar)
        self.__mode = 'tester'

        self.__loader_menu = ttk.Combobox(self.__master, width=30,
                                          values=[])
        self.__loader_menu.set('Choose image loader')
        self.__loader_menu.grid(row=0, column=0)

        self.__predictor_menu = ttk.Combobox(self.__master, width=30,
                                             values=[])
        self.__predictor_menu.set('Choose image predictor')
        self.__predictor_menu.grid(row=1, column=0)

        self.__normalizer_menu = ttk.Combobox(self.__master, width=30,
                                              values=[])
        self.__normalizer_menu.set('Choose image normalizer')
        self.__normalizer_menu.grid(row=2, column=0)

        self.__recognizer_menu = ttk.Combobox(self.__master, width=30,
                                              values=[])
        self.__recognizer_menu.set('Choose image recognizer')
        self.__recognizer_menu.grid(row=3, column=0)

        self.__process_button = tk.Button(self.__master, text="Test", command=self.__test)
        self.__process_button.grid(row=4, column=0)

    def registerImgLoaders(self, loaders: List[Tuple[ImgLoader, str]]):
        for l, n in loaders:
            self.__loaders[n] = l
            self.__loader_menu['values'] = (*self.__loader_menu['values'], n)

    def registerImgPredictors(self, predictors: List[Tuple[ImgPredictor, str]]):
        for p, n in predictors:
```

```
            self.__predictors[n] = p
            self.__predictor_menu['values'] = (*self.__predictor_menu['values'], n)

    def registerImgNormalizers(self, normalizers: List[Tuple[ImgNormalizer, str]]):
        for no, n in normalizers:
            self.__normalizers[n] = no
            self.__normalizer_menu['values'] = (*self.__normalizer_menu['values'], n)

    def registerImgOCRs(self, ocrs: List[Tuple[ImgOCR, str]]):
        for o, n in ocrs:
            self.__recognizers[n] = o
            self.__recognizer_menu['values'] = (*self.__recognizer_menu['values'], n)

    def __compose_algo(self):
        self.__loader = self.__loaders[self.__loader_menu.get()]
        self.__predictor = self.__predictors[self.__predictor_menu.get()]
        self.__normalizer = self.__normalizers[self.__normalizer_menu.get()]
        self.__recognizer = self.__recognizers[self.__recognizer_menu.get()]

    def __test(self):
        self.__compose_algo()
        self.__clear()
        clear_button = tk.Button(self.__master, text="Clear", com-
mand=self.__tester_clear)
        clear_button.grid(row=4, column=1)
        self.__widgets.append(clear_button)
        self.__master.geometry("500x655")
        if self.currentImg is None:
            self.currentImgPath = filedialog.askopenfilename()
        tm = time.time()
        self.currentImg = self.__loader.load_img(self.currentImgPath)
        tm_load = (time.time() - tm) * 1000
        self.__show_text(f'{tm_load:.2f} ms', 0, 1)
        if self.currentImg is not None:
            tm = time.time()
            boxes, scores = self.__predictor.predict(self.currentImg)
            tm_pred = (time.time() - tm) * 1000
            self.__show_image(self.__draw_box(self.currentImg.copy(), boxes,
scores), 5, 0)
            self.__show_text(f'{tm_pred:.2f} ms', 1, 1)
```

```
            tm = time.time()
            normalized_img = self.__normalizer.normalize(self.currentImg, boxes,
scores)
            tm_norm = (time.time() - tm) * 1000
            self.__show_image(normalized_img, 6, 0)
            self.__show_text(f'{tm_norm:.2f} ms', 2, 1)

            tm = time.time()
            number = self.__recognizer.read(normalized_img)
            tm_read = (time.time() - tm) * 1000
            self.__show_text(f'Number: {number}', 7, 0)
            self.__show_text(f'{tm_read:.2f} ms', 3, 1)

    def __draw_box(self, img, boxes, scores):
        max_score_index = np.argmax(scores.numpy())
        x1, y1, x2, y2 = boxes[max_score_index].numpy().astype(int)
        return cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

    def __show_image(self, img, row, column):
        image_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        img = Image.fromarray(image_bgr)
        img_tk = ImageTk.PhotoImage(img)
        label = tk.Label(self.__master)
        label.config(image=img_tk)
        label.image = img_tk
        label.grid(row=row, column=column)
        self.__widgets.append(label)

    def __show_text(self, txt, row, column):
        text = tk.Label(self.__master, text=txt)
        text.grid(row=row, column=column)
        self.__widgets.append(text)

    def __clear(self):
        for w in self.__widgets:
            w.destroy()

    def __tester_clear(self):
        self.__clear()
```

```
        self.currentImg = None
        self.currentImgPath = None
        self.__master.geometry("400x200")

    def __metrics_clear(self):
        self.__clear()
        self.__dark = tk.BooleanVar()
        self.__light = tk.BooleanVar()
        self.__contrast = tk.BooleanVar()
        self.__noise = tk.BooleanVar()
        self.__shift = tk.BooleanVar()

        self.__dark_option = tk.Checkbutton(self.__master, text="dark", variable=self.__dark)
        self.__dark_option.grid(row=5, column=0)
        self.__widgets.append(self.__dark_option)
        self.__light_option = tk.Checkbutton(self.__master, text="light", variable=self.__light)
        self.__light_option.grid(row=6, column=0)
        self.__widgets.append(self.__light_option)
        self.__contrast_option = tk.Checkbutton(self.__master, text="contrast", variable=self.__contrast)
        self.__contrast_option.grid(row=7, column=0)
        self.__widgets.append(self.__contrast_option)
        self.__noise_option = tk.Checkbutton(self.__master, text="noise", variable=self.__noise)
        self.__noise_option.grid(row=8, column=0)
        self.__widgets.append(self.__noise_option)
        self.__shift_option = tk.Checkbutton(self.__master, text="shift", variable=self.__shift)
        self.__shift_option.grid(row=9, column=0)
        self.__widgets.append(self.__shift_option)

    def start(self):
        self.__master.mainloop()

    def __switch_mode(self):
        if self.__mode == 'tester':
            self.__enter_metrics_mode()
        elif self.__mode == 'metrics':
```

```
            self.__enter_tester_mode()

    def __enter_tester_mode(self):
        self.__clear()
        self.__mode = 'tester'
        self.currentImg = None
        self.currentImgPath = None
        self.__master.geometry("400x200")
        self.__process_button.config(command=self.__test)

    def __enter_metrics_mode(self):
        self.__clear()
        self.__mode = 'metrics'
        self.currentImg = None
        self.currentImgPath = None
        self.__master.geometry("500x655")
        self.__process_button.config(command=self.__calculate_metrics)
        self.__dark = tk.BooleanVar()
        self.__light = tk.BooleanVar()
        self.__contrast = tk.BooleanVar()
        self.__noise = tk.BooleanVar()
        self.__shift = tk.BooleanVar()

        self.__dark_option = tk.Checkbutton(self.__master, text="dark", variable=self.__dark)
        self.__dark_option.grid(row=5, column=0)
        self.__widgets.append(self.__dark_option)
        self.__light_option = tk.Checkbutton(self.__master, text="light", variable=self.__light)
        self.__light_option.grid(row=6, column=0)
        self.__widgets.append(self.__light_option)
        self.__contrast_option = tk.Checkbutton(self.__master, text="contrast", variable=self.__contrast)
        self.__contrast_option.grid(row=7, column=0)
        self.__widgets.append(self.__contrast_option)
        self.__noise_option = tk.Checkbutton(self.__master, text="noise", variable=self.__noise)
        self.__noise_option.grid(row=8, column=0)
        self.__widgets.append(self.__noise_option)
```

```
        self.__shift_option = tk.Checkbutton(self.__master, text="shift", variable=self.__shift)
        self.__shift_option.grid(row=9, column=0)
        self.__widgets.append(self.__shift_option)

    def __calculate_metrics(self):
        self.__create_testing_repository()
        load_res, pred_res, norm_res, ocr_res, best = self.__do_calculate_metrics()
        self.__clear()
        return_button = tk.Button(self.__master, text="Return", command=self.__metrics_clear)
        return_button.grid(row=4, column=1)
        self.__widgets.append(return_button)
        self.__show_text(f'Best: {best[0]} -> {best[1]} -> {best[2]} -> {best[3]}', 5, 0)
        metrics_button = tk.Button(self.__master, text="Show results",
                                   command=lambda: self.__show_metrics(load_res, pred_res, norm_res, ocr_res))
        metrics_button.grid(row=6, column=0)
        self.__widgets.append(metrics_button)

    def __show_metrics(self, load_res, pred_res, norm_res, ocr_res):
        root = tk.Toplevel(self.__master)
        root.title("Метрики")
        root.geometry("500x200")
        buttons = tk.Frame(root)
        buttons.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        tree = tk.Frame(root)
        tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        res_view = ttk.Treeview(tree, show='headings')
        load_res_button = tk.Button(buttons, text="Loaders",
                                    command=lambda: self.__fill_tree(res_view, load_res))
        load_res_button.grid(row=0)
        pred_res_button = tk.Button(buttons, text="Predictors",
                                    command=lambda: self.__fill_tree(res_view, pred_res))
        pred_res_button.grid(row=1)
        norm_res_button = tk.Button(buttons, text="Normalizers",
                                    command=lambda: self.__fill_tree(res_view, norm_res))
        norm_res_button.grid(row=2)
        ocr_res_button = tk.Button(buttons, text="Recognizers",
```

```
                                command=lambda: self.__fill_tree(res_view, ocr_res))
        ocr_res_button.grid(row=3)
        res_view.pack(expand=True)

    def __fill_tree(self, tree, df):
        for item in tree.get_children():
            tree.delete(item)
        tree["columns"] = list(df.columns)
        for col in df.columns:
            tree.heading(col, text=col)

        for i in range(len(df)):
            tree.insert("", "end", values=list(df.iloc[i]))

        for col in df.columns:
            max_width = max(len(col), max(len(str(item)) for item in df[col]))
            tree.column(col, width=max_width*8)
        return tree

    def __create_testing_repository(self):
        dataPath = filedialog.askdirectory()
        repositoryPath = './data/test_repository'
        if os.path.exists(repositoryPath):
            shutil.rmtree(repositoryPath)
        os.makedirs(repositoryPath)
        if self.__dark.get(): dark_options = [1, 0.9, 0.8, 0.7]
        else: dark_options = []
        if self.__light.get(): light_options = [1, 1.1, 1.2, 1.3]
        else: light_options = []
        if self.__contrast.get(): contrast_options = [0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3]
        else: contrast_options = []
        if self.__noise.get(): noise_options = [0, 0.05, 0.075, 0.1]
        else: noise_options = []
        if self.__shift.get(): shift_options = [-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3]
        else: shift_options = []

        image_files = [f for f in os.listdir(dataPath) if f.endswith(('jpg', 'jpeg', 'png',
'bmp'))]

        for image_file in image_files:
```

```
            image_path = os.path.join(dataPath, image_file)
            image = cv2.imread(image_path)
            temp_list = [image]

            if image is not None:
                temp_list = self.__do_dark(temp_list, dark_options)
                temp_list = self.__do_light(temp_list, light_options)
                temp_list = self.__do_contrast(temp_list, contrast_options)
                temp_list = self.__do_noise(temp_list, noise_options)
                temp_list = self.__do_shift(temp_list, shift_options)

                base_name, ext = os.path.splitext(image_file)
                for idx, image in enumerate(temp_list):
                    output_file_name = f"{base_name}{'!!'}{idx}{ext}"
                    output_path = os.path.join(repositoryPath, output_file_name)
                    cv2.imwrite(output_path, image)

                file_name = f'{base_name}.txt'
                shutil.copy(os.path.join(dataPath,                                file_name),
os.path.join(repositoryPath, file_name))

    def __do_dark(self, imgs, dark_options):
        if not dark_options:
            return imgs
        darkened_images = []

        for image in imgs:
            for option in dark_options:
                darkened_image = np.clip(image * option, 0, 255).astype(np.uint8)
                darkened_images.append(darkened_image)

        return darkened_images

    def __do_light(self, imgs, light_options):
        if not light_options:
            return imgs
        lightened_images = []

        for image in imgs:
            for option in light_options:
```

```
            lightened_image = np.clip(image * option, 0, 255).astype(np.uint8)
            lightened_images.append(lightened_image)

    return lightened_images

def __do_contrast(self, imgs, contrast_options):
    if not contrast_options:
        return imgs
    contrast_images = []

    for image in imgs:
        for option in contrast_options:
            contrast_image = cv2.convertScaleAbs(image, alpha=option, beta=0)
            contrast_images.append(contrast_image)

    return contrast_images

def __do_noise(self, imgs, noise_options):
    if not noise_options:
        return imgs
    noise_images = []

    for image in imgs:
        for option in noise_options:
            noise = np.random.randn(*image.shape) * option
            noise_image = np.clip(image + noise * 255, 0, 255).astype(np.uint8)
            noise_images.append(noise_image)

    return noise_images

def __do_shift(self, imgs, shift_options):
    if not shift_options:
        return imgs
    shift_images = []

    for image in imgs:
        for option in shift_options:
            shift_image = image.astype(np.float32)
            shift_value = option * shift_image
            shift_image += shift_value
```

```
                shift_image = np.clip(shift_image, 0, 255).astype(np.uint8)
                shift_images.append(shift_image)

        return shift_images

    def __do_calculate_metrics(self):
        repositoryPath = './data/test_repository'
        names = []
        imgs = []
        boxes = []
        labels = []
        bd = {}
        load_res = pd.DataFrame(columns=['Name', 'Mean time'])
        pred_res = pd.DataFrame(columns=['Name', 'Mean time', 'iou75', 'iou90', 'iou95'])
        norm_res = pd.DataFrame(columns=['Name', 'Mean time', 'Readability'])
        ocr_res = pd.DataFrame(columns=['Name', 'Mean time', 'Perfect match rate', 'Character error rate'])
        for name, loader in self.__loaders.items():
            nms, temp, mean_tm = self.__test_loader(loader, repositoryPath)
            load_res.loc[len(load_res)] = [name, mean_tm]
            if not imgs:
                imgs = temp
            if not names:
                names = nms

        info_files = [f for f in os.listdir(repositoryPath) if f.endswith('.txt')]

        for file_name in info_files:
            file_path = os.path.join(repositoryPath, file_name)
            with open(file_path, 'r') as file:
                line = file.readline().strip()
                parts = line.split(' ')
                number = int(parts[0])
                bbox = list(map(lambda x: x * 416, map(float, parts[1:5])))
                label = ' '.join(parts[5:])
                fn, _ = os.path.splitext(file_name)
                bd[fn] = (number, bbox, label)

        for name in names:
```

```
            _, bbox, label = bd[name]
            boxes.append(bbox)
            labels.append(label)

        bboxes = []
        scores = []
        mean_iou = 0
        for name, predictor in self.__predictors.items():
            bb, sc, mean_tm, iou75, iou90, iou95 = self.__test_predictor(imgs, boxes,
predictor)
            pred_res.loc[len(pred_res)] = [name, mean_tm, iou75, iou90, iou95]
            if sum([iou75, iou90, iou95]) / len([iou75, iou90, iou95]) > mean_iou:
                mean_iou = sum([iou75, iou90, iou95]) / len([iou75, iou90, iou95])
                bboxes = bb
                scores = sc

        rois = []
        for norm_name, normalizer in self.__normalizers.items():
            readability = 0.0
            norm_tm = 0.0
            temp, mean_tm = self.__test_normalizer(imgs, bboxes, scores, normalizer)
            rois = temp
            for ocr_name, recognizer in self.__recognizers.items():
                mean_tm, perfect_match_rate, average_cer = self.__test_ocr(rois, labels,
recognizer)
                readability += (1 - average_cer)
                norm_tm += mean_tm
                ocr_res.loc[len(ocr_res)] = [ocr_name, mean_tm, perfect_match_rate,
average_cer]
            norm_res.loc[len(norm_res)] = [norm_name,
                                           norm_tm / (len(self.__recognizers) ),
                                           readability / (len(self.__recognizers))]
            ocr_res = ocr_res.groupby('Name').mean().reset_index()

        load_best = load_res.loc[load_res['Mean time'].idxmin(), 'Name']
        pred_best = pred_res.loc[(pred_res[['iou75', 'iou90',
'iou95']].mean(axis=1)).idxmax(), 'Name']
        norm_best = norm_res.loc[norm_res['Readability'].idxmax(), 'Name']
        ocr_best = ocr_res.loc[ocr_res['Character error rate'].idxmin(), 'Name']
```

```
        return (load_res, pred_res, norm_res, ocr_res,
                (load_best, pred_best, norm_best, ocr_best))

    def __test_loader(self, loader:ImgLoader, imgs_path):
        image_files = [f for f in os.listdir(imgs_path) if f.endswith(('jpg', 'jpeg', 'png',
'bmp'))]
        img_paths = []
        names = []
        for image_file in image_files:
            img_paths.append(os.path.join(imgs_path, image_file))
            names.append(image_file.split('!!')[0])

        tm = time.time()
        imgs = loader.load_imgs(img_paths)
        mean_tm = (time.time() - tm) * 1000 / len(imgs)
        return names, imgs, mean_tm

    def __test_predictor(self, imgs, true_boxes, predictor:ImgPredictor):
        tm = time.time()
        boxes = []
        scores  = []
        for img in imgs:
            box, score = predictor.predict(img)
            boxes.append(box)
            scores.append(score)
        mean_tm = ((time.time() - tm) * 1000) / len(imgs)

        iou = []
        iou75 = 0
        iou90 = 0
        iou95 = 0
        for bboxes, score, img, true_box in zip(boxes, scores , imgs, true_boxes):
            if len(bboxes) == 0:
                iou.append(0)
            else:
                max_score_index = np.argmax(score.numpy())
                bbox = bboxes[max_score_index].numpy().astype(int)
                iou.append(self.__calculate_iou(bbox, true_box))
        iou75_count = sum(i > 0.75 for i in iou)
        iou75 = (iou75_count / len(iou)) * 100 if len(iou) > 0 else 0
```

```
    iou90_count = sum(i > 0.90 for i in iou)
    iou90 = (iou90_count / len(iou)) * 100 if len(iou) > 0 else 0
    iou95_count = sum(i > 0.95 for i in iou)
    iou95 = (iou95_count / len(iou)) * 100 if len(iou) > 0 else 0

    return boxes, scores , mean_tm, iou75, iou90, iou95

def __calculate_iou(self, box1, box2):
    # box: [x1, y1, x2, y2]
    x1_inter = max(box1[0], box2[0])
    y1_inter = max(box1[1], box2[1])
    x2_inter = min(box1[2], box2[2])
    y2_inter = min(box1[3], box2[3])

    intersection_area = max(0, x2_inter - x1_inter) * max(0, y2_inter - y1_inter)

    area_box1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area_box2 = (box2[2] - box2[0]) * (box2[3] - box2[1])

    union_area = area_box1 + area_box2 - intersection_area

    return intersection_area / union_area if union_area > 0 else 0

def __test_normalizer(self, imgs, bboxes, bscores, normalizer:ImgNormalizer):
    tm = time.time()
    lst = []
    for img, boxes, scores in zip(imgs, bboxes, bscores):
        lst.append(normalizer.normalize(img, boxes, scores))
    mean_tm = (time.time() - tm) * 1000 / len(imgs)
    return lst, mean_tm

def __test_ocr(self, imgs, labels, ocr:ImgOCR):
    tm = time.time()
    lst = []
    for img in imgs:
        lst.append(ocr.read(img))
    mean_tm = (time.time() - tm) * 1000 / len(imgs)

    total_predictions = len(lst)
    perfect_matches = 0
```

```
        total_lev_distance = 0
        for predicted, reference in zip(lst, labels):
            if predicted == reference:
                perfect_matches += 1

            lev_distance = self.__lev_distance(reference, predicted)
            total_lev_distance += lev_distance

        perfect_match_rate = (perfect_matches / total_predictions) * 100 if total_predictions > 0 else 0
        average_cer = total_lev_distance / sum(len(ref) for ref in labels) if labels else 0

        return mean_tm, perfect_match_rate, average_cer

    def __lev_distance(self, s1, s2):
        n, m = len(s1), len(s2)
        if n > m:
            s1, s2 = s2, s1
            n, m = m, n

        current_row = range(n + 1)
        for i in range(1, m + 1):
            previous_row, current_row = current_row, [i] + [0] * n
            for j in range(1, n + 1):
                add, delete, change = previous_row[j] + 1, current_row[j - 1] + 1, previous_row[j - 1]
                if s1[j - 1] != s2[i - 1]:
                    change += 1
                current_row[j] = min(add, delete, change)

        return current_row[n]
```