**Содержание**

[Введение 8](#_Toc195472574)

[1 Аналитический обзор 9](#_Toc195472575)

[1.1 Цель исследования и задачи 9](#_Toc195472576)

[1.2 Обзор задачи классификации изображения 9](#_Toc195472577)

[1.3 Обзор методов машинного обучения для решения задачи детекции на изображении 11](#_Toc195472578)

[1.4 Методы на основе пиксельных классификаторов 12](#_Toc195472579)

[1.5 Методы обнаружения ключевых точек 13](#_Toc195472580)

[1.6 Метод Template matching 13](#_Toc195472581)

[1.7 Вывод по главе 13](#_Toc195472582)

[2 Описание модели классификации изображений 15](#_Toc195472583)

[2.1 Модель Roboflow Object Detection 3.0 15](#_Toc195472584)

[2.2 Архитектура 17](#_Toc195472585)

[2.3 Алгоритм применения модели Roboflow Object Detection 3.0 для задачи обнаружения объектов 18](#_Toc195472586)

[2.4 Вывод по главе 19](#_Toc195472587)

[3 Программное конструирование 20](#_Toc195472588)

[3.1 Обоснование выбора средств разработки 20](#_Toc195472589)

[3.2 Описание работы основных методов программы 22](#_Toc195472590)

[3.3 Пример работы программы 23](#_Toc195472591)

[3.3 Вывод по главе 26](#_Toc195472592)

[4 Тестирование программного средства 27](#_Toc195472593)

[4.1 Описание процесса тестирования обученной модели 27](#_Toc195472594)

[4.2 Описание процесса тестирования интерфейса 31](#_Toc195472595)

[4.3 Выводы по главе 34](#_Toc195472596)

[Заключение 35](#_Toc195472597)

[Cписок использованных источников 36](#_Toc195472598)

[Приложение Б. Исходный код программного средства 37](#_Toc195472599)

Введение

Распознавание растений на поле во время уборки урожая — это важный аспект современного сельского хозяйства. Точное определение видов культур, сорняков и их состояния позволяет оптимизировать процессы сбора урожая, минимизировать потери и повысить эффективность сельскохозяйственных работ. Понимание и автоматизация этого процесса с помощью технологий искусственного интеллекта (ИИ) способствует увеличению урожайности, снижению затрат и улучшению экологической устойчивости агропромышленного комплекса. В настоящее время технологии ИИ активно внедряются в сельское хозяйство, предлагая инновационные решения для анализа и мониторинга растительных культур. Они позволяют автоматизировать процессы, которые раньше требовали значительных временных и трудовых ресурсов.

Сельскохозяйственные угодья часто сталкиваются с такими проблемами, как смешение культур и сорняков, неравномерное созревание растений, поражение болезнями или вредителями. Применение искусственного интеллекта для распознавания растений даёт возможность решить эти проблемы за счёт автоматизированного анализа полей в реальном времени. Современные технологии ИИ, включая машинное обучение и компьютерное зрение, позволяют создавать системы для анализа изображений и видео с сельскохозяйственных полей. Эти системы обучаются на больших наборах данных, содержащих изображения различных культур, сорняков и их состояний в разных условиях. Анализируя такие данные, ИИ может классифицировать растения, определять их границы, выявлять болезни, оценивать зрелость и даже прогнозировать урожайность.

Целью данного проекта является разработка и реализация модели глубокого обучения для распознавания растений на поле во время уборки урожая.

## 1 Аналитический обзор

В данной главе рассматриваются цели исследования, задачи классификации изображений, методы машинного обучения для сегментации изображений.

### 1.1 Цель исследования и задачи

Ниже рассмотрены основные цели исследования:

1. реализация модели для распознавания растений на поле;
2. создание приложения с графическим интерфейсом с возможностями проводить детекцию растений на отдельном изображении, на каждом изображении в датасете, а также генерировать информацию о предсказанных боксах модели в формате JSON.

В соответствии с целью появляются следующие задачи:

1. сформировать обучаемый датасет, состоящий из множества изображений поля с растениями;
2. провести обучение на выбранном датасете модели Roboflow Object Detection 3.0 (Accurate);
3. провести тестирование обученной модели;
4. разработать программный модуль для взаимодействия с моделью через Roboflow API;
5. разработать графический интерфейс для взаимодействия с моделью;
6. протестировать работу программного средства.

### 1.2 Обзор задачи классификации изображения

Классификация изображений – это практическая задача, решаемая при помощи технологий искусственного интеллекта, заключающаяся в разделении изображения на классы присутствующих на нем объектов. Задача классификации подразделяется на сегментацию и детекцию.

Сегментация (Segmentation) – точное разделение изображения на семантически значимые области с присвоением каждому пикселю метки класса.

Детекция (Object Detection) – обнаружение объектов на изображении с указанием их местоположения (ограничивающие рамки, bounding boxes) и классификация.

Для поиска необходимых растений на поле требуется решить задачу детекции. Ниже рассмотрена подробная классификация типов данной задачи:

1. Биомедицинская детекция. В медицинской сфере детекция позволяет обнаруживать органы и патологические изменения на снимках, выявлять аномальные клеточные структуры, а также локализовывать опухоли и очаги воспаления для последующей диагностики.
2. Сельскохозяйственная детекция. В агропромышленном секторе детекция применяется для обнаружения и классификации культурных растений, выявления вредителей и болезней сельхозкультур, а также для определения степени зрелости плодов.
3. Транспортная детекция. В сфере транспорта и автономного вождения детекция используется для распознавания транспортных средств и участников движения, идентификации дорожных знаков и светофоров, а также для выявления потенциальных препятствий на пути движения.
4. Геопространственная детекция. При работе с картографическими данными детекция помогает обнаруживать различные объекты на спутниковых снимках, выявлять здания и сооружения, а также фиксировать изменения ландшафта с течением времени.
5. Промышленная детекция. В производственной сфере детекция обеспечивает контроль качества выпускаемой продукции, позволяет выявлять производственные дефекты и осуществлять мониторинг состояния оборудования.
6. Экологический мониторинг. В экологической сфере детекция помогает выявлять зоны загрязнения, обнаруживать важные природные объекты на изображениях и видео, а также проводить анализ изменений окружающей среды. [1][2]

Существующие методы не во всех случаях могут быть корректны, зачастую детекция получается неполной, либо неточной. Неточности детекции могут сделать задачу, например, для обработки медицинских изображений некорректной, поэтому необходимо улучшать существующие методы.

### 1.3 Обзор методов машинного обучения для решения задачи детекции на изображении

Методы машинного обучения для детекции объектов включают в себя различные подходы.

Методы на основе скользящего окна анализируют изображение, последовательно проверяя различные области на наличие объектов.

Обучение с учителем реализуется через архитектуры CNN (такие как R-CNN, Fast R-CNN, Faster R-CNN и YOLO), где модель обучается предсказывать координаты ограничивающих рамок и классы объектов.

Методы без учителя могут включать кластеризацию признаков или анализ текстур для выделения потенциальных объектов. [4]

Современные подходы также используют трансформеры (DETR) и другие гибридные архитектуры. Каждый метод обладает своими сильными сторонами и ограничениями, а выбор оптимального подхода определяется спецификой задачи детекции, требованиями к скорости обработки и доступностью размеченных данных.

### 1.4 Методы на основе пиксельных классификаторов

Классификаторы объектов — это методы машинного обучения, которые определяют наличие и местоположение объектов на изображении путём анализа характерных признаков. Они составляют основу многих подходов к детекции объектов.

Ключевыми особенностями классификаторов объектов являются:

1. Локальный анализ. Обрабатывают отдельные регионы изображения с помощью скользящего окна.
2. Классификаторы использую признаки HOG (гистограммы ориентированных градиентов), цветовые гистограммы, текстуры, локальные бинарные шаблоны (LBP).

Основными алгоритмами, применяющимися в классификаторах объектов, являются следующие:

1. Метод опорных векторов (SVM). Строит разделяющие гиперплоскости для классификации регионов.
2. Каскадные классификаторы (например, Viola-Jones). Последовательно отсеивают фоновые области.
3. Бустинг деревьев решений. Улучшает качество детекции за счёт ансамблевого подхода.

Преимуществами классификаторов объектов является относительно простая реализация, хорошая производительность при детекции жёстко заданных объектов (лица, дорожные знаки), а также умеренные вычислительные затраты (по сравнению с глубокими нейронными сетями).

К недостаткам данных методов машинного обучения можно отнести чувствительность к изменению масштаба и ориентации объектов, необходимость тщательной настройки параметров сканирования, а также ограниченную точность при работе с разнообразными объектами.

Основными областями применения классификаторов объектов являются:

1. Детекция лиц в системах видеонаблюдения.
2. Распознавание дорожных знаков для ADAS-систем.
3. Обнаружение простых объектов в embedded-системах с ограниченными ресурсами.

### 1.5 Методы обнаружения ключевых точек

Обнаружение ключевых точек, также известное как локализация ключевых точек или обнаружение ориентиров, представляет собой задачу компьютерного зрения, которая включает в себя идентификацию и локализацию определенных точек интереса на изображении. В задачах компьютерного зрения ключевые точки представляют собой суставы человеческого тела, лицевые ориентиры или выступающие точки на объектах.

Подходы к обнаружению ключевых точек в объектах, основанные на глубоком обучении, используют сверточные нейронные сети (CNN), такие как предобученные модели Roboflow Object Detection 3.0 или YOLOv8.

### 1.6 Метод Template matching

Template matching — метод, основанный на нахождении места на изображении, наиболее похожем на шаблон. “Похожесть” изображения задается определенной метрикой. То есть, шаблон "накладывается" на изображение, и считается расхождение между изображением и шаблоном. Положение шаблона, при котором это расхождение будет минимальным, и будет означать место искомого объекта. [6]

### 1.7 Вывод по главе

В данной главе рассмотрена задача классификации изображений, классификация применений задачи детекции как подвида задачи классификации изображений, методы решения этой задачи. Рассмотрено, какие этапы включает процесс решения задачи детекции и какие методы для этого применяются.

Для решения задачи детекции требуется собрать необходимые данные, выбрать и обучить модель, а затем оценить ее работу. Особое внимание было уделено архитектуре нейронных сетей, таким как CNN, и их роли в классификации изображений.

Методы классификации изображений используются для распознавания лиц, диагностики заболеваний, автоматического аннотирования фотографий, обеспечения безопасности и в автономных транспортных системах, а также для задачи распознавания растений на поле в сельском хозяйстве.

Таким образом, задачи детекции и задачи классификации изображений в целом играют ключевую роль в развитии технологий машинного обучения и компьютерного зрения, открывая новые возможности для исследований и практических применений.

## 2 Описание модели классификации изображений

В данной главе обоснован выбор модели для инициализации изображений, её архитектура и алгоритм применения.

### 2.1 Модель Roboflow Object Detection 3.0

Roboflow 3.0 Object Detection (Accurate) — это современная модель для обнаружения объектов, разработанная на базе передовых технологий компьютерного зрения. Хотя она не является прямой реализацией популярной модели YOLOv8, в ней используются многие прогрессивные подходы, характерные для последних версий YOLO, а также собственные инновации платформы Roboflow.

Ключевые особенности Roboflow 3.0 Object Detection (Accurate):

1. Оптимизированная архитектура. Модель использует эффективные механизмы внимания для улучшенного выделения значимых областей изображения, комбинирует многоуровневые признаки для точного обнаружения объектов разных размеров.
2. Усовершенствованные методы обучения. Применяет расширенные техники аугментации данных для повышения устойчивости модели. Поддерживает обучение с mixed precision для ускорения процесса без потери точности.
3. Масштабируемость и адаптивность. Позволяет настраивать архитектуру под конкретные задачи детекции. Модель также оптимизирована для работы с различными типами входных данных.

Модель Roboflow 3.0 Object Detection (Accurate) имеет следующие преимущества:

1. Высокая точность обнаружения. Показывает конкурентные результаты на стандартных бенчмарках, эффективно работает с разнообразными классами объектов.
2. Баланс скорости и точности. Обеспечивает хорошую производительность в реальном времени. Оптимизирована для развертывания в production-средах.
3. Интеграция с экосистемой Roboflow. Поддерживает все возможности платформы Roboflow для обработки данных. Легко адаптируется под конкретную постановку задачи с помощью инструментов аннотирования.
4. Гибкость применения. Может использоваться как для стандартных, так и для специализированных задач детекции. Поддерживает transfer learning для быстрого адаптирования под новые домены.

Проводя сравнительный анализ отличий выбранной модели от распространенной сегодня модели YOLOv8, можно выделить следующие важные пункты:

1. Более специализирована на работе с данными, подготовленными в экосистеме Roboflow.
2. Имеет оптимизированную для промышленного применения архитектуру.
3. Предлагает уникальные инструменты постобработки результатов.

Данная модель особенно хорошо подходит для задач, где критически важны:

1. Точность детекции в сложных условиях.
2. Интеграция с существующими pipeline обработки изображений.
3. Быстрое прототипирование и внедрение решений.
4. Для достижения максимальной эффективности рекомендуется использовать все преимущества платформы Roboflow, включая инструменты аннотирования, аугментации и мониторинга качества модели.

### 2.2 Архитектура

Модель Roboflow Object Detection 3.0 (Accurate) представляет собой современную архитектуру для обнаружения объектов, оптимизированную для работы с данными, подготовленными в экосистеме Roboflow. Ее работа состоит из нескольких взаимосвязанных этапов, обеспечивающих высокую точность детекции.

Архитектура модели Roboflow Object Detection 3.0 (Accurate) основана на современном подходе к обнаружению объектов, сочетающем эффективность и высокую точность. В основе системы лежит усовершенствованная CNN-архитектура, выполняющая роль бэкбона. Эта базовая сеть специально оптимизирована для задач детекции и включает механизмы эффективного уменьшения разрешения (downsampling) с одновременным сохранением ключевых признаков. Особенностью архитектуры является поддержка multi-scale feature extraction, позволяющая модели эффективно работать с объектами различных размеров - от крупных элементов до мелких деталей.

Важнейшим компонентом системы выступает механизм агрегации признаков (neck), реализующий улучшенную версию Feature Pyramid Network (FPN). Этот модуль обеспечивает комплексный обмен информацией между разными уровнями абстракции, что значительно улучшает качество детекции. Дополнительные skip-connections в архитектуре помогают сохранять важную пространственную информацию на всех этапах обработки изображения.

Детекторная головка (head) модели представляет собой многоуровневую систему, способную обнаруживать объекты в различных масштабах. В ней реализован адаптивный механизм работы с anchor boxes, автоматически подстраивающийся под особенности конкретного набора данных. Особое внимание уделено оптимизированной функции потерь, которая обеспечивает точное позиционирование ограничивающих рамок (bbox) и минимизирует ошибки локализации. Совокупность этих технических решений позволяет модели достигать высокой точности детекции при работе с разнообразными типами объектов и сценариями.

Модель Roboflow Object Detection 3.0 (Accurate) использует сбалансированное обучение с оптимизированными гиперпараметрами, такими как скорость обучения и вес декай, для предотвращения переобучения и улучшения обобщающей способности. Включение постепенного увеличения разрешения (прогрессивного ресайза) на поздних этапах обучения помогает точнее детектировать мелкие объекты. Кроме того, ансамблирование предобученных моделей и применение фокус-лосса (focal loss) позволяют минимизировать влияние дисбаланса классов, что особенно важно в сложных сценариях с неравномерным распределением объектов. [8]

### 2.3 Алгоритм применения модели Roboflow Object Detection 3.0 для задачи обнаружения объектов

Общий алгоритм работы модели Roboflow Object Detection 3.0 можно описать следующим образом:

Шаг 1: Собрать набор данных для индивидуального обучения модели.

Шаг 2: Разметить данные с помощью платформы Roboflow (это платформа, предоставляющая инструменты для компьютерного зрения и машинного обучения).

Шаг 3: Обучить модель компьютерного зрения на пользовательских данных, при этом важно разделить данные на обучающий набор и тестовый набор. Обучающий набор используется для обучения модели тому, как делать прогнозы, а тестовый набор используется для оценки точности модели. Процент разделения (80–20%) является общепринятым, но точное соотношение может зависеть от размера набора данных и конкретной задачи, над которой идёт работа.

Шаг 4: После того, как выполнены шаги предварительной обработки, такие как сбор данных, маркировка данных, разделение данных и создание пользовательского файла конфигурации, можно начать обучение Roboflow Object Detection 3.0 на пользовательских данных.

После обучения модели её можно использовать для прогнозирования и сравнения с новыми данными.

### 2.4 Вывод по главе

В данной главе была описана модель Roboflow Object Detection 3.0 (Accurate), а также подробно рассмотрена её архитектура, подробный алгоритм применения, а также особенности обучения.

Проведена оценка преимуществ и недостатков модели Roboflow Object Detection 3.0. Модель обладает хорошей точностью детекции при правильно подобранном наборе изначальных данных, отлично взаимодействует с сервисом Roboflow, в котором реализован эффективный механизм разметки начального датасета, к ней прилагается удобный API для запуска ее работы.

## 3 Программное конструирование

В данной главе обоснован выбор средств разработки, а также описаны основные методы и процессы программного средства.

### 3.1 Обоснование выбора средств разработки

Для разработки программного средства для взаимодействия с обученной моделью нейронной сети, был выбран язык программирования Python.

Python – высокоуровневый интерпретируемый язык программирования общего назначения. Он обладает простым и понятным синтаксисом, что делает его доступным для новичков, но при этом обеспечивает мощные возможности для опытных разработчиков. Python широко используется в различных областях, таких как веб-разработка, анализ данных, искусственный интеллект, научные исследования, автоматизация задач и многое другое.

Преимущества Python:

1. Простота и читаемость кода. Python разработан с упором на читаемость кода, что делает его легким для изучения и понимания.
2. Большое количество библиотек и фреймворков. Python имеет огромное сообщество разработчиков, которые создали множество библиотек и фреймворков для решения различных задач. Например, для машинного обучения существуют библиотеки TensorFlow, PyTorch, scikit-learn; для веб-разработки — Django, Flask, FastAPI, Starlette; для научных вычислений — NumPy, SciPy, Sympy и многие другие.
3. Поддержка различных платформ. Python поддерживает различные операционные системы, включая Windows, macOS и Linux, что делает его универсальным инструментом для разработки кроссплатформенных приложений.
4. Активное сообщество. В Python активно развивается сообщество разработчиков, которые активно обсуждают и развивают язык, создают учебные материалы, библиотеки и фреймворки. [6]

Python является одним из самых популярных языков программирования в мире и продолжает набирать популярность благодаря своей простоте, мощным возможностям и широкому спектру применения.

В качестве библиотеки для создания графического клиентского интерфейса была выбрана Python Tkinter.

Библиотека Python Tkinter является простой в освоении, предоставляет все необходимые базовые инструменты для построения графического интерфейса, такие как задание размерностей, цвета, шрифты, создание мультиэкранного окна приложения. Также Python Tkinter дает возможность пользователю импортировать необходимые ему файлы и папки, что является ключевой возможностью разрабатываемого интерфейса в контексте импорта путей к конкретному изображению, датасету с рядом изображения, на которых затем осуществляется детекция, а также путей для сохранения сгенерированных программным средством результатов. [7]

Для обучения нейросети, способной распознать растения на поле, выбрана модель Roboflow Object Detection 3.0 – современная модель для обнаружения объектов от команды Roboflow. Для программного взаимодействия с обученной моделью предоставляется удобный API, напрямую взаимодействующий с сервисом и позволяющий разрабатывать пользовательский интерфейс с возможностями для генерации предсказаний, запуска детекции по конкретному изображению или по всему датасету целиком.  
 При выборе между двумя вариациями модели Roboflow Object Detection 3.0 – Fast и Accurate – был сделан выбор в пользу Accurate из соображений точности результатов детекции.

### 3.2 Описание работы основных методов программы

Для выполнения поставленной задачи был реализован программный модуль для взаимодействия с Roboflow API для запуска работы модели и проведения различных расчетов метрик качества работы модели, и клиентский интерфейс для взаимодействия с программным средством.

Основными классами программного модуля по взаимодействию с API являются:

1. Класс BoxParser для расчета координат боксов.
2. Тестовый класс BoxVisualizer для отрисовки боксов с помощью библиотеки компьютерного зрения OpenCV.
3. Класс DatasetFetch для получения данных ground-truth (ручной разметки боксов) c сервиса Roboflow.
4. Вспомогательный класс CheckImage для проверки корректной структуры изображения, предлагаемого для проведения детекции.
5. Класс Predict, содержащий различные методы для проведения предсказаний моделью, визуализации данных предсказаний и сохранения полученных результатов проведенной нейронной сетью детекции.

Ключевыми методами корневого модуля приложения являются get\_images\_json() класса DatasetFetch, получающий по запросу к API ground-truth информацию о размеченных боксах, методы класса Predict, такие как image\_predict\_json() для сохранения координат предсказанных боксов по изображению и image\_visualize\_prediction() для визуализации изображения с нанесёнными на него предсказанными боксами.

Разработанный клиетский интерфейс состоит из основного класса-контроллера интерфейса GUI, а также ряда классов для каждого экрана приложения на базе класса tk.Frame библиотеки Python Tkinter, в частности, начальный экран приложения EntryScreen, экран запуска детекции по конкретному изображению ImageDetectionScreen, экран запуска детекции по каждому изображению из указанного набора DatasetDetectionScreen, экран для передачи пути к датасету для вычисления по нему оценочных метрик DatasetEvaluationScreen.

Программное средство имеет общую входную точку – класс Main, который импортирует основной контроллер GUI с соответствующими внутренними вызовами методов корневых классов приложения.

Весь исходный код приложения представлен в Листинге программного средства.

### 3.3 Пример работы программы

Рассмотрим базовую логику работы программы. При запуске приложения через основной класс Mаin пользователь видит начальный экран с доступными ему опциями, представленными на рисунке 1:

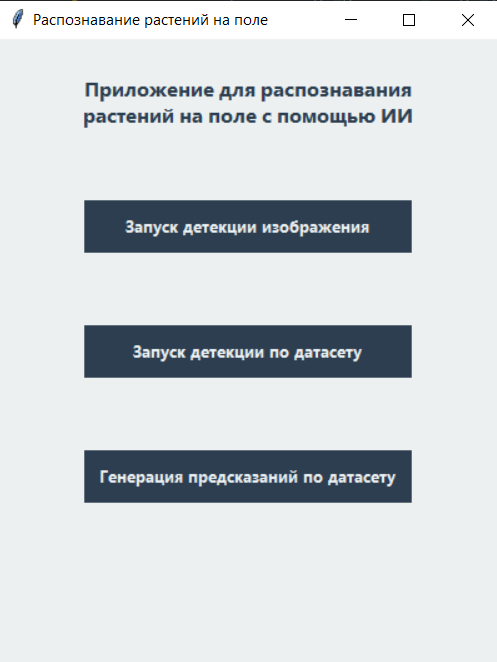


Рисунок 1 – Начальное окно при запуске программы

Выбрав, например, опцию «Запуск детекции изображения» и прописав на открывшемся экране программы, представленном на рисунке 2, необходимые данные, такие как путь к изображению и путь для сохранения результатов, пользователь получает в указанной им папке новое изображение с нанесёнными на него предсказанными моделью боксами, представленное на рисунке 3:

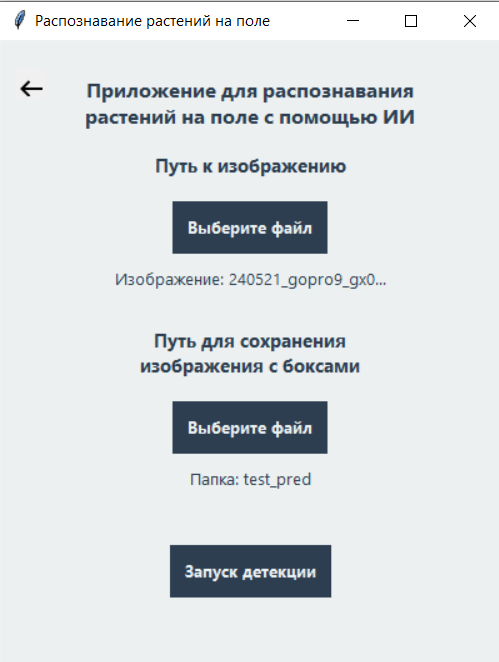


Рисунок 2 – Экран для ввода данных для проведения детекции по необходимому изображению



Рисунок 3 – Результат детекции, проведенной моделью по заданному изображению

Если пользователю требуется получить данные о работе модели в исходном JSON-формате, он может воспользоваться опцией «Генерация предсказаний по датасету», передав путь к папке с его набором изображений и путь к папке для сохранения предсказаний модели в формате JSON-файлов, соответствующих каждому изображению в указанном наборе. Экран ввода данных представлен на рисунке 4, а результаты работы программы отражены на рисунках 5-6.

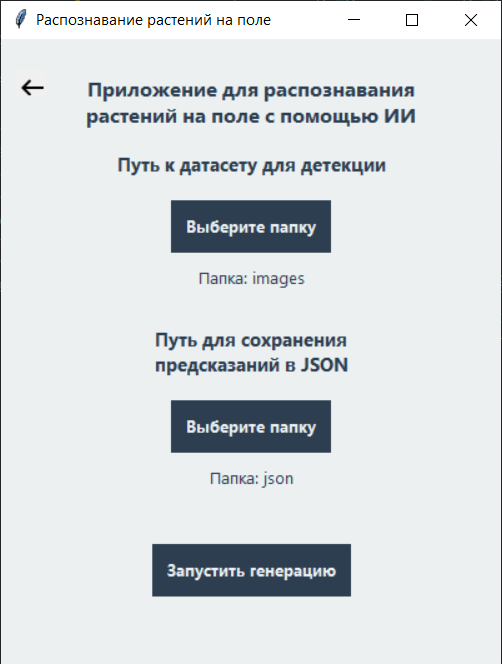


Рисунок 4 – Экран для ввода данных для генерации предсказаний по указанному датасету

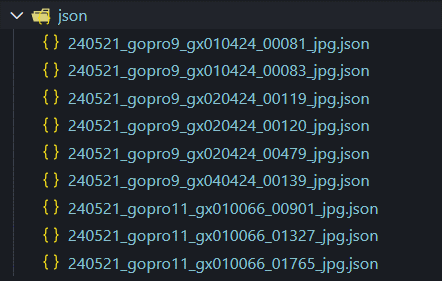


Рисунок 5 – Папка с информацией о координатах предсказанных боксов в формате JSON по каждому изображению

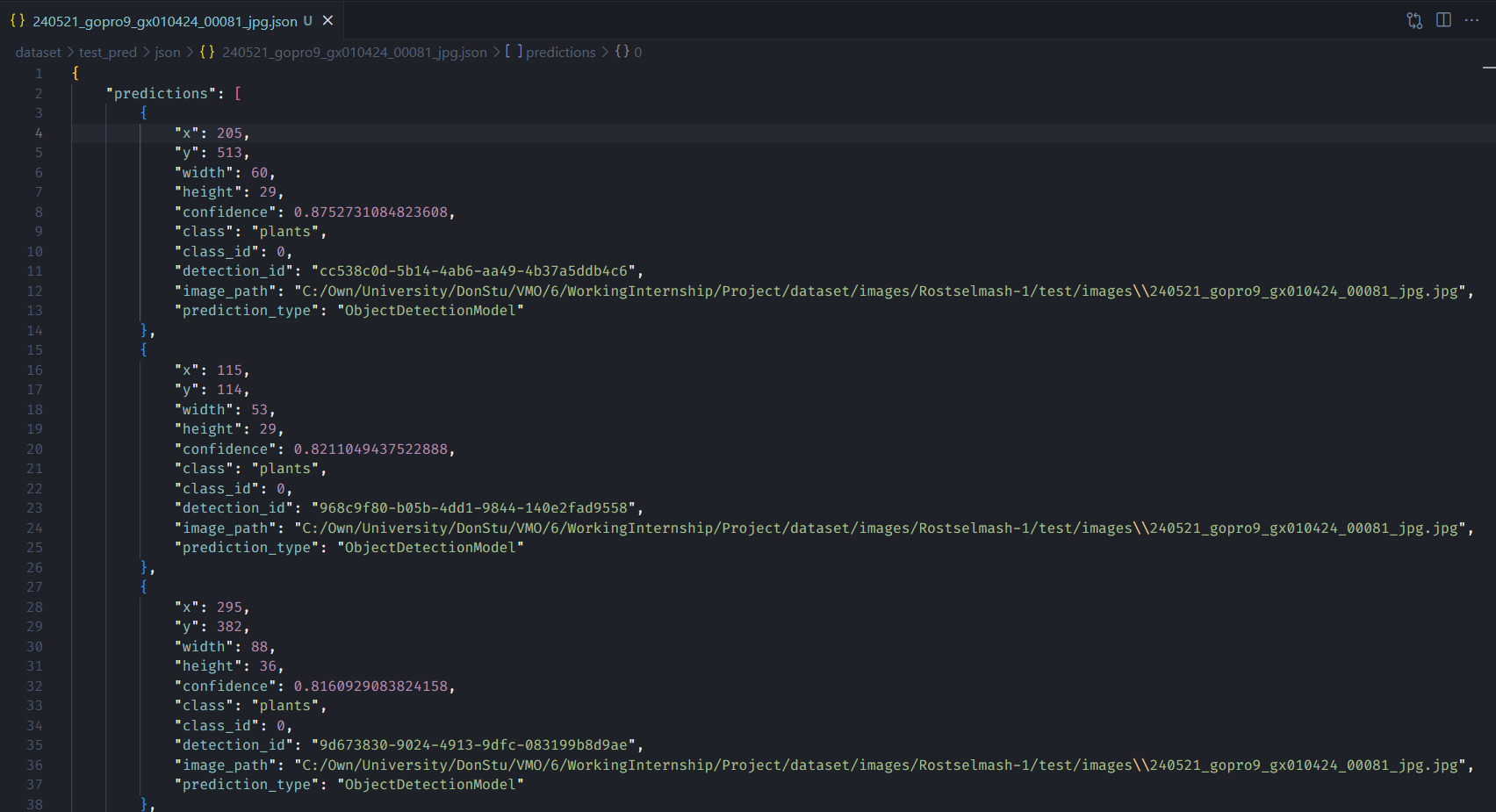


Рисунок 6 – Один из результирующих JSON-файлов по одному из изображений в наборе

При разработке программного интерфейса акцент ставился на простоту, понятность и удобство использования программы. Клиентский интерфейс скрывает всю внутреннюю реализацию по взаимодействию с моделью, преобразованию данных, обработке передаваемых путей, оставляя только самое необходимое для пользователя.

### 3.3 Вывод по главе

В данной главе рассмотрен и обоснован выбор языка программирования и средств разработки, описаны основные методы и классы программного средства, приведен базовый пример работы программы.

В процессе выбора технологий учитывались такие критерии, как производительность, удобство пользования конечным продуктом, скорость и эффективность разработки.

## 4 Тестирование программного средства

В данной главе рассмотрены результаты тестирования всех доступных пользовательских опций, предоставляемых интерфейсом программного средства, а также детали обучения модели и её тестирование.

### 4.1 Описание процесса тестирования обученной модели

При обучении модели использовался датасет из 100 изображений с растениями на поле. Пример изображения из датасета представлен на рисунке 7.



Рисунок 7 – Одно из изображения из датасета для обучения модели-распознавателя

При обучении датасета использовали 50 изображений, 20 изображения отведены в категорию valid и 10 отведены на тестирование модели.

В качестве модели была взята Roboflow Object Detection 3.0 (Accurate), подробное описание модели приведено в разделе 3.1. Перед обучением модели была проведена разметка всех ground-truth боксов средствами платформы Roboflow.

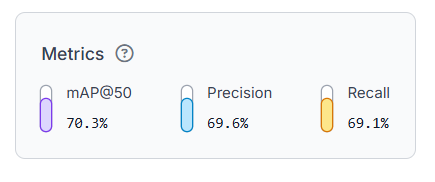
После обучения модель характеризуется следующими показателями метрик, представленными на рисунке 8.  
 

Рисунок 8 – Основные характеристические метрики модели

При обучении на относительно небольшом датасете из 50 изображений и на одном классе изображений plants достигнуты показатели 70.3% по метрике Mean Average Precision (средняя точность среди всех классов), 69.6% по метрике Precision (общая точность предсказаний) и 69.1% по метрике Recall (как часто модель точно спрогнозировала истинный результат).

Для оценки качества соотношения истинных размеченных боксов и предсказанных боксов модели построена матрица ошибок (confusion matrix), представленная на рисунке 9.

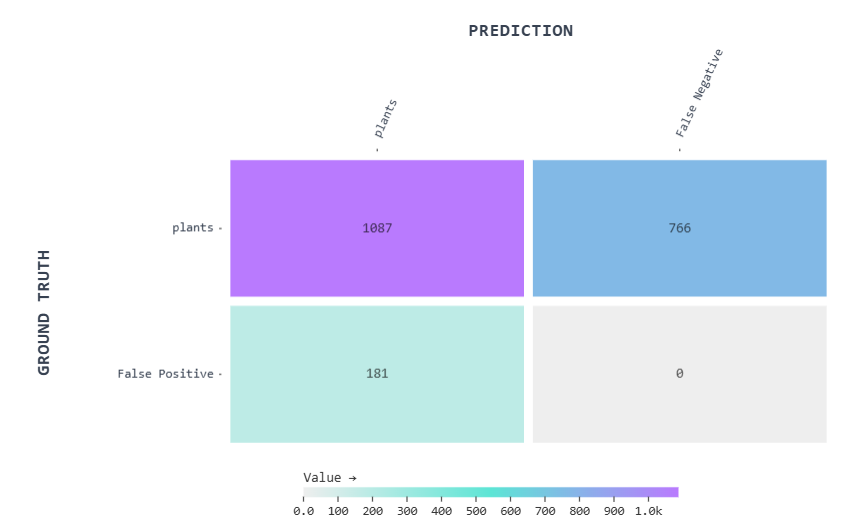


Рисунок 9 – Матрица ошибок (confusion matrix) для обученной модели

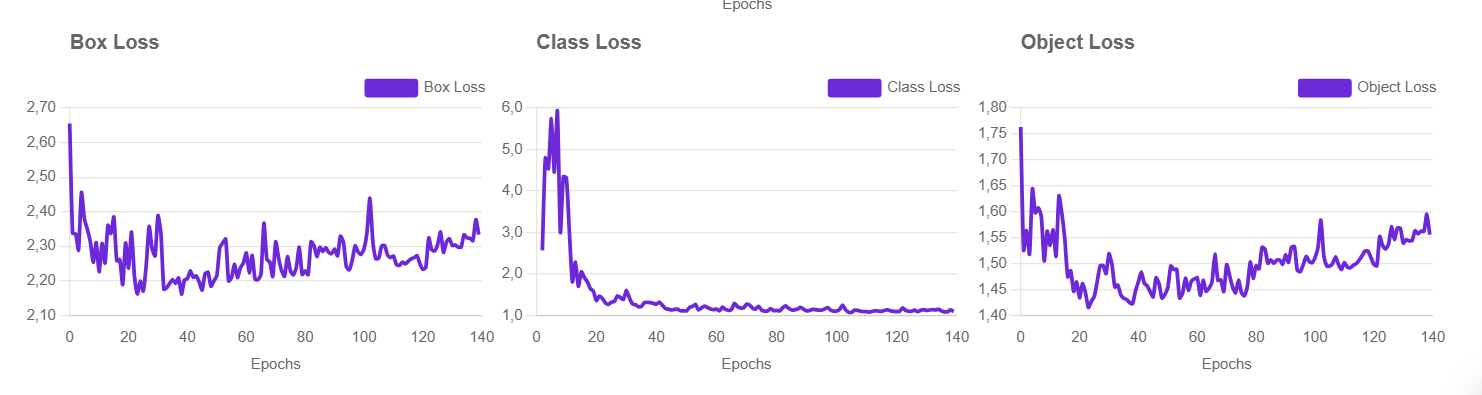
Для оценки непосредственно самого процесса обучения модели построены соответствующие графики количества потерь в зависимости от эпохи обучения модели, представленные на рисунке 10.  
 

Рисунок 10 – Потери боксов, классовые потери и потери объектов при обучении модели

Глядя на описанные выше метрики по обученной модели, можно сделать вывод, что модель следует дообучать на большем количестве изображений в датасете, текущие показатели обоснованы относительно небольшим размером изначального датасета.

Проведем тестирование модели при помощи разработанного программного интерфейса. Указав в качестве пути к датасету папку с изображениями в категории test, получим результаты детекции для 10 изображений в категории. Некоторые из них представлены на рисунках 11-13.

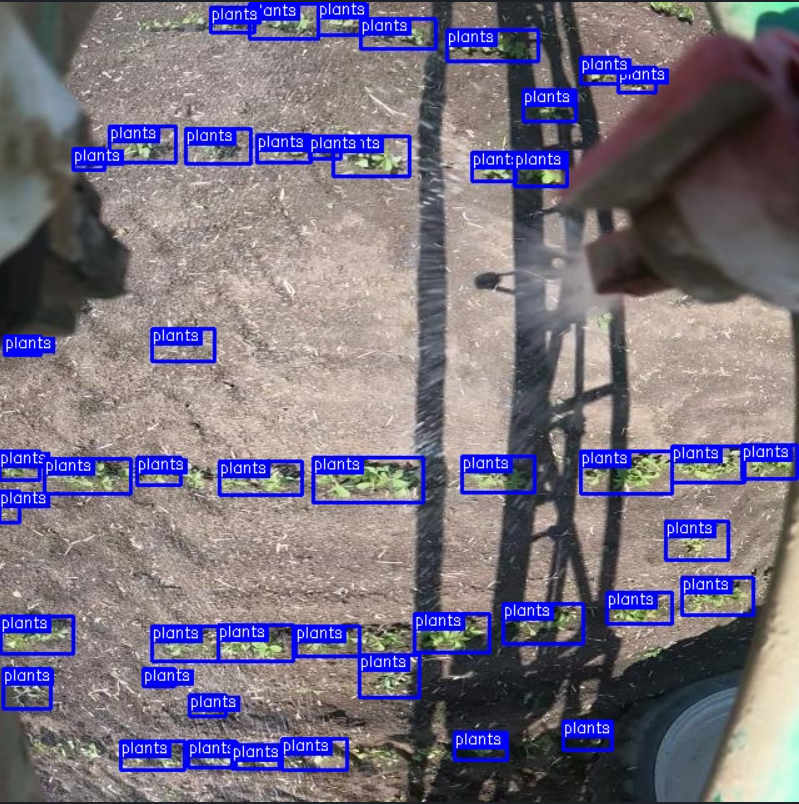


Рисунок 11 – Первое изображение датасета в категории test с предсказанными боксами



Рисунок 12 – Второе изображение датасета в категории test с предсказанными боксами



Рисунок 13 – Третье изображение датасета в категории test с предсказанными боксами

Анализируя результаты тестирования, можно заключить, что модель обучена корректно и решает свою задачу, беря в расчёт изначальный размер датасета. Модель можно дообучить, собрав более объемный датасет.

### 4.2 Описание процесса тестирования интерфейса

При запуске программы пользователю предлагается на выбор 3 опции: «Запуск детекции по изображению», «Запуск детекции по датасету» и «Генерация предсказаний по датасету». Стартовый экран представлен на рисунке 12.

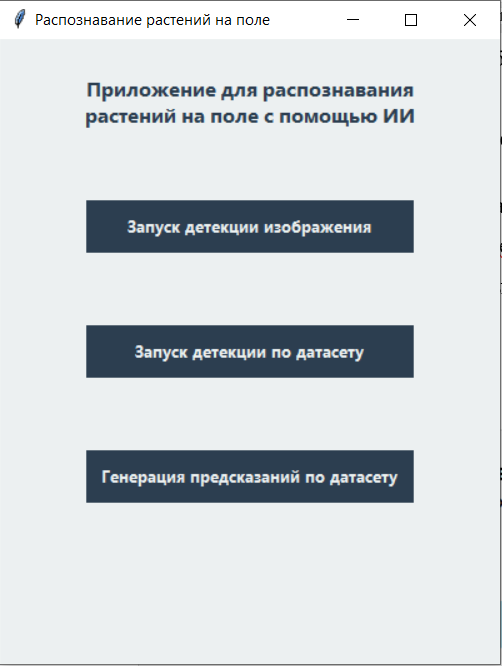


Рисунок 12 – Начальное окно при запуске программы

Нажимая на любую из кнопок на данном экране, пользователь попадает в стандартное меню импорта путей для входных данных и для сохранения данных. Экраны опций «Запуск детекции изображения», «Запуск детекции по датасету» и «Генерация предсказаний по датасету» представлены на рисунках 13-15.

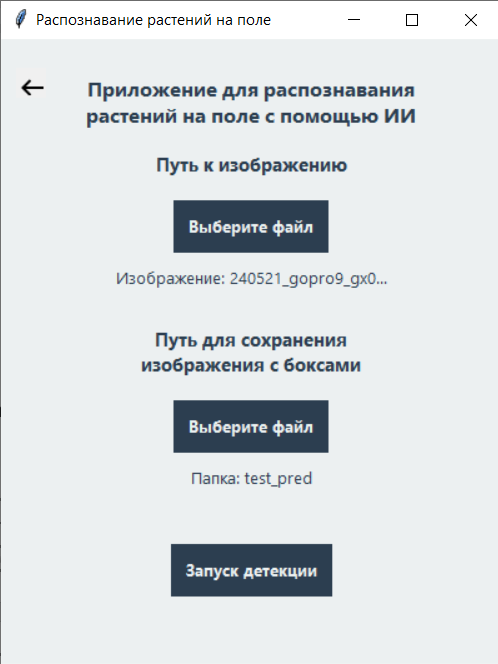


Рисунок 13 – Экран интерфейса опции «Запуск детекции изображения»

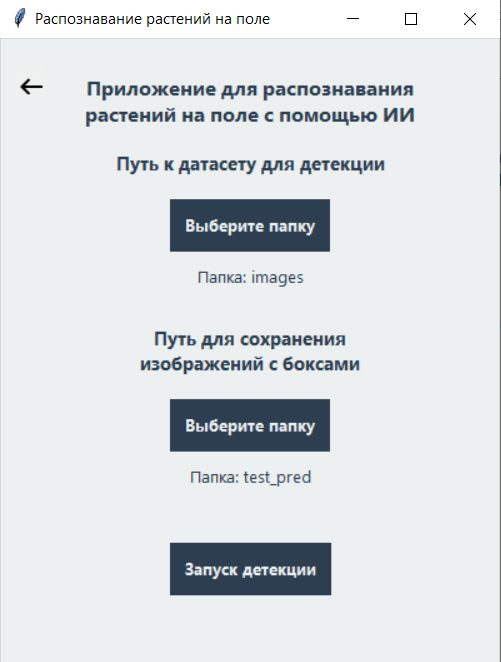


Рисунок 14 – Экран интерфейса опции «Запуск детекции изображения»

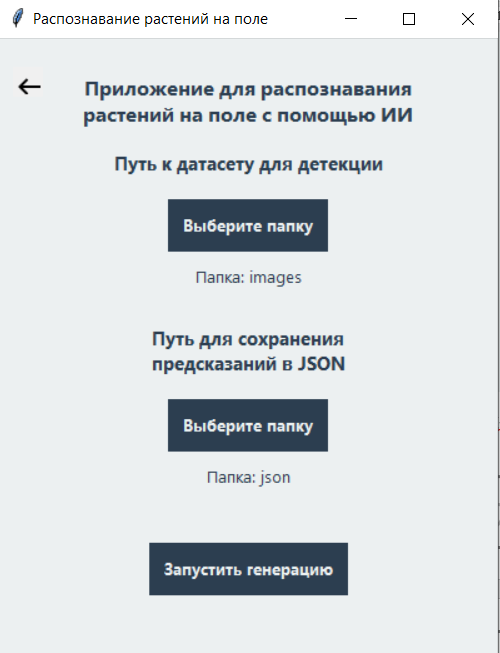


Рисунок 15 – Экран интерфейса опции «Генерация предсказаний по датасету»

При некорректном импорте либо при отсутствии импорта какого-то из путей в нижнем левом углу выводится пользовательское сообщение о необходимости импорта соответствующего пути. Примеры пользовательских сообщений представлены на рисунках 16-17.



Рисунок 16 – Пользовательское сообщение о необходимости ввода пути до изображения и до папки для сохранения предсказания



Рисунок 16 – Пользовательское сообщение о необходимости ввода пути до папки для сохранения предсказания, если введен путь только до изображения

Примеры работы приложения при выборе каждой из соответствующих опций представлены в разделе 3.3.

### 4.3 Выводы по главе

В результате тестирования обученной модели и соответствующего интерфейса для взаимодействия с ней можно установить, что программное средство работает корректно, удобно, просто и понятно в использовании. Также установлено, что обученную модель при более объемном датасете можно обучить на большую точность детекции при необходимости.

## Заключение

Задача детекции объектов на изображении остается крайне актуальной в промышленном и коммерческом ключе. Грамотное применение технологий искусственного интеллекта позволяет резко сократить расходы на ручную классификацию объектов на снимках, автоматизируя данный процесс, в частности в сфере сельского хозяйства.

В ходе данной работы были изучены принципы работы нейронных сетей, задача детекции растений на множественных снимках поля при уборке урожая, а также методы обучения модели нейронной сети для решения данной задачи.

Для решения поставленной задачи была обучена модель Roboflow Object Detection 3.0 (Accurate) на предварительно размеченных изображениях с множеством растений на поле. Для взаимодействия с моделью был разработан соответствующий программный модуль и клиентский интерфейс для понятного взаимодействия с программным средством.

Реализованное приложение содержит все базовые функции, требуемые для решения задачи детекции растений на поле, такие как детекция по изображению, по набору изображений, генерация предсказаний в формате JSON для получения результатов модели отдельно от клиентского интерфейса для дальнейшего самостоятельного анализа.

Созданное программное средство можно легко расширять дополнительными модулями для решения более сложных задач, например, использование более совершенной модели, распознающей несколько различных классов растений, проведения более сложного анализа данных, полученных при уборке урожая, либо провести дополнительное обучение существующей модели на более объемном начальном датасете.

# **Cписок использованных источников**

1. **Сегментация изображений в Computer Vision** [Электронный ресурс] // Data Light. – URL: https://data-light.ru/blog/segmentacia-izobrazenij-kompjuternoe-zrenie/ (дата обращения: 03.07.2024).
2. **Сегментация изображений в машинном обучении: виды, задачи, алгоритмы** [Электронный ресурс] // Training Data. – URL: https://trainingdata.ru/markup-dark-side-ml/tpost/bxdpm7blz1-segmentatsiya-izobrazhenii (дата обращения: 04.07.2024).
3. **Детекция объектов. R-CNN, Fast R-CNN, Faster R-CNN. Часть 1** [Электронный ресурс] // Хабр. – URL: https://habr.com/ru/articles/789942/ (дата обращения: 13.04.2025).
4. **Нахождение объектов без учителя (Unsupervised Object Detection)** [Электронный ресурс] // Хабр. – URL: https://habr.com/ru/articles/463991/ (дата обращения: 13.04.2025).
5. Template matching в компьютерном зрении [Электронный ресурс] // OTUS. – URL: https://otus.ru/nest/post/2478/ (дата обращения: 13.04.2025)
6. **Преимущества языка Python — плюсы и минусы** [Электронный ресурс] // Skysmart. – URL: https://skysmart.ru/articles/programming/preimushestva-i-nedostatki-python (дата обращения: 13.04.2025).
7. **Tkinter — Интерфейс Python для Tcl/Tk** [Электронный ресурс] // Python Docs. – URL: https://python-docs.ru/library/tkinter (дата обращения: 13.04.2025).
8. Gallagher J. **Announcing Roboflow Train 3.0** [Электронный ресурс] // Roboflow Blog. – 11.07.2023. – URL: https://blog.roboflow.com/roboflow-train-3-0/ (дата обращения: 13.04.2025).

Приложение Б. Исходный код программного средства

***Листинг Б.1– Исходный код корневых модулей программы***

Листинг Б.1.1 – Исходный код модуля BoxParser

import json

import numpy as np

from utilities.constants import \*

from core.image.CheckImage import CheckImage

import cv2

import os

from PIL import Image

class BoxParser:

    def \_\_init\_\_(self, image\_path, gt\_path, pred\_path):

        self.image\_path = image\_path

        self.gt\_path = gt\_path

        self.pred\_path = pred\_path

        # Создаем абсолютный путь к папке логов

        log\_dir = os.path.join('C:', 'Own', 'University', 'DonStu', 'VMO', '6', 'WorkingInternship', 'Project', 'dataset', 'test\_pred')

        self.log\_file = os.path.join(log\_dir, 'boxes\_log.txt')

        # Создаем папку, если она не существует

        try:

            os.makedirs(log\_dir, exist\_ok=True)

            print(f"Папка для логов создана: {log\_dir}")

        except Exception as e:

            print(f"Ошибка при создании папки для логов: {str(e)}")

        if not os.path.exists(self.gt\_path):

            raise FileNotFoundError(f"Файл {self.gt\_path} не найден.")

        if not os.path.exists(self.pred\_path):

            raise FileNotFoundError(f"Файл {self.pred\_path} не найден.")

        self.img\_width, self.img\_height = self.\_get\_image\_size()

    def \_log\_boxes(self, message):

        try:

            with open(self.log\_file, 'a', encoding='utf-8') as f:

                f.write(f"{message}\n")

            print(f"Запись в лог успешна: {self.log\_file}")

        except Exception as e:

            print(f"Ошибка при записи в лог: {str(e)}")

            print(f"Путь к лог-файлу: {self.log\_file}")

    # Получение размеров текущего изображения

    def \_get\_image\_size(self):

        if CheckImage(self.image\_path):

            try:

                with Image.open(self.image\_path) as img:

                    width, height = img.size

                    return width, height

            except FileNotFoundError:

                print(f"Ошибка: Файл '{self.image\_path}' не найден.")

            except Exception as e:

                print(f"Ошибка при чтении файла: {e}")

    # Подготовка боксов ground truth

    def prepare\_ground\_truth\_boxes(self):

        boxes = {}

        try:

            self.\_log\_boxes(f"\n=== Ground Truth Boxes for {os.path.basename(self.image\_path)} ===")

            with open(self.gt\_path, 'r', encoding='utf-8') as gt:

                for line in gt:

                    parts = line.strip().split()

                    if len(parts) != 5:

                        continue

                    cls, cx, cy, w, h = parts

                    cx, cy, w, h = map(float, (cx, cy, w, h))

                    if cls == '0':

                        cls = 'plants'

                    x = cx \* self.img\_width

                    y = cy \* self.img\_height

                    width = w \* self.img\_width

                    height = h \* self.img\_height

                    x0 = x - width / 2

                    y0 = y - height / 2

                    x1 = x + width / 2

                    y1 = y + height / 2

                    if cls not in boxes:

                        boxes[cls] = []

                    boxes[cls].append((x0, y0, x1, y1, 1.0))  # confidence=1 для GT

                    log\_message = f"Ground Truth: class={cls}, box=({x0:.2f}, {y0:.2f}, {x1:.2f}, {y1:.2f})"

                    print(log\_message)

                    self.\_log\_boxes(log\_message)

        except Exception as e:

            error\_message = f"Ошибка при чтении ground truth: {str(e)}"

            print(error\_message)

            self.\_log\_boxes(error\_message)

            return None

        return boxes

    # Подготовка предсказанных боксов

    def prepare\_prediction\_boxes(self):

        boxes = {}

        try:

            self.\_log\_boxes(f"\n=== Prediction Boxes for {os.path.basename(self.image\_path)} ===")

            with open(self.pred\_path, 'r', encoding='utf-8') as pred:

                data = json.load(pred)

                # Выводим информацию о структуре данных

                print(f"JSON data type: {type(data)}")

                if isinstance(data, dict):

                    print(f"JSON keys: {data.keys()}")

                    if 'predictions' in data:

                        print(f"Number of predictions: {len(data['predictions'])}")

                        if len(data['predictions']) > 0:

                            print(f"First prediction: {data['predictions'][0]}")

                # Проверяем формат данных

                if isinstance(data, dict) and 'predictions' in data:

                    predictions = data['predictions']

                    for box in predictions:

                        if isinstance(box, dict):

                            cls = str(box.get('class', '0'))

                            if cls == '0':

                                cls = 'plants'

                            # Получаем координаты в формате (x, y, width, height)

                            x = float(box.get('x', 0))

                            y = float(box.get('y', 0))

                            width = float(box.get('width', 0))

                            height = float(box.get('height', 0))

                            # Преобразуем в формат (x0, y0, x1, y1)

                            x0 = x

                            y0 = y

                            x1 = x + width

                            y1 = y + height

                            conf = float(box.get('confidence', 0.0))

                            if cls not in boxes:

                                boxes[cls] = []

                            boxes[cls].append((x0, y0, x1, y1, conf))

                            log\_message = f"Prediction: class={cls}, box=({x0:.2f}, {y0:.2f}, {x1:.2f}, {y1:.2f}), confidence={conf:.4f}"

                            print(log\_message)

                            self.\_log\_boxes(log\_message)

                else:

                    print(f"Неподдерживаемый формат JSON: {type(data)}")

                    return None

        except Exception as e:

            error\_message = f"Ошибка при чтении предсказаний: {str(e)}"

            print(error\_message)

            self.\_log\_boxes(error\_message)

            return None

        return boxes

    # Получение JSON и заданного файла

    def \_get\_json(self):

        with open(self.pred\_path, "r", encoding="utf-8") as f:

            return json.load(f)

Листинг Б.1.2 – Исходный код модуля BoxVisualizer

import cv2

class BoxVisualizer:

    def \_\_init\_\_(self, image\_path, gt\_boxes, pred\_boxes):

        self.image\_path = image\_path

        self.cv2\_im = cv2.imread(self.image\_path)

        if self.cv2\_im is None:

            raise ValueError(f"Ошибка: Не удалось загрузить изображение {self.image\_path}")

        self.gt\_boxes = gt\_boxes

        self.pred\_boxes = pred\_boxes

    def draw\_boxes(self):

        for cls in self.gt\_boxes.keys():

            gt\_boxes = self.gt\_boxes.get(cls, [])

            pred\_boxes = self.pred\_boxes.get(cls, [])

            # Отрисовка Ground Truth боксов (синие)

            for x0, y0, x1, y1, \_ in gt\_boxes:

                cv2.rectangle(self.cv2\_im, (x0, y0), (x1, y1), (255, 0, 0), 2)

            # Отрисовка предсказанных боксов (голубые)

            for x0, y0, x1, y1, \_ in pred\_boxes:

                cv2.rectangle(self.cv2\_im, (x0, y0), (x1, y1), (100, 149, 237), 2)

    def show\_image(self, resize\_ratio=1.0):

        self.draw\_boxes()

        if resize\_ratio != 1.0:

            new\_size = (int(self.cv2\_im.shape[1] / resize\_ratio), int(self.cv2\_im.shape[0] / resize\_ratio))

            self.cv2\_im = cv2.resize(self.cv2\_im, new\_size, interpolation=cv2.INTER\_LINEAR)

        cv2.imshow("Image with Boxes", self.cv2\_im)

        cv2.waitKey(0)

        cv2.destroyAllWindows()

Листинг Б.1.3 – Исходный код модуля BoxVisualizer

import roboflow

import requests

import json

import os

from concurrent.futures import ThreadPoolExecutor

from utilities.constants import API\_TOKEN, PROJECT\_ID, BASE\_FOLDER

class DatasetFetch:

    def \_\_init\_\_(self, api\_token, workspace, project\_id, base\_folder):

        self.rf = roboflow.Roboflow()

        self.workspace = workspace

        self.project\_id = project\_id

        self.project = self.rf.project(self.project\_id)

        self.api\_key = api\_token

        self.base\_folder = base\_folder

    @staticmethod

    def get\_json(url):

        r = requests.get(url)

        return json.dumps(r.json(), indent=4)

    @staticmethod

    def write\_to\_json(directory, j):

        d = json.loads(j)

        image\_name = d['image']['name']

        if not os.path.exists(os.path.join(os.getcwd(), directory)):

            path = os.path.join(os.getcwd(), directory)

            os.mkdir(path)

        with open(f'{directory}/{image\_name}.json', 'w', encoding='utf-8') as file:

            file.write(j)

    @staticmethod

    def pretty\_dict(dic):

        for key, val in dic.items():

            print(f"{key}: {val}")

    def get\_images\_ids(self):

        records = []

        for page in self.search\_all(

            offset=0,

            limit=100,

            class\_name="plants",

            in\_dataset=True,

            fields=["id", "name"],

        ):

            records.extend(page)

        return [{"id": record["id"], "name": record["name"]} for record in records]

    def create\_url(self, image\_id):

        base\_url = "https://api.roboflow.com"

        return f'{base\_url}/{self.workspace}/{self.project\_id}/images/{image\_id}?api\_key={self.api\_key}'

    def get\_images\_json(self, data, directory):

        img\_id = data["id"]

        url = self.create\_url(img\_id)

        js = self.get\_json(url)

        d = json.loads(js)

        image\_name = d['image']['name']

        self.write\_to\_json(directory, js)

        print(f"Информации об изображении {image\_name}.jpg получена!")

    def get\_json\_parallel(self):

        data = self.get\_images\_ids()

        with ThreadPoolExecutor() as executor:

            results = list(executor.map(lambda b: self.get\_images\_json(b, self.base\_folder), data))

        return results

# Пример запуска.

fetch = DatasetFetch(API\_TOKEN, WORKSPACE, PROJECT\_ID, BASE\_FOLDER)

fetch.get\_json\_parallel()

Листинг Б.1.4 – Исходный код модуля CheckImage

from PIL import Image

import cv2

import numpy as np

from utilities.constants import \*

import os

class CheckImage:

    def \_\_init\_\_(self, image\_path):

        self.image\_path = image\_path

    def check\_all\_errors(self):

        self.check\_if\_exists()

        self.check\_permission()

        self.check\_integrity()

        self.check\_cv2\_read()

        print(f"Изображение {self.image\_path} открыто корректно!")

        return True

    def check\_if\_exists(self):

        # Проверка существования файла

        if not os.path.exists(self.image\_path):

            raise FileNotFoundError(f"Файл {self.image\_path} не существует.")

    def check\_permission(self):

        # Проверка прав доступа

        if not os.access(self.image\_path, os.R\_OK):

            raise PermissionError(f"Нет прав на чтение файла {self.image\_path}.")

    def check\_integrity(self):

        try:

            img = Image.open(self.image\_path)

            img.verify()  # Проверка целостности

        except Exception as e:

            print(f"Файл поврежден: {e}")

    def check\_cv2\_read(self):

        # Попытка чтения

        image = cv2.imread(self.image\_path)

        if image is None:

            raise ValueError(

                f"OpenCV не смог прочитать файл {self.image\_path} (возможно, битый или неподдерживаемый формат).")

Листинг Б.1.5 – Исходный код модуля Predict

from roboflow import Roboflow

from concurrent.futures import ThreadPoolExecutor, as\_completed

import os

import json

from utilities.constants import API\_TOKEN, PROJECT\_ID, WORKSPACE\_NAME, PROJECT\_VERSION

class Predict:

    def \_\_init\_\_(self, path\_to\_dataset="", predict\_path="", path\_to\_save=""):

        self.model = self.\_model\_init()

        self.path\_to\_dataset = path\_to\_dataset

        self.predict\_path = predict\_path

        self.path\_to\_save = path\_to\_save

    @staticmethod

    def \_model\_init():

        rf = Roboflow(API\_TOKEN)

        project = rf.workspace(WORKSPACE\_NAME).project(PROJECT\_ID)

        version = project.version(PROJECT\_VERSION)

        return version.model

    def image\_generate\_image\_json(self, file, path\_to\_save):

        filename = os.fsdecode(file)

        path\_to\_save = os.fsdecode(path\_to\_save)

        image\_name, extension = os.path.splitext(os.path.basename(filename))

        path\_to\_save = f"{path\_to\_save}/{image\_name}.json"

        prediction\_json = self.model.predict(filename, confidence=40, overlap=30).json()

        self.save\_json(path\_to\_save, prediction\_json)

        print(f"Предсказание по изображению {image\_name}{extension} успешно сохранено в файл {path\_to\_save}!")

    def image\_generate\_dataset\_json(self, dataset\_path, path\_to\_save):

        dataset\_filename = os.fsdecode(dataset\_path)

        save\_path = os.fsdecode(path\_to\_save)

        if os.path.exists(save\_path):

            for file in os.listdir(save\_path):

                file\_path = os.path.join(save\_path, file)

                try:

                    if os.path.isfile(file\_path):

                        os.unlink(file\_path)

                except Exception as e:

                    print(f"Ошибка при удалении файла {file\_path}: {e}")

        image\_paths = [os.path.join(dataset\_filename, filename) for filename in os.listdir(dataset\_filename)]

        with ThreadPoolExecutor() as executor:

            futures = [executor.submit(self.image\_generate\_image\_json, image\_path, path\_to\_save)

                       for image\_path in image\_paths]

            for future in as\_completed(futures):

                future.result()

        print(f"Предсказания по датасету успешно сгенерированы и сохранены!")

    def image\_visualize\_prediction(self, file, ext=".jpg"):

        filename = os.fsdecode(file)

        image\_name, extension = os.path.splitext(os.path.basename(filename))

        path\_to\_save = f"{self.predict\_path}/{image\_name}\_processed.{ext}"

        self.model.predict(filename, confidence=40, overlap=30).save(path\_to\_save)

        print(f"Изображение {image\_name}{extension} с предсказанными боксами успешно сохранено!")

    def image\_visualize\_dataset(self, dataset\_path, ext='.jpg'):

        dataset\_filename = os.fsdecode(dataset\_path)

        image\_paths = [os.path.join(dataset\_filename, filename) for filename in os.listdir(dataset\_filename)]

        with ThreadPoolExecutor() as executor:

            futures = [executor.submit(self.image\_visualize\_prediction, image\_path, ext) for image\_path in image\_paths]

            for future in as\_completed(futures):

                future.result()

        print("Детекция по датасету успешно завершена!")

    @staticmethod

    def save\_json(path, content):

        try:

            os.makedirs(os.path.dirname(path), exist\_ok=True)

            with open(path, 'w', encoding='utf-8') as f:

                json.dump(content, f, ensure\_ascii=False, indent=4)

        except IOError as e:

            print(f"Ошибка записи файла {path}: {str(e)}")

            raise

    @staticmethod

    def print\_pretty\_json(json\_obj):

        print(json.dumps(json\_obj, indent=2))

***Листинг Б.2 – Исходный код корневых модулей интерфейса***

Листинг Б.2.1 – Исходный код входной точки Main

# Интерфейс

from gui.GUI import \*

# Файл с константами.

from utilities.constants import \*

# Класс входной точки в программу.

class Main:

    def \_\_init\_\_(self):

        # Класс интерфейса.

        self.gui = GUI(WINDOW\_WIDTH, WINDOW\_HEIGHT, WINDOW\_BG)

        self.gui.run()

# Запуск приложения.

app = Main()

Листинг Б.2.2 – Исходный код контроллера интерфейса GUI

from tkinter import \*

# Экраны приложения.

from gui.screens.EntryScreen import EntryScreen as EntryScreenCLS

from gui.screens.entry\_options.ImageDetectionScreen import ImageDetectionScreen as ImageDetectionScreenCLS

from gui.screens.entry\_options.DatasetDetectionScreen import DatasetDetectionScreen as DatasetDetectionScreenCLS

from gui.screens.entry\_options.DatasetPredictionScreen import DatasetPredictionScreen as DatasetPredictionScreenCLS

from gui.styles.theme import Theme

class GUI(Tk):

    def \_\_init\_\_(self, width, height, bg):

        Tk.\_\_init\_\_(self)

        # Сохраняем размеры окна как атрибуты

        self.width = width

        self.height = height

        # Настройка темы

        Theme.configure\_styles()

        # Настройка окна

        self.title("Распознавание растений на поле")

        self.geometry(f"{width}x{height}")

        self.configure(bg=Theme.COLORS['light'])

        # Создание контейнера для фреймов

        container = Frame(self)

        container.pack(side="top", fill="both", expand=True)

        container.grid\_rowconfigure(0, weight=1)

        container.grid\_columnconfigure(0, weight=1)

        # Словарь для хранения фреймов

        self.frames = {}

        # Создание и добавление фреймов

        for F in (EntryScreenCLS, ImageDetectionScreenCLS, DatasetDetectionScreenCLS,

                 DatasetPredictionScreenCLS):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        # Показ начального фрейма

        self.show\_frame(EntryScreenCLS)

    def show\_frame(self, cont):

        frame = self.frames[cont]

        frame.tkraise()

    def back(self, current\_frame):

        self.show\_frame(EntryScreenCLS)

    def get\_screen(self, screen\_class):

        return self.frames.get(screen\_class)

    def run(self):

        self.mainloop()

Листинг Б.2.3 – Исходный код родительского экрана ProgramScreen

# Модуль интерфейса.

from tkinter import \*

from tkinter import filedialog as fd

from gui.styles.theme import Theme

class ProgramScreen(Frame):

    def \_\_init\_\_(self, parent, controller):

        Frame.\_\_init\_\_(self, parent)

        self.controller = controller

        self.main\_label = Theme.create\_title\_label(self,

            "Приложение для распознавания\nрастений на поле с помощью ИИ"

        )

        self.main\_label.place(relx=0.5, rely=0.1, anchor=CENTER)

        # Кнопка для возвращения на предыдущий экран интерфейса

        self.canvas = Canvas(self, width=20, height=20, cursor="hand2")

        self.canvas.place(anchor=CENTER, relx=0.06, rely=0.07)

        self.backup\_img = PhotoImage(file="images/backup-arrow.png")

        self.canvas.create\_image(0, 0, anchor=NW, image=self.backup\_img)

        self.canvas.bind("<Button-1>", lambda event: self.controller.back(self.\_\_class\_\_))

    def process\_image\_path(self, screen\_class, status\_prop, filetype):

        filetypes = (

            ('JPEG Images', '\*.jpg'),

            ('JPEG Images', '\*.jpeg'),

            ('PNG Images', '\*.png'),

            ('All files', '\*.\*')

        )

        path = fd.askopenfilename(

            title='',

            initialdir='/',

            filetypes=filetypes)

        screen = self.controller.get\_screen(screen\_class)

        if screen:

            setattr(screen, status\_prop, path)

            self.set\_entry\_status(screen, path, filetype)

        return path

    def process\_dir\_path(self, screen\_class, status\_prop, filetype):

        path = fd.askdirectory(

            title='',

            initialdir='/'

        )

        # Если передан сам экран, используем его

        if isinstance(screen\_class, ProgramScreen):

            screen = screen\_class

        else:

            # Иначе получаем экран из контроллера

            screen = self.controller.get\_screen(screen\_class)

        if screen and path:

            setattr(screen, status\_prop, path)

            self.set\_entry\_status(screen, path, filetype)

        return path

    @staticmethod

    def set\_entry\_status(screen, path, filetype):

        filename = path.split("/")[-1]

        for import\_type, prop, ftp in getattr(screen, 'import\_config'):

            # Проверяем, что тип файла (ftp) соответствует переданному filetype

            if filetype == ftp:

                # Для папок показываем просто "Папка: имя\_папки"

                if import\_type == 'Папка':

                    file\_status = f'Папка: {filename if filename else "не задана"}'

                # Для файлов показываем тип файла и его имя

                else:

                    file\_status = f'{ftp}: {filename if filename else "не задан"}'

                if len(file\_status) > 30:

                    file\_status = file\_status[:30] + '...'

                getattr(screen, prop).config(text=file\_status)

                if file\_status != f"{ftp}: не задан":

                    getattr(screen, 'user\_message').configure(text="")

Листинг Б.2.4 – Исходный код начального экрана EntryScreen

from core.prediction.Predict import Predict

import matplotlib.colors as mcolors

from gui.screens.ProgramScreen import \*

from gui.custom.ColoredButton import \*

from utilities.constants import \*

from gui.styles.theme import Theme

from tkinter import ttk

from tkinter import filedialog as fd

from gui.screens.entry\_options.ImageDetectionScreen import ImageDetectionScreen as ImageDetectionScreenCLS

from gui.screens.entry\_options.DatasetDetectionScreen import DatasetDetectionScreen as DatasetDetectionScreenCLS

from gui.screens.entry\_options.DatasetPredictionScreen import DatasetPredictionScreen as DatasetPredictionScreenCLS

# Класс интерфейса настроек аппроксимации.

class EntryScreen(Frame):

    def \_\_init\_\_(self, parent, controller):

        Frame.\_\_init\_\_(self, parent)

        self.controller = controller

        self.\_entries\_data = ""

        self.\_current\_file = ""

        self.current\_file\_label = ""

        # Настройка фона

        self.configure(bg=Theme.COLORS['light'])

        # Заголовок

        self.main\_label = Theme.create\_title\_label(self,

            "Приложение для распознавания\nрастений на поле с помощью ИИ"

        )

        self.main\_label.place(relx=0.5, rely=0.1, anchor=CENTER)

        # Кнопки опций

        self.start\_image\_detection\_button = Theme.create\_rounded\_button(self,

            "Запуск детекции изображения",

            lambda: self.show\_option(ImageDetectionScreenCLS),

            width=30

        )

        self.start\_image\_detection\_button.place(relx=0.5, rely=0.3, anchor=CENTER)

        self.start\_dataset\_detection\_button = Theme.create\_rounded\_button(self,

            "Запуск детекции по датасету",

            lambda: self.show\_option(DatasetDetectionScreenCLS),

            width=30

        )

        self.start\_dataset\_detection\_button.place(relx=0.5, rely=0.5, anchor=CENTER)

        self.start\_dataset\_prediction = Theme.create\_rounded\_button(self,

            "Генерация предсказаний по датасету",

            lambda: self.show\_option(DatasetPredictionScreenCLS),

            width=30

        )

        self.start\_dataset\_prediction.place(relx=0.5, rely=0.7, anchor=CENTER)

        # Сообщение пользователю

        self.user\_message = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['danger']

        )

        self.user\_message.place(relx=0.05, rely=0.93, anchor=W)

    def show\_option(self, option\_cls, \*args):

        self.controller.show\_frame(option\_cls)

Листинг Б.2.4 – Исходный код экрана ImageDetectionScreen

from core.prediction.Predict import Predict

import matplotlib.colors as mcolors

from gui.screens.ProgramScreen import \*

from gui.custom.ColoredButton import \*

from gui.custom.Input import \*

from utilities.constants import \*

from gui.styles.theme import Theme

from tkinter import ttk

from tkinter import filedialog as fd

import os

# Интерфейсный экран для запуска детекции по изображению.

class ImageDetectionScreen(ProgramScreen):

    def \_\_init\_\_(self, parent, controller):

        ProgramScreen.\_\_init\_\_(self, parent, controller)

        self.image\_path = ''

        self.box\_path = ''

        self.import\_config = [

            ('Файл', 'image\_file\_status', 'Изображение'),

            ('Файл', 'detected\_file\_status', 'Папка'),

        ]

        # Настройка фона

        self.configure(bg=Theme.COLORS['light'])

        # Поле ввода пути к изображению

        self.path\_to\_detect\_label = Theme.create\_subtitle\_label(self,

            "Путь к изображению"

        )

        self.path\_to\_detect\_label.place(relx=0.5, rely=0.2, anchor=CENTER)

        self.path\_to\_detect\_import = Theme.create\_rounded\_button(self,

            "Выберите файл",

            self.process\_detection\_path

        )

        self.path\_to\_detect\_import.place(relx=0.5, rely=0.3, anchor=CENTER)

        # Лейбл с информацией об импорте изображения

        self.image\_file\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.image\_file\_status.place(relx=0.5, rely=0.38, anchor=CENTER)

        self.image\_file\_status.config(text="Изображение: не задано")

        # Поле ввода пути для сохранения

        self.path\_to\_detected\_label = Theme.create\_subtitle\_label(self,

            "Путь для сохранения\nизображения с боксами"

        )

        self.path\_to\_detected\_label.place(relx=0.5, rely=0.5, anchor=CENTER)

        self.path\_to\_detected\_import = Theme.create\_rounded\_button(self,

            "Выберите файл",

            self.process\_box\_path

        )

        self.path\_to\_detected\_import.place(relx=0.5, rely=0.62, anchor=CENTER)

        # Лейбл с информацией об импорте папки для сохранения

        self.detected\_file\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.detected\_file\_status.place(relx=0.5, rely=0.7, anchor=CENTER)

        self.detected\_file\_status.config(text="Папка: не задана")

        # Кнопка для запуска детекции

        self.start\_image\_detection\_button = Theme.create\_rounded\_button(self,

            "Запуск детекции",

            self.start\_image\_detection

        )

        self.start\_image\_detection\_button.place(relx=0.5, rely=0.85, anchor=CENTER)

        # Сообщение пользователю

        self.user\_message = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['danger']

        )

        self.user\_message.place(relx=0.05, rely=0.95, anchor=W)

    def start\_image\_detection(self, \*args):

        if self.\_validate\_paths():

            if (not self.image\_path.endswith('.jpg') and

                    not self.image\_path.endswith('.jpeg') and

                    not self.image\_path.endswith('.png')):

                self.user\_message.configure(text="Путь должен быть к изображению .jpg/.jpeg или .png!", fg="red")

                return

            prediction\_instance = Predict(predict\_path=self.box\_path)

            prediction\_instance.image\_visualize\_prediction(self.image\_path)

    def \_validate\_paths(self):

        if not self.image\_path and not self.box\_path:

            self.user\_message.configure(text='Не указано изображение и папка для предсказания!')

            return False

        if not self.image\_path:

            self.user\_message.configure(text='Не указан путь к изображению!')

            return False

        if not self.box\_path:

            self.user\_message.configure(text='Не указан путь к папке для записи предсказания!')

            return False

        return True

    # ImageDetectionScreen.process\_detection\_file()

    def process\_detection\_path(self, \*args):

        self.process\_image\_path(self.\_\_class\_\_, 'image\_path', 'Изображение')

    def process\_box\_path(self, \*args):

        self.process\_dir\_path(self.\_\_class\_\_, 'box\_path', 'Папка')

Листинг Б.2.5 – Исходный код экрана DatasetDetection

from core.prediction.Predict import Predict

import matplotlib.colors as mcolors

from gui.screens.ProgramScreen import \*

from gui.custom.ColoredButton import \*

from gui.custom.Input import \*

from utilities.constants import \*

from gui.styles.theme import Theme

from tkinter import ttk

from tkinter import filedialog as fd

import os

# Интерфейсный экран для запуска детекции по изображению.

class DatasetDetectionScreen(ProgramScreen):

    def \_\_init\_\_(self, parent, controller):

        ProgramScreen.\_\_init\_\_(self, parent, controller)

        self.dataset\_path = ''

        self.json\_path = ''

        self.import\_config = [

            ('Папка', 'dataset\_to\_detect\_status', 'Датасет'),

            ('Папка', 'save\_path\_status', 'Папка'),

        ]

        # Настройка фона

        self.configure(bg=Theme.COLORS['light'])

        # Поле ввода пути к датасету

        self.path\_to\_detect\_label = Theme.create\_subtitle\_label(self,

            "Путь к датасету для детекции"

        )

        self.path\_to\_detect\_label.place(relx=0.5, rely=0.2, anchor=CENTER)

        self.path\_to\_detect\_import = Theme.create\_rounded\_button(self,

            "Выберите папку",

            self.process\_detection\_path

        )

        self.path\_to\_detect\_import.place(relx=0.5, rely=0.3, anchor=CENTER)

        # Лейбл с информацией об импорте папки для детекции

        self.dataset\_to\_detect\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.dataset\_to\_detect\_status.place(relx=0.5, rely=0.38, anchor=CENTER)

        self.dataset\_to\_detect\_status.config(text="Датасет: не задан")

        # Поле ввода пути для сохранения

        self.path\_to\_json\_label = Theme.create\_subtitle\_label(self,

            "Путь для сохранения\nизображений с боксами"

        )

        self.path\_to\_json\_label.place(relx=0.5, rely=0.5, anchor=CENTER)

        self.path\_to\_json\_import = Theme.create\_rounded\_button(self,

            "Выберите папку",

            self.process\_json\_path

        )

        self.path\_to\_json\_import.place(relx=0.5, rely=0.62, anchor=CENTER)

        # Лейбл с информацией об импорте папки для сохранения

        self.save\_path\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.save\_path\_status.place(relx=0.5, rely=0.7, anchor=CENTER)

        self.save\_path\_status.config(text="Папка: не задана")

        # Кнопка для запуска детекции

        self.start\_image\_detection\_button = Theme.create\_rounded\_button(self,

            "Запуск детекции",

            self.start\_dataset\_detection

        )

        self.start\_image\_detection\_button.place(relx=0.5, rely=0.85, anchor=CENTER)

        # Сообщение пользователю

        self.user\_message = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['danger']

        )

        self.user\_message.place(relx=0.05, rely=0.95, anchor=W)

    def start\_dataset\_detection(self, \*args):

        if self.\_validate\_paths():

            prediction\_instance = Predict(predict\_path=self.json\_path)

            prediction\_instance.image\_visualize\_dataset(self.dataset\_path)

    def \_validate\_paths(self):

        if not self.dataset\_path and not self.json\_path:

            self.user\_message.configure(text='Не указано изображение и папка для предсказания!')

            return False

        if not self.dataset\_path:

            self.user\_message.configure(text='Не указан путь к изображению!')

            return False

        if not self.json\_path:

            self.user\_message.configure(text='Не указан путь к папке для записи предсказания!')

            return False

        return True

    def process\_detection\_path(self, \*args):

        self.process\_dir\_path(self.\_\_class\_\_, 'dataset\_path', 'Датасет')

    def process\_json\_path(self, \*args):

        self.process\_dir\_path(self.\_\_class\_\_, 'json\_path', 'Папка')

Листинг Б.2.6 – Исходный код экрана DatasetPrediction

from core.prediction.Predict import Predict

import matplotlib.colors as mcolors

from gui.screens.ProgramScreen import \*

from gui.custom.ColoredButton import \*

from gui.custom.Input import \*

from utilities.constants import \*

from gui.styles.theme import Theme

from tkinter import ttk

from tkinter import filedialog as fd

import os

# Интерфейсный экран для запуска детекции по изображению.

class DatasetPredictionScreen(ProgramScreen):

    def \_\_init\_\_(self, parent, controller):

        ProgramScreen.\_\_init\_\_(self, parent, controller)

        self.import\_config = [

            ('Папка', 'image\_file\_status', 'Датасет'),

            ('Папка', 'json\_file\_status', 'Папка'),

        ]

        self.dataset\_path = ''

        self.json\_path = ''

        # Настройка фона

        self.configure(bg=Theme.COLORS['light'])

        # Поле ввода пути к датасету

        self.path\_to\_dataset\_label = Theme.create\_subtitle\_label(self,

            "Путь к датасету для детекции"

        )

        self.path\_to\_dataset\_label.place(relx=0.5, rely=0.2, anchor=CENTER)

        self.path\_to\_dataset\_import = Theme.create\_rounded\_button(self,

            "Выберите папку",

            self.process\_detection\_path

        )

        self.path\_to\_dataset\_import.place(relx=0.5, rely=0.3, anchor=CENTER)

        self.image\_file\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.image\_file\_status.place(relx=0.5, rely=0.38, anchor=CENTER)

        self.image\_file\_status.config(text="Датасет: не задан")

        # Поле ввода пути для сохранения

        self.path\_to\_json\_label = Theme.create\_subtitle\_label(self,

            "Путь для сохранения\nпредсказаний в JSON"

        )

        self.path\_to\_json\_label.place(relx=0.5, rely=0.5, anchor=CENTER)

        self.path\_to\_json\_export = Theme.create\_rounded\_button(self,

            "Выберите папку",

            self.process\_json\_path

        )

        self.path\_to\_json\_export.place(relx=0.5, rely=0.62, anchor=CENTER)

        self.json\_file\_status = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['text']

        )

        self.json\_file\_status.place(relx=0.5, rely=0.7, anchor=CENTER)

        self.json\_file\_status.config(text="Папка: не задана")

        # Кнопка для запуска генерации

        self.start\_image\_detection\_button = Theme.create\_rounded\_button(self,

            "Запустить генерацию",

            self.start\_dataset\_prediction

        )

        self.start\_image\_detection\_button.place(relx=0.5, rely=0.85, anchor=CENTER)

        # Сообщение пользователю

        self.user\_message = Label(self,

            font=Theme.FONTS['body'],

            bg=Theme.COLORS['light'],

            fg=Theme.COLORS['danger']

        )

        self.user\_message.place(relx=0.05, rely=0.95, anchor=W)

    def start\_dataset\_prediction(self, \*args):

        extension = ''

        dataset\_path = os.fsencode(self.dataset\_path)

        json\_path = os.fsencode(self.json\_path)

        if not dataset\_path and not json\_path:

            self.user\_message.configure(text="Пути к датасету или папке сохранения\nнекорректны или не заданы!")

            return

        if not dataset\_path and json\_path:

            self.user\_message.configure(text="Путь к датасету некорректен\nили не задан!")

            return

        if dataset\_path and not json\_path:

            self.user\_message.configure(text="Путь к папке сохранения\nнекорректен или не задан!")

            return

        prediction\_instance = Predict()

        prediction\_instance.image\_generate\_dataset\_json(dataset\_path, json\_path)

    def process\_detection\_path(self, \*args):

        self.process\_dir\_path(self.\_\_class\_\_, 'dataset\_path', 'Датасет')

    def process\_json\_path(self, \*args):

        self.process\_dir\_path(self.\_\_class\_\_, 'json\_path', 'Папка')