МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

**«Северный (Арктический) федеральный университет имени М.В. Ломоносова»**

Высшая школа информационных технологий и автоматизированных систем

(наименование высшей школы)

**КУРСОВОЙ ПРОЕКТ**

|  |  |  |
| --- | --- | --- |
| По дисциплине | | Интеграция моделей глубокого/машинного обучения в программные |
| решения (модуль) | | |
|  | | |
| На тему | Разработка методики определения оставленных предметов в учреждении на основе | |
| нейронных сетей | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ф.И.О.  обучающихся | Наименование направления подготовки / специальности | Курс | Группа | Ф.И.О. руководителей должность / уч. степень / звание |
| Анащенко  Артем  Альбертович | 09.04.02 Информационные системы и технологии | 2 | 151266 | Васендина Ирина Сергеевна, доцент, к.т.н. |
| Гайнутдинов Владислав Ренатович | Гурьев Александр Тимофеевич, доцент, д.т.н. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Признать, что проект выполнен и защищен с отметкой |  |  |  |  |
|  |  | (отметка прописью) |  | (дата) |
| Руководитель |  |  |  | И. С. Васендина |
|  |  | (подпись руководителя) |  | (инициалы, фамилия) |

Архангельск 2024

Оглавление

[Введение 3](#_Toc164895409)

[1 Задача детекции оставленных предметов 4](#_Toc164895410)

[1.1 Обоснование темы работы 4](#_Toc164895411)

[1.2 Цель и задачи работы 4](#_Toc164895412)

[1.3 Актуальность выбранной темы 5](#_Toc164895413)

[1.4 Научная новизна 5](#_Toc164895414)

[1.5 Практическая значимость работы 6](#_Toc164895415)

[1.5.1 Умные города и видеоаналитика 6](#_Toc164895416)

[1.5.2 Проект цифровизации городского хозяйства «умный город» 7](#_Toc164895417)

[2 Обоснование применяемых алгоритмов, методов и средств 9](#_Toc164895418)

[2.1 Выбор модели 9](#_Toc164895419)

[2.2 Обучение модели 12](#_Toc164895420)

[2.3 Алгоритм трекинга объектов 15](#_Toc164895421)

[3 Формирование набора данных 19](#_Toc164895422)

[4 Проектирование программного средства 29](#_Toc164895423)

[Заключение 33](#_Toc164895424)

[Список использованных источников 34](#_Toc164895425)

[Приложение А (обязательное) листинг программного кода 1 37](#_Toc164895426)

[Приложение Б (обязательное) листинг программного кода 2 48](#_Toc164895427)

[Приложение В (обязательное) листинг программного кода 3 49](#_Toc164895428)

ВВЕДЕНИЕ

Обнаружение и поиск потерянных предметов по записям камер видеонаблюдения является критически важной задачей, которая может использоваться в различных отраслях.

В машинном обучении не существует конкретного алгоритма, предназначенного исключительно для решения детекции потерянных предметов, так как эта задача может быть решена с помощью различных алгоритмов и методов машинного обучения.

В данной курсовой работе рассматривается создание полноценного веб-приложения для обнаружения оставленных предметов. Были изучены источники по теме исследования, подготовлены данные для обучения. Также был выбран трекер, обучена модели, создан алгоритм определения оставленной вещи, который был интегрирован в приложение Flask.

1. **Задача детекции оставленных предметов**

## Обоснование темы работы

Тема магистерской диссертации - Разработка методики определения оставленных предметов в учреждении на основе нейронных сетей

Объект исследования – Система, позволяющая классифицировать объекты на изображениях с камер видеонаблюдения.

Предмет исследования - Модели, алгоритмы и численные методы классификации объектов на цифровых изображениях.

Задача предполагает обучение классификатора на основе набора данных с метками, содержащего изображения различных оставленных предметов, таких как чемоданы, рюкзаки, сумки и т.д. Классификатор учится идентифицировать эти предметы на основе их визуальных характеристик. После обучения классификатор используется для обнаружения оставленных предметов на записях камер видеонаблюдения в реальном времени в таких общественных местах, как учебные заведения [1, 2].

В целом это интересная и сложная область исследований в области машинного обучения, которая может оказать значительное влияние на различные отрасли промышленности и улучшить повседневную жизнь людей.

## Цель и задачи работы

Цель исследования состоит во внедрении разработанной системы определения оставленных вещей в образовательных учреждениях.

Задачи исследования:

- изучение способов определения оставленных предметов с камер видеонаблюдения;

- разработка алгоритма определения оставленных предметов в учреждении на основе нейронных сетей;

- оценка эффективности методики на практике;

- внедрение разработанной методики в учреждении.

## Актуальность выбранной темы

Эта задача имеет практическое применение в различных отраслях, таких как логистика, транспорт и безопасность. Например, в аэропортах система может помочь обнаружить потерянный багаж, а на вокзалах - потерянные кошельки и телефоны. Кроме того, система может использоваться в общественных местах, чтобы помочь найти потерянные вещи их владельцев, тем самым уменьшая количество оставленных вещей на складах вокзалов [1, 2].

Проект «Умный город» направлен на формирование эффективной системы управления городским хозяйством, создание безопасных и комфортных условий для жизни горожан и повышение конкурентоспособности российских городов.

Нашей команде как части проекта САФУ определена тематика диссертации «Разработка методики определения оставленных предметов в учреждении на основе нейронных сетей». Проект диссертации будет реализован на инфраструктуре IT-парка «Цифровая Арктика».

IT-парк создан совместно САФУ им. М.В. Ломоносова и правительством Архангельской области в рамках программы развития университета. Основная цель IT-парка - создание арктической межрегиональной IT-экосистемы для формирования цифровых компетенций населения, подготовки кадров и реализации технологических проектов цифровой экономики. На территории располагается 15 IТ-лабораторий, виртуальная лаборатория, конференц-залы, коворкинг-пространства, пространства для проектной работы, IТ-кафе.

## Научная новизна

Данная задача представляет собой сложную проблему в компьютерном зрении и машинном обучении, поскольку потерянные предметы могут быть обнаружены в разных направлениях, при разных условиях освещения и на разных фонах. На данный момент не существует конкретного алгоритма, предназначенного исключительно для решения детекции оставленных предметов, так как эта задача может быть решена с помощью различных алгоритмов и методов машинного обучения. Для решения этой проблемы можно использовать такие методы, как обнаружение объектов, семантическая сегментация и сегментация экземпляров, чтобы обнаружить и точно классифицировать потерянные предметы в сложных сценах. В рамках научной работы был разработан алгоритм определения оставленных вещей с камер видеонаблюдения [1, 2].

## Практическая значимость работы

### Умные города и видеоаналитика

Умные города представляют собой комплексное решение для повышения качества жизни населения, оптимизации инфраструктуры и снижения уровня преступности. Использование современных технологий, таких как Интернет вещей (IoT), машинное обучение и видеоаналитика, позволяет собирать и анализировать большие объемы данных для принятия эффективных решений на уровне городского управления.

Видеоаналитика является одним из ключевых инструментов умных городов и позволяет обеспечивать безопасность населения, контролировать дорожное движение, следить за состоянием инфраструктуры и предотвращать экстренные ситуации.

Под видеоаналитикой часто понимают использование программных алгоритмов для анализа изображений с камер наблюдения и определения условий, при которых должна срабатывать сигнализация, например, если нарушитель перемещается в контролируемую зону.

Видеоаналитика представляет собой программное обеспечение (ПО) для работы с видеоконтентом. Основное преимущество этой технологии заключается в том, что она позволяет автоматизировать иногда утомительную задачу мониторинга изображений с камер наблюдения и обрабатывать большие объемы информации, которые не могут быть обработаны человеком-оператором на контрольной панели [3].

Последние достижения в области распознавания образов с помощью видеоанализа широко используют биометрические данные человека - черты лица, отпечатки пальцев, рисунок радужной оболочки глаза и другие - в качестве ключевых элементов классификации [3].

Ниже представлены основные задачи, стоящие перед анализом изображений [3]:

- идентификация и классификация номерных знаков и лиц;

- идентификация оставленных объектов;

- идентификация объекта, входящего или выходящего из наблюдаемой зоны;

- идентификация движения, его направления и размера объекта;

- автоматическое отслеживание движения;

- обнаружение дыма и масс;

- обнаружение параметров покупки;

- снижение и идентификация помех;

- подсчет людей, проходящих через зону контроля.

### Проект цифровизации городского хозяйства «Умный город»

Проект «Умный город» направлен на формирование эффективной системы управления городским хозяйством, создание безопасных и комфортных условий для жизни горожан и повышение конкурентоспособности российских городов [4].

Проект базируется на 5 ключевых принципах [5]:

- ориентация на человека;

- технологичность городской инфраструктуры;

- повышение качества управления городскими ресурсами;

- комфортная и безопасная среда;

- акцент на экономической эффективности, в том числе, сервисной составляющей городской среды.

Проект реализуется с 2018 года в рамках национального проекта «Жилье и городская среда» и национальной программы «Цифровая экономика Российской Федерации» [4].

Цели проекта [5]:

- комплексное повышение эффективности городской инфраструктуры;

- цифровая трансформация городского пространства и автоматизация процессов.

Направления проекта [4]:

- городская среда;

- безопасный город;

- цифровое городское управление;

- благосостояние людей;

- инвестиционный климат.

В рамках проекта создан Национальный Центр компетенций проекта «Умный город», который будет заниматься разработкой, внедрением и популяризацией технологий, оборудования, программ, направленных на повышение уровня цифровизации городского хозяйства, а также подготовкой и оказанием содействия проектам международного сотрудничества по вопросам жилищной политики, городского развития и управления природными ресурсами, прежде всего касающимся создания и функционирования «умных городов» [3].

В рамках участия Северного (Арктического) федерального университета имени М.В. Ломоносова в проекте «Умный город» поставлена задача «Безопасная городская среда на основе видеоаналитики»

Нашей команде как части проекта САФУ определена тематика диссертации «Разработка методики мониторинга и оценки использования МТО образовательного учреждения с использованием глубокого обучения». Проект диссертации будет реализован на инфраструктуре IT-парка «Цифровая Арктика».

IT-парк создан совместно САФУ им. М.В. Ломоносова и правительством Архангельской области в рамках программы развития университета. Основная цель IT-парка - создание арктической межрегиональной IT-экосистемы для формирования цифровых компетенций населения, подготовки кадров и реализации технологических проектов цифровой экономики. На территории располагается 15 IТ-лабораторий, виртуальная лаборатория, конференц-залы, коворкинг-пространства, пространства для проектной работы, IТ-кафе.

1. **Обоснование применяемых алгоритмов, методов и средств**

## Выбор модели

Задача распознавания объектов на изображении относится к компьютерному зрению и обработке изображений. Мы приняли решение использовать предварительно обученную модель с дообучением на собранном наборе данных по тематике AOD.

Предварительно обученные модели — это модели глубокого обучения, которые были обучены на большом наборе данных, таком как ImageNet, содержащем более 1,2 миллиона изображений. Это дает им ряд преимуществ при решении задачи распознавания объектов. Во-первых, они обеспечивают более высокую точность, поскольку уже научились распознавать распознавать широкий спектр объектов на начальном этапе обучения. Во-вторых, им требуется меньше обучающих данных, поскольку они уже обучены на большом наборе данных. В-третьих, они требуют меньше вычислительных ресурсов, так как могут быть точно настроены для конкретных задач с меньшим количеством параметров, чем при обучении модели с нуля [6].

Были рассмотрены следующие популярные предварительно обученные модели:

- Faster R-CNN;

- YOLO.

1. Faster R-CNN - это алгоритм обнаружения объектов, предложенный Шаоцином Реном в 2015 году, состоящий из четырех частей: сети извлечения признаков (feature extraction network, FEN), сети региональных предположений (region proposal network, RPN), объединения регионов интереса (RoI pooling) и полносвязного слоя (рис. 3). RPN состоит из серии конволюционных слоев, за которыми следует головка предложений регионов, обеспечивающая оценки объективности и дельты регрессии для каждого предложения регионов. Faster R-CNN демонстрирует отличную производительность на нескольких эталонных наборах данных, таких как COCO и Pascal VOC [7, 8, 9].

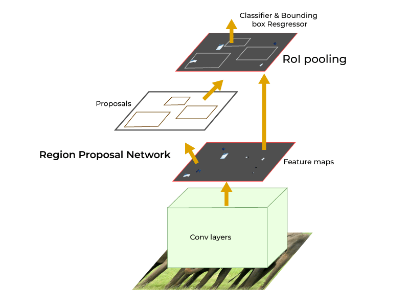


Рисунок 3 - Архитектура Faster R-CNN

Faster R-CNN находит применение в таких областях, как автономное вождение, видеонаблюдение и медицинская визуализация, поскольку она способна точно и эффективно распознавать объекты в видеопотоках в реальном времени. Faster R-CNN также можно использовать для решения таких задач, как семантическая сегментация и сегментация экземпляров, модифицируя сеть обнаружения для предсказания меток или масок пикселей для каждого объекта на изображении.

2. YOLO (You Only Look Once) — это одноэтапный алгоритм глубокого обучения, который использует сверточные нейронные сети (CNN) для одновременного предсказания ограничительных рамок и классов объектов для нескольких объектов на изображении. В отличие от других алгоритмов обнаружения объектов, таких как Faster R-CNN, которые используют сеть предложений регионов (RPN) для генерации предложений о потенциальном местоположении объектов, YOLO предсказывает ограничительные рамки и классы объектов непосредственно по изображению.

Архитектура YOLO состоит из одной CNN, которая делит входное изображение на NxN сетей. Каждая ячейка сети предсказывает ограничительные рамки и классы объектов для объектов в своей области. После определения ограничительных рамок YOLO выбирает наиболее подходящие из них, используя метод Intersection over Union (IoU) (рис. 4) [8, 10].

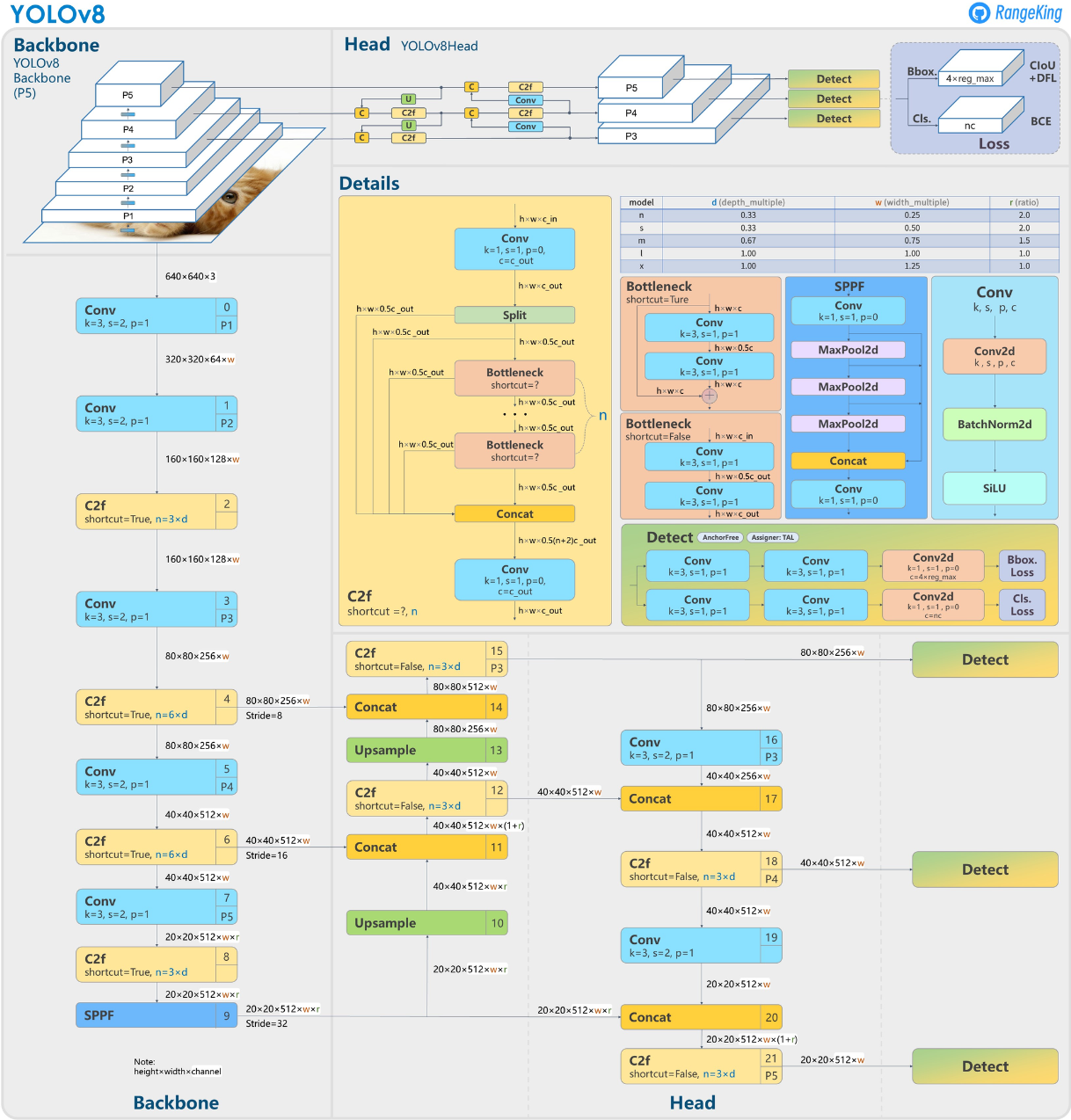


Рисунок 4 - Архитектура YOLOv8

YOLO имеет ряд преимуществ перед другими моделями для задачи распознавания объектов:

1) Одним из основных преимуществ YOLO является ее работа в режиме реального времени при распознавании объектов с низкой задержкой в видео и потоках. Faster R-CNN, несмотря на высокую точность, обычно работают медленнее и требуют больше вычислительных ресурсов для вывода, что невозможно в приложениях реального времени с высокой частотой кадров или большим объемом входных данных.

2) YOLO выполняет обнаружение объектов за один проход по входному изображению, что снижает вычислительную сложность и время ожидания по сравнению с многоступенчатыми детекторами, такими как Faster R-CNN, которые требуют нескольких проходов по изображению для генерации и уточнения предложений, а затем классификации и локализации предложений на отдельном этапе с использованием отдельной сети или ветви сетевой архитектуры.

3) Хотя YOLO работает в режиме реального времени, она достигла наивысшей точности на нескольких эталонных наборах данных, таких как COCO и Pascal VOC. Это объясняется использованием единой сетевой архитектуры для локализации и классификации объектов, захватывающей как пространственную, так и семантическую информацию в изображении.

4) YOLO обеспечивает большую гибкость в количестве и размере предсказываемых границ, которые могут быть адаптированы к конкретным требованиям приложения и положению объектов в сцене.

Таким образом было принято решение использовать модель YOLO v8 в своей работе.

## Обучение модели

Для обучения была выбрана модель yolov8n, а ее дообучение проводилось на разных устройствах:

* + - 1. Устройство с процессором Intel Xeon CPU E5-2680 и 62 Гбайт оперативной памяти;
      2. Устройство с процессором Apple M1 и 16 Гбайт оперативной памяти;
      3. Устройство с графическим процессором Nvidia Tesla T4 и 12 Гбайт оперативной памяти.

При обучении модели использовались следующие гиперпараметры:

- Количество эпох – 30;

- Размер изображения – 640;

- Размер батча – от 10 до 72;

- Оптимайзер – AdamW(lr=0.00125, momentum=0.9);

- lr0 – 0.01;

- lrf – 0.01;

- box – 7.5;

- cls – 0.5;

- dfl – 1.5;

Наилучшие результаты производительности показало устройство с графическим процессором Nvidia Tesla T4, затрачивая в среднем 20 минут на эпоху.

Код обучения представлен в листинге 1. Полный код представлен в приложении Б.

Листинг 1 – Код обучения модели

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

data\_path = '/content/AOD\_dataset/data.yaml'

model.train(

data = data\_path,

epochs = 30,

batch = -1, # AutoBatch

imgsz = 640,

name = 'yolov8n\_gpu',

seed = 42

)

Результаты обучения представлены в таблице 2.

Таблица 2 – Результаты обучения модели

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| epoch | precision(B) | recall(B) | mAP50(B) | mAP50-95(B) | box\_loss | cls\_loss | dfl\_loss |
| 1 | 0.49333 | 0.37069 | 0.37629 | 0.18717 | 1.5459 | 2.2385 | 1.9662 |
| 2 | 0.32535 | 0.16343 | 0.17112 | 0.06734 | 2.1698 | 2.6123 | 3.3042 |
| 3 | 0.25659 | 0.19508 | 0.15 | 0.06481 | 2.0217 | 3.3986 | 2.6687 |
| 4 | 0.36174 | 0.33378 | 0.30552 | 0.15542 | 1.724 | 2.0751 | 2.1584 |
| 5 | 0.40927 | 0.37045 | 0.34372 | 0.18025 | 1.5859 | 1.7507 | 1.9272 |
| 6 | 0.46724 | 0.45034 | 0.40073 | 0.20772 | 1.5529 | 1.876 | 1.9645 |
| 7 | 0.61554 | 0.45243 | 0.47113 | 0.25864 | 1.5222 | 1.6198 | 1.9745 |
| 8 | 0.61265 | 0.39557 | 0.44314 | 0.2513 | 1.4188 | 1.6926 | 1.7887 |
| 9 | 0.60754 | 0.49721 | 0.54026 | 0.32707 | 1.286 | 1.3879 | 1.7062 |
| 10 | 0.5069 | 0.43278 | 0.44253 | 0.23155 | 1.5155 | 1.6734 | 1.9106 |
| 11 | 0.58543 | 0.49156 | 0.52691 | 0.3119 | 1.2777 | 1.3166 | 1.6666 |
| 12 | 0.59806 | 0.52507 | 0.55085 | 0.33767 | 1.2914 | 1.2733 | 1.685 |
| 13 | 0.71766 | 0.49997 | 0.57772 | 0.36035 | 1.2551 | 1.2321 | 1.6331 |
| 14 | 0.67183 | 0.55139 | 0.58181 | 0.35872 | 1.2116 | 1.199 | 1.5933 |
| 15 | 0.68064 | 0.57325 | 0.61194 | 0.38103 | 1.2226 | 1.1529 | 1.597 |
| 16 | 0.66505 | 0.55055 | 0.59428 | 0.37527 | 1.1778 | 1.1631 | 1.568 |
| 17 | 0.65564 | 0.54686 | 0.5906 | 0.3664 | 1.1847 | 1.173 | 1.5659 |
| 18 | 0.68397 | 0.58341 | 0.61905 | 0.39461 | 1.1242 | 1.1139 | 1.4997 |
| 19 | 0.66837 | 0.57997 | 0.62809 | 0.408 | 1.0965 | 1.063 | 1.4799 |
| 20 | 0.64459 | 0.6188 | 0.64145 | 0.41257 | 1.0801 | 1.0375 | 1.4707 |
| 21 | 0.70559 | 0.59689 | 0.64138 | 0.42301 | 1.0714 | 1.0332 | 1.4642 |
| 22 | 0.68998 | 0.614 | 0.64768 | 0.42937 | 1.0366 | 0.98563 | 1.4322 |
| 23 | 0.7161 | 0.62596 | 0.66083 | 0.42767 | 1.0569 | 0.9912 | 1.4474 |
| 24 | 0.72276 | 0.62591 | 0.6684 | 0.44555 | 1.0136 | 0.93698 | 1.392 |
| 25 | 0.7398 | 0.60587 | 0.66115 | 0.44073 | 1.0085 | 0.93797 | 1.3866 |
| 26 | 0.70254 | 0.63129 | 0.66618 | 0.45135 | 0.98159 | 0.92533 | 1.3695 |
| 27 | 0.74729 | 0.63078 | 0.67056 | 0.45056 | 0.97883 | 0.91475 | 1.3688 |
| 28 | 0.70351 | 0.64695 | 0.67943 | 0.4595 | 0.96065 | 0.90824 | 1.3466 |
| 29 | 0.72947 | 0.63616 | 0.6817 | 0.46297 | 0.95433 | 0.88665 | 1.338 |
| 30 | 0.72208 | 0.65509 | 0.68437 | 0.46582 | 0.94418 | 0.87212 | 1.3326 |

По результатам обучения была выбрана модель, обученная на 30 эпохах.

На рисунке 6 представлен график результатов обучения.

A group of graphs showing different results

Description automatically generated with medium confidence

Рисунок 6 – результаты обучения

На рисунке 7 представлены обработанные обученной моделью изображения.

A collage of a photo of a backpack

Description automatically generated

Рисунок 7 – обработанные изображения

## Алгоритм трекинга объектов

Далее необходимо выбрать алгоритм трекинга объектов.

Поскольку в кадре предполагается множество объектов, то были рассмотрены только алгоритмы с трекингом множества объектов (MOT). Принцип выбора алгоритма основывается на выборе лучшего соотношения точности и скорости работы – насколько быстро алгоритм обрабатывает видеокадр и прогнозирует метку и местоположение объекта. Были отобраны и рассмотрены следующие алгоритмы:

- DeepSORT;

- FairMOT;

- MDNet;

- BoT-SORT;

- ByteTrack.

DeepSORT использует дополнительную сверточную нейронную сеть в качестве средства извлечения признаков. Данные признаки позволяют определять идентичность объекта в различных сценариях и позволяют алгоритму различать движущиеся объекты. Из преимуществ можно выделить высокую производительность в реальном времени и простую реализацию, однако точность остается невысокой. Архитектура представлена на рисунке 8.

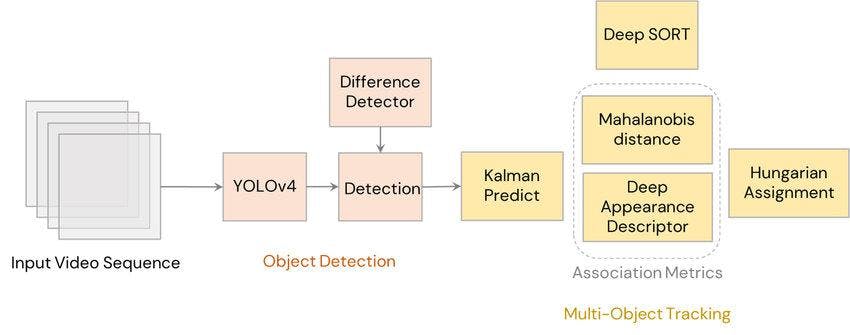


Рисунок 8 – Архитектура DeepSort

FairMOT используется предварительно обученную модель (R-CNN) для обнаружения объектов в видеопоследовательности, а затем использует нейронную сеть для извлечения особенностей обнаруженного объекта. Алгоритм обладает очень высокой точностью, однако требует очень больших вычислительных ресурсов. Архитектура представлена на рисунке 9.

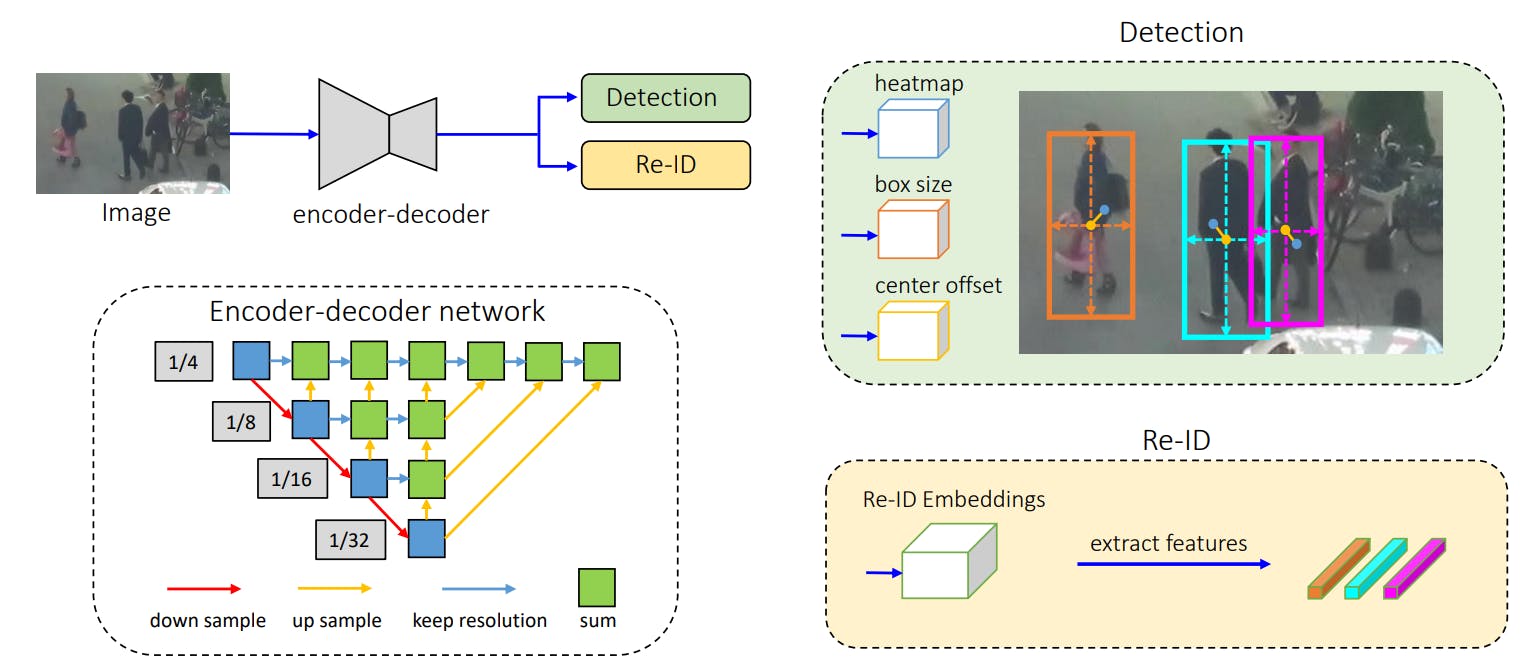


Рисунок 9 – Архитектура FairMOT

MDNet состоит из двух модулей. Первый – архитектура CNN, а второй состоит из параллельных полносвязных уровней, каждый из которых обрабатывает информацию, специфичную для предметной области. Данный алгоритм обладает отличной производительностью, однако требует большого количества данных для обучения, а также испытывает проблемы с производительностью при несбалансированных данных. Архитектура представлена на рисунке 10.

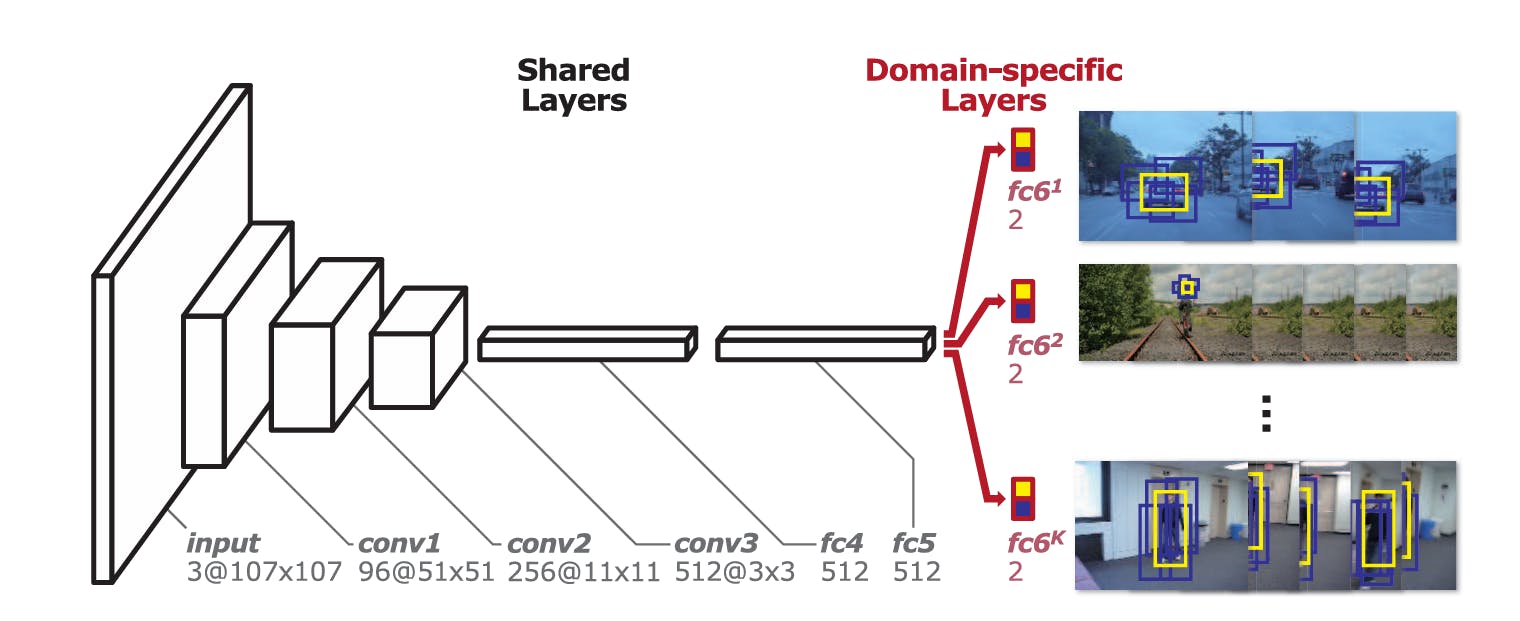


Рисунок 10 – Архитектура MDNet

BoT-SORT и ByteTrack входят в набор фреймворка YOLO. BoT-SORT превосходит ByteTrack в точности, в то время как ByteTrack показывает более высокую производительность. BoT-SORT наиболее эффективен в сценариях, включающих повторную идентификацию и новую ассоциацию трекера, по сравнению с ByteTrack. На рисунке 11 представлено сравнение точности.

A group of people walking in a large room

Description automatically generated

Рисунок 11 – Сравнение точности BoT-SORT и ByteTrack

Из всех алгоритмов наиболее подходящим для задачи был выбран BoT-SORT, поскольку его можно использовать в реальном времени при небольших вычислительных мощностях, но с хорошей точностью. Нам не требутся высокая производительность ByteTrack, так как в кадре будут отсутствовать быстро движущиеся объекты и их количество не будет превышать 60 кадров в секунду.

1. **Формирование НАбора данных**

Для решения поставленной задачи потребуются большие объемы предварительно подготовленных данных. Потребность в таких наборах существует для любой задачи, независимо от отрасли применения.

В рамках данной работы рассмотрим общедоступные наборы данных, которые возможно использовать для задачи AOD, размещенные Roboflow.

Roboflow — это платформа искусственного интеллекта, предназначенная для облегчения создания и внедрения моделей компьютерного зрения. Официальный сайт представляет собой коллекцию наборов данных и API для компьютерного зрения с открытым исходным кодом. На сайте представлено более 200 миллионов изображений, 200 000 наборов данных и 50 000 точно настроенных моделей. Пользователи могут искать и фильтровать соответствующие проекты, оценивать состояние наборов данных и точность моделей, а также опробовать модели в браузере, используя изображения или видеоданные. Сайт также предлагает облачные API для масштабируемого развертывания и возможность запуска моделей на пограничных устройствах с помощью Roboflow Inference Server. Наборы данных, размещенные на Roboflow возможно скачать в популярных форматах: YOLOv8, YOLOv5, YOLOv7, MT-YOLOv6, COCO JSON, YOLO Darknet, Pascal VOC XML, TFRecord, CreateML JSON, Other Formats [11].

В проекте использовались следующие данные:

1) За основу взят набор данных Abandoned Objects detection Computer Vision Project. Этот набор данных был создан David Sujitha и содержит 4826 изображений. Набор включает такие классы, как Handbag, Human-body, Luggage-and-bags, Wheelchair [12].

2) New Backpacks Computer Vision Project - набор данных, размещенный на Roboflow, с открытым исходным кодом для обнаружения рюкзаков. Он был создан пользователем Common Objects и содержит 1047 изображений. Набор создан для решения задачи Object Detection, субъекты набора – рюкзаки, класс – luggage [13].

3) Abandoned Object Detection Computer Vision Project - набор данных, размещенный на Roboflow, с открытым исходным кодом для анализа брошенных сумок с помощью беспилотников. Он был создан пользователем Aradhyas workspace и содержит 138 изображений. Датасет создан для решения задачи Object Detection, субъекты набора – сумки и чемоданы, классы - Bag, Person [14].

4) Abandoned Bags Computer Vision Project - набор данных, размещенный на Roboflow, с открытым исходным кодом для анализа брошенных сумок с помощью беспилотников. Он был создан пользователем Drone Analysis и содержит 4 версии с 2514, 2271, 2028 и 1434 изображений соответственно. При скачивании в формате YOLOv8 разметка искажена и не соответствует требованиям. При скачивании YOLO Darknet разметка изображений в нужном формате, однако потребовалось реорганизовать датасеты. Набор создан для решения задачи Object Detection, субъекты набора – брошенные сумки, класс – luggage [15].

Для объединения описанных ранее наборов данных использовались следующие библиотеки:

- os;

- shutil;

- yaml;

- random;

- matplotlib;

- cv2.

Модуль os предоставляет портативный способ использования множества функций для работы с операционной системой [16].

Модуль shutil содержит набор функций высокого уровня для обработки файлов, групп файлов, и папок [17].

Модуль PyYAML предназначен для работы с YAML в Python. Он не входит в стандартную библиотеку модулей [18].

Модуль random предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности [19].

Matplotlib — это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций на Python [20].

OpenCV (Open Source Computer Vision Library) — это библиотека с открытым исходным кодом, включающая несколько сотен алгоритмов компьютерного зрения [21].

В первую очередь была создана директория merging\_dataset' и файл data.yaml (листинг 1).

Листинг 1 – Создание директории для объединения наборов данных

merging\_dataset\_path = 'datasets/merging\_dataset'

os.makedirs(merging\_dataset\_path, exist\_ok=True)

split\_group\_list = ['train', 'valid', 'test']

for group in split\_group\_list:

os.makedirs(f'{merging\_dataset\_path}/{group}/images', exist\_ok=True)

os.makedirs(f'{merging\_dataset\_path}/{group}/labels', exist\_ok=True)

train\_path = os.path.abspath(f'{merging\_dataset\_path}/train')

val\_path = os.path.abspath(f'{merging\_dataset\_path}/valid')

test\_path = os.path.abspath(f'{merging\_dataset\_path}/test')

merging\_dataset\_data = {'train': f'{train\_path}',

'val': f'{val\_path}',

'test': f'{test\_path}',

'nc': 4,

'names': ['Handbag', # 0

'Human-body', # 1

'Luggage-and-bags', # 2

'Wheelchair' # 3

] # 3

}

with open(f'{merging\_dataset\_path}/data.yaml', 'w') as file:

documents = yaml.dump(merging\_dataset\_data, file, sort\_keys=False)

Для объединения наборов 1-3 были написаны функции vizualize\_img, merge\_dataset и merger\_check, представленные в листингах 2-4.

Листинг 2 – функция vizualize\_img

def vizualize\_img(image\_path,

label\_path):

with open(label\_path, "r") as file:

labels = file.read().strip().split("\n")

file.close()

image = cv2.imread(image\_path)

for label in labels:

if len(label.split()) != 5:

continue

class\_id, x\_center, y\_center, width, height = map(float, label.split())

x\_min = int((x\_center - width/2) \* image.shape[1])

y\_min = int((y\_center - height/2) \* image.shape[0])

x\_max = int((x\_center + width/2) \* image.shape[1])

y\_max = int((y\_center + height/2) \* image.shape[0])

cv2.rectangle(image, (x\_min, y\_min), (x\_max, y\_max), (0, 255, 0), 3)

cv2.putText(image, f'{class\_id}', (x\_min, y\_min-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (36,255,12), 2)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

Листинг 3 – Функция merge\_dataset

def merge\_dataset(old\_dataset\_path,

merging\_dataset\_path,

map\_dict):

split\_group\_list = ['train', 'valid', 'test']

for group in split\_group\_list:

copytree(f'{old\_dataset\_path}/{group}/images',

f'{merging\_dataset\_path}/{group}/images', dirs\_exist\_ok=True)

copytree(f'{old\_dataset\_path}/{group}/labels',

f'{merging\_dataset\_path}/{group}/labels', dirs\_exist\_ok=True)

for group in split\_group\_list:

label\_file\_list = os.listdir(f'{old\_dataset\_path}/{group}/labels')

for label\_file in label\_file\_list:

with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

for line in data:

line\_index = data.index(line)

class\_id, x\_center, y\_center, width, height = map(float, line.split())

class\_id = str(int(class\_id))

class\_id = class\_id.replace(class\_id,map\_dict[class\_id])

data[line\_index] = class\_id + f' {x\_center} {y\_center} {width} {height}\n'

new\_data = data.copy()

file.close()

with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'w') as file:

for new\_line in new\_data:

file.write(new\_line)

file.close()

Листинг 4 – Функция merger\_check

def merger\_check(img\_name,

old\_dataset\_path,

new\_dataset\_path):

fig = plt.subplots(1, 2, figsize=(16,8))

image\_path1 = f"{old\_dataset\_path}/train/images/{img\_name}.jpg"

label\_path1 = f"{old\_dataset\_path}/train/labels/{img\_name}.txt"

plt.subplot(121).set\_title(old\_dataset\_path)

vizualize\_img(image\_path1, label\_path1)

image\_path2 = f"{new\_dataset\_path}/train/images/{img\_name}.jpg"

label\_path2 = f"{new\_dataset\_path}/train/labels/{img\_name}.txt"

plt.subplot(122).set\_title(new\_dataset\_path)

vizualize\_img(image\_path2, label\_path2)

Для преобразования четырех версий 4 набора из формата YOLO Darknet в формат YOLOv8 были написаны функции merge\_yolo\_darknet\_to\_yolo\_v8 и merger\_check\_from\_yolo\_darknet, представленные в листингах 5-6.

Листинг 5 – Функция merge\_yolo\_darknet\_to\_yolo\_v8

def merge\_yolo\_darknet\_to\_yolo\_v8(old\_dataset\_path,

merging\_dataset\_path,

map\_dict):

split\_group\_list = ['train', 'valid', 'test']

for group in split\_group\_list:

all\_group\_files = os.listdir(f'{old\_dataset\_path}/{group}')

for file in all\_group\_files:

if '.txt' in file:

copyfile(f'{old\_dataset\_path}/{group}/{file}',

f'{merging\_dataset\_path}/{group}/labels/{file}')

elif '.jpg' in file:

copyfile(f'{old\_dataset\_path}/{group}/{file}',

f'{merging\_dataset\_path}/{group}/images/{file}')

for group in split\_group\_list:

all\_group\_files = os.listdir(f'{old\_dataset\_path}/{group}')

label\_file\_list = []

for file in all\_group\_files:

if '.txt' in file:

label\_file\_list.append(file)

for label\_file in label\_file\_list:

with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

for line in data:

line\_index = data.index(line)

class\_id, x\_center, y\_center, width, height = map(float, line.split())

class\_id = str(int(class\_id))

class\_id = class\_id.replace(class\_id,map\_dict[class\_id])

data[line\_index] = class\_id + f' {x\_center} {y\_center} {width} {height}\n'

new\_data = data.copy()

file.close()

with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'w') as file:

for new\_line in new\_data:

file.write(new\_line)

file.close()

Листинг 6 – Функция merger\_check\_from\_yolo\_darknet

def merger\_check\_from\_yolo\_darknet(img\_name,

old\_dataset\_path,

new\_dataset\_path):

fig = plt.subplots(1, 2, figsize=(16,8))

image\_path1 = f"{old\_dataset\_path}/train/{img\_name}.jpg"

label\_path1 = f"{old\_dataset\_path}/train/{img\_name}.txt"

plt.subplot(121).set\_title(old\_dataset\_path)

vizualize\_img(image\_path1, label\_path1)

image\_path2 = f"{new\_dataset\_path}/train/images/{img\_name}.jpg"

label\_path2 = f"{new\_dataset\_path}/train/labels/{img\_name}.txt"

plt.subplot(122).set\_title(new\_dataset\_path)

vizualize\_img(image\_path2, label\_path2)

Далее набор данных 1 был скопирован в созданную директорию в листинге 1, остальные датасеты скопированы в соответствии с форматом (листинг 7).

Листинг 7 – Объединение наборов данных

dataset1\_path = 'datasets/Abandoned-Objects-detection-1'

for group in split\_group\_list:

copytree(f'{dataset1\_path}/{group}/images',

f'{merging\_dataset\_path}/{group}/images',dirs\_exist\_ok=True)

copytree(f'{dataset1\_path}/{group}/labels', f'{merging\_dataset\_path}/{group}/labels',dirs\_exist\_ok=True)

merging\_dataset\_path = 'datasets/merging\_dataset'

map\_dict = {'0': '2'

}

v8\_datasets = {'New-Backpacks-1': map\_dict,

'Abandoned-Object-Detection-2': map\_dict}

for dataset, map\_dict in v8\_datasets.items():

old\_dataset\_path = f"datasets/{dataset}"

merge\_dataset(old\_dataset\_path, merging\_dataset\_path, map\_dict)

merging\_dataset\_path = 'datasets/merging\_dataset'

map\_dict = {'0': '2'

}

darknet\_datasets = {'Abandoned Bags.v1i.darknet': map\_dict,

'Abandoned Bags.v2i.darknet': map\_dict,

'Abandoned Bags.v3i.darknet': map\_dict,

'Abandoned Bags.v4i.darknet': map\_dict}

for dataset, map\_dict in darknet\_datasets.items():

old\_dataset\_path = f"datasets/{dataset}"

merge\_yolo\_darknet\_to\_yolo\_v8(old\_dataset\_path, merging\_dataset\_path, map\_dict)

После успешного запуска обучения модели нами было принято решение объединить все данные в одну папку и сбалансировать набор данных по классам с равным процентным распределением между train test val.

Была создана отдельная директория full\_data для объединения всех данных и скопированы данные (листинг 8).

Листинг 8 – Объединение и копирование данных

merging\_dataset\_path = 'datasets/full\_data'

os.makedirs(merging\_dataset\_path, exist\_ok=True)

os.makedirs(f'{merging\_dataset\_path}/images', exist\_ok=True)

os.makedirs(f'{merging\_dataset\_path}/labels', exist\_ok=True)

new\_dataset\_path = 'datasets/full\_data'

old\_dataset\_path = 'datasets/merging\_dataset'

split\_group\_list = ['train', 'valid', 'test']

for group in split\_group\_list:

copytree(f'{old\_dataset\_path}/{group}/images',

f'{new\_dataset\_path}/images',dirs\_exist\_ok=True)

copytree(f'{old\_dataset\_path}/{group}/labels',

f'{new\_dataset\_path}/labels',dirs\_exist\_ok=True)

Следующим этапом была осуществлена проверка на наличие дубликатов с помощью библиотки hashlib и их последующее удаление. Hashlib – Модуль, предоставляющий набор функций для работы с криптографическими хеш-функциями [22].

Для этого была написана функция duplicate\_removal (листинг 9).

Листинг 9 – Функция duplicate\_removal

def duplicate\_removal(dataset\_path):

directory = f'{dataset\_path}/images'

hashes = set()

for filename in os.listdir(directory):

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

digest = hashlib.sha1(open(path,'rb').read()).digest()

if digest not in hashes:

hashes.add(digest)

else:

os.remove(path)

label\_name = filename.replace('.jpg','.txt')

label\_path = f'{dataset\_path}/labels/{label\_name}'

os.remove(label\_path)

В результате было удалено 73 изображения.

Балансировка данных по классам включала в себя следующие этапы:

1 этап. Функция number\_of\_objects\_by\_class для подсчета количества объектов каждого класса (листинг 10)

Листинг 10 - Функция number\_of\_objects\_by\_class

def number\_of\_objects\_by\_class(labels\_path):

label\_file\_list = os.listdir(labels\_path)

num\_Handbag, num\_Human\_body = 0, 0

num\_Luggage\_and\_bags, num\_Wheelchair = 0, 0

for label\_file in label\_file\_list:

with open(f'{labels\_path}/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

for line in data:

class\_id = int(line[0])

if class\_id == 0:

num\_Handbag += 1

elif class\_id == 1:

num\_Human\_body += 1

elif class\_id == 2:

num\_Luggage\_and\_bags += 1

elif class\_id == 3:

num\_Wheelchair += 1

file.close()

print(f"{os.path.dirname(labels\_path)}\n"

f"Handbag: {num\_Handbag}\n"

f"Human-body: {num\_Human\_body}\n"

f"Luggage-and-bags: {num\_Luggage\_and\_bags}\n"

f"Wheelchair: {num\_Wheelchair}\n")

Таким образом общее количество объектов по классам следующее: Handbag 1946, Human-body 5623, Luggage-and-bags 11260 и Wheelchair 1464.

2 этап. Создаем списки с названиями изображений с разметкой по классам начиная с класса с наименьшим количеством объектов – Wheelchair (листинг 11).

Листинг 11 – Списки данных по классам

labels\_path = 'datasets/full\_data/labels'

label\_file\_list = os.listdir(labels\_path)

Wheelchair\_files = []

Handbag\_files = []

Human\_body\_files = []

for label\_file in label\_file\_list:

with open(f'{labels\_path}/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

class\_list = [int(line[0]) for line in data]

if 3 in class\_list:

Wheelchair\_files.append(label\_file)

label\_file\_list.remove(label\_file)

file.close()

for label\_file in label\_file\_list:

with open(f'{labels\_path}/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

class\_list = [int(line[0]) for line in data]

if 0 in class\_list:

Handbag\_files.append(label\_file)

label\_file\_list.remove(label\_file)

file.close()

for label\_file in label\_file\_list:

with open(f'{labels\_path}/{label\_file}', 'r') as file:

data = file.read().splitlines(True)

class\_list = [int(line[0]) for line in data]

if 1 in class\_list:

Human\_body\_files.append(label\_file)

label\_file\_list.remove(label\_file)

file.close()

Luggage\_and\_bags\_files = label\_file\_list

3 этап. Функция class\_dataset\_split для распределения на train, val, test в соотношении 80%, 10% и 10% (листинг 12)

Листинг 12 – Функция class\_dataset\_split

def class\_dataset\_split(classes\_files):

test\_val\_num = int(len(classes\_files) \* 0.1)

train\_num = len(classes\_files) - test\_val\_num \* 2

train\_files.extend(random.sample(classes\_files, train\_num))

interim\_list = [file for file in classes\_files if (file not in train\_files)]

val\_files.extend(random.sample(interim\_list, test\_val\_num))

test\_files.extend([file for file in interim\_list if (file not in val\_files)])

train\_files, val\_files, test\_files = [], [], []

all\_files = [Wheelchair\_files,

Handbag\_files,

Human\_body\_files,

Luggage\_and\_bags\_files]

for classes\_files in all\_files:

class\_dataset\_split(classes\_files)

4 этап. Создаем директорию AOD\_dataset для сбалансированного набора данных и копируем данные (листинг 13).

Листинг 13 – Создание директории и копирование данных

merging\_dataset\_path = 'datasets/AOD\_dataset'

os.makedirs(merging\_dataset\_path, exist\_ok=True)

split\_group\_list = ['train', 'valid', 'test']

for group in split\_group\_list:

os.makedirs(f'{merging\_dataset\_path}/{group}/images', exist\_ok=True)

os.makedirs(f'{merging\_dataset\_path}/{group}/labels', exist\_ok=True)

train\_path = os.path.abspath(f'{merging\_dataset\_path}/train')

val\_path = os.path.abspath(f'{merging\_dataset\_path}/valid')

test\_path = os.path.abspath(f'{merging\_dataset\_path}/test')

merging\_dataset\_data = {'train': f'{train\_path}',

'val': f'{val\_path}',

'test': f'{test\_path}',

'nc': 4,

'names': ['Handbag', # 0

'Human-body', # 1

'Luggage-and-bags', # 2

'Wheelchair' # 3

] # 3

}

with open(f'{merging\_dataset\_path}/data.yaml', 'w') as file:

documents = yaml.dump(merging\_dataset\_data, file, sort\_keys=False)

merging\_dataset\_path = 'datasets/AOD\_dataset'

old\_dataset\_path = 'datasets/full\_data'

split\_group\_list = ['train', 'valid', 'test']

grop\_file\_list = [train\_files, val\_files, test\_files]

for index, group in enumerate(split\_group\_list):

group\_files = grop\_file\_list[index]

for file in group\_files:

copyfile(f'{old\_dataset\_path}/labels/{file}',

f'{merging\_dataset\_path}/{group}/labels/{file}')

image\_name = file.replace('.txt','.jpg')

copyfile(f'{old\_dataset\_path}/images/{image\_name}', f'{merging\_dataset\_path}/{group}/images/{image\_name}')

5 этап. Сравнение данных train до и после балансировки, представлено на рисунке 5.

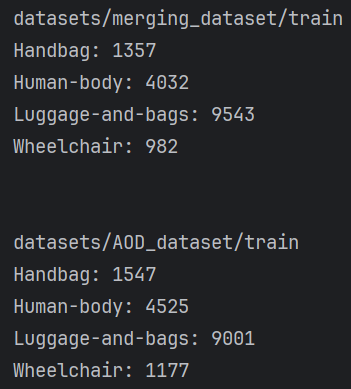


Рисунок 5 – Сравнение данных до и после балансировки

1. **проектирование программного средства**

Для реализации системы обнаружения был выбран формат веб-приложения, поскольку такой формат имеет множество преимуществ:

1. Большая совместимость с различными устройствами, поскольку для работы необходимо только наличие браузера
2. Контроль обновления приложений со стороны разработчика
3. Отсутствие необходимости обеспечивать обратную совместимость со старыми версиями приложения при добавлении нового функционала или изменении старого
4. Приложение может быть развернуто как локально (на сервере заказчика), так и удаленно

Для реализации веб-приложения был выбран следующий стек:

1. Язык программирования Python 3.10
2. Фреймворк Flask
3. Библиотека OpenCV для работы с видео
4. Фреймворк ultralytics для работы с моделями машинного обучения
5. Утилита poetry для фиксации зависимостей приложения
6. База данных PostgreSQL

В разрабатываемом приложении использование базы данных необходимо для системы авторизации. В базе хранится логин и хэш пароля пользователя. Аутентификация основана на стандарте HttpBasicAuth. Экран авторизации представлен на рисунке 6.

A screenshot of a computer

Description automatically generated

Рисунок 6 – Экран авторизации

После авторизации пользователь попадает на основной экран приложения (рисунок 7).

A screenshot of a video

Description automatically generated

Рисунок 7 – Основной экран приложения

Изображение с камеры захватывалось и нарезалось покадрово, после чего каждый кадр обрабатывался моделью, далее на этот кадр наносилась разметка в соответствии с предсказанными данными.

В веб-приложении Flask было создано три endpoint [23]:

1. Корневой маршрут (‘/’)
2. Маршрут потокового видео (‘/video\_feed’)
3. Маршрут получения числа обнаруженных объектов (‘/objects\_count’)

На корневой маршрут возвращается сгенерированная HTML-страница, на котором располагается объект видео. Этот объект обращается к роуту ‘/video\_feed’, который возвращает потоковое видео. Обработчик этого эндпоинта обращается к потоку камеры видеонаблюдения, захватывает изображение и обрабатывает его через модель.

В приложении также используется Pydantic для валидации полей моделей, при помощи него создан объект настроек с определяющими для запуска приложения параметрами. Это не позволяет запуститься приложению без указанных настроек, что в свою очередь упрощает отладку ошибок при развертывании приложения у заказчика.

Приложение построено на основе архитектурного паттерна MVC (Model – View – Controller), которой позволил разделить данные и логику приложения на три независимых друг от друга компонента: модель, вид и контроллер. Это позволит модифицировать в дальнейшем отдельные части приложения, не изменяя при этом остальные. На рисунке 8 представлена схема паттерна [24].



Рисунок 8 – Паттерн MVC

Рекомендуется держать базу данных и приложение на отдельных серверах. Диаграмма развертывания представлена на рисунке 9.

A diagram of a computer network

Description automatically generated

Рисунок 9 – Диаграмма развертывания

Заключение

В ходе написания курсовой работы были выполнены поставленные задачи. Были собраны и подготовлены данные, обучена модель, а также написан алгоритм определения оставленных вещей.

В качестве перспектив развития данного проекта, возможно внедрение детекции дополнительных классов, таких как ноутбуки, телефоны и т.д. Также можно улучшать алгоритм, тестируя его в сценах с различным освещением и местоположением камер.

Список использованных источников

1. Селедец, И. Е. Обнаружение аномалий с помощью машинного обучения [Электронный ресурс]/ И. Е. Селедец, Д. И. Борунов, В. А. Рычков // Синергия Наук. – 2020. – № 44. – С. 415-422. – EDN GXVBWU. – Электрон. дан. – Режим доступа: https://www.elibrary.ru/item.asp?id=42802631

2. Коновалов, И. А. Обзор методов машинного обучения в контексте решения задачи обнаружения аномалий в сетевом трафике [Электронный ресурс]/ И. А. Коновалов, Ю. И. Бауман // Аллея науки. – 2018. – Т. 2, № 3(19). – С. 732-746. – EDN YWRIPQ. – Электрон. дан. – Режим доступа: https://www.elibrary.ru/item.asp?id=32834567

3. Проект Цифровизации городского хозяйства «Умный город» [Электронный ресурс]/ Минстрой России [офиц. сайт] – Электрон. дан. – Режим доступа: https://minstroyrf.gov.ru/trades/gorodskaya-sreda/proekt-tsifrovizatsii-gorodskogo-khozyaystva-umnyy-gorod

4. Сайт проекта «Умный город» [Электронный ресурс]/ Ведомственный проект Минстроая России [офиц. сайт] – Электрон. дан. – Режим доступа: https://russiasmartcity.ru/

5. Шах, Д. А. Методы видеоаналитики в деятельности учреждений и органов уголовно-исполнительной системы [Электронный ресурс] / Д. А. Шах // Актуальные проблемы пенитенциарной науки и практики. – 2019. – № 2(12). – С. 103-105. – EDN NHVYSF. – Электрон. дан. – Режим доступа: https://www.elibrary.ru/item.asp?id=48247873

6. Побережник, В. И. Классификация структур информационной модели с помощью предварительно обученных нейронных сетей [Электронный ресурс]/ В. И. Побережник // Приложение математики в экономических и технических исследованиях. – 2020. – № 1(10). – С. 147-154. – Электрон. дан. – Режим доступа: https://elibrary.ru/item.asp?id=44280118

7. Сирота, А. А. Анализ алгоритмов поиска объектов на изображениях с использованием различных модификаций сверточных нейронных сетей [Электронный ресурс]/ А. А. Сирота, Е. Ю. Митрофанова, А. И. Милованова // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. – 2019. – № 3. – С. 123-137. – Электрон. дан. – Режим доступа: https://elibrary.ru/item.asp?id=41187782

8. Тимошкин, М. С. Сравнение YOLO V5 и Faster R-CNN для обнаружения людей на изображении в потоковом режиме [Электронный ресурс]/ М. С. Тимошкин, А. Н. Миронов, А. С. Леонтьев // Международный научно-исследовательский журнал. – 2022. – № 6-1(120). – С. 137-146. – DOI 10.23670/IRJ.2022.120.6.020. – Электрон. дан. – Режим доступа: https://elibrary.ru/item.asp?id=48705942

9. Ren Shaoqing. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / Shaoqing Ren, Kaiming He, Ross Girshick et al. // arXiv:1506.01497v1. – 2015. – 10 p.

10. Использование нейронной сети Yolov8 для детектирования на изображении лабораторного грызуна (крысы или мыши) в установке «Квадратное открытое поле» [Электронный ресурс]/ И. Л. Гринин, С. В. Кравченко, А. В. Чумаков, А. С. Тарасов // NovaUm.Ru. – 2023. – № 43. – С. 24-30. – Электрон. дан. – Режим доступа: https://elibrary.ru/item.asp?id=54167475

11. Roboflow Universe [Электронный ресурс] / Roboflow [офиц. сайт] – Электрон. дан. – Режим доступа: https://universe.roboflow.com/

12. Abandoned Objects detection Computer Vision Project [Электронный ресурс] / Roboflow [офиц. сайт] – Электрон. дан. – Режим доступа: https://universe.roboflow.com/david-sujitha/abandoned-objects-detection-rwmxc

13. New Backpacks Computer Vision Project [Электронный ресурс] / Roboflow [офиц. сайт] – Электрон. дан. – Режим доступа: https://universe.roboflow.com/common-objects/new-backpacks

14. Abandoned Object Detection Computer Vision Project [Электронный ресурс] / Roboflow [офиц. сайт] – Электрон. дан. – Режим доступа: https://universe.roboflow.com/aradhyas-workspace/abandoned-object-detection

15. Abandoned Bags Computer Vision Project [Электронный ресурс] / Roboflow [офиц. сайт] – Электрон. дан. – Режим доступа: https://universe.roboflow.com/drone-analysis/abandoned-bags-2afuo

16. Модуль os [Электронный ресурс] / Python 3 для начинающих [офиц. сайт] – Электрон. дан. – Режим доступа: https://pythonworld.ru/moduli/modul-os.html

17. Модуль shutil [Электронный ресурс] / Python 3 для начинающих [офиц. сайт] – Электрон. дан. – Режим доступа: https://pythonworld.ru/moduli/modul-shutil.html

18. Работа с файлами в формате YAML [Электронный ресурс] / Python для сетевых инженеров [офиц. сайт] – Электрон. дан. – Режим доступа: https://pyneng.readthedocs.io/ru/latest/book/17\_serialization/yaml.html

19. Модуль random [Электронный ресурс] / Python 3 для начинающих [офиц. сайт] – Электрон. дан. – Режим доступа: https://pythonworld.ru/moduli/modul-random.html

20. Matplotlib 3.8.4 documentation [Электронный ресурс] / Matplotlib [офиц. сайт] – Электрон. дан. – Режим доступа: https://matplotlib.org/stable/index.html

21. OpenCV-Python Tutorials [Электронный ресурс] / OpenCV Documentation [офиц. сайт] – Электрон. дан. – Режим доступа: https://docs.opencv.org/4.x/d6/d00/tutorial\_py\_root.html

22. hashlib [Электронный ресурс] / Python Docs [офиц. сайт] – Электрон. дан. – Режим доступа: <https://docs.python.org/3/library/hashlib.html>

23. Flask documentation [Электронный ресурс]/ Flask [офиц. сайт] – Электрон. дан. – Режим доступа: https://flask.palletsprojects.com/en/3.0.x/, свободный (дата обращения: 17.01.2024) – Загл. С экрана.

24. MVC [Электронный ресурс]/ Skillfactory media [офиц. сайт] – Электрон. дан. – Режим доступа: https://blog.skillfactory.ru/glossary/mvc/, свободный (дата обращения: 17.01.2024) – Загл. С экрана.

Приложение А  
(обязательное)  
Листинг программного кода 1

Объединение наборов данных

# создаем директорию для объединенного набора данных  
merging\_dataset\_path = 'datasets/merging\_dataset'  
os.makedirs(merging\_dataset\_path, exist\_ok=True)  
split\_group\_list = ['train', 'valid', 'test']  
  
for group in split\_group\_list:  
 os.makedirs(f'{merging\_dataset\_path}/{group}/images', exist\_ok=True)  
 os.makedirs(f'{merging\_dataset\_path}/{group}/labels', exist\_ok=True)  
   
train\_path = os.path.abspath(f'{merging\_dataset\_path}/train')  
val\_path = os.path.abspath(f'{merging\_dataset\_path}/valid')  
test\_path = os.path.abspath(f'{merging\_dataset\_path}/test')  
  
# разметка из Abandoned-Objects-detection-1   
merging\_dataset\_data = {'train': f'{train\_path}',  
 'val': f'{val\_path}',  
 'test': f'{test\_path}',  
   
 'nc': 4,  
  
 'names': ['Handbag', # 0  
 'Human-body', # 1  
 'Luggage-and-bags', # 2  
 'Wheelchair' # 3  
 ] # 3  
 }  
  
with open(f'{merging\_dataset\_path}/data.yaml', 'w') as file:  
 documents = yaml.dump(merging\_dataset\_data, file, sort\_keys=False)

def vizualize\_img(image\_path,   
 label\_path):  
 """  
 Визуализация изображения с рамками  
 указать путь к image и label  
 """  
   
 with open(label\_path, "r") as file:  
 labels = file.read().strip().split("\n")  
 # print(labels)  
 file.close()  
   
 image = cv2.imread(image\_path)  
   
 for label in labels:  
 if len(label.split()) != 5:  
 continue  
 class\_id, x\_center, y\_center, width, height = map(float, label.split())  
 x\_min = int((x\_center - width/2) \* image.shape[1])  
 y\_min = int((y\_center - height/2) \* image.shape[0])  
 x\_max = int((x\_center + width/2) \* image.shape[1])  
 y\_max = int((y\_center + height/2) \* image.shape[0])  
   
 cv2.rectangle(image, (x\_min, y\_min), (x\_max, y\_max), (0, 255, 0), 3)  
 cv2.putText(image, f'{class\_id}', (x\_min, y\_min-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (36,255,12), 2)  
   
 plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  
 plt.axis('off')  
 # print(image.shape)

def merge\_dataset(old\_dataset\_path,   
 merging\_dataset\_path,   
 map\_dict):  
 """  
 Указать path папок старого и нового датасетов, карту изменения классов  
 """  
   
 split\_group\_list = ['train', 'valid', 'test']  
   
 # копируем images и labels  
 for group in split\_group\_list:  
 copytree(f'{old\_dataset\_path}/{group}/images',   
 f'{merging\_dataset\_path}/{group}/images',dirs\_exist\_ok=True)  
   
 copytree(f'{old\_dataset\_path}/{group}/labels',   
 f'{merging\_dataset\_path}/{group}/labels',dirs\_exist\_ok=True)  
   
 # редактируем labels   
 for group in split\_group\_list:  
   
 # получаем список файлов label из old\_dataset  
 label\_file\_list = os.listdir(f'{old\_dataset\_path}/{group}/labels')  
   
 # каждый label\_file открывается в merging\_dataset\_path  
 for label\_file in label\_file\_list:  
   
 with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 for line in data:  
 # получаем индекс строки  
 line\_index = data.index(line)  
   
 class\_id, x\_center, y\_center, width, height = map(float, line.split())  
   
 # замена class\_id по map\_dict  
 class\_id = str(int(class\_id))  
 class\_id = class\_id.replace(class\_id,map\_dict[class\_id])  
   
 data[line\_index] = class\_id + f' {x\_center} {y\_center} {width} {height}\n'  
 new\_data = data.copy()  
   
 file.close()  
   
   
 with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'w') as file:  
 for new\_line in new\_data:  
 file.write(new\_line)  
   
 file.close()  
  
  
def merger\_check(img\_name,   
 old\_dataset\_path,   
 new\_dataset\_path):  
 """  
 Проверка merge\_dataset на train image  
 """  
   
 fig = plt.subplots(1, 2, figsize=(16,8))  
   
 image\_path1 = f"{old\_dataset\_path}/train/images/{img\_name}.jpg"  
 label\_path1 = f"{old\_dataset\_path}/train/labels/{img\_name}.txt"  
 plt.subplot(121).set\_title(old\_dataset\_path)  
 vizualize\_img(image\_path1, label\_path1)  
   
 image\_path2 = f"{new\_dataset\_path}/train/images/{img\_name}.jpg"  
 label\_path2 = f"{new\_dataset\_path}/train/labels/{img\_name}.txt"  
 plt.subplot(122).set\_title(new\_dataset\_path)  
 vizualize\_img(image\_path2, label\_path2)

def merge\_yolo\_darknet\_to\_yolo\_v8(old\_dataset\_path,   
 merging\_dataset\_path,   
 map\_dict):  
  
 split\_group\_list = ['train', 'valid', 'test']  
   
 # копируем images и labels в merging\_dataset   
 for group in split\_group\_list:  
   
 all\_group\_files = os.listdir(f'{old\_dataset\_path}/{group}')  
   
 for file in all\_group\_files:  
 if '.txt' in file:  
 copyfile(f'{old\_dataset\_path}/{group}/{file}',   
 f'{merging\_dataset\_path}/{group}/labels/{file}')  
   
 elif '.jpg' in file:  
 copyfile(f'{old\_dataset\_path}/{group}/{file}',   
 f'{merging\_dataset\_path}/{group}/images/{file}')  
   
 # редактируем labels merging\_dataset  
 for group in split\_group\_list:  
   
 # получаем список файлов label.txt  
 all\_group\_files = os.listdir(f'{old\_dataset\_path}/{group}')  
   
 label\_file\_list = []  
 for file in all\_group\_files:  
 if '.txt' in file:  
 label\_file\_list.append(file)  
   
 # каждый label\_file открывается в new\_dataset и изменяется class\_id  
 for label\_file in label\_file\_list:  
   
 with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 for line in data:  
 # получаем индекс строки  
 line\_index = data.index(line)  
   
 class\_id, x\_center, y\_center, width, height = map(float, line.split())  
   
 # замена class\_id по map\_dict  
 class\_id = str(int(class\_id))  
 class\_id = class\_id.replace(class\_id,map\_dict[class\_id])  
   
 data[line\_index] = class\_id + f' {x\_center} {y\_center} {width} {height}\n'  
 new\_data = data.copy()  
   
 file.close()   
   
   
 with open(f'{merging\_dataset\_path}/{group}/labels/{label\_file}', 'w') as file:  
 for new\_line in new\_data:  
 file.write(new\_line)  
   
 file.close()  
   
  
def merger\_check\_from\_yolo\_darknet(img\_name,   
 old\_dataset\_path,   
 new\_dataset\_path):  
 """  
 Проверка merge\_dataset на train image  
 """  
  
 fig = plt.subplots(1, 2, figsize=(16,8))  
   
 image\_path1 = f"{old\_dataset\_path}/train/{img\_name}.jpg"  
 label\_path1 = f"{old\_dataset\_path}/train/{img\_name}.txt"  
 plt.subplot(121).set\_title(old\_dataset\_path)  
 vizualize\_img(image\_path1, label\_path1)  
   
 image\_path2 = f"{new\_dataset\_path}/train/images/{img\_name}.jpg"  
 label\_path2 = f"{new\_dataset\_path}/train/labels/{img\_name}.txt"  
 plt.subplot(122).set\_title(new\_dataset\_path)  
 vizualize\_img(image\_path2, label\_path2)

"""  
1) Abandoned-Objects-detection-1   
4826 Total Images 416х416  
копируем набор данных  
"""  
  
dataset1\_path = 'datasets/Abandoned-Objects-detection-1'  
  
for group in split\_group\_list:  
 copytree(f'{dataset1\_path}/{group}/images',   
 f'{merging\_dataset\_path}/{group}/images',dirs\_exist\_ok=True)  
   
 copytree(f'{dataset1\_path}/{group}/labels',   
 f'{merging\_dataset\_path}/{group}/labels',dirs\_exist\_ok=True)

"""  
2) New-Backpacks-1   
1047 Total Images 416x416  
names:  
0 - luggage  
 изменяем class\_id: 0 -> 2  
  
3) Abandoned-Object-Detection-2   
272 Total Images 640x640  
names:  
0 - Bag  
 изменяем class\_id: 0 -> 2  
"""  
  
merging\_dataset\_path = 'datasets/merging\_dataset'  
map\_dict = {'0': '2'  
 }  
  
v8\_datasets = {'New-Backpacks-1': map\_dict,  
 'Abandoned-Object-Detection-2': map\_dict}  
  
for dataset, map\_dict in v8\_datasets.items():  
 old\_dataset\_path = f"datasets/{dataset}"  
 merge\_dataset(old\_dataset\_path, merging\_dataset\_path, map\_dict)

merging\_dataset\_path = 'datasets/merging\_dataset'  
  
#проверка New-Backpacks-1  
img\_name = 'dc\_luggage\_batch\_1\_9DQ7B8UA490CQ1GP\_jpg.rf.1995b8b81514be4a88e3cdea29ee60b5'  
merger\_check(img\_name, 'datasets/New-Backpacks-1', merging\_dataset\_path)  
  
#проверка Abandoned-Object-Detection-2  
img\_name = 'video1\_mp4-59\_jpg.rf.b609f0a1c597536648fb75b98686958a'  
merger\_check(img\_name, 'datasets/Abandoned-Object-Detection-2', merging\_dataset\_path)

"""  
4) Abandoned Bags.v1i.darknet   
2514 Total Images 640x640  
  
5) Abandoned Bags.v2i.darknet   
2271 Total Images 640x640  
  
6) Abandoned Bags.v3i.darknet   
2028 Total Images 640x640  
  
7) Abandoned Bags.v4i.darknet   
1434 Total Images 640x640  
  
names:  
 0: luggage  
 изменяем class\_id: 0 -> 2  
"""  
  
merging\_dataset\_path = 'datasets/merging\_dataset'  
map\_dict = {'0': '2'  
 }  
  
darknet\_datasets = {'Abandoned Bags.v1i.darknet': map\_dict,  
 'Abandoned Bags.v2i.darknet': map\_dict,  
 'Abandoned Bags.v3i.darknet': map\_dict,  
 'Abandoned Bags.v4i.darknet': map\_dict}  
  
for dataset, map\_dict in darknet\_datasets.items():  
 old\_dataset\_path = f"datasets/{dataset}"  
 merge\_yolo\_darknet\_to\_yolo\_v8(old\_dataset\_path, merging\_dataset\_path, map\_dict)

merging\_dataset\_path = 'datasets/merging\_dataset'  
  
# проверка Abandoned Bags.v1i.darknet  
img\_name = 'dc\_luggage\_batch\_1\_00R5A4HYKR4PFHS7\_jpg.rf.1698aba690375ba759352fdac605b6c9'  
merger\_check\_from\_yolo\_darknet(img\_name, 'datasets/Abandoned Bags.v1i.darknet', merging\_dataset\_path)  
  
# проверка Abandoned Bags.v2i.darknet  
img\_name = 'dc\_luggage\_batch\_1\_0CJ800W956Y4TUC1\_jpg.rf.1cc04af4dc58b29785885913efdfd9b7'  
merger\_check\_from\_yolo\_darknet(img\_name, 'datasets/Abandoned Bags.v2i.darknet', merging\_dataset\_path)  
  
# проверка Abandoned Bags.v3i.darknet  
img\_name = 'abandonedbag\_2\_mp4-5\_jpg.rf.5670fd536906715a7db5c724139c4632'  
merger\_check\_from\_yolo\_darknet(img\_name, 'datasets/Abandoned Bags.v3i.darknet', merging\_dataset\_path)  
  
# проверка Abandoned Bags.v4i.darknet  
img\_name = 'dc\_luggage\_batch\_1\_SVPOM9V1XL2O8037\_jpg.rf.101512ab83a8f853e3c5f3e1a10064b3'  
merger\_check\_from\_yolo\_darknet(img\_name, 'datasets/Abandoned Bags.v4i.darknet', merging\_dataset\_path)

# создаем директорию для балансировки набора данных  
merging\_dataset\_path = 'datasets/full\_data'  
  
os.makedirs(merging\_dataset\_path, exist\_ok=True)  
  
os.makedirs(f'{merging\_dataset\_path}/images', exist\_ok=True)  
os.makedirs(f'{merging\_dataset\_path}/labels', exist\_ok=True)

# копируем данные  
new\_dataset\_path = 'datasets/full\_data'  
old\_dataset\_path = 'datasets/merging\_dataset'  
  
split\_group\_list = ['train', 'valid', 'test']  
for group in split\_group\_list:  
 copytree(f'{old\_dataset\_path}/{group}/images',   
 f'{new\_dataset\_path}/images',dirs\_exist\_ok=True)  
   
 copytree(f'{old\_dataset\_path}/{group}/labels',   
 f'{new\_dataset\_path}/labels',dirs\_exist\_ok=True)

def duplicate\_removal(dataset\_path):  
 """  
 Проверка на дубликаты по всем изображениям  
 """  
 directory = f'{dataset\_path}/images'  
   
 hashes = set()  
   
 del\_num = 0  
 # в датасете есть дубликаты с другим именем изображения  
 for filename in os.listdir(directory):  
 path = os.path.join(directory, filename)  
   
 #шифруем image  
 digest = hashlib.sha1(open(path,'rb').read()).digest()  
   
 if digest not in hashes:  
 # если не дубликат, то добавляем в hashes  
 hashes.add(digest)  
   
 else:  
 # удаляем image  
 os.remove(path)  
   
 label\_name = filename.replace('.jpg','.txt')  
 label\_path = f'{dataset\_path}/labels/{label\_name}'  
 # удаляем label  
 os.remove(label\_path)  
   
 print(f'{path}\n{label\_path}\n\n')  
 del\_num += 1  
  
 print(f'Удалено {del\_num} дубликатов изображений')

def number\_of\_objects\_by\_class(labels\_path):  
 """  
 Общее количество объектов по классам в директории  
 """  
   
 # получаем список файлов label  
 label\_file\_list = os.listdir(labels\_path)  
   
 # 0, 1  
 num\_Handbag, num\_Human\_body = 0, 0   
 # 2 ,3   
 num\_Luggage\_and\_bags, num\_Wheelchair = 0, 0  
   
 for label\_file in label\_file\_list:  
   
 with open(f'{labels\_path}/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 for line in data:  
   
 class\_id = int(line[0])  
   
 if class\_id == 0:  
 num\_Handbag += 1  
   
 elif class\_id == 1:  
 num\_Human\_body += 1  
   
 elif class\_id == 2:  
 num\_Luggage\_and\_bags += 1  
   
 elif class\_id == 3:  
 num\_Wheelchair += 1  
   
 file.close()  
   
 print(f"{os.path.dirname(labels\_path)}\n"  
 f"Handbag: {num\_Handbag}\n"  
 f"Human-body: {num\_Human\_body}\n"  
 f"Luggage-and-bags: {num\_Luggage\_and\_bags}\n"  
 f"Wheelchair: {num\_Wheelchair}\n")

number\_of\_objects\_by\_class('datasets/full\_data/labels')

labels\_path = 'datasets/full\_data/labels'  
label\_file\_list = os.listdir(labels\_path)

"""  
Добавляем по классам от наименьшего   
общего количества объектов данного класса  
"""  
  
Wheelchair\_files = []  
Handbag\_files = []  
Human\_body\_files = []  
  
for label\_file in label\_file\_list:  
   
 with open(f'{labels\_path}/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 class\_list = [int(line[0]) for line in data]  
   
 if 3 in class\_list:  
 Wheelchair\_files.append(label\_file)  
 label\_file\_list.remove(label\_file)  
   
 file.close()  
   
for label\_file in label\_file\_list:  
   
 with open(f'{labels\_path}/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 class\_list = [int(line[0]) for line in data]  
   
 if 0 in class\_list:  
 Handbag\_files.append(label\_file)  
 label\_file\_list.remove(label\_file)  
   
 file.close()  
   
for label\_file in label\_file\_list:  
   
 with open(f'{labels\_path}/{label\_file}', 'r') as file:   
 data = file.read().splitlines(True)  
   
 class\_list = [int(line[0]) for line in data]  
   
 if 1 in class\_list:  
 Human\_body\_files.append(label\_file)  
 label\_file\_list.remove(label\_file)  
   
 file.close()  
  
Luggage\_and\_bags\_files = label\_file\_list

def class\_dataset\_split(classes\_files):  
 # train, val, test = 80%, 10%, 10%  
 test\_val\_num = int(len(classes\_files) \* 0.1)  
 train\_num = len(classes\_files) - test\_val\_num \* 2  
   
 # добавляет train\_num случайных файлов в train\_files  
 train\_files.extend(random.sample(classes\_files, train\_num))  
   
 # список файлов которых нет в train\_files   
 interim\_list = [file for file in classes\_files if (file not in train\_files)]  
   
 # добавляет test\_val\_num случайных файлов в val\_files  
 val\_files.extend(random.sample(interim\_list, test\_val\_num))  
   
 # добавляет оставшиеся файлы в test\_files  
 test\_files.extend([file for file in interim\_list if (file not in val\_files)])  
   
train\_files, val\_files, test\_files = [], [], []  
  
all\_files = [Wheelchair\_files,   
 Handbag\_files,   
 Human\_body\_files,   
 Luggage\_and\_bags\_files]  
  
for classes\_files in all\_files:  
 class\_dataset\_split(classes\_files)

# создаем директорию для нового набора данных  
merging\_dataset\_path = 'datasets/AOD\_dataset'  
  
os.makedirs(merging\_dataset\_path, exist\_ok=True)  
  
split\_group\_list = ['train', 'valid', 'test']  
  
for group in split\_group\_list:  
 os.makedirs(f'{merging\_dataset\_path}/{group}/images', exist\_ok=True)  
 os.makedirs(f'{merging\_dataset\_path}/{group}/labels', exist\_ok=True)  
  
train\_path = os.path.abspath(f'{merging\_dataset\_path}/train')  
val\_path = os.path.abspath(f'{merging\_dataset\_path}/valid')  
test\_path = os.path.abspath(f'{merging\_dataset\_path}/test')  
  
# за основу взяты class из Abandoned-Objects-detection-1   
merging\_dataset\_data = {'train': f'{train\_path}',  
 'val': f'{val\_path}',  
 'test': f'{test\_path}',  
   
 'nc': 4,  
  
 'names': ['Handbag', # 0  
 'Human-body', # 1  
 'Luggage-and-bags', # 2  
 'Wheelchair' # 3  
 ] # 3  
 }  
  
with open(f'{merging\_dataset\_path}/data.yaml', 'w') as file:  
 documents = yaml.dump(merging\_dataset\_data, file, sort\_keys=False)

# копируем данные  
merging\_dataset\_path = 'datasets/AOD\_dataset'  
old\_dataset\_path = 'datasets/full\_data'  
  
split\_group\_list = ['train', 'valid', 'test']  
grop\_file\_list = [train\_files, val\_files, test\_files]  
  
for index, group in enumerate(split\_group\_list):  
   
 group\_files = grop\_file\_list[index]  
   
 for file in group\_files:  
 copyfile(f'{old\_dataset\_path}/labels/{file}',   
 f'{merging\_dataset\_path}/{group}/labels/{file}')  
   
 image\_name = file.replace('.txt','.jpg')  
 copyfile(f'{old\_dataset\_path}/images/{image\_name}',   
 f'{merging\_dataset\_path}/{group}/images/{image\_name}')

Приложение Б  
(обязательное)  
Листинг программного кода 2

Вывод результатов обучения

import gdown

url = ''

output\_path = 'data.zip'

gdown.download(url, output\_path, quiet=False,fuzzy=True)

!unzip data.zip

import torch

torch.cuda.is\_available()

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

project = 'YOLO\_results'

name = 'yolov8n\_colab\_gpu'

data\_path = '/content/AOD\_dataset/data.yaml'model.train(

data = data\_path,

epochs = 30,

batch = -1, # AutoBatch

imgsz = 640,

project = 'YOLO\_results',

name = 'yolov8n\_gpu',

seed = 42

)

import locale

locale.getpreferredencoding = lambda: "UTF-8"

!zip -r results.zip /content/YOLO\_results

from google.colab import files

files.download('results.zip')

Приложение В  
(обязательное)  
Листинг программного кода 3

Код приложения

from flask import Flask, render\_template, Response, jsonify

from app.auth import auth

from app.camera import Camera

app = Flask(\_\_name\_\_)

camera = Camera()

@app.route('/video\_feed')

def video\_feed():

return Response(camera.gen\_frames\_2(),

mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/')

@auth.login\_required

def index():

return render\_template('index.html')

@app.route('/objects-count')

def objects\_count():

person\_count, things\_count, left\_things\_count = camera.get\_count()

return jsonify({

'person\_count': person\_count,

'things\_count': things\_count,

'left\_things\_count': left\_things\_count

})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='127.0.0.1', port=8000, debug=True)

# todo: save logpass to db in future

from flask\_httpauth import HTTPBasicAuth

from werkzeug.security import check\_password\_hash, generate\_password\_hash

from app.db import engine

from app.models import User

auth = HTTPBasicAuth()

users = {

"artem": generate\_password\_hash("password"),

"vlad": generate\_password\_hash("password")

}

@auth.verify\_password

def verify\_password(username, password):

# with Session(engine) as db:

# user = db.query(User).filter\_by(username=username).first()

# if not user:

# return

# if not check\_password\_hash(user.password\_hash, password):

# return

if username in users and \

check\_password\_hash(users.get(username), password):

return username

import math

import random

import cv2

import numpy as np

from ultralytics import YOLO

from app import settings

class Camera:

def \_\_init\_\_(self):

self.camera = cv2.VideoCapture(settings.camera)

self.model = YOLO(settings.path\_model)

self.item\_person\_vector = dict()

def get\_count(self):

person\_count = 0

items\_count = len(self.item\_person\_vector)

left\_items\_count = sum(1 for v in self.item\_person\_vector.values() if v is None)

return person\_count, items\_count, left\_items\_count

def gen\_frames(self):

while True:

success, frame = self.camera.read()

if not success:

break

else:

person\_count = 0

laptop\_count = 0

results = self.model(frame, stream=True)

for r in results:

boxes = r.boxes

for box in boxes:

# bounding box

x1, y1, x2, y2 = box.xyxy[0]

x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # convert to int values

# put box in cam

cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 3)

# confidence

confidence = math.ceil((box.conf[0] \* 100)) / 100

# print("Confidence --->", confidence)

# class name

cls = int(box.cls[0])

if cls == 0:

person\_count += 1

elif cls == 1:

laptop\_count += 1

# print("Class name -->", settings.class\_names[cls])

# object details

org = [x1, y1]

font = cv2.FONT\_HERSHEY\_SIMPLEX

fontScale = 1

color = (255, 0, 0)

thickness = 2

print(f'PERSONS: {person\_count}, LAPTOPS: {laptop\_count}')

cv2.putText(frame, settings.class\_names[cls], org, font, fontScale, color, thickness)

ret, buffer = cv2.imencode('.jpg', frame)

frame = buffer.tobytes()

yield (b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

def gen\_frames\_2(self):

colors = [(random.randint(0, 255), random.randint(0, 255),

random.randint(0, 255)) for j in range(20)]

# Высота и ширина видео файла

cap\_height = self.camera.get(cv2.CAP\_PROP\_FRAME\_HEIGHT).\_\_int\_\_()

cap\_width = self.camera.get(cv2.CAP\_PROP\_FRAME\_WIDTH).\_\_int\_\_()

# Разрешение изображения должно быть кратно сетевому шагу 32 для YOLOv8. Иначе warning и авто корректировка

cap\_height = round(cap\_height / 32) \* 32

cap\_width = round(cap\_width / 32) \* 32

# дистанция для определения владельца предмета

distance\_threshold = 405.0

# {item\_id: person\_id}

item\_person\_association = dict()

while self.camera.isOpened():

# Читаем кадр из видео

success, frame = self.camera.read()

if not success:

break

results = self.model.track(

frame, persist=True, # Сохранение идентификаторов ранее обнаруженных объектов

imgsz=(cap\_height, cap\_width), # Размер изображения для вывода

# tracker='botsort\_custom.yaml'

)

# Получение объекта Boxes

boxes = results[0].boxes

# Если на кадре есть объекты

if boxes.id is not None:

class\_dict = results[0].names # словарь классов в model

objs\_box = boxes.xywh.cpu() # боксы объектов в кадре

list\_objs\_id = boxes.id.int().cpu().tolist() # id объектов в кадре

objs\_cls = boxes.cls.int().cpu().tolist() # классы объектов в кадре

objs\_scores = boxes.conf.tolist() # score объектов в кадре

# Получаем списки индексов предметов и людей на текущем кадре

items\_class = ['Handbag', 'Luggage-and-bags', 'Wheelchair',

'backpack', 'handbag', 'suitcase', 'bird'] # stock yolo class

person\_class = ['Human-body', 'person'] # stock yolo class

item\_index\_list = [index for index, value in enumerate(objs\_cls) if

class\_dict[value] in items\_class]

person\_index\_list = [index for index, value in enumerate(objs\_cls) if

class\_dict[value] in person\_class]

'''

Сопоставление предмет-человек для всех предметов в кадре

'''

for item\_index in item\_index\_list:

item\_id = list\_objs\_id[item\_index]

# если для предмета не определен владелец

if item\_id not in item\_person\_association:

# центр бокса предмета [item\_x, item\_y]

item\_center\_point = [float(i) for i in objs\_box[item\_index][:2]]

# определяем владельца

for person\_index in person\_index\_list:

# центр бокса человека [person\_x, person\_y]

person\_center\_point = [float(i) for i in objs\_box[person\_index][:2]]

vector\_between\_centers\_boxes = np.array([item\_center\_point,

person\_center\_point],

np.int32).reshape((-1, 1, 2))

person\_center\_point\_with\_coef = [person\_center\_point[0] \*\* 2 / cap\_width,

person\_center\_point[1] \*\* 2 / cap\_height]

'''Евклидово расстояние-это норма L2 вектора

функция np.linalg.norm() по умолчанию использует ord=None (for vectors 2-norm)'''

distance\_between\_boxes = np.linalg.norm(np.array([item\_center\_point,

person\_center\_point\_with\_coef],

np.int32).reshape((-1, 1, 2)))

# если расстояние между боксами удовлетворяет условию определяем человека как владельца предмета

if distance\_between\_boxes < distance\_threshold:

item\_person\_association[item\_id] = list\_objs\_id[

person\_index] # {item\_id: person\_id}

self.item\_person\_vector[item\_id] = vector\_between\_centers\_boxes # {item\_id: vector}

# если расстояние не удовлетворяет условию и нет владельца - владелец и вектор None

else:

if item\_id not in item\_person\_association:

item\_person\_association[item\_id] = None # {item\_id: person\_id}

self.item\_person\_vector[item\_id] = None # {item\_id: vector}

# если для предмета определен владелец

else:

person\_id = item\_person\_association[item\_id]

# если владелец в кадре - обновляем значение вектора

if person\_id in list\_objs\_id:

person\_index = list\_objs\_id.index(person\_id)

# центр бокса предмета [item\_x, item\_y]

item\_center\_point = [float(i) for i in objs\_box[item\_index][:2]]

# центр бокса человека [person\_x, person\_y]

person\_center\_point = [float(i) for i in objs\_box[person\_index][:2]]

vector\_between\_centers\_boxes = np.array([item\_center\_point,

person\_center\_point],

np.int32).reshape((-1, 1, 2))

self.item\_person\_vector[item\_id] = vector\_between\_centers\_boxes # {item\_id: vector}

# если владелец не в кадре - вектор None

else:

# {item\_id: vector}

self.item\_person\_vector[item\_id] = None

'''если предмет не в кадре - вектор None'''

for item\_id in item\_person\_association:

if item\_id not in list\_objs\_id:

# {item\_id: vector}

self.item\_person\_vector[item\_id] = None

'''Визуализация на кадр'''

# Отображение связи предмет-человек

cv2.polylines(frame,

[vector for vector in self.item\_person\_vector.values()],

isClosed=True, color=(0, 0, 255), # BGR red

thickness=5)

# надпись "оставлено" для объектов

for item\_id in item\_person\_association.keys():

# если объект в кадре

if item\_id in list\_objs\_id:

item\_index = list\_objs\_id.index(item\_id)

try:

# item\_person\_vector = {item\_id: vector}

item\_vector = self.item\_person\_vector[item\_id]

# если вектор None

if item\_vector is None:

x, y, w, h = [int(i) for i in objs\_box[item\_index]]

text\_item = 'leave'

cv2.rectangle(frame,

(x - w // 2, y - 13),

(x + w // 2 + len(text\_item) + 1, y + 2),

(0, 0, 0), # черный цвет

-1)

cv2.putText(frame, text\_item, (x - w // 2 + 5, y),

cv2.FONT\_HERSHEY\_PLAIN, 1, (0, 0, 255), 2)

except (KeyError, ValueError):

continue

# Для каждого объекта в кадре

for obj\_id in list\_objs\_id:

# Данные объекта

obj\_index = list\_objs\_id.index(obj\_id)

x, y, w, h = [int(i) for i in objs\_box[obj\_index]]

# Отображение бокса объекта

cv2.rectangle(frame,

(x - w // 2, y - h // 2), # верхний левый угол

(x + w // 2, y + h // 2), # нижний правый угол

(colors[obj\_index % len(colors)]), # уникальный цвет по индексу

3)

ret, buffer = cv2.imencode('.jpg', frame)

frame = buffer.tobytes()

yield (b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')