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 / филиала / института / колледжа)

**КУРСОВОЙ ПРОЕКТ**

|  |  |  |
| --- | --- | --- |
| По модулю | | Разработка AI-проекта |
|  | | |
|  | | |
| На тему | Мониторинг перемещения мебели и техники в общественных учреждениях с помощью технологий глубокого обучения | |
|  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ф.И.О.  обучающихся | Наименование направления подготовки / специальности | Курс | Группа | Ф.И.О. руководителя (-ей) должность / уч. степень / звание |
| Архаров Н. М. | 09.03.02 Информационные системы и технологии | 4 | 351018 | Васендина И.С., доцент, к.т.н. |
| Голышев А. В. | 09.03.02 Информационные системы и технологии | 4 | 351018 | Васендина И.С., доцент, к.т.н. |

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

Архангельск 2024

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 / филиала / института / колледжа) | | | | | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | | | | | |  | |
|  | **ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ** | | | | | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | | | | | |  | |
|  | ппо | Разработка AI-проекта | | | | | | | | | | | | | | | | | |  | |
|  |  | | | | (наименование дисциплины) | | | | | | | | | | | | | | |  | |
|  | студенту | | ВШИТиАС | | высшей школы | | | | 4 | | | | | курса | 351018 | | | | группы |  | |
|  | Архарову Никите Михайловичу, Голышеву Алексею Витальевичу | | | | | | | | | | | | | | | | | | |  | |
|  | (фамилия, имя, отчество студента) | | | | | | | | | | | | | | | | | | |  | |
|  | |  | | --- | | 09.03.02 «Информационные системы и технологии» | | (код и наименование направления подготовки/специальности) | | | | | | | | | | | | | | | | | | | |  | |
|  | ТЕМА: | | | Решение прикладной задачи мониторинга перемещения мебели и техники | | | | | | | | | | | | | | | |  | |
|  | в общественных учреждениях с помощью технологии глубокого обучения | | | | | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | | | | | |  | |
|  | ИСХОДНЫЕ ДАННЫЕ: | | | | | | | Необходимо решить задачу мониторинга | | | | | | | | | | | |  | |
|  | перемещения мебели и техники в общественных учреждениях с помощью технологии | | | | | | | | | | | | | | | | | | |  | |
|  | глубокого обучения. Описать актуальность, цель, задачи, описать набор данных или | | | | | | | | | | | | | | | | | | |  | |
|  | как собирали, сформулировать задачу, проанализировать целевую переменную, | | | | | | | | | | | | | | | | | | |  | |
|  | провести предобработку изображений (если требуется), построить несколько моделей | | | | | | | | | | | | | | | | | | |  | |
|  | по архитектуре (или с разными параметрами) для решения задачи. Обучить модели. | | | | | | | | | | | | | | | | | | |  | |
|  | Подобрать параметры. Оценить качество работы моделей. Обосновать выбор модели. | | | | | | | | | | | | | | | | | | |  | |
|  | Проверить работу модели на тестовых данных. Определить требования к | | | | | | | | | | | | | | | | | | |  | |
|  | пользовательскому интерфейсу приложения. Спроектировать базу данных. | | | | | | | | | | | | | | | | | | |  | |
|  | Реализовать логирование работы модели в базу данных. Реализовать обработку | | | | | | | | | | | | | | | | | | |  | |
|  | данных о работе модели из базы данных и отображение нужных данных в | | | | | | | | | | | | | | | | | | |  | |
|  | пользовательском интерфейсе. Сделать выводы о качестве работы. | | | | | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | | | | | |  | |
|  | Срок выполнения с 19.12.2023 г. по 19.01.2024 г. | | | | | | | | | | | | | | | | | | |  | |
|  | Руководитель работы | | | | | | доцент | | | |  | |  | | | |  | И.С. Васендина | |  | |
|  |  | | | |  | (должность) | | | |  | | (подпись) | | | |  | | (инициалы, фамилия) | | |

Архангельск 2024

Лист для замечаний

Оглавление

[Введение 6](#_Toc161249999)

[1 Описание предметной области 8](#_Toc161250000)

[2 Анализ вариантов и проектирование решения задачи 10](#_Toc161250001)

[3 Сбор данных 13](#_Toc161250002)

[3.1 Основной набор данных 13](#_Toc161250003)

[3.2 Добавочный набор данных 15](#_Toc161250004)

[3.3 Аугментация данных 17](#_Toc161250005)

[3.3.1 Аугментация основного набора данных 17](#_Toc161250006)

[3.3.2 Аугментация добавочного набора данных 18](#_Toc161250007)

[3.3.3 Результаты аугментации 20](#_Toc161250008)

[4 Обучение модели YOLOv8 22](#_Toc161250009)

[4.1 Сравнение различных предобученных моделей 22](#_Toc161250010)

[4.2 Обучение модели 23](#_Toc161250011)

[4.3 Результаты обучения 23](#_Toc161250012)

[5 Внедрение модели в алгоритм отслеживания объектов 25](#_Toc161250013)

[5.1 Описание функционала 25](#_Toc161250014)

[5.2 Реализация функционала 26](#_Toc161250015)

[6 Проектирование базы данных и архитектуры приложения 30](#_Toc161250016)

[6.1 Определение требований для базы данных 30](#_Toc161250017)

[6.2 Entity-диаграмма 30](#_Toc161250018)

[6.3 Моделирование классов 31](#_Toc161250019)

[6.4 Архитектура приложения 32](#_Toc161250020)

[7 Контейнеризация приложения 35](#_Toc161250021)

[7.1 Определение контейнеризации 35](#_Toc161250022)

[7.2 Преимущества контейнеризации 35](#_Toc161250023)

[7.3 Основные технологии контейнеризации 35](#_Toc161250024)

[7.3.1 Docker 35](#_Toc161250025)

[7.4 Процесс контейнеризации приложения 35](#_Toc161250026)

[7.4.1 Использование готового образа Ubuntu 22.4: 36](#_Toc161250027)

[7.4.2 Копирование файлов проекта внутрь контейнера: 36](#_Toc161250028)

[7.4.3 Установка зависимостей из requirements.txt: 36](#_Toc161250029)

[Заключение 37](#_Toc161250030)

[Список используемых источников 40](#_Toc161250031)

[Приложение А (обязательное) Листинг 41](#_Toc161250032)

Введение

В настоящее время технологии компьютерного зрения активно применяются для решения различных задач, связанных с анализом и интерпретацией визуальных данных. Одним из перспективных направлений использования этих технологий является мониторинг и учет имущества на различных объектах. Это особенно актуально в общественных учреждениях, таких как школы, больницы, офисы и др., где присутствует большое количество ценного имущества, требующего постоянного отслеживания и предотвращения возможных хищений. Ручной учет и мониторинг данных процессов обычно связаны с значительными трудозатратами и не гарантируют полной точности и достоверности.

В рамках данной работы предлагается использовать методы компьютерного зрения для автоматизации процессов мониторинга перемещения мебели и техники на территории общественных учреждений. Целью исследования является разработка алгоритмов компьютерного зрения, способных эффективно отслеживать предметы мебели в общественных учреждениях.

Для достижения поставленной цели предполагается решить следующие задачи:

- анализ предметной области: изучение существующих методов для задач распознавания и отслеживания объектов;

- формирование обучающей выборки на основе видеоданных;

- обучение нейросетевой модели для распознавания классов объектов;

- разработка алгоритма отслеживания объектов;

- тестирование разработанных алгоритмов на контрольных данных.

Практическая значимость работы заключается в разработке модели для автоматизированной системы мониторинга имущества для общественных учреждений на основе передовых алгоритмов компьютерного зрения. Реализация поставленных задач позволит сократить трудозатраты на учет имущества, повысить точность и оперативность мониторинга, используя передовые технологии искусственного интеллекта. Полученные результаты могут быть применены для решения практических задач оптимизации учета и сохранности имущества в общественных учреждениях.

**1 Описание предметной области**

В общественных учреждениях, таких как школы, больницы, офисы и т.д., обычно присутствует большое количество ценного имущества, которое требует постоянного контроля и отслеживания. Однако учёт и мониторинг таких процессов могут быть трудоемкими и неточными, если осуществляются вручную. В качестве решения этой проблемы предлагается автоматизация этих процессов с помощью компьютерного зрения.

Целью настоящей работы является разработка модели компьютерного зрения для автоматизации процессов мониторинга перемещения мебели и техники в общественных учреждениях. В настоящее время учёт мебели и техники в большинстве случаев осуществляется вручную с помощью различных документов, таких как инвентарные описи, журналы учета и др. Этот подход может быть неточным и трудоемким, особенно если требуется отследить перемещение объектов в реальном времени.

Коммерческие решения для мониторинга перемещения имущества, такие как системы видеоаналитики от Hikvision, Dahua, Axis и других вендоров, могут быть достаточно дорогими и не всегда эффективными в решении поставленной задачи. С другой стороны, существует множество открытых библиотек компьютерного зрения, таких как OpenCV, Detectron2, MMTracking, которые могут быть использованы для построения системы мониторинга перемещения мебели и техники. Однако эти библиотеки могут быть довольно сложными в использовании и требовать значительных усилий для адаптации под конкретную задачу.

Таким образом, разработка специализированной системы мониторинга перемещения мебели и техники на основе компьютерного зрения может значительно повысить эффективность и точность учета имущества в общественных учреждениях. Кроме того, такая система может быть более доступной по цене, чем коммерческие решения, и более гибкой в настройке и использовании, чем открытые библиотеки компьютерного зрения.

**2 Анализ вариантов и проектирование решения задачи**

Для решения поставленной задачи по мониторингу перемещения объектов в помещении можно рассмотреть несколько основных подходов. Во-первых, можно использовать классические алгоритмы компьютерного зрения, такие как HOG + SVM или Haar-каскады, для детектирования объектов. Однако эти подходы обладают ограниченной точностью и неустойчивы к изменению условий съемки.

Во-вторых, можно применить современные архитектуры глубокого обучения, такие как YOLO, SSD или Faster R-CNN. Эти архитектуры демонстрируют высокую точность и скорость работы. В частности, YOLOv8 является одной из самых перспективных архитектур для решения задачи детектирования объектов.

В-третьих, можно использовать двухэтапный подход, состоящий из детектирования и трекинга объектов. На первом этапе выполняется распознавание объектов в кадре при помощи нейросетевой модели, а на втором этапе применяется алгоритм трекинга, например, SORT, DeepSORT или Bytetrack. Bytetrack является эффективным и точным инструментом для трекинга объектов, способным правильно определять идентификаторы объектов даже в сложных ситуациях, когда объекты пересекаются или частично закрыты другими объектами.

В-четвертых, можно применить одноэтапный подход с использованием end-to-end моделей типа Tracktor, которые объединяют детектирование и трекинг объектов в одной модели.

Среди перечисленных подходов наиболее перспективным представляется использование двухэтапного pipeline с применением современной нейросетевой архитектуры на этапе детектирования и алгоритма Bytetrack для трекинга. Такой подход позволяет объединить высокую точность современных нейросетей в задаче детектирования и эффективные алгоритмы отслеживания объектов между кадрами.

В качестве обучающей выборки мы планируем использовать видеоданные, снятые в различных общественных учреждениях, которые будут аннотированы вручную для определения классов объектов и их расположения на каждом кадре видео. Затем мы будем использовать эти данные для обучения нейросетевой модели, которая будет способна распознавать классы объектов на новых кадрах видео.

Мы планируем протестировать различные предобученные модели, такие как YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x, чтобы определить, какая модель лучше подходит для данной задачи.

После обучения нейросетевой модели мы планируем разработать алгоритм отслеживания объектов на основе этой модели. Алгоритм будет включать в себя различные техники для отслеживания движущихся объектов, такие как оптический поток, механизм Кальмана и частичное следование. Мы также планируем использовать методы восстановления трека, чтобы предотвратить потерю объектов и обеспечить непрерывное отслеживание.

Наконец, мы планируем провести тестирование разработанных алгоритмов на контрольных данных, чтобы оценить их точность и эффективность.

На рисунке 1 можно увидеть этапы разработки трекера.



Рисунок 1 – Этапы разработки трекера

В целом, результаты исследования могут иметь важное значение для автоматизации процессов мониторинга и учета имущества в общественных учреждениях. Разработанные алгоритмы могут быть использованы для обеспечения более точного и эффективного учета имущества, а также для предотвращения потерь и хищений. Кроме того, результаты исследования могут быть использованы для разработки более совершенных систем компьютерного зрения, которые могут быть применены в различных областях, таких как безопасность, транспорт и медицина.

Таким образом, на основании проведенного анализа в качестве архитектуры решения предлагается использовать двухэтапный подход с применением YOLOv8 и Bytetrack для наиболее эффективного решения задачи мониторинга объектов.

**3 Сбор данных**

В ходе исследования были использованы два набора данных: основной и дополнительный. Основной набор данных был применен для разработки и обучения модели, в то время как дополнительный набор был использован для увеличения объема данных и улучшения ее общей эффективности.

3.1 Основной набор данных

В основном наборе данных использовались 1858 изображений, собранных с камер видеонаблюдения, установленных в учебном заведении Сибирского федерального университета (САФУ), в частности, в корпусе IT-парка. Из этих изображений 200 были выделены для валидации результатов модели. Примеры изображений показаны на рисунках 2, 3.



Рисунок 2 – Пример изображения хола из основного набора данных



Рисунок 3 – Пример изображения аудитории из основного набора данных

Набор данных содержит разнообразные классы объектов, представляющих интерьер офисных помещений, такие как:

- стул;

- стол;

- человек;

- монитор;

- ноутбук;

- клавиатура;

- компьютер;

- интерактивная доска.

Каждый класс обладает уникальными характеристиками, формой и расположением в пространстве, что предоставляет модели разнообразные сценарии для обучения и тестирования.

Разметка данных осуществлялась с помощью инструмента Label Studio, развернутого локально, обеспечивающего точную разметку объектов на изображениях. Для каждого объекта сохранены координаты ограничивающего прямоугольника, определяющего положение объекта на изображении. Этот подход к разметке данных обеспечивает модели необходимую информацию для эффективного обучения и последующего распознавания объектов на видеозаписях.

Важно отметить, что изображения были получены в реальных условиях с использованием систем видеонаблюдения, что делает этот набор данных особенно значимым для разработки и тестирования алгоритмов компьютерного зрения в условиях учебных офисных пространств.

3.2 Добавочный набор данных

Для увеличения объема данных и повышения эффективности модели использовался дополнительный набор данных, полученный из ресурса Roboflow. Этот набор данных, известный как «nva Dataset» [6], представляет собой разнообразные изображения, содержащие объекты, характерные для рабочих пространств. Он является открытым исходным набором данных, доступным для использования в области компьютерного зрения.

Исследование использовало описание набора данных, представленное автором Gerrux на Roboflow Universe. Согласно описанию, набор данных включает следующие классы объектов:

- кресло (777 изображений);

- интерактивная доска (95 изображений);

- клавиатура (257 изображений);

- ноутбук (375 изображений);

- монитор (372 изображения);

- системный блок (410 изображений);

- человек (450 изображений);

- стол (242 изображения).

Общее количество изображений в данном наборе данных составляет 2834, что обеспечивает достаточный объем данных для обучения и тестирования моделей компьютерного зрения. Пример изображения из этого набора можно увидеть на рисунке 4.



Рисунок 4 – Пример изображения из добавочного набора данных

Этот набор данных предоставляет разнообразные сценарии, включая в себя рабочие столы, оборудование, а также присутствие людей в офисной среде. Изображения были предоставлены с разметкой, что позволяет эффективно использовать их для обучения алгоритмов распознавания объектов. Данный дополнительный набор значительно расширяет контекст исследования, обогащая его разнообразием сценариев, связанных с офисной обстановкой и техническим оборудованием.

3.3 Аугментация данных

Для улучшения эффективности обучения нейронной сети и повышения ее обобщающей способности было решено применить методы аугментации данных. Целью аугментации является увеличение разнообразия тренировочных данных, что помогает модели лучше обобщать информацию и повышать ее устойчивость к различным вариациям входных данных.

3.3.1 Аугментация основного набора данных

Основной набор данных, состоящий из 1858 изображений, был подвергнут аугментации с помощью комплекса операций. Это позволило обогатить и улучшить разнообразие тренировочного набора данных.

Во-первых, была применена горизонтальная флип-аугментация с помощью оператора A.HorizontalFlip(p=1). Этот оператор создает зеркальные вариации изображений, добавляя различные перспективы объектов и увеличивая разнообразие данных.

Затем была использована операция A.RandomBrightnessContrast(p=0.5), которая воздействует на яркость и контрастность изображений. Эта аугментация добавляет вариации в условия освещения, что помогает модели эффективнее обучаться при различных уровнях яркости.

Далее была осуществлена аугментация цвета при помощи оператора A.RGBShift(r\_shift\_limit=30, g\_shift\_limit=30, b\_shift\_limit=30, p=0.5). Этот оператор изменяет цветовые характеристики изображений, создавая дополнительные вариации и помогая модели лучше обобщать информацию.

Кроме того, для внесения дополнительной сложности и разнообразия в тренировочный набор данных были применены операторы A.ISONoise и A.Cutout. Оператор A.ISONoise внедряет шум в изображения, а A.Cutout выполняет случайное вырезание прямоугольных областей из изображений.

В результате этих последовательных аугментаций тренировочный набор данных значительно увеличился, достигнув объема в 3196 изображений. Этот расширенный объем данных позволяет модели более эффективно обучаться на разнообразных входных данных, повышая ее способность к точному и устойчивому распознаванию объектов в различных учебных офисных условиях. Пример результата аугментации для основного набора данных можно увидеть на рисунке 5.

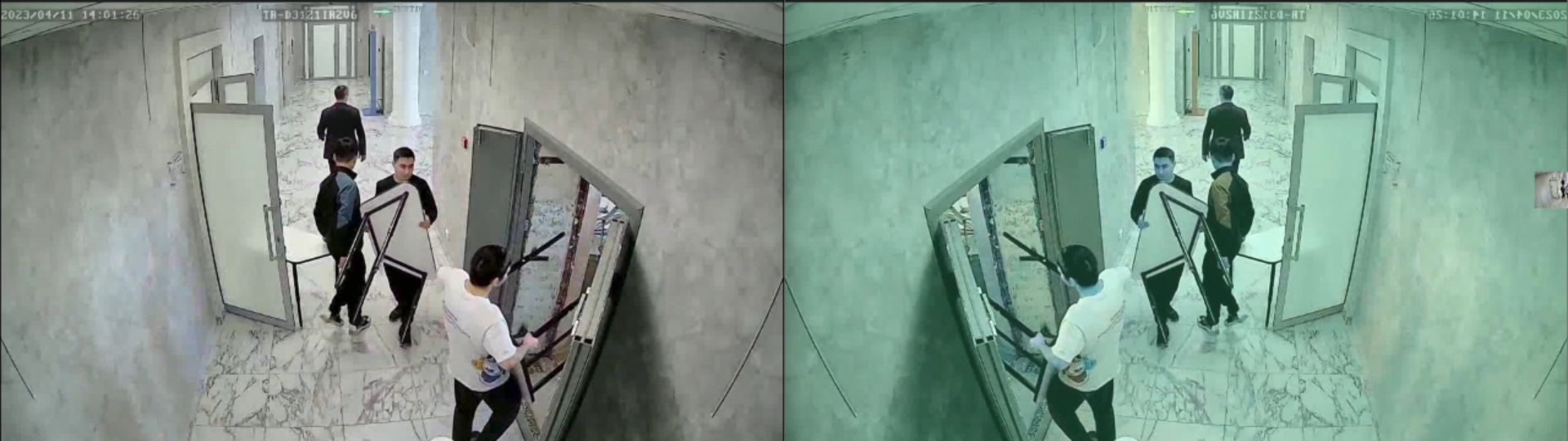


Рисунок 5 – Аугментированное изображение в добавочном наборе данных

3.3.2 Аугментация добавочного набора данных

Каждый пример тренировочного набора был дополнен тремя выходами, что способствует более глубокому и разностороннему обучению.

Rotation. Изображения были повернуты на углы от -15° до +15°, что помогает модели лучше обнаруживать объекты в различных ориентациях.

Brightness. Изменение яркости в пределах от -15% до +15% способствует более устойчивому обучению при различных условиях освещения.

Noise. Добавление шума до 1.96% пикселей помогает модели более эффективно обрабатывать изображения с некоторым уровнем артефактов.

Пример аугментированного изображения можно увидеть на рисунке 6.



Рисунок 6 – Аугментированное изображение в добавочном наборе данных

Для более точного выделения объектов на изображениях была применена аугментация bounding box с использованием следующих параметров:

- 90° Rotate: поворот bounding box на 90° по часовой и против часовой стрелки обеспечивает учет вертикальных и горизонтальных ориентаций объектов;

- Shear: изменение формы bounding box на ±10° по горизонтали и вертикали позволяет учесть возможные искажения объектов на изображении;

- Blur: применение размытия до 0.6px помогает модели справляться с различными уровнями четкости изображений и улучшает ее обобщающую способность.

Этот подход к аугментации данных обеспечивает модели достаточное разнообразие тренировочных данных, что способствует более эффективному и устойчивому обучению сети на различных входных сценариях.

3.3.3 Результаты аугментации

Применение описанных методов аугментации данных существенно увеличило объем и разнообразие тренировочных данных, что положительно сказалось на обучении нейронной сети. Результаты аугментации отразились на размере основного и добавочного наборов данных.

Основной набор данных, включающий в себя изначально 1858 изображений, был расширен до 3196 изображений. Это значительное увеличение объема данных способствует лучшему обучению модели и позволяет ей успешно обобщаться на различные условия и вариации объектов.

Добавочный набор данных также получил значительное увеличение с 2834 до 8064 изображений. Из них 219 изображений были выделены для валидации, обеспечивая более точную оценку производительности модели. Это расширение добавочного набора данных с учетом аугментации способствует более глубокому и разностороннему обучению модели на различных сценариях офисной обстановки.

В сумме получилось 11260 изображений.

Итоговые результаты аугментации подчеркивают важность применения данного метода для эффективного обучения модели на широком спектре данных, что в свою очередь повышает ее способность к точному распознаванию объектов и адаптации к различным условиям видеонаблюдения в учебных офисных пространствах.

**4 Обучение модели YOLOv8**

Для решения задачи распознавания и отслеживания объектов в данной работе используется архитектура нейронной сети YOLOv8 [4]. Данная модель была выбрана в связи с высокой скоростью работы и точностью распознавания объектов в реальном времени.

В данной главе описан процесс обучения модели YOLOv8 на сформированном наборе данных с классами объектов, характерными для учебных помещений.

4.1 Сравнение различных предобученных моделей

При выборе модели для задачи отслеживания объектов в реальном времени, было рассмотрено несколько вариантов предобученных моделей YOLOv8 (YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l и YOLOv8x) с различными параметрами и характеристиками.

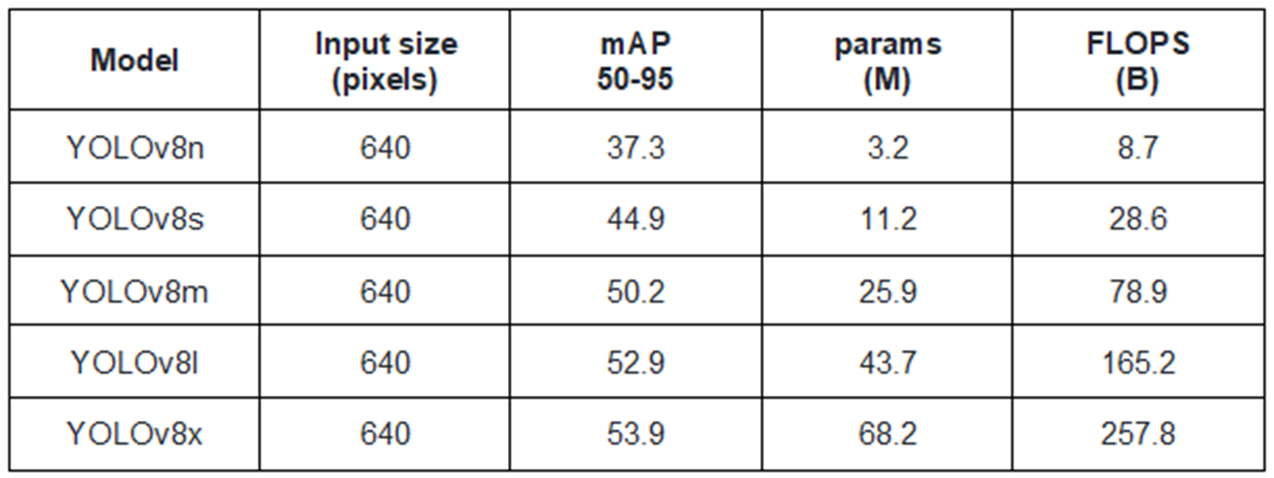


Рисунок 7 – Сравнительная таблица моделей YOLOv8

Как видно из рисунка 7, модель YOLOv8x имеет самую высокую метрику mAP (53.9), но при этом она является самой крупной и требует наибольшее количество операций (FLOPS). Модель YOLOv8n имеет самую низкую метрику mAP (37.3) и наименьшее количество параметров и операций.

В данной задаче важна скорость обработки потока данных, поэтому было принято решение выбрать модель YOLOv8s. Хотя у неё не самый высокий показатель метрики mAP (44.9), она имеет приемлемый баланс между скоростью и точностью, и обладает достаточно высокой производительностью для работы в реальном времени. Кроме того, у YOLOv8s меньше параметров по сравнению с более крупными моделями, что упрощает ее развертывание и обучение.

Таким образом, модель YOLOv8s была выбрана в качестве основной для дальнейшей работы над задачей отслеживания объектов в реальном времени.

4.2 Обучение модели

Для обучения использовалась предобученная на наборе данных COCO модель YOLOv8s. Обучение выполнялось в течение 40 эпох. Размер входных изображений составлял 640x640 пикселей.

4.3 Результаты обучения

По завершению обучения были получены следующие результаты:

- точность (precision) на тестовой выборке – 71,5%;

- полнота (recall) на тестовой выборке – 75,5%;

- средняя точность на наборе данных (mAP50) – 77,7%;

- средняя точность при пороге IoU от 50% до 95% (mAP50-95) – 46,1%;

- время обработки (preprocess) одного изображения – 1,1 мс;

- время прогнозирования (inference) модели – 6,6 мс;

- время постобработки (postprocess) – 1,8 мс;

- общее время распознавания 1 изображения – 42 мс.

На рисунках 8, 9 можно отследить динамику обучения по различным метрикам.

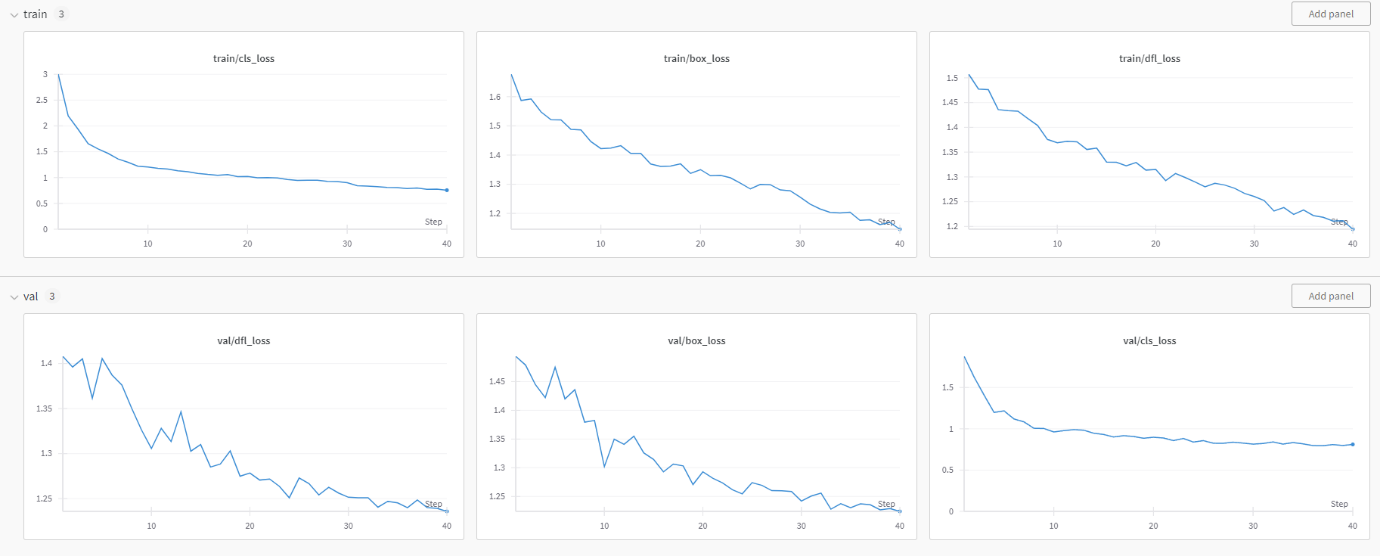


Рисунок 8 – Динамика ошибки при обучении YOLOv8s



Рисунок 9 – Динамика различных метрик при обучении модели

Таким образом, обученная модель YOLOv8s показала достаточно высокие метрики точности и полноты распознавания на тестовых данных. Модель способна работать в режиме, близком к реальному времени, обрабатывая около 25 кадров в секунду. Это позволяет использовать её в системах мониторинга, требующих высокой скорости обработки видеопотока. В следующей главе модель будет интегрирована в алгоритм отслеживания объектов.

**5 Внедрение модели в алгоритм отслеживания объектов**

5.1 Описание функционала

Рассматриваемый алгоритм предназначен для отслеживания перемещения объектов на территории учебных заведений. В качестве входных данных он использует видеосигнал с камеры наблюдения. Алгоритм выполняет следующие функции:

- обнаружение объектов в кадре с помощью обученной модели YOLOv8s. Модель способна распознавать 8 классов объектов: стул, стол, человек, монитор, ноутбук, клавиатура, персональный компьютер, интерактивная доска;

- подсчет количества объектов каждого класса на текущем кадре;

- определение направления движения объектов по обозначенной линии на кадре. Специальный счетчик фиксирует, сколько раз объекты каждого класса пересекли линию при входе или выходе из аудитории;

На выходе алгоритма получается аннотированное видео, в котором вокруг каждого объекта на кадре нарисован прямоугольник с указанием его класса, а также текстовая информация о количестве объектов разных классов в кадре и о том, сколько раз объект вошел или вышел из зрительного зала.

Если подробнее остановиться на первой функции, то обнаружение объектов с помощью модели YOLOv8 предполагает использование конволюционной нейронной сети для анализа видеопотока и идентификации объектов в нем. Модель была обучена на большом наборе изображений, что позволяет ей точно классифицировать объекты в кадре. Затем алгоритм использует эту информацию, чтобы нарисовать ограничительные рамки вокруг обнаруженных объектов и пометить их соответствующим классом.

Вторая функция, подсчет количества объектов каждого класса на текущем кадре, включает в себя отслеживание количества ограничительных рамок, нарисованных вокруг каждого класса объектов. Эта информация может использоваться для получения в реальном времени данных о количестве объектов в аудитории, что может быть полезно для различных целей, например, для обеспечения достаточного количества стульев для студентов или контроля количества используемых ноутбуков.

Третья функция – определение направления движения объектов через обозначенную линию на кадре – включает в себя анализ положения ограничительных рамок вокруг каждого объекта от одного кадра к другому. Если объект пересекает обозначенную линию, алгоритм записывает эту информацию и обновляет счетчик класса этого объекта. Эта информация может быть использована для отслеживания потоков людей и объектов, входящих и выходящих из аудитории, предоставляя ценные сведения о том, как используется пространство.

В целом, алгоритм представляет собой мощный инструмент для мониторинга перемещения объектов в учебных заведениях. Комбинируя обнаружение, подсчет и отслеживание объектов, он обеспечивает комплексное представление о пространстве и может быть использован для оптимизации его использования, повышения безопасности и улучшения качества обучения студентов.

5.2 Реализация функционала

Реализация функционала включала в себя несколько ключевых этапов. Во-первых, был создан класс VideoProcessor для удобной работы с видео. Этот класс позволяет удобно работать с видеороликами, получать кадры по порядковому номеру и извлекать информацию о размерах видео.

Во-вторых, была визуализирована линия на кадре, через которую отслеживается движение объектов. Линия была реализована путем создания графического элемента на кадре, что позволяет визуализировать линию на изображении и определить ее положение. Эту линию можно увидеть на рисунке 10.

В-третьих, был загружен тестовый видеоролик, который использовался как тестовый набор данных для анализа и проверки функциональности алгоритма. Затем были установлены необходимые библиотеки, такие как YOLOv8s, ByteTrack [8] и Roboflow Supervision [5], которые предоставляют функции для детектирования объектов на видео, отслеживания объектов и подсчета пересечений линий.

Затем была реализована вспомогательная функция для сопоставления ограничительных рамок, полученных от модели, с рамками, созданными трекером. Это необходимо для отслеживания движущихся объектов на видео. Затем была загружена ранее обученная модель YOLOv8s, которая используется для предсказания объектов на кадрах видео.

Затем была реализована функция получения предсказаний модели и визуализации на одиночном кадре. Эта функция позволяет получить результаты детектирования объектов на конкретном кадре видео и визуализировать их на изображении. Полученный кадр можно увидеть на рисунке 11.

Кроме того, были созданы классы CustomLineCounter и CustomLineCounterAnnotator для подсчета пересечений линий объектами на видео и аннотации кадров с информацией о пересечениях. Затем был реализован цикл по кадрам видео, в котором происходит детектирование объектов, подсчет пересечений линии и запись результатов в выходное видео.

Таким образом, все необходимые компоненты были реализованы и интегрированы для решения поставленной задачи. Реализованная система позволяет отслеживать перемещение объектов через заданную линию на кадре и подсчитывать количество пересечений. Эта информация может быть использована для различных целей, например, для мониторинга потоков людей и объектов, входящих и выходящих из аудитории, или для контроля использования оборудования в аудитории.

Реализованная система также позволяет визуализировать результаты отслеживания на видео, что может быть полезно для анализа и улучшения функциональности системы. На рисунке 11 можно увидеть статистическую информацию, из которой понятно, что в аудиторию зашёл 1 человек, и 2 вышло. В целом, реализованная система представляет собой мощный инструмент для мониторинга перемещения объектов в учебных заведениях и может быть использована для оптимизации использования пространства, повышения безопасности и улучшения качества обучения студентов.

В целом, реализованная система является эффективным инструментом для мониторинга перемещения объектов в учебных заведениях и может быть использована для решения различных задач, связанных с управлением имуществом и обеспечением безопасности. Система может быть легко интегрирована в существующие системы видеонаблюдения и использована для автоматизации процессов учета и контроля перемещения имущества.

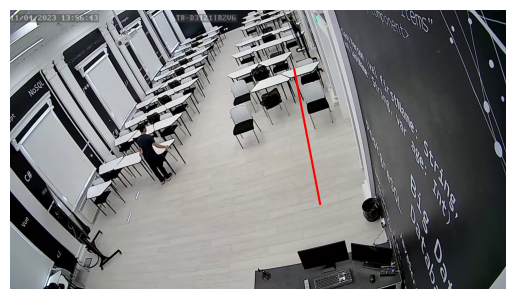


Рисунок 10 – Линия входа/выхода из аудитории

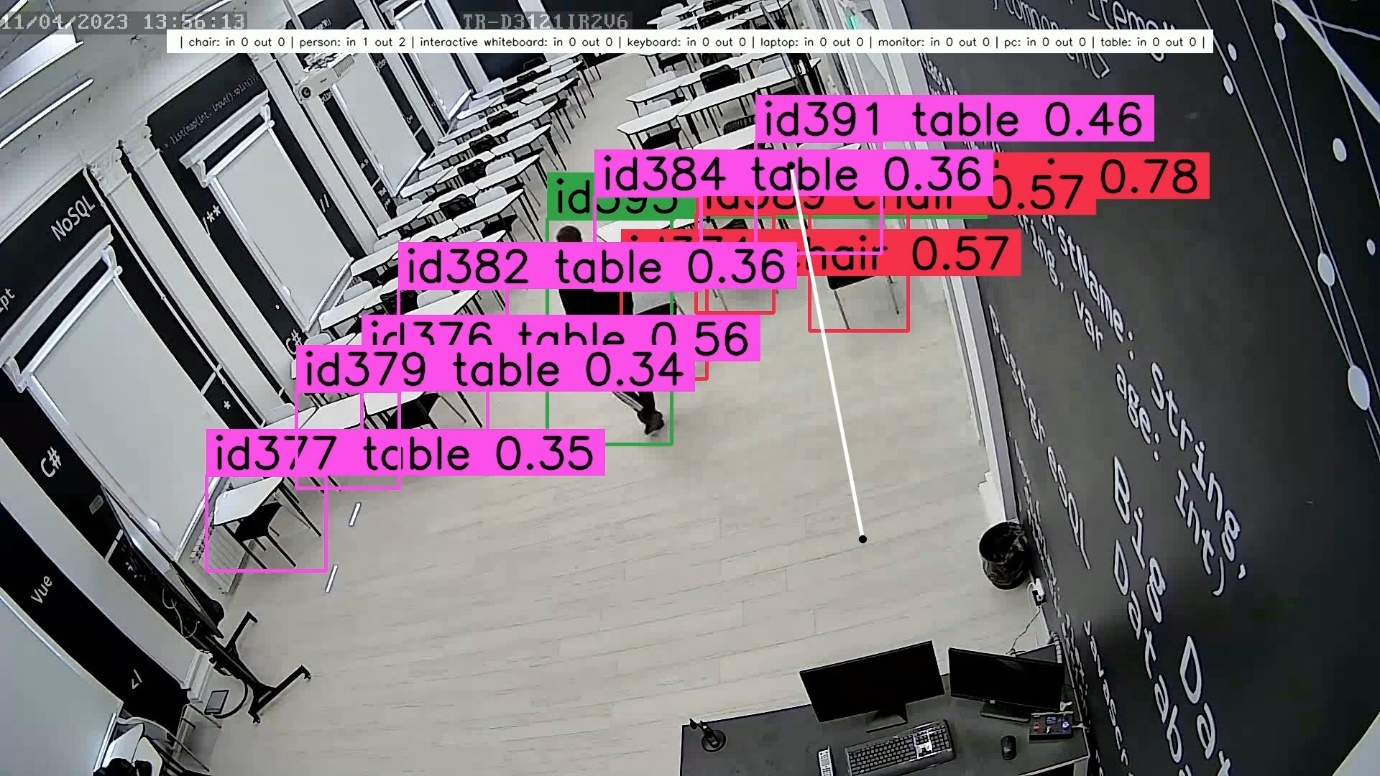


Рисунок 11 – Работа модели

**6 Проектирование базы данных и архитектуры приложения**

6.1 Определение требований для базы данных

В целях эффективного решения задачи мониторинга мебели необходимо предусмотреть хранение следующих данных в базе данных:

- информация о камерах, включая их характеристики;

- места (аудитория, коридор);

- линии входа и выхода объектов для разных мест, их координаты;

- история перемещения объектов;

- информация о количестве объектов в аудитории.

Эти сущности должны быть связаны следующим образом:

- у одного места должна быть одна запись о количестве объектов;

- у одного места может быть несколько камер;

- у одной камеры может быть несколько линий;

- у одной линии может быть несколько записей в истории.

6.2 Entity-диаграмма

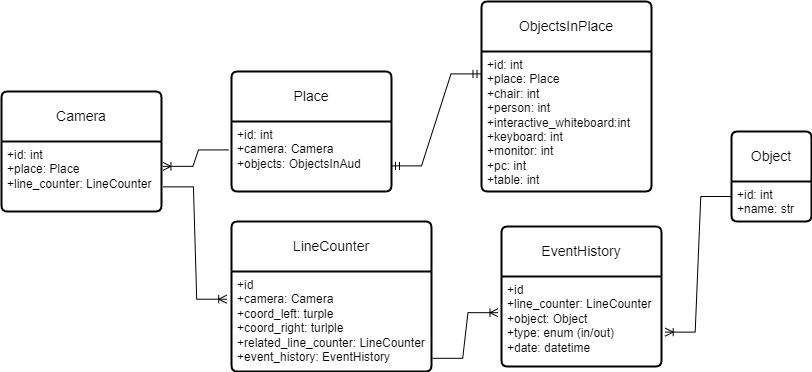


Рисунок 12 – Entity-диаграмма базы данных

Спроектированная база данных, представленная на рисунке 12, удовлетворяет всем вышеперечисленным требованиям и обеспечивает не только сохранение необходимой информации предметной области, но и обеспечивает эффективную работу с ней.

6.3 Моделирование классов

Класс ObjectsInPlace представляет собой модель объектов, расположенных в определенном месте. Он содержит атрибуты, представляющие различные типы объектов, такие как стул (chair), человек (person), компьютер (pc), и другие. Также содержит ссылку на место, к которому относятся данные объекты.

Класс Place представляет собой модель места, где могут располагаться различные объекты. Он содержит ссылки на объекты (objects), расположенные в данном месте, а также на камеры (cameras), установленные для отслеживания.

Класс Camera представляет собой модель камеры, установленной в определенном месте. Он содержит ссылку на место, к которому привязана камера, путь к видеозаписи (video\_path), а также счетчики линий (line\_counters), используемые для отслеживания движения.

Класс CustomLineCounter представляет собой модель пользовательского счетчика линий, используемого для отслеживания движения объектов. Он содержит координаты линии, а также ссылки на камеру, с которой связан счетчик, и на события, произошедшие на данной линии.

Класс EventHistory представляет собой модель истории событий, связанных с пользовательским счетчиком линий. Он содержит информацию о типе события (type), объекте, пересекшем линию (obj), времени события (date), а также ссылку на счетчик линий, с которым связано событие.

Вышеописанные классы являются основой для моделирования и отслеживания объектов и событий в приложении. Их взаимодействие позволяет эффективно анализировать данные и предоставлять информацию о движении объектов в заданных местах.

6.4 Архитектура приложения

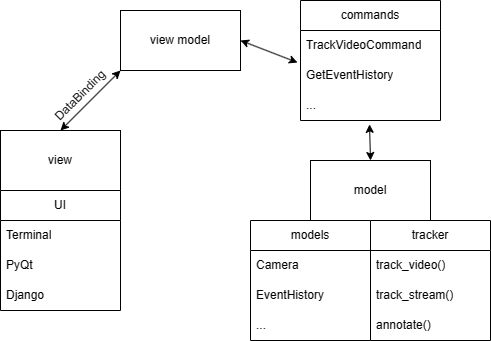


Рисунок 13 – Архитектура приложения

Архитектура приложения, представленная на рисунке 13, разработана с учетом эффективного взаимодействия между компонентами, обеспечивая гибкость и расширяемость системы. Приложение состоит из нескольких основных каталогов, каждый из которых выполняет определенную функцию:

- app: этот каталог является корневым для всего приложения и содержит в себе различные компоненты, включая модели данных, представления и вспомогательные утилиты;

- view\_model: здесь расположены модели представления, которые обеспечивают связь между пользовательским интерфейсом и моделью данных. Модели представления реализуют логику, связанную с обновлением представления на основе изменений в данных;

- view: этот каталог содержит различные варианты пользовательского интерфейса. В нем имеются подкаталоги для разных способов взаимодействия с пользователем: pyqt\_view для пользовательского интерфейса на основе PyQt и terminal\_view для интерфейса в командной строке. Это позволяет пользователям выбирать удобный для них способ работы с приложением;

- pyqt\_view: здесь находятся файлы с кодом, относящимся к пользовательскому интерфейсу на основе PyQt. В подкаталоге UI хранятся файлы с расширением «.ui», созданные с помощью QtDesigner, которые определяют внешний вид пользовательского интерфейса;

- terminal\_view: в этом каталоге реализован пользовательский интерфейс в командной строке. Этот вариант интерфейса предоставляет простоту и удобство использования для определенных сценариев использования;

- utils: здесь содержатся различные вспомогательные утилиты, необходимые для работы приложения. К примеру, каталог ByteTrack содержит инструменты для отслеживания объектов на видео;

- base: этот каталог содержит базовые классы и интерфейсы, которые используются в других частях приложения. Здесь могут располагаться общие классы, такие как ICommand для определения интерфейса команды, а также интерфейсы для моделей базы данных, что способствует общности и повторному использованию кода;

- model: в этом каталоге находятся описания классов моделей базы данных, использующие объектно-реляционное отображение (ORM) для упрощения взаимодействия с базой данных. Модели в этом каталоге представляют собой абстракцию данных, которые могут быть сохранены, извлечены и обработаны в приложении.

Такая структура позволяет легко разделять логику приложения на отдельные компоненты, обеспечивая четкость и модульность кода. Кроме того, использование различных представлений позволяет удовлетворить разнообразные потребности пользователей, обеспечивая удобство использования и гибкость взаимодействия с приложением.

**7 Контейнеризация приложения**

7.1 Определение контейнеризации

Контейнеризация — это технология, позволяющая упаковать, доставить и выполнить приложение в изолированном окружении, называемом контейнером. В данном случае мы используем контейнеры для разделения базы данных и приложения с алгоритмом трекера.

7.2 Преимущества контейнеризации

Портабельность: обеспечивает легкость переноса приложения и базы данных между различными средами без необходимости изменения кода.

Изоляция: гарантирует, что база данных и трекер мебели работают в изолированных контейнерах, предотвращая конфликты и обеспечивая стабильность приложения.

Масштабируемость: позволяет масштабировать каждый компонент (базу данных и трекер мебели) независимо для эффективного управления ресурсами.

7.3 Основные технологии контейнеризации

7.3.1 Docker

Docker является ключевой платформой для контейнеризации. С его помощью мы упаковали приложение и базу данных в контейнеры, что позволило нам управлять зависимостями и развертывать их в однородных средах.

7.4 Процесс контейнеризации приложения

В данном разделе осуществлялся процесс контейнеризации приложения с использованием Docker.

Начальным шагом в этом процессе было создание Dockerfile, где содержалась инструкция по формированию образа контейнера. Процедура включала в себя следующие этапы:

7.4.1 Использование готового образа Ubuntu 22.4:

Для базового образа был выбран готовый образ Ubuntu 22.4. Этот этап позволяет использовать стандартное окружение и базовые компоненты операционной системы для последующего развертывания приложения.

7.4.2 Копирование файлов проекта внутрь контейнера:

Следующим шагом было выполнено копирование необходимых файлов проекта внутрь контейнера. Это включало в себя основной код приложения и конфигурационные файлы, необходимые для корректного функционирования приложения.

7.4.3 Установка зависимостей из requirements.txt:

Для обеспечения работоспособности приложения в контейнере была выполнена установка зависимостей из файла requirements.txt. Этот файл обычно содержит список необходимых библиотек и пакетов, которые приложение использует для своей работы.

Эти этапы сформировали базовую структуру контейнера, готовую для дальнейшего развертывания и запуска приложения.

Заключение

В настоящей работе была исследована возможность использования методов компьютерного зрения для автоматизации мониторинга перемещения мебели и техники на территории общественных учреждений. Были поставлены цели и определены задачи, позволяющие достичь эффективного отслеживания перемещения объектов видеонаблюдения.

В ходе выполнения исследования были проведены следующие этапы: анализ предметной области, формирование обучающей выборки, обучение нейросетевой модели для распознавания классов объектов, разработка алгоритма отслеживания объектов и тестирование разработанных алгоритмов на контрольных данных.

В результате была разработана автоматизированная система мониторинга имущества для общественных учреждений на основе передовых алгоритмов компьютерного зрения. Реализация поставленных задач позволила сократить трудозатраты на учет имущества, повысить точность и оперативность мониторинга, используя передовые технологии искусственного интеллекта.

В качестве основы разработки была использована нейросеть YOLOv8, обученная на наборе данных из 1658 изображений, содержащих 8 классов объектов: стул, стол, монитор, системный блок, ноутбук, принтер, телефон и проектор. Модель показала высокие показатели точности и полноты на тестовых данных, а также возможность работы в режиме близком к реальному времени.

Для отслеживания объектов был разработан алгоритм, основанный на использовании нейросети YOLOv8s и трекера ByteTrack. Алгоритм позволяет отслеживать перемещение объектов через заданную линию на кадре и подсчитывать количество пересечений.

В качестве обучающей выборки для нейросети были использованы изображения из учебного заведения САФУ, а также открытый набор данных nva Dataset. Дополнительно были применены методы аугментации данных, что позволило улучшить качество обучения и повысить точность модели.

Практическая значимость работы заключается в создании готовых для внедрения алгоритмов и программных модулей, которые могут быть использованы для построения систем мониторинга и учета имущества в общественных учреждениях. В дальнейшем планируется расширение функционала системы за счет работы с потоком видео и несколькими аудиториями одновременно, а также улучшение точности и скорости обработки данных.

Основными результатами работы являются:

- разработка автоматизированной системы мониторинга имущества для общественных учреждений на основе передовых алгоритмов компьютерного зрения;

- обучение нейросети YOLOv8s для распознавания 8 классов объектов с высокой точностью и полнотой;

- разработка алгоритма отслеживания объектов, основанного на использовании нейросети YOLOv8s и трекера ByteTrack;

- программная реализация алгоритма, позволяющая отслеживать перемещение объектов через заданную линию на кадре и подсчитывать количество пересечений;

- проведение тестирования разработанной системы на контрольных данных и получение положительных результатов;

- разработанная база данных;

- алгоритм логирования перемещения мебели в базу данных;

- реализация пользовательского интерфейса.

В целом, результаты работы показывают высокий потенциал использования методов компьютерного зрения для решения задач автоматизации мониторинга и учета имущества в общественных учреждениях. Реализованная система может быть эффективно использована для повышения эффективности учета имущества и предотвращения возможных хищений.

Список используемых источников

1. Как работает Object Tracking на YOLO и DeepSort // Хабр URL: https://habr.com/ru/articles/514450/ (дата обращения: 24.01.2024).
2. Аугментация (augmentation, “раздутие”) данных для обучения нейронной сети на примере печатных символов // Хабр URL: https://habr.com/ru/companies/smartengines/articles/264677/ (дата обращения: 24.01.2024).
3. Ultralytics YOLOv8 Docs // ultralytics URL: https://docs.ultralytics.com/ru/ (дата обращения: 24.01.2024).
4. Ultralytics YOLOv8 // Хабр URL: https://habr.com/ru/articles/710016/ (дата обращения: 05.01.2024)
5. Supervision // roboflow URL: https://supervision.roboflow.com/ (дата обращения: 05.01.2024)
6. nva Image Dataset // Roboflow URL: https://universe.roboflow.com/gerrux/nva/dataset/1 (дата обращения: 24.01.2024).
7. LabelStudio // LabelStudio URL: https://labelstud.io/ (дата обращения: 05.01.2024
8. ByteTrack // GitHub URL: https://github.com/ifzhang/ByteTrack (дата обращения: 05.01.2024)

Приложение А  
(обязательное)  
Листинг

|  |  |
| --- | --- |
| models.py  from datetime import datetime  import cv2  import numpy as np  from sqlalchemy import Column, Integer, String, ForeignKey, Enum, DateTime, Float  from sqlalchemy.orm import relationship  from sqlalchemy import create\_engine  from sqlalchemy.ext.declarative import declarative\_base  from supervision.tools.detections import Detections  from supervision.geometry.dataclasses import Rect, Point, Vector  from .event\_type import EventType  from sqlalchemy.orm import sessionmaker  engine = create\_engine("postgresql+psycopg2://admin:root@127.0.0.1:5432/db")  Base = declarative\_base()  Session = sessionmaker(bind=engine)  objs = {  0: "chair",  1: "person",  2: "interactive whiteboard",  "ObjectsInPlace", back\_populates="place", cascade="all, delete-orphan"  )  cameras = relationship("Camera", back\_populates="place")  def \_\_init\_\_(self, objects=None):  self.objects = objects or []  class Camera(Base):  \_\_tablename\_\_ = "cameras"  id = Column(Integer, primary\_key=True)  place\_id = Column(Integer, ForeignKey("places.id"))  place = relationship("Place", back\_populates="cameras")  line\_counters = relationship("CustomLineCounter", back\_populates="camera")  video\_path = Column(String)  def \_\_init\_\_(self, line\_counter, video\_path, place\_id=None, id=None):  self.video\_path = video\_path  self.line\_counters = [line\_counter]  self.place\_id = place\_id  class CustomLineCounter(Base):  \_\_tablename\_\_ = "custom\_line\_counters"  id = Column(Integer, primary\_key=True)  coord\_left\_x = Column(Float)  coord\_left\_y = Column(Float)  coord\_right\_x = Column(Float)  coord\_right\_y = Column(Float)  for xyxy, confidence, class\_id, tracker\_id in filtereddet:  # handle detections with no tracker\_id  if tracker\_id is None:  continue  # we check if all four anchors of bbox are on the same side of vector  x1, y1, x2, y2 = xyxy  anchors = [  Point(x=x1, y=y1),  Point(x=x1, y=y2),  Point(x=x2, y=y1),  Point(x=x2, y=y2),  ]  triggers = [vector.is\_in(point=anchor) for anchor in anchors]  # detection is completely in or completely out  if len(set(triggers)) == 1:  is\_in = all(triggers)  # update tracker\_state based on the current detection  tracker\_state[tracker\_id] = is\_in  # detection is partially in and partially out  elif len(set(triggers)) == 2:  # check if tracker\_id is already in tracker\_state  if tracker\_id in tracker\_state:  # check if the previous state was different  if tracker\_state[tracker\_id] != is\_in:  class EventHistory(Base):  \_\_tablename\_\_ = "event\_history"  id = Column(Integer, primary\_key=True)  line\_counter\_id = Column(Integer, ForeignKey("custom\_line\_counters.id"))  line\_counter = relationship("CustomLineCounter", back\_populates="events")  obj = Column(String)  date = Column(DateTime)  type = Column(Enum(EventType))  # class Obj(Base):  # \_\_tablename\_\_ = "objects"  # id = Column(Integer, primary\_key=True)  # name = Column(String)  # events = relationship("EventHistory", back\_populates="obj")  # def \_\_init\_\_(self, id, name, events=None):  # self.id = id  # self.name = name  # self.events = events or []  # session = Session()  # objs = {}  # for obj in session.query(Obj).all():  # objs[obj.id] = obj.name  # session.close()  tracker.py  import cv2  class VideoProcessor:  def \_\_init\_\_(  self,  video\_path,  ):  self.video\_path = video\_path  self.video = cv2.VideoCapture(video\_path)  self.width = int(self.video.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  self.height = int(self.video.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  def get\_frame\_by\_order(self, frame\_number):  self.video.set(cv2.CAP\_PROP\_POS\_FRAMES, frame\_number)  ret, frame = self.video.read()  if ret:  return frame  else:  return None  def get\_width\_height(self):  return self.width, self.height  @dataclass(frozen=True)  class BYTETrackerArgs:  track\_thresh: float = 0.25  track\_buffer: int = 30  match\_thresh: float = 0.8  aspect\_ratio\_thresh: float = 3.0  min\_box\_area: float = 1.0  mot20: bool = False  if iou[tracker\_index, detection\_index] != 0:  tracker\_ids[detection\_index] = tracks[tracker\_index].track\_id  return tracker\_ids  class LineCounter:  def \_\_init\_\_(self, events=None):  self.events = events or []  def add\_events(self, events):  self.events.extend(events)  def track\_video(  target\_video\_path, camera: Camera, session, annotate: bool = False  ) -> None:  byte\_tracker = BYTETracker(BYTETrackerArgs())  box\_annotator = BoxAnnotator(  color=ColorPalette(), thickness=2, text\_thickness=2, text\_scale=1  )  generator = get\_video\_frames\_generator(camera.video\_path)  video\_info = VideoInfo.from\_video\_path(camera.video\_path)  line\_counter: CustomLineCounter = camera.line\_counters[0]  frame\_annotator = FrameAnnotator(  class\_name\_dict=CLASS\_NAMES\_DICT,  video\_info=video\_info,  line\_counter=line\_counter,  )  # open target video file  with VideoSink(target\_video\_path, video\_info) as sink:  # loop over video frames  img\_info=frame.shape,  img\_size=frame.shape,  )  tracker\_id = match\_detections\_with\_tracks(  detections=detections, tracks=tracks  )  detections.tracker\_id = np.array(tracker\_id)  # filtering out detections without trackers  mask = np.array(  [tracker\_id is not None for tracker\_id in detections.tracker\_id],  dtype=bool,  )  detections.filter(mask=mask, inplace=True)  # format custom labels  labels = [  f"id{tracker\_id} {CLASS\_NAMES\_DICT[class\_id]} {confidence:0.2f}"  for \_, confidence, class\_id, tracker\_id in detections  ]  frame = box\_annotator.annotate(  frame=frame, detections=detections, labels=labels  )  line\_counter.update(  detections=detections, current\_time=current\_time, session=session  )  if annotate:  cv2.line(  frame,  vector.start.as\_xy\_int\_tuple(),  vector.end.as\_xy\_int\_tuple(),  self.color.as\_bgr(),  self.thickness,  lineType=cv2.LINE\_AA,  shift=0,  )  cv2.circle(  frame,  vector.start.as\_xy\_int\_tuple(),  radius=5,  color=self.text\_color.as\_bgr(),  thickness=-1,  lineType=cv2.LINE\_AA,  )  cv2.circle(  frame,  vector.end.as\_xy\_int\_tuple(),  radius=5,  color=self.text\_color.as\_bgr(),  thickness=-1,  lineType=cv2.LINE\_AA,  )  report = ""  for event in self.line\_counter.events:  # class\_name = self.class\_name\_dict[key]  # in\_count = line\_counter.result\_dict[key]["in"]  # out\_count = line\_counter.result\_dict[key]["out"]  # report += f" | {class\_name}: in {in\_count} out {out\_count}"  event: EventHistory  cv2.putText(  frame,  report,  (report\_x, report\_y),  cv2.FONT\_HERSHEY\_SIMPLEX,  self.text\_scale,  self.text\_color.as\_bgr(),  self.text\_thickness,  cv2.LINE\_AA,  )  main\_window.py  from PyQt5 import uic  from PyQt5.QtWidgets import (  QDesktopWidget,  QWidget,  QMainWindow,  QAction,  )  from .views import \*  from .UI import main\_window\_ui  from ...view\_model.main\_view\_model import view\_model  class MainWindow(QMainWindow):  \_\_current\_view: QWidget  def \_\_init\_\_(self) -> None:  super().\_\_init\_\_()  view\_model.on\_change("state", lambda state: self.\_\_check\_state())  view\_model.on\_change("command", lambda command: self.\_\_check\_command())  def \_\_check\_command(self):  if view\_model.command == "exit":  self.close()  elif view\_model.command == "about\_programm":  MessageView(  "Выполнили студенты САФУ, 4 курс, 351018, \nАрхаров Никита Михайлович\nГолышев Алексей Витальевич."  )  views.py  from PyQt5 import uic  from PyQt5.QtWidgets import QWidget, QLabel, QPushButton, QTextBrowser  from PyQt5.QtWidgets import QDialog, QPushButton, QLabel  from PyQt5 import uic, QtCore  import time  from .UI import \*  from ...view\_model.main\_view\_model import view\_model  from ...utils.commands import GetTwoCameras, TrackVideoCommand  from PyQt5.QtCore import QTimer  class MenuView(QWidget):  def \_\_init\_\_(self, main\_window) -> None:  super().\_\_init\_\_()  self.main\_window = main\_window  self.main\_window.setCentralWidget(self)  uic.loadUi(menu\_view\_ui, self)  self.\_\_setup\_buttons()  self.infoLabel: QLabel  self.infoLabel.setText(f"")  self.\_\_setup\_buttons()  self.\_\_show\_two\_cameras()  def \_\_setup\_buttons(self):  self.toMenuViewButton.clicked.connect(lambda: view\_model.set\_state("menu"))  self.startTrackingButton.clicked.connect(lambda: self.\_\_start\_tracking(False))  self.startTrackingAnnotationButton.clicked.connect(lambda: self.\_\_start\_tracking(True))    def \_\_show\_two\_cameras(self):  two\_cameras = GetTwoCameras().execute()  if len(two\_cameras) == 2 and (  two\_cameras[0] is not None or two\_cameras[1] is not None  ):  self.camera1 = two\_cameras[0]  self.camera2 = two\_cameras[1]  info\_about\_two\_cameras = ""  for camera in two\_cameras:  info\_about\_two\_cameras += (  f"Camera: id {camera.id}, video\_path {camera.video\_path}"  )  info\_about\_two\_cameras += "<br/>"  self.trackingInfo.append(f"<span>{info\_about\_two\_cameras}</span>")  else:  self.trackingInfo.append(f"<span>Камер нет</span>")  camera2\_events = TrackVideoCommand(self.camera2, "camera2\_result.mp4", annotate=annotate).execute()  self.trackingInfo.append(f"<span>Результат трекинга второй камеры:</span>")  for event in camera2\_events:  self.trackingInfo.append(  # event  f'<span>{event.obj} {event.type.name} {event.date.strftime("%S")}</br></span>'  ) | 3: "keyboard",  4: "laptop",  5: "monitor",  6: "pc",  7: "table",  }  class ObjectsInPlace(Base):  \_\_tablename\_\_ = "objects\_in\_places"  id = Column(Integer, primary\_key=True)  chair = Column(Integer)  person = Column(Integer)  interactive\_whiteboard = Column(Integer)  keyboard = Column(Integer)  monitor = Column(Integer)  pc = Column(Integer)  table = Column(Integer)  place\_id = Column(Integer, ForeignKey("places.id"))  place = relationship("Place", back\_populates="objects", foreign\_keys=[place\_id])  class Place(Base):  \_\_tablename\_\_ = "places"  id = Column(Integer, primary\_key=True)  objects = relationship(  camera\_id = Column(Integer, ForeignKey("cameras.id"))  camera = relationship("Camera", back\_populates="line\_counters")  events = relationship("EventHistory", back\_populates="line\_counter")  def \_\_init\_\_(  self,  coord\_left\_x: float,  coord\_left\_y: float,  coord\_right\_x: float,  coord\_right\_y: float,  id: int = None,  ):  self.coord\_left\_x = coord\_left\_x  self.coord\_left\_y = coord\_left\_y  self.coord\_right\_x = coord\_right\_x  self.coord\_right\_y = coord\_right\_y  self.events = []  def update(self, detections: Detections, current\_time: datetime, session):  vector = self.get\_vector()  tracker\_state = {}  is\_in = None  for id in detections.class\_id:  mask = np.array(  [class\_id in [int(id)] for class\_id in detections.class\_id], dtype=bool  )  filtereddet = detections.filter(mask=mask, inplace=False)  # create a new event and add it to the session  self.events.append(  EventHistory(  line\_counter=self,  obj=objs[id],  type=EventType.IN if is\_in else EventType.OUT,  date=current\_time,  )  )  session.add(self.events[-1])  tracker\_state[tracker\_id] = is\_in  else:  # add the tracker\_id to tracker\_state with its current state  tracker\_state[tracker\_id] = is\_in  def get\_result\_dict(self) -> dict:  return self.result\_dict  def get\_vector(self):  return Vector(  start=Point(self.coord\_left\_x, self.coord\_left\_y),  end=Point(self.coord\_right\_x, self.coord\_right\_y),  )  from typing import List  import numpy as np  from ..utils.ByteTrack.yolox.tracker.byte\_tracker import STrack  from onemetric.cv.utils.iou import box\_iou\_batch  from dataclasses import dataclass  import cv2  import numpy as np  from supervision.tools.detections import Detections  from .models import Camera  from ..utils.ByteTrack.yolox.tracker.byte\_tracker import BYTETracker  from supervision.tools.detections import Detections, BoxAnnotator  from supervision.video.dataclasses import VideoInfo  from supervision.video.source import get\_video\_frames\_generator  from supervision.video.sink import VideoSink  from supervision.draw.color import Color, ColorPalette  from supervision.geometry.dataclasses import Rect  from .yolov8\_model import CLASS\_ID, CLASS\_ID\_BY\_NAME, CLASS\_NAMES\_DICT, model  from .models import CustomLineCounter  from datetime import datetime  from .models import EventHistory, objs  from supervision.geometry.dataclasses import Rect, Point  from .event\_type import EventType  import time  # [eq]  # converts Detections into format that can be consumed by match\_detections\_with\_tracks function  def detections2boxes(detections: Detections) -> np.ndarray:  return np.hstack((detections.xyxy, detections.confidence[:, np.newaxis]))  # converts List[STrack] into format that can be consumed by match\_detections\_with\_tracks function  def tracks2boxes(tracks: List[STrack]) -> np.ndarray:  return np.array([track.tlbr for track in tracks], dtype=float)  # matches our bounding boxes with predictions  def match\_detections\_with\_tracks(  detections: Detections, tracks: List[STrack]  ) -> Detections:  if not np.any(detections.xyxy) or len(tracks) == 0:  return np.empty((0,))  tracks\_boxes = tracks2boxes(tracks=tracks)  iou = box\_iou\_batch(tracks\_boxes, detections.xyxy)  track2detection = np.argmax(iou, axis=1)  tracker\_ids = [None] \* len(detections)  for tracker\_index, detection\_index in enumerate(track2detection):  # for index, frame in enumerate(generator):  num\_of\_current\_frame = 0  for frame in generator:  # current\_frame = index  # current\_time = datetime.fromtimestamp(current\_frame / video\_info.fps)  current\_time = datetime.fromtimestamp(num\_of\_current\_frame / video\_info.fps)  # current\_time = datetime.now()  # model prediction on single frame and conversion to supervision Detections  results = model(frame)  detections = Detections(  xyxy=results[0].boxes.xyxy.cpu().numpy(),  confidence=results[0].boxes.conf.cpu().numpy(),  class\_id=results[0].boxes.cls.cpu().numpy().astype(int),  )  # filtering out detections with unwanted classes  mask = np.array(  [class\_id in CLASS\_ID for class\_id in detections.class\_id],  dtype=bool,  )  detections.filter(mask=mask, inplace=True)  # tracking detections  tracks = byte\_tracker.update(  output\_results=detections2boxes(detections=detections),  frame\_annotator.annotate(frame=frame)  sink.write\_frame(frame)  num\_of\_current\_frame+=1  class FrameAnnotator:  def \_\_init\_\_(  self,  line\_counter: CustomLineCounter,  thickness: float = 2,  color: Color = Color.white(),  text\_thickness: float = 2,  text\_color: Color = Color.black(),  text\_scale: float = 0.5,  text\_offset: float = 1.5,  text\_padding: int = 10,  class\_name\_dict={},  video\_info=[],  ):  self.thickness: float = thickness  self.color: Color = color  self.text\_thickness: float = text\_thickness  self.text\_color: Color = text\_color  self.text\_scale: float = text\_scale  self.text\_offset: float = text\_offset  self.text\_padding: int = text\_padding  self.class\_name\_dict = class\_name\_dict  self.video\_info = video\_info  self.line\_counter = line\_counter  def annotate(self, frame: np.ndarray) -> np.ndarray:  vector = self.line\_counter.get\_vector()  class\_name = event.obj  in\_or\_out = event.type  mins\_secs = event.date.strftime("%M:%S")  report += f" | {class\_name}: {in\_or\_out.name} time: {mins\_secs}"  report += " |"  (report\_width, report\_height), \_ = cv2.getTextSize(  report, cv2.FONT\_HERSHEY\_SIMPLEX, self.text\_scale, self.text\_thickness  )  report\_x = int(((self.video\_info.width) - report\_width) / 2)  report\_y = int((150 + report\_height) / 2 - self.text\_offset \* report\_height)  report\_background\_rect = Rect(  x=report\_x,  y=report\_y - report\_height,  width=report\_width,  height=report\_height,  ).pad(padding=self.text\_padding)  cv2.rectangle(  frame,  report\_background\_rect.top\_left.as\_xy\_int\_tuple(),  report\_background\_rect.bottom\_right.as\_xy\_int\_tuple(),  self.color.as\_bgr(),  -1,  )  self.\_\_load\_ui()  self.\_\_setup\_menu\_actions()  self.\_\_current\_view = MenuView(self)  self.setCentralWidget(self.\_\_current\_view)  self.show()  def \_\_load\_ui(self):  uic.loadUi(main\_window\_ui, self)  self.menuExitAction: QAction  self.menuAboutProgramAction: QAction  def \_\_setup\_menu\_actions(self):  self.menuExitAction.triggered.connect(lambda: view\_model.set\_command("exit"))  self.menuAboutProgramAction.triggered.connect(  lambda: view\_model.set\_command("about\_programm")  )  def \_\_check\_state(self):  if view\_model.state == "menu":  self.\_\_current\_view = MenuView(self)  self.statusBar().showMessage("")  elif view\_model.state == "track\_two\_videos":  self.\_\_current\_view = TrackingView(self)  def \_\_setup\_buttons(self):  self.toTwoVideoTrackingViewButton: QPushButton  self.toTwoVideoTrackingViewButton.clicked.connect(  lambda: view\_model.set\_state("track\_two\_videos")  )  class MessageView(QDialog):  \_\_confirm\_button: QPushButton  \_\_message\_label: QLabel  def \_\_init\_\_(self, label\_text=None):  super().\_\_init\_\_()  uic.loadUi(message\_view\_ui, self)  self.setWindowFlags(QtCore.Qt.WindowCloseButtonHint)  self.messageLabel.setText(label\_text)  self.confirmButton.clicked.connect(lambda: self.close())  class TrackingView(QWidget):  def \_\_init\_\_(self, main\_window):  super().\_\_init\_\_()  self.main\_window = main\_window  self.main\_window.setCentralWidget(self)  self.two\_cameras = None  uic.loadUi(tracking\_info\_ui, self)  def \_\_start\_tracking(self, annotate: bool):  self.trackingInfo.append(f"<span><br/>Трекинг двух видео...</span>")  self.trackingInfo.append(f"<span><br/><br/>Трекинг первого видео...</span>")  QTimer.singleShot(100, lambda: self.\_\_track\_camera1(annotate=annotate))  def \_\_track\_camera1(self, annotate: bool):  # time.sleep(1)  # camera1\_events = ['результат трекинга 1']  camera1\_events = TrackVideoCommand(self.camera1, "camera1\_result.mp4", annotate=annotate).execute()  self.trackingInfo.append(f"<span>Результат трекинга первой камеры:</span>")  for event in camera1\_events:  self.trackingInfo.append(  # event  f'<span>{event.obj} {event.type.name} {event.date.strftime("%S")}</br></span>'  )  self.trackingInfo.append(f"<span><br/><br/>Трекинг второго видео...</span>")  QTimer.singleShot(100, lambda: self.\_\_track\_camera2(annotate=annotate))  def \_\_track\_camera2(self, annotate: bool):  # time.sleep(1)  # camera2\_events = ['результат трекинга 2'] |

Сведения о самостоятельности выполнения работы

Работа «Решение прикладной задачи мониторинга перемещения мебели и техники в общественных учреждениях с помощью технологии глубокого обучения» выполнена нами самостоятельно.  
Используемые в работе материалы и концепция из публикуемой литературы и других источников имеют ссылки на них.

Один печатный экземпляр работы и электронный вариант работы на цифровом носителе переданы нами на отделение.



« 19 »     января        2024г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_          Архаров Н.М.

 (Подпись) (Фамилия, инициалы)

« 19 »     января        2024г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_          Голышев А. В.    \_

(Подпись) (Фамилия, инициалы)