

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

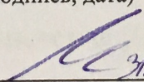
на тему

**ПРОГРАММА ОЦЕНКИ РАССТОЯНИЯ ДО ОБЪЕКТА
НА ИЗОБРАЖЕНИИ С ПОМОЩЬЮ НЕЙРОННЫХ
СЕТЕЙ**

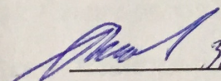
Студент

 30.05.2024 Шмелев Данила Валерьевич
(подпись, дата) (ФИО полностью)

Руководитель

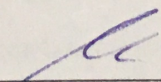
 31.05.24 М.А. Митрохин
(подпись, дата) (фамилия, инициалы)

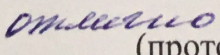
Нормоконтролёр

 31.05.24 А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 № 11)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от 04.06.24 № 10)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза, 2024

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«19» 04 2024г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Шмелев Данила Валерьевич гр. 20ВВП2 факультета вычислительной техники.
2. Тема работы «Программа оценки расстояния до объекта на изображении с помощью нейронных сетей».

Тема утверждена приказом ПГУ № 405/6 от "19" апреля 2024 г.

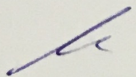
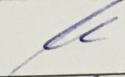
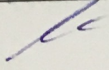
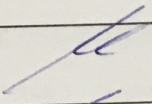
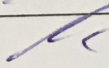
3. Руководитель бакалаврской работы д.т.н., доцент кафедры ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программное обеспечение для измерения расстояния от объекта на изображении до камеры. Язык программирования Python, среда разработки PyCharm. Программа должна обрабатывать различные типы входных данных и использовать следующие технологии: PyTorch.

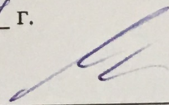
5. Перечень подлежащих разработке вопросов.

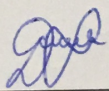
Требуется произвести анализ предметных областей и фреймворков для работы с моделями глубокого обучения. Выбрать технологии разработки.

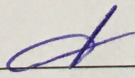
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области методов обнаружения объектов на изображении	32	19.04.2024- 22.04.2024	
2	Анализ предметной области методов измерения расстояния	32	23.04.2024- 27.04.2024	
3	Анализ инструментов разработки	65	28.04.2024- 02.05.2024	
4	Разработка программного обеспечения на основе проведенного анализа	94	03.05.2024- 23.05.2024	
5	Отладка и тестирование	49	24.04.2024- 27.05.2024	
6	Оформление выпускной работы	32	28.05.2024- 30.05.2024	

Дата выдачи задания "19" апреля 2024 г.

Руководитель бакалаврской работы  М.А. Митрохин
(подпись, дата)

Задание к исполнению принял студент  Д.В. Шмелев
(подпись, дата)

Работу к защите допустить
Декан факультета  31.05.2024 Л.Р. Фионова
(подпись, дата)

Реферат

Работа содержит 54 л., 11 рис., 4 табл., 18 источн., 3 прил.

НЕЙРОННЫЕ СЕТИ, ИЗМЕРЕНИЕ РАССТОЯНИЯ, МОДЕЛИ ГЛУБОКОГО ОБУЧЕНИЯ, PYTHON, YOLOV8, UNIDERTH

Цель работы – изучение моделей глубокого обучения в области обнаружения объекта на изображении и прогнозировании карты глубины всего изображения.

Объектом разработки является программное обеспечение, которое позволит пользователю обрабатывать изображения при помощи нейронных сетей.

В результате проведенной работы разработано программное обеспечение с использованием фреймворка глубокого обучения PyTorch на языке программирования Python.

Результаты данной работы применены в научно-исследовательской работе АО «НПП «Цифровые решения».

ПГУ 1.090301.40.001 ПЗ

Изм	Лист	№ докум.	Подпись	Дата	Литера	Лист	Листов
Разраб.		Шмелев Д.В.		30.05.24	Программа оценки расстояния до объекта на изображении с помощью нейронных сетей		
Пров.		Митрохин М. А.		31.05.24			
Н. Контр.		Бычков А.С.		31.05.24	Пояснительная записка		
Утв.		Митрохин М.А.		31.05.24			
					ФВТ, каф. ВТ гр. 20ВВП2		

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Обзор предметной области	7
1.1 Анализ темы	7
1.2 Анализ технологий	8
1.2.1 Обнаружение объекта	9
1.2.2 Измерение расстояния	11
1.3 Постановка задачи	17
2 Выбор технологий разработки	18
2.1 Выбор языка программирования	18
2.1.1 Язык программирования Python	18
2.1.2 Язык программирования C++	19
2.1.3 Язык программирования C#	19
2.2 Выбор фреймворка	20
2.2.1 PyTorch	20
2.2.2 TensorFlow	21
2.2.3 Keras	22
2.3 Выбор IDE	24
2.3.1 PyCharm	24
2.3.2 Spyder	25
2.3.3 VS Code	26
3 Проектирование ПО	27
3.1 YOLOv8	28
3.2 UniDepth	31
3.3 Архитектура разрабатываемой системы	33
4 Программная реализация	37
4.1 Настройка окружения	37
4.2 Описание модулей	38
5 Тестирование	46

5.1 Формирование тестового набора данных	46
5.2 Тестирование работы ПО	48
6 Безопасность жизнедеятельности	53
6.1 Вред здоровью при работе с компьютером	53
6.2 Снижение вредного воздействия на здоровье при работе с ПК	53
6.3 Техника безопасности при работе с компьютером	54
Заключение	55
Список использованных источников	56
Приложение А – UML-диаграммы	58
Приложение А.1 – Диаграмма классов	59
Приложение А.2 – Диаграмма прецедентов	60
Приложение А.3 – Диаграмма последовательностей	61
Приложение А.4 – Диаграмма компонентов	62
Приложение Б – Листинг программы	63
Приложение В – Презентация	80

Перечень сокращений и обозначений

В данной дипломной работе были применены следующие сокращения и обозначения

ПК – персональный компьютер

ПО – программное обеспечение

GPU (англ. Graphics Processing Unit) – графический процессор

IDE (англ. Integrated Development Environment) – интегрированная среда разработки

KITTI (англ. Karlsruhe Institute of Technology and Toyota Technological Institute) – набор данных, который ориентирован для исследований в области робототехники и автономном вождении

mAP (англ. mean Average Precision) – метрика, которая показывает среднюю точность обнаружения объектов на изображении

YOLO (англ. You Only Look Once) – архитектура нейронных сетей, предназначенная для детекции объектов на изображении. Отличительной особенностью YOLO является подход к решению задачи детекции

					ПГУ 1.090301.40.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

С развитием технологий нейронные сети становятся всё более важным инструментом в различных областях науки и техники. Модели глубокого обучения решают очень много задач в области обработки изображений. Одной из таких задач является измерение расстояния от объектов на изображении до модуля камеры. Использование нейронных сетей для этой цели открывает новые перспективы в обработке данных и повышении точности измерений.

Традиционные методы измерения расстояния, такие как использование лазерных дальномеров или стереозрение, имеют определенные ограничения в точности и применимости. В то время как нейронные сети, благодаря своей способности к обучению на больших объемах данных и адаптации к сложным условиям, могут предложить новые подходы к решению проблемы измерения расстояния [1].

Целью данной дипломной работы является исследование возможностей применения нейронных сетей для измерения расстояния. В работе будут рассмотрены основные принципы работы нейронных сетей, а также методы их применения в задачах измерения расстояния. Также будет проведен анализ результатов экспериментов с различными архитектурами нейронных сетей для измерения расстояния с использованием различных типов входных данных.

Данный дипломный проект внедряет перспективные подходы к усовершенствованию методов измерения расстояния, что представляет значительный потенциал для инноваций в различных сферах применения. Полученные результаты и разработанные технологии могут содействовать созданию более точных и эффективных средств измерения, что способствует развитию областей робототехники, медицинской диагностики, автомобильной навигации и т.д.

1 Обзор предметной области

1.1 Анализ темы

На сегодняшний день, задача измерения расстояния от камеры до объекта на изображении имеет широкий спектр применений в различных сферах жизнедеятельности человека. Ниже описаны возможные области применения, где решение поставленной задачи может быть востребовано:

- автомобильная промышленность;
- медицина;
- робототехника;
- виртуальная и дополненная реальность;
- безопасность и наблюдение.

В области автомобильной промышленности автопилоты и системы помощи водителям требуют точного определения расстояния до других транспортных средств, пешеходов, препятствий и дорожных знаков на изображениях с камер. Измерение расстояния на изображениях помогает определять безопасное расстояние для автомобилей и позволяет системам аварийного торможения реагировать на опасные ситуации [2].

В медицине измерение размеров внутренних органов и опухолей на медицинских изображениях (например, снимках МРТ, КТ) может помочь лечащему врачу в диагностике и планировании дальнейшего лечения пациента. Определение расстояния до медицинских инструментов и аппаратуры на хирургических изображениях помогает врачам-хирургам более эффективно управлять процессом операции [3].

В области робототехники роботы, использующие зрительные системы, могут измерять расстояние до объектов для навигации в неизвестной среде и выполнения различных задач, таких как сортировка, упаковка и сборка [4]. Дроны и автономные летательные аппараты могут использовать измерение расстояния на изображениях для навигации и избегания столкновений в реальном времени.

Что касается AR/VR, то виртуальные и дополненные системы реальности используют измерение расстояния для более точного отображения виртуальных объектов в сцене и взаимодействия пользователя с ними [5]. Также оценка расстояния до объектов вокруг пользователя помогает создавать более реалистичные и интуитивно понятные интерфейсы виртуальной и дополненной реальности.

В сфере безопасности системы видеонаблюдения используют измерение расстояния на изображениях с камеры для обнаружения подозрительных объектов или определения опасных ситуаций в общественных местах. Стоит также упомянуть, что определение расстояния до наблюдаемых объектов позволяет более эффективно контролировать доступ к защищенным объектам [6].

Все рассмотренные области возможного применения измерения расстояния от камеры до объекта на изображении указывают на важность данной темы на сегодняшний день в человеческой жизнедеятельности.

Таким образом, развитие и применение технологий измерения расстояния на изображениях играет важную роль в современном обществе, способствуя повышению эффективности, безопасности и улучшению качества жизни людей.

1.2 Анализ технологий

Для того, чтобы эффективно решить поставленную задачу необходимо выполнить два основных этапа, каждый из которых представляет из себя ключевой компонент всего процесса.

Первый этап — обнаружение объекта. На этом этапе особое внимание уделяется разработке и применению алгоритмов и методов, которые помогают точно обнаружить интересующий объект на изображении или в видеопотоке. Данный этап включает в себя использование современных технологий компьютерного зрения и машинного обучения для автоматического распознавания ключевых признаков искомого объекта. Например, для решения задачи обнаружения объекта, можно применять сверточные нейронные сети, каскады

Хаара и многие другие подходы, которые способны обнаруживать объекты различных форм, размеров и классов.

Вторым этапом является, непосредственно, измерение расстояния. Задача данного этапа состоит в том, чтобы определить расстояние до обнаруженного объекта на изображении до камеры. Для решения этой задачи может потребоваться использование различных методов измерения расстояния, таких как анализ карты глубины изображения, использование лазерных дальномеров или других технологий. В качестве примера, можно упомянуть про техники машинного обучения для измерения расстояния, которые способны прогнозировать карту глубины всего изображения в реальных единицах измерения.

В итоге, разделение поставленной задачи на эти две подзадачи позволит более эффективно решить исходную проблему и получить более точные и верные результаты.

1.2.1 Обнаружение объекта

На сегодняшний день, задача обнаружения объектов на изображениях имеет большое разнообразие методов решения, которые охватывают широкий спектр подходов от традиционных методов, основанных на пороговой обработке, до современных методов, которые используют модели глубокого обучения. Среди последних наиболее эффективным методом считается тот, который базируется на применении сверточных нейронных сетей. В настоящее время именно этот метод демонстрирует самые высокие показатели как по производительности, так и по точности обнаружения объектов на изображениях.

На сегодняшний момент в области глубокого обучения существует несколько ведущих архитектур нейронных сетей, которые успешно решают задачу детекции объектов на изображении. Среди них можно выделить такие как:

- Faster R-CNN;
- YOLO (You Only Look Once);
- SSD (Single Shot MultiBox Detector);
- Mask R-CNN;

– RetinaNet.

Faster R-CNN – это алгоритм обнаружения объектов, который объединяет методы регионального предложения (Region Proposal Network, RPN) и сверточные нейронные сети (Convolutional Neural Networks, CNN). RPN генерирует предложения областей, которые могут содержать объекты, а затем эти предложения классифицируются и точно обрамляются при помощи CNN. Он известен своей высокой точностью и эффективным использованием ресурсов. Однако у Faster R-CNN есть недостатки, такие как относительно медленная скорость обработки изображений, сложность в реализации и требовательность к данным для обучения [7].

Следующим по очереди идёт алгоритм YOLO. YOLO представляет собой алгоритм обнаружения объектов на изображениях. Он отличается от многих других методов тем, что применяется к изображению только один раз, что обеспечивает высокую скорость обработки. Алгоритм основан на глубокой сверточной нейронной сети, которая разбивает изображение на сетку ячеек и для каждой ячейки предсказывает прямоугольники и вероятности объектов. Это позволяет YOLO обнаруживать несколько объектов на одном изображении. Его сильные стороны включают высокую скорость обработки, способность обнаруживать несколько объектов и относительную простоту в реализации. Однако есть и недостатки, такие как ограниченная точность, проблемы с мелкими объектами и перекрытием объектов на изображении [8]. В целом, YOLO остается одним из самых популярных и широко используемых методов обнаружения объектов на изображениях благодаря своей эффективности и простоте в использовании.

Что касается SSD (Single Shot MultiBox Detector), то данный метод обнаружения объектов на изображениях, который объединяет в себе этапы обнаружения и классификации. Он отличается высокой скоростью работы, позволяя обнаруживать объекты в реальном времени. Благодаря использованию единой сверточной сети для всех задач, он может быть более эффективным с точки зрения использования ресурсов. SSD также способен обнаруживать объекты различных размеров, используя множество фиксированных аспектных отношений.

					ПГУ 1.090301.40.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Однако у него есть некоторые ограничения: он не всегда эффективен при обнаружении мелких объектов, требует значительных вычислительных ресурсов и может иметь ограничения в точности обнаружения объектов с высокой детализацией из-за использования ограниченного разрешения сверточных слоев [9].

Mask R-CNN расширяет архитектуру Faster R-CNN, добавляя возможность точной сегментации объектов на изображении. Он включает в себя компоненты, такие как Region Proposal Network (RPN), Feature Pyramid Network (FPN) и ROIAlign для эффективного выделения объектов разного размера и точного вычисления их масок. Плюсы Mask R-CNN включают точную сегментацию объектов и многообразие применений, однако его сложность и вычислительная нагрузка могут быть минусами, требующими значительных ресурсов для обучения и применения модели. Важно также иметь достаточный объем размеченных данных для эффективного обучения модели [10].

Последним рассматриваемым детектором является RetinaNet. Данная нейронная сеть была разработана для обнаружения объектов на изображениях с использованием Focal Loss для борьбы с проблемой дисбаланса классов. Она известна своей высокой точностью и способностью обнаруживать объекты разных размеров одновременно благодаря использованию Feature Pyramid Network (FPN). Однако, для обработки изображений с высоким разрешением или с плотными объектами, её использование может потребовать значительных вычислительных ресурсов. Кроме того, на изображениях с низким разрешением могут возникнуть трудности при обнаружении мелких объектов [11].

Все эти архитектуры представляют собой важные инструменты в области обнаружения объектов на изображениях, и выбор конкретной зависит от конкретной задачи и требований к ресурсам.

1.2.2 Измерение расстояния

В сфере исследований, посвященных оценке глубины сцены на основе изображений, обнаруживается необходимость в тщательном рассмотрении различных методов, разработанных для достижения этой цели. Это обеспечивает

не только обзор существующих подходов, но и выявление их сильных и слабых сторон в контексте решения конкретных задач.

В данной работе предпринимается попытка охватить широкий спектр методов, используемых для оценки глубины сцены на основе изображений. Среди них выделяются следующие:

- метод подобия треугольников;
- метод стереозрения;
- LiDAR;
- метод глубокого обучения.

В рамках данного исследования рассматривается геометрический метод, который может быть применен для измерения расстояния от объекта до камеры на основе известных размеров объекта на изображении и дополнительной информации о его геометрии. Принцип работы метода подобия треугольников основан на том, что при подобии двух треугольников отношения длин их сторон равны, а отношения площадей соответствуют квадратам этих длин [12].

Для применения данного метода к задаче измерения расстояния от объекта до камеры предлагается следующий алгоритм:

- определение размеров объекта на изображении. Путем измерения высоты, ширины или диаметра объекта на изображении в пикселях устанавливаются его размеры;
- оценка физических размеров объекта. С помощью известных физических размеров объекта в реальном мире, например, с использованием линейки, определяются его физические размеры;
- расчёт расстояния. Используя фокусное расстояние камеры (F), размеры объекта на изображении ($Size_{image}$) и его физические размеры ($Size_{real}$), производится вычисление расстояния от объекта до камеры (D) по формуле:

$$D = \frac{F \times Size_{real}}{Size_{image}}. \quad (1.1)$$

Предложенный метод представляет собой относительно простой и точный способ оценки расстояния от объекта до камеры, основанный на геометрических

принципах подобия треугольников. Однако для его применения необходимо знать физические размеры объекта, что может являться ограничивающим фактором в некоторых случаях.

Следующим по очереди методом измерения расстояния является метод стереозрения. Данный метод представляет собой технику компьютерного зрения, использующую пару камер для измерения глубины сцены. Он основан на принципе стереозрения, который заключается в том, что объект, наблюдаемый с двух различных точек зрения, создает немного различающиеся изображения на каждой камере. Эти различия могут быть использованы для определения расстояния до объекта.

Метод включает в себя процесс сопоставления каждой точки на одном изображении с соответствующей ей точкой на другом изображении. После этого проводится триангуляция, позволяющая определить координаты прообраза точек в трехмерном пространстве. Зная трехмерные координаты прообраза, глубина объекта вычисляется как расстояние от камеры до плоскости, на которой находится объект. Принцип работы данного метода отображен на рисунке 1.1.

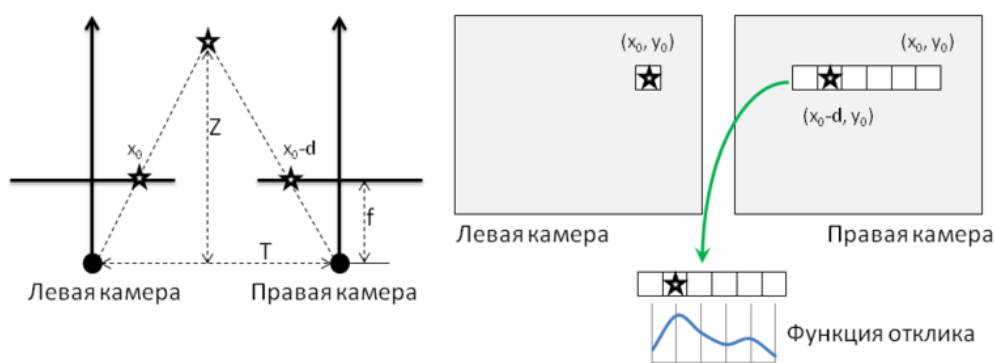


Рисунок 1.1 – Вычисление карты глубины

Для упрощения процесса сопоставления точек на изображениях используется метод ректификации, который выравнивает изображения таким образом, чтобы все эпиполярные линии стали параллельны сторонам изображения, обычно горизонтальным. Это достигается путем выравнивания эпиполярных линий так, чтобы для каждой точки с координатами (x_0, y_0) на первом изображении

соответствующая ей эпиполярная линия на втором изображении задавалась уравнением $x = x_0$.

Процесс ректификации включает в себя ремеппинг изображений и коррекцию дисторсий. Результатом являются ректифицированные изображения, на которых точки, соответствующие друг другу, находятся в одной и той же строке изображения. Примеры ректифицированных изображений представлены на рисунке 1.2.

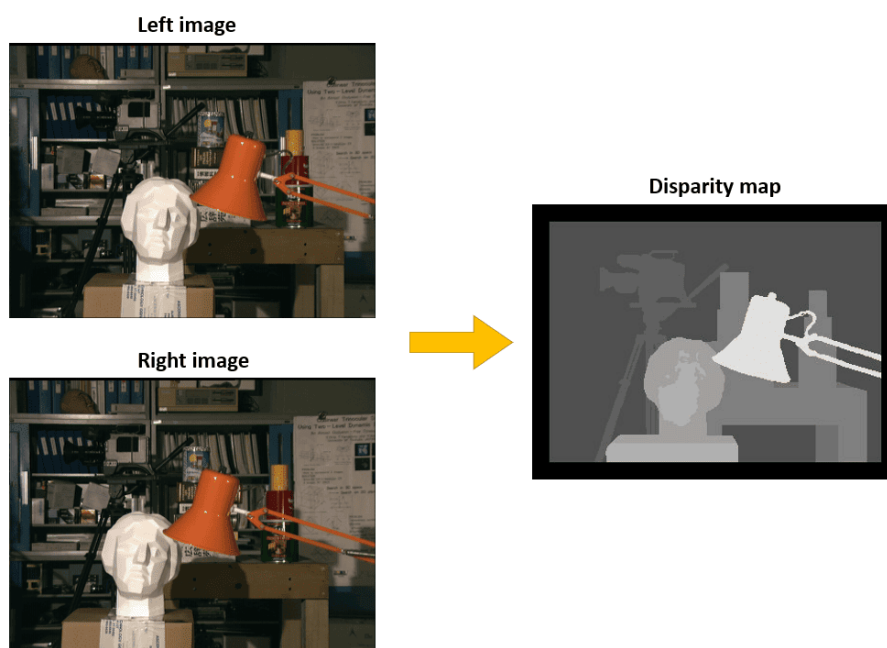


Рисунок 1.2 – Пример ректифицированных изображений, и соответствующей им disparity map

Для каждого пикселя на левом изображении с координатами (x_0, y_0) осуществляется поиск соответствующего пикселя на правом изображении. Предполагается, что координаты искомого пикселя на правом изображении равны $(x_0 - d, y_0)$, где d представляет собой смещение, также известное как диспаратность.

Фактически, значения глубины обратно пропорциональны величине смещения пикселей. Если использовать обозначения, представленные на левой половине рисунка 1.1, зависимость между диспаратностью и глубиной может быть выражена следующим образом:

$$\frac{T-d}{Z-f} = \frac{T}{Z} \rightarrow Z = \frac{fT}{d}, \quad (1.2)$$

Вследствие обратной зависимости между глубиной и диспаратностью, разрешающая способность систем стереозрения, основанных на данном методе, демонстрирует более высокую эффективность на близких расстояниях и снижается на дальних [13].

Преимущество данного метода заключается в возможности получения точных измерений расстояния до объектов на сцене без предварительного знания параметров объектов. Однако следует отметить, что данный метод может ограничиваться сложностью обработки данных и требовательностью к вычислительным ресурсам, необходимым для расчета глубины сцены.

Light Detection and Ranging (LiDAR) представляет собой технологию измерения расстояний и определения формы объектов с применением лазерного излучения. Она основана на принципе измерения времени, необходимого для того, чтобы лазерный импульс отразился от объекта и вернулся обратно к детектору.

Один из методов использования LiDAR с камерой для измерения расстояний называется методом временной развертки (time-of-flight). В этом методе LiDAR генерирует короткие лазерные импульсы, направленные на объект. После отражения от объекта импульс возвращается к детектору LiDAR. Измерение времени, прошедшего от момента отправки импульса до его возврата, используется для определения расстояния до объекта [14].

При использовании камеры в сочетании с LiDAR ее изображение используется для точного определения местоположения объекта в пространстве. Камера помогает сопоставить данные, полученные с помощью LiDAR, с визуальной информацией об объекте на сцене. Общая иллюстрация работы технологии LiDAR представлена на рисунке 1.3.

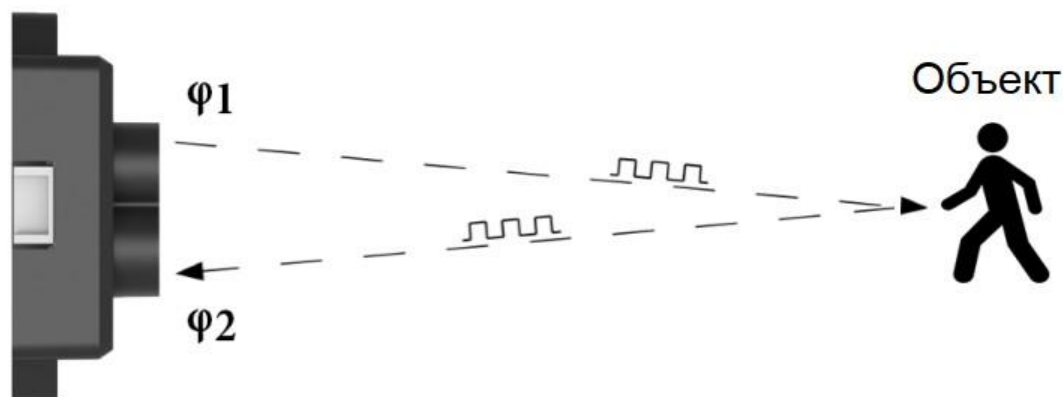


Рисунок 1.3 – Иллюстрация работы LiDAR

Используя технологию LiDAR, расчет расстояния до объекта может быть выполнен согласно формуле (1.3):

$$D = \frac{c}{2} \times \frac{1}{2\pi f} \times \Delta\varphi, \quad (1.3)$$

где f – частота сканирующих импульсов;

c – скорость света в оптической среде;

$\Delta\varphi$ – фазовый сдвиг;

D – искомое расстояние.

Таким образом, интеграция камеры с LiDAR позволяет эффективно решать поставленные задачи. Однако главным недостатком данной технологии является ее высокая стоимость.

Что касается метода использования моделей глубокого обучения, то оценка глубины, как ключевой аспект в области компьютерного зрения, осуществляется путем измерения расстояния от каждого пикселя изображения до камеры, основываясь на монокулярных или стереоизображениях разнообразных сцен. Традиционные подходы включают в себя применение различных геометрических методов для анализа взаимосвязи между изображениями. Новаторские техники, появившиеся в последнее время, позволяют осуществлять оценку глубины непосредственно, с минимальными потерями, путем регрессии или создания нового представления на основе последовательности изображений. Популярные

тестовые наборы данных, такие как KITTY и NYUv2, широко используются для оценки эффективности моделей [15].

Недавние исследования в области разработки архитектур моделей привели к созданию нескольких успешных решений, которые продемонстрировали высокую эффективность в прогнозировании карт глубины. Эти достижения стали возможны благодаря развитию методов глубокого обучения и продвинутому пониманию процессов извлечения признаков из данных для решения сложных задач в области компьютерного зрения.

1.3 Постановка задачи

Целью данной дипломной работы является разработка программного обеспечения для измерения расстояния от камеры до объекта на изображении. Для достижения поставленной цели предполагается использование и интеграция двух моделей нейронных сетей для обнаружения объектов и для прогнозирования карты глубины изображения.

Реализация программного обеспечения предполагает создание пакета модулей, способного анализировать цветные изображения и объединять результаты работы двух моделей. Для достижения этой цели будут использованы современные инструменты и технологии глубокого обучения.

Системные требования ПО:

- ОС. Windows, macOS или Linux;
- CPU. Многоядерный процессор (4 и более ядер);
- GPU. GPU с поддержкой CUDA;
- HDD/SSD. Минимальное пространство – 25 ГБ.

В конечном итоге, разрабатываемое программное обеспечение должно корректно обеспечивать интеграцию двух моделей глубокого обучения, на основе обработки которых, будет определяться и прогнозироваться расстояние от объектов на изображении до камеры.

2 Выбор технологий разработки

В данном разделе был проведён анализ существующих технологий и инструментов разработки, которые нужны в реализации поставленной задачи. Рассмотрены особенности языков программирования в области машинного обучения. Также были рассмотрены фреймворки глубокого обучения и интегрированные инструменты разработки.

2.1 Выбор языка программирования

Выбор языка программирования играет ключевую роль в разработке ПО. Ниже рассмотрены несколько языков программирования, учитывая их эффективность в области обработки изображений, наличие библиотек компьютерного зрения и фреймворков глубокого обучения. В настоящее время Python, C++ и C# выделяются как одни из наиболее популярных языков программирования для работы с нейронными сетями и решения задач компьютерного зрения.

2.1.1 Язык программирования Python

Python — это язык программирования, который используется в интернет-приложениях, разработке программного обеспечения, науке о данных и машинном обучении. Разработчики используют Python, потому что он эффективен, прост в изучении и работает на разных платформах.

Python обладает богатым выбором фреймворков для работы с моделями глубокого обучения, такими как TensorFlow, PyTorch и Keras, которые обеспечивают широкий спектр инструментов для создания, обучения и применения различных архитектур нейронных сетей.

Язык Python имеет огромное количество библиотек, включая одну из самых популярных библиотек компьютерного зрения OpenCV, которая предоставляет возможности для работы с изображениями, обнаружения объектов и измерения расстояний.

Из недостатков стоит отметить то, что интерпретируемость Python делает данный язык менее производительным по сравнению с компилируемыми языками программирования, такими как C++.

2.1.2 Язык программирования C++

C++ — это язык программирования, который был разработан в 80-х годах прошлого века как расширение языка C. Этот язык отличается от Си тем, что имеет больший набор возможностей, которые включают объектно-ориентированное программирование и шаблоны. Высокая производительность и прямой доступ к памяти делают данный язык программирования хорошим выбором для создания сложных архитектур нейронных сетей и работы с графикой.

Преимуществом данного языка является то, что в C++, также как и в Python, имеется библиотека компьютерного зрения OpenCV, которая предоставляет широкий набор функций для обработки изображений, детекции объектов и других задач компьютерного зрения.

К недостаткам относится то, что выбор библиотек для работы с нейронными сетями на C++ не так обширен, как в Python, даже с учётом того, что существуют такие фреймворки, как TensorFlow и Caffe, которые имеют собственные интерфейсы на этом языке программирования.

2.1.3 Язык программирования C#

C# – это объектно-ориентированный язык программирования. Он был создан в период с 1998 по 2002 год командой инженеров Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота.

Язык входит в семью C-подобных языков. Синтаксис приближен к Java и C++. C# популярен за счет своей «простоты». Простоты для современных программистов и больших команд разработчиков, чтобы те могли в сжатые сроки создавать функциональные и производительные приложения. Этому способствуют нетипичные конструкции языка и специфичный синтаксис, помогающий максимально органично реализовать намеченные функции. В области компьютерного зрения на C# существует несколько библиотек, которые предоставляют инструменты для обработки изображений и решения различных

задач компьютерного зрения. Например, Emgu CV представляет собой обертку для библиотеки OpenCV, которая позволяет использовать функционал OpenCV в C#-приложениях.

Язык программирования C#, также как и языки Python и C++, обладает фреймворками глубокого обучения, которые предоставляют возможности для работы с нейронными сетями на этом языке. Например, ML.NET — это фреймворк для машинного обучения, разработанный Microsoft, который включает инструменты для создания, обучения и использования нейронных сетей.

Несмотря на сильные стороны C++ и C#, Python чаще всего выбирается для разработки в области компьютерного зрения и нейронных сетей благодаря обширному функционалу и удобству. Большинство ведущих архитектур нейронных сетей имеют реализацию именно на этом языке. После проведенного анализа, в качестве основного языка программирования для разработки программного обеспечения был выбран язык Python.

2.2 Выбор фреймворка

После выбора основного языка программирования необходимо выбрать фреймворк глубокого обучения для работы с нейронными сетями. Именно поэтому самым важным фактором является его совместимость с языком программирования Python. Будут рассмотрены и другие факторы, такие как производительность, гибкость и наличие поддержки сообщества пользователей. Ниже описаны наиболее популярные фреймворки машинного обучения, а именно PyTorch, TensorFlow и Keras.

2.2.1 PyTorch

PyTorch — это фреймворк для языка программирования Python, предназначенный для машинного обучения. Он включает в себя набор инструментов для работы с моделями, используется в обработке естественного языка, компьютерном зрении и других похожих направлениях. Данный фреймворк создан на базе Torch, библиотеки для языка Lua, которая предназначена для математических расчетов и машинного обучения. Он обладает широким списком

					ПГУ 1.090301.40.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

инструментов для разработки и обучения моделей глубокого обучения, включая рекуррентные нейронные сети, сверточные нейронные сети и другие. PyTorch широко применяется как в научных исследованиях, так и в промышленных приложениях, благодаря своей гибкости и производительности.

Главным преимуществом PyTorch является его динамический граф вычислений, который строится по мере выполнения кода. Граф не существует как фиксированная структура, а динамически выстраивается непосредственно при вычислениях. Это обеспечивает гибкость в определении и изменении архитектуры моделей в процессе обучения, что значительно упрощает экспериментирование с различными типами архитектур и разработку самих моделей глубокого обучения.

Также стоит отметить, что данный фреймворк наследует идеологию ЯП Python, а именно простоту синтаксиса. PyTorch предоставляет пользователю довольно обширный набор функций и модулей для создания, обучения и эксплуатации разнообразных архитектур нейронных сетей, а также включает различные методы оптимизации моделей и основные функции потерь для обучения.

Стоит сказать и про недостатки, среди которых следует упомянуть, что главное преимущество данного фреймворка, а именно динамический граф вычислений, имеет более низкую производительность при выполнении сложных архитектур моделей глубокого обучения на больших наборах данных.

Но несмотря на это, PyTorch всё ещё остается одним из самых популярных фреймворков машинного обучения. Самые популярные архитектуры имеет свою реализацию именно на PyTorch. Кроме этого, активная поддержка сообщества помогает в разработке и использовании моделей глубокого обучения.

2.2.2 TensorFlow

TensorFlow — открытый Python-фреймворк машинного обучения (Machine Learning), созданный командой Google Brain, объединяет множество моделей и алгоритмов глубокого обучения (Deep Learning). TensorFlow предоставляет API-интерфейс на Python, а сами вычисления происходят на высокопроизводительном C++. Данный фреймворк может обучать глубокие нейронные сети для

классификации рукописных цифр, распознавания изображений, встраивания слов, рекуррентных нейронных сетей, моделей для машинного перевода, обработки естественного языка (NLP) и т.д.

Самым большим преимуществом TensorFlow является абстракция. Вместо того, чтобы заниматься мельчайшими деталями реализации алгоритмов или выяснять правильные способы привязки вывода одной функции ко входу другой, разработчик может сосредоточиться на общей логике приложения.

Также стоит отметить, что TensorFlow имеет высокую производительность при использовании графических процессоров, которые используются для ускорения матричных вычислений.

Главный недостаток TensorFlow – статический граф вычислений, который менее интуитивно понятен для новичков и требует дополнительных усилий для отладки и изменения архитектур моделей. Но начиная с версии TensorFlow 2.0 данный недостаток был утрачен. Также фреймворк менее гибок по сравнению с некоторыми альтернативами, что может затруднять работу с нестандартными моделями или экспериментами.

В общем и целом, TensorFlow является популярным фреймворком для разработки модели глубокого обучения, который обеспечивает обширный спектр возможностей разработки архитектур моделей и хорошую производительность их выполнения. Но во внимание стоит учесть, что у него более высокий входной порог для неопытного разработчика архитектур нейронных сетей, даже при наличии подробной документации.

2.2.3 Keras

Keras — это библиотека для языка программирования Python, которая предназначена для глубокого машинного обучения. Она позволяет быстрее создавать и настраивать модели — схемы, по которым распространяется и подсчитывается информация при обучении. Но сложных математических вычислений Keras не выполняет и используется как надстройка над другими библиотеками.

Одним из ключевых преимуществ Keras является то, что он предлагает поддержку несколько бэкенд-движков, включая Tensorflow, Theano и CNTK. Любой из них может быть выбран на основе требований проекта. Также стоит упомянуть что у Keras один из лучших примеров документации. Она представляет каждую функцию последовательно и очень подробно. Примеры кода также полезны и просты для понимания. У Keras есть отличная поддержка со стороны сообщества. Многие разработчики предпочитают Keras для участия в соревнованиях по Data Science. Также многие исследователи публикуют свой код и руководства для широкой аудитории.

Keras очень медленный при работе на GPU, а его операции занимают больше времени в сравнении с бэкендом. Поэтому скоростью приходится жертвовать в угоду удобству использования. Также инструменты Keras для подготовки данных не так хороши, как в случае с другими пакетами, например scikit-learn. Они не подходят для построения базовых алгоритмов машинного обучения: кластерного анализа или метода главных компонент. Нет и возможности динамического создания графиков.

Keras остается популярным выбором для быстрого прототипирования и разработки нейронных сетей благодаря своей простоте использования и гибкости. Этот фреймворк подходит как для начинающих, так и для опытных исследователей в области машинного обучения и глубокого обучения.

После анализа трех вышеупомянутых фреймворков для глубокого обучения было принято решение использовать PyTorch в качестве основного инструмента разработки. Это решение обусловлено рядом ключевых факторов, делающих PyTorch предпочтительным выбором. Во-первых, динамический граф вычислений в PyTorch предоставляет хорошую гибкость при создании архитектур глубокого обучения. Этот подход упрощает экспериментирование с различными моделями и изменениями архитектуры прямо в процессе обучения. Во-вторых, PyTorch очень эффективно использует графические процессоры, что способствует ускорению процесса обучения на больших объемах данных. Данная особенность позволяет эффективно обучать сложные модели нейронных сетей.

Наконец, активное сообщество пользователей PyTorch обеспечивает доступ к обширной базе знаний, ресурсам и литературе, что является важным аспектом при разработке и отладке моделей глубокого обучения.

Таким образом, выбор PyTorch обоснован его техническими возможностями, эффективностью использования ресурсов и поддержкой сообщества, что делает его предпочтительным инструментом для разработки алгоритмов глубокого обучения.

2.3 Выбор IDE

При выборе интегрированной среды разработки ПО для решения, поставленной задачи на языке программирования Python и с использованием фреймворка PyTorch стоит учесть несколько очень важных аспектов, которые, в дальнейшем, могут повлиять на эффективность и удобство разработки.

Наиболее популярные IDE для разработки на языке программирования Python включают в себя следующие варианты:

- PyCharm;
- Spyder;
- VS Code.

Каждая из этих сред разработки имеет собственные особенности и преимущества, которые могут быть значимыми при разработке собственного проекта. Окончательное решение в выборе конкретной IDE зависит только от индивидуальных предпочтений разработчика, а также требований к функциональности, удобству использования и интеграции с фреймворком PyTorch.

2.3.1 PyCharm

PyCharm — кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик.

PyCharm имеет ряд преимуществ по сравнению с другими средствами разработки. Из основных, можно выделить интуитивно понятный интерфейс, который упрощает навигацию по проекту. Также данная среда разработки имеет

доступ к использованию отладчика для пошагового анализа выполнения кода. Интеллектуальный анализ кода поможет выявить потенциальные проблемы и предложить варианты их решения.

К недостаткам можно отнести то, что PyCharm имеет две версии: бесплатную и платную. Даже с учётом того, что бесплатная версия предоставляет обширный функционал для разработки своих проектов на Python, но только в платной версии предусмотрена работа с Jupyter Notebook, который очень часто применяется при разработке нейронных сетей. Также стоит отметить, что данная IDE требует большого количества ресурсов ПК, в частности, памяти и места на жёстком диске.

Для некоторых пользователей наличие платной версии с расширенными функциями может быть минусом, особенно если их требования к функциональности не оправдывают необходимость приобретения лицензии. Тем не менее, бесплатная версия PyCharm Community Edition также предоставляет множество инструментов и возможностей для эффективной разработки на Python, включая поддержку фреймворка PyTorch.

2.3.2 Spyder

Spyder — это бесплатная научная среда разработки с открытым исходным кодом, написанная на языке программирования Python для Python и разработанная учеными, инженерами и аналитиками данных. Spyder представляет собой уникальное сочетание расширенных функций редактирования, анализа, отладки и профилирования комплексного инструмента разработки с возможностями исследования данных, интерактивного выполнения, глубокой проверки и прекрасной визуализации научного пакета. Она обеспечивает обширные возможности, включая встроенную консоль IPython, инструменты анализа данных и удобную интеграцию с основными библиотеками для научных вычислений.

Spyder имеет некоторые ограничения и недостатки. В частности, он не так популярен, как другие IDE для Python, такие как PyCharm или Visual Studio Code. Данный фактор сказывается на доступности ресурсов разработки, таких как обучающие материалы и сообщество разработчиков, что негативно влияет на

разработку проекта при работе с этой средой. Кроме того, Spyder имеет сложности с интеграцией систем контроля версий, что отталкивает неопытных разработчиков.

Spyder остается отличным выбором для разработки на Python, особенно в области научных вычислений, анализа данных и машинного обучения. Его специализированный функционал и интеграция с основными библиотеками делают его хорошим вариантом для разработчиков, нацеленных на выполнение научных задач и требующих высокой производительности.

2.3.3 VS Code

Visual Studio Code (VS Code) — текстовый редактор, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга.

VS Code является его обширная система плагинов. Большое количество таких плагинов, специально были разработаны для работы с Python, включая интеграцию с PyTorch, что позволяет возможность собственной настройки среды разработки в соответствии с индивидуальными потребностями разработчика. Visual Studio Code обладает широкой популярностью среди разработчиков благодаря своей многоплатформенной поддержке, простому интерфейсу и возможности интеграции с системами контроля версий, такими как Git. Благодаря активному сообществу разработчиков, в VS Code постоянно появляются новые плагины.

В сравнении с специализированными IDE, такими как Spyder, у VS Code, может быть, ограниченное количество интегрированных инструментов и библиотек для научных вычислений и анализа данных, что следует учитывать как недостаток данной среды разработки.

В результате анализа, в качестве интегрированной среды разработки был выбран PyCharm. Данное решение обосновывается наличием в PyCharm привлекательного интерфейса, превосходной интеграцией с системой контроля версий Git, а также обширным набором функций.

3 Проектирование ПО

Целенаправленное проектирование позволит создать эффективное и масштабируемое программное обеспечение, способное решать поставленную задачу с высокой точностью и надежностью.

В данной главе описан процесс проектирования программного обеспечения, целью которого является измерение расстояния от объекта на изображении до камеры. Как предварительно упоминалось, целесообразным представляется разбиение задачи на две подзадачи: обнаружение объектов на изображении и прогнозирование карты глубины для входного изображения.

Для решения первой подзадачи была выбрана модель YOLOv8, которая является современным одноэтапным детектором. В свою очередь, для решения подзадачи измерения расстояния была предпочтительно выбрана современная модель UniDepth, способная прогнозировать карту глубины для входного изображения.

На рисунке 3.1 приведено краткое описание структуры работы разрабатываемого продукта.

Входное изображение должно проходить необходимую предобработку для формирования тензора нужной размерности, который, в дальнейшем, поступит на вход моделям глубокого обучения. Перед моделью YOLOv8 стоит задачи обработать входные данные, сформировав ограничивающие рамки и наименование класса объекта на изображении. В свою очередь, модель UniDepth должна сформировать карту глубины входного изображения в реальных единицах измерения. После получения результатов моделей, нужно произвести обработку выходных тензоров и на их основе рассчитать расстояние от объекта до камеры. Далее происходит визуализация результата с помощью библиотеки OpenCV.

					ПГУ 1.090301.40.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

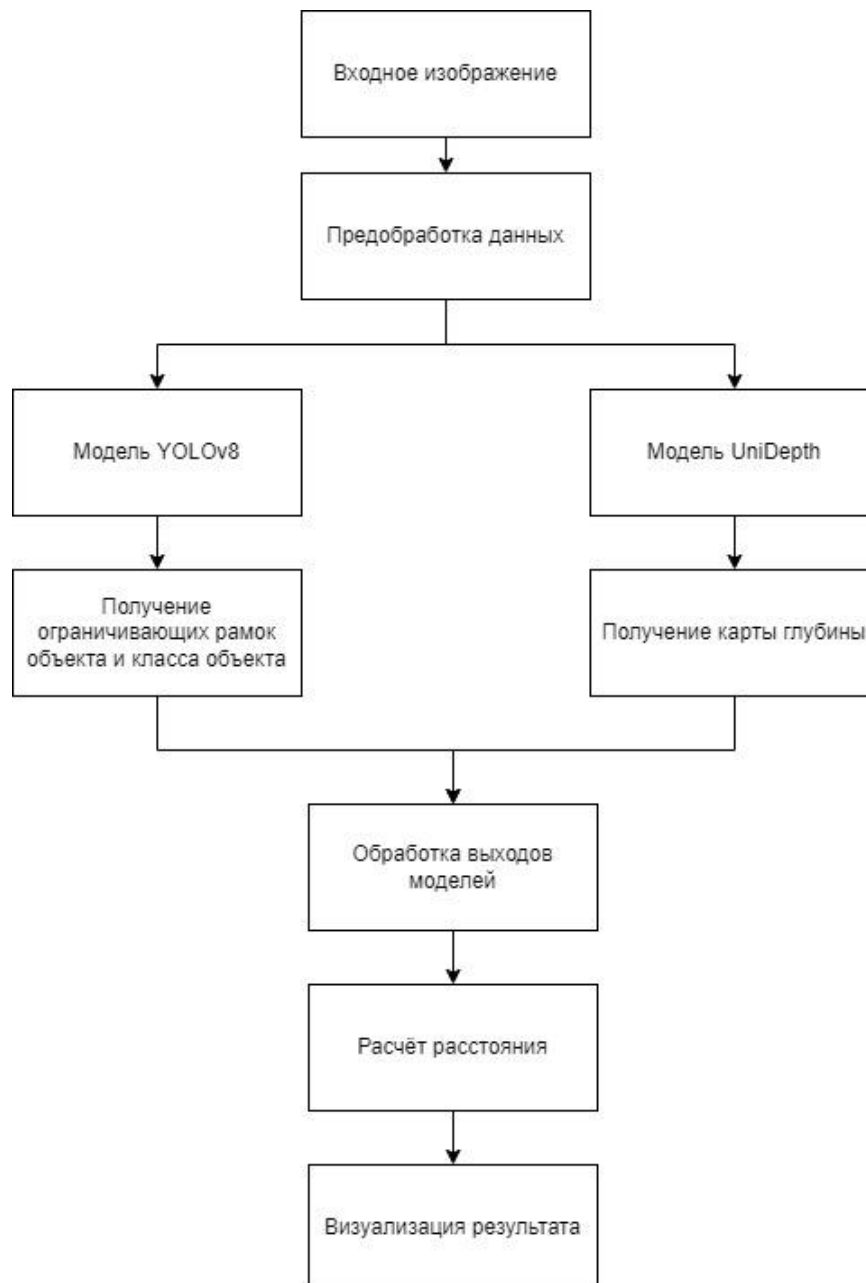


Рисунок 3.1 – Структура работы ПО

3.1 YOLOv8

YOLOv8 представляет собой модель обнаружения объектов на изображении, которая основана на архитектуре YOLO (You Only Look Once). Эта модель является улучшенной версией предыдущих моделей YOLO и предназначена для более точного и быстрого обнаружения объектов.

Архитектура YOLOv8 включает в себя несколько ключевых компонентов. Первый из них — это Backbone, который отвечает за извлечение признаков из

входного изображения. В YOLOv8 используется модифицированный вариант архитектуры CSPDarknet.

Далее идет компонент Neck, который объединяет признаки, извлеченные из различных слоев Backbone, и создает более общие признаки. В YOLOv8 Neck состоит из нескольких слоев, выполняющих различные функции, такие как уменьшение размерности и свертка.

И наконец, последний компонент — это Head. Эта часть модели отвечает за предсказание местоположения и классов объектов. В YOLOv8 Head также состоит из нескольких слоев, которые выполняют функции предсказания координат, классов и других характеристик объектов.

Иллюстрация архитектуры YOLOv8 представлена на рисунке 3.2.

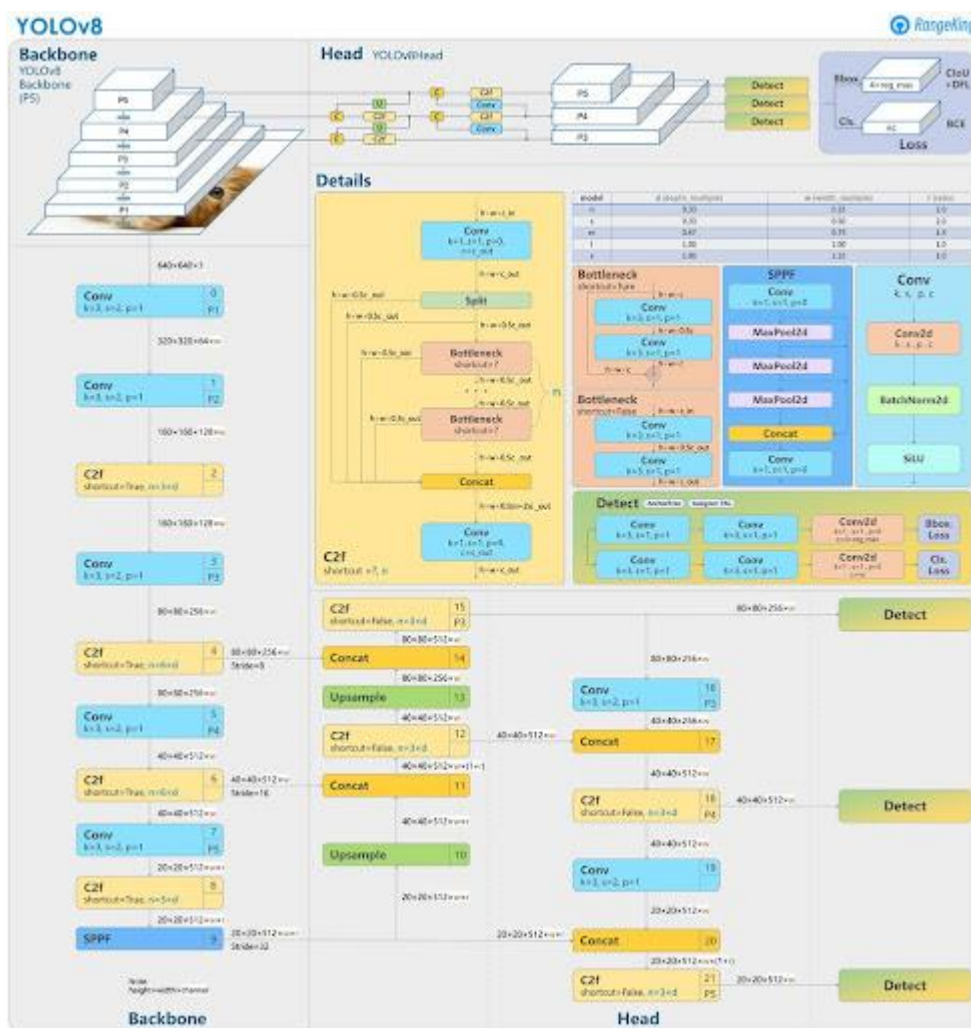


Рисунок 3.2 – Архитектура YOLOv8

Методика обнаружения объектов, используемая в YOLOv8, опирается на алгоритм YOLO, предполагающий, что каждый пиксель изображения может быть классифицирован как объект определенного класса. Обученная модель YOLOv8 анализирует большой набор данных, чтобы предсказывать вероятность того, что каждый пиксель является объектом конкретного класса, а также его координаты.

После завершения обучения YOLOv8 может применяться для обнаружения объектов на новых изображениях. Путем анализа входного изображения модель генерирует список объектов с соответствующими координатами и вероятностями принадлежности к определенным классам.

Репозиторий Ultralytics предоставляет возможность загрузки заранее обученных моделей с различной точностью. Всего доступно пять вариантов таких моделей, отличающихся производительностью и количеством параметров внутри архитектуры [16]. Подробное описание доступных моделей представлено в таблице 3.1.

Таблица 3.1 – Модели YOLOv8

Модель	Размер изображения	mAP 50-95	Количество параметров, млн.	FLOPs, млрд.
YOLOv8n	640	37.3	3.2	8.7
YOLOv8s	640	44.9	11.2	28.6
YOLOv8m	640	50.2	25.9	78.9
YOLOv8l	640	52.9	43.7	165.2
YOLOv8x	640	53.9	68.2	257.8

Метрика mAP50-95 представляет собой среднее значение точности (Precision) для порогов от 50 % до 95 %. Это позволяет оценить качество алгоритма обнаружения объектов на различных уровнях уверенности. Высокое значение mAP свидетельствует о более точном обнаружении объектов. Данная метрика широко

применяется в различных задачах, включая распознавание лиц, обнаружение пешеходов и другие.

Модели YOLOv8 были обучены на наборе данных от COCO, содержащем 80 предварительно обученных классов, и оценены с использованием значения mAP50-95 на наборе данных COCO val2017.

YOLOv8 обладает рядом преимуществ по сравнению с предыдущими моделями YOLO. Она обеспечивает более высокую точность обнаружения объектов и улучшенную скорость обработки изображений, что делает ее привлекательным выбором для задач, требующих быстрого и точного обнаружения объектов.

Однако следует отметить, что у модели YOLOv8 есть некоторые ограничения. Она может быть менее эффективной при обнаружении мелких объектов или объектов, находящихся на большом расстоянии от камеры. Кроме того, для успешного обучения требуется значительный объем данных, что может быть проблематично для некоторых пользователей.

Модель YOLOv8 представляет собой мощный инструмент для обнаружения объектов на изображениях. Она характеризуется высокой точностью и оперативностью работы, что делает ее пригодной для различных задач, включая автономную навигацию, обеспечение безопасности и другие области применения.

В контексте поставленной подзадачи, модель YOLOv8 будет принимать в качестве входных данных изображение и возвращать результаты в виде ограничивающих рамок объектов и их классификации.

3.2 UniDepth

UniDepth представляет собой передовую модель глубокого обучения, разработанную для точной монокулярной оценки метрической глубины на изображении. Она предназначена для восстановления метрических трехмерных сцен на входном изображении без использования дополнительной информации, стремясь к универсальному и гибкому решению данной задачи [17].

Архитектура модели UniDepth представлена двумя ключевыми модулями: модулем камеры и модулем глубины. На рисунке 3.3 приведена схема архитектуры нейронной сети UniDepth.

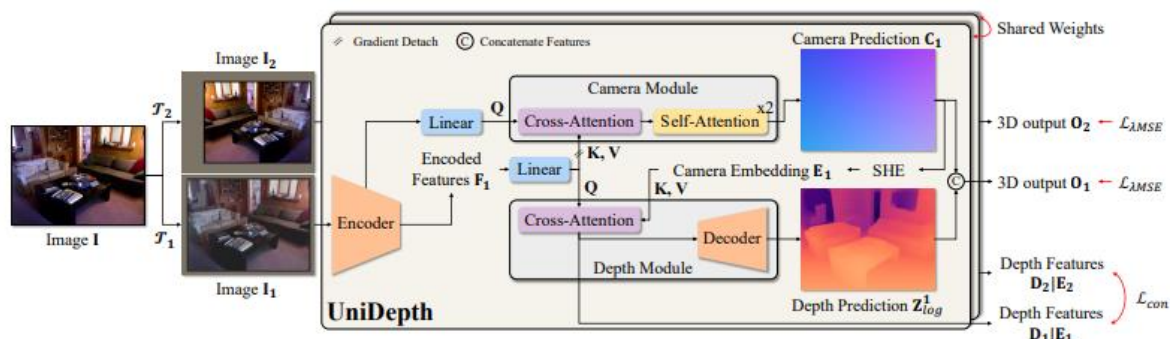


Рисунок 3.3 – Архитектура UniDepth

Модуль камеры играет значимую роль в формировании окончательных прогнозов трехмерных сцен, учитывая угловую плотность изображения для азимута и высоты. Эти встроенные характеристики помогают модулю глубины приблизительно определить глобальную шкалу глубины входной сцены, что способствует предотвращению сбоя в масштабировании сцены и снижает необходимость полного прогнозирования глубины.

Модуль глубины, в свою очередь, использует полный прогноз модуля камеры для формирования карты глубины для входного изображения.

UniDepth включает блок Encoder, который может быть как сверточной нейронной сетью, так и трансформером (ViT). Блок Decoder встроен в модуль глубины, откуда и происходит восстановление признаков и формирование карты глубины для входного изображения

Модель UniDepth проявляет превосходство по метрикам перед всеми своими конкурентами. Например, в специализированном бенчмарке от KITTI, она занимает первое место в рамках своей задачи. Детальное сравнение модели UniDepth с аналогами представлено в таблице 3.2.

Таблица 3.2 – Сравнение UniDepth с аналогами

Модель	SILog	sqErrorRel	absErrorRel	iRMSE
UniDepth	8.13	1.09	6.54	8.24
1PNet	9.46	1.45	7.62	9.75
MSFusion	9.59	1.56	7.81	10.32

Ключевой метрикой является SILog, которая показывает логарифмическую погрешность, не зависящую от масштаба $[\log(m) \times 100]$. Для спрогнозированной карты глубины (y) и истинной (y^*), каждая из которых содержит n пикселей, индексированных по i , можно определить SILog как:

$$D(y, y^*) = \frac{1}{n} \sum_i d_i^2 - \frac{1}{n^2} (\sum_i d_i)^2, \quad (3.1)$$

где:

$$d_i = \log y_i - \log y_i^*, \quad (3.2)$$

Метрика sqErrorRel показывает относительную квадратичную погрешность (в процентах), а метрика absErrorRel – относительную абсолютную погрешность (в процентах).

В случае метрики iRMSE, то данная метрика есть ничто иное, как среднеквадратичная погрешность определения обратной глубины $[\frac{1}{\text{км}}]$.

В конечном итоге, модель UniDepth возвращает карту глубины в реальных единицах измерения, в отличие от аналогов, таких как MiDAS или MonoDepth, для входного изображения. Карта глубины представлена тензором размерности $[1, 1, \text{Image}_{\text{WIDTH}}, \text{Image}_{\text{HEIGHT}}]$, где каждый элемент указывает на спрогнозированное расстояние от камеры до точки на изображении.

В результате анализа двух моделей, было принято решение объединить модели YOLOv8 и UniDepth в одну модель, которая будет прогнозировать расстояние от объекта на изображении до камеры, на основе выходов двух моделей.

3.3 Архитектура разрабатываемой системы

Цель данного подраздела заключается в подробном описании архитектуры модулей, их взаимосвязи и функциональности. Рассматривается структура

программного обеспечения, основные его компоненты, их предназначение и способы взаимодействия для обеспечения эффективной работы системы измерения расстояния от объектов на изображении до камеры.

Для достижения поставленной цели предполагается использование двух основных компонентов: модели обнаружения объектов и модели глубины. В качестве модели обнаружения объектов планируется использовать YOLOv8 – один из наиболее эффективных и точных алгоритмов обнаружения объектов в реальном времени. В свою очередь, для модели глубины будет использоваться UniDepth – инновационная модель, позволяющая предсказывать карту глубины изображения в реальных масштабах.

Предполагается, что пользователь сможет выбирать тип входных данных, которые он хочет обработать с помощью нейронных сетей, а именно изображения, видео или видеопоток с камеры. Для более детального описания использования разрабатываемого продукта была создана диаграмма вариантов использования, представленная на рисунке 3.4.

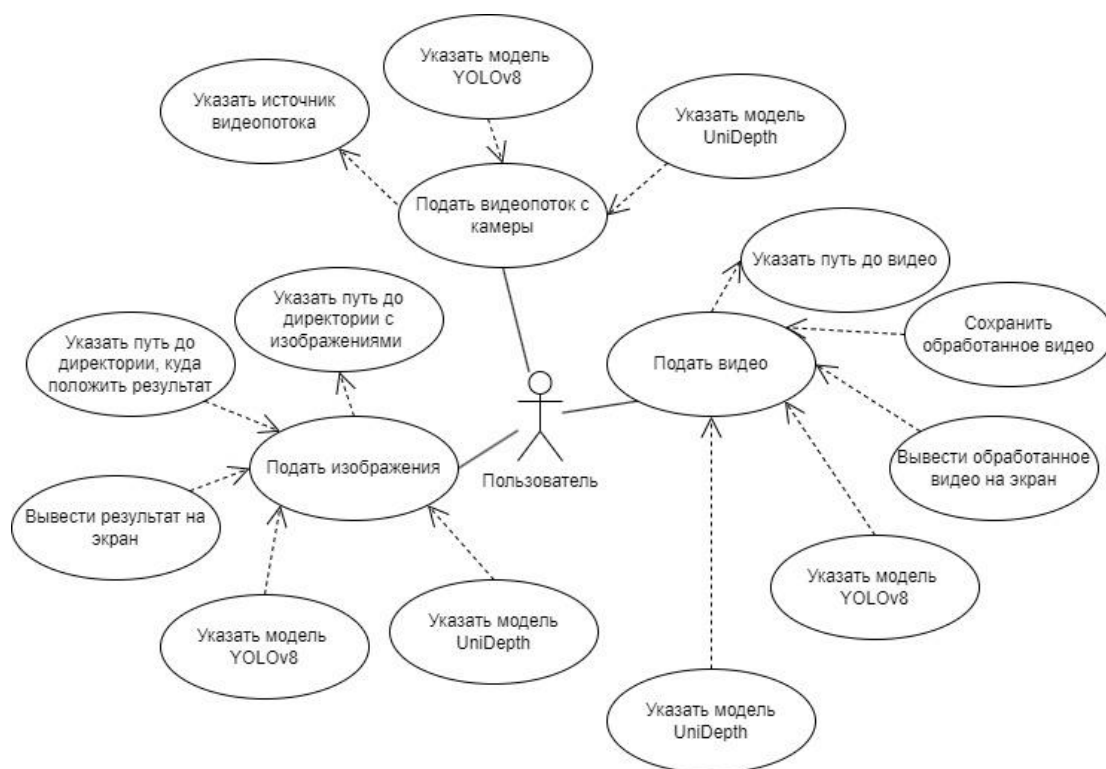


Рисунок 3.4 – Диаграмма вариантов использования

Как отражено на представленной выше диаграмме, пользователь имеет возможность выбора режима работы:

- изображения;
- видео;
- видеопоток с камеры.

В режиме "Изображения" пользователь должен указать путь к директории с изображениями, которые требуется обработать. Помимо этого, пользователь также может задать дополнительные аргументы, которые будут использоваться при обработке входных данных и после неё.

В случае выбора режима "Видео", пользователь должен указать путь к видеофайлу, который необходимо обработать. Программа также может принимать дополнительные параметры, такие как версии заранее обученных моделей YOLOv8 и UniDepth, путь для сохранения результатов обработки и указание на вывод обработанного видеопотока на экран устройства пользователя.

В режиме "Видеопоток с камеры" пользователь должен указать устройство, с которого будет производиться обработка кадров в реальном времени. Дополнительные параметры могут включать выбор моделей YOLOv8 и UniDepth.

В целях удовлетворения потребностей пользователя необходимо разработать систему модулей на языке программирования Python. Данная система будет реализовывать все требования, отображенные на диаграмме использования, приведенной на рисунке 3.4. На диаграмме компонентов, представленной на рисунке 3.5, продемонстрировано взаимодействие всех модулей системы.

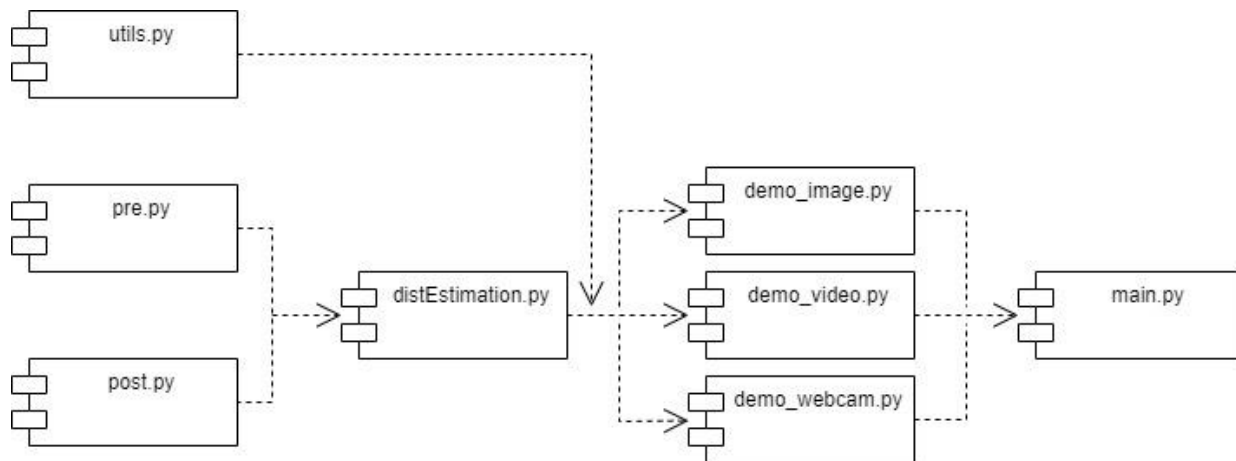


Рисунок 3.5 – Диаграмма компонентов

Согласно диаграмме компонентов (рисунок 3.5), пользователь будет взаимодействовать с одним основным файлом, а именно `main.py`, который принимает входные аргументы для управления последующей работой программы. Кроме того, присутствуют три модуля: `demo_image.py`, `demo_video.py` и `demo_webcam.py`, каждый из которых выполняет определенную задачу.

Однако ключевым модулем является `distEstimation.py`, который отвечает за интеграцию двух моделей YOLOv8 и UniDepth и расчет расстояния от объекта на изображении.

Модули `pre.py` и `post.py` отвечают соответственно за предварительную и последующую обработку входных и выходных данных для нейронных сетей.

Модуль `utils.py` содержит функции для десериализации аргументов, переданных пользователем в модуль `main.py`.

4 Программная реализация

В данном разделе описывается процесс создания программного обеспечения, включая выбор необходимых инструментов и реализацию.

Основной целью программной реализации является создание гибкого и эффективного инструмента, который способен обрабатывать изображения и видео с высокой точностью, определяя расстояние до обнаруженных объектов на изображении в реальном времени.

Для достижения этой цели были использованы современные модели компьютерного зрения, такие как YOLOv8 для обнаружения объектов и UniDepth для предсказания карты глубины изображения.

4.1 Настройка окружения

Для успешной реализации проекта и выполнения задач дипломной работы необходимо правильно настроить окружение разработки. В данном подразделе будет описана процедура настройки Python окружения с использованием инструмента управления пакетами conda.

После установки Anaconda на конечное устройство, процедура начинается с создания виртуального окружения с использованием следующей команды:

```
conda create -n <название окружения> python=3.10
```

Далее необходимо выполнить активацию предварительно созданного окружения:

```
conda activate <название окружения>
```

Для обеспечения корректной функциональности проекта необходимо установить инструментальный CUDA версии 11.8 с использованием средства управления пакетами conda. Для этого следует выполнить указанную команду:

```
conda install nvidia/label/cuda-11.8.0::cuda-toolkit
```

Затем необходимо установить все необходимые пакеты Python в пределах созданного окружения. Для этого рекомендуется перейти в каталог проекта и выполнить следующие команды:

```
pip install models/UniDepth --extra-index-url
https://download.pytorch.org/whl/cu118
pip install models/YOLOv8
```

После завершения всех указанных этапов настройки, окружение будет полностью подготовлено, и проект готов к использованию для достижения поставленных целей.

4.2 Описание модулей

В этом подразделе представлено детальное описание каждого модуля, включенного в конечный продукт. Каждый модуль разбирается с точки зрения его функциональности, обеспечиваемых возможностей, входных и выходных данных, а также методов и алгоритмов, реализованных в его рамках. Это позволяет полностью охватить характеристики каждого модуля, понять его цель и вклад в общую функциональность системы, а также понять, как он взаимодействует с другими компонентами.

Ниже приведён перечень всех модулей проекта:

- main.py;
- demo_image.py;
- demo_video.py;
- demo_webcam.py;
- distEstimation.py;
- pre.py;
- post.py;
- utils.py.

Основным модулем проекта является main.py, в котором прописана вся логика взаимодействия пользователя с пакетом. После отправки пользователем необходимых аргументов через CLI модулю main.py, данный модуль запустит

дочерний процесс, который начнёт исполнение одного из трёх модулей, а именно demo_image.py, demo_video.py или demo_webcam.py.

Каждый из трёх модулей содержит методы:

- preprocess();
- process();
- postprocess().

Метод preprocess() необходим для подготовки выбранных пользователем моделей и создания экземпляра класса DistEstimation. Ниже представлен код функции preprocess():

```
def preprocess() -> DistEstimation:
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

    YOLOv8_model_name = args['YOLOv8_model']
    UniDepth_model_name = args['UniDepth_model']

    UniDepth_version = UniDepth_model_name.split('_')[1]

    pretrained_models_dir = 'models/pretrained'

    YOLO_dir = 'YOLOv8'
    YOLOv8_model_name = '{}.pt'.format(YOLOv8_model_name)
    YOLOv8_path = os.path.join(pretrained_models_dir, YOLO_dir,
YOLOv8_model_name)

    YOLOv8_model = YOLO(YOLOv8_path).to(device)

    UniDepth_path = os.path.join(pretrained_models_dir,
UniDepth_model_name)
    if UniDepth_version == 'V1':
        UniDepth_model =
UniDepthV1.from_pretrained(UniDepth_path).to(device)
    else:
        UniDepth_model =
UniDepthV2.from_pretrained(UniDepth_path).to(device)

    distEstimation_model =
DistEstimation(YOLOv8_model=YOLOv8_model,
UniDepth_model=UniDepth_model)

    return distEstimation_model
```

В функции postprocess() реализована логика визуализации результата работы нейронных сетей, а именно отрисовка ограничивающих рамок найденного объекта,

класса объекта и спрогнозированного расстояния от этого объекта до камеры. Ниже приведен код для последующей обработки выходов моделей:

```
def postprocess(distEstimation_predict: dict, cv_frame:
cv.typing.MatLike) -> cv.typing.MatLike:
    class_name = distEstimation_predict['class_name']
    bbox = distEstimation_predict['bbox']
    distance = distEstimation_predict['distance']

    print('Объект: {}'.format(class_name))
    print('Местоположение: {}'.format(bbox))
    print('Расстояние: {:.2f}\n'.format(distance))

    X1 = bbox[0]
    Y1 = bbox[1]
    X2 = bbox[2]
    Y2 = bbox[3]

    in_channels = 3
    bbox_size = in_channels * (X2 - X1) * (Y2 - Y1)
    fontScale = bbox_size / cv_frame.size * np.square(np.pi)

    red_color = (60, 20, 220)
    green_color = (50, 205, 50)
    blue_color = (180, 130, 70)

    thickness = 2

    fontFace = cv.FONT_HERSHEY_COMPLEX
    lineType = cv.LINE_AA

    class_name_X_coord = X1
    class_name_Y_coord = Y1 - 16

    distance_X_coord = int(np.mean((X1, X2)))
    distance_Y_coord = Y1 - 16

    class_name_text = '{}'.format(class_name)
    distance_text = '{:.2f} м'.format(distance)

    cv.rectangle(img=cv_frame, pt1=(X1, Y1), pt2=(X2, Y2),
color=green_color, thickness=thickness)
    cv.putText(img=cv_frame, text=class_name_text,
org=(class_name_X_coord, class_name_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=red_color, thickness=thickness,
lineType=lineType)
    cv.putText(img=cv_frame, text=distance_text,
org=(distance_X_coord, distance_Y_coord),
fontFace=fontFace, fontScale=fontScale,
```

```

        color=blue_color, thickness=thickness,
lineType=lineType)

    return cv_frame

```

В методе process() реализована общая логика обработки, а именно получение входных данных, вызов метода обработки изображения, вызов метода последующей обработки изображения и т.д. Ниже приведен код функции, выполняющей обработку входных данных:

```

def process(distEstimation_model: DistEstimation, input_images:
list) -> None:
    result_dir = args['result_dir']
    show = args['show']

    print('Обработка изображений:')

    for input_image in tqdm(input_images):
        input_image_name = input_image.filename
        input_image_name = input_image_name.split('/')
        input_image_name = input_image_name.pop()

        print('\nИзображение: {}\n'.format(input_image_name))

        np_input_image = np.array(input_image)
        cv_image = cv.cvtColor(np_input_image,
cv.COLOR_RGB2BGR)

        distEstimation_predicts =
distEstimation_model.predict(input_frame=input_image)

        for distEstimation_predict in distEstimation_predicts:
            cv_image =
postprocess(distEstimation_predict=distEstimation_predict,
cv_frame=cv_image)

            if result_dir is not None:
                if not os.path.isdir(result_dir):
                    os.mkdir(result_dir)
                result_path = os.path.join(result_dir,
input_image_name)
                cv.imwrite(filename=result_path, img=cv_image)

            if show:
                cv.imshow(winname=input_image_name, mat=cv_image)

```

Модуль distEstimation.py описывается класс DistEstimation со всеми его атрибутами и методами. В конструктор класса необходимо будет передать

экземпляры моделей YOLOv8 и UniDepth, которые были получены из описанных выше модулей. Ключевым методом данного класса является метод predict(), который получив входное изображение, сначала производит предобработку этого изображения для того, чтобы отправить на вход каждой из моделей. После получения выходов моделей, вызывается функция последующей обработки для того, чтобы рассчитать расстояние, на основе карты глубины, спрогнозированной моделью UniDepth, от объекта, найденного моделью YOLOv8, до камеры. Ниже представлен код описания класса DistEstimation:

```
class DistEstimation:

    def __init__(self, YOLOv8_model, UniDepth_model):
        self.YOLOv8_model = YOLOv8_model
        self.UniDepth_model = UniDepth_model
        self.classes = {
            0: 'Человек',
            1: 'Велосипед',
            2: 'Автомобиль',
            3: 'Мотоцикл',
            4: 'Самолёт',
            5: 'Автобус',
            6: 'Поезд',
            7: 'Грузовик',
            8: 'Лодка',
            15: 'Кошка',
            16: 'Собака',
            39: 'Бутылка',
            41: 'Чашка',
            46: 'Банан',
            47: 'Яблоко',
            49: 'Апельсин',
            63: 'Ноутбук'
        }

    def predict(self, input_frame: Image) -> list:
        classes_indexes = list(self.classes.keys())

        preprocess = PreProcess(input_frame=input_frame)

        input_frame_YOLOv8, input_frame_UniDepth =
preprocess.preprocess()

        YOLOv8_predict =
self.YOLOv8_model.predict(input_frame_YOLOv8, verbose=False,
classes=classes_indexes)
```

```

        UniDepth_predict =
self.UniDepth_model.infer(input_frame_UniDepth)

        classes = self.classes
        postprocess =
PostProcess(YOLOv8_predict=YOLOv8_predict,
UniDepth_predict=UniDepth_predict, classes=classes)

        distEstimation_predicts = postprocess.postprocess()

        return distEstimation_predicts

```

Модули pre.py и post.py отвечают за обработку входных и выходных данных моделей YOLOv8 и UniDepth соответственно.

В модуле pre.py описан класс PreProcess, который, получая входное изображение в формате PIL.Image. Данный класс имеет один метод process(), который преобразует входное изображение в torch.tensor необходимой размерности для дальнейшей передачи на вход моделям глубокого обучения. Приведенный ниже код реализует преобразование входного изображения в формат torch.tensor, который будет отправлен на вход модели UniDepth:

```

def preprocess(self) -> (Image, torch.Tensor):
    input_frame_YOLOv8 = self.input_frame

    np_input_frame = np.array(self.input_frame)
    input_frame_UniDepth =
torch.from_numpy(np_input_frame).permute(2, 0, 1)

    return input_frame_YOLOv8, input_frame_UniDepth

```

Модуль post.py описывает класс PostProcessing, в конструктор которого поступают выходы моделей YOLOv8 и UniDepth. Данный класс, так же как и PreProcessing, реализован метод process(), задача которого обработать выход модели YOLOv8, получив координаты ограничивающей рамки объекта и наименование класса, к которому данный объект принадлежит. Далее необходимо преобразовать выходной torch.tensor модели UniDepth и на основе полученных координат ограничивающей рамки рассчитать среднее значение для карты глубины, находящейся внутри этой рамки. После расчёта расстояния все значимые

значения сохраняются в словарь для дальнейшего удобства при обращении к этим данным. Функция обработки выходов моделей глубокого обучения YOLOv8 и UniDepth, а также расчёта расстояния от объекта до камеры представлена ниже:

```
def postprocess(self) -> list:
    distEstimation_predicts = []

    depth_map = self.UniDepth_predict['depth']
    depth_map = depth_map.squeeze()

    YOLOv8_predicts = self.YOLOv8_predict

    classes = self.classes

    for YOLOv8_predict in YOLOv8_predicts:
        YOLOv8_bboxes = YOLOv8_predict.bboxes.xyxy

        for i in range(len(YOLOv8_bboxes)):
            class_index =
YOLOv8_predict.bboxes.cls[i].item()

            X1 = YOLOv8_bboxes[i][0].int().item()
            Y1 = YOLOv8_bboxes[i][1].int().item()
            X2 = YOLOv8_bboxes[i][2].int().item()
            Y2 = YOLOv8_bboxes[i][3].int().item()

            bbox = [X1, Y1, X2, Y2]

            bbox_depth_map = depth_map[Y1:Y2 + 1, X1:X2 +
1]

            distance = torch.mean(bbox_depth_map).item()

            predicted_class_name = classes[class_index]
            predicted_bbox = bbox
            predicted_distance = distance

            distEstimation_predict = {
                'class_name': predicted_class_name,
                'bbox': predicted_bbox,
                'distance': predicted_distance
            }

        distEstimation_predicts.append(distEstimation_predict)

    return distEstimation_predicts
```

Модуль utils.py содержи набор функций, которые необходимы для преобразования передаваемых аргументов модулям demo_image.py, demo_video.py

и demo_webcam.py. Ниже представлена функция для преобразования аргументов, используемых в модуле demo_image.py:

```
def read_image_args(args: list) -> dict:
    args_ = args[2:]

    input_dir = args_[0]

    if args_[1] == 'None':
        result_dir = None
    else:
        result_dir = args_[1]

    if args_[2] == 'True':
        show = True
    else:
        show = False

    YOLOv8_model = args_[3]
    UniDepth_model = args_[4]

    send_args = {
        'input_dir': input_dir,
        'result_dir': result_dir,
        'show': show,
        'YOLOv8_model': YOLOv8_model,
        'UniDepth_model': UniDepth_model
    }

    return send_args
```

5 Тестирование

5.1 Формирование тестового набора данных

Для проверки функциональной пригодности разработанного программного обеспечения требуется сформировать тестовый набор данных, который обеспечит оценку точности измерений и общую эффективность работы системы.

В качестве исходной основы выбран набор данных KITTI Object Detection Evaluation 2012, содержащий 7481 изображение в обучающей выборке и 7518 изображений в тестовой выборке. Общее количество размеченных объектов в наборе данных составляет 80 256 [18]. Формат всех изображений — цветной PNG. Разметка изображений представлена в следующем стандарте:

<Класс объекта> <Усечение> <Видимость> <Угол наблюдения>
<Ограничивающая рамка> <Габариты> <Местоположение> <Угол вращения>
<Степень уверенности>

В представленном формате разметки данных для объектов определены следующие характеристики:

- Класс объекта. Определяет категорию объекта, такую как «человек2», «автомобиль» и другие;
- Усечение. Число с плавающей точкой от 0.0 до 1.0, указывающее на степень обрезки объекта на изображении;
- Видимость. Оценка видимости объекта на изображении, выраженная целочисленными значениями: 0 (полностью виден), 1 (частично не виден), 2 (значительно не виден) и 3 (неизвестно);
- Угол наблюдения. Угол, под которым объект наблюдается на изображении, измеренный в радианах в диапазоне от $-\pi$ до π ;
- Ограничивающая рамка. Координаты прямоугольника, охватывающего объект на изображении, заданные как координаты верхнего левого и правого нижнего углов;

- Габариты. Размеры объекта в реальном мире, включая высоту, ширину и длину, измеренные в метрах;
- Местоположение. Расположение объекта в трехмерных координатах камеры (x, y, z), измеренное в метрах;
- Угол вращения. Угол наклона камеры вокруг оси Y, выраженный в радианах в диапазоне от $-\pi$ до π ;
- Степень уверенности. Число с плавающей точкой, отражающее уверенность в определении класса объекта.

В рамках исследования набора данных от KITTI принято решение разработать собственный тестовый набор данных, используя тренировочные изображения из указанного датасета. Тестовый набор данных будет содержать два основных класса объектов: «Человек» и «Автомобиль», поскольку они являются ключевыми для задачи измерения расстояния от объекта до камеры. Для каждого из этих классов будет отобрано 36 изображений, и для каждого изображения будет создана соответствующая разметка данных.

Структура тестового набора данных представлена на рисунке 5.1:



Рисунок 5.1 – Структура тестового набора данных

Пример изображения класса «Автомобиль» из тестового набора данных представлен на рисунке 5.2.



Рисунок 5.2 – Изображение класса «Автомобиль»

На рисунке 5.3 приведён пример файла разметки для изображения из тестового набора данных.

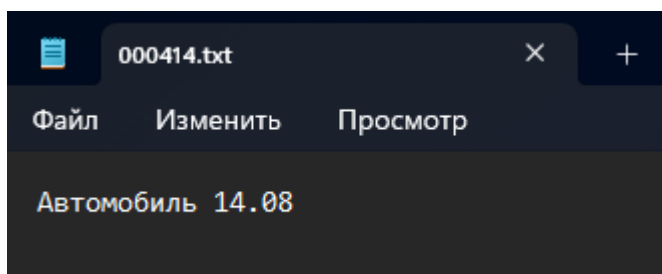


Рисунок 5.3 – Пример разметки для класса «Автомобиль»

В связи с акцентированием внимания на задаче измерения расстояния от объекта до камеры, принято решение упростить структуру разметки данных для собранного тестового набора. В соответствии с этим решением, в разметке сохранены только следующие данные:

- наименование класса объекта;
- z-координата объекта относительно камеры, выраженная в метрах.

5.2 Тестирование работы ПО

После успешного формирования тестового набора данных можно перейти к процессу проверки работоспособности разработанного программного обеспечения. В рамках этого процесса были созданы таблицы 5.1 и 5.2, содержащие основные результаты тестирования. Каждая из этих таблиц включает в себя сведения о разметке тестового набора данных, представленного наименованием

класса объекта (Класс_{GT}) и его расстоянием до камеры (D_{GT}). Кроме того, в таблицах представлены результаты работы программного обеспечения: спрогнозированный класс объекта на изображении ($\text{Класс}_{\text{PRED}}$) и рассчитанное с использованием карты глубины UniDepth расстояние до камеры (D_{PRED}).

Таблица 5.1 – Класс «Человек»

Изображение	Класс_{GT}	$\text{Класс}_{\text{PRED}}$	D_{GT} , м	D_{PRED} , м
000477.png	Человек	Человек	9.16	9.50
000294.png	Человек	Человек	7.40	9.58
000058.png	Человек	Человек	7.29	9.95
000407.png	Человек	Человек	8.31	9.63
000209.png	Человек	Человек	7.63	9.07
000199.png	Человек	Человек	7.49	9.24
000633.png	Человек	Человек	6.70	9.19
000103.png	Человек	Человек	9.15	9.78
000054.png	Человек	Человек	7.80	9.64
000298.png	Человек	Человек	9.03	9.24
000288.png	Человек	Человек	8.71	9.97
000646.png	Человек	Человек	8.81	9.83
000520.png	Человек	Человек	8.73	10.54
000598.png	Человек	Человек	9.37	9.73
000465.png	Человек	Человек	8.57	9.74
000263.png	Человек	Человек	10.71	11.51
000000.png	Человек	Человек	8.41	10.31
000368.png	Человек	Человек	7.87	8.62
000365.png	Человек	Человек	7.24	9.83
000398.png	Человек	Человек	7.41	9.62
000701.png	Человек	Человек	7.60	9.28
000109.png	Человек	Человек	8.71	9.73

Изображение	Класс _{GT}	Класс _{PRED}	D _{GT} , м	D _{PRED} , м
000130.png	Человек	Человек	8.43	9.78
000509.png	Человек	Человек	8.85	9.52
000070.png	Человек	Человек	6.89	8.88
000337.png	Человек	Человек	8.98	10.19
000028.png	Человек	Человек	8.51	9.22
000498.png	Человек	Человек	9.80	10.47
000671.png	Человек	Человек	8.47	9.44
000695.png	Человек	Человек	8.66	9.92
000327.png	Человек	Человек	8.67	10.42
000324.png	Человек	Человек	7.45	9.64
000532.png	Человек	Человек	9.00	10.39
000193.png	Человек	Человек	7.36	9.82
000290.png	Человек	Человек	6.44	7.80
000381.png	Человек	Человек	10.40	10.85

Из анализа данных в таблице 5.1 следует, что модель YOLOv8 продемонстрировала высокую точность в классификации объектов на изображениях, не допустив ни одной ошибки. Кроме того, можно рассчитать среднюю погрешность между прогнозируемым расстоянием и действительным. Для класса объектов «Человек» на тестовом наборе данных эта средняя погрешность составляет 17,5%.

В таблице 5.2 показан результат тестирования на для класса «Автомобиль».

Таблица 5.2 – Класс «Автомобиль»

Изображение	Класс _{GT}	Класс _{PRED}	D _{GT} , м	D _{PRED} , м
000546.png	Автомобиль	Автомобиль	10.43	12.95
000779.png	Автомобиль	Автомобиль	13.26	18.08
001197.png	Автомобиль	Автомобиль	40.04	47.29

Изображение	Класс _{GT}	Класс _{PRED}	D _{GT} , м	D _{PRED} , м
000158.png	Автомобиль	Автомобиль	30.37	33.12
001321.png	Автомобиль	Автомобиль	9.65	10.91
001385.png	Автомобиль	Автомобиль	46.82	50.68
000834.png	Автомобиль	Автомобиль	56.37	67.67
000414.png	Автомобиль	Автомобиль	14.08	19.01
000148.png	Автомобиль	Автомобиль	41.96	47.81
000090.png	Автомобиль	Автомобиль	18.74	18.18
000933.png	Автомобиль	Автомобиль	26.95	29.16
001018.png	Автомобиль	Автомобиль	11.20	11.55
001239.png	Автомобиль	Автомобиль	14.05	18.60
001341.png	Автомобиль	Автомобиль	23.73	25.76
000993.png	Автомобиль	Автомобиль	43.98	51.35
000891.png	Автомобиль	Автомобиль	31.15	31.53
000905.png	Автомобиль	Автомобиль	32.82	35.22
000687.png	Автомобиль	Автомобиль	51.40	59.91
000094.png	Автомобиль	Автомобиль	19.16	20.54
001619.png	Автомобиль	Автомобиль	10.54	11.16
000810.png	Автомобиль	Автомобиль	17.76	19.07
000839.png	Автомобиль	Автомобиль	24.69	30.05
001399.png	Автомобиль	Автомобиль	50.71	57.85
000434.png	Автомобиль	Автомобиль	39.17	41.91
001155.png	Автомобиль	Автомобиль	27.23	29.73
000247.png	Автомобиль	Автомобиль	40.02	41.15
000815.png	Автомобиль	Автомобиль	22.78	24.51
001408.png	Автомобиль	Автомобиль	41.89	42.59
000693.png	Автомобиль	Автомобиль	27.86	29.89
000020.png	Автомобиль	Автомобиль	15.58	17.05
001066.png	Автомобиль	Автомобиль	12.85	16.97
000828.png	Автомобиль	Автомобиль	28.19	30.51

Изображение	Класс _{GT}	Класс _{PRED}	D _{GT} , м	D _{PRED} , м
001202.png	Автомобиль	Автомобиль	56.62	57.37
000960.png	Автомобиль	Автомобиль	58.91	65.84
000202.png	Автомобиль	Автомобиль	45.59	46.91
001135.png	Автомобиль	Автомобиль	14.81	16.14

Анализ результатов тестирования для класса «Автомобиль» позволяет сделать вывод о высокой точности модели YOLOv8 в задаче классификации объектов на входных изображениях. Средняя погрешность для данного класса составила 12,2%, что объяснимо, учитывая обучение модели UniDepth на наборе данных от KITTI, где значительная часть изображений содержит проезжие части дорог и автострад.

Исходя из этих результатов, можно утверждать, что разработанное программное обеспечение успешно справляется с задачами классификации объектов и оценки расстояния до них на входных изображениях.

6 Безопасность жизнедеятельности

Пренебрежение правилами безопасности при работе с компьютером может привести к негативным последствиям для пользователя в виде ущерба для здоровья и имущества. Даже соблюдая технику безопасности как с работой со сложным бытовым устройством, вы можете получить не прямой вред своему здоровью, если не будете соблюдать рекомендации врачей, специально разработанных для пользователей ПК.

6.1 Вред здоровью при работе с компьютером

Продолжительная эксплуатация персонального компьютера может привести к накопительному отрицательному воздействию на здоровье человека, т.е. ущерб долгое время может не проявляться вовсе, а незаметно и постепенно накапливаться. Одними из самых опасных воздействий на здоровье пользователя ПК являются:

- высокая нагрузка на органы зрения, которая вызывает такие синдромы как «красный глаз» или «сухой глаз»;
- болезнь суставов, вызванная неправильной позой при продолжительной работе;
- нагрузка на психику и нервную систему, возникающая при долгой концентрации внимания на большом потоке информации длительное время.

6.2 Снижение вредного воздействия на здоровье при работе с ПК

Для того, чтобы максимально эффективно убрать отрицательное влияние на здоровье пользователя, необходимо соблюдать перечень простых правил, перечисленных ниже:

- соблюдать безопасное расстояние от пользователя до монитора. Безопасным расстоянием считается 40 – 75 сантиметров;
- соблюдать правильный угол наклона экрана монитора в 25 – 30 градусов;
- держать прямую осанку при работе с компьютером. Для этого необходимо правильно отрегулировать рабочее кресло или стул;

– при работе с периферией ввода/вывода, а именно клавиатурой и мышью, стараться держать руки немного согнутыми, а локти должны располагаться на столе или подлокотниках. Это предотвратит напряжение в кистях;

– при работе с компьютером, ноги не должны быть согнуты под стул или кресло, а должны быть выпрямлены вперед с упором в твердую поверхность;

– каждый час необходимо делать перерыв от работы с компьютером и выполнять небольшую разминку для снятия напряжения в суставах и мышцах.

6.3 Техника безопасности при работе с компьютером

Как и для электрического устройства, персональный компьютер также имеет технику безопасности при эксплуатации. Ниже приведен список основных правил при работе с ПК:

– не рекомендуется включать компьютер в розетки без заземления. Розетки и вилки должны быть цельными и без повреждений;

– нельзя работать с компьютером при открытом корпусе системного блока;

– не включать компьютер в помещении с высокой влажностью;

– нельзя оставлять без присмотра работающий ПК на длительное время.

– запрещается работать с компьютером, который имеет внешние повреждения корпуса или изоляцию силовых кабелей. Необходимо провести замену соответствующих кабелей или обратиться в сервисный центр.

Соблюдая технику безопасности при работе с компьютером и вышеперечисленные рекомендации, можно свести риск получить травму и нанести ущерб здоровью пользователя к минимуму.

Заключение

В ходе выполнения данной квалификационной работы были получены навыки использования моделей глубокого обучения для решения задачи измерения расстояния от объекта на изображении до модуля камеры. Также был выбран стек технологий для написания собственного программного обеспечения, которое позволяет решить поставленную задачу. Результатом работы является пакет модулей, который написан на языке программирования Python, а также отчёт, котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор всех его функциональных особенностей.

Разработанное программное обеспечение позволяет пользователю выбрать входные данные (изображение, видео, видеопоток с камеры) для дальнейшей обработки через модели глубокого обучения. После проверки работоспособности разработанного продукта было выяснено, что созданное программное обеспечение соответствует всем требованиям, которые были поставлены в начале работы.

На основе полученного опыта при работе с нейронными сетями, разработанный пакет можно улучшить, спроектировав собственную архитектуру модели глубокого обучения.

					ПГУ 1.090301.40.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, Feng Qian. «Monocular Depth Estimation Based On Deep Learning: An Overview», 2020.
2. T. Zhou, J. Kretschmar, M. Khansari, J. Lehrmann, D. D. Lee. «End-to-End Deep Learning for Autonomous Driving», 2016.
3. John J. Han, Ayberk Acar, Callahan Henry, Jie Ying Wu. «Depth Anything in Medical Images: A Comparative Study», 2024.
4. Xingshuai Dong, Matthew A. Garratt, Sreenatha G. Anavatti, Hussein A. Abbass. «Towards Real-Time Monocular Depth Estimation for Robotics: A Survey», 2021.
5. Abdelrahman Taha, Qi Qu, Sam Alex, Ping Wang, William L. Abbott, Ahmed Alkhateeb. «Millimeter Wave MIMO based Depth Maps for Wireless Virtual and Augmented Reality», 2021.
6. Alessio Mingozzi, Andrea Conti, Filippo Aleotti, Matteo Poggi, Stefano Mattoccia. «Monitoring social distancing with single image depth estimation», 2021.
7. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», 2015.
8. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection», 2015.
9. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. «SSD: Single Shot MultiBox Detector», 2015.
10. Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. «Mask R-CNN», 2017.
11. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár. «Focal Loss for Dense Object Detection», 2017.
12. Adrian Rosebrock. «Find distance from camera to object/marker using Python and OpenCV». [Электронный ресурс]. URL: <https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>. (дата обращения 12.04.2024)

13. Статья «Основы стереозрения», 2011. [Электронный ресурс]. URL: <https://habr.com/ru/articles/130300/>. (дата обращения 14.04.2024)

14. Статья «Что такое лидар». [Электронный ресурс]. URL: <https://amperka.ru/page/lidar-sensors-guide>

15. René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, Vladlen Koltun. «Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer», 2019.

16. GitHub-репозиторий «Ultralytics». [Электронный ресурс]. URL: <https://github.com/ultralytics/ultralytics>. (дата обращения 21.04.2024)

17. Luigi Piccinelli, Yung-Hsu Yang, Christos Sakaridis, Mattia Segu, Siyuan Li, Luc Van Gool, Fisher Yu. «UniDepth: Universal Monocular Metric Depth Estimation», 2024.

18. A. Geiger, P. Lenz, C. Stiller, R. Urtasun. Описание набора данных KITTI «Object Detection Evaluation 2012». [Электронный ресурс]. URL: https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d. (дата обращения 04.05.2024)

					ПГУ 1.090301.40.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

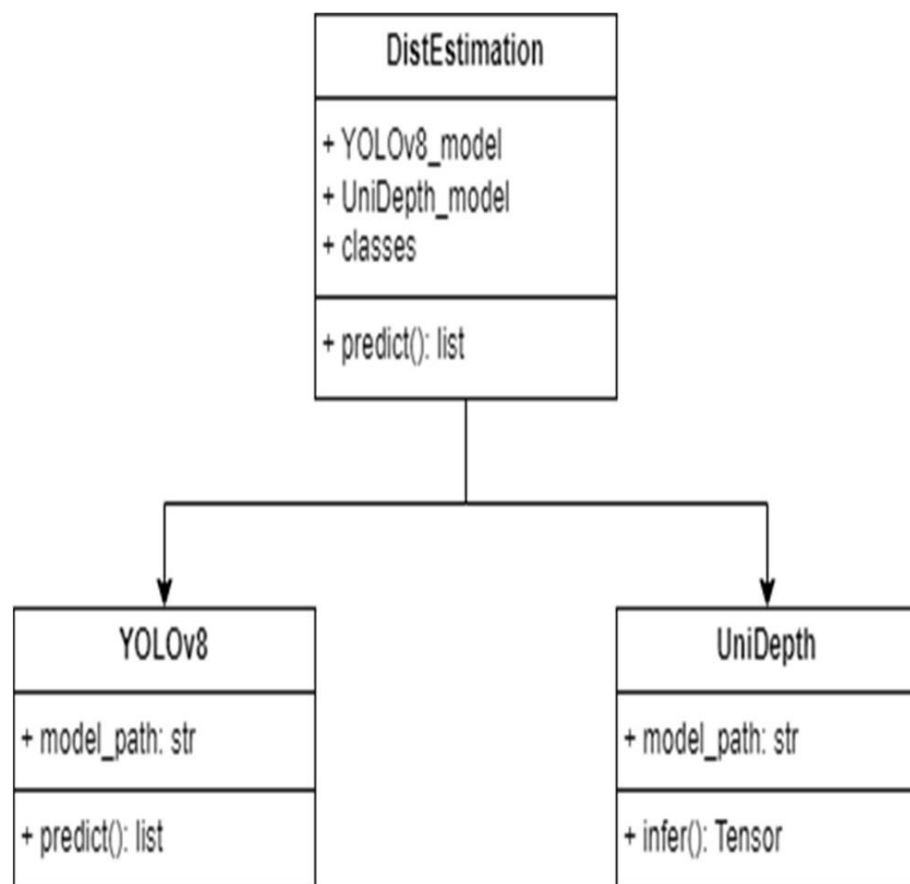
(Обязательное)

Программа оценки расстояния до объекта на изображении с помощью нейронных
сетей

UML-диаграммы

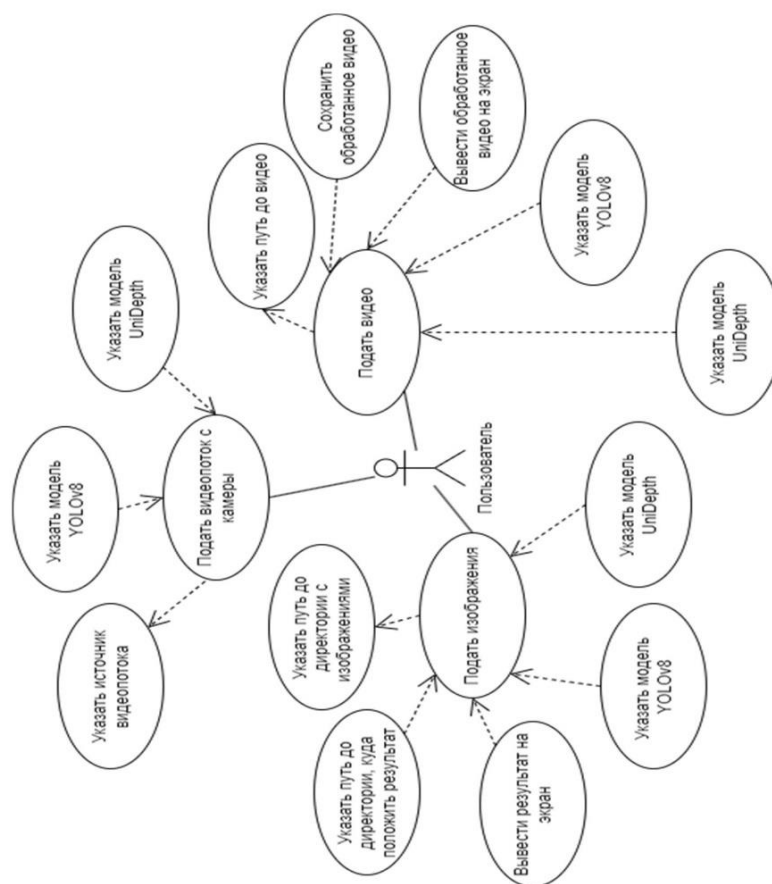
					ПГУ 1.090301.40.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма классов



				ПГУ 1.090301.40. 002 Д11		
Имя	Инициал	И.О.Фамилия	Подпись	Дата	Лист	Листов
Разработчик	Шенгеладзе Д.В.					
Проверен	Митропольский М.А.					
Содержание работ по объектам на разработанных проектах инженерных сетей Дизайн-проект					ФБТ, каф. ВТ	
					ар. 208ВВ/П2	
М. Утвердил	Выполнено А.С.					
Утвержден	Митропольский М.А.					

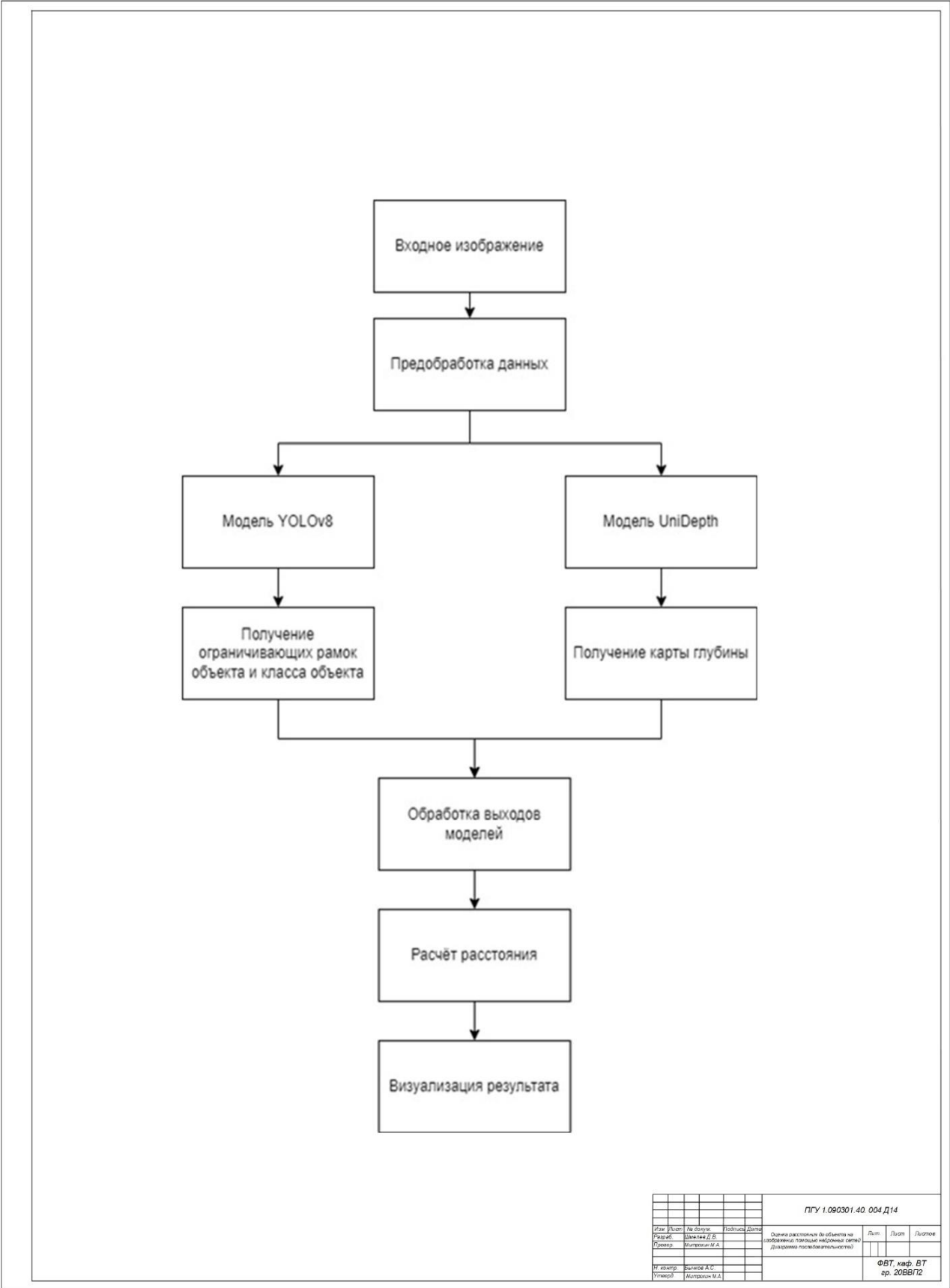
Приложение А.2 – Диаграмма прецедентов

[illegible]

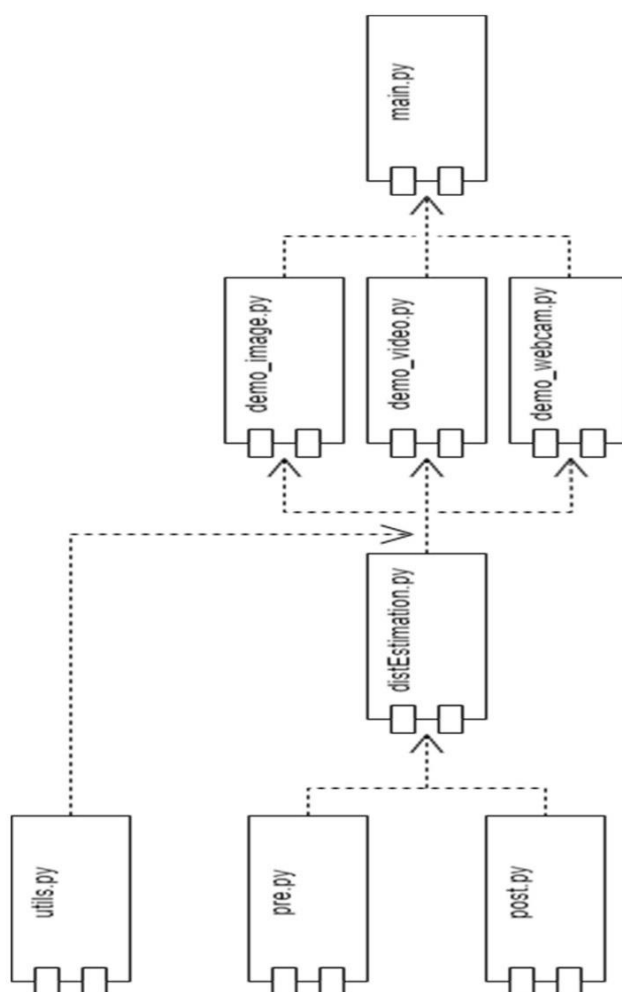
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.40.001 ПЗ

Приложение А.3 – Диаграмма последовательностей



Приложение А.4 – Диаграмма компонентов



ПУУ 19030301.40.005.018									
Содержание расчёта по объёмам и количеству работ				Длина		Листы		Листов	
Длина				Ширина		Площадь		Объём	
№	Наименование	Единица измерения	Количество	Длина	Ширина	Площадь	Объём	Длина	Листов
1	Работы по устройству	м	100	10	10	100	10	10	100
2	Работы по устройству	м	100	10	10	100	10	10	100
3	Работы по устройству	м	100	10	10	100	10	10	100
4	Работы по устройству	м	100	10	10	100	10	10	100
5	Работы по устройству	м	100	10	10	100	10	10	100
6	Работы по устройству	м	100	10	10	100	10	10	100
7	Работы по устройству	м	100	10	10	100	10	10	100
8	Работы по устройству	м	100	10	10	100	10	10	100
9	Работы по устройству	м	100	10	10	100	10	10	100
10	Работы по устройству	м	100	10	10	100	10	10	100
11	Работы по устройству	м	100	10	10	100	10	10	100
12	Работы по устройству	м	100	10	10	100	10	10	100
13	Работы по устройству	м	100	10	10	100	10	10	100
14	Работы по устройству	м	100	10	10	100	10	10	100
15	Работы по устройству	м	100	10	10	100	10	10	100
16	Работы по устройству	м	100	10	10	100	10	10	100
17	Работы по устройству	м	100	10	10	100	10	10	100
18	Работы по устройству	м	100	10	10	100	10	10	100
19	Работы по устройству	м	100	10	10	100	10	10	100
20	Работы по устройству	м	100	10	10	100	10	10	100
21	Работы по устройству	м	100	10	10	100	10	10	100
22	Работы по устройству	м	100	10	10	100	10	10	100
23	Работы по устройству	м	100	10	10	100	10	10	100
24	Работы по устройству	м	100	10	10	100	10	10	100
25	Работы по устройству	м	100	10	10	100	10	10	100
26	Работы по устройству	м	100	10	10	100	10	10	100
27	Работы по устройству	м	100	10	10	100	10	10	100
28	Работы по устройству	м	100	10	10	100	10	10	100
29	Работы по устройству	м	100	10	10	100	10	10	100
30	Работы по устройству	м	100	10	10	100	10	10	100
31	Работы по устройству	м	100	10	10	100	10	10	100
32	Работы по устройству	м	100	10	10	100	10	10	100
33	Работы по устройству	м	100	10	10	100	10	10	100
34	Работы по устройству	м	100	10	10	100	10	10	100
35	Работы по устройству	м	100	10	10	100	10	10	100
36	Работы по устройству	м	100	10	10	100	10	10	100
37	Работы по устройству	м	100	10	10	100	10	10	100
38	Работы по устройству	м	100	10	10	100	10	10	100
39	Работы по устройству	м	100	10	10	100	10	10	100
40	Работы по устройству	м	100	10	10	100	10	10	100
41	Работы по устройству	м	100	10	10	100	10	10	100
42	Работы по устройству	м	100	10	10	100	10	10	100

Приложение Б

(Обязательное)

Программа оценки расстояния до объекта на изображении с помощью нейронных
сетей

Листинг программы

					ПГУ 1.090301.40.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

Файл main.py

```
import argparse
import subprocess

def args_to_list(args):
    args_ = []
    for key in args.keys():
        arg = str(args[key])
        args_.append(arg)

    return args_

def run_process(mode: str, args):
    if mode == 'image':
        command = ['python', './demo/demo_image.py'] + args
        subprocess.run(command)
    elif mode == 'video':
        command = ['python', './demo/demo_video.py'] + args
        subprocess.run(command)
    elif mode == 'webcam':
        command = ['python', './demo/demo_webcam.py'] + args
        subprocess.run(command)
    else:
        print('Выбран некорректный режим работы!')

if __name__ == '__main__':
    parser =
    argparse.ArgumentParser(description='dist_estimation')
    subparsers = parser.add_subparsers(dest='mode')

    image_parser = subparsers.add_parser(name='image')
    image_parser.add_argument('--input_dir', required=True,
    help='Путь до директории, с входными изображениями')
    image_parser.add_argument('--result_dir', required=False,
    help='Путь до директории, в
    которую нужно поместить результат')
    image_parser.add_argument('--show', required=False,
    action='store_true', help='Вывести результат на экран')
    image_parser.add_argument('--YOLOv8', required=False,
    default='yolov8n',
    help='Модель YOLOv8, для
    обнаружения объектов')
    image_parser.add_argument('--UniDepth', required=False,
    default='UniDepth_V1_ViT114',
    help='Модель UniDepth, для
    прогнозирования карты глубины входного изображения')

    video_parser = subparsers.add_parser(name='video')
```

```

        video_parser.add_argument('--video_path', required=True,
                                   help='Путь до входного
видеопотока')
        video_parser.add_argument('--result_dir', required=False,
                                   help='Путь до директории, в
которую нужно поместить результат')
        video_parser.add_argument('--show', required=False,
action='store_true', help='Вывести результат на экран')
        video_parser.add_argument('--YOLOv8', required=False,
default='yolov8n',
                                   help='Модель YOLOv8, для
обнаружения объектов')
        video_parser.add_argument('--UniDepth', required=False,
default='UniDepth_V2_ViTl14',
                                   help='Модель UniDepth, для
прогнозирования карты глубины входного изображения')

        webcam_parser = subparsers.add_parser(name='webcam')
        webcam_parser.add_argument('--source', required=False,
default=0, help='Источник видеопотока')
        webcam_parser.add_argument('--YOLOv8', required=False,
default='yolov8n',
                                   help='Модель YOLOv8, для
обнаружения объектов')
        webcam_parser.add_argument('--UniDepth', required=False,
default='UniDepth_V1_ViTl14',
                                   help='Модель UniDepth, для
прогнозирования карты глубины входного изображения')

        args = parser.parse_args()
        mode = args.mode

        args_ = vars(args)
        send_args = args_to_list(args=args_)

        run_process(mode=mode, args=send_args)

```

Файл demo_image.py

```

import os
import sys
import torch

import cv2 as cv
import numpy as np

from PIL import Image
from tqdm import tqdm
from ultralytics import YOLO
from unidepth.models import UniDepthV1
from unidepth.models import UniDepthV2
from models.distEstimation import DistEstimation

```

```

from utils.utils import read_image_args

def collect_input_images(input_dir: str) -> list:
    input_images = os.listdir(input_dir)

    PIL_images = []

    for input_image in input_images:
        PIL_image_path = os.path.join(input_dir, input_image)
        PIL_image = Image.open(PIL_image_path)
        PIL_images.append(PIL_image)

    return PIL_images

def preprocess() -> DistEstimation:
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

    YOLOv8_model_name = args['YOLOv8_model']
    UniDepth_model_name = args['UniDepth_model']

    UniDepth_version = UniDepth_model_name.split('_')[1]

    pretrained_models_dir = 'models/pretrained'

    YOLO_dir = 'YOLOv8'
    YOLOv8_model_name = '{}.pt'.format(YOLOv8_model_name)
    YOLOv8_path = os.path.join(pretrained_models_dir, YOLO_dir,
YOLOv8_model_name)

    YOLOv8_model = YOLO(YOLOv8_path).to(device)

    UniDepth_path = os.path.join(pretrained_models_dir,
UniDepth_model_name)
    if UniDepth_version == 'V1':
        UniDepth_model =
UniDepthV1.from_pretrained(UniDepth_path).to(device)
    else:
        UniDepth_model =
UniDepthV2.from_pretrained(UniDepth_path).to(device)

    distEstimation_model =
DistEstimation(YOLOv8_model=YOLOv8_model,
UniDepth_model=UniDepth_model)

    return distEstimation_model

def postprocess(distEstimation_predict: dict, cv_frame:
cv.typing.MatLike) -> cv.typing.MatLike:
    class_name = distEstimation_predict['class_name']

```

```

bbox = distEstimation_predict['bbox']
distance = distEstimation_predict['distance']

print('Объект: {}'.format(class_name))
print('Местоположение: {}'.format(bbox))
print('Расстояние: {:.2f}\n'.format(distance))

X1 = bbox[0]
Y1 = bbox[1]
X2 = bbox[2]
Y2 = bbox[3]

in_channels = 3
bbox_size = in_channels * (X2 - X1) * (Y2 - Y1)
fontScale = bbox_size / cv_frame.size * np.square(np.pi)

red_color = (60, 20, 220)
green_color = (50, 205, 50)
blue_color = (180, 130, 70)

thickness = 2

fontFace = cv.FONT_HERSHEY_COMPLEX
lineType = cv.LINE_AA

class_name_X_coord = X1
class_name_Y_coord = Y1 - 16

distance_X_coord = int(np.mean((X1, X2)))
distance_Y_coord = Y1 - 16

class_name_text = '{}'.format(class_name)
distance_text = '{:.2f} м'.format(distance)

cv.rectangle(img=cv_frame, pt1=(X1, Y1), pt2=(X2, Y2),
color=green_color, thickness=thickness)
cv.putText(img=cv_frame, text=class_name_text,
org=(class_name_X_coord, class_name_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=red_color, thickness=thickness,
lineType=lineType)
cv.putText(img=cv_frame, text=distance_text,
org=(distance_X_coord, distance_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=blue_color, thickness=thickness,
lineType=lineType)

return cv_frame

def process(distEstimation_model: DistEstimation, input_images:
list) -> None:
    result_dir = args['result_dir']

```

```

show = args['show']

print('Обработка изображений:')

for input_image in tqdm(input_images):
    input_image_name = input_image.filename
    input_image_name = input_image_name.split('/')
    input_image_name = input_image_name.pop()

    print('\nИзображение: {}'.format(input_image_name))

    np_input_image = np.array(input_image)
    cv_image = cv.cvtColor(np_input_image,
cv.COLOR_RGB2BGR)

    distEstimation_predicts =
distEstimation_model.predict(input_frame=input_image)

    for distEstimation_predict in distEstimation_predicts:
        cv_image =
postprocess(distEstimation_predict=distEstimation_predict,
cv_frame=cv_image)

        if result_dir is not None:
            if not os.path.isdir(result_dir):
                os.mkdir(result_dir)
            result_path = os.path.join(result_dir,
input_image_name)
            cv.imwrite(filename=result_path, img=cv_image)

        if show:
            cv.imshow(winname=input_image_name, mat=cv_image)

if __name__ == '__main__':
    args = sys.argv
    args = read_image_args(args=args)

    distEstimation_model = preprocess()

    input_dir = args['input_dir']
    input_images = collect_input_images(input_dir=input_dir)

    process(distEstimation_model=distEstimation_model,
input_images=input_images)

```

Файл demo_video.py

```

import os
import sys
import torch

```

```

import cv2 as cv
import numpy as np

from PIL import Image
from tqdm import tqdm
from ultralytics import YOLO
from unidepth.models import UniDepthV1
from unidepth.models import UniDepthV2
from models.distEstimation import DistEstimation
from utils.utils import read_video_args

def preprocess() -> DistEstimation:
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

    YOLOv8_model_name = args['YOLOv8_model']
    UniDepth_model_name = args['UniDepth_model']

    UniDepth_version = UniDepth_model_name.split('_')[1]

    pretrained_models_dir = 'models/pretrained'

    YOLO_dir = 'YOLOv8'
    YOLOv8_model_name = '{}.pt'.format(YOLOv8_model_name)
    YOLOv8_path = os.path.join(pretrained_models_dir, YOLO_dir,
YOLOv8_model_name)

    YOLOv8_model = YOLO(YOLOv8_path).to(device)

    UniDepth_path = os.path.join(pretrained_models_dir,
UniDepth_model_name)
    if UniDepth_version == 'V1':
        UniDepth_model =
UniDepthV1.from_pretrained(UniDepth_path).to(device)
    else:
        UniDepth_model =
UniDepthV2.from_pretrained(UniDepth_path).to(device)

    distEstimation_model =
DistEstimation(YOLOv8_model=YOLOv8_model,
UniDepth_model=UniDepth_model)

    return distEstimation_model

def postprocess(distEstimation_predict: dict, cv_frame:
cv.typing.MatLike) -> cv.typing.MatLike:
    if not distEstimation_predict:
        return cv_frame

    distEstimation_predict = distEstimation_predict[0]

```

```

class_name = distEstimation_predict['class_name']
bbox = distEstimation_predict['bbox']
distance = distEstimation_predict['distance']

print('Объект: {}'.format(class_name))
print('Местоположение: {}'.format(bbox))
print('Расстояние: {:.2f} м\n'.format(distance))

X1 = bbox[0]
Y1 = bbox[1]
X2 = bbox[2]
Y2 = bbox[3]

fontScale = 1.0

red_color = (60, 20, 220)
green_color = (50, 205, 50)
blue_color = (180, 130, 70)

thickness = 2

fontFace = cv.FONT_HERSHEY_COMPLEX
lineType = cv.LINE_AA

class_name_X_coord = 16
class_name_Y_coord = 32

distance_X_coord = 16
distance_Y_coord = 64

class_name_text = '{}'.format(class_name)
distance_text = '{:.2f} м'.format(distance)

cv.rectangle(img=cv_frame, pt1=(X1, Y1), pt2=(X2, Y2),
color=green_color, thickness=thickness)
cv.putText(img=cv_frame, text=class_name_text,
org=(class_name_X_coord, class_name_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=red_color, thickness=thickness,
lineType=lineType)
cv.putText(img=cv_frame, text=distance_text,
org=(distance_X_coord, distance_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=blue_color, thickness=thickness,
lineType=lineType)

return cv_frame

def process(distEstimation_model: DistEstimation) -> None:
    video_path = args['video_path']
    result_dir = args['result_dir']
    show = args['show']

```

```

video_name = video_path.split('/')
video_name = video_name.pop()

print('Видео: {}'.format(video_name))

cap = cv.VideoCapture(video_path)

if result_dir is not None:
    if not os.path.isdir(result_dir):
        os.mkdir(result_dir)
    frame_width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))

    fourcc = cv.VideoWriter_fourcc(*'XVID')

    fps = 30.0

    result_video_path = os.path.join(result_dir,
video_name)
    result_video = cv.VideoWriter(result_video_path,
fourcc, fps, (frame_width, frame_height))

    total_frames = int(cap.get(cv.CAP_PROP_FRAME_COUNT))

    progress_bar = tqdm(total=total_frames)

    while cap.isOpened():
        ret, cv_frame = cap.read()

        if not ret:
            break

        PIL_frame = cv.cvtColor(cv_frame, cv.COLOR_BGR2RGB)

        distEstimation_predict =
distEstimation_model.predict(input_frame=PIL_frame)

        res_frame =
postprocess(distEstimation_predict=distEstimation_predict,
cv_frame=cv_frame)

        progress_bar.update()

        if result_dir is not None:
            result_video.write(res_frame)

        if show:
            winname = '{}'.format(video_name)
            cv.imshow(winname=winname, mat=res_frame)

        if cv.waitKey(1) & 0xFF == ord('q'):
            break

```

```

if __name__ == '__main__':
    args = sys.argv
    args = read_video_args(args=args)

    distEstimation_model = preprocess()
    process(distEstimation_model=distEstimation_model)

```

Файл demo_webcam.py

```

import os
import sys
import torch

import cv2 as cv
import numpy as np

from PIL import Image
from tqdm import tqdm
from ultralytics import YOLO
from unidepth.models import UniDepthV1
from unidepth.models import UniDepthV2
from models.distEstimation import DistEstimation
from utils.utils import read_webcam_args

def preprocess() -> DistEstimation:
    device = device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")

    YOLOv8_model_name = args['YOLOv8_model']
    UniDepth_model_name = args['UniDepth_model']

    UniDepth_version = UniDepth_model_name.split('_')[1]

    pretrained_models_dir = 'models/pretrained'

    YOLO_dir = 'YOLOv8'
    YOLOv8_model_name = '{}.pt'.format(YOLOv8_model_name)
    YOLOv8_path = os.path.join(pretrained_models_dir, YOLO_dir,
YOLOv8_model_name)

    YOLOv8_model = YOLO(YOLOv8_path).to(device)

    UniDepth_path = os.path.join(pretrained_models_dir,
UniDepth_model_name)
    if UniDepth_version == 'V1':
        UniDepth_model =
UniDepthV1.from_pretrained(UniDepth_path).to(device)
    else:

```

					ПГУ 1.090301.40.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

        UniDepth_model =
UniDepthV2.from_pretrained(UniDepth_path).to(device)

        distEstimation_model =
DistEstimation(YOLOv8_model=YOLOv8_model,
UniDepth_model=UniDepth_model)

        return distEstimation_model

def postprocess(distEstimation_predict: dict, cv_frame:
cv.typing.MatLike) -> cv.typing.MatLike:
    if not distEstimation_predict:
        return cv_frame

    distEstimation_predict = distEstimation_predict[0]

    class_name = distEstimation_predict['class_name']
    bbox = distEstimation_predict['bbox']
    distance = distEstimation_predict['distance']

    print('Объект: {}'.format(class_name))
    print('Местоположение: {}'.format(bbox))
    print('Расстояние: {:.2f} м\n'.format(distance))

    X1 = bbox[0]
    Y1 = bbox[1]
    X2 = bbox[2]
    Y2 = bbox[3]

    fontScale = 1.0

    red_color = (60, 20, 220)
    green_color = (50, 205, 50)
    blue_color = (180, 130, 70)

    thickness = 2

    fontFace = cv.FONT_HERSHEY_COMPLEX
    lineType = cv.LINE_AA

    class_name_X_coord = 16
    class_name_Y_coord = 32

    distance_X_coord = 16
    distance_Y_coord = 64

    class_name_text = '{}'.format(class_name)
    distance_text = '{:.2f} м'.format(distance)

    cv.rectangle(img=cv_frame, pt1=(X1, Y1), pt2=(X2, Y2),
color=green_color, thickness=thickness)

```

```

        cv.putText(img=cv_frame, text=class_name_text,
org=(class_name_X_coord, class_name_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=red_color, thickness=thickness,
lineType=lineType)
        cv.putText(img=cv_frame, text=distance_text,
org=(distance_X_coord, distance_Y_coord),
fontFace=fontFace, fontScale=fontScale,
color=blue_color, thickness=thickness,
lineType=lineType)

    return cv_frame

def process(distEstimation_model: DistEstimation) -> None:
    source = args['source']

    print('Камера: {}'.format(source))

    cap = cv.VideoCapture(source)

    while cap.isOpened():
        ret, cv_frame = cap.read()

        if not ret:
            break

        PIL_frame = cv.cvtColor(cv_frame, cv.COLOR_BGR2RGB)

        distEstimation_predict =
distEstimation_model.predict(input_frame=PIL_frame)

        res_frame =
postprocess(distEstimation_predict=distEstimation_predict,
cv_frame=cv_frame)

        winname = 'Camera: {}'.format(source)
        cv.imshow(winname=winname, mat=res_frame)

        if cv.waitKey(1) & 0xFF == ord('q'):
            break

if __name__ == '__main__':
    args = sys.argv
    args = read_webcam_args(args=args)

    distEstimation_model = preprocess()
    process(distEstimation_model=distEstimation_model)

```

Файл pre.py

					ПГУ 1.090301.40.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```

import torch

import numpy as np

from PIL import Image

class PreProcess:
    def __init__(self, input_frame: Image):
        self.input_frame = input_frame

    def preprocess(self) -> (Image, torch.Tensor):
        input_frame_YOLOv8 = self.input_frame

        np_input_frame = np.array(self.input_frame)
        input_frame_UniDepth =
torch.from_numpy(np_input_frame).permute(2, 0, 1)

        return input_frame_YOLOv8, input_frame_UniDepth

```

Файл post.py

```

import torch

import numpy as np

class PostProcess:
    def __init__(self, YOLOv8_predict, UniDepth_predict,
classes):
        self.YOLOv8_predict = YOLOv8_predict
        self.UniDepth_predict = UniDepth_predict
        self.classes = classes

    def postprocess(self) -> list:
        distEstimation_predicts = []

        depth_map = self.UniDepth_predict['depth']
        depth_map = depth_map.squeeze()

        YOLOv8_predicts = self.YOLOv8_predict

        classes = self.classes

        for YOLOv8_predict in YOLOv8_predicts:
            YOLOv8_bboxes = YOLOv8_predict.bboxes.xyxy

            for i in range(len(YOLOv8_bboxes)):
                class_index =
YOLOv8_predict.bboxes.cls[i].item()

                X1 = YOLOv8_bboxes[i][0].int().item()

```

```

        Y1 = YOLOv8_bboxes[i][1].int().item()
        X2 = YOLOv8_bboxes[i][2].int().item()
        Y2 = YOLOv8_bboxes[i][3].int().item()

        bbox = [X1, Y1, X2, Y2]

        bbox_depth_map = depth_map[Y1:Y2 + 1, X1:X2 +
1]

        distance = torch.mean(bbox_depth_map).item()

        predicted_class_name = classes[class_index]
        predicted_bbox = bbox
        predicted_distance = distance

        distEstimation_predict = {
            'class_name': predicted_class_name,
            'bbox': predicted_bbox,
            'distance': predicted_distance
        }

distEstimation_predicts.append(distEstimation_predict)

    return distEstimation_predicts

```

Файл distEstimation.py

```

import torch

from PIL import Image

from ultralytics import YOLO
from unidepth.models import UniDepthV1
from unidepth.models import UniDepthV2
from processing.pre import PreProcess
from processing.post import PostProcess

class DistEstimation:

    def __init__(self, YOLOv8_model, UniDepth_model):
        self.YOLOv8_model = YOLOv8_model
        self.UniDepth_model = UniDepth_model
        self.classes = {
            0: 'Человек',
            1: 'Велосипед',
            2: 'Автомобиль',
            3: 'Мотоцикл',
            4: 'Самолёт',
            5: 'Автобус',
            6: 'Поезд',

```

```

        7: 'Грузовик',
        8: 'Лодка',
        15: 'Кошка',
        16: 'Собака',
        39: 'Бутылка',
        41: 'Чашка',
        46: 'Банан',
        47: 'Яблоко',
        49: 'Апельсин',
        63: 'Ноутбук'
    }

    def predict(self, input_frame: Image) -> list:
        classes_indexes = list(self.classes.keys())

        preprocess = PreProcess(input_frame=input_frame)

        input_frame_YOLOv8, input_frame_UniDepth =
preprocess.preprocess()

        YOLOv8_predict =
self.YOLOv8_model.predict(input_frame_YOLOv8, verbose=False,
classes=classes_indexes)
        UniDepth_predict =
self.UniDepth_model.infer(input_frame_UniDepth)

        classes = self.classes
        postprocess =
PostProcess(YOLOv8_predict=YOLOv8_predict,
UniDepth_predict=UniDepth_predict, classes=classes)

        distEstimation_predicts = postprocess.postprocess()

        return distEstimation_predicts

```

Файл utils.py

```

def read_image_args(args: list) -> dict:
    args_ = args[2:]

    input_dir = args_[0]

    if args_[1] == 'None':
        result_dir = None
    else:
        result_dir = args_[1]

    if args_[2] == 'True':
        show = True
    else:
        show = False

```

```

YOLOv8_model = args_[3]
UniDepth_model = args_[4]

send_args = {
    'input_dir': input_dir,
    'result_dir': result_dir,
    'show': show,
    'YOLOv8_model': YOLOv8_model,
    'UniDepth_model': UniDepth_model
}

return send_args

def read_video_args(args: list) -> dict:
    args_ = args[2:]

    video_path = args_[0]

    if args_[1] == 'None':
        result_dir = None
    else:
        result_dir = args_[1]

    if args_[2] == 'True':
        show = True
    else:
        show = False

    YOLOv8_model = args_[3]
    UniDepth_model = args_[4]

    send_args = {
        'video_path': video_path,
        'result_dir': result_dir,
        'show': show,
        'YOLOv8_model': YOLOv8_model,
        'UniDepth_model': UniDepth_model
    }

    return send_args

def read_webcam_args(args: list) -> dict:
    args_ = args[2:]

    source = args_[0]
    source = int(source)

    YOLOv8_model = args_[1]
    UniDepth_model = args_[2]

    send_args = {

```

```
        'source': source,  
        'YOLOv8_model': YOLOv8_model,  
        'UniDepth_model': UniDepth_model  
    }  
  
    return send_args
```

					ПГУ 1.090301.40.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Оценка расстояния до объекта на изображении помощью нейронных сетей

Презентация

					ПГУ 1.090301.40.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

ПРОГРАММА ОЦЕНКИ РАССТОЯНИЯ ДО ОБЪЕКТА НА ИЗОБРАЖЕНИИ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

АВТОР РАБОТЫ: СТУДЕНТ ГРУППЫ 208ВВ12 ШМЕЛЕВ Д.В.
НАУЧНЫЙ РУКОВОДИТЕЛЬ: Д.Т.Н., ДОЦЕНТ МИТРОХИН М.А.

1

Цель и объект работы

Цель работы: изучение моделей глубокого обучения в области обнаружения объектов и прогнозирования карты глубины изображения.

Объектом разработки является программное обеспечение, основной задачей которого является обрабатывать изображение в формате RGB в трёх режимах работы:

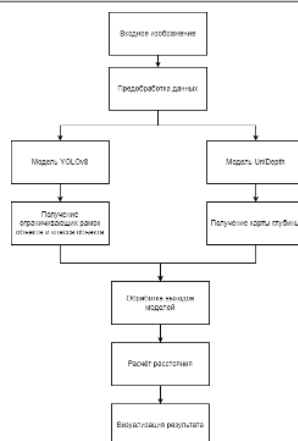
- Изображение;
- Видео;
- Вideoпоток с камеры в режиме реального времени

2

					ПГУ 1.090301.40.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		



Структура работы ПО



Результаты работ интегрированных моделей глубокого обучения

YOLOv8

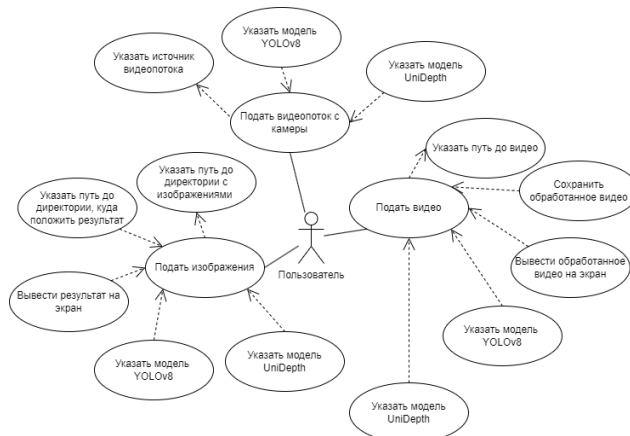


UniDepth



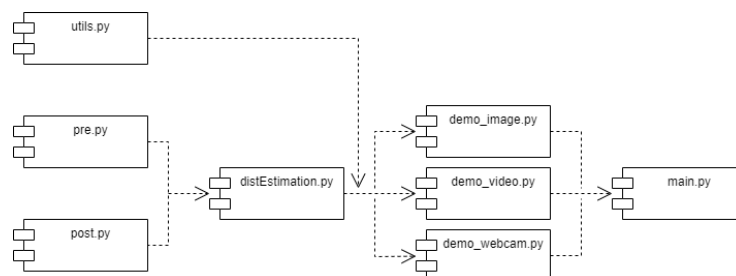
5

Варианты использования



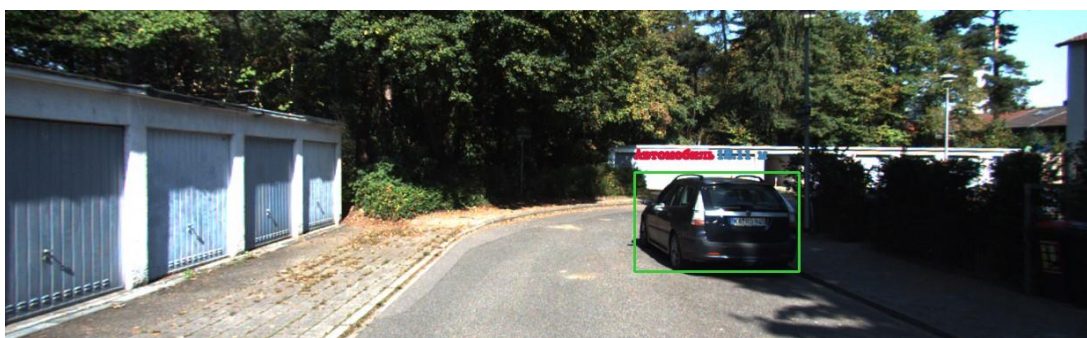
6

Архитектура ПО



7

Результат работы ПО



8

Сравнение полученных результатов

Исходное изображение с верной разметкой.

Класс: Автомобиль

Расстояние: 11.20 м



Исходное изображение с прогнозированной разметкой

Класс: Автомобиль

Расстояние: 12.11 м



9

Заключение

В ходе выполнения выпускной квалификационной работы, было разработано специализированное программное обеспечения для решения задачи измерения расстояния от объекта на изображении до камеры.

Реализованную программу можно будет использовать в различных областях применения:

- Автономное вождение
- AR/VR
- Сфера безопасности и наблюдения

10

					ПГУ 1.090301.40.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

Благодарю за внимание!

					ПГУ 1.090301.40.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		