

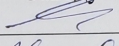
Е.И. Гурин

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» 04 2023 г.

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Еремин Александр Александрович гр.19ВВП2 факультета вычислительной техники.

2. Тема работы «Программа оценки расстояния до объектов по изображениям»

Тема утверждена приказом ПГУ №458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы д.т.н., заведующий кафедрой ВТ ПГУ Митрохин М.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

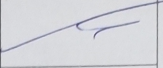
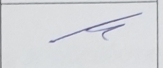
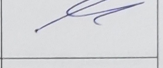
Назначение разработки – приложение по определению расстояния до объектов по изображениям.

Разработать программное обеспечение для определения расстояния до объектов по изображениям.

5. Перечень подлежащих разработке вопросов

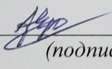
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать инструменты для разработки, реализовать приложение.

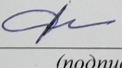
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	26.04.23 - 29.04.23	
2.	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	15 %	30.04.23 - 04.05.23	
3.	Подбор и обработка набора данных	20 %	05.05.23 - 09.05.23	
4.	Обучение и настройка нейронных сетей	25 %	10.05.23 - 21.05.23	
5.	Тестирование и отладка	15 %	22.05.23 - 24.05.23	
6.	Выполнение задания по БЖД	5 %	25.05.23 - 27.05.23	
7.	Оформление выпускной работы	10 %	28.05.23 - 31.05.23	

Дата выдачи задания "26" апреля 2023г.

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

Задание к исполнению принял студент  Еремин А.А.
(подпись, дата)

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

27.06.2023

Реферат

Бакалаврская работа содержит 66 листов, 19 рисунков, 6 источников, 3 приложения

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, МАШИННОЕ ОБУЧЕНИЕ, АНАЛИЗ ДАННЫХ *PYTHON, PYTORCH, OPENCV, ANACONDA, NUMPY*

Цель работы – разработать приложение, позволяющее по изображению определять расстояния до объектов

В ходе разработки необходимо осуществить поиск, подбор и обработку набора данных, для обучения детектора объектов. Обучить и настроить детектор для определения объектов на изображении, получить по изображению карту глубины и предсказать расстояние до объектов

В результате проведенной работы подобран и преобразован набор данных, обучен детектор, настроена сеть по получению карты глубины, использована система по оценке расстояния до объектов

ПГУ1.090301.28.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Еремин А.А.		
Провер.		Митрохин М.А.		
Н. Контр.		Бычков А.С.		
Утверд.		Митрохин М.А.		

Программа оценки расстояния до объектов по изображениям
Пояснительная записка

Лит.	Лист	Листов
	2	
ФВТ, каф. ВТ гр. 19ВВП2		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Разработка программы оценки расстояния до объектов по изображениям...	7
1.1 Анализ предметной области	7
1.1.1 Обзор существующих решений.....	8
1.1.2 Постановка задачи.....	11
1.2 Выбор средств разработки	12
1.2.1 Дистрибутив языков <i>Anaconda</i>	12
1.2.2 Язык программирования <i>Python</i>	13
1.2.3 Среда разработки <i>Spyder</i>	13
1.2.4 Фреймворк <i>Pytorch</i>	14
1.2.5 Библиотека <i>OpenCV</i>	14
1.2.6 Библиотека <i>Numpy</i>	15
1.2.7 Библиотека <i>Pandas</i>	15
1.3 Архитектура приложения.....	15
1.4 Реализация программного обеспечения	17
1.4.1 Поиск и подбор набора данных	17
1.4.2 Обработка набора данных	18
1.4.3 Обучение и настройка нейронных сетей.	24
1.5 Руководство пользователя.....	37
2 Безопасность жизнедеятельности.....	39
2.1 Требования к помещениям для работы с ПК	39
2.2 Требования к организации рабочих мест	40
Заключение	42

Список использованных источников	43
Приложение А - Архитектуры нейронных сетей и <i>UML</i> -диаграмма	44
Приложение Б - Листинг файлов.....	50
Приложение В - Презентация	62

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

ПК – персональный компьютер

ПО – программное обеспечение

ЯП – язык программирования

LSTM (Long short-term memory) - разновидность архитектуры рекуррентных нейронных сетей

KITTI (Karlsruhe Institute of Technology and Toyota Institute) - популярный набор данных, содержащий широкую информацию для исследований в области робототехники и автономных автомобилей

mAP (mean Average Precision) - метрика для измерения точности объектных детекторов

YOLO (You Only Look Once) - одноименная архитектура обнаружения объектов и искусственная нейронная сеть, на основе этой архитектуры

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		5

Введение

В современном мире технологии компьютерного зрения, основанные в большинстве случаев на технологии нейронных сетей, становятся все более важными в современном мире. Одной из серьёзных и необходимых заданий в области компьютерного зрения является определение расстояния до объектов по изображению. Эта задача имеет широкий спектр применений, начиная от автопилотов, автономных роботов и спортивных систем до систем безопасности и виртуальной реальности, медицинской диагностики и автоматизации.

Главной целью решения такой задачи является быстрая и точная оценка расстояния до объекта, что позволяет автономным системам принимать решения на основе этого расстояния. Например, в автомобильных автопилотах, определение расстояния является важным фактором, серьёзно влияющим на поведение системы принятия решений для обеспечения безопасности водителя и пассажиров.

В данной работе будут рассмотрены основные принципы и методы оценок расстояний до объектов по изображениям, будут выделены плюсы и минусы различных решений в данной области. На основе полученной информации и анализа предметной области будет создано приложение для оценки расстояния до объектов по изображению.

					ПГУ1.090301.28.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дат		

1 Разработка программы оценки расстояния до объектов по изображениям

1.1 Анализ предметной области

В современном мире для определения расстояния от наблюдателя до объекта используются специальные приборы - дальномеры. Существуют разные типы дальномеров: Лазерные, ультразвуковые, радиолокационные.

Все они работают по одному принципу: замеру времени, за которое сигнал от дальномера доходит до объекта, отражается от него и доходит до наблюдателя.

Использование дальномеров имеет свои преимущества и недостатки.

К преимуществам дальномеров можно отнести:

- высокую точность;
- быструю скорость;
- возможность измерения больших расстояний.

К недостаткам можно отнести:

- сильную чувствительность к видимости;
- высокую стоимость;
- наличие дополнительных знаний для правильного использования определенных типов дальномеров.

Из-за своих недостатков подобные приборы не всегда могут быть использованы в сферах, использующих встраиваемые системы. Тогда были разработаны методы определения расстояния до объектов, используя лишь изображение.

Среди преимуществ использования таких методов выделяют:

- большую гибкость;
- возможность определять расстояние до нескольких объектов сразу;
- использование дополнительной информации сцены;

Недостатками таких методов являются:

- зависимость от качества изображения;
- ограничение на дальность объектов;
- возможная вычислительная сложность;
- точность.

В целом, алгоритмы, использующие лишь одиночное изображение, для определения расстояния, стали пользоваться спросом из-за своей доступности, гибкости и простоты внедрения. Такие качества и делают их привлекательными для широкого круга пользователей, даже мобильных приложений. Помимо этого, область компьютерного зрения постоянно развивается и уже сейчас можно увидеть решения, серьёзно конкурирующие с дальномерами в точности.

1.1.1 Обзор существующих решений

Существует несколько методов определения расстояния до объектов по изображению. Чаще всего алгоритмы используют дополнительную информацию, например, о камере или параметры объекта.

Самый простой метод способ определить расстояние до объекта по изображению — это использовать формулу тонкой линзы [1]. Схема лучей представлена на рисунке 1.

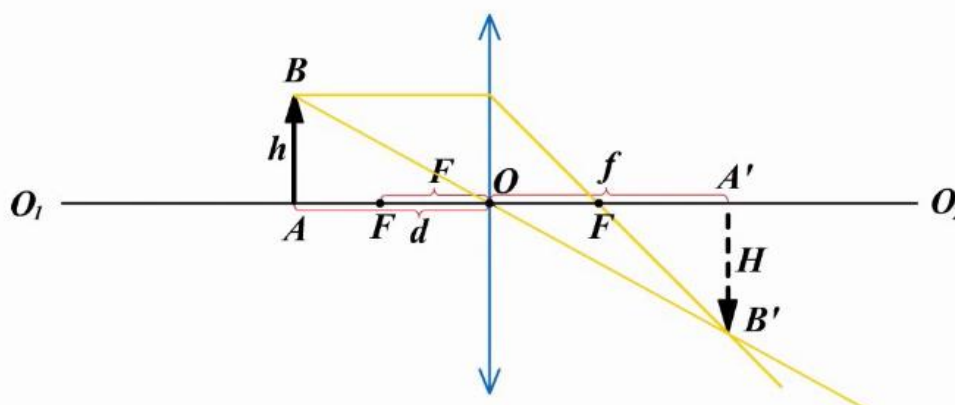


Рисунок 1 – Схема лучей тонкой линзы

На рисунке 1, d — это расстояние от объекта до линзы, f - расстояние от линзы до изображения, F - фокусное расстояние линзы, h - высота объекта, H - высота объекта на изображении.

Основной формулой для расчета является:

$$\frac{1}{d} \pm \frac{1}{f} = \pm \frac{1}{F}$$

Главное в этой формуле - избавиться от величины f и выделить d . В конечном итоге для определения расстояния до объекта понадобится знать: h - реальную высоту объекта, H - размер изображения объекта на матрице камеры, исходя из физических параметров камеры, F - фокусное расстояние, которое описывается в метаданных самой фотографии.

К достоинствам данного метода относится:

- простота использования;
- широкое применение.

К недостаткам такого метода относится:

- дополнительные требования к исходным данным;
- ограниченная точность.

Более серьезным и сложным подходом оценки расстояния является стереозрение. Метод стереозрения основан на том, как сам человек определяет расстояние до объектов - с помощью бинокулярного зрения, посредством определения мозгом разницы между изображениями, полученными от каждого глаза. Чтобы реализовать такой метод, понадобится две предварительно откалиброванных камеры, расположенные на определенном расстоянии друг от друга. Путем определения точек, находящихся одновременно на двух изображениях и разницы между ними, можно определить глубину каждой точки и определить расстояние до объекта. Данный метод очень популярен в задачах оценки расстояний до объектов.

Преимущества метода стереозрения:

- высокая точность;

- независимость от освещения и цвета;
- информация о глубине.

Недостатки метода:

- требование к настройке и калибровке;
- чувствительность к шуму;
- дополнительное оборудование;
- вычислительная сложность;
- требование к наличию нескольких изображений.

Метод использования *LIDAR* - представляет из себя метод сканирования местности с помощью специального устройства - *LIDAR*, которое формирует карту глубины. Путем наложения карты глубины на изображение можно определить расстояние до объектов.

Преимуществами такого метода являются:

- высокая точность;
- большой обзор;
- трехмерная информация;
- независимость от освещения.

Недостатки *LIDAR*:

- требование к дополнительному оборудованию;
- сложность обработки данных, полученных с *LIDAR*;
- высокие затраты.

Метод оценки глубины с использованием технологий нейронных сетей и машинного обучения, как и метод стереозрения является одним из наиболее популярных и интересных методов. Его суть заключается в обучении модели, которая по фотографии будет формировать карту глубины для изображения. Зная глубину пикселей объекта и реальное расстояние до объекта можно определить зависимость между ними и обучить другую модель предсказывать непосредственно расстояние до объектов.

Преимущества метода оценки глубины:

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		10

- применим к одиночным изображениями;
- не требует никакого дополнительного оборудования;
- гибкость в использовании.

Недостатками метода являются:

- зависимость от качества исходных данных для обучения;
- ограниченная дальность;
- вычислительная сложность;
- зависимость от качества изображений.

Проанализировав существующие решения, можно сделать вывод, что большинство из них требуют какой-либо дополнительной информации или оборудования, за исключением метода оценки глубины, использующего исключительно монокулярную информацию. Таким образом, выбор решения зависит от изначального оборудования пользователя, требуемой точности, условий, в которых будет получено изображение, и вычислительной мощности.

Каждый метод обладает своими преимуществами и недостатками, что позволяет подобрать решение почти к любой ситуации, в зависимости от предпочтений пользователя.

1.1.2 Постановка задачи

В последнее время большую популярность и интерес в исследованиях представляют системы оценок расстояния до объектов, основанных на технологии искусственных нейронных сетей и машинного обучения. Благодаря ним уходит потребность в самостоятельной разметке объектов и использовании дополнительного оборудования.

Таким образом, необходимо реализовать программу, которая по изображению сначала определяет количество и тип объектов, а потом расстояние до этих самых объектов.

Для этого сначала нужно подобрать набор данных, на котором будет обучен детектор, для обнаружения объектов в кадре. Этот же набор данных

будет использован для формирования карт глубины, с помощью которых можно будет определить расстояние до объектов, размеченных детектором. Помимо этого, набор данных должен обладать информацией о расстоянии.

1.2 Выбор средств разработки

Правильный выбор средств разработки сильно облегчает и помогает в создании ПО. Хорошо подобранные технологии существенно увеличивают качество конечного продукта, повышают эффективность разработки и облегчают процесс проектирования.

Чтобы реализовать программу оценки расстояния до объектов по изображению были использованы следующие языки и технологии:

- *Anaconda* – дистрибутив языков программирования *Python* и *R*;
- *Python* – интерпретируемый, высокоуровневый, объектно-ориентированный язык программирования с динамической типизацией;
- *Spyder* – интегрированная среда разработки для языка программирования *Python*;
- *Pytorch* – фреймворк машинного обучения и глубокого обучения, написанный для языка *python*;
- *OpenCV* – библиотека компьютерного зрения и обработки изображений;
- *Numpy* – библиотека для языка *Python* для работы с многомерными массивами;
- *Pandas* – библиотека для языка *Python* для обработки и анализа данных.

1.2.1 Дистрибутив языков *Anaconda*

Anaconda — это дистрибутив языков программирования *Python* и *R* для научных вычислений и анализа данных. Данный дистрибутив содержит всё необходимой для быстрой и легкой настройки виртуальных сред и встроенный интерпретатор.

Основными особенностями данной среды разработки являются:

- управление виртуальными средами для управления библиотеками и фреймворками для каждого отдельного проекта, что позволяет избегать конфликты между ними;
- удобный интерфейс для простой настройки и установки необходимых пакетов и программ;
- кроссплатформенность, позволяющая запускать данный дистрибутив на различных операционных системах;
- широкая база, содержащая большое количество научных библиотек и инструментов, как для анализа, обработки и визуализации данных, так и для обучения, настройки и анализа моделей машинного обучения и искусственных нейронных сетей.

1.2.2 Язык программирования *Python*

Python — это высокоуровневый язык программирования, который разработан в 1991 году Гвидо ван Россумом [2, 3].

Главными преимуществами *Python* являются простота и легкая читаемость кода, написанного на нем. Именно благодаря гибкости, динамической типизации и развитой экосистеме большая часть проектов и библиотек, посвященных анализу и обработке данных, написана именно на языке *Python*. Область использования языка программирования не ограничивается только областью машинного обучения, он также востребован в разработке веб-приложений, создании компьютерных игр и в научных вычислениях.

1.2.3 Среда разработки *Spyder*

Spyder — это интегрированная среда разработки для языка программирования *Python*. *Spyder* предоставляет удобную среду для написания, отладки и запуска кода.

Главные преимущества *Spyder*:

- удобный и простой интерфейс для работы с файлами и кодом;

- широкий функционал для исследовательской работы, такой как отображение всех изображений, графиков и значений переменных в отдельном окне с персональной настройкой;
- легкость и скорость работы по сравнению с другими *IDE*;
- кроссплатформенность.
- ориентированность на аналитиков данных и инженеров по искусственному интеллекту.

1.2.4 Фреймворк *Pytorch*

Pytorch — это фреймворк машинного обучения для *Python* с открытым исходным кодом.

Данный фреймворк предоставляет интерфейс для создания, обучения и настройки нейронных сетей. Именно на нем написан один из самых эффективных детекторов для выделения или сегментации объектов на изображении - *YOLO* [4].

Pytorch, в сравнении со своим главным конкурентом *TensorFlow*, отличается простотой использования, но при этом менее производительный для крупномасштабных проектов и с относительно молодым сообществом.

1.2.5 Библиотека *OpenCV*

OpenCV — это библиотека компьютерного зрения и обработки изображений. С помощью этой библиотеки легко выполнять манипуляции с изображениями, такие как:

- фильтрация;
- изменение размера;
- аффинные преобразования и преобразование цветового пространства;
- удаление шумов;
- обнаружение особых точек;
- выделение границ объектов.

OpenCV чаще других библиотек для обработки изображений используется в проектах, связанных с компьютерным зрением.

1.2.6 Библиотека *Numpy*

Numpy — это библиотека, предоставляющая мощные инструменты для работы с многомерными массивами и выполнения математических операций над ними.

Numpy содержит все необходимое для быстрых и оптимизированных расчетов различных типов массивов, так как большая часть данных для обучения и обработки представлена именно в этой структуре данных. Благодаря своей производительности *Numpy* является неотъемлемой частью большинства проектов в области научных вычислений.

1.2.7 Библиотека *Pandas*

Pandas - это библиотека, предназначенная для обработки и анализа данных. Она предоставляет структуры данных, с помощью которых удобно визуализировать и изменять данные для анализа. В общем и целом, *Pandas* оперирует со структурой данных, называемой *DataFrame*. *DataFrame* - это таблица данных с метками столбцов и строк и *Pandas* позволяет выполнять различные операции над этой таблицей, такие как фильтрация, сортировка, группировка, объединение и т.п. Также такую таблицу можно конвертировать в различные форматы: *CSV*, *JSON*, *HTML*, *SQL*.

1.3 Архитектура приложения

В зависимости от задачи, архитектура приложений, основанных на нейронных сетях, может сильно варьироваться. Они могут быть сделаны на основе всех основных видов современных архитектур или встроены в них.

В общем виде, приложение по оценке расстояния до объектов показано на рисунке 2.

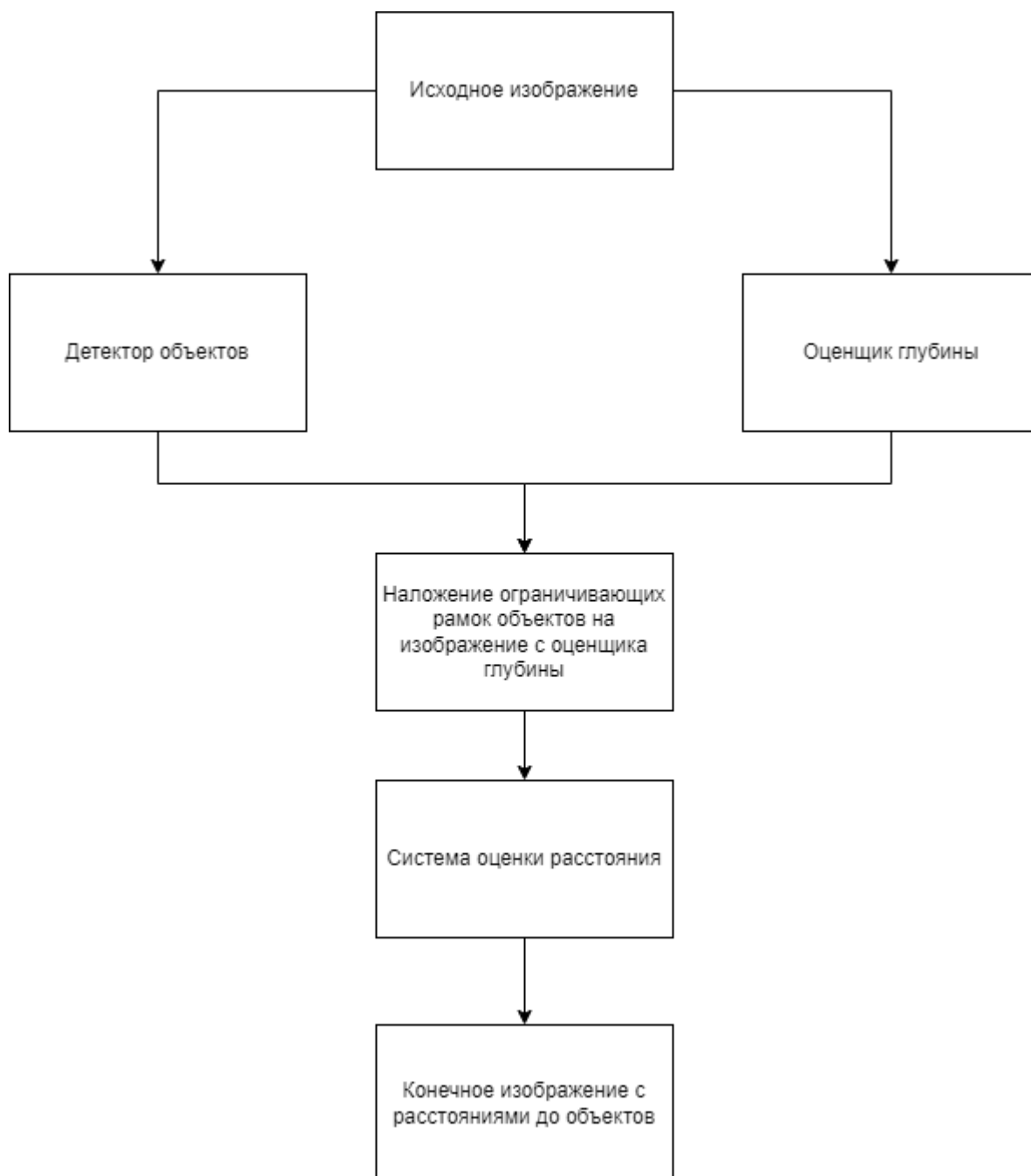


Рисунок 2 – Архитектура приложения

Необходимо сделать несколько шагов: поиск и подбор набора данных, предобработка набора данных, поиск и настройка нейронной сети, обучение и оптимизация нейронной сети.

1.4 Реализация программного обеспечения

1.4.1 Поиск и подбор набора данных

Подбор набора данных является очень важным шагом, сильно влияющим на конечное качество и эффективность нейронной сети. Хороший набор данных имеет правильную разметку изображений, разнообразие данных и, что самое главное, хорошее качество самих изображений. В зависимости от задачи, существует множество информационных наборов для компьютерного зрения. Самые популярные среди них: *ImageNet*, содержащий 14 миллионов изображений, *MNIST*, содержащий рукописные цифры, *MS COCO*, содержащий изображения для сегментации и распознавания объектов. В данной работе будет использован набор данных *KITTI*.

KITTI - является одним из наиболее известных и широко используемых наборов данных для разработки и тестирования алгоритмов компьютерного зрения и восприятия автомобилей. Его главная особенность — это его направленность на помощь в исследованиях в области автономного вождения. Поэтому он содержит широкую информацию как о 2D, так и о 3D сцене, данные *LiDAR*, стереокамеры, но при этом не содержит данных для сегментации. Всего набор данных содержит 7481 фотографий дорожных магистралей и городской местности, сделанных на автономной платформе. Сама автономная платформа представляет из себя автомобиль, с установленными на его крыше датчиками и камерами. Сами изображения имеют расширение 1242 на 375 пикселей и следующие классы: машина, автобус, грузовик, трамвай, пешеход, велосипедист, другое и то, что не имеет значения. Пример изображения из набора данных представлен на рисунке 3.



Рисунок 3 – Изображение из набора данных

Размеченное изображение представлено на рисунке 4.



Рисунок 4 – Размеченное изображение

1.4.2 Обработка набора данных

После выбора набора данных необходимо преобразовать его метки в нужный формат для детектора *YOLO*, а также правильно оформить файловую структуру. Изначальный формат меток представлен на рисунке 5.

```

Truck 0.00 0 -1.57 599.41 156.40 629.75 189.25 2.85 2.63 12.34
0.47 1.49 69.44 -1.56
Car 0.00 0 1.85 387.63 181.54 423.81 203.12 1.67 1.87
3.69 -16.53 2.39 58.49 1.57
Cyclist 0.00 3 -1.65 676.60 163.95 688.98 193.93 1.86 0.60 2.02
4.59 1.32 45.84 -1.55
DontCare -1 -1 -10 503.89 169.71 590.61
190.13 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 511.35 174.96 527.81
187.45 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 532.37 176.35 542.68
185.27 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 559.62 175.83 575.40
183.15 -1 -1 -1 -1000 -1000 -1000 -10

```

Рисунок 5 – Формат аннотации *KITTI*

В этом файле аннотации содержится следующая информация по порядку:

- тип объекта. Машина, пассажир, велосипедист и т.д;
- усечение;
- степень закрытия объекта другими объектами. Полностью виден, частично закрыт, в значительной степени закрыт или неизвестно;
- угол наблюдения объекта;
- 4 значения двумерной ограничивающей рамки. Содержит координаты левой, верхней, правой и нижней границ объекта;
- 3 значения трехмерного измерения объекта. Высота, ширина и длина в метрах;
- 3 значения координаты локации объекта. Координаты x, y, z в метрах;
- угол вращения;
- степень уверенности в правильности определения объекта.

Большая часть данных в аннотации не будет использовано для детектора, помимо этого, *KITTI* использует свой собственный формат аннотации и ограничивающей рамки, поэтому необходимо преобразовать метки в формат *YOLO* и удалить ненужную информацию для обучения. *YOLO* формат аннотации представлен на рисунке 6.

```

0 0.593229 0.475463 0.158333 0.456481
1 0.363021 0.402315 0.083333 0.069444
1 0.436198 0.397685 0.025521 0.056481
2 0.489583 0.338889 0.087500 0.229630
0 0.634896 0.392130 0.026042 0.106481
0 0.668229 0.388426 0.026042 0.091667
0 0.697656 0.394444 0.016146 0.081481
1 0.933073 0.418981 0.133854 0.136111
2 0.047396 0.390278 0.093750 0.236111

```

Рисунок 6 – Формат аннотации YOLO

Здесь содержится информация о типе объекта и его ограничивающей двумерной рамки в формате: центр x, центр y, ширина и высота в диапазоне от 0 до 1.

Ниже представлен фрагмент кода для преобразования:

```

from PIL import Image
import argparse
import os
import csv

OUT_LABELS_DIR = "labels"

KEY_PEDESTRIAN = "Pedestrian"
KEY_CYCLIST = "Cyclist"
KEY_CAR = "Car"
KEY_VAN = "Van"
KEY_MISC = "Misc"
KEY_TRUCK = "Truck"
KEY_PERSON_SITTING = "Person_sitting"
KEY_TRAM = "Tram"
KEY_DONT_CARE = "DontCare"

CLAZZ_NUMBERS = {
    KEY_PEDESTRIAN : 0,
    KEY_CYCLIST : 1,
    KEY_CAR : 2,
    KEY_DONT_CARE : 3
}

def getSampleId(path):
    basename = os.path.basename(path)
    return os.path.splitext(basename)[0]

def resolveClazzNumberOrNone(clazz, use_dont_care):
    if clazz == KEY_CYCLIST:
        return CLAZZ_NUMBERS[KEY_CYCLIST]

```

```

    if clazz == KEY_PEDESTRIAN:
        return CLAZZ_NUMBERS[KEY_PEDESTRIAN]
    if clazz == KEY_CAR:
        return CLAZZ_NUMBERS[KEY_CAR]
    if clazz == KEY_DONT_CARE:
        return CLAZZ_NUMBERS[KEY_DONT_CARE]
    return None

def convertToYoloBBox(bbox, size):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (bbox[0] + bbox[1]) / 2.0
    y = (bbox[2] + bbox[3]) / 2.0
    w = bbox[1] - bbox[0]
    h = bbox[3] - bbox[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def readRealImageSize(img_path):
    return Image.open(img_path).size

def readFixedImageSize():
    return (1242, 375)

def parseSample(lbl_path, img_path, use_dont_care):
    with open(lbl_path) as csv_file:
        reader = csv.DictReader(csv_file, fieldnames=["type",
"truncated", "occluded", "alpha", "bbox2_left", "bbox2_top",
"bbox2_right", "bbox2_bottom", "bbox3_height", "bbox3_width",
"bbox3_length", "bbox3_x", "bbox3_y", "bbox3_z", "bbox3_yaw",
"score"], delimiter=" ")
        yolo_labels = []
        for row in reader:
            clazz_number = resolveClazzNumberOrNone(row["type"], use_dont_care)
            if clazz_number is not None:
                size = readRealImageSize(img_path)
                bbox = (
                    float(row["bbox2_left"]),
                    float(row["bbox2_right"]),
                    float(row["bbox2_top"]),
                    float(row["bbox2_bottom"])
                )
                yolo_bbox = convertToYoloBBox(bbox, size)
                yolo_label = (clazz_number,) + yolo_bbox
                yolo_labels.append(yolo_label)
    return yolo_labels

def parseArguments():

```

```

        parser = argparse.ArgumentParser(description="Generates
labels for training darknet on KITTI.")
        parser.add_argument("label_dir",
help="data_object_label_2/training/label_2    directory;    can    be
downloaded from KITTI.")
        parser.add_argument("image_2_dir",
help="data_object_image_2/training/image_2    directory;    can    be
downloaded from KITTI.")
        parser.add_argument("--training-samples",            type=int,
default=0.8, help="percentage of the samples to be used for
training between 0.0 and 1.0.")
        parser.add_argument("--use-dont-care",
action="store_false", help="do not ignore 'DontCare' labels.")
        args = parser.parse_args()
        if args.training_samples < 0 or args.training_samples > 1:
            print("Invalid argument {} for --training-samples.
Expected a percentage value between 0.0 and 1.0.")
            exit(-1)
        return args

def main():
    args = parseArguments()

    if not os.path.exists(OUT_LABELS_DIR):
        os.makedirs(OUT_LABELS_DIR)

    sample_img_pathes = []
    for dir_path, sub_dirs, files in os.walk(args.label_dir):
        for file_name in files:
            if file_name.endswith(".txt"):
                lbl_path = os.path.join(dir_path, file_name)
                sample_id = getSampleId(lbl_path)
                img_path = os.path.join(args.image_2_dir,
"{}.png".format(sample_id))
                sample_img_pathes.append(img_path)
                yolo_labels = parseSample(lbl_path, img_path,
args.use_dont_care)
                with open(os.path.join(OUT_LABELS_DIR,
"{}.txt".format(sample_id)), "w") as yolo_label_file:
                    for lbl in yolo_labels:
                        yolo_label_file.write("{} {} {} {}
{}\n".format(*lbl))

            print("Writing training and test sample ids...")
            first_test_sample_index = int(args.training_samples *
len(sample_img_pathes))
            with open("kitti_train.txt", "w") as train_file:
                for sample_index in range(first_test_sample_index):
                    train_file.write("{}\n".format(sample_img_pathes[sample_index]))
            with open("kitti_test.txt", "w") as test_file:

```

```

        for sample_index in range(first_test_sample_index,
len(sample_img_pathes)):

test_file.write("{}\n".format(sample_img_pathes[sample_index]))

if __name__ == "__main__":
    main()

```

С помощью него все файлы аннотации для каждого изображения из формата *KITTI* переводятся в формат *YOLO*, при этом было принято решение отсечь некоторые классы и оставить лишь пешеходов, машины, велосипедистов и то, что не имеет значение.

Помимо этого, файловая структура расположения изображений и их соответствующих аннотаций была сделана следующим образом. (рис 7.).

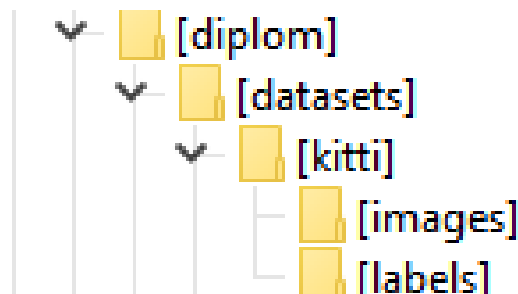


Рисунок 7 – Формат файловой структуры

Также был написан скрипт для генерации текстовых файлов, содержащих пути ко всем файлам с изображениями и аннотациям. Код файла *DatasetHandler.py* представлен ниже:

```

import os
import glob

DATASET_PATH = "./datasets/kitti/"
PATH_IMAGES = "./datasets/kitti/images/"
PATH_LABELS = "./datasets/kitti/labels/training/label_2/"
TRAIN_NAME_TXT = "train.txt"
TEST_NAME_TXT = "test.txt"

def TxtPatchCreator(path, path_save):
    path_train_images = path + "training/image_2/"
    path_test_images = path + "testing/image_2/"

    list_training = os.listdir(path_train_images)
    list_test = os.listdir(path_test_images)

```

```

train_file_path = path_save + "train.txt"
test_file_path = path_save + "test.txt"

train_file = open(train_file_path, "w")
test_file = open(test_file_path, "w")

for name_img in list_training:

train_file.write(os.path.join(os.path.abspath(path_train_images),
name_img))
        train_file.write("\n")

        for name_img in list_test:

test_file.write(os.path.join(os.path.abspath(path_test_images),
name_img))
        test_file.write("\n")

train_file.close()
test_file.close()
TxtPatchCreator(PATH_IMAGES, DATASET_PATH)

```

Результат работы представлен на рисунке 8.



The image shows a screenshot of a text file with a list of 12 image paths. Each path is on a new line and follows the format: E:\diplom\datasets\kitti\images\training\image\000000.png, where the number at the end of the path increases from 000000 to 000012.

Рисунок 8 – Формат файла с путями до изображений

1.4.3 Обучение и настройка нейронных сетей.

В качестве детектора была выбрана самая современная на данный момент сверточная нейронная сеть на архитектуре *YOLO* - *YOLOv8*. Она представляет из себя одну из самых популярных и эффективных моделей для задачи обнаружения объектов. Основная идея такого детектора состоит в том, чтобы

применить сверточную нейронную сеть к изображению только один раз и сразу получить предсказание о наличии и позиции всех объектов на изображении. Такой подход, в отличие от других, позволяет существенно увеличить скорость обучения и прохода для предсказания.

При обнаружении объектов, *YOLO* разбивает входное изображение на сетку ячеек и применяет сверточные фильтры для каждой ячейки. Каждая ячейка отвечает за предсказание ограничивающей рамки и класса объекта внутри неё. Архитектура детектора представлена на рисунке 9.

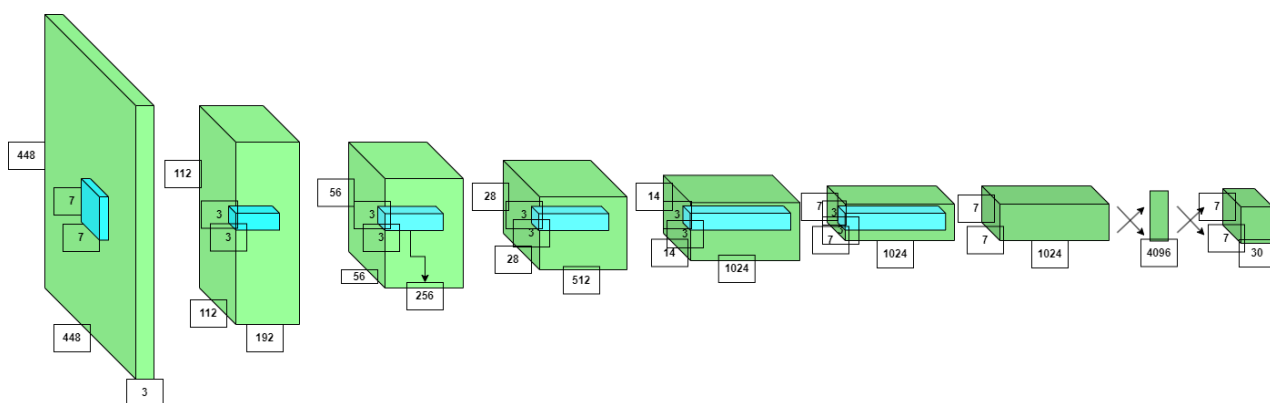


Рисунок 9 – Архитектура *YOLO*

Перед тем, как запустить непосредственно обучение, был создан файл *kitti.yaml*, содержащий путь к набору данных и номера и названия классов (рисунок 10).

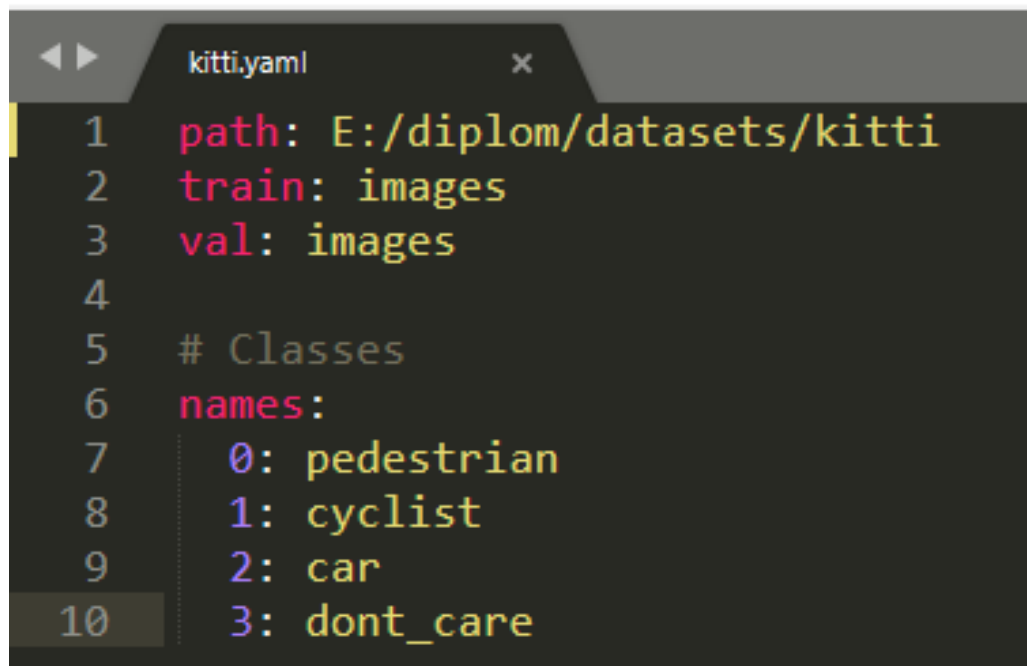


Рисунок 10 – Файл классов и путей до набора данных

Фрагмент кода для обучения представлен ниже:

```

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

from ultralytics import YOLO

# Load the model.
model = YOLO('yolov8m.pt')

# Training.
results = model.train(

data='E:/diplom/yolov8/ultralytics/ultralytics/datasets/kitti.yaml',
    imgsz=640,
    epochs=10,
    batch=8,
    name='yolov8m_kitti_10ep')

model =
YOLO("C:/Users/Limitless/runs/detect/yolov8m_kitti_10ep/weights/best.pt")

```

Обучение происходило на видеокарте *Nvidia Geforce GTX 1050 TI*. Общее время обучения 4 часа на 10 эпохах. Размер изображений: 640x640 пикселей.

В результате сеть показала отличные данные, не только по точности определения объекта, но и по метрике mAP . Результаты обучения представлены на рисунках 11 и 12.

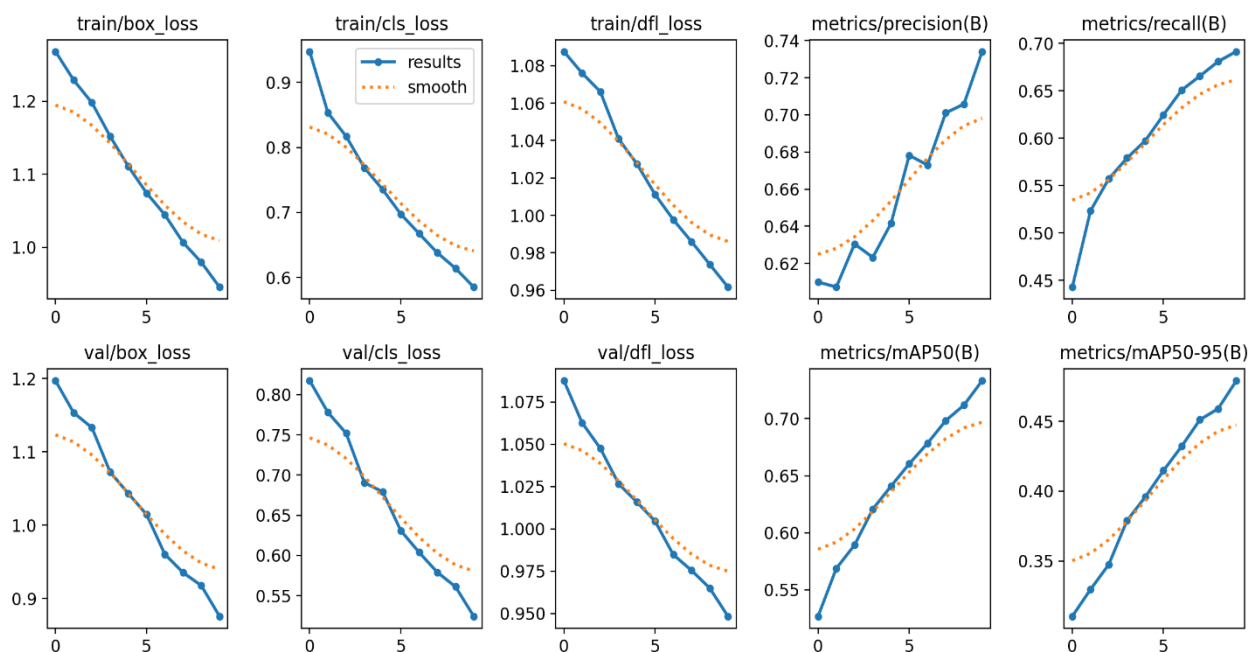


Рисунок 11 – Результат обучения 1

```
Validating runs\detect\yolov8m_kitti_10ep\weights\best.pt...
Ultralytics YOLOv8.0.121 Python-3.8.16 torch-2.0.1 CUDA:0 (NVIDIA GeForce GTX 1050 Ti, 4096MiB)
Model summary (fused): 218 layers, 25842076 parameters, 0 gradients
Class      Images  Instances  Box(P)      R      mAP50  mAP50-95): 100%|██████████| 468/468 [03:22<00:00,
2.31it/s]
      all       7481      46151      0.734      0.691      0.733      0.479
  pedestrian  7481      4487      0.819      0.745      0.813      0.476
    cyclist  7481      1627      0.719      0.814      0.832      0.543
        car  7481     28742      0.889      0.928      0.96      0.783
    dont_care 7481     11295      0.509      0.278      0.329      0.114
Speed: 0.1ms preprocess, 17.2ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs\detect\yolov8m_kitti_10ep
```

Рисунок 12 – Результат обучения 2

Изображения, пройденные через детектор показаны на рисунках 13,14.



Рисунок 13 – Размеченное детектором изображение 1

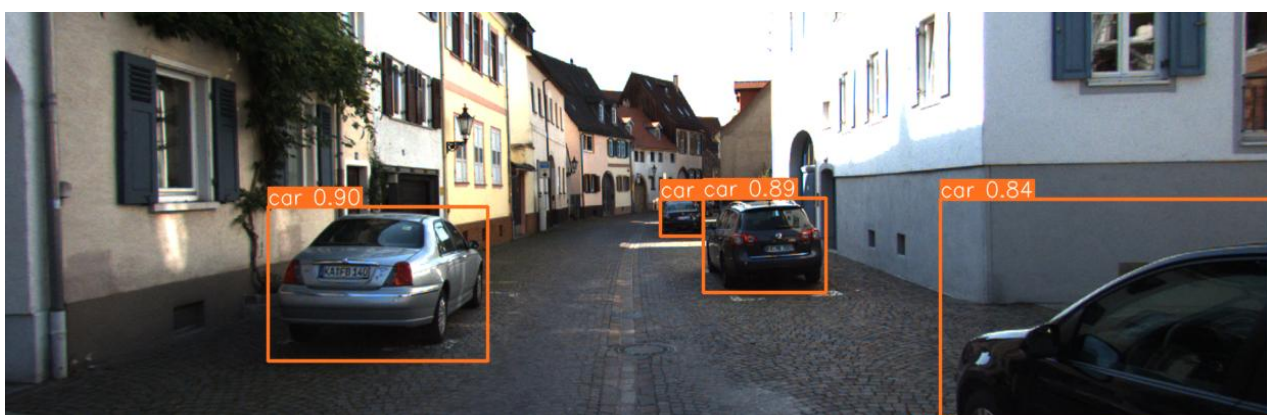


Рисунок 14 – Размеченное детектором изображение 2

Следующим этапом была использована сеть *MIDAS* [5]. *MDAS* — это нейронная сеть модели энкодер-декор, которая использует сверточную нейронную сеть для извлечения признаков и обработки изображений. Главной задачей для *MIDAS* было использование её для генерации карт глубины. Карта глубины позволяет оценить расстояние до объектов на изображении. *MIDAS* предварительно обучена на 12 наборах данных, поэтому не требует какой-либо настройки или дообучения. Изображение карты глубины представлено на рисунке 15.

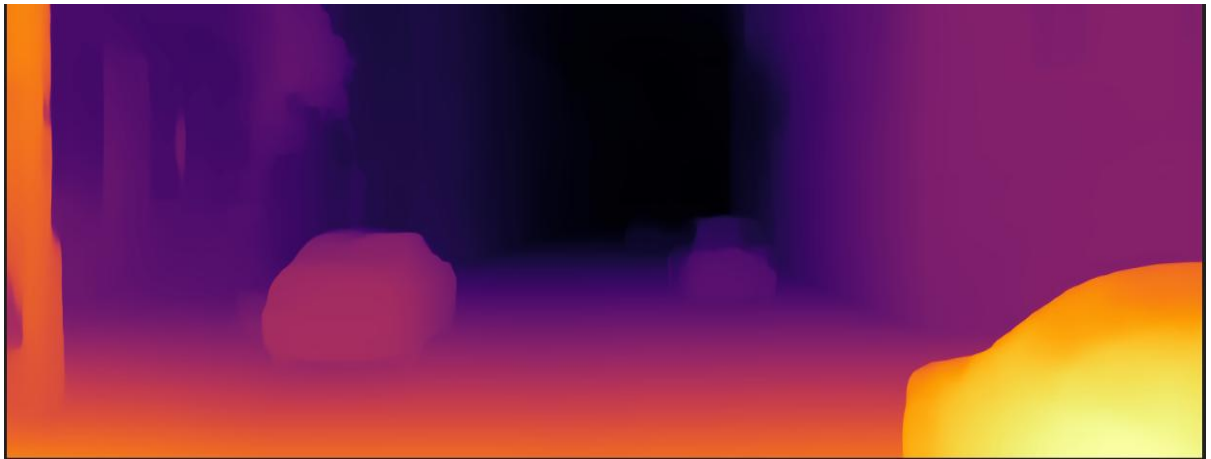


Рисунок 15 – Карта глубины

Фрагмент кода для запуска *MIDAS* представлен ниже:

```
def process(device, model, model_type, image, input_size,
target_size, optimize, use_camera):
    global first_execution

    if "openvino" in model_type:
        if first_execution or not use_camera:
            print(f"    Input resized to
{input_size[0]}x{input_size[1]} before entering the encoder")
            first_execution = False

        sample = [np.reshape(image, (1, 3, *input_size))]
        prediction = model(sample)[model.output(0)][0]
        prediction = cv2.resize(prediction, dsize=target_size,
                                interpolation=cv2.INTER_CUBIC)
    else:
        sample =
torch.from_numpy(image).to(device).unsqueeze(0)

        if optimize and device == torch.device("cuda"):
            if first_execution:
                print("    Optimization to half-floats
activated. Use with caution, because models like Swin require\n"
                    "    float precision to work properly and
may yield non-finite depth values to some extent for\n"
                    "    half-floats.")
                sample =
sample.to(memory_format=torch.channels_last)
                sample = sample.half()

            if first_execution or not use_camera:
                height, width = sample.shape[2:]
```

```

        print(f"    Input resized to {width}x{height}
before entering the encoder")
        first_execution = False

        prediction = model.forward(sample)
        prediction = (
            torch.nn.functional.interpolate(
                prediction.unsqueeze(1),
                size=target_size[:-1],
                mode="bicubic",
                align_corners=False,
            )
            .squeeze()
            .cpu()
            .numpy()
        )

    return prediction

def create_side_by_side(image, depth, grayscale):

    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    print("Device: %s" % device)

    model, transform, net_w, net_h = load_model(device,
model_path, model_type, optimize, height, square)

    if input_path is not None:
        image_names = glob.glob(os.path.join(input_path, "*"))
        num_images = len(image_names)
    else:
        print("No input path specified. Grabbing images from
camera.")

    if output_path is not None:
        os.makedirs(output_path, exist_ok=True)

    print("Start processing")

    if input_path is not None:
        if output_path is None:
            print("Warning: No output path specified. Images
will be processed but not shown or stored anywhere.")
            for index, image_name in enumerate(image_names):

                print("    Processing {} ({} / {})".format(image_name,
index + 1, num_images))

                # input

```

```

original_image_rgb = utils.read_image(image_name)
# in [0, 1]
image = transform({"image":
original_image_rgb})["image"]

# compute
with torch.no_grad():
    prediction = process(device, model,
model_type, image, (net_w, net_h), original_image_rgb.shape[1::-1],
optimize, False)

# output
if output_path is not None:
    filename = os.path.join(
        output_path,
os.path.splitext(os.path.basename(image_name))[0] + '-' +
model_type
    )
    if not side:
        utils.write_depth(filename, prediction,
grayscale, bits=2)
    else:
        original_image_bgr =
np.flip(original_image_rgb, 2)
        content =
create_side_by_side(original_image_bgr*255, prediction,
grayscale)
        cv2.imwrite(filename + ".png", content)
        utils.write_pfm(filename + ".pfm",
prediction.astype(np.float32))

else:
    with torch.no_grad():
        fps = 1
        video = VideoStream(0).start()
        time_start = time.time()
        frame_index = 0
        while True:
            frame = video.read()
            if frame is not None:
                original_image_rgb = np.flip(frame, 2) #
in [0, 255] (flip required to get RGB)
                image = transform({"image":
original_image_rgb/255})["image"]

                prediction = process(device, model,
model_type, image, (net_w, net_h),
original_image_rgb.shape[1::-1], optimize, True)

```

```

original_image_bgr =
np.flip(original_image_rgb, 2) if side else None
content =
create_side_by_side(original_image_bgr, prediction, grayscale)
cv2.imshow('MiDaS Depth Estimation - Press
Escape to close window ', content/255)

if output_path is not None:
    filename = os.path.join(output_path,
'Camera' + '-' + model_type + '_' + str(frame_index))
    cv2.imwrite(filename + ".png",
content)

alpha = 0.1
if time.time()-time_start > 0:
    fps = (1 - alpha) * fps + alpha * 1 /
(time.time()-time_start) # exponential moving average
    time_start = time.time()
print(f"\rFPS: {round(fps,2)}", end="")

if cv2.waitKey(1) == 27: # Escape key
    break

frame_index += 1

print()

print("Finished")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument('-i', '--input_path',
                        default=None,
                        help='Folder with input images (if no
input path is specified, images are tried to be grabbed '
                        'from camera)'
                    )

    parser.add_argument('-o', '--output_path',
                        default=None,
                        help='Folder for output images'
                    )

    parser.add_argument('-m', '--model_weights',
                        default=None,
                        help='Path to the trained weights of
model'
                    )

    parser.add_argument('-t', '--model_type',
                        default='dpt_swin2_large_384',

```

```

        help='Model type: '
            'dpt_beit_large_512,
dpt_beit_large_384, dpt_beit_base_384, dpt_swin2_large_384, '
            'dpt_swin2_base_384,
dpt_swin2_tiny_256, dpt_swin_large_384, dpt_next_vit_large_384, '
            'dpt_levit_224, dpt_large_384,
dpt_hybrid_384, midas_v21_384, midas_v21_small_256 or '
            'openvino_midas_v21_small_256'
        )

    parser.add_argument('-s', '--side',
                        action='store_true',
                        help='Output images contain RGB and
depth images side by side'
    )

    parser.add_argument('--optimize', dest='optimize',
                        action='store_true', help='Use half-float optimization')
    parser.set_defaults(optimize=False)

    parser.add_argument('--height',
                        type=int, default=None,
                        help='Preferred height of images feed
into the encoder during inference. Note that the '
                        'preferred height may differ from
the actual height, because an alignment to multiples of '
                        '32 takes place. Many models
support only the height chosen during training, which is '
                        'used automatically if this
parameter is not set.'
    )

    parser.add_argument('--square',
                        action='store_true',
                        help='Option to resize images to a
square resolution by changing their widths when images are '
                        'fed into the encoder during
inference. If this parameter is not set, the aspect ratio of '
                        'images is tried to be preserved
if supported by the model.'
    )

    parser.add_argument('--grayscale',
                        action='store_true',
                        help='Use a grayscale colormap instead
of the inferno one. Although the inferno colormap, '
                        'which is used by default, is
better for visibility, it does not allow storing 16-bit '
                        'depth values in PNGs but only 8-
bit ones due to the precision limitation of this '
                        'colormap.'
    )

    args = parser.parse_args()

```

```

if args.model_weights is None:
    args.model_weights = default_models[args.model_type]

torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark = True

run(args.input_path, args.output_path, args.model_weights,
    args.model_type, args.optimize, args.side, args.height,
    args.square, args.grayscale)

```

Ограничивающие рамки были наложены на изображения с картой глубины с помощью следующего алгоритма:

```

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

from ultralytics import YOLO

model = YOLO('yolov8m.pt')

result =
model.predict(source="E:/diplom/datasets/kitti/images/000050.png"
)

import cv2
import random

names = model.names
img = cv2.imread("E:/diplom/midas/output/000050-
dpt_beit_large_512.png")

t1 = round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1
for r in result:
    boxes = r.boxes
    for box in boxes:
        b = box.xyxy[0]
        c = box.cls
        x1, x2 = (int(b[0]), int(b[1])), (int(b[2]),
int(b[3]))
        color = (0, 0, 255)
        cv2.rectangle(img, x1, x2, color, t1, cv2.LINE_AA)
        cv2.putText(img, names[int(c)], (x1[0], x1[1] - 2), 0,
t1 / 3, [225, 255, 255], thickness=t1, lineType=cv2.LINE_AA)
        cv2.imwrite("output.png", img)

```

На рисунке 16 представлен результат работы алгоритма

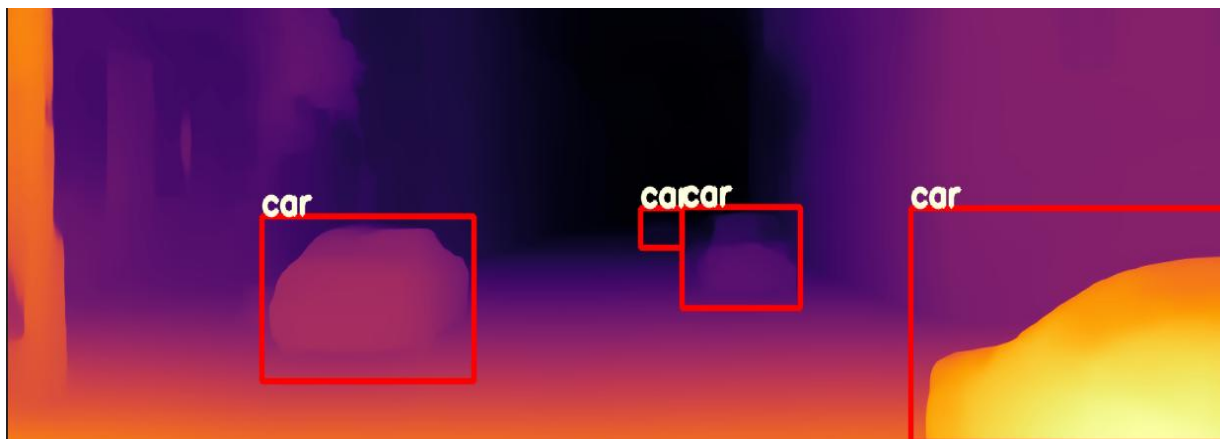


Рисунок 16 – Результат разметки на карте глубины

Последним этапом будет запуск системы оценки расстояния до объектов. Система оценки расстояния основана на получении информации с карты глубины и ограничивающих рамок объектов [6]. Её основная логика — это получить на вход изображение карты глубины и рамки объектов. Сама система представляет из себя алгоритм машинного обучения, который предварительно обучен на похожем наборе данных. На выход алгоритм выдаст расстояния для каждого объекта. По итогу мы получим изображение с предсказанным расстоянием до каждого объекта в метрах (рисунок 17). Код модели представлен ниже:

```
class Zloc_Estimaotor(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim,
output_dim=1):
        super().__init__()

        self.rnn = nn.LSTM(input_dim, hidden_dim, layer_dim,
batch_first=True, bidirectional=False)

        #Layer
        layersize=[306, 154, 76]
        layerlist= []
        n_in=hidden_dim
        for i in layersize:
            layerlist.append(nn.Linear(n_in,i))
            layerlist.append(nn.ReLU())
            #layerlist.append(nn.BatchNorm1d(i))
            #layerlist.append(nn.Dropout(0.1))
            n_in=i
        layerlist.append(nn.Linear(layersize[-1],1))
```

```

#layerlist.append(nn.Sigmoid())

self.fc=nn.Sequential(*layerlist)

def forward(self, x):
    out, hn = self.rnn(x)
    output = self.fc(out[:,-1])
    return output

```

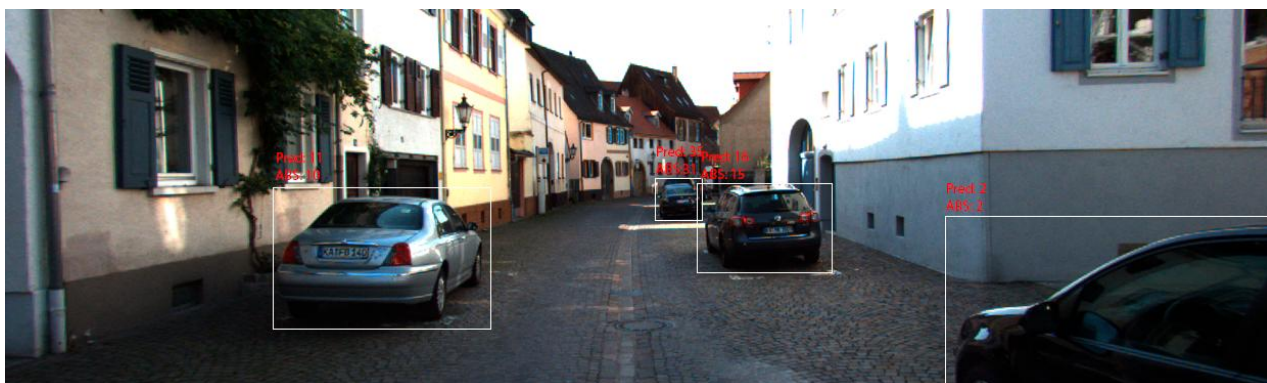


Рисунок 17 – Результат работы системы оценки расстояния.

1.5 Руководство пользователя

В результате, первое, что должен сделать пользователь — это подготовить, разметить и преобразовать свой собственный набор данных, для обучения детектора. Для этого можно воспользоваться программой *Labelimg*. Сохранить аннотацию нужно в формате *YOLO*. Далее, нужно загрузить оригинальный репозиторий *YOLOv8*, с помощью команды *git clone*. Далее, нужно запустить файл *DatasetHandler.py*, который создаст пути к изображениям и к аннотациям. В конечном итоге, необходимо оформить файловую структуру (рисунок 18).



Рисунок 18 – Результат работы оценки расстояния.

Запустив файл *train.py* и указав путь к набору данных можно начинать процесс обучения. Длительность процесса может достигать от нескольких минут до нескольких дней или часов, в зависимости от количества эпох обучения, количества изображений и объектов на них.

В конце обучения пользователь получает обученную модель, способную определять объекты на изображении. С помощью файла *YoloVisualize.py* можно визуализировать работу детектора.

Следующим этапом будет запуск сети для генерации карты глубины на изображении. Для этого необходимо лишь поместить нужные для этого изображения в любую папку и запустить *MIDAS* командой *python run.py --model_type dpt_beit_large_512 --input_path <папка с изображениями> --output_path <папка для сохранения новых изображений>*. В результате у пользователя будут изображения с картой глубины (рисунок 19).

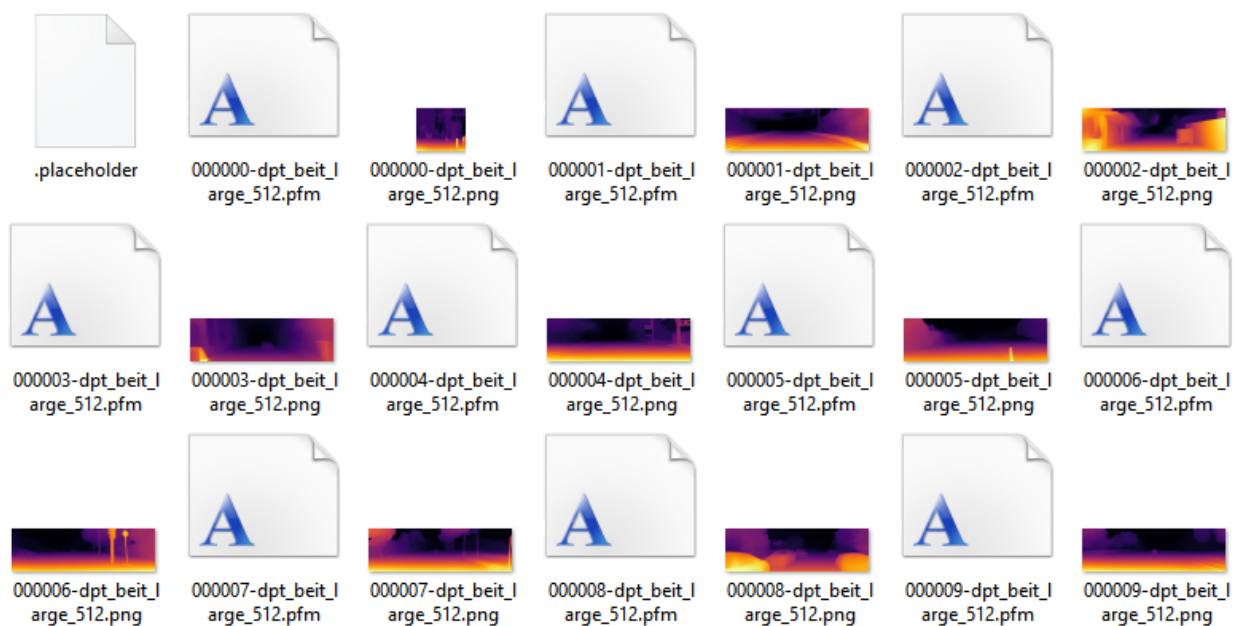


Рисунок 19 – Изображения с картой глубины.

Далее нужно запустить систему оценки расстояния.

2 Безопасность жизнедеятельности

2.1 Требования к помещениям для работы с ПК

Для работы с компьютером требуется рабочее место, которое обеспечивает комфорт, эффективность и безопасность.

Рабочее пространство должно иметь достаточно места для размещения компьютера, стола, стула и других необходимых предметов. Рекомендуется, чтобы помещение имело площадь не менее 6-8 квадратных метров.

В помещении должна быть хорошая вентиляция, поддерживающая свежий воздух и регулицию температуры.

В помещении должно быть достаточно розеток для подключения компьютера и других устройств. Проводка должна быть в исправном состоянии, без повреждений. Рекомендуется использовать стабилизаторы напряжения или ИБП (источники бесперебойного питания), чтобы защитить оборудование от возможных скачков напряжения или перебоев в электроснабжении.

Желательно минимизировать внешние шумы, которые могут отвлекать пользователя персонального компьютера. Если помещение находится рядом с источниками шума, необходимо использовать звукоизоляционные материалы или наушники.

Искусственное освещение в помещениях для эксплуатации ПК должно осуществляться системой общего равномерного освещения. В качестве источников света при искусственном освещении следует преимущественно использовать люминесцентные лампы с рассеивателями и экранирующими решетками. Применение светильников без рассеивателей и экранирующих решеток не допускается. В светильниках местного освещения допускается использование ламп накаливания, в том числе галогенных. Для поддержания требуемых значений освещенности в помещениях для использования ПК

следует регулярно чистить стекла оконных рам и светильников не реже двух раз в год и своевременно заменять перегоревшие лампы.

Помещения, где располагаются рабочие места с ПК, должны быть оборудованы защитным заземлением. В этих помещениях следует ежедневно проводить влажную уборку и регулярно проветривать помещение после каждого часа работы на ПК. Также необходимо обеспечить наличие систем отопления, кондиционирования воздуха, чтобы обеспечить оптимальную температуру и влажность.

В помещении, где размещаются рабочие места с ПК, необходимо иметь аптечку первой медицинской помощи для оказания неотложной помощи при несчастных. Также, для обеспечения безопасности, необходимо иметь углекислотный огнетушитель, который может быть использован для тушения пожаров, а также пожарные извещатели.

2.2 Требования к организации рабочих мест

При размещении ПК в помещениях культурно-развлекательных учреждений с использованием жидкокристаллических или плазменных экранов площадь на одно рабочее место должна быть не менее 4,5 м².

Для обеспечения комфортного пространства между рабочими столами с мониторами расстояние между ними должно быть не менее 2 метров, а расстояние между боковыми поверхностями мониторов - не менее 1,2 метра.

Рабочие столы следует располагать таким образом, чтобы мониторы были ориентированы боковой стороной к окнам, таким образом естественный свет падает преимущественно слева.

Для выполнения творческой работы, требующей значительного умственного напряжения, рекомендуется изолировать рабочие места с компьютерами друг от друга перегородками высотой 1,5–2,0 метра.

Экран монитора должен находиться на расстоянии 600–700 мм от глаз пользователя, с учетом размеров алфавитно-цифровых знаков и символов, но

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		40

не ближе 500 мм. Клавиатура должна быть расположена на поверхности стола на расстоянии 100–300 мм от края.

Рабочий стол может иметь различные конструкции, отвечающие современным требованиям эргономики и позволяющие удобно разместить на рабочей поверхности необходимое оборудование. Желательно использование столов, которые имеют отдельную рабочую поверхность для размещения клавиатуры. Могут применяться столы с регулируемой и нерегулируемой высотой рабочей поверхности. Если стол не регулируется, его высота должна быть в пределах от 680 до 800 мм.

Для обеспечения физиологически правильной рабочей позы в течение рабочего дня рекомендуется использовать подъемно-поворотные рабочие стулья с регулируемыми по высоте и углам наклона сиденьями и спинками.

Рабочее место должно быть организовано с учетом принципов эргономики. Рабочий стол должен быть достаточно просторным для размещения на нём всей необходимой периферии для работы. Стул обязательно должен поддерживать правильную осанку, а периферия при использовании не должна вызывать напряжение или дискомфорт в теле.

Удобство рабочей позы работника в положении "сидя" достигается регулированием взаимного положения места для сидения и рабочей поверхности, в том числе ее высоты и размеров, а также высоты и угла наклона подставки для ног при ее применении.

При организации рабочего места должна быть обеспечена возможность смены рабочей позы занятыми на нем работниками.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано программное обеспечение для определения расстояния до объектов по изображениям. Помимо определения расстояния, с помощью предложенных инструментов можно определять объекты на фото и отрисовывать карту глубины.

Процесс разработки включал в себя анализ требований, анализ архитектуры приложения, подбор и обработку набора данных, обучение и настройку искусственных нейронных сетей.

Разработанное программное обеспечение представляет практическую ценность для пользователей, предоставляя им возможность автоматизированно определять объекты на фото, а также предсказывать расстояние до них.

В целом, выполнение данной работы позволило получить ценный опыт и глубже понять процессы, связанные с технологиями глубокого обучения и компьютерного зрения.

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		42

Список использованных источников

1. *rosuchebnik.*: [Электронный ресурс] URL: <https://rosuchebnik.ru/material/linzy-fokusnoe-rasstoyanie-linz-opticheskaya-sila-linz-formula-tonkoy-linzy-7200/> (Дата обращения: 19.05.2023г)
2. Марк Лутц. Изучаем *Python*. — СПб.: ООО “Диалектика”, 2019. — 832с.
3. Официальная документация *Python*.: [Электронный ресурс]. URL: <https://docs.python.org/3/> (Дата обращения: 23.05.2023г)
4. *Ultralytics.*: [Электронный ресурс] URL: <https://github.com/ultralytics/ultralytics> (Дата обращения: 25.05.2023г)
5. *Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, Vladlen Koltun. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer, – TPAMI, 2019.*
6. *Armin Masoumian, David G. F. Marei, Saddam Abdulwahab, Julian Cristiano, Domenec Puig, Hatem A. Rashwan: Absolute distance prediction based on deep learning object detection and monocular depth estimation models. – CCIA, 2021* δ

Приложение А

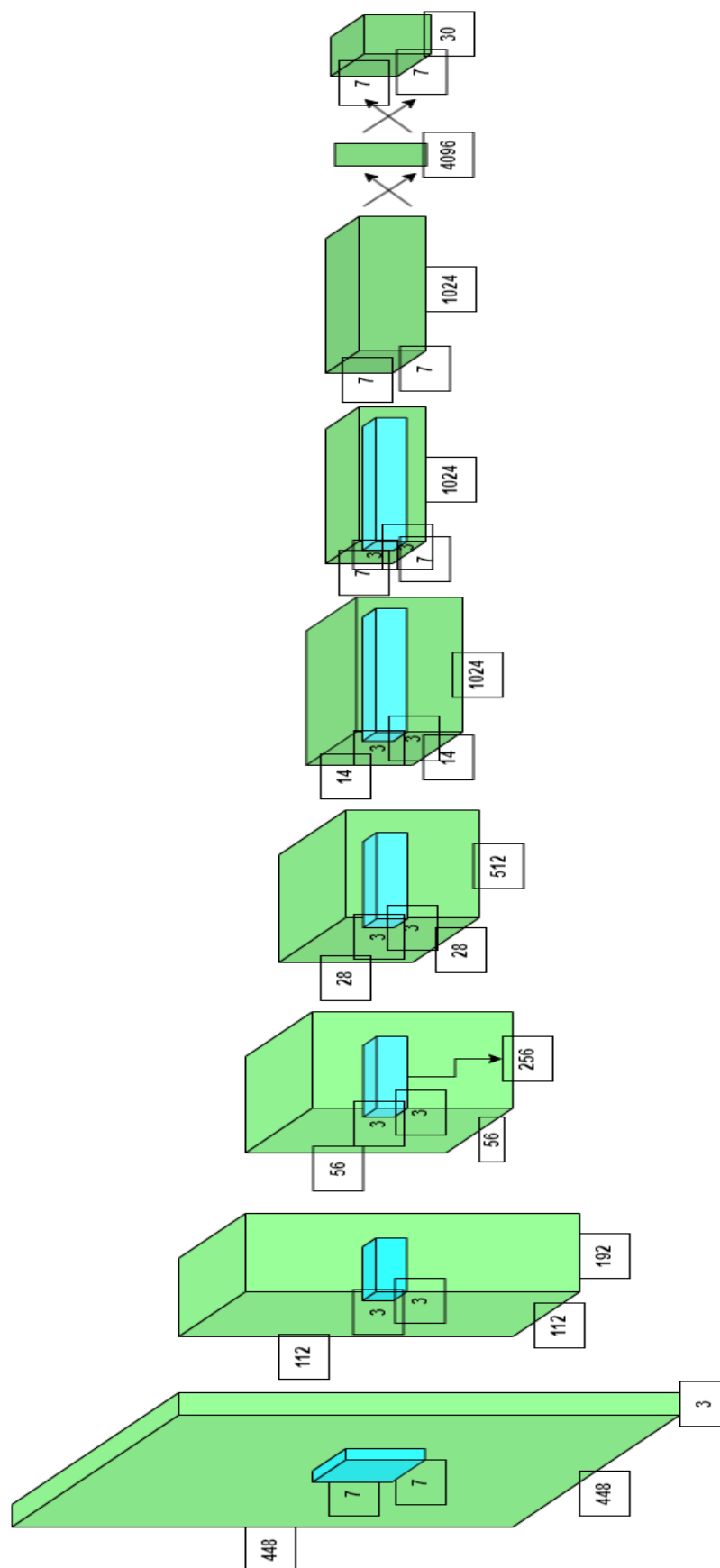
(обязательное)

Программа оценки расстояния до объектов по изображениям

Архитектуры нейронных сетей и UML диаграмма

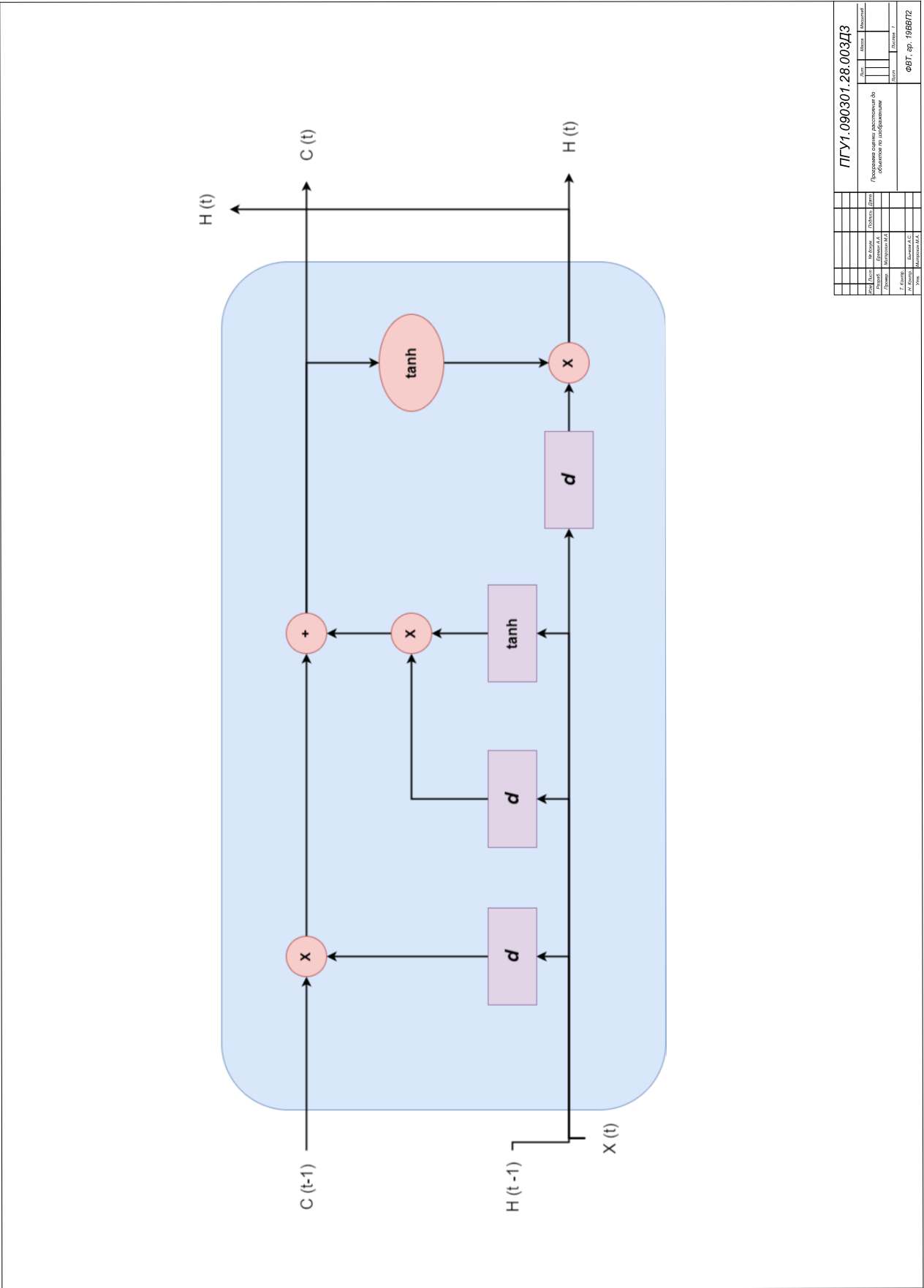
					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		44

Приложение А.1 – Архитектура детектора YOLO



ПГУ1.090301.28.003Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Приложение А.2 – Архитектура LSTM модели



Приложение А.4 – UML – диаграмма деятельности

ПГУ1.090301.28.003Д3			
Директор	Зам. директора	Менеджер	Исполнитель
Иванов И.И.	Петров П.П.	Сидоров С.С.	Кузнецов К.К.
Дата	Место	Подпись	Подпись
10.10.2023	г. Москва	Иванов И.И.	Петров П.П.
ФБТ, 09.1986П2			

Изм.	Лист	№ докум.	Подпись	Дат

Приложение Б

(обязательное)

Программа оценки расстояния до объектов по изображениям

Листинг файлов

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		50

Файл «DatasetHandler.py»

```
import os
import glob

DATASET_PATH = "./datasets/kitti/"
PATH_IMAGES = "./datasets/kitti/images/"
PATH_LABELS = "./datasets/kitti/labels/training/label_2/"
TRAIN_NAME_TXT = "train.txt"
TEST_NAME_TXT = "test.txt"

def TxtPatchCreator(path, path_save):
    path_train_images = path + "training/image_2/"
    path_test_images = path + "testing/image_2/"

    list_training = os.listdir(path_train_images)
    list_test = os.listdir(path_test_images)

    train_file_path = path_save + "train.txt"
    test_file_path = path_save + "test.txt"

    train_file = open(train_file_path, "w")
    test_file = open(test_file_path, "w")

    for name_img in list_training:
        train_file.write(os.path.join(os.path.abspath(path_train_images),
                                      name_img))
        train_file.write("\n")

    for name_img in list_test:
        test_file.write(os.path.join(os.path.abspath(path_test_images),
                                      name_img))
        test_file.write("\n")

    train_file.close()
    test_file.close()

TxtPatchCreator(PATH_IMAGES, DATASET_PATH)
```

Файл «Kitti_label.py»

```
from PIL import Image
import argparse
import os
import csv
```

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		51

```

OUT_LABELS_DIR = "labels"

KEY_PEDESTRIAN = "Pedestrian"
KEY_CYCLIST = "Cyclist"
KEY_CAR = "Car"
KEY_VAN = "Van"
KEY_MISC = "Misc"
KEY_TRUCK = "Truck"
KEY_PERSON_SITTING = "Person_sitting"
KEY_TRAM = "Tram"
KEY_DONT_CARE = "DontCare"

CLAZZ_NUMBERS = {
    KEY_PEDESTRIAN : 0,
    KEY_CYCLIST : 1,
    KEY_CAR : 2,
    KEY_DONT_CARE : 3
}

def getSampleId(path):
    basename = os.path.basename(path)
    return os.path.splitext(basename)[0]

def resolveClazzNumberOrNone(clazz, use_dont_care):
    if clazz == KEY_CYCLIST:
        return CLAZZ_NUMBERS[KEY_CYCLIST]
    if clazz == KEY_PEDESTRIAN:
        return CLAZZ_NUMBERS[KEY_PEDESTRIAN]
    if clazz == KEY_CAR:
        return CLAZZ_NUMBERS[KEY_CAR]
    if clazz == KEY_DONT_CARE:
        return CLAZZ_NUMBERS[KEY_DONT_CARE]
    return None

def convertToYoloBBox(bbox, size):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (bbox[0] + bbox[1]) / 2.0
    y = (bbox[2] + bbox[3]) / 2.0
    w = bbox[1] - bbox[0]
    h = bbox[3] - bbox[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def readRealImageSize(img_path):
    return Image.open(img_path).size

def readFixedImageSize():

```

```

        return (1242, 375)

def parseSample(lbl_path, img_path, use_dont_care):
    with open(lbl_path) as csv_file:
        reader = csv.DictReader(csv_file, fieldnames=["type",
            "truncated", "occluded", "alpha", "bbox2_left", "bbox2_top",
            "bbox2_right", "bbox2_bottom", "bbox3_height", "bbox3_width",
            "bbox3_length", "bbox3_x", "bbox3_y", "bbox3_z", "bbox3_yaw",
            "score"], delimiter=" ")
        yolo_labels = []
        for row in reader:
            clazz_number = resolveClazzNumberOrNone(row["type"],
            use_dont_care)
            if clazz_number is not None:
                size = readRealImageSize(img_path)
                bbox = (
                    float(row["bbox2_left"]),
                    float(row["bbox2_right"]),
                    float(row["bbox2_top"]),
                    float(row["bbox2_bottom"])
                )
                yolo_bbox = convertToYoloBBox(bbox, size)
                yolo_label = (clazz_number,) + yolo_bbox
                yolo_labels.append(yolo_label)
    return yolo_labels

def parseArguments():
    parser = argparse.ArgumentParser(description="Generates
    labels for training darknet on KITTI.")
    parser.add_argument("label_dir",
    help="data_object_label_2/training/label_2 directory; can be
    downloaded from KITTI.")
    parser.add_argument("image_2_dir",
    help="data_object_image_2/training/image_2 directory; can be
    downloaded from KITTI.")
    parser.add_argument("--training-samples", type=int,
    default=0.8, help="percentage of the samples to be used for
    training between 0.0 and 1.0.")
    parser.add_argument("--use-dont-care", action="store_false",
    help="do not ignore 'DontCare' labels.")
    args = parser.parse_args()
    if args.training_samples < 0 or args.training_samples > 1:
        print("Invalid argument {} for --training-samples.
        Expected a percentage value between 0.0 and 1.0.")
        exit(-1)
    return args

def main():
    args = parseArguments()

    if not os.path.exists(OUT_LABELS_DIR):
        os.makedirs(OUT_LABELS_DIR)

```

```

sample_img_paths = []
for dir_path, sub_dirs, files in os.walk(args.label_dir):
    for file_name in files:
        if file_name.endswith(".txt"):
            lbl_path = os.path.join(dir_path, file_name)
            sample_id = getSampleId(lbl_path)
            img_path = os.path.join(args.image_2_dir,
"{}.png".format(sample_id))
            sample_img_paths.append(img_path)
            yolo_labels = parseSample(lbl_path, img_path,
args.use_dont_care)
            with open(os.path.join(OUT_LABELS_DIR,
"{}.txt".format(sample_id)), "w") as yolo_label_file:
                for lbl in yolo_labels:
                    yolo_label_file.write("{} {} {} {}
{}\n".format(*lbl))

    print("Writing training and test sample ids...")
    first_test_sample_index = int(args.training_samples *
len(sample_img_paths))
    with open("kitti_train.txt", "w") as train_file:
        for sample_index in range(first_test_sample_index):
train_file.write("{}\n".format(sample_img_paths[sample_index]))
    with open("kitti_test.txt", "w") as test_file:
        for sample_index in range(first_test_sample_index,
len(sample_img_paths)):
test_file.write("{}\n".format(sample_img_paths[sample_index]))

if __name__ == "__main__":
    main()

```

Файл «KittiVisualizer.py»

```

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from skimage import io
from matplotlib.patches import Rectangle

colors = sns.color_palette('Paired', 9 * 2)
names = ['Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting',
'Cyclist', 'Tram', 'Misc', 'DontCare']
file_id = '000010'

if __name__ == '__main__':

    # load image

```

```

img =
np.array(io.imread(f'E:/diplom/datasets/kitti/images/000050.png')
, dtype=np.int32)

# load labels
with
open(f'E:/diplom/datasets/kitti/kitti_labels/training/label_2/000
050.txt', 'r') as f:
    labels = f.readlines()

fig = plt.figure()
# draw image
plt.imshow(img)

for line in labels:
    line = line.split()
    lab, _, _, _, x1, y1, x2, y2, _, _, _, _, _, _ = line
    x1, y1, x2, y2 = map(float, [x1, y1, x2, y2])
    if lab != 'DontCare':
        plt.gca().add_patch(Rectangle((x1, y1), x2 - x1, y2 - y1,
                                       linewidth=2,
                                       edgecolor='red',
                                       facecolor='none'))
        plt.text(x1 + 3, y1 + 3, lab,
                 bbox=dict(facecolor="blue", alpha=0.5),
                 fontsize=7, color='red')

plt.axis('off')
plt.tight_layout()
plt.show()

```

Файл «MidasRun.py»

```

import os
import glob
import torch
import utils
import cv2
import argparse
import time

import numpy as np

from imutils.video import VideoStream
from midas.model_loader import default_models, load_model

first_execution = True
def process(device, model, model_type, image, input_size,
target_size, optimize, use_camera):

    global first_execution

```

```

    if "openvino" in model_type:
        if first_execution or not use_camera:
            print(f"        Input resized to
{input_size[0]}x{input_size[1]} before entering the encoder")
            first_execution = False

        sample = [np.reshape(image, (1, 3, *input_size))]
        prediction = model(sample)[model.output(0)][0]
        prediction = cv2.resize(prediction, dsize=target_size,
                                interpolation=cv2.INTER_CUBIC)
    else:
        sample = torch.from_numpy(image).to(device).unsqueeze(0)

        if optimize and device == torch.device("cuda"):
            if first_execution:
                print("    Optimization to half-floats activated.
Use with caution, because models like Swin require\n"
                    "        float precision to work properly and may
yield non-finite depth values to some extent for\n"
                    "        half-floats.")
                sample = sample.to(memory_format=torch.channels_last)
                sample = sample.half()

            if first_execution or not use_camera:
                height, width = sample.shape[2:]
                print(f"        Input resized to {width}x{height} before
entering the encoder")
                first_execution = False

            prediction = model.forward(sample)
            prediction = (
                torch.nn.functional.interpolate(
                    prediction.unsqueeze(1),
                    size=target_size[::-1],
                    mode="bicubic",
                    align_corners=False,
                )
                .squeeze()
                .cpu()
                .numpy()
            )

        return prediction

def create_side_by_side(image, depth, grayscale):

    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    print("Device: %s" % device)

```

```

    model, transform, net_w, net_h = load_model(device,
model_path, model_type, optimize, height, square)

    if input_path is not None:
        image_names = glob.glob(os.path.join(input_path, "*"))
        num_images = len(image_names)
    else:
        print("No input path specified. Grabbing images from
camera.")

    if output_path is not None:
        os.makedirs(output_path, exist_ok=True)

    print("Start processing")

    if input_path is not None:
        if output_path is None:
            print("Warning: No output path specified. Images will
be processed but not shown or stored anywhere.")
            for index, image_name in enumerate(image_names):

                print("  Processing {} ({} / {})".format(image_name,
index + 1, num_images))

                # input
                original_image_rgb = utils.read_image(image_name)  #
in [0, 1]
                image = transform({"image":
original_image_rgb})["image"]

                # compute
                with torch.no_grad():
                    prediction = process(device, model, model_type,
image, (net_w, net_h), original_image_rgb.shape[1::-1],
optimize, False)

                # output
                if output_path is not None:
                    filename = os.path.join(
                        output_path,
os.path.splitext(os.path.basename(image_name))[0] + '-' +
model_type
                    )
                    if not side:
                        utils.write_depth(filename, prediction,
grayscale, bits=2)
                    else:
                        original_image_bgr =
np.flip(original_image_rgb, 2)
                        content =
create_side_by_side(original_image_bgr*255, prediction,
grayscale)

```

Изм.	Лист	№ докум.	Подпись	Дат

ПГУ1.090301.28.001 ПЗ

Лист

57

```

        cv2.imwrite(filename + ".png", content)
        utils.write_pfm(filename + ".pfm",
prediction.astype(np.float32))

    else:
        with torch.no_grad():
            fps = 1
            video = VideoStream(0).start()
            time_start = time.time()
            frame_index = 0
            while True:
                frame = video.read()
                if frame is not None:
                    original_image_rgb = np.flip(frame, 2) # in
[0, 255] (flip required to get RGB)
                    image = transform({"image":
original_image_rgb/255})["image"]

                    prediction = process(device, model,
model_type, image, (net_w, net_h),

original_image_rgb.shape[1::-1], optimize, True)

                    original_image_bgr =
np.flip(original_image_rgb, 2) if side else None
                    content =
create_side_by_side(original_image_bgr, prediction, grayscale)
                    cv2.imshow('MiDaS Depth Estimation - Press
Escape to close window ', content/255)

                    if output_path is not None:
                        filename = os.path.join(output_path,
'Camera' + '-' + model_type + '_' + str(frame_index))
                        cv2.imwrite(filename + ".png", content)

                    alpha = 0.1
                    if time.time()-time_start > 0:
                        fps = (1 - alpha) * fps + alpha * 1 /
(time.time()-time_start) # exponential moving average
                        time_start = time.time()
                        print(f"\rFPS: {round(fps,2)}", end="")

                    if cv2.waitKey(1) == 27: # Escape key
                        break

                    frame_index += 1

            print()

            print("Finished")

if __name__ == "__main__":

```

```

parser = argparse.ArgumentParser()

parser.add_argument('-i', '--input_path',
                    default=None,
                    help='Folder with input images (if no
input path is specified, images are tried to be grabbed '
                        'from camera)'
                    )

parser.add_argument('-o', '--output_path',
                    default=None,
                    help='Folder for output images'
                    )

parser.add_argument('-m', '--model_weights',
                    default=None,
                    help='Path to the trained weights of
model'
                    )

parser.add_argument('-t', '--model_type',
                    default='dpt_swin2_large_384',
                    help='Model type: '
                        'dpt_beit_large_512,
dpt_beit_large_384, dpt_beit_base_384, dpt_swin2_large_384, '
                        'dpt_swin2_base_384,
dpt_swin2_tiny_256, dpt_swin_large_384, dpt_next_vit_large_384, '
                        'dpt_levit_224, dpt_large_384,
dpt_hybrid_384, midas_v21_384, midas_v21_small_256 or '
                        'openvino_midas_v21_small_256'
                    )

parser.add_argument('-s', '--side',
                    action='store_true',
                    help='Output images contain RGB and depth
images side by side'
                    )

parser.add_argument('--optimize', dest='optimize',
                    action='store_true', help='Use half-float optimization')
parser.set_defaults(optimize=False)

parser.add_argument('--height',
                    type=int, default=None,
                    help='Preferred height of images feed
into the encoder during inference. Note that the '
                        'preferred height may differ from
the actual height, because an alignment to multiples of '
                        '32 takes place. Many models support
only the height chosen during training, which is '
                        'used automatically if this
parameter is not set.'
                    )

```

Изм.	Лист	№ докум.	Подпись	Дат

```

        )
        parser.add_argument('--square',
                              action='store_true',
                              help='Option to resize images to a square
resolution by changing their widths when images are '
                              'fed into the encoder during
inference. If this parameter is not set, the aspect ratio of '
                              'images is tried to be preserved if
supported by the model.'
        )
        parser.add_argument('--grayscale',
                              action='store_true',
                              help='Use a grayscale colormap instead of
the inferno one. Although the inferno colormap, '
                              'which is used by default, is better
for visibility, it does not allow storing 16-bit '
                              'depth values in PNGs but only 8-bit
ones due to the precision limitation of this '
                              'colormap.'
        )

    args = parser.parse_args()

    if args.model_weights is None:
        args.model_weights = default_models[args.model_type]

    # set torch options
    torch.backends.cudnn.enabled = True
    torch.backends.cudnn.benchmark = True

    # compute depth maps
    run(args.input_path, args.output_path, args.model_weights,
        args.model_type, args.optimize, args.side, args.height,
        args.square, args.grayscale)

```

Файл «YoloRun.py»

```

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

from ultralytics import YOLO

model = YOLO('yolov8m.pt')

result =
model.predict(source="E:/diplom/datasets/kitti/images/000050.png"
)

import cv2
import random

```

```

names = model.names
img = cv2.imread("E:/diplom/midas/output/000050-
dpt_beit_large_512.png")

tl = round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1
for r in result:
    boxes = r.bboxes
    for box in boxes:
        b = box.xyxy[0]
        c = box.cls
        x1, x2 = (int(b[0]), int(b[1])), (int(b[2]), int(b[3]))
        color = (0, 0, 255)
        cv2.rectangle(img, x1, x2, color, tl, cv2.LINE_AA)
        cv2.putText(img, names[int(c)], (x1[0], x1[1] - 2), 0, tl
/ 3, [225, 255, 255], thickness=tl, lineType=cv2.LINE_AA)

cv2.imwrite("output.png", img)

```

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		61

Приложение В

(обязательное)

Программа оценки расстояния до объектов по изображениям

Презентация

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		62

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

Программа оценки расстояния до объектов по изображениям

Автор работы: Еремин А.А

Руководитель ВКР: Митрохин М.А., д.т.н., зав. кафедрой ИВТ

1

Цель работы и сформированные задачи

Целью работы является разработка программы позволяющая оценить расстояние до определенных объектов по изображениям. В список детектируемых объектов входят:

- машины;
- велосипедисты;
- пешеходы.

Для реализации данной цели были поставлены следующие задачи:

- 1) поиск и подбор набора данных;
- 2) обработка набора данных;
- 3) преобразование аннотаций;
- 4) обучение и настройка нейронных сетей;

2

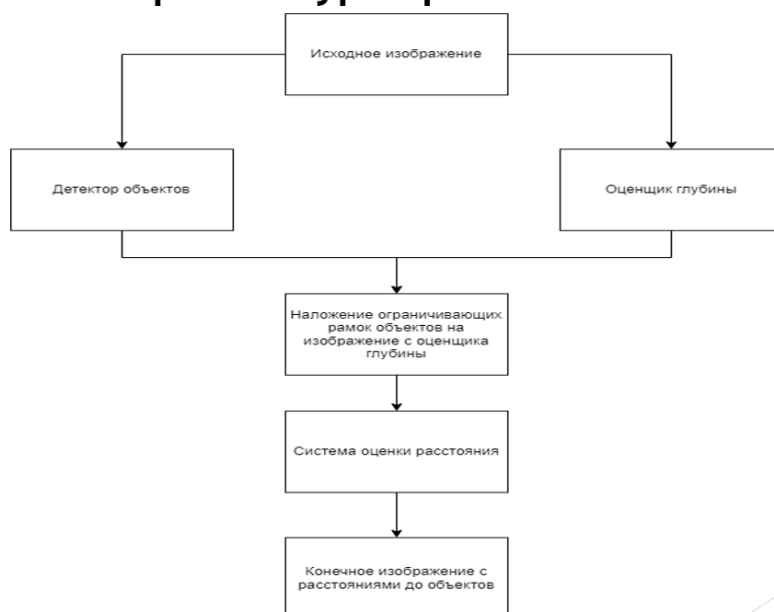
Изм.	Лист	№ докум.	Подпись	Дат

ПГУ1.090301.28.001 ПЗ

Лист

63

Архитектура приложения



Выбор инструментов и технологий



Изм.	Лист	№ докум.	Подпись	Дат

ПГУ1.090301.28.001 ПЗ

Лист

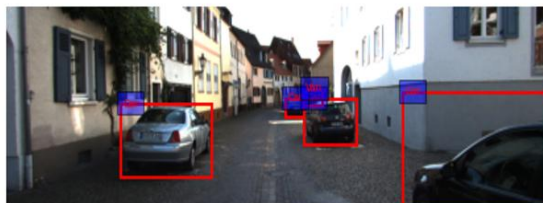
64

Детектирование объектов на изображении

Изначальное изображение:



Размеченное изображение:



5

Примеры обнаружения объектов на изображении с помощью детектора



6

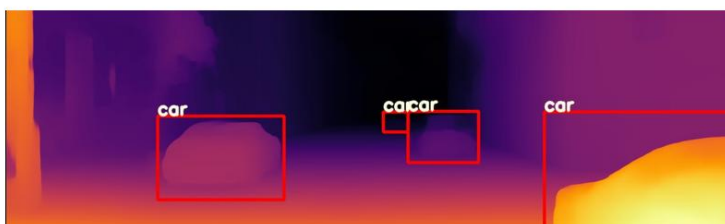
Изм.	Лист	№ докум.	Подпись	Дат

ПГУ1.090301.28.001 ПЗ

Лист

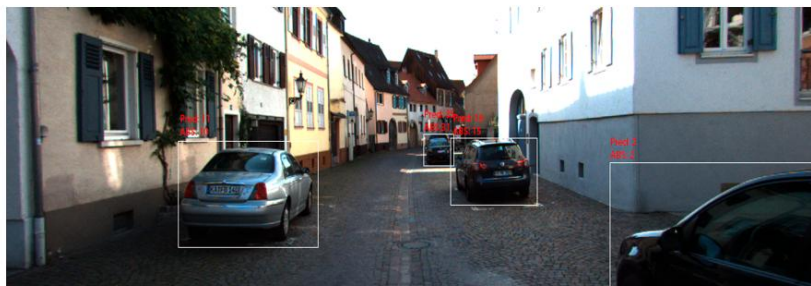
65

Получение карты глубины



7

Оценка расстояния до объектов



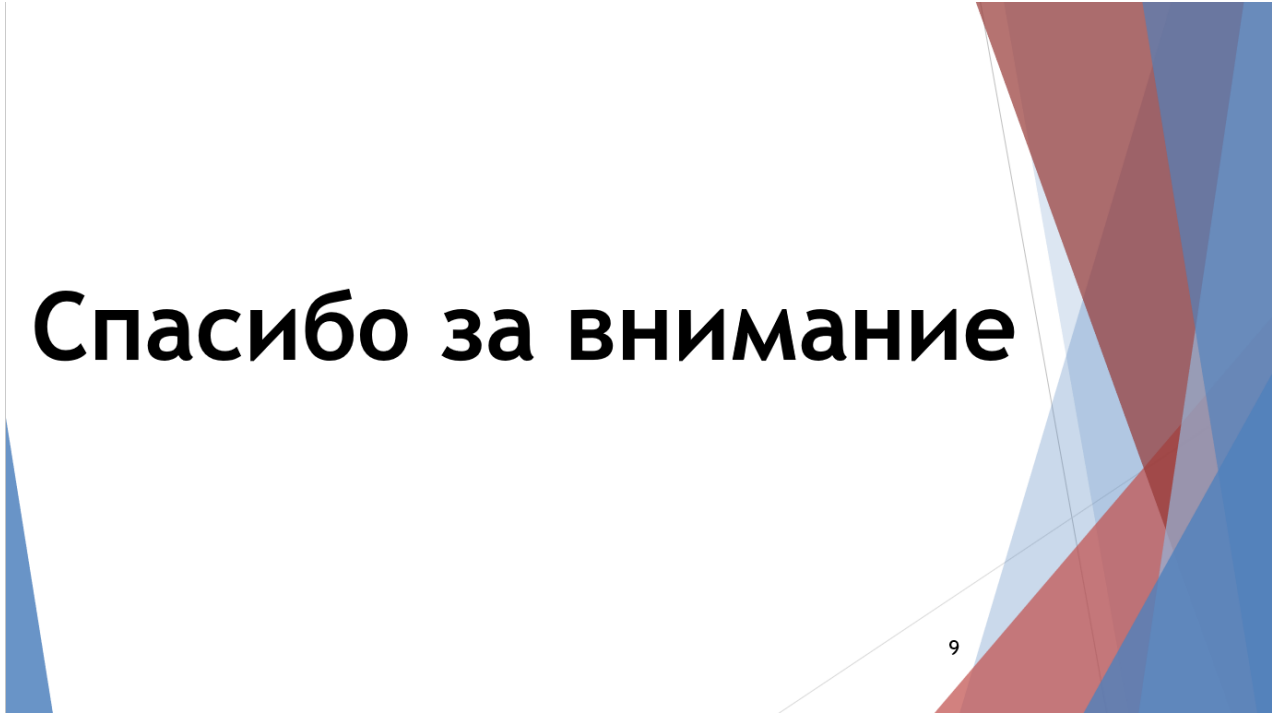
8

Изм.	Лист	№ докум.	Подпись	Дат

ПГУ1.090301.28.001 ПЗ

Лист

66



Спасибо за внимание

9

					ПГУ1.090301.28.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		67