Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Интеллектуальная система мониторинга и распознавания загрязнений водоемов

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР		А. В. Конев
	(подпись, дата)	(инициалы, фамилия)

Группа ПО-11б

Руководитель ВКР		Р. А. Томакова
	(подпись, дата)	(инициалы, фамилия)
Нормоконтроль		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой		А. В. Малышев
	(подпись, дата)	(инициалы, фамилия)

Курск 2025 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Конева А. В., шифр 21-06-0146, группа ПО-11б

1. Тема «Интеллектуальная система мониторинга и распознавания загрязнений водоемов» утверждена приказом ректора ЮЗГУ от «04» апреля 2025 г. № 1696-с.

2. Срок предоставления работы к защите «9» июня 2025 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) проанализировать предметную область;

2) разработать архитектуру нейронной сети;

3) сформировать датасет и обучить нейронную сеть;

4) спроектировать настольное приложение для взаимодействия с системой;

5) сконструировать и протестировать настольное приложение.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: датасет из изображений, изображения для анализа, файл весов нейронной сети.

2) Выходными данными для программной системы являются: проанализированное изображение, файлы весов нейронной сети после обучения.

4. Содержание работы (по разделам):

4.1. Введение.

4.1. Анализ предметной области.

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к данным программной системы, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, описание используемых библиотек, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение.

4.6. Список использованных источников.

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ.

Лист 2. Цель и задачи разработки.

Лист 3. Концептуальная модель сайта.

Лист 4. Еще плакат.

Руководитель ВКР		Р. А. Томакова
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению		А. В. Конев
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 91 странице. Работа содержит 34 иллюстрации, 10 таблиц, 20 библиографических источников и 4 листа графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: разливы нефти, нейронные сети, мониторинг, спутниковые снимки, распознавание объектов, информационная система, экология, загрязнение водоемов, компьютерное зрение, U-Net, интеллектуальная система, машинное обучение, обработка изображений, предобработка данных, пользовательский интерфейс.

Объектом разработки является система мониторинга водоемов для выявления загрязнений на основе анализа изображений поверхности воды.

Целью выпускной квалификационной работы является разработка интеллектуальной системы для распознавания характерных пятен нефтяных разливов на поверхности водоемов.

В процессе создания системы была разработана архитектура нейронной сети, приложение с графическим интерфейсом для взаимодействия пользователей с системой, сформирован датасет, обучена и протестирована разработанная нейронная сеть.

ABSTRACT

The volume of work is 91 page. The work contains 34 illustrations, 10 tables, 20 bibliographic sources and 4 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. The layout of the site, including the connection of components, is presented in annex B.

List of keywords: oil spills, neural networks, monitoring, satellite imagery, object recognition, information system, ecology, water pollution, computer vision, U-Net, intelligent system, machine learning, image processing, data preprocessing, user interface.

The object of the research is the waterbody monitoring system for detectin pollution based on water surface imagery analysis.

The goal of this work is to develop an intelligent system for detecting characteristic oil spill patches on the surface of waterbodies.

During the development process, a neural network architecture was designed, a graphical user interface desktop application was developed for user interaction with the system, a dataset was prepared and the developed neural network architecture was trained and tested.

СОДЕРЖАНИЕ

Сведения о ВКРБ (Графический материал / Сведения о ВКРБ.png) Лист 1

Цель и задачи разработки (Графический материал / Цель и задачи разработки.png) Лист 2

Концептуальная модель сайта (Графический материал / Концептуальная модель сайта.png) Лист 3

Еще плакат (Графический материал / Еще плакат.png) Лист 4

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС – интеллектуальная система.

ИТ – информационные технологии.

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

НС – нейронная сеть.

СНС – сверточная нейронная сеть.

U-Net – архитектура нейронной сети для сегментации изображений.

CUDA – технология параллельных вычислений NVIDIA.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

ВВЕДЕНИЕ

Вода является одним из важнейших природных ресурсов, как для поддержания жизнедеятельности биологических систем, так и для обеспечения разнообразных промышленно-технологических процессов. В настоящее время водные ресурсы активно используются в различных ключевых отраслях промышленности и жизнедеятельности человека. В металлургии вода применяется в процессах флотации - обогащения руд полиметаллов, а также для охлаждения доменных печей. Химическая промышленность активно использует воду для очищения и охлаждения оборудования на различных нефтеперерабатывающих производствах, как компонент для создания продуктов нефтехимической отрасли, таких как пластмассы, реагенты и растворители, удобрения. Вода также активно применяется как один из материалов для создания фармацевтической продукции, такой как инъекционные растворы, препараты для наружного применения(спреи и капли). Энергетический сектор также использует воду для охлаждения энергетического оборудования на атомных и теплоэлектростанциях. Кроме того, вода так же обеспечивает работу гидроэлектростанций.

Загрязнение водоемов негативно сказывается на экологии, здоровье населения, препятствует развитию экономики, поэтому обеспечение экологической безопасности водоемов является одной из наиболее важных задач экологии. В настоящее время особую актуальность приобретает проблема нефтяных разливов.

По этим и другим причинам в настоящее время остро стоит вопрос сохранения чистоты водоемов. Одним из способов достижения этой цели является распознавание загрязнений на поверхности воды.

Цель настоящей работы – разработка интеллектуальной системы для распознавания характерных пятен нефтяных разливов на поверхности водоемов. Для достижения поставленной цели необходимо решить *следующие задачи:*

– провести анализ предметной области;

– сформирофать датасет обучающих изображений поверхности воды;
– разработать архитектуру нейронной сети;
– обучить нейронную сеть;
– провести оценку достоверности полученных результатов;
– спроектировать настольное приложение для анализа изображений;
– реализовать приложение, используя графический интерфейс.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы изложен на 91 странице.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии анализа предметной области рассматриваются экологические аспекты загрязнения водоемов, причины и последствия разливов нефти, а также методы их распознавания.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемой интеллектуальной системе.

В третьем разделе на стадии технического проектирования представлены проектные решения для системы.

В четвертом разделе приводится список классов и их методов, использованных при разработке системы, производится тестирование разработанного настольного приложения.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 Нефтяные разливы

1.1.1 Экологическая опасность нефтяных разливов

Разлив нефти – аварийный выброс нефти или нефтепродуктов в окружающую среду. Нефтяные разливы происходят с высокой частотой и являются серьезной проблемой нефтегазовой отрасли. Каждый разлив приводит к масштабным последствиям для окружающей среды и требует значительных финансовых затрат для устранения самого разлива и минимизации экологического ущерба.

Ярким примером является инцидент на буровой платформе Deepwater Horizon, принадлежавшей британской нефтегазовой компании British Petrolium. 20 апреля 2010 года, в результате нарушения технологических норм цементирования скважины, произошел взрыв, который привел к гибели 11 человек и неконтролируемому выбросу нефти в Мексиканский залив на протяжении нескольких месяцев. В окружающую среду попало около 400 тысяч тонн нефти. Эта катастрофа, ставшая крупнейшим в истории США разливом нефти, привела к массовой гибели животных, таких как дельфины, морские черепахи, тем самым нанеся колоссальный ущерб морским экосистемам, по некоторым оценкам сохраняющийся и по сей день. Расходы на ликвидацию происшествия, а также затраты на штрафы и компенсации, составили около 65 миллиардов долларов.

16 марта 1978 года французский супертанкер Amoco Cadiz, попав в шторм, потерял управление и разбился о скалы у берегов французского региона Бретань. Это происшествие привело к выбросу 220 тысяч тонн сырой нефти в Атлантический океан, загрязнив более 300 километров французского побережья. Пролившаяся нефть образовала пятно протяженностью 19 километров и поникла в песок на глубину до 50 сантиметров, образовав асфальтоподобные корки, сохранявшиеся на протяжении нескольких лет. В защищенных от волн районах нефть сохранялась до 10 лет, тем самым замедляя

восстановление экосистемы, понесшей масштабный ущерб – загрязнение вызвало гибель около 20 тысяч птиц, множества донных организмов, таких как моллюски и ракообразные. Рыбаки сообщали о сокращении уловов в два раза, а сама рыба нередко имела различные язвы и опухоли. Компания Amoco, владелец танкера, выплатила Франции компенсацию в размере 230 миллионов долларов.

Не менее разрушительными оказались последствия крушения танкера Prestige, затонувшего в ноябре 2002 года. Получив повреждения у побережья Испании, корпус судна раскололся, что привело к попаданию примерно 60 тысяч тонн нефти в Атлантический океан. Растянувшееся на более чем 2000 километров вдоль побережья Галисии нефтяное пятно поставило под угрозу существование 25 охраняемых видов животных, уничтожило популяции рыб, кораллов, моллюсков. Согласно государственной оценке, зачистка побережья Галисии обошлась в 2,5 миллиарда евро, а нанесенный ущерб составил 368 миллионов евро.

В июле 2010 года нефтепровод, принадлежащий американской корпорации Enbridge, разорвало в результате коррозии трубы, что послужило причиной попадания более 2,7 тонн битумной нефти в реку Каламазу, пролившейся на 40 километров по руслу реки и осевшей на дне. Осевшая нефть отравила донные отложения реки, что привело к массовой гибели рыб, популяции которых начали возвращаться только спустя 10 лет. Кроме того, токсины сделали речную воду непригодной для питья, а из-за ядовитых испарений нефти некоторые жители близлежащих районов жаловались на головные боли и тошноту. Поскольку убирать пролившуюся нефть пришлось не только с поверхности, но и со дна реки, очистные мероприятия были значительно затруднены, что повысило стоимость их проведения – она составила примерно 1,2 миллиарда долларов.

Утром 15 декабря 2024 года в Керченском проливе в результате шторма танкеры «Волгонефть-212» и «Волгонефть-239», перевозившие в общей сложности около 9000 тонн мазута, потерпели крушение. По различным оценкам в море попало от 2,5 до 5 тысяч тонн мазута, разлившегося по пло-

щади более 400 м^2. Загрязнение затронуло около 70 километров береговой линии и привело к гибели более 200 птиц и 70 дельфинов, среди которых были охраняемые виды. Реагируя на происшествие, власти вели режим чрезвычайной ситуации на федеральном уровне и объявили о начале масштабных очистных работ с участием 300 человек и 60 единиц специальной техники, продолжающихся по сей день.

Вследствие принятия множества международных законов и соглашений, совершенствования технологических процессов можно отметить уменьшение частоты крупных происшествий, приводящих к масштабным разливам нефти. Однако, некоторые исследования[1] отмечают, что информация о разливах нефти не всегда достоверна, утверждая, что многие менее крупные инциденты никогда не попадают в поле зрения государственных служб контроля и СМИ, что приводит к отсутствию данных об этих инцидентах в международных и государственных базах.

1.1.2 Причины и источники загрязнений

Основными источниками разливов нефти являются различные утечки на этапах добычи, хранения и транспортировки нефти и нефтепродуктов. Причинами возникновения утечек могут являться как стихийные бедствия, такие как штормы, землетрясения и эрозия, так и техногенные факторы, включая отказы оборудования, человеческие ошибки. В таблице 1.1 указаны основные источники попадания нефти в мировой океан[2].

Таблица 1.1 – Основные источники попадания нефти в мировой океан

Источник	млн. т/год
Морская транспортировка (кроме аварийных разливов)	1,83
Аварийные разливы	0,3
Речной сток, включая сточные воды городов	1,9
Сточные воды прибрежной зоны	0,8

Продолжение таблицы 1.1

Источник	млн. т/год
Естественные нефтяные скважины	0,6
Добыча нефти в море	0,08
Всего	**5,51**

Наиболее часто разливы нефти появляются в результате аварии на танкере, перевозящем нефть или нефтепродукты. Среди причин возникновения таких аварий можно выделить столкновения, посадки на мель, штормовые повреждения, нарушение герметичности резервуаров. Эти происшествия нередко приводят к масштабным выбросам нефти в морскую или речную среду, при этом загрязняя не только воду, но и зачастую прибрежные зоны, в районе которых произошла авария.

Морская добыча нефти проводится с помощью масштабных комплексов, использующих комплекс инженерно-технических решений. Такие происшествия, как повреждения герметизации скважин, отказ оборудования, ошибки при цементировании или нарушения технологических регламентов приводят к выбросам добываемой нефти из подземных резервуаров, что в свою очередь приводит к разливам, плохо поддающимся локализации.

Трубопроводы, используемые для транспортировки нефти на большие расстояния, могут дать течь вследствие коррозии труб, механических повреждений, ошибок в обслуживании. В результате в воде может образоваться нефтяной разлив, который может долго оставаться незамеченным, если трубопровод проходит под водой. Кроме того, серьезную угрозу загрязнения представляет преднамеренное разрушение нефтяной инфраструктуры, особенно в зонах активных боевых действий. Атаки на эти объекты не только приводят к масштабным утечкам, но и затрудняют обнаружение и ликвидацию последствий разливов.

Промышленные предприятия, связанные с переработкой нефти, могут сбрасывать загрязненные нефтью или продуктами ее переработки воды в

близлежащие водоемы. Наряду с этим отходы ремонтных мастерских, автозаправочных станций также могут попадать в водоемы, что приводит к еще большему загрязнению.

Нефтяные утечки происходят не только по вине человека, но и в ходе естественного просачивания из нефтеносных пластов земли через трещины в коре. Чаще всего это происходит в районах активной геологической активности. Несмотря на относительно незначительные объемы, эти утечки также оказывают негативное влияние на экосистемы.

1.2 Методы мониторинга разливов нефти

Своевременное обнаружение и ликвидация нефтяных разливов – ключевые задачи для сохранения экологической безопасности водоемов. В связи с этим широко распространены различные виды мониторинга состояния водоемов. среди которых выделяются визуальные методы наблюдения за поверхностью воды. В отличие от лабораторных исследований, визуальное исследование позволяет намного быстрее охватить большие территории и не требует физического контакта с объектом, что позволяет оперативно обнаружить новые разливы нефти, особенно в труднодоступных акваториях.

Одним из самых распространенных методов визуального мониторинга является спутниковая съемка. Современные спутники, оснащенные высокоточными сенсорами, позволяют получить качественные снимки поверхности воды и хорошо отображают аномалии, такие как нефтяные пятна. Спутниковая съемка охватывает сотни квадратных километров поверхности Земли в одном снимке, а использование групп спутников позволяет получать новые данные с высокой периодичностью. Кроме того, использование многих спутников бесплатно, а данные размещены в открытом доступе, что снижает стоимость процедур мониторинга. Однако, зависимость космической съемки от погодных условий и времени суток значительно осложняет наблюдение за поверхностью водоемов и является существенным недостатком этого метода.

В условиях плохой видимости, ограничивающих работу спутников, применяются радиолокационные спутниковые снимки. В отличие от оптиче-

ских сенсоров, радары полагаются на электромагнитные сигналы и фиксирую отраженный поверхностью воды отклик, что и позволяет вести съемку в любую погоду и время суток. На радарных снимках нефтяные пятна обычно проявляются в качестве темных областей, что повышает их информативность в задачах обнаружения нефтяных пятен и пленок. Так, исследование радиолокационных снимков Керченского пролива с использованием спутников Sentinel-1A/B за период с 2017 по 2021 годы показало высокую точность этого метода и позволило исследовать частоту появления, масштаб нефтяных разливов и их источники на основании 2597 пятен, появившихся на снимках за этот период[3]. Несмотря на гибкость этого метода, полученные снимки нередко проявляют сторонние объекты, например различные биологические образования, создавая визуально похожие участки на изображениях. Наличие таких ложных данных требует участия эксперта, отличающего действительные нефтяные пятна от схожих объектов, в процессе мониторинга. Кроме того, радиолокационные изображения значительно дороже, что еще больше повышает стоимость мониторинга.

Наконец, высокую эффективность показал метод аэросъемки поверхностей воды. Вместе с развитием беспилотных летательных аппаратов, или БПЛА, наблюдение за водоемами с малой высоты стало высокоэффективным и экономически оправданным методом мониторинга. БПЛА часто оснащаются камерами высокого разрешения, что позволяет получать самые качественные снимки из перечисленных методов, а высокая степень гибкости в построении маршрутов и простота управления делают этот метод самым доступным. Недостатками аэросъемки с помощью БПЛА являются ограниченность радиуса и времени полета, что делает его малоэффективным для мониторинга больших поверхностей воды, таких как моря. Беспилотные аппараты часто используются в составе систем мониторинга вместе с спутниковыми системами для уточняющих съемок в определенных секторах, снимки которых получены из космоса.

Перечисленные выше дистанционные методы используются для получения визуальных данных, являющихся источником информации для обна-

ружения нефтяных разливов. Чтобы обнаружить загрязнение, необходимо проанализировать полученные данные. Учитывая высокое количество изображений, полученных для анализа, этот процесс занимает значительное время. Более того, если изображения получаются в режиме реального времени, например в формате видеосъемки, то для обнаружения разлива нефти необходимо постоянное присутствие наблюдателя. В настоящее время для задач анализа изображений все чаще используются нейронные сети.

1.3 Нейронные сети

1.3.1 История появления нейронных сетей

Идеи, которые легли в основу современных нейронных сетей, берут свое начало в философии и математике, а основные принципы исследовались задолго до появления вычислительных систем как таковых. Правила, руководящие рациональной частью мышления были сформированы Аристотелем в далеких 384 – 322 годах до нашей эры. Впоследствии формальные правила рассуждений и процессы мышления стали предметом исследований не менее известных философов, среди которых были Томас Гоббс (1588-1679), Рене Декарт (1596-1650) и Рудольф Карнап (1891-1970)[4]. Известные математики, такие как Джордж Буль (1815-1864) и Готтлоб Фреге (1848-1925), внесли вклад в разработку логических систем, впоследствии ставших источниками исследований моделирования искусственного интеллекта.

Первая модель искусственного нейрона была предложена Уорреном МакКаллоком и Уолтером Питтсом в 1943 году. Модель представленного ими логического устройства показана на рисунке 1.1.

Модель состоит из входов x_1, x_2, ... , x_n, весов w_1, w_2, ... , w_n, сумматора и функции активации. Попадая на входы, каждое значение x_i умножается на значение соответствующего веса w_i, обозначающего его важность, после чего поступает на сумматор, вычисляющий сумму всех поступивших на него

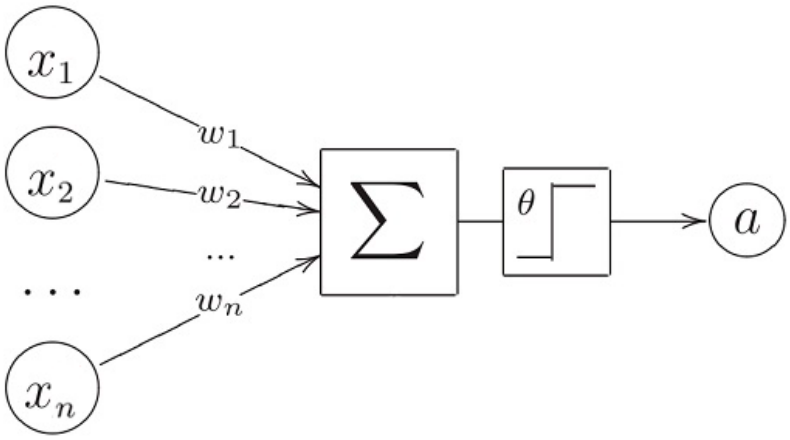

Рисунок 1.1 – Искусственный нейрон

сигналов:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Полученная сумма затем сравнивается с пороговым значением θ. Если сумма больше, то нейрон возбуждается (выходное значение a = 1), в противном случае остается в состоянии покоя (выходное значение a = 0). Данная модель может выполнять только бинарные операции, получая на вход значения 0 и 1, требует ручного задания весов, также являющихся бинарными, и порогового значения, что значительно ограничивает её возможности.

Несмотря на ограничения, предложенный МакКаллоком и Питтсем искусственный нейрон показал способность выполнять базовые логические операции, такие как И, ИЛИ и НЕ. Эта модель стала основой для дальнейшей теории нейронных вычислений, а все современные нейронные сети состоят из доработанных версий оригинальных нейронов.

Спустя несколько лет идеи, взяв за основу исследования МакКаллока и Питтса, американский ученый Фрэнк Розенблатт предложил схему перцептрона - устройства, моделирующего восприятие информации головным мозгом человека. Перцептрон состоит из входных датчиков, ассоциативных и реагирующих элементов[5]. Схема простейшего перцептрона показана на рисунке 1.2.

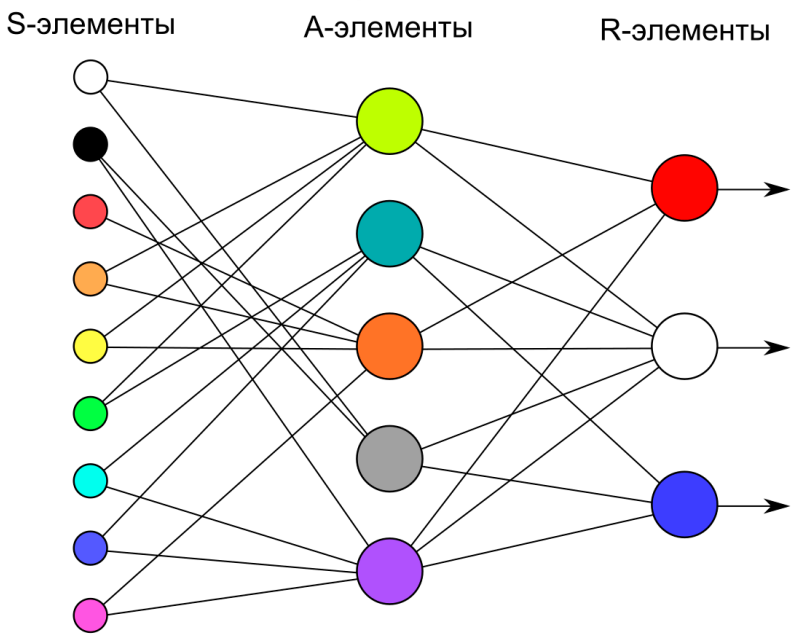

Рисунок 1.2 – Схема однослойного перцептрона

Однослойный перцептрон подходит для решения задачи классификации данных. Для обучения задаются не только входные значения, но и известные ожидаемые результаты. Как и в случае работы нейрона МакКаллока - Питтса, входные значения (S-слой) передаются на ассоциативные элементы (A-слой), где вычисляется взвешенная сумма поступивших сигналов, являющихся произведениями входных значений и весов связей и, наконец, если сумма превышает заданный порог θ, то результат подается на реагирующий элемент (R-слой). Если выходное значение не совпадает с ожидаемым, то веса связей S–A модифицируются по формуле:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \cdot \delta \cdot X_j,$$

где t и t_1 – номера текущей и следующей итераций процесса, η – коэффициент обучения, находящийся в диапазоне $0<\eta<1$, δ – разность между ожидаемым и полученным значением, i и j – номера входа и нейрона в слое соответственно, после чего перцептрон заново обрабатывает входные значения до тех пор, пока все выходные значения не совпадут с ожидаемыми.

Однако, несмотря на успехи в исследованиях, дальнейшее изучение нейронных сетей столкнулось с скептицизмом. Основной причиной стала невозможность однослойных сетей решать многие простые задачи, среди которых операция «исключающего ИЛИ». Формальное доказательство невозможности решения нелинейных задач однослойным перцептроном было предоставлено Марвином Минским и Сеймуром Папертом в книге «Перцептроны», выпущенной в 1969 году.

После выхода книги, исследовательский интерес к области нейронных систем значительно упал, но не исчез полностью. В 1970-х активно проводились исследования многослойных перцептронов, был разработан алгоритм обратного распространения ошибки для их обучения, позволяющий преодолеть многие ограничения, обозначенные в работе «Перцептроны», но впоследствии оказавшийся не универсальным. Также разрабатывались альтернативные модели нейронных сетей, такие как когнитрон, предложенная Кунихико Фукусимой в 1975 году, ставшая первой моделью, реализующей обучение без учителя. Основной идеей когнитрона является чередование возбуждающих и подавляющих слоев, что позволяет извлекать отдельные признаки и обобщать изображения. Обучение без учителя основано на конкуренции между нейронами – чем сильнее конкретный вход, тем больший вес ему присваивается, что приводит к заглушению соседних нейронов в конкретном слое. Через пять лет Фукусима представил доработанную и более устойчивую версию когнитрона – неокогнитрон, состоящий из чередующихся слоев S-элементов, отвечающих за выделение признаков, и C-элементов, подавляющих искажения. Когнитрон и неокогнитрон эффективны для решения задач распознавания образов и стал основой для современных сверточных нейронных сетей.

Ключевым моментом развития нейронных сетей явилась разработка метода обратного распространения ошибки. Впервые этот метод был описан в 1974 году Александром Галушкиным и Полом Вербосом, одновременно и независимо сформулировавшим его, и формализован в 1986 году Дэвидом Румельхартом, Джеффри Хинтоном и Рональдом Уильямсом. Основной

идеей стало распространение ошибки в обратном прямому распространению сигналов направлении. Метод реализует вычисление градиента ошибки по каждому весу нейрона в сети, начиная с выходного слоя и двигаясь к входному, что позволяет корректировать веса для минимизации ошибки предсказания. Появление обратного распространения предоставляет возможность обучения многослойных нейронных сетей на сложных зависимостях и выполнять более точную классификацию.

1.3.2 Современные типы нейронных сетей

В настоящее время нейронные сети являются эффективными инструментами машинного обучения, успешно применяемые в широком спектре задач[9]. На сегодняшний день разработано большое количество различных архитектур нейронных сетей, обладающих уникальными свойствами и наиболее подходящих для реализации конкретных задач.

Многослойный перцептрон, также известный как сеть прямого распространения, является базовой архитектурой полносвязной нейронной сети. Многослойный перцептрон состоит из входного слоя, одного или нескольких скрытых слоев и выходного слоя, причем каждый элемент конкретного слоя связан со всеми элементами следующего за ним слоя.

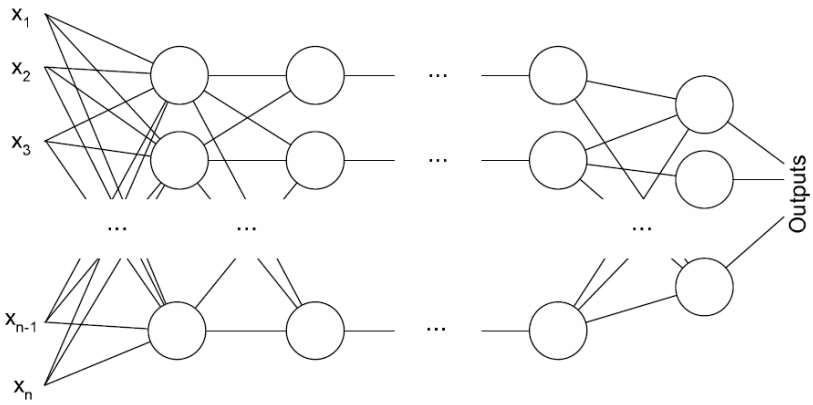

Рисунок 1.3 – Многослойный перцептрон

Во время обработки информации данные последовательно передаются от входа к выходу без использования сторонних циклов или обратной связи. Обучение полносвязных сетей выполняется при помощи алгоритма обратно-

го распространения ошибки, используемого вместе с параллельным спуском. Эти нейронные сети позволяют аппроксимировать и приближать практически любые непрерывные функции. Недостатками данной модели является плохая масштабируемость на задачи с большими входными данными, а также большие временные затраты на обучение, что делает ее непригодной для работы с изображениями.

Рекуррентная нейронная сеть разработана для работы с последовательными данными, например текстом, аудиосигналами[10]. Отличительной чертой этого типа является наличие внутренней памяти, позволяющей учитывать контекст предыдущих вычислений при обработке новых данных. Базовой архитектурой рекуррентной сети является сеть входных, скрытых и выходных узлов, каждый из которых соединен с остальными. На вход каждого шага вместе с входными данными подается результат обработки предыдущего шага, что позволяет моделировать временные зависимости. В рамках этого подхода следует отметить, что в процессе обработки данных могут возникать проблемы с исчезновением или резким увеличением градиента ошибки. Эти недостатки могут быть устранены в процессе разработки усовершенствованной сети с долгой краткосрочной памятью, LTSM. LTSM является сетью с ячейками памяти, позволяющими сохранять информацию на длительные промежутки времени. Обычно формируется с использованием «вентилей», использующихся для контроля информации на входах и выходах памяти блоков.

1.3.3 Сверточные нейронные сети

Сверточная нейронная сеть (СНС) – особый класс многослойных перцептронов, разработанный с целью распознавания образов[17]. Существенным моментом, определяющим практическую эффективность, является то, что СНС обладают высокой степенью инвариантности к пространственным искажениям, таким как смещения и повороты. На рисунке 1.5 приведена типовая архитектура СНС.

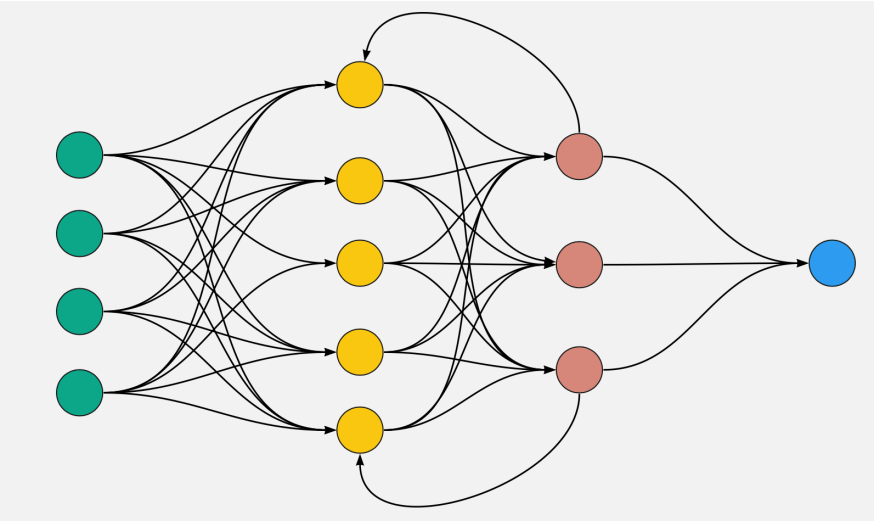

Рисунок 1.4 – Рекуррентная нейронная сеть

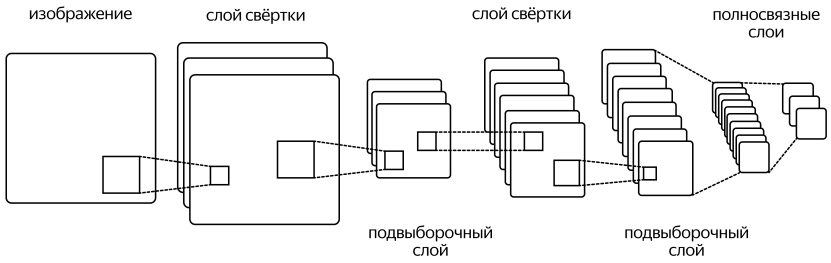

Рисунок 1.5 – Архитектура сверточной нейронной сети

Сверточные нейронные сети работают с изображениями, представленными в виде тензоров – трехмерных массивов чисел или массивов матриц чисел. Для моделирования и отображения подобной информации следует учитывать формат представления изображений в компьютерах – наборы пикселей, содержащих значения интенсивности каждого цветного канала. Данные значения варьируются в диапозоне от 0 до 255, а каналов чаще всего три – красный, зеленый, синий.

СНС состоят из слоев свертки и субдискретизации, полносвязной нейронной сети и выходного слоя. Сверточный слой представляет собой набор карт признаков, имеющих свое ядро свертки – матрицу весовых коэффициентов, устанавливаемых в процессе обучения. Назначением этого слоя является выделение признаков входного изображения и формирования карты признаков, представляющей тензор, в котором каждый канал отвечает за отдельный

признак, такой как границы или текстуры. Для выделения этих признаков используются ядра, также называемые фильтрами – наборы тензоров одинакового размера. Количество тензоров в ядре определяет глубину выходного 3D-массива. Ядра используются в процессе вычисления нового значения выбранного пикселя с учетом значения окружающих его пикселей, называемом сверткой. Фильтр накладывается на левую верхнюю часть изображения, после чего производится покомпонентное умножение значений фильтра и изображения, после чего перемещается по изображению с определенным шагом до тех пор, пока не охватит его полностью[18]. После получения каналов каждого фильтра, результрующие матрицы объединяются в единый тензор.

Входными параметрами сверточного слоя являются:

– тензор размерностью W_1xH_1xD_1, где W_1 – высота, H_1 – ширина, D_1 – глубина, или количество каналов;

– fc – количество фильтров;

– fs – размер фильтров;

– S – шаг свертки;

– P – количество добавленных пикселей.

На выходе слоя получают тензор размером W_2xH_2xD_2, где $W_2 = (W_1 - f_s + 2P)/S + 1$, $H_2 = (H_1 - f_s + 2P)/S + 1$, $D_2 = fc$. Примерами фильтров являются слои подывыборки (пуллинга), сокращающие пространственные размерности изображения в указанное количество раз и слои активации, представляющие из себя функцию, применяющуюся к каждому числу входного изображения.

Размер карт признаков сверточного слоя одинаковый и вычисляется по формуле:

$$(w, h) = (mW - kW + 1, mH - kH + 1)$$

, где (w, h) – размер сверточной карты, mW, mH – ширина и высота предыдущей карты, kW, kH – ширина и высота ядра.

Подвыборочный слой предназначен для уменьшения размерности карт предыдущего слоя, уплотняя изображения до менее подробных. Производит-

ся подвыборка при помощи операции макспуллинга – выбора максимального значения.

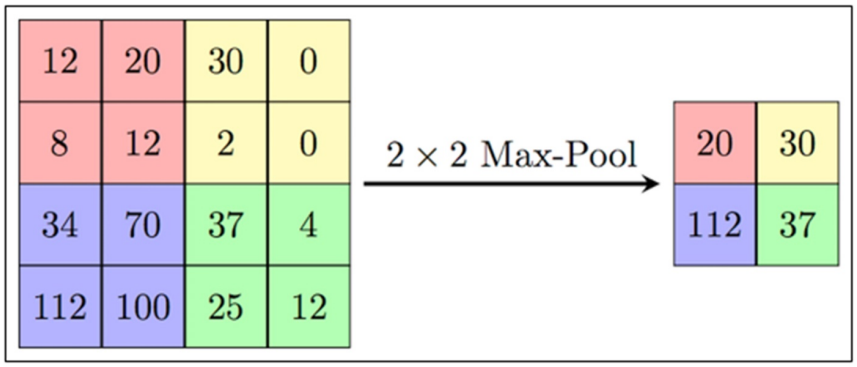

Рисунок 1.6 – Операция макспуллинга

Как правило, каждая карта этого слоя имеет размерность 2x2, что позволяет уменьшить изображение в два раза. Благодаря уменьшению размера увеличивается скорость вычислений, упрощается поиск признаков более высокого уровня на следующих сверточных слоях, так как ненужные детали отбрасываются.

Полносвязный слой является последним в структуре сверточной нейронной сети и представляет собой слой обычного многослойного перцептрона. Его цель – классификация при помощи нелинейной функции, улучшая качество распознавания. Выходной слой связан со всеми нейронами предыдущего слоя, количество нейронов которого обычно соответствует количеству распознаваемых классов.

Сверточные нейронные сети обучаются при помощи алгоритма обратного распространения ошибки. Сначала выполняется прямое распространение от первого слоя к последнему, после чего на выходном слое вычисляется ошибка, передающаяся обратно. На каждом слое вычисляются градиенты обучаемых параметров,, используемые для обновления весов при помощи градиентного спуска в конце обратного распространения.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу бакалавра «Интеллектуальная система мониторинга и распознавания загрязнений водоемов».

2.2 Цель и назначение разработки

Программная система предназначения для автоматического распознавания пятен нефтяных разливов на изображениях и их визуального выделения.

Пользователи должны иметь возможность загружать собственные изображения для анализа системой. Также должен быть реализован функционал сохранения полученных результатов программного анализа .

Задачами данной разработки являются:

– создание интеллектуальной системы распознавания на основе технологий нейронных сетей;

– обучение созданной интеллектуальной системы для распознавания пятен нефтяных разливов;

– разработка функционала для тестирования точности анализа полученной системы;

– реализация настольного приложения с графическим интерфейсом для взаимодействия пользователей с системой.

2.3 Требования пользователя к программной системе

2.3.1 Требования к данным программной системы

Для обучения нейронной сети и анализа изображений на предмет наличия пятен разливов нефти программной системе требуется датасет, состоящий из изображений в формате JPEG, сохраненные в отдельной папке. Кроме того, для сохранения параметров обученной модели нейронной сети исполь-

зуются файлы формата .pth, содержащие веса и смещения модели. Для анализа изображений и тестирования нейронной сети также требуются файлы в формате .pth, содержащие параметры, предварительно полученные после обучения нейронной сети.

2.3.2 Функциональные требования к программной системе

Разработанная программа должна реализовывать следующие функциональные возможности:

– обучение нейронной сети: после выбора директории с изображениями, директории сохранения полученных параметров и параметров обучения, программа запускает обучение нейронной сети и, по его завершении, сохраняет полученный набор параметров в указанную пользователем директорию;

– тестирование нейронной сети: после загрузки тестового изображения и предварительно сохраненного файла параметров нейронной сети, программа должна обрабатывать полученное изображение при помощи модели сети, выводить результат анализа в виде бинарного изображения, идеальный ожидаемый результат, полученный с помощью простого алгоритма обработки и метрики, оценивающие эффективность работы нейронной сети;

– распознавание нефтяных пятен: программа должна распознавать пятна разливов нефти на фотографиях при помощи предоставленных пользователем параметров на загруженном изображении;

– сохранение результатов: пользователь должен иметь возможность сохранения проанализированных изображений на жесткий диск.

На рисунке 2.1 предоставлены функциональные требования к системе, представленные в виде диаграммы прецедентов[6].

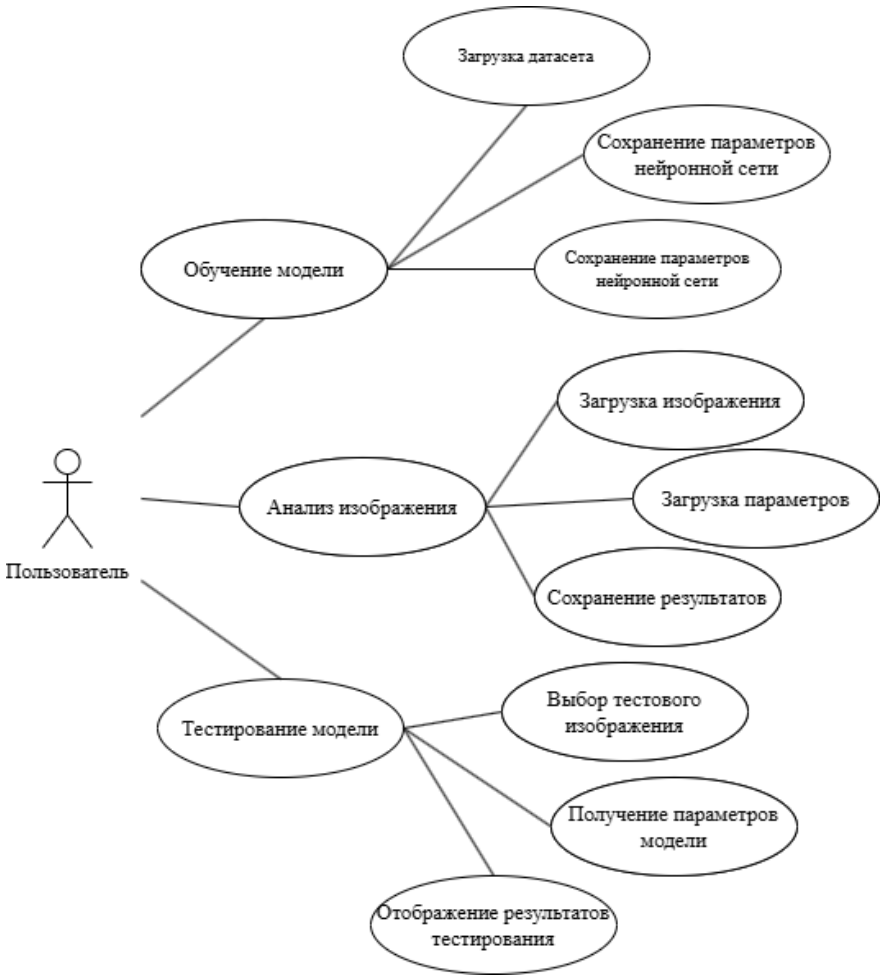

Рисунок 2.1 – Диаграмма прецедентов

2.3.2.1 Сценарий использования «Обучение нейронной сети»

Заинтересованные лица и их требования: пользователь желает обучить модель нейронной сети для распознавания нефтяных пятен на поверхности водоемов.

Предусловие: программа запущена, выбран режим «Обучение».

Постусловие: программа сохраняет файл, содержащий параметры весов нейронной сети.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку «Выбрать папку» .
2. Программа открывает диалоговое окно выбора папки.
3. Пользователь выбирает папку, содержащую изображения для обучения нейронной сети.

4. Программа отображает путь до выбранной папки в специальном поле.

5. Пользователь нажимает на кнопку «Выбрать путь сохранения».

6. Программа открывает диалоговое окно выбора папки сохранения.

7. Пользователь выбирает директорию, в которой будет сохранен файл, содержащий параметры нейронной сети.

8. Программа отображает путь до выбранной директории сохранения в специальном поле.

9. Пользователь выбирает количество изображений, обрабатываемых нейронной сетью одновременно (размер батча).

10. Пользователь указывает количество эпох обучения.

11. Пользователь нажимает на кнопку «Обучить нейросеть».

12. Программа начинает процесс обучения нейронной сети на полученных изображениях, используя заданные параметры.

13. Программа отображает прогресс процесса обучения пользователю.

14. Программа сохраняет полученный файл, содержащий параметры нейронной сети в папку, указанную пользователем, после завершения процесса обучения.

2.3.2.2 Сценарий использования «Тестирование нейронной сети»

Заинтересованные лица и их требования: пользователь желает протестировать работу предварительно обученную нейронную сеть и узнать качество сегментации.

Предусловие: программа запущена, выбран режим «Тестирование», имеется файл с настройками нейронной сети.

Постусловие: программа отображает метрики качества сегментации, а также сравнение ожидаемых и полученных результатов.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку «Выбрать изображение».

2. Программа открывает диалоговое окно выбора изображения, используемого для тестирования нейронной сети.

3. Пользователь выбирает изображение.

4. Программа отображает путь до выбранного изображения в специальном поле.

5. Пользователь нажимает на кнопку «Выбрать настройки».

6. Программа открывает диалоговое окно выбора файла настроек нейронной сети.

7. Пользователь выбирает файл настроек нейронной сети.

8. Программа отображает путь до выбранного файла настроек в специальном поле.

9. Пользователь нажимает на кнопку «Запустить тестирование».

10. Программа выполняет предварительную обработку изображения.

11. Программа создает ожидаемую маску на основании загруженного изображения при помощи порогового алгоритма.

12. Программа обрабатывает обработанное изображение, используя нейронную сеть с загруженными параметрами.

13. Программа рассчитывает метрики качества сегментации изображения.

14. Программа отображает ожидаемую и полученную нейронной сетью маски классов, а также метрики качества сегментации.

2.3.2.3 Сценарий использования «Анализ изображения»

Заинтересованные лица и их требования: пользователь желает проанализировать изображение на наличие нефтяного разлива.

Предусловие: программа запущена, выбран режим «Анализ изображения», имеется файл с настройками нейронной сети.

Постусловие: программа отображает результаты анализа изображения.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку «Выбрать изображение».

2. Программа открывает диалоговое окно выбора анализируемого изображения.

3. Пользователь выбирает необходимое изображение.

4. Программа отображает путь до выбранного изображения в специальном поле и выводит его в главное окно.

5. Пользователь нажимает на кнопку «Выбрать модель».

6. Программа открывает диалоговое окно выбора файла с настройками модели.

7. Пользователь выбирает файл настроек нейронной сети.

8. Программа отображает путь до выбранного файла в специальном поле.

9. Пользователь нажимает на кнопку «Анализировать».

10. Программа выполняет предварительную обработку изображения.

11. Программа анализирует обработанное изображение на предмет наличия пятен нефтяных разливов с помощью нейронной сети.

12. Программа отображает результат анализа на исходной версии изображения, загруженной пользователем в главном окне программы.

2.3.2.4 Сценарий использования «Сохранение результата анализа»

Заинтересованные лица и их требования: пользователь желает сохранить результат анализа изображения нейронной сетью.

Предусловие: программа запущена, выбран режим «Анализ изображения», изображение предварительно загружено и проанализированно.

Постусловие: программа сохраняет результат обработки в виде изображения на жесткий диск.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку «Сохранить результат».

2. Программа открывает диалоговое окно выбора папки для сохранения результата анализа.

3. Пользователь выбирает необходимую папку.

4. Программа сохраняет результат анализа в указанную папку в виде изображения.

5. Программа уведомляет пользователя об успешном сохранении изображения.

2.3.3 Требования пользователя к интерфейсу приложения

Приложение должно иметь следующие экраны:

1. Экран «Анализ изображения». Основной экран, реализующий функционал распознавания пятен нефти на изображениях поверхности водоемов, где реализована возможность загрузки изображения для поиска нефтяных пятен, просмотр результата работы нейронной сети и его сохранения на жесткий диск.

2. Экран «Обучение». Экран, позволяющий обучать нейронную сеть искать пятна разливов нефти на изображениях. Должен содержать возможность выбора папки, содержащей датасет для обучения, и директории сохранения файла, содержащего настройки параметров нейронной сети, полученные во время ее обучения.

3. Экран «Тестирование». Экран для тестирования обученной модели, позволяющий выбрать файл настроек нейронной сети, тестовое изображение, а также отображающий исходное изображение, ожидаемый результат и фактический результат обработки тестового изображения нейронной сетью.

2.4 Нефункциональные требования к программной системе

2.4.1 Требования к надежности

Программная система должна обеспечивать стабильную работу в различных условиях эксплуатации. В процессе работы приложения могут возникнуть следующие аварийные ситуации:

– отсутствие файлов изображений в папке, выбранной как директория хранения датапака;

– попытка загрузить в программу поврежденное изображение;

– ошибки загрузки или сохранения файлов настроек моделей или изображений из-за проблем с файловой системой;

– ошибки в работе программы из-за загрузки пользователем файлов некорректных форматов.

Для предотвращения аварийных ситуаций программа должна корректно обрабатывать исключения при работе с файлами, предоставляя пользователям информативные сообщения об ошибках. В случае проблем с отсутствием прав доступа к директории сохранения файлов, полученных в результате работы программы, программа должна открывать диалоговое окно с выбором другой директории.

2.4.2 Требования к аппаратному обеспечению

Для корректной работы программного продукта требуется центральный процессор с количеством ядер от 6 и выше с частотой ядра от 2.4 ГГц. Размер необходимой оперативной памяти - 8 Гб и выше. Кроме того, для отрисовки графического интерфейса требуется видеокарта с объемом графической памяти 4 Гб и выше, монитор. Наконец, для управления программой необходимы клавиатура и мышь.

Для ускорения процесса обучения нейронной сети возможно использование вычислительных ресурсов графического адаптера NVIDIA с поддержкой технологии CUDA.

2.4.3 Требования к программному обеспечению

Для запуска и работы программы требуется компьютер под управлением операционной системы Windows 10 или Windows 11. При использовании графических адаптеров NVIDIA с поддержкой технологии CUDA для ускорения обучения нейронной сети необходима последняя версия драйверов соответствующего адаптера, а также программы CUDA Toolkit и cuDNN.

2.5 Требования к оформлению документации

Требования к стадиям разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их

назначения и области применения, этапам и содержанию работ устанавливаются ГОСТ 19.102–77. и ГОСТ 34.601-90.

Программная документация должна включать в себя:

– анализ предметной области;
– техническое задания;
– технический проект;
– рабочий проект.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Необходимо спроектировать и разработать интеллектуальную систему распознавания пятен разливов нефти на поверхности водоемов.

Интеллектуальная система состоит из модели нейронной сети и приложения, отвечающего за взаимодействие пользователей с системой. В приложении необходимо иметь возможность взаимодействовать с нейронной сетью при помощи графического интерфейса пользователя. Под взаимодействием понимается загрузка изображений, содержащих пятна нефтяных разливов, возможность распознавания пятен при помощи модели нейронной сети, а также её обучение и тестирование.

3.2 Обоснование выбора технологии проектирования

Технологии, языки программирования и архитектурные решения, использованные для создания интеллектуальной системы, отвечают современным практикам разработки и позволяют достичь высокой производительности и отказоустойчивости программы.

3.2.1 Язык программирования Python

Для реализации программной системы был использован язык Python. Python – язык программирования высокого уровня обладающий высокой степенью гибкости[12] и широко используемый как в научной отрасли, так и для коммерческой разработки. Особенно хорошо Python подходит для построения интеллектуальных систем и анализа изображений благодаря богатой системе внешних библиотек, позволяющих значительно упростить построение модели нейронной сети[11], подготовку данных, работу с изображениями, создание графического интерфейса пользователя. Интуитивный синтаксис языка позволяет снизить количество синтаксических ошибок, что в свою очередь значительно ускоряет разработку программных продуктов. Встроенная поддержка различных парадигм программирования позволяет удобно

реализовывать модульную архитектуру приложения с использованием классов, функций и последовательного кода. Кроссплатформенность языка облегчает адаптацию системы для работы на различных операционных систем, что при желании позволит легко создать нативные версии приложений для таких операционных систем, как Linux и macOS[13].

3.2.2 Описание библиотеки PyTorch

Для разработки модели нейронной сети была выбрана библиотека PyTorch. PyTorch является открытой библиотекой, содержащей широкий набор инструментов для машинного обучения. Данная библиотека активно применяется в разнообразных научных исследованиях, использующих модели нейронных сетей. PyTorch зарекомендовала себя благодаря гибкости, лаконичному синтаксису и обширным возможностям построения и обучения нейронных сетей[16].

Для определения структуры вычислений PyTorch использует динамический вычислительный граф, что означает построение графа операций непосредственно в процессе выполнения программы вместо предварительного определения его структуры. Данный подход значительно упрощает модификацию и отладку архитектуры модели нейронной сети, позволяет удобно реализовывать операции обучения, тестирования и анализа при помощи полученной модели нейронной сети.

3.2.3 Описание библиотек OpenCV и Pillow

OpenCV и Pillow (PIL) — две широко используемые библиотеки для обработки изображений и компьютерного зрения. OpenCV обеспечивает высокую производительность при работе с изображениями, поддерживая такие операции, как выделение контуров объектов и визуализация результатов. В свою очередь, Pillow предлагает удобный интерфейс для базовой обработки изображений, включая изменение размеров, форматов, наложение масок и простые преобразования. В данной работе OpenCV применяется преимущественно для задач выделения контуров, тогда как Pillow используется для

предварительной обработки и модификации изображений, а также их сохранения.

3.2.4 Описание фреймворка PyQt5

PyQt5 является набором расширений кросплатформенного графического фреймворка Qt 6 версии для языка Python. PyQt5 позволяет создавать графические интерфейсы пользователя различной сложности – от простых оконных приложений до многооконных систем с развитой логикой взаимодействий.

Одним из ключевых преимуществ PyQt5 является большое число готовых элементов графического интерфейса, таких как кнопки, выпадающие списки, таблицы, позволяющих строить полноценный интуитивный пользовательский интерфейс. Графический интерфейс PyQt можно разрабатывать как вручную в коде, так и при помощи визуального редактора Qt Designer, возвращающего пользователю готовый Python-код для графического интерфейса. Кроссплатформенность PyQt5 позволяет запускать приложения, разработанные для операционной системы Windows на компьютерах с Linux и macOS не требуя значительных изменений, что позволяет без особых усилий создавать мультиплатформенные версии приложений.

3.2.5 Описание библиотеки NumPy

Библиотека NumPy предназначена для работы с многомерными массивами и матрицами, а также позволяет проводить сложные математические вычисления с высокой скоростью. NumPy предлагает большое количество функций для работы с массивами, таких как арифметические и логические операции, линейная алгебра.

NumPy позволяет преобразовывать изображения и их маски в числовые массивы для дальнейшей обработки, а также объединять и нормализовать эти маски. Компактный синтаксис и высокая производительность делают его подходящим для практически любых проектов, связанных с обработкой данных[15].

3.3 Архитектура программной системы

На рисунке 3.1 в виде UML-диаграммы[7] представлена архитектура программной системы.

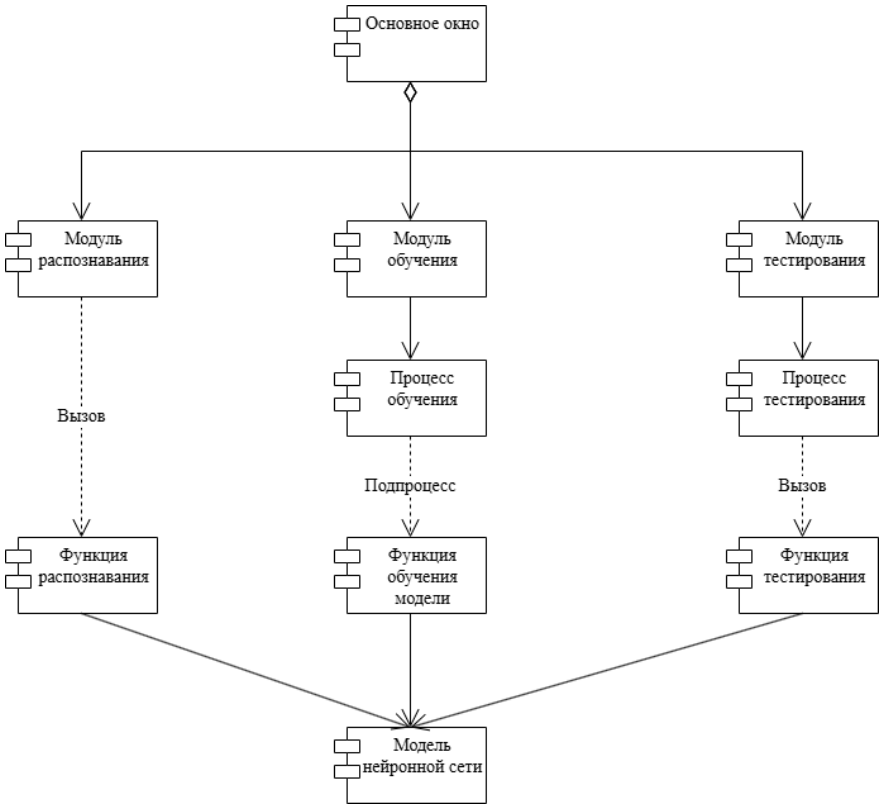

Рисунок 3.1 – Архитектура системы

Интеллектуальная система состоит из следующих компонентов:

1. Основное окно. Основное окно отвечает за вызов главного окна программной системы и инициализацию трех режимов работы программы и переключение между ними.

2. Модуль распознавания. Данный модуль вызывает окно для распознавания изображения, позволяющее выбрать изображение и параметры модели для анализа нейронной сетью, запускает процесс распознавания и отображает результат.

3. Функция распознавания. Использует модель нейронной сети и ее параметры для анализа полученного изображения, возвращая результат работы и сохраняя его в виде файла.

4. Модуль обучения. Вызывает окно, позволяющее выбрать директорию, содержащую изображения для обучения нейронной сети, и указать директорию для сохранения файла параметров обученной нейронной сети. Кроме того, данный модуль запускает процесс, отвечающий за обучение нейронной сети и отображение прогресса завершения обучения.

5. Функция обучения модели. Компонент загружает изображения для обучения, преобразует их в требуемый для обучения нейронной сети вид и обучает модель, сохраняя полученный в результате файл параметров.

6. Модуль тестирования. Вызывает окно, позволяющее выбрать параметры модели, тестовое изображение и запускает процесс, отвечающий за тестирование модели нейронной сети с сохраненными параметрами и возвращает результаты тестирования. Также отображает полученные метрики и результаты.

7. Функция тестирования. Выполняет оценку точности модели нейронной сети с параметрами, загруженными из предварительно сохраненного файла. Сравнивает результат анализа изображения нейронной сетью и ожидаемый результат, полученный при помощи пороговой фильтрации.

8. Модель нейронной сети. Хранит в себе структуру используемой в системе нейронной сети.

3.4 Описание нейронной сети

Для распознавания нефтяных пятен на изображениях поверхностей водоемов была спроектирована и разработана модель сверточной нейронной сети. Сверточные нейронные сети – однонаправленные модели, разработанные для распознавания образов на изображениях.

3.4.1 Метод распознавания пятен нефти

Для определения пятен нефти на изображениях сверточные нейронные сети могут применяться в соответствии с одним из следующих подходов:

– распознавание объектов на изображении;

– семантическая сегментация.

Распознавание объектов обучает нейронную сеть распознавать образы, выделяя их при помощи ограничивающих рамок прямоугольной области, окружающих распознанный объект. Нейронная сеть, обученная при помощи семантической сегментации, выполняет попиксельную разметку, присваивая каждому пикселю метку определенного класса. В отличие от распознавания объектов, семантическая сегментация позволяет снизить погрешность распознавания и повысить точность локализации, минимизируя возможность ложного распознавания сторонних объектов, окружающих нефтяные разливы, зачастую представляющие собой пятна неправильной формы.

Для создания интеллектуальной системы был выбран семантический подход сегментации, основанный на распознавании определенных уровней яркости пикселей. В большинстве случаев при переводе изображения, содержащего пятна разлива нефти, в полутоновое, объект интереса (пятно) будет представлен скоплениями темных пикселей. Обучив нейронную сеть распознавать пиксели, яркость которых ниже определенного порогового значения, можно эффективно обнаруживать и выделять нефтяные разливы на изображении.

3.4.2 Архитектура U-Net

В данной работе была реализована нейронная сеть типа U-Net, адаптированная под особенности предметной области.

Архитектура U-Net была представлена в 2015 году группой исследователей из Университета Фрайбугра. Данная архитектура получила широкое распространение благодаря способности выделять объекты различной формы и масштаба с высокой точностью даже на маленьких объемах обучающих выборок. Характерной особенностью U-Net является симметричная структура: левая часть сети, или энкодер, выполняет последовательное сжатие входного изображения, а правая, или декодер – восстановление пространственного разрешения до исходного размера[19].

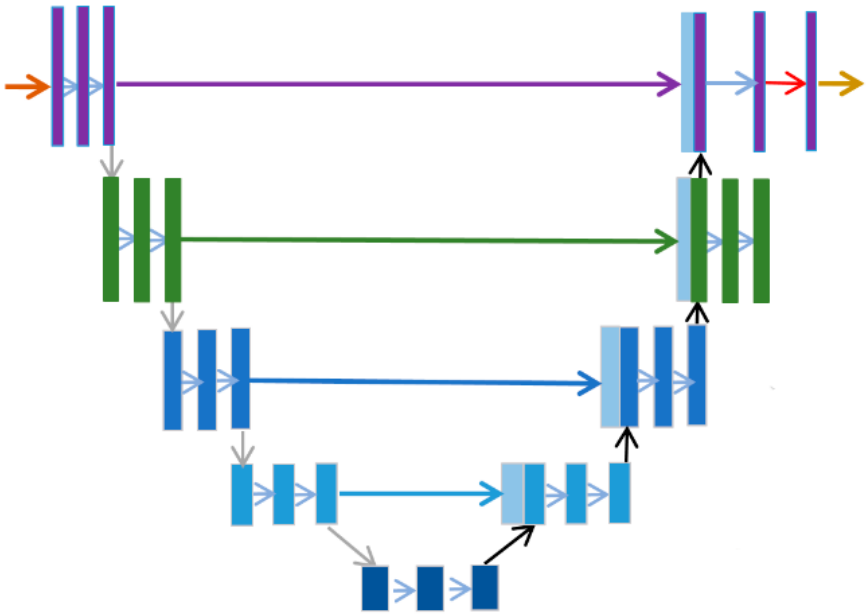

Рисунок 3.2 – Архитектура сети U-Net

Ключевым элементом U-Net являются прямые соединения между соответствующими уровнями энкодера и декодера, позволяющие использовать признаки, извлеченные на ранних этапах обработки, передавая их на этапы восстановления изображения, что предотвращает потерю пространственной информации. Отмеченные особенности позволяют с легкостью адаптировать U-Net для использования в различных областях, например медицинская диагностика, спутниковый мониторинг, обработка изображений, полученных с беспилотных летательных аппаратов.

3.4.3 Структура нейронной сети

Реализация модели U-Net состоит из энкодера, узкого места и декодера. Энкодер состоит из двух уровней, содержащих по два сверточных слоя, преобразующих 1 канал в 64 и 64 канала в 128, создавая 64 и 128 карт признаков соответственно, каждая из которых отвечает за определенные особенности изображения, такие как горизонтальные и вертикальные края, их комбинации, текстуры. Каждый из сверточных слоев, используемых в уровнях, имеет размерность маски 3x3, проходящей по изображению с шагом 1 (маска смещается на 1 пиксель) и извлекающей вышеуказанные признаки с сохранением размеров изображения. После извлечения энкодер уменьшает изоб-

ражение в 2 раза, используя маску размерностью 2x2, смещающуюся на 2 пикселя за шаг. Операция уменьшения повторяется после каждого извлечения признаков, в общем уменьшая размеры изображения в 4 раза.

Блок узкого места продолжает процесс извлечения признаков, распознавая абстрактные признаки, такие как крупные структуры и формы, добавляя дополнительные 128 каналов до общей суммы 256, перемещая маску размерности 3x3 шагом в 1. Размер самого изображения при этом остается неизменным.

Слой декодера возвращает изображению исходные размеры. При помощи перемещения маски размерностью 2x2 с шагом 2 увеличивает размер полученного из предыдущего этапа изображения в 2 раза, растягивая каждый пиксель в блок размером 2x2 и корректируя значения при помощи весов, вместе с этим комбинируя данные признаков, уменьшая размерность каналов в 2 раза до 128. Затем слой уточняет признаки, полученные из предыдущего шага и признаки, полученные энкодером на 2 шаге свертки, и возвращает изображение с размерностью, полученной на предыдущем шаге и 256 картами признаков, содержащую признаки как высокого, так и низкого уровней. Наконец, сверточный блок использует маску 3x3 с шагом 1 и уточняет карты признаков высокого уровня из узкого места и низкого уровня из энкодера, тем самым уменьшая количество каналов в два раза, до 128. Данный процесс повторяется два раза, по одному на каждый уровень декодера, восстанавливая исходные размеры изображения и объединяя карты признаков. Финальный слой объединяет оставшиеся 64 карты признаков в единую маску, значения которой обозначают вероятность принадлежности пикселя к классу разлива, которая, после применения порога функцией активации, изменяется в бинарную маску (1, если пиксель является частью разлива и 0, если нет).

3.4.4 Обучение нейронной сети

Обучение модели нейронной сети происходит на наборе полутоновых изображений формата JPEG разрешения 624x320. Целевые маски для обучения создаются на основании яркостных признаков при помощи алгорит-

ма бинаризации с порогом 0,5. Размер целевых масок совпадает с размером входного изображения. Датасет загружается из отдельной директории и полностью используется для обучения без разделения на обучающую, валидационную и тестовую выборки.

Обучение происходит на протяжении 25 эпох на группах из двух изображений. Дополнительные аугментации не применяются. Обучение проводится на центральном процессоре, при этом реализована возможность обучения на графическом процессоре при наличии поддерживаемого устройства. На каждой эпохе для каждой группы изображений входные данные и маски отправляются на вычислительное устройство, модель делает предсказание, после чего вычисляется потеря по формуле:

$$L = -\left[y \cdot \log(p) + (1 - y) \cdot \log(1 - p)\right],$$

где y – идеальное значение, p – предсказанная вероятность. Наконец, градиенты обнуляются, вычисляются заново, после чего веса модели обновляются.

3.5 Проектирование пользовательского интерфейса

На основании требований к пользовательскому интерфейсу, представленных в пункте 2.3.3, был разработан графический интерфейс программной системы[8].

На рисунке 3.3 представлен макет интерфейса окна «Анализ изображения». Макет содержит следующие элементы:

1. Кнопка переключения режимов окна.
2. Поле, содержащее путь до анализируемого изображения.
3. Кнопка выбора изображения для анализа.
4. Поле, содержащее путь до выбранного файла параметров нейронной сети.
5. Кнопка выбора файла параметров нейронной сети.
6. Поле для отображения загруженного изображения и результатов анализа.
7. Кнопка для запуска распознавания нефтяных пятен.

8. Кнопка для сохранения результатов анализа нейронной сети.

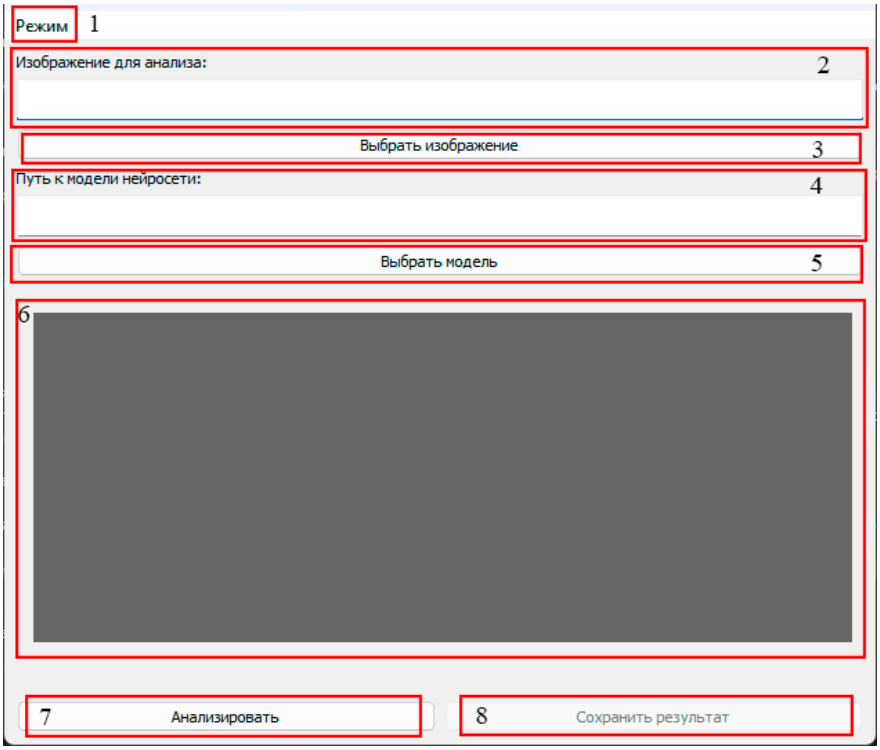

Рисунок 3.3 – Макет интерфейса окна «Анализ изображения»

На рисунке 3.4 представлен макет интерфейса окна «Обучение». Макет содержит следующие элементы:

1. Кнопка переключения режимов окна.
2. Поле, содержащее путь до выбранной папки с изображениями для обучения нейронной сети.
3. Кнопка выбора папки с изображениями для обучения нейронной сети.
4. Поле, содержащее путь до папки, в которую необходимо сохранить параметры модели.
5. Кнопка для выбора папки, в которую необходимо сохранить параметры модели.
6. Элемент для выбора количества изображений, обрабатываемых одновременно.
7. Поле для выбора количества эпох обучения нейронной сети.
8. Шкала прогресса завершения эпохи обучения нейронной сети.

9. Поле для отображения информации о процессе обучения нейронной сети.

10. Кнопка для запуска обучения.

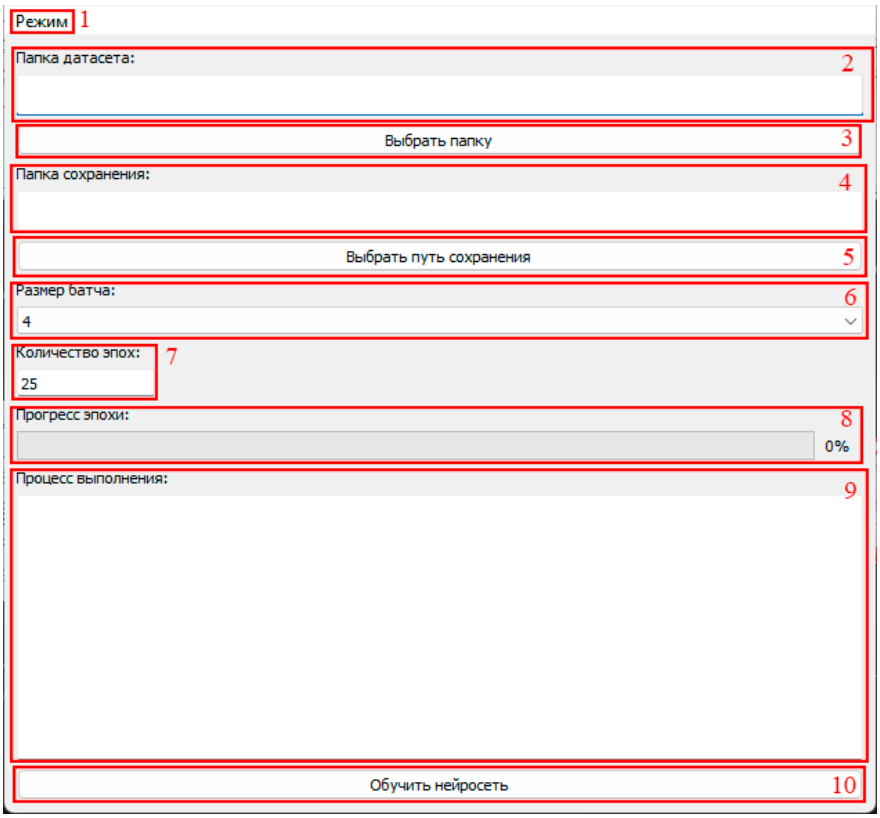

Рисунок 3.4 – Макет интерфейса окна «Обучение»

На рисунке 3.5 представлен макет интерфейса окна «Тестирование». Макет содержит следующие элементы:

1. Кнопка переключения режимов окна.
2. Поле, содержащее путь до тестового изображения.
3. Кнопка выбора изображения для тестирования нейронной сети.
4. Поле, содержащее путь до выбранного файла параметров нейронной сети.
5. Кнопка выбора файла параметров нейронной сети.
6. Поле вывода метрик тестирования нейронной сети.
7. Поле для отображения исходного изображения.
8. Поле для отображения ожидаемого результата обработки.

9. Поле для отображения фактического результата обработки изображения нейронной сети.

10. Кнопка для запуска тестирования.

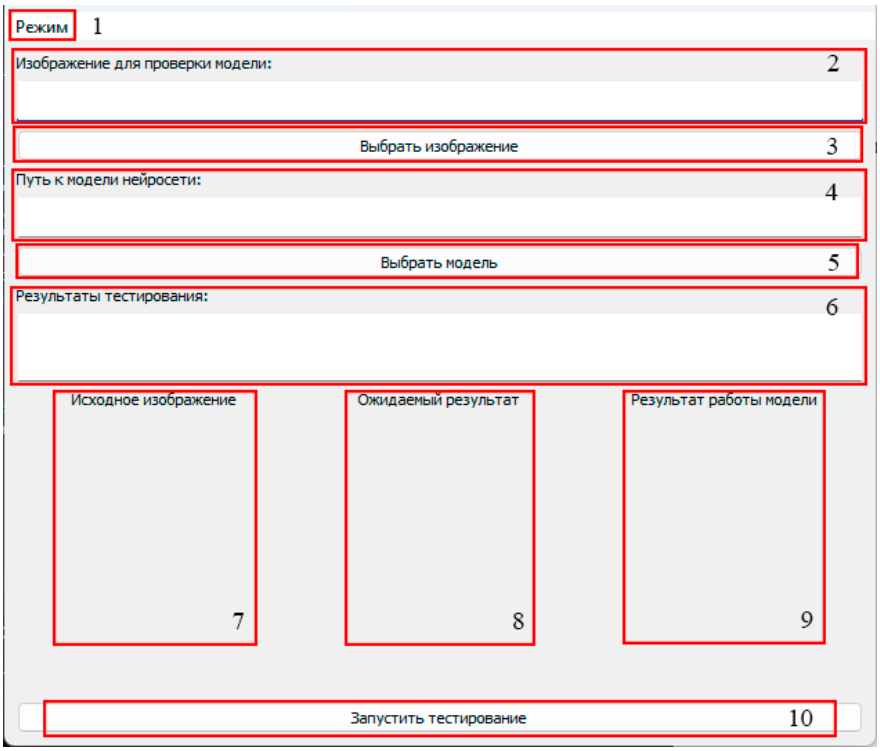

Рисунок 3.5 – Макет интерфейса окна «Тестирование»

4 Рабочий проект

4.1 Спецификация компонентов и классов программы

4.1.1 Модули программной системы

4.1.1.1 Модуль main.py

Модуль предоставляет графический интерфейс с меню для переключения между тремя режимами работы: анализ изображений, обучение и тестирование нейронной сети.

Класс – MainWindow.

Описание класса MainWindow: Класс предназначен для управления главным окном приложения и переключения между режимами работы программы. Базовый класс – QMainWindow, стандартный класс библиотеки PyQt5. Интерфейсы: панель меню, позволяющая переключать режимы работы программы; центральный виджет QStackedWidget, отображающий интерфейс активного режима работы. Константы: отсутствуют. Внутренние поля:

– stack: QStackedWidget – содержит три виджета графического интерфейса для каждого режима работы программы;

– analysis_widget: ImageAnalysisWidget – виджет, содержащий графический интерфейс режима анализа изображений;

– training_widget: TrainingWidget – виджет, содержащий графический интерфейс режима обучения нейронной сети;

– testing_widget: TestingWidget – виджет, содержащий графический интерфейс режима тестирования нейронной сети.

Методы класса представлены в таблице 4.1.

Таблица 4.1 – Методы класса MainWindow

Название метода	Область видимости	Назначение метода
__init__	Внутренний	Инициализирует главное окно программы, задает его параметры, заголовок, панель меню с действиями для переключения режимов
switch_mode	Общедоступный	Переключает графический интерфейс в соответствии с выбранным режимом работы

Метод __init__ не имеет входных и возвращаемых данных. Метод switch_mode. Входные данные: index (тип int) – идентификатор виджета активного режима окна. Возвращаемых данных нет.

4.1.1.2 Модуль model.py

Модуль определяет структуру нейронной сети UNet, используемой для обнаружения разливов нефти.

Класс – UNet.

Описание класса UNet: Класс реализует модель UNet для сегментации и распознавания пятен нефтяных разливов на поверхности водоемов. Базовый класс – nn.Module, стандартный класс библиотеки PyTorch. Интерфейсы: общедоступные методы __init__ и forward. Константы отсутствуют. Внутренние поля:

– enc1: nn.Sequential – первый блок энкодера;

– enc2: nn.Sequential – второй блок энкодера;

– pool: nn.MaxPool2d – слой максимального пуллинга, уменьшающий разрешение;

– bottleneck: nn.Sequential – блок узкого места;

– upconv2: nn.ConvTranspose2d – второй слой повышения разрешения;

– dec2: nn.Sequential – второй блок декодера;

– upconv1: nn.ConvTranspose2d – первый слой повышения разрешения;

– dec1: nn.Sequential – первый блок декодера;

– final_conv: nn.Conv2d – финальный сверточный слой, создающий выходную маску признаков.

Методы класса представлены в таблице 4.2

Таблица 4.2 – Методы класса UNet

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует архитектуру UNet
forward	Общедоступный	Выполняет прямой проход через нейронную сеть, возвращая итог сегментации
conv_block	Внутренний	Определяет сверточный блок с двумя сверточными слоями и функциями активации

Метод __init__. Входные данные:

– in_channels (тип int, значение по умолчанию – 1) – количество входных каналов;

– out_channels (тип int, значение по умолчанию – 1) – количество выходных каналов.

Возвращаемых данных нет.

Метод forward. Входные данные: x (объект torch.Tensor формы (batch_size, in_channels, height, width)) – входной тензор. Возвращаемые данные: объект torch.Tensor – выходная маска признаков сходной формы.

Метод conv_block. Входные данные:

– in_c (тип int) – входные каналы.

– out_c (тип int) – выходные каналы.

Возвращаемые данные: объект nn.Sequential – контейнер сверточного блока.

4.1.1.3 Модуль test.py

Модуль содержит функции для оценки обученной модели UNet на одном изображении. Не содержит классов. Методы модуля:

1. dice_coefficient. Вычисляет коэффициент Dice для предсказанной маски. Входные данные:

– pred (тип torch.Tensor) – предсказанная маска;

– target (тип torch.Tensor) – целевая маска;

– smooth (тип float, значение по умолчанию – $1e^{-6}$) – стабилизирующая константа для избежания деления на ноль.

Возвращаемые данные – коэффициент Dice в виде float-числа.

2. iou_score. Вычисляет пересечение по объединению (коэффициент IoU) для предсказанной маски. Входные данные:

– pred (тип torch.Tensor) – предсказанная маска;

– target (тип torch.Tensor) – целевая маска;

– smooth (тип float, значение по умолчанию – $1e^{-6}$) – стабилизирующая константа для избежания деления на ноль.

Возвращаемые данные – коэффициент пересечения по объединению в виде float-числа.

3. load_image. Загружает изображение и выполняет предобработку для дальнейшего анализа нейронной сетью. Входные данные:

– path (тип str) – путь к загружаемому изображению;

– size (тип tuple) – целевой размер изображения для дальнейшей обработки нейронной сетью.

Возвращаемые данные: нормализованный массив изображения типа np.ndarray.

4. run_evaluation. Выполняет оценку точности обученной модели нейронной сети и возвращает результаты обработки и метрики. Входные данные:

– image_path (тип str) – путь к анализируемому изображению;

– weights (тип str) – путь к оцениваемым весам модели;

– threshold (тип float, значение по умолчанию = 0,5) – порог для бинаризации масок

Возвращаемые данные – значения коэффициентов Dice и IoU, исходное изображение, ожидаемую маску, полученную при помощи пороговой бинаризации и предсказанную сетью маску в виде словаря dict.

4.1.1.4 Модуль train.py

Модуль управляет загрузкой датасета и обучением нейронной сети.

Классы: dataset и Trainer.

Описание класса dataset: Класс загружает и предобрабатывает изображения и создает псевдомаски на основании порога для обучения нейронной сети. Базовый класс – Dataset, стандартный класс библиотеки PyTorch. Интерфейсы – общедоступные методы __init__, __len__, __getitem__, общедоступные атрибуты image_dir, images. Константы отсутствуют. Внутренние поля:

– image_dir: str – папка, содержащая датасет;
– images: list – список имен файлов изображений для обучения;
– threshold: float – порог для создания обучающих масок.

Методы класса представлены в таблице 4.3.

Таблица 4.3 – Методы класса dataset

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует датасет, проверяя директорию и загружая имена файлов изображений
__len__	Общедоступный	Определяет количество изображений в датасете

Продолжение таблицы 4.3

Название метода	Область видимости	Назначение метода
__getitem__	Общедоступный	Загружает, предобрабатывает и создает обучающую маску для изображений датасета

Метод __init__. Входные данные:

– image_dir (тип str) – путь к папке с датасетом;

– threshold (тип float, значение по умолчанию – 0,5) – порог для создания обучающих масок.

Метод __len__. Входные данные отсутствуют. Возвращаемые данные – количество изображений в датасете в виде int-числа.

Метод __getitem__. Входные данные: idx (тип int) – порядковый номер обрабатываемого изображения. Возвращаемые данные – тензор изображения и маски в виде кортежа tuple[torch.Tensor, torch.Tensor].

Описание класса Trainer: Класс управляет обучением модели нейронной сети и отправляет сигналы о состоянии процесса обучения для обновления графического интерфейса. Базовый класс – QObject, стандартный класс библиотеки PyQt5. Интерфейсы – сигналы о состоянии процесса обучения, общедоступный метод __init__. Константы отсутствуют. Внутренние поля:

– epoch_start_signal: pyqtSignal[int, int] – сигнал начала эпохи, содержащий её номер и общее количесвто эпох;

– epoch_complete_signal: pyqtSignal[int, float] – сигнал завершения эпохи, содержащий её номер и среднее значение потери при обучении;

– batch_progress_signal: pyqtSignal[int, int, float] – сигнал прогресса обработки батча, содержащий его номер, потерю, общее число батчей;

– training_complete_signal: pyqtSignal[str] – сигнал завершения обучения.

Методы класса представлены в таблице 4.4

Таблица 4.4 – Методы класса Trainer

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует процесс обучения с определенными параметрами
run	Общедоступный	Обучает нейронную сеть, отправляя сигналы о прогрессе, и сохраняет параметры модели по завершении обучения

Метод __init__. Входные данные:

– image_dir (тип str) – путь датасета;

– save_path (тип str) – путь сохранения параметров обученной модели;

– batch_size (тип int, значение по умолчанию – 1) – размер батча;

– epochs (тип int, значение по умолчанию – 25) – количество обучающих эпох;

– lr (тип float, значение по умолчанию – $1e^{-4}$) – скорость обучения;

– threshold (тип float, значение по умолчанию – 0,5) – порог для обучающих масок.

Возвращаемых данных нет.

Функция run не имеет входных и возвращаемых данных.

В модуле train.py также реализован метод main(), предназначенная для получения аргументов обучения из графического интерфейса и запуска обучения при помощи Trainer. Метод не имеет входных и возвращаемых данных.

4.1.1.5 Модуль detect.py

Модуль используется для анализа изображений с использованием обученной модели нейронной сети, обнаруживая и выделяя пятна нефтяных разливов. Не содержит классов. Методы модуля:

1. load_model. Загружает и инициализирует модель нейронной сети с указанными параметрами. Входные данные – model_path (тип str) – путь к

файлу параметров модели. Возвращаемые данные – элемент класса Unet модуля model.py с загруженными из файла весами.

2. analyze_return. Метод обрабатывает изображение, используя нейронную сеть, создает бинарную маску разлива и возвращает исходное изображение с выделенными распознанными пятнами нефтяных разливов. Входные данные:

– image_path (тип str) – путь к анализируемому изображению;

– model_path (тип str) – путь к файлу параметров нейронной сети;

– threshold (тип float, значение по умолчанию – 0,5) – порог для бинаризации маски.

Возвращаемые данные – изображение с выделенными распознанными нефтяными пятнами в формате np.ndarray.

4.1.1.6 Модуль analyze_ui.py

Модуль содержит графический интерфейс режима анализа изображений с использованием предварительно обученной модели нейронной сети.

Класс – ImageAnalysisWidget.

Описание класса ImageAnalysisWidget: Класс содержит графический интерфейс, позволяющий выбрать анализируемое изображение, файл параметров нейронной сети, запустить процесс анализа и сохранить результат и отображающий этот результат. Базовый класс – QWidget, стандартный класс библиотеки PyQt5. Интерфейсы – поля для путей к изображению и модели, кнопки для выбора этих путей, начала анализа, сохранения результата, область отображения изображения. Константы отсутствуют. Внутренние поля класса:

– image_path: str – путь к выбранному изображению;

– model_path: str – путь к выбранному файлу параметров нейронной сети;

– result_img: np.ndarray – проанализированное изображение с разметкой;

– path_label: QLabel – надпись «Изображение для анализа:»;

– path_field: QTextEdit – текстовое поле для отображения пути к выбранному изображению;

– select_button: QPushButton – кнопка для выбора изображения;

– model_label: QLabel – надпись «Путь к модели нейросети:»;

– model_path_field: QTextEdit – текстовое поле для отображения пути к выбранному файлу параметров нейронной сети;

– select_model_button: QPushButton – кнопка для выбора файла параметров нейронной сети;

– image_label: QLabel – поле для отображения входного или проанализированного изображения;

– analyze_button: QPushButton – кнопка для анализа изображения;

– save_button: QPushButton – кнопка для сохранения результата анализа.

Методы класса представлены в таблице 4.5

Таблица 4.5 – Методы класса ImageAnalysisWidget

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует графический интерфейс, включая его макет и элементы
select_image	Общедоступный	Открывает диалоговое окно выбора изображения, отображает путь к нему и само изображение в интерфейсе
select_model	Общедоступный	Открывает диалоговое окно выбора файла настроек модели и отображает путь к нему

Продолжение таблицы 4.5

Название метода	Область видимости	Назначение метода
analyze_image	Общедоступный	Анализирует загруженное изображение при помощи нейронной сети и отображает результат распознавания
save_result	Общедоступный	Открывает диалоговое окно выбора папки сохранения результата анализа изображения и сохраняет результат

Методы в этом классе не имеют входных и возвращаемых данных.

4.1.1.7 Модуль train_ui.py

Модуль содержит графический интерфейс для настройки параметров и запуска процесса обучения модели нейронной сети.

Классы: TrainingThread, TrainingWidget.

Описание класса TrainingThread: Данный класс выполняет процесс обучений нейронной сети в отдельном потоке для избежания блокировки и зависания графического интерфейса. Базовый класс – QThread, стандартный класс библиотеки PyQt5. Интерфейсы – сигналы для передачи прогресса обучения и обработки ошибок, функция, управляющая процессом обучения. Константы отсутствуют. Внутренние поля:

– trainer: Trainer – экземпляр класса Trainer из train.py для обучения нейронной сети;

– error_signal: pyqtSignal[str] – сигнал для передачи сообщений об ошибках.

Методы класса представлены в таблице 4.6

Таблица 4.6 – Методы класса TrainigThread

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует поток обучения нейронной сети с экземпляром класса Trainer
run	Общедоступный	Выполняет обучение нейронной сети с обработкой ошибок

Метод __init__. Входные данные:

– dataset_path (тип str) – путь к датасету;

– model_path (тип str) – путь к директории сохранения модели;

– batch_size (тип int) – размер батча;

– epochs (тип int) – количество эпох обучения.

Возвращаемых данных нет.

Метод run не имеет входных и возвращаемых данных.

Описание класса TrainingWidget: Класс предоставляет графиеский интерфейс для выбора параметров обучения и отображения прогресса. Базовый класс – QWidget, стандартный класс PyQt5. Интерфейсы – поля ввода расположений датасета и сохранения результата, кнопки выбора путей, запуска обучения, шкала прогресса завершения обучения, текстовое поле для отображения сведений об обучении. Константы отсутствуют. Внутренние поля:

– dataset_label: QLabel – надпись «Папка датасета:»;

– dataset_path_field: QTextEdit – поле для отображения выбранной папки датасета;

– select_dataset_button: QPushButton – кнопка для выбора папки датасета;

– model_label: QLabel – надпись «Папка сохранения:»;

– model_path_field: QTextEdit – поле для отображения выбранной папки сохранения;

– select_model_button: QPushButton – кнопка для выбора папки сохранения;

– batch_size_label: QLabel – надпись «Размер батча:»;

– batch_size_combo: QComboBox – выпадающий список для выбора размера батча;

– epochs_label: QLabel – надпись «Количество эпох:»;

– epochs_field: QLineEdit – поле для ввода количества эаох обучения;

– progress_label: QLabel – надпись «Прогресс эпохи:»;

– progress_bar: QProgressBar – шкала прогресса обучения нейронной сети;

– log_label: QLabel – надпись «Процесс выполнения:»;

– output_text: QTextEdit – поле для отображения сведений о процессе обучения;

– train_button: QPushButton – кнопка для запуска обучения;

– thread: TrainingThread – поток для процесса обучения.

Методы класса представлены в таблице 4.6.

Таблица 4.7 – Методы класса TrainigWidget

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует графический интерфейс, включая макет и элементы для настройки обучения и отображения прогресса
select_dataset_folder	Общедоступный	Открывает диалоговое окно для выбора папки, содержащей датасет
select_model_path	Общедоступный	Открывает диалоговое окно для выбора папки сохранения весов обученной нейронной сети

Продолжение таблицы 4.7

Название метода	Область видимости	Назначение метода
run_training	Общедоступный	Проверяет корректность параметров обучения, заданных пользователем, отображает ошибки при наличии, создает поток обучения, подключает сигналы для обновления графического интерфейса
on_epoch_start	Общедоступный	Обновляет отображаемый журнал обучения нейронной сети записью о начале эпохи и сбрасывает шкалу прогресса при начале новой эпохи обучения
on_epoch_complete	Общедоступный	Обновляет отображаемый журнал информацией о завершении эпохи и заполняет шкалу прогресса по окончании эпохи обучения
on_batch_progress	Общедоступный	Обновляет шкалу прогресса в процессе обучения
append_output	Общедоступный	Добавляет текст в область отображения журнала
show_error	Общедоступный	Отображает диалоговое окно ошибки

Методы __init__, select_dataset_folder, select_model_path, run_training не имеют входных и возвращаемых данных.

Метод on_epoch_start. Входные данные:

– epoch (тип int) – номер текущей эпохи обучения;

– total_epochs (тип int) – общее количество эпох.

Возвращаемых данных нет.

Метод on_epoch_complete. Входные данные:

– epoch (тип int) – номер завершенной эпохи;

– avg_loss (тип float) – среднее значение потери за эпоху обучения.

Возвращаемых данных нет.

Метод on_batch_progress. Входные данные:

– batch (тип int) – номер текущего батча;

– total_batches (тип int) – общее количество батчей;

– loss (тип float) – значение потери для текущего батча.

Возвращаемых данных нет.

Метод append_output. Входные данные – text (тип str) – текст, добавляемый в журнал обучения. Возвращаемых данных нет.

Метод show_error. Входные данные – error_message (тип str) – отображаемое сообщение об ошибке. Возвращаемых данных нет.

4.1.1.8 Модуль test_ui.py

Модуль предоставляет графический интерфейс для тестирования модели нейронной сети.

Классы: TestingThread, TestingWidget.

Описание класса TestingThread: Класс выполняет оценку модели в отдельном потоке, чтобы избежать зависаний графического интерфейса. Базовый класс – QThread, стандартный класс PyQt5. Интерфейсы класса – сигналы для обновления графического интерфейса. Константы отсутствуют. Внутренние поля:

– image_path: str – путь к тестовому изображению;

– model_path: str – путь к файлу настроек модели нейронной сети;

– output_signal: pyqtSignal[str] – сигнал для сообщений о результатах тестирования;

– results_signal: pyqtSignal[dict] – сигнал для результатов оценки;

– error_signal: pyqtSignal[str] – сигнал для сообщений об ошибках.

Методы класса представлены в таблице 4.8

Таблица 4.8 – Методы класса TestingThread

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует поток, передавая в него пути к тестовому изображению и папке модели
run	Общедоступный	Выполняет оценку нейронной сети, отправляя результаты или ошибки через сигналы

Метод __init__. Входные данные:

– image_path (тип str) – путь к тестовому изображению;

– model_path (тип str) – путь к файлу параметров нейронной сети.

Возвращаемых данных нет.

Метод run не имеет входных и возвращаемых данных.

Описание класса TestingWidget: Класс содержит графический интерфейс для выбора тестового изображения и файла весов модели, запуска оценки и отображения полученных метрик точности нейронной сети, целевой и предсказанной масок. Базовый класс – QWidget, стандартный класс PyQt5. Интерфейсы класса – поля для отображения путей к тестовому изображению и файлу весов, кнопки для выбора этих файлов и запуска тестирования, области для отображения изображений и метрик. Константы отсутствуют. Внутренние поля:

– image_path: str – путь к тестовому изображению;

– model_path: str – путь к тестируемым параметрам модели;

– path_label: QLabel – надпись «Изображение для проверки модели:»;

– image_path_field: QTextEdit – поле для отображения пути к выбранному изображению;

– select_image_button: QPushButton – кнопка для выбора изображения;

– model_label: QLabel – надпись «Путь к параметрам нейросети:»;

– model_path_field: QTextEdit – поле для отображения пути к выбранному файлу параметров нейронной сети;

– select_model_button: QPushButton – кнопка для выбора модели;

– output_text: QTextEdit – поле для отображения метрик точности нейронной сети;

– input_text_label: QLabel – надпись «Исходное изображение»;

– input_image_label: QLabel – поле для отображения изображения;

– gt_text_label: QLabel – надпись «Ожидаемый результат»;

– gt_mask_label: QLabel – поле для отображения целевой маски;

– pred_text_label: QLabel – надпись «Результат работы модели»;

– pred_mask_label: QLabel – поле для отображения предсказанной маски;

– test_button: QPushButton – кнопка для запуска тестирования;

– thread: TestingThread – поток для выполнения тестирования и оценки.

Методы класса представлены в таблице 4.9

Таблица 4.9 – Методы класса TestingWidget

Название метода	Область видимости	Назначение метода
__init__	Общедоступный	Инициализирует графический интерфейс, включая макет и элементы для тестирования и отображения результатов
select_image	Общедоступный	Открывает диалоговое окно для выбора тестового изображения и добавляет путь в специальное поле

Продолжение таблицы 4.9

Название метода	Область видимости	Назначение метода
select_model	Общедоступный	Открывает диалоговое окно для выбора файла весов нейронной сети и добавляет путь в специальное поле
run_testing	Общедоступный	Проверяет входные данные, запускает поток тестирования и подключает сигналы для отображения результатов и ошибок
show_results	Общедоступный	Отображает исходное изображения и результаты тестирования в графическом интерфейсе
append_output	Общедоступный	Добавляет текст в область отображения результатов тестирования
show_error	Общедоступный	Отображает диалоговое окно ошибки
numpy_to_qimage	Внутренний	Преобразует маски результатов из массивов в изображения

Методы __init__, select_image, select_model, run_testing не имеют входных и возвращаемых значений.

Метод show_results. Входное значение – results (тип dict) – словарь с результатами тестирования нейронной сети. Возвращаемых данных не имеет.

Метод append_output. Вохдное значение – text (тип str) – текст, отображаемый в области результатов тестирования. Возвращаемых данных не имеет.

Метод show_error. Входное значение – error_message (тип str) – отображаемое сообщение об ошибке. Возвращаемых данных не имеет.

Метод numpy_to_qimage. Входные значения:

– np_array (тип np.ndarray) – массив, представляющий изображение;

– is_grayscale (тип bool, значение по умолчанию – True) – флаг, определяющий, является ли изображение полутоновым.

Возвращаемое значение – преобразованное изображение в формате QImage.

4.2 Модульное тестирование разработанной программной системы

Модульный тест для класса UNet из модуля model.py представлен на рисунке 4.1.

```
import unittest
import torch
from model import UNet

class TestUNet(unittest.TestCase):
  def test_model_initialization(self):
    model = UNet(in_channels=1, out_channels=1)
    self.assertIsInstance(model, UNet, "Должен создаваться экземпляр UNet")

  def test_forward_pass(self):
    model = UNet(in_channels=1, out_channels=1)
    input_tensor = torch.randn(1, 1, 320, 624)
    output = model(input_tensor)
    self.assertEqual(output.shape, (1, 1, 320, 624), "Некорректный размер
        вывода")
    self.assertTrue(torch.all(output >= 0), "Выход должен быть ≥ 0")
    self.assertTrue(torch.all(output <= 1), "Выход должен быть ≤ 1")
```

Рисунок 4.1 – Модульный тест класса UNet

Результат тестирования:

```
..
----------------------------------------------------------------------
Ran 2 tests in 2.025s

OK
```

Рисунок 4.2 – Результат тестирования класса UNet

Модульный тест для вычисления метрик модуля test.py представлен на рисунке 4.3.

```python
import unittest
import torch
from test import dice_coefficient, iou_score

class TestMetrics(unittest.TestCase):
  def test_dice_coefficient(self):
    pred = torch.tensor([1, 1, 0, 0], dtype=torch.float32)
    target = torch.tensor([1, 0, 1, 0], dtype=torch.float32)
    dice = dice_coefficient(pred, target)
    self.assertTrue(0 <= dice <= 1, "Dice должен быть в [0, 1]")

  def test_iou_score(self):
    pred = torch.tensor([1, 1, 0, 0], dtype=torch.float32)
    target = torch.tensor([1, 0, 1, 0], dtype=torch.float32)
    iou = iou_score(pred, target)
    self.assertTrue(0 <= iou <= 1, "IoU должен быть в [0, 1]")
```

Рисунок 4.3 – Модульный тест функций подсчета метрик модуля test.py

Результат тестирования:

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.244s

OK
```

Рисунок 4.4 – Результат тестирования функций подсчета метрик модуля test.py

Модульный тест для класса analyze_return из модуля detect.py представлен на рисунке 4.5.

```python
import unittest
import numpy as np
import cv2
import os
import torch
from unittest.mock import patch
from detect import analyze_return

  class TestDetect(unittest.TestCase):
    def setUp(self):
      self.test_img = np.random.randint(0, 255, (100, 100, 3), dtype=np.uint8
          )
      self.img_path = "test_image.jpg"
      cv2.imwrite(self.img_path, self.test_img)

    def tearDown(self):
      if os.path.exists(self.img_path):
      os.remove(self.img_path)

    @patch("detect.load_model")
    def test_analyze_save(self, mock_load_model):
      mock_model = mock_load_model.return_value
      mock_model.return_value = torch.sigmoid(torch.randn(1, 1, 320, 624))


      result = analyze_return(self.img_path, "dummy_model.pth")
      self.assertIsInstance(result, np.ndarray, "Должен возвращаться numpy-
          массив")
      self.assertEqual(result.shape[2], 3, "Изображение должно быть 3-
          канальным (BGR)")
```

Рисунок 4.5 – Модульный тест класса analyze_return

Результат тестирования:

```
.
----------------------------------------------------------------------
Ran 1 test in 1.116s

OK
```

Рисунок 4.6 – Результат тестирования класса analyze_return

4.3 Системное тестирование разработанной программной системы

Для проведения системного тестирования был использован файл весов, полученный после обучения нейронной сети на датасете, состоящем из 777 изображений. Обучение проводилось на протяжении после 25 эпох.

На рисунке 4.7 представлено окно режима работы «Анализ изображения».

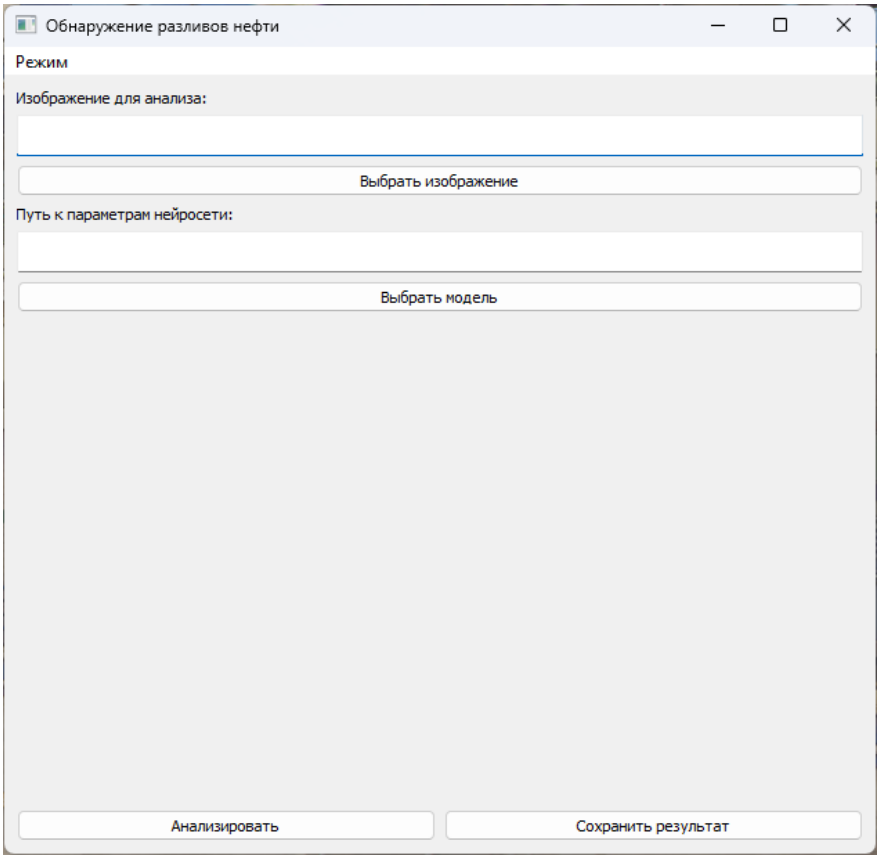

Рисунок 4.7 – Окно программы в режиме «Анализ изображения»

На рисунке 4.8 представлено диалоговое окно выбора анализируемого изображения.

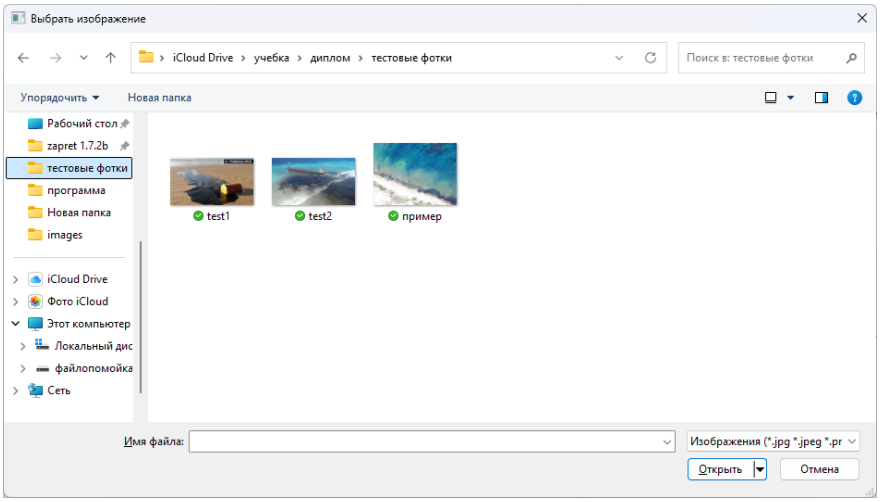

Рисунок 4.8 – Диалоговое окно выбора анализируемого изображения

На рисунке 4.9 представлено диалоговое окно выбора файла весов модели нейронной сети для анализа.

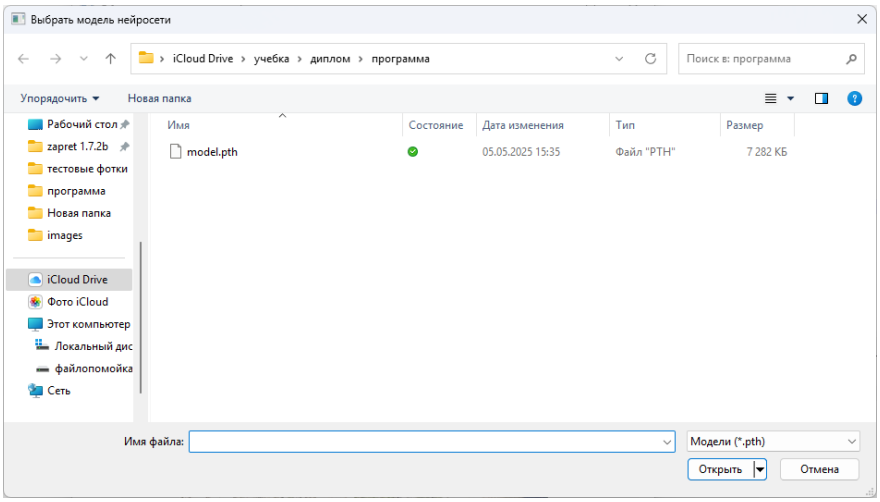

Рисунок 4.9 – Диалоговое окно выбора файла весов

На рисунке 4.10 представлено отображение результата распознавания нефтяных пятен на поверхности водоемов.

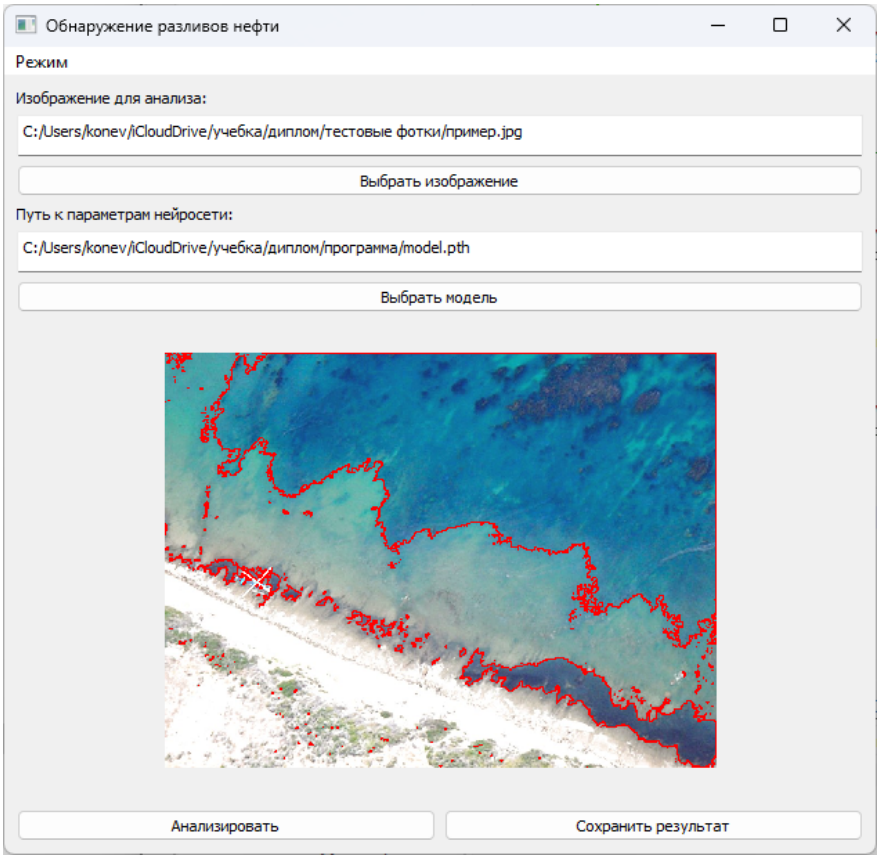

Рисунок 4.10 – Результат распознавания нефтяных пятен

На рисунках 4.11 и 4.12 показано сохранение полученного нейросетью результата.

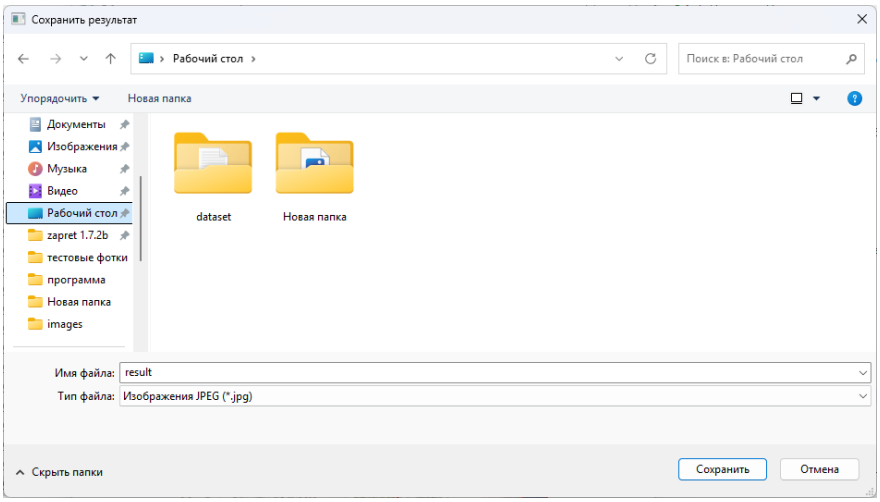

Рисунок 4.11 – Диалоговое окно выбора расположения сохранения

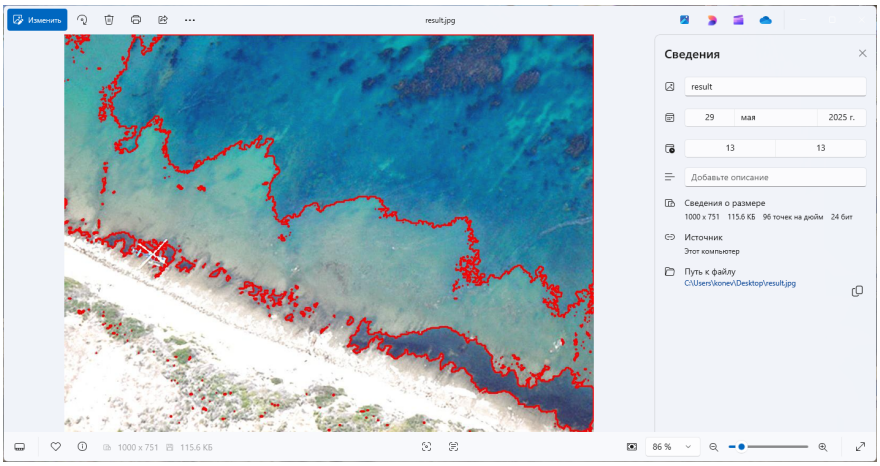

Рисунок 4.12 – Сохраненный результат

На рисунке 4.13 представлено окно режима работы «Обучение».

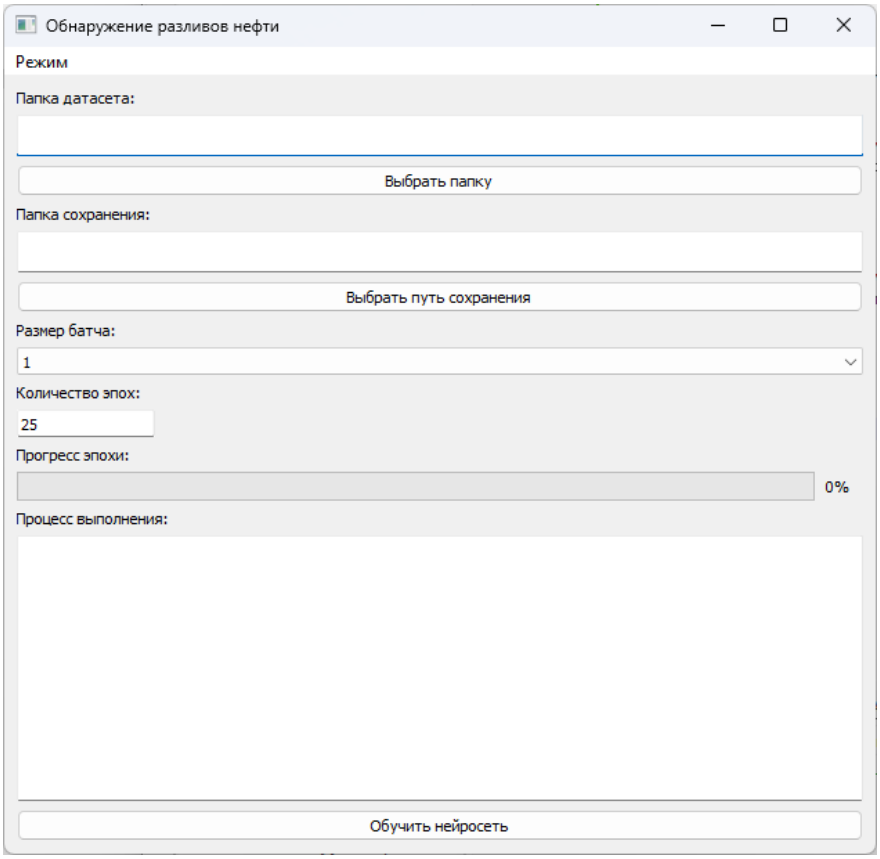

Рисунок 4.13 – Окно режима«Обучение»

На рисунке 4.14 представлено диалоговое окно выбора датасета.

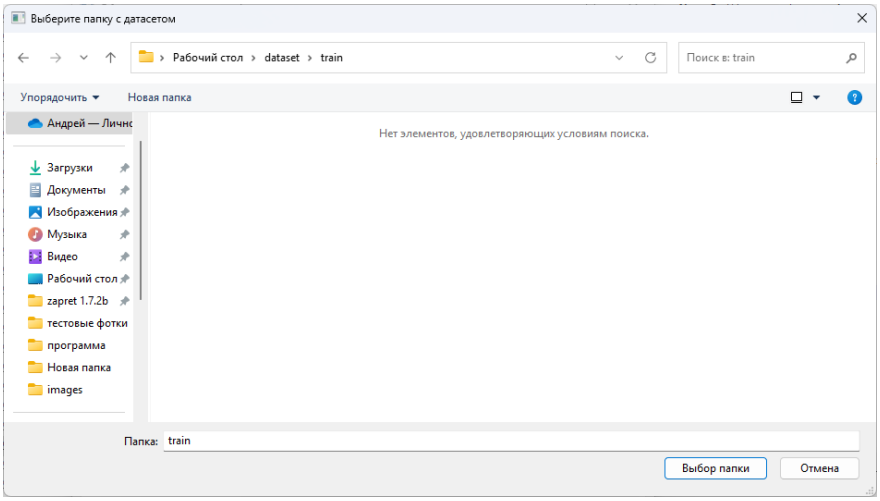

Рисунок 4.14 – Диалоговое окно выбора датасета

На рисунке 4.15 представлено диалоговое окно выбора расположения сохранения весов.

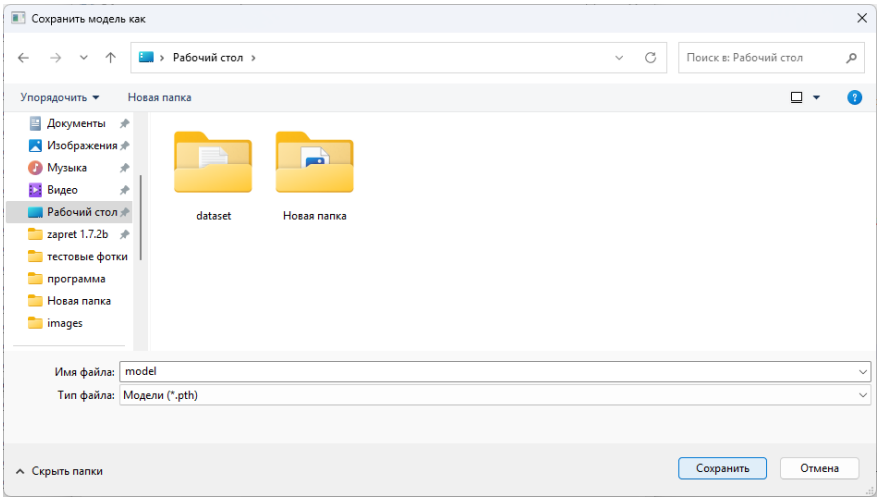

Рисунок 4.15 – Диалоговое окно выбора расположения сохранения весов

На рисунках 4.16, 4.17 и 4.18 отображены процесс обучения, интерфейс программы после завершения обучения и полученный файл весов модели нейронной сети.

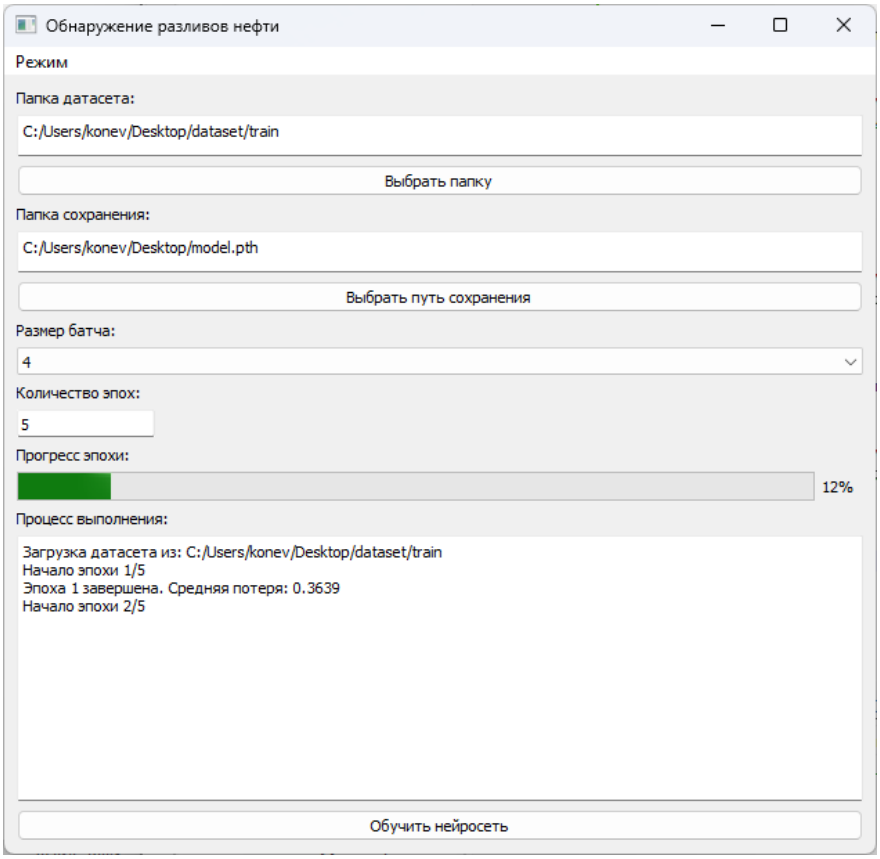

Рисунок 4.16 – Процесс обучения нейронной сети

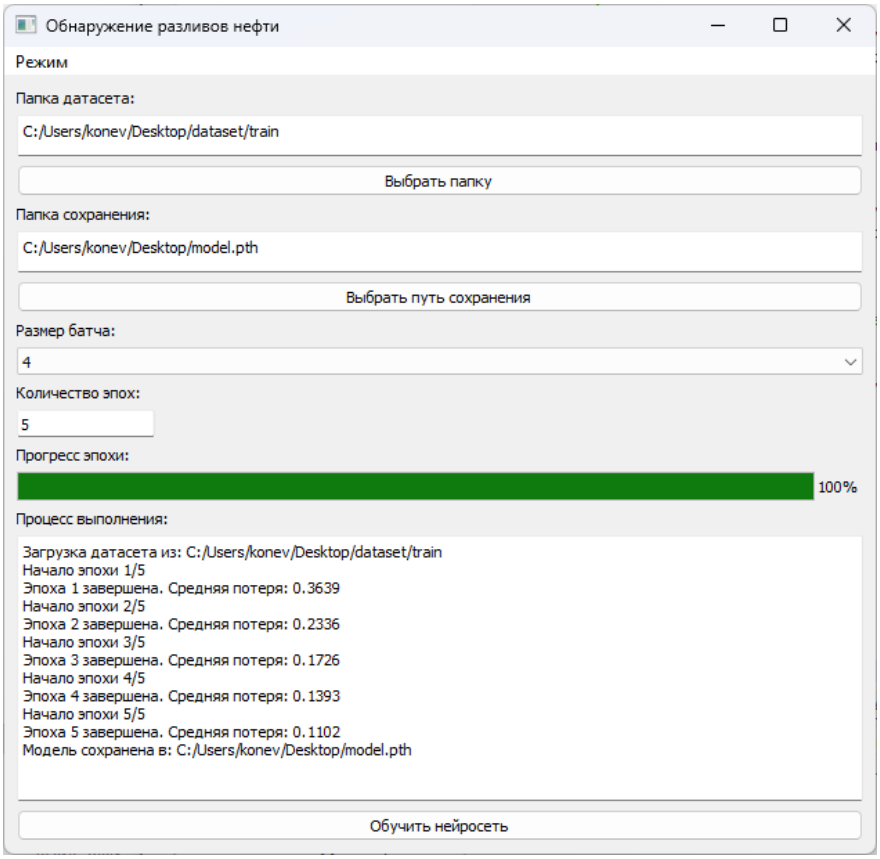

Рисунок 4.17 – Интерфейс программы после завершения обучения

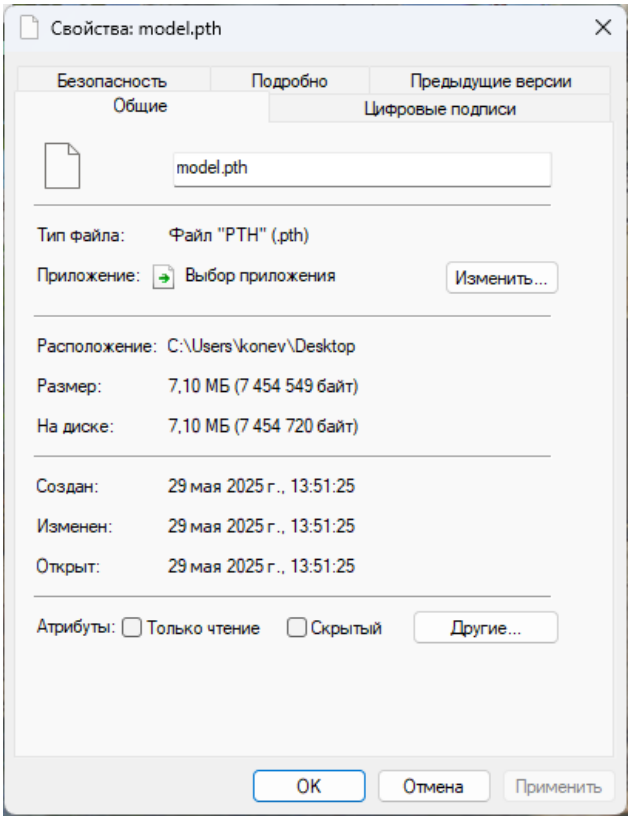

Рисунок 4.18 – Файл весов модели

На рисунке 4.19 изображено окно программы в режиме «Тестирование».

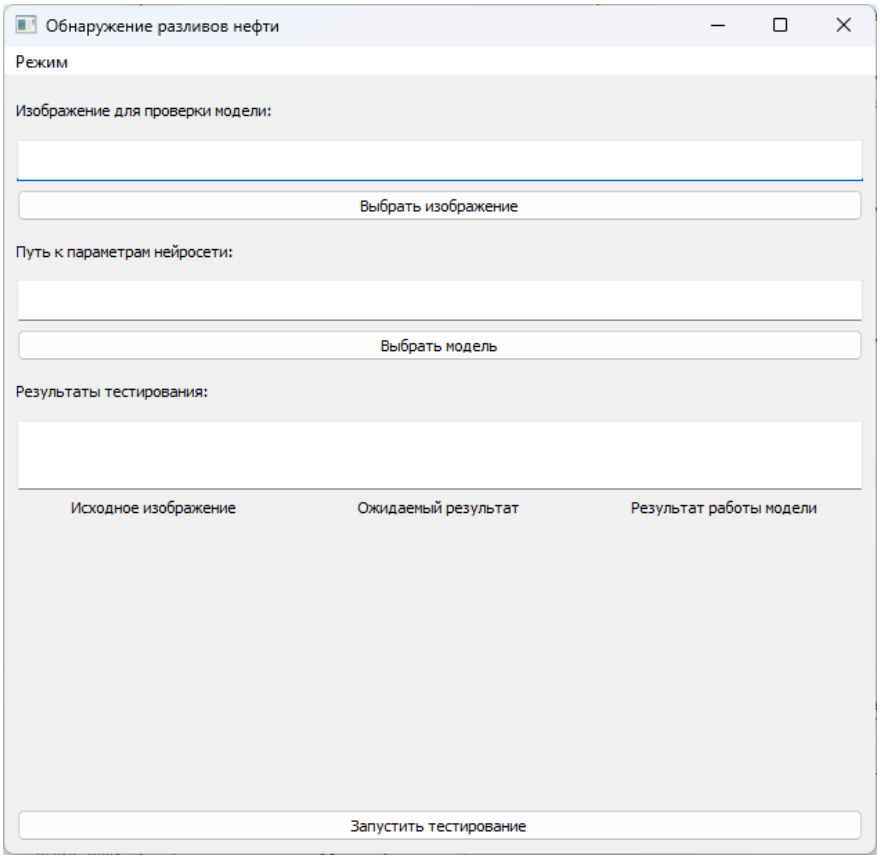

Рисунок 4.19 – Окно режима «Тестирование»

На рисунке 4.20 изображено диалоговое окно выбора тестового изображения.

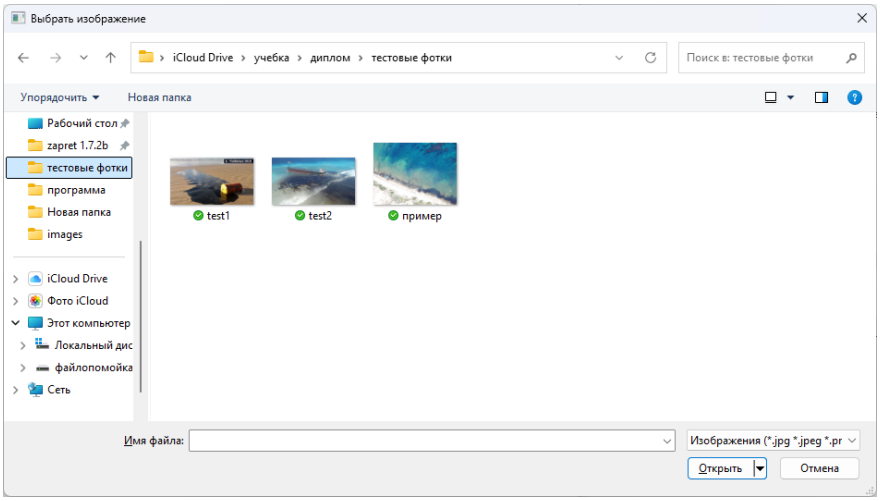

Рисунок 4.20 – Диалоговое окно выбора тестового изображения

На рисунке 4.21 изображено диалоговое окно выбора тестируемых весов.

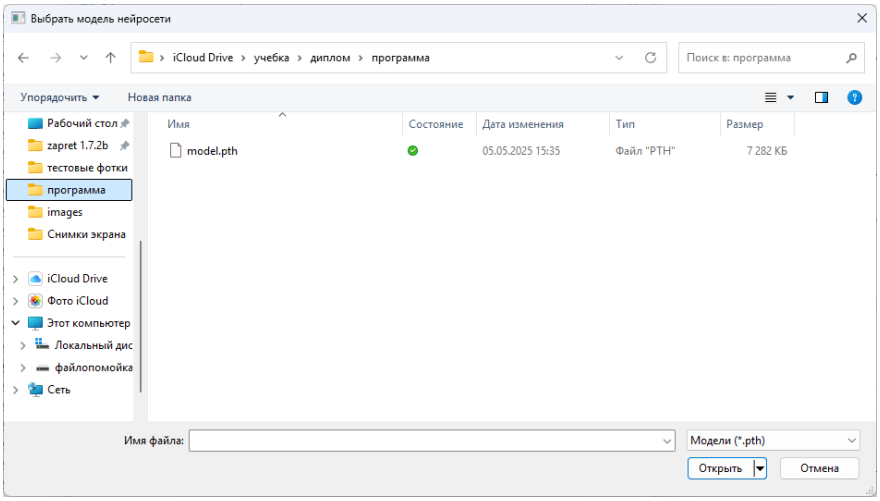

Рисунок 4.21 – Диалоговое окно выбора тестируемой модели

На рисунке 4.22 отображены результаты тестирования выбранных весов модели нейронной сети.

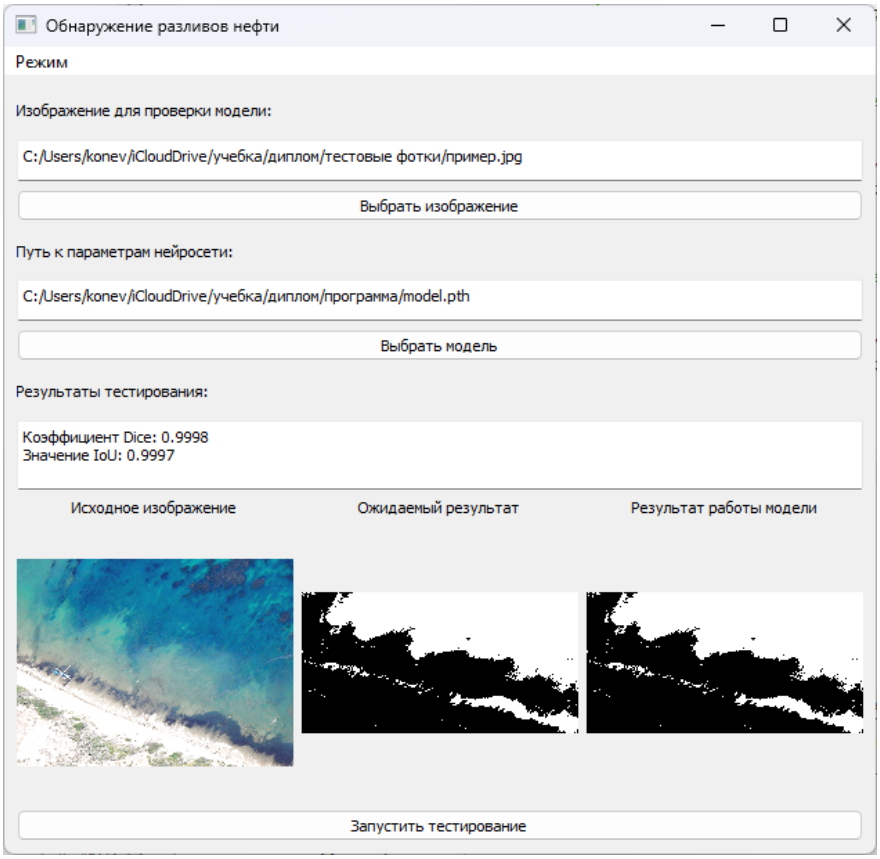

Рисунок 4.22 – Результаты тестирования

4.4 Сборка программной системы

Программные компоненты представляют собой файлы исходных кодов программной системы.

Для сборки и компиляции программной системы использовалась библиотека Pyinstaller[20], позволяющая упаковать все необходимые файлы в один исполняемый файл формата .exe. Данный файл может быть запущен без предварительной установки.

Интерпретация исходных кодов на языке Python выполняется встроенным в исполняемый файл интерпретатором языка и не требует отдельной установки интерпретатора и библиотек на целевую систему.

Все программные компоненты собраны в один исполняемый файл, готовый к запуску в среде Windows.

ЗАКЛЮЧЕНИЕ

Развитие нейронных сетей и методов машинного обучения открыло широкие возможности для автоматизации задач анализа изображений. Современные технологии позволяют обрабатывать большие объемы визуальных данных и своевременно выявлять потенциальные угрозы окружающей среде.

В условиях роста количества загрязнений водоемов, вызванных разливами нефти, применение интеллектуальных систем является эффективным инструментом мониторинга. Их использование позволяет значительно облегчить процесс обнаружения и снизить его стоимость, увеличивая при этом скорость применения.

Для решения задачи распознавания пятен нефтяных разливов была разработана интеллектуальная система на основе сверточной нейронной сети архитектуры U-Net. Система реализована в виде настольного приложения с графическим интерфейсом.

Основные результаты работы:

1. Проведен анализ предметной области. Проведено исследование причин возникновения разливов и нейронных сетей, являющихся наиболее эффективным методом автоматического распознавания.

2. Разработана концептуальная модель интеллектуальной системы, определены основные требования к системе и аппаратному обеспечению.

3. Осуществлено проектирование интеллектуальной системы. Разработана архитектура настольного приложения и нейронной сети. Разработан пользовательский интерфейс приложения.

4. Реализована интеллектуальная система, проведено модульное и системное тестирование разработанной нейронной сети и настольного приложения.

Все требования, объявленные в техническом задании, были полностью реализованы. Все задачи, поставленные в начале разработки проекта, были решены.

Готовый рабочий проект представлен в виде настольного приложения с графическим интерфейсом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алексеев Д. В. Сравнительный анализ баз данных по разливам нефти и нефтепродуктов с морских судов / Д. В. Алексеев, А. А. Лентарёв // Вестник Государственного университета морского и речного флота имени адмирала С. О. Макарова. — 2022. — Т. 14. — № 6. — С. 891–904. DOI: 10.21821/2309-5180-2022-14-6-891-904. — Текст: непосредственный.

2. Владимиров В. А. Разливы нефти: причины, масштабы, последствия // Стратегия гражданской защиты: проблемы и исследования. — 2014. — № 1. — URL: https://cyberleninka.ru/article/n/razlivy-nefti-prichiny-masshtaby-posledstviya (дата обращения: 10.05.2025). – Текст : непосредственный.

3. Клименко С. К., Иванов А. Ю., Терелева Н. В. Пленочные загрязнения Керченского пролива по данным пятилетнего радиолокационного мониторинга: современное состояние и основные источники // Исследование Земли из космоса. –2022. – № 3. – С. 37–54. — DOI: 10.31857/S0205961422030071. – Текст : непосредственный.

4. Горбачевская Е. Н., Краснов С. С. История развития нейронных сетей // Вестник ВУиТ. — 2015. — № 1 (23). — URL: https://cyberleninka.ru/article/n/istoriya-razvitiya-neyronnyh-setey (дата обращения: 11.05.2025). – Текст : непосредственный.

5. Митина О. А., Ломовцев П. П. Перцептрон в задачах бинарной классификации // НАУ. — 2021. — № 66-1. — URL: https://cyberleninka.ru/article/n/pertseptron-v-zadachah-binarnoy-klassifikatsii (дата обращения: 11.05.2025). – Текст : непосредственный.

6. Фаулер М. UML. Основы. — 3-е изд. — СПб.: Символ-Плюс, 2015. — 192 с. — ISBN 978-5-93286-060-1. — Текст: непосредственный.

7. Халяпин Д. Б. UML. Проектирование систем реального времени, параллельных и распределенных приложений / Д. Б. Халяпин. — Москва: Озон, 2023. — 352 с. — ISBN 978-5-6048804-3-2. — Текст: непосредственный.

8. Мюллер-Брокманн Й. Модульные системы в графическом дизайне. — М.: Студия Артемия Лебедева, 2018. — 176 с. — ISBN 978-5-98062-140-7. — Текст: непосредственный.

9. Хайкин С. Нейронные сети: полный курс / Саймон Хайкин; пер. с англ. Н. Н. Куссуль. — 2-е изд., испр. — М.: Вильямс, 2018. — 1103 с. — ISBN 978-5-907144-22-4. — Текст: непосредственный.

10. Ростовцев В. С. Искусственные нейронные сети: учебник. — 5-е изд., стер. — СПб.: Лань, 2025. — 216 с. — ISBN 978-5-507-50568-5. — Текст: непосредственный.

11. Шолле Ф. Глубокое обучение на Python / пер. с англ. — М.: ДМК Пресс, 2018. — 384 с. — ISBN 978-5-4461-0770-4. — Текст: непосредственный.

12. Лутц М. Изучаем Python. Том 1: учебное пособие / М. Лутц. — 5-е изд. — М.: Вильямс, 2020. — 832 с. — ISBN 978-5-907144-52-1. — Текст: непосредственный.

13. Лутц М. Изучаем Python. Том 2 / М. Лутц. — 5-е изд. — М.: Вильямс, 2020. — 720 с. — ISBN 978-5-907144-53-8. — Текст: непосредственный.

14. Маккинни У. Python и анализ данных. — 2-е изд. — М.: ДМК Пресс, 2019. — 544 с. — ISBN 978-5-97060-590-5. — Текст: непосредственный.

15. Бендер Д. Python для анализа данных. — М.: ДМК Пресс, 2015. — 482 с. — ISBN 978-5-97060-315-4. — Текст: непосредственный.

16. Пойнтер Я. Программируем с PyTorch: создание приложений глубокого обучения. — СПб.: Питер, 2020. — 256 с. — ISBN 978-5-4461-1677-5. — Текст: непосредственный.

17. Бычков А. Г., Киселёва Т. В., Маслова Е. В. Использование сверточных нейросетей для классификации изображений // Вестник Сибирского государственного индустриального университета. — 2022. — № 1. — С. 15–19. — Текст: непосредственный.

18. Годунов А. И., Баланян С. Т., Егоров П. С. Сегментация изображений и распознавание объектов на основе технологии сверточных нейронных се-

тей // Надежность и качество сложных систем. — 2021. — № 3. — С. 71–73. — Текст: непосредственный.

19. Сорокин А. Б., Железняк Л. М., Зикеева Е. А. Сверточные нейронные сети: примеры реализаций: учебное пособие. — М.: МИРЭА, 2020. — 1 электрон. опт. диск (CD-ROM). — Текст: непосредственный.

20. Васильев А. Н. Программирование на Python в примерах и задачах. — М.: Бомбора, 2021. — 384 с. — ISBN 978-5-04-103199-2. — Текст: непосредственный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.4.

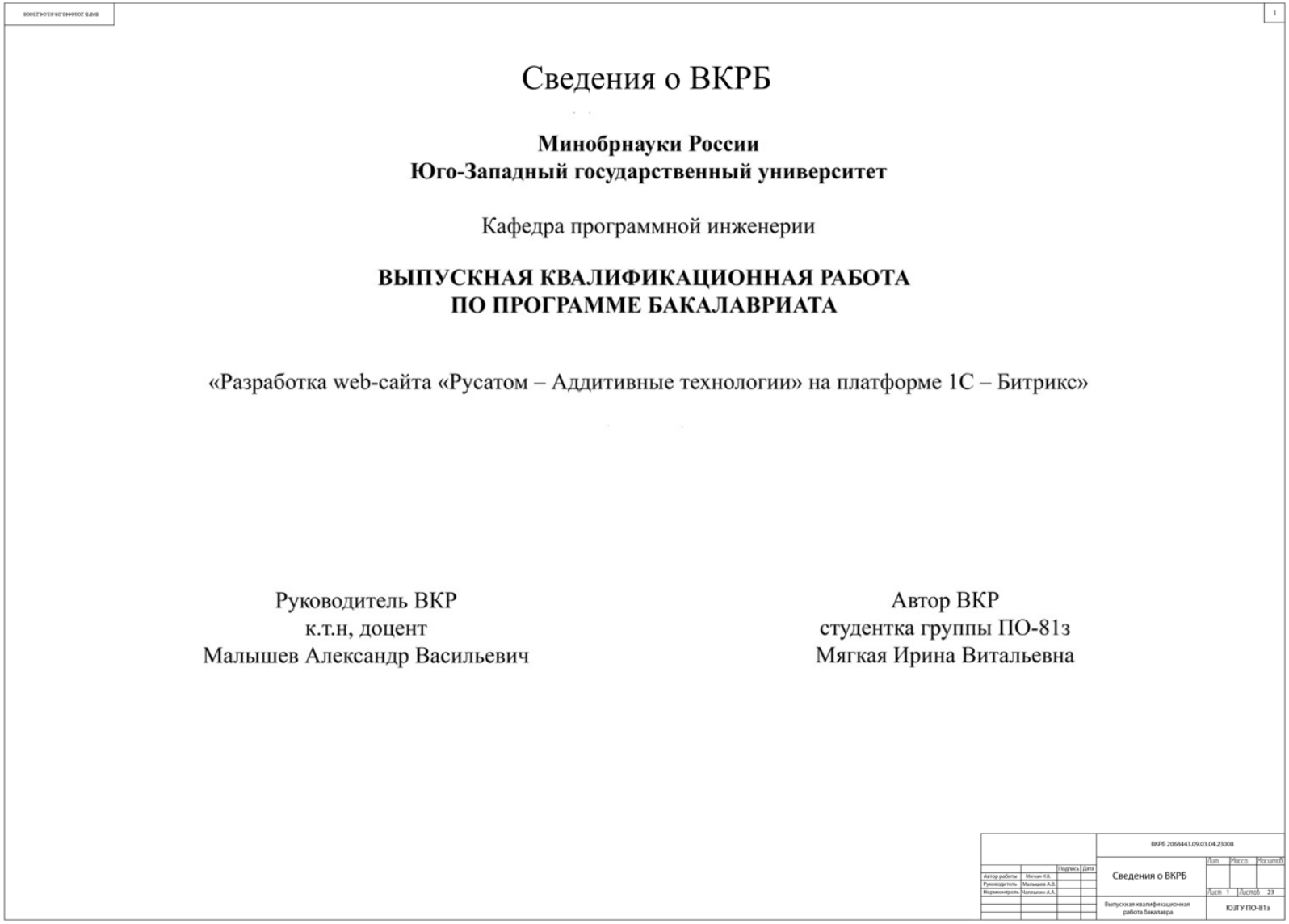

Рисунок А.1 – Сведения о ВКРБ

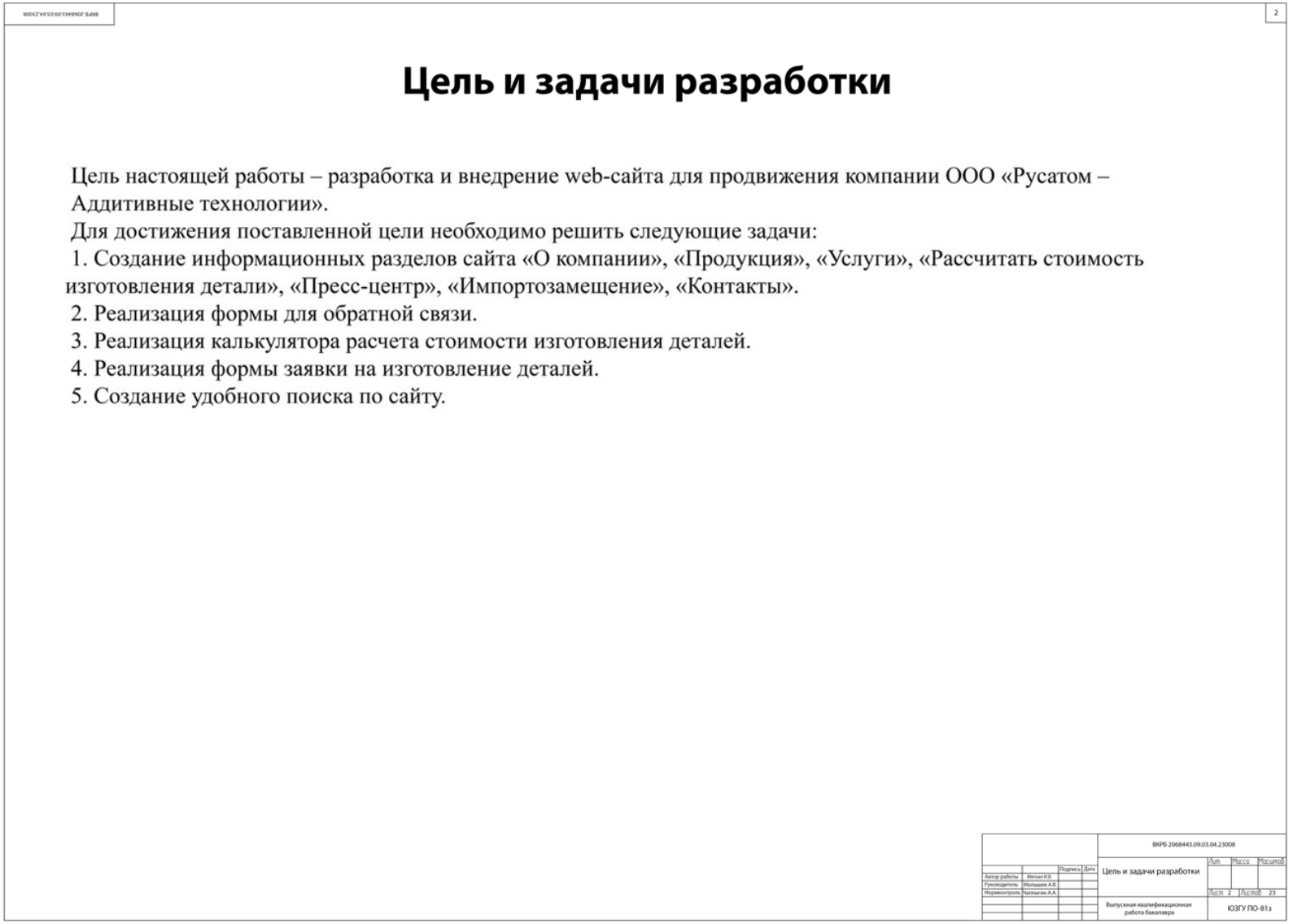

Рисунок А.2 – Цель и задачи разработки

Рисунок А.3 – Концептуальная модель сайта

Рисунок А.4 – Еще плакат

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

model.py

```python
import torch
import torch.nn as nn

class UNet(nn.Module):
    def __init__(self, in_channels=1, out_channels=1):
        super(UNet, self).__init__()

        def conv_block(in_c, out_c):
            return nn.Sequential(
                nn.Conv2d(in_c, out_c, 3, padding=1),
                nn.ReLU(inplace=True),
                nn.Conv2d(out_c, out_c, 3, padding=1),
                nn.ReLU(inplace=True)
            )

        self.enc1 = conv_block(in_channels, 64)
        self.enc2 = conv_block(64, 128)
        self.pool = nn.MaxPool2d(2, 2)

        self.bottleneck = conv_block(128, 256)

        self.upconv2 = nn.ConvTranspose2d(256, 128, 2, 2)
        self.dec2    = conv_block(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, 2, 2)
        self.dec1    = conv_block(128, 64)

        self.final_conv = nn.Conv2d(64, out_channels, 1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        b  = self.bottleneck(self.pool(e2))
        d2 = self.upconv2(b)
        d2 = torch.cat([d2, e2], dim=1)
        d2 = self.dec2(d2)
        d1 = self.upconv1(d2)
        d1 = torch.cat([d1, e1], dim=1)
        d1 = self.dec1(d1)
        return torch.sigmoid(self.final_conv(d1))
```

train.py

```python
import torch
from torch.utils.data import Dataset, DataLoader
import os
import numpy as np
from model import UNet
import argparse
from PyQt5.QtCore import QObject, pyqtSignal
```

```
from PIL import Image

class dataset(Dataset):
    def __init__(self, image_dir, threshold=0.5):
        self.image_dir = image_dir
        self.threshold = threshold
        self.images = [
            img for img in os.listdir(image_dir)
            if img.lower().endswith(('.jpg', '.jpeg'))
        ]

        if not self.images:
            raise ValueError(f"Папка {image_dir} не содержит изображений
               формата .jpg или .jpeg")

        for img_name in self.images:
            img_path = os.path.join(image_dir, img_name)
            try:
                with Image.open(img_path) as img:
                    img.verify()
            except Exception as e:
                raise ValueError(f"Папка содержит поврежденные изображения")

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        path = os.path.join(self.image_dir, self.images[idx])
        img = Image.open(path).convert("L")
        if img is None:
            raise ValueError(f"Ошибка загрузки изображения: {path}")
        img = img.resize((624, 320))
        img = np.array(img).astype(np.float32) / 255.0
        mask = (img < self.threshold).astype(np.float32)
        img = np.expand_dims(img, axis=0)
        mask = np.expand_dims(mask, axis=0)
        return torch.tensor(img), torch.tensor(mask)

class Trainer(QObject):
    epoch_start_signal = pyqtSignal(int, int)
    epoch_complete_signal = pyqtSignal(int, float)
    batch_progress_signal = pyqtSignal(int, int, float)
    training_complete_signal = pyqtSignal(str)

    def __init__(self, image_dir, save_path, batch_size=1, epochs=25, lr=1e
       -4, threshold=0.5):
        super().__init__()
        self.image_dir = image_dir
        self.save_path = save_path
        self.batch_size = batch_size
        self.epochs = epochs
        self.lr = lr
        self.threshold = threshold

```

```
    def run(self):
        self.training_complete_signal.emit(f"Загрузка датасета из: {self.image_dir}")
        train_dataset = dataset(self.image_dir, threshold=self.threshold)
        dataloader = DataLoader(train_dataset, batch_size=self.batch_size, shuffle=True)

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        model = UNet().to(device)
        criterion = torch.nn.BCELoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=self.lr)

        for epoch in range(self.epochs):
            total_loss = 0
            self.epoch_start_signal.emit(epoch + 1, self.epochs)
            for i, (imgs, masks) in enumerate(dataloader, 1):
                imgs, masks = imgs.to(device), masks.to(device)
                preds = model(imgs)
                loss = criterion(preds, masks)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                total_loss += loss.item()
                self.batch_progress_signal.emit(i, len(dataloader), loss.item())

            avg_loss = total_loss / len(dataloader)
            self.epoch_complete_signal.emit(epoch + 1, avg_loss)

        torch.save(model.state_dict(), self.save_path)
        self.training_complete_signal.emit(f"Модель сохранена в: {self.save_path}")

def main():
    parser = argparse.ArgumentParser(description="Обучение нейросети UNet")
    parser.add_argument('--data', required=True, help="Путь к папке с изображениями")
    parser.add_argument('--output', required=True, help="Путь для сохранения модели (model.pth)")
    parser.add_argument('--batch-size', type=int, default=4, help="Размер батча (по умолчанию: 4)")
    parser.add_argument('--epochs', type=int, default=25, help="Количество эпох (по умолчанию: 25)")
    args = parser.parse_args()

    trainer = Trainer(args.data, args.output, batch_size=args.batch_size, epochs=args.epochs)
    trainer.run()

if __name__ == "__main__":
    main()
```

detect.py

```
import torch
```

```
import numpy as np
import cv2
from model import UNet
from PIL import Image
import io

DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def load_model(model_path):
    model = UNet()
    model.load_state_dict(torch.load(model_path, map_location=DEVICE))
    model.to(DEVICE)
    model.eval()
    return model

def analyze_return(image_path, model_path, threshold=0.5):
    model = load_model(model_path)

    with open(image_path, "rb") as f:
        pil_img = Image.open(io.BytesIO(f.read()))
        pil_img = pil_img.convert("RGB")
        original_img = np.array(pil_img)[:, :, ::-1]
        img_gray = np.array(pil_img.convert("L"))

    orig_h, original_w = img_gray.shape
    img_norm = img_gray.astype(np.float32) / 255.0
    img_resized = cv2.resize(img_norm, (624, 320))
    img_tensor = torch.tensor(np.expand_dims(img_resized, axis=(0, 1)), dtype
       =torch.float32).to(DEVICE)

    with torch.no_grad():
        pred = model(img_tensor)
        if not torch.is_floating_point(pred):
            raise ValueError("Model output is not a floating-point tensor")

    mask = (pred.squeeze().cpu().numpy() > float(threshold)).astype(np.uint8)
    mask = cv2.resize(mask, (original_w, orig_h))

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
       CHAIN_APPROX_SIMPLE)
    result = original_img.copy()
    cv2.drawContours(result, contours, -1, (0, 0, 255), 2)

    return result
```

Автор ВКР		А. В. Конев
	(подпись, дата)	
Руководитель ВКР		Р. А. Томакова
	(подпись, дата)	
Нормоконтроль		А. А. Чаплыгин
	(подпись, дата)	

Место для диска