

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ИСПОЛЬЗОВАНИЕ ИСКУССТВЕННЫХ
НЕЙРОННЫХ СЕТЕЙ ДЛЯ РЕШЕНИЯ ЗАДАЧ
ПОЗИЦИОНИРОВАНИЯ ОБЪЕКТОВ НА ЦИФРОВЫХ
КАРТАХ

Студент

Экимов Дмитрий Витальевич
(подпись, дата) (ФИО полностью)

Руководитель

Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

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

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

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

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
25 июня 2025 г. № 17)

Секретарь ГЭК

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

Пенза 2025

УТВЕРЖДАЮ

Зав. кафедрой ВТ

«29» 09

М.А. Митрохин

2023 г.

ЗАДАНИЕ

на выполнение магистерской работы

1. Студент гр. 23ВВим1 факультета ВТ направление 09.04.01
Экимов Дмитрий Витальевич

(фамилия, имя, отчество)

2. Руководитель ВКР доцент кафедры МОиПЭВМ, ПГУ

(Должность и место работы)

Такташкин Денис Витальевич

(фамилия, имя, отчество)

3. Время выполнения магистерской работы с _____ по _____

4. Тема магистерской работы Использование искусственных нейронных сетей для решения
задач позиционирования объектов на цифровых картах

Тема утверждена приказом ПГУ № 986/п от 29.09.2023г.

5. Задание на магистерскую работу: предмет и назначение разработки, функциональные, программные, технические требования и т.п.

5.1. Разрабатываемый модуль предназначен для позиционирования объектов на цифровых картах с использованием искусственных нейронных сетей.

5.2 Требования к функциональности: Модуль позволяет:

система должна предоставлять доступ к картам с помощью которых искусственная нейронная сеть разместит объекты на цифровой карте и предоставит доступ пользователю.

пользователь должен иметь возможность самостоятельно размещать точку интереса вокруг которой будут позиционироваться объекты, а также возможность скачивать результат работы.

программный комплекс должен предоставлять корректные данные и реагировать на возникающие ошибки в работе веб-интерфейса или искусственной нейронной сети,

сгенерированные изображения должны экспортироваться архивом содержащим файлы в формате PNG,

интерфейс должен быть понятен пользователям без глубоких технических знаний.

5.3 Требования к ИО: В качестве испытательного оборудования используются изображения как входные данные, поступающие от пользователей через веб-интерфейс и API системы.

5.4 Требования к ПО: Python, Flask 2.3.3, TensorFlow 2.13.0, OpenCV 4.8.1, LabelMe3, Yandex Static Maps API, Visual Studio Code, U-Net.

5.5 Требования к аппаратному обеспечению: Операционная система Microsoft Windows 10 и выше,

6. Содержание пояснительной записки магистерской работы: перечень подлежащих разработке, расчетов, обоснований, описаний

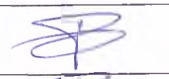
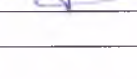
6.1 Анализ предметной области и аналогов, техническое задание

6.2 Проектирование программных средств

6.3 Разработка программных средств

6.4 Анализ результатов

7. Календарный график выполнения магистерской работы

Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1. Анализ предметной области и аналогов, техническое задание	15 %	25.11.2023	
2. Проектирование программных средств	25 %	25.04.2024	
3. Разработка ПО	35%	25.07.2024	
4. Анализ результатов разработки	15%	25.03.2025	
5. Оформление ПЗ	10 %	30.05.2025	
6. Нормоконтроль		31.05.2025	

Дата выдачи задания

«28» 09 2023г.

Руководитель магистерской работы



Такташкин Д.В.

(фамилия, имя, отчество)

Задание к исполнению принял



«28» 09. 2023г.

Экимов Д.В.

Студент

(фамилия, имя, отчество)

Магистерскую работу к защите допустить

Декан ФВТ



/ Л.Р. Фионова

(Подпись, дата)

27.06.2025

РЕФЕРАТ

Пояснительная записка содержит 135 листов, 19 рисунков, 4 таблицы, 25 источников.

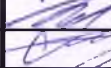
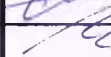
ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, СЕМАНТИЧЕСКАЯ СЕГМЕНТАЦИЯ, ЦИФРОВЫЕ КАРТЫ, ПОЗИЦИОНИРОВАНИЕ ОБЪЕКТОВ, U-NET, ГЕОИНФОРМАЦИОННЫЕ СИСТЕМЫ, ОБНАРУЖЕНИЕ ОБЪЕКТОВ, ГЛУБОКОЕ ОБУЧЕНИЕ, LABELME, TENSORFLOW, OPENC, PYTHON

Целью магистерской диссертации является разработка программного комплекса, использующего методы искусственных нейронных сетей для автоматического позиционирования объектов на цифровых картах на основе семантической сегментации спутниковых изображений.

В результате проделанной работы проведен анализ предметной области, выбрана среда разработки, проведен анализ аналогичных продуктов, проектирование и разработка приложения.

В результате выполнения работы разработано программное обеспечение, реализующее автоматическую семантическую сегментацию спутниковых изображений с использованием архитектуры нейронной сети U-Net, а также модуль позиционирования объектов инфраструктуры на цифровых картах на основе полученных масок.

При разработке приложения использовались язык программирования Python и JavaScript, фреймворк Flask для серверной части, библиотека TensorFlow для построения нейросети U-Net, инструменты OpenCV и Pillow для обработки изображений, а также редактор Visual Studio Code. Разметка данных выполнялась в LabelMe, визуализация и взаимодействие с картами – через Yandex Maps API.

					ПГУ 1.090401.15.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Использование искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Экимов Д.В.		26.06.25					
Провер.		Такташкин Д.В.		26.06.25			2	135	
Н. Контр		Бычков А.С.		26.06.25		ФВТ, каф. ВТ			
Реценз.		Петеркин С.		26.06.25		гр. 23ВВИм1			
Утверд.		Митрохин М.А.		26.06.25					

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	3
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Актуальность	7
2 Предпроектное исследование	9
2.1 Обзор существующих методов.....	9
2.1.1 Обнаружение объектов с использованием алгоритмов обнаружения объектов	9
2.1.2 Сегментация с помощью сверточных нейронных сетей (CNN)	11
2.1.3 Сегментация с помощью рекуррентных нейронных сетей (RNN) ..	15
2.2 Обоснование выбора средств разработки.....	16
3 Архитектура нейронной сети для семантической сегментации картографических данных.....	21
3.1 Семантическая сегментация данных.....	21
3.2 Классификация объектов и формирование масок	22
3.3 Обоснование выбора архитектуры U-Net.....	26
3.4 Функция потерь	31
3.5 Обучение модели и анализ датасета	32
4 Техническое задание	34
4.1 Основание для разработки	34
4.2 Назначение разработки.....	34
4.3 Требования к программе	34
4.3.1 Требования к функциональным характеристикам	34
4.3.2 Требования к надежности	35
4.3.3 Требования к составу и параметрам технических средств.....	35
4.3.4 Требования к информационной и программной совместимости	36
4.4 Требования к программной документации	36
4.5 Стадии и этапы разработки	37
4.6 Порядок контроля и приемки	37
5 Разработка программного обеспечения	38

5.1	Алгоритм размещения объектов на карте	38
5.2	Диаграмма вариантов использования	39
5.3	Диаграмма развертывания	40
5.4	Разработка структуры приложения	41
6	Описание программы	44
6.1	Общие сведения	44
6.2	Функциональное назначение	44
6.3	Используемые технические средства	44
6.4	Вызов и загрузка	45
6.5	Входные данные	45
6.6	Выходные данные	45
7	Программа и методика испытаний	47
7.1	Объект испытаний	47
7.2	Цель испытаний	47
7.3	Средства и порядок испытаний	47
7.4	Методы испытаний	47
7.5	Результаты тестирования и проверки функциональности	53
8	Описание применения	56
8.1	Назначение применения	56
8.2	Условия применения	56
8.3	Описание задачи	56
8.4	Инструкция развертывания	60
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	63
	ПРИЛОЖЕНИЕ А Листинг программы	66
	ПРИЛОЖЕНИЕ А.1 Файл «combined_app.py»	67
	ПРИЛОЖЕНИЕ А.2 Файл «yandex_maps.py»	76
	ПРИЛОЖЕНИЕ А.3 Файл «map_segmentation_service.py»	78
	ПРИЛОЖЕНИЕ А.4 Файл «index.html»	104
	ПРИЛОЖЕНИЕ А.5 Файл «style.css»	119
	ПРИЛОЖЕНИЕ Б Презентация	123

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей магистерской диссертации применяются следующие термины и определения

API (Application Programming Interface) – Программный интерфейс приложения набор определений и протоколов для взаимодействия между программными компонентами

Categorical Crossentropy – Функция потерь, применяемая для многоклассовой классификации, измеряющая расхождение между предсказанным распределением и истинным

CNN – Сверточная нейронная сеть. Класс глубоких нейронных сетей, предназначенных для обработки изображений и извлечения признаков с использованием сверточных фильтров

F1-score – Гармоническое среднее между precision и recall, используемое для оценки точности классификации

Flask – Легковесный микрофреймворк на Python для создания веб-приложений и REST API

GRU – Тип архитектуры рекуррентной нейронной сети. Она используется в глубоком обучении для обработки последовательных данных

IoU (Intersection over Union) – Метрика для оценки качества сегментации, отражающая степень пересечения предсказанной и истинной масок

JS – Мультипарадигменный язык программирования

LabelMe – Инструмент разметки изображений, используемый для создания масок сегментации вручную

LSTM – Специализированная архитектура рекуррентных нейронных сетей, разработанная для обработки последовательностей данных, где важен контекст

Mask R-CNN – Архитектура нейронной сети, совмещающая детекцию объектов и сегментацию масок на изображении

NumPy – Библиотека Python для работы с многомерными массивами, матрицами и функциями линейной алгебры

OpenCV (Open-Source Computer Vision Library) – Библиотека компьютерного зрения с открытым исходным кодом, применяемая для обработки изображений и видео

PIL / Pillow – Библиотека Python для обработки изображений, включая открытие, преобразование и сохранение в различных форматах

Pixel accuracy – Метрика качества сегментации, отражающая долю пикселей, правильно классифицированных моделью

ReLU (Rectified Linear Unit) – Функция активации, применяемая в нейросетях, обнуляющая отрицательные значения и сохраняющая положительные

RGB – Цветовая модель, использующая три канала: красный (Red), зелёный (Green) и синий (Blue)

RNN – Тип нейронных сетей, специально разработанный для обработки последовательных данных

Softmax – Функция активации, преобразующая выходные значения модели в вероятностное распределение по классам

Swin Transformer – Архитектура на основе self-attention, применяемая в задачах сегментации и обработки изображений высокого разрешения

TensorFlow – Платформа для машинного обучения и нейронных сетей, предназначенная для построения, обучения и внедрения моделей ИИ

UML – Унифицированный язык моделирования. Это графический язык, который с помощью диаграмм и схем описывает разнообразные процессы и структуры

U-Net – Архитектура сверточной нейронной сети для задач семантической сегментации изображений, включающая симметричный энкодер-декодер с skip-соединениями

YOLOv5 – Семейство моделей нейронных сетей для детектирования объектов в реальном времени

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

API (Application Programming Interface) – программный интерфейс приложения

CNN (Convolutional Neural Network) – свёрточная нейронная сеть

GRU (Gated Recurrent Unit) – упрощённая рекуррентная сеть с единицей обновления

IDE (Integrated Development Environment) – интегрированная среда разработки

IoU (Intersection over Union) – перекрытие по отношению к объединению

JS (JavaScript) – язык программирования JavaScript

LSTM (Long Short-Term Memory) – длинная краткосрочная память

PIL (Python Imaging Library) – библиотека изображений Python

ReLU (Rectified Linear Unit) – выпрямленная линейная единица

RGB (Red, Green, Blue) – красный, зелёный, синий (цветовая модель)

RNN (Recurrent Neural Network) – рекуррентная нейронная сеть

TTS (Text-to-Speech) – преобразование текста в речь

UML (Unified Modeling Language) – унифицированный язык моделирования

VS Code (Visual Studio Code) – визуальная студия кода

ВВЕДЕНИЕ

В современном мире цифровые карты играют важную роль в различных областях, от навигации и транспортной инфраструктуры до городского планирования и экологического мониторинга. Однако точное позиционирование объектов на цифровых картах остается сложной задачей, требующей эффективных методов обработки данных и анализа пространственной информации. В последние годы искусственные нейронные сети стали популярны в области анализа геопро пространственных данных благодаря их способности обрабатывать большие объемы информации и автоматически извлекать признаки из изображений или других типов геоданных. В связи с этим, использование искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах становится все более актуальным и перспективным направлением исследований. На данный момент существует достаточно много универсальных ГИС, использование которых позволяет решать множество задач. Но чаще всего для этого требуется наличие специалиста, так как решение прикладных задач в ГИС в основном основывается на знаниях картографических данных, а также математических методах и алгоритмов обработки [1].

В данной научной работе будет рассмотрено применение различных методов искусственных нейронных сетей для обнаружения, классификации и позиционирования объектов на цифровых картах. Также будут исследованы их возможности и перспективы в контексте современных задач геоинформационного анализа.

По теме исследования опубликована 1 научная работа.

1 Актуальность

Тема «Использование искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах» имеет высокую актуальность по ряду причин.

Использование искусственных нейронных сетей позволяет достичь более высокой точности в определении и позиционировании объектов на цифровых картах. Нейронные сети способны обнаруживать и анализировать сложные паттерны и зависимости в геоданных, что повышает точность результата. Например, применение сверточных нейронных сетей, таких как архитектура YOLOv5, обеспечивает высокие значения точности ($\text{precision} \approx 0,89$) и полноты ($\text{recall} \approx 0,86$) при детектировании объектов на спутниковых изображениях.

Автоматизация процесса определения и позиционирования объектов на карте упрощает создание и обновление картографических данных. Это особенно важно в условиях постоянного изменения окружающей среды, таких как изменения в городской застройке или состоянии природных ресурсов. Использование методов глубокого обучения, в том числе Mask R-CNN и U-Net, позволяет автоматически выявлять изменения объектов на местности, способствуя регулярному и оперативному обновлению пространственных данных.

С увеличением объема доступных геоданных и спутниковых изображений становится всё более сложным обрабатывать и анализировать эти данные вручную. Использование искусственных нейронных сетей позволяет обрабатывать большие объемы данных эффективно и быстро, что увеличивает производительность и точность анализа. В рамках программных решений, таких как ArcGIS Pro от компании ESRI, уже реализованы инструменты глубокой обработки изображений, позволяющие выполнять классификацию, сегментацию и извлечение объектов с использованием предобученных моделей.

Нейронные сети могут использоваться для решения различных задач на цифровых картах, таких как обнаружение и классификация объектов, сегментация изображений, определение типов поверхности и многое другое. Это позволяет расширить функциональность картографических приложений и улучшить пользовательский опыт. Например, возможность автоматического определения объектов или изменений на карте может значительно улучшить удобство использования приложений для пользователей. Кроме того, современные модели, включая трансформеры, например, Swin Transformer, демонстрируют способность эффективно обрабатывать изображения высокого разрешения и учитывать контекст сцены, что особенно важно для точного позиционирования объектов [2].

С развитием технологий и методов глубокого обучения нейронные сети становятся всё более мощными и эффективными инструментами для анализа данных. Применение их в области картографии обещает привнести новые возможности и улучшить качество картографических продуктов, обеспечивая более точное и быстрое позиционирование объектов на цифровых картах. Архитектуры, использующие методы transfer learning, также показывают высокую обобщающую способность при переносе знаний с больших датасетов на задачи позиционирования в специфических геопространственных условиях.

Таким образом, использование искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах является актуальным направлением исследований и разработок в области геоинформационных систем и картографии.

2 Предпроектное исследование

2.1 Обзор существующих методов

2.1.1 Обнаружение объектов с использованием алгоритмов обнаружения объектов

Идентификация и точная локализация разнообразных объектов на фото или видеоматериалах достигается через применение метода обнаружения объектов. Данный процесс охватывает распознавание наличия объектов в кадре и генерацию контуров-областей вокруг них для их точного выделения.

Компоненты распознавания объектов через объектные детекторы:

- использование нейросетей,
- архитектуры детекторов объектов,
- процесс обучения,
- применение в различных областях.

Многообразие архитектур для детекции объектов включает в себя такие системы, как Faster R-CNN, YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector) среди прочих. Они различаются по ряду характеристик и преимуществ, однако объединяются общим принципом использования предварительно обученных искусственных нейронных сетей для определения видов объектов и вычисления их позиций в рамках представленного изображения [3].

Для тренировки систем распознавания объектов необходима обширная база аннотированных данных, включая ярлыки классов и локализацию объектов на снимках. Искусственный интеллект совершенствуется на основе этих данных через алгоритмы оптимизации, например, метод градиентного спуска, целью которых является снижение разницы между выходными данными сети и фактическим положением вещей.

Системы распознавания объектов осуществляют автоматизированное идентификацию лиц, обнаружение автомобилей на дорожном полотне и контроль качества в промышленности. Используемые в навигационном программном обеспечении и системах обеспечения безопасности, эти

технологии эффективно классифицируют дорожные знаки по их типам. Они умеют узнавать знаки, регулирующие скоростной режим, знаки предупреждения, а также таблички, указывающие направления движения, что способствует повышению безопасности и комфорта использования.

Применение системы распознавания объектов охватывает идентификацию таких структур, как здания, мосты, аэропорты и стадионы на картах, обогащая их подробными сведениями о этих объектах. Это значительно улучшает качество цифрового картографирования, предоставляя ценную информацию для сферы туризма, проектирования городской инфраструктуры и различных прикладных услуг.

Системы обнаружения объектов также применимы для распознавания естественных форм рельефа на картах, включая водные потоки, водоемы, древесные массивы и горные массивы. Это способствует исследованию географических характеристик региона и организации активного досуга.

Обнаружение объектов может также охватывать идентификацию элементов городской инфраструктуры, включая светофоры, перекрестки, железнодорожные вокзалы и дорожные артерии. Этот процесс имеет важное значение для создания систем регулирования дорожного движения и стратегического планирования транспортной сети.

Распознавание объектов на картах способствует идентификации и аннотированию значимых локаций, включая кафе, выставки, зелёные зоны и культурные памятники. Это может улучшить информативность цифровых карт для путешественников и любителей туризма.

Следовательно, применение алгоритмов распознавания объектов позволяет значительно повысить ценность и актуальность цифровых картографических материалов, обогащая их детальной информацией о разнообразных элементах и характеристиках ландшафта.

Картографические сервисы, такие как Google Maps, Яндекс Карты и 2ГИС, применяют технологии обнаружения объектов. Аналогично, системы

OpenStreetMap и Apple Maps тоже задействуют алгоритмы обнаружения объектов.

2.1.2 Сегментация с помощью сверточных нейронных сетей (CNN)

Эта технология сегментации эффективно идентифицирует, оценивает и категоризирует элементы на основе анализа визуальной информации, что обеспечивает ее широкое применение в создании и обновлении цифровых картографических данных [4].

Обычно, методика сегментации с использованием сверточных нейронных сетей (CNN) включает следующие этапы:

- подготовка данных,
- обучение модели,
- тестирование модели,
- применение модели.

В ходе предобработки данных для обучения нейронной сети производится ресайзинг и нормализация изображений, а также аннотация с правдивыми данными. Искусственная нейронная сеть анализирует этот датасет, абстрагируя и изучая паттерны для точной идентификации элементов на различных изображениях. По окончании тренировочного процесса проводится валидация модели на внешнем наборе данных, который ранее не участвовал в тренинге. Этот этап критически важен для оценки эффективности работы алгоритма сегментации и его способности применять наработанные знания к неизвестным образцам.

По завершении тренировочного процесса и валидации, модель готова к сегментации элементов на свежих снимках или в электронных картах. Приложение модели к визуальному контенту позволяет классифицировать каждый пиксель по принадлежности к конкретной категории объектов.

Преимущества применения сегментации на основе свёрточных нейронных сетей в географическом картографировании:

- высокоточное и эффективное распознавание элементов на графических материалах,

- автоматическая обработка географической информации,
- гибкость в обработке разнообразных объектов данных и адаптация к изменяющимся условиям.

Данная технология широко используется в многочисленных приложениях для определения и категоризации объектов в цифровом картографировании, исследовании динамики ландшафтов, наблюдении за природными изменениями. Среди всемирно известных организаций, применяющих свёрточные нейронные сети (CNN) в сочетании с географическими информационными системами, можно отметить:

Яндекс.Навигатор, ведущее навигационное приложение в России, применяет методы компьютерного зрения и алгоритмы машинного обучения для анализа снимков, полученных с камер смартфонов пользователей, и классификации дорожных элементов, включая разметку полос, дорожные знаки и прочие компоненты. Интерфейс данной системы приведен на рисунке 1.

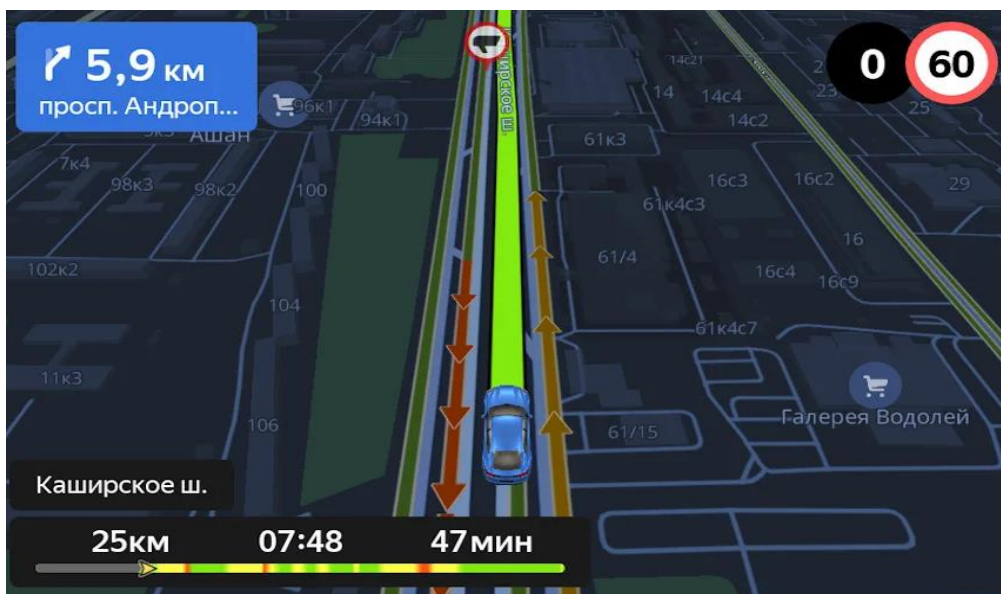


Рисунок 1 – Система навигации Яндекс.Навигатор

С использованием передовых навигационных систем, приложение в реальном времени анализирует данные о динамике дорожной ситуации, предлагая пользователям своевременные и релевантные обновления.

Компания ДубльГИС широко признана за разработку инновационного веб-сервиса картографии, активно собирающего и предоставляющего

сведения о различных бизнесах и учреждениях по всей России и за её пределами. В рамках этого продукта широко применяются передовые разработки в области искусственного интеллекта, включая применение сверточных алгоритмов сверточных нейронных сетей для всестороннего анализа данных, касающихся земельных участков и разметки территорий. Эффективная обработка этих информационных массивов приводит к получению следующих ключевых результатов:

- разделение элементов на цифровом изображении карты, таких как идентификация и выделение структурных компонентов, включая здания, проезжие части, водные объекты и прочие инфраструктурные объекты,
- автоматизированная идентификация и категоризация сущностей на фотографиях, таких как разнообразные категории сооружений, торговые точки, путепроводы,
- определение и сверка местоположения объектов на географической карте с информационной базой данных предприятий для актуализации и пополнения сведений.

Интерфейс рассматриваемой системы приведен ниже на рисунке 2.

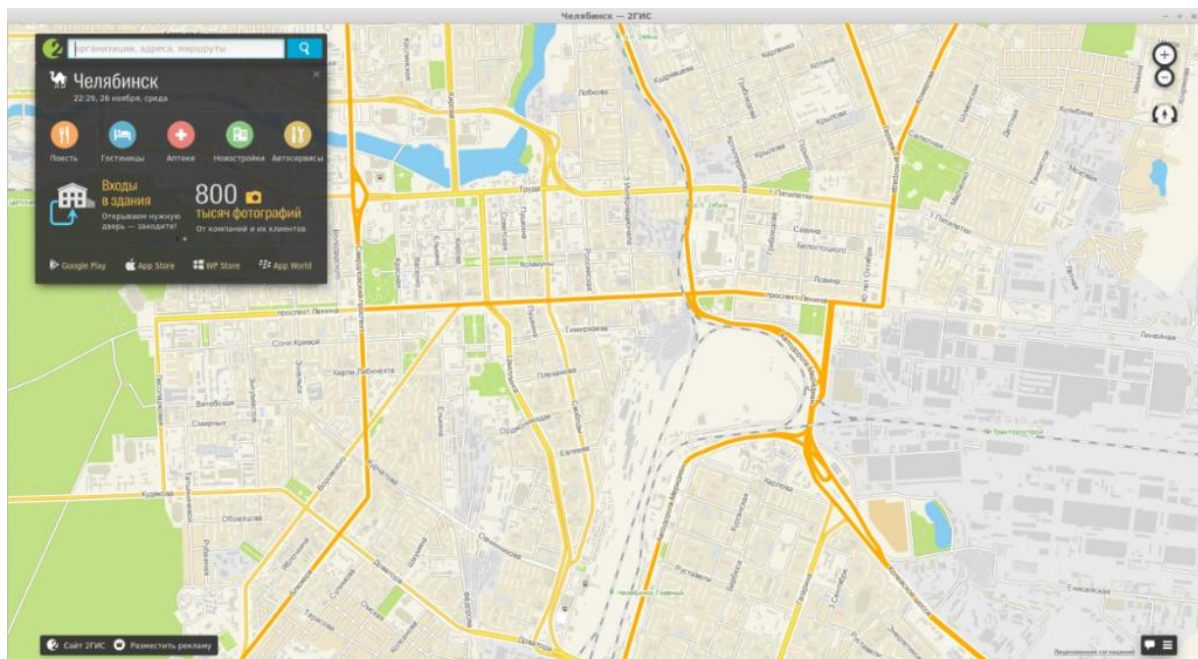


Рисунок 2 – Система навигации 2ГИС

Свёрточные нейронные сети в системе 2ГИС используются в картографических продуктах не только для улучшения качества и точности данных, но и обеспечения простоты и удобства работы сервиса для клиентов.

Google Maps является глобальным веб-сервисом карт и навигации, разработанным технологическим гигантом Google. Он обеспечивает детализированную информацию о маршрутизации, транспортных заторах, точках интереса и топографических особенностях для пользователей по всему миру. Для повышения точности и детализации картографических материалов Google применяет методы искусственного интеллекта, включая свёрточные нейронные сети (CNN) и алгоритмы компьютерного зрения, что позволяет эффективно анализировать изображения земной поверхности и выделять на них отдельные элементы и объекты. Интерфейс системы приведен на рисунке 3.

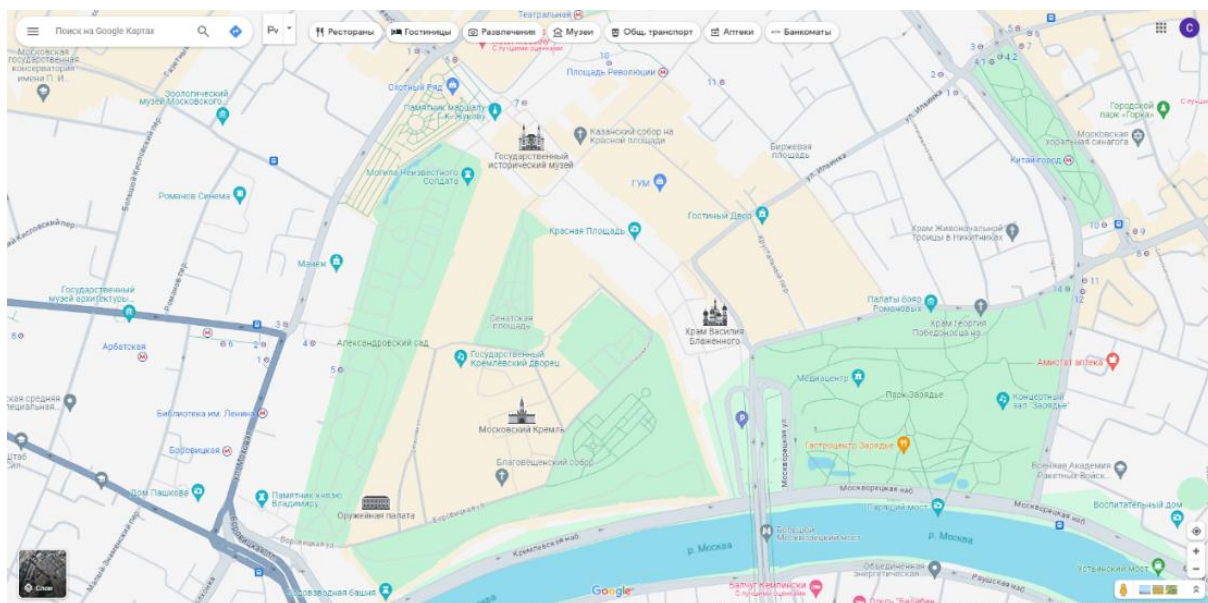


Рисунок 3 – Система Google Maps

Использование свёрточных нейронных сетей (CNN) в системе способствует оптимизации точности и качества геопространственных данных в геоинформационных системах (ГИС) за счёт улучшения процесса сегментации объектов.

2.1.3 Сегментация с помощью рекуррентных нейронных сетей (RNN)

Этот подход применяется для исследования временных рядов данных на географических картах, таких как идентификация шаблонов перемещения или динамики изменения атрибутов объектов. Особенно эффективен он в выявлении трендов эволюции уличного движения. Для таких аналитических операций рекуррентные нейросети (RNN) подходят для обработки геопространственных данных, охватывая диапазон задач от распознавания последовательностей до прогнозирования временных последовательностей [5].

RNN могут анализировать временные данные о трафике на цифровых картах и предсказывать будущие трафиковые потоки. На основе исторических данных о движении транспорта, погодных условиях и событиях на дорогах RNN могут предсказывать уровень загруженности дорог в будущем.

Пример интерфейса разметки дорожного трафика приведен на рисунке 4.



Рисунок 4 – Разметка дорожного трафика

Этот вид искусственной нейронной сети имеет возможность обрабатывать информацию с систем глобального позиционирования (GPS) и прочие связанные с передвижением данные для идентификации

закономерностей и трендов в движении людей или транспорта на картах. К примеру, он может выявлять маршруты с высокой проходимостью или районы с интенсивной деятельностью.

Рекуррентные нейронные сети применяются для исследования динамики ландшафта через анализ геопространственных данных, таких как колебания рельефа или изменения во флоре. Они находят применение в отслеживании экологических сдвигов или в разработке стратегий землепользования.

Кроме того, их применение включает в себя улучшение планирования маршрутов через анализ времени в пути, учитывая прогнозы движения, и включает переменные, такие как дорожная активность, для предоставления оптимального пути с учетом существующих временных рамок [6].

2.2 Обоснование выбора средств разработки

В качестве основного языка разработки программного модуля был выбран Python версии 3.7. Это решение обусловлено широким распространением языка в сфере машинного обучения, наличием большого числа специализированных библиотек, удобством синтаксиса, а также активной поддержкой со стороны профессионального сообщества. Средства, предоставляемые Python, позволяют эффективно реализовывать алгоритмы глубокого обучения, осуществлять обработку графических данных и взаимодействовать с внешними интерфейсами через API [7, 8].

JavaScript, в сочетании с HTML5/CSS3, используется для реализации клиентской части веб-приложения. Этот выбор обусловлен необходимостью создания интерактивного, отзывчивого пользовательского интерфейса для визуализации результатов сегментации и взаимодействия с картами [9].

Flask 2.3.3 – это легковесный микрофреймворк для создания REST API и веб-приложений. Благодаря своей простоте и гибкости, Flask обеспечивает быструю интеграцию модели машинного обучения с веб-интерфейсом, позволяя организовать отправку изображений, вызов сегментации и отображение результатов [10].

TensorFlow 2.13.0 это одна из ведущих платформ для построения и обучения нейросетевых моделей. TensorFlow обеспечивает высокую производительность, поддержку GPU, экспорт моделей (ONNX, SavedModel, TorchScript) и богатый набор инструментов для мониторинга (TensorBoard) [11].

OpenCV 4.8.1, Pillow (PIL) 10.0.1 и NumPy 1.24.3 используются для предварительной обработки изображений, преобразования масок, масштабирования и других операций, необходимых для подготовки данных и визуализации результатов.

Scikit-learn 1.3.0 применяется для разделения выборки, оценки метрик, построения confusion matrix и анализа результатов сегментации, а также Matplotlib 3.7.2 для наглядной визуализации результатов сегментации, сравнения предсказаний и масок [12].

LabelMe 3 используется для ручной разметки спутниковых изображений. Он позволяет создавать полигональные маски с привязкой к определённым классам, что необходимо для обучения моделей семантической сегментации. Yandex Maps API, Static Maps API, Geocoding API и JavaScript API позволяют интегрировать картографические данные в пользовательский интерфейс, обеспечивая географическую привязку и поиск по координатам или адресу [13].

Visual Studio Code используется как основная IDE благодаря своей лёгкости, поддержке Python, интеграции с системами контроля версий и расширяемости. Поддержка Jupyter-ноутбуков, отладчиков, Git и Docker делает VS Code удобным инструментом для разработки как серверной, так и клиентской части проекта [13].

OpenCV (Open Source Computer Vision Library) – это одна из наиболее широко используемых библиотек компьютерного зрения с открытым исходным кодом, предоставляющая мощный инструментарий для обработки изображений [15]. В рамках проекта библиотека применяется для:

- морфологической фильтрации маски (операции эрозии, дилатации, открытия и закрытия) с целью устранения шумов и выделения связных компонент пригодных зон,
- выделения контуров (функция `cv2.findcontours`) и анализа геометрических характеристик объектов,
- построения бинарных и цветовых масок, а также их заливки,
- геометрического размещения объектов на изображении путём нанесения фигур (прямоугольников, кругов, линий) и аннотаций (`cv2.rectangle`, `cv2.putText`),
- интерполяции и масштабирования (например, для совмещения изображения с маской или пересчёта координат между различными разрешениями).

Использование OpenCV позволяет эффективно оперировать с пиксельными данными и выполнять сложные преобразования без значительных затрат вычислительных ресурсов.

PIL (Python Imaging Library)/Pillow – это активно поддерживаемое расширение библиотеки PIL, предназначенное для создания, обработки и сохранения изображений [16]. В контексте реализации визуализации размещения объектов библиотека выполняет следующие функции:

- создание холста для итогового изображения, объединяющего исходную карту, маску и графику размещения,
- сохранение изображений в нужном формате (PNG, JPEG), в том числе с прозрачностью или наложением,
- конвертация между форматами и режимами изображений (RGB ↔ RGBA, массив ↔ Image),
- финальная отрисовка элементов, особенно в случаях, когда необходима высокая точность позиционирования с сохранением альфа-канала.

Благодаря Pillow удаётся обеспечить корректную интеграцию визуализации в веб-интерфейс приложения, а также экспорт изображений для внешнего использования.

Для задачи семантической сегментации спутниковых изображений выбрана архитектура U-Net, хорошо зарекомендовавшая себя в медицине, спутниковой съёмке и других областях, где требуется точная локализация объектов.

U-Net – одна из самых популярных архитектур для семантической сегментации, предложенная Олафом Роннебергером и соавторами в 2015 году. Ключевая особенность U-Net – наличие skip-connections, которые соединяют соответствующие уровни encoder и decoder, что позволяет сохранять детальную пространственную информацию [17].

Преимущества U-Net:

- высокая точность на малых выборках,
- хорошо работает с медицинскими и спутниковыми изображениями,
- простота модификации и расширения,
- поддержка генерации масок с последующей визуализацией, наложением на исходную карту и возможностью извлечения статистики по площади объектов.

Схема U-Net:

- Encoder: несколько блоков Conv2D + MaxPooling2D,
- Decoder: блоки UpSampling2D + Conv2D,
- Skip-connections: объединяют карты признаков encoder и decoder на каждом уровне.

Исходя из анализа выбранных технологий и инструментов, можно сделать вывод о целесообразности использования Python 3.7 в качестве основного языка разработки, фреймворка Flask 2.3.3 для построения серверной части и REST API, JavaScript совместно с HTML/CSS для реализации клиентского интерфейса, а также библиотеки TensorFlow 2.13.0 и архитектуры U-Net для решения задачи семантической сегментации

спутниковых изображений. Для реализации и внедрения модели были использованы дополнительные программные библиотеки и инструменты, такие как OpenCV, Pillow, NumPy, Scikit-learn, Matplotlib, LabelMe, а также Yandex Maps API. Эти компоненты позволяют организовать полный процесс разработки – от предварительной разметки и обучения модели до предсказания, визуализации результатов и интеграции с геоданными. В качестве основной среды разработки использовался Visual Studio Code.

3 Архитектура нейронной сети для семантической сегментации картографических данных

3.1 Семантическая сегментация данных

Семантическая сегментация представляет собой метод анализа изображений, при котором каждому пикселю присваивается соответствующая категория. Это позволяет не только выявлять объекты на изображении, но и точно очерчивать их границы. Такая задача имеет особую значимость в сфере цифровой картографии, где необходимо автоматизированно определять участки, занятые различными объектами инфраструктуры – зданиями, дорожной сетью, водными объектами, лесными зонами и другими элементами местности. В последние годы наибольшую популярность получили свёрточные нейронные сети (Convolutional Neural Networks, CNN), а также их специализированные архитектуры для сегментации, такие как U-Net, DeepLab, SegNet и др. В данной главе рассматривается архитектура нейронной сети, предназначенной для решения задачи семантической сегментации картографических данных, а также обосновывается выбор конкретных технических решений [18].

Задача семантической сегментации формализуется следующим образом: имеется множество спутниковых изображений $X = \{x_i\}$, для каждого из которых задана маска разметки $Y = \{y_i\}$, где каждому пикселю соответствует метка одного из классов $C = \{c_1, \dots, c_K\}$. Требуется построить функцию $f_\theta(x)$, параметризованную весами нейронной сети θ , которая для любого входного изображения x предсказывает маску $\hat{y}$, максимально близкую к истинной разметке. В нашем случае классы включают: поле, лес, вода, дорога, здание.

Свёрточные нейронные сети – это класс глубоких нейронных сетей, специально разработанных для обработки изображений. Основная идея заключается в использовании сверточных фильтров, которые автоматически извлекают локальные признаки на разных уровнях абстракции. Классическая

архитектура CNN состоит из чередующихся сверточных и pooling-слоев, за которыми следуют полносвязные слои [19].

Основные преимущества CNN:

- инвариантность к сдвигу объектов на изображении,
- эффективное использование параметров за счет локальных связей и совместного использования весов,
- возможность построения глубоких моделей для сложных задач.

Для задач сегментации применяются архитектуры типа encoder-decoder. Encoder (кодировщик) последовательно уменьшает пространственное разрешение изображения, извлекая все более абстрактные признаки, а decoder (декодировщик) восстанавливает пространственное разрешение, используя полученные признаки для точного позиционирования объектов.

Encoder включает блоки из двух сверточных слоев с функцией активации ReLU и слоя пакетной нормализации, за которыми следует слой подвыборки (MaxPooling). С увеличением глубины растет количество фильтров, что позволяет извлекать более абстрактные признаки. Decoder симметричен encoder и использует слои UpSampling и skip-соединения с соответствующими слоями encoder, что позволяет восстанавливать пространственную структуру с сохранением детализированных признаков. Также часть модели, называемая Bottleneck, содержит два сверточных слоя с увеличенным количеством фильтров, выполняющих сжатие пространственного представления и извлечение высокоуровневых признаков.

3.2 Классификация объектов и формирование масок

В рамках настоящей работы рассматривается задача семантической сегментации спутниковых изображений, направленная на автоматическую разметку цифровой карты и последующее использование для размещения объектов. Решение этой задачи предполагает применение свёрточных нейронных сетей, обученных на размеченных изображениях, полученных с помощью спутников или цифровых карт.

Основная идея состоит в том, чтобы подать на вход модели изображение определённого участка местности, а на выходе получить маску, где каждый пиксель классифицирован в соответствии с определённым классом (здание, дорога, водоём и т.д.). Для этого требуется предварительно обучить модель на размеченном датасете, в котором каждой исходной карте соответствует соответствующая маска-аннотация.

Для подготовки модели требуется набор данных, состоящий из пар «изображение – маска». Маска – это изображение, где каждый пиксель имеет значение, соответствующее одному из классов. Датасет формировался вручную с использованием инструмента LabelMe, позволяющего создавать полигональные аннотации и экспортировать их в формат PNG. Изображения, полученные из API Яндекс.Карт, приводились к фиксированному размеру 448 на 448 пикселей.

Маски аннотировались в соответствии со следующими классами:

- здания,
- дороги,
- лесные массивы,
- поля,
- водоёмы.

Каждому классу соответствовал уникальный цвет, зафиксированный в конфигурационном файле модели. Пример разметки приведен на рисунке 5.

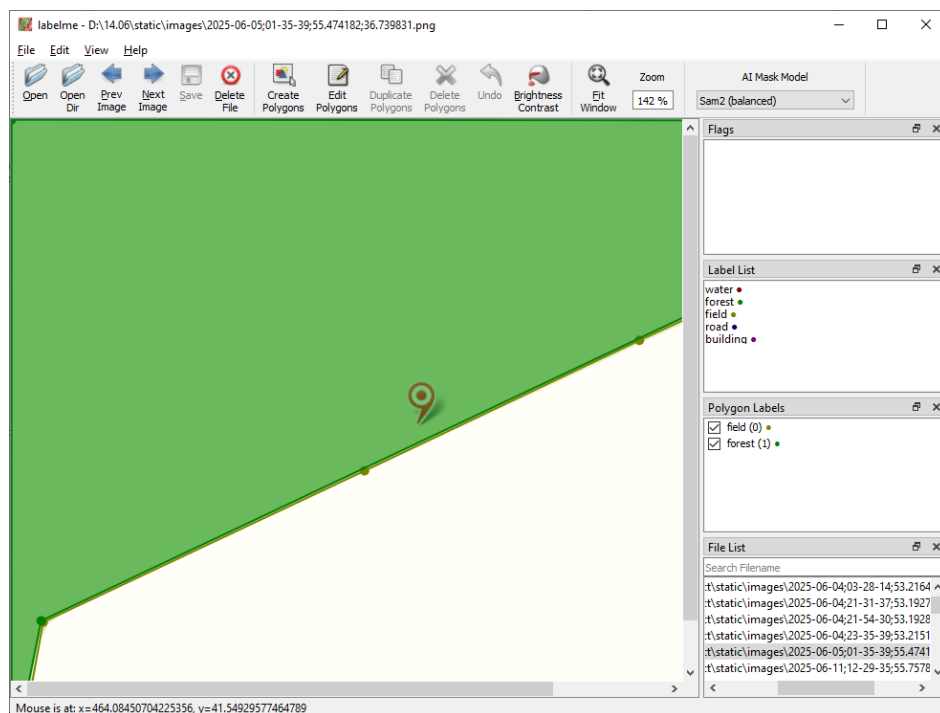


Рисунок 5 – Интерфейс Labelme3

В процессе формирования маски модель U-Net возвращает тензор размерности (448, 448, 5), где 5 – это число классов, а также применяется операция argmax по последней размерности для выбора наиболее вероятного класса для каждого пикселя. Каждый класс переводится в соответствующий RGB-цвет. Маска используется не только для визуализации, но и как основа для логики расположения объектов, например, поиск подходящих участков для размещения ПВР.

Веса классов – это коэффициенты, которые увеличивают влияние редких классов на функцию потерь. Математически это выражается следующим образом: Для функции потерь L с весами классов w_c для каждого класса c :

$$L_{\text{weighted}} = \sum_{c=1}^K w_c \cdot L_c ,$$

где L_c – потери для класса c ;

K – количество классов.

Вычисляются веса по формуле:

$$w_c = \frac{N}{K \cdot N_c},$$

где N – общее количество образцов;

N_c – количество образцов класса c ;

K – количество классов.

Расчетные значения весов для нашего набора данных:

- field (поле): от 0.8 до 1.2 (наиболее частый класс),
- forest (лес): от 1.5 до 2.5 (средняя частота),
- water (вода): от 2.0 до 3.5 (редкий класс),
- road (дорога): от 2.5 до 4.0 (очень редкий класс),
- building (здания): от 3.0 до 5.0 (наиболее редкий класс).

Интерпретация весов:

- вес 1.0 означает стандартное влияние класса,
- вес > 1.0 увеличивает влияние класса на обучение,
- вес < 1.0 уменьшает влияние класса.

Альтернативный подход к решению проблемы дисбаланса, который автоматически уменьшает вес легко классифицируемых образцов:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t),$$

где p_t – вероятность правильной классификации;

α_t – вес класса;

γ – фокусный параметр.

Можно использовать данную функцию потерь (Focal Loss) в качестве альтернативы стандартной кросс-энтропии, поскольку она позволяет эффективно справляться с дисбалансом классов за счёт снижения вклада

легко классифицируемых образцов и фокусировки обучения на более трудных примерах.

3.3 Обоснование выбора архитектуры U-Net

Для задачи семантической сегментации спутниковых изображений была выбрана архитектура U-Net как наиболее подходящее решение. U-Net изначально разработана для биомедицинской сегментации, но зарекомендовала себя и в других областях, включая анализ спутниковых снимков. В отличие от методов классификации или детекции, требующих определения объекта целиком, U-Net позволяет отнести каждый пиксель изображения к определённому классу, формируя маску сегментации. Характерной особенностью U-Net является сочетание сжимающего пути, захватывающего контекстные признаки, и расширяющего пути для точной локализации объектов. Такое устройство обеспечивает высокое качество сегментации даже при относительно небольшом объёме обучающих данных, что важно, поскольку разметка спутниковых карт трудоёмка и дорога. Кроме того, U-Net доказала эффективность в пиксельной классификации спутниковых изображений, успешно выделяя такие объекты как здания и дороги [20].

Не менее важно, что архитектура U-Net относительно компактна по сравнению с некоторыми современными моделями. Реализация содержит порядка 487 тысяч параметров и 25 слоёв, что позволяет обучать её с нуля на доступном датасете без необходимости в предобученном энкодере. Несмотря на небольшой размер, U-Net способна учитывать, как глобальный контекст сцены, так и мелкие детали за счёт особой структуры. Таким образом, выбор U-Net обусловлен балансом между точностью сегментации и вычислительной эффективностью модели, а также успешным опытом применения U-Net в смежных задачах сегментации изображений, что делает её оптимальным решением для текущей задачи.

Архитектура модели построена по классической схеме U-образной сети. Она включает кодировщик (encoder), бутылочное горлышко (bottleneck)

и декодировщик (decoder), связанные между собой skip-соединениями. На вход модель получает изображения размером 448×448 с тремя каналами (RGB). Выходом является тензор размера $448 \times 448 \times 5$, где каждому пикселю соответствует вероятностное распределение по пяти классам (поле, лес, вода, дорога, здание). Ниже составлена таблица, отражающая основные блоки архитектуры, размерности выходных тензоров и функциональное назначение каждого этапа.

Таблица 1 – Описание блоков U-Net

Имя блока	Описание	Размерность
ConvBlock1	3 свёртки 3×3 с ReLU	$448 \times 448 \times N1$
MaxPool1	2×2 max pooling	$224 \times 224 \times N1$
ConvBlock2	3 свёртки 3×3 с ReLU	$224 \times 224 \times N2$
MaxPool2	2×2 max pooling	$112 \times 112 \times N2$
ConvBlock3	3 свёртки 3×3 с ReLU	$112 \times 112 \times N3$
MaxPool3	2×2 max pooling	$56 \times 56 \times N3$
Bottleneck	2×3 conv + ReLU, затем 3×3 conv	$56 \times 56 \times N4$
UpSample1	2×2 upsampling	$112 \times 112 \times N3$
DecoderConv1	3 свёртки после concat skip3	$112 \times 112 \times N3$
UpSample2	2×2 upsampling	$224 \times 224 \times N2$
DecoderConv2	3 свёртки после concat skip2	$224 \times 224 \times N2$
UpSample3	2×2 upsampling	$448 \times 448 \times N1$
DecoderConv3	3 свёртки после concat skip1	$448 \times 448 \times N1$

Кодировщик последовательно извлекает признаки из изображения, уменьшая пространственное разрешение и увеличивая глубину признаков. Он состоит из нескольких свёрточных блоков и операций понижающей дискретизации. В модели используется три уровня понижения разрешения – три этапа pooling, что делит изображение на 4 основных масштаба признаков включая исходный. На каждом уровне кодировщика выполняются две

последовательные свёртки 3×3 с активацией ReLU без изменения ширины/высоты за счёт `padding='same'`, после которых применяется MaxPooling 2×2 для уменьшения размера карты признаков в 2 раза. Количество фильтров при этом увеличивается по мере сжатия, чтобы компенсировать потерю пространственной информации. Конкретно, входное изображение $448 \times 448 \times 3$ сначала пропускается через первый свёрточный блок: две свёртки с 16 фильтрами. В результате получаем тензор $448 \times 448 \times 16$.

Далее MaxPooling снижает размер до 224×224 , что является половиной по каждой оси, при сохранении 16 каналов: получаем $224 \times 224 \times 16$. Затем второй свёрточный блок извлекает более сложные признаки: свёртка 3×3 с 32 фильтрами, за ней ещё одна 3×3 с 32 фильтрами. На выходе этого блока – тензор $224 \times 224 \times 32$. Далее вновь применяется pooling 2×2 , и разрешение уменьшается до 112×112 , каналы: 32 тензор $112 \times 112 \times 32$. Третий свёрточный блок кодировщика имеет 64 фильтра в свёртках: после двух свёрток получаем $112 \times 112 \times 64$. Последний этап энкодера – третий pooling, дающий тензор $56 \times 56 \times 64$. Обобщённо, на каждом уровне кодировщика количество фильтров удваивается $16 \rightarrow 32 \rightarrow 64 \rightarrow 128$, а размер карты признаков уменьшается вдвое. Это позволяет сети постепенно сконцентрироваться на более абстрактных и крупномасштабных признаках, сохраняя при этом информацию о локальных деталях в весах фильтров. В конце сжимающего пути достигается самое низкое разрешение 56×56 в нашей модели и максимальная семантическая насыщенность признаков 128 каналов.

Боттлneck или центральная часть U-Net – это слой, который находится на пересечении кодировщика и декодировщика. Он оперирует на самом низком разрешении изображения, где собрана наиболее обобщённая информация о сцене. В реализации боттлneck состоит из двух подряд идущих свёрточных слоёв 3×3 с 128 фильтрами каждый и активацией ReLU. Входом для них служит тензор $56 \times 56 \times 64$, полученный после последнего pooling энкодера, а выходом – тензор $56 \times 56 \times 128$.

За счёт увеличенного числа фильтров модель может на этом уровне распознавать сложные комбинации признаков, например, контекстуальные связи между объектами инфраструктуры [20]. Центральный слой U-Net играет роль информационного узла, здесь признаки максимально сконцентрированы и обобщены, хотя пространственная информация значительно сгрублена. Беспосадочная архитектура отсутствие полносвязных слоёв позволяет обработать тензоры любой размерности, однако в нашей задаче размер входа фиксирован. Поскольку мы использовали свёртки с заполнением краёв – «same padding», размер тензоров в буттлнеке (56×56) точно соответствует результату деления исходного разрешения на 2^3 , так как три раза применялся pooling 2×2 . Это отличается от оригинального U-Net, где использовались свёртки без паддинга и приходилось обрезать карты признаков из-за уменьшения ширины границы на каждом слое.

В данном случае же применение «same padding» устраняет необходимость обрезки – декодер получает карты признаков той же размерности, что и требуемое разрешение на данном уровне. Таким образом, буттлнек концентрирует контекст изображения: для каждого из 128 каналов обобщается информация с рецептивного поля размером 56×56 , что соответствует области 448×448 исходного изображения, учитывая все свёртки и pooling. После этого этапа модель имеет высокоуровневое «понимание» сцены – например, где в общем расположены крупные объекты, но для восстановления точных границ необходим декодировщик.

Декодировщик выполняет постепенное восстановление пространственного разрешения сегментационной маски, комбинируя обобщённые признаки буттлнека с деталями, сохранёнными на этапах кодирования. Каждый уровень декодера соответствует уровню кодировщика в обратном порядке и включает увеличение размерности, конкатенацию со skip-признаками и два свёрточных слоя.

На первом уровне декодера входом служит тензор $56 \times 56 \times 128$ из буттлнека. Применяется увеличение размерности, чтобы удвоить ширину и

высоту тензора до 112×112 . Используется слой `UpSampling2D` в Keras (ближайший сосед), что эквивалентно некому обобщению транспонированной свёртки без дополнительных параметров. Полученный тензор $112 \times 112 \times 128$ затем объединяется с пропущенными по skip-соединению признаками с соответствующего уровня энкодера. В частности, из кодировщика на уровень 112×112 был сохранён тензор $112 \times 112 \times 64$.

После конкатенации каналов получается тензор $112 \times 112 \times 192$ 128 каналов от декодера + 64 каналов от энкодера. Эта операция реализует основной принцип U-Net – добавление локальной детальной информации из энкодера к обобщённым признакам декодера. Благодаря этому декодер способен точнее восстанавливать мелкие детали объектов, например, контуры дорог или границы водоёмов, которые были бы потеряны без skip-соединений. Далее на объединённый тензор $112 \times 112 \times 192$ накладывается сверточный блок. В первом из них мы уменьшаем число каналов до 64. Эта свёртка фактически смешивает информацию декодера и сохранённые признаки энкодера и снижает избыточность каналов после конкатенации. Вторая свёртка уровня также имеет 64 фильтра, сохраняя размер $112 \times 112 \times 64$.

После этого уровня декодера разрешение 112×112 и 64 каналов совпадают с характеристиками слоя энкодера, от которого брались skip-признаки, но теперь эти 64 каналов содержат комбинированную информацию о соответствующей части изображения. Аналогичным образом, следующий уровень декодера берет тензор $112 \times 112 \times 64$, увеличивает его до $224 \times 224 \times 64$, конкатенирует с сохранёнными признаками энкодера размером $224 \times 224 \times 32$, получая тензор $224 \times 224 \times 96$. Затем две свёртки: первая снижает каналы до 32, вторая уточняет до $224 \times 224 \times 32$. Наконец, последний уровень декодера увеличивает тензор $224 \times 224 \times 32$ до исходного разрешения $448 \times 448 \times 32$, присоединяет признаки начального уровня энкодера, получая $448 \times 448 \times 48$, и пропускает через две заключительные свёртки с 16 фильтрами каждая. В результате формируется тензор $448 \times 448 \times 16$, уже сопоставимый по пространственным размерам с оригинальным изображением.

Все skip-соединения в U-Net осуществляются путём конкатенации карт признаков по каналному измерению. Это позволяет декодеру на каждом шаге получать дополнительные признаки высокого разрешения. Skip-соединения эффективно решают проблему утраты тонких граней объектов, возникающую из-за сильного сжатия изображения в энкодере. Они же облегчают протекание градиентов обратно через сеть, способствуя более стабильному обучению глубоких слоёв модели. Таким образом, декодер, обладающий информацией от соответствующих слоёв энкодера, способен реконструировать маску с высокой точностью.

После прохождения всех уровней декодера получается тензор размером $448 \times 448 \times 16$, который несёт интегрированную информацию о признаках каждого класса в каждом пикселе. Однако для получения окончательных предсказаний классов требуется свернуть эти 16-канальные признаки в необходимое число классов. Для этого в конце сети используется свёртка Conv2D с ядром 1×1 , которая сопоставляет каждый вектор признаков размерности 16 к 5 выходным каналам – по числу классов. Эта операция даёт тензор $448 \times 448 \times 5$, где каждое значение отражает не нормированное значение активации соответствия данного пикселя определённому классу. Далее применяется softmax-преобразование по каналам, чтобы получить вероятности классов для каждого пикселя. Формально выход U-Net можно представить, как тензор, к которому покомпонентно применяется softmax, получая распределение вероятностей по классам для каждой координаты. В процессе обучения эти вероятности сравниваются с истинной разметкой через функцию потерь.

3.4 Функция потерь

Функция потерь (loss function) – это мера, которая оценивает, насколько хорошо модель предсказывает правильные метки. В процессе обучения модели семантической сегментации одним из наиболее важных аспектов является выбор подходящей функции потерь, поскольку именно она определяет то, как модель будет оценивать ошибки и корректировать свои

параметры. В представленной работе используется стандартная для задач многоклассовой сегментации функция категориальной перекрёстной энтропии [21].

Данная функция потерь особенно эффективна в случае задач сегментации изображений, где каждому пикселю изображения ставится в соответствие один из множества заранее заданных классов. Для каждого пикселя модель формирует вероятностное распределение по всем классам с помощью слоя активации softmax, а функция потерь измеряет, насколько полученные моделью вероятности отличаются от реального класса, указанного в разметке.

$$L = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \cdot \log(\hat{y}_{i,c}),$$

где N – количество пикселей в изображении;

C – количество классов;

$y_{i,c}$ – индикатор истинного класса для пикселя i и класса c ;

$\hat{y}_{i,c}$ – предсказанная моделью вероятность принадлежности пикселя i классу c .

Значение функции потерь минимизируется в процессе обучения, что приводит к увеличению вероятности правильных классов, назначаемых моделью каждому пикселю.

3.5 Обучение модели и анализ датасета

Для обучения разработанной модели семантической сегментации был подготовлен специализированный датасет спутниковых снимков. Датасет включает 200 изображений местности размером 448×448 пикселей с соответствующими масками-разметками. Распределение классов на этих масках существенно неравномерно: преобладают пиксели класса «поле», тогда как классы «вода» и «здания» встречаются реже всего.

Модель U-Net обучалась в среде TensorFlow на GPU NVIDIA с учетом ранней остановки по критерию минимизации функции потерь. В качестве функции потерь выбрана категориальная кросс-энтропия, взвешенная по классам, – это позволило компенсировать доминирование фона и уделить больше внимания редким объектам. Обучение проводилось на протяжении 50 эпох с использованием оптимизатора Adam ($\eta=0,001$).

Достигнутое качество сегментации оценивается метрикой Intersection over Union (IoU): совокупный IoU по пяти классам составил $\approx 0,78$, а средняя точность пикселей – порядка 85%. Эти результаты свидетельствуют о достаточной точности модели для практического применения [22].

4 Техническое задание

4.1 Основание для разработки

Разработка программного комплекса с использованием искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах ведется на основании задания, выданного руководителем дипломного проекта кандидатом технических наук, доцентом кафедры МОиПЭВМ Такташкиным Д.В. и утвержденного заведующим кафедрой ВТ Митрохиным М.А.

4.2 Назначение разработки

Разрабатываемый программный комплекс предназначен для автоматизированного распознавания, сегментации и последующего позиционирования объектов инфраструктуры на спутниковых и цифровых картографических изображениях. Комплекс реализован на основе современных методов глубокого обучения, в частности – сверточных нейронных сетей архитектуры U-Net [23].

4.3 Требования к программе

4.3.1 Требования к функциональным характеристикам

Программное обеспечение должно обеспечивать следующий функционал:

- получение спутниковых изображений заданного участка местности путём обращения к внешнему сервису Yandex Static Maps API по заданным координатам и уровню масштабирования,
- предобработка изображений (448×448, RGB),
- генерация схем позиционирования объектов на цифровой карте,
- сохранение результатов сегментации и позиционирования объектов,
- на основе результатов сегментации автоматически должны определяться подходящие участки для размещения объектов.

Программное обеспечение должно иметь интуитивно понятный интерфейс, обеспечивать комфортную работу для пользователей, не имеющих опыта работы с персональным компьютером.

4.3.2 Требования к надежности

Разрабатываемое программное обеспечение должно обладать устойчивостью к ошибкам, возникающим при взаимодействии с внешними API или сервисами, например, при отсутствии информации по заданным координатам. В подобных случаях пользователь должен получать понятные и пояснительные сообщения о возникших проблемах.

Также программный модуль должен обрабатывать случаи подачи входных изображений с неподходящими размерами или форматами. При необходимости такие данные должны быть автоматически преобразованы к требуемым параметрам, либо пользователю следует выдать уведомление с рекомендациями по корректировке.

Система должна обеспечивать воспроизводимость результатов: при повторной обработке одного и того же изображения модель обязана выдавать идентичную маску сегментации, тем самым подтверждая предсказуемость и стабильность работы алгоритма.

Программное обеспечение должно обеспечивать минимальное время восстановления работоспособности после сбоев или плановых перезапусков модели и сервисов, поддерживая непрерывность работы системы.

4.3.3 Требования к составу и параметрам технических средств

Для эффективного функционирования разрабатываемого программного комплекса необходимо обеспечить следующие технические средства:

Серверное оборудование:

Центральный процессор CPU: Intel Core i5 или эквивалентный с частотой не менее 2,5 ГГц.

Объём оперативной памяти RAM: не менее 8 Гб, рекомендуется 16 Гб.

Постоянное хранилище данных SSD или HDD: не менее 50 Гб свободного пространства.

Видеокарта: NVIDIA серии GTX/RTX с поддержкой CUDA, не менее 4 Гб памяти, для повышения производительности нейросетевой модели.

Рабочее место пользователя:

Персональный компьютер с веб-браузером (Google Chrome, Mozilla Firefox, Microsoft Edge последних версий).

Разрешение монитора не менее 1920×1080 пикселей для комфортного отображения веб-интерфейса.

Сетевое оборудование:

Стабильное подключение к сети Интернет с пропускной способностью не менее 10 Мбит/с для получения картографических данных и взаимодействия с внешними сервисами.

Использование оборудования с указанными характеристиками обеспечит стабильную и производительную работу разработанного программного комплекса в условиях эксплуатации.

4.3.4 Требования к информационной и программной совместимости

Разработка программного обеспечения должна осуществляться в среде разработки Microsoft Visual Studio Code. Сервисная часть ПО должна быть реализована на языке программирования Python 3.7 и выше. Для реализации нейросетевых компонентов ПО должно использовать фреймворки глубокого обучения TensorFlow (не ниже версии 2.x) и Keras [24].

Разработанное приложение должно работать под управлением операционной системы Windows 10 и выше. Веб-интерфейс должен обеспечивать совместимость с современными браузерами Google Chrome, Mozilla Firefox, Microsoft Edge последних версий.

4.4 Требования к программной документации

Программное обеспечение должно сопровождаться пояснительной запиской, включающей следующую программную документацию:

- техническое задание,
- описание программы,
- программа и методика испытаний,
- описание применения.

Структура и содержание программной документации соответствовать требованиям стандартам Единой системы программной документации (ЕСПД).

4.5 Стадии и этапы разработки

Разработка программного обеспечения должна осуществляться в соответствии со следующими этапами:

- предпроектное исследование,
- разработка и утверждение технического задания,
- разработка технического проекта,
- разработка и отладка программного обеспечения,
- испытания программного обеспечения,
- внедрение программного обеспечения.

Каждый из перечисленных этапов является неотъемлемой частью жизненного цикла программного обеспечения и обеспечивает последовательный и контролируемый процесс разработки. Соблюдение данной структуры позволяет минимизировать риски, повысить качество конечного продукта и обеспечить его соответствие требованиям заказчика и специфике решаемой задачи.

4.6 Порядок контроля и приемки

Для проверки правильности работы программного обеспечения должны быть разработаны тестовые примеры, в которых должны быть учтены все возможные сочетания входных данных, отражающие работу всех его частей. Должны быть протестированы действия программного обеспечения на возможные реакции пользователя: запуск программы, заполнение полей ввода, выбор пункта меню, нажатие кнопок.

Приёмка программного обеспечения должна включать проверку его работоспособности, корректности выполнения функций на основе тестовых данных и наличие пояснительной записки с полной программной документацией.

5 Разработка программного обеспечения

5.1 Алгоритм размещения объектов на карте

Алгоритм автоматического размещения объектов ПВР на выбранном участке реализован модульно и опирается на результаты сегментации карты. Ниже описана последовательность действий алгоритма.

Для поиска подходящей площадки анализируется результирующая маска сегментации местности. Алгоритм выделяет связные области класса «поле» достаточного размера, пригодные для размещения лагеря. Если подходящая область не найдена, процесс прекращается и пользователю выдаётся сообщение об ошибке «В данной области невозможно разместить все объекты. Попробуйте выбрать другое место».

Определяется требуемое число объектов для заданной вместимости лагеря. Например, число палаток рассчитывается исходя из норм плотности размещения людей по техническому заданию, N человек на палатку. Аналогично, на основе санитарных нормативов вычисляется минимально необходимое число туалетов и душевых, гарантируя минимум по 2 туалета и 2 душа даже для малого лагеря.

В границах выбранного поля размещаются палатки. Алгоритм расставляет палатки прямоугольной сеткой с учётом нормативных расстояний – не менее 2,5–3 м друг от друга. Для каждой новой палатки проверяется, что она не пересекается с уже размещёнными и сохраняет требуемый отступ. Размещение продолжается, пока не будет размещено заданное число палаток или пока свободное место не исчерпано [25].

После палаток алгоритм выделяет область под санитарные объекты – блок туалетов и душевых. Выбирается свободный участок на границе поля, отстоящий от палаток на требуемое расстояние ≥ 50 м. Все туалеты и душевые располагаются компактно, единым блоком. В коде решение реализовано попыткой «вписать» прямоугольный блок в свободное пространство поля. При успешном размещении объекты визуализируются,

например, туалеты помечаются условным синим цветом, душевые – голубым.

Алгоритм последовательно добавляет прочие элементы инфраструктуры. В частности, предусматривается установка мусорных контейнеров на краю поля с отступом ~25 м от жилой зоны и размещение генератора с запасом топлива на максимально удалённой точке поля ≥ 50 м от палаток. В коде реализован поиск точки, максимально удалённой от всех палаток, для размещения топливного блока.

Полевая кухня и медицинский пункт располагаются рядом друг с другом, образуя хозяйственно-медицинскую зону. Алгоритм пытается разместить кухню и медпункт, плечом к плечу в свободной части поля. При этом проверяется условие доступности: оба блока должны поместиться рядом на поле и с учётом минимального промежутка в несколько метров между ними и другими объектами.

После успешного размещения всех элементов формируется итоговая схема – карта с нанесёнными условными обозначениями всех объектов пункта временного развертывания. Эта схема сохраняется как изображение и выводится пользователю.

5.2 Диаграмма вариантов использования

При проектировании программного продукта необходимо определить функциональные возможности, взаимодействующие лица и взаимосвязи между ними. StarUML может позволить создать все необходимые UML-диаграммы, для определения структуры и функционала будущего приложения. Диаграмма вариантов использования отображает функционал будущего продукта и лица, которые будут взаимодействовать с ним. Разработанная диаграмма приведена ниже на рисунке 6.



Рисунок 6 – Диаграмма вариантов использования

Исходя из данной диаграммы, пользователь может выполнять следующие действия: вводить координаты вручную, выбирать точку на карте, получать карту, просматривать статистику и скачивать результат. Ключевым действием является получение карты, которое включает в себя обязательные этапы: генерацию маски, наложения, плана размещения и получение статистики. Эти этапы обозначены связями `<<include>>`, что указывает на их обязательное выполнение при получении карты.

5.3 Диаграмма развертывания

Диаграмма развёртывания используется для того, чтобы визуализировать аппаратные узлы системы, каналы связи между ними и размещение программных файлов. Благодаря описанным UML-диаграммам вариантов использования и развёртывания описана структура и функционал разрабатываемой системы. Диаграмма развертывания приведена на рисунке 7.

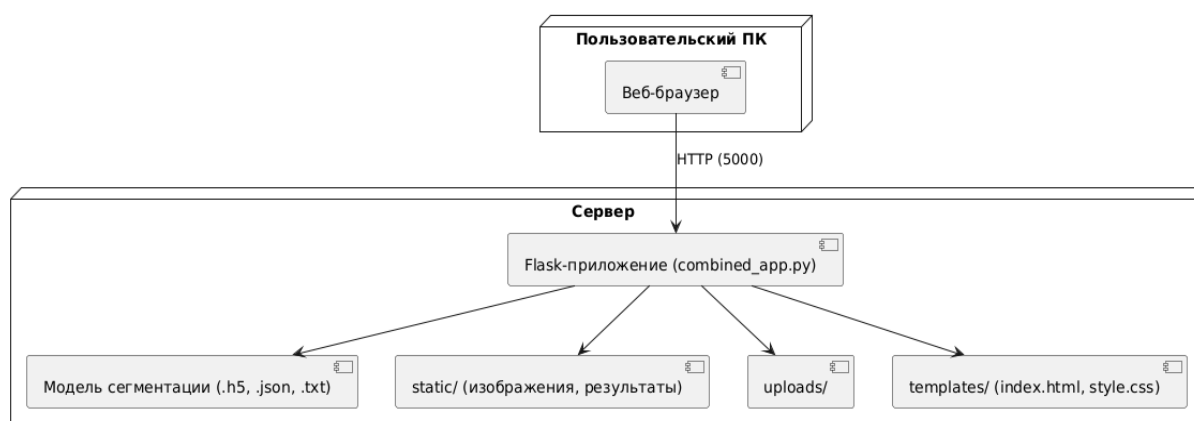


Рисунок 7 – Диаграмма развертывания

5.4 Разработка структуры приложения

На основе функциональной модели системы определены основные функции, которые должно выполнять программное обеспечение. Описание функций программного обеспечения представлено в таблице 2. Каждая из функций отражает ключевые требования к системе и обеспечивает реализацию поставленных задач в рамках заданной области применения.

Таблица 2 – Функции системы

Уровень	Наименование функции	Описание функции
1	Открытие приложения	Осуществить подключение к веб-приложению через браузер
2	Установка координат на карте	Установить маркер на карте для указания точки интереса
2.1	Запрос изображения по координатам	Получить изображение выбранной области через API внешнего сервиса
2.2	Обработка изображения нейросетевой моделью	Выполнить сегментацию изображения на классы объектов инфраструктуры
3	Генерация схемы размещения объектов	Автоматически сгенерировать схему размещения объектов ПВР на подходящих участках

Продолжение таблицы 2

3.1	Отображение результата	Визуализировать цветную маску сегментации и схему размещения на исходном изображении
4	Сохранение результата	Сохранить полученные изображения в архив

На основе определенных функций разработана схема «функции-модули», представленная на рисунке 8.



Рисунок 8 – Схема функции модули

Список функций и модулей представлен в таблице 3.

Таблица 3 – Списки функции и модулей

Функция	Модуль
Получение изображения по координатам	yandex_maps.py
Сегментация изображения с помощью	map_segmentation_service.py /

нейросети	segmentation_model
-----------	--------------------

Продолжение таблицы 3

Функция	Модуль
Формирование маски и overlay-визуализации	combined_app.py
Генерация схемы размещения объектов	встроено в combined_app.py
Веб-интерфейс и API взаимодействие	test_app.py / combined_app.py

Анализ структуры программного комплекса позволяет выделить основные функции и соответствующие им модули, реализующие ключевую логику системы. Такое соответствие обеспечивает модульность, упрощает поддержку, тестирование и возможную масштабируемость решения в будущем.

6 Описание программы

6.1 Общие сведения

Программное обеспечение предназначено для автоматизации процессов анализа цифровых карт и генерации инфраструктурных схем на основе спутниковых изображений.

Решение реализовано с использованием языка программирования Python, с применением библиотек TensorFlow, Flask, OpenCV и NumPy. Разработка и отладка велись в среде Visual Studio Code. Интерфейс предоставляется в виде веб-приложения, доступного через интернет-браузер.

6.2 Функциональное назначение

Программное обеспечение обеспечивает выполнение следующих функций:

- автоматическое получение спутникового изображения по координатам,
- сегментация изображения с помощью свёрточной нейронной сети,
- визуализация результатов в виде цветной маски и наложения на исходное изображение,
- генерация схемы размещения объектов инфраструктуры на подходящих участках,
- сохранение полученных результатов.

6.3 Используемые технические средства

Программное обеспечение реализуется как клиент-серверное веб-приложение.

В качестве сервера предполагается использование персонального компьютера с характеристиками не ниже следующих:

- процессор с тактовой частотой не ниже 2,8 ГГц,
- объем оперативной памяти не менее 16 ГБ,
- объем свободного места на жестком диске – не менее 10 ГБ,
- рекомендуется наличие видеокарты с поддержкой CUDA, например, NVIDIA GTX/RTX для ускорения инференса модели.

Рабочее место пользователя должно обеспечивать доступ к веб-браузеру и стабильное интернет-соединение.

6.4 Вызов и загрузка

Для запуска программного обеспечения необходимо:

- запустить серверную часть (Flask-приложение),
- открыть клиентскую часть через браузер, подключившись к локальному адресу или удалённому серверу,
- установить координату на карте и инициировать сегментацию нажатием соответствующей кнопки.

При запуске программного обеспечения локально взаимодействие происходит через сервер localhost.

6.5 Входные данные

Входными данными являются:

- координаты (широта и долгота) выбранной области,
- изображение карты, полученное по этим координатам,
- параметры, встроенные в модель, для идентификации классов.

После установки координат автоматически выполняется запрос к внешнему картографическому API, например, Yandex Static Maps для получения соответствующего спутникового изображения. Полученное изображение преобразуется к требуемому формату и разрешению (448×448 пикселей, RGB) и подаётся на вход свёрточной нейронной сети.

6.6 Выходные данные

Выходными данными программного комплекса являются:

- сегментационная маска изображения, классифицирующая каждый пиксель по типу объекта (здание, дорога, вода, лес, поле),
- визуализированное наложение маски (overlay) на исходное изображение,
- схема размещения объектов инфраструктуры (палаток, санитарных зон и т.д.), построенная на основе сегментации,

– итоговые файлы (изображения и схемы), пригодные для дальнейшей обработки или интеграции в ГИС.

Все выходные изображения могут быть сохранены в формате PNG или переданы в другие модули системы для последующей обработки или интеграции в геоинформационные платформы.

7 Программа и методика испытаний

7.1 Объект испытаний

Объектом испытаний является программное обеспечение для автоматического получения изображений, их сегментации с использованием сверточной нейронной сети и генерации схем размещения объектов инфраструктуры.

7.2 Цель испытаний

Целью испытаний является проверка корректности функционирования программного обеспечения, а также установление соответствия реализованных функций требованиям, изложенным в техническом задании.

7.3 Средства и порядок испытаний

Проведение испытаний программного обеспечения должно осуществляться на персональных компьютерах с минимальными характеристиками, указанными в техническом задании.

Для проверки работоспособности используются заранее подготовленные тестовые сценарии, включающие:

- ввод координат для получения карты,
- автоматическое получение изображения,
- выполнение сегментации,
- генерацию схемы размещения,
- визуализацию и сохранение результата.
- Эти сценарии позволят подтвердить корректность выполнения ключевых этапов

7.4 Методы испытаний

Проверка правильности работы программного обеспечения будет включать в себя верификацию всех функциональных возможностей, изложенных в техническом задании. Основной метод испытаний – функциональное тестирование, направленное на проверку реакции системы на различные сценарии ввода и корректность выполнения ключевых операций.

После открытия веб-страницы появляется окно с картой, управляющими элементами и кнопками, обеспечивающими ввод координат и запуск сегментации (рисунок 9).

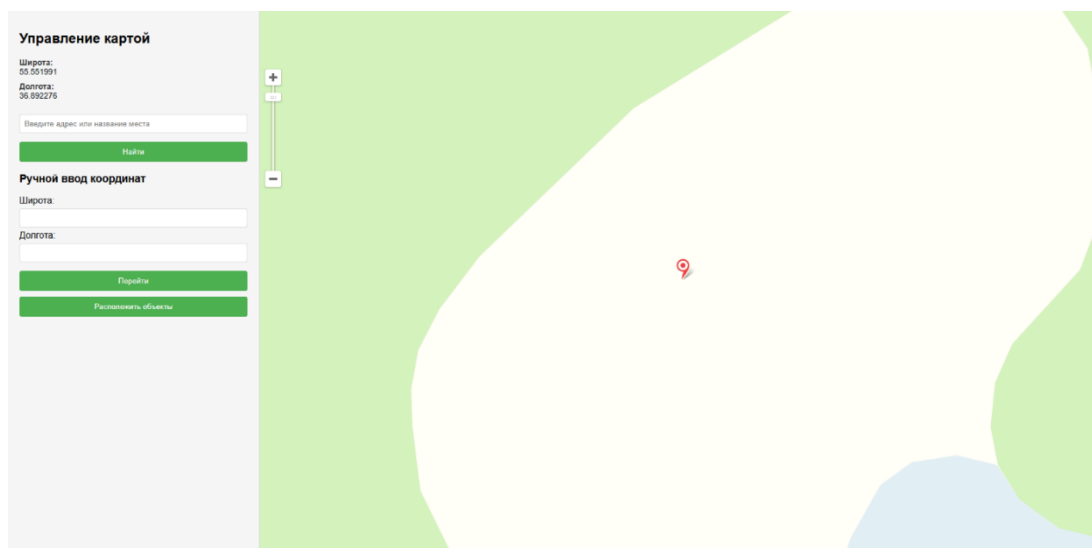


Рисунок 9 – Главное окно

Пользователь вводит координаты вручную или выбирает точку на карте, после чего инициирует загрузку спутникового изображения и передачу его на нейросетевой модуль.

Испытания включают последовательную проверку следующих действий:

- корректность загрузки карты и визуализации интерфейса,
- корректное выполнение запроса к картографическому API и получение изображения,
- успешная подача изображения в модель и получение маски сегментации,
- генерация схемы размещения объектов ПВР на основе сегментации,
- проверка сообщений об ошибках при некорректном вводе или неподходящих условиях,
- возможность сохранения результата и визуального анализа сегментированного изображения.

Функциональные блоки тестируются изолированно и в составе общего бизнес-процесса, включая негативные сценарии, например, отсутствие

подходящего поля или сбой подключения к внешним сервисам. Оценка результатов производится путём сопоставления фактического поведения системы с ожидаемым, указанным в спецификациях.

С помощью ввода названия населенного пункта можно переместить маркер в искомую точку. Если оставить поле пустым и нажать кнопку «Найти» должно выйти сообщение «Введите адрес или название места» (рисунок 10).



Рисунок 10 – Уведомление о пустом поле вводе поиска

Если попытаться перейти по координатам, не заполнив поле ввода, появится уведомление об ошибке. Продемонстрировано на рисунке 11.

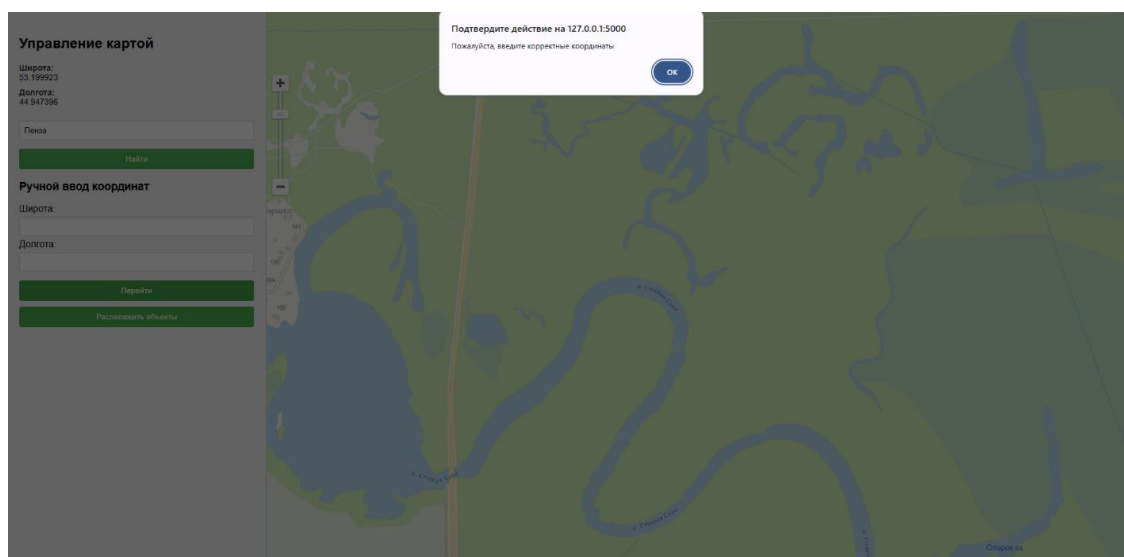


Рисунок 11 – Уведомление о пустом поле вводе поиска

Если попытаться разместить объекты там, где недостаточно места для объектов или нет вовсе, система выдаст текст об ошибке и прекратит сегментировать карту. Сообщение об ошибке представлено на рисунке 12.

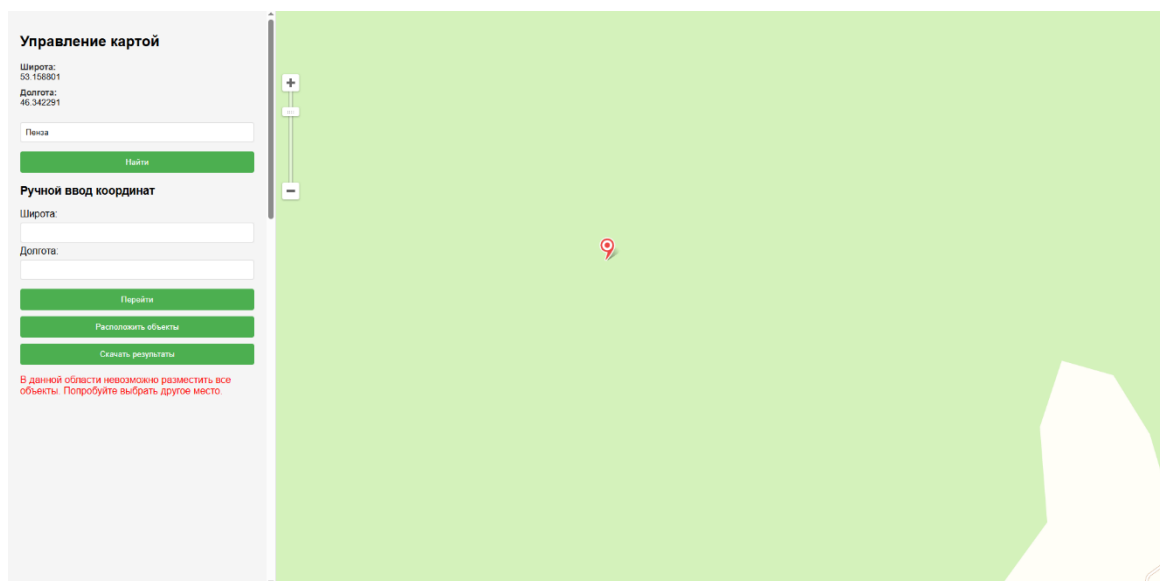


Рисунок 12 – Уведомление о неподходящем месте для размещения объектов

Если место, где установлен маркер устраивает, после нажатия на кнопку «Расположить объекты», начнется обработка изображения после чего в левом столбце появится кнопка «Скачать результаты» и изображения с сегментацией карты, размещением объектов на участке карты вокруг маркера и т.д. Для загрузки будет представлен архив с изображениями. Успешное выполнение программы выдаст несколько изображений (рисунки 13–16).

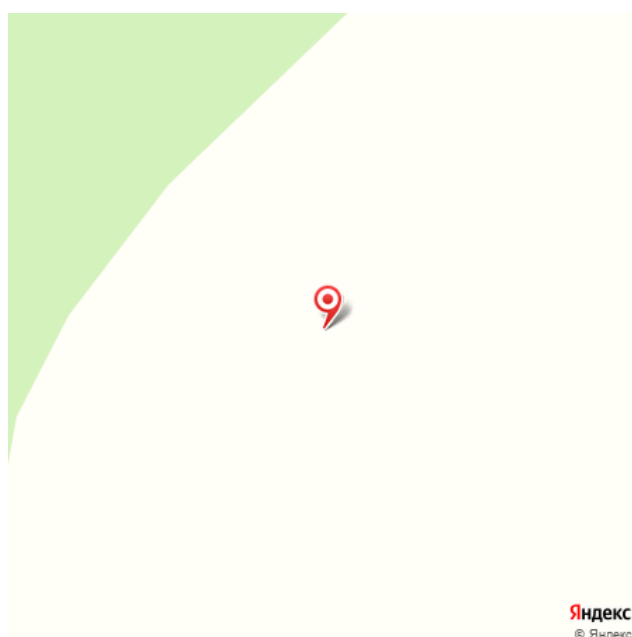


Рисунок 13 – Карта без разметки

На изображении представлена исходная карта, полученная с использованием внешнего картографического сервиса. Установленный маркер указывает на выбранную пользователем точку интереса, вблизи границы растительности и открытого пространства.

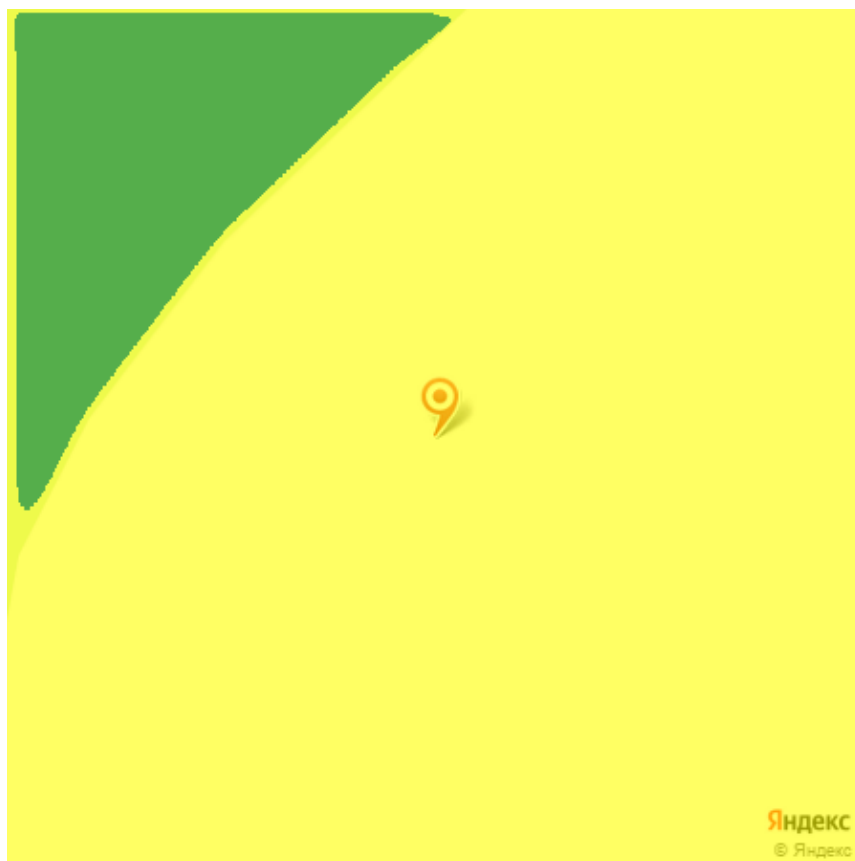


Рисунок 14 – Маска цветов поверх карты

На изображении показан результат семантической сегментации. Каждому пикселю присвоен определённый класс местности. Цвета обозначают следующие классы. Зелёный – лесной массив, Жёлтый – открытое поле. Маркер сохранён для наглядного сопоставления.

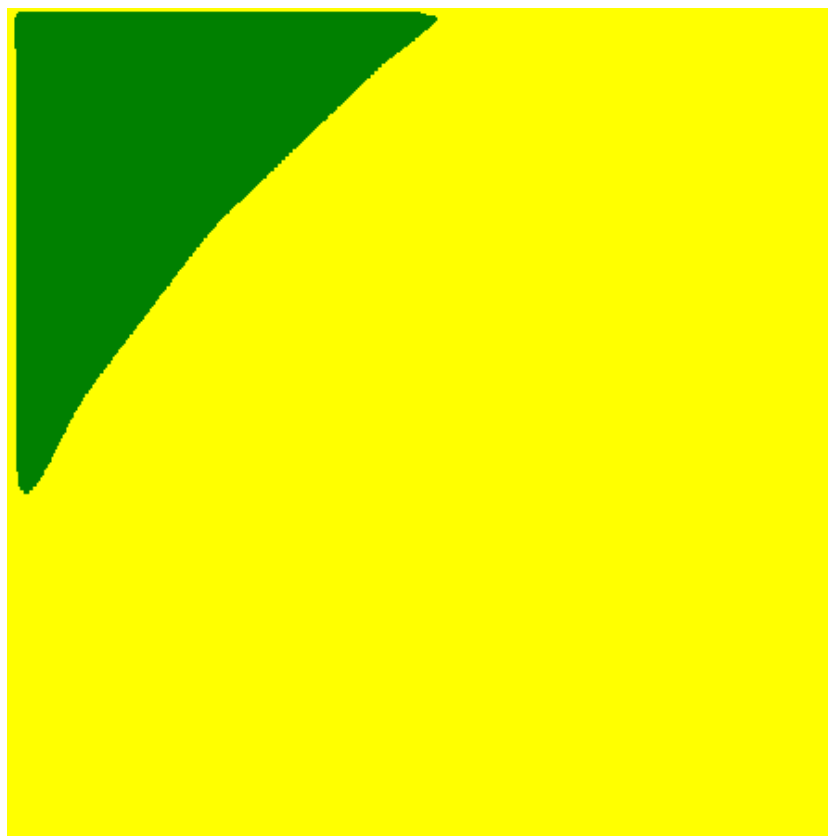


Рисунок 15 – Разметка, полученная из анализа карты

На рисунке представлена бинаризованная сегментационная маска, сформированная нейросетевой моделью. Здесь жёлтым цветом выделена область, классифицированная как «поле» – пригодная для размещения объектов инфраструктуры, а зелёным – участок леса, не подлежащий использованию. Эта карта является промежуточным результатом анализа, на основании которого принимается решение о возможности генерации схемы размещения ПВР.

Таблица 4 – Сценарии функционального тестирования программы

Сценарий испытаний	Описание и условия	Ожидаемый результат	Фактический результат
Получение изображения (корректный ввод)	Пользователь указывает корректные координаты местности и нажимает «Найти».	Отображается спутниковая карта заданного участка; сервер возвращает изображение.	Изображение успешно получено и выведено на карту.
Получение изображения (ошибка ввода)	Не задано название места и координаты; нажата кнопка поиска.	Появляется предупреждение «Введите адрес или название места».	Выведено корректное уведомление об отсутствующих данных.
Сегментация карты (успешный случай)	После получения изображения пользователь нажимает «Распознать объекты». На выбранном участке присутствуют различные объекты.	Модель сегментирует изображение; слева отображается цветная маска и кнопка «Скачать результаты».	Сегментация выполнена, маска накладывается на карту; кнопка скачивания активна.
Размещение ПВР (нет свободного поля)	Пользователь выбрал точку на карте, где отсутствуют пригодные поля (например, сплошная застройка или водоём).	Программа выводит сообщение об ошибке и не выполняет сегментацию.	Выдано предупреждение «В данной области...», алгоритм корректно остановлен.

Продолжение таблицы 4

Сценарий испытаний	Описание и условия	Ожидаемый результат	Фактический результат
Размещение ПВР (успешный случай)	Пользователь выбрал координаты с большим полем; нажата кнопка «Разместить объекты».	На карту накладываются значки палаток, кухни, пр. (схема ПВР); предлагается скачать архив с изображениями.	Объекты успешно размещены; план лагеря отображён; архив доступен для скачивания.

Из таблицы 4 видно, что все основные функции программного комплекса прошли проверку. Программа предсказуемо реагирует на некорректный ввод данных, выдавая понятные сообщения пользователю (сценарии 2 и 4), и корректно выполняет ключевые задачи по сегментации и генерации схемы размещения объектов (сценарии 3 и 5). Кроме перечисленных, проведены дополнительные тесты на производительность: среднее время сегментации изображения размером 448×448 составило ~0,4 с на GPU и ~1,2 с на CPU, что укладывается в допустимые рамки для интерактивного веб-приложения.

Таким образом, результаты испытаний подтверждают работоспособность и надёжность разработанного комплекса.

8 Описание применения

8.1 Назначение применения

Программное обеспечение предназначено для автоматизированного анализа изображений цифровых карт и размещения объектов с использованием нейросети. Система применяется в задачах пространственного планирования, ситуационного реагирования и визуального анализа геоданных с использованием сверточных нейронных сетей.

8.2 Условия применения

Программное обеспечение реализовано в виде клиент-серверного веб-приложения.

В качестве сервера должен использоваться персональный компьютер с минимальными характеристиками:

- процессор с тактовой частотой не ниже 2,8 ГГц,
- объём оперативной памяти не менее 16 ГБ,
- наличие видеокарты с поддержкой CUDA (рекомендуется, но не обязательно),
- подключение к сети Интернет для работы с внешними картографическими API.

В качестве пользовательского рабочего места применяется любой персональный компьютер или ноутбук с установленным современным браузером Google Chrome, Mozilla Firefox и др. и стабильным интернет-соединением.

8.3 Описание задачи

После запуска программного обеспечения в браузере открывается веб-интерфейс с картой и панелью управления.

Пользователь устанавливает точку интереса на карте, после чего нажимает кнопку для размещения объектов, изображение обрабатывается нейросетевой моделью и если разметка карты включает в себя поля, то мы можем разместить там пункт временного развертывания.

Результат отображается в виде наложенной цветной маски и графических обозначений, показывающих размещение элементов. Пользователь может сохранить архив с изображениями, обработанными нейросетью, при скачивании пользователь получает результат разметки и размещения объектов (рисунки 16–19).

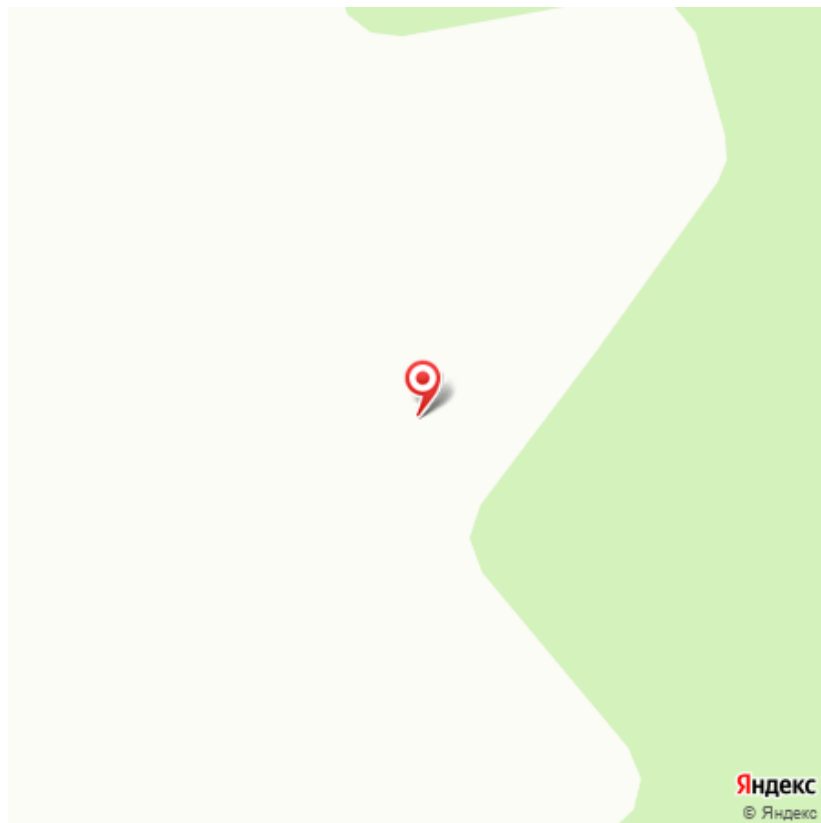


Рисунок 16 – Карта без разметки

На данном изображении показана исходная, неизменённая карта участка, полученная до запуска нейросетевой обработки. Маркер указывает выбранную пользователем точку интереса. На этом этапе изображение ещё не прошло сегментацию и не содержит элементов разметки или схемы размещения объектов.



Рисунок 17 – Маска цветов поверх карты

На изображении представлена карта с наложенной маской сегментации, полученной в результате обработки нейросетевой моделью. Жёлтым цветом обозначена область, классифицированная как «поле», зелёным – участок леса. Цветовая маска накладывается поверх оригинального изображения (overlay), сохраняя исходную географическую ориентацию и визуальные ориентиры. Маркер остаётся на месте для сопоставления с предыдущим изображением.

На следующем этапе маска преобразуется в итоговую форму – удаляется подложка карты, а результат сегментации сохраняется в виде чистой разметки, где каждый пиксель закодирован в соответствии с классом объекта. Это упрощает последующую работу с изображением, в частности – проведение анализа пригодности участка и размещение элементов ПВР.



Рисунок 18 – Разметка, полученная из анализа карты

На изображении представлена итоговая маска сегментации, сформированная нейросетевой моделью. Здесь каждому пикселю присвоен один из классов: жёлтым цветом обозначено поле (открытая территория), зелёным – лесной массив. Эта карта служит основой для анализа пригодности участка и последующего автоматического размещения объектов ПВР. Визуализация упрощена и не содержит исходного спутникового изображения – отображена только классификация поверхности.

На основе итоговой маски сегментации, содержащей классифицированные области местности, система переходит к этапу построения схемы размещения объектов ПВР. Алгоритм автоматически определяет пригодные участки (в данном случае – поле), рассчитывает размещение палаток, санитарных, технических и хозяйственных объектов с учётом нормативных отступов, и формирует финальную схему, представленную на следующем изображении.

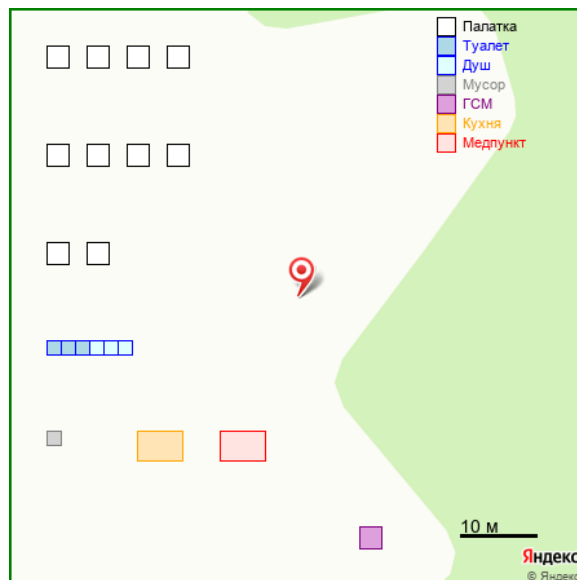


Рисунок 19 – Карта с размещенными объектами

На изображении представлена итоговая схема размещения объектов пункта временного развертывания, автоматически сгенерированная на основе сегментации местности. Объекты инфраструктуры расположены в соответствии с нормативными требованиями: палатки (чёрные), санитарная зона с туалетами и душевыми (оттенки синего), пункт питания (оранжевый), медицинская палатка (красная), контейнер для мусора (серый) и блок с GSM (фиолетовый). Маркер указывает исходную точку, выбранную пользователем. Масштабная линейка в правом нижнем углу позволяет визуально оценить расстояния между объектами.

8.4 Инструкция развертывания

Для запуска backend-части приложения требуется установленный Python версии 3.7 или выше, а также необходимые зависимости, указанные в файле requirements.txt.

Откройте терминал и перейдите в каталог проекта:

```
cd ./map-segmentation-app
```

Затем запустите Flask-приложение, выполнив команду:

```
python combined_app.py
```

После запуска backend-сервер будет доступен по адресу <http://localhost:5000/>, и станет возможна обработка запросов на сегментацию и генерацию схем.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была изучена предметная область автоматической семантической сегментации спутниковых изображений и генерации инфраструктурных схем. Определена актуальность разработки программного обеспечения, способного распознавать такие объекты как здания, дороги, водоёмы, леса и поля на цифровых картах с применением нейросетевых методов. Проведён анализ существующих решений в области геоинформационных технологий и компьютерного зрения, а также сравнительный обзор архитектур сверточных нейронных сетей, используемых для задач сегментации.

На основе анализа было обосновано применение архитектуры U-Net, как одной из наиболее эффективных для семантической сегментации с ограниченными вычислительными ресурсами. Использование собственной, «чистой» реализации U-Net без предобученного энкодера позволило достичь баланса между точностью и быстродействием модели, при этом количество параметров удалось сохранить в пределах 487 тыс., что делает модель пригодной для использования в прикладных веб-сервисах.

В рамках проекта разработан программный комплекс, включающий backend на Python (Flask + TensorFlow/Keras) и веб-интерфейс, взаимодействующий с пользователем через карту. Приложение реализует полный цикл обработки: от получения изображения по координатам до визуализации сегментации и генерации схемы размещения объектов.

При разработке алгоритма автоматического размещения объектов ПВР использовались положения ГОСТ Р 22.3.18–2021 «Безопасность в чрезвычайных ситуациях. Пункты временного размещения населения. Общие требования», а также санитарные и пространственные нормативы, регламентирующие расположение санитарных зон, палаток, пунктов питания и объектов жизнеобеспечения. Это обеспечивает соответствие сгенерированных схем установленным государственным стандартам, что

особенно важно при применении комплекса в реальных условиях ЧС, мобилизационного планирования или инфраструктурного проектирования.

Проведённые испытания подтвердили корректность работы программного обеспечения и соответствие его функциональности требованиям технического задания.

Разработка представленного программного комплекса позволяет автоматизировать трудоёмкие процессы картографического анализа и может быть использована в системах гражданской обороны, градостроительного планирования. На основе проделанной работы можно сделать вывод о целесообразности дальнейшего развития проекта и его интеграции в прикладные геоинформационные решения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рыбаков Д. А. Актуальность и доступность нейросетей в современном обществе // Вестник науки. 2023. №7 (64). URL: <https://cyberleninka.ru/article/n/aktualnost-i-dostupnost-neyrosetey-v-sovremennom-obschestve> (дата обращения: 03.12.2023);
2. Питенко А.А. Нейросетевой анализ в ГИС. // Автореф. дисс. канд. техн. наук 05.13.16, Красноярск, 2000;
3. Галушкин, А.И. Нейронные сети: основы теории. / А.И. Галушкин. - М.: РиС, 2014. – 496 с.;
4. Галушкин, А.И. Нейронные сети: история развития теории: Учебное пособие для вузов. / А.И. Галушкин, Я.З. Цыпкин. - М.: Альянс, 2015. – 840 с.;
5. Хайкин Саймон, Нейронные сети: полный курс, 2-е издание.: Пер. с англ. - М.: Издательский дом «Вильямс», 2006. – 1104 с.: ил. – Парал. тит. англ.
6. Экимов Д. В., Такташкин Д. В. Использование искусственных нейронных сетей для решения задач позиционирования объектов на цифровых картах // Вестник Пензенского государственного университета. 2025. № 1. С. 140–144.
7. Python Software Foundation. *Python 3.7. Official documentation*. – URL: <https://docs.python.org/3.7/> (дата обращения: 16.03.2024).
8. Шолле Ф. Глубокое обучение на Python / Ф. Шолле ; пер. с англ. – СПб. : Питер, 2018. – 384 с.
9. JavaScript – Википедия [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/JavaScript> (Дата последнего обращения: 13.03.2025)
10. Flask Documentation. URL: <https://flask.palletsprojects.com> (дата обращения: 16.03.2024).
11. TensorFlow Documentation [Электронный ресурс]. URL: <https://www.tensorflow.org/> (дата обращения: 16.03.2024).

12. Scikit-learn Documentation [Электронный ресурс]. URL: <https://scikit-learn.org/> (дата обращения: 17.03.2024).
13. LabelMe: Image Polygonal Annotation Tool. URL: <https://github.com/wkentaro/labelme> (дата обращения: 26.06.2025)
14. Visual Studio Code Documentation. URL: <https://code.visualstudio.com/docs> (дата обращения: 26.06.2025).
15. OpenCV Documentation [Электронный ресурс]. URL: <https://docs.opencv.org/> (дата обращения: 16.03.2024).
16. Pillow (PIL Fork) Documentation. URL: <https://pillow.readthedocs.io> (дата обращения: 18.03.2024).
17. U-Net: CNN для сегментации изображений – Ultralytics (на русском языке). URL: <https://www.ultralytics.com/ru/glossary/u-net> (дата обращения: 25.03.2023)
18. Николенко С.В. Глубокое обучение. Погружение в мир нейронных сетей. – М.: Олимп-Бизнес, 2018. – 400 с.
19. Гафаров Ф. М., Галимянов А. Ф. Искусственные нейронные сети и их приложения: учебное пособие / Ф.М. Гафаров, А. Ф. Галимянов. – Казань: Изд-во Казанского университета, 2018. – 121 с.
20. Сойер П. Цифровая обработка изображений / П. Сойер. – М.: Мир, 2006. – 456 с.
21. Перминов А. И., Турдаков Д. Ю., Беляева О. В. Функции потерь для обучения моделей сегментации изображений документов. Труды ИСП РАН, т. 34, вып. 2, 2022, стр. 89–110.
22. Головкин В. Л. Искусственные нейронные сети: основы, модели, обучение. – Минск: ТетраСистемс, 2021. – 456 с.
23. Гудфеллоу Я., Бенджио Ю., Курвиль А. Глубокое обучение / Я. Гудфеллоу, Ю. Бенджио, А. Курвиль ; пер. с англ. – М. : ДМК Пресс, 2018. – 656 с.
24. Keras Documentation [Электронный ресурс]. URL: <https://keras.io/> (дата обращения: 10.04.2025).

25. ГОСТ Р 22.3.18–2021. Безопасность в чрезвычайных ситуациях. Пункты временного размещения населения, пострадавшего в чрезвычайной ситуации. Общие требования и порядок приемки в эксплуатацию / утв. 09.02.2021 г. Федеральным агентством по техническому регулированию и метрологии (Росстандарт). – М.: Стандартинформ, 2021. – 56 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Использование искусственных нейронных сетей для решения задач
позиционирования объектов на цифровых картах

Листинг программы

ПРИЛОЖЕНИЕ А.1

Файл «combined_app.py»

```
"""
```

```
Объединенное Flask-приложение для работы с картами и их  
сегментации
```

```
Включает функционал получения карт через Yandex Maps API и их  
сегментацию
```

```
"""
```

```
from flask import Flask, render_template, jsonify, request,  
send_file, send_from_directory, Response  
from yandex_maps import YandexMapsAPI
```

```
import os
```

```
import base64
```

```
import io
```

```
import numpy as np
```

```
from PIL import Image
```

```
import cv2
```

```
from werkzeug.utils import secure_filename
```

```
from map_segmentation_service import get_segmentation_service
```

```
import math
```

```
import zipfile
```

```
import urllib.parse
```

```
app = Flask(__name__, static_folder='static')
```

```
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 # 16MB max
```

```
# Инициализация API ключей Яндекс Карт
```

```
STATIC_API_KEY = "56294bfb-d75f-4c06-8ca9-70ec176c8b84" # Ключ  
для Static API
```

```
JS_API_KEY = "283de992-5692-49cd-9a1c-e005992579c1" # Ключ для  
JavaScript API и геокодера
```

```
# Инициализация API
```

```
maps_api = YandexMapsAPI(api_key=STATIC_API_KEY)
```

```

# Конфигурация папок
UPLOAD_FOLDER = 'uploads'
OUTPUT_FOLDER = 'static/output' # Перемещаем output в static
IMAGE_FOLDER = 'static/images'

# Создание необходимых директорий
for folder in [UPLOAD_FOLDER, OUTPUT_FOLDER, IMAGE_FOLDER]:
    os.makedirs(folder, exist_ok=True)

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'bmp'}

def allowed_file(filename):
    """Проверка допустимого расширения файла"""
    return '.' in filename and filename.rsplit('.',
1)[1].lower() in ALLOWED_EXTENSIONS

def array_to_base64(image_array):
    """Конвертация numpy array в base64 для передачи в веб"""
    if len(image_array.shape) == 3:
        image = Image.fromarray(image_array.astype(np.uint8))
    else:
        image = Image.fromarray(image_array.astype(np.uint8),
mode='L')

    buffer = io.BytesIO()
    image.save(buffer, format='PNG')
    img_str = base64.b64encode(buffer.getvalue()).decode()
    return f"data:image/png;base64,{img_str}"

def get_meters_per_pixel(lat, zoom):
    """Вычисляет сколько метров в 1 пикселе для Web Mercator"""
    return 156543.03392 * math.cos(math.radians(lat)) / (2 **
zoom)

def get_bounds(lat, lon, zoom, size):
    meters_per_pixel = get_meters_per_pixel(lat, zoom)

```

```

    half_width = (size[0] / 2) * meters_per_pixel
    half_height = (size[1] / 2) * meters_per_pixel
    dlat = half_height / 111320
    dlon = half_width / (40075000 * math.cos(math.radians(lat))
/ 360)
    return [
        [lat + dlat, lon - dlon], # левый верхний
        [lat - dlat, lon + dlon], # правый нижний
    ]

@app.route('/')
def index():
    """Главная страница с картой"""
    return render_template('index.html', api_key=JS_API_KEY)

@app.route('/segmentation')
def segmentation():
    """Страница сегментации"""
    return render_template('segmentation.html')

@app.route('/api/info')
def api_info():
    """Информация о сервисе сегментации"""
    service = get_segmentation_service()
    return jsonify({
        'status': 'running',
        'model_loaded': service.model is not None,
        'classes': service.class_names,
        'input_size': service.input_size,
        'class_colors': {name: service.class_colors[i] for i,
name in enumerate(service.class_names)}
    })

@app.route('/save_location', methods=['POST'])
def save_location():
    """Сохранение и сегментация карты по координатам"""

```

```

try:
    data = request.json
    if not data:
        return jsonify({
            'success': False,
            'message': 'Не получены данные JSON'
        }), 400

    lat = data.get('lat')
    lon = data.get('lon')
    datetime = data.get('datetime')
    coords = data.get('coords')
    zoom = int(data.get('zoom', 19))
    size = tuple(data.get('size', (450, 450)))

    if lat is None or lon is None:
        return jsonify({
            'success': False,
            'message': 'Не указаны координаты (lat, lon)'
        }), 400

    filename = f"{datetime};{coords}.png"
    full_path = os.path.join(IMAGE_FOLDER, filename)
    print(f"Сохранение карты: lat={lat}, lon={lon}")
    maps_api.save_map(
        lat=lat,
        lon=lon,
        filename=full_path,
        zoom=zoom,
        size=size,
        pt=f"{lon},{lat},pm2rdm"
    )
    if not os.path.exists(full_path):
        raise Exception("Файл не был создан")
    service = get_segmentation_service()
    image = Image.open(full_path).convert('RGB')

```

```

print("Начинаю сегментацию...")
results = service.segment_image(image)
print("Сегментация завершена!")
segmentation_filename = f"segmentation_{filename}"
overlay_filename = f"overlay_{filename}"
pvr_plan_filename = f"pvr_plan_{filename}"
segmentation_path = os.path.join(OUTPUT_FOLDER,
segmentation_filename)
overlay_path = os.path.join(OUTPUT_FOLDER,
overlay_filename)
pvr_plan_path = os.path.join(OUTPUT_FOLDER,
pvr_plan_filename)

Image.fromarray(results['colored_mask']).save(segmentation_path)
Image.fromarray(results['overlay']).save(overlay_path)
meters_per_pixel = get_meters_per_pixel(float(lat),
zoom)

num_people = 30
pvr_img, placement_success = service.draw_pvr_on_field(
    results['mask'],
    image_size=service.input_size,
    meters_per_pixel=meters_per_pixel,
    num_people=num_people,
    save_path=pvr_plan_path,
    base_image=image
)

if not placement_success:
    return jsonify({
        'success': False,
        'message': 'В данной области невозможно
разместить все объекты. Попробуйте выбрать другое место.'
    }), 400

bounds = get_bounds(float(lat), float(lon), zoom, size)
return jsonify({

```

```

        'success': True,
        'message': 'Карта успешно сохранена и
сегментирована',
        'image_url': f'/static/images/{filename}',
        'segmentation_url':
f'/static/output/{segmentation_filename}',
        'overlay_url': f'/static/output/{overlay_filename}',
        'pvr_plan_url':
f'/static/output/{pvr_plan_filename}',
        'meters_per_pixel': meters_per_pixel,
        'num_people': num_people,
        'statistics': results['statistics'],
        'class_names': results['class_names'],
        'class_colors': results['class_colors'],
        'bounds': bounds
    })
except Exception as e:
    print(f"Ошибка при обработке: {str(e)}")
    return jsonify({
        'success': False,
        'message': f'Ошибка при обработке: {str(e)}'
    }), 500

@app.route('/api/segment', methods=['POST'])
def segment_image():
    """API для сегментации загруженного изображения"""
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'Нет файла изображения'}),
400

        file = request.files['image']
        if file.filename == '':
            return jsonify({'error': 'Файл не выбран'}), 400

        if not allowed_file(file.filename):

```

```

        return jsonify({'error': 'Неподдерживаемый формат
файла'}), 400

    # Загружаем изображение
    image = Image.open(file.stream).convert('RGB')
    original_size = image.size

    # Получаем сервис сегментации
    service = get_segmentation_service()

    # Выполняем сегментацию
    print("Начинаю сегментацию...")
    results = service.segment_image(image)
    print("Сегментация завершена!")

    # Конвертируем результаты в base64
    response_data = {
        'success': True,
        'original_size': original_size,
        'processed_size': service.input_size,
        'colored_mask':
array_to_base64(results['colored_mask']),
        'overlay': array_to_base64(results['overlay']),
        'statistics': results['statistics'],
        'class_names': results['class_names'],
        'class_colors': results['class_colors']
    }

    return jsonify(response_data)

except Exception as e:
    print(f"Ошибка сегментации: {e}")
    return jsonify({'error': f'Ошибка обработки:
{str(e)}'}), 500

@app.route('/api/test')

```

```

def test_segment():
    """Тестовая сегментация с примером"""
    try:
        service = get_segmentation_service()

        # Создаем тестовое изображение
        test_image = np.random.randint(0, 255, (448, 448, 3),
dtype=np.uint8)
        test_image_pil = Image.fromarray(test_image)

        # Выполняем сегментацию
        results = service.segment_image(test_image_pil)

        response_data = {
            'success': True,
            'message': 'Тестовая сегментация выполнена',
            'statistics': results['statistics'],
            'class_names': results['class_names']
        }

        return jsonify(response_data)

    except Exception as e:
        return jsonify({'error': f'Ошибка тестирования: {str(e)}'}), 500

@app.route('/download_results')
def download_results():
    """Скачать только текущие результаты в виде архива"""
    from io import BytesIO

    # Получаем параметры из query string
    files = request.args.getlist('file')
    if not files:
        return 'Не указаны файлы для архивации', 400
    memory_file = BytesIO()
    with zipfile.ZipFile(memory_file, 'w') as zf:

```

```

        for rel_path in files:
            # Декодируем URL и убираем ведущий слэш
            rel_path
        urllib.parse.unquote(rel_path).lstrip('/')
        abs_path = os.path.join(app.root_path, rel_path)
        if os.path.exists(abs_path):
            zf.write(abs_path, os.path.basename(abs_path))
        memory_file.seek(0)
        return send_file(memory_file, download_name='results.zip',
as_attachment=True)

if __name__ == '__main__':
    print("Запуск объединенного приложения карт и сегментации")
    print("Доступно по адресу: http://localhost:5000")
    print("Карта: http://localhost:5000/")
    print("Тест API: http://localhost:5000/api/test")
    print("Информация: http://localhost:5000/api/info")

    app.run(debug=True, host='0.0.0.0', port=5000)

```

ПРИЛОЖЕНИЕ А.2

Файл «yandex_maps.py»

```
import os
import requests
from PIL import Image
from io import BytesIO

class YandexMapsAPI:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://static-maps.yandex.ru/1.x/"

    def get_map(self, lat, lon, zoom=12, size=(450, 450), map_type="map",
pt=None):
        """
        Получение статической карты
        :param lat: Широта центра карты
        :param lon: Долгота центра карты
        :param zoom: Уровень масштабирования (0-17)
        :param size: Размер изображения (ширина, высота)
        :param map_type: Тип карты (map - схема, sat - спутник, sat,skl -
спутник с названиями)
        :param pt: Маркер на карте в формате "lon,lat,pm2rdm"
        :return: объект PIL.Image
        """
        params = {
            "ll": f"{lon},{lat}",
            "z": zoom,
            "size": f"{size[0]},{size[1]}",
            "l": map_type,
            "apikey": self.api_key
        }

        if pt:
            params["pt"] = pt

        response = requests.get(self.base_url, params=params)
        if response.status_code == 200:
            return Image.open(BytesIO(response.content))
        else:
```

```
        raise Exception(f"Ошибка при получении карты:
{response.status_code}")
```

```
def save_map(self, lat, lon, filename, **kwargs):
    """
    Сохранение карты в файл
    """
    image = self.get_map(lat, lon, **kwargs)
    image.save(filename)
    print(f"Карта сохранена в файл: {filename}")

def save_satellite(self, lat, lon, filename, zoom=16, size=(450, 450),
with_labels=True):
    """
    Сохранение спутникового снимка
    :param lat: Широта
    :param lon: Долгота
    :param filename: Имя файла для сохранения
    :param zoom: Уровень масштабирования (0-17)
    :param size: Размер изображения
    :param with_labels: Добавлять ли названия объектов
    """
    map_type = "sat,skl" if with_labels else "sat"
    self.save_map(
        lat=lat,
        lon=lon,
        filename=filename,
        zoom=zoom,
        size=size,
        map_type=map_type
    )
```

ПРИЛОЖЕНИЕ А.3

Файл «map_segmentation_service.py»

```
"""
Сервис для сегментации спутниковых карт
Использует обученную U-Net модель для определения:
- Лесов, полей, дорог, воды, болот
"""

import numpy as np
import cv2
import tensorflow as tf
from PIL import Image, ImageDraw, ImageFont
import json
import os

class MapSegmentationService:
    def __init__(self,
model_path='balanced_segmentation_model.h5',
config_path='balanced_model_config.json'):
        self.model_path = model_path
        self.config_path = config_path
        self.model = None
        self.class_names = ['field', 'forest', 'water', 'road',
'building'] # Правильные классы с building
        self.class_colors = {
            0: [255, 255, 0], # field - желтый
            1: [0, 128, 0], # forest - зеленый
            2: [0, 0, 255], # water - синий
            3: [128, 128, 128], # road - серый
            4: [255, 0, 0], # building - красный
        }
        self.input_size = (448, 448)

        # Сбалансированные пороги (компромисс между точностью и
полнотой)
        self.class_thresholds = {
```

```

        0: 0.3, # field - низкий порог (фон)
        1: 0.5, # forest - средний порог
        2: 0.7, # water - высокий порог (уменьшаем ложные
срабатывания)
        3: 0.6, # road - высокий порог
        4: 0.3, # building - СБАЛАНСИРОВАННЫЙ порог
(компромисс между точностью и полнотой)
    }

    # Загружаем модель при инициализации
    self.load_model()

def load_model(self):
    """Загрузка обученной модели"""
    try:
        print("Загружаю модель сегментации...")
        self.model =
tf.keras.models.load_model(self.model_path)
        print("Модель успешно загружена!")
        print(f" Входной размер: {self.input_size}")
        print(f" Классы: {self.class_names}")
        return True
    except Exception as e:
        print(f"Ошибка загрузки модели: {e}")
        return False

def preprocess_image(self, image):
    """Подготовка изображения для модели"""
    # Если изображение в формате PIL, конвертируем в numpy
    if isinstance(image, Image.Image):
        # Убеждаемся, что изображение в RGB формате
        image = image.convert('RGB')
        image = np.array(image)

    # Убеждаемся, что у нас есть 3 канала
    if len(image.shape) == 2:

```

```

        image = np.stack([image, image, image], axis=-1)
    elif image.shape[-1] == 4: # RGBA
        image = image[:, :, :3] # Берем только RGB

    # Изменяем размер до нужного размера
    image_resized = cv2.resize(image, self.input_size)

    # Нормализация (0-1)
    image_normalized = image_resized.astype(np.float32) /
255.0

    # Добавляем batch dimension
    image_batch = np.expand_dims(image_normalized, axis=0)

    return image_batch

def predict(self, image):
    """Предсказание с настраиваемыми порогами"""
    if self.model is None:
        raise Exception("Модель не загружена!")

    # Подготавливаем изображение
    processed_image = self.preprocess_image(image)

    # Получаем предсказания
    predictions = self.model.predict(processed_image,
verbose=0)
    probabilities = predictions[0]

    # Создаем маску с учетом порогов
    mask = np.zeros(probabilities.shape[:2], dtype=np.uint8)

    # Для каждого пикселя определяем класс с учетом порогов
    for y in range(probabilities.shape[0]):
        for x in range(probabilities.shape[1]):
            pixel_probs = probabilities[y, x]

```

```

        # Проверяем, какие классы превышают свои пороги
        valid_classes = []
        for class_id, threshold in
self.class_thresholds.items():
            if pixel_probs[class_id] >= threshold:
                valid_classes.append((class_id,
pixel_probs[class_id]))

        if valid_classes:
            # Выбираем класс с максимальной вероятностью
среди валидных

            best_class = max(valid_classes, key=lambda
x: x[1])[0]

            mask[y, x] = best_class
        else:
            # Если ни один класс не превышает порог,
выбираем field (фон)

            mask[y, x] = 0

        # Применяем цветовую фильтрацию для зданий
        mask = self.apply_color_filtering(mask, image,
probabilities)

    return mask, probabilities

    def apply_color_filtering(self, mask, original_image,
probabilities):
        """Применение цветовой фильтрации для улучшения точности
зданий"""

        if isinstance(original_image, Image.Image):
            original_image = np.array(original_image)

            image_resized = cv2.resize(original_image,
self.input_size)

            filtered_mask = mask.copy()

```

```

# Фильтрация зданий по цвету
building_mask = mask == 4
if np.sum(building_mask) > 0:
    for y in range(mask.shape[0]):
        for x in range(mask.shape[1]):
            if mask[y, x] == 4: # Если это здание
                pixel_color = image_resized[y, x]
                if len(pixel_color.shape) == 0: #
Скалярное значение
                    r = g = b = pixel_color
                else:
                    r, g, b = pixel_color[0],
pixel_color[1], pixel_color[2]
                building_prob = probabilities[y, x, 4]

            # Более мягкая фильтрация светлых
пикселей
            if r > 220 and g > 220 and b > 220:
                # Убираем только если вероятность
очень низкая
                if building_prob < 0.4:
                    filtered_mask[y, x] = 0 #
Переклассифицируем как поле

            # Более мягкая фильтрация зеленых
пикселей
            elif g > r + 20 and g > b + 20:
                # Убираем только если вероятность
очень низкая
                if building_prob < 0.5:
                    filtered_mask[y, x] = 1 #
Переклассифицируем как лес

    return filtered_mask

```

```

def create_colored_mask(self, mask):
    """Создание цветной маски сегментации"""
    h, w = mask.shape
    colored_mask = np.zeros((h, w, 3), dtype=np.uint8)

    for class_id, color in self.class_colors.items():
        colored_mask[mask == class_id] = color

    return colored_mask

def overlay_on_original(self, original_image, colored_mask,
alpha=0.6):
    """Наложение маски на оригинальное изображение"""
    if isinstance(original_image, Image.Image):
        original_image = np.array(original_image)

    # Изменяем размер оригинального изображения до размера
маски
    original_resized = cv2.resize(original_image,
self.input_size)

    # Убеждаемся, что оба изображения имеют одинаковый тип
данных
    original_resized = original_resized.astype(np.uint8)
    colored_mask = colored_mask.astype(np.uint8)

    # Смешиваем изображения
    overlay = cv2.addWeighted(original_resized, 1-alpha,
colored_mask, alpha, 0)

    return overlay

def get_class_statistics(self, mask):
    """Получение статистики по классам"""
    total_pixels = mask.size
    stats = {}

```

```

for class_id, class_name in enumerate(self.class_names):
    pixel_count = np.sum(mask == class_id)
    percentage = (pixel_count / total_pixels) * 100
    stats[class_name] = {
        'pixels': int(pixel_count),
        'percentage': round(percentage, 2)
    }

return stats

def segment_image(self, image_path_or_array):
    """Полная сегментация изображения с результатами"""
    # Загружаем изображение
    if isinstance(image_path_or_array, str):
        image = Image.open(image_path_or_array).convert('RGB')
    else:
        image = image_path_or_array

    # Получаем предсказания
    mask, probabilities = self.predict(image)

    # Создаем цветную маску
    colored_mask = self.create_colored_mask(mask)

    # Создаем overlay
    overlay = self.overlay_on_original(image, colored_mask)

    # Получаем статистику
    stats = self.get_class_statistics(mask)

    return {
        'mask': mask,
        'colored_mask': colored_mask,
        'overlay': overlay,

```

```

        'probabilities': probabilities,
        'statistics': stats,
        'class_names': self.class_names,
        'class_colors': self.class_colors
    }

def save_results(self, results, output_dir='output'):
    """Сохранение результатов сегментации"""
    os.makedirs(output_dir, exist_ok=True)

    # Сохраняем цветную маску
    mask_path = os.path.join(output_dir,
                              'segmentation_mask.png')
    cv2.imwrite(mask_path,
cv2.cvtColor(results['colored_mask'], cv2.COLOR_RGB2BGR))

    # Сохраняем overlay
    overlay_path = os.path.join(output_dir,
                              'segmentation_overlay.png')
    cv2.imwrite(overlay_path,
cv2.cvtColor(results['overlay'], cv2.COLOR_RGB2BGR))

    # Сохраняем статистику
    stats_path = os.path.join(output_dir, 'statistics.json')
    with open(stats_path, 'w', encoding='utf-8') as f:
        json.dump(results['statistics'], f,
ensure_ascii=False, indent=2)

    return {
        'mask_path': mask_path,
        'overlay_path': overlay_path,
        'stats_path': stats_path
    }

def draw_block(self, draw, x, y, w, h, fill="white",
outline="black"):

```

```

        draw.rectangle([x, y, x + w, y + h], fill=fill,
outline=outline)

    def get_font(self, size=14):
        # Попробовать подключить arial.ttf из папки проекта или
стандартный
        font_path = os.path.join(os.path.dirname(__file__),
'arial.ttf')
        try:
            return ImageFont.truetype(font_path, size)
        except Exception:
            return ImageFont.load_default()

    def draw_pvr_plan(self, field_coords=(50, 50, 500, 400),
image_size=(600, 500), meters_per_pixel=1.0, num_people=30,
save_path=None):
        img = Image.new("RGB", image_size, color='white')
        draw = ImageDraw.Draw(img)
        font = self.get_font(14)
        font_bold = self.get_font(16)

        # --- Заголовок ---
        draw.text((20, 10), f'ПВР на {num_people} человек
(масштаб: {meters_per_pixel:.2f} м/пиксель)', fill="black",
font=font_bold)

        # --- ГОСТ параметры ---
        tent_area_per_person = 3 # м² на 1 человека
        tent_size = 3 # м (палатка 3x3)
        tents_needed = int(np.ceil(num_people *
tent_area_per_person / (tent_size * tent_size)))
        tents_per_group = 4
        group_gap = 15 # м между группами
        tent_gap_side = 2.5 # м между боковыми сторонами
        tent_gap_end = 10 # м между торцами
        # Санитарная зона

```

```

toilets_needed = max(2, int(np.ceil(num_people / 13)))
showers_needed = max(2, int(np.ceil(num_people / 12)))
wc_gap = 25 # м от палаток
# Мусор
trash_gap = 25 # м от палаток
# ГСМ
fuel_gap = 50 # м от палаток
# Кухня и медпункт
kitchen_gap = 20 # м от палаток (вне жилой зоны)
med_gap = 20 # м от палаток (вне жилой зоны)
# Размеры объектов
wc_size = 2
shower_size = 2
trash_size = 2
fuel_size = 3
kitchen_size = (6, 4)
med_size = (6, 4)

def m2px(m):
    return int(m / meters_per_pixel)

# --- Поле ---
draw.rectangle(field_coords,
               outline="green",
fill="lightgreen")
    base_x,    base_y    =    field_coords[0]    +    m2px(5),
field_coords[1] + m2px(5)
    field_w = field_coords[2] - field_coords[0]
    field_h = field_coords[3] - field_coords[1]

# --- Палатки группами ---
tent_w = m2px(tent_size)
tent_h = m2px(tent_size)
gap_side = m2px(tent_gap_side)
gap_end = m2px(tent_gap_end)
group_gap_px = m2px(group_gap)
tents_drawn = 0

```

```

        group_x = base_x
        group_y = base_y
        for group in range(int(np.ceil(tents_needed /
tents_per_group))):
            for i in range(min(tents_per_group, tents_needed -
tents_drawn)):
                x = group_x + i * (tent_w + gap_side)
                y = group_y
                self.draw_block(draw, x, y, tent_w, tent_h,
fill="white", outline="black")
                tents_drawn += 1
                group_x += (tent_w + gap_side) * tents_per_group +
group_gap_px

# --- Группировка санитарной зоны (туалеты и душевые) ---
-

wc_shower_total = toilets_needed + showers_needed
wc_shower_size = 2 # м
wc_shower_w = m2px(wc_shower_size * wc_shower_total)
wc_shower_h = m2px(wc_shower_size)
found = False
for dx in range(0, field_w - wc_shower_w, 2):
    for dy in range(0, field_h - wc_shower_h, 2):
        xg = base_x + dx
        yg = base_y + dy
        if self.is_area_free_and_on_field(field_mask,
occupied, xg, yg, wc_shower_w, wc_shower_h, min_gap):
            for i in range(toilets_needed):
                xt = xg + i * m2px(wc_shower_size)
                yt = yg
                self.draw_block(draw, xt, yt,
m2px(wc_shower_size), m2px(wc_shower_size), fill="lightblue",
outline="blue")
                self.mark_area_occupied(occupied, xt,
yt, m2px(wc_shower_size), m2px(wc_shower_size), min_gap)
            for i in range(showers_needed):

```

```

            xt = xg + (toilets_needed + i) *
m2px(wc_shower_size)

            yt = yg
            self.draw_block(draw, xt, yt,
m2px(wc_shower_size), m2px(wc_shower_size), fill="lightcyan",
outline="blue")

            self.mark_area_occupied(occupied, xt,
yt, m2px(wc_shower_size), m2px(wc_shower_size), min_gap)
            found = True
            break

        if found:
            break

# --- Mycop ---
for obj, count, size, fill, outline in [
    ("trash", 1, trash_size, "lightgray", "gray")
]:
    for i in range(count):
        found = False
        for dx in range(0, field_w - m2px(size), 2):
            for dy in range(0, field_h - m2px(size), 2):
                x = base_x + dx
                y = base_y + dy
                if
self.is_area_free_and_on_field(field_mask, occupied, x, y,
m2px(size), m2px(size), min_gap):
                    self.draw_block(draw, x, y,
m2px(size), m2px(size), fill=fill, outline=outline)
                    self.mark_area_occupied(occupied, x,
y, m2px(size), m2px(size), min_gap)
                    found = True
                    break

            if found:
                break

# --- ГСМ (максимально далеко от палаток) ---

```

```

fuel_size_px = m2px(fuel_size)
max_dist = -1
best_pos = None
for dx in range(0, field_w - fuel_size_px, 2):
    for dy in range(0, field_h - fuel_size_px, 2):
        x = base_x + dx
        y = base_y + dy
        if self.is_area_free_and_on_field(field_mask,
occupied, x, y, fuel_size_px, fuel_size_px, min_gap):
            tent_mask = occupied & (mask == 0)
            tent_ys, tent_xs = np.where(tent_mask)
            if len(tent_xs) == 0:
                dist = 0
            else:
                dist = np.min(np.sqrt((tent_xs - x) ** 2
+ (tent_ys - y) ** 2))
            if dist > max_dist:
                max_dist = dist
                best_pos = (x, y)
    if best_pos:
        self.draw_block(draw, best_pos[0], best_pos[1],
fuel_size_px, fuel_size_px, fill="plum", outline="purple")
        self.mark_area_occupied(occupied, best_pos[0],
best_pos[1], fuel_size_px, fuel_size_px, min_gap)

# --- Кухня и медпункт рядом друг с другом ---
kitchen_w, kitchen_h = m2px(kitchen_size[0]),
m2px(kitchen_size[1])
med_w, med_h = m2px(med_size[0]), m2px(med_size[1])
found = False
for dx in range(0, field_w - kitchen_w - med_w -
min_gap, 2):
    for dy in range(0, field_h - max(kitchen_h, med_h),
2):
        xk = base_x + dx
        yk = base_y + dy

```

```

        xm = xk + kitchen_w + min_gap
        ym = yk
        if self.is_area_free_and_on_field(field_mask,
occupied, xk, yk, kitchen_w, kitchen_h, min_gap) and \
            self.is_area_free_and_on_field(field_mask,
occupied, xm, ym, med_w, med_h, min_gap):
            self.draw_block(draw, xk, yk, kitchen_w,
kitchen_h, fill="moccasin", outline="orange")
            self.mark_area_occupied(occupied, xk, yk,
kitchen_w, kitchen_h, min_gap)
            self.draw_block(draw, xm, ym, med_w, med_h,
fill="mistyrose", outline="red")
            self.mark_area_occupied(occupied, xm, ym,
med_w, med_h, min_gap)
            found = True
            break
    if found:
        break

# --- Масштабная линейка и легенда ---
bar_length_m = 10
bar_length_px = int(bar_length_m / meters_per_pixel)
x0 = image_size[0] - bar_length_px - 40
y0 = image_size[1] - 40
x1 = x0 + bar_length_px
y1 = y0
draw.line((x0, y0, x1, y1), fill="black", width=3)
draw.text((x0, y0 - 15), f'{bar_length_m} м',
fill="black", font=font)

# --- Легенда: размещение в самом свободном углу ---
legend_items = [
    ("Палатка", "white", "black"),
    ("Туалет", "lightblue", "blue"),
    ("Душ", "lightcyan", "blue"),
    ("Мусор", "lightgray", "gray"),

```

```

        ("ГСМ", "plum", "purple"),
        ("Кухня", "moccasin", "orange"),
        ("Медпункт", "mistyrose", "red"),
    ]
    legend_w = 110
    legend_row_h = 15
    legend_icon = 14
    legend_h = len(legend_items) * legend_row_h + 8
    margin = 7
    corners = [
        (margin, margin), # левый верхний
        (image_size[0] - legend_w - margin, margin), #
        # правый верхний
        (margin, image_size[1] - legend_h - margin), #
        # левый нижний
        (image_size[0] - legend_w - margin, image_size[1] -
        legend_h - margin), # правый нижний
    ]
    # Для каждого угла считаем минимальное расстояние до
    # занятого пикселя
    occ_ys, occ_xs = np.where(occupied)
    best_corner = corners[0]
    max_min_dist = -1
    for cx, cy in corners:
        if len(occ_xs) == 0:
            min_dist = 1e9
        else:
            dists = np.sqrt((occ_xs - cx) ** 2 + (occ_ys -
            cy) ** 2)
            min_dist = np.min(dists)
            if min_dist > max_min_dist:
                max_min_dist = min_dist
                best_corner = (cx, cy)
    legend_x, legend_y = best_corner

    # --- Полупрозрачная подложка под легенду ---

```

```

        # legend_bg_color = (240, 240, 240, 180)
        # legend_bg = Image.new("RGBA", (legend_w, legend_h),
legend_bg_color)
        # img_rgba = img.convert("RGBA")
        # img_rgba.alpha_composite(legend_bg, dest=(legend_x,
legend_y))
        # img = img_rgba.convert("RGB")
        # draw = ImageDraw.Draw(img)

        # Более компактный шрифт
        font_legend = self.get_font(11)
        for i, (label, fill, outline) in
enumerate(legend_items):
            y = legend_y + i * legend_row_h
            self.draw_block(draw, legend_x, y, legend_icon,
legend_icon, fill=fill, outline=outline)
            draw.text((legend_x + legend_icon + 6, y + 1),
label, fill=outline, font=font_legend)

        if save_path:
            img.save(save_path)
        return img

    def is_area_free_and_on_field(self, mask, occupied, x, y, w,
h, min_gap=0):
        x0, y0 = int(x - min_gap), int(y - min_gap)
        x1, y1 = int(x + w + min_gap), int(y + h + min_gap)
        h_mask, w_mask = mask.shape
        if x0 < 0 or y0 < 0 or x1 > w_mask or y1 > h_mask:
            return False
        area = mask[y0:y1, x0:x1]
        occ = occupied[y0:y1, x0:x1]
        return np.all(area == 0) and not np.any(occ)

    def mark_area_occupied(self, occupied, x, y, w, h,
min_gap=0):

```

```

x0, y0 = int(x - min_gap), int(y - min_gap)
x1, y1 = int(x + w + min_gap), int(y + h + min_gap)
occupied[y0:y1, x0:x1] = True

def draw_pvr_on_field(self, mask, image_size=(600, 500),
meters_per_pixel=1.0, num_people=30, save_path=None,
base_image=None):
    """
    Находит область 'field' на маске и рисует на ней схему
    ПВР с учетом масштаба.

    :param mask: numpy-массив маски сегментации
    :param image_size: размер итогового изображения
    :param meters_per_pixel: сколько метров в 1 пикселе
    :param num_people: количество людей
    :param save_path: путь для сохранения (опционально)
    :param base_image: PIL.Image (оригинальная карта), если
    None — рисуем на белом фоне
    :return: PIL.Image
    """
    field_class_index = 0 # индекс класса 'field'
    ys, xs = np.where(mask == field_class_index)
    if len(xs) == 0 or len(ys) == 0:
        # Нет поля — рисуем по умолчанию
        field_box = (50, 50, image_size[0] - 100,
image_size[1] - 100)
    else:
        x0, y0 = int(xs.min()), int(ys.min())
        x1, y1 = int(xs.max()), int(ys.max())
        # Немного уменьшаем рамку, чтобы не упираться в край
        pad = 10
        x0, y0 = max(x0 - pad, 0), max(y0 - pad, 0)
        x1, y1 = min(x1 + pad, image_size[0]), min(y1 + pad,
image_size[1])
        field_box = (x0, y0, x1, y1)

    # --- Используем оригинальное изображение как фон ---

```

```

        if base_image is not None:
            img = base_image
        else:
            base_image = Image.new("RGB", image_size, color='white')
            draw = ImageDraw.Draw(base_image)
            font = self.get_font(14)
            font_bold = self.get_font(16)

        def m2px(m):
            return int(m / meters_per_pixel)

        # --- Поле ---
        draw.rectangle(field_box, outline="green", width=2)
        base_x, base_y = field_box[0] + m2px(5), field_box[1] +
m2px(5)

        field_w = field_box[2] - field_box[0]
        field_h = field_box[3] - field_box[1]
        field_mask = mask
        occupied = np.zeros_like(mask, dtype=bool)

        # --- ГОСТ параметры ---
        tent_area_per_person = 3 # м² на 1 человека
        tent_size = 3 # м (палатка 3x3)
        tents_needed = int(np.ceil(num_people
tent_area_per_person / (tent_size * tent_size)))
        tents_per_group = 4
        group_gap = 15 # м между группами
        tent_gap_side = 2.5 # м между боковыми сторонами
        tent_gap_end = 10 # м между торцами
        toilets_needed = max(2, int(np.ceil(num_people / 13)))
        showers_needed = max(2, int(np.ceil(num_people / 12)))
        wc_gap = 25 # м от палаток
        trash_gap = 25 # м от палаток
        fuel_gap = 50 # м от палаток
        kitchen_gap = 20 # м от палаток (вне жилой зоны)

```

```

med_gap = 20          # м от палаток (вне жилой зоны)
wc_size = 2
shower_size = 2
trash_size = 2
fuel_size = 3
kitchen_size = (6, 4)
med_size = (6, 4)
min_gap_m = 5        # Минимальный отступ между объектами,
ГОСТ (можно увеличить)
min_gap = m2px(min_gap_m)

# --- Палатки группами ---
tent_w = m2px(tent_size)
tent_h = m2px(tent_size)
gap_side = m2px(tent_gap_side)
gap_end = m2px(tent_gap_end)
group_gap_px = m2px(group_gap)
tents_drawn = 0
tents_in_group = tents_per_group
num_groups = int(np.ceil(tents_needed / tents_in_group))
for group in range(num_groups):
    tents_in_this_group = min(tents_in_group,
tents_needed - tents_drawn)
    group_width = tents_in_this_group * tent_w +
(tents_in_this_group - 1) * gap_side
    group_height = tent_h
    found = False
    for dx in range(0, field_w - group_width, 2):
        for dy in range(0, field_h - group_height, 2):
            xg = base_x + dx
            yg = base_y + dy
            # Проверяем всю группу с учётом min_gap
            group_ok = True
            for i in range(tents_in_this_group):
                xt = xg + i * (tent_w + gap_side)
                yt = yg

```

```

            if not
self.is_area_free_and_on_field(field_mask, occupied, xt, yt,
tent_w, tent_h, min_gap):

                group_ok = False
                break
            if group_ok:
                # Размещаем всю группу
                for i in range(tents_in_this_group):
                    xt = xg + i * (tent_w + gap_side)
                    yt = yg
                    self.draw_block(draw, xt, yt,
tent_w, tent_h, fill="white", outline="black")
                    self.mark_area_occupied(occupied,
xt, yt, tent_w, tent_h, min_gap)
                    tents_drawn += 1
                found = True
                break
            if found:
                break

# --- Группировка санитарной зоны (туалеты и душевые) --
-

wc_shower_total = toilets_needed + showers_needed
wc_shower_size = 2 # м
wc_shower_w = m2px(wc_shower_size * wc_shower_total)
wc_shower_h = m2px(wc_shower_size)
found = False
for dx in range(0, field_w - wc_shower_w, 2):
    for dy in range(0, field_h - wc_shower_h, 2):
        xg = base_x + dx
        yg = base_y + dy
        if self.is_area_free_and_on_field(field_mask,
occupied, xg, yg, wc_shower_w, wc_shower_h, min_gap):
            for i in range(toilets_needed):
                xt = xg + i * m2px(wc_shower_size)
                yt = yg

```

```

                self.draw_block(draw,          xt,          yt,
m2px(wc_shower_size),  m2px(wc_shower_size),  fill="lightblue",
outline="blue")

                self.mark_area_occupied(occupied,          xt,
yt, m2px(wc_shower_size), m2px(wc_shower_size), min_gap)
                for i in range(showers_needed):
                    xt  =  xg  +  (toilets_needed  +  i)  *
m2px(wc_shower_size)

                    yt = yg
                    self.draw_block(draw,          xt,          yt,
m2px(wc_shower_size),  m2px(wc_shower_size),  fill="lightcyan",
outline="blue")

                    self.mark_area_occupied(occupied,          xt,
yt, m2px(wc_shower_size), m2px(wc_shower_size), min_gap)
                    found = True
                    break

                if found:
                    break

# --- Mycop ---
for obj, count, size, fill, outline in [
    ("trash", 1, trash_size, "lightgray", "gray")
]:
    for i in range(count):
        found = False
        for dx in range(0, field_w - m2px(size), 2):
            for dy in range(0, field_h - m2px(size), 2):
                x = base_x + dx
                y = base_y + dy
                if
self.is_area_free_and_on_field(field_mask,          occupied,          x,          y,
m2px(size), m2px(size), min_gap):
                    self.draw_block(draw,          x,          y,
m2px(size), m2px(size), fill=fill, outline=outline)
                    self.mark_area_occupied(occupied,  x,
y, m2px(size), m2px(size), min_gap)

```

```

        found = True
        break
    if found:
        break

# --- ГСМ (максимально далеко от палаток) ---
fuel_size_px = m2px(fuel_size)
max_dist = -1
best_pos = None
for dx in range(0, field_w - fuel_size_px, 2):
    for dy in range(0, field_h - fuel_size_px, 2):
        x = base_x + dx
        y = base_y + dy
        if self.is_area_free_and_on_field(field_mask,
occupied, x, y, fuel_size_px, fuel_size_px, min_gap):
            tent_mask = occupied & (mask == 0)
            tent_ys, tent_xs = np.where(tent_mask)
            if len(tent_xs) == 0:
                dist = 0
            else:
                dist = np.min(np.sqrt((tent_xs - x) ** 2
+ (tent_ys - y) ** 2))
            if dist > max_dist:
                max_dist = dist
                best_pos = (x, y)
    if best_pos:
        self.draw_block(draw, best_pos[0], best_pos[1],
fuel_size_px, fuel_size_px, fill="plum", outline="purple")
        self.mark_area_occupied(occupied, best_pos[0],
best_pos[1], fuel_size_px, fuel_size_px, min_gap)

# --- Кухня и медпункт рядом друг с другом ---
kitchen_w, kitchen_h = m2px(kitchen_size[0]),
m2px(kitchen_size[1])
med_w, med_h = m2px(med_size[0]), m2px(med_size[1])
found = False

```

```

        for dx in range(0, field_w - kitchen_w - med_w -
min_gap, 2):
            for dy in range(0, field_h - max(kitchen_h, med_h),
2):

                xk = base_x + dx
                yk = base_y + dy
                xm = xk + kitchen_w + min_gap
                ym = yk
                if self.is_area_free_and_on_field(field_mask,
occupied, xk, yk, kitchen_w, kitchen_h, min_gap) and \
                    self.is_area_free_and_on_field(field_mask,
occupied, xm, ym, med_w, med_h, min_gap):
                    self.draw_block(draw, xk, yk, kitchen_w,
kitchen_h, fill="moccasin", outline="orange")
                    self.mark_area_occupied(occupied, xk, yk,
kitchen_w, kitchen_h, min_gap)
                    self.draw_block(draw, xm, ym, med_w, med_h,
fill="mistyrose", outline="red")
                    self.mark_area_occupied(occupied, xm, ym,
med_w, med_h, min_gap)
                    found = True
                    break
            if found:
                break

# --- Масштабная линейка и легенда ---
bar_length_m = 10
bar_length_px = int(bar_length_m / meters_per_pixel)
x0 = image_size[0] - bar_length_px - 40
y0 = image_size[1] - 40
x1 = x0 + bar_length_px
y1 = y0
draw.line((x0, y0, x1, y1), fill="black", width=3)
draw.text((x0, y0 - 15), f'{bar_length_m} м',
fill="black", font=font)

```

```

# --- Легенда: размещение в самом свободном углу ---
legend_items = [
    ("Палатка", "white", "black"),
    ("Туалет", "lightblue", "blue"),
    ("Душ", "lightcyan", "blue"),
    ("Мусор", "lightgray", "gray"),
    ("ГСМ", "plum", "purple"),
    ("Кухня", "moccasin", "orange"),
    ("Медпункт", "mistyrose", "red"),
]
legend_w = 110
legend_row_h = 15
legend_icon = 14
legend_h = len(legend_items) * legend_row_h + 8
margin = 7
corners = [
    (margin, margin), # левый верхний
    (image_size[0] - legend_w - margin, margin), #
    # правый верхний
    (margin, image_size[1] - legend_h - margin), #
    # левый нижний
    (image_size[0] - legend_w - margin, image_size[1] -
    legend_h - margin), # правый нижний
]
# Для каждого угла считаем минимальное расстояние до
занятого пикселя
occ_ys, occ_xs = np.where(occupied)
best_corner = corners[0]
max_min_dist = -1
for cx, cy in corners:
    if len(occ_xs) == 0:
        min_dist = 1e9
    else:
        dists = np.sqrt((occ_xs - cx) ** 2 + (occ_ys -
cy) ** 2)
        min_dist = np.min(dists)

```

```

        if min_dist > max_min_dist:
            max_min_dist = min_dist
            best_corner = (cx, cy)
    legend_x, legend_y = best_corner

    # --- Полупрозрачная подложка под легенду ---
    # legend_bg_color = (240, 240, 240, 180)
    # legend_bg = Image.new("RGBA", (legend_w, legend_h),
legend_bg_color)
    # img_rgba = img.convert("RGBA")
    # img_rgba.alpha_composite(legend_bg, dest=(legend_x,
legend_y))
    # img = img_rgba.convert("RGB")
    # draw = ImageDraw.Draw(img)

    # Более компактный шрифт
    font_legend = self.get_font(11)
    for i, (label, fill, outline) in
enumerate(legend_items):
        y = legend_y + i * legend_row_h
        self.draw_block(draw, legend_x, y, legend_icon,
legend_icon, fill=fill, outline=outline)
        draw.text((legend_x + legend_icon + 6, y + 1),
label, fill=outline, font=font_legend)

    # После размещения всех палаток:
    success = tents_drawn == tents_needed
    if save_path:
        img.save(save_path)
    return img, success

# Глобальный экземпляр сервиса
segmentation_service = None

def get_segmentation_service():
    """Получение глобального экземпляра сервиса"""

```

```

global segmentation_service
if segmentation_service is None:
    segmentation_service = MapSegmentationService()
return segmentation_service

if __name__ == "__main__":
    # Тестирование сервиса
    service = MapSegmentationService()
    print("Сервис сегментации инициализирован")
    print(" Доступные классы: {service.class_names}")
    print(" Пороги классификации: {service.class_thresholds}")

```

ПРИЛОЖЕНИЕ А.4

Файл «index.html»

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Сегментация карт</title>
    <script src="https://api-
maps.yandex.ru/2.1/?apikey=283de992-5692-49cd-9a1c-
e005992579c1&lang=ru_RU" type="text/javascript"></script>
    <link rel="stylesheet" href="{ url_for('static',
filename='style.css') }}">
</head>
<body>
    <div class="container">
        <div class="sidebar">
            <h2>Управление картой</h2>
            <div class="coordinates">
                <div class="coordinate-label">Широта:</div>
                <span id="latitude">0.000000</span>
                <div class="coordinate-label">Долгота:</div>
                <span id="longitude">0.000000</span>
            </div>

            <div class="search-section">
                <input type="text" id="search-input"
class="search-input"
placeholder="Введите адрес или название
места">

                <button
onclick="searchLocation()">Найти</button>

                <div id="search-error" class="error-
message"></div>
```

```

</div>

<div class="manual-coordinates">
  <h3>Ручной ввод координат</h3>
  <div>
    <label for="lat-input">Широта:</label>
    <input type="number" id="lat-input"
step="0.000001" class="coordinate-input">
  </div>
  <div>
    <label for="lon-input">Долгота:</label>
    <input type="number" id="lon-input"
step="0.000001" class="coordinate-input">
  </div>
  <button
onclick="goToCoordinates()">Перейти</button>
</div>

  <button onclick="saveLocation()">Расположить
объекты</button>

  <button id="download-btn" class="hidden">Скачать
результаты</button>
  <div id="message"></div>

  <div class="saved-images">
    <div class="image-label hidden" id="label-
original">Исходная карта:</div>
    <img id="saved-image" class="hidden">

    <div class="image-label hidden" id="label-
mask">Маска сегментации:</div>
    <img id="segmentation-image" class="hidden">

    <div class="image-label hidden" id="label-
overlay">Наложение:</div>
    <img id="overlay-image" class="hidden">

```

```

        <div class="image-label hidden" id="label-
pvr">План ПБР:</div>
        <img id="pvr-plan-image" class="hidden">
    </div>

    <div class="segmentation-results hidden"
id="segmentation-results">
        <h3>Результаты сегментации</h3>
        <div class="statistics" id="statistics"></div>
    </div>
</div>
<div class="map-container">
    <div id="map"></div>
</div>
</div>

<div id="toast" class="toast"></div>

<script type="text/javascript">
    let myMap;
    let myPlacemark;

    // Проверяем загрузку API
    if (!window.ymaps) {
        console.error('API Яндекс.Карт не загружен');
        alert('Ошибка загрузки API Яндекс.Карт');
    } else {
        console.log('API Яндекс.Карт успешно загружен');
    }

    // Функция для форматирования даты и времени
    function getFormattedDateTime() {
        const now = new Date();
        const year = now.getFullYear();

```

```

        const month = String(now.getMonth() + 1).padStart(2,
'0');

        const day = String(now.getDate()).padStart(2, '0');
        const hours = String(now.getHours()).padStart(2,
'0');

        const minutes = String(now.getMinutes()).padStart(2,
'0');

        const seconds = String(now.getSeconds()).padStart(2,
'0');

        const date = `${year}-${month}-${day}`;
        const time = `${hours}-${minutes}-${seconds}`;
        return `${date};${time}`;
    }

    // Функция для форматирования координат
    function formatCoordinates(lat, lon) {
        return `${lat.toFixed(6)};${lon.toFixed(6)}`;
    }

    // Словарь для перевода названий классов
    const classTranslations = {
        'background': 'Фон',
        'forest': 'Лес',
        'field': 'Поле',
        'road': 'Дорога',
        'water': 'Вода',
        'swamp': 'Болото'
    };

    function displaySegmentationResults(results) {
        const resultsDiv =
document.getElementById('segmentation-results');
        const statisticsDiv =
document.getElementById('statistics');
        statisticsDiv.innerHTML = '';
    }

```

```

// Отображаем статистику по классам

Object.entries(results.statistics).forEach(([className, stats])
=> {
    const percentage = stats.percentage;
    const classIndex = results.class_names.indexOf(className);
    const color = results.class_colors[classIndex];

    const itemDiv = document.createElement('div');
    itemDiv.className = 'statistics-item';

    const colorBox = document.createElement('span');
    colorBox.className = 'color-box';
    colorBox.style.backgroundColor = `rgb(${color.join(',')})`;

    const nameSpan = document.createElement('span');
    nameSpan.textContent = classTranslations[className] || className; // Используем перевод

    const percentSpan = document.createElement('span');
    percentSpan.textContent = `${percentage}%`;

    itemDiv.appendChild(colorBox);
    itemDiv.appendChild(nameSpan);
    itemDiv.appendChild(percentSpan);

    statisticsDiv.appendChild(itemDiv);
});

resultsDiv.classList.remove('hidden');
}

```

```

// Функция поиска места
function searchLocation() {
    const searchQuery = document.getElementById('search-
input').value;
    const errorDiv = document.getElementById('search-
error');

    if (!searchQuery) {
        errorDiv.textContent = 'Введите адрес или
название места';
        errorDiv.style.display = 'block';
        return;
    }

    // Проверяем доступность API перед поиском
    if (!window.ymaps || !ymaps.geocode) {
        console.error('API геокодера недоступен');
        errorDiv.textContent = 'Сервис поиска временно
недоступен';
        errorDiv.classList.add('error-message');
        return;
    }

    errorDiv.textContent = '';
    errorDiv.style.display = 'none';

    ymaps.geocode(searchQuery)
        .then(function (res) {
            const firstGeoObject =
res.geoObjects.get(0);
            console.log('firstGeoObject:',
firstGeoObject);

            if (!firstGeoObject) {
                errorDiv.textContent = 'Место не
найдено';
                errorDiv.style.display = 'block';
            }
        })

```

```

        return;
    }

    const coords =
firstGeoObject.geometry.getCoordinates();
    const bounds =
firstGeoObject.properties.get('boundedBy');
    console.log('coords:', coords);
    console.log('bounds:', bounds);

    // Обновляем метку
    setPlacemark(coords);

    // Обновляем отображаемые координаты
    updateCoordinates(coords);

    // Центрируем и масштабируем карту
    if (bounds) {
        myMap.setBounds(bounds, {
            checkZoomRange: true
        });
    } else {
        myMap.setCenter(coords, 19);
    }
})
.catch(function (error) {
    console.error('Ошибка геокодирования:',
error);

    errorDiv.textContent = 'Ошибка при поиске
места';

    errorDiv.classList.add('error-message');
});
}

// Инициализация карты
ymaps.ready(function() {

```

```

        console.log('API готов к использованию');
        try {
            init();
            console.log('Карта успешно инициализирована');
        } catch (e) {
            console.error('Ошибка при инициализации карты:',
e);

            alert('Ошибка при инициализации карты');
        }
    });

function init() {
    // Создаем карту
    myMap = new ymaps.Map('map', {
        center: [55.76, 37.64],
        zoom: 19,
        controls: ['zoomControl']
    });

    // Отключаем все элементы на карте
    myMap.options.set({
        // Отключаем отображение всех POI
        yandexMapDisablePoiInteractivity: true,
        // Отключаем кнопку "Открыть в Яндекс.Картах"
        suppressMapOpenBlock: true,
        // Отключаем копирайты
        copyrightLogoVisible: false,
        copyrightProvidersVisible: false,
        copyrightUaVisible: false
    });

    // Получаем слой карты
    var layer = myMap.layers.get(0).get(0);

    // Отключаем все надписи на карте
    layer.options.set({

```

```

        // Отключаем отображение текста
        enableMapText: false,
        // Отключаем отображение иконок
        enableMapMarks: false,
        // Отключаем отображение меток
        enableMapMarkers: false,
        // Отключаем отображение POI
        enableMapPoi: false
    });

    // Обработчик клика по карте
    myMap.events.add('click', function (e) {
        const coords = e.get('coords');
        setPlacemark(coords);
        updateCoordinates(coords);
    });
}

function setPlacemark(coords) {
    if (myPlacemark) {
        myMap.geoObjects.remove(myPlacemark);
    }
    myPlacemark = new ymaps.Placemark(coords, {}, {
        preset: 'islands#redDotIcon'
    });
    myMap.geoObjects.add(myPlacemark);
}

function updateCoordinates(coords) {
    document.getElementById('latitude').textContent =
coords[0].toFixed(6);
    document.getElementById('longitude').textContent =
coords[1].toFixed(6);
}

function showToast(message, color='#333') {

```

```

const toast = document.getElementById('toast');
toast.textContent = message;
toast.style.backgroundColor = color;
toast.className = 'toast show';
setTimeout(() => {
    toast.className = 'toast';
}, 4000);
}

function saveLocation() {
    if (!myPlacemark) {
        alert('Сначала выберите точку на карте');
        return;
    }

    const coords =
myPlacemark.geometry.getCoordinates();
    const datetime = getFormattedDateTime();
    const coordsStr = formatCoordinates(coords[0],
coords[1]);
    const messageDiv =
document.getElementById('message');

    messageDiv.innerHTML = '<p style="color:
blue;">Обработка...</p>';

    fetch('/save_location', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        },
        body: JSON.stringify({
            lat: coords[0],
            lon: coords[1],
            datetime: datetime,

```

```

        coords: coordsStr
    })
})
.then(response => {
    console.log('HTTP status:', response.status);
    return response.json().then(data => {
        console.log('Ответ сервера:', data);
        if (!response.ok) {
            throw data;
        }
        return data;
    });
})
.then(data => {
    console.log('Полученные данные:', data);
    if (data.success) {
        messageDiv.innerHTML = `<p style="color:
green;">${data.message}</p>`;
        const savedImage =
document.getElementById('saved-image');
        savedImage.src = data.image_url + '?t=' +
new Date().getTime();
        savedImage.classList.remove('hidden');
        document.getElementById('segmentation-
image').src = data.segmentation_url;
        document.getElementById('segmentation-
image').classList.remove('hidden');
        document.getElementById('overlay-image').src
= data.overlay_url;
        document.getElementById('overlay-
image').classList.remove('hidden');
        // --- Удаляю overlay ПБР на карту ---
        if (window.pvrOverlayPlacemark) {
myMap.geoObjects.remove(window.pvrOverlayPlacemark);
        window.pvrOverlayPlacemark = null;

```

```

    }
    if (window.pvrOverlayLayer) {

myMap.layers.remove(window.pvrOverlayLayer);
        window.pvrOverlayLayer = null;
    }
    // Просто отображаем картинку ПВР-плана в
меню

        document.getElementById('pvr-plan-
image').src = data.pvr_plan_url;
        document.getElementById('pvr-plan-
image').classList.remove('hidden');
        // Отображаем статистику
        displaySegmentationResults({
            statistics: data.statistics,
            class_names: data.class_names,
            class_colors: data.class_colors
        });
        showResultLabels();
        // Формируем ссылку для скачивания только
текущих файлов

        const downloadBtn =
document.getElementById('download-btn');
        const params = [
            data.image_url,
            data.segmentation_url,
            data.overlay_url,
            data.pvr_plan_url
        ].map(f => 'file=' +
encodeURIComponent(f)).join('&');
        downloadBtn.onclick = function() {
            window.location.href =
'/download_results?' + params;
        };
        downloadBtn.classList.remove('hidden');
    } else {

```

```

        messageDiv.innerHTML = `<p style="color:
red;">${data.message}</p>`;

        const downloadBtn =
document.getElementById('download-btn');
        downloadBtn.classList.add('hidden');
    }
})
.catch(error => {
    console.error('Ошибка (catch):', error);
    messageDiv.innerHTML = `<p style="color:
red;">${error.message || 'Ошибка при размещении объектов'}</p>`;
    });
}

// Функция перехода к координатам
function goToCoordinates() {
    const lat = parseFloat(document.getElementById('lat-
input').value);
    const lon = parseFloat(document.getElementById('lon-
input').value);

    if (isNaN(lat) || isNaN(lon)) {
        alert('Пожалуйста, введите корректные
координаты');
        return;
    }

    if (lat < -90 || lat > 90) {
        alert('Широта должна быть в диапазоне от -90 до
90 градусов');
        return;
    }

    if (lon < -180 || lon > 180) {
        alert('Долгота должна быть в диапазоне от -180
до 180 градусов');
    }
}

```

```

        return;
    }

    // Переходим к указанным координатам
    myMap.setCenter([lat, lon], 19);

    // Устанавливаем метку
    setPlacemark([lat, lon]);
    updateCoordinates([lat, lon]);

    // Очищаем поля ввода
    document.getElementById('lat-input').value = '';
    document.getElementById('lon-input').value = '';
}

// Добавляем обработчик Enter для поля поиска
document.getElementById('search-
input').addEventListener('keypress', function(e) {
    if (e.key === 'Enter') {
        searchLocation();
    }
});

function showResultLabels() {
    document.getElementById('label-
original').classList.remove('hidden');
    document.getElementById('label-
mask').classList.remove('hidden');
    document.getElementById('label-
overlay').classList.remove('hidden');
    document.getElementById('label-
pvr').classList.remove('hidden');
}
</script>
</body>
</html>

```


ПРИЛОЖЕНИЕ А.5

Файл «style.css»

```
body, html {
    margin: 0;
    padding: 0;
    height: 100%;
    font-family: Arial, sans-serif;
}
.container {
    display: flex;
    height: 100%;
}
.sidebar {
    width: 400px;
    padding: 20px;
    background-color: #f5f5f5;
    border-right: 1px solid #ddd;
    overflow-y: auto;
}
.map-container {
    flex-grow: 1;
    position: relative;
}
#map {
    width: 100%;
    height: 100%;
}
.coordinates {
    margin: 10px 0;
    font-size: 14px;
}
button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 15px;
```

```

    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
    margin-top: 10px;
}
button:hover {
    background-color: #45a049;
}
.saved-images img {
    max-width: 100%;
    margin-top: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
}
.image-label {
    font-weight: bold;
    margin-top: 15px;
    color: #333;
}
.coordinate-input {
    width: 100%;
    padding: 8px;
    margin: 5px 0;
    border: 1px solid #ddd;
    border-radius: 4px;
    box-sizing: border-box;
}
/* Убираем стрелки из полей ввода чисел */
input[type="number"]::-webkit-outer-spin-button,
input[type="number"]::-webkit-inner-spin-button {
    -webkit-appearance: none;
    margin: 0;
}
input[type="number"] {
    -moz-appearance: textfield;

```

```

}
.coordinate-label {
    font-weight: bold;
    margin-top: 10px;
    color: #333;
}
.search-section {
    margin-top: 20px;
}
.search-input {
    width: 100%;
    padding: 8px;
    margin: 5px 0;
    border: 1px solid #ddd;
    border-radius: 4px;
    box-sizing: border-box;
}
.error-message {
    color: #ff0000;
    font-size: 12px;
    margin-top: 5px;
}
.segmentation-results {
    margin-top: 20px;
    padding: 10px;
    background-color: #fff;
    border-radius: 4px;
    border: 1px solid #ddd;
}
.statistics {
    margin-top: 10px;
    font-size: 14px;
}
.statistics-item {
    display: flex;
    justify-content: space-between;

```

```
    margin: 5px 0;
    padding: 3px 0;
    border-bottom: 1px solid #eee;
}
.color-box {
    width: 20px;
    height: 20px;
    display: inline-block;
    margin-right: 5px;
    vertical-align: middle;
    border: 1px solid #ddd;
}
.hidden {
    display: none !important;
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Использование искусственных нейронных сетей для решения задач
позиционирования объектов на цифровых картах

Презентация

ИСПОЛЬЗОВАНИЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ РЕШЕНИЯ ЗАДАЧ ПОЗИЦИОНИРОВАНИЯ ОБЪЕКТОВ НА ЦИФРОВЫХ КАРТАХ

Выполнил: ст. группы 23ВВИм1

Экимов Д.В.

Введение

В современном мире цифровые карты играют ключевую роль в навигации, городском планировании и мониторинге окружающей среды.

Однако задача точного позиционирования объектов остаётся сложной из-за высокой плотности информации и разнообразия объектов.

Применение искусственных нейронных сетей (ИНС) позволяет:

- ▶ автоматически извлекать признаки с изображений,
- ▶ эффективно обрабатывать большие объёмы геоданных,
- ▶ точно определять местоположение объектов на карте.

В данной работе рассматривается использование ИНС для решения задач семантической сегментации изображений карты и размещения инфраструктурных объектов.

Актуальность темы исследования

В условиях стремительного роста объёма геопространственных данных возрастает потребность в автоматизированных методах анализа и обновления цифровых карт.

Искусственные нейронные сети (ИНС) обеспечивают:

- ▶ высокую точность идентификации объектов;
- ▶ адаптацию к сложным и разнородным данным;
- ▶ автоматическую обработку спутниковых изображений.

Современные модели (YOLOv5, U-Net, Swin Transformer) демонстрируют точность до 89% (precision) при позиционировании объектов на изображениях высокого разрешения.

Использование ИНС упрощает:

- ▶ обновление инфраструктурных данных;
- ▶ отслеживание изменений на местности;
- ▶ интеграцию с ГИС и планировочными системами.

Анализ существующих решений

В современных ГИС активно применяются методы искусственного интеллекта для распознавания и классификации объектов:

Яндекс.Навигатор

- ▶ Использует компьютерное зрение для анализа дорожной инфраструктуры.
- ▶ Распознаёт разметку, дорожные знаки, полосы.

2ГИС

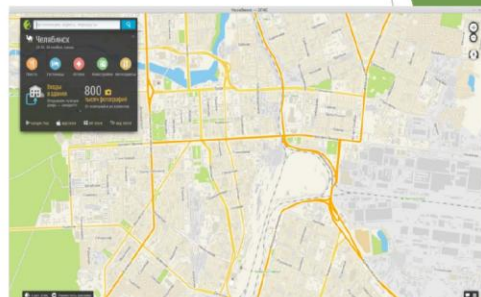
- ▶ Сегментирует карты, выделяя здания, дороги, водоёмы.
- ▶ Применяет CNN для обновления данных и поиска объектов.

Google Maps

- ▶ Использует сверточные нейросети (CNN) и AI для детальной обработки картографических изображений.
- ▶ Автоматически классифицирует элементы на спутниковых снимках.

Все решения используют CNN и другие модели ИНС для улучшения качества карт и автоматизации их поддержки.

Анализ существующих решений



Методы обнаружения объектов

Обнаружение объектов — это задача нахождения и классификации объектов на изображении с определением их координат.

Используемые архитектуры:

- ▶ YOLO (You Only Look Once) — детекция в реальном времени;
- ▶ Faster R-CNN — точная, но медленная модель;
- ▶ SSD (Single Shot Detector) — компромисс между скоростью и точностью.

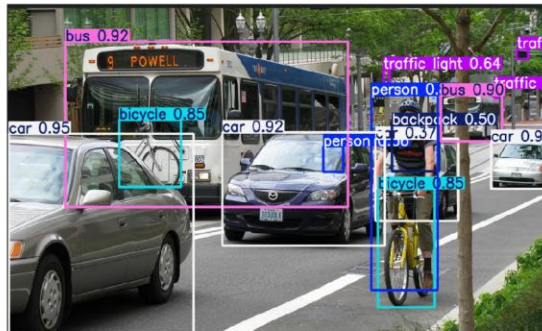
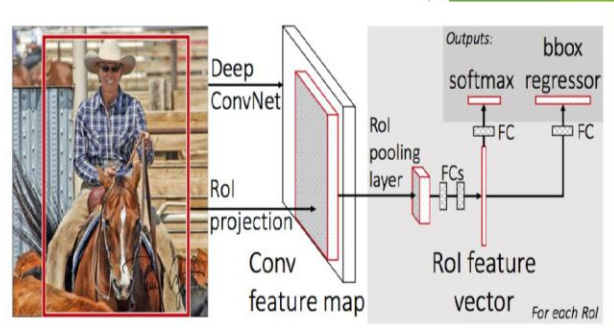
Компоненты:

- ▶ Свёрточная нейросеть (CNN) для извлечения признаков;
- ▶ Регрессионные блоки для предсказания координат;
- ▶ Softmax/activation для определения класса.

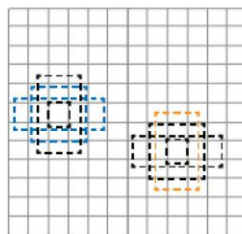
Применение:

- ▶ Навигационные приложения (распознавание знаков, дорог);
- ▶ Инфраструктурные карты (поиск зданий, мостов, вокзалов);
- ▶ Городское планирование и контроль трафика.

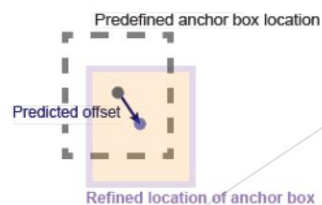
Методы обнаружения объектов



Ground truth image and bounding boxes



Anchor boxes at each predefined location in each feature map



Семантическая сегментация с использованием CNN

Семантическая сегментация — это процесс, при котором каждый пиксель изображения классифицируется по типу объекта.

Используемые подходы:

- **Сверточные нейронные сети (CNN)** — извлекают пространственные признаки;
- **U-Net, DeepLab, SegNet** — архитектуры сегментации на уровне пикселей.

Этапы:

1. Предобработка изображений (нормализация, аугментация);
2. Обучение модели на размеченных масках;
3. Применение модели к новым изображениям;
4. Построение маски (цвета = классы объектов).

Применение:

- Автоматическое обновление карт;
- Выделение полей зданий, дорог, лесов, водоёмов;
- Анализ изменений на местности.
- Размещение объектов

Семантическая сегментация с использованием RNN

RNN (рекуррентные нейронные сети) применяются для обработки последовательных данных, включая видео, временные ряды и последовательности изображений.

Возможности RNN в сегментации:

- Учитывают контекст соседних фрагментов;
- Эффективны при анализе временных или пространственно зависимых картинок;
- Используются в связке с CNN в архитектурах типа ReSeg, ConvLSTM U-Net.

Ограничения:

- Сложнее обучаются, чем CNN;
- Меньше применимы к одиночным снимкам без временной структуры;
- Требуют больше вычислительных ресурсов.

В проекте не использовались, так как:

- данные — одиночные спутниковые изображения;
- CNN (U-Net) показали лучшую точность и простоту реализации.

Выбор технологий и инструментов разработки

Разработка программного комплекса велась с использованием современных и открытых технологий.

Бэкэнд:

- ▶ Python — основной язык разработки;
- ▶ Flask — лёгкий веб-фреймворк для API и веб-интерфейса;
- ▶ TensorFlow / Keras — для обучения и запуска нейросетевой модели (U-Net);
- ▶ OpenCV — для обработки изображений.

Фронтенд:

- ▶ HTML + CSS + JavaScript — формирование пользовательского интерфейса;
- ▶ Yandex.Maps API — отображение карт и взаимодействие с координатами;
- ▶ OpenStreetMap API — получение фоновых карт.

Дополнительные инструменты:

- ▶ LabelMe — разметка изображений;

Архитектура нейросети U-Net

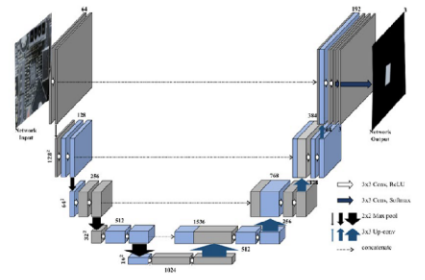
U-Net — архитектура свёрточной нейросети, предназначенная для семантической сегментации изображений. Разработана для медицинской визуализации, но эффективно применяется в геоинформационных задачах.

Структура модели:

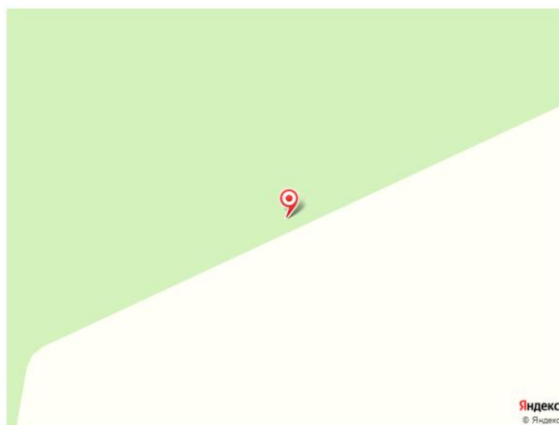
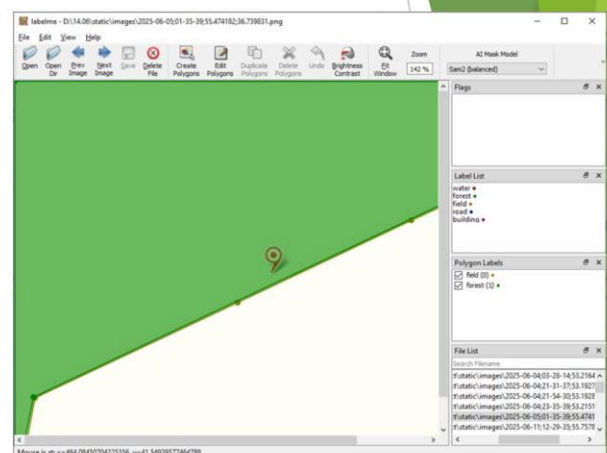
- **Encoder (сжимающая часть):** извлекает признаки, постепенно уменьшая размер изображения;
- **Decoder (восстанавливающая часть):** восстанавливает пространственную структуру и формирует маску;
- **Скип-соединения (skip connections):** связывают соответствующие уровни энкодера и декодера, передавая детальные признаки.

Особенности реализации в проекте:

- Модель построена «с нуля» на Keras (TensorFlow);
- Без предобученного энкодера (облегчает вес, ~487 тыс. параметров);
- Входное изображение: $448 \times 448 \times 3$;
- Выход: карта классов $448 \times 448 \times C$, где C — количество классов.
- U-Net обеспечивает точную сегментацию с малым числом параметров



Разметка обучающих данных и процесс обучения



Маски сегментации



Формирование схемы размещения объектов ПВР

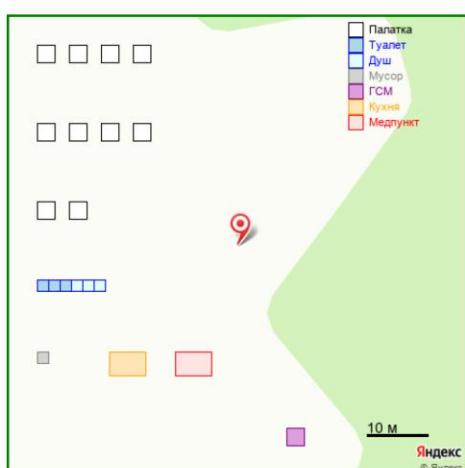
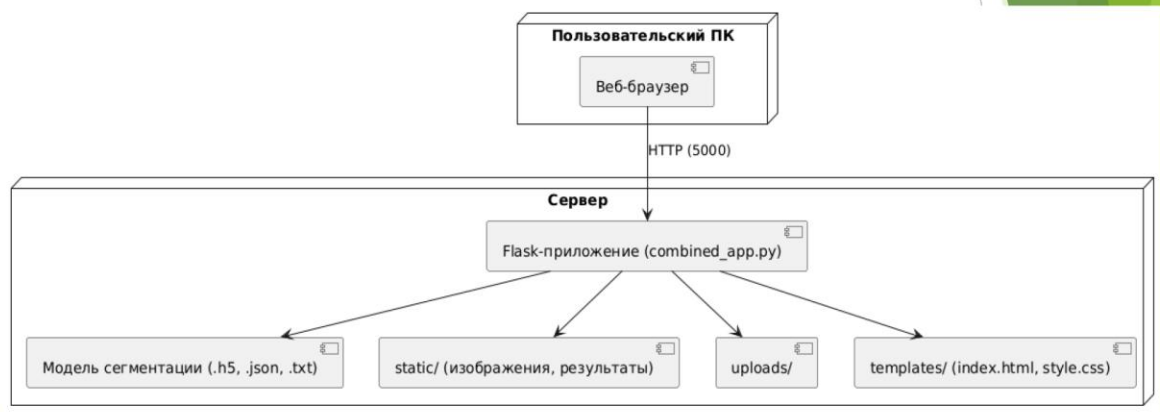


Диаграмма вариантов использования



Диаграмма развертывания



Структура приложения

Сервер: Flask-приложение, Python, работает с моделью, API.

Клиент:

Веб-интерфейс (HTML + JS), открывается в браузере.

Модули:

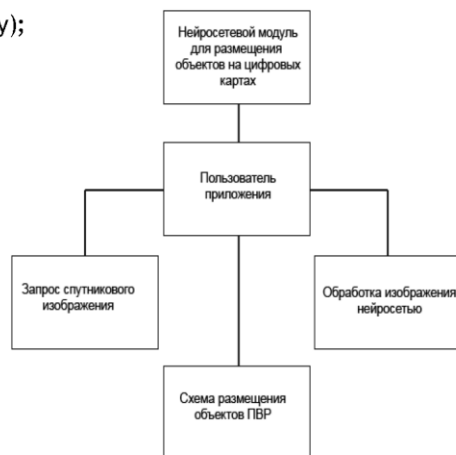
загрузка изображения (yandex_maps.py);

нейросетевая сегментация (segmentation_model);

визуализация (combined_app.py);

веб-интерфейс (test_app.py).

Схема функции модули



Интерфейс веб-приложения

Пользовательский интерфейс разработан в виде веб-приложения, доступного через браузер.

Основные функции:

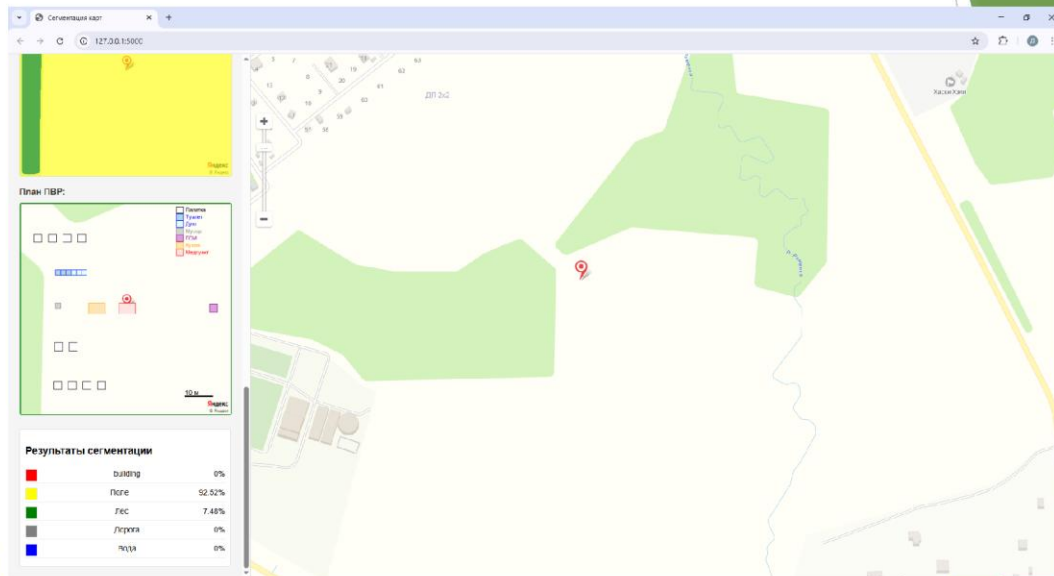
- Ввод координат или выбор области на карте;
- Загрузка спутникового изображения;
- Запуск нейросетевой сегментации;
- Визуализация результата в виде цветной маски;
- Генерация схемы размещения ПБР согласно нормативам.

Интерфейс реализован с использованием:

- HTML, CSS, JavaScript;
- Интерактивная карта на основе Leaflet/OpenStreetMap;
- Взаимодействие с backend через REST API (Flask).

Все действия выполняются через понятный и минималистичный интерфейс без необходимости устанавливать ПО.

Интерфейс веб-приложения



Результаты тестирования модели

Модель: U-Net

Обучение на 200 спутниковых изображениях (размер 448×448)
Использована взвешенная **категориальная кросс-энтропия**

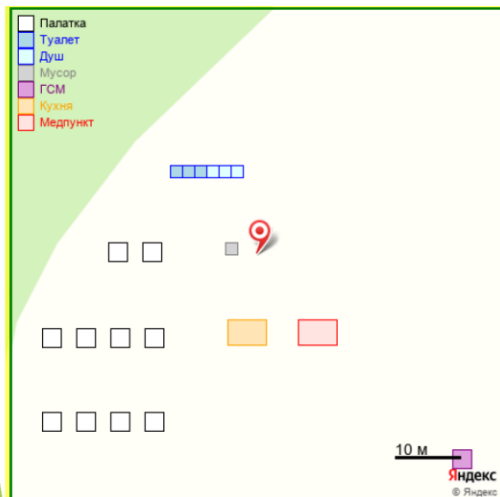
Основные метрики:

- **Pixel Accuracy:** 85%
- **IoU (Intersection over Union):** 0.78 (в среднем по 5 классам)
- **F1-score:** > 0.8 для классов "здание" и "поле"

Примеры вывода:

- Исходное изображение
- Предсказанная маска
- Наложение маски на оригинал

Результаты тестирования модели



Метрика	Значение
Точность по пикселям	85%
Среднее IoU	0.78
F1-score	> 0.80

Заключение

В ходе работы разработан программный комплекс для автоматической семантической сегментации спутниковых изображений и генерации схем размещения объектов.

Обоснован выбор архитектуры **U-Net**, показавшей высокую точность при малом числе параметров (~487 тыс.), что делает её подходящей для веб-сервисов.

Модель успешно распознаёт ключевые объекты инфраструктуры: здания, дороги, поля, леса, водоёмы.

Разработан веб-интерфейс на Python (Flask), реализующий полный цикл: от загрузки изображения по координатам до вывода итоговой схемы.

Алгоритм размещения объектов учитывает нормативы ГОСТ и может применяться в системах ЧС и городском планировании.

Испытания подтвердили корректность работы и соответствие функционала требованиям.

Разработка имеет прикладной потенциал и может быть интегрирована в реальные геоинформационные системы.

Спасибо за внимание!