

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«Северный (Арктический) федеральный университет имени М.В. Ломоносова»

Высшая школа информационных технологий и автоматизированных систем
(наименование высшей школы / филиала / института / колледжа)

КУРСОВОЙ ПРОЕКТ

По модулю
решения

Интеграция моделей глубого обучения в программные

На тему Проектирование и разработка содержательной части и программной
оболочки системы машинного обучения для сегментации морских сцен

| Ф.И.О. обучающихся | Наименование направления подготовки / специальности | Курс | Группа | Ф.И.О. руководителя (-ей) должность / уч. степень / звание |
|------------------------------------|--|------|--------|---|
| Абдурахманова Карина Отабековна | 09.04.02 Информационные системы и технологии | 2 | 151466 | Березовский Владимир Валерьевич, к.ф.-м.н., профессор кафедры ИСиИБ |
| Архаров Никита Михайлович | 09.04.02 Информационные системы и технологии | 2 | 151466 | Березовский Владимир Валерьевич, к.ф.-м.н., профессор кафедры ИСиИБ |

Признать, что проект выполнен и
защищен с отметкой

(отметка прописью)

(дата)

Руководитель

(подпись руководителя)

В.В. Березовский

(инициалы, фамилия)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«Северный (Арктический) федеральный университет имени М.В. Ломоносова»

Высшая школа информационных технологий и автоматизированных систем
(наименование высшей школы / филиала / института / колледжа)

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

по Интеграция моделей глубокого обучения в программные решения (модуль)
(наименование дисциплины)

студенту ВШИТИАС высшей школы 2 курса 151466 группы
Абдурахмановой Карине Отабековне, Архарову Никите Михайловичу

(фамилия, имя, отчество студента)

09.04.02 «Информационные системы и технологии»

(код и наименование направления подготовки/специальности)

ТЕМА: Проектирование и разработка содержательной части и программной оболочки системы машинного обучения для сегментации морских сцен.

ИСХОДНЫЕ ДАННЫЕ: Сделать реализацию инфраструктуры полного MLOps цикла (получение и хранение данных, разработка моделей, тестирование, Интеграция, выпуск и управление инфраструктурой)

1. Выделить эталонный датасет для обучения.
2. Обучить кардинально другую модель и сделать сравнение.
3. Сделать тюнинг гиперпараметров моделей.
4. Код должен быть размещен на github
5. Найти место для размещения эталонных данных

Срок выполнения с 15.09.2025 г. по 13.01.2026 г.

Руководитель работы

профессор

В.В. Березовский

ЛИСТ ДЛЯ ЗАМЕЧАНИЙ

ОГЛАВЛЕНИЕ

| | |
|--|----|
| Лист для замечаний..... | 3 |
| Оглавление..... | 4 |
| Введение..... | 5 |
| 1 Анализ готовых решений | 7 |
| 2 Использование готовых инструментов для работы с набором данных | 9 |
| 3. Обработка данных..... | 11 |
| 3.1 Получение и распаковка данных..... | 11 |
| 3.2 Разработка dataloader'ов..... | 11 |
| 3.2.1 Семантический датасет | 12 |
| 3.2.2 Паноптический датасет | 13 |
| 3.3 Преимущества текущего подхода | 15 |
| 4.1 DeepLabV3 | 15 |
| 4.2 DeepLabV3 с backbone ResNet-50..... | 16 |
| 4.3 DeepLabV3 с backbone MobileNetV3..... | 16 |
| 5 Обучение модели | 18 |
| 5.1 Подбор гиперпараметров | 18 |
| 5.2 Описание параметров обучения и результатов | 20 |
| 6 Контейнеризация модели | 22 |
| 6.1 Общая структура контейнера | 22 |
| 6.2 Сервис инференса | 22 |
| 6.3 Сборка и запуск контейнера | 23 |
| 6.4 Результаты контейнеризации..... | 23 |
| Заключение | 25 |
| Список используемых источников..... | 27 |
| Приложение А (обязательное) Листинг кода обучения моделей..... | 28 |
| Приложение Б (обязательное) Листинг кода использования модели..... | 32 |
| Приложение В Сведения о самостоятельности выполнения работы | 34 |

ВВЕДЕНИЕ

Современные геоинформационные системы (ГИС) играют ключевую роль в анализе и визуализации пространственных данных, обеспечивая эффективное принятие решений в различных областях: от городского планирования до экологического мониторинга и управления ресурсами. Одним из перспективных направлений развития ГИС является интеграция методов компьютерного зрения и глубокого обучения, что позволяет автоматически выделять и классифицировать объекты на изображениях, создавать высокоточные тематические карты и реализовывать функции анализа в реальном времени.

Особую актуальность эта задача приобретает в области морской и прибрежной среды, где необходимо оперативно обрабатывать большие объёмы визуальной информации, включая спутниковые и беспилотные данные. Существующие подходы к семантической и паноптической сегментации позволяют выделять объекты различных классов на изображениях с высокой точностью, что делает их незаменимыми инструментами для построения интеллектуальных ГИС.

В настоящей работе рассматривается процесс проектирования и разработки содержательной части и программной оболочки геоинформационной системы, способной обрабатывать визуальные данные морской среды с использованием методов глубокого обучения. Основное внимание уделено следующим аспектам:

- подготовка и обработка исходных данных, включая изображения и соответствующие аннотации для задачи сегментации;
- разработка специализированных загрузчиков данных (dataloader) для семантической и паноптической сегментации;
- выбор архитектур нейронных сетей, оптимальных для обработки изображений заданного типа, с учётом баланса между точностью и вычислительными ресурсами;

- проведение экспериментов по обучению моделей, настройке гиперпараметров и оценке качества сегментации;
- реализация программной оболочки, обеспечивающей удобное использование разработанных моделей в рамках ГИС.

Цель работы заключается в создании гибкой и эффективной системы обработки визуальных данных, которая может быть интегрирована в ГИС для решения задач автоматической сегментации и классификации объектов на изображениях морской среды. Полученные результаты демонстрируют применимость современных методов глубокого обучения в практических геоинформационных задачах и позволяют сделать выводы о выборе оптимальной архитектуры для конкретной предметной области.

1 АНАЛИЗ ГОТОВЫХ РЕШЕНИЙ

Для реализации инфраструктуры MLOps был выбран датасет LaRS (Lakes, Rivers and Seas) — первый масштабный и разнообразный набор данных для паноптического обнаружения препятствий на воде. Основная цель LaRS — предоставить исследователям в области морской робототехники и автономного судовождения эталонный материал для обучения и оценки моделей, способных распознавать как статические, так и динамические препятствия в сложных водных сценах.

Особенности датасета LaRS:

- наибольшая разнообразность среди существующих датасетов для морской среды: сцены с озер, рек и морей, снятые вручную, из публичных видео и существующих коллекций.
- более 4000 ключевых кадров с разметкой по пикселям и 9 предшествующих кадров для использования временной информации, что в сумме составляет около 40 000 кадров;
- каждый ключевой кадр размечен по 11 классам объектов и фонов и имеет 19 глобальных атрибутов сцены.

Предоставлены три типа разметки: паноптическая, семантическая и атрибуты сцены.

Разметка датасета:

- паноптическая разметка в формате COCO: включает файл `panoptic_annotations.json` и маски в директории `panoptic_masks`. Категории включают статические препятствия, воду, небо, а также динамические объекты — лодки, пловцов, животных и др;
- семантическая разметка: PNG-маски с пиксельными метками трех классов — препятствия, вода, небо. Метка 255 используется для игнорируемых областей.

На рисунке 1 представлен пример разметки.

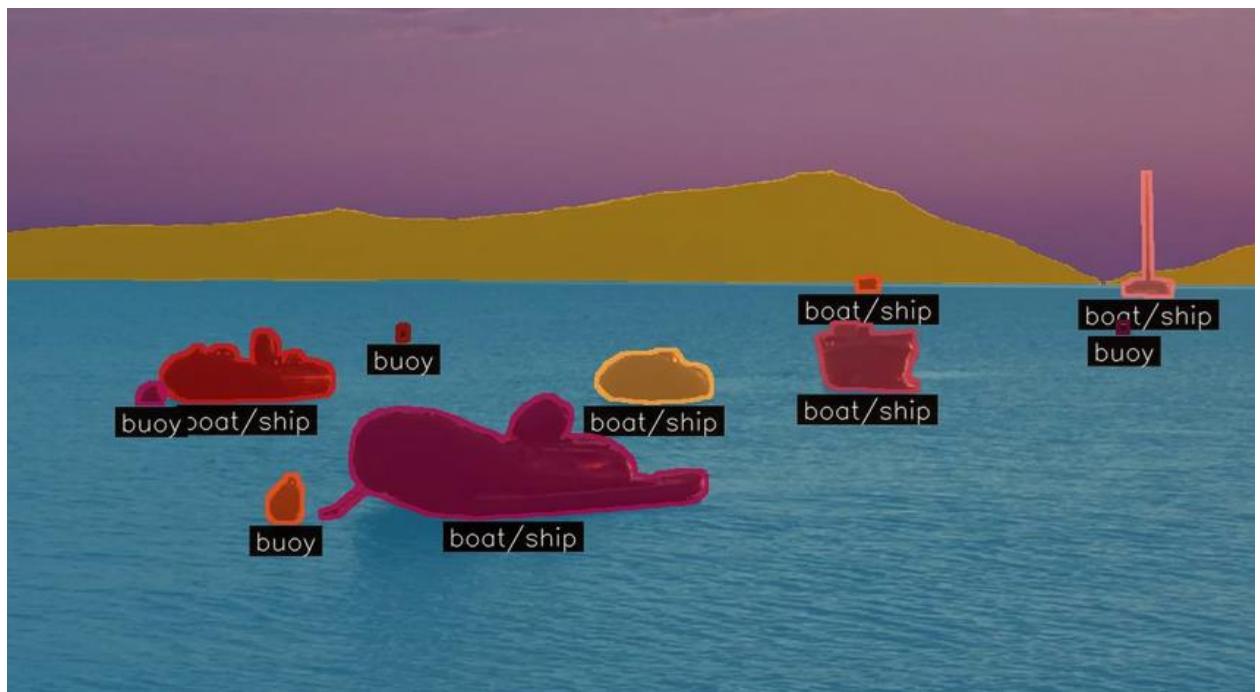


Рисунок 1 – Пример разметки LaRS

Атрибуты сцены хранятся в файле `image_annotations.json` и содержат информацию о глобальных характеристиках каждой сцены.

Датасет LaRS предоставляет обширный и разнообразный материал для обучения моделей компьютерного зрения, включая задачи сегментации и обнаружения препятствий на воде, что делает его идеальным эталонным набором для реализации MLOps-пайплайна в проекте автономных судов.

2 ИСПОЛЬЗОВАНИЕ ГОТОВЫХ ИНСТРУМЕНТОВ ДЛЯ РАБОТЫ С НАБОРОМ ДАННЫХ

Для работы с датасетом LaRS разработан репозиторий на GitHub — LaRS Segmentation Starter Kit (MMSegmentation) [4]. Этот репозиторий представляет собой форк MMSegmentation и предоставляет базовый набор инструментов для проведения экспериментов по сегментации на LaRS.

Основные возможности репозитория:

- Dataloader для LaRS — готовые функции загрузки и предобработки данных;
- конфигурации для различных моделей сегментации — включая FCN, UNet, DeepLabv3, BiSeNet, STDC, SegFormer, Segmenter и KNet;
- утилиты для обучения и инференса — скрипты для запуска тренировок и получения предсказаний на тестовых данных LaRS.

Репозиторий предоставляет готовую инфраструктуру и конфигурации для быстрого старта работы с датасетом, что упрощает интеграцию LaRS в MLOps-пайплайн и позволяет сосредоточиться на разработке моделей и сравнении их производительности.

Особенности запуска и ограничения:

- для корректной работы требуется Python 3.10, в то время как Google Colab использует версию 3.12, что делает запуск невозможным без изменения окружения;
- при локальной сборке dev container с использованием Python 3.10 и всех необходимых зависимостей (mmcv-full, mmengine, mmdet, требования из requirements.txt) возникали определённые ошибки при запуске скрипта обучения (tools/train.py). Это связано с несовместимостью некоторых библиотек и сложностью их настройки в современных версиях окружения.

Исправление этих ошибок оказалось слишком трудоёмким и проблематичным, поэтому было принято решение обрабатывать набор данных самостоятельно, что обеспечивает более гибкую работу с данными и упрощает дальнейшую интеграцию в MLOps-пайплайн. Таким образом, несмотря на

существование готовых инструментов, для целей данного проекта был выбран подход самостоятельной обработки набора данных, что позволяет полностью контролировать процесс подготовки данных и настройки моделей.

3. ОБРАБОТКА ДАННЫХ

Для реализации MLOps-пайплайна и обучения моделей сегментации на датасете LaRS первым этапом была проведена полная подготовка данных: скачивание, распаковка и создание структуры загрузки для дальнейшего использования в моделях.

3.1 Получение и распаковка данных

Набор данных LaRS представлен двумя архивами: изображения и аннотации. Для их загрузки использовались стандартные инструменты Python (листинг 1)

Листинг 1 – Получение набора данных

```
# Загрузка изображений
!wget -O lars_v1.0.0_images.zip
"https://box.vicos.si/lars/lars_v1.0.0_images.zip"
with zipfile.ZipFile("lars_v1.0.0_images.zip", 'r') as zip_ref:
    zip_ref.extractall()
# Загрузка аннотаций
!wget -O lars_v1.0.0_annotations.zip
"https://box.vicos.si/lars/lars_v1.0.0_annotations.zip"
with zipfile.ZipFile("lars_v1.0.0_annotations.zip", 'r') as zip_ref:
    zip_ref.extractall()
```

После распаковки в рабочей директории появились все изображения и соответствующие разметки, структурированные по разделам набора данных (train, val, test) и типам аннотаций (семантические и паноптические).

3.2 Разработка dataloader'ов

Для эффективной работы с датасетом были разработаны два собственного dataloader'a: один для семантической сегментации, другой — для паноптической. Это позволило гибко управлять загрузкой изображений и масок, а также адаптировать данные под обучение любых моделей.

3.2.1 Семантический датасет

Семантическая разметка включает три класса: препятствия, вода и небо. Класс LaRSSemanticDataset реализует стандартный интерфейс PyTorch Dataset (листинг 2).

Листинг 2 – Класс сематического набора данных

```
class LaRSSemanticDataset(Dataset):
    def __init__(self, root_dir, split="train", img_size=(256, 256)):
        self.split_dir = os.path.join(root_dir, split)
        self.images_dir = os.path.join(self.split_dir, "images")
        self.masks_dir = os.path.join(self.split_dir,
                                     "semantic_masks") if split != "test" else None
        list_path = os.path.join(self.split_dir, "image_list.txt")
        with open(list_path, "r") as f:
            self.files = [x.strip() for x in f.readlines()]
        self.img_size = img_size
    def __len__(self):
        return len(self.files)
    def _find_file(self, folder, name):
        for ext in [".png", ".jpg", ".jpeg"]:
            path = os.path.join(folder, name + ext)
            if os.path.exists(path):
                return path
        raise FileNotFoundError(f"No file found for {name} in {folder}")
    def __getitem__(self, idx):
        name = self.files[idx]
        img_path = self._find_file(self.images_dir, name)
        image = Image.open(img_path).convert("RGB")
        image = TF.resize(image, self.img_size)
        image = TF.to_tensor(image)
        if self.masks_dir:
            mask_path = self._find_file(self.masks_dir, name)
            mask = Image.open(mask_path)
            mask = TF.resize(mask, self.img_size,
                             interpolation=Image.NEAREST)
            mask = torch.from_numpy(np.array(mask)).long()
        else:
            mask = None
        return (image, mask) if mask is not None else image
```

Особенности:

- поддержка основных форматов изображений (.png, .jpg, .jpeg);
- масштабирование изображений и масок до одинакового размера для удобства обучения моделей.
- поддержка загрузки данных без масок (например, для тестового набора).

Сематический набор данных включает в себя 3 класса: вода, небо, объект. Пример предсказания представлен на рисунке 2.

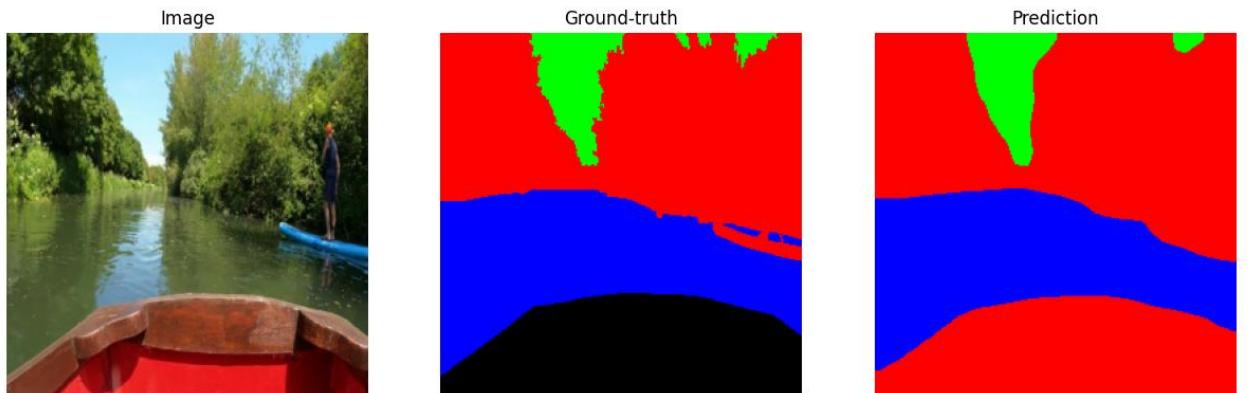


Рисунок 2 – Пример предсказания сематической разметки

3.2.2 Паноптический датасет

Для паноптической сегментации предусмотрен класс LaRSPanopticDataset, который обрабатывает 11 классов объектов и фонов. Этот dataloader учитывает соответствие между ID категорий и индексами классов для моделей (листиング 3).

Листинг 3 – Класс паноптического набора данных

```
class LaRSPanopticDataset(Dataset):
    def __init__(self, root_dir, split="train", img_size=(256, 256)):
        self.split_dir = os.path.join(root_dir, split)
        self.images_dir = os.path.join(self.split_dir, "images")
        self.masks_dir = os.path.join(self.split_dir,
                                     "panoptic_masks")
        self.img_size = img_size
        list_path = os.path.join(self.split_dir, "image_list.txt")
        with open(list_path, "r") as f:
            self.files = [x.strip() for x in f.readlines()]

        ann_path = os.path.join(self.split_dir,
                               "panoptic_annotations.json")
        with open(ann_path, "r") as f:
            self.ann = json.load(f)
            self.file_to_mask = {x["file_name"]: x["id"] for x in
                                self.ann["images"]}
            self.cat_ids = sorted([c["id"] for c in
                                  self.ann["categories"]])
            self.id2idx = {cid: i for i, cid in enumerate(self.cat_ids)}
    def __len__(self):
        return len(self.files)
    def _find_file(self, folder, name):
        for ext in [".png", ".jpg", ".jpeg"]:
            path = os.path.join(folder, name + ext)
```

Продолжение листинга 3

```
        if os.path.exists(path):
            return path
        raise FileNotFoundError(f"No file found for {name} in
{folder}")
    def __getitem__(self, idx):
        name = self.files[idx]
        img_path = self._find_file(self.images_dir, name)
        image = Image.open(img_path).convert("RGB")
        image = TF.resize(image, self.img_size)
        image = TF.to_tensor(image)
        mask_path = self._find_file(self.masks_dir, name)
        mask = Image.open(mask_path)
        mask = TF.resize(mask, self.img_size,
interpolation=Image.NEAREST)
        mask = np.array(mask)
        if mask.ndim == 3:
            mask = mask[...,0]
        mask_idx = np.zeros_like(mask, dtype=np.int64)
        for k,v in self.id2idx.items():
            mask_idx[mask==k] = v
        mask_idx = torch.from_numpy(mask_idx)
        return image, mask_idx
```

Особенности:

- автоматическое преобразование исходных ID категорий в непрерывные индексы классов для корректной работы функции потерь;
- поддержка масок с несколькими каналами, при необходимости используется только первый канал;
- масштабирование изображений и масок для унификации входных данных моделей.

Пример предсказания представлен на рисунке 3.

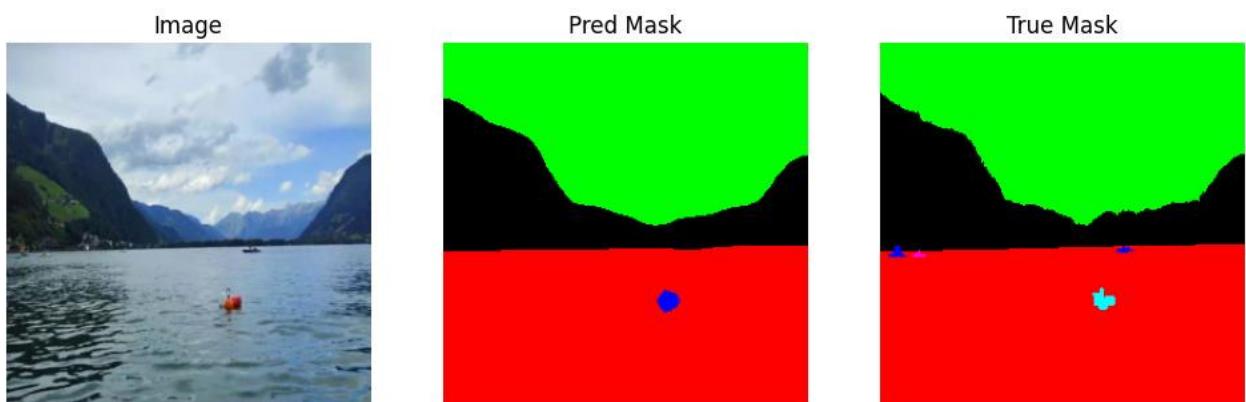


Рисунок 3 – Пример предсказания паноптической разметки

3.3 Преимущества текущего подхода

Создание собственных dataloader'ов позволило:

- гибко работать с разными типами разметки (семантическая и паноптическая);
- контролировать процесс загрузки, преобразования и масштабирования данных;
- подготовить данные в формате, готовом к обучению любых моделей сегментации, без привязки к сторонним конфигурациям;
- обеспечить возможность последующей интеграции в MLOps-пайплайн, где датасет используется напрямую в PyTorch DataLoader.

Таким образом, этап обработки данных обеспечивает полный контроль над набором данных и является фундаментом для построения эффективного пайплайна обучения и тестирования моделей сегментации на LaRS.

4 Выбор модели

Для решения задачи семантической сегментации водной поверхности и препятствий были выбраны две модели: DeepLabV3 с backbone ResNet-50 [1] и DeepLabV3 с backbone MobileNetV3 [2]. Выбор данных архитектур обусловлен их распространённостью, стабильностью и различием в вычислительной сложности, что позволяет провести наглядное сравнение моделей.

4.1 DeepLabV3

Архитектура DeepLabV3 [5] является одной из наиболее известных и широко используемых моделей для семантической сегментации изображений. Ключевой особенностью данной архитектуры является использование atrous (dilated) convolutions, которые позволяют увеличивать receptive поле без потери пространственного разрешения. Это особенно важно для задач сегментации, где необходимо точно выделять границы объектов.

Дополнительно в DeepLabV3 применяется модуль ASPP (Atrous Spatial Pyramid Pooling), который позволяет учитывать контекст на разных масштабах. Для сцен водной среды это критично, так как объекты могут значительно различаться по размеру: от небольших буйков и пловцов до крупных судов и береговых объектов.

4.2 DeepLabV3 с backbone ResNet-50

Модель DeepLabV3-ResNet50 использует сверточную сеть ResNet-50 в качестве базовой архитектуры извлечения признаков. Данный backbone обладает следующими преимуществами:

- хорошо изученная и проверенная архитектура, часто используемая в задачах компьютерного зрения;
- способность извлекать богатые и устойчивые признаки за счёт глубокой структуры и остаточных связей;
- подходит для обучения на относительно сложных сценах с большим разнообразием объектов и фонов.

Использование ResNet-50 делает модель более ресурсоёмкой, однако потенциально позволяет достичь более высокой точности сегментации, что важно при анализе качества разметки и сравнении моделей.

4.3 DeepLabV3 с backbone MobileNetV3

Вторая выбранная модель — DeepLabV3-MobileNetV3 — ориентирована на более лёгкие и эффективные вычисления. MobileNetV3 разработан специально для сценариев с ограниченными вычислительными ресурсами и обладает следующими преимуществами:

- значительно меньшее количество параметров по сравнению с ResNet-50;
- более высокая скорость обучения и инференса;
- потенциальная применимость в задачах реального времени и на встраиваемых устройствах.

Для задачи автономного судовождения и обработки видеопотока в реальном времени такие свойства являются особенно актуальными, так как модели могут быть использованы непосредственно на борту судна.

5 ОБУЧЕНИЕ МОДЕЛИ

После подготовки данных и выбора архитектур следующим этапом стало обучение моделей семантической сегментации. В рамках данной работы обучение проводилось с использованием фреймворка PyTorch, что позволило гибко управлять процессом обучения, экспериментировать с гиперпараметрами и собирать метрики качества.

В качестве основной метрики качества использовалась mIoU (mean Intersection over Union), так как она является стандартной метрикой для задач сегментации и хорошо отражает качество предсказаний по всем классам.

5.1 Подбор гиперпараметров

Для повышения качества модели был выполнен подбор гиперпараметров. Основной целью данного этапа являлся поиск таких значений параметров обучения, при которых модель показывает наилучшее качество на валидационной выборке.

В рамках работы рассматривались следующие гиперпараметры:

- скорость обучения (learning rate);
- размер батча (batch size).

Для упрощения экспериментов использовался перебор фиксированного набора значений (grid search). Были выбраны следующие значения:

- learning rate: 1e-3, 1e-4;
- batch size: 2, 4;

Перебор всех комбинаций осуществлялся с использованием стандартного подхода, при котором каждая пара гиперпараметров оценивается независимо.

Процесс подбора был организован следующим образом:

- для каждой комбинации learning rate и batch size создавались новые DataLoader для обучающей и валидационной выборок;
- инициализировалась модель DeepLabV3 с backbone ResNet-50, предобученная на датасете COCO;

- последний слой классификатора заменялся для соответствия числу классов текущей задачи;
- в качестве оптимизатора использовался Adam, а в качестве функции потерь — CrossEntropyLoss с игнорированием пикселей с меткой 255;
- обучение проводилось в ускоренном режиме (одна короткая эпоха), так как целью этапа являлось сравнение гиперпараметров, а не финальное обучение модели;
- после обучения выполнялась оценка модели на валидационной выборке с вычислением значения mIoU.

Для каждой конфигурации сохранялось среднее значение mIoU по всей валидационной выборке. По результатам экспериментов выбиралась комбинация гиперпараметров, обеспечивающая наилучшее качество.

В ходе экспериментов были получены следующие значения mIoU:

- learning rate = 0.001, batch size = 2 → mIoU ≈ 0.49;
- learning rate = 0.001, batch size = 4 → mIoU ≈ 0.46;
- learning rate = 0.0001, batch size = 2 → mIoU ≈ 0.55;
- learning rate = 0.0001, batch size = 4 → mIoU ≈ 0.55 (наилучший результат).

Наилучшее качество было достигнуто при следующих параметрах обучения:

- learning rate = 0.0001;
- batch size = 4.

Данная конфигурация обеспечила максимальное значение метрики mIoU на валидационной выборке и была выбрана для дальнейшего обучения и сравнения моделей.

Выходы по подбору гиперпараметров

Проведённый подбор показал, что:

- более низкое значение скорости обучения ($1e-4$) обеспечивает более стабильное обучение и лучшее качество сегментации;

- увеличение размера батча до 4 положительно влияет на итоговую метрику при выбранной скорости обучения;
- даже ограниченный перебор гиперпараметров позволяет заметно улучшить качество модели по сравнению с использованием значений «по умолчанию».

5.2 Описание параметров обучения и результатов

Обучение моделей семантической сегментации проводилось в течение 30 эпох с использованием обучающей и валидационной выборок. В качестве функции потерь применялась кросс-энтропия с игнорированием пикселей, относящихся к служебному классу (значение 255), что позволило исключить из расчётов некорректно размеченные или неиспользуемые области изображения. Оптимизация параметров нейронных сетей осуществлялась с помощью алгоритма Adam с фиксированной скоростью обучения.

В процессе обучения после каждой эпохи выполнялась оценка качества модели на валидационной выборке с использованием метрики mean Intersection over Union (mIoU). Для предотвращения деградации качества и фиксации наилучшего состояния модели применялся механизм сохранения весов: сохранялась модель, показавшая максимальное значение mIoU на валидации за всё время обучения.

Модель DeepLabV3-ResNet50 продемонстрировала устойчивый рост качества на обучающей выборке: значение функции потерь постепенно снижалось, а метрика mIoU увеличивалась по мере роста числа эпох. Уже на ранних этапах обучения (1–5 эпохи) наблюдалось быстрое улучшение качества сегментации, что свидетельствует о высокой выразительной способности архитектуры.

Максимальное значение mIoU на валидационной выборке составило 0.6062, достигнутое на 26-й эпохе. После этого наблюдались колебания значения метрики, что указывает на приближение модели к пределу обобщающей способности на данном наборе данных. Несмотря на

дальний рост mIoU на обучающей выборке, улучшение результатов на валидации не происходило, что подтверждает целесообразность использования сохранённой лучшей версии модели.

В целом, модель DeepLabV3-ResNet50 обеспечивает высокое качество сегментации, однако характеризуется значительной вычислительной сложностью и длительным временем обучения.

Модель DeepLabV3-MobileNetV3 обучалась при тех же параметрах, что и модель с ResNet-50, что обеспечивает корректность сравнения результатов. Данная архитектура демонстрировала более высокую скорость обучения за счёт меньшего количества параметров и оптимизированной структуры базовой сети.

В ходе обучения наблюдался стабильный рост качества сегментации без резких колебаний значений метрики на валидационной выборке. Максимальное значение mIoU составило 0.5791, достигнутое на 30-й эпохе обучения. В отличие от более тяжёлой модели, MobileNetV3 показала более плавную динамику изменения метрик и меньшую склонность к переобучению.

Несмотря на несколько более низкое итоговое значение mIoU по сравнению с DeepLabV3-ResNet50, данная модель обладает существенными преимуществами с точки зрения вычислительной эффективности и может быть использована в задачах, где важны скорость работы и ограниченные аппаратные ресурсы.

Проведённые эксперименты показывают, что обе модели являются пригодными для задачи семантической сегментации морских сцен. DeepLabV3-ResNet50 обеспечивает более высокое качество сегментации, тогда как DeepLabV3-MobileNetV3 демонстрирует лучший баланс между точностью и вычислительной сложностью. Выбор конкретной архитектуры может осуществляться в зависимости от требований к точности и доступным вычислительным ресурсам.

6 КОНТЕЙНЕРИЗАЦИЯ МОДЕЛИ

Для обеспечения воспроизводимости, удобного развертывания и интеграции обученной модели в геоинформационную систему была реализована контейнеризация с использованием Docker. Контейнеризация позволяет запускать модель на любых системах с установленным Docker, обеспечивая идентичное окружение и корректную работу всех зависимостей.

6.1 Общая структура контейнера

В качестве базового образа использовался официальный образ PyTorch с поддержкой CUDA, что обеспечивает наличие предустановленной библиотеки PyTorch с поддержкой GPU. Это критически важно для ускоренного обучения и инференса сегментационных моделей. В рабочую директорию контейнера копируются все зависимости проекта и основной код, что позволяет запускать сервис инференса в стандартизированном окружении.

6.2 Сервис инференса

Для взаимодействия с моделью реализован сервер на базе FastAPI. Сервис выполняет следующие функции:

Загружает выбранную архитектуру DeepLabV3 с настроенным классификатором под количество классов.

Производит предобработку входного изображения, включая масштабирование и преобразование в формат, пригодный для модели.

Выполняет предсказание маски классов и возвращает результат в формате изображения, пригодного для визуализации или дальнейшей обработки.

Такой подход обеспечивает быстрый и удобный доступ к модели через HTTP-запросы, позволяя интегрировать сегментацию изображений в более крупные информационные системы.

6.3 Сборка и запуск контейнера

Контейнер собирается в единую самодостаточную среду, которая включает все зависимости и код проекта. После сборки контейнер можно запускать на любой машине с установленным Docker, включая поддержку GPU, что позволяет максимально эффективно использовать вычислительные ресурсы.

Интерфейс сервиса реализован через REST API, что позволяет отправлять изображения на сервер и получать результат в виде маски. Такой подход упрощает автоматизацию процесса инференса и интеграцию модели в внешние сервисы.

Пример маски, получающейся на выходе представлен на рисунке 4.



Рисунок 4 – Пример маски в результате работы модели

6.4 Результаты контейнеризации

Контейнеризация позволила:

Обеспечить стабильную и воспроизводимую работу модели на любых устройствах с поддержкой Docker и GPU.

Упростить развертывание модели для интеграции в геоинформационную систему.

Создать стандартизованный интерфейс взаимодействия с моделью через REST API, облегчая автоматизацию обработки изображений.

Таким образом, контейнер предоставляет готовую платформу для быстрого тестирования и промышленного применения сегментационной модели, минимизируя проблемы с совместимостью и настройкой окружения.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была проведена разработка содержательной части и программной оболочки геоинформационной системы для обработки визуальных данных с использованием методов глубокого обучения. Основные результаты исследования включают:

- подготовку и обработку данных: был собран набор изображений и соответствующих аннотаций, разработаны специализированные загрузчики данных для семантической и паноптической сегментации, что обеспечило гибкую и удобную работу с различными типами разметки;
- выбор и настройку моделей: для решения задачи сегментации были выбраны архитектуры DeepLabV3 с различными бэкбонами (ResNet50 и MobileNetV3), что позволило оценить эффективность моделей с точки зрения точности и производительности;
- обучение и оценку моделей: проведено обучение моделей с подбором гиперпараметров и отслеживанием метрики mIoU. Максимальные значения mIoU составили 0.6062 для DeepLabV3-ResNet50 и 0.5791 для DeepLabV3-MobileNetV3, что демонстрирует достаточно высокую точность сегментации объектов различных классов;
- сохранение и использование лучших моделей: реализован механизм сохранения модели с наилучшей производительностью на валидационном наборе, что обеспечивает возможность последующего применения в геоинформационной системе без повторного обучения;
- контейнеризация модели: разработан Docker-контейнер для развертывания обученной модели с предустановленным окружением PyTorch и всеми зависимостями. Контейнер предоставляет REST API через FastAPI для инференса изображений, что обеспечивает стабильную и воспроизводимую работу модели на любых системах с поддержкой GPU, а также упрощает интеграцию сегментационной модели в геоинформационную систему.

В результате проделанной работы создана гибкая и масштабируемая система, которая может быть использована для автоматической сегментации изображений в рамках ГИС, включая задачи морского мониторинга и анализа объектов на различных визуальных данных.

Дальнейшее развитие проекта может включать:

- интеграцию дополнительных архитектур нейронных сетей для повышения точности сегментации;
- использование более крупных и разнообразных наборов данных;
- разработку визуального интерфейса для анализа результатов и управления процессом сегментации.

Таким образом, работа демонстрирует практическую применимость современных методов глубокого обучения для обработки пространственных данных и подтверждает возможность создания эффективных инструментов для автоматизированного анализа изображений в геоинформационных системах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- 1 Что такое ResNet-50 и какова его роль в компьютерном зрении? // ultralytics URL: <https://www.ultralytics.com/ru/blog/what-is-resnet-50-and-what-is-its-relevance-in-computer-vision> (дата обращения: 09.01.2026)
- 2 MobileNet V3 // Torchvision main documentation URL: <https://docs.pytorch.org/vision/main/models/mobilenetv3.html> (дата обращения: 09.01.2026)
- 3 LaRS-segmentation-train-and-inference // GitHub URL: <https://github.com/KinitoSs/LaRS-segmentation-train-and-inference/tree/main> (дата обращения: 09.01.2026)
- 4 LaRS: A Diverse Panoptic Maritime Obstacle Detection Dataset and Benchmark // GitHub Pages URL: <https://lojzezust.github.io/lars-dataset/> (дата обращения: 09.01.2026)
- 5 DeepLabV3 // Torchvision main documentation URL: <https://docs.pytorch.org/vision/main/models/deeplabv3.html> (дата обращения: 09.01.2026)

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода обучения моделей

```
import os
import json
import zipfile
from tqdm import tqdm
from PIL import Image
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import models, transforms as T
import torchvision.transforms.functional as TF
device = "cuda" if torch.cuda.is_available() else "cpu"
num_epochs = 30
lr = 1e-4
batch_size = 4
print("Device:", device)
!wget -O lars_v1.0.0_images.zip "https://box.vicos.si/lars/lars_v1.0.0_images.zip"
with zipfile.ZipFile("lars_v1.0.0_images.zip", 'r') as zip_ref:
    zip_ref.extractall()
!wget -O lars_v1.0.0_annotations.zip "https://box.vicos.si/lars/lars_v1.0.0_annotations.zip"
with zipfile.ZipFile("lars_v1.0.0_annotations.zip", 'r') as zip_ref:
    zip_ref.extractall()
class LaRSPanopticDataset(Dataset):
    def __init__(self, root_dir, split="train", img_size=(256, 256)):
        self.split_dir = os.path.join(root_dir, split)
        self.images_dir = os.path.join(self.split_dir, "images")
        self.masks_dir = os.path.join(self.split_dir, "panoptic_masks")
        self.img_size = img_size
        # список файлов
        list_path = os.path.join(self.split_dir, "image_list.txt")
        with open(list_path, "r") as f:
            self.files = [x.strip() for x in f.readlines()]
        # Паноптическая аннотация JSON
        ann_path = os.path.join(self.split_dir, "panoptic_annotations.json")
        with open(ann_path, "r") as f:
            self.ann = json.load(f)
            # словарь: filename -> mask id
            self.file_to_mask = {x["file_name"]: x["id"] for x in self.ann["images"]}
            # категории: id -> индекс (0..num_classes-1)
            self.cat_ids = sorted([c["id"] for c in self.ann["categories"]])
            self.id2idx = {cid: i for i, cid in enumerate(self.cat_ids)}
    def __len__(self):
        return len(self.files)
    def _find_file(self, folder, name):
        for ext in [".png", ".jpg", ".jpeg"]:
            path = os.path.join(folder, name + ext)
            if os.path.exists(path):
                return path
        raise FileNotFoundError(f"No file found for {name} in {folder}")

```

```

def __getitem__(self, idx):
    name = self.files[idx]
    # ---- image ----
    img_path = self._find_file(self.images_dir, name)
    image = Image.open(img_path).convert("RGB")
    image = TF.resize(image, self.img_size)
    image = TF.to_tensor(image)  # [3,H,W]
    # ---- mask ----
    mask_path = self._find_file(self.masks_dir, name)
    mask = Image.open(mask_path)
    mask = TF.resize(mask, self.img_size,
                     interpolation=Image.NEAREST)
    mask = np.array(mask)  # [H,W] или [H,W,3]
    if mask.ndim == 3:  # берем только первый канал
        mask = mask[:, :, 0]
    # переводим ID в индексы 0..num_classes-1
    mask_idx = np.zeros_like(mask, dtype=np.int64)
    for k, v in self.id2idx.items():
        mask_idx[mask==k] = v
    mask_idx = torch.from_numpy(mask_idx)  # [H,W]
    return image, mask_idx

root_dir = "/content"
img_size = (256, 256)
batch_size = 4
train_dataset = LaRSPanopticDataset(root_dir, split="train",
                                     img_size=img_size)
val_dataset = LaRSPanopticDataset(root_dir, split="val",
                                   img_size=img_size)
train_loader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2,
    drop_last=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=2,
    drop_last=False
)
def compute_mIoU(pred, target, num_classes, ignore_index=255):
    pred = pred.argmax(1)  # [B,H,W]
    pred = pred.cpu().numpy()
    target = target.cpu().numpy()
    ious = []
    for cls in range(num_classes):
        pred_i = pred == cls
        target_i = target == cls
        mask = target != ignore_index
        intersection = np.logical_and(pred_i, target_i) & mask
        union = np.logical_or(pred_i, target_i) & mask
        if union.sum() == 0:
            ious.append(np.nan)
        else:

```

```

        ious.append(intersection.sum() / union.sum())
    return np.nanmean(ious)
def train_model(model, model_name):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss(ignore_index=255)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    best_miou = 0.0
    for epoch in range(num_epochs):
        print(f"\n[{model_name}] Epoch {epoch+1}/{num_epochs}")
        # ===== TRAIN =====
        model.train()
        train_losses = []
        train_mious = []
        for imgs, masks in tqdm(train_loader):
            imgs = imgs.to(device)
            masks = masks.to(device).long()
            optimizer.zero_grad()
            outputs = model(imgs) ["out"]
            loss = criterion(outputs, masks)
            loss.backward()
            optimizer.step()
            miou = compute_mIoU(outputs, masks, num_classes)
            train_losses.append(loss.item())
            train_mious.append(miou)
        print(
            f"Train | Loss: {np.mean(train_losses):.4f}, "
            f"mIoU: {np.mean(train_mious):.4f}"
        )
        # ===== VALIDATION =====
        model.eval()
        val_mious = []
        with torch.no_grad():
            for imgs, masks in val_loader:
                imgs = imgs.to(device)
                masks = masks.to(device).long()

                outputs = model(imgs) ["out"]
                miou = compute_mIoU(outputs, masks, num_classes)
                val_mious.append(miou)

        val_miou = np.mean(val_mious)
        print(f"Val | mIoU: {val_miou:.4f}")
        # ===== SAVE BEST =====
        if val_miou > best_miou:
            best_miou = val_miou
            torch.save(
                model.state_dict(),
                f"{model_name}_best.pth"
            )
            print(f"🔥 Saved best model ({best_miou:.4f})")
    print(f"\n✅ {model_name} finished. Best mIoU: {best_miou:.4f}")
num_classes = 11
model1 = models.segmentation.deeplabv3_resnet50(pretrained=True)
model1.classifier[4] = nn.Conv2d(256, num_classes, kernel_size=1)
train_model(model1, "DeepLabV3_ResNet50")
model2 =
models.segmentation.deeplabv3_mobilenet_v3_large(pretrained=True)

```

```
model2.classifier[4] = nn.Conv2d(256, num_classes, kernel_size=1)
train_model(model2, "DeepLabV3_MobileNetV3")
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг кода использования модели

```
Dockerfile:  
FROM pytorch/pytorch:2.9.1-cuda12.6-cudnn9-runtime  
WORKDIR /app  
# Зависимости  
COPY requirements.txt .  
RUN pip install uv  
RUN uv pip install --system --no-cache-dir -r requirements.txt  
# Код  
COPY . .  
EXPOSE 8000  
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]  
  
app.py:  
from fastapi import FastAPI, UploadFile, File, Response  
from PIL import Image  
import io  
import numpy as np  
import cv2  
from inference import load_model, predict  
app = FastAPI()  
model = load_model("mobilenet")  
@app.post("/predict")  
async def predict_image(file: UploadFile = File(...)):  
    image = Image.open(io.BytesIO(await file.read())).convert("RGB")  
    mask = predict(image, model)  
    mask_png = (mask * 20).astype("uint8")  
    _, encoded = cv2.imencode(".png", mask_png)  
    return Response(  
        content=encoded.tobytes(),  
        media_type="image/png"  
)  
  
inference.py:  
import torch  
import torchvision.models as models  
from torchvision import transforms  
import numpy as np  
from PIL import Image  
# Устройство  
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"  
# Глобальный трансформ для всех изображений  
transform = transforms.Compose([  
    transforms.Resize((256, 256)),  
    transforms.ToTensor()  
)  
def load_model(model_type="resnet50", num_classes=11):  
    """  
    Загружает модель DeepLabV3 с заданной архитектурой и количеством  
    классов.  
    """  
    if model_type == "resnet50":  
        model = models.segmentation.deeplabv3_resnet50(  
            weights=None, pretrained_backbone=False, aux_loss=False
```

```

        )
weights = "models/DeepLabV3_ResNet50_best.pth"
else:
    model = models.segmentation.deeplabv3_mobilenet_v3_large(
        weights=None, pretrained_backbone=False, aux_loss=False
    )
    weights = "models/DeepLabV3_MobileNetV3_best.pth"
# Меняем классификатор под нужное количество классов
model.classifier[4] = torch.nn.Conv2d(256, num_classes, 1)
# Загружаем веса, пропуская несоответствующие ключи
state_dict = torch.load(weights, map_location=DEVICE)
model.load_state_dict(state_dict, strict=False)
model.to(DEVICE)
model.eval()
return model
def predict(image: Image.Image, model):
    """
    Делает предсказание маски для одного изображения PIL.Image.
    Возвращает numpy-массив с индексами классов [H, W].
    """
    x = transform(image).unsqueeze(0).to(DEVICE)
    with torch.no_grad():
        out = model(x) ["out"]
        mask = out.argmax(1).squeeze().cpu().numpy()
    return mask

```

ПРИЛОЖЕНИЕ В
Сведения о самостоятельности выполнения работы

Работа по проекту «Проектирование и разработка содержательной части и программной оболочки системы машинного обучения для сегментации морских сцен» выполнена нами самостоятельно.

Используемые в работе материалы и концепции из публикуемой литературы и других источников имеют ссылки на них.

Электронный вариант работы на цифровом носителе переданы мной на кафедру/отделение.

«9» января 2026 г.  Абдурахманова К. О.

(подпись)

«9» января 2026 г.  Архаров Н. М.

(подпись)