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 технологи и автоматизированных систем

(наименование высшей школы / филиала / института / колледжа)

**КУРСОВОЙ ПРОЕКТ**

|  |  |  |
| --- | --- | --- |
| По модулю | | Глубокое обучение. Продвинутый уровень |
|  | | |
|  | | |
| На тему | Определение сплоченности льда с помощью нейронных сетей | |
|  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ф.И.О.  обучающихся | Наименование направления подготовки / специальности | Курс | Группа | Ф.И.О. руководителя(-ей) должность / уч. степень / звание |
| Пахнев Дмитрий Владимирович | 09.04.02 Информационные системы и технологии | 2 | 151266 | Васендина И.С., доцент, к.т.н. |
| Майоров Даниил Михайлович | 09.04.02 Информационные системы и технологии | 2 | 151266 | Васендина И.С., доцент, к.т.н. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Признать, что проект выполнен и защищен с отметкой |  |  |  | 26.04.2024 |
|  |  | (отметка прописью) |  | (дата) |
| Руководитель |  |  |  | И.С. Васендина |
|  |  | (подпись руководителя) |  | (инициалы, фамилия) |

Архангельск 2024

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 модулю | | | Глубокое обучение. Продвинутый уровень | | | | | | | | | | | |  | |
|  |  | | | | (наименование дисциплины) | | | | | | | | | | |  | |
|  | студентам | ВШИТАС | | | высшей школы | | | 2 | | | курса | 151266 | | | группы |  | |
|  | Пахнев Дмитрий Владимирович, Майоров Даниил Михайлович | | | | | | | | | | | | | | |  | |
|  | (Ф.И.О. студентов) | | | | | | | | | | | | | | |  | |
|  | |  |  | | --- | --- | | 09.04.02 «Информационные системы и технологии» | | | (код и наименование направления подготовки/специальности) | | | | | | | | | | | | | | | | |  | |
|  | ТЕМА: | | Определение сплоченности льда с помощью нейронных сетей | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | |  | |
|  | ИСХОДНЫЕ ДАННЫЕ: | | | | |  | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | | | |  | |
|  | ЗАДАНИЕ: | | | | | | | | | | | | | | |  | |
|  | 1. Проанализировать предметную область | | | | | | | | | | | | | | |  | |
|  | 1. Создать набор данных для обучения моделей | | | | | | | | | | | | | | |  | |
|  | 1. Обучить несколько моделей на созданном наборе данных | | | | | | | | | | | | | | |  | |
|  | 1. Интегрировать обученную модель в десктоп приложение | | | | | | | | | | | | | | |  | |
|  |
|  |
|  |
|  |  | | | | | | | | | | | | | | |  | |
|  | Руководитель работы | | | | | | доцент, к.т.н. | |  |  | | |  | И.С. Васендина | |  | |
|  |  | | | |  | | (должность) | |  | (подпись) | | |  | (инициалы, фамилия) | | |

Архангельск 2024

ОГЛАВЛЕНИЕ

[1. Проанализировать предметную область 2](#_Toc164972757)

[2. Создать набор данных для обучения моделей 2](#_Toc164972758)

[3. Обучить несколько моделей на созданном наборе данных 2](#_Toc164972759)

[4. Интегрировать обученную модель в десктоп приложение 2](#_Toc164972760)

[Введение 4](#_Toc164972761)

[1 Описание исследования 5](#_Toc164972762)

[1.1 Анализ предметной области 5](#_Toc164972763)

[1.2 Анализ существующих исследований 7](#_Toc164972764)

[2 Создание набора данных 12](#_Toc164972765)

[2.1 Источники данных 12](#_Toc164972766)

[2.2 Классы сплоченности 12](#_Toc164972767)

[2.3 Обработка данных 13](#_Toc164972768)

[2.4 Разметка данных 16](#_Toc164972769)

[3 Обучение моделей машинного обучения 23](#_Toc164972770)

[4 Интеграция модели в приложение 29](#_Toc164972771)

[Заключение 34](#_Toc164972772)

[Список использованных источников 35](#_Toc164972773)

[Приложение А (обязательное) программный код для обучения моделей 36](#_Toc164972774)

[Приложение Б (обязательное) программный код «desktop приложения» 46](#_Toc164972775)

ВВЕДЕНИЕ

В ходе выполнения данного проекта были поставлены следующие задачи:

* исследовать предметную область;
* создать набор данных для сегментации спутниковых радиолокационных снимков арктического региона;
* обучить несколько моделей с разными гиперпараметрами;
* интегрировать лучшую модель в приложение.

В результате выполнения задачи должен быть разработан готовый набор данных, содержащий в себе изображения радиолокационных снимков льдов, а также маски к данным изображениям. Также должно быть проведено обучение нескольких нейронных сетей на созданном наборе данных и интегрирована лучшая модель в приложение.

1. Описание исследования
   1. Анализ предметной области

Вопрос исследования и освоения Арктического региона имеет стратегическое значение для Российской Федерации в силу целого ряда факторов. Во-первых, в данном регионе проходит Северный морской путь, который представляет собой кратчайший морской маршрут между европейскими и азиатскими портами. По сравнению с традиционными маршрутами через Суэцкий канал или вокруг Африки, Северный морской путь значительно сокращает время и расстояние перевозок, что делает его экономически более выгодным для международных торговых компаний.

Во-вторых, Северный морской путь обладает геополитическими преимуществами для России. Большая часть акватории этого маршрута проходит через российские арктические воды, что позволяет России контролировать и регулировать его использование. Это дает России рычаги влияния в области международной торговли и навигации, а также возможность развивать собственную арктическую инфраструктуру, такую как порты, судоремонтные базы и средства навигационного обеспечения.

Кроме того, Арктический регион обладает значительными запасами природных ресурсов, таких как нефть, газ, уголь, редкие металлы и другие полезные ископаемые. По мере таяния арктических льдов и развития технологий добычи, эти ресурсы становятся все более доступными для разработки. Контроль над арктическими ресурсами может обеспечить России существенные экономические выгоды и укрепить ее позиции на мировых рынках энергоносителей и минерального сырья.

Помимо экономических факторов, освоение Арктики имеет важное научное и экологическое значение. Изучение арктических экосистем, климатических процессов и их влияния на глобальное изменение климата представляет большой интерес для международного научного сообщества. Россия, как страна, обладающая обширными арктическими территориями, может внести существенный вклад в эти исследования.

Таким образом, освоение Арктического региона является стратегической задачей для России, охватывающей экономические, геополитические, научные и экологические аспекты. Развитие арктической инфраструктуры, обеспечение безопасности судоходства, рациональное использование природных ресурсов и сохранение уникальных арктических экосистем – все эти вопросы требуют комплексного подхода и серьезных инвестиций со стороны российского государства.

Присутствие ледовой обстановки в арктическом регионе создает опасность для судов и повышает риски аварийных ситуаций. Это делает проведение судоходства в данном регионе более сложным и опасным. Для таких зон обычно создаются карты, обозначающие зоны, в которых находится тот или иной тип льда с определенной сплоченностью. При их создании используется метод основанный на измерении радиолокационного сигнала, отраженного от поверхности льда.

Спутник отправляет радиолокационный сигнал к поверхности льда, который отражается обратно к спутнику. Измеряется время задержки между отправкой и приемом сигнала, а также его интенсивность. На основе этих данных строятся изображения, которые позволяют определить степень сплоченности льда. Преимуществом радиолокационных систем является их способность работать в любое время суток и при любых погодных условиях, поскольку радиоволны не зависят от солнечного освещения и могут проникать сквозь облачный покров. Это обеспечивает непрерывный мониторинг арктических льдов даже в периоды полярной ночи или сильной облачности.

Сплоченность льда определяется визуально и оценивается по 10-балльной шкале, в которой 0 баллов – это открытая вода, и 10 баллов – цельный лед. На рисунке 1 можно увидеть примеры сплоченности льда.

Также можно сказать, что сплоченность льда в секторе – это отношение площади открытой воды и льдин. Для разных секторов создаются карты сплоченности. Из-за разнородности льда в океане, различных искажений радиолокационных спутниковых снимков, погодных условий визуальный анализ радиолокационных спутниковых снимков может быть весьма трудным, это может стать причиной неточности карт. В этой задаче могут помочь алгоритмы глубокого обучения, которые так же могут и предсказать, как будет меняться ледовая обстановка в будущем.

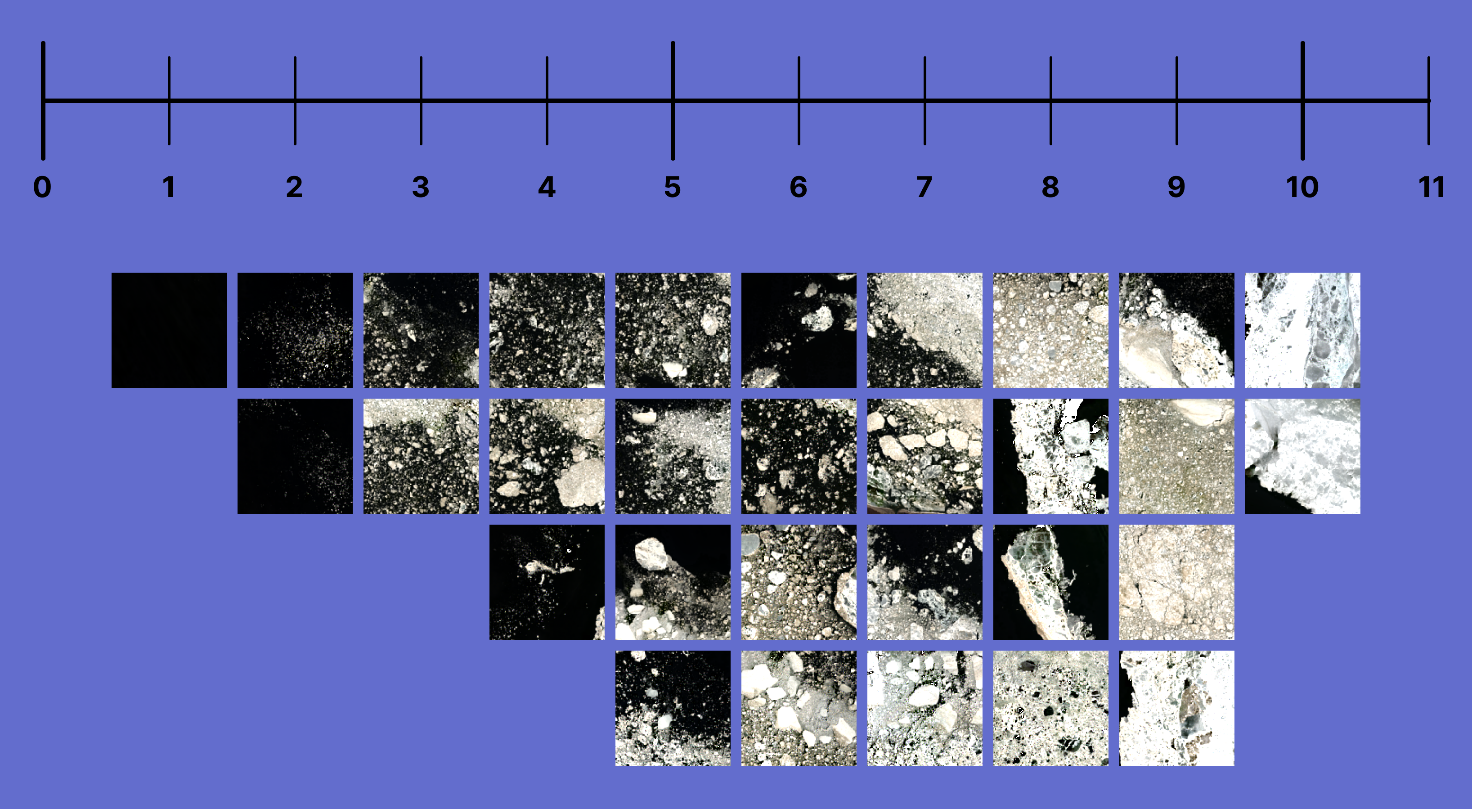


Рисунок - 1 – Сплоченность льда по 10 бальной шкале

* 1. Анализ существующих исследований

Существуют методы построения маршрутов судов с учетом ледовой обстановки. Среди них распространены алгоритмы нахождения кратчайшего пути в графе. Описанный способ предполагает использование множества точек изображения с картой ледовой обстановки, а затем соединение этих точек с близлежащими. Веса предлагается определять исходя из опасности обледенения в указанном районе для судна (в зависимости от самого судна, а также от характеристик льда). Предложено использовать алгоритм Дейкстры в качестве алгоритма нахождения кратчайшего пути. Данный алгоритм был разработан нидерландским учёным Эдсгер Дейкстра в 1959 году и предполагает собой эффективный метод нахождение самого короткого пути в графе. На каждом из шагов алгоритм рассматривает ближайшую не посещённую раннее точку и обновляет расстояние до соседних точек в том случае, если новый путь оказывается короче. Процесс повторяется до посещения всех имеющихся вершин. Как отмечают авторы, даже при самых худших сценариях работа алгоритма на сгенерированном графе занимает вполне приемлемое время. Вопрос вычисления весов графов является сложным, но авторы предлагают в качестве решения использование нечеткого системного моделирования функции, экспертных оценок и использование ретроспективных данных [1].

Кроме того, существуют методы построения маршрута во время движения судна. Для этого определённая часть команды, постоянно находящаяся на мостике корабля, занимается визуальным мониторингом округи, фиксацией показателей датчиков и спутниковых радаров. Поэтому у судов нет определённого маршрута и каждый раз приходится корректировать его в зависимости от появляющихся или пропадающих ледовых образований, из-за чего цели экспедиций иногда могут не достигаться в полном объёме.

Метод определения ледовой обстановки также известен по патенту РФ No 2425400. Суть метода заключается в отправке двух беспилотных летательных аппаратов (БПЛА), один из которых ведет видеосъемку, а второй ретранслирует сигнал. Использование этого метода не позволяет объективно оценить толщину льда. В способе по патенту No 2631966 эта задача была решена с помощью акустического измерительного аппаратно-программного комплекса. Антенны этого комплекса контактируют с поверхностью льда в заданных точках для проведения измерений. Использование этого метода обеспечивает информативность и точность ледовой разведки, но является крайне трудоемким [2].

Также используются нейросетевые модели, основанные на данных, состоящих из радиолокационных снимков. Чаще всего такие модели используют снимки со спутников типа Sentinel разработанные в рамках программы Copernicus. Sentinel-1 [3,4]. Один из таких подходов описан в статье Wang YR и Li XM «Arctic sea ice cover data from spaceborne synthetic aperture radar by deep learning». Так как радиолокационные снимки делаются в достаточно нестабильных условиях, обусловленных влиянием атмосферных воздействий, электромагнитных помехов и шумов, то после получения данных со спутника они проходят постобработку и калибровку для минимизации систематических ошибок и неравномерностей данных при дальнейшем их использовании и обучении. Изображения весьма большие и неудобны для обработки, потому они уменьшаются в размерах, в результате чего 1 пиксель изображения имеет размер 400м в реальности. Данные с разной поляризацией обрабатываются, в результате чего объединяются в RGB-композит с ненастоящими цветами. С помощью SVM классификатора были обработаны снимки, отделена открытая вода и лед друг от друга. Модифицированным модели U-net передаются изображения, разбитые на участки 256x256 пикселей. После обучения модели могут уже использоваться для неразмеченных снимков. В ходе исследований авторами было обнаружено, что различные модификации U-net имеют разную специализацию и подходят для работы с определенными типами снимков, из-за чего они были обучены на разных наборах данных и объединены в некий ансамбль, позволяющий получить более точные оценки. [5]

Другим подходом, описанным в статье Lyu H., Huang W., Mahdianpari M. «Eastern arctic sea ice sensing: First» является сопоставление с радиолокационными снимками цифровых карт канадской ледовой службы. Разметка данных представляла из себя 2 этапа: автоматический и ручной. Автоматическая проекция цифровой карты дает хороший результат, однако некоторые пиксели в однородных участках воды или льда ошибочно размечаются как участки иного класса, из-за чего необходима дополнительная ручная разметка. Для обучения модели использовалась архитектура NFNet. Основной идеей архитектуры NFNet является использование параметрической активации, а не слоев нормализации, чтобы сеть могла эффективно обучаться на больших масштабах и показывать высокие результаты при классификации изображений. Результаты обучения сравнивались с моделью Random Forest. Модель показала хорошие результаты по классификации молодых льдов. Радиолокационные снимки с двойной поляризацией, как и в исследовании, описанном ранее, позволили обучить модели для сегментации поверхности океана и ледяного покрова [6].

Кроме того, стоит упомянуть существующие открытые наборы данных в данной области. Например, канадским инженером в области data science Alexander Sylvester был создан набор данных, основанный на данных канадской ледовой службы [7, 8]. Набор сформирован с помощью языка Python, исходный код скрипта для формирования набора данных представлен на github.

Алгоритм выгрузки работает следующим образом: берется векторная карта сплоченности льда Гудзонского залива, разбивается на квадраты с определенной площадью, а для этих квадратов выгружаются спутниковые из SentinelHub на определенные даты снимки с относительно небольшим покрытием облаками.

В результате чего получается изображение ледовой обстановки в определенной области, а также соразмерная ей векторную карта с отмеченными на ней зонами сплоченности. При наличии векторной маски и спутниковых снимков хорошего качества получается набор данных, который собирается без предварительной разметки нашей командой.

Векторные карты сплоченности льда Гудзонского залива берутся с сайта канадской ледовой службы. Это подразделение канадской метеорологической службы, занимающееся задачей предоставления информации для безопасной навигации в морском пространстве Канады. Данная служба с определенными периодами формирует различные карты, описывающие состояние морских зон в данном регионе, включая сплоченность льдов.

В набор данных содержится 3392 спутниковых снимков Гудзонского залива в канадской Арктике с 1 января 2016 по 31 июля 2018. Данные из этого набора являются устаревшими, а также рассматривают другую часть Арктики.

Вышеописанные подходы в качестве исходных данных использует радиолокационные снимки и не учитывает информацию о характеристиках льда, такую как солёность пористость, толщину и другие характеристики, которые характеризируют состояние льда и его проходимость.

1. Создание набора данных
   1. Источники данных

Карты о типах и сплоченности льда в различных секторах представляются многими сервисами. Например такие карты представляют Арктический и антарктический научно-исследовательский институт (ААНИИ) на своем сайте [10].Такие карты представляют и научно-исследовательские центры других стран. Если рассматривать такие карты как потенциальные источники данных для будущего набора данных, служащего для обучения нейронных сетей, то желательно использовать изображения с географической привязкой, что позволит в геоинформационных системах сопоставить карту и спутниковый снимок. ААНИИ предоставляют такие карты, но публикуют в открытых источниках довольно редко. В данный момент мы остановились на наборе данных от National Oceanic and Atmospheric Administration (NOAA ) (США) [11].

Спутниковые радиолокационные снимкив формате geotiff для исследования были взяты с сайта Alaska Satellite Facility (ASF) (США) [12]. Для исследования использовались снимки со спутника Sentinel-1.

* 1. Классы сплоченности

Для решения задачи было принято решение выделать 4 класса:

* высокая сплоченность льда (обильное скопление ледяных образований, по шкале сплоченности 6-10 баллов);
* низкая сплоченность льда (незначительное количество льда, по шкале сплоченности 3-6 баллов);
* вода;
* суша.

Эти классы были определены для более точного анализа данных и выделения основных характеристик окружающей среды. Высокая сплоченность льда указывает на плотное обледенение поверхности, что может создавать препятствия для движения и работ. Низкая сплоченность льда, напротив, означает минимальное наличие ледяных образований, что упрощает передвижение.

Вода и суша являются основными категориями, определяющими наличие или отсутствие льда. Их выделение позволяет более точно учитывать различия в условиях окружающей среды и принимать соответствующие меры для безопасной работы.

* 1. Обработка данных

Обработка данных перед разметкой необходима для подготовки данных к анализу и классификации. Это включает в себя удаление дубликатов, пропущенных значений, ошибок, а также преобразование данных в нужный формат или структуру.

Радиолокационные снимки необходимо конвертировать в формат «jpg» поскольку инструмент, предназначенный для разметки данных, поддерживает только данный тип файлов. Для выполнения этой задачи воспользуемся программой «SNAP», которая позволяет перевести формат данных «feotiff» в «jpg». Откроем приложение и выведем скачанное радиолокационное изображение, пример показан на рисунке 2 [13].

Далее нажимаем правой кнопкой мыши на корневую папку радиолокационного изображения и выбираем «Open RGB image window». В открывшейся форме ставим настройки, который показаны на рисунке 3. В результате выполнения действия появится изображение, которое можно будет экспортировать. При экспортировании изображения необходимо выставить настройки, которые показаны на рисунке 4.

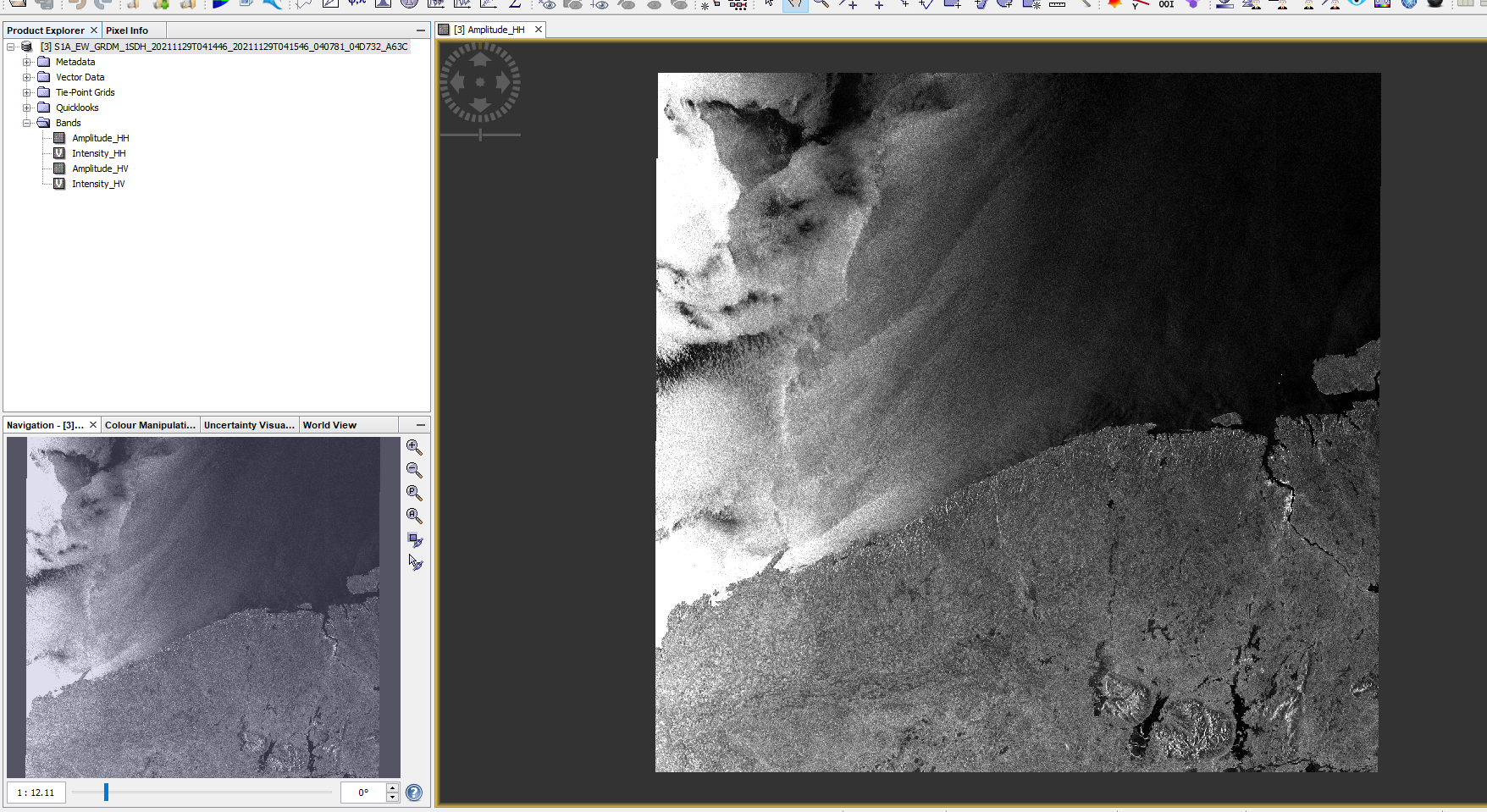


Рисунок - 2 – Радиолокационное изображение в «SNAP»

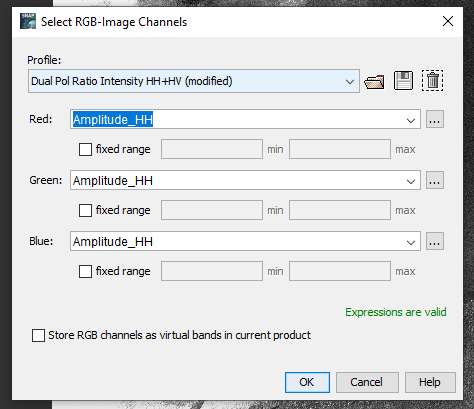


Рисунок - 3 – Параметры для создания изображения

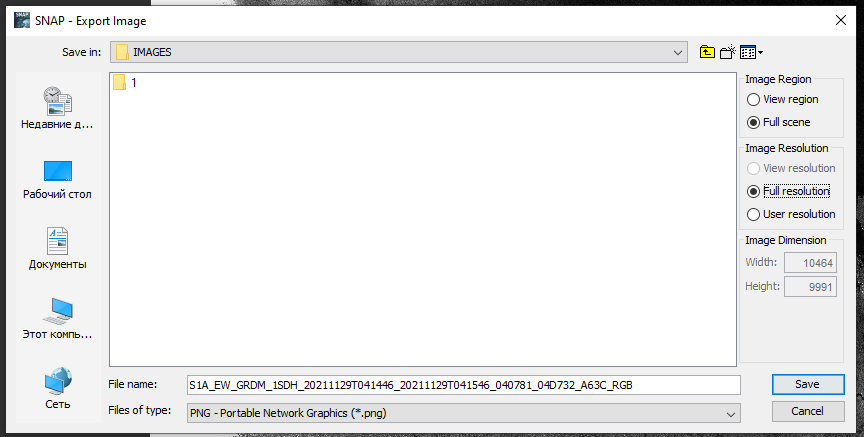


Рисунок - 4 – Настройки для экспортирования изображения

Также необходимо сопоставить изображения, которые описывались в источниках данных с помощью инструмента «QGIS» чтобы при разметке данных правильно выделить области на определенные классы. Был выбран данный инструмент так как:

* является бесплатным;
* поддерживает различные операционные системы;
* имеет мощный функционал.

На рисунке 5 показан пример как выглядит рабочее пространство в «QGIS» при загруженных изображениях. Человек, который будет размечать данные, может без проблем выделить типы сплоченности льда на изображении.

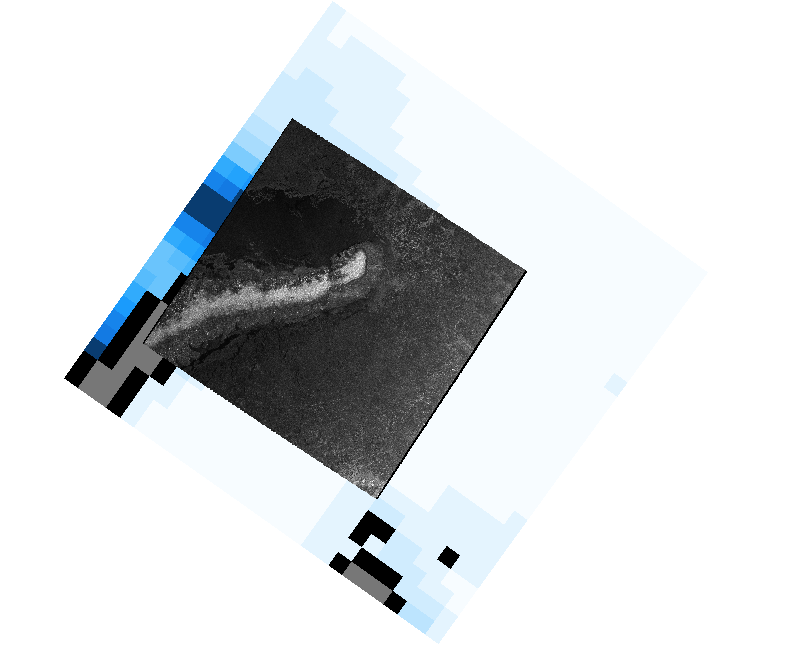


Рисунок - 5 – Наложение изображений в «QGIS»

* 1. Разметка данных

Размеченные данные используются в машинном обучении для обучения моделей на основе этих данных. Без размеченных данных алгоритмам будет сложно или даже невозможно выявить закономерности и предсказать результаты на основе имеющихся данных. Кроме того, разметка данных позволяет улучшить качество и точность моделей машинного обучения. Таким образом, разметка данных оказывает огромное влияние на развитие информационных технологий и науки в целом. Она позволяет улучшить качество и эффективность алгоритмов машинного обучения, обработки естественного языка и компьютерного зрения, а также создать новые технологии и продукты, которые помогут улучшить качество жизни людей.

Для разметки данных использовался инструмент «roboflow» так как функциональные возможности позволяют с большой скоростью и высокой точностью размечать изображения [14]. Также после создания набора данных его можно выгрузить в формате «coco json» для обучения нейронной сети.

В данном инструменте можно изначально настроить необходимые классы, которые будут использоваться при разметке. Пример созданных классов показано на рисунке 6.

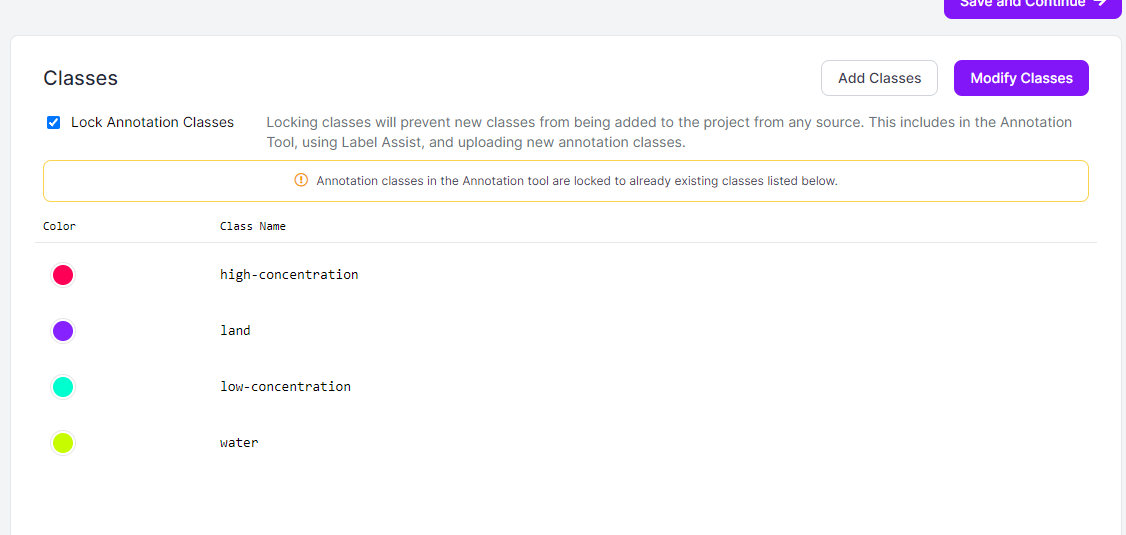


Рисунок - 6 – Созданные классы в «Roboflow»

На рисунке 7 показан основной интерфейс при разметке изображения. В левой части можно увидеть какие классы уже выделены на изображении. В правой части находится основной инструмент для разметки. Хотелось бы выделить «Label assist» c помощью которого можно эффективно выделять части изображения автоматически с помощью встроенных алгоритмов и подсказок, что позволяет сэкономить большое количество времени.

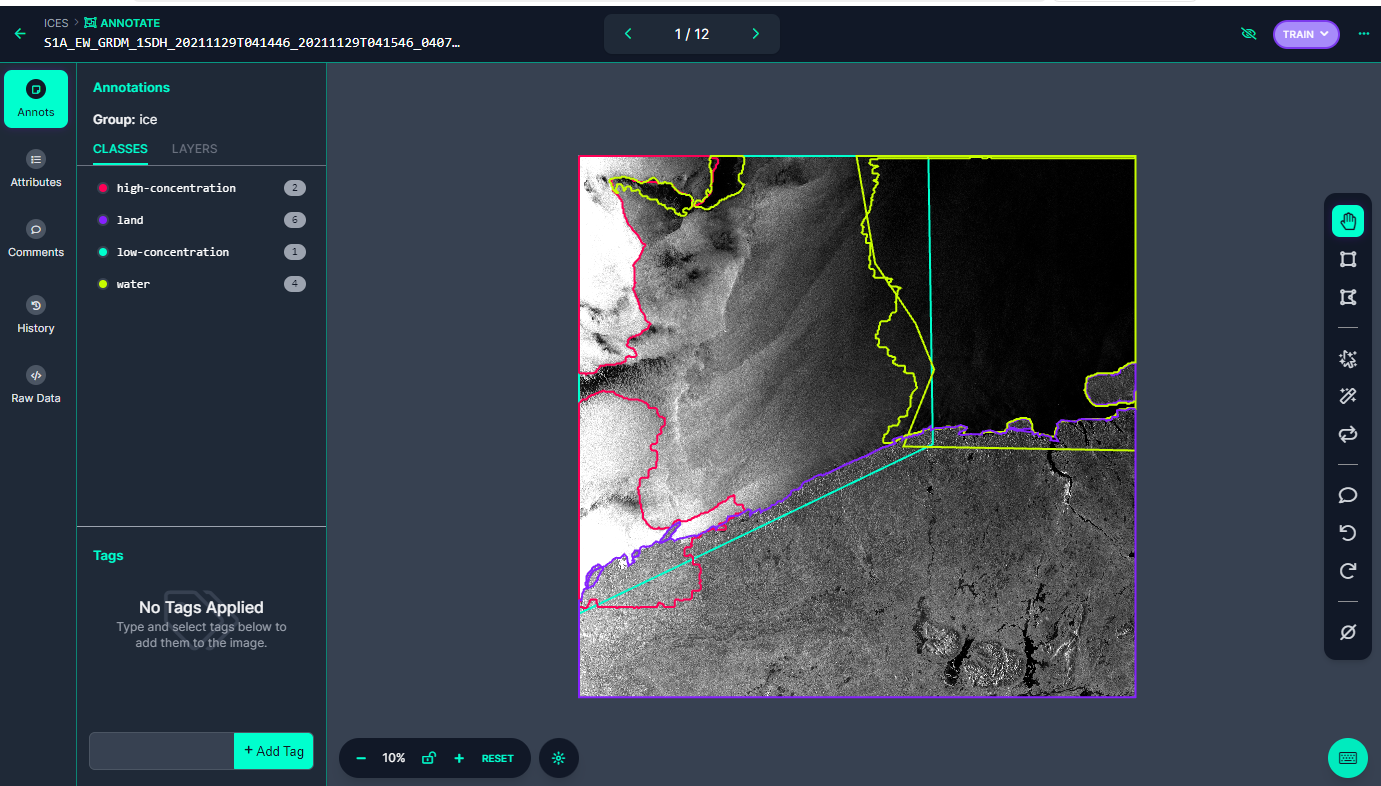


Рисунок - 7 – Основной интерфейс разметки изображений

В результате разметки получаем результат, который показан на рисунке 8.

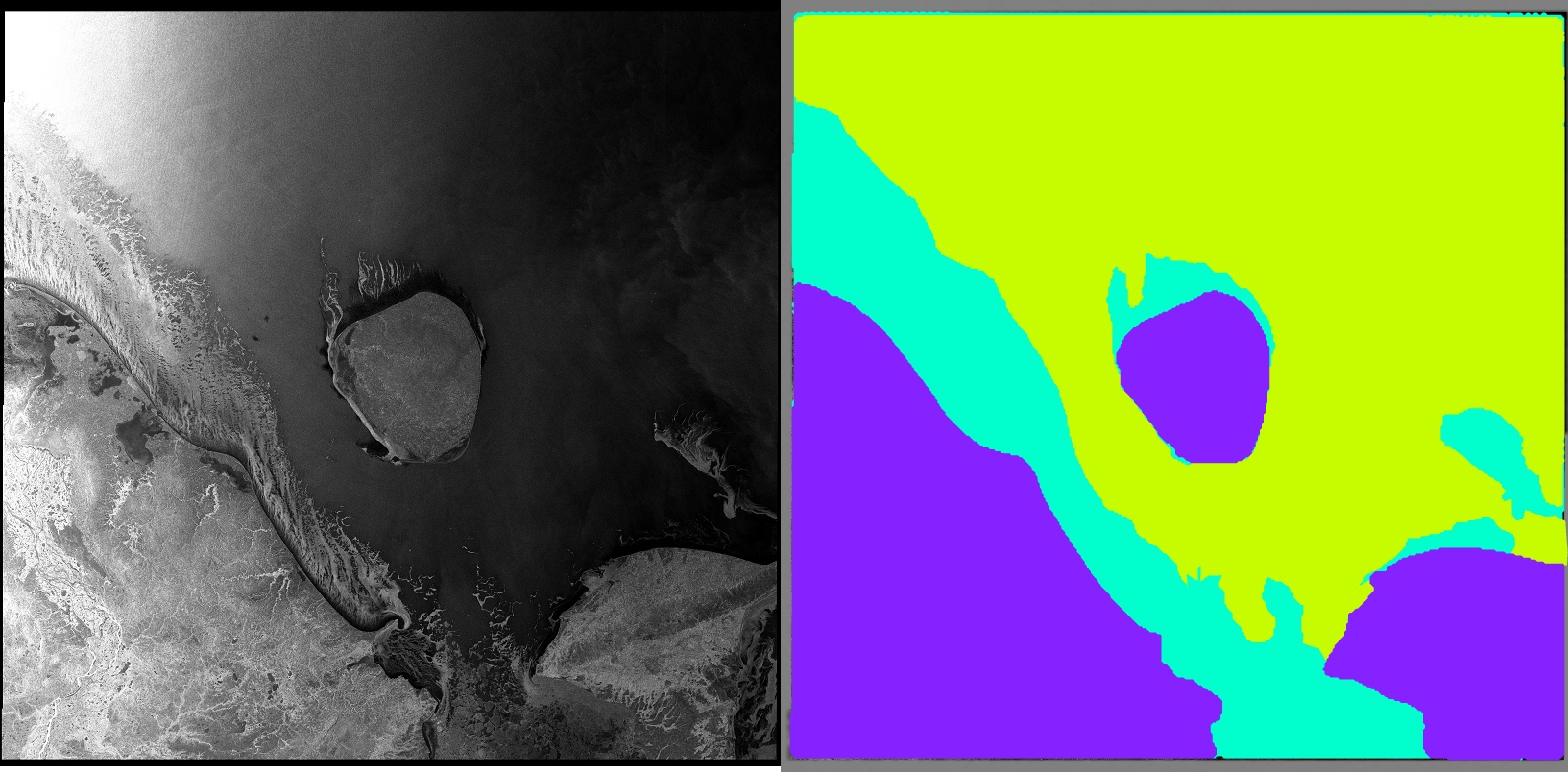


Рисунок - 8 – Разметка одного снимка в «Roboflow»

Следующим шагом выгружает готовый набор данных из «roboflow» в формате «coco json» и создаем маски с помощью скрипта, который продемонстрирован в листинге 1.

Листинг 1 – Создание маски

import json

import numpy as np

import skimage

import tifffile

import os

import shutil

def create\_mask(image\_info, annotations, output\_folder):

# Create an empty mask as a numpy array

mask\_np = np.zeros((image\_info['height'], image\_info['width']), dtype=np.uint16)

# Counter for the object number

object\_number = 1

for ann in annotations:

if ann['image\_id'] == image\_info['id']:

# Extract segmentation polygon

for seg in ann['segmentation']:

# Convert polygons to a binary mask and add it to the main mask

rr, cc = skimage.draw.polygon(seg[1::2], seg[0::2], mask\_np.shape)

#skimage.draw.set\_color(mask\_np,(rr,cc),1)

mask\_np[rr, cc] = ann['category\_id']

object\_number += 1 # We are assigning each object a unique integer value (labeled mask)

# Save the numpy array as a TIFF using tifffile library

mask\_path = os.path.join(output\_folder, image\_info['file\_name'].replace('.tif', '\_mask.tif'))

tifffile.imwrite(mask\_path, mask\_np)

print(f"Saved mask for {image\_info['file\_name']} to {mask\_path}")

def main(json\_file, mask\_output\_folder, image\_output\_folder, original\_image\_dir):

# Load COCO JSON annotations

with open(json\_file, 'r') as f:

data = json.load(f)

images = data['images']

annotations = data['annotations']

# Ensure the output directories exist

if not os.path.exists(mask\_output\_folder):

os.makedirs(mask\_output\_folder)

Окончание листинга 1

if not os.path.exists(image\_output\_folder):

os.makedirs(image\_output\_folder)

for img in images:

# Create the masks

create\_mask(img, annotations, mask\_output\_folder)

# Copy original images to the specified folder

original\_image\_path = os.path.join(original\_image\_dir, img['file\_name'])

new\_image\_path = os.path.join(image\_output\_folder, os.path.basename(original\_image\_path))

shutil.copy2(original\_image\_path, new\_image\_path)

print(f"Copied original image to {new\_image\_path}")

if \_\_name\_\_ == '\_\_main\_\_':

original\_image\_dir = r"F:\seg11\train\images" # Where your original images are stored

json\_file = r"F:\seg11\train\\_annotations.coco.json"

mask\_output\_folder = r"F:\seg11\train\masks" # Modify this as needed. Using val2 so my data is not overwritten

image\_output\_folder = r"F:\seg11\train\output" #

main(json\_file, mask\_output\_folder, image\_output\_folder, original\_image\_dir)

Так как при обучении нейронная сеть не может обрабатывает изображения размером 10000 на 10000 пикселей изображения нужно разрезать. Для этого воспользуемся скриптом, который описан в листинге 2.

Листинг 2 – Нарезание изображений

from PIL import Image

import os

def crop\_images(image\_paths, mask\_paths, output\_dir, square\_size, overlap):

# Создаем папки для сохранения нарезанных изображений и масок

cropped\_images\_dir = os.path.join(output\_dir, "cropped\_images")

cropped\_masks\_dir = os.path.join(output\_dir, "cropped\_masks")

if not os.path.exists(cropped\_images\_dir):

os.makedirs(cropped\_images\_dir)

if not os.path.exists(cropped\_masks\_dir):

os.makedirs(cropped\_masks\_dir)

for image\_path, mask\_path in zip(image\_paths, mask\_paths):

# Получаем название файла (без расширения)

filename = os.path.splitext(os.path.basename(image\_path))[0]

# Открываем изображение и маску

image = Image.open(image\_path)

mask = Image.open(mask\_path)

Продолжение листинга 2

# Преобразуем маску в черно-белый формат

mask = mask.convert("L")

width, height = image.size

for left in range(0, width - square\_size + 1, square\_size - overlap):

for top in range(0, height - square\_size + 1, square\_size - overlap):

right = min(left + square\_size, width)

bottom = min(top + square\_size, height)

if not is\_black\_frame(mask, left, top, right, bottom) and not is\_full\_of\_same\_class(mask, left, top, right, bottom):

cropped\_image = image.crop((left, top, right, bottom))

cropped\_mask = mask.crop((left, top, right, bottom))

cropped\_image.save(os.path.join(cropped\_images\_dir, f"{filename}\_{left}\_{top}.jpg"),

format='JPEG')

cropped\_mask.save(os.path.join(cropped\_masks\_dir, f"{filename}\_{left}\_{top}.jpg"),

format='JPEG')

def is\_black\_frame(mask, left, top, right, bottom):

# Проверяем, является ли рамка на маске черной

for x in range(left, right):

for y in range(top, bottom):

pixel = mask.getpixel((x, y))

if pixel == 0:

return True

return False

def is\_full\_of\_same\_class(mask, left, top, right, bottom):

classes\_count = [0,0,0,0]

all = 0

# Проверяем, является ли рамка на маске черной

for x in range(left, right):

for y in range(top, bottom):

all+=1

pixel = mask.getpixel((x, y))-1

classes\_count[pixel]+=1

for class\_count in classes\_count:

if(class\_count/all>0.9):

return True

return False

image\_folder = r"F:\seg11\train\images"

mask\_folder = r"F:\seg11\train\masks"

output\_dir = r"F:\seg11\train\crop\_dataset\_fixed"

Окончание листинга 2

image\_paths = [os.path.join(image\_folder, filename) for filename in os.listdir(image\_folder) if

filename.endswith(".jpg")]

mask\_paths = [os.path.join(mask\_folder, filename) for filename in os.listdir(mask\_folder) if filename.endswith(".jpg")]

square\_size = 992

overlap = 248

crop\_images(image\_paths, mask\_paths, output\_dir, square\_size, overlap)

Из получившийся изображении необходимо оставить те, которые имеют несколько классов чтобы сеть могла научиться распознавать различные объекты на изображениях и классифицировать их правильно. Разнообразие классов в наборе данных позволяет сети обучаться на различных типах изображений и улучшать свою способность к обобщению. Кроме того, обучение на множестве классов помогает предотвратить переобучение и повысить обобщающую способность модели.

В результате получился набор данных размером в 241 изображение. К каждому изображению имеется маска с набором классов. На рисунке 9 показан пример изображения и маски.

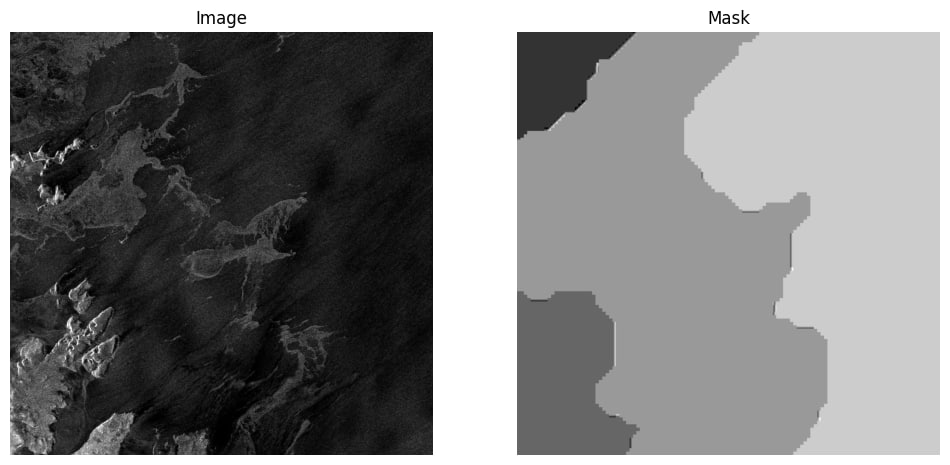


Рисунок - 9 – Изображение и маска

1. Обучение моделей машинного обучения

Для решения данной задачи были выбраны модели «U-NET» и «U-NET++» так как они себя отлично зарекомендовали в задачах сегментации благодаря использованию «skip-connections», которые позволяют передавать информацию из более глубоких слоев обратно к более поверхностным слоям. Это помогает избежать потери информации при агрегации признаков и повышает качество сегментации. Классические архитектуры моделей представлены на рисунках 10, 11.

Выборка набора данных для обучения модели производилась в следующем соотношении:

* 70% на обучение модели;
* 15% на тестовый;
* 15% на проверочный.

На вход подавались изображения в формате 3 канала 448 на 448. На выход же получался вектор из 4 классов 448 на 448.

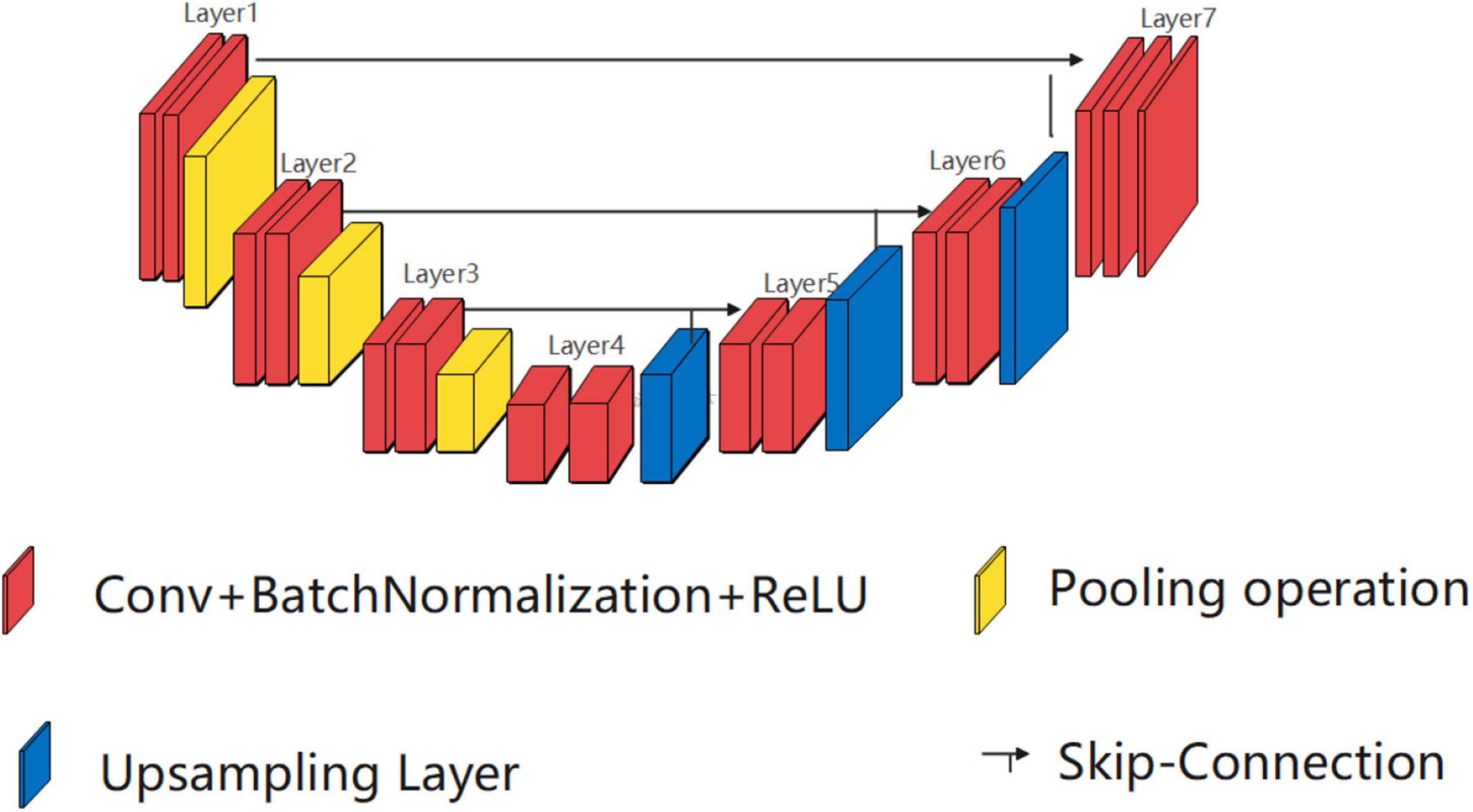


Рисунок - 10 – Архитектура «U-NET»

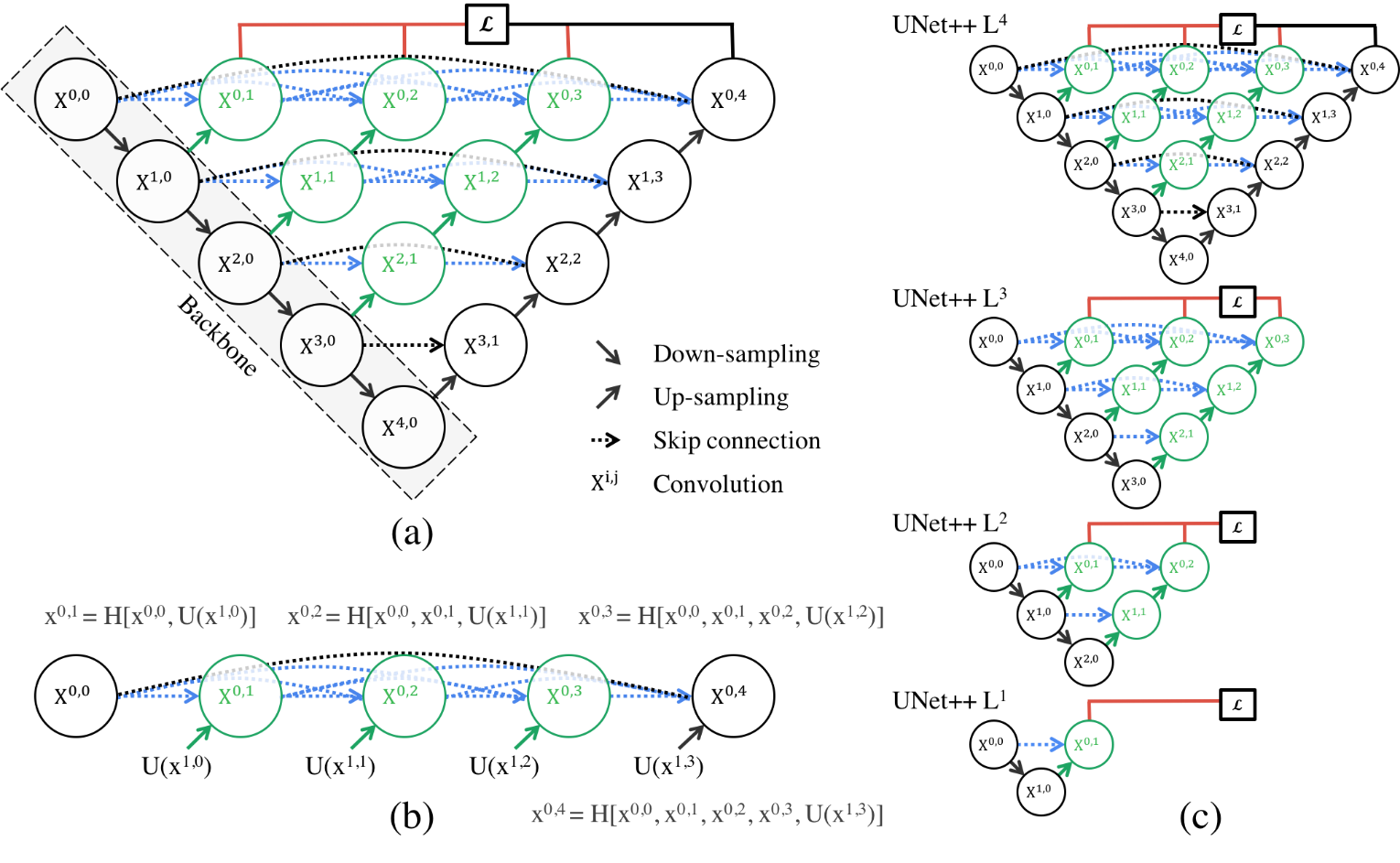


Рисунок - 11 – Архитектура «U-NET++»

Обучение моделей происходило всего в 10 эпох. Для определения качества моделей были использованы «DiceLoos» и «Intersection over Union» метрики. При обучении моделей менялись такие параметры как:

* back bone;
* encoder weights.

«Back bone» является основной частью сети так как она принимает изображение и делает извлечение признаков с разных уровней (receptive fields). Хороший «back bone» может значительно улучшить работу сети, помогая ей выявлять сложные зависимости в данных и делать более точные прогнозы.

Модели брались из библиотеки «segmentation\_models.pytorch», которая имеет 9 моделей архитектур для бинарной и мультиклассовой сегментации. Имеется 124 доступных энкодера. Также можно выделить что все кодировщики имеют предварительно обученные веса для быстрой и лучше сходимости. В таблице 1 представлены энкодеры и веса, которые можно использовать для обучения моделей.

Таблица 1 – Таблица энкодеров с весами

|  |  |  |
| --- | --- | --- |
| Encoder | Weights | Params, M |
| resnext50\_32x4d | imagenet/ssl/swsl | 22M |
| resnext101\_32x4d | ssl/swsl | 42M |
| resnext101\_32x8d | imagenet/instagram/ssl/swsl | 86M |
| resnext101\_32x16d | instagram/ssl/swsl | 191M |
| resnext101\_32x32d | instagram | 466M |
| resnext101\_32x48d | instagram | 826M |

Наилучший результат продемонстрировала модель с «back bone» «resnext50-32x4d», достигнув точности сегментации 69%. Архитектура ResNeXt относится к остаточным сетям (ResNet) и обладает более высокой способностью к обобщению по сравнению со стандартными сверточными сетями благодаря применению группированных свертык. Параметр 32x4d означает, что в этой архитектуре использовались 32 группы с 4 путями в каждой.

С другой стороны, худший результат с точностью 57% показала модель с «back bone» "resnext101\_32x16d". Эта архитектура является более глубокой версией ResNeXt, состоящей из 101 слоя. Однако увеличение глубины сети не всегда гарантирует лучшие результаты, особенно при ограниченных объемах данных для обучения.

Анализируя результаты, можно сделать вывод, что выбор подходящей базовой архитектуры имеет критическое значение для достижения высокой точности в задачах сегментации изображений с использованием U-Net. Более легкие и компактные архитектуры, такие как resnext50-32x4d, могут обеспечивать лучшие результаты по сравнению с более глубокими и сложными моделями, особенно при недостатке больших объемов данных для обучения.

Таблица 2 – Результаты обучения моделей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Back bone | Encoder weights | Model | Кол-во эпох | Dice loss | Iou score |
| resnext50-32x4d | imagenet | U-NET | 10 | 0.24 | 0.69 |
| resnext50-32x4d | imagenet | U-NET++ | 10 | 0.27 | 0.67 |
| resnext101\_32x16d | instagram | U-NET | 10 | 0.77 | 0.57 |
| resnext101\_32x16d | instagram | U-NET++ | 10 | 0.32 | 0.61 |
| resnext101\_32x8d | imagenet | U-NET | 10 | 0.33 | 0.63 |
| resnext101\_32x8d | imagenet | U-NET | 5 | 0.33 | 0.59 |
| resnext101\_32x8d | imagenet | U-NET++ | 10 | 0.33 | 0.59 |

На рисунке 12 можно увидеть эпохи обучения лучше модели по результату.

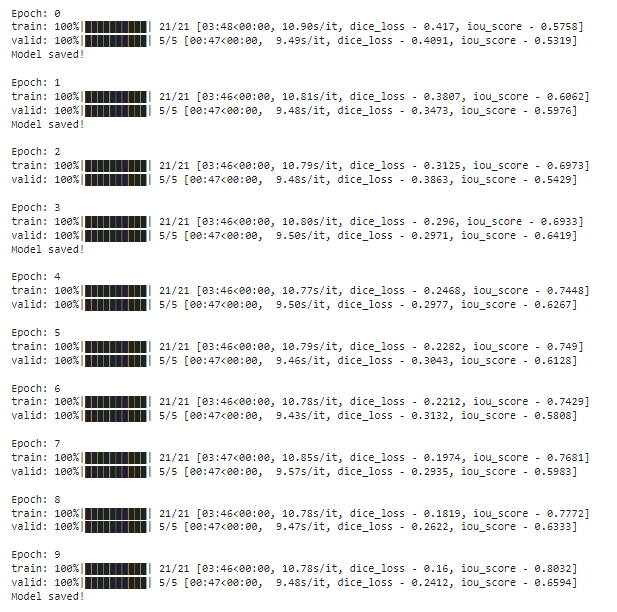


Рисунок - 12 – Эпохи обучения модели

Результаты работы обученной модели можно увидеть на рисунках 13, 14, 15. Код для обучения модели имеется в приложении А.

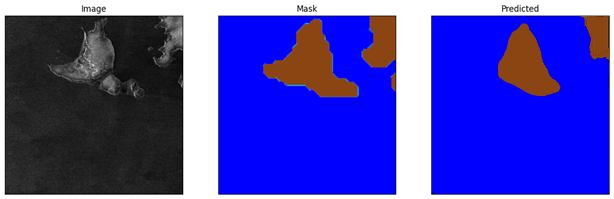


Рисунок - 13 – Результат в визуальном формате 1



Рисунок - 14 – Результат в визуальном формате 2



Рисунок - 15 – Результат в визуальном формате 3

1. Интеграция модели в приложение

Реализация «Desktop приложения» производилась с помощью программной библиотеки «PyQt». Она позволяет создавать кроссплатформенные приложения с использованием графического фреймворка «Qt». «PyQt» обеспечивает доступ к функционалу «Qt», позволяет создавать интерактивные и красивые пользовательские интерфейсы, а также обладает мощными возможностями для работы с событиями, графикой, мультимедиа.

С помощью инструмента «Qt designer» создадим форму с помощью, которой пользователь сможет загружать изображения радиолокационных приложений и получать маски с помощью обученной модели. Форма состоит из двух кнопок «Загрузить изображение» и «Создать маску», которые позволяют взаимодействовать с приложением. Также имеется два объекта «label», которые будут выводить изображения для пользователя. Как выглядит форма можно увидеть на рисунке 16. В приложении Б можно увидеть код для реализации формы.

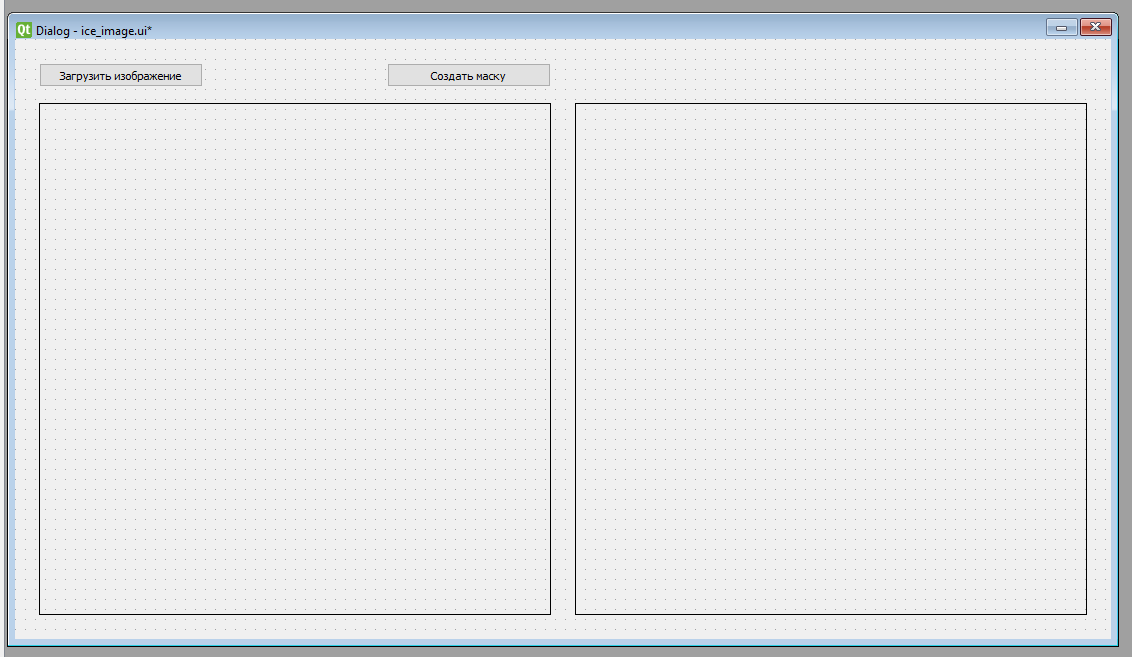


Рисунок - 16 – Спроектированная форма

Чтобы реализовать обработку изображения в маску с помощью обученной модели были созданы классы внутри приложения, которые описаны ниже.

Класс «model handler» необходим для инициализации модели внутри приложения. Также данный класс содержит метод «predict», который позволяет получить предсказание в виде маски. В качестве входного параметра метод получает тензор, а на выходе получает маску. Реализацию класса можно увидеть в листинге 3.

Листинг 3 – Model handler

import torch.nn as nn

from segmentation\_models\_pytorch import Unet

class model\_handler:

RESIZE\_SIZE = 448

def \_\_init\_\_(self):

# Определение функции потерь

criterion = nn.CrossEntropyLoss()

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

ENCODER = 'resnext50\_32x4d'

ENCODER\_WEIGHTS = 'imagenet'

CLASSES = ['high-concentration', 'ground', 'low-concentration', "water"]

ACTIVATION = 'softmax2d' # could be None for logits or 'softmax2d' for multiclass segmentation

# Определение архитектуры U-Net

self.model = Unet(encoder\_name=ENCODER,

encoder\_weights=ENCODER\_WEIGHTS,

classes=len(CLASSES),

activation=ACTIVATION).to(device)

self.model = torch.load('./resnext50\_32x4d.pth', map\_location=torch.device('cpu'))

def predict(self, input):

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

x\_tensor = torch.from\_numpy(input).to(device).unsqueeze(0)

pr\_mask = self.model.predict(x\_tensor)

pr\_mask = pr\_mask.squeeze().cpu().numpy()

for x in range(self.RESIZE\_SIZE):

for y in range(self.RESIZE\_SIZE):

maxx = pr\_mask[0, x, y]

maxq = 0

for q in range(4):

if (pr\_mask[q, x, y] >= maxx):

maxx = pr\_mask[q, x, y]

maxq = q

for q in range(4):

if (q == maxq):

Окончание листинга 3

pr\_mask[q, x, y] = 1

continue

pr\_mask[q, x, y] = 0

return pr\_mask

Класс «preprocessor» позволяет пред обрабатывать входное изображение для получения тензора, который уже подается в модель для получения необходимой маски. Реализацию класса можно увидеть в листинге 4.

Листинг 4 – Preprocessor

import os

from PIL import Image

import numpy as np

import segmentation\_models\_pytorch as smp

import albumentations as albu

class image\_preprocessor:

RESIZE\_SIZE = 448

ENCODER = 'resnext50\_32x4d'

ENCODER\_WEIGHTS = 'imagenet'

def preprocess(self, path):

image = Image.open(path).convert("RGB")

image = image.resize((self.RESIZE\_SIZE, self.RESIZE\_SIZE))

image = np.array(image)

preprocessing\_fn = smp.encoders.get\_preprocessing\_fn(self.ENCODER, self.ENCODER\_WEIGHTS)

if preprocessing\_fn:

sample = self.get\_preprocessing(preprocessing\_fn)(image=image)

image = sample['image']

return image

def get\_preprocessing(self, pf):

"""Construct preprocessing transform

Args:

preprocessing\_fn (callbale): data normalization function

(can be specific for each pretrained neural network)

Return:

transform: albumentations.Compose

"""

\_transform = [

albu.Lambda(image=pf),

albu.Lambda(image=self.to\_tensor),

]

return albu.Compose(\_transform)

def to\_tensor(self, x, \*\*kwargs):

return x.transpose(2, 0, 1).astype('float32')

Класс «image convertor» предназначен для конвертации полученной маски в удобный формат для отображения классов сплоченности льда. На листинге 5 можно увидеть представленный класс.

Эти классы взаимодействуют друг с другом, обеспечивая полную цепочку обработки изображения в маску с помощью обученной модели. Например, в листинге 6 можно увидеть событие, которое создается при нажатии на кнопку «Создать маску», которое использует все три класса для создания маски и отображения её в форме. Код этих классов можно легко расширить или изменить в соответствии с потребностями конкретного проекта.

Листинг 5 – Image convertor

import numpy as np

from PIL import Image

class image\_convertor:

RESIZE\_SIZE = 448

color\_palette = [

[0, 191, 255], # Класс 0 - Голубой (высокая сплоченность льда)

[139, 69, 19], # Класс 1 - Коричневый (земля)

[65, 105, 225], # Класс 2 - Светло-голубой (низкая сплоченность льда)

[0, 0, 255], # Класс 3 - Синий (вода)

[0, 0, 0] # Класс 4 - НЕИЗВЕСТНО

]

def mask\_to\_image(self, mask):

img = np.zeros((self.RESIZE\_SIZE, self.RESIZE\_SIZE, 3), dtype=np.uint8)

for i in range(self.RESIZE\_SIZE):

for j in range(self.RESIZE\_SIZE):

found = False

for q in range(4):

if (mask[q, i, j] == 1):

img[i, j] = self.color\_palette[q]

found = True

break

if (not found):

img[i, j] = self.color\_palette[4]

ouput = Image.fromarray(img)

return ouput

Результат работы приложения можно увидеть на рисунке 17.

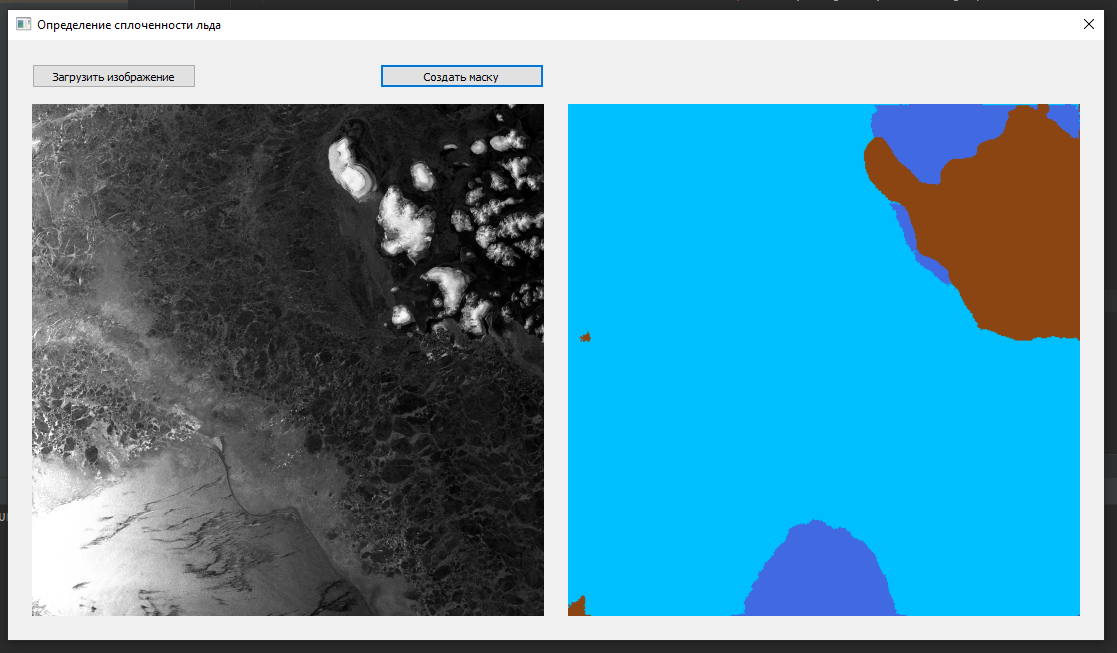


Рисунок - 17 – Рабочее приложение

Листинг 6 – Create\_mask\_m

def create\_mask\_m(self):

ip = image\_preprocessor()

image = ip.preprocess(self.path)

mh = model\_handler()

pr\_mask = mh.predict(image)

ic = image\_convertor()

mask = ic.mask\_to\_image(pr\_mask)

mask.save('mask.png')

pixmapImagen = QtGui.QPixmap("mask.png").scaled(512,

512,

QtCore.Qt.KeepAspectRatio,

QtCore.Qt.SmoothTransformation)

self.mask.setPixmap(pixmapImagen)

ЗАКЛЮЧЕНИЕ

В результате реализации проекта была исследована предметная область, создан набор данных, который включает в себя набор изображения и масок в 241 единицу. Также были обучены несколько нейронных сетей с разными гиперпараметрами на созданном наборе данных, которые сегментируют изображения на определённые классы. Был проведен анализ полученных моделей и интегрирована лучшая из них в «desktop приложение». Для улучшения результата можно провести аугментацию набора данных, попробовать использовать другие модели с разными гиперпараметрами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ольховик Е. А., Афонин А. Б., Тезиков А. Л. МЕТОДИКА ИССЛЕДОВАНИЯ ЛЕДОВОЙ ОБСТАНОВКИ НА СЕВЕРНОМ МОРСКОМ ПРОСТРАНСТВЕ. – 2019.
2. Гриняк В.М., Акмайкин Д.А., Иваненко Ю.С. Планирование маршрута судна с учетом ледовой обстановки. – 2022.
3. Донцов А. А., Суторихин И. А. Применение данных космических аппаратов Sentinel-1 и Sentinel-2 для мониторинга ледовой обстановки рек и водохранилищ // Материалы XI Всероссийской конференции с международным участием «Современные проблемы оптики. – 2021. – С. 171.
4. Sentinel Technologies [Электронный ресурс]. — Режим доступа: https://www.sentinel.com/ (15.04.2024) – Загл. с экрана.
5. Ван Ю.Р., Ли Х.М. Данные о морском ледовом покрове Арктики с помощью космического радара с синтезированной апертурой с помощью глубокого обучения //Earth System Science Data. – 2021. – Т. 13. – Нет. 6. – С. С. 2723-2742.
6. Лю Х., Хуан В., Махдианпари М. Зондирование морского льда в восточной части Арктики: первые результаты данных миссии группировки RADARSAT //Дистанционное зондирование. – 2022. – Т. 14. – Нет. 5. – С. 1165.
7. Arctic Sea Ice Image Masking [Электронный ресурс]. — Режим доступа: https://www.kaggle.com/datasets/alexandersylvester/arctic-sea-ice-image-masking (15.04.2024) – Загл. с экрана.
8. Canadian Ice Service [Электронный ресурс]. — Режим доступа: https://www.canada.ca/en/environment-climate-change/services/ice-forecasts-observations/latest-conditions.html (15.04.2024) – Загл. с экрана.
9. Арктический и антарктический научно-исследователь­ский институт (ААНИИ) [Электронный ресурс] – Режим доступа: http://old.aari.ru (дата обращения: 15.04.2024)
10. Sea Ice Index, Version 3 [Электронный ресурс] – Режим доступа: [https://nsidc.org/data/g02135/versions/3 (дата обращения: 15.04.2024)](https://nsidc.org/data/g02135/versions/3%20(дата%20обращения:%2015.04.2024))
11. ASF Data search [Электронный ресурс] – Режим доступа: [https://search.asf.alaska.edu/#/ (дата обращения:15.02.2024)](https://www.minstroyrf.gov.ru/docs/117294/)
12. QGIS [Электронный ресурс] – Режим доступа: [https://qgis.org/en/site/ (дата обращения:15.04.2024)](https://qgis.org/en/site/%20(дата%20обращения:15.04.2024))
13. SNAP [Электронный ресурс] – Режим доступа: https://step.esa.int/main/download/snap-download/ [(дата обращения:15.04.2024)](https://www.minstroyrf.gov.ru/docs/117294/)
14. Roboflow [Электронный ресурс] – Режим доступа: https://roboflow.com/ [(дата обращения:15.04.2024)](https://www.minstroyrf.gov.ru/docs/117294/)

Приложение А  
(обязательное)  
Программный код для обучения моделей

# -\*- coding: utf-8 -\*-

"""Ice segmentation baseline.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1XSm6GKf3eKRT0rShLADA5hpzzkY\_zWnk

"""

!pip install segmentation-models-pytorch

!pip install torchinfo

# Data handling

import pandas as pd

import numpy as np

# Data visualization

import matplotlib.pyplot as plt

from PIL import Image

import cv2

# Preprocessing

from sklearn.model\_selection import train\_test\_split as tts

from segmentation\_models\_pytorch.encoders import get\_preprocessing\_fn

# Torch

import torch

from torch import nn, optim

from torch.utils.data import Dataset, DataLoader

from torchinfo import summary

from torchvision import transforms

import segmentation\_models\_pytorch as smp

from segmentation\_models\_pytorch import Unet

# Metrics

from segmentation\_models\_pytorch.metrics import get\_stats, iou\_score

# os

import os

# Path

from pathlib import Path

# tqdm

from tqdm.auto import tqdm

# warnings

import warnings

warnings.filterwarnings("ignore")

import os

import cv2

import random

import matplotlib.pyplot as plt

# Путь к папке с масками и изображениями

dataset\_folder = "/content/drive/MyDrive/crop\_dataset\_fixed/"

images\_folder = os.path.join(dataset\_folder, "cropped\_images")

masks\_folder = os.path.join(dataset\_folder, "cropped\_masks")

# Получаем список файлов изображений

image\_files = os.listdir(images\_folder)

# Выбираем три случайных изображения

random\_images = random.sample(image\_files, 3)

# Показываем каждую пару изображение-маска

for image\_file in random\_images:

# Путь к изображению

image\_path = os.path.join(images\_folder, image\_file)

# Прочитаем изображение с помощью OpenCV

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # Переводим цветовую схему для корректного отображения в matplotlib

# предполагается, что маска имеет тот же формат, что и изображение

mask\_path = os.path.join(masks\_folder, image\_file)

# Прочитаем маску с помощью OpenCV

mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

# Выведем изображение и маску

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Изображение

axes[0].imshow(image)

axes[0].set\_title('Image')

axes[0].axis('off')

# Маска

axes[1].imshow(mask, cmap='gray')

axes[1].set\_title('Mask')

axes[1].axis('off')

plt.show()

import os

import random

import shutil

dataset\_folder = "/content/drive/MyDrive/crop\_dataset\_fixed/"

train\_folder = "train\_data"

val\_folder = "val\_data"

test\_folder = "test\_data"

os.makedirs(train\_folder, exist\_ok=True)

os.makedirs(train\_folder+"/masks", exist\_ok=True)

os.makedirs(train\_folder+"/images", exist\_ok=True)

os.makedirs(val\_folder, exist\_ok=True)

os.makedirs(val\_folder+"/masks", exist\_ok=True)

os.makedirs(val\_folder+"/images", exist\_ok=True)

os.makedirs(test\_folder, exist\_ok=True)

os.makedirs(test\_folder+"/masks", exist\_ok=True)

os.makedirs(test\_folder+"/images", exist\_ok=True)

image\_files = os.listdir(images\_folder)

random.shuffle(image\_files)

train\_size = int(0.7 \* len(image\_files))

val\_size = int(0.15 \* len(image\_files))

test\_size = len(image\_files) - train\_size - val\_size

train\_images = image\_files[:train\_size]

val\_images = image\_files[train\_size:train\_size+val\_size]

test\_images = image\_files[train\_size+val\_size:]

for image\_file in train\_images:

fixed\_name = image\_file.replace(".jpg","").replace(".","") +".jpg"

shutil.copy(os.path.join(images\_folder, image\_file), os.path.join(train\_folder+"/images", fixed\_name))

shutil.copy(os.path.join(masks\_folder, image\_file), os.path.join(train\_folder+"/masks", fixed\_name.replace(".jpg", "\_mask.jpg")))

for image\_file in val\_images:

fixed\_name = image\_file.replace(".jpg","").replace(".","") +".jpg"

shutil.copy(os.path.join(images\_folder, image\_file), os.path.join(val\_folder+"/images", fixed\_name))

shutil.copy(os.path.join(masks\_folder, image\_file), os.path.join(val\_folder+"/masks", fixed\_name.replace(".jpg", "\_mask.jpg")))

for image\_file in test\_images:

fixed\_name = image\_file.replace(".jpg","").replace(".","") +".jpg"

shutil.copy(os.path.join(images\_folder, image\_file), os.path.join(test\_folder+"/images", fixed\_name))

shutil.copy(os.path.join(masks\_folder, image\_file), os.path.join(test\_folder+"/masks", fixed\_name.replace(".jpg", "\_mask.jpg")))

import torch

from torch import nn, optim

from torch.utils.data import Dataset, DataLoader

from torchinfo import summary

from torchvision import transforms

import segmentation\_models\_pytorch as smp

from segmentation\_models\_pytorch import Unet

# Metrics

from segmentation\_models\_pytorch.metrics import get\_stats, iou\_score

SEED = 123

import os

from PIL import Image

from torch.utils.data import Dataset

from torch.utils.data import Dataset as BaseDataset

from torchvision.transforms.functional import resize

RESIZE\_SIZE=448

class CustomDataset(Dataset):

def \_\_init\_\_(self, root\_dir, image\_transforms:transforms, mask\_transforms:transforms):

self.root\_dir = root\_dir

self.image\_transforms = image\_transforms

self.mask\_transforms = mask\_transforms

self.image\_folder = os.path.join(root\_dir, "images")

self.mask\_folder = os.path.join(root\_dir, "masks")

self.image\_files = os.listdir(self.image\_folder)

self.mask\_files = os.listdir(self.mask\_folder)

def \_\_len\_\_(self):

return len(self.image\_files)

def \_\_getitem\_\_(self, idx):

img\_name = os.path.join(self.image\_folder, self.image\_files[idx])

image = Image.open(img\_name).convert("RGB")

image = image.Resize((RESIZE\_SIZE,RESIZE\_SIZE))

image = self.image\_transforms(image)

mask\_name = os.path.join(self.mask\_folder, self.mask\_files[idx])

mask = Image.open(mask\_name).convert("L") # Преобразование в черно-белый режим

mask = mask.Resize((RESIZE\_SIZE,RESIZE\_SIZE))

mask = self.mask\_transforms(mask)

for x in range(RESIZE\_SIZE):

for y in range(RESIZE\_SIZE):

if(mask[x,y]>4):

mask[x,y] = 0

else:

mask[x,y]-=1

return image, mask

class NewCustomDataset(BaseDataset):

"""CamVid Dataset. Read images, apply augmentation and preprocessing transformations.

Args:

images\_dir (str): path to images folder

masks\_dir (str): path to segmentation masks folder

class\_values (list): values of classes to extract from segmentation mask

augmentation (albumentations.Compose): data transfromation pipeline

(e.g. flip, scale, etc.)

preprocessing (albumentations.Compose): data preprocessing

(e.g. noralization, shape manipulation, etc.)

"""

CLASSES = ['high-concentration', 'ground', 'low-concentration',"water"]

def \_\_init\_\_(

self,

images\_dir,

masks\_dir,

classes=None,

augmentation=None,

preprocessing=None,

):

self.ids = os.listdir(images\_dir)

print(self.ids)

print(os.listdir(masks\_dir))

self.images\_fps = [os.path.join(images\_dir, image\_id) for image\_id in self.ids]

self.masks\_fps = [os.path.join(masks\_dir, image\_id) for image\_id in self.ids]

# convert str names to class values on masks

self.class\_values = [self.CLASSES.index(cls.lower()) for cls in classes]

self.augmentation = augmentation

self.preprocessing = preprocessing

def \_\_getitem\_\_(self, i):

image = Image.open(self.images\_fps[i]).convert("RGB")

image = image.resize((RESIZE\_SIZE,RESIZE\_SIZE))

image = np.array(image)

mask = Image.open(self.masks\_fps[i].replace(".jpg", "\_mask.jpg")).convert("L") # Преобразование в черно-белый режим

mask = mask.resize((RESIZE\_SIZE,RESIZE\_SIZE))

mask = np.array(mask)

for x in range(RESIZE\_SIZE):

for y in range(RESIZE\_SIZE):

if(mask[x,y]>4):

mask[x,y] = 0

else:

mask[x,y]-=1

# extract certain classes from mask (e.g. cars)

masks = [(mask == v) for v in self.class\_values]

mask = np.stack(masks, axis=-1).astype('float')

# apply augmentations

if self.augmentation:

sample = self.augmentation(image=image, mask=mask)

image, mask = sample['image'], sample['mask']

# apply preprocessing

if self.preprocessing:

sample = self.preprocessing(image=image, mask=mask)

image, mask = sample['image'], sample['mask']

return image, mask

def \_\_len\_\_(self):

return len(self.ids)

import albumentations as albu

def get\_training\_augmentation():

train\_transform = [

albu.HorizontalFlip(p=0.5),

albu.ShiftScaleRotate(scale\_limit=0.5, rotate\_limit=0, shift\_limit=0.1, p=1, border\_mode=0),

albu.PadIfNeeded(min\_height=320, min\_width=320, always\_apply=True, border\_mode=0),

albu.RandomCrop(height=320, width=320, always\_apply=True),

albu.IAAAdditiveGaussianNoise(p=0.2),

albu.IAAPerspective(p=0.5),

albu.OneOf(

[

albu.CLAHE(p=1),

albu.RandomBrightness(p=1),

albu.RandomGamma(p=1),

],

p=0.9,

),

albu.OneOf(

[

albu.IAASharpen(p=1),

albu.Blur(blur\_limit=3, p=1),

albu.MotionBlur(blur\_limit=3, p=1),

],

p=0.9,

),

albu.OneOf(

[

albu.RandomContrast(p=1),

albu.HueSaturationValue(p=1),

],

p=0.9,

),

]

return albu.Compose(train\_transform)

def get\_validation\_augmentation():

"""Add paddings to make image shape divisible by 32"""

test\_transform = [

albu.PadIfNeeded(384, 480)

]

return albu.Compose(test\_transform)

def to\_tensor(x, \*\*kwargs):

return x.transpose(2, 0, 1).astype('float32')

def get\_preprocessing(preprocessing\_fn):

"""Construct preprocessing transform

Args:

preprocessing\_fn (callbale): data normalization function

(can be specific for each pretrained neural network)

Return:

transform: albumentations.Compose

"""

\_transform = [

albu.Lambda(image=preprocessing\_fn),

albu.Lambda(image=to\_tensor, mask=to\_tensor),

]

return albu.Compose(\_transform)

def visualize(\*\*images):

"""PLot images in one row."""

n = len(images)

plt.figure(figsize=(16, 5))

for i, (name, image) in enumerate(images.items()):

plt.subplot(1, n, i + 1)

plt.xticks([])

plt.yticks([])

plt.title(' '.join(name.split('\_')).title())

plt.imshow(image)

plt.show()

x\_train\_dir = "train\_data/images/"

y\_train\_dir = "train\_data/masks/"

x\_val\_dir = "train\_data/images/"

y\_val\_dir = "train\_data/masks/"

CLASSES = ['high-concentration', 'ground', 'low-concentration',"water"]

dataset = NewCustomDataset(x\_train\_dir, y\_train\_dir, classes=['high-concentration', 'ground', 'low-concentration',"water"])

image, mask = dataset[10] # get some sample

print(mask.shape)

visualize(

image=image,

ice\_mask=mask.squeeze(),

)

import gc

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision.transforms import Compose, ToTensor

import torchvision.transforms as transforms

from tqdm import tqdm

# Определение функции потерь

criterion = nn.CrossEntropyLoss()

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

ENCODER = 'resnext101\_32x8d'

ENCODER\_WEIGHTS = 'imagenet'

CLASSES = ['high-concentration', 'ground', 'low-concentration',"water"]

ACTIVATION = 'softmax2d' # could be None for logits or 'softmax2d' for multiclass segmentation

# Определение архитектуры U-Net

model = Unet( encoder\_name=ENCODER,

encoder\_weights=ENCODER\_WEIGHTS,

classes=len(CLASSES),

activation=ACTIVATION,)

preprocessing\_fn = smp.encoders.get\_preprocessing\_fn(ENCODER, ENCODER\_WEIGHTS)

x\_train\_dir = "train\_data/images/"

y\_train\_dir = "train\_data/masks/"

x\_val\_dir = "val\_data/images/"

y\_val\_dir = "val\_data/masks/"

CLASSES = ['high-concentration', 'ground', 'low-concentration',"water"]

train\_dataset = NewCustomDataset(

x\_train\_dir,

y\_train\_dir,

preprocessing=get\_preprocessing(preprocessing\_fn),

classes=CLASSES,

)

val\_dataset = NewCustomDataset(

x\_val\_dir,

y\_val\_dir,

preprocessing=get\_preprocessing(preprocessing\_fn),

classes=CLASSES,

)

BATCH\_SIZE = 8

NUM\_WORKERS = os.cpu\_count()

train\_loader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=BATCH\_SIZE, shuffle=False)

#for param in model.encoder.parameters():

# param.requires\_grad = False

#summary(model = model,

# input\_size = [8, 3, 896, 896],

# col\_width = 15,

# col\_names = ['input\_size', 'output\_size', 'num\_params', 'trainable'],

# row\_settings = ['var\_names'])

import segmentation\_models\_pytorch.utils

loss = smp.utils.losses.DiceLoss()

metrics = [

smp.utils.metrics.IoU(threshold=0.5),

]

optimizer = torch.optim.Adam([

dict(params=model.parameters(), lr=0.0001),

])

train\_epoch = smp.utils.train.TrainEpoch(

model,

loss=loss,

metrics=metrics,

optimizer=optimizer,

device=device,

verbose=True,

)

valid\_epoch = smp.utils.train.ValidEpoch(

model,

loss=loss,

metrics=metrics,

device=device,

verbose=True,

)

import gc

max\_score = 0

for i in range(0, 5):

gc.collect()

print('\nEpoch: {}'.format(i))

train\_logs = train\_epoch.run(train\_loader)

valid\_logs = valid\_epoch.run(val\_loader)

# do something (save model, change lr, etc.)

if max\_score < valid\_logs['iou\_score']:

max\_score = valid\_logs['iou\_score']

torch.save(model, './best\_model.pth')

print('Model saved!')

if i == 25:

optimizer.param\_groups[0]['lr'] = 1e-5

print('Decrease decoder learning rate to 1e-5!')

best\_model = torch.load('./best\_model.pth')

x\_test\_dir = "test\_data/images/"

y\_test\_dir = "test\_data/masks/"

test\_dataset = NewCustomDataset(

x\_test\_dir,

y\_test\_dir,

preprocessing=get\_preprocessing(preprocessing\_fn),

classes=CLASSES,

)

test\_dataloader = DataLoader(test\_dataset)

test\_epoch = smp.utils.train.ValidEpoch(

model=best\_model,

loss=loss,

metrics=metrics,

device=device,

)

logs = test\_epoch.run(test\_dataloader)

logs

test\_dataset\_vis = NewCustomDataset(

x\_test\_dir, y\_test\_dir,

classes=CLASSES,

)

for i in range(10):

n = np.random.choice(len(test\_dataset))

image\_vis = test\_dataset\_vis[n][0].astype('uint8')

image, gt\_mask = test\_dataset[n]

gt\_mask = gt\_mask.squeeze().transpose(1, 2, 0)

x\_tensor = torch.from\_numpy(image).to(device).unsqueeze(0)

pr\_mask = best\_model.predict(x\_tensor)

pr\_mask = (pr\_mask.squeeze().cpu().numpy().round()).transpose(1, 2, 0)

print(gt\_mask)

visualize(

image=image\_vis,

mask=gt\_mask,

predicted = pr\_mask

)

def mask\_to\_image(mask, color\_palette):

img = np.zeros((RESIZE\_SIZE, RESIZE\_SIZE, 3), dtype=np.uint8)

for i in range(RESIZE\_SIZE):

for j in range(RESIZE\_SIZE):

found = False

for q in range(4):

if(mask[i,j,q]==1):

img[i,j] = color\_palette[q]

found = True

break

if(not found):

img[i,j] = color\_palette[4]

return img

color\_palette = [

[0, 191, 255], # Класс 0 - Голубой (высокая сплоченность льда)

[139, 69, 19], # Класс 1 - Коричневый (земля)

[65, 105, 225], # Класс 2 - Светло-голубой (низкая сплоченность льда)

[0, 0, 255], # Класс 3 - Синий (вода)

[0,0,0] #Класс 4 - НЕИЗВЕСТНО

]

for i in range(10):

n = np.random.choice(len(test\_dataset))

image\_vis = test\_dataset\_vis[n][0].astype('uint8')

image, gt\_mask = test\_dataset[n]

gt\_mask = gt\_mask.squeeze().transpose(1, 2, 0)

target\_mask\_image = mask\_to\_image(gt\_mask, color\_palette)

x\_tensor = torch.from\_numpy(image).to(device).unsqueeze(0)

pr\_mask = best\_model.predict(x\_tensor)

pr\_mask = (pr\_mask.squeeze().cpu().numpy().round()).transpose(1, 2, 0)

predicted\_mask\_image = mask\_to\_image(pr\_mask, color\_palette)

visualize(

image=image\_vis,

mask=target\_mask\_image,

predicted = predicted\_mask\_image

)

Приложение б  
(обязательное)  
Программный код «desktop приложения»

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'ice\_image.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

import preprocessor

from preprocessor import image\_preprocessor

from model\_handler import model\_handler

from image\_convertor import image\_convertor

from PIL import Image

import sys

from os import getcwd

class QLabelClickable(QtWidgets.QLabel):

clicked = QtCore.pyqtSignal()

def \_\_init\_\_(self, parent=None):

super(QLabelClickable, self).\_\_init\_\_(parent)

def mousePressEvent(self, event):

self.clicked.emit()

class Ui\_Dialog(QtWidgets.QDialog):

path = ""

def \_\_init\_\_(self, parent=None):

super(Ui\_Dialog, self).\_\_init\_\_(parent)

self.setWindowTitle("Определение сплоченности льда")

self.setWindowFlags(QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.MSWindowsFixedSizeDialogHint)

self.setFixedSize(1096, 600)

self.path = ""

self.initUI()

def initUI(self):

self.add\_image = QtWidgets.QPushButton("Загрузить изображение", self)

self.add\_image.setGeometry(QtCore.QRect(24, 24, 164, 24))

self.add\_image.setObjectName("add\_image")

self.add\_image.clicked.connect(self.seleccionarImagen)

self.create\_mask = QtWidgets.QPushButton("Создать маску", self)

self.create\_mask.setGeometry(QtCore.QRect(372, 24, 164, 24))

self.create\_mask.setObjectName("create\_mask")

self.create\_mask.clicked.connect(self.create\_mask\_m)

self.image = QtWidgets.QLabel(self)

self.image.setGeometry(QtCore.QRect(24, 64, 512, 512))

self.image.setObjectName("image")

self.mask = QtWidgets.QLabel(self)

self.mask.setGeometry(QtCore.QRect(560, 64, 512, 512))

self.mask.setObjectName("mask")

def seleccionarImagen(self):

imagen, extension = QtWidgets.QFileDialog.getOpenFileName(self,

"Выбор картинки",

"D:\Dowlands\ICES\_P\READY\_IMAGES\images",

"Image (\*.png \*.jpg)",

options=QtWidgets.QFileDialog.Options())

if imagen:

pixmapImagen = QtGui.QPixmap(imagen).scaled(512,

512,

QtCore.Qt.KeepAspectRatio,

QtCore.Qt.SmoothTransformation)

self.image.setPixmap(pixmapImagen)

self.path = imagen

def create\_mask\_m(self):

ip = image\_preprocessor()

image = ip.preprocess(self.path)

mh = model\_handler()

pr\_mask = mh.predict(image)

ic = image\_convertor()

mask = ic.mask\_to\_image(pr\_mask)

mask.save('mask.png')

pixmapImagen = QtGui.QPixmap("mask.png").scaled(512,

512,

QtCore.Qt.KeepAspectRatio,

QtCore.Qt.SmoothTransformation)

self.mask.setPixmap(pixmapImagen)

if \_\_name\_\_ == "\_\_main\_\_":

app = QtWidgets.QApplication(sys.argv)

main = Ui\_Dialog()

main.show()

sys.exit(app.exec\_())