

ОГЛАВЛЕНИЕ

Нормативные ссылки	12
Определения, обозначения и сокращения	13
Введение	14
1 Анализ предметной области.....	17
1.1 Описание предметной области	17
1.1.1 Рыбный промысел	17
1.1.2 Компьютерное зрение	19
1.2 Существующие решения и исследования	26
1.3 Методы отслеживания рыбы	31
1.4 Формирование требований.....	33
2 Разработка методики и алгоритмов	35
2.1 Схема исследования	35
2.2 Основные программные инструменты.....	35
2.3 Метрики и критерии исследования	39
2.3.1 Метрики детектирования	40
2.3.2 Метрики отслеживания.....	44
2.4 Архитектуры нейронных сетей.....	46
2.4.1 R-CNN.....	48
2.4.2 SSD.....	50
2.4.3 YOLO	51
3 Разработка и тестирование системы	53
3.1 Проектирование системы	53
3.2 Набор данных и работа с ним	55
3.2.1 Сбор данных.....	55
3.2.2 Удаление дубликатов	59
3.2.3 Унифицирование.....	60
3.2.4 Аугментации	64
3.3 Разработка модели машинного обучения.....	70
3.4 Разработка приложения	74

3.5 Тестирование приложения.....	76
Заключение.....	77
Список использованных источников.....	79
Приложение А.....	91
Приложение Б	95
Приложение В.....	101
Приложение Г	106

НОРМАТИВНЫЕ ССЫЛКИ

В настоящем документе используются ссылки на следующие нормативные документы:

ГОСТ Р 56696-2015 Возобновляемые источники сырья. Аквакультура. Термины и определения

ГОСТ Р 24668-2022 Информационные технологии. Искусственный интеллект. Структура управления процессами для анализа больших данных (ИСО/МЭК 24668:2022)

ГОСТ Р 70462.1-2022 Информационные технологии. Интеллект искусственный. Оценка робастности нейронных сетей. Часть 1. Обзор (ISO/IEC TR 24029-1:2021)

ПНСТ 776-2022 Информационные технологии. Интеллект искусственный. Управление рисками

ГОСТ Р 59895-2021 Технологии искусственного интеллекта в образовании. Общие положения и терминология

ГОСТ Р 59898-2021 Оценка качества систем искусственного интеллекта. Общие положения

ГОСТ Р 59385-2021 Информационные технологии. Искусственный интеллект. Ситуационная видеоаналитика. Термины и определения

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящем текстовом документе применяются следующие определения, обозначения и сокращения:

ИИ (artificial intelligence, AI) – искусственный интеллект.

МО (machine learning, ML) – машинное обучение.

Компьютерное зрение (Computer Vision, CV) – теория и технология создания машин, которые могут производить обнаружение, отслеживание и классификацию объектов.

Свёрточная нейронная сеть (convolutional neural network, CNN) – специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание объектов. Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

ВНИРО – Всероссийский научно-исследовательский институт рыбного хозяйства и океанографии.

Детектирование объектов на изображении – это процесс автоматического определения и локализации различных объектов.

Сегментация – это процесс разделения цифрового изображения на несколько сегментов, которые выделяют границы объектов.

ВВЕДЕНИЕ

Одной из наиболее важных задач рыбного промысла является отслеживание рыбы и ее классификация в местах прохода. Данные о количестве и видовом распределении проходящей рыбы обладают многими возможностями использования. Начиная с основных – это отслеживание редких рыб, находящихся под угрозой исчезновения. И производственных потребностей – в превентивной адаптации рыбной промышленности для более качественной обработки рыбы. А также в научных целях – ведение статистики, анализ влияния различных факторов на количество проходящей рыбы. [1]

Способов отслеживания рыбы достаточно много, наиболее развитой технологией является акустическая телеметрия. Однако данная технология не всегда подходит для отслеживания рыбы, например из-за невозможности точно определить видовую принадлежность рыбного ресурса.

На данный момент в следствии анализа множества исследований по теме, можно предложить, что оптические методы анализа пространства начнут заменять акустические и лазерные. Особенно на близких расстояниях до 1.5 метров. Оптические методы в современном их виде могут выдавать точности даже при построении 3D моделей сравнимые с лазерными, наиболее точными. [2]. Однако важно заметить, что замена гидролокационных методов оптическими при использовании на большой удаленности или на глубине в ближайшее время не предвидится из-за малого количества света, проникающего под воду и общем снижении световой проницаемости воды.

Однако, стоит отметить, что с развитием технологий машинного обучения оптические системы отслеживания рыбы получили дополнительное развитие. Большое количество исследований в этой области подтверждают данное предположение. [3].

На данный момент отслеживание рыбных ресурсов скорее носит скорее косвенный характер и анализируется на основе количества выловленной рыбы. [4]. Использование оптического метода контроля подразумевает большую

возможную точность анализа количества и вида проходящей рыбы, без прямого на нее влияния.

Наиболее перспективными технологиями в этой сфере можно назвать сверточные нейронные сети. Данные модели искусственного интеллекта могут в режиме реального времени, то есть чаще 30 раз в секунду, определить расположение рыбы, ее вид и даже ее контур. В совокупности с системами трекинга можно получить подсчет проходящей рыбы с точностью более 85%.[5]. А при использовании систем 3D реконструкции можно определить длину каждой проплывшей рыбы. При использовании данных с видеокамер и их уточнения с помощью сонаров, можно получить большую точность [6]

Исходя из указанных выше данных, можно сделать вывод, что сфера оптического отслеживания рыб на основе нейронных сетей активно развивается последние несколько лет. Однако, большинство проанализированных работ содержат данные о решении данной задачи в лабораторных условиях, в условиях рыбных ферм, или с валидацией результатов на крайне малом количестве данных. Также большинство сделанных исследований относятся в период с 2016 по 2022 год. Относительно современных исследований с использованием новых архитектур нейронных сетей и с использованием нескольких источников данных крайне мало. Возможно, это связано с количеством времени, необходимым на апробацию данных, рецензирование и публикацию.

Таким образом можно сделать вывод, что сфера оптического отслеживания рыбных ресурсов нуждается в новых разработках с использованием большого количества исходных данных для обучения нейронных сетей, с рассмотрением новых архитектур и с концентрированием на использование в реальных условиях замутненной и темной воды.

Цель выпускной квалификационной работы – разработать методику отслеживания, подсчёта и анализа рыбных ресурсов с использованием компьютерного зрения. Исследование и разработка будет проводиться в рамках договора с Северным филиалом федерального государственного бюджетного

научного учреждения «Всероссийский научно-исследовательский институт рыбного хозяйства и океанографии».

Задачи исследования:

- анализ предметной области;
- формирование методов и методологии исследования;
- разработка и тестирование системы отслеживания рыбы.

В качестве объекта исследования будут рассматриваться системы отслеживания рыбных ресурсов. Предмет исследования: системы трекинга рыбных ресурсов на основе компьютерного зрения.

Достигнуть результата планируется с помощью подбора исходных данных для обучения моделей детектирования, рассмотрение новых архитектур сверточных нейронных сетей и сравнение их работы. В результате работы необходимо получить конечную систему отслеживания рыбы по данным с видеокамеры в режиме реального времени.

Следует отметить, что в процессе разработки будет уделяться особое внимание возможности работы модели в затемненных и загрознённых условиях. При успешной апробации система будет введена в использование всероссийским научно-исследовательским институт рыбного хозяйства и океанографии.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание предметной области

1.1.1 Рыбный промысел

Рыба – это востребованный ресурс. Обработка данного ресурса тяжелый и трудоемкий процесс. С момента ловли рыбы до представления конечному потребителю рыба проходит через множество технологических операций. [7]

Рыба - водное позвоночное животное с непостоянной температурой тела, дышащее жабрами и имеющее плавники. Рыбы — это достаточно обширная группа живых существ. К ним относятся акулы и скаты, как представители хрящевых рыб, так и, например осетры, которые являются костными рыбами. В зоологии отдельно выделяется ихтиология – наука, посвященная всестороннему изучению рыб.

Одним из наиболее развитых в России институтов по изучению рыбы является Всероссийский научно-исследовательский институт рыбного хозяйства и океанографии [8].

Рыбный промысел является одной из ключевых сфер в экономике Архангельской области. Всего на территории региона находятся 19 предприятий, осуществляющих океанический судовой промысел. Архангельская область представляет 11% общего улова рыбы из Северного бассейна и 2% среди общероссийского показателя. Объем добычи на 2024 год составил порядка 85 тысяч тон. Основными объектами промысла в Белом море являются: сельдь, корюшка, навага, камбала и горбуша. Всего на рыбном промысле задействовано более 150 организаций. [9]

Одной из интересных статистик для данных организаций является миграция рыб. Миграция – это целесообразное перемещение рыбы на различные расстояния. Например: нерестовые, нагульные и зимовальные миграции. Некоторые миграции регулярные, от ежегодных до ежедневных. А некоторые причины миграций до сих пор не выяснены. Наиболее интересными

параметрами миграции являются конечные пункты, сроки, пути, скорости и объемы перемещения. [10]

В среднем скорость миграции рыб равна 1-3 длин тела в секунду. Данную скорость рыба можно поддерживать достаточно долго. При этом в большинстве случаев максимальная длина тела рыбы около 30 сантиметров в секунду, среди наиболее распространенных. Таким образом в рамках наших решений необходимо обеспечить возможность детектирования рыбы, движущейся со скоростью около 1 метр в секунду. [10]

Важным параметром для системы оптического отслеживания рыбы является цветовосприимчивость. В целом зрение рыбы похоже на человеческое, однако смещено в сторону коротких волн. [11] Таким образом рыбы хорошо видят синие, зеленые цвета и ультрафиолетовые волны. При этом красные и инфракрасные волны практически «черные» для рыбы. Таким образом можно сделать вывод, что использование инфракрасных подсветок для камер не будет сильно влиять на прохождение рыбы.

Однако результат нашей работы может также влиять фототаксис. Фототаксис – отношение рыбы к источнику света может быть как положительным, так и отрицательным. И даже может меняться в зависимости от стадии развития. Однако многие рыбы положительно относятся к небольшому освещению, возможно в связи с большим количеством кислорода в воде на малой глубине. Также существуют исследования, в ходе которых удалось ускорить рост рыб, облучая их зеленым цветом. Данный цвет имеет более короткую длину волны и хорошо проходит сквозь воду. Также хорошо влияние может наблюдаться при синем цвете. Однако также многие рыбы демонстрируют сложное отношение к свету. Некоторые могут пугаться, затормаживаться и избегать [12]. Пример проникновения света в глубь воды изображен на рисунке 1.1.

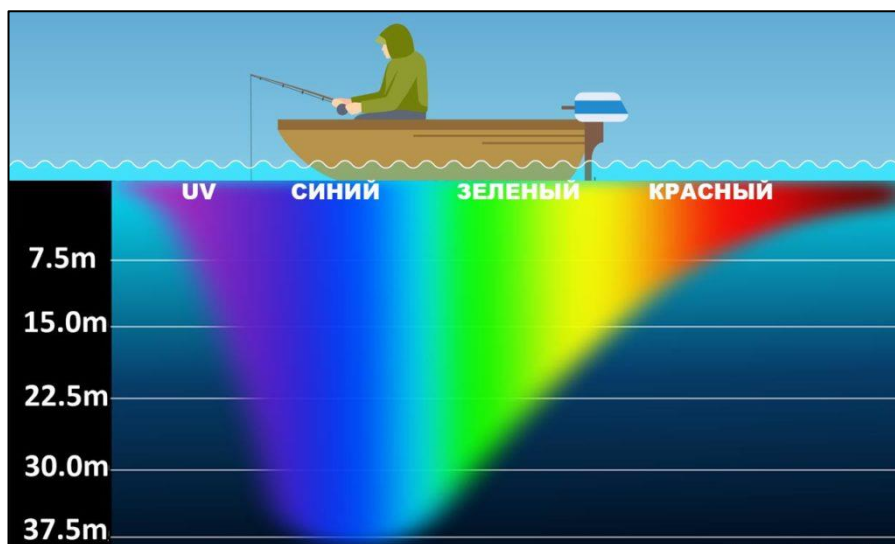


Рисунок 1.1 – проникаемость цвета по воду

Также важным параметром является фототропизм среди рыб заключается в ориентировании спиной к свету. Рыбы с нарушением вестибулярного аппарата могут переворачиваться. Прочие будут вставать под углом к свету или избегать данных участков. Таким образом можно сделать вывод, что освещение снизу рыб, в том числе отражающее дно светлым цветом может негативно сказаться на прохождении рыб. [10]

1.1.2 Компьютерное зрение

Компьютерное зрение или техническое зрение – это некоторые технологические системы позволяющие обнаруживать, отслеживать и классифицировать объекты. В общем случае можно обобщить данную сферу в трех задачах: получение данных, обработка данных, анализ данных. Отдельно в рамках компьютерного зрения можно выделить машинное зрение. Данный термин принято использовать в системах с инженерным уклоном, например на производстве.

Компьютерное зрение применяется во многих сферах жизнедеятельности человека. От достаточно простых – например классификации овощей и распознавание текста [13] до

узкоспециализированных задач в медицине, например детектирование пневмонии по радиограмме грудной клетки [14].

Наиболее ярко представленной технологией в сфере компьютерного зрения являются модели глубокого обучения. Точкой отсчета развития данных моделей принято считать 2012 год. В этом году на соревнованиях LSVRC с большим отрывом в задаче классификации картинок побеждает глубокая нейронная сеть [15]. В 2015 году на этих же соревнованиях модель искусственного интеллекта классифицирует 1000 классов картинок лучше человека. [16]. В 2021 году нейросеть может определять пол и возраст человека по следу от обуви лучше экспертов криминалистов. [17].

Развитие систем отслеживания рыбы началось практически сразу с активным внедрением нейронных сетей в другие сферы. Сильное влияние в этап зарождения оказали два проекта: Fish4Knowledge [18] и LifeCLEF Fish Task 2014-2015 [19]. Оба проекта представили наборы данных для автоматизации обработки, и анализа визуальных данных с целью изучения морской экосистемы и различных зависимостей, и эффектов.

Первые исследования, которые удалось найти в открытых источниках, относятся к 2016 году. [20]. В этом году наиболее развитой архитектурой нейронных сетей можно было считать R-CNN. Данная архитектура позволяла давать достаточно точные прогнозы и при этом на определенной мощности работать практически в режиме реального времени.

Второй этап развития сферы отслеживания рыбы можно определить к 2021-2022 году. В это время начали активно развиваться новые архитектуры нейронных сетей – YOLO и SSD. Данные архитектуры позволяли быстрее и с определенного этапа развития точнее детектировать объекты, в том числе рыб. В это же время начали внедряться новые подходы в трекинге объектов, например использование фильтра Калмана. [21]

В настоящее время можно отследить моду не на создание принципиально новых универсальных архитектур, а на адаптацию

существующих под различные задачи. В том числе с использованием новых инструментов из других сфер. Например, использование OpticalFlow [22].

Однако, учитывая все нововведения, общая суть остается неизменной. Для детектирования рыб необходимо обладать достаточно большим и качественным датасетом. Обучить на основе этого набора данных одну из распространённых архитектур нейросетей и по возможности доработать ее под конкретную задачу.

Для более глубокого понимания принципов работы нейронных сетей следует рассмотреть их виды на основе решаемых задач. В общем случае компьютерное зрение основанное на нейронных сетях решает шесть основных задач, в зависимости от ожидаемого результата:

- классификация - разделение изображений на группы по основному объекту в кадре;

- локализация - определение доминирующего объекта с помощью определения его центра или выделения объекта прямоугольником, может использовать в совокупности с классификацией;

- распознавание (детектирование, обнаружение) - поиск нескольких объектов на изображении с последующей классификацией;

- семантическая сегментация - классификация каждого пикселя изображения в зависимости от класса, которые он описывает, например: трава, люди, овцы.

- сегментация экземпляров — классификация каждого пикселя по принадлежности к классу и определенному экземпляру класса, например: первая овца, другая овца.

- определение ключевых точек (отслеживание позиции) — поиск определенных точек на изображении, например: сгиб локтя, центр зрачка. Общая схема и с примера приведенных задач изображена на рисунке 1.2.

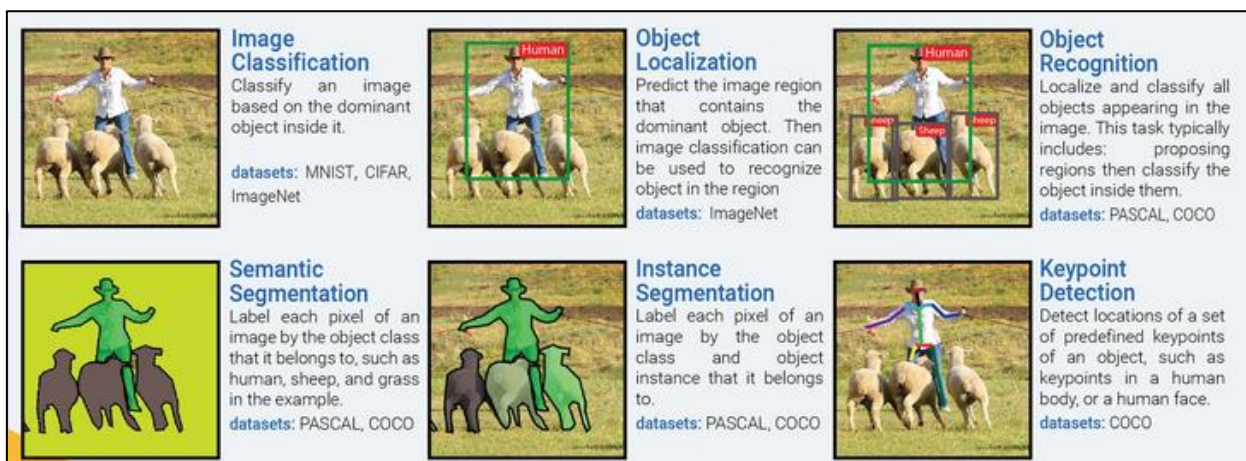


Рисунок 1.2 – Задачи глубоких сетей компьютерного зрения

В поставленной цели работы ключевая задача это распознаваний или как говорят – детектирование. Однако в зависимости от успешности первой задачи могут использоваться и другие задачи. Например, сегментация рыбы, для определения ее параметров: длины, ширины и т. д. Классификация как таковая уже будет использоваться для детектирования изображения, однако можно попробовать дополнительно классифицировать виды рыб.

На более глубоком уровне абстракции нейронные сети работают несколько сложнее. На примере сверточной нейронной сети для классификации представленных цифр. Нейронная сеть состоит из нейронов – перцептронов, разделенных на слои и соединенных в определенном порядке. На первый слой подается изображение, как таблица значений. Каждый перцептрон считывает значения из ячеек и в зависимости от их значений может либо активироваться, либо нет. Активированный нейрон передает информацию на следующие подключенную к нему нейроны. Такая операция происходит несколько раз. В конечном счете мы получаем 10 нейронов на выходе и нейрон с самым большим значением, скорее всего будет указывать на представленную на входе цифру.

В общем случае принято считать, что каждый слой отвечает за вытягивание определенного признака, первый, например ищет какой пиксель закрашен, а какой нет. Второй слой объединяет закрашенные пиксели в палочки. Далее эти палочки могут складываться в символы. Каждый

нейронной отвечает за свой признак, и его активация указывает на наличие этого признака на фото. Например, если у нас активировался нейрон, отвечающий за горизонтальную палочку сверху и нейрон диагональной палочки, то есть основания предполагать, что на изображении цифра 7. Пример простой нейронной сети изображен на рисунке 1.3.

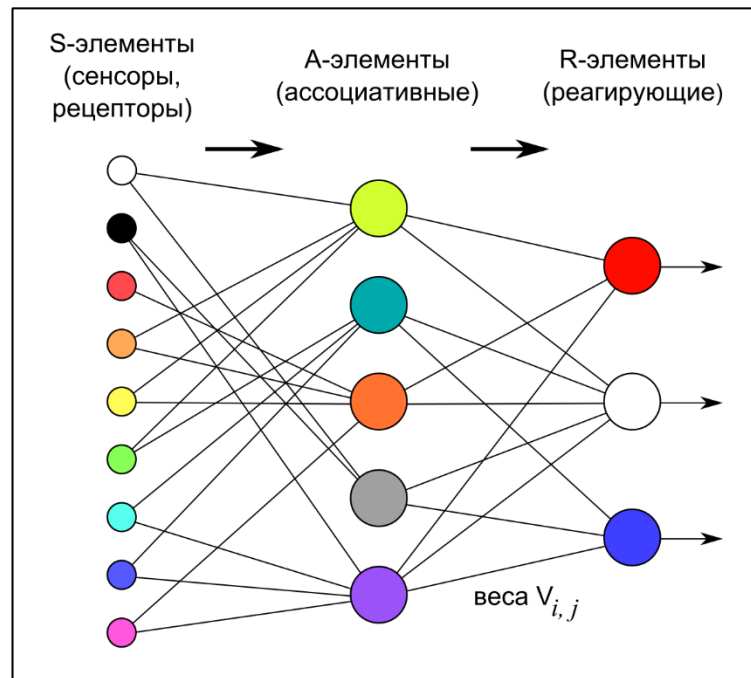


Рисунок 1.3 – Пример нейронной сети

Слои нейронных сетей зачастую типизированы. Основным слоем сверточных нейронных сетей является слой свертки. Свертка – это математическая операция над матрицами. По сути, представляет поэлементное умножение частей матрицы на некоторую другую матрицу. Сумма произведений будет значением ячейки новой матрицы. Матрица умножения называется ядром свертки. Данное ядро скользящим окном передвигается по исходной матрице. Принято считать, что свертка позволяет выявить признаки объектов. Пример процесса свертки отображен на рисунке 1.4.

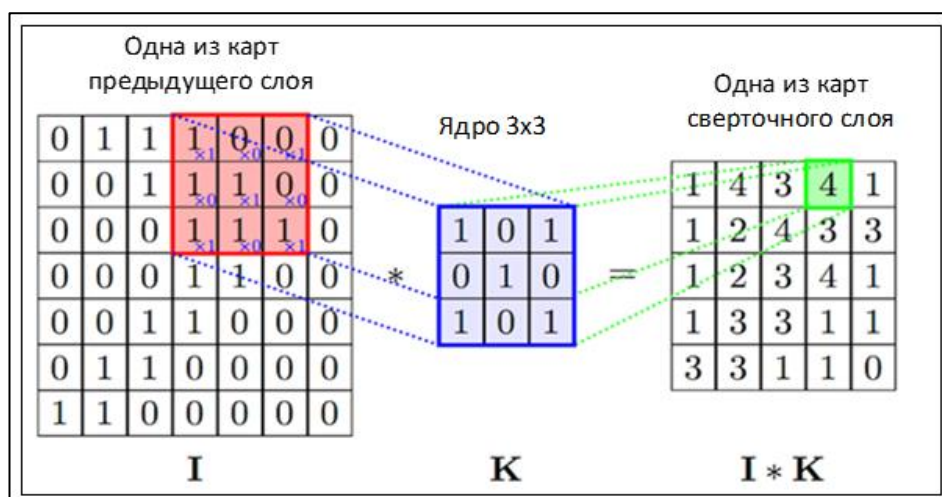


Рисунок 1.4 – Свертка

После слоя свертки обычно идет слой выборки. Слой выборки выбирает наибольшее значение в отдельных частях матрица. Таким образом из матрицы 4 на 4, выбрав максимальное значение в каждой четверти, мы получим матрицу 2 на 2. Слои свертки и выборки чередуются и в конечном счете объединяются в полносвязном слое, которые делает финальное предположение. Считается, что каждый слой свертки позволяет выявить признаки объектов, передвигаясь по уровню абстракции в сторону высокого уровня: пиксели, линии, чешуя, рыба, косяк. Примерные алгоритм работы сверточной нейронной сети 1.5

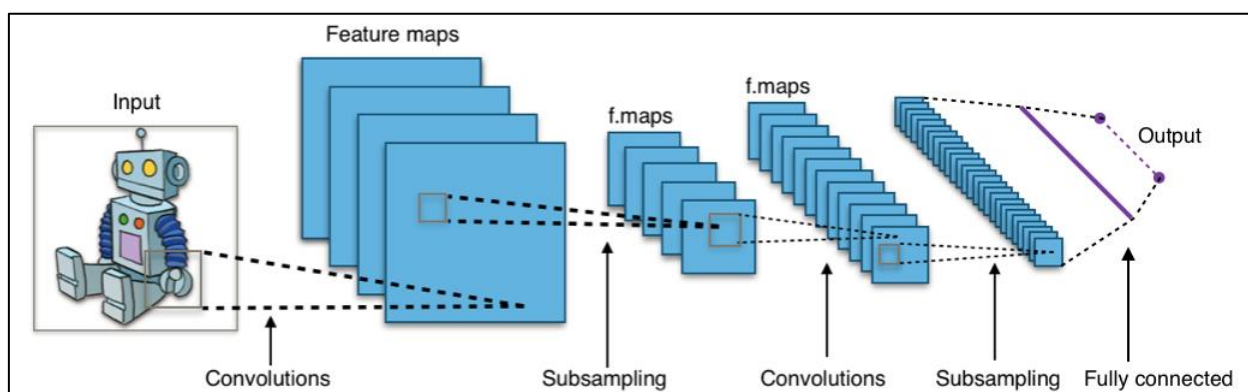


Рисунок 1.5 Схема работы сверточной нейронной сети

Процесс обучения нейронных сетей состоит в основном из метода обратного распространения ошибки. Нейронная сеть сделала предположение и ошиблась, мы можем обнаружить путь, которым прошла эта ошибка и изменить параметры нейронов в сети на этом пути. Таким образом, ошибаясь

и изменяясь в местах, где произошла ошибка, нейронная сеть подбирает нужные параметры и учиться – старается минимизировать ошибки.

Процесс создание рабочей модели нейронной сети, состоит из 5 основных этапов:

- получение исходных данных;
- предобработка данных;
- обучение;
- постобработка;
- интеграция.

Получение исходных данных состоит в подборе или сборе набора размеченных данных, в случае обучения с учителем.

Предобработка данных позволяет нам немного расширить исходных набор данных с помощью аугментаций изображений, например: размытия или вращения. Также предобработка позволяет адаптировать изображения и приблизить к реальным изображениям, которые будут использоваться, например: имитировать сжатие jpeg.

На этапе обучение нейронной сети мы получаем некоторую модель, которая может выполнять поставленную задачу.

В этап постобработки модели мы можем провести квантизацию модели, то есть уменьшения размерности параметров нейронной сети. Это может позволить ускорить работу нейронной сети. Или, например на основе работы обученной нейронной сети обучить меньшую нейронную сеть, если в наборе данных много неразмеченных изображений.

Также к этапу постобработки нейронной сети можно отнести перенос обучения. Например, у нас есть 1000 изображений рыбы, выделенной квадратами и 100, где у рыбы выделен контур. 100 изображений точно не хватит для обучения модели сегментации. Однако мы можем сначала обучить модель выделения объекта квадратом, а потом на основе уже полученного изображения, где точно есть рыба, выделить ее контур.

Также такой подход применяется если данных мало. Например, у нас есть модель, определяющая коров. Большую часть базовых функций нейронной сети уже обучены, она может находить четыре ноги, туловище и голову с хвостом, условно. Наша задача найти овец. Задача похожая, мы можем немного изменить выходную часть нейронной сети и тем самым обучить ее на маленьком количестве данных.

Также можно выделить возможность тонкой настройки сети. Это когда основная задача модели не меняются, но входящие данные могут изменяться. Например, у нас есть модель поиска пластиковых бутылок, которая обучалась на данных с суши. А наша модель должна искать бутылки на воде. Используя малое количество данных, мы можем дообучить модель для использования в реках и озерах.

Существуют также другие способы пост обработки модели. Например: дропаут – это выбрасывание отдельных нейронов из сети с дообучением модели. Таким образом определение признаков объектов равномерно распределится между нейронами, что поможет избежать переобучения.

В конечном счете полученная модель интегрируется в какую-нибудь программу, например для ее использования с помощью интерфейса, а не программирования.

1.2 Существующие решения и исследования

Наиболее весомым исследованием в области отслеживания рыб является «Обзор методов для отслеживания рыб на основе компьютерного зрения» от Китайского Агрокультурного Университета. [23] Данное исследование проводит крупномасштабный обзор исследований за 7 лет с 2017 до 2023 года. Данное исследование рассматривает как архитектуры нейросетей, так и аппаратные подходы к детектированию рыбы, а также дополнение данных с помощью вспомогательных алгоритмов. Аналогично рассматриваются наиболее популярные наборы данных, метрики и выносятся предположения о дальнейшем развитии.

Исходя из исследований, приведенных выше, можно сделать вывод, что задача отслеживания рыбы делится на два этапа. Первый этап – обнаружение, подразумевает использование детектора на отдельных кадрах видео. Второй этап – отслеживание, то есть сопоставление кадров, найденных объектов на этапе обнаружения и объединение их в единый объект. Распределение моделей и алгоритмов, трекинга и детекции относительно используемости отображено на рисунке 1.6.

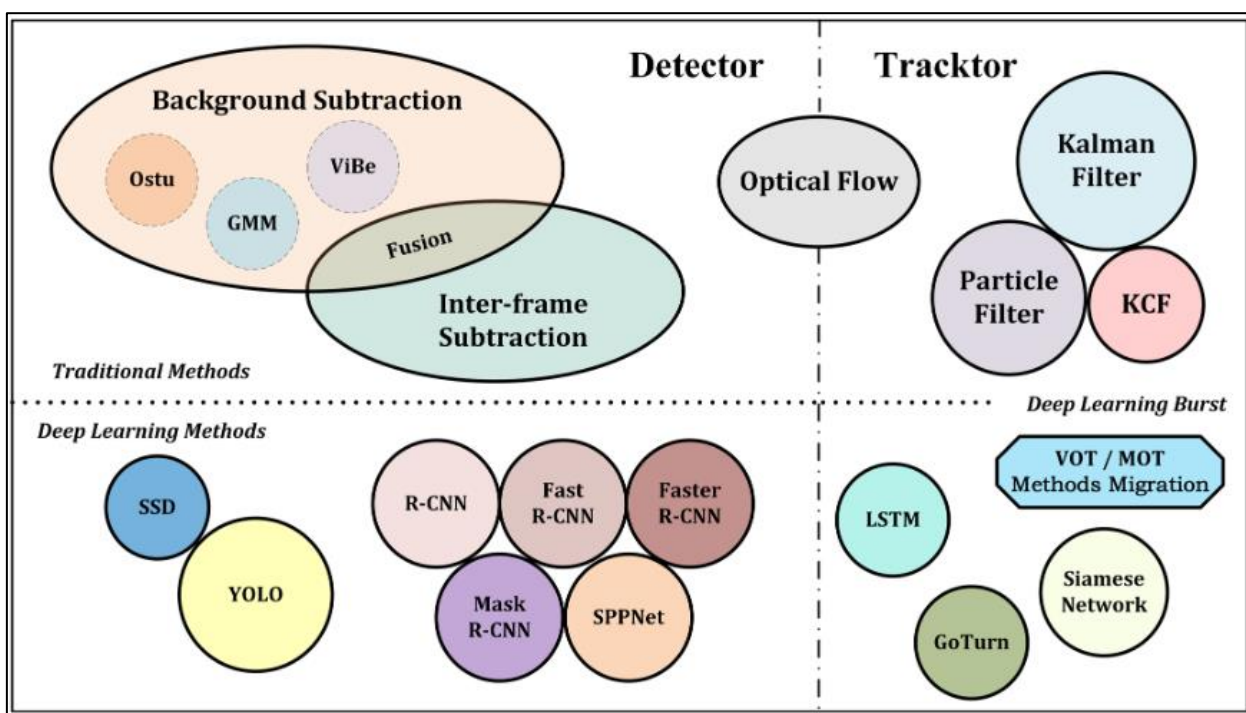


Рисунок 1.6 – Распределение детекторов и трекеров

В качестве традиционных алгоритмов используют GMM, Ostu, ViBe. Однако данные алгоритмы практически не пригодны для детектирования рыбы, так как дают низкий результат и, по сути, определяют выделяющиеся объекты. Но они достаточно хорошо себя раскрывают при работе в мутной воде. В настоящее время эти алгоритмы в основном используются в совокупности с другими алгоритмами, например в этом исследовании создана гибридная система из трех источников данных: Optical Flow, GMM, Grayscale image. [24]. Однако данная система требует весьма высокой производительности. С точки зрения выгоды использования ресурсов, есть смысл использовать более сложные модели глубокого обучения.

На основе проанализированных исследований можно сделать вывод, что YOLO модели на данный момент являются доминирующей архитектурой для обнаружения рыб. Это подтверждают исследования [25, 26, 27, 28]. Самая новая архитектура, используемая в найденных исследованиях – YOLOv8. [29]. Самая последняя и обладающая наиболее высокими метриками – YOLOv10 [30].

Также существуют решения основанные на R-CNN архитектуре. [31]. Решения основанные на двухстадийных моделях обладают большей точностью, но меньшей скоростью обработки данных. Согласно исследованию [32] Среднее отклонение по точности работы около 1 процента. При этом Скорость работы YOLOv5 превосходит в 11 раз Faster R-CNN.

Также существуют исследования по сегментации рыб различными способами, однако они обладают более низкой распространенностью. Можно предположить, что наиболее важным параметром является наличие рыбы и ее перемещение, нежели ее форма. Среди исследований можно отметить опыт доработки модели YOLOv5 сегментирующей головой. [5]. Данный алгоритм в аквариумной воде получил 95% точности по задаче детектирования и 98% точности в задаче сегментации. Скорость работы порядка 115 кадров в секунду с вычислениями на видеокарте RTX3060. Что является весьма хорошим результатом.

Особенно среди исследований сегментации рыб выделяются модели, построенные на визуальных трансформерах. Данные модели позволяют производить обучение без использования разметки данных. Как утверждают авторы исследования им удалось обучить модуль сквозной сегментации рыбы с самообучением. [33]. Однако данные модели сложны в обучении и время обработки данных достаточно высокое в стандартных реализациях архитектур этих моделей.

В качестве отдельного способа обнаружения рыбы можно выделить алгоритм Optical Flow. В общем случае оптический поток представляет собой графическое представление перемещений объектов в кадре. Существует

множество алгоритмов для определения оптического потока. Также оптический поток можно использовать как самостоятельную модель. [34]. Однако в наибольшей степени данный алгоритм раскрывается в совместной работе с моделями глубокого обучения. В определенном смысле этот алгоритм помогает обрабатывать мутные фотографии. Однако, следует отметить, что если использовать этот алгоритм для помощи в отслеживании объектов, то он налагает весьма большие ограничения по производительности аппаратной части и сильно увеличивается время работы.

Среди алгоритмов отслеживания найденных объектов выделяются алгоритмы SORT. Данные алгоритмы на основе фильтра Калмана могут уточнять положения объектов и проводить трекинг. Используются массово практически во всех моделях компьютерного зрения. В 2022 году удалось адаптировать алгоритм SORT для отслеживания рыбы по данным с сонара. Точность алгоритма составила 5%, что сопоставима с человеческой ошибкой. [35]

Весомым исследованием в системе трекинга рыбы можно назвать работу FishMOT [36], которая основывается на использовании данных о совпадении прямоугольников обнаружения объектов и фильтре Калмана. Для этого используют данные за несколько предыдущих кадров. Также предусмотрен функционал при слишком низкой оценке объекта, с вычитанием фона объекта и принятия решения не только на основе прямоугольника выделения, но и на основе пересечения сегментов объектов. Это позволяет не перепутать объекты, когда они пересекаются. Данный алгоритм получил довольно высокие метрики и обогнал все наиболее популярные универсальные трекеры. Стоит отметить, что использование вычитание фона на основе фильтров применимо только в данной задаче. В качестве входных данных подаются мальки рыб в чашке на белом фоне. Это весьма идеализированные условия по сравнению с реальными условиями низкой освещенности и высокой мутности.

В некоторых случаях используют несколько источников данных. Например, в данном исследовании проект «The Ocean Aware» планирует

использовать в совокупности сонар и оптическую камеру. Для сегментации и классификации. Они используют две модели: YOLOv3 и Mask R-CNN. Первая модель показала лучшие результаты 0.73 против 0.62 mAP, что является большим разрывом. Также в этом исследовании используется Norfair метод сортировки, который показывает удовлетворительные результаты. Кроме прочего в данной статье проводится анализ точности трекинга с изменением параметров задержки создания объекта и количества кадров, которые использовались при обучении. Наиболее высокой точностью обладает инициализация через 11 повторяющихся кадров с указанным объектом. Однако данный параметр может быть индивидуальным для разных проектов. Что более интересно – количество кадров в секунду для лейблинга видео практически не повлияло на итоговые результаты. Данные результаты будут полезны при подготовке датасета

Интересным исследованием, связанным с предметной областью, можно назвать исследование восстановления цветов под водой. В данной сфере есть несколько публикаций. Наиболее интересной из них считаю исследование школы морской науки и технологий [37]. В данной публикации рассматривается подход из двух параллельных этапов. В первом определяется доминирующий цвет изображения. Это может быть зеленое фото, синее и желтое в зависимости от цвета мутной воды. На основе выбранного основного цвета получается цветное фото с небольшой потерей деталей. Детали восстанавливаются с помощью корректировки контраста серой версии исходного фото. Далее полученные результаты совмещаются и получается новое фото с восстановленными цветами. Существуют и более сложные алгоритмы [38]. Интересно, что результаты данной работы оказались применимы для не только для восстановления подводных снимков, но и для просмотра сквозь туман, песчаный шторм и изучения фото, сделанных при слабой освещенности. Существуют также алгоритмы, основанные на нейронных сетях. [39]. Результаты исследований могут быть полезны для нашей работы, так как позволяют усилить признаки объектов под водой и возможно улучшить

результаты. В таком случае рекомендуется добавить данный алгоритм в качестве предобработки изображений. Примерный алгоритм восстановления цвета можно наблюдать на рисунке 1.7.



Рисунок 1.7 – Алгоритм восстановления цвета

1.3 Методы отслеживания рыбы

Резюмируя проанализированные исследования, можно разделить способы отслеживания рыбы по целевым параметрам:

- количество сущностей для отслеживания;
- точность работы;
- определение траектории движения;
- определение параметров рыбы их видов;
- время обработки одного кадра;
- трехмерная реконструкция.

По этапам отслеживания рыбы выделяют два основных: позиционирование (нахождения рыб на отдельном кадре) и отслеживание (совмещение рыб на нескольких кадрах в одну сущность, экземпляр).

С распространением нейронных сетей глубокого обучения задачу трекинга (определения и отслеживания) можно также разделить решения на отдельные подходы:

- эвристические подходы, например: основанные на вычитании не движимых частей изображения (выделение фона) и фильтрования Калмана для отслеживания;

- основанные на обучении, например: обучение сегментации на основе изображений с выделенными контурами рыб и обучение отдельной модели для трекинга;

- смешанные, например: обучение модели на исходных данных для позиционирования и отслеживания на основе фильтра Калмана.

Методы отслеживания уже найденных объектов тоже можно разделить на 2 группы: эвристические и основанные на обучении.

Эвристические методы отслеживания:

- фильтры, в основном основанные на фильтре Калмана;
- оптический поток, используется для определения движения объектов между кадрами за счет изменения пикселей;

- SORT алгоритмы, используют фильтр Калмана для прогнозирования состояния объекта и венгерский алгоритм для расчета сходства объектов на основе пересечения квадратов выделения.

Методы отслеживания на основе обучения:

- долгая коротко-временная память (LSTM) – рекуррентная сеть позволяющая прогнозировать траекторию рыбы на основе генетических алгоритмов;

- GOTURN – модель, обученная на простых обобщенных характеристиках движения, за счет структуры прямого распространения обладает высокой скоростью;

- Сиамские сети – сети, основанные на сопоставлении двух экземпляров и нахождения их корреляции;

- Графовые сети (GNN) – отслеживают объект на основе анализа взаимоотношения между объектами;

- Трансформеры – нейронные сети, позволяющие совместить функционал позиционирования и отслеживания в одной модели.

Вспомогательные модули поиска и отслеживания рыбы:

- улучшение изображения – различные алгоритмы и нейронные сети удаления шума, повышения разрешения, цветокоррекции и так далее.

- Ре-идентификация – программные модели, позволяющие сопоставить объект на разные кадры за счет особенностей, например: окраса, вида, размера, что позволяет увеличить точность трекинга при окклюзии двух объектов.

1.4 Формирование требований

Основываясь на проведенном исследовании решений и методов, можно выдвинуть требования к разрабатываемой системе, учитывая следующие выводы:

- современные архитектуры нейронных сетей позволяют достичь скорость работы эвристических методов при точности глубоких нейронных сетей, при этом компенсировать не использование алгоритмов предобработки или обработки из нескольких источников данных для сокращения аппаратных мощностей;

- современные решения позволяют получить отслеживание рыб с человеческой точностью при работе в режиме реального времени;

Таким образом можно сформировать следующие требования:

- работа на современных компьютерах с частотой выше 30 кадров в секунду;

- возможность идентификации объектов между кадрами;

- возможность отслеживания перемещения объектов через центральную вертикаль кадра;

- точность определения сравнимая с человеческой, около 5% ошибок;

- определение параметров рыбы: высота, ширина описывающего прямоугольника;

- наличие интеграции в программу для использования с помощью удобного интерфейса.

Поставленные требования позволят разработать новую, комплексную систему компьютерного зрения, основывающуюся на оптических данных. В качестве усовершенствования подходов необходимо проанализировать новые

архитектуры моделей глубокого обучения и сравнить их точность, скорость выполнения.

Также необходимо провести дополнительный поиск, анализ и обработку данных из открытых источников. Найти новые данные, не указанные в изученных исследованиях. В дополнение необходимо ввести новую систему подсчета количества прошедшей рыбы и разработать систему считывания параметров рыбы: длины и ширины описывающего квадрата. Это позволит в будущем дополнить систему датчиками расстояния, сонарами или стереозрением и получить привязку к реальным размерам проплывающей рыбы.

Таким образом проводимая работа позволит актуализировать данные, указанные в изученных исследованиях и специализировать их на определенное задаче – подсчета и анализа проплывающей рыбы.

2 РАЗРАБОТКА МЕТОДИКИ И АЛГОРИТМОВ

2.1 Схема исследования

После вынесения требований системы и постановки основных ее характеристик можно выработать поэтапную схему исследования:

- выбор предмета исследования (описано ранее);
- анализ исследований (описано ранее);
- поставка гипотезы и задач (описано ранее);
- подбор инструментов;
- сбор исходных данных;
- определение метрик работы моделей CNN;
- анализ популярных архитектур моделей;
- обучение моделей детектирования, сравнение;
- постобработка модели детектирования;
- разработка системы отслеживания;
- анализ возможных дополнений системы;
- конечная интеграция в графическую систему управления.

Дополнительно в рамках исследования требуется проверить следующие предположения:

- обработать набор данных в черно-белый формат;
- использовать hsv представление изображений;
- использовать восстановление цветов перед детектированием

2.2 Основные программные инструменты

В качестве основного инструмента для написания программ будет использоваться язык программирования python. Это наиболее популярный язык программирования для анализа данных и разработки моделей глубокого обучения [40]. В основном python выбирают по трем причинам: это чистый и лаконичный код, наличие интерпретатора, большое сообщество разработчиков и как следствие большое количество модулей.

Язык программирования Python остается наиболее популярным не только в сфере разработки нейросетей, но и в целом является самым популярным языком программирования[41]. Это позволяет ему обладать большим сообществом разработчиков, работающих над синтаксисом языка, работе его функций, стандартных и дополнительных библиотек (модулей), а также выпускающих большое количество обучающих материалов. На рисунке 2.1 распределение популярности языков программирования.






May 2024	May 2023	Change	Programming Language		Ratings	Change
1	1			Python	16.33%	+2.88%
2	2			C	9.98%	-3.37%
3	4	▲		C++	9.53%	-2.43%
4	3	▼		Java	8.69%	-3.53%
5	5			C#	6.49%	-0.94%

Рисунок 2.1 – Рейтинг популярности языков программирования

Среди языков программирования также используется Rlang. Однако данный язык обладает крайне слабым сообществом и как следствие малым количеством модулей, низкой их проработанностью. Аналогичная ситуация с языком программирования Java, который используется в сфере обработки данных. Также в недавнее время появился новый язык программирования mojo. Этот язык представлен, как усовершенствованная версия языка python с частичной обратной совместимостью. Данный проект представлен другими разработчиками – не основателями python. К сожалению, данный язык еще не прошел проверку временем и не обрел сообщество и достаточное количество модулей. Однако в некоторых тестах использования нейронных сетей удалось

получить преимущество в 250 раз [42]. Одним из главных преимуществ языка является многопоточность, что позволяет использовать несколько ядер.

После выбора языка необходимо перейти к выбору основного фреймворка. Обычно выделяют два основных фреймворка – это TensorFlow и PyTorch. На основе запросов в поисковике Google можно сделать вывод, что последний год доминирующим фреймворком по запросу является PyTorch. Динамика запросов в поисковой сети отображена на рисунке 2.2

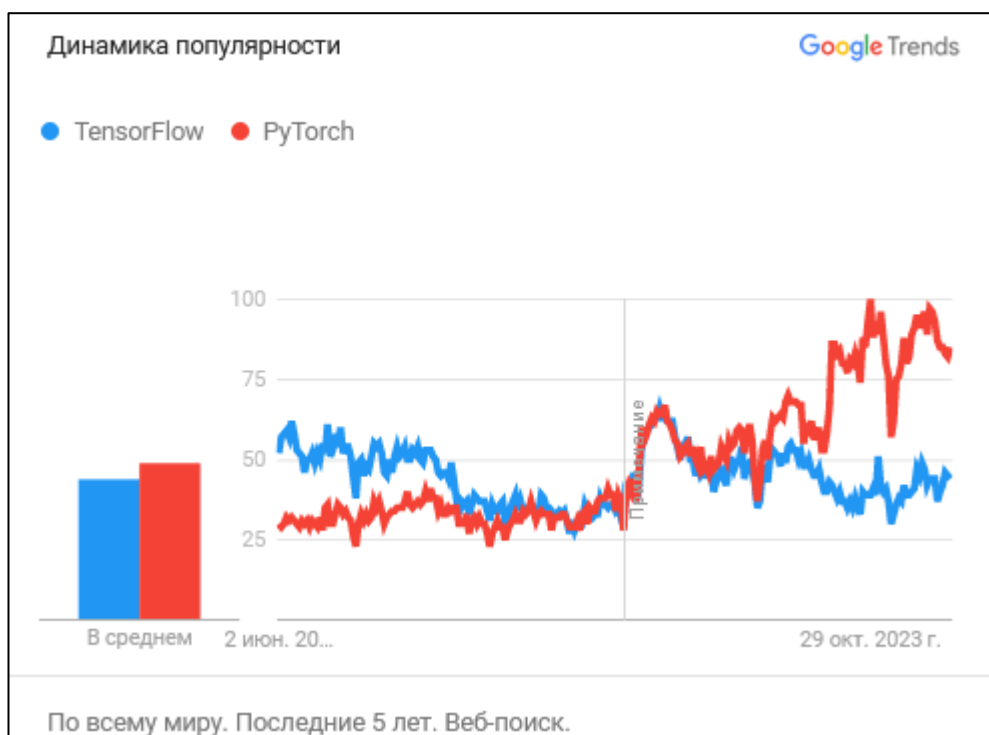


Рисунок 2.2 – Динамика запросов TensorFlow и PyTorch

Противоположный вывод можно сделать исходя из анализа открытых репозиторий в системе Github [43]. В данном сравнении участвует также Keras, который по сути является высокоуровневой оболочкой для TensorFlow и PyTorch. Таким образом можно сделать вывод, что последнее время набирает большую популярность PyTorch. Учитывая все факторы, был сделан выбор в пользу PyTorch, так как данный инструмент более знаком и с ним имеется опыт работы. Также был выбран PyTorch, потому что на его основе построена система YOLO от компании Ultralytics [44]. Сравнение репозиторий отображено на рисунке 2.3




pytorch/pytorch 		keras-team/keras 		tensorflow/tensorflow 	
★ Stars	79,155	★ Stars	61,120	★ Stars	183,261
🔗 Forks	21,331	🔗 Forks	19,352	🔗 Forks	73,995
📄 Open issues	14,229	📄 Open issues	211	📄 Open issues	2,969
📅 Age	8 years ago	📅 Age	9 years ago	📅 Age	9 years ago
➕ Last commit	12 hours ago	➕ Last commit	12 hours ago	➕ Last commit	12 hours ago
⚙ License	NOASSERTION	⚙ License	Apache-2.0	⚙ License	Apache-2.0
🔗 Language	Python	🔗 Language	Python	🔗 Language	C++
Remove repo		Remove repo		Remove repo	

Рисунок 2.3 – Репозитории фреймворков нейронных сетей Python

YOLO - на данный момент это одни из самых используемых моделей компьютерного зрения. Данные модели приняли необычный путь развития и выпускают новую архитектуру по мере разработки, сохраняя общее название. На данный момент последней версией является 10, самой используемой является версия 8. [45]. На основе проанализированных исследований можно сделать вывод, что данные модели продемонстрируют наилучший результат по метрикам и скорости обучения.

Наиболее популярным модулем для компьютерного зрения можно назвать модуль OpenCV [46]. Этот модуль позволяет провести полную разработку систему компьютерного зрения и содержит в себе как алгоритмы для простой обработки изображений, так и предобученные нейронные сети.

Основная разработка будет вестись в формате ноутбуков (IPython Notebook)[47]. Данный формат благодаря использованию интерпретируемого языка программирования позволяет разрабатывать программу по частям, запуская их отдельно и проверяя вывод данных. Это сильно упрощает процесс разработки нейронных сетей и анализа данных.

Для языка Python не требуется полноценной среды разработки. Достаточно редактора кода. Редактор кода VS Code обладает низким порогом вхождения и позволяет быстро приступить к разработке [48]. Также данный

редактор обладает большим количеством плагинов, для автоматического дополнения кода, анализа и использования системы ноутбуков [49]. На данный момент это один из самых популярных редакторов кода и с ним имеется большой опыт работы. Интерфейс VS Code отображен на рисунке 2.4.

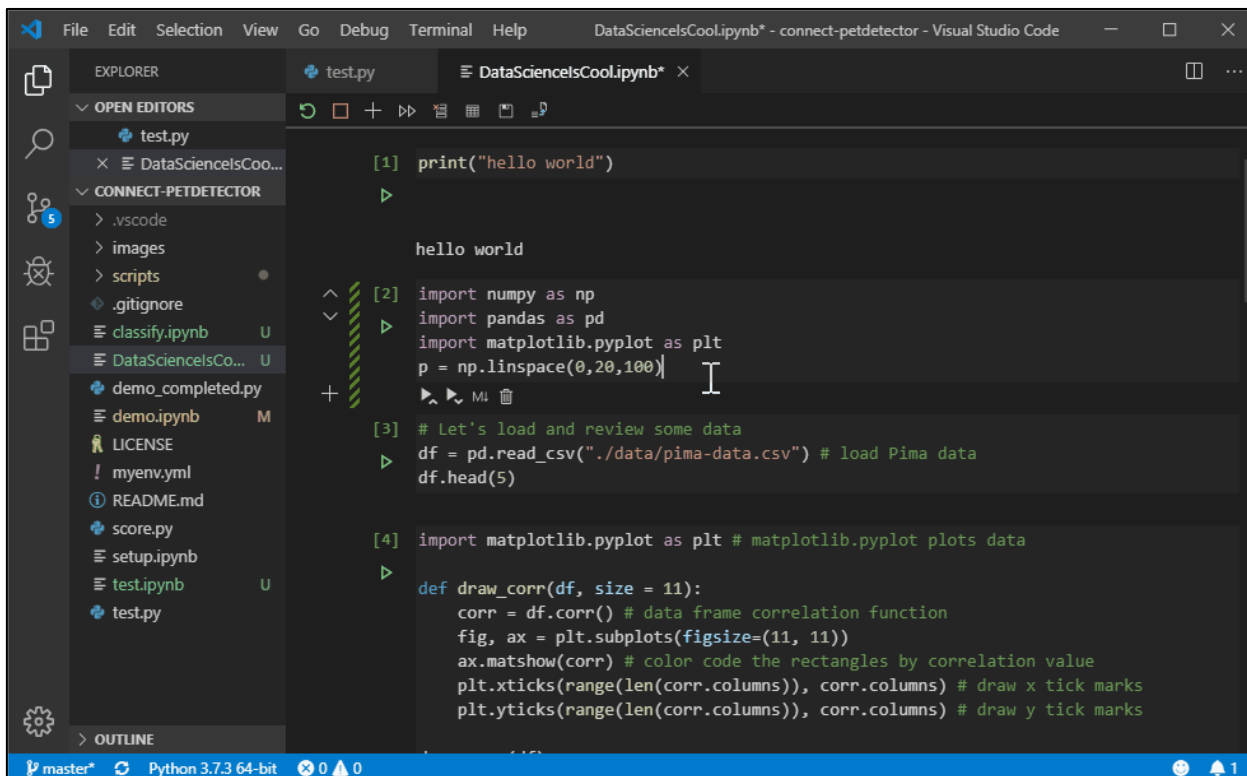


Рисунок 2.4 – Интерфейс VS Code

Для разработки графического системного приложения будет использоваться модуль PyQT. Это один из самых популярных фреймворков для написания графических интерфейсов. Он позволяет в короткие сроки разрабатывать базовые интерфейсы для программ. [50]. Также благодаря большому сообществу разработчиков, он обладает обширной документацией и большим количеством учебного материала, в том числе по интеграции данного модуля с другими системами, например OpenCV [51].

2.3 Метрики и критерии исследования

Итоговая модель будет оцениваться путем подсчета отклонения (bias). Отклонение – это величина, описывающая разницу истинного значения от предсказанного. Мы будем изменять ее в процентах. Например: наша система

предположила, что проплыло 95 рыб, а на самом деле 100. Таким образом наше отклонение составления 5% от истинного значения.

Однако для анализа результатов обучения необходимо вывести и уточнить метрики оценки качества работы нейронной сети.

2.3.1 Метрики детектирования

В метриках машинного обучения почти все метрики отталкиваются от понятия матрицы ошибок. Данная матрица представляется из себя четыре возможных варианта исхода прогнозирования: истинно положительный, ложно положительный, ложно отрицательный и истинно отрицательный. Например: нам нужно определить яблоко ли на изображении или груша. TP – мы предположили, что это яблоко и оно им и оказалось. Или FP – мы предположили, что это яблоко, а оказалось, что это груша. Графическое отображение матрицы ошибок изображена на рисунке 2.5

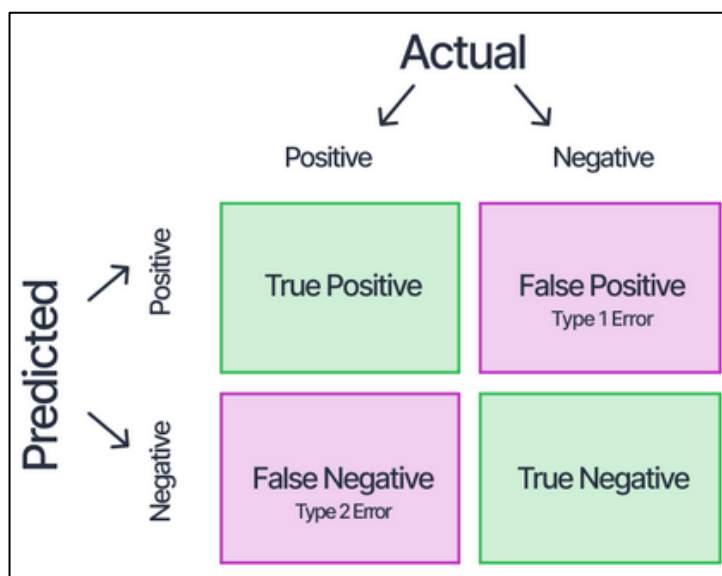


Рисунок 2.5 – Матрица ошибок

С другой стороны, кроме отслеживания правильности предположения, в задачах обнаружения, когда необходимо выделить объект квадратом, важным параметром является точность выставления этого квадрата. Обычно для отображения этого параметра используется метрика IoU (Intersection over Union). Данная метрика отображает количество пространства, которое было

предсказано верно относительно общего предсказанного и истинного пространства. IoU в том числе используется для определения матрица ошибок. В данном случае устанавливается порог, выше которого предположение считается верным. На изображении приведен пример для порога 0.5. Пример распределения IoU изображен на рисунке 2.6

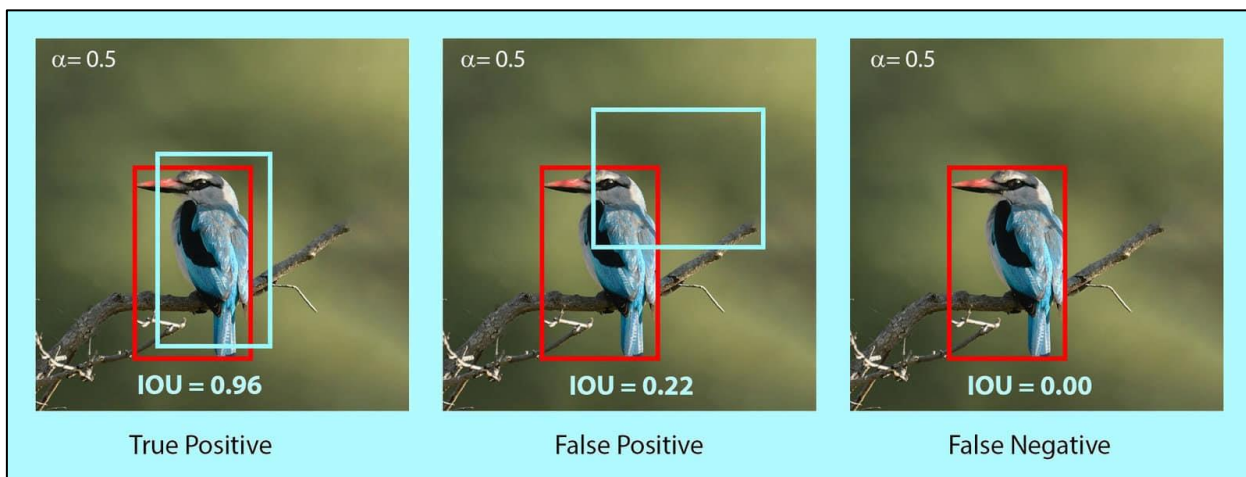


Рисунок 2.6 – Распределение IoU

Основываясь на этой концепции, выводятся другие метрики. Самой простой из них является Accuracy. Эта метрика высчитывает количество верно угаданных относительно всех предположений. Вычисляется по формуле 1.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Однако в большинстве случаев мы не можем использовать эту метрику. Например: у нас есть классификатор банковских переводов. 90 переводов не мошеннические. Наш классификатор определил верно 80 из них. 10 переводов мошеннические. Мы определили верно только 5. Таким образом мы предсказали верно 85% ответов, что является не плохой точностью. Однако если мы пометим просто все ответы, как не мошеннические, то получим точность 90%. Со второй стороны, в данной задаче важнее охватить все мошеннические случаи, чем ложно определить не мошеннические.

Часто для оценки работы используются две метрики *precision* (точность, не путать с *accuracy*) и *recall* (полнота). Вычисляются соответственно по формуле 2 и 3.

$$precision = \frac{TP}{TP + FP} \quad (2)$$

$$recall = \frac{TP}{TP + FN} \quad (3)$$

Precision можно интерпретировать как количество образцов, фактически принадлежащих к положительному классу и правильно определенных, из всех образцов, которые, по прогнозам модели, относятся к положительному классу. *Recall* показывает количество образцов, которые правильно предсказаны как принадлежащие к положительному классу, среди всех образцов, которые действительно принадлежат к положительному классу. Именно метрика *recall* была бы наиболее популярна в задаче с определением мошеннических переводов. Однако наиболее часто нужно найти баланс между этими двумя метриками. Например, нужно найти 95% мошеннических переводов, но не бить ложную тревогу в 50% случаев.

Часто, использование двух метрик одновременно довольно сложно. Например, если нужно найти в автоматическом режиме лучшую модель среди нескольких. Удобней сравнивать по одному параметру. Есть много метрик, совмещающих *Precision* и *Recall*. Одной из них является метрика *F*. Она позволяет определить гармоническое соотношение точности и полноты для положительных классов. Наиболее часто она встречается с параметром 1, это означает что находится среднее количество полноты и точности. Параметр от 0 до 1 будет отдавать приоритет точности, а при параметрах больше 1, приоритет отдается полноте. Сама метрика будет стремиться к 1 при точности и полноте равных единицы и к 0 если один и параметров стримится к нулю. Вычисление *F1* и *F* с коэффициентом производится соответственно по формулам 4 и 5.

$$F1 = \frac{2 * precision * recall}{precision + recall} \quad (4)$$

$$F1_{\beta} = (1 + \beta^2) = \frac{precision * recall}{(\beta^2 * precision) + recall} \quad (5)$$

Также существует еще один способ объединения этих двух основных метрик – это метрика AP (Average Precision). Глобально эта метрика описывает площадь под кривой precision-recall. Измеряя для каждого объекта одного класса по очереди правильность определения, аккумулирую результаты в TP, FP, FN мы сможем каждый раз рассчитывать значения точности и полноты. В результаты мы получим некоторое количество изменений. Таким образом сможем отследить зависимость изменения полноты от точности. Далее полученные значения накладывают на график, интерполируют и считают площадь фигуры под кривой. Пример графика изображен на рисунке 2.7

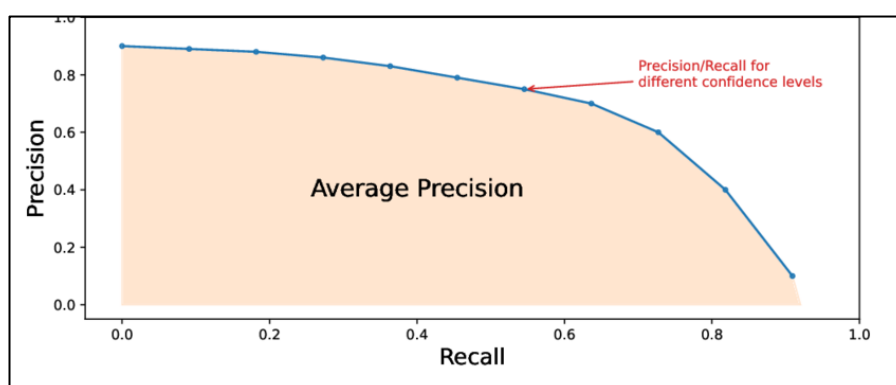


Рисунок 2.7 – График Average Precision

Исходя из этой метрики были придуманы производные, которые считают, насколько хорошо работает модель при разных порогах IoU. Наиболее популярны: AP50 – порог IoU = 50%; AP75 – порог IoU 75%; AP50-95 – это когда высчитывается AP от порога 50% до 95 с шагом 5%, далее берется среднее арифметическое всех результатов. На данный момент метрика AP50-95 или как пишут чаще – $AP@[0.5:0.05:0.95]$ является наиболее популярной.

В задачах, когда нам нужно находить несколько классов вводят метрику mAP. Зачастую это просто среднее арифметическое AP за все классы.

2.3.2 Метрики отслеживания

Кроме метрик точности определения объектов на отдельных фотографиях также существуют метрики отслеживания точности трекинга. Иначе говоря, насколько хорошо работает программа по объединению одной сущности на разных кадрах.

Глобально данные метрики принято делить на 2 группы: Комплексные и Специализированные.

Комплексные метрики:

- MOTA (Multiple Object Tracking Accuracy) – это метрика позволяющая оценить точность отслеживания нескольких объектов, учитывая ошибки, ложные срабатывания и пропуски, формула 6;

- IDF1 (ID F1 Score) – это метрика позволяющая оценить точность и полноты идентификаторов, формула 7;

- HOTA (Higher Order Tracking Accuracy) — это метрика, описывающая среднее геометрическое обнаружения и точности ассоциации, формула 8.

Пример распределения ошибок при трекинге отображен на рисунке 2.8

$$MOTA = 1 - \frac{\sum(FN + FP + ID_{sw})}{\sum GT} \quad (6)$$

где FN – ложноотрицательный;

FP – ложноположительный;

ID – смена идентификатора;

GT – истинное количество объектов.

$$IDF_1 = \frac{TP}{(TP + 0,5FP + 0,5FN)} \quad (7)$$

$$HOTA_{\alpha} = \sqrt{\sum_{c \in \{TP\}} \frac{A(c)}{(|TP| + |FN| + |FP|)}} \quad (8)$$

$$A(c) = \frac{|TPA(c)|}{(|TPA(c)| + |FNA(c)| + |FPA(c)|)}$$

где α — порог IoU;

c — количество положительных траекторий выборки;

TPA — пересечение между двумя траекториями;

FPA — траектории вне пересечения прогнозируемой траектории;

FNA — обнаружения вне пересечения истинной траектории.

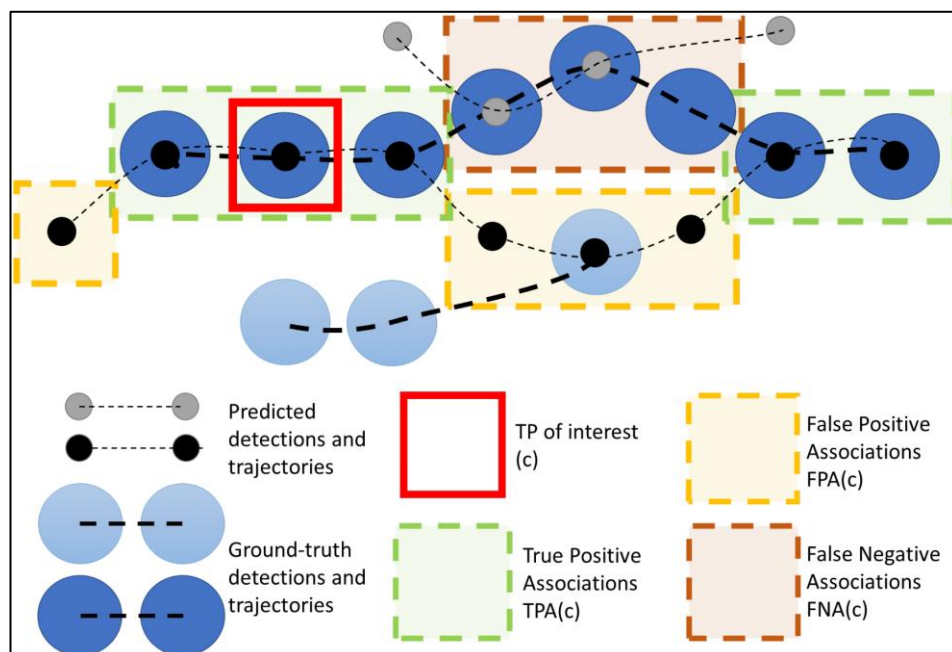


Рисунок 2.8 – Пример оценки трекинга

Специализированные метрики:

- MT (Mostly Tracked) — это метрика показывающая, какая доля объектов отслеживалась большую часть времени (представлена более 80% времени);

- ML (Mostly Lost) – это метрика показывающая, какая доля объектов пропущена большую часть времени (представлена менее 20% времени).

- Rcll (Recall) – это метрика оценивающая, какая доля объектов была корректно отслежена;

- D Sw (ID Switches) – это метрика учитывающая, сколько раз трекер переключался между разными объектами, то есть сколько раз менялся идентификатор отслеживаемого объекта.

2.4 Архитектуры нейронных сетей

Одними из первых исследований, которые проводились в сфере обнаружения объектов и принесли значительное влияние — это так называемые каскады Хаара. Хотя первое исследование по теме датируется еще 1940 годами.[52] Каскад Хаара — это некоторое описание объекта, основывающееся на локальных изменениях текстуры изображения[53]. Например, если мы видим светлую область с затемнением внутри, при этом еще есть темная полоса выше на всю длину области, то есть основания полагать, что это глаз и бровь. Если мы видим два глаза примерно на одном уровне и светлый промежуток между ними, и темная линия под светлым участком, то можно сказать, что это похоже на лицо. Далее создавались алгоритмы машинного обучения поиска и обучения функции Хаара, например AdaBoost[54]. Эти алгоритмы стали началом больших исследований в области машинного обучения и компьютерного зрения.

Следующим этапом развития можно назвать появление сверхточных нейронных сетей на основе регионов или сетей многократного обнаружения объектов. Ярким представителем является R-CNN[55]. Эти модели по сути своей являлись развитием классификаторов. Исходное изображение разделялось каким-то способом на области интереса. Данных способов было много, например: модель RPN. После этого каждый регион отдельно анализировался, пытаясь предсказать что находится в данной области. Наиболее ярким и известным алгоритмом является Faster R-CNN[56]. Эти алгоритмы являлись достаточно точными, но весьма медленными из-за предварительного шага обнаружения регионов. В системах реального времени они были слабо применимы по сравнению с современным.

Более популярными последнее время становятся алгоритмы, которые обрабатывают изображение за один проход. Это два схожих алгоритма YOLO (You Look Only Once) и SSD (Single Shot Detector), хотя чаще их все-таки разделяют.

YOLO используется сетку. Разбивает изображения на блоки и в каждом блоке пытается выделить еще N количество блоков[57]. После этого для каждого найденного блока проходит классификация. Далее блоки соединяются в одно большое предположение. Некоторую область, ограниченную рамкой, с какой-то вероятностью содержащий конкретный объект.

SSD используется карту признаков с каждого сверточного слоя [58]. После получения всех карт они объединяются и запускается дополнительная свертка, которая прогнозирует ограничивающие рамки и вероятности.

Первые версии SSD превосходили YOLO. Однако YOLOv3 и YOLOv4 начали обгонять модели с SDD, например RetinaNet [59] и стали выдавать полноценный предсказания в реально времени, то есть быстрее 30 кадров в секунду. На данный наиболее известной и быстрой моделью является YOLOv10 [60]. Данные модели находятся в открытом доступе и распространяются в том числе в переобученном состоянии.

Современные исследования в области детектирования и классификации рыбных ресурсов в большинстве случаев используют модель YOLO начиная с 3 версии. Так как считают ее наиболее успешной. [27, 1, 5]. В некоторых исследованиях модель YOLO видоизменяется, для улучшенной классификации [61], сегментации [5] или дополняется другими алгоритмами фильтрации и трекинга [36] для получения более точных результатов. Наиболее активно применяются Optical Flow [2] и фильтр Калмана для уточнения данных за несколько кадров [62] или для использования вместе с данными с сонаров и других датчиков[1, 63]. Так же в некоторых исследованиях используются алгоритмы 3D реконструкции [64], в том числе с помощью стереозрения [65].

Начиная с 2020 года набирают популярность модели основанные на DETR [66]. Данные модели весьма долго обрабатывают данные, по сравнению с теми же SSD моделями, однако активно развиваются. Идея данных моделей основывается на предварительном получении карты признаков и энкодировании-декодировании данных. Таким образом можно получить как сегментированные изображения, так и детектированные.

Наиболее развитой моделью сейчас можно назвать YOLOv10 архитектуру. Данная модель использует сквозное определение объектов, как у DETR моделей. В основе она использует SCPNet[60].

Также среди исследований можно выделить публикации связанный с использованием *amba* блоков, которые могут заменить популярные на данный момент трансформеры[67]. Возможно, нейронные сети в будущем будут использовать *amba* блоки или GELAN архитектуру. Данные технологии можно выделить как наиболее перспективные.

2.4.1 R-CNN

Глобально, у задачи детектирования два подзадачи: найти возможные объекты и классифицировать их. Можно по очереди проходить по объекту скользящим окном и пытаться классифицировать каждое решение, лучшие метрики вывести в результат, но это займет много времени.

Чтобы ускорить эту задачу была разработана архитектура R-CNN, которая сужает поиск возможных положений объекта. Алгоритм Region Proposal (Selective Search) принимает на вход исходное изображение и на выходе прогнозирует возможные положения объектов. Далее каждое полученное предположение мы классифицируем [55]. Схема работы R-CNN модели отображена на рисунке 2.9

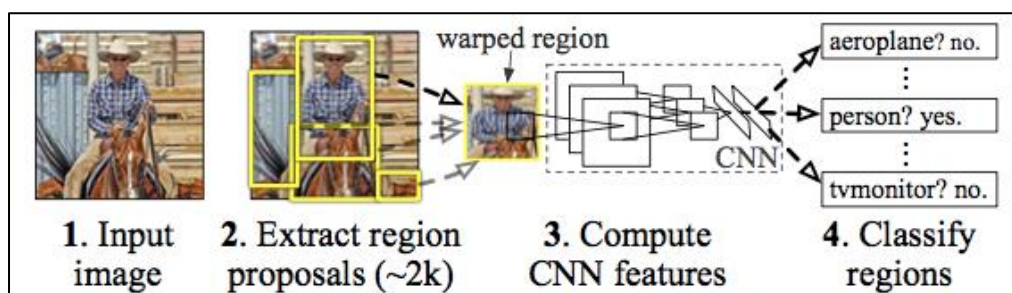


Рисунок 2.9 – Схема работы R-CNN модели

И данная архитектура может быть хорошо и достаточно точна. Однако в случае детектирования большого количества предположений на первой стадии, вторая стадия может обрабатывать данные слишком долго. Собственно, данные модели так и называются – двухстадийные.

Исходя из основного ограничения этой модели была выведена новая модель – Fast R-CNN. В данной модели исходное изображение проходит не только через предположение объектов. Но и через сверточную сеть, которая позволяет сократить количество регионов. Таким образом количество выполнений второй стадии сокращается, за счет вытягивания признаков в первой стадии [68]. Схема работы Fast R-CNN отображена на рисунке 2.10

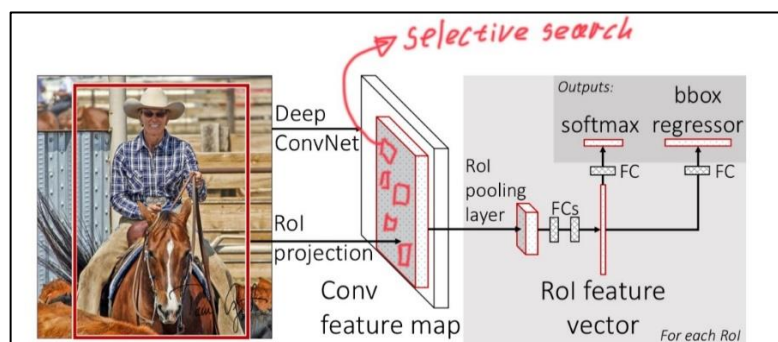


Рисунок 2.10 – Схема работы Fast R-CNN

Буквально через 3 месяца после релиза Fast R-CNN, вышла новая модель Faster R-CNN. Исходная модель Была изменена не сильно. Теперь подбор предположений о расположении объектов происходит за счет отдельной нейронной сети, которую можно обучать. Глобально схема работы не изменилась [56]. На данный момент эти модели считаются устаревшими, так как выпущены более 8 лет назад.

Продолжением развития R-CNN сетей можно считать выпуск новой архитектуры Mask R-CNN. В данном случае в Faster R-CNN добавили еще одну группу слоев свертки, позволяющую выделить контур объекта. Также имеются некоторые доработки со стороны алгоритма, совмещающего карту признаков и предположения объектов. [69] Данная модель используется в исследованиях и в настоящее время.

2.4.2 SSD

Модель SSD (Single Shot Detector) модель построена на принципе вытягивания признаков объектов разной величины. Таким образом слоя свертки исходного изображения постепенно поступают друг в друга с уменьшением разрешения. Между каждым блоком свертки вытягивается признаки и проходят через блоки классификации и детектирования. Далее выходы всех классификаторов со всех величин объединяются с помощью блока слоев Fast NMS. [58]

Внутри каждого классификатора есть некоторое количество стандартных блоков, которые проверяются на наличие объектов. Вытягиваются блоки с наибольшей вероятностью, при этом границы блоков могут подстраиваться относительно стандартных и вычисляется вероятность каждого блока. Обобщенная архитектура модели изображена на рисунке 3.3

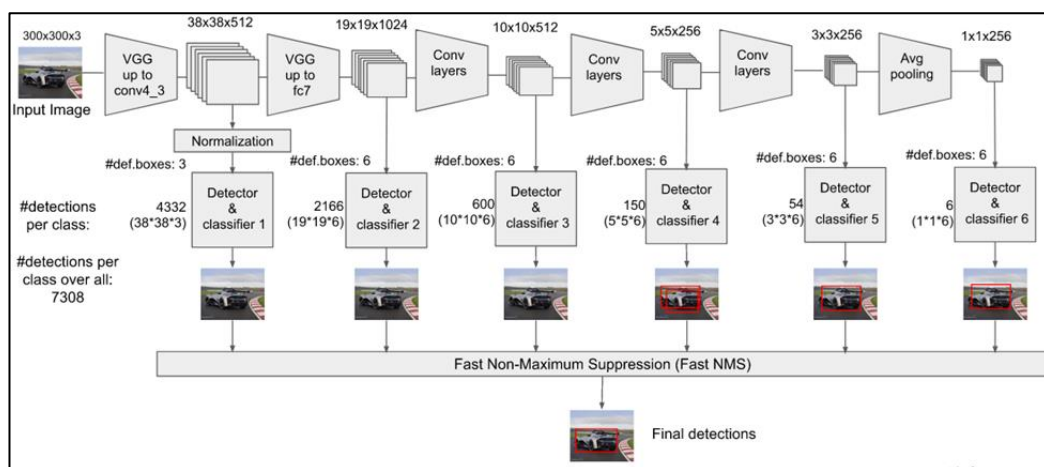


Рисунок 2.11 – Архитектура SSD модели

2.4.3 YOLO

Архитектура моделей YOLO (you only look once) весьма похожа на модель SSD. Скорее даже на оборот. SSD вышел позднее модели YOLO и перенял многие аспекты реализации у нее. Так же, как и модель SSD эта модель работает за один проход модели. Также модель YOLO это первая модель, которая была успешно запущена на мобильном телефоне.

YOLO архитектура на высоком уровне абстракции представляет собой простую модель, где исходное изображение разбивается на сетки. На основе задачи регрессии строится несколько ограничивающих прямоугольников, параллельно рассчитывается классовая принадлежность частей изображения на основе простой сетки. Далее два предположения совмещаются и получаются прямоугольники, выделяющие объекты определенного класса. [70]

Изначально в процессе развития модели YOLO были медленнее моделей R-CNN и SSD. Однако спустя несколько итераций YOLO выбрался в явные лидеры рейтингов моделей. Большой вклад с моделью YOLO внес Алексей Бочковский. Он разработал архитектуру YOLOv4, которая прекрасно совмещала скорость работы и точность. Особенность работы архитектуры YOLO отображена на рисунке 2.12

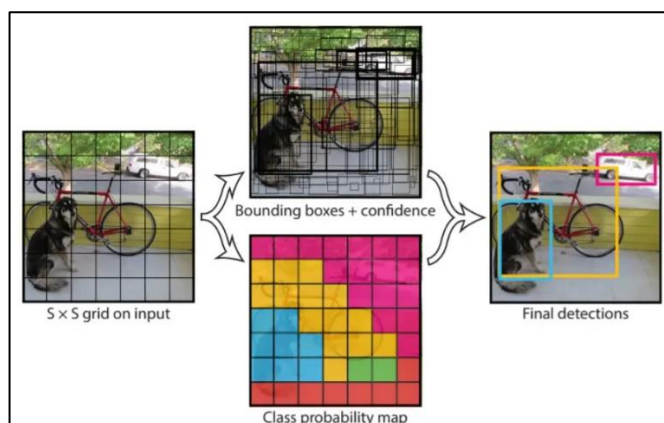


Рисунок 2.12 – Принцип работы сетей YOLO

Четвертая версия использует много нововведений, таким как WRC слои, CSP, CmBN, SAT. И многие другие. Также при ее обучении используются

несколько новых подходов к обучению, аугментация мозаикой, дропаут и потери CIoU [71]. Обобщённая архитектура модели YOLOv4 отображена на рисунке 2.13

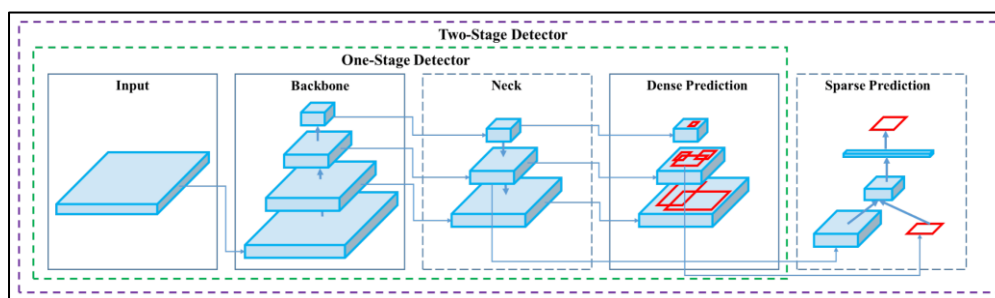


Рисунок 2.13 – Архитектура YOLOv4

YOLOv4 и ее достижения сильно продвинули нейронные сети и привели много новых разработчиков. Некоторое время выпускали новые версии, постепенно улучшая скорость и точность работы. Также через некоторые время образовался проект ultralytics. На данный момент именно этот проект поддерживает основную разработку YOLO. Хотя сама архитектура разрабатывается контрибьюторами со всего мира и любой может предложить новую архитектуру.

Также компанией YOLO разрабатывается YOLO API для работы с нейронными сетями, полностью пряча весь процесс за высокоуровневым интерфейсом. Что позволяет быстро разработать модель для конкретной задачи. Однако и снижает возможности их модификации.

Свое образным прорывом стала разработка модель YOLOv8. Она в большей мере основана на YOLOv4. Тоже использует данные мозаики для обучения. В данной модели пробуются новый подход, без генерации стандартных описывающих квадратов, ищется центр объекта. Что позволяет ускорить обучение и снижается количество прогнозов прямоугольников. Основным блоком является C2f. Эти блоки объединяют данные с нескольких слоев для вычисления признаков объектов разной величины. Также в данной версии различается голов. В данном случае классификация и регрессия выполняются в разных модуля, что ускоряет работу модели. [72]

3 РАЗРАБОТКА И ТЕСТИРОВАНИЕ СИСТЕМЫ

3.1 Проектирование системы

Разрабатываемая система нацелена на циклическую обработку кадров, поступающих с видеокамеры и последующий подсчет количества проплывшей рыбы.

Таким образом можно спроектировать систему, как циклическую. Программа в циклическом режиме обрабатывает каждое фото. Шаги по обработке фото для проведения инференса:

- предобработка изображения;
- инференс;
- постобработка предположений;
- отслеживании объектов;
- подсчет параметров
- вывод изображения на экран;
- подсчет показателей.

Этап предобработки изображений включает в себя изменение размера изображения и преобразование цветовых пространств. Дополнительно данный шаг может включать в себя цветовую предобработку изображений, например восстановление цвета, если камера используется под водой или вырывание контраста для нормализации освещения.

Инференс – это процесс получения прогноза расположения рыбы по входящему изображению. Выходящие из модели данные представлю собой массивы из цифровых значений разного формата.

Из полученных после инференса значений необходимо отобрать предположения достаточно высокой уверенности, преобразовать их данные в подходящий для вывода формат, а также объединить несколько предположений со схожим расположением в кадре в единый объект. Также на этапе пост-постобработки нам необходимо отправить полученные данные в систему отслеживания.

Система отслеживания имея несколько кадров сможет выдвинуть предположения об ассоциации объектов между кадрами. Итоговые ассоциации необходимо обработать. Например, посчитать сколько индивидуальных объектов пересекло центр экрана кадра.

Итоговые данные по объектам необходимо нанести на исходное изображения и вернуть пользователю в графический интерфейс, чтобы пользователь мог вручную убедиться в корректности данных и в случае необходимости настроить необходимые параметры. Параметре прошедшей рыбы также нужно вернуть пользователю и отобразить на экране.

Интерфейс пользователя должен обладать полями и кнопками, необходимыми для настройки работы системы, а также в интерфейсе должно быть предусмотрено место для вывода изображения. Также интерфейс должен включать дополнительные функции: выбор ввода изображения: файл или камера, и кнопку записи вывода в видео файл. Макет интерфейса можно наблюдать на рисунке 3.1

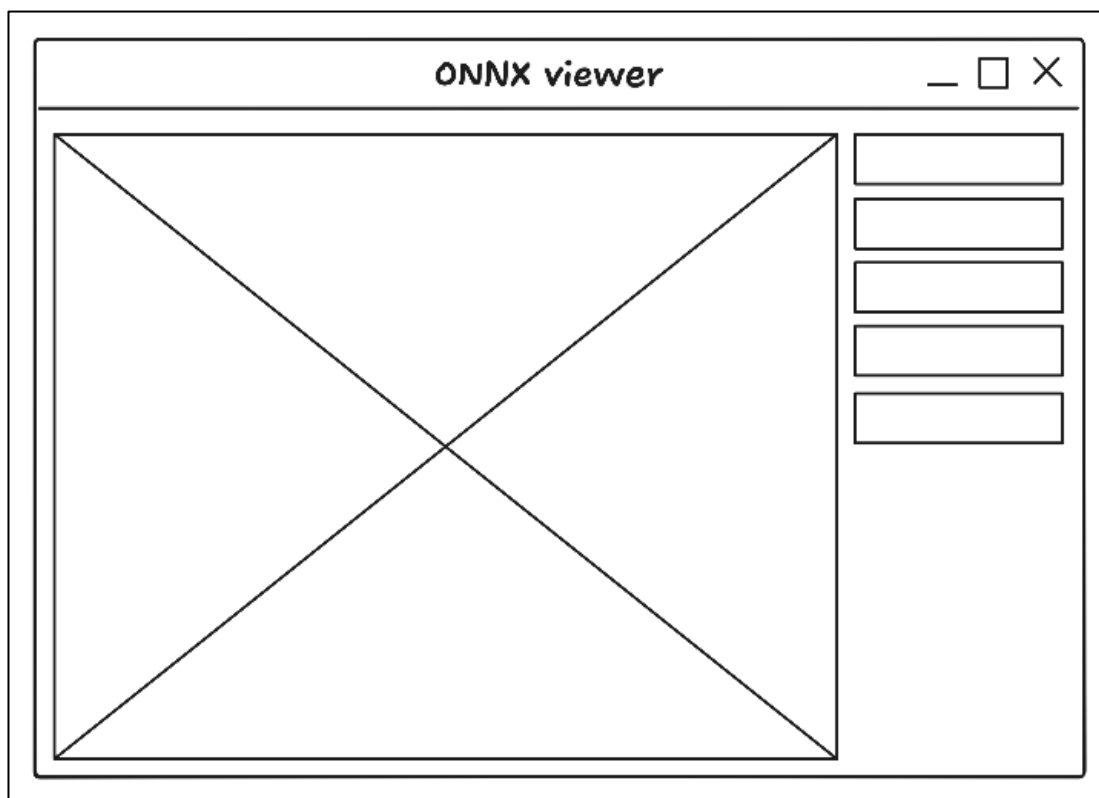


Рисунок 3.1 – Макет интерфейса

3.2 Набор данных и работа с ним

3.2.1 Сбор данных

Разработка модели глубокого обучения начинается с подбора данных. Для выбранной задачи нам необходимы изображения рыб, приоритетно под водой, в чистой и замутненной воде. Изображения должны содержать разное количество рыб, выделять из квадратами равномерно распределяя по кадру.

Было подобрано 9 различных датасетов. В таблице 1.1 представлена сводная информация по ним.

Таблица 1.1 – Сводная информация по наборам данных

№	Название	Тип	Detect	Segment	Class	Local	Countour
1	DeepFish [73]	Фото	-	662	39 770	3 202	-
2	Fishclef 2015 [19]	Видео	-	98	-	-	-
3	Fishnet [74]	Фото	94 806	-	-	-	
4	fishRecognition_GT [75]	Фото	-	27 370	-	-	-
5	LABELED-FISHES-IN-THE-WILD [76]	Фото, Видео	207 кадр.	1 видео	-	-	-
6	Fish detection and tracking [77]	Видео	-	-	-	-	17
7	Underwater Object Detection Dataset [78]	Фото	7 556	-	-	-	-
8	Deep Fish Object Detection [79]	Фото	5 568	-	-	-	-
9	Luderick seagrass [80]	Фото	4 280	-	-	-	4 280

Представленные наборы данных содержат большое количество фотографий, однако многие из них содержат одну рыбу, некоторые сделаны на воздухе, и в некоторых группы рыб объединены единым объектом, а также в наборах данных присутствуют дублирования. Данные факторы могут ухудшить результаты обучения модель.

Проблемы использования картинок с одной рыбой на кадр можно решить с помощью аугментаций – изменений исходных изображений. Аугментация «Мозаика» составляет из нескольких изображений единое по типу коллажа. Это позволит сократить кол-во изображений и распределить квадраты определения рыбы по несколько штук равномерно по кадру. Работа с размытием, яркостью, контрастностью и тоном позволит создать изображений имитирующие загрязнение воды, время суток и погодные условия. Примеры изображений приведены на рисунке 3.2. и рисунке 3.3.



Рисунок 3.2 – Пример изображения из датасета DeepFish



Рисунок 3.3 – Пример изображения из датасета Luderick seagrass

Таким образом после анализа датасетов и принятия решений о его конечном виде можно выделить следующие задачи:

- разрезать видео по кадрам;
- преобразовать xml, csv и txt подписи в единый вид;
- удалить дубликаты;
- переименовать файлы;
- отчистить от дублирования лейблов;
- разбить на обучающую, тестовую и валидационную выборки.

В качестве конечной иерархии файлов в датасете был выбран вид YOLO в следующем распределении, представленном на изображении 3.4.

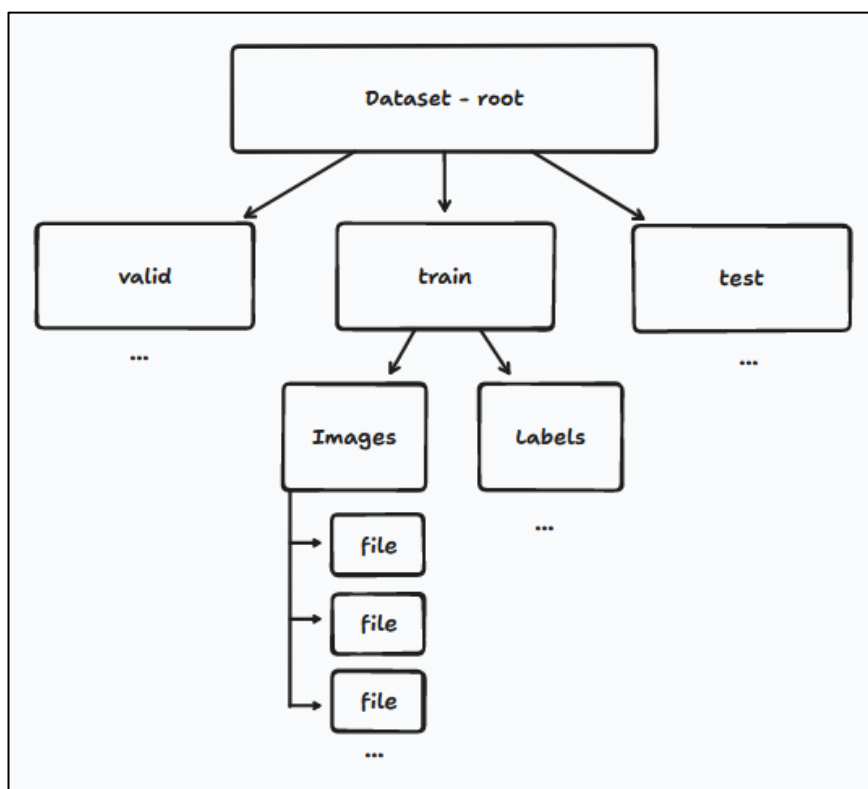


Рисунок 3.4 – Иерархий файлов в наборе данных

Обработка данных будет произведена на языке программирования python. Этот язык позволяет быстро реализовать все необходимые операции с файлами. В качестве первой задачи обработки данных было выбрано разрезать видео на кадры и сохранить отдельно по фреймам. В данном случае для каждого видео существует файл, в котором описаны подписи к отдельным фреймам. Поэтому будут сохранены кадры, которые указаны в файле подписей.

Большинство подписей сохранены с форматом указания x и y , координаты верхнего левого и нижнего правого угла или ширины, высоты объекта. Данный формат необходимо преобразовать в формат пропорций, пригодный для использования семейством YOLO. Данный формат легко реализуется с помощью python и позволяет изменять размер изображения, без изменения его подписи. Пример программы для разрезания видео из датасета LABELED-FISHES-IN-THE-WILD [6] приведено в листинге 3.1.

Листинг 3.1 – Разделение видео по кадрам и преобразование подписи

```
import cv2 as cv
import numpy as np
file = open("./Test_ROV_video_h264_full_marks.dat", "r")
cap = cv.VideoCapture("./Test_ROV_video_h264_full.mp4")
frameNum = 0
if (cap.isOpened() == False):
    print("Some error")
fish = file.readline().split(' ')
fr = fish.pop(0).split('(')[1][3:-1]
width = cap.get(cv.CAP_PROP_FRAME_WIDTH)
height = cap.get(cv.CAP_PROP_FRAME_HEIGHT)
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        if frameNum == int(fr):
            num = int(fish.pop(0))
            writeFile = open("labels/frame%d.txt" % frameNum, 'a+')
            content = ''
            for i in range(num):
                x, y = int(fish.pop(0)), int(fish.pop(0))
                w, h = int(fish.pop(0)), int(fish.pop(0))
                content += '0' + ' ' + \
                    str((x + w/2)/width) + ' ' + \
                    str((y + h/2)/height) + ' ' + \
                    str(w/width) + ' ' + \
                    str(h/height) + '\n'
            writeFile.write(content)
            writeFile.close()
            cv.imwrite("frames/frame%d.jpg" % frameNum, frame)
            fish = file.readline().split(' ')
            fr = fish.pop(0).split('(')[1][3:-1]
            frameNum += 1
            if cv.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
cv.destroyAllWindows()
```

3.2.2 Удаление дубликатов

Удаление файлов дубликатов можно сделать двумя способами: удаление идентичных фото и удаление похожих фото. Удаление идентичных фото достаточно простой подход, реализуется за счет использования функций хэширования изображения и последующего сравнения хэшей. Удаление дубликатов более сложная задача, для ее решения используются модели на слоях трансформеров. Данные модели возвращают входных данные в виде многомерного вектора, где каждое измерение должно представляться как логическое значение содержания изображения.

Полученные многомерные вектора можно сравнить с помощью формул вычисления расстояния между векторами. Чем ближе значения, тем похожей должны быть картинки, в теории. Данный подход весьма сложен в вычислении и занимает продолжительное время. Для датасета состоящего из 120 000 картинок нужно порядка 100 Гб свободного места и около 1000 часов для вычислений на современном CPU. В конечном счете был выбран способ хэширования, которым было удалено 750 дубликатов, пример его использования приведен в листинге 3.2.

Листинг 3.2 – Удаление дубликатов изображений

```
import os
import hashlib
# split_group_list = ['test', 'train', 'valid']
split_group_list = ['images']
dataset_path = '.'
completed = 0
num_images = len(os.listdir(f'{dataset_path}/{split_group_list[0]}'))
for group in split_group_list:
    directory = f'{dataset_path}/{group}'
    hashes = set()
    del_num = 0
    for filename in os.listdir(directory):
        path = os.path.join(directory, filename)
        digest = hashlib.shal(open(path, 'rb').read()).digest()
        if digest not in hashes:
            hashes.add(digest)
        else:
            os.remove(path)
            label_name = filename.replace('.jpg', '.txt')
            label_path = f'{dataset_path}/labels/{label_name}'
            os.remove(label_path)
            del_num += 1
```

Продолжение листинга 3.2 – Удаление дубликатов изображения

```
    complited += 1
    print(f'Выполнено: {complited} / {num_images}', end='\r')
    print(f'{group}: удалено {del_num} изображений')
print('Дубликаты удалены')
```

3.2.3 Унифицирование

После анализа полученных данных были выделены следующие ошибки: дублирование лейблов, запись лейблов подряд и наличие не рыбы в датасете. Для исправления данных ошибок использовался код, приведенный в листинге 3.3.

Листинг 3.3 – Очистка датасета от ошибок подписи

```
import os

dataset_path = '.'
labels = os.listdir('labels')

labels_len = len(labels)
counter_img = 0
counter_dub = 0
counter_over = 0
counter_nf = 0
fin_line = []
to_remove = []

for label in labels:
    label_file = open(f'labels/' + label, 'r')
    lines = label_file.readlines()
    old_len = len(lines)
    lines = list(set(lines))
    new_len = len(lines)
    if new_len < old_len:
        counter_dub += 1
    for line in lines:
        if len(line.split(' ')) > 5:
            fin_line.append(' '.join(line.split(' ')[:5]))
            fin_line.append('0 ' + ' '.join(line.split(' ')[5:]))
            counter_over += 1

        elif line[0] != '0':
            counter_nf += 1
        else:
            fin_line.append(line)
    label_file.close()

    label_file = open(f'labels/' + label, 'w')
    label_file.writelines(fin_line)
```

Продолжение листинга 3.3 – Очистка датасета от ошибок подписи

```
label_file.close()

to_remove = []
fin_line = []
counter_img += 1
print(f'Progress: {counter_img}/{labels_len} over:{counter_over}
nf:{counter_nf}', end='\r')

print(f'Progress: {counter_img}/{labels_len} over:{counter_over}
nf:{counter_nf}')
```

После полной обработки изображений и их подписей их можно объединить в две папки: `images`, `labels` и распределить по выборкам. В формате YOLO распределение на обучающую, тестовую и валидационную выборку принято производить заранее. В некоторых других подходах это делается в процессе непосредственно перед обучением модели. Для распределения данных по выборкам в определенных пропорциях использовалась программа, указанная в листинге 3.4.

Листинг 3.4 – Разбиение данных на выборки

```
import os
from random import shuffle
from shutil import copyfile

split_group = ['train', 'test', 'valid']
split_size = [0.7, 0.2, 0.1]
counter = 0

images = os.listdir('images')
labels = os.listdir('labels')
shuffle(images)

images_len = len(images)
labels_len = len(labels)

train_len = int(images_len * split_size[0])
test_len = int(images_len * split_size[1])
val_len = int(images_len * split_size[2])

train_len += images_len - test_len - val_len - train_len

print('train:', train_len)
print('test:', test_len)
print('val:', val_len)
print('src:', images_len)
print('sum:', train_len + test_len + val_len)
```

Продолжение листинга 3.4 – Разбиение данных на выборки

```
train = images[:train_len]
test = images[train_len:train_len+test_len]
valid = images[train_len+test_len:]
groups = [train, test, valid]

i = 0
for group in split_group:
    for image in groups[i]:
        copyfile(f'images/{image}',
                  f'fishes/{group}/images/{image}')

        copyfile(f'labels/{image.replace(".jpg", ".txt")}',
                  f'fishes/{group}/labels/{image.replace(".jpg",
".txt")}')

        counter += 1
        print('Progress:', counter, '/', images_len, end='\r')

    i += 1
```

Для проверки датасета на правильно преобразования данных была разработана программа, которая перебирает все изображения и рисует на них квадрат подписи согласно указанным данным. Код программы указан в листинге 3.5.

Листинг 3.5 – Код предпросмотра и проверки подписей

```
import cv2
import time
img = cv2.imread('./frames/frame1991.jpg')
fileRead = open('./labels/frame1991.txt')
width = img.shape[1]
height = img.shape[0]
# print(width, height)

for line in fileRead.readlines():
    line = line.split(' ')
    x1, y1 = int(int(float(line[1])*width)-
int(float(line[3])*width)/2), int(int(float(line[2])*height)-
int(float(line[4])*height)/2)
    x2, y2 = int(int(float(line[1])*width)+int(float(line[3])*width)/2),
int(int(float(line[2])*height)+int(float(line[4])*height)/2)
    print((x1, y1),
          (x2, y2))
    img = cv2.rectangle(img,
                        (x1, y1),
                        (x2, y2),
                        (255, 255, 0),
                        3)
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Для итоговой проверки набора данных было выбрано видео с мутной водой и большим количеством рыбы. Данное видео является наиболее сложным и приближенным к реальным условиям использования. Поэтому тестовая модель будет доучена на десятой части изображений из видео в случайном порядке. Кадр из данного видео представлен на рисунке 3.5. Следует утонить, что перед конечным внедрением модели, она также будет доучена на основе реальных данных с места использования.

На основе данных итоговой проверки модели были сделаны следующие выводы:

- в наборе данных представлены фото с выделением косяка рыб в качестве одного объекта;
- в наборе данных в большей мере представлены фотографии, на которых рыба одна и помещена ровно в центр кадра;
- цвет воды имеет слишком большой вес при определении рыбы.



Рисунок 3.5 – Кадр из видео итогового тестирования модели

Разметка производилась в сервисе roboflow. Также одна из первых версий набора данных была выгружена из этого сервиса. Roboflow позволят

пройти полный набор задач по созданию датасета от сбора данных и его разметки, до выгрузки в определенном формате и аугментации. Пример интерфейса разметки данных приведен на рисунке 3.6

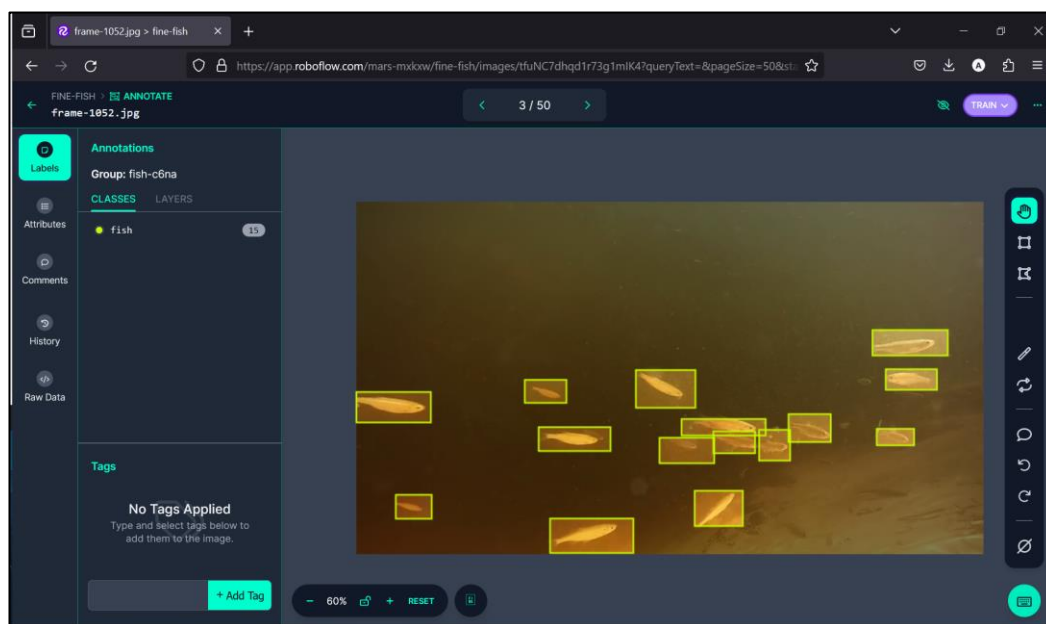


Рисунок 3.6 – Интерфейс разметки данных Roboflow

3.2.4 Аугментации

При обучении тестовой модели были обнаружены ошибки набора данных, возникающие при обучении. Пример приведен на рисунке 3.7. В данном случае в районе 10 эпохи обучения происходит сильный скачок показателей, ухудшающий результаты обучения. Можно предположить, что в данном случае где-то в наборе данных есть аномалии.

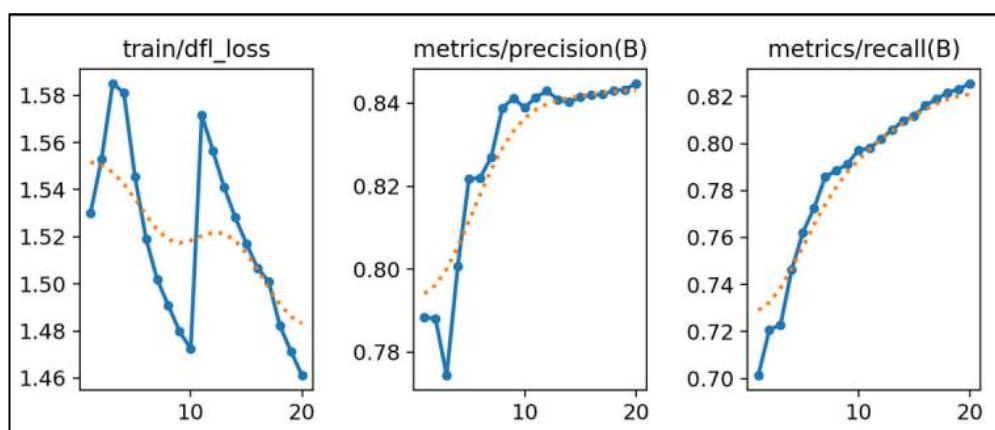


Рисунок 3.7 – Результаты обучения

В рамках визуальной проверки работы нейронной сети, можно заметить, что рыбы в центре кадра определяются значительно лучше, чем по бокам. Это может служить указателем на то, что в обучающем наборе данных доминируют изображения с одним объектом по центру кадра.

Однако при обучении модели использовались различные аугментации, смещающие изображение, конкатенирующие несколько изображений в одно, и другие. Это должно было снизить влияние распределения bbox. Но оказалось недостаточным. Распределение объектов по кадру представлен, но на рисунке 3.8. Можно обратить внимание, что кроме большого количества центрированных изображений, так же изображения часто занимают всю ширину и почти всю высоту кадра. Можно предположить, что это влияние набора данных fishnet.

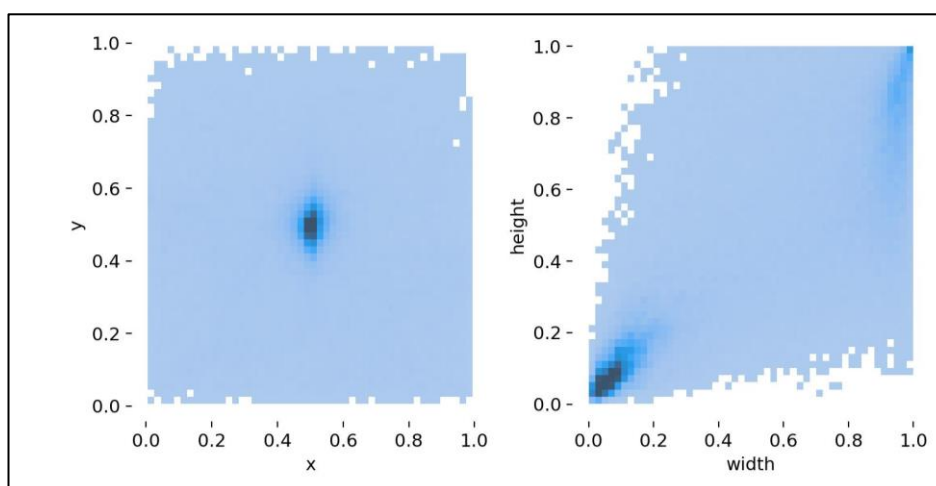


Рисунок 3.8 – Анализ распределения bbox

Обучение набора данных на 10 эпох занимает около 3 часов используя видеокарту GPU P100. Для нашей задачи это значительный временной промежуток. Вследствие ограниченности ресурсов, актуальной задачей будет ускорение обучения, по средствам сжатия датасета.

Учитывая три данных фактора, было принято решение провести предварительную аугментацию данных методом мозаики с 1, 4 и 9 изображениями со случайными параметрами размера каждой фотографии. Работать было принято только с датасетом fishnet, так как в остальных

используемых наборах данных так ярко не прослеживались указанные ошибки.

Сам по себе набор данных fishnet занимает примерно $\frac{2}{3}$ изображений из общего количества. Обработывая фотографии только из этого датасета, мы снизим его влияние и влияние ошибок, представленных в нем на общее обучение. Так же это позволит разнообразить изображения, представленные в одном батче. Исходя из указанных выше параметров планируется заменить 14 исходных фотографий (коллажи по 1, 4 и 9) изображений на 3 новых. Таким образом можно сжать набор данных примерно в 4.67 раза. Что снизит его влияние на набор данных в целом до $\frac{1}{3}$, что все еще очень много, но уже намного лучше.

Для обработки изображений мозаикой была разработана программа на языке программирования python. Данная программа считывает все изображения в указанной директории. Берет из нее случайное количество изображений – 1, 4 или 9. Если было выбрано одно изображения, оно не обрабатывается. Если было выбрано 4 или 9 изображений, то данные изображения преобразуются к квадратные и конкатенируются в новый коллаж. Количество места для каждого изображения выбирается случайным образом до 0.3 до 0.7 по высоте и ширине для 4 изображений, и от 0.25 до 0.40 для 9 изображений. Подписи к данным фотографиям обрабатываются аналогичным образом и тоже объединяются в единый файл. Таким образом из примерно 85 тыс. исходных изображений было получено 19 тыс. объединенных изображений. Полный программный код вы можете наблюдать на листинге A1. Пример получившихся фото вы можете наблюдать на рисунке 3.9.

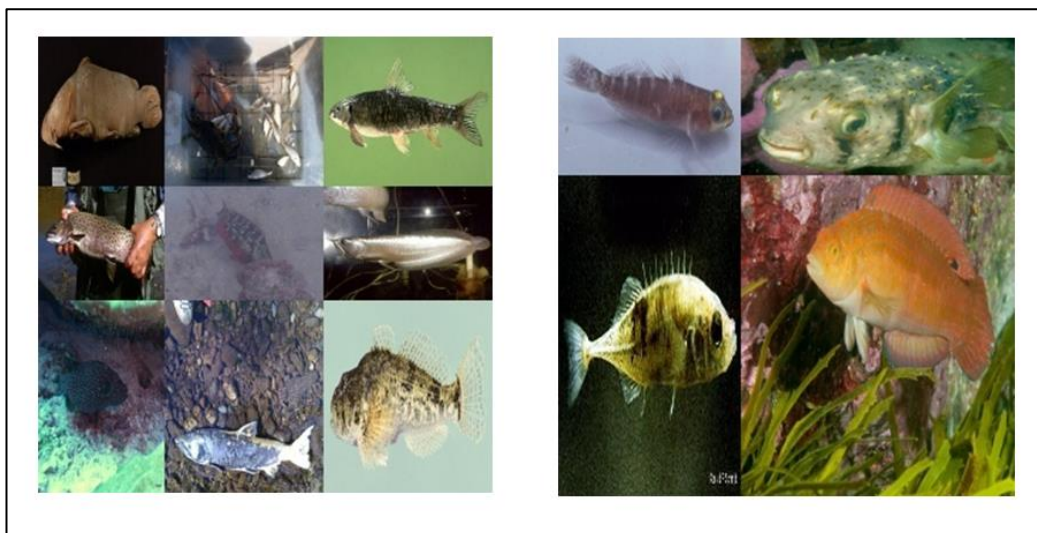


Рисунок 3.9 – Пример аугментированных изображений

Итоговый собранный датасет включает в себя 120 194 изображения, разделенных на train, test и valid группы в пропорция 70%, 20%, 10% соответственно. Также набор данных содержит 9 375 изображений без рыб. Все итоговые изображения были сжаты до разрешения 320 на 320 пикселе, что позволит уменьшить датасет в размере примерно в 3 раза, с 15 гигабайт до 5. Также это позволит увеличить батч и соответственно сократить влияние ошибок разметки на датасет

Полученные изображения имеют разрешение 320 на 320 пикселей. И при этом рыбы остаются достаточно различимыми. Таким образом итоговый набор данных сократился с 15 ГБ до 1.8 ГБ и с 120 тыс. изображений до 45 тыс. При этом количество индивидуальных рыб не изменилось и составило около 162 тыс. объектов. Итоговый собранный датасет разделен на train, test и valid группы в пропорция 70%, 20%, 10% соответственно.

Благодаря сокращению количества изображений практический в 3 раза удалось более подробно рассмотреть изображения набора данных и найти не подходящие. В данном случае примеры изображений представлены на рисунке 3.10. Данный изображения были получены из видео путем покадрового разрезания. Можно предположить, что они появились до начала обработки данных и присутствовали в исходном наборе. Было принято решение удалить данные фотографии.



Рисунок 3.10 – Пример «битых» изображений

После обработки данных была проведена повторная проверка распределения bbox по изображению. Его результаты представлены на рисунке 3.11. Можно наблюдать, что изображения стали гораздо более равномерно распределены по кадру. Также средняя ширина и высота значительно снизились. Обработку набора данных можно считать успешной.

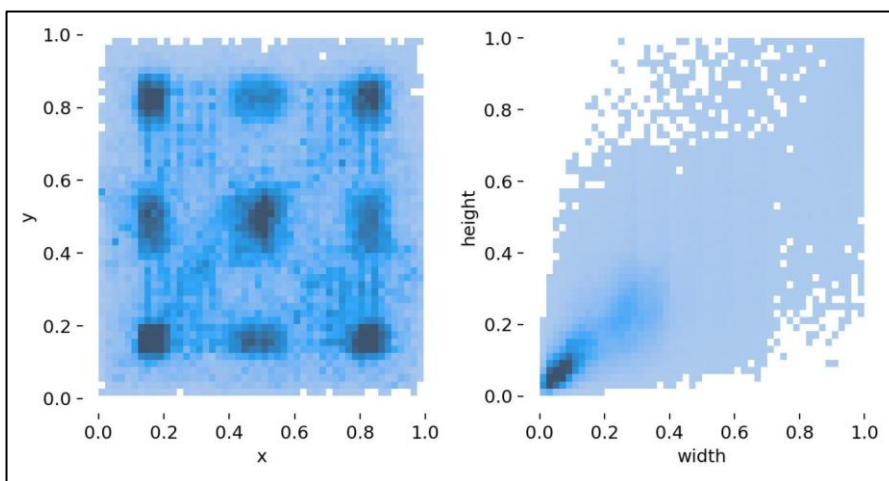


Рисунок 3.11 – Анализ распределения bbox после аугментации

Для улучшения результатов обучения, как было сказано выше, часто используется аугментация изображений. Это позволяет увеличить разнообразие исходных данных. Обычно аугментации проводят непосредственно перед обучением, чтобы не увеличивать набор данных.

Однако мы провели аугментацию «Мозаика» заранее. Это тоже является распространенной практикой.

Среди способов предобработки изображений и аугментации, как было сказано выше, в большей мере используются следующие: HSV, Мозаика, Поворот и Отражение. HSV модель цвета позволяет отделить цвет пикселя в отдельную переменную от насыщенности и яркости. Изменение параметров S и V позволит имитировать условия различной освещенности. В совокупности с этой аугментацией можно использовать изменение экспозиции и контрастности, что также позволит имитировать другие изображения, созданные в других условиях и выработать в модели резистивность к этим условиям. «Мозаика» — это форма объединения изображений в единое целое. Оно позволяет оптимизировать датасет, который создан, например с использованием одной рыбы на фото или с помещением рыб только в центр кадра. Поворот и отражение позволит создать изображения, в которых рыба будет направляться в различных направлениях. Пример работы некоторых аугментаций изображен на рисунке 3.12.



Рисунок 3.12 – Аугментации изображения.

3.3 Разработка модели машинного обучения

После сбора датасета и настройки среды разработки можно перейти к обучению модели. В качестве основной архитектуры для обучения модели была выбрана архитектура YOLOv8. Это современная архитектура обладает наибольшей распространенностью среди моделей детектирования объектов.

В качестве тестирования были обучены 2 модели на двух датасетах. Модели: YOLOv8n и YOLOv8m. Датасеты были в двух видах: черно-белом и цветном. Было выдвинуто предположение, что черно-белый датасет позволит быстрее обучить модель вследствие отбрасывания таких параметров, как блики, тени, загрязнённость воды и прочие.

Результаты обучения, следующие:

- YOLOv8n, Черно-белый – mAP50 0.97739;
- YOLOv8m, Черно-белый – mAP50 0.96389;
- YOLOv8n, Цветной – mAP50 0.98018;
- YOLOv8m, Цветной – mAP50 0.95929.

Все модели обучались с батчем 16 и обучались 100 эпох. В результате наибольшими метриками обладала модель YOLOv8n обученная на цветном датасете. Пример детектирования данной модели представлен на рисунке 3.13

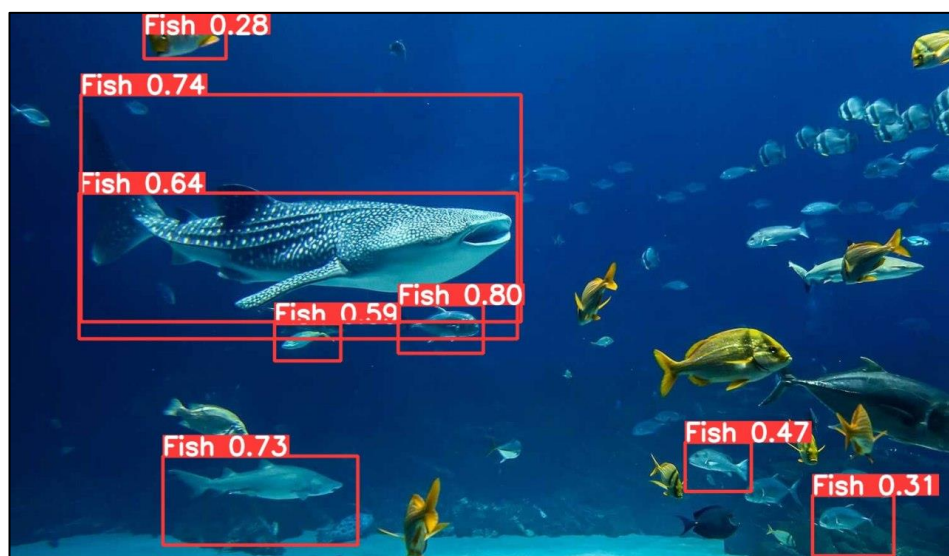


Рисунок 3.13 – Детектирование модели YOLOv8n после обучения

Таким образом предположение об использовании черно-белого датасета оказалось ложным. Данная аугментация не привела к увеличению итоговых метрик. Важно уточнить, что при данном эксперименте использовались не полные версии датасета, так что сравнение метрик с другими экспериментами не допустимо.

Далее необходимо было выделить доминирующую архитектуру, среди семейства YOLO. Было выбрано две наиболее популярных на данный момент версии: 10 и 8. Так как 9 версия вышла буквально на несколько месяцев ранее 10 версии, она не обзавелась популярностью. Наиболее используемой осталась 8 версия, а также она получила релиз 8.2. 10 же версия предлагает принципиально новый подход и в исследованиях показывает значительно лучшие результаты.

В рамках сравнения производительности моделей yolov8n и yolov10n. Было проведено обучение на 100 эпох. 8 версия модели обучилась за 7 часов на видеокарте nvidia P100. В сравнении 10 версия обучилась за 4 часа и получила следующие метрики: AP50 – 0.8737, AP50-95 – 0.5688. Против 0.8649 и 0.5599 у 8 версии соответственно. Скорость детектирования так же незначительно улучшилась. Однако конкретные замеры, не проводились. Таким образом можно сделать логичный вывод, что модель 10 версии является приоритетной.

Модель yolov8n также обучалась на датасете без предварительного аугментирования мозаикой. За 20 эпох, обученных удалось получить лучшие метрики 0.8899 и 0.5934, AP50 и AP50-95. Однако визуальное тестирование на новых данных модель не прошла и косяк рыб стабильно выделялся как единый объект при разном пороговом коэффициенте на тестовом видео. Из чего можно сделать вывод, что использование мозаики положительно сказалось не только на объеме данных, но и на обучаемости.

Для сравнения двух доминирующих архитектур было проведено обучение модели Faster R-CNN. Данная модель обучалась на том же наборе данных. Из-за использования разных фреймворков для обучения аугментации

могли различаться. Анализируя результаты инференса, можно сделать вывод, что в общем случае Faster R-CNN показал себя лучше. Однако в некоторых задачах косяк рыб выделялся единым прямоугольником. Можно предположить, что в наборе данных присутствуют изображения с подобной подписью. Однако, можно в некоторой степени влиять на подобные случаи при использовании итоговой модели за счет параметра объединения прямоугольников.

Время инференса данных моделей примерно различается значительно. Модель Faster R-CNN провела полный анализ изображения за 0,33 секунды. В это время модель YOLOv10 затратила на это 0,09 секунды. Что практически в 3,5 раза быстрее. Учитывая, что одним из требований является работа в режиме реального времени, выгоднее выбрать модель YOLOv10. Учитывая наличие богатой предобработки изображений и постобработки результатов только модель YOLO уложиться в указанные сроки. Для модели Faster R-CNN потребуется улучшение аппаратной части оборудования. Пример результата работы модели Faster R-CNN приведен в первой строке, YOLOv10 во второй строке изображения 3.14.

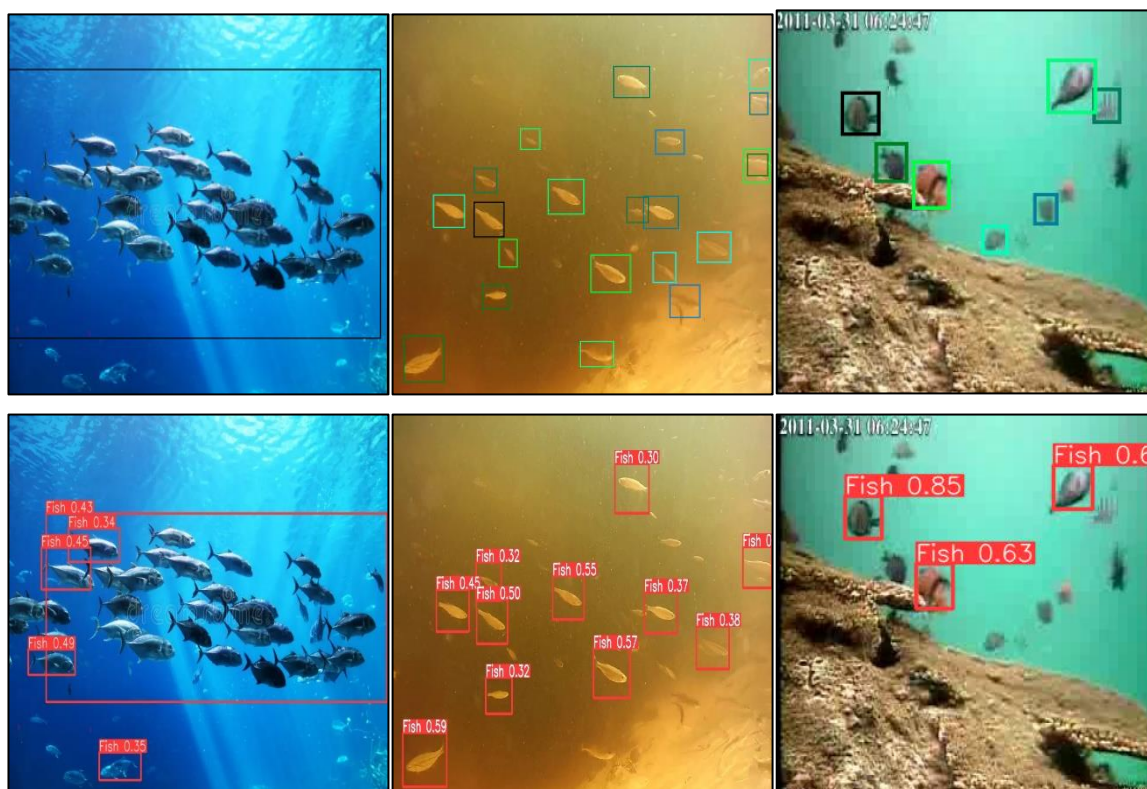


Рисунок 3.14 – Результаты работы нейронных сетей Faster R-CNN и YOLO

Можно предположить, что архитектура R-CNN может лучше справляться в задаче поиска рыбы, так как алгоритм предположения регионов объекта передает лучшие предположения из-за однообразного фона. Полный код обучения модели Faster R-CNN представлен в листинге B1. Код инференса модели Faster R-CNN представлен в листинг Г1.

Таким образом в конечном счете была выбрана архитектура YOLOv10n и обучена с батчем 256 за 250 эпох. По графикам обучения можно сделать вывод, что обучения прошло успешно. Можно заметить сравнительно низкую метрику mAP50-95, что говорит о том, что большинство объектов определяется с низкой вероятностью. Это может говорить как о разнообразии данных, не полной видимости рыб, так и о малом количестве видимых признаков рыб в наборе данных. В некоторых случаях можно сделать вывод, что набор данных плохо размечен или исходные фото плохого качества. Итоговые метрики обучения модели YOLOv10n представлены на рисунке 3.14.

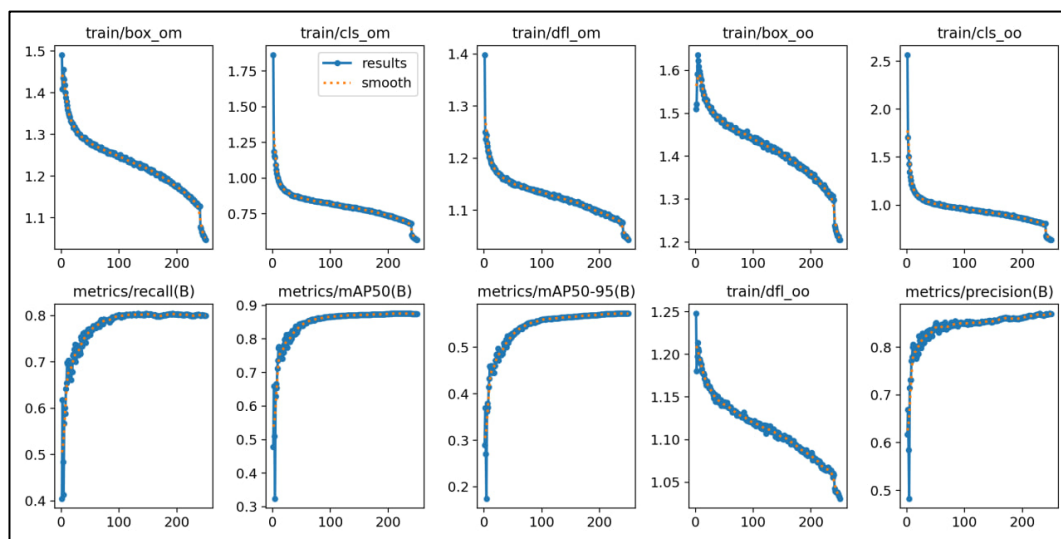


Рисунок 3.14 – Метрики обучения модели YOLOv10n

Дополнительно были обучены несколько моделей семейства YOLO для выбора наилучшей. Каждая модель оценивалась по двум параметрам: время инференса и метрики AP50-95. Результат обучения представлен на рисунке 3.15. Лучшие модели левее и выше.

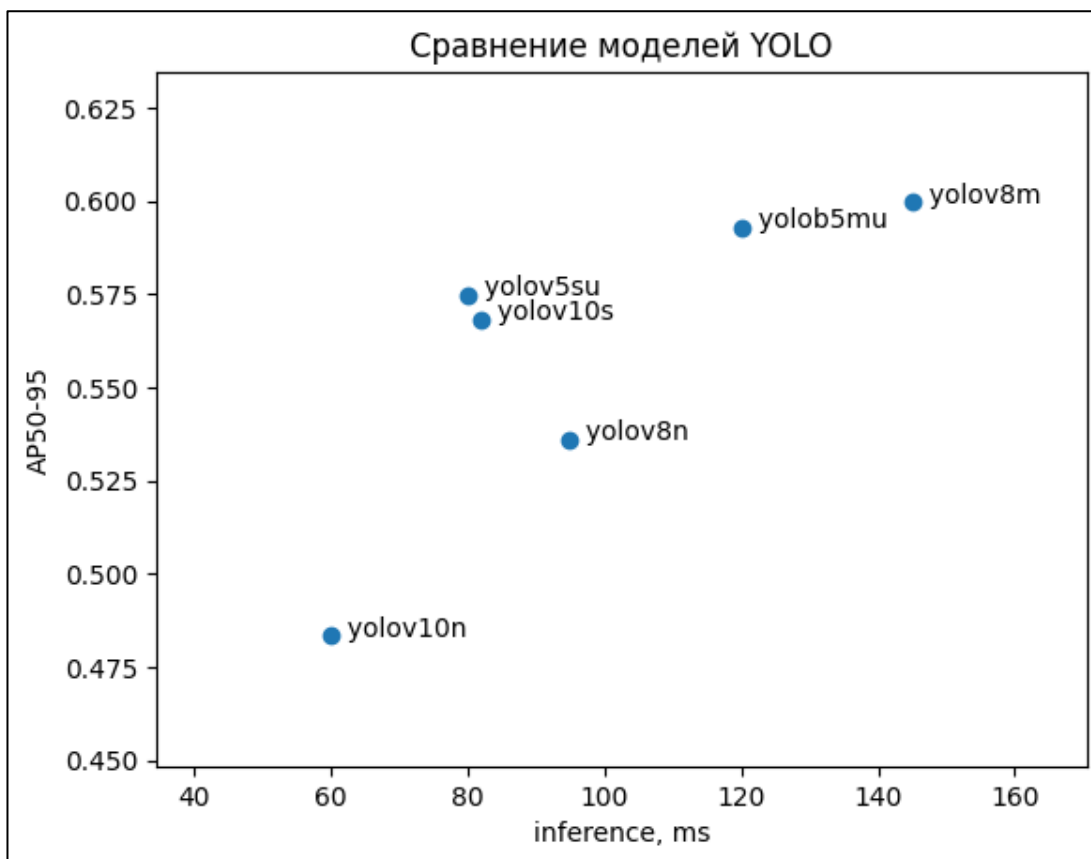


Рисунок 3.15 – Результаты обучения моделей YOLO

3.4 Разработка приложения

Для использования разработанной модели неспециалистом необходимо упростить взаимодействие с моделью. Наиболее простым способом является разработка графического приложения. Разрабатываемое приложение должно обладать следующим функционалом:

- иметь возможность получать входные данные из видео и с подключенной видеокамеры;
- выводить итоговое изображение с размеченными рыбами на экран пользователя;
- иметь возможность записи выводного изображения в файл;
- редактирование первичных параметров работы модели;
- считать количество уникальных рыб, прошедших через центр кадра.

Этапы работы приложения можно условно разделить на шесть шагов:

- предобработка изображения;

- детектирование рыб;
- постобработка данных;
- отслеживание рыбы;
- детектирование пересечения центра;
- вывод данных пользователю.

В первом шаге необходимо входное изображение трансформировать таким же образом, как обучающие данные, в данном случае это означает перевести его в тензор и сжать до размера 320 на 320 пикселей.

Детектирование рыбы – это собственно получение сырых данных из модели глубокого обучения. В разрабатываемом приложении будет использоваться API ONNXRUNTIME. Это библиотека для запуска моделей в формате onnx. Данный формат широко распространен и позволит экспортировать многие другие виды моделей.

Постобработка будет состоять из преобразования вида квадрата выделения рыбы в формат, подходящий для модуля opencv. Этот модуль будет осуществлять основную работу по обработке изображения.

После получения координат квадрата выделения рыбы можно загрузить эти данные в модель трекинга. Данная модель позволит объединять объекты с разных кадров единой сущностью и таким образом отслеживать перемещение конкретной рыбы. За это будет отвечать модель DeepSort.

Имея список уникальных объектов, можно отслеживать их перемещение через центр кадра и обратно. Таким образом можно грубо посчитать количество рыбы, прошедшей через место съемки в одну и в другую сторону.

В конечном счете нам необходимо вывести эти данные пользователю. За отрисовку квадратов выделения будет отвечать модуль opencv в реализации для языка python. Для разработки графического интерфейса использовался модуль PyQt. Он достаточно распространен и прост в обращении, а также позволяет реализовать весь необходимый функционал. Макет интерфейса изображен на рисунке 3.15. Основной программный код приложения содержится в листинге Б1.



Рисунок 3.15 – Финальный интерфейс приложения

3.5 Тестирование приложения

Итоговое тестирование проводилось на модели yolov5su, как самой перспективной для конечного использования.

В результате тестирования разработанная программа проводила полную обработку одного кадра изображения в самой высоконагруженной сцене за 142 миллисекунды, что составляет примерно 7 кадров в секунды. Вычисления производились на ЦП Intel Core i5 12450H, который обладает примерно 42.4 GFLOPS . вследствие чего можно уверенно сказать, что ПК оснащенный среднее видеокартой с мощностью более 10 TFLOPS сможет обеспечить вычисления в режиме реального времени. Стоит отметить, что в дальнейшем в системе планируется добавление второй камеры и развитие системы с помощью 3D реконструкции, запас мощности является оправданным.

На тестовом видео система смогла отследить перемещение 52 рыб из 56, что означает погрешность около 8%. Данный результат можно считать удовлетворительным.

ЗАКЛЮЧЕНИЕ

В результате работы над магистерской диссертацией на тему «Разработка программного средства для подсчёта и анализа проходящей рыбы в реках Архангельской области на основе компьютерного зрения», было проведено исследование текущего состояния систем мониторинга прохода рыбы. Были изучены основные исследования на тему разработки систем компьютерного зрения для отслеживания рыбы. Также были изучены современные архитектуры глубоких нейронных сетей и возможности их использования в сфере разработки системы отслеживания рыбы.

Основываясь на анализе предметной области, были выделены требования для конечной системы: возможность работы с частотой кадров выше 30, отклонение данных сравнимое с человеческой ошибкой.

Для достижения поставленной цели был проведен анализ открытых источников наборов данных обнаружения рыбы. Полученные изображения были собраны и обработаны для использования в обучении нейронной сети.

На основе полученного набора данных был обучены наиболее перспективные модели нейронных сетей глубокого обучения. Исходя из результатов первичного обучения были сделаны выводы об ошибках в наборе данных. Набор данных был несколько раз доработан для достижения наилучшего результата.

На основе обученных моделей было разработано графическое приложение для использования модели на постоянной основе неспециалистом в сфере нейронных сетей. Разработанная программа имеет первичные настройки для адаптирования работы нейронной сети в конкретных условиях.

В дальнейшем при использовании полученной системы необходимо провести до обучение исходной модели на данных с места использования. При размещении камеры подводой есть возможность воспользоваться моделями восстановления цвета, что увеличит количества признаков.

Также существует возможность переноса обучения нейронной сети на дополнительную задачу классификации для определения вида рыбы и задачу сегментации.

В случае развития системы и дополнения ее сенсорами расстояния до объекта или стереозрением существует возможность оценки размеров проводящей рыбы. Для качественной оценки проходящей рыбы необходимо дополнить существующую систему детектирования, системой сегментации.

Резюмируя, можно сказать, что разрабатываемая система готова к внедрению и наладке на месте использования. Поставленные задачи в рамках работы выполнены, цели достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Automated Detection, Classification and Counting of Fish in Fish Passages With Deep Learning [Электронный ресурс]:[официальный сайт]/ Frontiers – Электрон. дан. - [2006–2024]. - Режим доступа: <https://www.frontiersin.org/articles/10.3389/fmars.2021.823173/full>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

2. A comparison of an RGB-D cameras performance and a stereo camera in relation to object recognition and spatial position determination [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. -[2008-2024]. - Режим доступа: https://www.researchgate.net/publication/350163097_A_comparison_of_an_RGB-D_cameras_performance_and_a_stereo_camera_in_relation_to_object_recognition_and_spatial_position_determination, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

3. Applications of deep learning in fish habitat monitoring: A tutorial and survey [Электронный ресурс]:[официальный сайт]/ ScienceDirect – Электрон. дан. - [1997–2024]. - Режим доступа: <https://www.sciencedirect.com/science/article/pii/S0957417423023436>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

4. Временной метод определения численности рыб в рыбопромысловых реках [Электронный ресурс]:[официальный сайт]/ Cyberleninka – Электрон. дан. - [2011–2024]. - Режим доступа: <https://cyberleninka.ru/article/n/vremennoy-metod-opredeleniya-chislennosti-ryb-v-rybopromyslovyh-rekakh>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

5. A multitask model for realtime fish detection and segmentation based on YOLOv5 [Электронный ресурс]:[официальный сайт]/ PeerJ – Электрон. дан. - [2012-2024]. - Режим доступа: <https://peerj.com/articles/cs-1262/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

6. Fish Tracking Using Acoustical and Optical Data Fusion in Underwater Environment [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон.

дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/321956687_Fish_Tracking_Using_Acoustical_and_Optical_Data_Fusion_in_Underwater_Environment, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

7. Технология переработки рыбы [Электронный ресурс]:[официальный сайт]/ UComplex – Электрон. дан. - [2013–2024]. - Режим доступа: <https://storage.ucomplex.org/files/books/2572/publication.pdf>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

8. Новости [Электронный ресурс]:[официальный сайт]/ Всероссийский научно-исследовательский институт рыбного хозяйства и океанографии – Электрон. дан. - [2000–2024]. - Режим доступа: <http://vniro.ru>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

9. Правительство Архангельской области [Электронный ресурс]:[официальный сайт]/ Рыбная промышленность Архангельской области – Электрон. дан. - [1998–2024]. - Режим доступа: https://dvinaland.ru/economics/fish_industry/, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

10. Образ жизни и поведение промысловых рыб [Электронный ресурс]:[официальный сайт]/ ВНИРО – Электрон. дан. - [2000–2024]. - Режим доступа: http://www.vniro.ru/files/publish/jarjombek_obraz_jizni_ryb.pdf, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

11. Can Fish See Color? Unhooking the Mystery for Anglers [Электронный ресурс]:[официальный сайт]/ Reel Coquina Crew – Электрон. дан. - [2019–2024]. - Режим доступа: <https://www.reelcoquinafishing.com/blogs/florida-fishing-blog/can-fish-see-color>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

12. Why can fishing lights only catch phototaxis fish? [Электронный ресурс]:[официальный сайт]/ MECREE – Электрон. дан. - [2014–2024]. - Режим доступа: <https://www.mecreeled.com/can-fishing-lights-catch-phototaxis-fish/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

13. Application of Machine Learning in Vegetable Classification and Recognition [Электронный ресурс]:[офиц. сайт]/ ResearchGate – Электрон. дан. - [2008–2024]. - Режим доступа: https://www.researchgate.net/publication/379530914_Application_of_Machine_Learning_in_Vegetable_Classification_and_Recognition, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

14. Artificial Intelligence-Based Detection of Pneumonia in Chest Radiographs [Электронный ресурс]:[офиц. сайт]/ NLM database – Электрон. дан. - [1999–2024]. - Режим доступа: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9221818/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

15. ImageNet Classification with Deep Convolutional Neural Networks [Электронный ресурс]:[офиц. сайт]/ NVIDIA – Электрон. дан. - [1993–2024]. - Режим доступа: %Ссылка%, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

16. Can Artificial Intelligence Identify Pictures Better than Humans? [Электронный ресурс]:[офиц. сайт]/ Entrepreneur – Электрон. дан. - [1994–2024]. - Режим доступа: <https://www.entrepreneur.com/science-technology/can-artificial-intelligence-identify-pictures-better-than/283990>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

17. Нейросеть определяет пол и возраст человека по следу от обуви [Электронный ресурс]:[офиц. сайт]/ NeuroHive – Электрон. дан. - [2018–2024]. - Режим доступа: %Ссылка%, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

18. Brief project overview [Электронный ресурс]:[офиц. сайт]/ Fsh4Knowledge – Электрон. дан. - [1996–2024]. - Режим доступа: <https://homepages.inf.ed.ac.uk/rbf/fish4knowledge/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

19. fishklef [Электронный ресурс]:[официальный сайт]/ lifeclef – Электрон. дан. - [2007–2024]. - Режим доступа: <https://www.imageclef.org/lifeclef/2015/fish>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

20. An effective and robust method for tracking multiple fish in video image based on fish head detection [Электронный ресурс]:[официальный сайт]/ BMC Bioinformatics – Электрон. дан. - [1999–2024]. - Режим доступа: <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-016-1138-y>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

21. Temperate fish detection and classification: a deep learning based approach [Электронный ресурс]:[официальный сайт]/ Springer Link – Электрон. дан. - [2009–2024]. - Режим доступа: <https://link.springer.com/article/10.1007/s10489-020-02154-9>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

22. Unsupervised Fish Trajectory Tracking and Segmentation [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: <https://arxiv.org/pdf/2208.10662>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

23. A Survey of Fish Tracking Techniques Based on Computer Vision [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2110.02551>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

24. Automatic fish detection in underwater videos by a deep neural network-based hybrid motion learning system [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/331638173_Automatic_fish_detection_in_underwater_videos_by_a_deep_neural_network-based_hybrid_motion_learning_system, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

25. YOLO-Fish: A robust fish detection model to detect fish in realistic underwater environment [Электронный ресурс]:[официальный сайт]/ ScienceDirect – Электрон. дан. - [1997–2024]. - Режим доступа:

<https://www.sciencedirect.com/science/article/pii/S1574954122002977>,

свободный (дата обращения : 10.06.2024). - Загл. с экрана.

26. YOLO-Based Fish Detection in Underwater Environments [Электронный ресурс]:[официальный сайт]/ MDPI – Электрон. дан. - [1996-2024]. - Режим доступа: <https://www.mdpi.com/2673-4931/29/1/44>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

27. Underwater Fish Detection using Deep Learning for Water Power Applications [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1811.01494>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

28. Automatic single fish detection with a commercial echosounder using YOLO v5 and its application for echosounder calibration [Электронный ресурс]:[официальный сайт]/ Frontiers – Электрон. дан. - [2006–2024]. - Режим доступа: <https://www.frontiersin.org/articles/10.3389/fmars.2023.1162064/full>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

29. YOLOv8 based fish detection and classification on fishnet dataset [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/380805365_YOLOv8_based_fish_detection_and_classification_on_fishnet_dataset, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

30. YOLOv10: Real-Time End-to-End Object Detection [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://www.arxiv.org/pdf/2405.14458>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

31. Faster R-CNN based Fish Detector for Smart Aquaculture System [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/359271669_Faster_R-

CNN_based_Fish_Detector_for_Smart_Aquaculture_System, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

32. Сравнение yolo v5 и faster r-cnn для обнаружения людей на изображении в потоковом режиме [Электронный ресурс]:[официальный сайт]/ Cyberleninka – Электрон. дан. - [2011–2024]. - Режим доступа: <https://cyberleninka.ru/article/n/sravnenie-yolo-v5-i-faster-r-cnn-dlya-obnaruzheniya-lyudey-na-izobrazhenii-v-potokovom-rezhime>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

33. Transformer-based Self-Supervised Fish Segmentation in Underwater Videos [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2206.05390>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

34. MSR-YOLO: Method to Enhance Fish Detection and Tracking in Fish Farms [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/340636645_MSR-YOLO_Method_to_Enhance_Fish_Detection_and_Tracking_in_Fish_Farms, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

35. The Caltech Fish Counting Dataset: A Benchmark for Multiple-Object Tracking and Counting [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2207.09295>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

36. FishMOT: A Simple and Effective Method for Fish Tracking Based on IoU Matching [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2309.02975>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

37. Underwater Image Restoration via Adaptive Color Correction and Contrast Enhancement Fusion [Электронный ресурс]:[официальный сайт]/ MDPI – Электрон. дан. - [1996-2024]. - Режим доступа: <https://www.mdpi.com/2072-4292/15/19/4699>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

38. Enhancing underwater image via adaptive color and contrast enhancement, and denoising [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2104.01073>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

39. Underwater color restoration and dehazing based on deep neural network [Электронный ресурс]:[официальный сайт]/ ResearchGate – Электрон. дан. - [2008-2024]. - Режим доступа: https://www.researchgate.net/publication/359911793_Underwater_color_restoration_and_dehazing_based_on_deep_neural_network, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

40. Нейронные сети на Python: как всё устроено [Электронный ресурс]:[официальный сайт]/ GeekBrains – Электрон. дан. - [2000-2024]. - Режим доступа: https://gb.ru/geek_university/developer?from=blog_vrezka_finish, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

41. TIOBE Index for June 2024 [Электронный ресурс]:[официальный сайт]/ TIOBE – Электрон. дан. - [2000–2024]. - Режим доступа: <https://www.tiobe.com/tiobe-index/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

42. MoJo: убийца Python и будущее AI [Электронный ресурс]:[официальный сайт]/ хабр – Электрон. дан. - [2006–2024]. - Режим доступа: <https://habr.com/ru/companies/raft/articles/808517/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

43. Github Compare [Электронный ресурс]:[официальный сайт]/ Github Compare – Электрон. дан. - [2019–2024]. - Режим доступа: <https://www.githubcompare.com/pytorch/pytorch+keras-team/keras+tensorflow/tensorflow>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

44. Get Started [Электронный ресурс]:[официальный сайт]/ PyTorch – Электрон. дан. - [2016–2024]. - Режим доступа: <https://pytorch.org/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

45. ultralytics [Электронный ресурс]:[официальный сайт]/ github – Электрон. дан. - [2007–2024]. - Режим доступа: <https://github.com/ultralytics/ultralytics>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

46. OpenCV modules [Электронный ресурс]:[официальный сайт]/ OpenCV – Электрон. дан. - [2005–2024]. - Режим доступа: <https://docs.opencv.org/4.10.0/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

47. Jupyter Notebook Documentation [Электронный ресурс]:[официальный сайт]/ Jupyter – Электрон. дан. - [2014–2024]. - Режим доступа: <https://jupyter-notebook.readthedocs.io/en/latest/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

48. Quick Start Guide for Python in VS Code [Электронный ресурс]:[официальный сайт]/ VS Code – Электрон. дан. - [1997–2024]. - Режим доступа: <https://code.visualstudio.com/docs/python/python-quick-start>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

49. Jupyter Notebooks in VS Code [Электронный ресурс]:[официальный сайт]/ VS Code – Электрон. дан. - [1997–2024]. - Режим доступа: <https://code.visualstudio.com/docs/datascience/jupyter-notebooks>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

50. PyQt [Электронный ресурс]:[официальный сайт]/ python – Электрон. дан. - [1995–2024]. - Режим доступа: <https://wiki.python.org/moin/PyQt>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

51. Display OpenCv camera on a PyQt app [Электронный ресурс]:[официальный сайт]/ Medium – Электрон. дан. - [1998–2024]. - Режим доступа: <https://medium.com/@ilias.info.tel/display-opencv-camera-on-a-pyqt-app-4465398546f7>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

52. A logical calculus of the ideas immanent in nervous activity [Электронный ресурс]:[официальный сайт]/ Carnegie Mellon University – Электрон. дан. - [1985–2024]. - Режим доступа: <https://www.cs.cmu.edu/~epxing/Class/10715/reading/McCulloch.and.Pitts.pdf>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

53. Rapid Object Detection using a Boosted Cascade of Simpl Features [Электронный ресурс]:[офиц. сайт]/ Carnegie Mellon University – Электрон. дан. - [1985–2024]. - Режим доступа: <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

54. A Short Introduction to Boosting [Электронный ресурс]:[офиц. сайт]/ Internic – Электрон. дан. - [2000–2024]. - Режим доступа: <https://www.site.uottawa.ca/~stan/csi5387/boost-tut-ppr.pdf>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

55. Rich feature hierarchies for accurate object detection and semantic segmentation [Электронный ресурс]:[офиц. сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1311.2524>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

56. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Электронный ресурс]:[офиц. сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1506.01497>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

57. YOLOv4: Optimal Speed and Accuracy of Object Detection [Электронный ресурс]:[офиц. сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2004.10934>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

58. SSD: Single Shot MultiBox Detector [Электронный ресурс]:[офиц. сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1512.02325>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

59. Focal Loss for Dense Object Detection [Электронный ресурс]:[офиц. сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1708.02002>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

60. yolov10 [Электронный ресурс]:[официальный сайт]/ github – Электрон. дан. - [2007–2024]. - Режим доступа: <https://github.com/THU-MIG/yolov10>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

61. Temperate Fish Detection and Classification: a Deep Learning based Approach [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2005.07518>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

62. Simple online and realtime tracking with a deep association metric [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1703.07402>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

63. Activity Segmentation and Fish Tracking From Sonar Videos by Combining Artifacts Filtering and a Kalman Approach[Электронный ресурс]:[официальный сайт]/ IEEE – Электрон. дан. - [1989-2024]. - Режим доступа: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10179857>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

64. Motion stereo at sea: Dense 3D reconstruction from image sequences monitoring conveyor systems on board fishing vessels [Электронный ресурс]:[официальный сайт]/ wiley – Электрон. дан. - [1994–2024]. - Режим доступа: <https://ietresearch.onlinelibrary.wiley.com/doi/epdf/10.1049/ipr2.12636>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

65. Intelligent Underwater Stereo Camera Design for Fish Metric Estimation Using Reliable Object Matching [Электронный ресурс]:[официальный сайт]/ IEEE – Электрон. дан. - [1989-2024]. - Режим доступа: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9804493>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

66. End-to-End Object Detection with Transformers [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2005.12872>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

67. FER-YOLO-Mamba: Facial Expression Detection and Classification Based on Selective State Space [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/2405.01828>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

68. Fast R-CNN [Электронный ресурс]:[официальный сайт]/ arXiv – Электрон. дан. - [1998–2024]. - Режим доступа: <https://arxiv.org/pdf/1504.08083>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

69. Mask R-CNN: архитектура современной нейронной сети для сегментации объектов на изображениях [Электронный ресурс]:[официальный сайт]/ хабр – Электрон. дан. - [2006–2024]. - Режим доступа: <https://habr.com/ru/articles/421299/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

70. A Guide to YOLOv8 in 2024 [Электронный ресурс]:[официальный сайт]/ viso.ai – Электрон. дан. - [2018–2024]. - Режим доступа: <https://viso.ai/deep-learning/yolov8-guide/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

71. YOLOv4: высокоскоростное и точное обнаружение объектов [Электронный ресурс]:[официальный сайт]/ Ultralytics – Электрон. дан. - [2014–2024]. - Режим доступа: <https://docs.ultralytics.com/ru/models/yolov4/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

72. Ultralytics YOLOv8 [Электронный ресурс]:[официальный сайт]/ хабр – Электрон. дан. - [2006–2024]. - Режим доступа: <https://habr.com/ru/articles/710016/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

73. An Underwater Fish Species Image Dataset for Deep Learning [Электронный ресурс]:[официальный сайт]/ github – Электрон. дан. - [2007–2024]. - Режим доступа: <https://alzayats.github.io/DeepFish/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

74. FishNet: A Large-scale Dataset and Benchmark for Fish Recognition, Detection, and Functional Traits Prediction [Электронный ресурс]:[официальный сайт]/

github – Электрон. дан. - [2007–2024]. - Режим доступа: <https://fishnet-2023.github.io/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

75. Fish Recognition Ground-Truth data [Электронный ресурс]:[официальный сайт]/ Fsh4Knowledge – Электрон. дан. - [1996–2024]. - Режим доступа: <https://homepages.inf.ed.ac.uk/rbf/fish4knowledge/GROUNDTRUTH/RECOG/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

76. Labeled Fishes in the Wild [Электронный ресурс]:[официальный сайт]/ NOAA – Электрон. дан. - [1997-2024]. - Режим доступа: <https://www.fisheries.noaa.gov/west-coast/science-data/labeled-fishes-wild>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

77. Fish Trajectory Ground Truth Dataset [Электронный ресурс]:[официальный сайт]/ Fsh4Knowledge – Электрон. дан. - [1996–2024]. - Режим доступа: <https://homepages.inf.ed.ac.uk/rbf/fish4knowledge/GROUNDTRUTH/BEHAVIOR/>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

78. Underwater Object Detection Dataset [Электронный ресурс]:[официальный сайт]/ kaggle – Электрон. дан. - [2009–2024]. - Режим доступа: <https://www.kaggle.com/datasets/slavkoprytula/aquarium-data-cots>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

79. Deep Fish Object Detection [Электронный ресурс]:[официальный сайт]/ kaggle – Электрон. дан. - [2009–2024]. - Режим доступа: <https://www.kaggle.com/datasets/vencerlanz09/deep-fish-object-detection>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

80. dataset-fish-detection-low-visibility [Электронный ресурс]:[официальный сайт]/ github – Электрон. дан. - [2007–2024]. - Режим доступа: <https://github.com/slopezmarcano/dataset-fish-detection-low-visibility>, свободный (дата обращения : 10.06.2024). - Загл. с экрана.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы аугментации «Мозаика»

Листинг А1 – Программный код для обработки изображений аугментацией «Мозаика»

```
# -*- coding: utf-8 -*-
#Developed by Alexander Kraynikov
#Email: krajnikov.a@edu.narfu.ru

import cv2
import os
import random
from shutil import copyfile

images_dir = 'old_images'
labels_dir = 'old_labels'
output_images_dir = 'images'
output_labels_dir = 'labels'

image_files = os.listdir(images_dir)
label_files = os.listdir(labels_dir)

num = [1, 4, 9]

total_images = len(image_files)
processed = 0
print('total images: ', total_images)

def label_scale(label, w, h, aw=0, ah=0):
    text = ''
    lbl_text = open(os.path.join(labels_dir,
label), 'r+').readlines()
```

```
for line in lbl_text:
    line = line.split(' ')

    out_text = ['0',
                str(float(line[1]) * w + aw),
                str(float(line[2]) * h + ah),
                str(float(line[3]) * w),
                str(float(line[4]) * h)]

    text += ' '.join(out_text) + '\n'

return text

def mosaic4(images, labels):
    salt = random.randint(3, 7) * 320
    w = [320 - int(salt/10), int(salt/10)]
    salt = random.randint(3, 7) * 320
    h = [320 - int(salt/10), int(salt/10)]

    img1 = cv2.resize(images[0], (w[0], h[0]),
interpolation = cv2.INTER_AREA)
    img2 = cv2.resize(images[1], (w[1], h[0]),
interpolation = cv2.INTER_AREA)
    img3 = cv2.resize(images[2], (w[0], h[1]),
interpolation = cv2.INTER_AREA)
    img4 = cv2.resize(images[3], (w[1], h[1]),
interpolation = cv2.INTER_AREA)
```

Продолжение листинга А1 – Программный код для обработки изображений аугментацией «Мозаика»

```

    label      = label_scale(labels[0],  w[0]/320,
h[0]/320)
    label  +=  label_scale(labels[1],  w[1]/320,
h[0]/320, w[0]/320)
    label  +=  label_scale(labels[2],  w[0]/320,
h[1]/320, 0, h[0]/320)
    label  +=  label_scale(labels[3],  w[1]/320,
h[1]/320, w[0]/320, h[0]/320)

    layout = [[img1, img2],
               [img3, img4]]

    image = cv2.vconcat([cv2.hconcat(line) for line
in layout])

    return image, label

def mosaic9(images, labels):
    salt1 = random.randint(25, 40) * 320
    salt2 = random.randint(25, 40) * 320
    w = [int(salt1/100), int(salt2/100), 320 -
int(salt2/100) - int(salt1/100)]
    salt1 = random.randint(25, 40) * 320
    salt2 = random.randint(25, 40) * 320
    h = [int(salt1/100), int(salt2/100), 320 -
int(salt2/100) - int(salt1/100)]

    img1 = cv2.resize(images[0],  (w[0],  h[0]),
interpolation = cv2.INTER_AREA)
    img2 = cv2.resize(images[1],  (w[1],  h[0]),
interpolation = cv2.INTER_AREA)

```

```

    img3 = cv2.resize(images[2],  (w[2],  h[0]),
interpolation = cv2.INTER_AREA)
    img4 = cv2.resize(images[3],  (w[0],  h[1]),
interpolation = cv2.INTER_AREA)
    img5 = cv2.resize(images[4],  (w[1],  h[1]),
interpolation = cv2.INTER_AREA)
    img6 = cv2.resize(images[5],  (w[2],  h[1]),
interpolation = cv2.INTER_AREA)
    img7 = cv2.resize(images[6],  (w[0],  h[2]),
interpolation = cv2.INTER_AREA)
    img8 = cv2.resize(images[7],  (w[1],  h[2]),
interpolation = cv2.INTER_AREA)
    img9 = cv2.resize(images[8],  (w[2],  h[2]),
interpolation = cv2.INTER_AREA)

    label      = label_scale(labels[0],  w[0]/320,
h[0]/320)
    label  +=  label_scale(labels[1],  w[1]/320,
h[0]/320, w[0]/320)
    label  +=  label_scale(labels[2],  w[2]/320,
h[0]/320, w[0]/320+w[1]/320)
    label  +=  label_scale(labels[3],  w[0]/320,
h[1]/320, 0, h[0]/320)
    label  +=  label_scale(labels[4],  w[1]/320,
h[1]/320, w[0]/320, h[0]/320)
    label  +=  label_scale(labels[5],  w[2]/320,
h[1]/320, w[0]/320+w[1]/320, h[0]/320)
    label  +=  label_scale(labels[6],  w[0]/320,
h[2]/320, 0, h[0]/320+h[1]/320)
    label  +=  label_scale(labels[7],  w[1]/320,
h[2]/320, w[0]/320, h[0]/320+h[1]/320)

```

Продолжение листинга А1 – Программный код для обработки изображений аугментацией «Мозаика»

```

    label += label_scale(labels[8], w[2]/320,
h[2]/320, w[0]/320+w[1]/320, h[0]/320+h[1]/320)

    layout = [[img1, img2, img3],
               [img4, img5, img6],
               [img7, img8, img9]]

    image = cv2.vconcat([cv2.hconcat(line) for line
in layout])

    return image, label

while len(image_files) > 10:
    func = random.choice(num)

    images = random.choices(image_files, k=func)

    for img in images:
        try:
            image_files.remove(img)
        except:
            print(img)

    if func == 1:
        copyfile(f'old_images/{images[0]}',
                f'images/{processed}.jpg')
        #
    copyfile(f'old_labels/{images[0].replace(".jpg",
".txt")}',
            #
            f'labels/lable_{processed}.txt')

```

```

        lbl_text =
open(f'old_labels/{images[0].replace(".jpg",
".txt")}', 'r+').readlines()
        #
    print(f'old_labels/{images[0].replace(".jpg",
".txt")}')
        # print(lbl_text)
        new_file =
open(f'labels/{processed}.txt', 'w')
        for line in lbl_text:
            new_file.write(line)

        processed += 1

    if func == 4:
        file_images = []
        name_labels = []

        for img in images:
            file = cv2.imread(f'old_images/{img}')
            if file is None:
                print(img)
            name_labels.append(img.replace(".jpg",
".txt"))
            file_images.append(file)

        image, label = mosaic4(file_images,
name_labels)

```

Продолжение листинга А1 – Программный код для обработки изображений аугментацией «Мозаика»

```
        cv2.imwrite(f'images/{processed}.jpg',
image)
        with open(f'labels/{processed}.txt', 'w')
as file:
            file.write(label)

        processed += 4

    if func == 9:
        file_images = []
        name_labels = []

        for img in images:
            file = cv2.imread(f'old_images/{img}')
            if file is None:
                print(img)
            name_labels.append(img.replace(".jpg",
".txt"))
            file_images.append(file)

        image, label = mosaic9(file_images,
name_labels)

        cv2.imwrite(f'images/{processed}.jpg',
image)
        with open(f'labels/{processed}.txt', 'w')
as file:
            file.write(label)
```

```
        processed += 9

        print('Progress:', processed, '/',
total_images, end='\r')

    for image in image_files:
        copyfile(f'old_images/{image}',
f'images/{processed}.jpg')

        lbl_text =
open(f'old_labels/{images[0].replace(".jpg",
".txt")}', 'r+').readlines()
        # print(f'old_labels/{images[0].replace(".jpg",
".txt")}')
        # print(lbl_text)
        new_file = open(f'labels/{processed}.txt', 'w')
        for line in lbl_text:
            new_file.write(line)

        processed += 1

        print('Progress:', processed, '/',
total_images, end='\r')

    print('finish')
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг основной программы

Листинг Б1 – Программный код графического интерфейса

```
# -*- coding: utf-8 -*-
#   Developed by Alexander Kraynikov
#   krajnikov.a@edu.narfu.ru
import os
import sys

from PyQt5 import QtGui
from PyQt5.QtWidgets import QWidget, QApplication,
QLabel, QVBoxLayout, \
    QPushButton, QHBoxLayout, QComboBox, QLineEdit
from PyQt5.QtGui import QPixmap, QDoubleValidator
from PyQt5.QtCore import pyqtSignal, Qt, QThread

import numpy as np
import onnxruntime as ort
import cv2 as cv
import random
import time

from YOLOv8 import YOLOv8

from deep_sort.deep_sort.tracker import Tracker as
DeepSortTracker
from deep_sort.tools import generate_detections as
gdet
from deep_sort.deep_sort import nn_matching
from deep_sort.deep_sort.detection import
Detection
import numpy as np
```

```
class Tracker:
    tracker = None
    encoder = None
    tracks = None

    def __init__(self, max_cosine_distance,
max_iou_distance, max_age, n_init):
        nn_budget = None

        encoder_model_filename =
'deep_sort/resources/networks/mars-small128.pb'
        #'model_data/mars-small128.pb'

        metric =
nn_matching.NearestNeighborDistanceMetric("cosine"
, max_cosine_distance, nn_budget)
        self.tracker = DeepSortTracker(metric,

max_iou_distance=max_iou_distance,

max_age=max_age,

n_init=n_init)
        self.encoder =
gdet.create_box_encoder(encoder_model_filename,
batch_size=1)
```

Продолжение листинга Б1 – Программный код графического интерфейса

```
def update(self, frame, detections):

    if len(detections) == 0:
        self.tracker.predict()
        self.tracker.update([])
        self.update_tracks()
        return

    bboxes = np.asarray([d[:-1] for d in
detections])
    bboxes[:, 2:] = bboxes[:, 2:] - bboxes[:,
0:2]
    scores = [d[-1] for d in detections]

    features = self.encoder(frame, bboxes)

    dets = []
    for bbox_id, bbox in enumerate(bboxes):
        dets.append(Detection(bbox,
scores[bbox_id], features[bbox_id]))

    self.tracker.predict()
    self.tracker.update(dets)
    self.update_tracks()

    def update_tracks(self):
        tracks = []
        for track in self.tracker.tracks:
            if not track.is_confirmed() or
track.time_since_update > 1:
                continue
            bbox = track.to_tlbr()
```

```
        id = track.track_id

        tracks.append(Track(id, bbox))

    self.tracks = tracks

class Track:
    track_id = None
    bbox = None

    def __init__(self, id, bbox):
        self.track_id = id
        self.bbox = bbox

class VideoThread(QThread):
    change_pixmap_signal = pyqtSignal(np.ndarray)

    def __init__(self, model_name, conf, iuo,
path=0, record=False):
        self.model_name = model_name
        self.conf = conf
        self.iuo = iuo
        self.record = record
        self.path = path

        self.colors = [(random.randint(0, 255),
random.randint(0, 255), random.randint(0, 255))
for j in range(20)]
        self.count = 0
        self.font = cv.FONT_HERSHEY_SIMPLEX
        self.thickness = 2
```

Продолжение листинга Б1 – Программный код графического интерфейса

```
# параметры

self.detection_threshold = 0.1 # Порог,
выше которого объект считается подтвержденным.
max_cosine_distance = 2 # Порог
соответствия. Образцы с большим расстоянием
считаются недействительным совпадением.
max_iou_distance = 2 # 0.7 Порог ворот.
Ассоциации со стоимостью, превышающей это
значение, игнорируются.
max_age = 30 # 30 Максимальное количество
пропущенных промахов перед удалением трека
n_init = 10 # 5 Количество
последовательных обнаружений до подтверждения
трека

self.tracker =
Tracker(max_cosine_distance, max_iou_distance,
max_age, n_init)

if self.record:
    fourcc =
cv.VideoWriter_fourcc(*'XVID')
    self.out = cv.VideoWriter("./videos/"
+ \
time.strftime("%Y-%m-%d_%H-%M-%S") + \
                    ".mp4",
                    fourcc,
                    10.0, (640, 480))
```

```
super().__init__()
self.is_run = True

def run(self):
    # cap = cv.VideoCapture(int(self.path))
    cap = cv.VideoCapture('D:\\dis\\test-
data\\test2-video.mp4')

    self.detector = YOLOv8("models/" +
self.model_name,
                                self.conf,
                                self.iuo)

    map = dict()
    stop_list = []
    self.fishcounter = 0
    while self.is_run:
        ret, frame = cap.read()
        if ret:

            boxes, scores, class_ids =
self.detector(frame)

            # print(boxes)

            # combined_img =
self.detector.draw_detections(frame)

            detections = []
            frame_center = 1280/2
            for box, class_id, score in
zip(boxes, class_ids, scores):
                x1, y1, x2, y2 =
box.astype(int)
```


Продолжение листинга Б1 – Программный код графического интерфейса

```
class_id = int(class_id)
if score > self.conf:
    detections.append([x1, y1,
x2, y2, score])
    print(self.fishcounter)

    self.tracker.update(frame,
detections)

    for track in self.tracker.tracks:
        bbox = track.bbox
        x1, y1, x2, y2 = bbox
        track_id = track.track_id

        if str(track_id) in map:
            if
float(map[str(track_id)]) > frame_center and
int(x1) < frame_center and track_id not in
stop_list:

                self.fishcounter += 1

stop_list.append(track_id)
        else:
            map[str(track_id)] =
str(x1)

            # img_tmp = frame

            # cv.rectangle(img_tmp,
(int(x1), int(y1)), (int(x2), int(y2)),
(self.colors[track_id % len(self.colors)]), -1)
            # frame =
cv.addWeighted(img_tmp, 0.3, frame, 1 - 0.3, 0)
```

```
frame = cv.putText(frame,
str(track_id), (int(x1), int(y1) - 30), self.font,
1, self.colors[track_id %
len(self.colors)], self.thickness, cv.LINE_AA)
cv.rectangle(frame, (int(x1),
int(y1)), (int(x2), int(y2)),
(self.colors[track_id % len(self.colors)]), 2)

        if self.record:
            # print("record")

self.out.write(cv.resize(frame, (640, 480)))

self.change_pixmap_signal.emit(frame)
cap.release()

def stop(self):
    self.is_run = False
    self.wait()

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("ONNX viewer")

        self.select_model = QComboBox()

self.select_model.addItem(os.listdir("models"))
```

Продолжение листинга Б1 – Программный код графического интерфейса

```
self.select_model.currentIndexChanged.connect(self
    .changemodel)

    self.thread =
VideoThread(os.listdir("models")[0],
            0.7,
            0.5)

self.thread.change_pixmap_signal.connect(self.upda
te_image)
    self.thread.start()

    self.image_width = 1280
    self.image_height = 720

    self.record = False

    self.image_label = QLabel(self)
    self.image_label.resize(self.image_width,
self.image_height)

    self.button_record = QPushButton("Запись")

self.button_record.clicked.connect(self.start_reco
rd)
    self.button_change =
QPushButton("Перезапустить")

self.button_change.clicked.connect(self.changemode
l)

    self.label_confidence = QLabel()
```

```
self.label_confidence.setText("confidence:")
    self.line_confidence = QLineEdit()

self.line_confidence.setValidator(QDoubleValidator
(0.01,1.00,2))
    self.line_confidence.setText("0.70")

    self.label_iuo = QLabel()
    self.label_iuo.setText("iuo:")
    self.label_path = QLabel()
    self.label_path.setText("path:")
    self.line_path = QLineEdit()
    self.line_path.setText('0')
    self.line_iuo = QLineEdit()

self.line_iuo.setValidator(QDoubleValidator(0.01,1
.00,2))
    self.line_iuo.setText("0.50")

    self.vbox = QVBoxLayout()
    self.hbox = QHBoxLayout()

    self.hbox.addWidget(self.image_label)
    self.vbox.addWidget(self.button_record)
    self.vbox.addWidget(self.select_model)

    self.vbox.addWidget(self.label_confidence)
    self.vbox.addWidget(self.line_confidence)
    self.vbox.addWidget(self.label_iuo)
    self.vbox.addWidget(self.line_iuo)
    self.vbox.addWidget(self.label_path)
    self.vbox.addWidget(self.line_path)
```

Продолжение листинга Б1 – Программный код графического интерфейса

```
self.vbox.addWidget(self.button_change)
self.vbox.addStretch()

self.hbox.addLayout(self.vbox)

self.setLayout(self.hbox)

def start_record(self):
    self.record = not self.record
    self.changemodel()

def changemodel(self):
    self.line_iuo.text

    self.thread.stop()
    self.thread =
VideoThread(self.select_model.currentText(),
float(self.line_confidence.text()),
float(self.line_iuo.text()),
self.record,

self.line_path.text())

self.thread.change_pixmap_signal.connect(self.update_image)
self.thread.start()

def closeEvent(self, event):
    self.thread.stop()
    event.accept()
```

```
def update_image(self, cv_img):
    qt_img = self.convert_cv_qt(cv_img)
    self.image_label.setPixmap(qt_img)

def convert_cv_qt(self, cv_img):
    rgb_image = cv.cvtColor(cv_img,
cv.COLOR_BGR2RGB)
    h, w, ch = rgb_image.shape
    bytes_per_line = ch * w
    convert_to_Qt_format =
QtGui.QImage(rgb_image.data,
w,
h,
bytes_per_line,
QtGui.QImage.Format_RGB888)
    p =
convert_to_Qt_format.scaled(self.image_width,
self.image_height,
Qt.KeepAspectRatio)
    return QPixmap.fromImage(p)

if __name__=="__main__":
    app = QApplication(sys.argv)
    a = App()
    a.show()
    sys.exit(app.exec_())
```

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг программы обучения

Листинг В1 – Программа обучения модели Faster R-CNN

```
# %%
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file
I/O (e.g. pd.read_csv)
import os

!pip install -U torchvision > /dev/null

# %%
import torch
import torchvision
from torchvision import datasets, models
from torchvision.transforms import functional as FT
from torchvision import transforms as T
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import DataLoader, sampler,
random_split, Dataset
import copy
import math
from PIL import Image
import cv2
import albumentations as A # our data
augmentation library

import matplotlib.pyplot as plt
%matplotlib inline
```

```
# %%
# remove arnings (optional)
import warnings
warnings.filterwarnings("ignore")
from collections import defaultdict, deque
import datetime
import time
from tqdm import tqdm # progress bar
from torchvision.utils import
draw_bounding_boxes

# %%
from albumentations.pytorch import ToTensorV2

# %%
def get_transforms(train=False):
    if train:
        transform = A.Compose([
            A.Resize(320, 320), # our input size
            A.HorizontalFlip(p=0.3),
            A.VerticalFlip(p=0.3),
            A.RandomBrightnessContrast(p=0.1),
            A.ColorJitter(p=0.1),
            ToTensorV2()
        ],
        bbox_params=A.BboxParams(format='coco'))
    else:
        transform = A.Compose([
            A.Resize(320, 320), # our input size
            A.HorizontalFlip(p=0.3),
            A.VerticalFlip(p=0.3),
            A.RandomBrightnessContrast(p=0.1),
            A.ColorJitter(p=0.1),
            ToTensorV2()
        ],
        bbox_params=A.BboxParams(format='coco'))
```

```

        else:
            transform = A.Compose([
                A.Resize(320, 320), # our input size
                ToTensorV2()
            ],
            bbox_params=A.BboxParams(format='coco'))
        return transform

# %%
class FishDataset(datasets.VisionDataset):
    def __init__(self, root, split='train',
        transform=None, target_transform=None,
        transforms=None):

        self.transforms = transforms

        self.root = os.path.join(root, split)
        self.images =
sorted(os.listdir(os.path.join(root, split,
'images'))))
        self.labels =
sorted(os.listdir(os.path.join(root, split,
'labels'))))

        def _load_image(self, id:int):
            image = cv2.imread(os.path.join(self.root,
'images', self.images[id]))
            image = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
            return image

        def _load_label(self, id:int):

```

```

            label = open(os.path.join(self.root,
'labels', self.labels[id])).readlines()
            return label

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

        def __getitem__(self, id):

            image = self._load_image(id)
            label = self._load_label(id)

            height, width, _ = image.shape

            bbox = [] # [[xmin, ymin, xmax, ymax], ...
]
            for line in label:

                splited = line.strip().split()

                w = float(splited[3]) * width
                h = float(splited[4]) * height

                min_x = int(float(splited[1]) * width
- w/2)
                min_y = int(float(splited[2]) * height
- h/2)

                bbox.append([min_x, min_y, w, h, 1])

            bbox = torch.Tensor(bbox)
            transformed = self.transforms(image=image,
bboxes=bbox)

```

```

        image = transformed['image']
        new_bbox = []
        for box in transformed['bboxes']:
            xmin = box[0]
            xmax = xmin + box[2]
            ymin = box[1]
            ymax = ymin + box[3]
            new_bbox.append([xmin, ymin, xmax,
ymax])

        boxes = torch.tensor(new_bbox,
dtype=torch.float32)

        target = {}

        # here is our transformed target
        target['boxes'] = boxes
        target['labels'] = torch.tensor([1 for i
in range(len(bbox))], dtype=torch.int64)
        target['image_id'] = id
        print(boxes)
        if len(boxes) > 0:
            target['area'] = (boxes[:, 3] -
boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0]) # we
have a different area
        else:
            target['area'] = 0
            target['boxes'] =
torch.zeros((0,4),dtype=torch.float32)
            target['iscrowd'] =
torch.zeros((len(boxes),), dtype=torch.int64)

        return image.div(255), target # scale
images

```

```

# %%
train_dataset =
FishDataset(root='D:\\dis\\datasets\\fish-
v9\\fishes', transforms=get_transforms(True))

# %%
len(train_dataset)

# %%
sample = train_dataset[3]
img_int = torch.tensor(sample[0] * 255,
dtype=torch.uint8)
plt.imshow(
    draw_bounding_boxes(
        img_int,
        sample[1]['boxes'],
        width=4
    ).permute(1, 2, 0))

# %%
model =
models.detection.fasterrcnn_mobilenet_v3_large_fpn
(pretrained=True)

in_features =
model.roi_heads.box_predictor.cls_score.in_feature
s # we need to change the head
model.roi_heads.box_predictor =
models.detection.faster_rcnn.FastRCNNPredictor(in_
features, 2)

# %%
def collate_fn(batch):

```

```

        return tuple(zip(*batch))

# %%
train_loader = DataLoader(train_dataset,
batch_size=4, shuffle=True, collate_fn=collate_fn)

# %%
images, targets = next(iter(train_loader))
images = list(image for image in images)
targets = [{k:v for k, v in t.items()} for t in
targets]
output = model(images, targets) # just make sure
this runs without error

# %%
device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
print('Using device:', device)
print()

#Additional Info when using cuda
if device.type == 'cuda':
    print(torch.cuda.get_device_name(0))
    print('Memory Usage:')
    print('Allocated:',
round(torch.cuda.memory_allocated(0)/1024**3,1),
'GB')
    print('Cached:    ',
round(torch.cuda.memory_reserved(0)/1024**3,1),
'GB')

# %%
model = model.to(device)

```

```

# %%
params = [p for p in model.parameters() if
p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.01,
momentum=0.9, nesterov=True, weight_decay=1e-4)
# lr_scheduler =
torch.optim.lr_scheduler.MultiStepLR(optimizer,
milestones=[16, 22], gamma=0.1) # lr scheduler

# %%
import sys

# %%
def train_one_epoch(model, optimizer, loader,
device, epoch):
    model.to(device)
    model.train()

#     lr_scheduler = None
#     if epoch == 0:
#         warmup_factor = 1.0 / 1000 # do lr
warmup
#         warmup_iters = min(1000, len(loader) -
1)

#         lr_scheduler =
optim.lr_scheduler.LinearLR(optimizer,
start_factor = warmup_factor,
total_iters=warmup_iters)

    all_losses = []
    all_losses_dict = []

    for images, targets in tqdm(loader):

```

```

        images = list(image.to(device) for image
in images)
        targets = [{k: torch.tensor(v).to(device)
for k, v in t.items()} for t in targets]

        loss_dict = model(images, targets) # the
model computes the loss automatically if we pass
in targets
        losses = sum(loss for loss in
loss_dict.values())
        loss_dict_append = {k: v.item() for k, v
in loss_dict.items()}
        loss_value = losses.item()

        all_losses.append(loss_value)
        all_losses_dict.append(loss_dict_append)

        if not math.isfinite(loss_value):
            print(f"Loss is {loss_value}, stopping
trainig") # train if loss becomes infinity
            print(loss_dict)
            sys.exit(1)

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()
#         if lr_scheduler is not None:
#             lr_scheduler.step() #

        all_losses_dict =
pd.DataFrame(all_losses_dict) # for printing
        print("Epoch {}, lr: {:.6f}, loss: {:.6f},
loss_classifier: {:.6f}, loss_box: {:.6f},
loss_rpn_box: {:.6f}, loss_object: {:.6f}".format(

```

```

        epoch, optimizer.param_groups[0]['lr'],
np.mean(all_losses),
        all_losses_dict['loss_classifier'].mean(),
        all_losses_dict['loss_box_reg'].mean(),
        all_losses_dict['loss_rpn_box_reg'].mean()
,
        all_losses_dict['loss_objectness'].mean()
    ))
num_epochs=10

for epoch in range(num_epochs):
    train_one_epoch(model, optimizer,
train_loader, device, epoch)
    #     lr_scheduler.step()
    test_dataset =
FishDataset(root='D:\\dis\\datasets\\fish-
v9\\fishes', split="test",
transforms=get_transforms(False))
    model.eval()
    torch.cuda.empty_cache()
    img, _ = test_dataset[5]
    img_int = torch.tensor(img*255, dtype=torch.uint8)
    with torch.no_grad():
        prediction = model([img.to(device)])
        pred = prediction[0]

# %%
fig = plt.figure(figsize=(14, 10))
plt.imshow(
    draw_bounding_boxes(
        img_int,
        pred['boxes'][pred['scores'] > 0.8],
        width=4
    ).permute(1, 2, 0))

```


ПРИЛОЖЕНИЕ Г

(обязательное)

Листинг программы инференса

Листинг В1 – Программа для инференса модели Faster R-CNN

```
from torchvision import models
model=
models.detection.fasterrcnn_mobilenet_v3_large_fpn
(pretrained=True)
in_features =
model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor =
models.detection.faster_rcnn.FastRCNNPredictor(in_
features, 2)
model.load_state_dict(torch.load('D:\\dis\\models\\
\\Faster R-CNN -30 epch.pth',
map_location=torch.device('cpu'))))
model.eval()
import matplotlib.pyplot as plt
device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
print('Using device:', device)
#Additional Info when using cuda
if device.type == 'cuda':
    print(torch.cuda.get_device_name(0))
print('Memory Usage:')
print('Allocated:',
round(torch.cuda.memory_allocated(0)/1024**3,1),
'GB')
print('Cached:', round(torch.cuda.memory_reserved
(0)/1024**3,1), 'GB')
```

```
from torchvision.utils import draw_bounding_boxes
import cv2,os,albumations as A,random,time,
torch,
from albumations.pytorch import ToTensorV2,
root = 'D:\\dis\\test-data\\fine-dataset'
imgs = os.listdir(root)
# img = cv2.imread(os.path.join(root,
random.choice(imgs)))
img = cv2.imread('D:\\dis\\datasets\\fish-
v8\\fishes\\valid\\images\\014980.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
transform = A.Compose([A.Resize(320,
320),ToTensorV2()])
img = transform(image=img)['image']
with torch.no_grad():
    prediction = model([img.div(255).to(device)])
    pred = prediction[0]
print(time.time_ns() - timer)
plt.imshow(
    draw_bounding_boxes(
        img,
        pred['boxes'][pred['scores'] > 0.3],
        width=3
    ).permute(1, 2, 0))
```