|  |  |  |  |
| --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное автономное образовательное учреждение высшего образования  **«Северный (Арктический) федеральный университет имени М. В. Ломоносова»**  Высшая школа информационных технологий и автоматизированных систем | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | |
| На тему: | Глубокое обучение. Продвинутый уровень (модуль) | | |
|  |  | | |
|  |  | | |
|  | | Выполнил обучающийся: | |
| Крайников Александр Николаевич | |
| (Ф.И.О.) | |
| Направление подготовки / специальность: | |
| 09.04.02 Информационные системы и технологии | |
| (код и наименование) | |
| Курс: | 2 |
| Группа: | 151266 |
| Руководитель: | |
| Васендина Ирина Сергеевна, ст. препод., к.т.н. | |
| (Ф.И.О. руководитель, должность/уч. степень/звание) | |

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

Архангельск 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

**«Северный (Арктический) федеральный университет имени М. В. Ломоносова»**

|  |
| --- |
| Высшая школа информационных технологий и автоматизированных систем |
| (наименования высшей школы) |
| Кафедра информационных систем и информационной безопасности |
| (наименование кафедры) |

**ЗАДАНИЕ ДЛЯ ПРОЕКТА**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| По дисциплине: | | Глубокое обучение. | | | | |
|  | | Продвинутый уровень (модуль) | | | | |
| студенту | ВШИТАС | высшей школы | 2 | курса | 151266 | группы |
| Крайников Александр Николаевич | | | | | | |
| (фамилия имя отчество студент, выполнившего работу) | | | | | | |
| 09.03.04 «Информационные системы и технологии» | | | | | | |
| Тема: | | Разработка системы отслеживания перемещений | | | | |
|  | | рыбы с помощью компьютерного зрения | | | | |
| Формулировка задания: | |  | | | | |
|  | Сбор датасета; | | | | | |
|  | Обучение модели детектирования рыбы; | | | | | |
|  | Разработка системы отслеживания перемещения рыбы; | | | | | |
|  | Разработка программного обеспечения для работы системы | | | | | |
|  | в режиме реального времени. | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Срок выполнения с | « » 2023 г. | по | « » 2024 г. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Руководитель работы | ст. препод |  |  |  | И. С. Васендина |
|  |  |  | (подпись) |  |  |

Архангельск 2024

ЛИСТ ДЛЯ ЗАМЕЧАНИЙ

Оглавление

[Введение 5](#_Toc164983145)

[1. Сбор и подготовка данных 8](#_Toc164983146)

[2. Разработка модели определения рыбы 18](#_Toc164983147)

[3. Разработка программной системы для отслеживания рыбы 22](#_Toc164983148)

[Заключение 24](#_Toc164983149)

[Список использованной литературы 25](#_Toc164983150)

[Приложение А 27](#_Toc164983151)

# Введение

Концепция индустрии 4.0 предсказывает нам ближайшее развитие производственных технологий. В рамках данной концепции активно рассматривается цифровизация, использование интеллектуальных систем и превентивные меры. Рыбная промышленность в том числе нуждается в нововведениях, связанных с этой сферой.[1]

Одна из проблем рыбной промышленности – это расчёт количества рыбы, которая будет доступна для обработки. Данный фактор напрямую зависит о того, сколько рыб протекает через русла. Зная это количество в примерных объемах можно предсказывать поведение рынка рыбных ресурсов.

Для решения данной задачи была выбрана технология компьютерного зрения. Необходимо разработать систему способную определять рыбу и отслеживать ее перемещение. В перспективе данная система должна определять вид рыбы и ее примерный размер.

Цель проекта – выработка оптимальной системы отслеживания рыбы.

Задачи проекта:

- собрать и подготовить набор данных;

- разработать модель отсаживания рыбы;

- разработать программную систему детектирования рыбы.

Задача отслеживания и классификации рыбных ресурсов уже решалась многими исследователями и разработчиками в области компьютерного зрения и глубокого обучения. Например, исследователи из проекта Ocean Aware в 2022 году опубликовали работу, в которой рассматривали применение моделей YOLOv3, YOLOv4 и Mask-RCNN для детектирования и сегментации изображений с рыбой. Также в данной публикации рассматривается применение алгоритма Norfair для объединения двух выделенных объектов между кадрами в качестве одной сущности, на основе данных о расположении и скорости передвижения. Также в данной работе данные с модели глубокого обучения и гидролокатора обрабатываются фильтром Калмана, что позволяет увеличить точность определения рыбы. По результатам данного исследования лучше всего себя показала модель YOLOv3, определяя точнее в среднем на 10% и быстрее в 3 раза. Однако к моменту написания работы еще не вышла полноценная модель YOLOv4, которая является улучшенной модель YOLOv3.[2] Пример обнаружения рыбы и оценки достоверности изображен на рисунке 1.1.

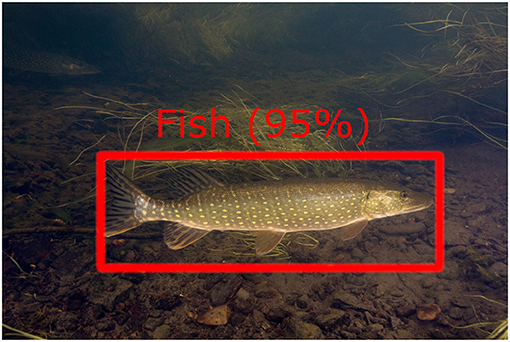


Рисунок 1.1 - Пример обнаружения рыбы и оценки достоверности

В другой публикации, также посвящённой обнаружению и классификации рыб алгоритмы CNN и YOLO используются в совокупности. Данная публикация используется наиболее современные модели этих архитектур, Fast-CNN и YOLOv3, соответственно. Изначально рыбу любого вида детектирует модель YOLO и выделяет ее квадратом. Полученное изображение передается в классификатор CNN. Этот подход позволяет получить точность определения 83,68%, что является достаточным результатом для продолжения исследования.[3]

Более современная публикация, выпущенная в 2023 году, описывает использование YOLOv5 для задачи сегментации изображений. Для противопоставления модель Mask-RCNN, которая по мнению авторов является медленной и не применимой в условия реального времени. Данные исследователи используют модель YOLOv5 для быстрого определения рыбы в задаче детектирования. В последствии модель YOLOv5 была модернизирована путем добавления сегментирующей головы, основываясь на идеях дизайна DeepLabv3+. В результате исследований удалось получить точность более 94%.[4]

Большая часть рассмотренных публикаций также используют аугментации для увеличения количества изображений самыми популярными аугментациями являются: HSV, Мозайка, Поворот и Отражение. Данные аугментации предобрабатывают некоторые изображения или создают новые, позволяя нормализовать классы, равномерно распределить рыб по фотографии, что в совокупности позволяет увеличить точность работы моделей.

## Сбор и подготовка 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 [5] | Фото | - | 662 | 39 770 | 3 202 | - |
| 2 | Fishclef 2015 [6] | Видео | - | 98 | - | - | - |
| 3 | Fishnet [7] | Фото | 94 806 | - | - | - |  |
| 4 | fishRecognition\_GT [8] | Фото | - | 27 370 | - | - | - |
| 5 | LABELED-FISHES-IN-THE-WILD [9] | Фото, Видео | 207 кадр. | 1 видео | - | - | - |
| 6 | Fish detection and tracking [10] | Видео | - | - | - | - | 17 |
| 7 | Underwater Object Detection Dataset [11] | Фото | 7 556 | - | - | - | - |
| 8 | Deep Fish Object Detection [12] | Фото | 5 568 | - | - | - | - |
| 9 | Luderick seagrass [13] | 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.

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. Примеры изображений приведены на рисунке 1.1. и рисунке 1.2.

Изображение выглядит как подводный, вода, природа, Организм

Автоматически созданное описание

Рисунок 1.1 – Пример изображения из датасета DeepFish

Изображение выглядит как рыба, вода, риф, подводный

Автоматически созданное описание

Рисунок 1.2 – 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 1.3.

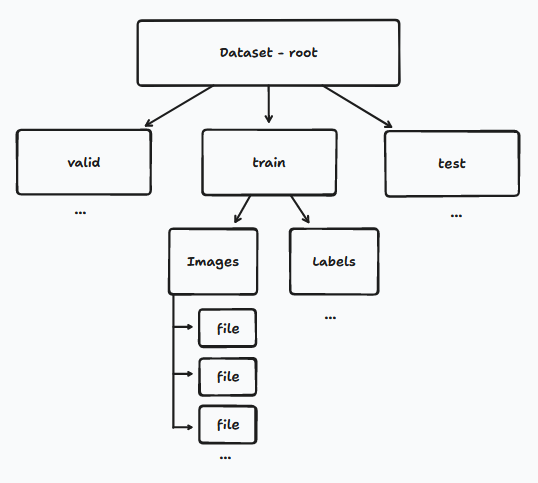


Рисунок 1.3 – Иерархий файлов в наборе данных

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 реализовывает 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 1.1.

Листинг 1.1 – 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))  # frame = cv.rectangle(frame,  # (x, y),  # (x+w, y+h),  # (255, 255, 0),  # 3)  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.imshow('Frame', frame)  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() |

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 1.2.

Листинг 1.2 – Удаление дубликатов изображений

|  |
| --- |
| import os  import hashlib  # split\_group\_list = ['test', 'train', 'valid']  split\_group\_list = ['images']  dataset\_path = '.'  complited = 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.sha1(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  complited += 1  print(f'Выполнено: {complited} / {num\_images}', end='\r')  print(f'{group}: удалено {del\_num} изображений')  print('Дубликаты удалены') |

Для сжатия изображений до разрешения 320 на 320 пикселей и переименования изображений и их лейблов использовался код, приведенный в листинге 1.3.

Листинг 1.3 – Переименование файлов и сжатие разрешения

|  |
| --- |
| import os  import cv2  images = os.listdir('images')  count\_name\_with\_sharp = 0  count\_images\_complited = 0  print(len(images))  for image in images:  # if '#' in image:  # new\_image = image.replace('#', '')  # os.rename('labels/' + image, 'labels/' + new\_image)  # count\_name\_with\_sharp += 1  frame = cv2.imread('images/' + image)  frame = cv2.resize(frame, (320, 320), interpolation = cv2.INTER\_AREA)  cv2.imwrite('images/' + image, frame)  count\_images\_complited += 1  print('Progress:', count\_images\_complited, '/', len(images), end='\r')  print(count\_images\_complited)  # print(count\_name\_with\_sharp) |

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 1.4.

Листинг 1.4 – 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 = [] |

Продолжение листинга 1.4 – Очистка датасета от ошибок подписи

|  |
| --- |
| 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)  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 1.5.

Листинг 1.5 – 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)  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 1.6.

Листинг 1.7 – 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() |

## Разработка Модели определения рыбы

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 2.1

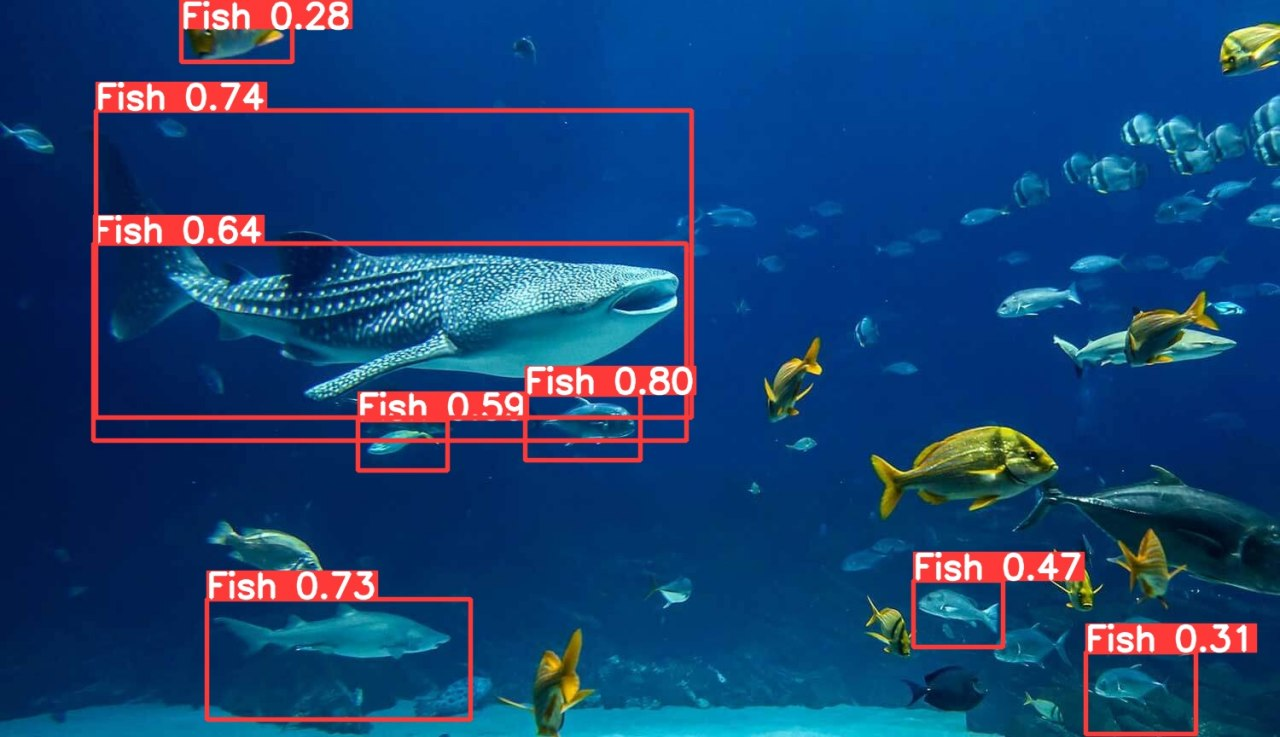


Рисунок 2.1 – Детектирование модели YOLOv8n после обучения

Для сравнения процесса обучения и проверки на обученность и переобученность использовались две метрики: mAP50 и mAP50-95. Также сравнивались потери при валидации, которые проводились после каждой эпохи. К сожалению, в процессе обучения были утеряны графики для модели YOLOv8m на черно-белом датасет. Однако данная модель показала сравнительно низки показатели, поэтому далее она не рассматривается. Примеры метрик указаны на рисунке 2.2.

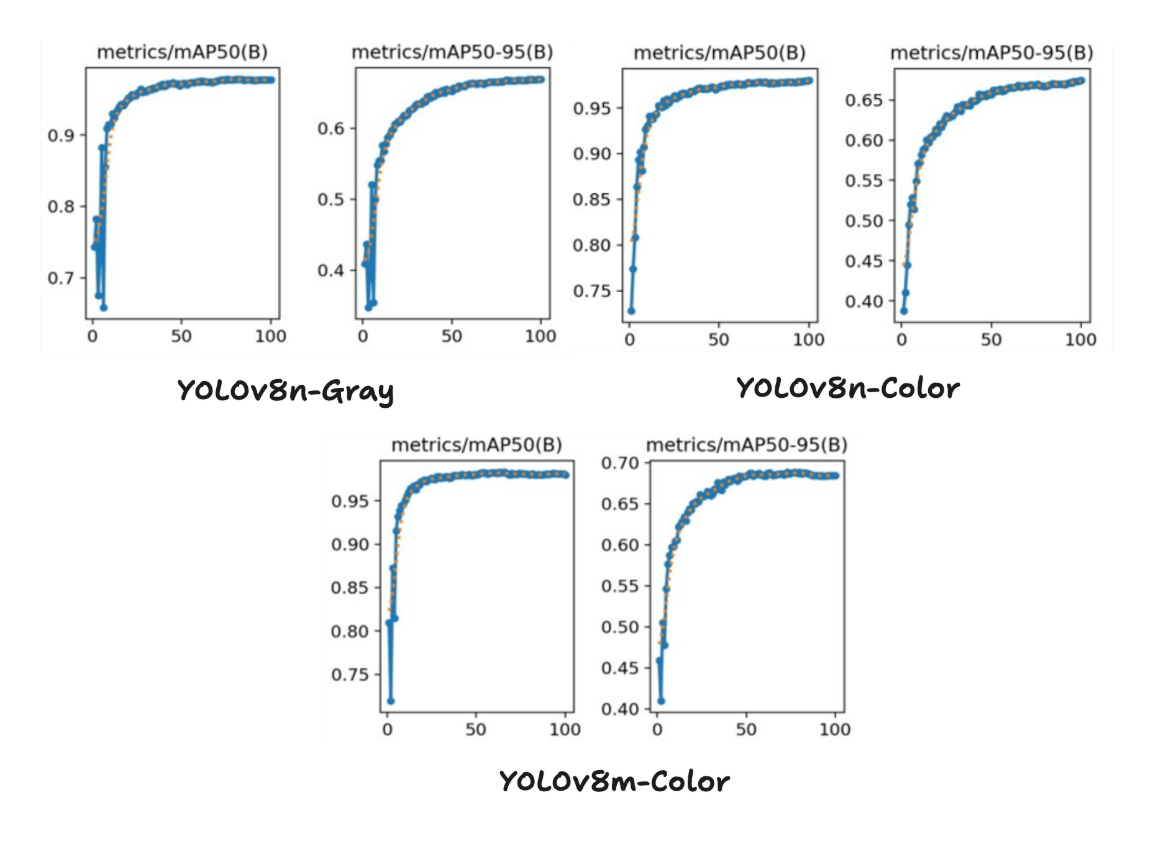


Рисунок 2.2 – Метрики обученности моделей

Исходя из указанных данных можно заметить, что метрики у моделей n не вышли на плато, а значит могли продолжить увеличиваться. В данном случае был смысл оставить эти модели обучаться еще на 20 эпох. На рисунке 2.3 указаны метрики потери. Исходя из указанных метрик потерь можно сделать вывод, что модель m после 50-60 эпохи начала переобучаться. Для данной модели определение рыб оказалось слишком простой задачей. Возможно, что данная модель проявит себя лучше при большей аугментации и большем наборе данных.

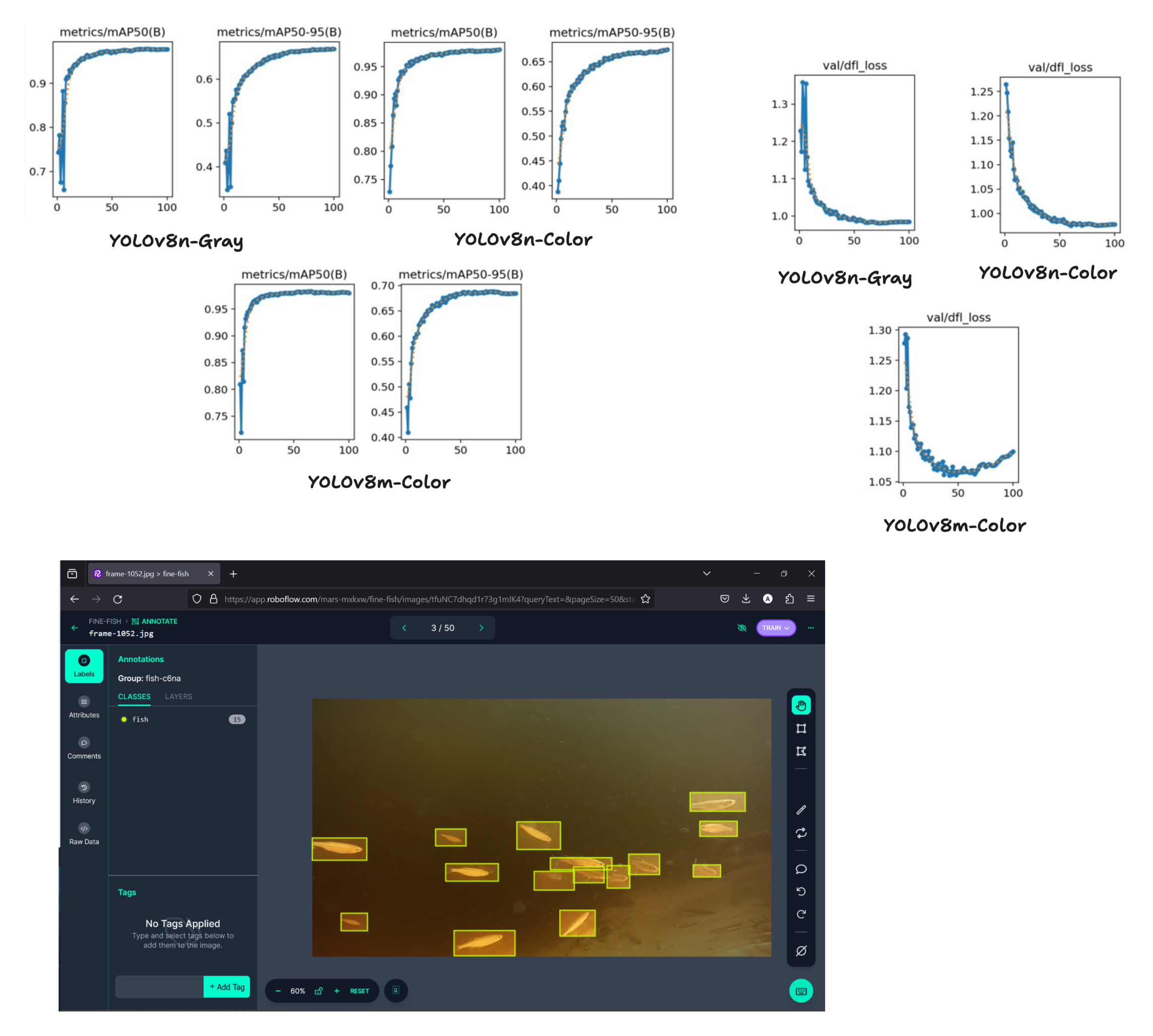


Рисунок 2.3 – Метрики потери моделей

Для итоговой проверки модели было выбрано видео с мутной водой и большим количеством рыбы. Данное видео является наиболее сложным и приближенным к реальным условиям использования. Поэтому модель была доучена на двадцатой части изображений из видео в случайном порядке в течение 5 эпох. Кадр из данного видео представлен на рисунке 2.4. 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 2.4 – 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 2.5

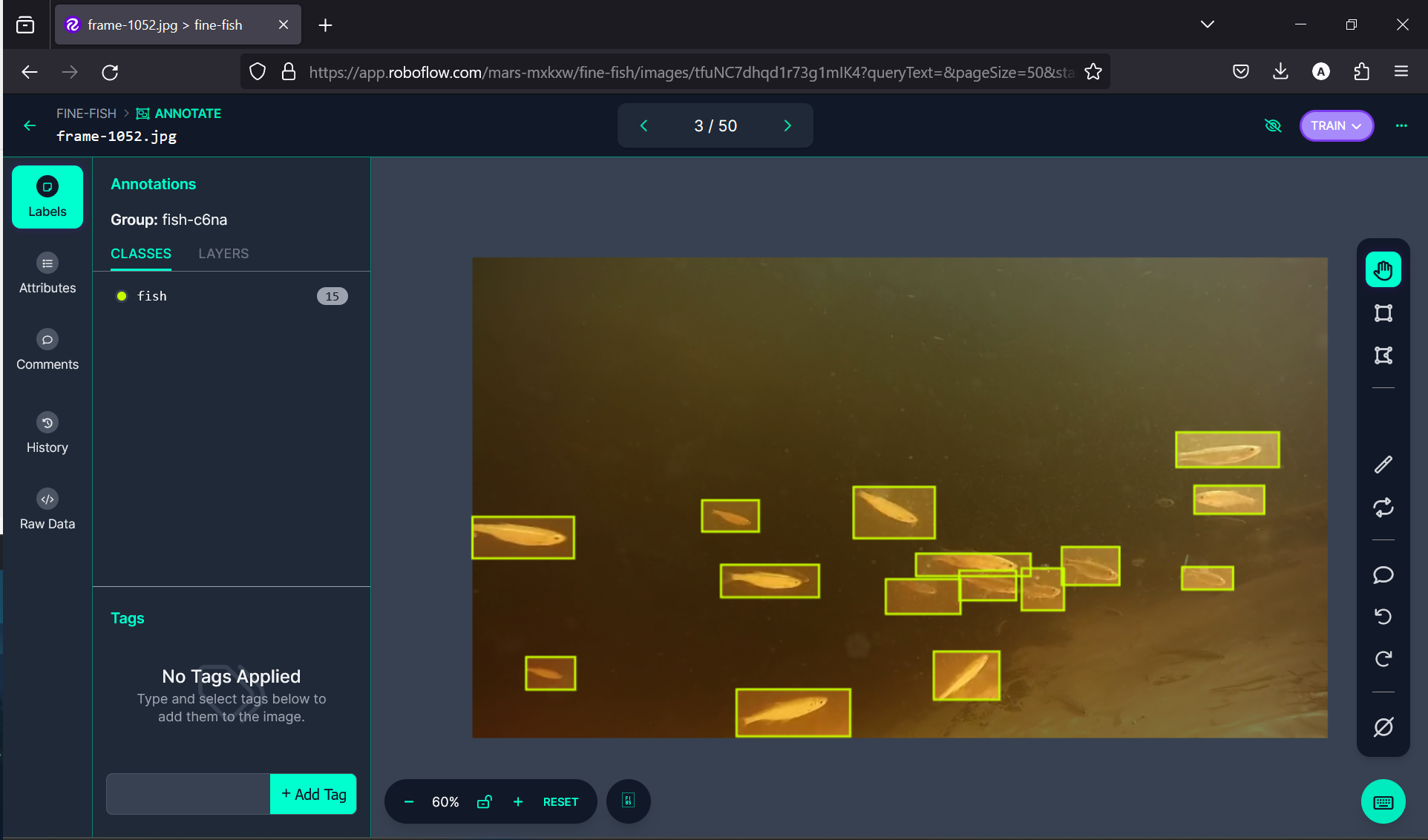


Рисунок 2.5 – Интерфейс разметки данных Roboflow

## Разработка программной системы для Отслеживания рыбы

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 данному 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 приложение 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 счет 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. Финальный вид приложения отображен на рисунке 3.1.



Рисунок 3.1 – Финальный интерфейс приложения

# Заключение

В процесс выполнения проекта было разработано приложение на основе нейронных сетей и компьютерного зрения, позволяющее отслеживать перемещение рыбы в режиме реального времени.

В дальнейшем развитии приложения можно разработать нейронную сеть сегментации изображения, это позволит увеличить точность детектирования рыбы и примерно определять размер рыбы. Также можно классифицировать рыбу, это позволит получать большую статистику о перемещении рыбного ресурса.

Для получения лучшего качества определения рыбы можно провести дополнительные эксперименты с другими архитектурами нейронных сетей и предобработкой датасета, а также использовать аугментации при обучении.

Цель проекта можно считать достигнутой. Все задачи реализованы. Проект можно считать выполненным.

# Список использованной литературы

1. Козлова Г. Г., Арбузова Т. А. ВЛИЯНИЕ ИНДУСТРИИ 4.0 НА ПРОМЫШЛЕННЫЕ ПРЕДПРИЯТИЯ // Международный журнал гуманитарных и естественных наук. 2021. №4-3. Режим доступа: https://www.qgis.org/ru/site/, свободный (дата обращения 10.04.2024);
2. Frontiers [электронный ресурс] – Электрон. Дан: Temperate fish detection and classification: a deep learning based approach – Режим доступа: https://www.frontiersin.org/articles/10.3389/fmars.2021.823173/full, свободный (дата обращения 20.04.2024);
3. Splinger Link [электронный ресурс] – Электрон. Дан: An Underwater Fish Species Image Dataset for Deep Learning – Режим доступа: https://link.springer.com/article/10.1007/s10489-020-02154-9, свободный (дата обращения 20.04.2024);
4. PeerJ [электронный ресурс] – Электрон. Дан: A multitask model for realtime fish detection and segmentation based on YOLOv5 – Режим доступа: https://peerj.com/articles/cs-1262/, свободный (дата обращения 20.04.2024);
5. DeepFish [электронный ресурс] – Электрон. Дан: An Underwater Fish Species Image Dataset for Deep Learning – Режим доступа: https://alzayats.github.io/DeepFish/, свободный (дата обращения 20.04.2024);
6. GitHub [электронный ресурс] – Электрон. Дан: FishCLEF-2015 – Режим доступа: https://github.com/perceivelab/FishCLEF-2015, свободный (дата обращения 20.04.2024);
7. FishNet [электронный ресурс] – Электрон. Дан: A Large-scale Dataset and Benchmark for Fish Recognition, Detection, and Functional Traits Prediction – Режим доступа: https://fishnet-2023.github.io/, свободный (дата обращения 20.04.2024);
8. F4K [электронный ресурс] – Электрон. Дан: h Recognition Ground-Truth data – Режим доступа: https://homepages.inf.ed.ac.uk/rbf/fish4knowledge/  
   GROUNDTRUTH/RECOG/, свободный (дата обращения 20.04.2024);
9. NOAA FISHERS [электронный ресурс] – Электрон. Дан: Labeled Fishes in the Wild – Режим доступа: https://www.fisheries.noaa.gov/west-coast/science-data/labeled-fishes-wild, свободный (дата обращения 20.04.2024);
10. F4K [электронный ресурс] – Электрон. Дан: Fish Trajectory Ground Truth Dataset – Режим доступа: https://homepages.inf.ed.ac.uk/rbf/  
    fish4knowledge/GROUNDTRUTH/BEHAVIOR/, свободный (дата обращения 20.04.2024);
11. Kaggle [электронный ресурс] – Электрон. Дан: Underwater Object Detection Dataset – Режим доступа: https://www.kaggle.com/datasets/  
    slavkoprytula/aquarium-data-cots, свободный (дата обращения 20.04.2024);
12. Kaggle [электронный ресурс] – Электрон. Дан: Deep Fish Object Detection – Режим доступа: https://www.kaggle.com/datasets/vencerlanz09/deep-fish-object-detection, свободный (дата обращения 20.04.2024);
13. GitHub [электронный ресурс] – Электрон. Дан: An annotated dataset for automated detection and counting of estuarine fish in poor visibility conditions – Режим доступа: https://www.qgis.org/ru/site/, свободный (дата обращения 20.04.2024

# Приложение А

(обязательное)

Листинг основной программы

Листинг А1 – 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)  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  # 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)  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.addItems(os.listdir("models"))  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.update\_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\_record)  self.button\_change = QPushButton("Перезапустить")  self.button\_change.clicked.connect(self.changemodel)  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)  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\_()) |