Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Байбак І. А.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О.К.
(прізвище та ініціали)
« 04 » червня 2025 р.

Рецензент: к.т.н., проф. каф. КСУ
Биков М. М.
(прізвище та ініціали)
« 05 » червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
« 06 » червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Байбаку Іллі Андрійовичу

1. Тема роботи: Програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі.
керівник роботи: Колесницький О.К, к.т.н., проф.
затверджені наказом вищого навчального закладу "20" березня 2025 року № 97

2. Термін подання студентом роботи 30 Травня 2025 р.

3. Вихідні дані до роботи :
кількість класів риб – не менше 5; роздільна здатність зображень - 416 x 416 пікселів; формат файлів зображення – jpeg, обсяг набору даних для навчання та тестування – не менше 500 зображень; використання об'єктно орієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи виявлення риб на підводних зображеннях та вибрати найбільш ефективний;
обрати нейронну мережу, проаналізквати її архітектуру
розробити модель виявлення риб на підводних зображеннях;
розробити алгоритм роботи модуля виявлення риб на підводних зображеннях;
спроектувати та програмно реалізувати модуль виявлення риб на підводних зображеннях за допомогою згорткової нейронної мережі;
протестувати роботу програми виявлення риб на підводних зображеннях на основі згорткової нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема алгоритму роботи програми;
структура нейронної мережі;
характеристики набору даних;
результати роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесницький О. К., к.т.н., проф. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	07.05.25	
2	Обгрунтування методу розв'язання задачі	08.05.25	11.05.25	
3	Проектування програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі	12.05.25	15.05.25	
4	Програмна реалізація модуля виявлення риб на підводних зображеннях	16.05.25	26.05.25	
5	Аналіз результатів тестування модуля виявлення риб на підводних зображеннях	27.05.25	01.06.25	
6	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Байбак І. А.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Колесницький О. К.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Байбак І. А. Програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі. Бакалаврська кваліфікаційна робота скаладається з 78 сторінок формату А4, на яких є 19 рисунків, 2 таблиці, список використаних джерел містить 19 найменувань.

Метою роботи є підвищення середньої повноти виявлення риб на підводних зображеннях.

У роботі було розроблено модель виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7. У розробленій моделі використовується перенесення навчання (Transfer Learning) для покращення виявлення риби спеціально для меншої риби і в умовах каламутної води. Модуль реалізовано мовою програмування Python з використанням спеціалізованих бібліотек Torch, NumPy, OpenCV та Matplotlib. Навчання та тестування нейромережі проводилось на наборі даних Ozfish, який містить 43 тисячі анотацій обмежувальних рамок у 1800 кадрах. Було оцінено 5 варіантів навчання моделі: 1) попередньо навчена, 2) перенесення навчання (без аугментації), 3) перенесення навчання (з аугментацією), 4) навчена «з нуля» (без аугментації), 5) навчена «з нуля» (з аугментацією). Найкращою по параметру середньої повноти (91,4%) є модель з перенесеним навчанням і 100% аугментацією, яка по параметру середньої точності (75,81%) знаходиться на тому ж рівні, що і попередньо навчена модель (76,28%). Тобто розроблений програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі має (91,4%) порівняно з найкращим аналогом (90,6%) збільшену на 0,8% середню повноту виявлення риб на підводних зображеннях.

Ключові слова: виявлення риб, підводні зображення, згорткова нейронна мережа.

ABSTRACT

Baybak I. A. Food product recognition software module based on a convolutional neural network. The bachelor thesis consists of 78 pages of A4 format, on which there are 19 figures, 2 tables, the list of used sources contains 19 names.

The aim of the work is to increase the average recall of fish detection in underwater images.

In the work, a fish detection model based on the YOLOv7 convolutional neural network was developed. The developed model uses transfer learning to improve fish detection specifically for smaller fish and in turbid water conditions. The module is implemented in the Python programming language using specialized libraries Torch, NumPy, OpenCV and Matplotlib. The neural network was trained and tested on the Ozfish dataset, which contains 43 thousand bounding box annotations in 1800 frames. 5 model training options were evaluated: 1) pre-trained, 2) transfer learning (without augmentation), 3) transfer learning (with augmentation), 4) trained "from scratch" (without augmentation), 5) trained "from scratch" (with augmentation). The best model in terms of average recall (91.4%) is the model with transferred learning and 100% augmentation, which in terms of average precision (75.81%) is at the same level as the pre-trained model (76.28%). That is, the developed software module for detecting fish in underwater images based on a convolutional neural network has (91.4%) an average completeness of detecting fish in underwater images increased by 0.8% compared to the best analogue (90.6%).

Keywords: fish detection, underwater images, convolutional neural network.

ЗМІСТ

ВСТУП

Актуальність досліджень. Виявлення риби на підводних зображеннях давно було важливим завданням для дослідників-екологів, щоб краще зрозуміти підводні середовища проживання. Однак це завдання може бути досить складним через низьку освітленість підводних зображень, складний фон і велике розмаїття видів риб. Наразі існуючі моделі виявлення риби не так добре працюють на крихітній рибі, яку важко відрізнити від фону через її розмір. Крім того, моделям виявлення риби часто важко виявити рибу на підводних зображеннях низької якості через каламутність води, що також ускладнює виявлення меншої риби.

Вхідними даними розроблюваного модуля є зображення, зроблені під водою. Результатом роботи модуля є зображення з передбаченими обмежувальними рамками, намальованими навколо кожної виявленої риби. Перспективним для цієї задачі виявлення риби на зображеннях є використання згорткових нейронних мереж, зокрема, архітектури YOLOv7. Також позитивний ефект може мати так зване «перенесення навчання», коли вже навчена модель мережі YOLOv7 донавчається на конкретне завдання на основі спеціалізованого набору даних. Було використано набір даних Ozfish, який містить зображення дрібної риби за умов каламутної води.

Метою дослідження є підвищення середньої повноти виявлення риб на підводних зображеннях за рахунок використання згорткової штучної нейронної мережі.

Об’єктом дослідження є процес комп’ютеризованого розпізнавання видів риб на підводних зображеннях на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розпізнавання видів риб на підводних зображеннях на основі згорткової нейронної мережі та точність їх роботи.

Задачі дослідження:

1. проаналізувати відомі методи виявлення риб на підводних зображеннях та обрати напрямок досліджень;
2. розробити структуру програмного модуля виявлення риб на підводних зображеннях на основі нейронної мережі;
3. обґрунтувати вибір архітектури нейромережі;
4. розробити алгоритм роботи програмного модуля виявлення риб на підводних зображеннях;
5. здійснити програмну реалізацію модуля виявлення риб на підводних зображеннях;
6. провести тестування програмного модуля виявлення риб на підводних зображеннях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ

1.1 Постановка задачі

Виявлення риби на підводних зображеннях має вирішальне значення для екологічних досліджень для покращення розуміння водних середовищ існування. Однак це завдання представляє значні труднощі через низьку освітленість підводних зображень, складний фон і велике розмаїття видів риб. Існуючі моделі виявлення часто погано працюють під час виявлення крихітної риби, особливо в умовах каламутної води.

У цій роботі використовується згорткова нейронна мережа як найбільш перспективна для обробки зображень, а також технологія перенесення навчання (Transfer Learning) для створення моделі виявлення риби, яка краще узагальнює рибу невеликого розміру та непрозорі зображення. Обґрунтованим є використання набору даних Ozfish, який містить зображення невеликих риб і фотографії, зроблені в каламутній воді. Вхідними даними розроблюваної моделі є зображення, зроблені під водою. Потім використовується модельна архітектура YOLOv7 (описана нижче) для виявлення риби. Результатом роботи модуля є зображення з передбаченими обмежувальними рамками, намальованими навколо кожної виявленої риби.

Оскільки метою даного проекту є дослідження впливу перенесення навчання (Transfer Learning) на це завдання, було оцінено набір даних Ozfish на трьох типах моделей. По-перше, було оцінено попередньо навчену модель виявлення риби YOLOv7 (яка була попередньо навчена на великому відкритому наборі даних про рибу із зображень у відкритому доступі). Потім було покращено цю попередньо навчену модель за допомогою Transfer Learning на Ozfish. Нарешті, було навчено ту саму архітектуру моделі на Ozfish з нуля (без попередньо навчених ваг). Також були проведені експерименти, щоб дослідити, як кількість даних впливає на Transfer Learning.

1.2 Огляд відомих методів виявлення об’єктів на зображеннях

Виявлення об’єктів — важливе завдання комп’ютерного зору визначення місцезнаходження та розпізнавання об’єктів, присутніх на зображенні/відео.

Безліч практичних завдань від автоматизації контролю на виробництві до конструювання роботизованих автомобілів безпосередньо пов'язані із завданням пошуку об'єктів на зображенні. Для її вирішення можна застосовувати дві різні стратегії, які залежать від умов зйомки – моделювання фону та моделювання об'єкта.

1) Моделювання фону - цей підхід можна використовувати якщо камера нерухома, тобто. ми маємо тло, що мало змінюється, і таким чином можна побудувати його модель. Усі точки зображення, які суттєво відхиляються від моделі фону, вважаємо об'єктами переднього плану. Таким чином можна вирішувати завдання виявлення та супроводу об'єкта.

2) Моделювання об'єкта - цей підхід більш загальний, застосовується у випадках коли фон постійно і істотно змінюється. На відміну від попереднього випадку, тут нам потрібно знати що ми хочемо знайти, тобто. необхідно побудувати модель об'єкта, та перевірити точки картинки на відповідність цій моделі.

Іноді умови завдання дозволяють комбінувати обидва підходи, це може суттєво покращити результати.

Вирішення задачі з моделюванням фону для нерухомої камерою є обмеженим у використанні, тому більш перспективною виявляється друга стратегія, тобто. моделювання об'єкта пошуку.

Розглянемо методи пошуку об'єкта на зображенні згідно другої стратегії в порядку зростання складності.

1) Колірні фільтри - якщо об'єкт істотно виділяється на тлі за кольором, можна підібрати відповідний фільтр.

2) Виділення та аналіз контурів - якщо ми знаємо, що об'єкт має форму, наприклад, кола, то можна пошукати кола на зображенні.

3) Зіставлення з шаблоном - у нас є зображення об'єкта, шукаємо в іншому зображенні області, що збігаються з цим зображенням об'єкта.

4) Робота з особливими точками - на картинці з об'єктом шукаємо особливості (наприклад, кути), які намагаємося порівняти з такими особливостями на іншому зображенні.

5) Методи машинного навчання - навчаємо класифікатор на картинках з об'єктом, у якийсь спосіб поділяємо зображення на частини, перевіряємо класифікатором кожну частину на наявність об'єкта.

Задача виявлення об'єктів на зображеннях є складною через такі причини:

− наявність різних умови освітленості, які залежать від типу, кількості та напрямку джерел світла;

− наявність різної роздільної здатності зображень, на яких знаходиться об’єкт, а також різні розміри самого об’єкта відносно зображення;

− наявність зашумленості зображення, яка обумовлена його якістю, а також можливі візуальні ефекти, які зменшують видимість об’єкта;

− наявність часткового перекриття об'єкта іншими об'єктами; необхідність локалізації і розпізнавання об’єкта, який може мати довільне положення у просторі.

Однак, існуючі системи виявлення об’єктів не завжди враховують перераховані особливості, що не дозволяє досягти прийнятного значення якості розпізнавання зображень.

Із розглянутих вище 5 груп методів виявлення об’єктів на зображенні найбільш перспективною є група методів машинного навчання, серед яких найбільшу ефективність має підгрупа нейромережевих методів.

Це пояснюється тим, що штучні нейронні мережі працюють на основі навчання на прикладах, а тому не потрібують якихось складних алгоритмів обробки та виділення корисної інформації із зображень, не потребують

формулювання якихось правил або критеріїв прийняття рішень. Все це виконує сама нейронна мережа, яка навчена на розпізнавання професійно підібраних прикладів тих чи інших зображень.

Виявлення об'єктів є досить популярним завданням комп'ютерного зору, яка включає ідентифікацію і виявлення об'єктів на зображеннях або відео. Це завдання є частиною багатьох додатків, наприклад, таких як безпілотні автомобілі, робототехніка, відеоспостереження і т. д. За минулі роки розроблено безліч алгоритмів та методів для пошуку об'єктів на зображеннях та їх положеннях. Найкраща якість виконання таких завдань досягається при використанні згорткових нейронних мереж:

Серед різних типів згорткових нейронних мереж найкраще зарекомендували себе для обробки зображень такі нейронні мережі:

- R-CNN;
- Mask R-CNN;
- Faster R-CNN;
- SSD (Single shot detectors);
- YOLO (You only look once).

Під час додаткових досліджень було виявлено, що Yolo, схоже, став стандартом, коли справа доходить до виявлення об’єктів з високою швидкістю і точністю. Тому YOLO (you only look once), створена в 2015 році, є однією з найпопулярніших архітектур нейронних мереж для таких завдань. З того часу з'явилося чимало версій даних алгоритмів. Останні випуски мережі призначені для таких завдань, як розпізнавання, виявлення та сегментація зображень.

Ми будемо розглядати архітектуру YOLO тільки для виявлення об'єктів на зображенні. В даному випадку мета алгоритму - передбачити клас об'єкта і намалювати обмежувальну рамку, яка визначає місце розташування об'єкта на вхідному зображенні.

Один із способів задач виявлення полягає у розбитті зображення на квадратні області, а потім класифікація цих областей на наявність об'єкта та класифікацію самого об'єкта. Таким чином, зображення проглядається двічі,

один раз для визначення областей де є об'єкт, а другий — для класифікації цього об'єкта. YOLO використовує інший підхід. Вихідне зображення стискається таким чином, щоб отримати квадратну матрицю, в кожній клітці якого записана інформація про наявність об'єкта та клас цього об'єкта на відповідній частині картинки. Для кожної комірки виводиться ймовірність класу, що визначається. Осередки, що мають ймовірність класу вище порогового значення, вибираються і використовуються для визначення розташування об'єкта на зображенні. Тобто YOLO переглядає картинку один раз, що суттєво збільшує швидкість обробки. Відрізняється високою швидкістю та точністю виявлення об'єктів. На виході роботи такої мережі ми хочемо отримати приблизно таке зображення (рис. 1.1).

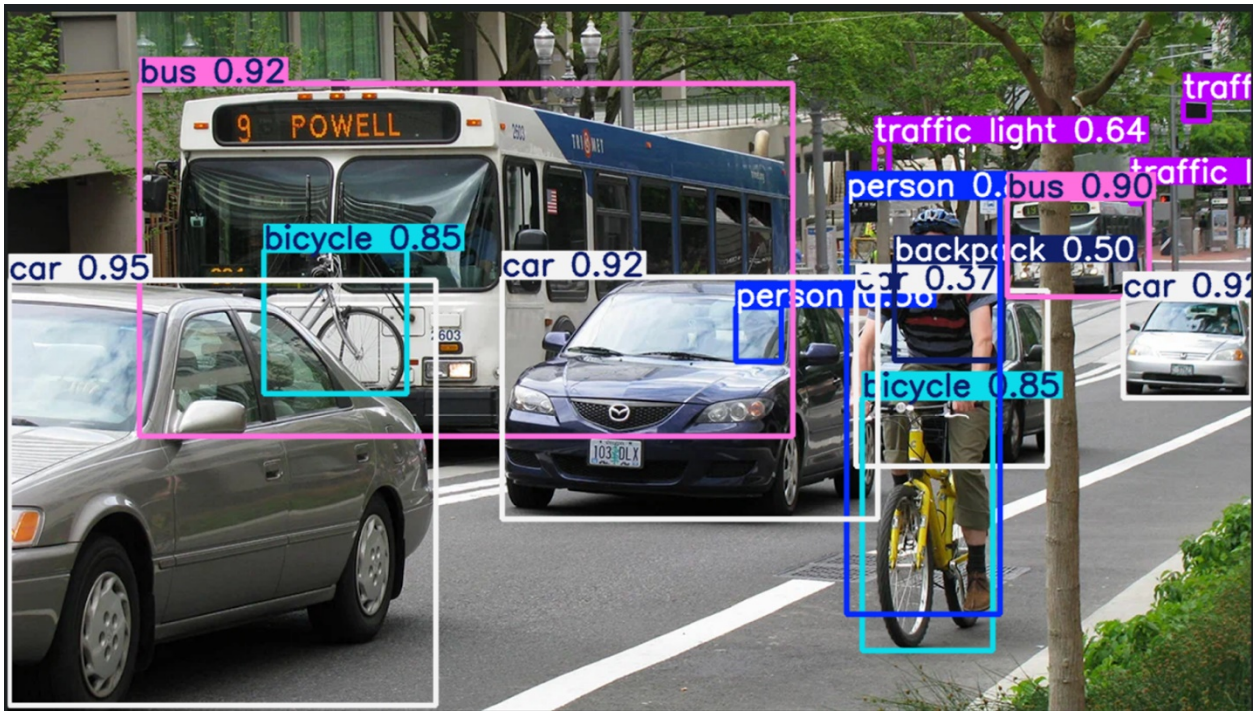


Рисунок 1.1 – Приклад зображення із виявленими та розпізнаними об’єктами

Таким чином, ми отримали зображення з позначенням виявлених об'єктів та значенням ймовірності приналежності до обраного класу.

Найбільш поширеними для виявлення об'єктів є моделі YOLOv5 та YOLOv7. Обидві версії YOLO безкоштовні від одного розробника Ultralytics,

що добре показують себе на порівняльних тестах і реалізовані на сучасній бібліотеці глибокого навчання PyTorch. Модель YOLOv7 позиціонується як більш нова та сучасна в порівнянні з 5-ю версією, хоча обидві версії активно розвиваються.

1.3 Огляд аналогів програмного забезпечення виявлення риб на підводних зображеннях

У роботі [1] описують кілька варіантів архітектури YOLO для виявлення риби. Основною моделлю є YOLOFish, надійна модель виявлення риби, опублікована в грудні 2022 року та прийнята в Екологічній інформатиці. YOLO-Fish є однією з найпоширеніших моделей виявлення риби, які використовуються сьогодні. Модель YOLO-Fish була навчена на наборі даних DeepFish і Ozfish, і було виявлено, що вона досить добре працює на DeepFish (приблизно 95% точності), але не так добре узагальнюється на наборі даних Ozfish (82% точності). Набір даних DeepFish містить зображення більшої риби із середнім вмістом 3-4 риби на кадр, тоді як набір даних Ozfish містить зображення дрібної риби із середнім вмістом 25 рибин на кадр. У статті [1] описано дві різні версії YOLO-Fish: YOLO-Fish1 і YOLO-Fish-2. Архітектура YOLO-Fish-1 зображена на рисунку 1.2, де шкала 1 відповідає виявленню великих об’єктів, шкала 2 відповідає виявленню середніх об’єктів, а шкала 3 відповідає виявленню малих об’єктів.

Модель використовує кілька шарів підвищення дискретизації для виділення ознак на основі масштабу та використовує просторову піраміду об’єднання (SPP - Spatial Pyramid Pooling), яка складається з 4 шарів макспулінгу, об’єднаних разом для покращення виділення ознак.

У роботі [2] використовували комбінацію Gaussian Mixture Models (GMM) і YOLO для виявлення та класифікації риби в двох підводних відеодатасетах: опорний датасет LifeCLEF 2015 із репозиторію Fish4Knowledge та датасет, зібраний Університетом Західної Австралії

(UWA). Вони досягли F-оцінки виявлення риби 95,47% і 91,2% на цих двох датасетах відповідно. У той час як GMM і оптичний потік допомогли виділити часові ознаки, модель значною мірою покладалася на YOLO для виявлення статичної риби, як ми маємо в нашому наборі даних зображень підводної риби.

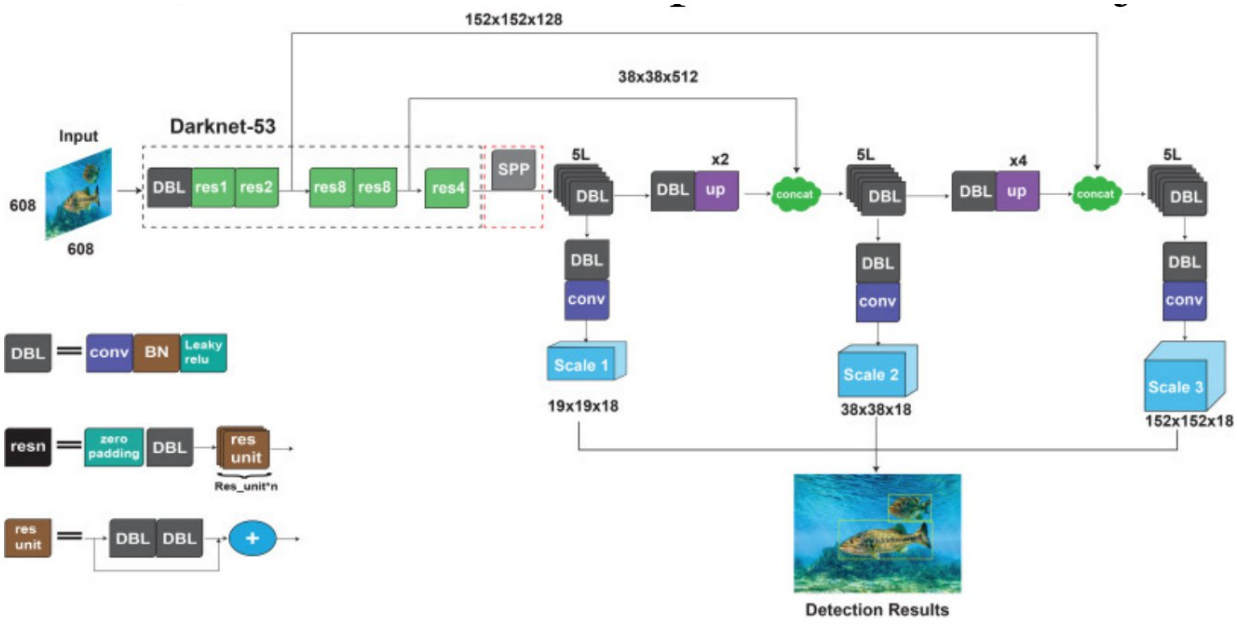


Рисунок 1.2 – Архітектура YOLO-Fish-1

У роботі [3] описують використання YOLO-v3 для виявлення та класифікації підводних об’єктів на зображеннях сонара бокового сканування, який все частіше використовується для підводного пошуку. Вони навчили модель на наборі з 7000 підводних зображень SSS і провели серію експериментів за допомогою мережі YOLO-v3. Оцінюючи використання mAP із пороговими значеннями 0,5, 0,55 і 0,6, вони виявили, що модель ефективно виявляє об’єкти в підводних даних, зібраних у реальному середовищі. Так само ми будемо використовувати порогове значення 0,5 і mAP як нашу метрику, оскільки ми зосереджені виключно на виявленні.

У роботі [4] оцінили YOLO-v3 для виявлення, підрахунку та класифікації риби з підводних відео з набору даних Ocqueoc River DIDSON. Вони тренували модель YOLOv3 із попередньо натренованими вагами від

ImageNet. Ми також будемо використовувати попередньо натреновані ваги для тренування нашої моделі. Початковими результатами YOLO-v3 було mAP від 0,00 до 0,66 для різних класів, розраховане з IoU 0,5. Однак після навчання з доповненими даними mAP збільшився до діапазону від 0,31 до 0,86 у різних класах. Ми також будемо використовувати аугментовані (розширені) дані для навчання нашої моделі.

У роботі [5] обговорюють використання перенесення навчання (Transfer Learning) для ініціалізації своїх ваг для навчання на своїх наборах даних Voith Hydro, Wells Dam та Igiugig. Однак вони використовують ваги, які були навчені виявляти різні класи, починаючи від котів і закінчуючи автомобілями, але не виявляють рибу. У цій роботі досліджується передача навчання на основі попередньо навчених ваг, призначених для виявлення риби.

У роботі [6] описують використання кодерів Co-scale conv-attentional image Transformer (CoAT), щоб перевершити кодери на основі CNN у виявленні риби на підводних зображеннях. Вони використовували три різні набори даних: DeepFish, Seagrass і Youtube-VOS. У статті було виявлено, що кодер на основі трансформера краще виявляв рибу, що перекривається, там, де кодери на основі CNN не справлялися. Однак було виявлено, що середня точність методів на основі YOLO краща.

У роботі [7] обговорюється модифікації моделі DETR, які дозволяють їй краще працювати на підводних зображеннях. У декодер вбудований механізм запам’ятовування запитів, який можна навчити, щоб пом’якшити вплив шуму, наприклад дифракції світла та зважених частинок у воді, які можуть зробити зображення розмитими. Вони також використовують AdaptFFN для відстеження дрібних деталей у малих об’єктах. До кожного кодера та декодера також додається легкий адаптерний модуль. Незважаючи на це, було виявлено, що моделі на основі YOLO працюють краще, ніж цей підхід щодо виявлення риби.

У роботі [8] порівнювали Faster R-CNN ResNet50, YOLO-v3 і SSD MobileNetV2 щодо виявлення підводної риби та класифікації на наборі даних

Fish4Knowledge. Вони виявили, що YOLO-v3 має найкраще значення повноти (recall), а SSD MobileNetV2 — найкраще значення точності (precision).

У роботі [9] використовували YOLO-v5 для виявлення риби на Ozfish і DeepFish. Вони досягли точності 0,898 і повноти 0,699 на Ozfish, з яким і ми працюємо. Однак у нашій реалізації ми будемо використовувати вдосконалену версію YOLO, YOLO-v7.

Також було переглянуто роботу [10], у якій описано підхід до виявлення підводної риби за допомогою Fast RCNN на датасеті ImageCLEF, який досяг mAP 81,4%.

1.4 Висновок

У розділі описано детальну постановку задачі виявлення риб на підводних зображеннях. Також розглядаються різні методи розв'язання задачі виявлення об’єктів на зображеннях і оцінюється їх застосовність до завдання виявлення риб на підводних зображеннях. Серед таких методів розпізнавання об`єктів як колірні фільтри, виділення та аналіз контурів, зіставлення з шаблоном, робота з особливими точками, методи машинного навчання як найбільш перспективний і застосовний до даної задачі було обрано метод машинного навчання, а точніше нейромережевий метод. Було показано, що з різних типів нейронних мереж найбільше підходить для вирішення поставленої задачі саме згорткові нейронні мережі. Крім цього, було оглянуто аналоги розроблюваного модуля.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Архітектура нейронної мережі YOLOv7

Модель YOLO (You Only Look Once) v7 є одною з останніх в сімействі моделей YOLO. Моделі YOLO є одноетапними детекторами об’єктів. Архітектура нейронної мережі YOLO заснована на FCNN (Fully Connected Neural Network, повнозв`язна нейронна мережа). Однак версії на основі Transformer також нещодавно були додані до сімейства YOLO.

Структура нейронної мережі YOLO складається з трьох основних компонентів.

- Хребет (Backbone),
- Шия (Neck),
- Голова (Head).

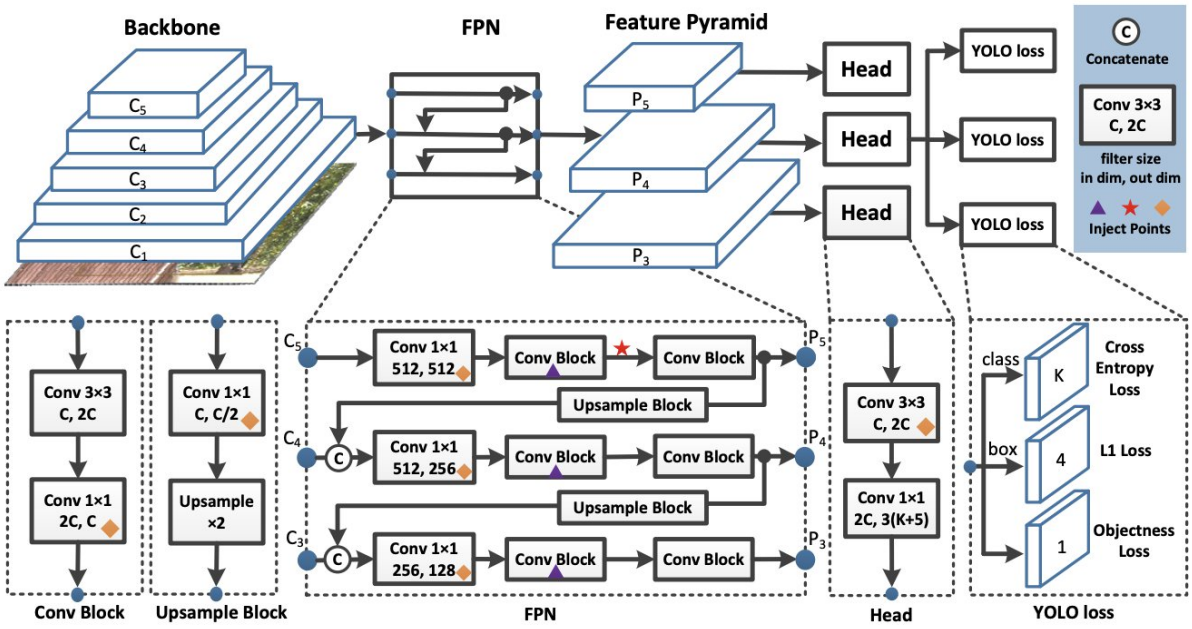


Рисунок 2.1 – Структура мережі YOLOv7

Хребет, в основному, виділяє основні характеристики зображення та передає їх у голову через шию. Шия збирає карти ознак, отримані хребтом, і створює піраміди ознак. Нарешті, голова складається з вихідних шарів нейронної мережі, які мають вихідні детектори.

Хребет YOLO — це згорткова нейронна мережа, яка об’єднує пікселі зображення, щоб сформувати об’єкти з різною деталізацією. Хребет зазвичай попередньо навчений на наборі даних класифікації, зазвичай ImageNet.

Шия YOLO (FPN на рис. 2.1) поєднує та змішує представлення згорткових шарів ConvNet перед передачею до голови передбачення.

Голова YOLO - це частина мережі, яка робить обмежувальну рамку та прогнозування класу. Вона керується трьома функціями втрат YOLO для класу, рамки та об’єктності.

YOLOv7 покращує швидкість і точність, запроваджуючи кілька архітектурних реформ. Подібно до Scaled YOLOv4, хребет YOLOv7 не використовує попередньо навчений хребет ImageNet. Натомість моделі навчаються повністю з використанням набору даних COCO. Подібності можна очікувати, оскільки YOLOv7 написаний тими ж авторами, що й Scaled YOLOv4, який є розширенням YOLOv4. В структуру YOLOv7 були внесені такі основні зміни:

- удосконалення архітектури;
- E-ELAN (розширена ефективна мережа агрегації шарів);
- масштабування моделі для моделей на основі конкатенації;
- тренувальний BoF (Bag of Freebies, мішок безкоштовних речей);
- планова репараметризована згортка;
- грубий для допоміжних і тонкий для основних втрат.

Удосконалення архітектури.

Архітектура походить від YOLOv4, Scaled YOLOv4 і YOLO-R. Використовуючи ці моделі як основу, були проведені подальші експерименти для розробки нових і вдосконалених YOLOv7.

E-ELAN (Extended Efficient Layer Aggregation Network).

E-ELAN є обчислювальним блоком хребта YOLOv7 (див. рис. 2.2). Він черпає натхнення з попередніх досліджень ефективності мережі. Він був розроблений шляхом аналізу наступних факторів, які впливають на швидкість і точність.

- Вартість доступу до пам'яті;
- Співвідношення каналів введення/виведення;
- Поелементна операція;
- Активації;
- Градієнтний шлях.

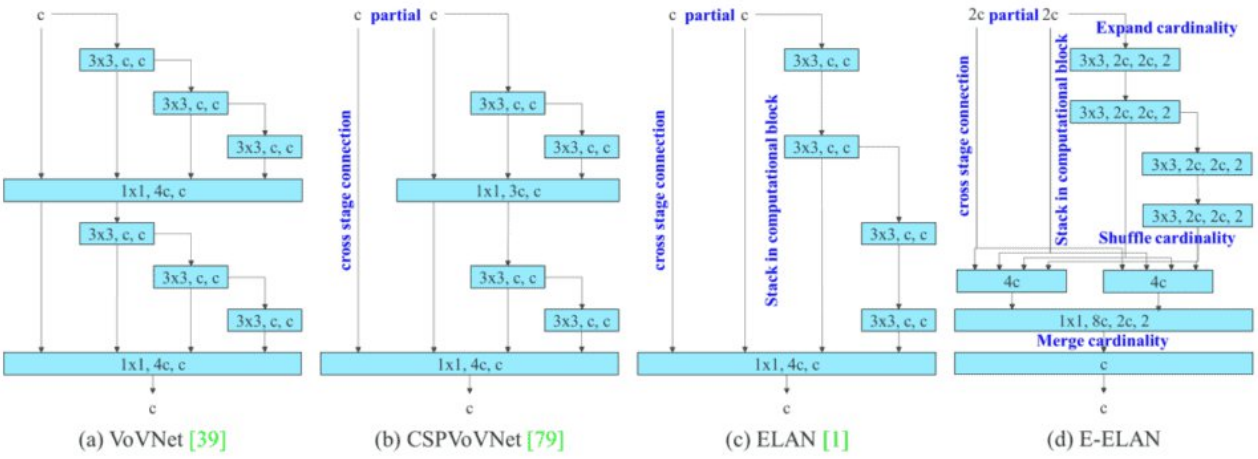


Рисунок 2.2 – E-ELAN і попередня робота над максимальною ефективністю шару

Простіше кажучи, архітектура E-ELAN дозволяє інфраструктурі краще навчатися. Він заснований на обчислювальному блоці ELAN.

Масштабування складної моделі в YOLOv7.

Для різних програм потрібні різні моделі. У той час як деяким потрібні високоточні моделі, інші надають перевагу швидкості. Масштабування моделі виконується, щоб задовольнити ці вимоги та зробити її придатною для різних обчислювальних пристроїв.

При масштабуванні розміру моделі враховуються такі параметри:

- Роздільна здатність (розмір вхідного зображення);
- Ширина (кількість каналів);
- Глибина (кількість шарів);
- Етап (кількість пірамід ознак).

NAS (пошук мережевої архітектури) є широко використовуваним методом масштабування моделі. Він використовується дослідниками для повторного перегляду параметрів, щоб знайти найкращі коефіцієнти масштабування. Однак такі методи, як NAS, виконують масштабування залежно від параметрів. Коефіцієнти масштабування в цьому випадку незалежні.

Автори статті YOLOv7 показують, що її можна додатково оптимізувати за допомогою підходу масштабування складної моделі. Тут ширина та глибина масштабуються узгоджено для моделей на основі конкатенації (рис. 2.3).

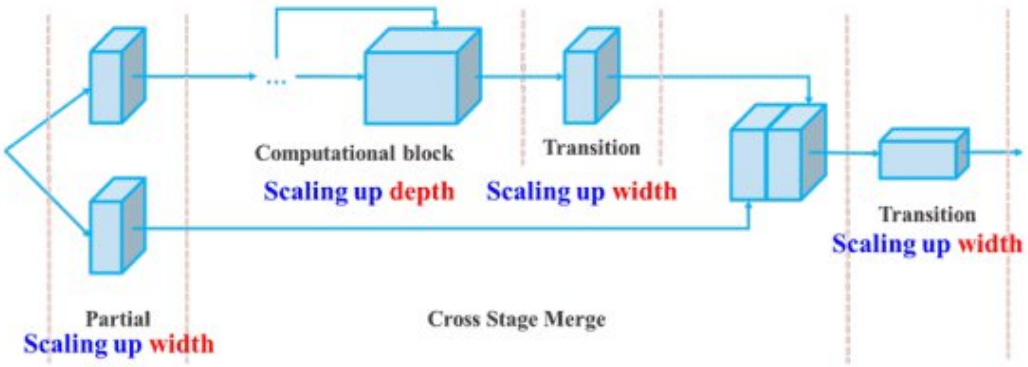


Рисунок 2.3 – Складене масштабування YOLOv7

Сумка безкоштовних речей (BoF), яку можна навчити в YOLOv7.

BoF або Bag of Freebies — це методи, які підвищують продуктивність моделі без збільшення вартості навчання. YOLOv7 представив такі методи BoF.

Запланована репараметризована згортка.

Повторна параметризація (репараметризація) — це техніка, яка використовується після навчання для покращення моделі. Це збільшує час

навчання, але покращує результати роботи мережі. Два типи повторної параметризації використовуються для фіналізації моделей: рівень моделі та рівен ансамбля модулів.

Повторну параметризацію на рівні моделі можна виконати двома способами.

1) Використовуючи різні навчальні дані, але однакові налаштування, навчити кілька моделей. Потім усереднити їх ваги, щоб отримати остаточну модель.

2) Взяти середнє значення ваг моделей у різні епохи.

Останнім часом репараметризація на рівні модуля набула значного поширення в дослідженнях. У цьому методі процес навчання моделі розбивається на кілька модулів. Виходи об’єднуються для отримання остаточної моделі. Автори статті YOLOv7 показують найкращі можливі способи виконання ансамблю на рівні модуля (рис. 2.4).

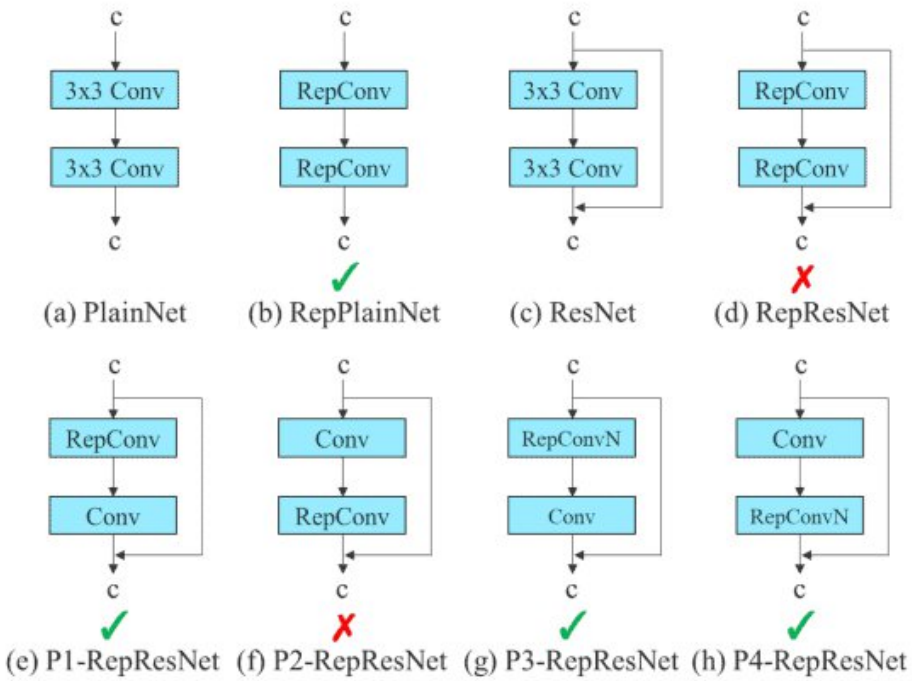


Рисунок 2.4 – Варіанти повторної параметризації

На рисунку 2.4 шар згортки 3×3 обчислювального блоку E-ELAN замінено на шар RepConv. Експерименти проводились, перемикаючи або замінюючи позиції RepConv, 3×3 Conv та підключення Identity. Стрілка залишкового обходу, показана на рисунку 2.4, є підключенням ідентифікатора. Це не що інше, як згортковий шар 1×1. Ми можемо бачити конфігурації, які працюють, і ті, які не працюють.

Включаючи RepConv, YOLOv7 також виконує повторну параметризацію на Conv-BN (пакетна нормалізація згортки), OREPA (онлайн-згорточна повторна параметризація) і YOLO-R для отримання найкращих результатів.

Грубий для допоміжних і тонкий для основних втрат.

Як відомо, архітектура YOLO складається з хребта, шиї та голови. Голова містить передбачувані результати. YOLOv7 не обмежується однією головою. Вона має кілька голів, щоб виконувати декілька завдань.

Однак, це не перший раз, коли представлено багатоголовий фреймворк. Глибоке кероване навчання - техніка, яку використовують моделі DL, використовує кілька голів. У YOLOv7 голова, відповідальна за кінцевий результат, називається основною головою (Lead Head). А голова, яка використовується для навчання середніх шарів, називається допоміжною головою (Auxiliary Head).

За допомогою втрати асистента оновлюються ваги допоміжних голів. Це дозволяє здійснювати глибоке кероване навчання, і модель навчається краще. Ці поняття тісно пов’язані з основною головою (Lead Head) і розпорядником міток (Label Assigner).

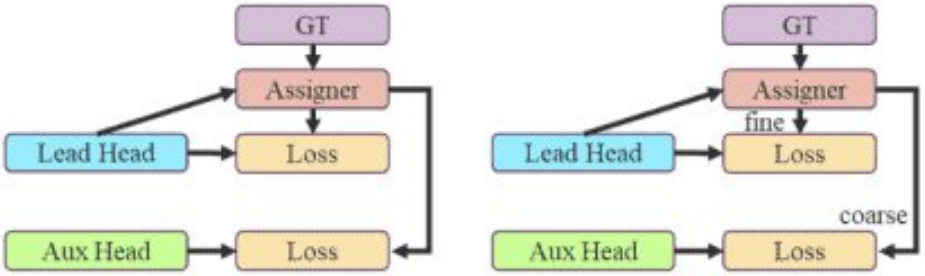
Label Assigner — це механізм, який порівнює результати передбачення мережі та дійсні результати, а потім призначає м’які мітки. Важливо зауважити, що програма призначення міток генерує м’які та грубі мітки замість жорстких.

Пристрій для присвоєння міток, керований основною головою та пристрій для присвоєння міток від грубого до тонкого, керований основною

головою (Lead Head Guided Label Assigner and Coarse-to-Fine Lead Head Guided Label Assigner).

Пристрій для присвоєння міток, керований основною головою (рис. 2.5), охоплює наступні три концепції.

- основна голова;
- допоміжна голова;
- пристрій для присвоєння міток.



а) б)

Рисунок 2.5 – Пристрій для присвоєння міток, керований основною головою (а) та пристрій для присвоєння міток від грубого до тонкого, керований основною головою (б)

Основна голова мережі YOLOv7 прогнозує кінцеві результати. На основі цих остаточних результатів генеруються м’які мітки. Важливою частиною є те, що втрати обчислюються як для основної голови, так і для допоміжної голови на основі тих самих м’яких міток, які генеруються. Зрештою, обидві голови навчаються використовувати м’які мітки. Це показано на лівому зображенні на рисунку 2.5.

Тут виникає питання: «Чому м’які мітки?». Причина для цього полягає в тому, що основна голова має відносно сильну здатність до навчання. Отже, м’яка мітка, згенерована з неї, повинна більш репрезентативно відображати розподіл і кореляцію між вихідними даними та цільовими даними. Дозволяючи дрібнішій допоміжній голові безпосередньо вивчати інформацію,

яку вивчила основна голова, основна голова зможе більше зосередитися на вивченні залишкової інформації, яка ще не була вивчена.

Тепер переходимо до міток від грубого до тонкого, як показано на рисунку 2.5б. У наведеному вище процесі генеруються два набори м’яких міток.

- тонка мітка для тренування основної голови;
- набір грубих міток для тренування допоміжної голови.

Тонкі мітки такі ж, як і безпосередньо згенеровані м’які мітки. Однак більше сіток розглядаються як позитивні цілі для створення грубих міток. Це досягається шляхом послаблення обмежень процесу призначення позитивної мітки.

2.2 Розробка моделі виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7

У нашому проекті використовується перенесення навчання (Transfer Learning) для створення моделі виявлення риби спеціально для меншої риби і оцінюється його позитивний вплив на вирішення цієї задачі.

Перенесення навчання – це метод машинного навчання, за якого знання, отримані з моделі, навченої на одному завданні, повторно використовуються для покращення навчання на пов'язаному, іншому завданні. Замість навчання нової моделі з нуля, що є ресурсомістким, трансферне навчання використовує знання попередньо навченої моделі, прискорюючи процес навчання та часто призводячи до кращої продуктивності.

Опис моделі.

Було використано YOLOv7 [12] як нашу основну модель для цього проекту. Було використано існуючу кодову базу [11], яка мала більшість налаштувань для навчання YOLOv7, яку ми використовували та модифікували для завантаження, обробки та навчання на нашому власному наборі даних. YOLOv7 — це найкращий сучасний детектор об’єктів, який загалом

перевершив усі інші детектори об’єктів як за швидкістю, так і за точністю. Це одноетапний детектор об’єктів, який передбачає обмежувальні рамки та ймовірності класу для кожного вхідного зображення.

Хребет мережі.

Для виділення ознак YOLOv7 пропускає вхідні зображення через частину нейромережі, яка називається хребет. Вхідні зображення спочатку проходять через ряд шарів CBS (який є послідовністю згорткового шару, пакетної нормалізації та функції активації SiLU - Sigmoid Linear Unit function). Після цього вихідні дані цих шарів пропускаються через чергування шарів ELAN і MPConv. Схема хребта мережі показана на рисунку 2.6.

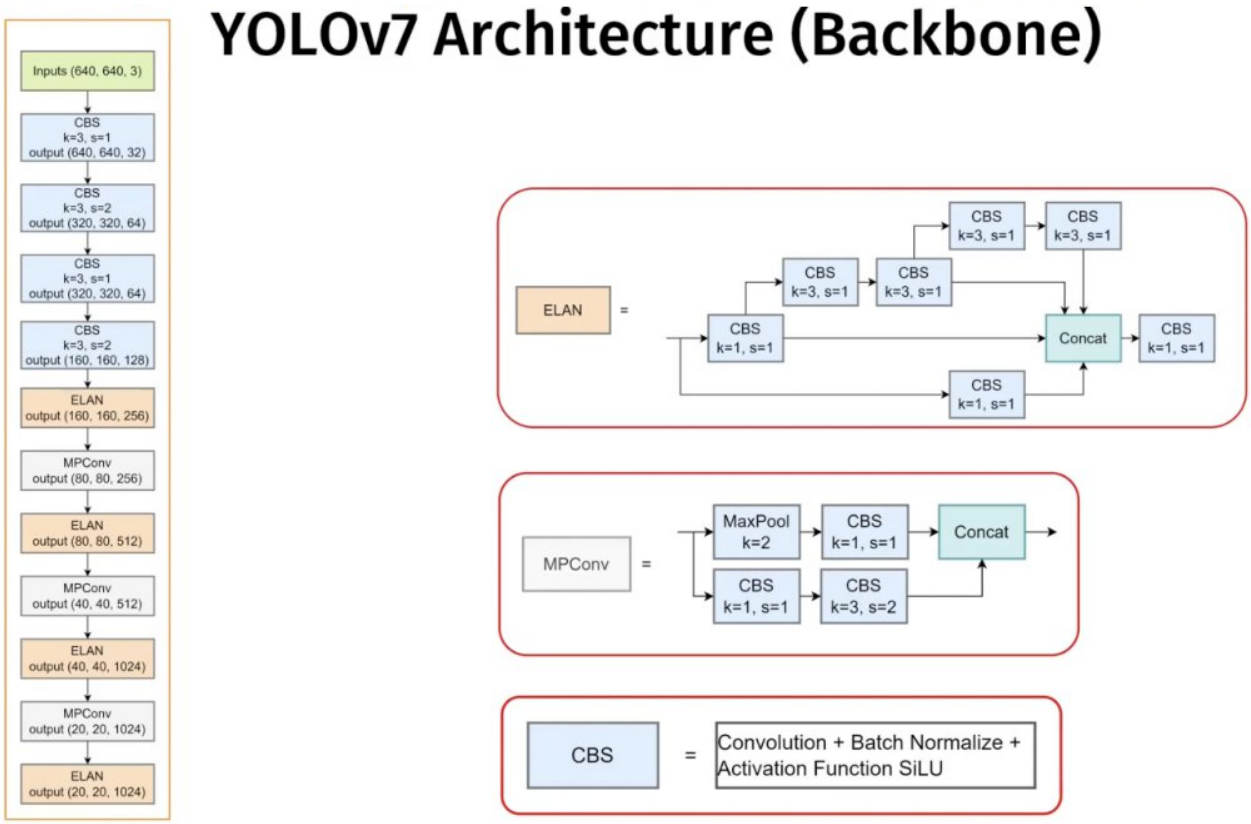


Рисунок 2.6 – Схема хребта мережі YOLOv7

ELAN (Efficient Layer Aggregation Network) — це структура для створення карт ознак шляхом комбінування результатів різних шарів, одночасно контролюючи найкоротший шлях градієнта, щоб точність не

погіршувалася під час застосування масштабування моделі. Масштабування моделі стосується вивчення кількості каналів і кількості шарів у кожному обчислювальному блоці. YOLOv7 використовує масштабування моделі в 3-х її частинах: хребті, шиї та голові.

Шия.

Мета шиї — обробити риси, виділені в хребті. Шия використовує модуль CSPSPP (мережа Cross Stage Partial Network with Spatial Pyramid Pooling (SPP) block) і PAN (Path Aggregation Network), а також блоки ELAN-H. Результатом є три різні рівні оброблених вхідних ознак. Повна схема шиї показана на рисунку 2.7.

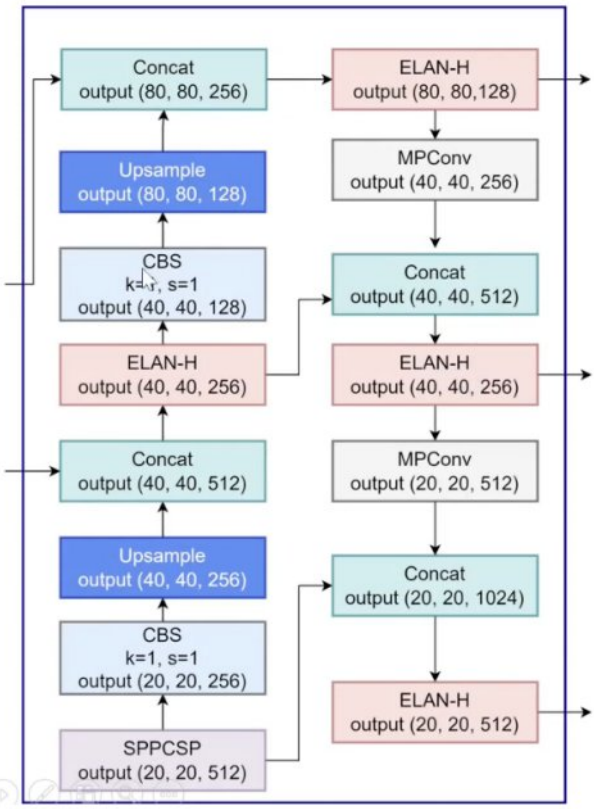


Рисунок 2.7 –Повна схема шиї мережі YOLOv7

Модуль CSPSPP використовує SPP (Spatial Pyramid Pooling), шар пулінгу (об’єднання), який вставляється між блоками CBS і дозволяє моделі обробляти вхідні дані змінного розміру. SPP складається з кількох шарів макспулінгу з бункерами різного розміру (5, 9 і 13), а вихідні дані цих шарів

об’єднані разом у вихідні дані фіксованого розміру. Зауважимо, що модуль CSPSPP (рис. 2.8) надсилає вхідні ознаки двома шляхами: один подається через шар SPP, а інший подається через простий згортковий шар 1 на 1 і об’єднується в кінці. Це робиться, щоб запобігти дублюванню інформації про градієнт, зменшити складність і зберегти деталі.

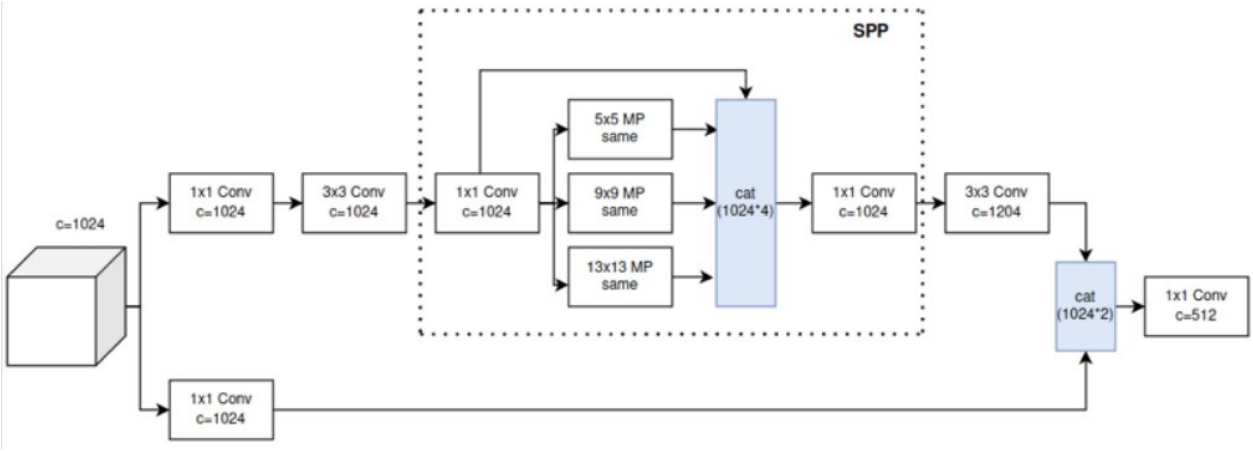


Рисунок 2.8 – Схема модуля CSPSPP шиї мережі YOLOv7

PAN покращує здатність моделі локалізувати інформацію, оптимізуючи шлях потоку інформації знизу вгору, щоб переконатися, що модель може використовувати детальні ознаки в різних масштабах. Деталі вищого рівня беруться з більш глибоких рівнів хребта та об’єднуються з деталями нижчого рівня з подальшої мережі. Це робиться за допомогою підвищення дискретизації (upsampling), що показано на рис.2.7), де роздільна здатність ознак вищого рівня збільшується, а потім об’єднується з картами ознак із попередніх шарів мережі.

Голова мережі.

Голова отримує три шкали вхідних ознак від шиї, які використовує для прогнозування малих, середніх і великих об’єктів. YOLOv7 використовує попередньо визначені опорні рамки як відправну точку для прогнозування обмежувальних рамок, і для кожного масштабу виділено три опорні рамки. Під час навчання для кожного блоку прив’язки модель вчиться прогнозувати

положення обмежувальної рамки, щоб краще відповідати об’єктам, які вона виявляє.

2.3 Функції втрат

Оскільки YOLOv7 передбачає як координати обмежувальної рамки, так і ймовірності класу, вона використовує кілька функцій втрат.

CIoU (Complete Intersection over Union) Loss використовується для вимірювання того, наскільки прогнозовані обмежувальні рамки відповідають справжнім обмежувальним рамкам. Ця функція втрат враховує перекриття між обмежувальними рамками, відстань між центральними точками та співвідношення сторін:

$$\text{CIoU Loss} = 1 - \text{IoU} + \frac{\rho^2(\mathbf{b}, \mathbf{b}^g)}{c^2} + \alpha v$$

де:

- IoU – це перетин між прогнозованим блоком **b** і істинним базовим блоком $\mathbf{b^g}$.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

- $\rho(\mathbf{b}, \mathbf{b^g})$ — евклідова відстань між центрами прогнозованого прямокутника та істинного прямокутника::

$$\rho(\mathbf{b}, \mathbf{b}^g) = \sqrt{(x - x^g)^2 + (y - y^g)^2}$$

де (x,y) і (x^g,y^g) – координати центру прогнозованого та істинного прямокутників відповідно.

- c — діагональна довжина найменшого обмежувального прямокутника, що охоплює як передбачуваний, так і істинний блоки:

$$c = \sqrt{(w + w^g)^2 + (h + h^g)^2}$$

де w і h — ширина та висота прогнозованого прямокутника, а w^g, h^g — ширина та висота істинного прямокутника.

- v — показник співвідношення сторін:

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^g}{h^g} - \arctan\frac{w}{h}\right)^2$$

- α – позитивний компромісний параметр:

$$\alpha = \frac{v}{(1 - \mathrm{IoU}) + v}$$

Втрата об’єктності стосується здатності моделі розрізняти фон і об’єкти. Це сума втрат бінарної перехресної ентропії для кожної обмежувальної рамки:

$$L_{\text{obj}} = -\left[t_i \log(p_i) + (1 - t_i)\log(1 - p_i)\right]$$

де:

- p_i – передбачувана оцінка об’єктності для і-ої обмежувальної рамки.
- t_i — основна мітка істинності для об’єктності (1, якщо об’єкт присутній, 0, якщо інакше).

Загальна втрата об’єктності з урахуванням усіх обмежувальних рамок становить:

$$\text{Objectness Loss} = \sum_i [-t_i \log(p_i) - (1 - t_i) \log(1 - p_i)]$$

Нарешті, Class Prediction Loss використовує двійкову перехресну втрату ентропії, щоб гарантувати, що кожному обмежувальному прямокутнику призначено правильний клас.

Втрата передбачення класу для окремої обмежувальної рамки та окремого класу визначається як:

$$L_{\text{cls},ij} = -[t_{ij} \log(p_{ij}) + (1 - t_{ij}) \log(1 - p_{ij})]$$

де:

- p_{ij} – прогнозована ймовірність для і-ої обмежувальної рамки, що належить до j-го класу.
- t_{ij} — основна мітка істинності для і-ої обмежувальної рамки та j-го класу (1, якщо об’єкт належить до класу, 0 в протилежному випадку).

Загальна втрата передбачення класу з урахуванням усіх обмежувальних рамок і всіх класів становить:

$$\text{Class Prediction Loss} = \sum_i \sum_j [-t_{ij} \log(p_{ij}) - (1 - t_{ij}) \log(1 - p_{ij})]$$

2.4 Розробка алгоритму роботи програмного модуля виявлення риб на підводних зображеннях

Відповідно до мети роботи та постановки задачі було розроблено алгоритм програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7, представлений на рисунку 2.9.

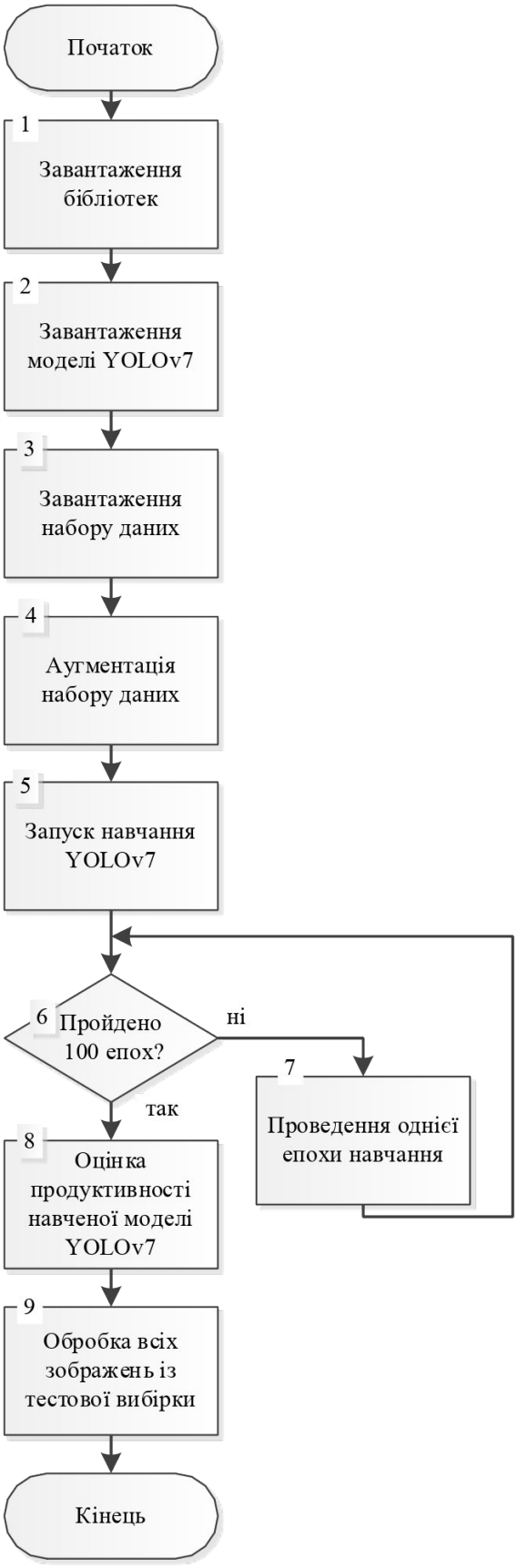


Рисунок 2.9 – Схема алгоритму роботи програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7

Розроблюваний програмний модуль виявлення риб на підводних зображеннях, якщо коротко, має виконувати такі дії: завантажити бібліотеки і модель YOLOv7, завантажити спеціальний набір даних у форматі YOLOv7, розширити цей набір шляхом аугментації даних, провести навчання YOLOv7, оцінити продуктивність навченої YOLOv7, виконати інференс YOLOv7 на тестових зображеннях.

Першим кроком в програмному модулі виявлення риб на підводних зображеннях буде завантаження необхідних бібліотек (блок 1), а другим кроком буде завантаження з репозиторію моделі згорткової нейронної мережі YOLOv7 (блок 2). Третім кроком будее завантаження набору даних зображень продуктів харчування Ozfish (блок 3), який містить зображення саме мілких риб, знятих в поганих умовах освіиленості під водою через каламутність води..

Далі переходимо до процесу аугментації даних з набору Ozfish (блок 4). Набір даних і процес аугментації детально описані в п. 3.2. Після цього запускаємо процес навчання згорткової нейронної мережі YOLOv7 (блок 5). Перед навчанням дані розділяються у співвідношенні 70:20:10 на тренувальні, валідаційні та тестувальні відповідно.

Умовний блок 6 перевіряє чи пройшло вже 100 епох навчання (взагалі кількість епох навчання в процесі дослідження результатів роботи програми змінювалась від 100 до 300). Блок 7 проводить 1 епоху навчання, параметри якого зазначені на початку розділу 4, навчання пакетне, розмір пакету 64.

Якщо вже пройдено 100 епох навчання, то переходимо до оцінки продуктивності (блок 8) навченої моделі згорткової нейронної мережі YOLOv7. Продуктивність оцінюється за такими показниками як середня точність та середня влучність виявлення риб на підводних зображеннях. Результати оцінювання параметрів продуктивності розробленого модуля при різних варіантах навчання моделі згорткової нейронної мережі YOLOv7 наведено у таблицях 4.1 та 4.2.

Далі переходимо до обробки всіх зображень із тестової вибірки. Завантажуємо зображення із папки з тестовими вхідними зображеннями,

обробляємо його на нейромережі. Після цього на зображеннях з’являються обмежувальні рамки, назва класу риб та коефіцієнт упевненості як показано на рисунку 4.1 в розділі 4.

2.5 Висновок

У розділі було проаналізовано архітектуру згорткової нейронної мережі YOLOv7 і на її основі розроблено модель виявлення риб на підводних зображеннях. У розробленій моделі використовується перенесення навчання (Transfer Learning) для покращення виявлення риби спеціально для меншої риби і в умовах каламутної води. Описано принципи роботи та навчання нейронної мережі YOLOv7 для модуля виявлення риб. Розроблено алгоритм роботи програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови програмування та спеціалізованих бібліотек

Для реалізації програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі було обрано мову програмування Python [13].

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Структури даних високого рівня разом із динамічним зв'язуванням та динамічною семантикою роблять її придатною для швидкої розробки програм, а також як засіб поєднання наявних можливостей та компонентів. Python підтримує модулі та пакети модулів, що сприяє повторному використанню коду. Інтерпретатор мови Python та стандартні бібліотеки доступні як у вихідній, так і у скомпільованій формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, аспектно-орієнтована, процедурна та функціональна.

Тобто, Python - це сучасна високорівнева мова програмування, яка привертає увагу розробників простотою, читабельністю та елегантністю. Завдяки простому синтаксису, Python має потужні функції, які задовольняють потреби як початківців, так і досвідчених програмістів-розробників.

Python застосовується для різних задач, у тому числі для веб-розробки, наукових досліджень, аналізу даних, штучного інтелекту, робототехніки та багато чого іншого.

Також для програмної реалізації модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7 нам

знадобляться спеціалізовані бібліотеки PyTorch, NumPy, OpenCV та Matplotlib.

PyTorch — відкрита бібліотека машинного навчання на основі бібліотеки Torch, що застосовується для задач комп'ютерного бачення та обробки природної мови [14]. Розробляє її переважно група дослідження штучного інтелекту компанії Facebook. Вона є вільним та відкритим програмним забезпеченням, що випускають під ліцензією Modified BSD. І хоча інтерфейс Python є більш відшліфованим, і головним зосередженням розробки, PyTorch також має зовнішній інтерфейс і для C++.

PyTorch — це пакет Python, який надає дві високорівневі функції:

- Тензорні обчислення (подібно до NumPy) з потужним прискоренням GPU,
- Глибокі нейронні мережі, побудовані на основі стрічкової системи автоградації.

Оскільки у згорткових нейронних мережах використовують багато операцій множення векторів на матриці, то для цієї роботи стане у пригоді бібліотека NumPy [15]. NumPy (скорочення від Numerical Python) — це бібліотека з відкритим кодом для мови програмування Python. Вона характеризкється такими можливостями:

- підтримання багатовимірних масивів (включно з матрицями);
- підтримання математичних функцій високого рівня, що призначені для роботи з багатовимірними масивами.

Математичні алгоритми, що реалізовані інтерпретованими мовами програмування (наприклад, Python), зачасту функціонують повільніше за алгоритми, які реалізовано компільованими мовами (напр., Фортран, C, Java). Бібліотека NumPy містить реалізації обчислювальних алгоритмів (у вигляді функцій та операторів), які оптимізовані для роботи з багатовимірними масивами. У результаті довільний алгоритм, який може бути виражений у вигляді послідовності операцій над масивами (матрицями) та реалізований із

використанням NumPy, працює так само швидко, як і еквівалентний йому код у MATLABi.

Так як при проектуванні програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі ми використовуємо набір даних підводних зображень для навчання та тестування згорткової нейронної мережі, то нам стане у пригоді бібліотека OpenCV.

OpenCV (скорочення від Open Source Computer Vision Library) - це потужна та універсальна бібліотека з відкритим вихідним кодом, що широко використовується в штучному інтелекті (ІІ) та машинному навчанні (МГО). Вона надає широкий набір інструментів та алгоритмів, спеціально розроблених для завдань комп'ютерного зору (CV) у реальному часі, обробки зображень та аналізу відео. Для спеціалістів з машинного навчання OpenCV є незамінним набором інструментів для роботи з візуальними даними, дозволяючи вирішувати завдання від базового завантаження та маніпулювання зображеннями до складного розуміння сцен. Його відкритий вихідний код, який підтримується OpenCV.org, сприяє формуванню великої спільноти та постійному розвитку, що робить її наріжною технологією в цій галузі. Вона легко доступна на різних платформах, включаючи Windows, Linux, macOS, Android та iOS, та пропонує інтерфейси для таких мов, як Python, C++, Java та MATLAB.

OpenCV відіграє важливу роль у конвеєрі штучного інтелекту та машинного навчання, особливо при роботі з візуальними даними. Вона надає фундаментальні інструменти для попередньої обробки даних - найважливішого кроку, що передує подачі зображень або відео моделі машинного навчання. Загальні етапи попередньої обробки, що виконуються OpenCV, включають зміну розміру, перетворення колірного простору (наприклад, BGR RGB, що часто необхідно для моделей, навчених на конкретних колірних порядках), придушення шуму за допомогою фільтрів типу гауссового розмиття, а також застосування різних перетворень для

поліпшення якості зображення або вилучення відповідності. Така попередня обробка істотно впливає продуктивність моделей глибокого навчання (DL).

Оскільки при проектуванні програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі ми виконуємо побудову обмежувальних рамок навколо зображень риб, то нам знадобитьсч бібліотека Matplotlib [16].

Matplotlib – це комплексна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій на мові програмування Python. Matplotlib має такі функціональні можливості:

- Створення сюжетів для публікацій.
- Створення інтерактивних фігур, які можна масштабувати, панорамувати, оновлювати.
- Налаштування візуального стилю і макету.
- Експорт у велике число форматів файлів.
- Інтегрування у JupyterLab і у графічний інтерфейс користувача.
- Використання великого набору зовнішніх пакетів, побудованих на основі Matplotlib.

Matplotlib може створювати публікації у цифровій якості для різних форматів друкованих копій та інтерактивних середовищах на різноманітних платформах. Matplotlib може бути використаний у сценаріях мови програмування Python, у серверах веб-додатків, у оболонках Python та у різноманітних наборах інструментів графічного інтерфейсу користувача.

3.2 Опис набору даних підводних зображень для навчання та тестування модуля

Набір даних, який ми оцінюємо та використовуємо для Transfer Learning, – це Ozfish [17], який містить 43 тисячі анотацій обмежувальних рамок у 1800 кадрах. Набір даних було зібрано за допомогою відеоматеріалів австралійської

програми Discoveries Data Commons. Приклади зображень з набору даних Ozfish представлені на рисунку 3.1

Рисунок 3.1 – Приклади зображень з набору даних Ozfish

Набір даних Ozfish містить у середньому 25 риб на кадр і містить рибу різних розмірів із великою кількістю дрібних риб, що робить його хорошим набором даних для використання в цьому проекті. Крім того, більшість знімків було зроблено при слабкому освітленні та в більш непрозорих умовах, що робить набір даних більш складним для виявлення риби. Вся база прикладів була поділена на навчальний, валідаційний та тестовий набори у співвідношенні 70-20-10 відсотків. Баланс класів риб у наборі даних Ozfish показано на рисунку 3.2.

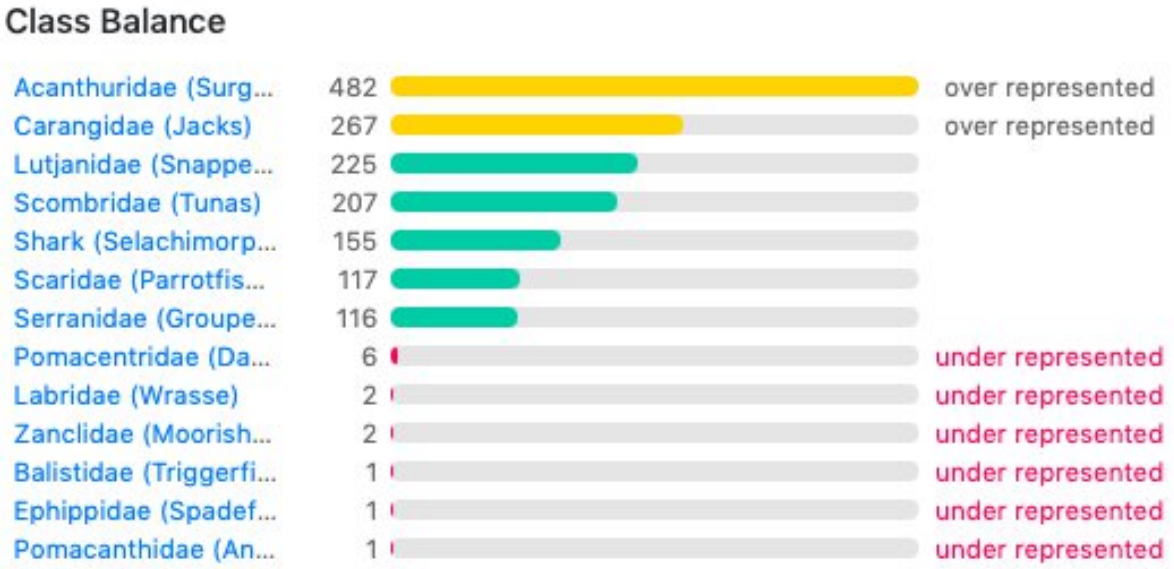


Рисунок 3.2 – Баланс класів риб у наборі даних Ozfish

Для доповнення (аугментації) даних ми провели експерименти з двома різними наборами даних: Ozfish без доповнення даних (загалом 680 зображень) і Ozfish із доповненням даних (загалом 4570 зображень). Збільшення даних включало поворот від -7° до +7°, кут огляду ±15°, горизонтальне та вертикальне розмиття до 1,5 пікселів і шум до 3 відсотків пікселів. Приклади аугментованих зображень з набору даних Ozfish представлені на рисунку 3.3

Роздільна здатність зображень у наборі даних Ozfish 416x416 пікселів.

Взагалі, можна підготувати власний спеціальний набір даних Якщо у вас уже є власні зображення (і, за бажанням, анотації), ви можете конвертувати свій набір даних за допомогою Roboflow [18], набору інструментів, які розробники використовують для швидкого й точного створення кращих моделей комп’ютерного зору. Понад 100 тисяч розробників використовують Roboflow для (автоматичної) анотації, перетворення форматів наборів даних (наприклад, у YOLOv7), навчання, розгортання та вдосконалення своїх наборів даних/моделей.

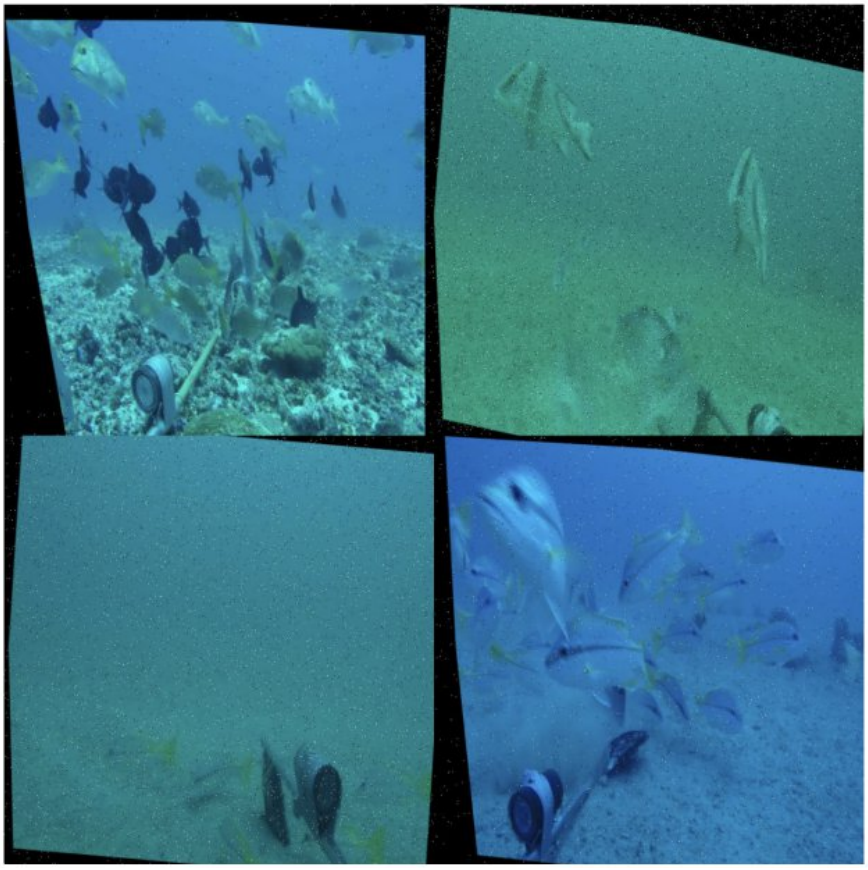


Рисунок 3.3 – Приклади аугментованих зображень з набору даних Ozfish

Roboflow - найбільша у світі колекція наборів даних комп'ютерного зору та API з відкритим кодом. Більше 350 мільйонів зображень, 500 000+ наборів даних, 100 000+ модифікованих моделей.

3.3 Програмна реалізація модуля виявлення риб на підводних зображеннях

Щоб програмно реалізувати модуль виявлення риб на підводних зображеннях, потрібно виконати такі дії:

- Встановити залежності YOLOv7
- Завантажити спеціальний набір даних у форматі YOLOv7
- Запустити навчання YOLOv7
- Оцінити продуктивність YOLOv7
- Виконати інференс YOLOv7 на тестових зображеннях

Встановлення залежностей YOLOv7.

На рисунку 3.4 показано фрагмент коду встановлення залежностей для нейромережі YOLOv7.

```
# Download YOLOv7 repository and install requirements
!git clone https://github.com/WongKinYiu/yolov7
%cd yolov7
!pip install -r requirements.txt
```

```
Cloning into 'yolov7'...
remote: Enumerating objects: 734, done.
remote: Counting objects: 100% (734/734), done.
remote: Compressing objects: 100% (387/387), done.
remote: Total 734 (delta 372), reused 648 (delta 332), pack-reused 0
Receiving objects: 100% (734/734), 67.48 MiB | 33.54 MiB/s, done.
Resolving deltas: 100% (372/372), done.
/content/yolov7
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
4)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 5))
(1.21.6)
Requirement already satisfied: opencv-python>=4.1.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
6)) (4.6.0.66)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 7))
(7.1.2)
Collecting PyYAML>=5.3.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (59
6 kB)
     |████████████████████████████████| 596 kB 4.7 MB/s
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 9))
(2.23.0)
```

...

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython-
>-r requirements.txt (line 34)) (0.2.5)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packages (from pexpect->ipython->-r requireme
nts.txt (line 34)) (0.7.0)
Installing collected packages: torch, torchvision, thop, PyYAML
```

Рисунок 3.4 – Фрагмент коду встановлення залежностей для нейромережі YOLOv7

Завантаження спеціального набору даних у форматі YOLOv7.

Далі ми завантажуємо наш набір даних у потрібному форматі. Використовуємо експорт YOLOv7 PyTorch. На рисунку 3.5 показано фрагмент коду завантаження спеціального набору даних.

```
!pip install roboflow

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting roboflow
  Downloading roboflow-0.2.14.tar.gz (18 kB)
Collecting certifi==2021.5.30
  Downloading certifi-2021.5.30-py2.py3-none-any.whl (145 kB)
     |████████████████████████████████| 145 kB 4.7 MB/s
Collecting chardet==4.0.0
  Downloading chardet-4.0.0-py2.py3-none-any.whl (178 kB)
     |████████████████████████████████| 178 kB 27.6 MB/s
Collecting cycler==0.10.0
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: glob2 in /usr/local/lib/python3.7/dist-packages (from roboflow) (0.7)
Requirement already satisfied: idna==2.10 in /usr/local/lib/python3.7/dist-packages (from roboflow) (2.10)
Collecting kiwisolver==1.3.1
  Downloading kiwisolver-1.3.1-cp37-cp37m-manylinux1_x86_64.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 62.3 MB/s
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from roboflow) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.7/dist-packages (from roboflow) (1.21.6)
Requirement already satisfied: opencv-python-headless>=4.5.1.48 in /usr/local/lib/python3.7/dist-packages (from roboflow) (4.
6.0.66)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.7/dist-packages (from roboflow) (7.1.2)
Collecting pyparsing==2.4.7

...

Successfully installed certifi-2021.5.30 chardet-4.0.0 cycler-0.10.0 kiwisolver-1.3.1 pyparsing-2.4.7 python-dotenv-0.20.0 re
quests-2.28.1 requests-toolbelt-0.9.1 roboflow-0.2.14 urllib3-1.26.6 wget-3.2

from roboflow import Roboflow
rf = Roboflow(api_key="")
project = rf.workspace("roboflow-gw7yv").project("fish-yzfml")
dataset = project.version(43).download("yolov7")

loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Fish-43 to yolov7pytorch: 100% [20326345 / 20326345] bytes
Extracting Dataset Version Zip to Fish-43 in yolov7pytorch:: 100%|██████████| 2732/2732 [00:01<00:00, 1491.75it/s]
```

Рисунок 3.5 – Фрагмент коду завантаження спеціального набору даних

Звертаємо увагу, що для цієї моделі потрібні анотації YOLO TXT, спеціальний файл YAML і впорядковані каталоги. Експорт Roboflow записує це для нас і зберігає у правильному місці.

Запуск навчання YOLOv7.

Далі розпочинаємо індивідуальне навчання мережі YOLOv7. У нашому випадку ми змінимо лише один із стандартних параметрів навчання YOLOv7: епохи. У нашому випадку ми будемо регулювати тільки кількість епох навчання від 100 до 300.

За бажанням можна налаштувати модель YOLOv7 за допомогою таких параметрів:

--weights, шлях до початкових ваг (значення за замовчуванням: 'yolo7.pt')

--cfg, шлях model.yaml (значення за замовчуванням: '')

--data, шлях до data.yaml (значення за замовчуванням: 'data/coco.yaml')

--hyp, шлях гіперпараметрів (значення за замовчуванням: 'data/hyp.scratch.p5.yaml')

--epochs, кількість циклів навчання (значення за замовчуванням: 300)

--batch-size, загальний розмір пакету для всіх GPU (значення за замовчуванням: 16)

--img-size, розміри зображення (значення за замовчуванням: [640, 640])

--rect, чи використовувати опцію неквадратичного навчання

--resume, відновити останнє навчання (значення за замовчуванням: False)

--nosave, зберегти лише останню контрольну точку

--test, перевірити лише останню епоху

--noautoanchor, вимкнути перевірку автоматичної прив'язки

-- evolve, розвивати гіперпараметри

--cache-images, кеш-зображення для швидшого навчання

--image-weights, використовувати зважений вибір зображень для навчання

--device, пристрій cuda, тобто 0 або 0,1,2,3 або cpu (значення за замовчуванням: '')

--багатомасштабний, варіювати розмір зображення +/- 50%%

--single-cls, навчати багатокласові дані як однокласові

--adam, використовувати оптимізатор torch.optim.Adam().

--sync-bn, використовувати SyncBatchNorm, доступний в режимі DDP

--local_rank, параметр DDP, (значення за замовчуванням: -1)

--workers, максимальна кількість завантажувачів даних (значення за замовчуванням: 8)

--project, зберегти в проект/ім'я (значення за замовчуванням: 'runs/train')

--name, зберегти до проекту/назви (значення за замовчуванням: 'exp')

--label-smoothing, епсилон згладжування мітки (значення за замовчуванням: 0,0)

--upload_dataset, завантажити набір даних як таблицю артефактів W&B

--bbox_interval, встановити інтервал реєстрації зображення обмежувальної рамки для W&B (значення за замовчуванням: -1)

--save_period, реєструвати модель після кожної епохи "save_period" (значення за замовчуванням: -1)

--artifact_alias, версія артефакту набору даних для використання (значення за замовчуванням: "останній")

Відредагувавши налаштування в клітинці train.py, запустимо навчання. Модель розпочне навчання та працюватиме протягом кількох хвилин або годин (залежно від того, наскільки великий наш набір даних, які варіанти навчання ми вибрали та який графічний процесор нам було призначено в лотереї Colab).

Далі завантажуємо стартовий чекпоінт нейромережі YOLOv7, навченої на базі зображень COCO:

```
# download COCO starting checkpoint
%cd /content/yolov7
!wget "https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt"
```

Потім запускаємо цю комірку, щоб почати навчання:

```
# run this cell to begin training
%cd /content/yolov7
!python train.py --batch 16 --cfg cfg/training/yolov7.yaml --epochs 100 --data /content/yolov7/Fish-43/data.yaml --weights
```

```
/content/yolov7
YOLOR 🚀 v0.1-103-g6ded32c torch 1.12.1+cu102 CUDA:0 (Tesla T4, 15109.75MB)

Namespace(adam=False, artifact_alias='latest', batch_size=16, bbox_interval=-1, bucket='', cache_images=False, cfg='cfg/train
ing/yolov7.yaml', data='/content/yolov7/Fish-43/data.yaml', device='0', entity=None, epochs=100, evolve=False, exist_ok=Fals
e, freeze=[0], global_rank=-1, hyp='data/hyp.scratch.p5.yaml', image_weights=False, img_size=[640, 640], label_smoothing=0.0,
linear_lr=False, local_rank=-1, multi_scale=False, name='exp', noautoanchor=False, nosave=False, notest=False, project='runs/
train', quad=False, rect=False, resume=False, save_dir='runs/train/exp', save_period=-1, single_cls=False, sync_bn=False, tot
al_batch_size=16, upload_dataset=False, weights='yolov7.pt', workers=8, world_size=1)
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_
lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, obj=0.7, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hs
v_v=0.4, degrees=0.0, translate=0.2, scale=0.9, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.15, c
opy_paste=0.0, paste_in=0.15, loss_ota=1
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)
Overriding model.yaml nc=80 with nc=26
```

У фрагменті вище видно які були встановлені параметри навчання. Потім програма виводить результати навчання по кожній епосі:

```
Starting training for 100 epochs...

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      0/99     9.84G   0.08421    0.7879    0.2024     1.075        42       640: 100% 60/60 [01:15<00:00,  1.26s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 9/9 [00:42<00:00,  4.68s/i
t]
                 all         272         660      0.0057      0.0335     0.00243    0.000582

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      1/99     10.7G   0.07421   0.03494    0.1971    0.3062        59       640: 100% 60/60 [00:55<00:00,  1.08it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 9/9 [00:17<00:00,  1.97s/i
t]
                 all         272         660     0.00413       0.076     0.00159     0.00031


...

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     98/99     10.7G   0.01572   0.01206   0.01389   0.04166        33       640: 100% 60/60 [00:53<00:00,  1.11it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 9/9 [00:03<00:00,  2.56it/
s]
                 all         272         660       0.529       0.577       0.319       0.228

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     99/99     10.7G   0.01523   0.01235   0.01382    0.0414        48       640: 100% 60/60 [00:53<00:00,  1.12it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 9/9 [00:04<00:00,  1.98it/
s]
                 all         272         660       0.534       0.556       0.322       0.233
Acanthuridae (Surgeonfishes)         272         112       0.369       0.652       0.378       0.262
Acanthuridae -Surgeonfishes-         272         112       0.368       0.643       0.378       0.265
  Carangidae (Jacks)         272          46        0.36       0.648       0.372       0.237
  Carangidae -Jacks-         272          46       0.366       0.674       0.391       0.248
   Labridae (Wrasse)         272           1           1           0           0           0
   Labridae -Wrasse-         272           1           1           0           0           0
Lutjanidae (Snappers)         272          62       0.427       0.613       0.408       0.295
Lutjanidae -Snappers-         272          62       0.412       0.629       0.406       0.279
Scaridae (Parrotfishes)         272          21       0.401        0.81       0.451       0.356
Scaridae -Parrotfishes-         272          21       0.415        0.81       0.453       0.366
  Scombridae (Tunas)         272          23       0.292           1         0.4       0.319
  Scombridae -Tunas-         272          23       0.279           1       0.377       0.287
Serranidae (Groupers)         272          30       0.455       0.613       0.472       0.328
Serranidae -Groupers-         272          30       0.466       0.567       0.475       0.321
Shark (Selachimorpha)         272          34         0.5       0.676       0.414       0.307
Shark -Selachimorpha-         272          34       0.501       0.679       0.422       0.317
Zanclidae (Moorish Idol)         272           1           1           0           0           0
Zanclidae -Moorish Idol-         272           1           1           0           0           0
100 epochs completed in 1.668 hours.
```

Вище в кінці виведеної інформації бачимо, що 100 епох навчання тривали протягом 1,668 годин.

Оцінка ефективності YOLOv7.

Ми можемо оцінити ефективність нашого спеціального навчання за допомогою сценарію оцінки. Зауважимо, що можна налаштувати наведені нижче спеціальні аргументи, які приймає detect.py.

```
# Run evaluation
!python detect.py --weights runs/train/exp/weights/best.pt --conf-thres 0.2 --source /content/yolov7/Fish-43/test/images
```

Виконання завдання на тестових зображеннях.

Наведений нижче фрагмент коду дозволяє виконати задачу виявлення риб для всіх тестових зображень та відобразити отримані результати.

```
#display inference on ALL test images
import glob
from IPython.display import Image, display

i = 0
limit = 10000 # max images to print
for imageName in glob.glob('/content/yolov7/runs/detect/exp2/*.jpg'): #assuming JPG
    if i < limit:
      display(Image(filename=imageName))
      print("\n")
    i = i + 1
```

Приклади оброблених тестових зображень представлені на рисунку 3.6.

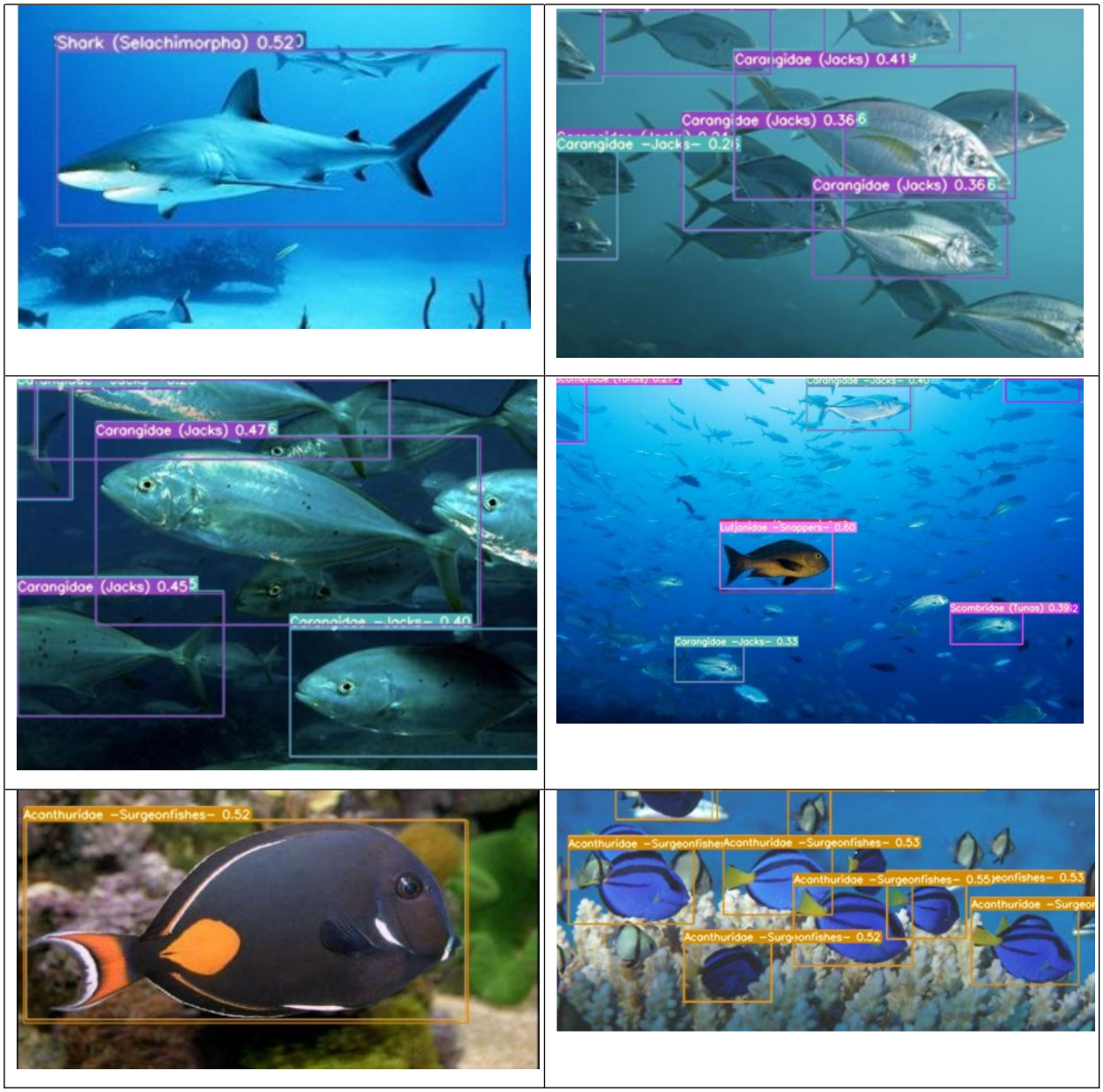


Рисунок 3.6 – Приклади оброблених тестових зображень

Після того, як наш перший тренувальний запуск буде завершено, потрібно використати навчену модель нейромережі, щоб визначити, які зображення є найбільш проблематичними, щоб дослідити, анотувати та покращити набір даних (і, отже, модель).

3.4 Висновок

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек Torch, NumPy, OpenCV та Matplotlib для програмної реалізації модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7. Проведено аналіз набору даних Ozfish, який містить 43 тисячі анотацій обмежувальних рамок у 1800 кадрах.. Описано основні етапи програмної реалізація та функціонування модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі, що складається з завантаження залежностей для YOLOv7, завантаження набору даних із зображеннями, запуску навчання моделі запропонованої нейромережі, оцінювання ефективності навчання нейромережі, виконання інференсу YOLOv7 на тестових зображеннях.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ

Розроблений програмний модуль реалізований у вигляді Jupiter Notebook, тому він не має користувацького інтерфейсу. Він виконується послідовно, завантажує необхідні бібліотеки та дані, навчає нейронну мережу, оцінює її продуктивність, а в кінці обробляє зображення тестової вибірки, які знаходяться у відповідні й папці. У результаті обробки на зображенні з'являються обмежувальні рамки, назви класу риб та коефіцієнт упевненості. Приклади оброблених зображень показані на рисунку 4.1.

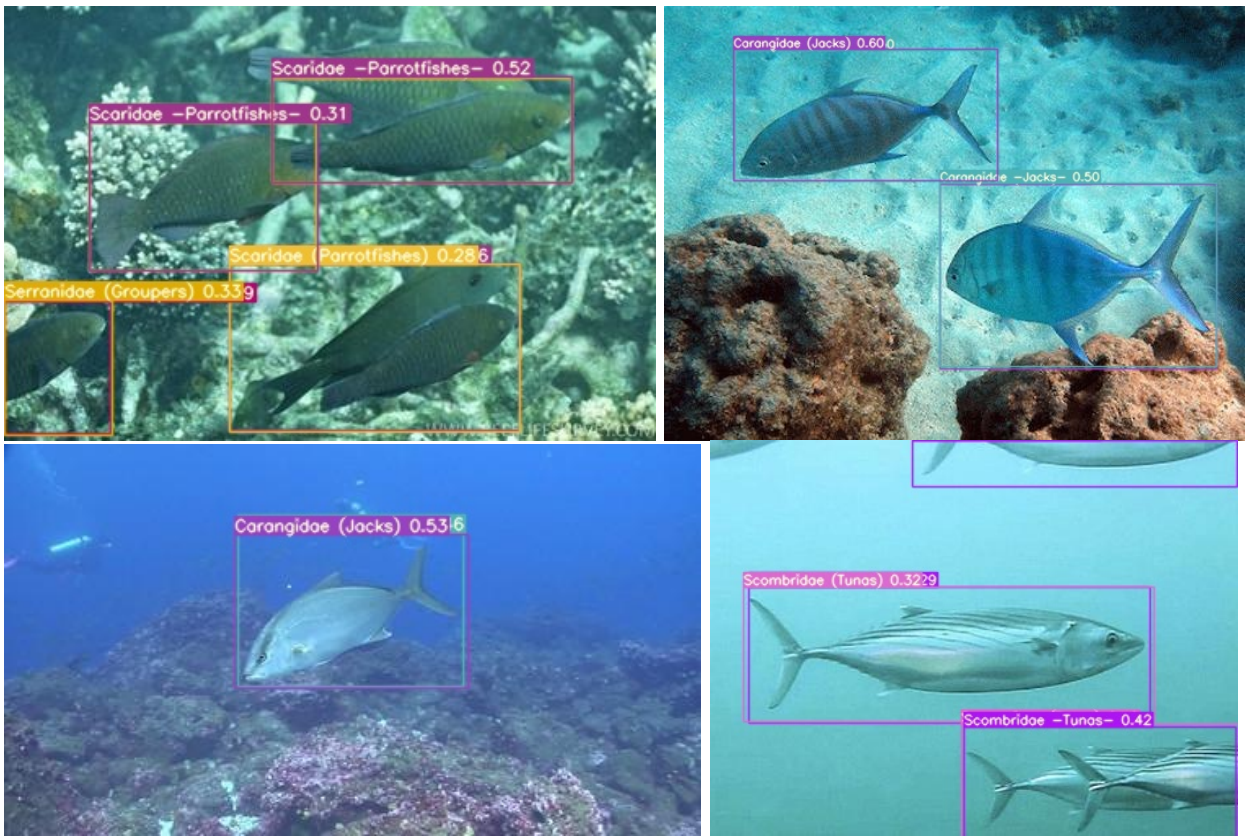


Рисунок 4.1 – Приклади підводних зображень, оброблених розробленим програмним модулем виявлення риб

Коефіцієнт упевненості (приймає значення від 0 до 1) показує імовірність того, що виявлена риба відноситься до зазначеного класу. Кількість тестових зображень 180 і всі вони виводяться на екран в роцесі виконання програми.

Оскільки метою нашого проекту є дослідження впливу Transfer Learning на процес виявлення риби на зображеннях, ми спочатку оцінили Ozfish на трьох типах моделей. Спочатку ми оцінили попередньо навчену модель виявлення риби YOLOv7 (яка була попередньо навчена на великому наборі даних про рибу). Потім ми покращили цю попередньо навчену модель за допомогою Transfer Learning на Ozfish. Нарешті, ми також навчили ту саму архітектуру моделі на Ozfish з нуля (без попередньо навчених ваг). Під час використання Transfer Learning і навчання з нуля ми використовували два різні набори даних: один із доповненням (аугментацією) і інший без доповнення. Після цього ми провели подальші експерименти, щоб дослідити, як кількість даних, що використовуються в Transfer Learning, впливає на продуктивність моделі.

Гіперпараметри.

При навчанні нейронної мережі були обрані такі параметри: швидкість навчання становила 0,01, метод навчання (оптимізатор) – стохастичного градієнтного спуску (SGD) з імпульсом, коефіцієнт імпульсу 0,937 і коефіцієнт спаду ваги 0,0005. Розмір пакету (міні-серії) становив 64. Було обрано саме такі параметри, оскільки це були початкові параметри, на яких навчалася попередньо підготовлена модель, і ми виявили, що ці параметри також найкраще працюють на наших початкових і точно налаштованих моделях. Щоб перевірити це, ми виконали грубий і точний пошук гіперпараметрів у сітці.

Щоб запобігти перенавчанню, ми побудували криві помилки навчання і тестування для кожної навченої моделі, щоб переконатися, що модель не перенавчається. Це допомогло нам визначити найкращу кількість епох для навчання моделі.

Метрики.

Для оцінки продуктивності моделі ми використовували два показники: середню точність (precision) і середню повноту (recall). Щоб обчислити їх, ми використали CIoU (як визначено вище) між прогнозованою та справжньою обмежувальними рамками.

Ми використовували порогове значення CIoU 0,5, тобто якщо CIoU ≥ 0,5, це вважається справді позитивним. Середню точність і середню повноту розраховували таким чином:

Середня точність = TP/(TP + FP).

Середня повнота = TP/(TP + FN).

Показники точності різних модифікацій моделі YOLOv7 на датасеті Ozfish, досліджені у цій роботі, наведені у табл. 4.1.

Таблиця 4.1 – Показники точності різних модифікацій моделі YOLOv7 на датасеті Ozfish

Модель	Середня точність	Середня повнота
Попередньо навчена	0,7628	0,9064
Перенесення навчання (без аугментації)	0,7365	0,7323
Перенесення навчання (з аугментацією)	0,7581	0,9140
«З нуля» (без аугментації)	0,7408	0,7312
«З нуля» (з аугментацією)	0,7366	0,7314

Хоча всі моделі, перелічені в таблиці 4.1, мають відносно близькі значення середньої точності (усі в діапазоні 3% або менше), значення середньої повноти значно відрізняються. Повнота відповідає здатності моделі мінімізувати помилкові негативи (тобто здатність правильно виявляти всі екземпляри риби на зображенні). Точність відповідає достовірності виявлених

зображень (тобто модель правильно визначає рибу, а не інші об’єкти). Таким чином, ми бачимо, що всі моделі в таблиці 4/1 були відносно подібними за точністю їх виявлення, але деякі моделі були набагато кращими за інших у виявленні всіх риб, присутніх на зображенні. Оскільки набір даних Ozfish містить у середньому 25 риб на зображення (що набагато більше, ніж у більшості інших наборів даних про рибу), Середня повнота (Mean Recall) є важливим показником, який ми повинні враховувати під час оцінки ефективності.

З наших результатів ми бачимо, що модель, навчена з нуля з аугментацією, показала найгірші показники як середньої точності, так і середньої повноти. У той час як модель «з нуля» без аугментації спрацювала трохи краще, обидві моделі «з нуля» не показали себе відносно добре. Той факт, що обидві моделі «з нуля» не показали належних результатів, підтверджує думку про те, що Ozfish є складними даними для навчання з нуля, оскільки моделі важко дізнатися про форму та особливості риби з рибою меншого розміру та більш непрозорими зображеннями.

З наших результатів ми бачимо, що модель Transfer Learning with augmentation працює дуже подібно до попередньо навченої моделі, з різницею в середній точності менше 0,5% і різницею в середній повноті менше 0,8%. Це показує нам, що завдання Transfer Learning для доповненого набору даних не мало помітного впливу на загальну продуктивність попередньо підготовленої моделі на Ozfish.

Проте, подальший якісний аналіз показує, що модель з Transfer Learning могла краще обробляти певні випадки зображень, знайдених в Ozfish. На рисунках 4.2 і 4.3 показано кілька зображень, які були випадками збою на попередньо навченій моделі, і їх відповідну продуктивність на моделі Transfer Learning з аугментацією.

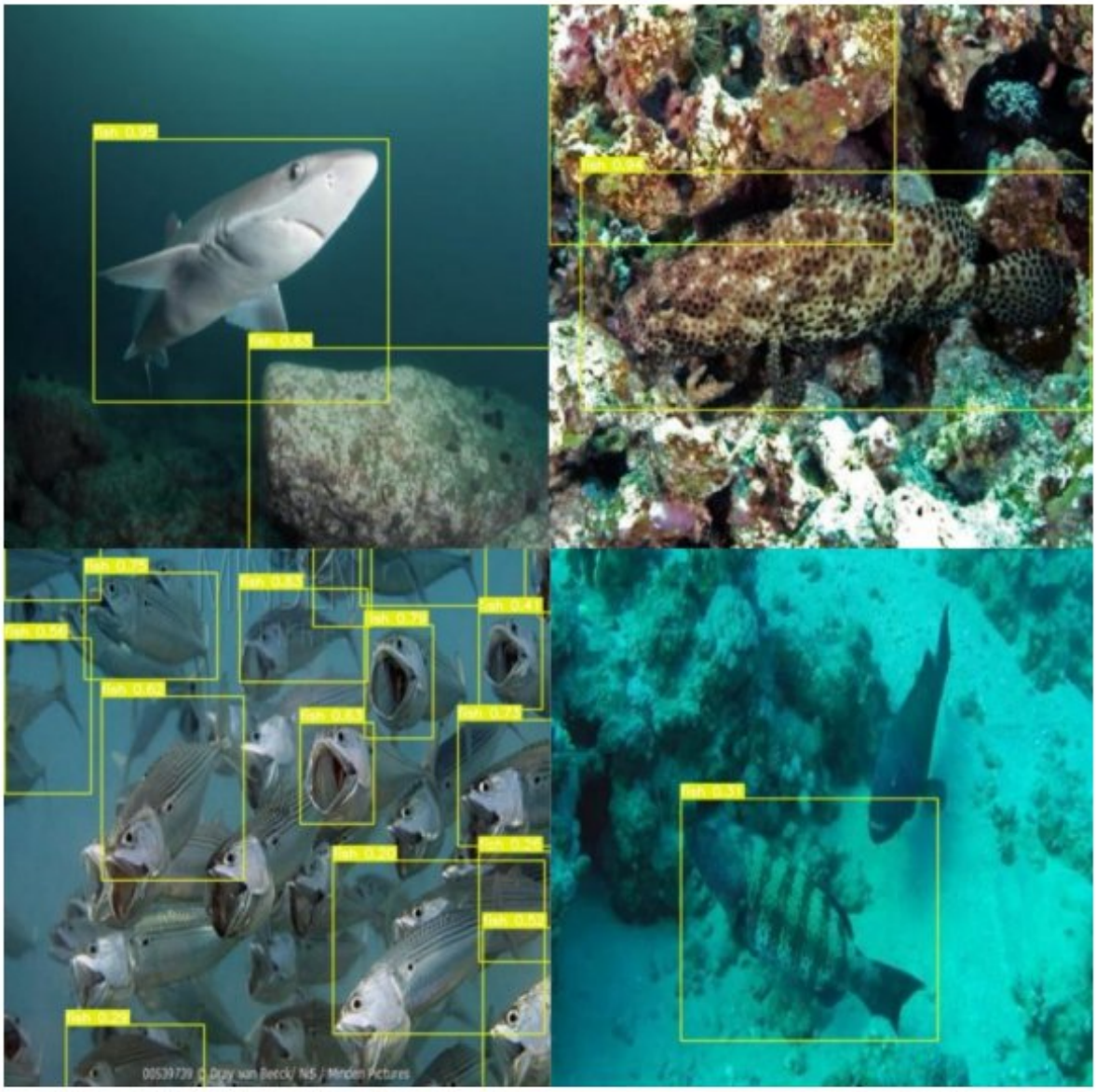


Рисунок 4.2 – Випадки відмов попередньо навченої моделі

Ми бачимо, що за допомогою Transfer Learning модель справді дізнається більше про ідентифікацію риби на фоні, особливо у випадках, коли риба зливається з фоном, але все ще має проблеми із зображеннями з багатьма рибами та має інші подібні випадки помилок. Цікаво, що наші результати показують суттєву різницю між середнім показником повноти для перенесеного навчання без доповнення (0,7323) і перенесеного навчання з доповненням (0,9140).

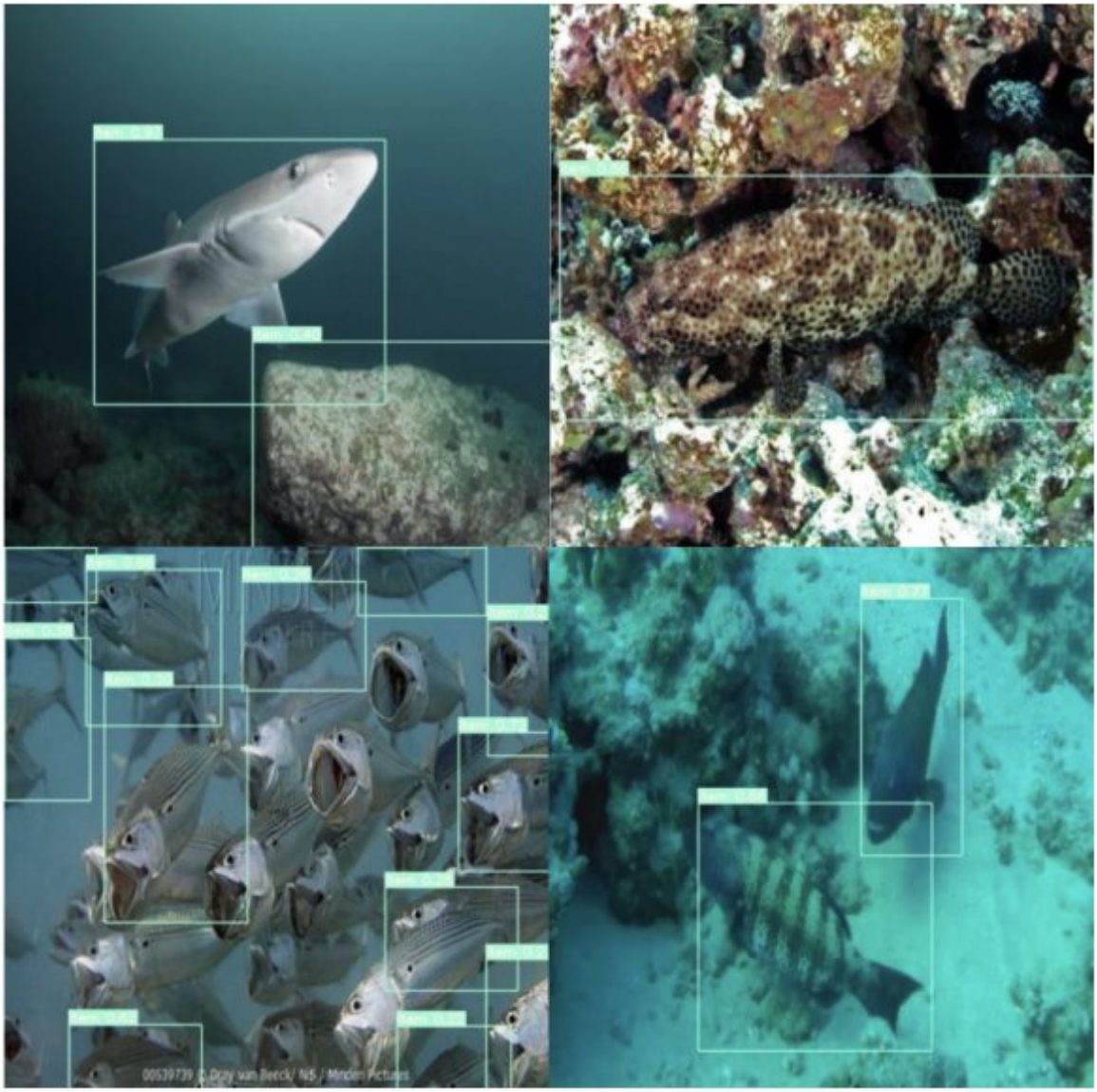


Рисунок 4.3 –Випадки з рисунку 4.2, оброблені моделлю Transfer Learning зі 100% доповненим набором даних

Таким чином, це виглядає так, що метод Transfer Learning без доповнення фактично значно знизив продуктивність попередньо навченої моделі замість того, щоб підвищити її. Ми припустили, що це потенційно пов’язано з розміром наборів даних, які використовуються для передачі навчання. У той час як набір даних, використаний для Transfer Learning без доповнення, містив лише 680 зображень від Ozfish, набір даних, використаний для Transfer Learning з доповненням, містив 4570 зображень. Таким чином, щоб дослідити вплив кількості даних на якість Transfer Learning, ми вирішили

провести подальші експерименти, у яких використали Transfer Learning на тій самій попередньо підготовленій моделі, використовуючи лише 25%, 50%, 75% і 88% доповнених навчальних даних Ozfish. Наші результати представлені в таблиці 4.2.

Таблиця 4.2 – Показники точності різних модифікацій моделі YOLOv7 на датасеті Ozfish з різними обсягами даних, використаних при перенесенні навчання

Модель	Середня точність	Середня повнота
«З нуля» (100% навчальних даних)	0,7366	0,7314
Попередньо натренована (без перенесення навчання)	0,7628	0,9064
Перенесення навчання (25% навчальних даних)	0,6848	0,6902
Перенесення навчання (50% навчальних даних)	0,7193	0,7203
Перенесення навчання (75% навчальних даних)	0,7472	0,7332
Перенесення навчання (88% навчальних даних)	0,7521	0,8379
Перенесення навчання (100% навчальних даних)	0,7581	0,9140

З таблиці 4.2 ми бачимо, що середня точність і середня повнота зростають, коли ми тренуємося з більшим відсотком доповнених даних Ozfish. Це відповідає нашій попередній теорії про кількість даних, що впливають на якість Transfer Learning. Ми помітили, що Transfer Learning з використанням 25% і 50% розширених даних навчання працює гірше, ніж модель з нуля, що свідчить про те, що 25% і 50% розширених даних навчання було недостатньо, щоб попередньо навчена модель навчилася виявляти рибу в наборі даних Ozfish. Потім ми бачимо, що Transfer Learning із використанням 75% доповнених навчальних даних Ozfish дає кращі результати, ніж модель, яка навчалася виключно на навчальних даних Ozfish, і оскільки ми збільшуємо

обсяг використовуваних даних ще більше до 88 і 100 відсотків, продуктивність моделі суттєво зростає (рис. 4.4). Це показує, що за наявності достатньої кількості навчальних даних модель може доповнити своє навчання, щоб також добре працювати на Ozfish. Таким чином, це демонструє ефективність Transfer Learning у створенні кращих моделей виявлення риби. Однак ми зазначаємо, що в наших експериментах ми не змогли суттєво покращити продуктивність попередньо навченої моделі на Ozfish, що підкреслює складність навчання моделі виявлення риби для дрібної риби та каламутної води.

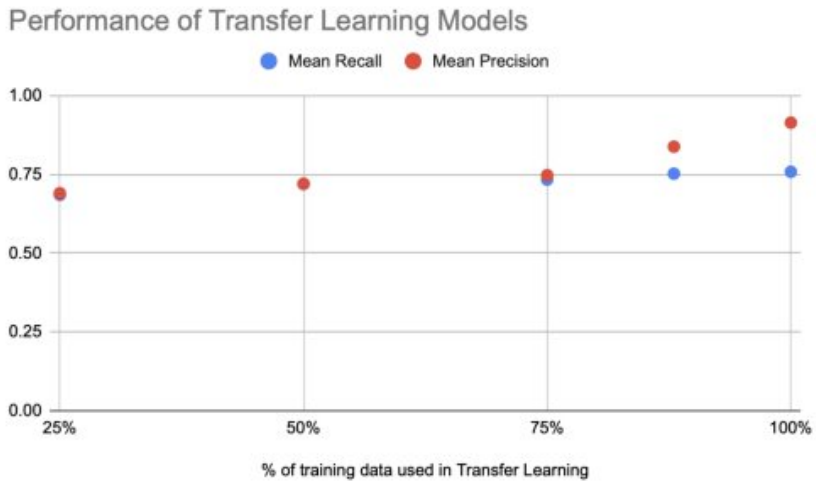


Рисунок 4.4 –Графіки залежності середньої точності та середньої повноти від відсотка даних, використаних у Transfer Learning

Експериментуючи з доповненими даними та навчаючись із використанням різних обсягів даних, ми побачили, що кількість даних, які використовуються в Transfer Learning, значно покращує вплив Transfer Learning на виявлення риби на підводних зображеннях. Виявлення риби в каламутній воді та дрібної риби продовжує залишатися складним завданням для моделей виявлення об’єктів, як показують наші результати.

У майбутньому слід продовжувати вдосконалювати модель виявлення риби, навчаючи її на більшій кількості даних про маленьку рибу в каламутній воді. Це включатиме пошук інших наборів даних, які містять дрібну рибу, або звернення до дослідників-екологів для створення власних наборів даних.

Також слід вивчити інші методи виявлення об’єктів, такі як SSD, Faster-RCNN і більш просунуті версії YOLO. Ці моделі виявлення об’єктів використовують різноманітні методи, які можуть краще чи гірше служити для виявлення риби на підводних зображеннях.

Із табл. 4.1 видно, що найкращою по параметру середньої повноти (91,4%) є модель з перенесеним навчанням і 100% аугментацією, яка по параметру середньої точності (75,81%) знаходиться на тому ж рівні, що і попередньо навчена модель (76,28%).

Таким чином, можна зробити висновок, що розроблений програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі має (91,4%) порівняно з найкращим аналогом (90,6%) збільшену на 0,8% середню повноту. Тобто мета роботи досягнута – точність виявлення риб на підводних зображеннях підвищена.

Висновок

У розділі в результаті тестування модуля було доведено його працездатність та відповідність поставленому завданню. Для оцінки якості роботи модуля використовувалась такі метрики як середня точність та середня повнота. Було оцінено 5 варіантів навчання моделі: 1) попередньо навчена, 2) перенесення навчання (без аугментації), 3) перенесення навчання (з аугментацією), 4) навчена «з нуля» (без аугментації), 5) навчена «з нуля» (з аугментацією). Найкращою по параметру середньої повноти (91,4%) є модель з перенесеним навчанням і 100% аугментацією, яка по параметру середньої точності (75,81%) знаходиться на тому ж рівні, що і попередньо навчена модель (76,28%). Тобто розроблений програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі має (91,4%) порівняно з найкращим аналогом (90,6%) збільшену на 0,8% середню повноту, а значить мета роботи досягнута – точність виявлення риб на підводних зображеннях підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу виявлення риб на підводних зображеннях на основі згорткової нейронної мережі. В ході аналізу предметної області було описано детальну постановку задачі виявлення риб на підводних зображеннях. Також розглянуто різні методи розв'язання задачі виявлення об’єктів на зображеннях і оцінено їх застосовність до завдання виявлення риб на підводних зображеннях. Серед таких методів розпізнавання об`єктів як колірні фільтри, виділення та аналіз контурів, зіставлення з шаблоном, робота з особливими точками, методи машинного навчання як найбільш перспективний і застосовний до даної задачі було обрано метод машинного навчання, а точніше нейромережевий метод. Було показано, що з різних типів нейронних мереж найбільше підходить для вирішення поставленої задачі саме згорткові нейронні мережі. Крім цього, було оглянуто аналоги розроблюваного модуля.

Також було проаналізовано архітектуру згорткової нейронної мережі YOLOv7 і на її основі розроблено модель виявлення риб на підводних зображеннях. У розробленій моделі використовується перенесення навчання (Transfer Learning) для покращення виявлення риби спеціально для меншої риби і в умовах каламутної води. Описано принципи роботи та навчання нейронної мережі YOLOv7 для модуля виявлення риб. Розроблено алгоритм роботи програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7.

Було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек Torch, NumPy, OpenCV та Matplotlib для програмної реалізації модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7. Проведено аналіз набору даних Ozfish, який містить 43 тисячі анотацій обмежувальних рамок у 1800 кадрах. Описано основні етапи програмної реалізація та функціонування модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі, що

складається з завантаження залежностей для YOLOv7, завантаження набору даних із зображеннями, запуску навчання моделі запропонованої нейромережі, оцінювання ефективності навчання нейромережі, виконання інференсу YOLOv7 на тестових зображеннях.

У результаті тестування модуля було доведено його працездатність та відповідність поставленому завданню. Для оцінки якості роботи модуля використовувалась такі метрики як середня точність та середня повнота. Було оцінено 5 варіантів навчання моделі: 1) попередньо навчена, 2) перенесення навчання (без аугментації), 3) перенесення навчання (з аугментацією), 4) навчена «з нуля» (без аугментації), 5) навчена «з нуля» (з аугментацією). Найкращою по параметру середньої повноти (91,4%) є модель з перенесеним навчанням і 100% аугментацією, яка по параметру середньої точності (75,81%) знаходиться на тому ж рівні, що і попередньо навчена модель (76,28%). Тобто розроблений програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі має (91,4%) порівняно з найкращим аналогом (90,6%) збільшену на 0,8% середню повноту, а значить мета роботи досягнута – точність виявлення риб на підводних зображеннях підвищена.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Muksit, A. A., Hasan, F., Emon, M. F. H. B., Haque, M. R., Anwary, A. R., Shatabda, S. (2022). YOLO-Fish: A robust fish detection model to detect fish in realistic underwater environment. Ecological Informatics, 72, 101847. https://doi.org/10.1016/j.ecoinf.2022.101847

2 Jalal, A., Salman, A., Mian, A., Shortis, M., Shafait, F. (2020). Fish detection and species classification in underwater environments using deep learning with temporal information. Ecological Informatics, 57, 101088. https://doi.org/10.1016/j.ecoinf.2020.101088

3 Y. Wang, J. Liu, S. Yu, K. Wang, Z. Han and Y. Tang, ”Underwater Object Detection based on YOLOv3 network,” 2021 IEEE International Conference on Unmanned Systems (ICUS), Beijing, China, 2021, pp. 571-575.

4 Kandimalla, V., Richard, M., Smith, F., Quirion, J., Torgo, L., & Whidden, C. (2022). Automated detection, classification and counting of fish in fish passages with deep learning. Frontiers in Marine Science, 8. https://doi.org/10.3389/fmars.2021.823173

5 Xu, W., & Matzner, S. (2018, November 5). Underwater Fish Detection using Deep Learning for Water Power Applications. arXiv.org. https://arxiv.org/abs/1811.01494

6 Saleh, A., Sheaves, M., Jerry, D.R., & Azghadi, M.R. (2022). Transformer-based Self-Supervised Fish Segmentation in Underwater Videos. ArXiv, abs/2206.05390.

7 Lin, X., Huang, X., & Wang, L. (2024). Underwater object detection method based on learnable query recall mechanism and lightweight adapter. PloS one, 19(2), e0298739. https://doi.org/10.1371/journal.pone.0298739

8 Pagire, V., Phadke, A. C., & Hemant, J. (2024). A deep learning approach for underwater fish detection. Journal of Integrated Science and Technology, 12(3). https://doi.org/10.62110/sciencein.jist.2024.v12.765

9 Marrable, D., Barker, K., Tippaya, S., Wyatt, M., Bainbridge, S., Stowar, M., & Larke, J. (2022). Accelerating species recognition and labelling of fish from

underwater video with Machine-Assisted Deep Learning. Frontiers in Marine Science, 9. https://doi.org/10.3389/fmars.2022.944582

10 Fast accurate fish detection and recognition of underwater images with Fast R-CNN. (2015, October 1). IEEE Conference Publication ─ IEEE Xplore. https://ieeexplore.ieee.org/document/7404464

11 Ghugare, P. P. (2022). Underwater Fish Detection. GitHub. https://github.com/pathikg/Underwaterfish-detection

12 Wang, C. Y., Bochkovskiy, A., Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 7464-7475).

13 Python [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

14 PyTorch [Електронний ресурс] – Режим доступу: https://pytorch.org/

15 OpenCV - Open Computer Vision Library [Електронний ресурс] – Режим доступу: https://opencv.org/

16 Matplotlib: Visualization with Python [Електронний ресурс] – Режим доступу: https://matplotlib.org/

17 Australian Institute Of Marine Science, 2020. Ozfish dataset - machine learning dataset for baited remote underwater video stations.

18 Solawetz, J. (2023, February). Fish Dataset. Roboflow Universe. Roboflow. Retrieved June 6, 2024, from https://universe.roboflow.com/roboflowgw7yv/fish-yzfml

19 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп'ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль виявлення риб на підводних зображеннях на основі згорткової нейронної мережі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 15,10 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колесницький О.К.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Байбак І.А.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

Фрагмент коду (навчання нейронної мережі)

```
import argparse
import logging
import math
import os
import random
import time
from copy import deepcopy
from pathlib import Path
from threading import Thread

import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

import test  # import test.py to get mAP after each epoch
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path, labels_to_image_weights,
init_seeds, \
    fitness, strip_optimizer, get_latest_run, check_dataset, check_file, check_git_status,
check_img_size, \
    check_requirements, print_mutation, set_logging, one_cycle, colorstr
from utils.google_utils import attempt_download
from utils.loss import ComputeLoss, ComputeLossOTA
from utils.plots import plot_images, plot_labels, plot_results, plot_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_dicts,
torch_distributed_zero_first, is_parallel
from utils.wandb_logging.wandb_utils import WandbLogger, check_wandb_resume

logger = logging.getLogger(__name__)

def train(hyp, opt, device, tb_writer=None):
    logger.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))
    save_dir, epochs, batch_size, total_batch_size, weights, rank, freeze = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.total_batch_size, opt.weights,
opt.global_rank, opt.freeze

    # Directories
    wdir = save_dir / 'weights'
    wdir.mkdir(parents=True, exist_ok=True)  # make dir
    last = wdir / 'last.pt'
    best = wdir / 'best.pt'
    results_file = save_dir / 'results.txt'

    # Save run settings
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.dump(vars(opt), f, sort_keys=False)

    # Configure
    plots = not opt.evolve  # create plots
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.load(f, Loader=yaml.SafeLoader)  # data dict
    is_coco = opt.data.endswith('coco.yaml')

    # Logging- Doing this before checking the dataset. Might update data_dict
    loggers = {'wandb': None}  # loggers dict
```

```
    if rank in [-1, 0]:
        opt.hyp = hyp  # add hyperparameters
        run_id = torch.load(weights, map_location=device).get('wandb_id') if
weights.endswith('.pt') and os.path.isfile(weights) else None
        wandb_logger = WandbLogger(opt, Path(opt.save_dir).stem, run_id, data_dict)
        loggers['wandb'] = wandb_logger.wandb
        data_dict = wandb_logger.data_dict
        if wandb_logger.wandb:
            weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp  # WandbLogger might update
weights, epochs if resuming

    nc = 1 if opt.single_cls else int(data_dict['nc'])  # number of classes
    names = ['item'] if opt.single_cls and len(data_dict['names']) != 1 else data_dict['names']
# class names
    assert len(names) == nc, '%g names found for nc=%g dataset in %s' % (len(names), nc,
opt.data)  # check

    # Model
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(rank):
            attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location=device)  # load checkpoint
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc,
anchors=hyp.get('anchors')).to(device)  # create
        exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and not opt.resume else []  #
exclude keys
        state_dict = ckpt['model'].float().state_dict()  # to FP32
        state_dict = intersect_dicts(state_dict, model.state_dict(), exclude=exclude)  #
intersect
        model.load_state_dict(state_dict, strict=False)  # load
        logger.info('Transferred %g/%g items from %s' % (len(state_dict),
len(model.state_dict()), weights))  # report
    else:
        model = Model(opt.cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  # create
    with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
    train_path = data_dict['train']
    test_path = data_dict['val']

    # Freeze
    freeze = [f'model.{x}.' for x in (freeze if len(freeze) > 1 else range(freeze[0]))]  #
parameter names to freeze (full or partial)
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
            print('freezing %s' % k)
            v.requires_grad = False

    # Optimizer
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batch_size), 1)  # accumulate loss before optimizing
    hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # scale weight_decay
    logger.info(f"Scaled weight_decay = {hyp['weight_decay']}")

    pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
    for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
            pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
            pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):
            pg1.append(v.weight)  # apply decay
        if hasattr(v, 'im'):
            if hasattr(v.im, 'implicit'):
                pg0.append(v.im.implicit)
            else:
                for iv in v.im:
                    pg0.append(iv.implicit)
        if hasattr(v, 'imc'):
            if hasattr(v.imc, 'implicit'):
                pg0.append(v.imc.implicit)
            else:
                for iv in v.imc:
                    pg0.append(iv.implicit)
        if hasattr(v, 'imb'):
            if hasattr(v.imb, 'implicit'):
                pg0.append(v.imb.implicit)
            else:
```

```
                for iv in v.imb:
                    pg0.append(iv.implicit)
        if hasattr(v, 'imo'):
            if hasattr(v.imo, 'implicit'):
                pg0.append(v.imo.implicit)
            else:
                for iv in v.imo:
                    pg0.append(iv.implicit)
        if hasattr(v, 'ia'):
            if hasattr(v.ia, 'implicit'):
                pg0.append(v.ia.implicit)
            else:
                for iv in v.ia:
                    pg0.append(iv.implicit)
        if hasattr(v, 'attn'):
            if hasattr(v.attn, 'logit_scale'):
                pg0.append(v.attn.logit_scale)
            if hasattr(v.attn, 'q_bias'):
                pg0.append(v.attn.q_bias)
            if hasattr(v.attn, 'v_bias'):
                pg0.append(v.attn.v_bias)
            if hasattr(v.attn, 'relative_position_bias_table'):
                pg0.append(v.attn.relative_position_bias_table)
        if hasattr(v, 'rbr_dense'):
            if hasattr(v.rbr_dense, 'weight_rbr_origin'):
                pg0.append(v.rbr_dense.weight_rbr_origin)
            if hasattr(v.rbr_dense, 'weight_rbr_avg_conv'):
                pg0.append(v.rbr_dense.weight_rbr_avg_conv)
            if hasattr(v.rbr_dense, 'weight_rbr_pfir_conv'):
                pg0.append(v.rbr_dense.weight_rbr_pfir_conv)
            if hasattr(v.rbr_dense, 'weight_rbr_1x1_kxk_idconv1'):
                pg0.append(v.rbr_dense.weight_rbr_1x1_kxk_idconv1)
            if hasattr(v.rbr_dense, 'weight_rbr_1x1_kxk_conv2'):
                pg0.append(v.rbr_dense.weight_rbr_1x1_kxk_conv2)
            if hasattr(v.rbr_dense, 'weight_rbr_gconv_dw'):
                pg0.append(v.rbr_dense.weight_rbr_gconv_dw)
            if hasattr(v.rbr_dense, 'weight_rbr_gconv_pw'):
                pg0.append(v.rbr_dense.weight_rbr_gconv_pw)
            if hasattr(v.rbr_dense, 'vector'):
                pg0.append(v.rbr_dense.vector)

    if opt.adam:
        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # adjust
beta1 to momentum
    else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)

    optimizer.add_param_group({'params': pg1, 'weight_decay': hyp['weight_decay']})  # add pg1
with weight_decay
    optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2), len(pg1),
len(pg0)))
    del pg0, pg1, pg2

    # Scheduler https://arxiv.org/pdf/1812.01187.pdf
    # https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#OneCycleLR
    if opt.linear_lr:
        lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf']  # linear
    else:
        lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)
    # plot_lr_scheduler(optimizer, scheduler, epochs)

    # EMA
    ema = ModelEMA(model) if rank in [-1, 0] else None

    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
            best_fitness = ckpt['best_fitness']

        # EMA
        if ema and ckpt.get('ema'):
            ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
            ema.updates = ckpt['updates']
```

```
        # Results
        if ckpt.get('training_results') is not None:
            results_file.write_text(ckpt['training_results'])  # write results.txt

        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if opt.resume:
            assert start_epoch > 0, '%s training to %g epochs is finished, nothing to resume.' %
(weights, epochs)
        if epochs < start_epoch:
            logger.info('%s has been trained for %g epochs. Fine-tuning for %g additional
epochs.' %
                        (weights, ckpt['epoch'], epochs))
            epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, state_dict

    # Image sizes
    gs = max(int(model.stride.max()), 32)  # grid size (max stride)
    nl = model.model[-1].nl  # number of detection layers (used for scaling hyp['obj'])
    imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_size]  # verify imgsz are gs-
multiples

    # DP mode
    if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)

    # SyncBatchNorm
    if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        logger.info('Using SyncBatchNorm()')

    # Trainloader
    dataloader, dataset = create_dataloader(train_path, imgsz, batch_size, gs, opt,
                                            hyp=hyp, augment=True, cache=opt.cache_images,
rect=opt.rect, rank=rank,
                                            world_size=opt.world_size, workers=opt.workers,
                                            image_weights=opt.image_weights, quad=opt.quad,
prefix=colorstr('train: '))
    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max label class
    nb = len(dataloader)  # number of batches
    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels are 0-%g' % (mlc,
nc, opt.data, nc - 1)

    # Process 0
    if rank in [-1, 0]:
        testloader = create_dataloader(test_path, imgsz_test, batch_size * 2, gs, opt,  #
testloader
                                       hyp=hyp, cache=opt.cache_images and not opt.notest,
rect=True, rank=-1,
                                       world_size=opt.world_size, workers=opt.workers,
                                       pad=0.5, prefix=colorstr('val: '))[0]

        if not opt.resume:
            labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                #plot_labels(labels, names, save_dir, loggers)
                if tb_writer:
                    tb_writer.add_histogram('classes', c, 0)

            # Anchors
            if not opt.noautoanchor:
                check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)
            model.half().float()  # pre-reduce anchor precision

    # DDP mode
    if cuda and rank != -1:
        model = DDP(model, device_ids=[opt.local_rank], output_device=opt.local_rank,
                    # nn.MultiheadAttention incompatibility with DDP
https://github.com/pytorch/pytorch/issues/26698
                    find_unused_parameters=any(isinstance(layer, nn.MultiheadAttention) for layer
in model.modules()))

    # Model parameters
    hyp['box'] *= 3. / nl  # scale to layers
    hyp['cls'] *= nc / 80. * 3. / nl  # scale to classes and layers
```

```
    hyp['obj'] *= (imgsz / 640) ** 2 * 3. / nl  # scale to image size and layers
    hyp['label_smoothing'] = opt.label_smoothing
    model.nc = nc  # attach number of classes to model
    model.hyp = hyp  # attach hyperparameters to model
    model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
    model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device) * nc  # attach
class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 1000)  # number of warmup iterations, max(3
epochs, 1k iterations)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of training
    maps = np.zeros(nc)  # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
    scheduler.last_epoch = start_epoch - 1  # do not move
    scaler = amp.GradScaler(enabled=cuda)
    compute_loss_ota = ComputeLossOTA(model)  # init loss class
    compute_loss = ComputeLoss(model)  # init loss class
    logger.info(f'Image sizes {imgsz} train, {imgsz_test} test\n'
                f'Using {dataloader.num_workers} dataloader workers\n'
                f'Logging results to {save_dir}\n'
                f'Starting training for {epochs} epochs...')
    torch.save(model, wdir / 'init.pt')
    for epoch in range(start_epoch, epochs):  # epoch ------------------------------------------
        model.train()

        # Update image weights (optional)
        if opt.image_weights:
            # Generate indices
            if rank in [-1, 0]:
                cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # class weights
                iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)  # image
weights
                dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)  #
rand weighted idx
            # Broadcast if DDP
            if rank != -1:
                indices = (torch.tensor(dataset.indices) if rank == 0 else
torch.zeros(dataset.n)).int()
                dist.broadcast(indices, 0)
                if rank != 0:
                    dataset.indices = indices.cpu().numpy()

        # Update mosaic border
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
        # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

        mloss = torch.zeros(4, device=device)  # mean losses
        if rank != -1:
            dataloader.sampler.set_epoch(epoch)
        pbar = enumerate(dataloader)
        logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 'total',
'labels', 'img_size'))
        if rank in [-1, 0]:
            pbar = tqdm(pbar, total=nb)  # progress bar
        optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar:  # batch -------------------------------------
            ni = i + nb * epoch  # number integrated batches (since train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255
to 0.0-1.0

            # Warmup
            if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio (obj_loss = 1.0 or
iou)
                accumulate = max(1, np.interp(ni, xi, [1, nbs / total_batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
                    # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0,
x['initial_lr'] * lf(epoch)])
                    if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'],
hyp['momentum']])

            # Multi-scale
            if opt.multi_scale:
```

```
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  # size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  # new shape
(stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilinear', align_corners=False)

            # Forward
            with amp.autocast(enabled=cuda):
                pred = model(imgs)  # forward
                if hyp['loss_ota'] == 1:
                    loss, loss_items = compute_loss_ota(pred, targets.to(device), imgs)  # loss
scaled by batch_size
                else:
                    loss, loss_items = compute_loss(pred, targets.to(device))  # loss scaled by
batch_size
                if rank != -1:
                    loss *= opt.world_size  # gradient averaged between devices in DDP mode
                if opt.quad:
                    loss *= 4.

            # Backward
            scaler.scale(loss).backward()

            # Optimize
            if ni % accumulate == 0:
                scaler.step(optimizer)  # optimizer.step
                scaler.update()
                optimizer.zero_grad()
                if ema:
                    ema.update(model)

            # Print
            if rank in [-1, 0]:
                mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
                mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9 if torch.cuda.is_available()
else 0)  # (GB)
                s = ('%10s' * 2 + '%10.4g' * 6) % (
                    '%g/%g' % (epoch, epochs - 1), mem, *mloss, targets.shape[0], imgs.shape[-1])
                pbar.set_description(s)

                # Plot
                if plots and ni < 10:
                    f = save_dir / f'train_batch{ni}.jpg'  # filename
                    Thread(target=plot_images, args=(imgs, targets, paths, f),
daemon=True).start()
                    # if tb_writer:
                    #     tb_writer.add_image(f, result, dataformats='HWC', global_step=epoch)
                    #     tb_writer.add_graph(torch.jit.trace(model, imgs, strict=False), [])  #
add model graph
                elif plots and ni == 10 and wandb_logger.wandb:
                    wandb_logger.log({"Mosaics": [wandb_logger.wandb.Image(str(x),
caption=x.name) for x in
                                                  save_dir.glob('train*.jpg') if x.exists()]})

            # end batch ------------------------------------------------------------------------
        # end epoch ----------------------------------------------------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups]  # for tensorboard
        scheduler.step()

        # DDP process 0 or single-GPU
        if rank in [-1, 0]:
            # mAP
            ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names', 'stride',
'class_weights'])
            final_epoch = epoch + 1 == epochs
            if not opt.notest or final_epoch:  # Calculate mAP
                wandb_logger.current_epoch = epoch + 1
                results, maps, times = test.test(data_dict,
                                                 batch_size=batch_size * 2,
                                                 imgsz=imgsz_test,
                                                 model=ema.ema,
                                                 single_cls=opt.single_cls,
                                                 dataloader=testloader,
                                                 save_dir=save_dir,
                                                 verbose=nc < 50 and final_epoch,
                                                 plots=plots and final_epoch,
```

```
                                                wandb_logger=wandb_logger,
                                                compute_loss=compute_loss,
                                                is_coco=is_coco)

            # Write
            with open(results_file, 'a') as f:
                f.write(s + '%10.4g' * 7 % results + '\n')  # append metrics, val_loss
            if len(opt.name) and opt.bucket:
                os.system('gsutil cp %s gs://%s/results/results%s.txt' % (results_file,
opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_loss',  # train loss
                    'metrics/precision', 'metrics/recall', 'metrics/mAP_0.5',
'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',  # val loss
                    'x/lr0', 'x/lr1', 'x/lr2']  # params
            for x, tag in zip(list(mloss[:-1]) + list(results) + lr, tags):
                if tb_writer:
                    tb_writer.add_scalar(tag, x, epoch)  # tensorboard
                if wandb_logger.wandb:
                    wandb_logger.log({tag: x})  # W&B

            # Update best mAP
            fi = fitness(np.array(results).reshape(1, -1))  # weighted combination of [P, R,
mAP@.5, mAP@.5-.95]
            if fi > best_fitness:
                best_fitness = fi
            wandb_logger.end_epoch(best_result=best_fitness == fi)

            # Save model
            if (not opt.nosave) or (final_epoch and not opt.evolve):  # if save
                ckpt = {'epoch': epoch,
                        'best_fitness': best_fitness,
                        'training_results': results_file.read_text(),
                        'model': deepcopy(model.module if is_parallel(model) else model).half(),
                        'ema': deepcopy(ema.ema).half(),
                        'updates': ema.updates,
                        'optimizer': optimizer.state_dict(),
                        'wandb_id': wandb_logger.wandb_run.id if wandb_logger.wandb else None}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                if (best_fitness == fi) and (epoch >= 200):
                    torch.save(ckpt, wdir / 'best_{:03d}.pt'.format(epoch))
                if epoch == 0:
                    torch.save(ckpt, wdir / 'epoch_{:03d}.pt'.format(epoch))
                elif ((epoch+1) % 25) == 0:
                    torch.save(ckpt, wdir / 'epoch_{:03d}.pt'.format(epoch))
                elif epoch >= (epochs-5):
                    torch.save(ckpt, wdir / 'epoch_{:03d}.pt'.format(epoch))
                if wandb_logger.wandb:
                    if ((epoch + 1) % opt.save_period == 0 and not final_epoch) and
opt.save_period != -1:
                        wandb_logger.log_model(
                            last.parent, opt, epoch, fi, best_model=best_fitness == fi)
                del ckpt

        # end epoch ----------------------------------------------------------------------------------
    # end training
    if rank in [-1, 0]:
        # Plots
        if plots:
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb_logger.wandb:
                files = ['results.png', 'confusion_matrix.png', *[f'{x}_curve.png' for x in
('F1', 'PR', 'P', 'R')]]
                wandb_logger.log({"Results": [wandb_logger.wandb.Image(str(save_dir / f),
caption=f) for f in files
                                              if (save_dir / f).exists()]})
        # Test best.pt
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start_epoch + 1,
(time.time() - t0) / 3600))
        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO
            for m in (last, best) if best.exists() else (last):  # speed, mAP tests
                results, _, _ = test.test(opt.data,
                                          batch_size=batch_size * 2,
```

```
                                            imgsz=imgsz_test,
                                            conf_thres=0.001,
                                            iou_thres=0.7,
                                            model=attempt_load(m, device).half(),
                                            single_cls=opt.single_cls,
                                            dataloader=testloader,
                                            save_dir=save_dir,
                                            save_json=True,
                                            plots=False,
                                            is_coco=is_coco)

        # Strip optimizers
        final = best if best.exists() else last  # final model
        for f in last, best:
            if f.exists():
                strip_optimizer(f)  # strip optimizers
        if opt.bucket:
            os.system(f'gsutil cp {final} gs://{opt.bucket}/weights')  # upload
        if wandb_logger.wandb and not opt.evolve:  # Log the stripped model
            wandb_logger.wandb.log_artifact(str(final), type='model',
                                            name='run_' + wandb_logger.wandb_run.id + '_model',
                                            aliases=['last', 'best', 'stripped'])
        wandb_logger.finish_run()
    else:
        dist.destroy_process_group()
    torch.cuda.empty_cache()
    return results

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolo7.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.p5.yaml',
help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all
GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train,
test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most
recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster
training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image
selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as
single-class')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available
in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not
modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader
workers')
    parser.add_argument('--project', default='runs/train', help='save to project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not
increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear LR')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing
epsilon')
    parser.add_argument('--upload_dataset', action='store_true', help='Upload dataset as W&B
artifact table')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='Set bounding-box image
logging interval for W&B')
    parser.add_argument('--save_period', type=int, default=-1, help='Log model after every
"save_period" epoch')
```

```
    parser.add_argument('--artifact_alias', type=str, default="latest", help='version of dataset
artifact to be used')
    parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers:
backbone of yolov7=50, first3=0 1 2')
    opt = parser.parse_args()

    # Set DDP variables
    opt.world_size = int(os.environ['WORLD_SIZE']) if 'WORLD_SIZE' in os.environ else 1
    opt.global_rank = int(os.environ['RANK']) if 'RANK' in os.environ else -1
    set_logging(opt.global_rank)
    #if opt.global_rank in [-1, 0]:
    #    check_git_status()
    #    check_requirements()

    # Resume
    wandb_run = check_wandb_resume(opt)
    if opt.resume and not wandb_run:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  # specified or
most recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        apriori = opt.global_rank, opt.local_rank
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:
            opt = argparse.Namespace(**yaml.load(f, Loader=yaml.SafeLoader))  # replace
        opt.cfg, opt.weights, opt.resume, opt.batch_size, opt.global_rank, opt.local_rank = '',
ckpt, True, opt.total_batch_size, *apriori  # reinstate
        logger.info('Resuming training from %s' % ckpt)
    else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else 'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_file(opt.cfg),
check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be specified'
        opt.img_size.extend([opt.img_size[-1]] * (2 - len(opt.img_size)))  # extend to 2 sizes
(train, test)
        opt.name = 'evolve' if opt.evolve else opt.name
        opt.save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok |
opt.evolve)  # increment run

    # DDP mode
    opt.total_batch_size = opt.batch_size
    device = select_device(opt.device, batch_size=opt.batch_size)
    if opt.local_rank != -1:
        assert torch.cuda.device_count() > opt.local_rank
        torch.cuda.set_device(opt.local_rank)
        device = torch.device('cuda', opt.local_rank)
        dist.init_process_group(backend='nccl', init_method='env://')  # distributed backend
        assert opt.batch_size % opt.world_size == 0, '--batch-size must be multiple of CUDA
device count'
        opt.batch_size = opt.total_batch_size // opt.world_size

    # Hyperparameters
    with open(opt.hyp) as f:
        hyp = yaml.load(f, Loader=yaml.SafeLoader)  # load hyps

    # Train
    logger.info(opt)
    if not opt.evolve:
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
            prefix = colorstr('tensorboard: ')
            logger.info(f"{prefix}Start with 'tensorboard --logdir {opt.project}', view at
http://localhost:6006/")
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer)

    # Evolve hyperparameters (optional)
    else:
        # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
        meta = {'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
                'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
                'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
                'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
                'box': (1, 0.02, 0.2),  # box loss gain
                'cls': (1, 0.2, 4.0),  # cls loss gain
                'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
                'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
                'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
```

```
                'iou_t': (0, 0.1, 0.7),  # IoU training threshold
                'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
                'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
                'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default gamma=1.5)
                'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
                'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation (fraction)
                'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
                'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
                'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
                'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
                'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
                'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction), range 0-
0.001
                'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
                'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
                'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
                'mixup': (1, 0.0, 1.0),   # image mixup (probability)
                'copy_paste': (1, 0.0, 1.0),  # segment copy-paste (probability)
                'paste_in': (1, 0.0, 1.0)}    # segment copy-paste (probability)

        with open(opt.hyp, errors='ignore') as f:
            hyp = yaml.safe_load(f)  # load hyps dict
            if 'anchors' not in hyp:  # anchors commented in hyp.yaml
                hyp['anchors'] = 3

        assert opt.local_rank == -1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save best result here
        if opt.bucket:
            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # download evolve.txt if
exists

        for _ in range(300):  # generations to evolve
            if Path('evolve.txt').exists():  # if evolve.txt exists: select best hyps and mutate
                # Select parent(s)
                parent = 'single'  # parent selection method: 'single' or 'weighted'
                x = np.loadtxt('evolve.txt', ndmin=2)
                n = min(5, len(x))  # number of previous results to consider
                x = x[np.argsort(-fitness(x))][:n]  # top n mutations
                w = fitness(x) - fitness(x).min()  # weights
                if parent == 'single' or len(x) == 1:
                    # x = x[random.randint(0, n - 1)]  # random selection
                    x = x[random.choices(range(n), weights=w)[0]]  # weighted selection
                elif parent == 'weighted':
                    x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted combination

                # Mutate
                mp, s = 0.8, 0.2  # mutation probability, sigma
                npr = np.random
                npr.seed(int(time.time()))
                g = np.array([x[0] for x in meta.values()])  # gains 0-1
                ng = len(meta)
                v = np.ones(ng)
                while all(v == 1):  # mutate until a change occurs (prevent duplicates)
                    v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s +
1).clip(0.3, 3.0)
                for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
                    hyp[k] = float(x[i + 7] * v[i])  # mutate

            # Constrain to limits
            for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1])  # lower limit
                hyp[k] = min(hyp[k], v[2])  # upper limit
                hyp[k] = round(hyp[k], 5)  # significant digits

            # Train mutation
            results = train(hyp.copy(), opt, device)

            # Write mutation results
            print_mutation(hyp.copy(), results, yaml_file, opt.bucket)

        # Plot results
        plot_evolution(yaml_file)
        print(f'Hyperparameter evolution complete. Best results saved as: {yaml_file}\n'
              f'Command to train a new model with these hyperparameters: $ python train.py --hyp
{yaml_file}')
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ВИЯВЛЕННЯ РИБ НА ПІДВОДНИХ ЗОБРАЖЕННЯХ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ»

Виконав: студент 4 курсу,групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Байбак І. А.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О.К.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

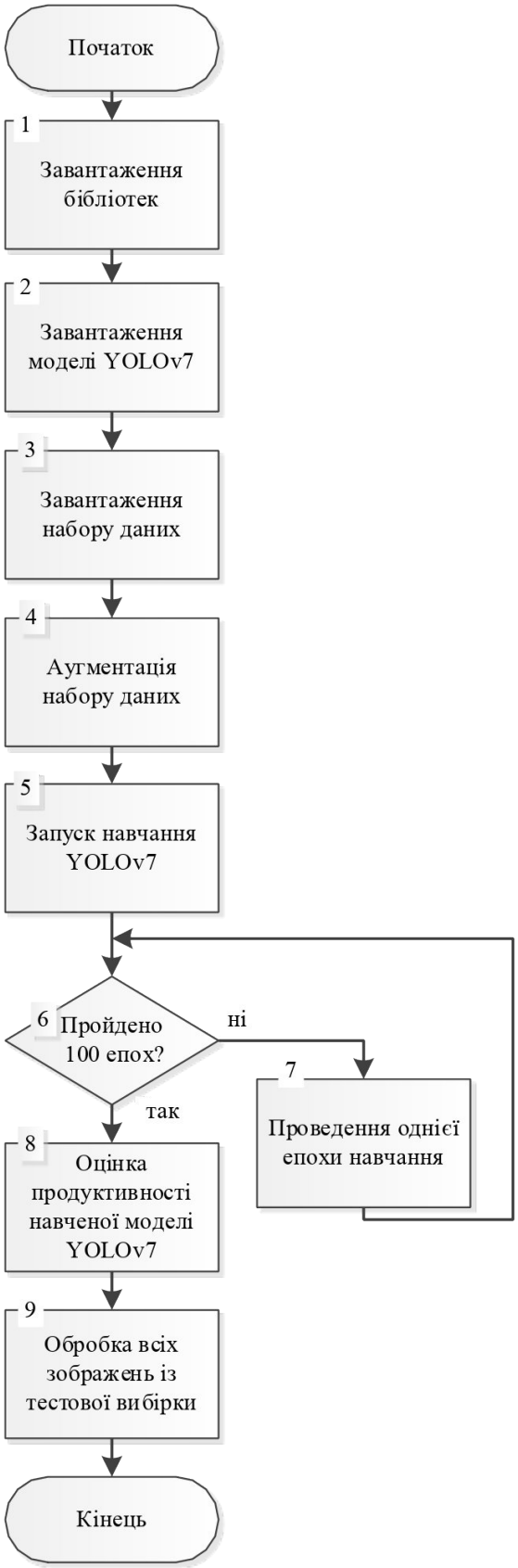


Рисунок В.1– Схема алгоритму роботи програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7

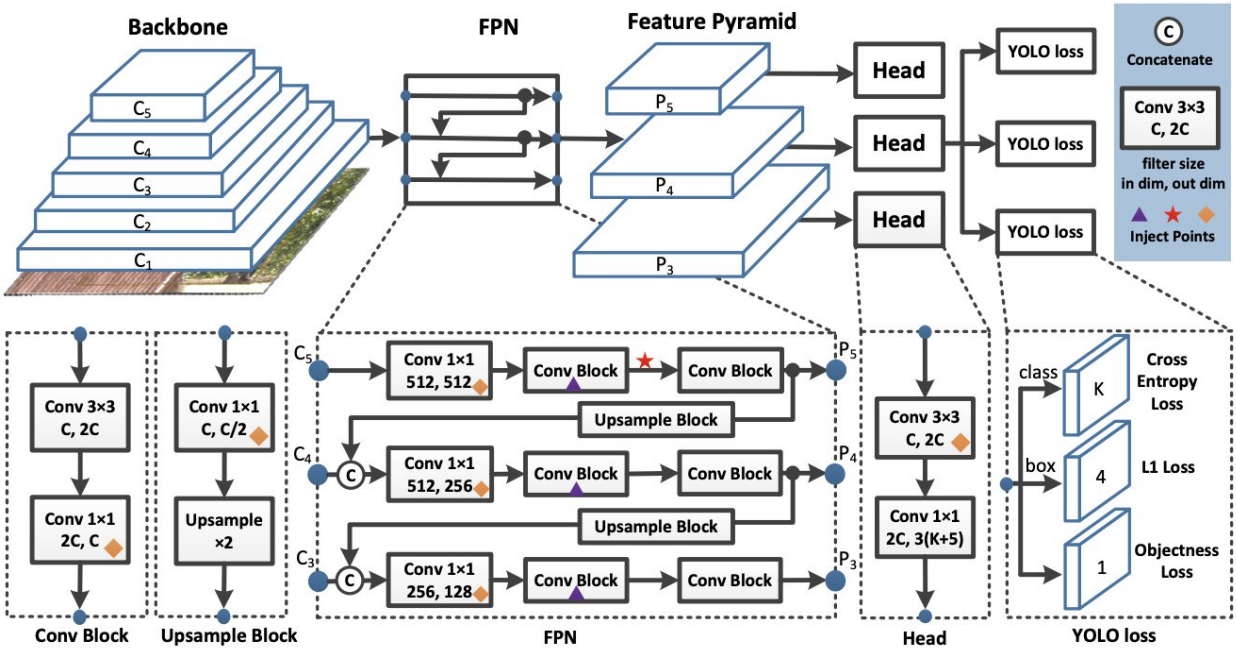


Рисунок В.2– Структура згорткової нейронної мережі YOLOv7

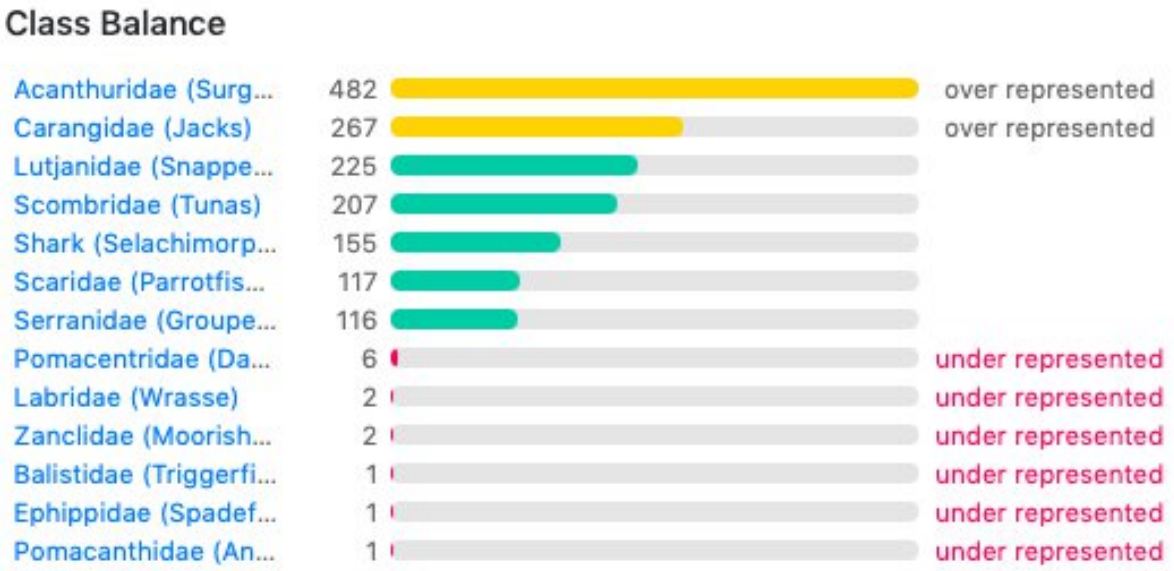


Рисунок В.3– Баланс класів риб у наборі даних Ozfish

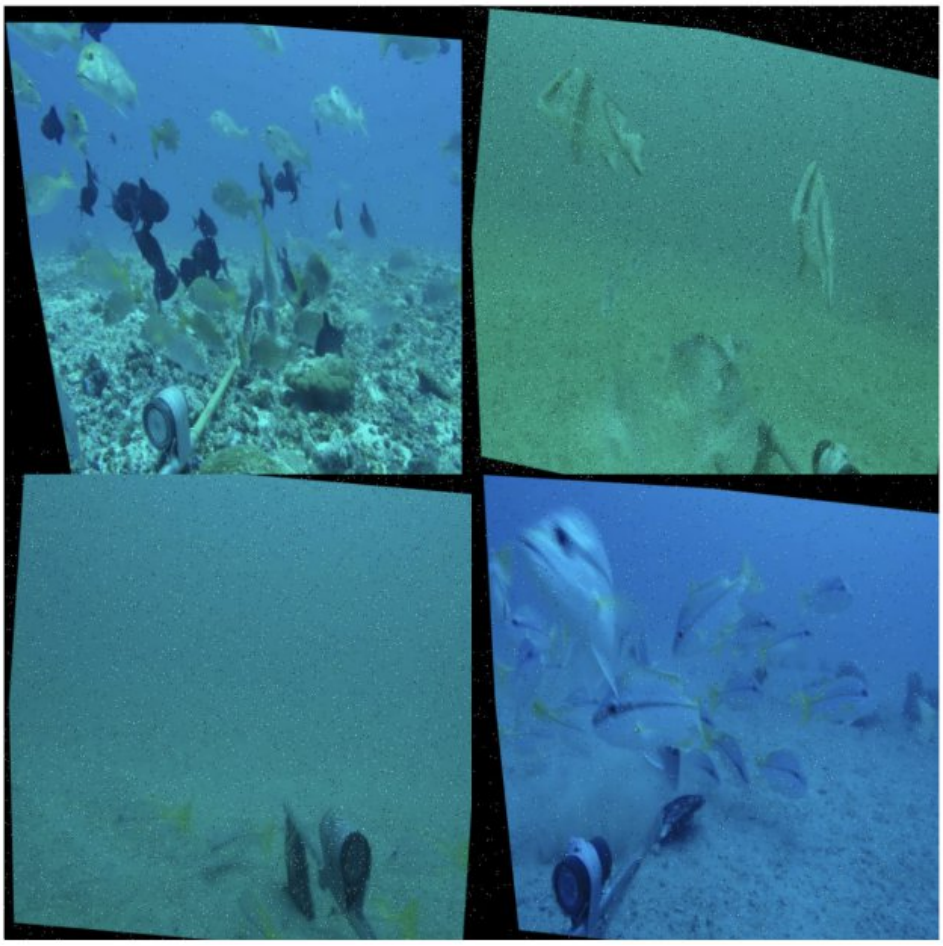


Рисунок В.4– Приклади аугментованих зображень з набору даних Ozfish

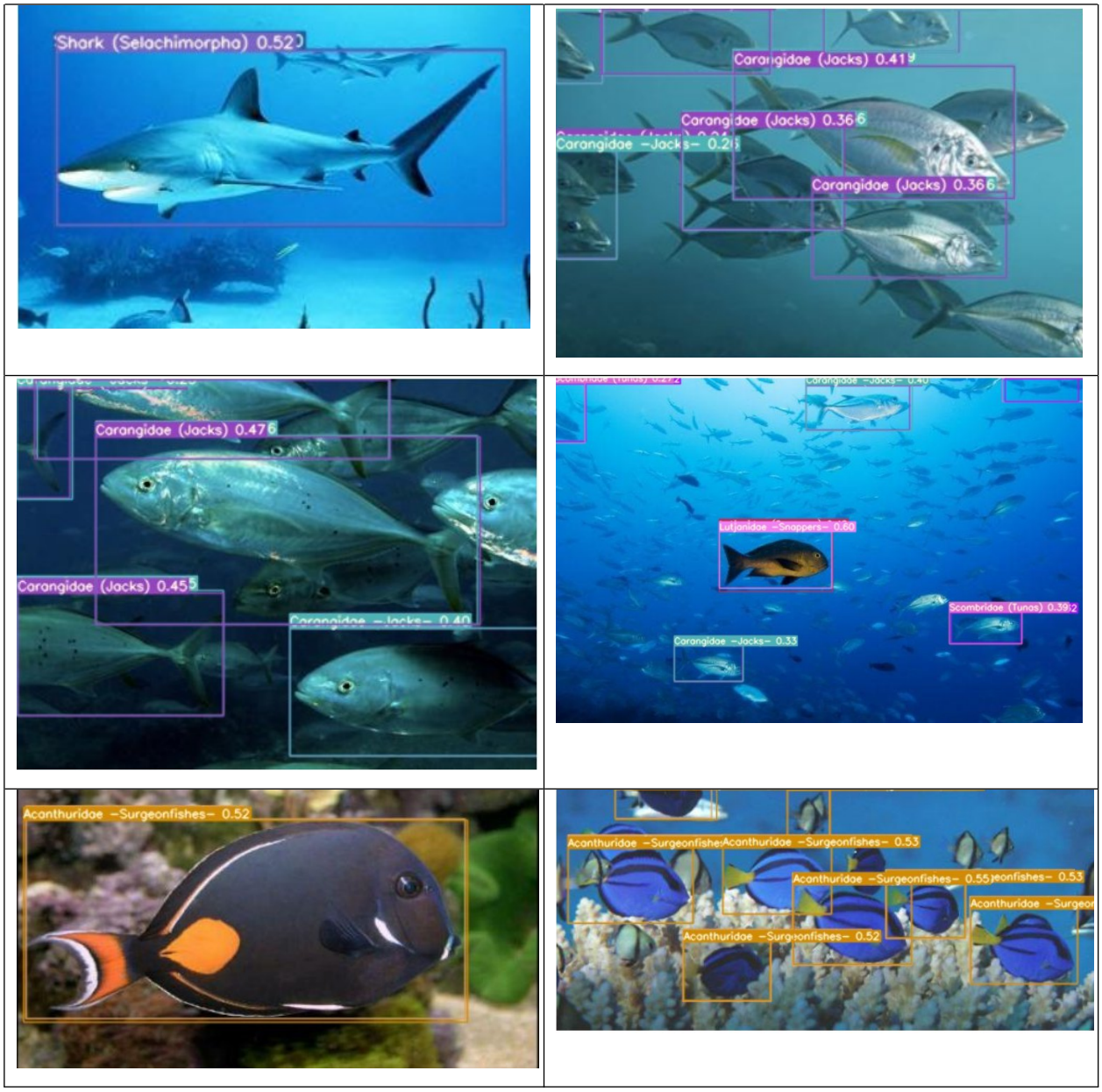


Рисунок В.5 – Результати роботи програмного модуля виявлення риб на підводних зображеннях на основі згорткової нейронної мережі YOLOv7

Модель	Середня точність	Середня повнота
Попередньо навчена	0,7628	0,9064
Перенесення навчання (без аугментації)	0,7365	0,7323
Перенесення навчання (з аугментацією)	0,7581	0,9140
«З нуля» (без аугментації)	0,7408	0,7312
«З нуля» (з аугментацією)	0,7366	0,7314

Рисунок В.6 – Показники точності різних модифікацій моделі YOLOv7 на датасеті Ozfish

Модель	Середня точність	Середня повнота
«З нуля» (100% навчальних даних)	0,7366	0,7314
Попередньо натренована (без перенесення навчання)	0,7628	0,9064
Перенесення навчання (25% навчальних даних)	0,6848	0,6902
Перенесення навчання (50% навчальних даних)	0,7193	0,7203
Перенесення навчання (75% навчальних даних)	0,7472	0,7332
Перенесення навчання (88% навчальних даних)	0,7521	0,8379
Перенесення навчання (100% навчальних даних)	0,7581	0,9140

Рисунок В.6 – Показники точності різних модифікацій моделі YOLOv7 на датасеті Ozfish з різними обсягами даних, використаних при перенесенні навчання