МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
«КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАДИМА ГЕТЬМАНА»

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Системи штучного інтелекту»

галузь знань	12 Інформаційні технології
спеціальність	122 Комп’ютерні науки

Форма навчання: денна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «**ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ ПІД ВОДОЮ НА ОСНОВІ МОДЕЛЕЙ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ**»

здобувача: Мазура Василя Вікторовича _____

Науковий керівник: д.т.н., професор Шевченко А.І. _____

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: к.е.н., доцент Тішков Б.О. _____

Київ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут «Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА

«Системи штучного інтелекту»

галузь знань 12 Інформаційні технології

спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____Рамазанов С.К.
«____» _____2024р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
«____» _____ 2024р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти__Мазура Василя Вікторовича_____

(прізвище, ім'я, по батькові)

_____очної (денної)_____ форми навчання

очної (денної), заочної, дистанційної

на підготовку кваліфікаційної магістерської роботи

на тему: *«ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ ПІД ВОДОЮ НА ОСНОВІ МОДЕЛЕЙ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ»*

Тему затверджено наказом ректора Університету від «__» _____ 2024 р. №_____

Кваліфікаційна магістерська робота виконується на матеріалах

_____

План кваліфікаційної магістерської роботи

Розділ І ТЕОРЕТИЧНИЙ РОЗДІЛ. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ПРОГРАМНОГО МОДУЛЮ НА ОСНОВІ ШТУЧНИЙ НЕЙРОННИХ МЕРЕЖ.

Розділ ІІ АНАЛІТИЧНИЙ РОЗДІЛ. ХАРАКТЕРИСТИКА ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ ТА ПОСТАНОВКА ЗАДАЧІ.

Розділ ІІІ КОНСТРУКТИВНИЙ РОЗДІЛ. РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ.

Об’єкт дослідження: Методи машинного навчання та комп’ютерного зору для розпізнавання вибухонебезпечних об’єктів під водою.
Предмет дослідження: Використання згорткової нейронної мережі, створення та підготовка датасету для її тренування.

Мета кваліфікаційної магістерської роботи: Доведення актуальності даної теми дослідження, її важливості та необхідності впровадження.

Завдання, які здобувач повинен виконати для досягнення поставленої мети:
У розділі I Проведено збір інформації та аналіз існуючих рішень в сфері розпізнавання підводних об'єктів за допомогою нейронних мереж, порівняно різні моделі та методи.

У розділі II Детально описано обрану модель нейронної мережі YOLOv8, її архітектури, принципів роботи та переваги. Поставлено конкретні задачі, що необхідно вирішити.

У розділі III Наведено етапи створення датасету, підготовки зображень, анотування, розроблено програмний модуль та базу даних, проведено тестування та проаналізовано результати.

Завдання підготував науковий керівник _____ _____
(підпис) *(ініціали, прізвище)*

«____» _____ 202_ р.

Завдання одержав здобувач _____ _____
(підпис) *(ініціали, прізвище)*

«____» _____ 202_ р.

РЕФЕРАТ

Кваліфікаційна магістерська робота містить 77 сторінок, 1 таблиць, _48_ рисунків, список літератури з 20 найменувань, 5 додатків.

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ ПІД ВОДОЮ НА ОСНОВІ МОДЕЛЕЙ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Об’єктом дослідження виступають методи машинного навчання та комп’ютерного зору для автоматичного розпізнавання вибухонебезпечних об’єктів під водою за допомогою відеоданих.

Суб’єктом дослідження є програмний модуль, який використовує моделі штучних нейронних мереж для виявлення та класифікації підводних об’єктів, таких як боєприпаси.

Предметом дослідження є використання згорткової нейронної мережі (CNN), створення та підготовка датасету для її тренування.

Мета кваліфікаційної магістерської роботи полягає в доведенні актуальності даної теми дослідження, її важливості та необхідності впровадження.

Завданнями кваліфікаційної магістерської роботи є розробка програмного модулю для розпізнавання підводних об’єктів, підготовка датасету та його анотування, тестування функціональності.

Апаратура використана при дослідженні: персональний комп’ютер, роутер, камера.

Одержані результати можуть бути використані у військовій сфері для розпізнавання вибухонебезпечних об’єктів на водоймах України.

ABSTRACT

The Master's thesis comprises 38 pages, 1 table, 2 figures, a bibliography with a list of sources, and appendices.

SOFTWARE MODULE FOR UNDERWATER OBJECT RECOGNITION BASED ON ARTIFICIAL NEURAL NETWORK MODELS

The research object is machine learning methods and computer vision for the automatic recognition of explosive objects underwater using video data.

The research subject is a software module that utilizes artificial neural network models for the detection and classification of underwater objects such as ammunition.

The research subject involves the use of a convolutional neural network (CNN), the creation, and preparation of a dataset for its training.

The goal of the Master's thesis is to demonstrate the relevance of this research topic, its importance, and the necessity of implementation.

The tasks of the Master's thesis are the development of a software module for the recognition of underwater objects, the preparation of the dataset and its annotation, and testing the accuracy of recognition.

Equipment used in the research: personal computer, router, camera.

The results obtained can be used in the military sector for the recognition of explosive objects in Ukrainian waters.

АНОТАЦІЯ

кваліфікаційної магістерської роботи

здобувача 6 курсу

Мазура Василя Вікторовича виконану на тему:

«Програмний модуль розпізнавання об'єктів під водою на основі моделей штучних нейронних мереж»

Кваліфікаційна магістерська робота присвячена актуальній проблемі розпізнавання вибухонебезпечних об’єктів в українських водоймах за допомогою моделей штучних нейронних мереж.

Кваліфікаційна магістерська робота складається з трьох розділів, логічно пов’язаних між собою.

Перший розділ є теоретичним. Проведено збір інформації та аналіз існуючих рішень в сфері розпізнавання підводних об'єктів за допомогою нейронних мереж, порівняно різні моделі та методи.

Другий розділ є проєктним. Детально описано обрану модель нейронної мережі YOLOv8, її архітектури, принципів роботи та переваги. Поставлено конкретні задачі, що необхідно вирішити.

Третій розділ – конструктивний. Наведено етапи створення датасету, підготовки зображень, анотування, розроблено програмний модуль та базу даних, проведено тестування та проаналізовано результати.

Висновки містять рекомендації щодо розробки та впровадження такої системи, також описано можливі варіанти покращення програмного модулю та алгоритмів навчання нейромережі.

ЗМІСТ

ВСТУП

Нейронні мережі, також відомі як штучні нейронні мережі (ANN), є одним із напрямків у створенні штучного інтелекту і спрямовані на відтворення аналітичної роботи людського мозку.

Вчені розробляють штучні нейронні мережі близько 70 років. Але якщо у 2016 році роботи, які давали цікаві відповіді на запитання, були вершиною їхніх можливостей, то сьогодні нове покоління нейронних мереж може писати книги, складати музику, розробляти маркетингові стратегії та навіть рятувати життя [1].

Розпізнавання об'єктів являє собою техніку комп’ютерного зору для ідентифікації об'єктів з відеопотоку чи зображень та є ключовим результатом алгоритмів глибокого та машинного навчання. Мета полягає у навчанні комп’ютера речам, які природні для людини, а саме отримання рівня розуміння того, що знаходиться на відео або зображенні [2] с.5.

Для розпізнавання об’єктів можна використовувати різноманітні підходи, проте останнім часом зростає популярність глибокого навчання. Моделі глибокого навчання використовуються для автоматичного вивчення властивостей об'єкта та його ідентифікації.

Ця технологія охопила багато аспектів людського життя. У 2023 році світовий ринок глибокого навчання становив майже 70 мільярдів доларів США, а до 2033 року перевищить 1185 мільярдів доларів США. Вони прогнозують безпрецедентні середньорічні темпи зростання на рівні 32,57% у наступному десятилітті [3].

Розглянемо два підходи розпізнавання об’єктів з використанням глибокого навчання:

1) Навчання з нуля. Для навчання такої моделі необхідно зібрати великий набір міток даних та спроектувати мережеву архітектуру,

яка вивчатиме функції й будуватиме модель. Даний підхід може принести неймовірні результати, проте вимагає дуже велику кількість даних, апаратних потужностей та кропітливого налаштування вагів і шарів.

2) Попередньо навчена модель. Значна кількість додатків використовує підхід передачі навчання, що передбачає точне налаштування перевіреної моделі. Передбачено початок з існуючої мережі (наприклад, GoogLeNet) та подачу нових даних з невідомими раніше класами. Даний метод менш трудомісткий та забезпечує більш швидкий результат за умови якісно побудованого та підготовленого датасету.

Також можливо розпізнавати об’єкти за допомогою стандартного підходу до машинного навчання. У такому випадку необхідно почати з колекції зображень та обрати до кожного з них відповідні функції. В більшості випадків, потрібно різні алгоритми машинного навчання та методи вилучення функцій з безліччю комбінацій для створення точної моделі розпізнавання об’єктів.

Кожна з моделей має ряд переваг та недоліків. Модель глибокого навчання гарно працює з великими датасетами, потребує більшої продуктивності від системи, проте пришвидшує процес навчання за допомогою використання потужностей графічного процесору (GPU). Машинне навчання хоч і має меншу ресурсозатратність, але більше підходить для невеликої кількості вхідних даних.

У даній роботі для розпізнавання об’єктів буде використовуватись глибоке навчання з підходом попередньо навченої моделі. Основним етапом стане побудова та підготовка датасету, що включатиме сім основних етапів.

І. Теоретичний розділ. Дослідження та аналіз підходів до створення програмного модулю на основі штучний нейронних мереж

1.1 Дослідження предметної області. Збір інформації та вивчення матеріалів з теми кваліфікаційної магістерської роботи.

Комп'ютерний зір - це галузь інформатики, яка зосереджена на створенні алгоритмів і технологій, що дають змогу комп'ютерам сприймати і аналізувати візуальну інформацію подібно до людського ока. Головна мета комп'ютерного зору - навчити комп'ютери розпізнавати об'єкти, ідентифікувати їхні особливості, рухи та інші характеристики на зображеннях або відео.

Ключові аспекти комп'ютерного зору включають в себе [4]:

1) Розпізнавання об'єктів: Це стосується здатності комп'ютера виявляти та ідентифікувати об'єкти на зображеннях або відео, включаючи обличчя, тварин, автомобілі та інші об'єкти.
2) Визначення руху: Системи комп'ютерного зору можуть відстежувати рух об'єктів на відео, що корисно для моніторингу рухомих об'єктів або їхньої взаємодії.
3) Визначення глибини: Деякі алгоритми комп'ютерного зору можуть оцінювати глибину об'єктів на зображеннях, що є важливим для створення тривимірних моделей або симуляцій.
4) Розпізнавання дій та жестів: Комп'ютерний зір може використовуватися для розпізнавання рухів рук, обличчя або тіла людини, що дозволяє взаємодіяти з комп'ютером за допомогою жестів.
5) Автоматизоване визначення контексту: Системи комп'ютерного зору можуть аналізувати зображення для визначення контексту навколишньої ситуації, що може бути корисним для автоматизованих систем і роботів.

Комп'ютерний зір в умовах підводного середовища представляє собою комплекс технологій та методів, спрямованих на розвиток систем візуального сприйняття, аналізу та інтерпретації зображень під водою. У порівнянні з атмосферними умовами, підводний світ видається великим викликом для комп'ютерного зору через специфічні обмеження, які включають в себе високий рівень розсіювання світла, знижену видимість, біологічні перешкоди та гідродинамічний вплив води на зображення.

Комп'ютерний зір у підводних умовах використовує спеціалізовані оптичні та акустичні сенсори, такі як водонепроникні камери та сонари, здатні працювати в умовах низької видимості та високого розсіювання світла. Ці сенсори зазвичай встановлюються на підводних апаратах, дронах або спеціальних роботах для збору зображень та відеоматеріалів. Важливим завданням у цій галузі є розробка алгоритмів обробки зображень та відео, які ефективно виявляють та класифікують об'єкти у водному середовищі. Такі алгоритми повинні справлятися з високим рівнем шуму на зображеннях, враховувати зміни в освітленості та видимості, а також розпізнавати об'єкти, навіть якщо вони частково приховані або змінили форму через гідродинамічні ефекти [5].

Додатково, у підводній робототехніці використовуються технології комп'ютерного зору для автономної навігації та управління підводними апаратами. Системи комп'ютерного зору можуть допомагати апаратам уникати перешкод, розпізнавати цільові об'єкти та вимірювати глибину.

Завдяки швидкому розвитку наукових досліджень і промислових технологій, сучасні системи виявлення об'єктів у підводному середовищі стали надзвичайно потужними та точними. Одним із ключових досягнень є використання автономних безпілотних підводних апаратів (АБПА). Ці сучасні АБПА оснащені високоякісними камерами, лазерними сканерами,

гідроакустичними системами та іншими сенсорами, які дозволяють їм здійснювати дослідження на значних глибинах. Вони використовуються для картографування дна океану, вивчення морських екосистем, досліджень археологічних об'єктів, а також для пошуку й у воєнних місіях [6]. Технології обробки зображень і штучного інтелекту виграють ключову роль у сучасних системах виявлення об'єктів у водяному середовищі. Алгоритми глибокого навчання дозволяють автоматично аналізувати великі обсяги зібраних даних, виявляти підводні об'єкти та робити точні класифікації.

У таблиці наведені основні тактико-технічні дані найбільш розповсюджених протимінних підводних апаратів, що знаходяться на озброєнні ВМС зарубіжних країн.

Таблиця 1.1.1 – Основні технічні дані протимінних підводних апаратів [7]

Тип апарата, країна-виробник	Маса апарата, кг	Розміри: довжина×ширина×висота, м	Швидкість ходу максимальна/робоча, вуз	Автономність, год	Довжина кабелю, м Дальність дії	Глибина занурення, м	Маса підривного заряду, кг	Перебуває на озброєнні
Дистанційно керовані підводні апарати								
PAP-104 Мк5, Франція	750	2,7×1,2×1,3	5,3	4–5	1000 –	300	128	Великобританія, ФРН, Франція, Нідерланди, Бельгія, ОАЕ та ін.
«Пінгвін-В3», ФРН	1350	3,5×1,6×1,4	8/3	60–120	1000 –	100–200	2×128	ФРН, Тайвань
AN/SQL-48, США	1130, 1247	3,67×1,2×1,2	6	необмежена	1000	більш 200	128	США

«Pluto», Італія	160	1,68×0,6×0,65	5	4×0,5	500	300	15	Італія, Іспанія
«Pluto Gigas», Італія	600	3,4×–×–	7,5	12	2000	1000	128	Італія
«Sea Fox», ФРН	42	1,31×0,39×0,2	6	-	1300	300	1,5	ФРН, США, Бельгія, Швеція, Нідерланди, Великобританія та ін
«K-Ster», Франція	50	1,4×0,23×0,23	6/3	1	1100	300	6	Франція
AN/WLD–1, США	7300	7,3×1,0×1,2	16/12	72	600 миль	200	-	США, Франція, Канада

Аналіз стану та перспектив розвитку підводних апаратів-міношукачів на флотах провідних морських держав показав, що протягом останніх 30 років зроблено велику кількість протимінних підводних апаратів, що свідчить про актуальність задачі боротьби з морськими мінами на морях, де в минулому велися активні воєнні дії. У зв’язку з цим створення такого виду морської техніки є актуальним для розв’язання протимінних задач у територіальних водах України.

1.2 Аналіз існуючих СШІ й інтелектуальних систем предметної області.

Існує безліч подібних алгоритмів, що вирішують широкий спектр завдань у галузі розпізнавання. Для комп'ютерного зору було розроблено багато різних архітектурних рішень нейронних мереж, але згорткова нейронна мережа (convolutional neural network (CNN)) є ключовою у багатьох моделях.

У даній роботі не будуть розглянуті моделі з архітектурою Regions With Convolution Neural Networks (R-CNNs) та їхні похідні, які чудово справляються з завданням розпізнавання в автономному режимі, але абсолютно не підходять для реалізації в реальному часі. Такі алгоритми використовують двоетапний підхід — спочатку визначають області, де об'єкти повинні бути знайдені, а потім виявляють об'єкти тільки в цих, окремих регіонах, використовуючи згорткові нейронні мережі.

Вибір моделі для виявлення об'єктів, яка відповідає вимогам як за швидкістю, так і за точністю стає важливим. Одною з таких моделей є YOLO (You Only Look Once) – це одноступінчастий детектор об'єктів, який задовольняє обом цілям (швидко і точно).

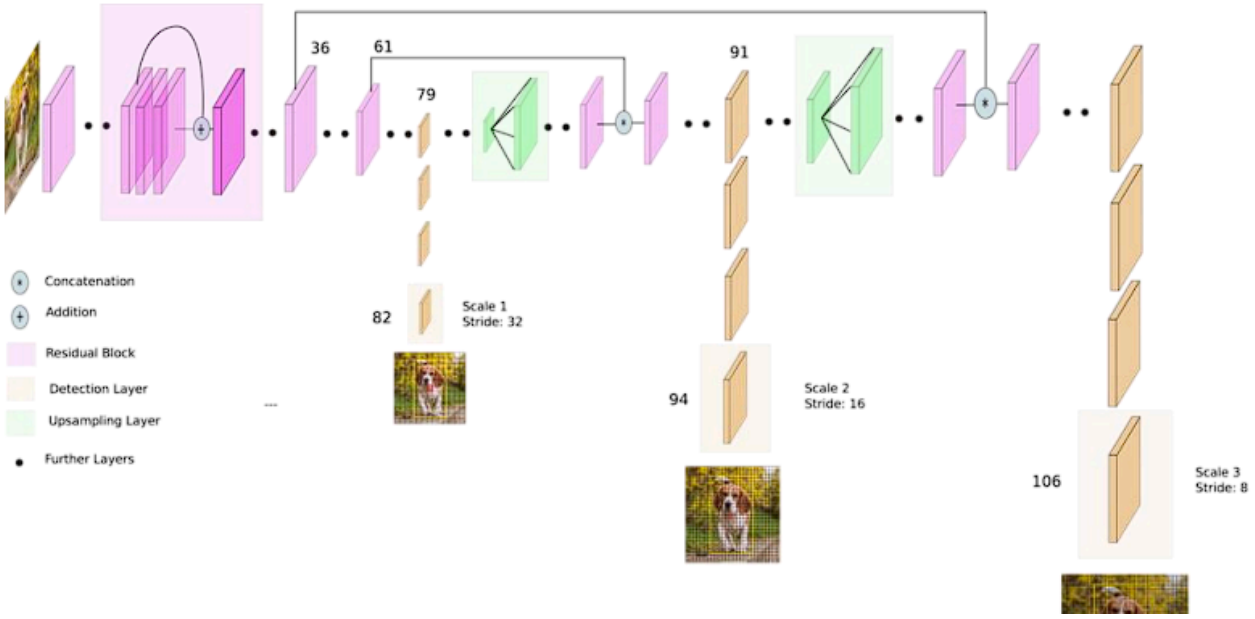


Рисунок 1.2.1 – Архітектура YOLO

(Джерело: https://link.springer.com/article/10.1007/s11042-023-17933-y)

Оскільки YOLO не проходить етап пропозиції регіону та передбачає лише обмежену кількість обмежувальних рамок, він здатний дуже швидко прогнозувати наявність та розташування об’єктів на зображенні. YOLO поєднує як створення пропозицій, так і класифікацію в один крок. Це усуває потребу в окремому етапі пропозиції регіону, значно зменшуючи витрати на обчислення та забезпечуючи швидший висновок. Це значна перевага YOLO порівняно з іншими алгоритмами розпізнавання об’єктів, які використовують двоетапний підхід.

Робочий процес YOLO:

1) Навчання мережі класифікації зображень.
2) Розбиття зображення на $S \times S$ клітини. Якщо центр об’єкта потрапляє в клітину, ця клітина «відповідає» за виявлення існування цього об’єкта. Кожна клітина передбачає розташування B обмежувальних рамок, оцінку достовірності та ймовірність класу об’єкта, залежну від існування об’єкта в обмежувальній рамці[1].

- Координати обмежувальної рамки визначаються кортежем із 4 значень–(x, y, w, h), де x і y встановлюються як зміщення розташування клітини.
- Оцінка достовірності (ймовірність помножена на перетин через об’єднання)
- Якщо клітина містить об’єкт, вона передбачає ймовірність належності цього об’єкта до кожного класу. Однак YOLO безпосередньо не виводить цю ймовірність. Замість цього він використовує функцію softmax як функцію активації на останньому рівні.
- Одне зображення містить $S \times S \times B$ обмежувальних рамок, кожна рамка відповідає 4 прогнозам розташування, оцінці достовірності і K умовних ймовірностей для класифікації об'єктів.

3) Останній рівень попередньо навченої згорткової нейронної мережі модифікується для виведення тензора прогнозування розміру $S \times S \times (5B+K)$.

YOLO має труднощі з розпізнаванням нестандартних об'єктів, а також груп дрібних об'єктів, що обумовлено використанням сітки. Крім того він демонструє чутливість до змін освітлення, має меншу стійкість до шуму та артефактів, присутніх у зображеннях, що пояснюється залежністю від згорткових нейронних мереж та навчальних даних, а також обмеженнями одноетапного підходу та функції втрат.

Переваги YOLO [8]:

- швидкість роботи. YOLO добре підходить для обробки зображень в реальному часі; – прогнози робляться однією нейронною мережею. Вся модель тренується разом.
- YOLO більш генералізована. Ця модель працює краще інших методів при застосуванні на інших типах зображень, наприклад на картинах;
- методи пропозиції регіонів обмежують класифікатор в конкретному регіоні. YOLO має доступ до цілого зображення під час прогнозу меж об’єктів. Використовуючи додатковий контекст, YOLO демонструє менше помилкових позитивів в фонових ділянках зображення;
- YOLO розпізнає один об’єкт в кожній клітині. Це забезпечує просторове різноманіття під час виконання прогнозів.

SSD – це алгоритм глибокого навчання, який використовується для розпізнавання об’єктів на зображеннях і відео.

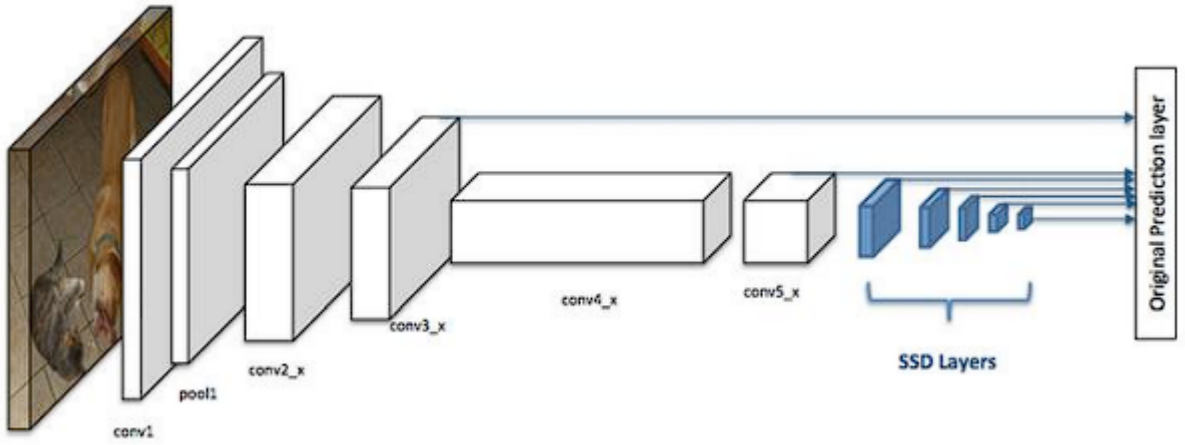


Рисунок 1.2.2 – Архітектура згорткової нейронної мережі з детектором SSD

(Джерело: https://link.springer.com/chapter/10.1007/978-3-031-24506-0_2)

Представлений в 2016 році, він вперше використовув пірамідальну ієрархію ознак згорткової нейронної мережі для точного виявлення об’єктів різного розміру. Як правило, архітектура SSD зазвичай складається з базової мережі, такої як VGG або ResNet, яка попередньо навчена на великому наборі даних класифікації зображень, наприклад ImageNet. Після цієї базової мережі слідує кілька додаткових шарів, які додаються поверх базової мережі. Їх можна розглядати як пірамідальне представлення зображень у різних масштабах. Карти ознак з більшою деталізацією знаходяться на ранніх рівнях мережі та володіють кращою роздільною здатністю, а карти ознак з меншою деталізацією розміщені на пізніших рівнях мережі та мають нижчу роздільну здатність. На відміну від YOLO, SSD не розбиває зображення на сітки довільного розміру, а передбачає зміщення попередньо визначених прив'язок кожного місця на карті ознак. Кожний блок має фіксований розмір і положення відносно відповідної клітини. Усі блоки прив’язки розміщують всю карту об’єктів у згортковий спосіб.

Основні особливості SSD:

- SSD здійснює багато прогнозів і має гарне покриття на зображенні.

- Через видалення регіонів і використання відео-послідовностей із більш низькою роздільною здатністю модель може працювати в режимі реального часу.
- Продуктивність SSD512 з використанням ЗНН (згорткової нейронної мережі) із низьким обсягом пам’яті HarDNet85 має наступні параметри: показник точності mAP@0.5 = 35,1; значення FPS = 34.

FPN – архітектура, яка розроблена для поєднання ознак зрізних рівнів згорткової мережі, щоб краще виявляти об’єкти в різних масштабах.

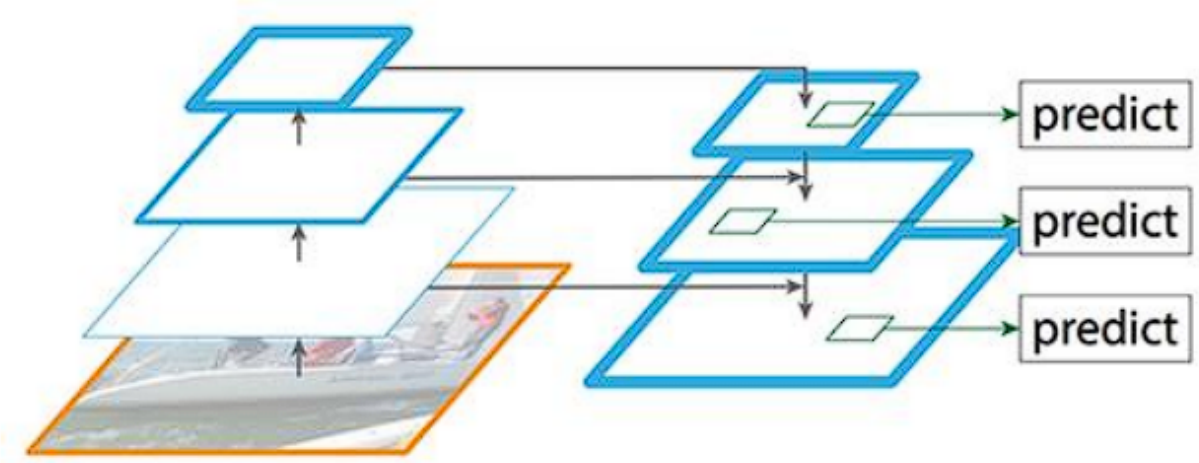


Рисунок 1.2.3 – Спадна архітектура FPN з незалежними прогнозами на всіх шарах

(Джерело: https://www.researchgate.net/figure/Feature-Pyramid-Network-FPN-structure_fig2_351830582)

Мережа бере зображення як вхідний сигнал і надає на виході пропорційно масштабовані карти ознак на кількох рівнях. Ці карти ознак містять інформацію про зображення на різних рівнях деталізації. Побудова піраміди включає шляхи знизу вгору та шляхи зверху вниз. У висхідному шляху магістральна мережа, як ResNet, використовується для вилучення ознак із зменшенням рівня просторової роздільної здатності. У міру того, як рівні роздільної здатності зменшуються, семантичне значення карт ознак збільшується, як вказує товщина рамок синього кольору. У

низхідному шляху карти ознак об’єднуються, щоб мати багате семантичне значення та точну просторову інформацію. Завдяки потужності побудови карт ознак із багатим семантичним значенням і високою просторовою точністю, FPN широко використовується в багатьох проблемах комп’ютерного зору.

Основні особливості FPN:

- У FPN використовується спадний шлях, який відновлює роздільну здатність завдяки багатій семантичній інформації.
- З горизонтальними зв’язками додається точніша просторова інформація про об’єкт.
- Спадний шлях і горизонтальні зв’язки підвищують точність на 8 пунктів у наборі даних MS COCO. Для невеликих об’єктів поліпшення збільшується на 12,9 балів.

RetinaNet –одноетапна модель виявлення об’єктів, яка добре працює з щільними та малими об’єктами.

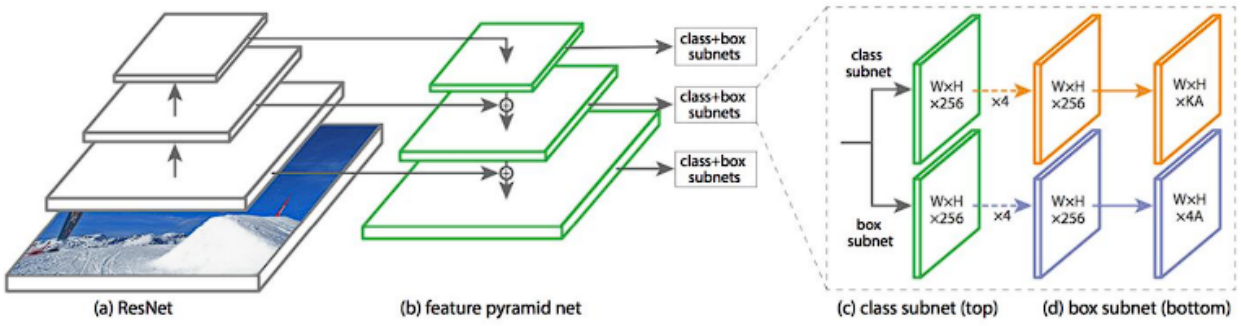


Рисунок 1.2.4 – Архітектура Retina Net

(Джерело: https://www.youtube.com/playlist?list=PLKlxghiZuIM59XVSjuye43qcHXRZLwQNN)

RetinaNet використовує FPN для отримання багаторівневих ознак. Нижчі рівні FPN мають високу роздільну здатність, що дозволяє краще локалізувати об’єкти, а верхні рівні мають більш абстрактні ознаки, що

допомагає класифікувати їх. На кожному рівні FPN RetinaNet застосовує дві підмережі:

1) Підмережа класифікації: Передбачає ймовірність присутності об’єкта в кожному просторовому місці для кожного блоку прив’язки та класу об’єктів. Підмережа регресії: Виконує регресію (прогнозування) координат обмежувальних рамок, які точно визначають положення об’єктів на зображенні.

У RetinaNet, як і в більшості алгоритмів виявлення об’єктів, існує проблема дисбалансу між фоном, який не містить об’єктів, і переднім планом, який містить об’єкти інтересу. Фокальна втрата призначена для присвоєння більшої ваги складним прикладам (фон з частковим об’єктом або з шумною текстурою), які легко класифікувати неправильно, а також зменшення ваги простих прикладів (порожній фон).

Основні особливості RetinaNET:

- Об'єднує дві архітектури FPN + ResNet і завдяки спеціальній функції помилки (focal loss) має більш високу точність
- RetinaNet перевершує по частоті кадрів попередні SSD і FPN.

1.3 Постановка проблеми та формування задач.

Постановка проблеми. Основним обмеженням методів виявлення та класифікації об’єктів є висока ступінь розсіювання світла у воді, що веде до втрати якості, роздільної здатності зображень та первинних кольорів об’єктів, що може ускладнювати їхню класифікацію. Також варто зосередитись на тому, що під водою знижується контрастність об'єктів через розсіювання та поглиблення, ускладнюючи їхнє розпізнавання. Крім того, біологічні об'єкти, такі як водорості та риби, можуть перешкоджати точному визначенню та класифікації цільових об'єктів.

Однак існують потенційні засоби подолання цих обмежень. Розробка високочутливих камер, які можуть пристосовуватися до умов низької видимості, є однією з можливих стратегій. Акустичні методи, які використовують акустичні хвилі для взаємодії з об'єктами, можуть бути менш чутливими до водяних умов. Використання алгоритмів глибокого навчання для аналізу та відновлення зображень може поліпшити розпізнавання об'єктів ускладнених умовах підводного середовища. Також можливе використання мультисенсорних систем, які об'єднують дані від різних типів сенсорів, що дозволяє отримати комплексну інформацію та знизити вплив окремих обмежень кожного типу сенсора.

Крім того, розвиток підводних роботів та їхнє використання в дослідженнях може забезпечити більшу гнучкість та можливості виявлення та класифікації об'єктів під водою. Незважаючи на складність завдань у цій галузі, постійні дослідження та розвиток нових технологій продовжують сприяти подоланню цих викликів і відкривають нові можливості для вивчення та збереження підводного світу.

Для покращення зображення можливе використання алгоритмів, які зменшать шуми, розмиття, підвищать чіткість. До таких алгоритмів можна віднести аналіз плям, детектор кутів Гарріса, детектор Канні.

Детектор Канні вважається одним з найпоширеніших алгоритмів для виявлення контурів на зображеннях. Розроблений у 1986 році Джоном Канні,

він залишається популярним завдяки своїй високій ефективності та точності. Процес алгоритму виявлення контурів Канні можна розглядати як послідовність п’яти етапів:

1) Згладжування (Smoothing): Спосіб згладжування
2) зниження рівня шуму.
3) Розрахунок градієнту (Gradient Calculation): Градієнти яскравості визначаються в зображенні за допомогою оператора Собеля, який виявляє ребра там, де градієнт досягає максимального значення.
4) Приглушення не-максимумів (Non-maximum Suppression) вдосконалює контури шляхом приглушення всіх пікселів, які не є локальними максимумами в напрямку градієнта.
5) Подвійний поріг (Double Thresholding) використовує дві різні порогові значення для визначення «сильних» та «слабких» кордонів.
6) Завершальне визначення межі (Edge Tracking by Hysteresis) включає в себе утримання сильних кордонів, тоді як слабкі кордони зберігаються лише у випадку, якщо вони з’єднуються з сильними.

Перетворення Гарріса, відоме як детектор кутів Гарріса, є важливим інструментом у галузі комп’ютерного зору та обробки зображень. Це важливе знаряддя для виявлення кутів та країв на зображеннях, а також визначення ключових точок, які можуть бути використані для подальших завдань, таких як розпізнавання об’єктів та відстеження руху. Детектор кутів Гарріса базується на принципах детектора кутів Моравека. Останній функціонує за допомогою аналізу локального вікна на зображенні та визначення середніх змін інтенсивності при зсуві цього вікна на невелику відстань в різних напрямках. Цей процес можна уявити у контексті трьох сценаріїв:

1) Пласка ділянка (тобто приблизно постійна за інтенсивністю) призводить лише до невеликої зміни при будь-якому зсуві
2) Перетин з краєм призводить до невеликої зміни при зсуві вздовж краю, але до значної зміни при зсуві перпендикулярно до краю

3) Якщо ділянка містить кут або ізольовану точку, то будь-який зсув призведе до значних змін. Кут можна виявити, знаходячи місце, де мінімальна зміна, спричинена будь-яким зсувом, є великою.

Blob Analysis (аналіз плям) – основний метод машинного зору, що ґрунтується на обстеженні послідовних областей зображення, виявляється важливим інструментом для сценаріїв, де об’єкти піддані перевірці чітко відокремлені від фону. Основні етапи аналізу плям такі:

1) Визначення порогового значення
2) Маркування з’єднаних компонентів
3) Фільтрація та аналіз згустків
4) Вилучення характеристик згустків
5) Відстеження та розпізнавання об’єктів

Аналіз плям є важливою технікою в галузі комп’ютерного зору та відіграє ключову роль у різноманітних сценаріях застосування, зокрема в областях, де важливо визначати та розпізнавати об’єкти на зображеннях або відеозаписах. Поліпшення цього методу сприяє підвищенню його ефективності в умовах складного тла та різноманітних умов освітлення.

Формування задач:

- Зібрати зображення для навчання моделі
- Підготувати та анотувати зображення
- Сформувати датасет
- Виконати процес тренування та перевірки навченої моделі
- Експортувати навчену модель для подальшого використання
- Розробити програмний модуль для розпізнавання об’єктів під водою на основі штучної нейронної мережі
- Виконати завершальне тестування

1.4 Обґрунтування вибору підходів і технологій для проектування та створення СШІ і їх компонентів.

За основу моделі нейронної мережі було взято YOLOv8 - це новітнє сімейство моделей виявлення об'єктів на базі YOLO від Ultralytics, що забезпечують найсучасніші характеристики. У порівнянні з попередніми версіями YOLO модель YOLOv8 працює швидше і точніше , забезпечуючи при цьому єдину структуру для навчання моделей та виконання. Він побудований як єдина платформа для навчання моделей виявлення об'єктів, сегментації екземплярів і класифікації зображень.

Для підготовки і анотування зображень було обрано CVAT (Computer Vision Annotation Tool). CVAT є потужним інструментом для інструкції зображень, розробленим спеціально для комп'ютерного зору. Він надає широкий набір функцій для різних типів анотацій, включаючи bounding boxes, полігони, ключові точки та інші.

При написанні програмного модулю було використано PyCharm. PyCharm – це одне з найпопулярніших інтегрованих середовищ розробки (IDE), спеціально розроблених для Python. Вона надає розширений набір інструментів та функціональності, спеціально створених для ефективної розробки. Підтримує контроль версій, перевірку коду на відповідність стандарту PEP8 та має вбудовану інтеграцію з пакетами Python.

Мовою програмування виступає Python. Вона ідеально підходить для написання нейронних мереж, має велику кількість бібліотек, сучасний мовний синтаксис та можливість використання парадигм програмування. Для роботи з машинним навчанням, нейромережами і штучним інтелектом найчастіше використовується мова Python. Вона може застосовуватися в якості основної мови або для реалізації окремих модулів.

ІІ. Аналітичний розділ. Характеристика штучної нейронної мережі та постановка задачі

2.1 Характеристика об’єкта дослідження.

Це нейронна мережа для об’єктного детектування, яка здатна швидко і точно визначати об’єкти на зображеннях та відео. Модель може обробляти зображення та відео в реальному часі (30 кадрів в секунду або більше). Така швидкість досягається за рахунок того, що YOLO пропускає крок виділення об’єктів на різних шарах зображення, які зазвичай присутні в інших видах архітектури об’єктного детектування. YOLO використовує CNN-архітектуру, яка має декілька блоків під назвою Convolutional Layer, в яких виконуються операції згортки та підсумовування. Модель також включає Fully Connected Layers та Detection Layers, які використовуються для визначення об’єктів на зображенні та виділення їх координат.

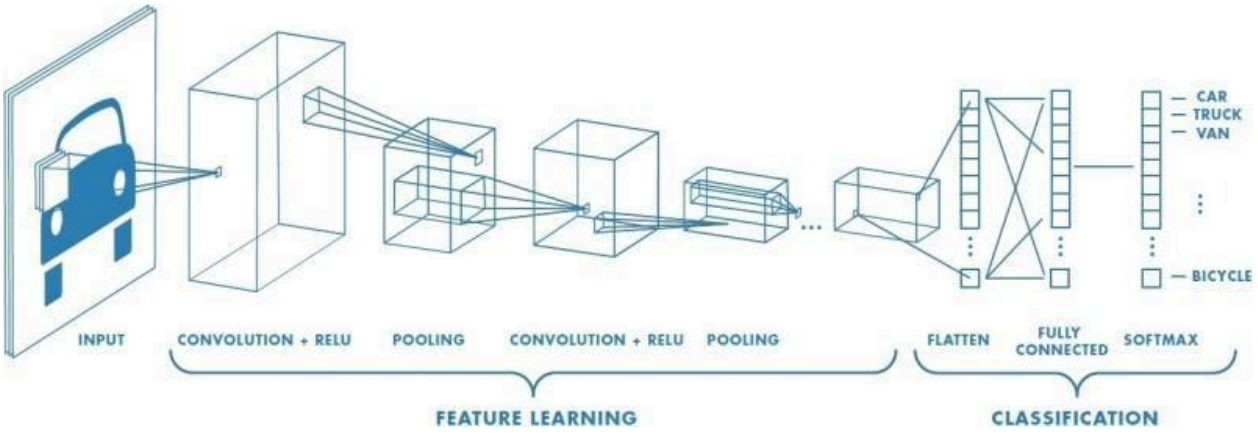


Рис.2.1.1 – архітектура CNN

(Джерело: https://hackaday.com/2024/02/12/understanding-deep-learning-free-mit-press-ebook-for-instructors-and-students/)

Архітектура YOLO складається з декількох блоків, які працюють послідовно та дозволяють здійснювати об’єктне детектування на зображенні або відео в реальному часі.

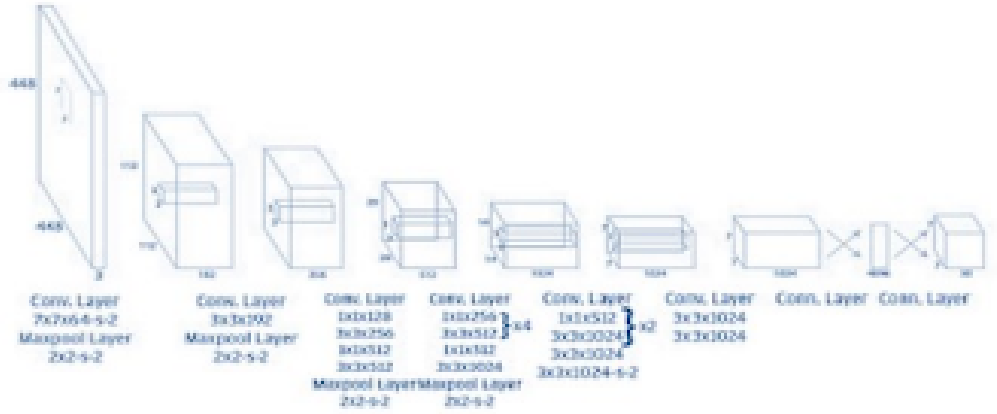


Рис.2.1.2 – архітектура YOLO

(Джерело: https://github.com/Martar189/Lession-Detection-with-YOLO)

Основні блоки архітектури YOLO:

- Блок згорткового шару (Convolutional Layer): призначений для зменшення розміру зображення та вилучення рис об’єктів з нього.
- Блок максимального пулінгу (Max-Pooling Layer): призначений для зменшення розміру зображення та підвищення інваріантності до зміщень.
- Блок згорткового шару + пакетна нормалізація (Convolutional Layer + Batch Normalization): призначений для подальшого вилучення рис об’єктів та стабілізації градієнтів під час навчання.
- Блок згорткового шару + пакетна нормалізація + функція активації (Convolutional Layer + Batch Normalization + Leaky ReLU): призначений для додавання нелінійності та підсилення розпізнавання об’єктів.
- Повнозв’язний шар (Fully Connected Layer): призначений для зведення вихідних значень з попередніх шарів до фінального розміру та отримання вектору ознак.
- Блок векторної розмітки (Vectorization Layer): призначений для перетворення вектора ознак в вектор розмітки, який містить інформацію про розташування та клас об’єкта.

- Блок об’єднання результуючих векторів (Anchors Layer): призначений для об’єднання результуючих векторів розмітки в один вектор, який містить інформацію про всі об’єкти на зображенні.
- Далі йде п’ять повторювальних блоків, кожен з яких складається зі свого власного детектора об’єктів, який працює на основі вихідних даних з передньої частини мережі.

Кожен детектор об’єктів використовує фільтри-якорі для прогнозування координат обмежуючої рамки і ймовірності класів для кожної рамки. Якщо ймовірність перевищує поріг, то рамка пов’язується з даними класу. YOLO використовує функцію втрати суми квадратів помилок, яку легко оптимізувати. Однак ця функція надає однакову вагу завданням класифікації та локалізації.

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathrm{I}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]$$
$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathrm{I}_{ij}^{obj}\left[\left(\sqrt{w}_i-\sqrt{\hat{w}}_i\right)^2+\left(\sqrt{h}_i-\sqrt{\hat{h}}_i\right)^2\right]$$
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathrm{I}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$
$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathrm{I}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2$$
$$+\sum_{i=0}^{S^2}\mathrm{I}_{i}^{obj}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

Рис.2.1.3 – Функція втрати суми квадратів помилок (2.1.1)

де I_{ij}^{obj} - позначає, чи присутній об’єкт у клітинці i, I_{ij}^{obj} - позначає j_{th} обмежувальну рамку, відповідальну за передбачення об'єкта в клітинці i, λ_{coord} і $\lambda_{nocoord}$ - це параметри регуляризації, які необхідні для балансування функції втрат.

Основна ідея YOLO полягає в тому, щоб розділити зображення на сітку, складаючи її з комірок, які накладаються на зображення. Кожна комірка відповідає деякому регіону на зображенні, але вона також може бути причетною до декількох об’єктів. Якщо центр обмежувальної рамки об’єкта знаходиться в цій сітці, то ця сітка відповідає за виявлення цього об’єкта. Кожен об’єкт позначається як bounding box з деякими характеристиками, такими як координати кутів, розміри, клас об’єкта тощо. Також кожна комірка сітки передбачає умовну ймовірність класу об’єкту $P_r(Class_i|Object)$. Ймовірність залежать від комірки сітки, яка містить об’єкт. Можна передбачити лише один набір ймовірностей класу на клітинку сітки, незалежно від кількості блоків сітки.

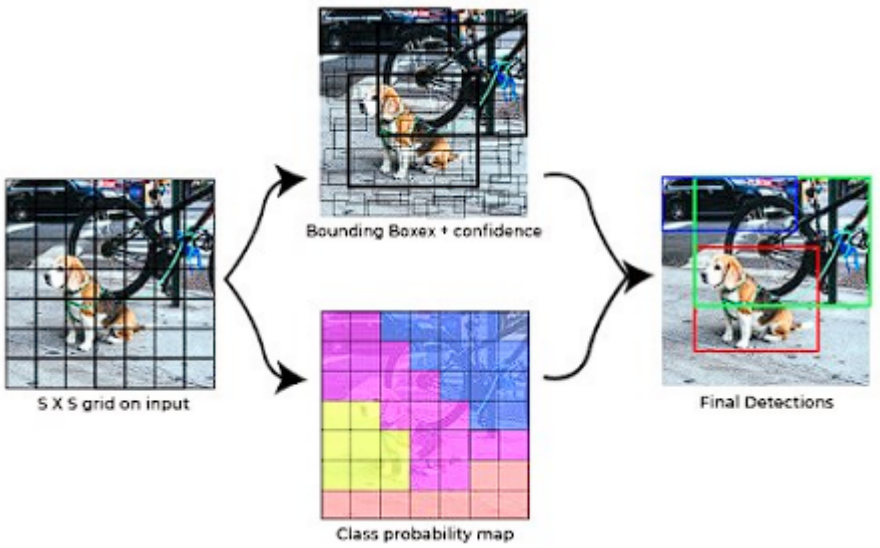


Рис.2.1.4 – Приклад накладання сітки на зображення

(Джерело:

https://www.researchgate.net/figure/Figure-courtesy-38-The-model-perform-detection-as-a-regression-problem-It-divides-the_fig3_344645063)

Оцінку надійності для кожного класу можна визначити наступним чином:

$$P_r(Class_i|Object) * P_r(Object) * IOU_{pred}^{truth} = P_r(Object) * IOU_{pred}^{truth}$$

(2.1.2)

Для визначення bounding box та класів об’єктів, YOLO використовує останній блок CNN, який з’єднує розміри зображення з вимірами сітки та використовує цю інформацію для прогнозування характеристик bounding box та класів об’єктів. Завдяки цій архітектурі YOLO здатна визначати об’єкти на зображеннях та відео в реальному часі, з високою точністю та відносно невисокими обчислювальними витратами. Однак, через те, що YOLO розділяє зображення на сітку комірок, вона може мати деяку схильність до помилок в обробці деяких об’єктів, а також відносно невеликої кількості об’єктів, розташованих дуже близько один до одного.

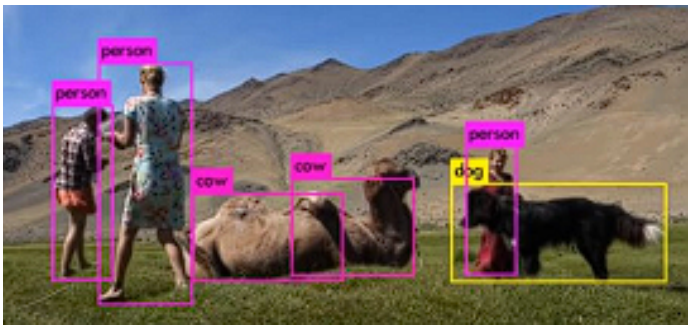


Рис.2.1.5 – Помилки YOLO в обробці ділянок відео зображення

(Джерело: https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e)

Для навчання YOLO використовують функцію втрат, яка обчислює різницю між прогнозованими та дійсними значеннями. Ця функція включає в себе складові, які стимулюють мережу прогнозувати об’єкти різних розмірів та забезпечувати більш точне позиціонування обмежуючих рамок.

Переваги:

- Обробляє кадри зі швидкістю від 45 кадрів/с до 150 кадрів/с.
- Краще знаходить рамки зображення.

Недоліки:

- Досить низький рівень запам’ятовування відокремлених об’єктів.
- Багато помилок локалізації порівняно з Faster R-CNN.

- Важко виявити близькі об’єкти, оскільки кожна сітка може запропонувати лише дві обмежувальні рамки.
- Погано виявляє дрібні предмети.

YOLO є ефективною для задачі об’єктного детектування, оскільки вона дозволяє швидко та точно прогнозувати обмежуючі рамки та класи об’єктів на зображеннях і відео. Вона також має високу швидкість роботи та низькі вимоги до обчислювальних ресурсів порівняно з деякими видами моделей.

2.2 Структура і характеристика СШІ.

Важливим етапом постає вибір певної модифікації YOLOv8, яка включатиме в себе необхідні функції, що необхідні для вирішення сформованих задач. YOLO-World Model є просунутою моделлю, що працює в режимі реального часу Ultralytics YOLOv8-підходом до вирішення завдань з виявлення відкритих словників. Ця інновація дозволяє знайти будь-який об'єкт на зображенні на основі описових текстів. Завдяки значному зниженню обчислювальних вимог за збереження конкурентоспроможної продуктивності, YOLO-World стає універсальним інструментом для безлічі додатків, заснованих на зорі.

YOLO-World перевершує існуючі детектори відкритих словників, включаючи MDETR та серію GLIP, за швидкістю та ефективністю у стандартних бенчмарках, демонструючи перевагу YOLOv8 на одному графічному процесорі NVIDIA V100. Нижче наведено порівняльну таблицю, що містить всі модифікації.

Таблиця 2.2.1 – Модифікації YOLOv8-World

Тип моделі	Підготовлені ваги	Підтримувані задачі	Заключення	Валідація	Тренування	Експорт
YOLOv8s-world	yolov8s-world.pt	Виявлення об’єктів	+	+	+	-
YOLOv8s-worldv2	yolov8s-worldv2.pt	Виявлення об’єктів	+	+	+	+
YOLOv8m-world	yolov8m-world.pt	Виявлення об’єктів	+	+	+	-
YOLOv8m-worldv2	yolov8m-worldv2.pt	Виявлення об’єктів	+	+	+	+
YOLOv8l-world	yolov8l-world.pt	Виявлення об’єктів	+	+	+	-
YOLOv8l-worldv2	yolov8l-worldv2.pt	Виявлення об’єктів	+	+	+	+
YOLOv8x-world	yolov8x-world.pt	Виявлення об’єктів	+	+	+	-
YOLOv8x-worldv2	yolov8x-worldv2.pt	Виявлення об’єктів	+	+	+	+

При аналізі цих показників орієнтуватись необхідно на worldv2, оскільки ці моделі підтримують детерміноване навчання та можуть бути експортовані. Із чотирьох моделей, які залишились, відсіювання зайвих спиратиметься на показники продуктивності та розмір моделі.

Тип моделі	mAP	mAP50	mAP75
yolov8s-world	37.4	52.0	40.6
yolov8s-worldv2	37.7	52.2	41.0
yolov8m-world	42.0	57.0	45.6
yolov8m-worldv2	43.0	58.4	46.8
yolov8l-world	45.7	61.3	49.8
yolov8l-worldv2	45.8	61.3	49.8
yolov8x-world	47.0	63.0	51.2
yolov8x-worldv2	47.1	62.8	51.4

Рис.2.2.1 – Передача нуля на наборі даних COCO

(Джерело: https://docs.ultralytics.com/ru/models/yolo-world/#zero-shot-transfer-on-coco-dataset)

Виходячи з таблиці та рисунка обираємо модель, яка підходить найбільше – yolov8x-worldv2.

III. Конструктивний розділ. Розроблення проектних рішень

3.1 Моделювання та проектування датасету

При відсутності необхідних датасетів, потрібно їх створювати з нуля. Нижче буде описано всі етапи, які необхідні для створення якісного датасету. Дані маніпуляції можна розділити на кілька кроків:

- Розмірність
- Реальні умови
- Формат та анотування
- Поділ на частини
- Витік даних

Крок 1 – розмірність. Розмір самих зображень, що знаходяться в датасеті, не такий важливий, перед подачею моделі на навчання дані нормалізуються, щоб відповідати розміру сітки (у нашому випадку це 640 на 640 пікселів). Розмір вхідного шару нейромережі - розмір зображень, з якими працюватиме сітка. При формуванні датасета важливо мати на увазі розмірність сітки, з якою працюємо і максимально близько підганяти розмір зображень до розміру моделі, щоб уникнути небажаного стиснення зображення. Для оптимізації розміру зображень (наближення до розміру сітки) було використано влаштований в переглядач зображень Windows інструмент.

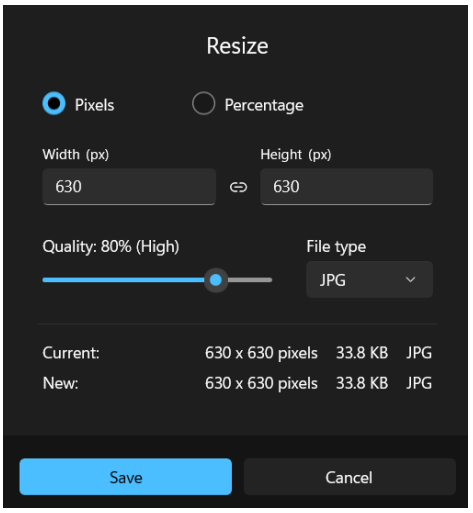


Рис.3.1.1 – оптимізація розміру зображення (Resize)

(Джерело: розроблено автором)

Крок 2 – реальні умови. Головне правило будь-якого датасета - зображення мають бути максимально наближені до реальних умов, у яких працюватиме модель нейромережі. Цей крок часто становить досить велику складність, оскільки доволі часто немає можливості зібрати реальні фото об'єктів до того, як проект буде запущено. Однак його не можна ігнорувати, адже від того, наскільки дані в датасеті близькі до реальних даних, залежить якість розпізнавання.

Крок 3 – формат та анотування. На даному кроці важливо звести всі зображення до одного формату. В даному випадку використовується популярний формат зображень «.jpg». Зведення до одного формату відбувалось влаштованими інструментами Windows. Далі потрібно анотувати всі зображення, тобто виділити зони інтересу на кожному зображенні (полігон, бокс, еліпс та інші). Для анотування використовувався онлайн-додаток CVAT.

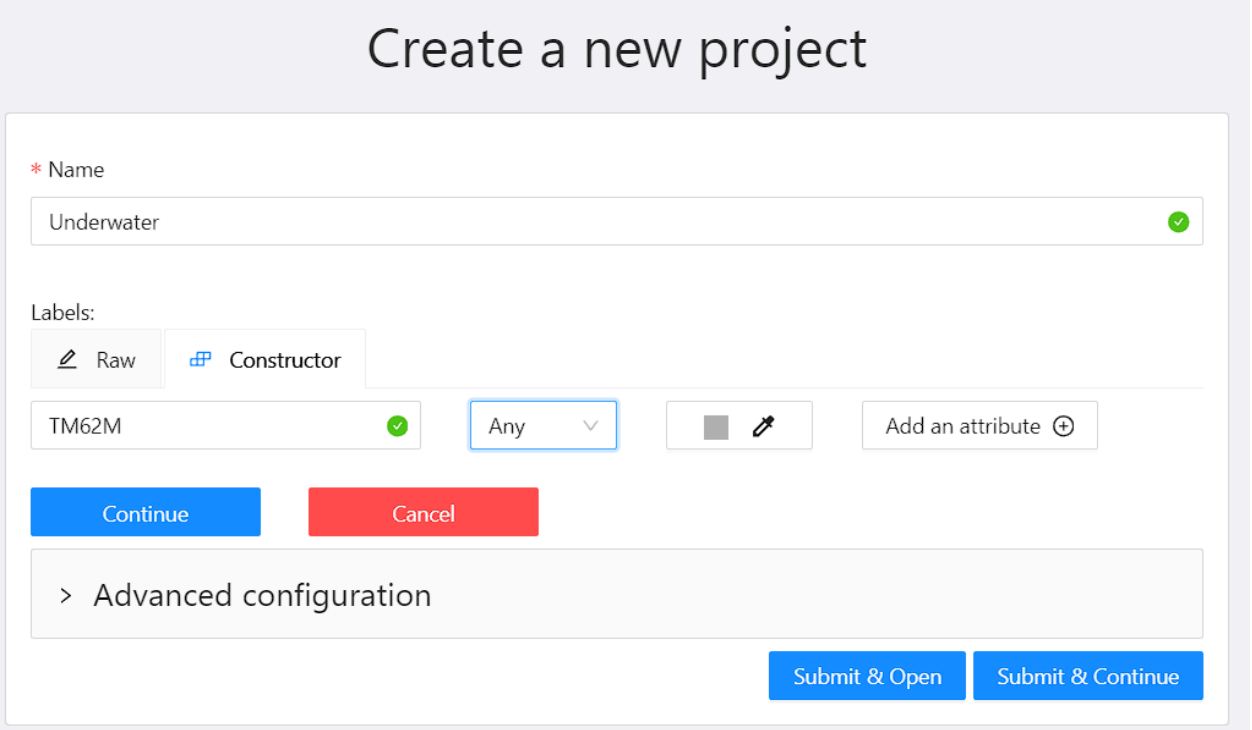


Рис.3.1.2 – створення проекту і атрибутів

(Джерело: розроблено автором)

Потрібно додати всі необхідні атрибути (відповідно до кількості розпізнаваних типів об’єктів), після чого натиснути «Submit & Open».

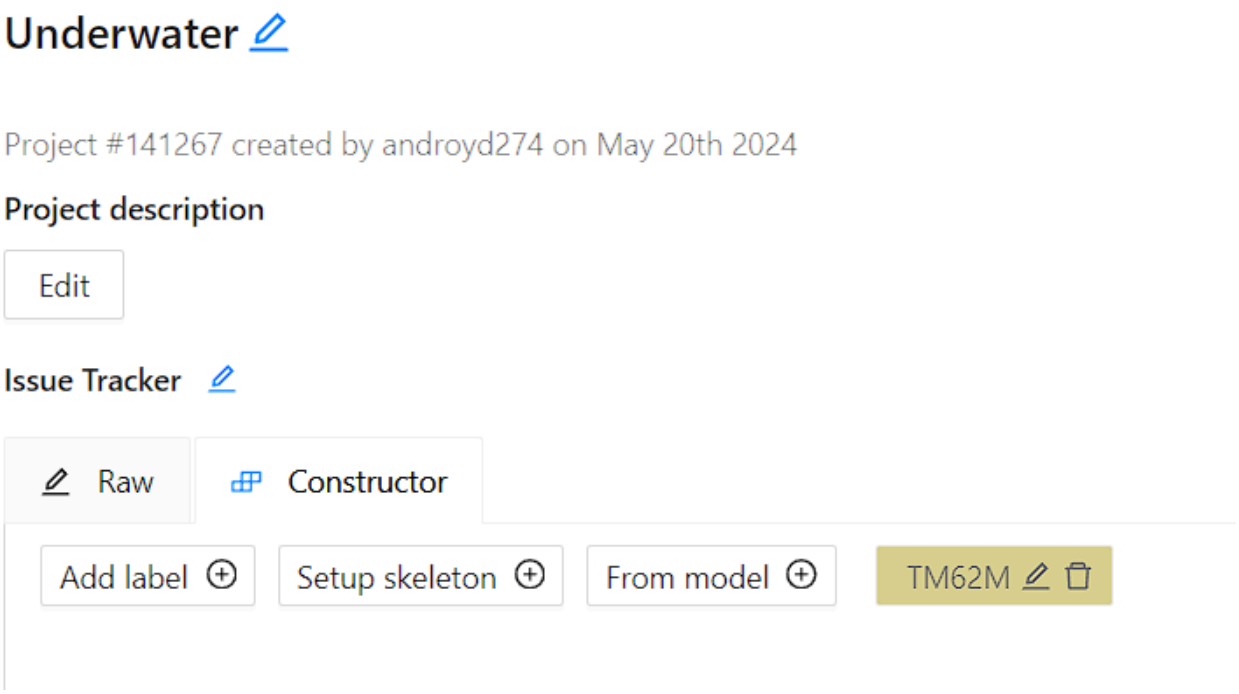


Рис.3.1.3 – створений проект з атрибутом (класом)

(Джерело: розроблено автором)

Створюємо задачу, обираємо підтекст – тренування або валідування (в залежності від етапу) та довантажуємо зображення для подальшого анотування.

Рис.3.1.4 – створення задачі

(Джерело: розроблено автором)

Переходимо в задачі, щоб розпочати анотування.

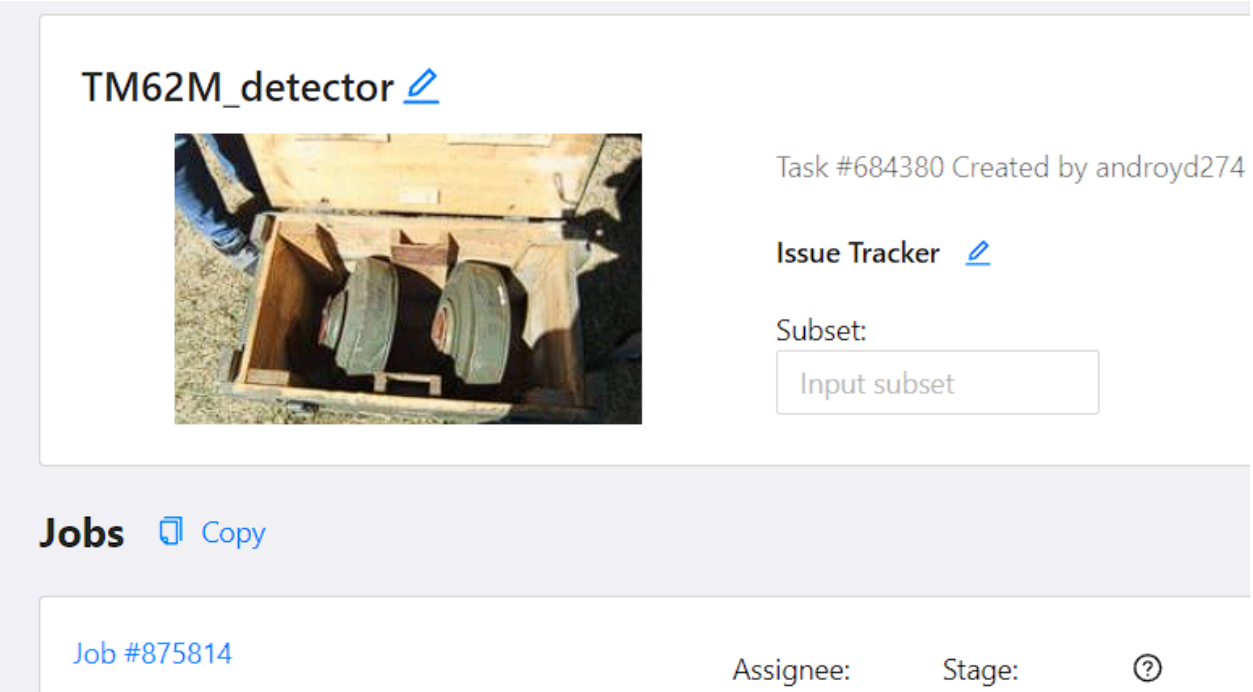


Рис.3.1.5 – перехід до анотування

(Джерело: розроблено автором)

Обираємо потрібний інструмент, перевіряємо атрибут і вмикаємо метод з двома точками.

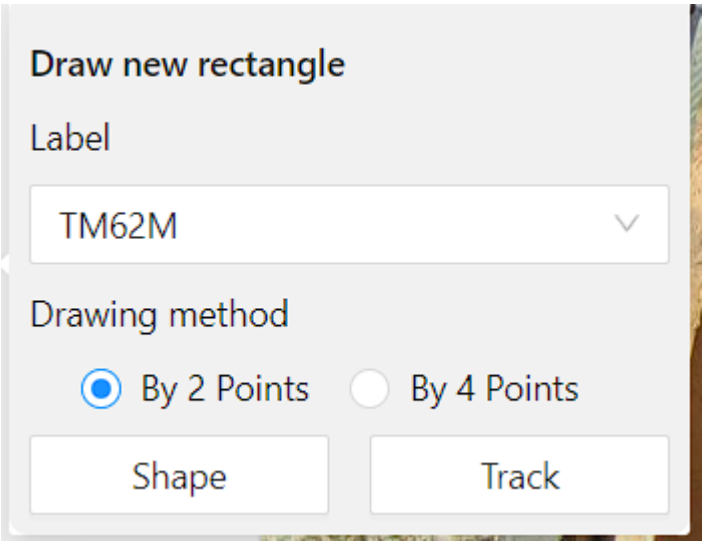


Рис.3.1.6 – вибір інструменту та методу

(Джерело: розроблено автором)

Розпочинаємо анотування. Використовуємо для розмітки прямокутники (rectangle shape).

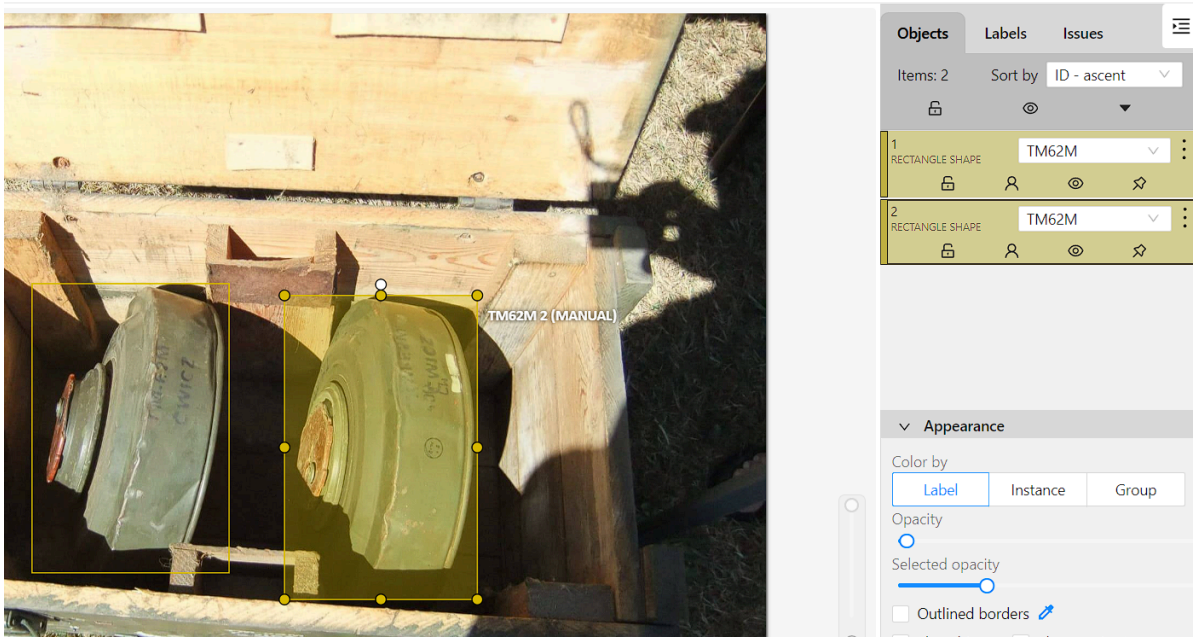


Рис.3.1.7 – анотування

(Джерело: розроблено автором)

Це доволі тривалий процес, оскільки кожне зображення необхідно анотувати вручну. Такий алгоритм повторюється для кожного атрибуту (класу).

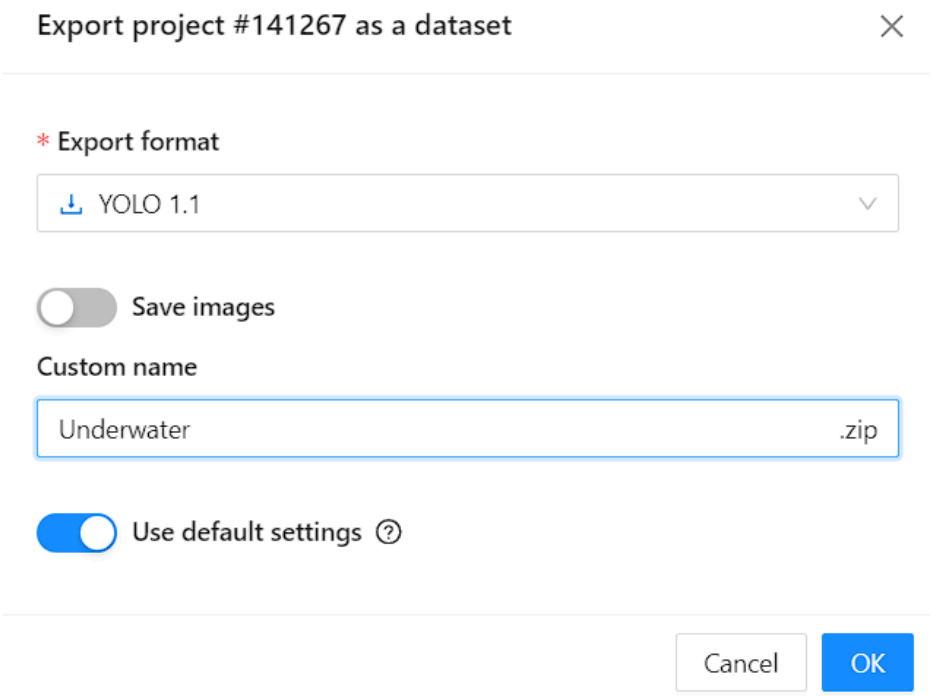


Рис.3.1.8 – експорт анотацій

(Джерело: розроблено автором)

Після завершення, проект з анотуванням було експортовано в підтримуваний YOLO формат. Загалом було анотовано 5 класів (160 зображень), а саме:

- TM62M (протитанкова міна)
- 9N210-235 (бойовий елемент від касетних реактивних ракет 9M55K Smerch та 9M27K Uragan)
- FAB500M54 (фугасна авіаційна бомба)
- FAB500M62 (фугасна авіаційна бомба)
- SeaMine (морська міна)

Усі ці вибухонебезпечні боєприпаси було знайдено в водоймах України, саме тому їх зображення використані для побудови датасету.

Крок 4 – поділ на частини. Далі необхідно розбити датасет на частини Train та Validate, інакше навчання нейромережі не вдасться. Зазвичай, при

поділі користуються таким правилом: 70-80% зображень йдуть у Train, 20-30% - у Validate.

Рис.3.1.9 – поділ на частини

(Джерело: розроблено автором)

Train - те, на чому навчається нейромережа. Після закінчення епохи тренування нейромережа повинна перевірити, чи правильно вона вивчила, як виглядає об'єкт, для чого вона звертається до сета Validate. Використовуючи знання, отримані за допомогою Train, нейромережа намагається вгадати який об'єкт знаходиться на зображенні з Validate. Якщо припущення нейромережі виявилося неправильним, робиться аналіз помилки для запобігання такій помилці в наступних ітераціях.

Без сета Validate неможливо навчити модель. Цей спосіб часто застосовується в системах машинного навчання, працює досить добре і є певним стандартом роботи з датасетом.

Крок 5 – витік даних. Якщо зображення, які на 90% ідентичні (розмір, фон, положення об’єкта), не можна розміщати одразу в навчальний та валідаційний набір. Інакше нейромережа не навчиться, а запам’ятає інформацію. Хоч вона і безпомилково визначить такий об’єкт, але коли стикнеться з ним в іншому ракурсі – не зможе його вірно визначити.

Рис.3.1.10 – приклад поділу

(Джерело: розроблено автором)

Важливо зробити так, щоб у сеті Validate не було картинок, які є у Train. Щоб уникнути витоку даних, можна використовувати скрипти та бібліотеки, які видаляють дублікати. Можна налаштувати поріг видалення – наприклад, видаляти ідентичні картинки або схожі на 90% картинки. Чим більше дублікатів та схожих зображень буде видалено, тим краще для нейромережі.

3.2 Розроблення програмного модулю

Розробку програмного модулю можна розділити на кілька частин – написання програми для навчання нейромережі, конфігураційний файл, програма для перевірки вірності шляхів анотацій та зображень, сам програмний модуль для розпізнавання.

Для написання коду (на мові програмування Python) було використано середовище розробки PyCharm Community Edition. Створимо проект на Python з використанням інтерпретатора venv.

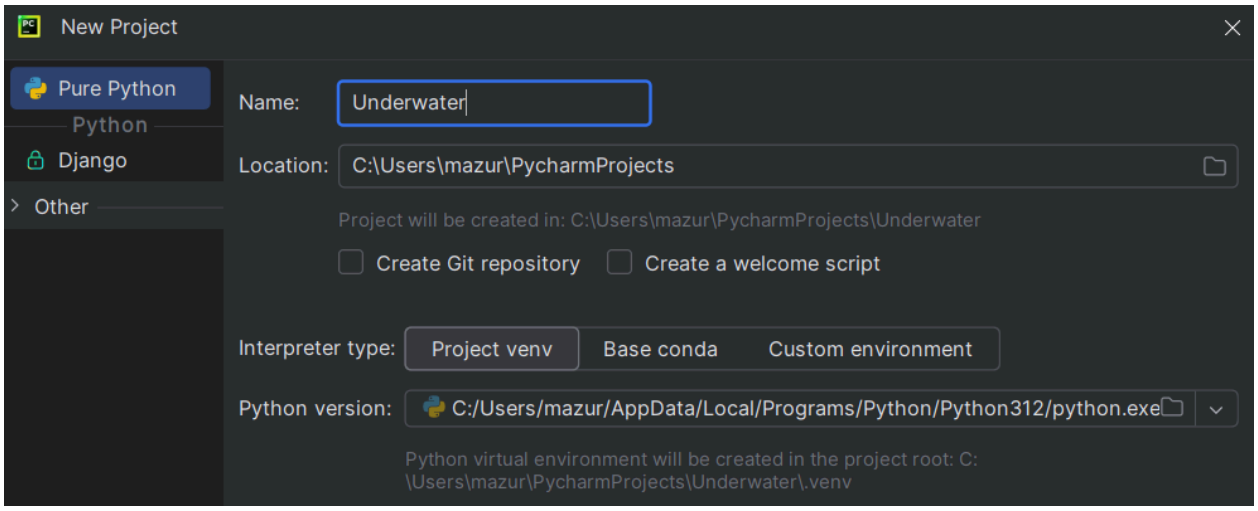


Рис.3.2.1 – створення нового проекту

Після цього переходимо в термінал для створення віртуального середовища та його активування.

```
(.venv) PS C:\Users\mazur\PycharmProjects\Underwater> python -m venv venv
(.venv) PS C:\Users\mazur\PycharmProjects\Underwater> venv\Scripts\activate
(venv) PS C:\Users\mazur\PycharmProjects\Underwater>
```

Рис.3.2.2 – активація віртуального середовища

Встановимо одразу всі необхідні бібліотеки через команду pip install, щоб потім до цього не повертатись:

- Ultralytics - Надає інструменти для використання і тренування моделей YOLO
- opencv-python (cv2) – Використовується для завантаження, відображення, обробки зображень та відео
- matplotlib - Бібліотека для створення статичних, анімованих та інтерактивних візуалізацій у Python
- torch - Потрібна для автоматичної обробки градієнтів, побудови моделей та виконання обчислень на GPU
- os - Використовується для роботи з файловою системою, перевірки існування файлів, створення каталогів та маніпулювання шляхами до файлів
- yaml - Використовується для читання та запису конфігураційних файлів, таких як data.yaml, який містить шляхи до даних та параметри моделі
- tkinter - Використовується для створення додатків з графічним інтерфейсом для взаємодії з користувачем
- sqlite3 - Використовується для збереження та організації результатів обробки зображень, збереження даних про розпізнані об'єкти
- threading - Використовується для виконання декількох потоків одночасно, що дозволяє виконувати декілька завдань паралельно, а саме, захоплення та обробку відеопотоку в реальному часі

Створимо файл train_yolov8.py для тренування моделі, де вкажемо всі необхідні параметри.

```python
# Імпорт необхідних бібліотек
from ultralytics import YOLO
import os
import torch

# Перевірка доступності GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Шлях до датасету
data_path = 'C:/Users/mazur/PycharmProjects/Underwater/dataset/data.yaml'

# Ініціалізація моделі YOLOv8x
model = YOLO('C:/Users/mazur/PycharmProjects/Underwater/yolov8x-worldv2.pt')

# Тренування моделі
model.train(
    data=data_path,
    epochs=50,
    imgsz=640,
    batch=8
```

Рис.3.2.3 – частина коду train_yolov8.py

Нижче наведено та описано алгоритм програми для тренування.

Алгоритм:

1) Імпортувати необхідні бібліотеки (ultralytics, os, torch).
2) Визначити пристрій для тренування (GPU або CPU).
3) Вказати шлях до конфігураційного файлу датасету.
4) Завантажити попередньо натреновану модель YOLOv8x.
5) Налаштувати параметри тренування моделі (шлях до датасету, кількість епох, розмір зображень, розмір пакету, кількість робітників, пристрій для тренування, ім'я для збереження результатів).
6) Запустити тренування моделі.
7) Зберегти натреновану модель у файл.
8) Вивести повідомлення про успішне збереження моделі.

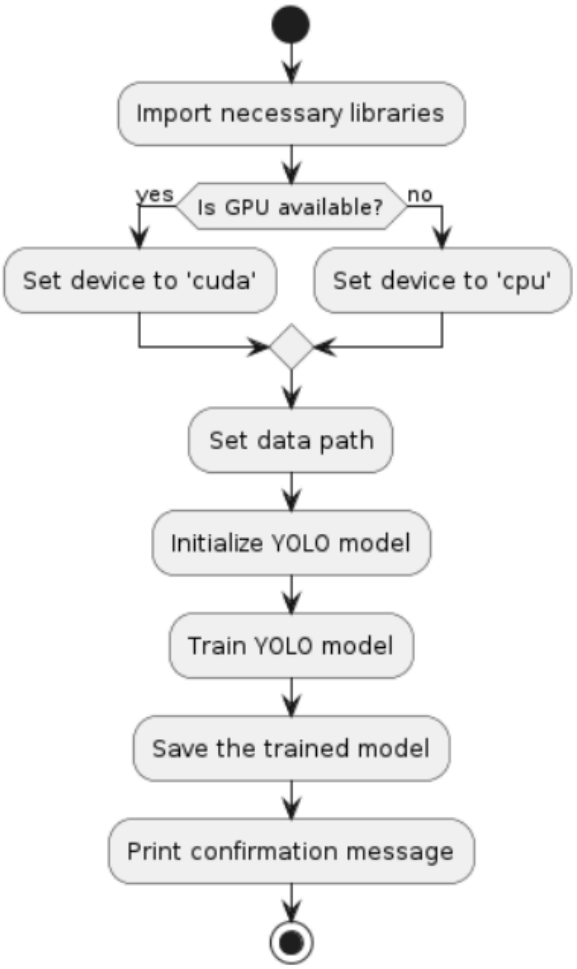


Рис.3.2.4 – алгоритм роботи

Далі потрібно створити конфігураційний файл data.yaml, в якому вказуються шляхи до зображень та анотацій, а також кількість класів у датасеті і їх назви в порядку використання.

```yaml
train: C:/Users/mazur/PycharmProjects/Underwater/dataset/images/train
val: C:/Users/mazur/PycharmProjects/Underwater/dataset/images/val

labels:
  train: C:/Users/mazur/PycharmProjects/Underwater/dataset/labels/train
  val: C:/Users/mazur/PycharmProjects/Underwater/dataset/labels/val

nc: 5
names: ['9N210-235', 'FAB500m54', 'FAB500m62', 'SeaMine', 'TM62M']
```

Рис.3.2.5 – data.yaml

Датасет має наступну структуру, яка не має порушуватись.

Рис.3.2.6 – структура датасету

В каталозі з датасетом є поділ на дві основні папки – зображення та мітки, кожна з яких має папки тренування та валідації. Images містить зображення, а labels – анотовані файли в форматі «.txt».

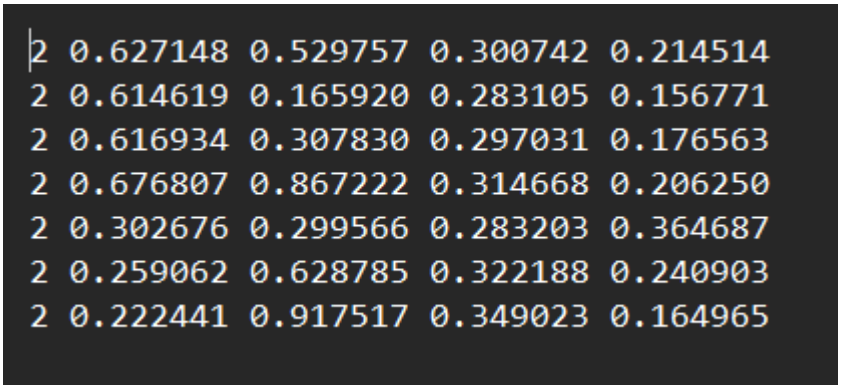


Рис.3.2.7 – один з файлів анотації

Кожне зображення має свій файл анотації з такою ж назвою (відрізняються лише формат файла). Анотація складається з чотирьох точок, які мають діапазон від 0 до 1. Чим більше розмічено об’єктів на зображенні, тим більше рядків з координатами в файлі.

Тепер створимо файл check-annotation.py для перевірки шляхів анотацій.

```
# Імпорт необхідних бібліотек
import os
import yaml

# Шлях до файлу data.yaml
data_yaml_path = 'C:/Users/mazur/PycharmProjects/Underwater/dataset/data.yaml'

# Перевірка існування файлів зображень та анотацій
with open(data_yaml_path, 'r') as file:
    data = yaml.safe_load(file)

# Отримання шляхів до зображень та анотацій
train_images_path = data['train']
val_images_path = data['val']
train_labels_path = data['labels']['train']
```

Рис.3.2.8 – частина коду check-annotation.py

Нижче наведено алгоритм роботи програми для перевірки.

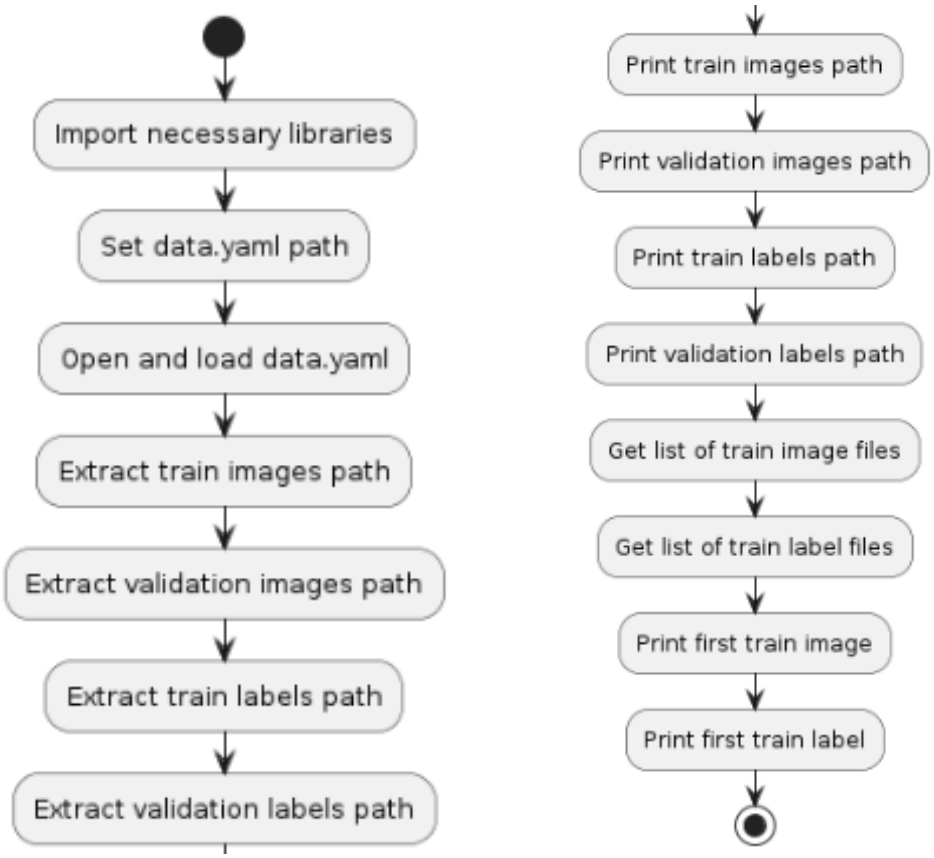


Рис.3.2.9-3.2.10 – алгоритм програми для перевірки шляхів

Як це працює:

1) Користувач вказує шлях до файлу data.yaml
2) Код відкриває і зчитує вміст файлу data.yaml
3) З файлу data.yaml витягуються шляхи до тренувальних та валідаційних зображень і анотацій
4) Виводяться ці шляхи для перевірки правильності
5) Код зчитує список файлів з тренувальних зображень і анотацій
6) Виводяться назви перших файлів з цих списків, що дозволяє перевірити наявність та правильність файлів.

3.3 Проектування забезпечувальних підсистем. Реалізація системи

Перейдемо до написання головної програми, яка буде виконувати усі необхідні функції для розпізнавання.

Створимо main.py, який буде складатись з двох основних класів, що містять в собі ряд функцій.

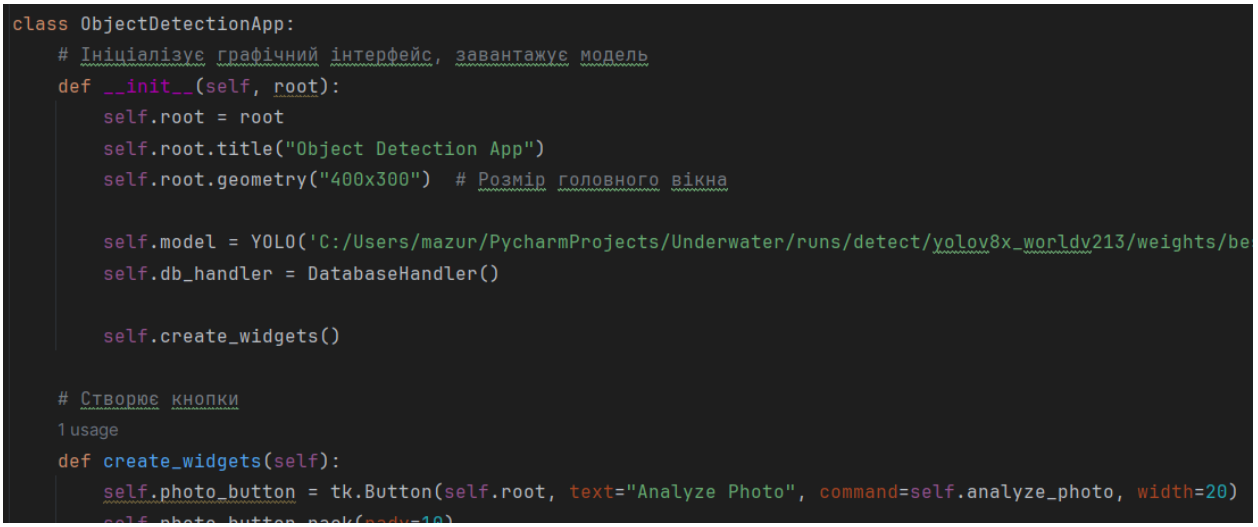
```
# Імпорт необхідних бібліотек
import tkinter as tk
from tkinter import filedialog
from ultralytics import YOLO
import cv2
import sqlite3
import threading

# Клас баз даних
2 usages
class DatabaseHandler:
    # Ініціалізує об'єкт і викликає setup_database
    def __init__(self, db_name='detection_results.db'):
        self.db_name = db_name
        self.setup_database()

    # Створює таблицю results
    1 usage
    def setup_database(self):
        conn = sqlite3.connect(self.db_name)
```

3.3.1 – бібліотеки і частина класу баз даних main.py

Клас баз даних ініціалізує об’єкти, генерує таблиці в базі даних з можливістю додавати нові стовпці без перезапису існуючих даних та зберігає результати аналізу.



```
class ObjectDetectionApp:
    # Ініціалізує графічний інтерфейс, завантажує модель
    def __init__(self, root):
        self.root = root
        self.root.title("Object Detection App")
        self.root.geometry("400x300")  # Розмір головного вікна

        self.model = YOLO('C:/Users/mazur/PycharmProjects/Underwater/runs/detect/yolov8x_worldv213/weights/be
        self.db_handler = DatabaseHandler()

        self.create_widgets()

    # Створює кнопки
    1 usage
    def create_widgets(self):
        self.photo_button = tk.Button(self.root, text="Analyze Photo", command=self.analyze_photo, width=20)
```

Рис.3.3.2 – частина класу розпізнавання main.py

Клас розпізнавання реалізує створення вікна з інтерфейсом та кнопками, містить функції для вибору джерела, з якого відбудеться розпізнавання, має можливість примусового припинення розпізнавання об’єктів.

Пояснення роботи програми:

1) Імпорт необхідних бібліотек
    a. tkinter: Бібліотека для створення графічного інтерфейсу
    b. filedialog: Діалог для вибору файлів
    c. YOLO: Імпорт з бібліотеки ultralytics для роботи з моделлю YOLO
    d. cv2: Інтерфейс OpenCV для обробки зображень та відео
    e. sqlite3: Бібліотека для роботи з базою даних SQLite
    f. threading: Бібліотека для багатопоточності.

2) Клас DatabaseHandler

    a. __init__: Ініціалізує об'єкт і викликає setup_database для налаштування бази даних

    b. setup_database: Створює таблицю results, якщо вона не існує, і додає стовпець object_names, якщо він відсутній

    c. save_to_database: Зберігає результати аналізу в базу даних.

3) Клас ObjectDetectionApp

    a. __init__: Ініціалізує графічний інтерфейс, завантажує модель YOLO і створює об'єкт для роботи з базою даних

    b. create_widgets: Створює кнопки для аналізу фото, відео та потокового відео, а також для зупинки потокового відео

    c. get_object_names: Витягує імена об'єктів з результатів аналізу

    d. analyze_photo: Відкриває діалог для вибору фото, аналізує його, показує результат і зберігає в базу даних

    e. analyze_video: Відкриває діалог для вибору відео, аналізує кожен кадр, показує результати і зберігає в базу даних

    f. analyze_stream: Захоплює відеопотік з веб-камери, аналізує кожен кадр, показує результати і зберігає в базу даних

    g. stop_stream: Зупиняє потокове відео.

4) Основний блок

    a. Ініціалізує графічний інтерфейс і запускає основний цикл програмного модулю.

Нижче представлено графічно алгоритм програми.

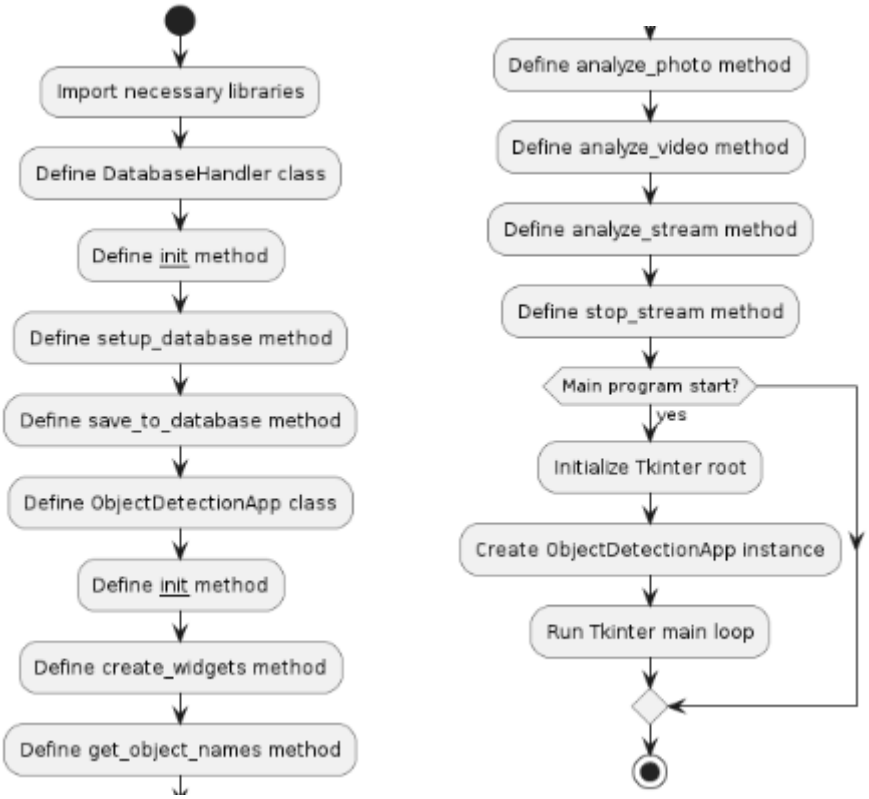


Рис.3.3.3-3.3.4 – алгоритм роботи main.py

3.4 Тестування системи

Початок запуску навчання виконується за допомогою запуску в консолі команди python train_yolov8.py.

```
(venv) PS C:\Users\mazur\PycharmProjects\Underwater> python train_yolov8.py
New https://pypi.org/project/ultralytics/8.2.20 available 😃 Update with 'pip install -U ultralytics'
engine\trainer: task=detect, mode=train, model=C:\Users\mazur\PycharmProjects\Underwater\yolov8x-worldv2.pt,
s=50, time=None, patience=100, batch=8, imgsz=640, save=True, save_period=-1, cache=False, device=cpu, worker
rue, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False, cl
eze=None, multi_scale=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=Fal
=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnostic_
ve_frames=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, show_box
se, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, moment
warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015,
, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0, auto_augment=randa
 save_dir=runs\detect\yolov8x_worldv214
Overriding model.yaml nc=80 with nc=5
```

Рис.3.4.1 – запуск навчання

Після запуску починається навчання з такими параметрами: 50 епох, розмір сітки 640*640, розмір пакету 8, працівників 4, пристрій – GPU при можливості.

```
train: Scanning C:\Users\mazur\PycharmProjects\Underwater\dataset\labels\train.cache... 130 images, 0 backgrounds, 0 corrupt: 100%|██████████| 130/130
val: Scanning C:\Users\mazur\PycharmProjects\Underwater\dataset\labels\val.cache... 30 images, 0 backgrounds, 0 corrupt: 100%|██████████| 30/30 [00:00<
Plotting labels to runs\detect\yolov8x_worldv214\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.001111, momentum=0.9) with parameter groups 104 weight(decay=0.0), 115 weight(decay=0.0005), 121 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\yolov8x_worldv214
Starting training for 50 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/50         0G      1.004      3.417      1.348         30        640:  41%|████      | 7/17 [02:54<04:14, 25.46s/it]
```

Рис.3.4.2 – навчання

Даний процес може зайняти велику кількість часу, залежно від налаштовуваних параметрів, потужності системи, розміру моделі та датасету.

Після навчання модель збережеться в каталозі проекту разом з графіками по результатах навчання.

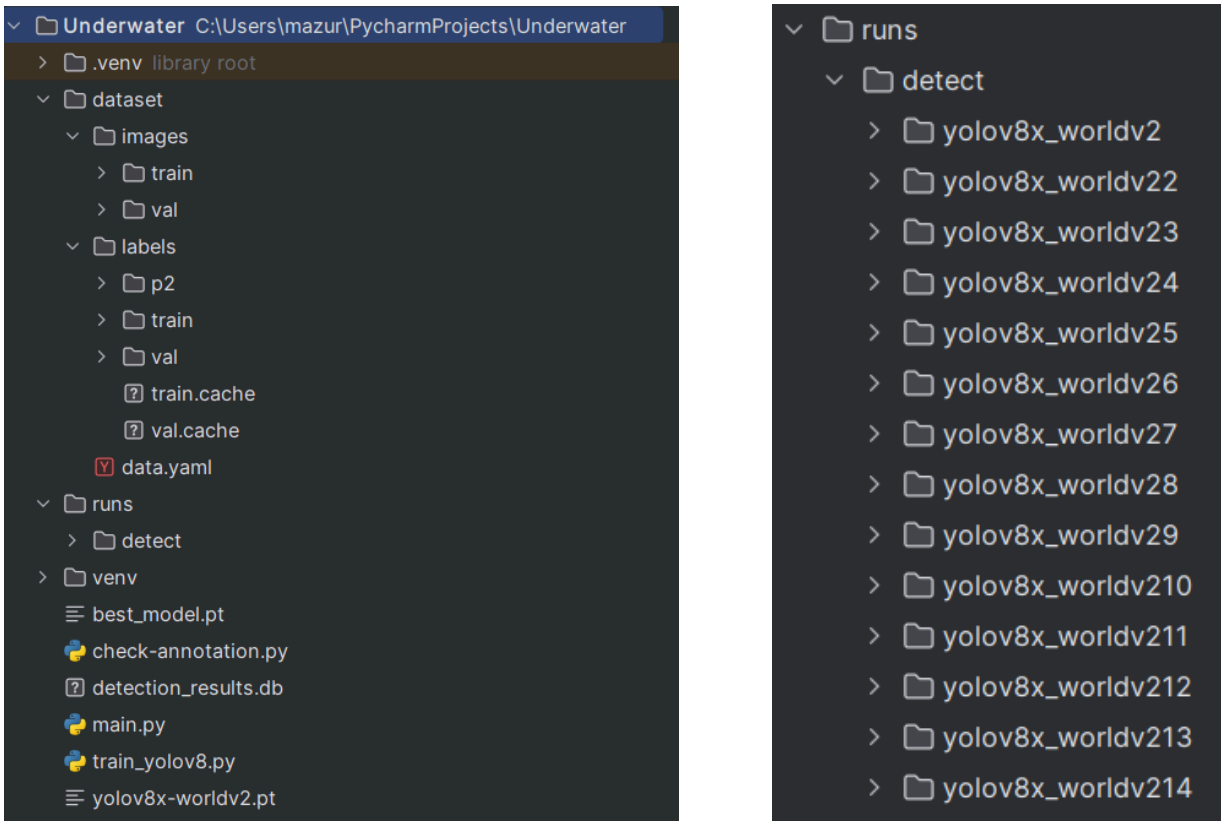


Рис.3.4.3-3.4.4 – структура проекту

Параметри для навчання моделі підбирались неодноразово, про що свідчить кількість спроб (14) навчити нейромережу (див. Рис.3.4.4).

Перейдемо до графічних результатів навчання, після чого протестуємо програму.

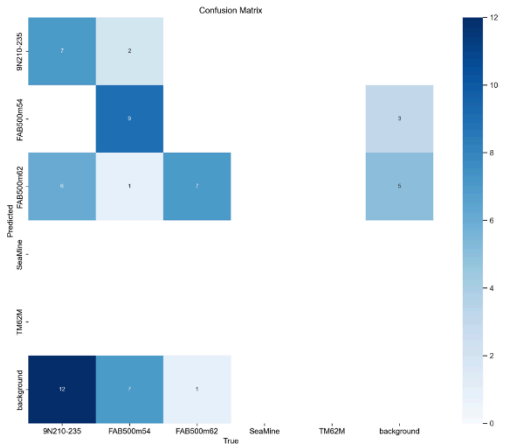


Рис.3.4.5 – Матриця плутанини

Матриця плутанини, яка показує, як часто модель плутає різні класи між собою. Відображає кількість правильних і неправильних класифікацій для кожного класу.

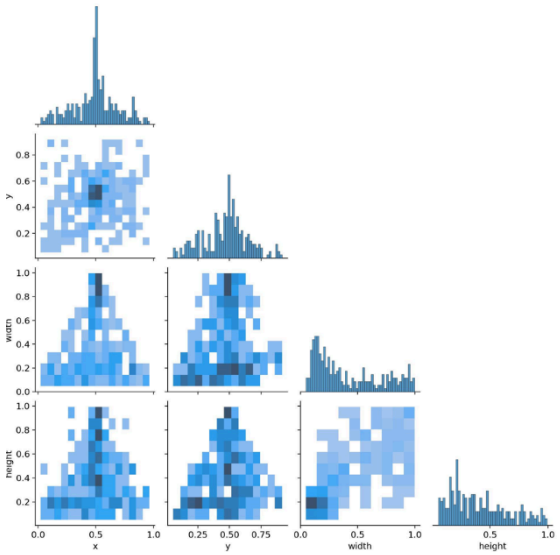


Рис.3.4.6 – корелограма міток

Корелограма міток, що показує кореляцію між різними класами. Це може бути кореляція спільної появи об'єктів різних класів на одному зображенні.

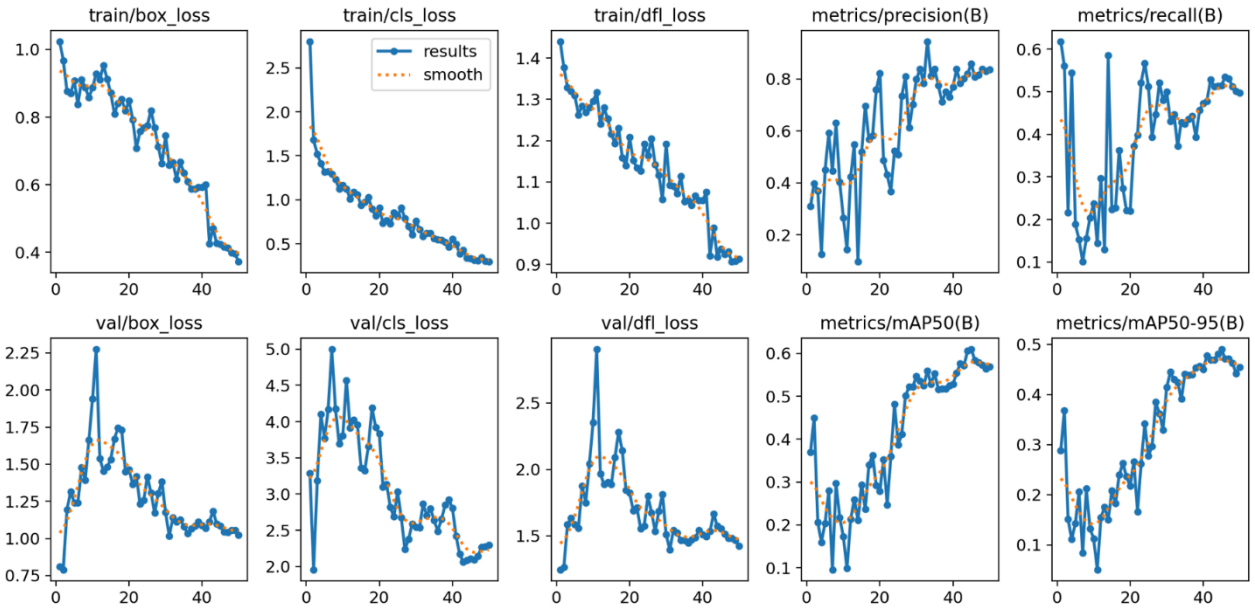


Рис.3.4.7 – Результати

Графік, який показує загальні результати навчання, такі як втрата (loss) та метрики точності для тренувальних та валідаційних даних протягом усіх епох.

Перейдемо до реального тестування працездатності та функцій програмного модулю.

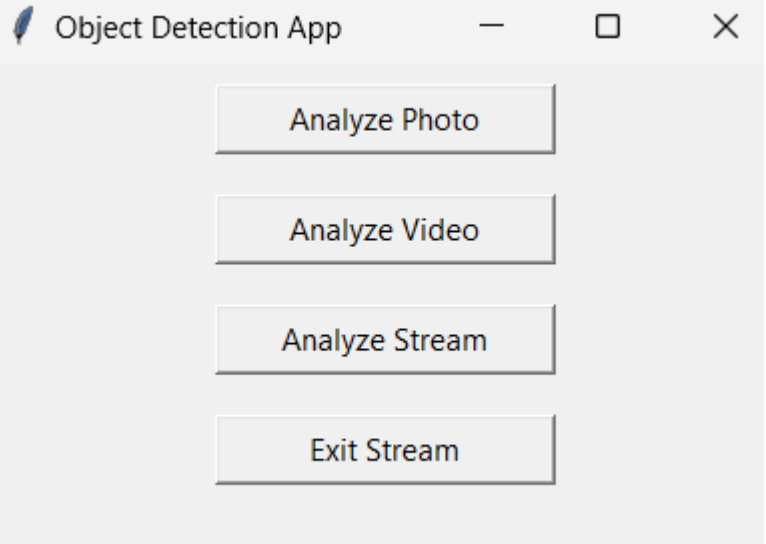


Рис.3.4.8 – головне вікно програми

Функції – розпізнавання з фото, розпізнавання з відео, розпізнавання в реальному часі з камери, примусове завершення розпізнавання.

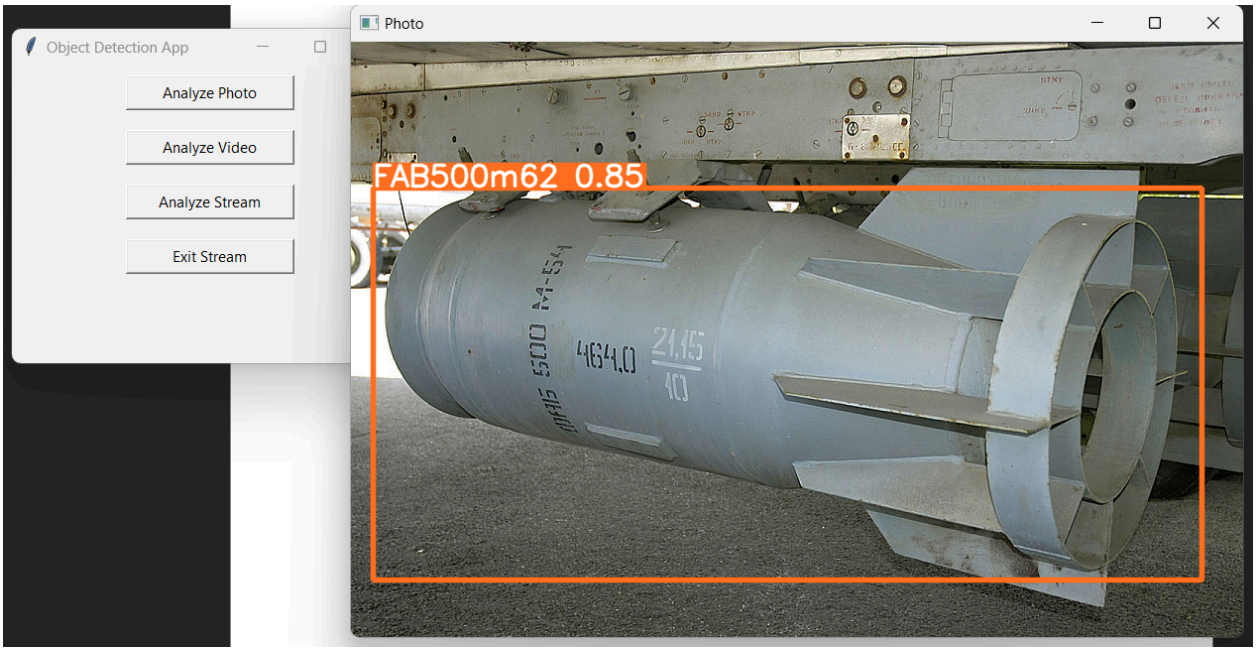


Рис.3.4.9 – аналіз з фото

Рис.3.4.10 – аналіз з відео

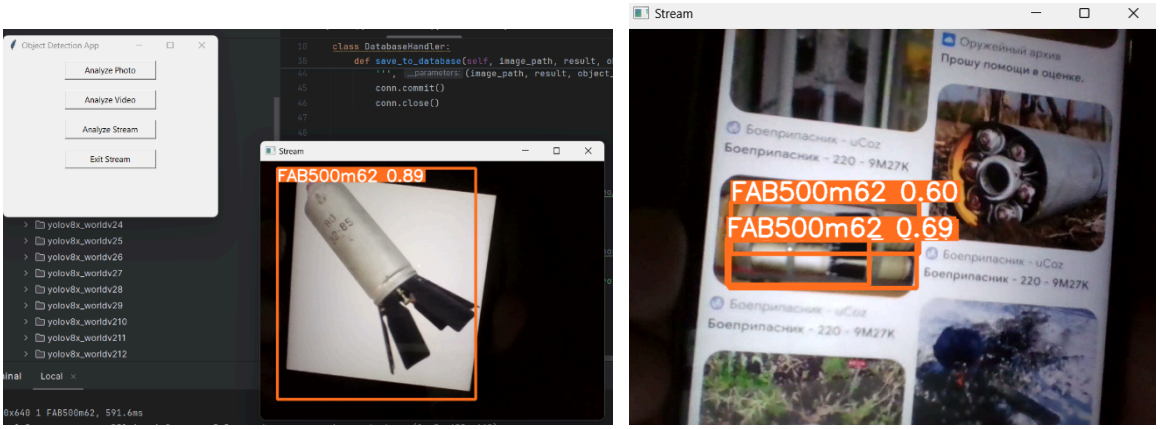


Рис.3.4.11-3.4.12 – розпізнавання в реальному часі

Дані в усіх випадках записуються в єдину базу даних та оновляються при наступному запуску програми, що дозволяє тримати всю інформацію в одній БД.

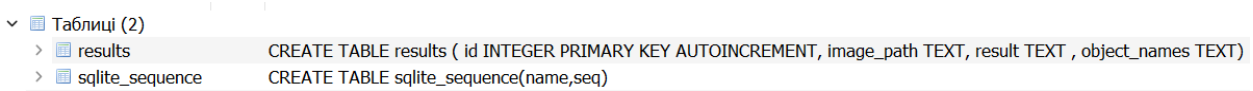
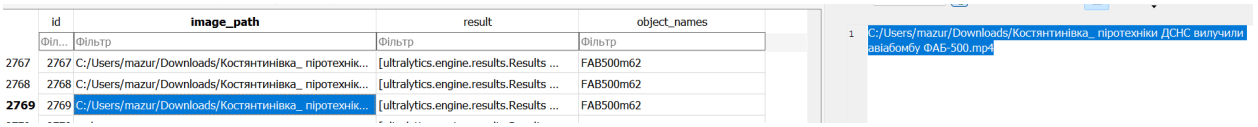


Рис.3.4.13 – структура БД



	id	image_path	result	object_names
	Філ...	Фільтр	Фільтр	Фільтр
2767	2767	C:/Users/mazur/Downloads/Костянтинівка_ піротехнік...	[ultralytics.engine.results.Results ...	FAB500m62
2768	2768	C:/Users/mazur/Downloads/Костянтинівка_ піротехнік...	[ultralytics.engine.results.Results ...	FAB500m62
2769	2769	C:/Users/mazur/Downloads/Костянтинівка_ піротехнік...	[ultralytics.engine.results.Results ...	FAB500m62

Рис.3.4.14 – база даних

Розпізнавання працює, хоч і має певні недоліки у вигляді похибок, що спричинено невеликим датасетом. Програмний модуль має потенціал та потребує більшої кількості часу та розміру анотованих датасетів для покращення показників розпізнавання та підвищення їх точності.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Програмний модуль розпізнавання об'єктів під водою на основі моделей штучних нейронних мереж» були досягнуті наступні результати.

Проведено аналіз існуючих методів машинного навчання та комп’ютерного зору для розпізнавання підводних об'єктів. Вивчено підходи до використання згорткових нейронних мереж у задачах розпізнавання об'єктів під водою.

Зібрано та підготовлено набір даних. Датасет був анотований та підготовлений для навчання моделей нейронних мереж у форматі YOLO. Обрано та налаштовано модель YOLOv8 для розпізнавання підводних об'єктів. Розроблено програмний модуль, що включає аналіз зображень, відео та потокових даних у реальному часі. Здійснено інтеграцію з базою даних для збереження результатів аналізу.

Проведено навчання моделі YOLOv8 на підготовленому датасеті. Оцінено точність роботи моделі за допомогою метрик точності. Проаналізовано результати навчання за допомогою візуалізації метрик.

Розроблений програмний модуль показав непогану ефективність у розпізнаванні об'єктів. Використання моделі YOLOv8 дозволило покращити показники точності та швидкості обробки зображень. Графічний інтерфейс забезпечує зручність використання та інтеграцію з базою даних для збереження результатів. Програмний модуль може бути використаний у різних сферах, таких як оборона, наукові дослідження та промисловість.

У майбутньому можливі такі напрями удосконалення розробленого програмного модуля – розширення датасету для підвищення точності моделі, оптимізація алгоритмів для роботи в реальному часі на вбудованих системах, інтеграція з іншими системами спостереження та моніторингу підводного середовища.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ладуба М. В. Нейромережі пишуть книги та рятують життя: що таке нейронна мережа і як вона працює [Електронний ресурс] / М. В. Ладуба. – 2023. – Режим доступу до ресурсу: https://mc.today/uk/shho-take-nejronna-merezha/.

2. Ковалюк Т. В. Нейромережне розпізнавання об’єктів / Т. В. Ковалюк, І. Б. Бондар. – Київ, 2020. – 26 с.

3. Глибоке навчання: що це таке? [Електронний ресурс] – Режим доступу до ресурсу: https://stfalcon.com/uk/blog/post/deep-learning-what-it-is.

4. Що таке комп’ютерний зір (Computer Vision, CV) [Електронний ресурс] – Режим доступу до ресурсу: https://thetransmitted.com/adlucem/shho-take-kompyuternyj-zir-computer-vision-cv/.

5. ПІДВОДНА ТЕХНІКА І ТЕХНОЛОГІЯ [Електронний ресурс] – Режим доступу до ресурсу: https://eir.nuos.edu.ua/bitstreams/0d3863d9-5bf8-4557-a7eb-928e60023cdf/download.

6. Hooton, E. R. Security to 100 atmospheres: fools rush in where angels fear to tread-but in modern naval warfare it is the robots or Unmanned Underwater Vehicles (UUV) which are now being sent to discover and destroy the hidden secrets, traps and dangers that lurk deep in Poseidon’s realm [Text] / E. R. Hooton. — Armada International, 2005.

7. Родін І. О. СТАН ТА НАПРЯМКИ РОЗВИТКУ ПІДВОДНИХ АПАРАТІВ-МІНОШУКАЧІВ ВІЙСЬКОВО-МОРСЬКИХ СИЛ ВІЙСЬКОВО-МОРСЬКИХ СИЛ ПРОВІДНИХ КРАЇН СВІТУ / І. О. Родін, Є. Е. М. // ЕЛЕКТРОННИЙ ВІСНИК НУК. – 2010. – №3.

8. Розпакування Yolov8: Ultralytics’ Viral Computer Vision Masterpiece [Електронний ресурс] – Режим доступу до ресурсу: https://www.unite.ai/uk/%D1%83%D0%BB%D1%8C%D1%82%D1%80%D0%B

0%D0%BB%D1%96%D1%82%D0%B8%D0%BA%D0%B8-%D0%B9%D0%BE%D0%BB%D0%BE%D0%B28-%D0%BF%D0%BE%D1%8F%D1%81%D0%BD%D0%B8%D0%B2/.

ДОДАТКИ

Додаток А. train_yolov8.py

```
# Імпорт необхідних бібліотек
from ultralytics import YOLO
import os
import torch

# Перевірка доступності GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Шлях до датасету
data_path = 'C:/Users/mazur/PycharmProjects/Underwater/dataset/data.yaml'

# Ініціалізація моделі YOLOv8x
model = YOLO('C:/Users/mazur/PycharmProjects/Underwater/yolov8x-worldv2.pt')

# Тренування моделі
model.train(
    data=data_path,
    epochs=50,
    imgsz=640,
    batch=8,
    workers=4,
    device=device,
    name='yolov8x_worldv2'
)

# Збереження моделі
model_path = 'best_model.pt'
model.save(model_path)

print(f"Модель збережено у {model_path}")
```

Додаток Б. check-annotation.py

```
# Імпорт необхідних бібліотек

import os

import yaml


# Шлях до файлу data.yaml

data_yaml_path = 'C:/Users/mazur/PycharmProjects/Underwater/dataset/data.yaml'


# Перевірка існування файлів зображень та анотацій

with open(data_yaml_path, 'r') as file:

    data = yaml.safe_load(file)


# Отримання шляхів до зображень та анотацій

train_images_path = data['train']

val_images_path = data['val']

train_labels_path = data['labels']['train']

val_labels_path = data['labels']['val']


# Виведення шляхів

print("Train images path:", train_images_path)

print("Validation images path:", val_images_path)

print("Train labels path:", train_labels_path)

print("Validation labels path:", val_labels_path)
```

```
# Перевірка існування першого зображення та його анотації
train_image_files = os.listdir(train_images_path)
train_label_files = os.listdir(train_labels_path)

print("First train image:", train_image_files[0])
print("First train label:", train_label_files[0])
```

Додаток В. data.yaml

```
train: C:/Users/mazur/PycharmProjects/Underwater/dataset/images/train

val: C:/Users/mazur/PycharmProjects/Underwater/dataset/images/val


labels:

  train: C:/Users/mazur/PycharmProjects/Underwater/dataset/labels/train

  val: C:/Users/mazur/PycharmProjects/Underwater/dataset/labels/val


nc: 5

names: ['9N210-235', 'FAB500m54', 'FAB500m62', 'SeaMine', 'TM62M']
```

Додаток Г. main.py

```
# Імпорт необхідних бібліотек
import tkinter as tk
from tkinter import filedialog
from ultralytics import YOLO
import cv2
import sqlite3
import threading


# Клас баз даних
class DatabaseHandler:
    # Ініціалізує об'єкт і викликає setup_database
    def __init__(self, db_name='detection_results.db'):
        self.db_name = db_name
        self.setup_database()


    # Створює таблицю results
    def setup_database(self):
        conn = sqlite3.connect(self.db_name)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS results (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            image_path TEXT,
            result TEXT
        )
    ''')
    # Додати новий стовпець, якщо він не існує
    cursor.execute("PRAGMA table_info(results)")
    columns = [column[1] for column in cursor.fetchall()]
    if 'object_names' not in columns:
        cursor.execute('''
            ALTER TABLE results ADD COLUMN object_names TEXT
        ''')
    conn.commit()
    conn.close()

# Зберігає результати аналізу
def save_to_database(self, image_path, result, object_names):
    conn = sqlite3.connect(self.db_name)
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO results (image_path, result, object_names)
        VALUES (?, ?, ?)
    ''', (image_path, result, object_names))
```

```
        conn.commit()
        conn.close()



# Клас розпізнавання
class ObjectDetectionApp:
    # Ініціалізує графічний інтерфейс, завантажує модель
    def __init__(self, root):
        self.root = root
        self.root.title("Object Detection App")
        self.root.geometry("400x300")  # Розмір головного вікна


        self.model = YOLO('C:/Users/mazur/PycharmProjects/Underwater/runs/detect/yolov8x_worldv213/weights/best.pt')
        self.db_handler = DatabaseHandler()


        self.create_widgets()


    # Створює кнопки
    def create_widgets(self):
        self.photo_button = tk.Button(self.root, text="Analyze Photo", command=self.analyze_photo, width=20)
```

```
        self.photo_button.pack(pady=10)

        self.video_button = tk.Button(self.root, text="Analyze Video",
command=self.analyze_video, width=20)
        self.video_button.pack(pady=10)

        self.stream_button = tk.Button(self.root, text="Analyze Stream",
command=self.analyze_stream, width=20)
        self.stream_button.pack(pady=10)

        self.exit_button = tk.Button(self.root, text="Exit Stream",
command=self.stop_stream, width=20)
        self.exit_button.pack(pady=10)

        self.stream_running = False

    # Витягує імена об'єктів з результатів
    def get_object_names(self, results):
        object_names = []
        for result in results:
            names = [result.names[int(cls)] for cls in result.boxes.cls]
            object_names.extend(names)
        return ', '.join(object_names)
```

```
    # Відкриває діалог для вибору фото
    def analyze_photo(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            img = cv2.imread(file_path)
            results = self.model(img)
            for result in results:
                annotated_img = result.plot(line_width=3)  # Збільшуємо товщину
ліній
                cv2.imshow('Photo', annotated_img)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
            object_names = self.get_object_names(results)
            self.db_handler.save_to_database(file_path, str(results), object_names)


    # Відкриває діалог для вибору відео
    def analyze_video(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            cap = cv2.VideoCapture(file_path)
            while cap.isOpened():
                ret, frame = cap.read()
```

```
            if not ret:
                break
            results = self.model(frame)
            for result in results:
                annotated_frame = result.plot(line_width=3)  # Збільшуємо товщину ліній
                cv2.imshow('Video', annotated_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            object_names = self.get_object_names(results)
            self.db_handler.save_to_database(file_path, str(results), object_names)
        cap.release()
        cv2.destroyAllWindows()

    # Захоплює відеопотік з веб-камери
    def analyze_stream(self):
        def stream():
            self.stream_running = True
            cap = cv2.VideoCapture(0)
            db_handler = DatabaseHandler()  # Create a new database handler for the thread
            while self.stream_running:
                ret, frame = cap.read()
```

```
            if not ret:
                break
            results = self.model(frame)
            for result in results:
                annotated_frame = result.plot(line_width=3)  # Збільшуємо товщину
ліній
                cv2.imshow('Stream', annotated_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    self.stream_running = False
                    break
            object_names = self.get_object_names(results)
            db_handler.save_to_database('webcam', str(results), object_names)
        cap.release()
        cv2.destroyAllWindows()

    thread = threading.Thread(target=stream)
    thread.start()

# Зупиняє потокове відео
def stop_stream(self):
    self.stream_running = False

# Ініціалізує графічний інтерфейс і запускає основний цикл додатку
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = ObjectDetectionApp(root)
    root.mainloop()
```

Додаток Д. Тези

ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО АГЕНТУ СИСТЕМИ ЗАХИСТУ ТА ДІАГНОСТИКИ ОПЕРАЦІЙНОЇ СИСТЕМИ

Сучасний розвиток інформаційних технологій відкриває безліч можливостей, але також ставить перед нами нові виклики та загрози для безпеки та ефективності комп'ютерних систем. Однією з ключових завдань у цій області є створення інтелектуальних агентів, здатних забезпечувати високий рівень захисту та діагностики операційних систем.

Процес розробки програмного забезпечення, що використовується для створення таких агентів, охоплює не тільки програмування та підтримку вихідного коду, але й включає планування проекту, оцінку його здійсненності, аналіз бізнес-вимог, а також тестування і випуск програмного продукту. Цей процес також включає управління проектами, управління персоналом та інші організаційні функції.

Ітераційні методи розробки програмного забезпечення дозволяють значно підвищити гнучкість, ефективність та адаптивність процесу. Ці методи допомагають реалізувати паралельну розробку та адаптацію проекту згідно з новими вимогами та змінами. Такий підхід стане особливо корисним у проекті, який передбачає розробку комплексного програмного забезпечення для діагностики та оптимізації роботи персональних комп'ютерів. Таке ПЗ забезпечить користувачам інструменти для аналізу стану системи, очищення тимчасових файлів, моніторингу навантаження на систему, автоматичного оновлення драйверів та програмного забезпечення, що вкрай важливо для підтримки оптимального рівня продуктивності та безпеки.

Для представлення структури програми використовують діаграми класів UML, які дозволяють визначити ключові класи та їх взаємодії, сприяючи глибокому розумінню та ефективному плануванню розробки, особливо у складних проектах. Це включає забезпечення зручного і зрозумілого

візуального представлення складних структур, що сприяє ефективному спілкуванню між розробниками та допомагає ідентифікувати потенційні проблеми на ранніх етапах розробки. Використання Python разом з діаграмами класів значно сприяє зниженню ризиків помилок у проектуванні, підвищує якість програмного продукту та забезпечує більшу гнучкість і масштабованість у реалізації проектів.

Проектування інтелектуального агента для системи захисту та діагностики операційної системи є складним завданням, яке потребує вдосконаленої архітектури та інтеграції різних технологій. Використання мови програмування Python разом із методологією Unified Modeling Language (UML) може значно сприяти успішному вирішенню цього завдання. Python з її об'єктно-орієнтованим підходом та багатою бібліотекою дозволяє швидко реалізовувати різноманітні функціональні компоненти системи, в той час як UML сприяє чіткому визначенню структури та взаємодії між ними.

Перспективи подальших досліджень у цій області включають розвиток більш складних моделей машинного навчання для прогнозування атак, вдосконалення механізмів швидкого реагування на інциденти та створення адаптивних систем захисту, які можуть самостійно налаштовуватися на зміни в небезпечному середовищі. Також важливим напрямком є інтеграція рішень на основі блокчейн технологій для підвищення прозорості та безпеки даних у мережі. Ці новітні підходи можуть відкрити нові можливості для забезпечення більш ефективного та надійного захисту операційних систем в майбутньому.