МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Катерина НЕСТЕРЕНКО
“____”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: “ Штучний інтелект для управління модулем пошуку, спостереження та наведення на об’єкт”

Виконавець: Михно Дмитро Петрович

Керівник: к.т.н доцент Задонцев Юрій Вікторович

Нормоконтролер: к.т.н доцент Гололобов Дмитро Олександрович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО

"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Михна Дмитра Петровича

1. Тема кваліфікаційної роботи: «Штучний інтелект для управління модулем пошуку, спостереження та наведення на об’єкт».
затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання проекту: з 02.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи : Програмний продукт який можна інтегрувати в систему модуля для управління спостереження та наведення на об’єкт

4. Зміст пояснювальної записки:
   1. Огляд штучного інтелекту в предметній області
   2. Аналіз сучасних технологій
   3. Проектування системи агентів для управління модулем пошуку, спостереження та наведення на об’єкт
   4. Реалізація системи управління модулем

5. Перелік обов'язкових слайдів презентації:
   1. Нейронні мережі

2. Згорткові нейронні мережі
3. Алгоритм YOLO
4. Алгоритм PPO
5. Принцип роботи програмного продукту

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи.	02.10.2023-03.10.2023	
2.	Підготовка та написання 1 розділу «Огляд штучного інтелекту в предметній області».	04.10.2023-20.10.2023	
3.	Підготовка та написання 2 розділу «Аналіз сучасних технологій». Відсилка керівнику.	23.10.2023-08.11.2023	
4.	Підготовка та написання 3 розділу «Проектування системи агентів для управління модулем пошуку, спостереження та наведення на об’єкт». Відсилка керівнику.	09.11.2023-24.11.2023	
5.	Підготовка та написання 4 розділу «Реалізація системи управління модулем». Відсилка керівнику.	27.11.2023-10.12.2023	
6.	Редагування та друк пояснювальної записки, графічного матеріалу Відсилка ПЗ для перевірки на плагіат одним файлом.	11.12.2023-15.12.2023.	
7.	Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	15.12.2023-16.12.2023	
8.	Передзахист кваліфікаційної роботи. Отримання рецензії.	16.12.2023.-17.12.2023	
9.	Здати секретарю ДЕК: ПЗ, ГМ, CD-R з електронними версіями ПЗ, ГМ, презентацію, відгук керівника, рецензію, довідку про успішність, 2 папки, 2 конверта)	18.12.2023-24.12.2023	
10.	Захист дипломної роботи перед ЕК	25.12.2023-31.12.2023	

Дата видачі завдання 02.10.2023 р.
Керівник дипломної роботи: _____к.т.н доцент Юрій ЗАДОНЦЕВ
Завдання прийняв до виконання: Дмитро МИХНО

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Штучний інтелект для управління модулем пошуку, спостереження та наведення на об’єкт»: 84 сторінок, 39 рисунків, 0 таблиці, 36 використаних джерел, 1 додаток.

ШТУЧНИЙ ІНТЕЛЕКТ ДЛЯ УПРАВЛІННЯ МОДУЛЕМ ПОШУКУ, СПОСТЕРЕЖЕННЯ ТА НАВЕДЕННЯ НА ОБ’ЄКТ, ШТУЧНИЙ ІНТЕЛЕКТ, АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ, СИСТЕМА УПРАВЛІННЯ МОДУЛЕМ, ПРОГРАМИ УПРАВЛІННЯ.

Об’єкт дослідження – методика та засіб яким можна виконати виявлення, трекінг об’єктів і прийняття рішень в управлінні модулем за допомоги штучного інтелекту

Мета дипломної роботи – Програмний продукт який можна інтегрувати в систему модуля для управління спостереження та наведення на об’єкт

.Метод дослідження - розробка системи штучного інтелекту та тренування нейронних мереж для штучного інтелекту, спрямованого на управління модулем пошуку, спостереження та наведення на ціль. Планується розробити штучний інтелект, який буде навчений ідентифікувати, відстежувати та приймати рішення стосовно цілей у динамічному середовищі. Цей ШІ буде здатен імітувати різні рівні складності та тактики поведінки залежно від сценаріїв використання.

Результатом цієї роботи є ефективно навчений штучний інтелект, який може отримувати та обробляти об'єкти з відеопотоку, а також автономно приймати рішення за допомогою середовища OpenAI Gym для керування модулем наведення на ціль. Розробка та дослідження виконувались під управлінням ОС Windows 11, з використанням Python як основної мови програмування як для скриптів, так і для нейронної мережі

ABSTRACT

Explanatory note to the qualification work "Artificial intelligence for managing the search, observation and targeting module": 84 pages, 39 figures, 0 tables, 36 used sources, 1 appendix.

ARTIFICIAL INTELLIGENCE FOR MANAGING THE SEARCH MODULE, OBSERVATION AND TARGETING, ARTIFICIAL INTELLIGENCE, ANALYSIS OF MODERN TECHNOLOGIES, MODULE MANAGEMENT SYSTEM, MANAGEMENT PROGRAMS.

The object of research is a methodology and tool that can be used to detect, track objects and make decisions in module management using artificial intelligence

Purpose of the thesis - A software product that can be integrated into the system of the module for controlling surveillance and targeting of the object.

The research method is the development of an artificial intelligence system and training of neural networks for artificial intelligence aimed at managing the search, observation and targeting module. It is planned to develop an artificial intelligence that will be trained to identify, track, and make decisions about targets in a dynamic environment. This AI will be able to simulate different levels of complexity and behavioral tactics depending on the scenarios of use.

The result of this work is an efficiently trained artificial intelligence that can receive and process objects from a video stream, as well as make autonomous decisions using the OpenAI Gym environment to control the targeting module. The development and research was performed under Windows 11, using Python as the main programming language for both scripts and the neural network.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ШІ – Штучний інтелект

ML (Machine learning) – Машинне навчання

ШНМ – Штучна нейронна мережа

CNN (Convolutional neural network) – Згорткова нейронна мережа

PyPI (Python Package Index) – каталог програмного забезпечення, написаного на мові програмування Python

Pip – система керування пакунками, яка використовується для встановлення та управління програмними пакетами, які написані на Python

ВСТУП

Сучасний світ характеризується стрімким розвитком технологій і постійною еволюцією в усіх сферах людської діяльності. Питання взаємодії комп’ютерними системами з навколишнім середовищем набуває особливої важливості та актуальності . Взаємодія комп’ютерних систем з оточуючим середовищем ставить складне завдання через те, що ці середовища часто виявляються надто складними і непередбачуваними. Різноманітні системи, які діють у нашому світі, потребують від комп’ютерів здатності швидко аналізувати ситуації, пристосовуватися до нових умов і взаємодіяти ефективно.

Технології, що базуються на штучному інтелекті, стають ключовим інструментом у реалізації даної взаємодії, надаючи машинам можливість самостійного «бачення», аналізу та інтерпретації оточуючого світу. Вони відкривають нові горизонти у побудові систем, які здатні розпізнавати об’єкти, розуміти контекст їхньої появи та динаміку руху, а також адаптовуватися до стрімко змінюваних умов реальності.

Слід зазначити що лише розпізнавання об’єктів і збір інформації недостатні для успішного управління таких систем. Прийняття рішень на основі отриманих даних відіграє критичну роль у забезпеченні точності та ефективності взаємодії. Тут штучний інтелект виступає в ролі аналітика та стратега. Він обробляє інформацію, оцінює ситуацію, враховуючи фактори як об’єктів, так і зовнішні впливи, і приймає рішення щодо оптимального керування системою.

Штучний інтелект для управління модулем пошуку, спостереження та наведення на об’єкт актуальний у низці галузей, які покликані розв'язувати проблеми спостереження та ідентифікації об'єктів. Наприклад, у військовій сфері, де необхідно швидко ідентифікувати та реагувати на загрози, штучний інтелект може сприяти автоматизації процесів виявлення та слідкування за об'єктами, а також наведення зброї та інших систем. У космічній індустрії, системи на основі штучного інтелекту можуть спрощувати задачі пошуку та взаємодії з космічними об'єктами, оптимізуючи траєкторії руху супутників та інших апаратів, а також забезпечувати

керування місіями у глибокий космос. У галузі медицини, технології на основі ШІ можуть сприяти високоточному направленню медичного обладнання, наприклад, для проведення радіотерапії або хірургічних втручань із високою точністю та мінімізацією помилок. У галузі безпеки та спостереження, системи ШІ можуть автоматизувати процеси розпізнавання осіб, транспортних засобів, а також виявлення аномальних ситуацій, що можуть вказувати на загрози або порушення. У робототехніці, алгоритми ШІ можуть підтримувати навігацію та взаємодію роботів із навколишнім середовищем, включаючи виявлення та маніпулювання об'єктами. Також важливою сферою є авіаційна індустрія, де алгоритми для наведення та слідкування можуть бути використані для поліпшення безпеки політів, контролю повітряного простору та оптимізації маршрутів. В цілому, штучний інтелект у сфері управління модулем пошуку та наведення може знайти застосування в будь-якій галузі, де критичні точність, швидкість реакції та надійність систем спостереження та управління.

РОЗДІЛ 1. ОГЛЯД ШТУЧНОГО ІНТЕЛЕКУ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Основи нейронних мереж

1.1.1 Поняття нейронної мережі

Штучні нейронні мережі — це моделі обчислень, натхненні структурою та функцією нейронів у мозку. Вони складаються з великої кількості взаємодіючих нейронів, які можуть адаптуватися та навчатися від великої кількості даних.

Біологічні нейрони в основі своєї будови і функції є ключовими елементами мозкової структури. Наш мозок налічує приблизно 10^10 таких нейронів, кожен з яких може мати декілька тисяч взаємозв'язків. Ці унікальні клітини не лише формують наші думки та спогади, але і використовують попередній досвід для прийняття рішень, що робить їх відмінними від інших клітин нашого організму. Основна структура біологічного нейрона включає тіло клітини, відоме як сома, дендрити, які приймають імпульси, та аксон, що передає імпульси від нейрона.

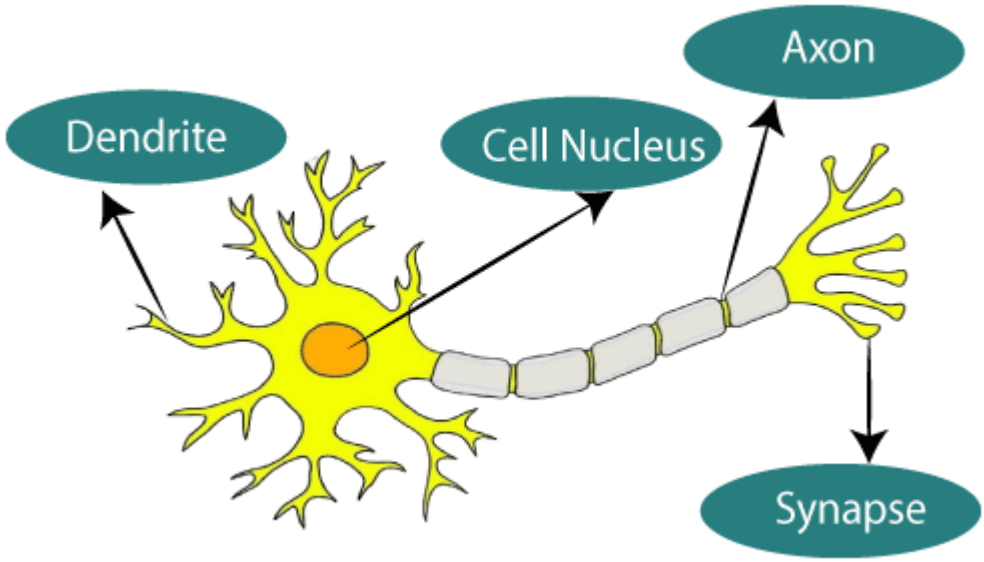


Рис.1.1 Біологічна нейронна мережа

Штучна нейронна мережа в сфері штучного інтелекту намагається відтворити принцип роботи нейронів людського мозку, дозволяючи комп’ютерам аналізувати інформацію та приймати рішення схожим на біологічний спосіб. Такі мережі

створюються за допомогою спеціального програмування, що імітує взаємодію мозкових клітин.

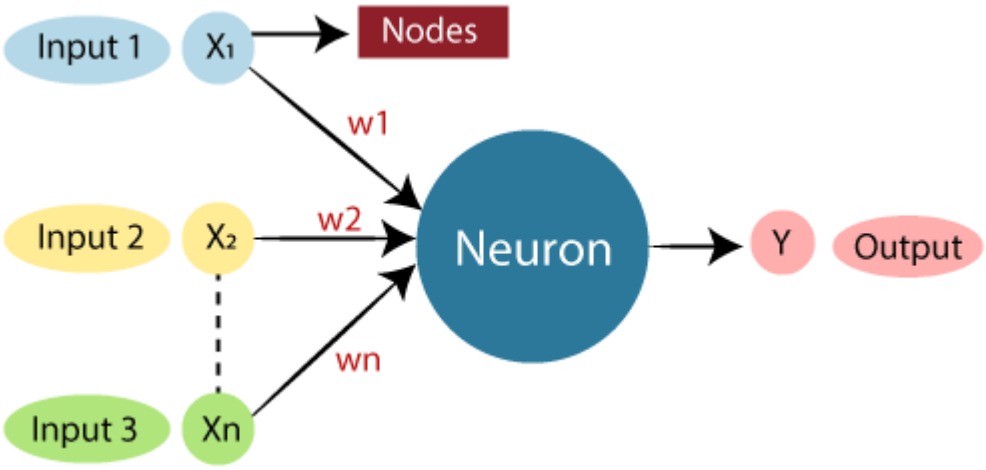


Рис.1.2 Штучна нейронна мережа

1.1.2 Історичний огляд Штучних нейроних мереж

Оглядучись назад винахідники здавна прагнули до створення машин, здатних мислити, та розумних інструментів. Це бажання зародилося ще в епоху Стародавньої Греції, де міфічні постаті, такі як Пігмаліон, Дедал і Гефест, символізували легендарних винахідників, а їх роботи - Галатея, Талос та Пандора - можна вбачити аналогії перших спроб створити живе з неодушевленого.

Ця ідея відродилася, коли у XX столітті з'явилися перші комп'ютери. З розвитком технологій ідея створення розумних машин отримала новий поштовх завдяки нейронним мережам. В історії їх розвитку можна виділити декілька ключових етапів.

У 1940-х роках, піонери такі як Воррен Маккаллох та Вальтер Піттс представили перші математичні моделі нейронів, намагаючись імітувати функціональну активність мозку на базовому рівні. Ці зусилля заклали фундамент для подальших досліджень у цій області.

У період 1950-60-х років, завдяки роботі Френка Розенблатта, світова наукова спільнота познайомилася з концепцією перцептрона — алгоритмом, що мав можливість самостійного навчання на основі даних. Втім, обмежені можливості цієї

моделі, які були детально висвітлені Марвіном Мінським та Сеймуром Папертом, призвели до скорочення інтересу до нейронних мереж на деякий час.

1980-90-і роки принесли новий ренесанс у сфері нейронних мереж завдяки розробці мультишарових структур та введенню алгоритму зворотного розповсюдження помилки. Цей алгоритм забезпечив навчання більш складних моделей, розширюючи межі можливого застосування нейронних мереж.

Проте, незважаючи на ці величезні кроки вперед, на початку 2000-х років нейронні мережі знову втратили популярність, поступаючись місцем іншим методам машинного навчання.

Цей тренд змінився тільки в останній декаді, коли нові алгоритми, потужні комп'ютерні системи та доступ до великих наборів даних призвели до відродження інтересу до глибоких нейронних мереж і поставили їх на передові штучного інтелекту.

1.1.3 Структура нейроної мережі

Штучні нейронні мережі представляють собою системи, які імітують структуру нейронів у людському мозку, об'єднуючи окремі елементи, звані вузли або штучні нейрони, у структуровані шари. Кожен вузол отримує вхідні дані, які потім множаться на відповідні ваги, визначаючи важливість цих даних для завдання. Після обчислення ваг і вхідних даних отримана сума проходить через функцію активації, яка вирішує, чи буде вузол активовано, та в якому ступені.

Мережі можуть мати різну структуру і глибину. Зазвичай вони містять три основних типи шарів: вхідний, один або декілька прихованих, та вихідний. Вхідний шар отримує інформацію з зовнішнього середовища або датчиків і передає її на наступні шари. Приховані шари обробляють дані і передають їх далі. Вихідний шар надсилає результати обчислень до інших систем або пристроїв.

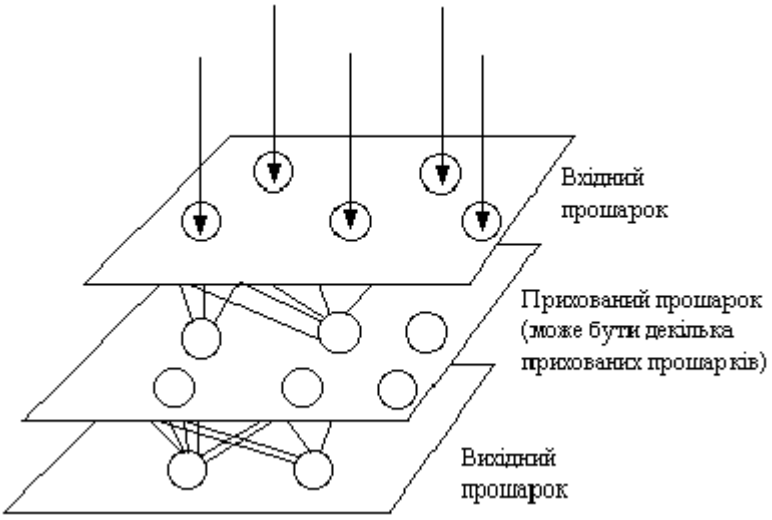


Рис.1.3 Схема штучної НМ

Напрямок зв’язку від одного нейрона до іншого є важливим аспектом нейромереж. У більшості мереж кожен нейрон прихованого прошарку отримує сигнали від всіх нейронів попереднього прошарку і зазвичай від нейронів вхідного прошарку. Після виконання операцій над сигналами нейрон передає свій вихід всім нейронам наступних прошарків, забезпечуючи передачу сигналу вперед (feedforward) на вихід. При зворотному зв’язку вихід нейронів прошарку скеровується до нейронів попереднього прошарку.

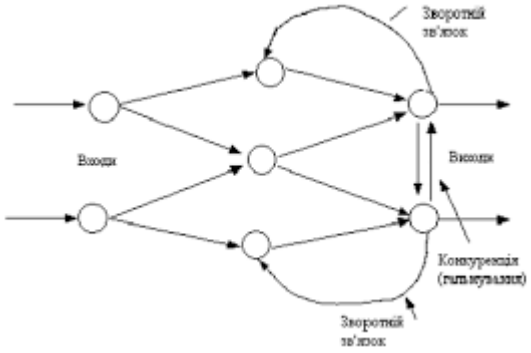


Рис.1.4 Схема штучної НМ зі зворотним зв’язком

1.1.4 Навчання нейроної мережі

Навчання нейронної мережі є ітеративним процесом адаптації внутрішніх параметрів системи до конкретного набору даних. Основна мета — мінімізувати різницю між виходом мережі та очікуваними результатами, які ми маємо в нашому навчальному наборі.

Параметри мережі включають в себе ваги та зсуви, які визначають, як вхід впливає на вихід. Важливим етапом перед навчанням є ініціалізація цих параметрів. Зазвичай вони ініціалізуються випадковими маленькими значеннями, що допомагає забезпечити динамічність навчання.

Кожен вхідний приклад подається на вхід мережі, де він проходить через різні шари, і на виході отримується прогноз. Функція втрат допомагає виміряти різницю між прогнозом мережі та дійсним значенням. Ця відмінність, або "помилка", є ключовою для корекції параметрів.

Для оптимізації параметрів використовують градієнтний спуск або його варіації. Це метод, який визначає, як потрібно змінювати параметри для зменшення помилки. Величина цих змін регулюється швидкістю навчання - невеликим числовим значенням, яке визначає, наскільки "швидко" мережа вчиться.

Процес навчання складається з декількох епох. Одна епоха — це один прохід через весь навчальний набір даних. Після кожної епохи помилка обчислюється, і параметри мережі коригуються. Навчання зазвичай триває доти, доки мережа не досягне певного критерію зупинки, такого як мінімальне зменшення помилки або досягнення певної кількості епох.

Підходи до навчання нейронних мереж:

- Навчання з учителем є підходом до машинного навчання, при якому модель навчається на основі заздалегідь підготовлених даних, що складаються з вхідних значень та відповідних їм очікуваних виходів. Основна мета полягає в тому, щоб підлаштувати модель таким чином, щоб вона могла передбачати виходи для нових, раніше невідомих вхідних даних. Для процесу навчання з учителем потрібен набір даних, який містить вхідні приклади та відповідні їм правильні виходи. Такий набір даних часто розділяється на дві частини: одна для безпосереднього навчання моделі, інша для валідації. Під час навчання, кожний вхідний приклад подається на модель, яка робить прогноз. Помилка обчислюється як різниця між прогнозом моделі та справжнім виходом. На основі цієї помилки відбувається корекція параметрів моделі з метою зменшення майбутніх помилок.

- Навчання без учителя є підходом машинного навчання, де модель працює із вхідними даними без попередньо вказаних відповідей або міток. Основна мета полягає в пошуку закономірностей, структур або відносин у вхідних даних.

- Навчання з частковим наглядом – це підхід до машинного навчання, який використовує як марковані, так і немарковані дані для навчання. Зазвичай використовується в ситуаціях, коли маркованих даних недостатньо або їх збір коштовний, але немаркованих даних доступно велика кількість. За допомогою напівконтрольованого навчання ви тренуєте початкову модель на кількох позначених зразках, а потім ітеративно застосовуєте модель до більшого набору даних.

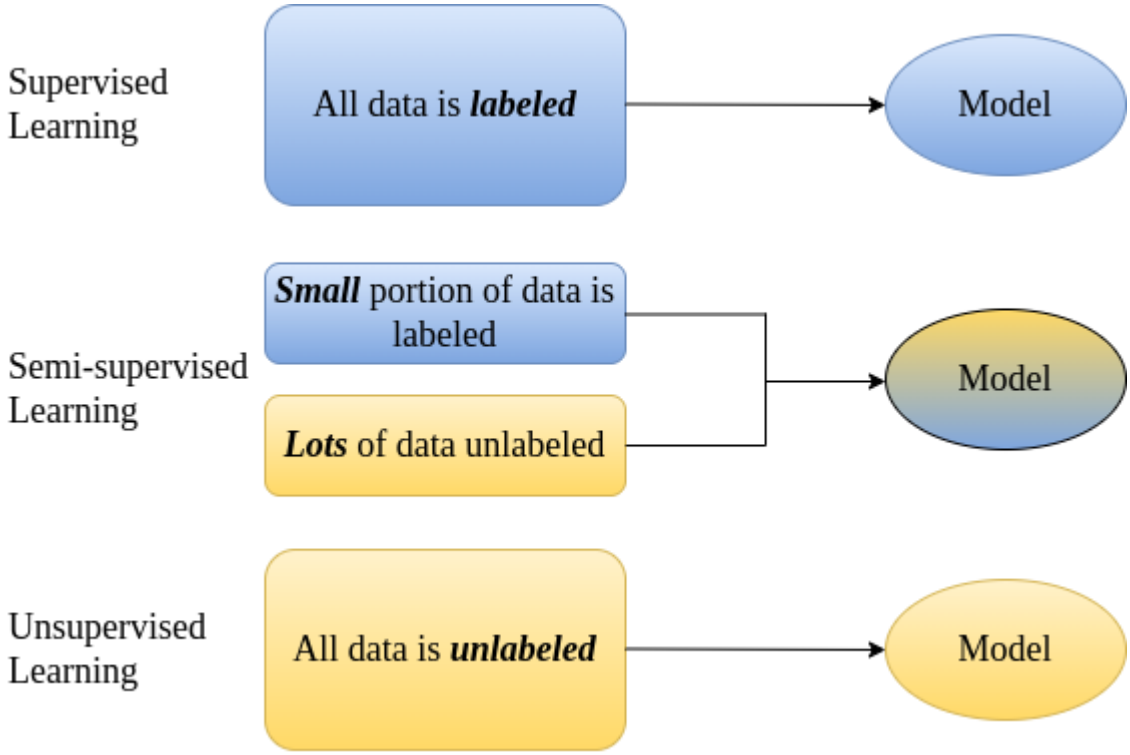


Рис.1.5 Діаграма трьох основних типів машинного навчання.

- Навчання з підкріпленням є підходом до машинного навчання, де агент навчається взаємодіяти із середовищем, щоб максимізувати певний кумулятивний показник нагороди. Відрізняється від інших підходів тим, що немає явно вказаних правильних відповідей, і агент має досліджувати середовище, щоб визначити оптимальні дії. В контексті навчання з підкріпленням, агент – це сутність, що приймає рішення. Агент взаємодіє із середовищем, виконуючи дії, і отримує

нагороди або покарання в залежності від результатів цих дій. Кожна дія, яку виконує агент в середовищі, призводить до отримання винагороди або покарання. Метою агента є максимізація сумарної винагороди за певний період часу. Функція втрат допомагає агенту оцінити, наскільки добре він виконує свої дії.

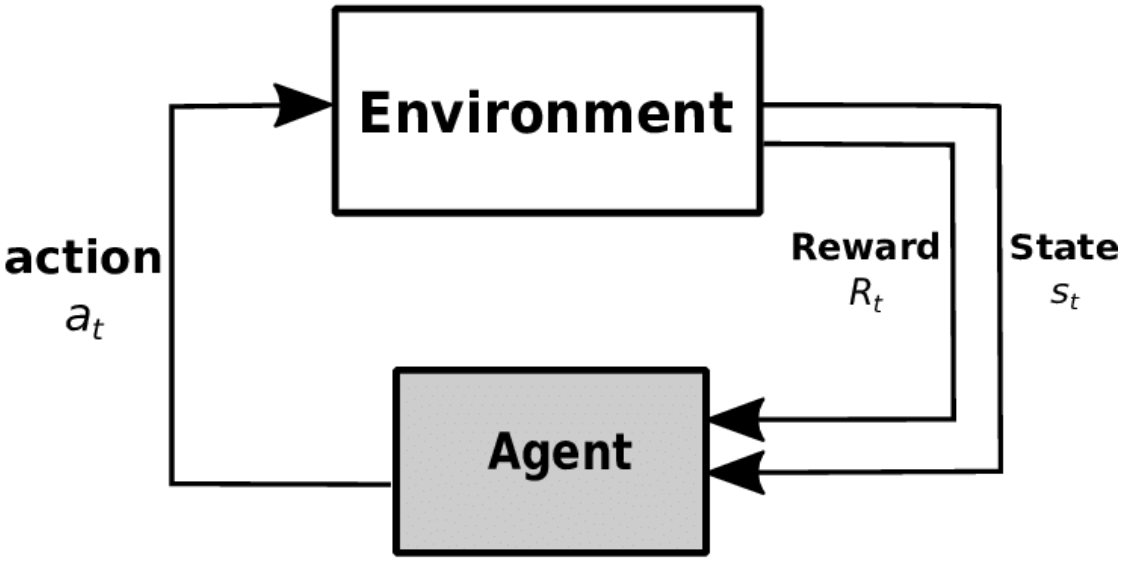


Рис.1.6 Навчання з підкріпленням

1.1.6 Глибинне навчання та глибинні нейромережі

Глибинне навчання – це підгалузь машинного навчання, яка зосереджена на створенні ієрархічних моделей. Ці моделі працюють за принципом послідовного витягування корисної інформації з вхідних даних, перетворюючи їх через безліч обробних шарів. Ці шари можуть виконувати як лінійні, так і нелінійні перетворення, допомагаючи виділити і зібрати ознаки на різних рівнях абстракції.

Серцем глибинного навчання є глибинні нейронні мережі. Вони є розширенням стандартних нейронних мереж і містять значно більше шарів обробки. Завдяки цій багатошаровості глибокі нейронні мережі можуть видобувати та об'єднувати ознаки на різних рівнях, просуваючись від вхідних даних до виходу.

Глибинні нейронні мережі відрізняються від звичайних нейронних мереж своєю здатністю моделювати більш складні взаємозв'язки завдяки збільшеній кількості елементів і зв'язків між ними. Крім того, вони можуть бути оптимізовані для конкретних задач, що робить їх дуже потужними інструментами для певних задач, але може обмежувати їх універсальність в інших ситуаціях.

1.2 Комп’ютерний зір

1.2.1 Поняття комп’ютерного зору

Комп'ютерний зір - це галузь дослідження, зосереджена на проблемі допомоги комп'ютерам бачити. Це мультидисциплінарна сфера, яку в широкому сенсі можна назвати підсферою штучного інтелекту та машинного навчання, яка може передбачати використання спеціалізованих методів і використання загальних алгоритмів навчання.

1.2.2 Історія розвитку комп’ютерного зору

У 1960-х роках вчені почали активні дослідження в області обробки зображень, проте обчислювальні засоби того часу суттєво обмежували можливості використання великих наборів даних та впровадження складних алгоритмів. Протягом 1970-х років було розроблено численні методи обробки зображень та розпізнавання образів. Особливо важливим стало винайдення перетворення Хафа, що спрямоване на ефективне виявлення ліній, еліпсів та інших геометричних форм на зображеннях.

У 1980-х та 1990-х роках основний акцент досліджень в області комп’ютерного зору був зміщений у бік алгоритмів машинного навчання. Це дозволило системам навчатися на основі великих датасетів і постійно вдосконалюватися, збільшуючи точність розпізнавання. Значущим моментом стала розробка алгоритму розпізнавання обличчя Віоли-Джонса у 2001 році, який виявився ефективним у виявленні облич на зображеннях в реальному часі.

Починаючи з 2000-х років, глибоке навчання почало революціонізувати галузь комп’ютерного зору. Зокрема, згорткові нейронні мережі дозволили створити ієрархічні представлення візуальних даних, значно підвищуючи точність розпізнавання об'єктів. Така архітектура мережі забезпечує високу ефективність у завданнях класифікації, відстеження руху та інших візуальних завданнях.

1.2.3 Згорточна нейронні мережі (CNN)

Згорточна нейронна мережа (CNN) — це мережева архітектура для глибокого навчання, яка навчається безпосередньо з даних. CNN особливо корисні для пошуку

шаблонів на зображеннях для розпізнавання об’єктів. Вони також можуть бути досить ефективними для класифікації неграфічних даних, таких як аудіо, часові ряди та сигнальні дані.

Згорточні нейронні мережі (CNN) складаються з послідовності шарів, розроблених для виявлення структурованих ієрархійних ознак та подальшої класифікації. Основними компонентами цих мереж є згорткові шари, які використовують набори вагових фільтрів для втягування специфічних ознак з вхідних даних. Далі, рівні агрегації застосовуються для просторового зменшення розмірності вихідних даних конволюційного шару, редукуючи обчислювальний обсяг та виокремлюючи найбільш важливі ознаки. Після кількох таких шарів, вихідна інформація підлягає процедурі вирівнювання та подається на повністю з'єднані шари для фінальної класифікації або регресійного аналізу ієрархічних ознак.

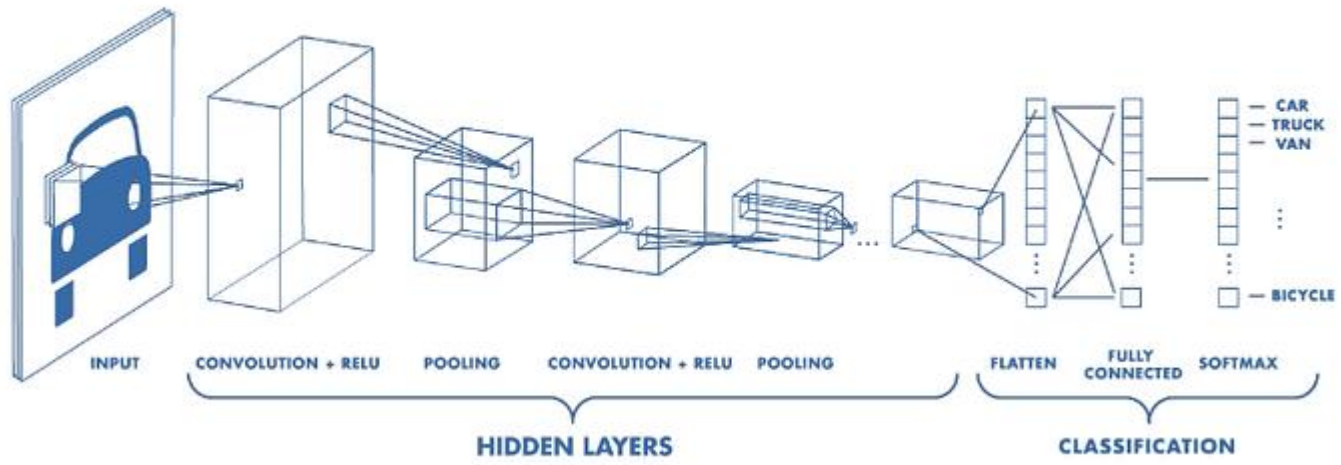


Рис.1.7 Архітектура згорткової нейронної мережі

Операції згортки в згорткових шарах нейронних мереж виконують ключову функцію, яка полягає у математичній обробці вхідних даних за допомогою ядра згортки, відомого як детектор ознак. Дана операція полягає у застосуванні детектора ознак до вхідного зображення, представленого у формі тензора x, що призводить до формування карти ознак m. Детектор ознак містить в собі кількість фільтрів, кожен з

яких являє собою тензор параметрів з певною розмірністю, яка забезпечує покриття лише обмеженої частини зображення, але відповідає його глибині.

Функціональність фільтрів не обмежується лише виявленням ознак та переходів, а також може включати застосування операцій збільшення різкості або розмивання зображення. Фільтри є параметричною частиною нейронної мережі, до якої застосовується процес навчання.

Кількість фільтрів є гіперпараметром нейронної мережі, оскільки збільшення їх числа може призвести до зростання кількості потенційно виявлених ознак та закономірностей. Проте, надмірна кількість фільтрів може спричинити повторення та нерелевантне виявлення ознак, що може ускладнити процес та призвести до перенавчання моделі. Збільшення кількості фільтрів веде до відповідного зростання кількості карт ознак, які визначають глибину вихідних даних і тим самим глибину вхідних даних для наступного шару нейронної мережі.

У процесі навчання фільтри оптимізуються для виявлення певних ознак зображення, які відображаються на картах ознак. Кожен фільтр відповідає за створення однієї карти ознак. Під час переміщення фільтра по зображенню, він формує карту ознак, оцінюючи рецептивні поля. Варіювання кроку переміщення фільтра (stride) є важливим параметром, що впливає на формування карти ознак. Хоча абсолютне положення кожної ознаки на карті не зберігається, їх відносне положення зберігається, що важливо для збереження властивостей зображення.

Матиматично це може виглядати так:

$$M(i,j) = (K * X)(i,j) = \sum_m \sum_n K(m,n)X(i-m, j-n),$$

де $M(i,j)$ – елемент карти ознак з координатами i та j,

X – вхідне зображення,

K – детектор ознак,

m, n – розмірності детектора ознак.

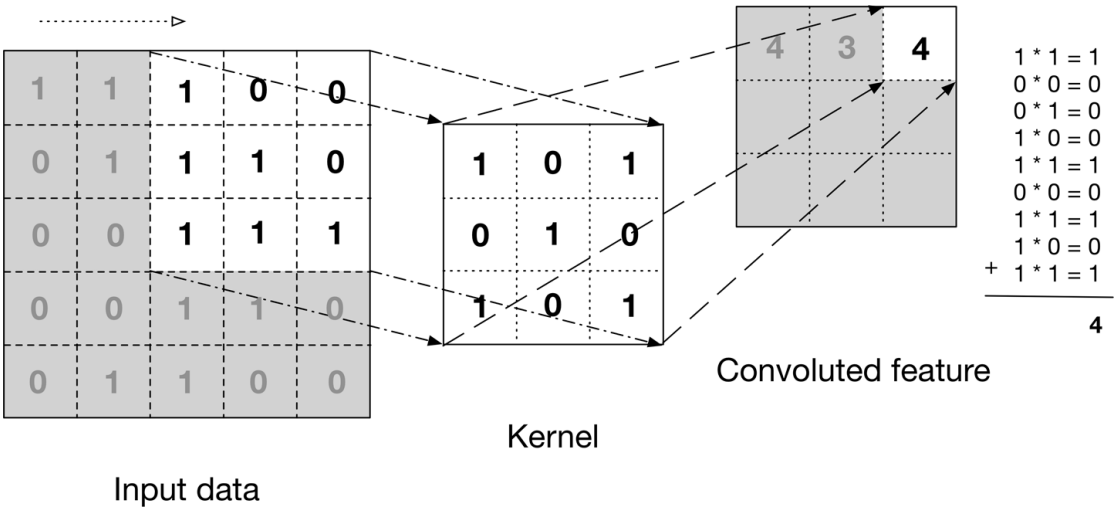


Рис.1.8 Фільтр CNN

Кожен елемент карти ознак у згорткових шарах нейронної мережі можна інтерпретувати як результат перетворення, яке відбувається через скалярний добуток значень з певної області вхідного зображення, відомої як рецептивне поле нейрона. Цей процес здійснюється у взаємодії з сусідніми значеннями на тій же карті ознак.

Здатність фільтрів виявляти певні ознаки зображення залежить від рівня нейронної мережі, на якому вони знаходяться. На початкових етапах мережі фільтри зазвичай розпізнають прості візуальні характеристики, такі як переходи кольору або зміни в освітленості. Однак, на більш глибоких рівнях мережі фільтри стають здатними розпізнавати складніші структури, включаючи кути, закруглення та навіть більш абстрактні образи. Це стає можливим завдяки інтеграції та аналізу інформації з попередніх шарів мережі, а також збільшенню кількості точок вихідного зображення, які впливають на формування кожної карти ознак на більш глибоких рівнях.

Також слід відзначити, що вплив окремої ознаки на результат аналізу зображення збільшується, коли ця ознака знаходиться ближче до центру зображення. Це пояснюється тим, що значення, пов'язані з такою ознакою, впливають на більшу кількість значень відповідної карти ознак, у порівнянні з ознаками, що розташовані ближче до країв зображення. Цей ефект досягається за рахунок більшого охоплення рецептивними полями фільтрів.

В згортковому продукті пікселі на краях зображення використовуються менше, ніж пікселі в середині, що призводить до втрати інформації з країв. Щоб вирішити цю проблему та зберегти інформацію по периферії, застосовується метод відступу (padding). При відступі до зображення додаються додаткові рамки, зазвичай заповнені нулями. Параметр "padding" позначає кількість елементів, доданих на кожну сторону зображення. Цей метод використовується для збереження просторової розмірності зображення після згорткової операції, щоб не втрачати інформацію на краях та дозволити мережі краще вчити ознаки, розташовані по периферії зображення.

Крок обходу зображення, відомий як stride (s), є важливим гіперпараметром у структурі нейронної мережі, який необхідно визначати на етапі розробки та конструювання моделі перед процесом навчання. Збільшення цього параметра веде до зменшення розмірності вихідних даних, що знижує загальну складність моделі та зменшує перекриття між рецептивними полями різних елементів зображення.

Ще одним ключовим гіперпараметром є доповнення нулями, відоме як padding (p). Ця техніка використовується для збереження первинної розмірності карт ознак у порівнянні з вхідними даними, а також для більш контрольованої обробки зображення. Доповнення нулями включає додавання нульових значень до країв вхідного зображення, забезпечуючи збільшення впливу значень, розташованих на краях зображення. Без застосування такого доповнення, крайні елементи зображення вносять свій вклад у формування карт ознак лише один раз, що призводить до втрати інформації з країв зображення.

Таким чином, обидва гіперпараметри, stride і padding, грають вирішальну роль у конфігурації згорткових шарів нейронної мережі, впливаючи на обробку вхідних зображень та формування карт ознак, забезпечуючи при цьому різні рівні складності та деталізації при аналізі вхідних даних.

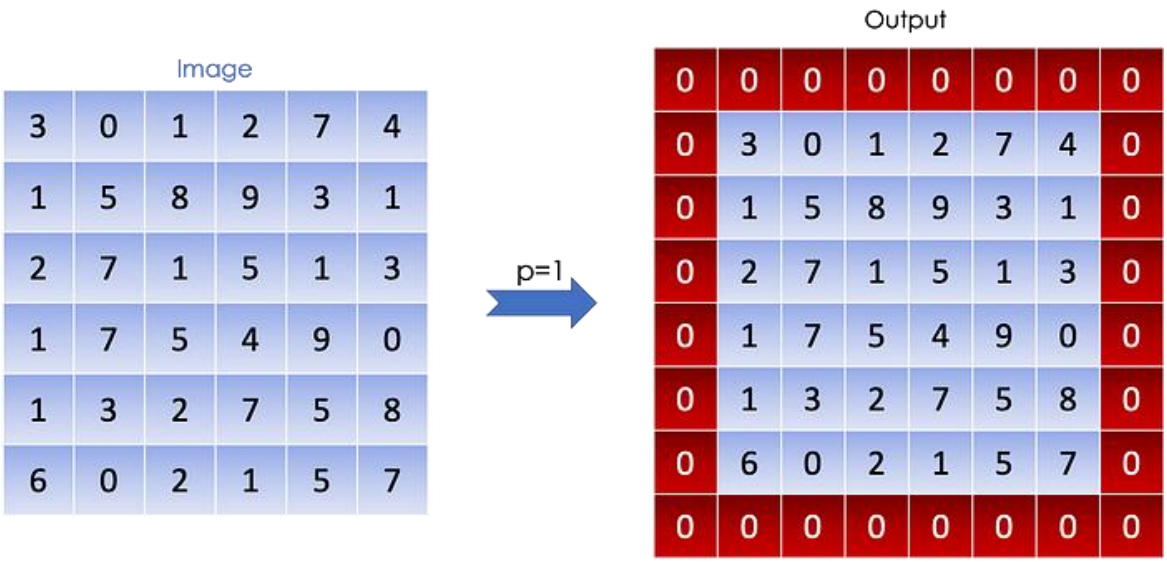


Рис.1.9 Падинг

Розмірність фільтрів у згорткових шарах нейронних мереж визначається з урахуванням того, що із збільшенням їх розмірів фільтри стають здатними виявляти більші та рідкісні ознаки, які можуть мати значний вплив на аналіз зображення. Проте, у сучасних дослідженнях спостерігається тенденція до зменшення розмірності фільтрів, оскільки це сприяє зниженню використання ресурсів, враховуючи, що кількість параметрів у фільтрах зростає квадратично із збільшенням їх розмірності.

Для компенсації зменшення розмірів рецептивних полів в таких умовах часто використовується вертикальне послідовне розташування згорток. Ця техніка дозволяє збільшити глибину мережі, а не її ширину, що є особливо корисним при використанні згорток великої розмірності. Подібний підхід також сприяє поліпшенню ефективності роботи мережі.

В деяких випадках використовуються згортки з розмірністю 1 для зменшення глибини даних перед проведенням операцій, які потребують значних обчислювальних ресурсів. Такий метод дозволяє оптимізувати обробку даних, забезпечуючи ефективність при одночасному зниженні використання ресурсів. Розмірність карт ознак залежить від вказаних гіперпараметрів та може бути обчислена наступним чином:

$$d_m = \frac{d_x - f + 2 * p}{s + 1},$$

де d_m – розмірність карт ознак (ширина або висота),

d_x – відповідна розмірність вхідних даних,

f – розмірність фільтрів,

p – доповнення нулями,

s – розмір кроку обходу зображення.

Рис.1.10 формула ромірності карт ознак

Також можна паралельно використовувати різні розміри фільтрів на одному рівні. Такий тип згортки називається груповими згортками, і він вимагає групування вихідних даних. для кожної розмірності окремо від інших груп, що потребуватиме використання різних згорток до цих груп на кожному наступному рівні. Це підвищує потужність мережі, оскільки кожна група згорток виконує завдання окремий згортковий шар, і він широко використовується в сучасних архітектурних стилів. Крім того, час від часу застосовується операція розширення, який використовує елементи фільтру замість сусідніх елементів зображення, а також до елементів, які мають певний крок. Це забезпечує значно зменшити розмір даних у випадку, якщо в них є значна кількість надлишкової інформацію та узагальнювати ознаки.

Агрегувальні шари, відомі також як шари субдискретизації, є ключовими елементами згорткових нейронних мереж, виконуючи функцію зменшення розмірності даних і одночасно зберігаючи важливі характеристики. Вони можуть бути як глобальними, так і локальними, обробляючи окремі групи вхідних даних. Агрегувальні шари ефективно запобігають перенавчанню, зберігаючи глибину вхідних даних.

Існують два основних типи агрегувальних шарів: усереднювальні та максимізаційні. Усереднювальні шари визначають середнє значення з групи нейронів попереднього шару для кожного елементу, тоді як максимізаційні шари вибирають максимальне значення з цієї групи. Формально операції для обох типів агрегування записуються наступним чином:

$$p(i,j) = \frac{\sum_{m,n} x(i-m, j-n)}{m * n},$$

для усередненого агрегування,

$$p(i,j) = \max_{i,j}\big(x(i-m, j-n)\big),$$

для максимизаційного агрегування,

$p(i,j)$ – значення елементу поточного рівня з координатами i та j,

x – вхідні дані з попередніх рівнів,

m, n – розмірність рецептивного поля.

Рис.1.11 Формули Усереднювального та максимізаційного агрегування

Агрегувальні шари в згорткових нейронних мережах служать для підвищення стійкості мережі до варіацій у вхідних даних, таких як зміна кута погляду чи положення об’єкта. Шляхом використання визначених критеріїв, таких як усереднення або максимізація, агрегувальні шари мінімізують вплив незначних змін на вихідні дані, забезпечуючи консистентність виявлення ознак незалежно від змін в положенні чи освітленні. Максимізаційне агрегування зазвичай краще впорається з цією задачею.

Агрегувальні шари мають декілька спільних гіперпараметрів з згортковими шарами, включаючи крок обходу зображення та розмірність рецептивного поля. В більшості мереж використовується рецептивне поле розміром 2x2, що дозволяє ефективно зменшувати розмірність даних. В деяких архітектурах агрегувальні шари замінюються згортковими шарами з більшим кроком. Розмірність результатів у агрегувальних шарах визначається аналогічно до розрахунку розмірності карт ознак у згорткових шарах.

В деяких випадках замість агрегування використовується субдискретизація, яка включає обчислення суми елементів у рецептивному полі з подальшим

множенням на параметр і додаванням параметра зсуву, застосовуючи нелінійну функцію активації. Формалізовано таку операцію можна записати так:

$$p(i,j) = f\left(w * \sum_{m,n} x(i-m, j-n) + b\right),$$

де $p(i,j)$ – значення елементу поточного рівня з координатами i та j,
x – вхідні дані з попередніх рівнів,
m, n – розмірність рецептивного поля.

Рис.1.12 Формула субдискретизації

У рамках архітектури згорткових нейронних мереж, повнозв'язні шари представляють собою компоненти, де кожен нейрон у шарі формує зв'язок з усіма нейронами попереднього шару. Це принципово відрізняється від згорткових шарів, де зв'язки є локалізованими. Використання повнозв'язних шарів на ранніх або проміжних етапах нейронної мережі може бути неоптимальним, оскільки це потенційно призводить до збільшення складності моделі та можливого ігнорування деталей, виявлених у процесі згортки та агрегації.

Однак, на завершальних етапах мережі повнозв'язні шари часто використовуються для інтеграції та синтезу інформації, підготовленої попередніми шарами, з метою виведення кінцевого результату. Для цього дані з попередніх шарів перетворюються у лінійний формат, тобто "випрямляються", для забезпечення їх подальшого оброблення. Повнозв'язні шари виконують обчислення, схоже на згорткові шари, включаючи обчислення скалярного добутку вхідних даних і вагових коефіцієнтів з наступним додаванням зсуву. Тому загальна формула повнозв’язного рівня схожа на формулу згорткового:

$$y = f(w^T * x + b).$$

Рис.1.13 Формула повнозв’язного рівня

Повнозв'язні рівні в згорткових нейронних мережах характеризуються використанням масиву параметрів w, відрізняючись від згорткових рівнів, де w представляє детектор ознак. На цих рівнях застосовуються функції активації, які або передають результати для обробки наступними рівнями, або ж використовуються для визначення вихідних результатів мережі. Вибір активаційних функцій на повнозв'язних рівнях залежить від їх положення у мережі та загальної цілі, з якою вона створена. Для формування виходів часто використовуються такі активаційні функції, як Softmax та сігмоїд.

Повнозв'язні рівні можна уявити як згорткові шари, де детектор ознак має розмірність 1x1 і де реалізовано всі можливі зв'язки між нейронами.

Рівень втрат у згорткових нейронних мережах, який зазвичай розташований на останньому етапі, відіграє вирішальну роль у процесі тренування мережі, оновлюючи її параметри. На цьому рівні використовується спеціально обрана функція втрат для визначення розбіжностей між фактичними значеннями тренувальних даних та прогнозами мережі. Ця розбіжність використовується як основа для подальшого оновлення параметрів мережі у процесі тренування.

Правильний вибір функції втрат є критично важливим для ефективності тренування мережі. Він визначає, наскільки швидко та ефективно мережа буде навчатися, а також її здатність до точної оцінки результатів. Функція втрат об'єднує всі переваги та недоліки мережі в єдине числове значення, яке допомагає оцінювати та порівнювати різні варіанти моделей за їх ефективністю.

Для згорткових нейронних мереж у задачах класифікації часто використовують функцію перехресної ентропії, оскільки вона дозволяє ефективно порівнювати імовірності, що прогнозуються мережею, з фактичними класами вхідних даних. Функція логарифмічних втрат, яка є варіантом перехресної ентропії, використовується разом з активаційними функціями Softmax або сігмоїди для обчислення кінцевих прогнозів у задачах багатокласової та бінарної класифікації

відповідно. Функція перехресної ентропії може бути знайдена за формулою:

$$L(p, y) = -\sum_{n} y_n \ln p_n \, , n \in [1, N],$$

де p – отримані за допомогою мережі ймовірності належності даних до вибраних класів,

y – справжні значення для вхідних даних,

N – кількість класів.

Рис.1.14 Формула перехресної ентропії

Функції активації в згорткових нейронних мережах вносять нелінійність після згорткових, агрегаційних та інших шарів, формуючи остаточні вихідні дані мережі. Важливість цих функцій полягає у можливості мережі вчитися розпізнавати нелінійні закономірності, а також у збереженні нелінійних залежностей між різними рівнями мережі. Вони також контролюють вплив окремих нейронів на кінцеві результати.

Функції активації зазвичай обмежують або зменшують діапазон значень вхідних даних та мають бути диференційовними для ефективного застосування алгоритму зворотного поширення помилки. Вибір функції активації впливає на початкове налаштування параметрів мережі перед тренуванням.

Однією з часто використовуваних функцій активації є випрямлена лінійна одиниця (ReLU), яка сприяє швидшому навчанню мережі порівняно з іншими функціями активації. ReLU ефективно вносить нелінійність, знижуючи при цьому ризик перенавчання мережі, та дозволяє навчати складніші моделі ефективніше. Формула знаходження значення ReLU:

$$y = f(x) = \max(0, x),$$

де y – елемент поточного рівня,

x – елемент з вхідних даних.

Рис.1.15 Формула ReLu

Функція активації ReLU, що використовується в нейронних мережах, ефективно усуває негативні значення, призначаючи їм нульові значення, тоді як додатні значення проходять через неї без змін. Ця функція прискорює обчислення з апаратного погляду, оскільки полягає лише в порівнянні з нулем. Похідна ReLU дорівнює нулю для від'ємних значень та одиниці для додатних, спрощуючи таким чином зворотне поширення помилки в мережі.

Незважаючи на свої переваги, ReLU вимагає хоча б одного додатного початкового значення для ефективного навчання, а також створює ефект розрідження активації, оскільки не усі початкові значення активуються. Введення ReLU було натхненно функціонуванням візуальної кори мозку в обробці візуальної інформації.

Крім ReLU, іншими популярними функціями активації в нейронних мережах є гіперболічний тангенс (tanh) та сігмоїдна функція. Сігмоїдна функція зокрема використовується на останньому рівні мереж у задачах бінарної класифікації, оскільки її діапазон значень [0; 1] дозволяє апроксимувати імовірності належності вхідних даних до двох класів.

$$y = f(x) = \frac{1}{1 + e^{-x}}.$$

Рис.1.15 Формула Сігмоїдної функції

У контексті багатокласової класифікації, на останньому рівні згорткових нейронних мереж часто застосовується функція Softmax. Ця функція ефективно використовує всі значення з попереднього рівня мережі, дозволяючи обчислювати

імовірності приналежності до кожного класу.

$$y_i = f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \forall i, j \in (1, N),$$

де y_i – елемент поточного шару з індексом i,

x_i – елемент попереднього шару з індексом i,

N – кількість елементів на попередньому шарі.

Рис.1.16 Формула Softmax

Функція Softmax в згорткових нейронних мережах виконує нормалізацію експоненційних значень з попереднього рівня, перетворюючи їх у діапазон [0; 1] для обчислення імовірностей класів. Ця функція може бути модифікована шляхом зміни основи експоненціальної функції, що впливає на те, як значення з попереднього рівня враховуються при обчисленні імовірностей. Збільшення основи акцентує увагу на великих значеннях, тоді як зменшення робить вплив різних значень більш рівномірним. Також Softmax є стабільною до зміни всіх вхідних значень на одне і те ж стале число, забезпечуючи консистентність у визначенні імовірностей класів.

РОЗДІЛ 2 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ

2.1 Розгляд та аналіз алгоритмів штучного інтелекту для виявлення та відстеження об’єктів

Штучний інтелект - це галузь комп'ютерної лінгвістики та інформатики, що стрімко розвивається і зосереджена на розробці інтелектуальних машин, здатних виконувати завдання, які зазвичай вимагають людського інтелекту. Такі завдання варіюються від простих, таких як розпізнавання мови та зображень, до більш складних, таких як ігри та керування транспортними засобами.

Штучний інтелект (ШІ) - це галузь комп'ютерних наук, що займається розробкою інтелектуальних машин, які можуть виконувати завдання, що зазвичай вимагають людського інтелекту. Системи ШІ призначені для навчання на основі досвіду, розпізнавання закономірностей і прийняття рішень на основі вхідних даних. Ці системи можна навчити виконувати конкретні завдання, наприклад, розпізнавати зображення, розуміти природну мову або грати в ігри. Технології штучного інтелекту включають широкий спектр методів, зокрема машинне навчання, обробку природної мови, робототехніку та експертні системи. Метою досліджень у галузі штучного інтелекту є створення машин, які можуть міркувати, розуміти і навчатися, як люди, і використовувати ці здібності для покращення життя людей і вирішення складних проблем .

У більшості випадків алгоритм розв'язання проблеми заздалегідь невідомий. Оскільки філософія не вирішила питання про природу і статус людського інтелекту, то немає точного визначення цієї науки. Хоча було висунуто багато гіпотез щодо штучного інтелекту, таких як тест Тюрінга та гіпотеза Ньюелла-Саймонса, також не існує точного критерію, за яким комп'ютер набуває "інтелекту". Сьогодні існують різні підходи до розуміння проблем штучного інтелекту та створення інтелектуальних систем.

Ця наука пов'язана з нейронауками, включаючи когнітивну нейронауку, системну нейронауку та обчислювальну нейронауку. Як і інші комп'ютерні науки, вона використовує математичні інструменти. Особливо важливими є філософія та робототехніка.

У березні 2023 року деякі дослідники описали свою роботу над ChatGPT на основі немодифікованого GPT-4, відкритого лише для розробників OpenAI, як ранню і незавершену версію потужного штучного інтелекту (AGI) У середині квітня 2023 року Сем Альтман, Ілон Маск, Стів Возняк, Юваль Галалі та інші співробітники OpenAI оголосили про призупинення навчання на GPT-5 після письмових запитів від 50 000 осіб, у тому числі понад 1800 генеральних директорів і 1500 викладачів.

Згідно з повідомленням ООН, 31 жовтня 2023 року Генеральний секретар ООН Антоніу Гутерріш створив Консультативний орган високого рівня зі штучного інтелекту - групу експертів, які надаватимуть рекомендації щодо використання штучного інтелекту .

Існують різні способи побудови систем штучного інтелекту. Наразі існує чотири різні методи:

Логічний підхід. Основою для вивчення логічного підходу є алгебра логіки. Всі програмісти знайомі з цією алгеброю з моменту вивчення операторів IF. Алгебра логіки отримала подальший розвиток у вигляді логіки предикатів і була розширена за рахунок додавання предметних символів і відношень між ними. Крім того, кожна така машина має блок генерації мети, і система виводу намагається довести цю мету у вигляді теореми. Якщо мета досягнута, то набір використаних правил дає ланцюжок дій, необхідних для реалізації мети (такі системи також називають експертними). Потужність такої системи визначається можливостями генератора цілей і механізму доведення теорем. Для досягнення більшої виразності в логічних підходах використовують нові напрямки. Основною відмінністю в цій галузі є те, що істинність твердження може приймати проміжні значення, такі як "не знаю" (0,5), "пацієнт швидше живий, ніж мертвий" (0,75) і "пацієнт швидше мертвий, ніж

живий" (0,25) на додаток до "так"/"ні" (1/0). Цей підхід більше схожий на людське судження, оскільки тут рідко можна отримати відповідь "так" чи "ні"..

Структурний підхід. Структурні підходи стосуються спроб створення штучного інтелекту шляхом моделювання структур людського мозку. Однією з перших таких спроб був перцептрон Френка Розенблата. Основною моделюючою одиницею перцептрона є нейрон. Пізніше з'явилися інші моделі, більш відомі як нейронні мережі (НМ), та їх застосування - нейрокомп'ютер. Ці моделі відрізняються структурою окремих нейронів, топологією зв'язків між ними та алгоритмами навчання. На початку 2000-х років найвідомішими варіантами НМ були НМ зі зворотним поширенням, мережі Кохонена та мережі Хопфілда, серед інших - стохастичні нейронні мережі. У широкому сенсі цей підхід відомий як коннекціонізм. Відмінності між логічним і структурним підходами не настільки фундаментальні, як може здатися. Алгоритми, які спрощують і вербалізують нейронні мережі, перетворюють моделі структурного підходу в явні логічні моделі. З іншого боку, ще в 1943 році Уоррен Маккалох і Волтер Піттс показали, що нейронні мережі можуть реалізувати будь-яку функцію логічної алгебри.

Еволюційний підхід. При побудові систем ШІ з використанням цього підходу основна увага приділяється побудові початкової моделі та правил, необхідних для того, щоб ця модель змінювалася (еволюціонувала). Також моделі можуть бути побудовані по-різному, це можуть бути НМ, набір логічних правил або інші моделі. Потім, при включенні комп'ютера, на основі перевірки моделі вибирається найбільш підходяща модель і на її основі за різними правилами створюється нова модель. Серед еволюційних алгоритмів класичним вважається генетичний алгоритм.

Імітаційний підхід. Цей підхід є класикою кібернетики, одним з основних понять якої є чорний ящик. Об'єкт моделювання дійсно є чорною скринькою. Для нас важливо не те, які моделі знаходяться в чорному ящику і як вони себе поводять, а те, щоб наші моделі поводилися однаково в схожих ситуаціях. Тому тут моделюється ще одна характеристика людини. Це здатність копіювати те, що роблять інші, без необхідності розбивати це на базові операції або формальні

визначення поведінки. Ця особливість може заощадити багато часу, особливо на початку.

Гібридні інтелектуальні системи намагаються поєднати ці дисципліни. Спеціалізовані правила висновку генеруються нейронними мережами, тоді як грубі правила виводяться на основі статистичного навчання. Багатообіцяючий новий підхід, який також називають доповненням інтелекту, розглядає досягнення штучного інтелекту в процесі еволюційного розвитку як постійний ефект доповнення людського інтелекту за допомогою технологій.

2.1.1 Алгоритм YOLO

Нейронна мережа YOLO (You Only Look Once) відрізняється від інших нейронних мереж тим, що вона використовується лише один раз під час аналізу зображення перед тим, як проаналізувати всю площину зображення. Іншими словами, нейромережа ділить зображення на своєрідну сітку однакових комірок і для кожної частини зображення оцінює своєрідну обмежувальну область, оскільки саме в цій області з найбільшою ймовірністю може бути розміщений шуканий об'єкт.

Перевага цього підходу полягає в тому, що при аналізі всього зображення мережа враховує весь контекст інформації для виявлення і розпізнавання. Це означає, що, на відміну від мережі RetinaNet, ця мережа має бути швидшою, оскільки відсутній двонаправлений аналіз зображення. Наприклад, за словами розробника Джозефа Редмона, базова модель цієї мережі обробляє зображення в реальному часі зі швидкістю 45 кадрів на секунду. При цьому відомо, що мережа має труднощі з пошуком об'єктів з високою точністю, але це ненавчена.

YOLOv8, остання версія алгоритму "You Only Look Once", є значним кроком вперед у галузі комп'ютерного зору та обробки зображень. Ця версія, розроблена Ultralytics, запроваджує нові оптимізації та вдосконалення, які підвищують ефективність виявлення об'єктів. Основні характеристики YOLOv8 включають поліпшену точність, прискорення процесу висновків, підтримку різноманітних структур даних, адаптивне навчання, розширені методи аугментації даних та налаштовувану архітектуру. Ця модель є високоадаптивною та підходить для

широкого спектру застосувань, включаючи автономні транспортні засоби, системи спостереження, роздрібну торгівлю, медичну візуалізацію, сільське господарство та робототехніку.

Архітектура YOLOv8 базується на конволюційній нейронній мережі, поділеній на дві основні частини: "хребет" та "голову". "Хребет" мережі базується на модифікованій версії архітектури CSPDarknet53, що включає 53 конволюційні шари та використовує часткові з'єднання між різними етапами для покращення потоку інформації. "Голова" мережі складається з декількох конволюційних шарів, за якими слідують повнозв'язні шари, відповідальні за передбачення обмежувальних рамок, оцінки об'єктів та ймовірності класів для виявлених об'єктів. Однією з ключових особливостей YOLOv8 є використання механізму самоуваги у "голові" мережі, який дозволяє моделі зосереджуватися на різних частинах зображення та коригувати важливість різних ознак залежно від їхньої актуальності до завдання

У мережах YOLO розпізнавання об'єктів реалізовано як єдину регресійну задачу, яка відокремлює обмежувальні рамки, пов'язані з ймовірностями належності об'єктів на зображенні до різних класів. Іншими словами, існує прямий зв'язок між координатами пікселів зображення та обмежувальних рамок і ймовірністю передбачення класу об'єкта.

Крім того, результати можуть накладатися один на одного, якщо аналізується вся площина зображення. Для усунення цього мережа YOLO оснащена алгоритмом "не максимального видалення". Згідно з цим алгоритмом, знаходиться кадр з максимальним індексом достовірності, а інші найближчі кадри, розташовані поблизу максимального перетину з першим кадром, просто ігноруються.

2.1.2 Алгоритм DEEPSORT

DeepSORT - це алгоритм відстеження, який поєднує моделі виявлення об'єктів, такі як Mask R-CNN і SSD, з алгоритмами відстеження SORT (Simple Online and Real-

time Tracking). Цей підхід є розширенням стандартного SORT і дозволяє відстежувати людей у відеопотоках з високою точністю і стабільністю.

DeepSORT використовує нейронні мережі для виявлення об'єктів і може розпізнавати людей на зображеннях. Алгоритм SORT потім використовується для відстеження цих об'єктів у часі, тобто для визначення траєкторій їхнього руху. Основна ідея полягає у використанні унікального ідентифікатора для кожного відстежуваного об'єкта. Це дозволяє відстежувати та ідентифікувати окремих людей у всьому відеопотоці. Крім того, DeepSORT використовує фільтр Калмана для прогнозування майбутнього положення об'єкта на основі його поточного стану та швидкості. Це забезпечує підвищену точність відстеження, навіть якщо об'єкт тимчасово перебуває поза зоною видимості - приклади застосування DeepSORT включають відстеження людей у відеоспостереженні, аналіз поведінки, розпізнавання об'єктів і багато інших завдань, які вимагають точного відстеження об'єктів у часі.

2.1.3. Алгоритм SORT

Алгоритм SORT описується ключовими компонентами як:

1. Виявлення
2. Оцінка станів об'єктів у майбутні кадри,
3. Пов’язування поточного виявлення з існуючими об’єктами
4. Управління тривалістю життя відстежуваних об'єктів

Виявлення: якість виявлення необхідна для виконання завдання відстеження, отже, підхід використовує систему виявлення Faster Region CNN (FrRCNN).

Оцінка: після того, як виявлення виконано, потрібно виконати представлення та модель руху, яка використовується для розповсюдження ідентичності цілі в наступному кадрі. Отже, тепер потрібно перенести виявлення від поточного кадру до наступного за допомогою лінійної моделі постійної швидкості. Коли виявлення пов’язано з ціллю, виявлена обмежувальна рамка використовується для оновлення цільового стану, де компоненти швидкості вирішуються оптимально за допомогою

фільтра Калмана. Якщо виявлення не пов’язане з ціллю, її стан просто прогнозується без корекції за допомогою моделі лінійної швидкості.

Пов’язування поточного виявлення з існуючими об’єктами: при призначенні виявлення існуючим цілям геометрія обмежувальної рамки кожної цілі оцінюється шляхом прогнозування її нового розташування в поточному кадрі. Потім матриця вартості призначення обчислюється як відстань перетину через об’єднання (IOU) між кожним виявленням і всіма прогнозованими обмежувальними рамками від існуючих цілей. Завдання вирішується оптимально за допомогою угорського алгоритму. Крім того, для відхилення призначень встановлюється мінімальний IOU, коли виявлення перекривання цілі менше, ніж IOU min. Це добре працює, коли ціль закрита оклюзійним об’єктом, у цьому випадку виявляється лише оклюзійний об’єкт. Управління тривалістю життя відстежуваних об'єктів: коли об’єкти входять у зображення та залишають його, необхідно створювати або знищувати унікальні ідентифікатори відповідно. Для створення трекерів розглядається будь-яке виявлення з перекриттям менше IOU min як ознаку існування невідстежуваного об’єкта. Трекер ініціалізується з використанням геометрії обмежувальної рамки зі швидкістю, встановленою на нуль. Оскільки в цей момент швидкість не спостерігається, коваріація компонента швидкості ініціалізується великими значеннями, що відображає цю невизначеність. Крім того, новий трекер проходить випробувальний період, коли мета має бути пов’язана з виявленнями, щоб накопичити достатньо доказів, щоб запобігти відстеженню помилкових спрацьовувань. Доріжки припиняються, якщо вони не виявлені для кадрів T Lost. Це запобігає необмеженому зростанню кількості трекерів і помилок локалізації, викликаних передбаченнями протягом тривалого часу без виправлень від детектора.

2.2 Аналіз алгоритмів навчання штучного інтелекту з підсиленням

Навчання з підкріпленням - це розділ машинного навчання, натхненний поведінковою психологією, який вивчає проблему того, яка поведінка програмного агента в певному середовищі максимізує суму його винагород. Через загальність проблеми, вона також вивчається в багатьох інших дисциплінах, включаючи теорію

ігор, теорію управління, дослідження операцій, теорію інформації, оптимізацію на основі моделювання, мультиагентні системи, колективний інтелект, статистику та генетичні алгоритми. У літературі з дослідження операцій та управління один з напрямків навчання з підкріпленням називається наближеним динамічним програмуванням. Проблеми навчання з підкріпленням також вивчалися в теорії оптимального керування, але основна увага приділялася існуванню і характеристиці оптимальних рішень, а не аспектам навчання і апроксимації. В економіці та теорії ігор навчання з підкріпленням використовується для пояснення того, як виникає рівновага в умовах обмеженої раціональності.

У машинному навчанні середовище часто розглядається як марковський процес прийняття рішень, і в цьому контексті більшість алгоритмів навчання з підкріпленням використовують динамічне програмування. Основна відмінність між класичними методами та алгоритмами навчання з підкріпленням полягає в тому, що останні не потребують знань про МПВ і зосереджуються на великих МПВ, що робить строгі методи незастосовними .

Навчання з підкріпленням відрізняється від стандартного навчання під контролем тим, що правильні пари вхід-вихід ніколи не подаються, а неоптимальна поведінка явно не змінюється. Також акцент робиться на інтерактивній продуктивності, включаючи пошук балансу між розвідкою та використанням. Баланс між дослідженням і використанням у навчанні з підкріпленням найбільш детально вивчався на прикладі задачі про багаторукого бандита і скінчені МПВ.

Базова модель навчання з підкріпленням складається з:м

- множини станів середовища S;
- множини дій A;
- правил переходу між станами;
- правил, які визначають скалярну безпосередню винагороду;
- правил, які описують, що спостерігає агент.

Ці правила зазвичай стохастичні. Спостереження зазвичай передбачає скалярну миттєву винагороду, пов'язану з крайовими переходами. У багатьох дослідженнях також передбачається, що агенти спостерігають за поточним станом середовища, і в

цьому випадку вони вважаються повністю спостережливими, в іншому випадку - частково спостережливими. Іноді поведінка агентів обмежена (наприклад, вони не можуть витратити більше грошей, ніж у них є).

Агенти навчання з підкріпленням взаємодіють з навколишнім середовищем у дискретні моменти часу.

Коли поведінка агента порівнюється з поведінкою агента, який з самого початку поводиться оптимально, різниця в результатах призводить до поняття "горе".

Зауважте, що для того, щоб поводитися близько до оптимального, агенти повинні розуміти довгострокові наслідки своєї поведінки: щоб максимізувати майбутній дохід, вони повинні піти до школи зараз, навіть якщо негайна фінансова винагорода, пов'язана з цим, є негативною.

Тому навчання з підкріпленням особливо підходить для завдань, що передбачають компроміси між довгостроковими і короткостроковими винагородами. Навчання з підкріпленням успішно застосовується до найрізноманітніших завдань, включаючи управління розкладом роботизованих ліфтів, телекомунікації, нарди, шашки та Alpha Go.

Є два елементи, які роблять навчання з підкріпленням потужним: використання прикладів для оптимізації продуктивності та використання функціонального підходу для роботи з великими середовищами. Ці два елементи дозволяють застосовувати навчання з підкріпленням до великомасштабних середовищ у наступних ситуаціях:

- модель середовища є відомою, але аналітичний розв'язок відсутній;
- задано лише імітаційну модель середовища (предмет оптимізації на основі імітації)
- єдиним способом збирання інформації про середовище є взаємодія з ним.

Перші дві з цих задач можна вважати задачами планування (оскільки модель існує в тій чи іншій формі), тоді як останню можна вважати реальною задачею навчання. Однак, використовуючи методологію навчання з підкріпленням, обидві задачі планування можна перетворити на задачі машинного навчання.

2.2.1 Алгоритм DQN

Глибоке Q-навчання — це алгоритм навчання з підкріпленням, який поєднує Q-навчання з методами глибокого навчання, щоб вивчити оптимальну політику для прийняття рішень у складних середовищах. DQN, або Deep Q-Network, — це конкретна реалізація глибокого Q-навчання, яка використовує глибоку нейронну мережу для апроксимації функції дії-цінності. Основна відмінність між ними полягає в тому, що глибоке Q-навчання є загальною концепцією, а DQN — це конкретна архітектура та алгоритм для впровадження глибокого Q-навчання.

Агент DQN — це агент навчання з посиленням, який використовує Deep Q-Network для вивчення оптимальної політики для прийняття рішень у складних середовищах. Агент взаємодіє з середовищем, спостерігає за поточним станом і вибирає дії на основі виходу DQN. Агент отримує відгуки у вигляді винагород або штрафів і оновлює DQN, щоб з часом покращити його продуктивність.

Глибока мережа Q (DQN) — це архітектура нейронної мережі, яка використовується в навчанні з підкріпленням для апроксимації функції дії-цінності, яка оцінює очікувану кумулятивну винагороду за виконання певної дії в певному стані. DQN дозволяють агентам навчатися на вхідних даних великого розміру, таких як зображення, і вирішувати складні завдання, поєднуючи потужність глибокого навчання з алгоритмами навчання з підкріпленням, такими як Q-навчання.

DQN не є застарілим, але його було вдосконалено та розширено різними методами та алгоритмами. Дослідники розробили методи вирішення таких проблем, як упередження переоцінки, масштабованість і багатоцільові завдання. Деякі з цих покращень включають Double DQN, Dueling DQN і Prioritized Experience Replay. DQN залишається основоположною технікою в навчанні з підкріпленням, і його варіанти продовжують використовуватися в різних сферах застосування та досліджень.

DQN обробляють багатовимірні вхідні дані за допомогою глибоких нейронних мереж, які здатні вивчати складні ієрархічні представлення вхідних даних. Згорткові нейронні мережі (CNN) часто використовуються в DQN для обробки вхідних даних зображення, оскільки вони можуть автоматично вивчати особливості та шаблони з

необроблених піксельних даних. Ця здатність обробляти багатовимірні вхідні дані дозволяє DQN вирішувати складні завдання, з якими важко справляються традиційні алгоритми навчання з підкріпленням.

Існують два методи, які є дуже важливими для навчання DQN.

1. Відтворення досвіду: оскільки навчальні батчі в типовому налаштуванні ОП(RL) сильно корельовані та менш ефективні для обробки даних, це призведе до більш складної конвергенції для мережі. Одним із способів вирішення проблеми вибірки батчів є відтворення досвіду. По суті, батчі переходів зберігаються, а потім випадково вибираються з «пула переходів» для оновлення знань.

2. Окрема цільова мережа: цільова мережа Q має таку ж структуру, як і мережу, яка оцінює значення. Кожен крок C, відповідно до наведеного вище псевдокоду, цільова мережа набуває значення основної мережі. Таким чином, коливання стають менш сильними, що призводить до більш стабільних тренувань.

Практичні застосування DQN включають адаптивне керування трафіком, де новий алгоритм на основі DQN під назвою TC-DQN+ використовується для швидкого та надійного прийняття рішень щодо трафіку. У грі Wizard, що виконує трюк, DQN дають змогу самовдосконалюючим агентам долати виклики надзвичайно нестаціонарного середовища. Крім того, багатодоменні діалогові системи можуть отримати користь від методів DQN, як продемонстровано алгоритмом NDQN для оптимізації політик багатодоменного діалогу.

У прикладі компанії йдеться про використання DQN у робототехніці, де параметризовані дії поєднують дії високого рівня з гнучким керуванням. Метод MP-DQN значно перевершує попередні алгоритми з точки зору ефективності даних і продуктивності конвергентної політики для різних робототехнічних завдань.

Підсумовуючи, Deep Q-Networks продемонстрували великий потенціал у навчанні з підкріпленням, дозволяючи агентам навчатися складним завданням на основі багатовимірних вхідних даних. Вирішуючи такі проблеми, як упередження переоцінки та масштабованість, дослідники продовжують розширювати межі продуктивності DQN, що призводить до практичних застосувань у різних сферах, включаючи контроль трафіку, ігри та робототехніку .

2.2.2 Алгоритм PPO

Proximal Policy Optimization (PPO) — це потужний алгоритм навчання з підкріпленням, який набув популярності завдяки своїй ефективності та результативності у вирішенні складних завдань.

PPO вирішує проблему оновлення політик у навчанні з підкріпленням за допомогою сурогатної цільової функції для обмеження розміру кроку під час кожного оновлення політики. Цей підхід забезпечує стабільне та ефективне навчання, але все ще є деякі проблеми з нестабільністю продуктивності та неефективністю оптимізації. Дослідники запропонували різні варіанти PPO для вирішення цих проблем, такі як PPO-динамічний, CIM-PPO та IEM-PPO, які зосереджені на покращенні ефективності дослідження, використовуючи метрику, індуковану корентропією, та включаючи внутрішні модулі дослідження відповідно. Останні дослідження в області PPO призвели до розробки нових алгоритмів і методик. Наприклад, PPO-λ вводить адаптивний механізм відсікання для кращої продуктивності навчання, тоді як PPO-RPE використовує відносну розбіжність Пірсона для регулярізації. Інші варіанти, такі як PPO-UE та PPOS, зосереджені на дослідженні з урахуванням невизначеності та функціональних методах відсікання для покращення швидкості конвергенції та продуктивності.

Ось кілька варіантів алгоритму PPO, які були запропоновані для вирішення таких проблем, як нестабільність продуктивності та неефективність оптимізації. Деякі приклади включають PPO-dynamic, який фокусується на підвищенні ефективності розвідки; CIM-PPO, який використовує метрику, викликану коретропією; і IEM-PPO, який включає внутрішні модулі дослідження. Інші варіанти, такі як PPO-λ, PPO-RPE, PPO-UE та PPOS, вводять адаптивні механізми відсікання, методи регулярізації, дослідження з урахуванням невизначеності та функціональні методи відсікання для покращення продуктивності навчання та швидкості конвергенції.

Було показано, що PPO перевершує інші алгоритми навчання з підкріпленням у різних завданнях, таких як завдання безперервного контролю та ігровий штучний

інтелект. Наприклад, у фізичному симуляторі MuJoCo PPO досяг кращої ефективності вибірки та сукупної винагороди порівняно з іншими алгоритмами. У ігровому штучному інтелекті PPO виробляє моделі, подібні до алгоритму Advantage Actor-Critic (A2C), коли інші параметри контролюються. Загалом PPO вважається потужним і ефективним алгоритмом навчання з підкріпленням завдяки його здатності вирішувати проблеми оновлення політики та ефективності дослідження.
Практичні застосування PPO включають завдання безперервного контролю, ігровий штучний інтелект та розробку чат-ботів. Наприклад, PPO використовувався для навчання агентів у фізичному симуляторі MuJoCo, досягаючи кращої ефективності вибірки та сукупної винагороди порівняно з іншими алгоритмами. У сфері ігрового штучного інтелекту було показано, що PPO створює ті самі моделі, що й алгоритм Advantage Actor-Critic (A2C), коли інші налаштування контролюються. Крім того, PPO було застосовано до чат-ботів для чат-чату, демонструючи покращену стабільність і продуктивність порівняно з традиційними методами градієнта політики.

Одне дослідження компанії стосується OpenAI, який використовував PPO в різних проектах, включаючи розробку свого набору інструментів Gym для дослідження закріпленого навчання. OpenAI's Gym надає дослідникам платформу для тестування та порівняння різних алгоритмів навчання з підкріпленням, включаючи PPO, для широкого кола завдань.

Підсумовуючи, оптимізація проксимальної політики є багатообіцяючим алгоритмом навчання з підкріпленням, який отримав значні успіхи за останні роки. Вирішуючи проблеми оновлення політики та ефективності дослідження, PPO має потенціал для революції в різних сферах, включаючи робототехніку, ігровий штучний інтелект та обробку природної мови. Оскільки дослідження продовжують удосконалювати та вдосконалювати PPO, його застосування, безсумнівно, буде розширюватися, ще більше зміцнюючи його позицію як провідного алгоритму навчання з підкріпленням .

2.2.3 Алгоритм A3C

Asynchronous Advantage Actor-Critic (A3C) — це потужний алгоритм навчання з підкріпленням, який дозволяє агентам вивчати оптимальні дії в складних середовищах. Він працює шляхом асинхронного оновлення політики агента та функцій цінностей, що забезпечує швидше навчання та кращу продуктивність порівняно з традиційними алгоритмами навчання з підкріпленням. A3C успішно застосовувався для різних завдань, таких як відеоігри, керування роботами та оптимізація трафіку.

Advantage Actor-Critic (A3C) — це алгоритм навчання з підкріпленням, який поєднує в собі сильні сторони як методів актора-критики, так і методів навчання переваг. Підхід актор-критик використовує дві окремі нейронні мережі: актор, який вивчає оптимальну політику, і критик, який оцінює функцію цінності. Навчання переваги, з іншого боку, зосереджується на вивченні відносної цінності дій, а не їхньої абсолютної цінності. Поєднуючи ці два підходи, A3C може навчатися ефективніше та досягати кращої продуктивності в складних середовищах.

A3C, або Asynchronous Advantage Actor-Critic, — це алгоритм навчання з підкріпленням, який дозволяє агентам вивчати оптимальні дії, взаємодіючи з середовищем і отримуючи зворотний зв’язок у формі винагород або штрафів. Це популярний алгоритм у сфері навчання з підкріпленням завдяки його здатності швидко навчатися та добре виконувати широкий спектр завдань.

Основною перевагою A3C є його асинхронна природа, яка дозволяє швидше навчатися та покращувати продуктивність порівняно з традиційними алгоритмами навчання з підкріпленням. Завдяки асинхронному оновленню політики та функцій агента, A3C може досліджувати кілька шляхів у середовищі одночасно, що сприяє більш ефективному навчанню та підвищенню продуктивності у складних завданнях.

A3C працює за допомогою кількох паралельних агентів для дослідження середовища та вивчення оптимальної політики. Кожен агент взаємодіє зі своєю власною копією середовища, асинхронно оновлюючи свою політику та функції значень. Це паралельне дослідження дозволяє A3C навчатися ефективніше та

досягати кращої продуктивності порівняно з традиційними алгоритмами навчання з підкріпленням, які покладаються на одного агента.

A3C успішно застосовувався для широкого кола завдань, включаючи відеоігри, керування роботами, оптимізацію трафіку та адаптивні алгоритми бітрейту для служб доставки відео. У кожному з цих додатків A3C продемонстрував свою здатність швидко навчатися та добре працювати, що робить його цінним інструментом для вирішення складних проблем прийняття рішень у різних областях. Основна відмінність A3C від інших алгоритмів навчання з підкріпленням полягає в його асинхронній природі. У той час як традиційні алгоритми навчання з підкріпленням покладаються на одного агента для дослідження середовища та вивчення оптимальної політики, A3C використовує кілька паралельних агентів для одночасного дослідження середовища. Це паралельне дослідження дозволяє A3C навчатися ефективніше та досягати кращої продуктивності у складних завданнях.
Останні дослідження A3C були зосереджені на покращенні його надійності, ефективності та можливості інтерпретації. Наприклад, алгоритм Adversary Robust A3C (AR-A3C) вводить агент змагальності, щоб зробити процес навчання більш стійким до перешкод, що забезпечує кращу продуктивність у шумному середовищі. Інше дослідження пропонує гібридну реалізацію CPU/GPU A3C, яка значно прискорює процес навчання порівняно з реалізацією лише на CPU. Дослідники також досліджували допоміжні завдання, такі як Terminal Prediction (TP), щоб підвищити продуктивність A3C.

2.3 Аналіз технологій імітації середовища для навчання штучного інтелекту

Машинне моделювання стало популярним у всьому світі при дослідженні складних систем завдяки значним перевагам, які можуть бути отримані тими, хто використовує цей метод.

Машинне моделювання як метод чисельної механіки для розв'язання складних задач доцільно використовувати за таких умов:
- непридатність або відсутність аналітичних методів розв’язання задач;

- цілковита впевненість в успішному створенні імітаційної моделі, яка адекватно описує досліджувану систему (процес), зокрема в тому, що вдасться зібрати всю необхідну інформацію про модельовану систему (процес), забезпечивши вірогідну імітацію на ЕОМ реальних ситуацій (будувати імітаційну модель стохастичних процесів, коли не можна дістати опис потрібних характеристик випадкових величин і подій, – марний замір);
- можливість використати сам процес побудови імітаційної моделі для попереднього дослідження системи, що моделюється, з метою напрацювання рекомендацій щодо поліпшення умов її функціонування.

Можливі цілі побудови імітаційних моделей, призначених для вивчення проблем корпоративного управління, включають дослідження існуючих функціональних систем, аналіз гіпотетичних функціональних систем і проектування більш повних систем.

Однак, успішне вирішення вищезазначених завдань за допомогою імітаційних моделей можливе лише за наявності правильної моделі. Тому при дослідженні складних економічних систем за допомогою імітаційних моделей першим кроком є визначення адекватності моделі реальному об'єкту.

Якщо модель є невідповідною, дослідник ризикує отримати недостовірні результати або зробити на їх основі помилкові висновки. Тому оцінка адекватності моделі є важливим кроком у моделюванні, яке саме по собі є великим і складним завданням. Перевірка надійності моделі називається валідацією.

2.3.1 Алгоритм OpenAI Gym

OpenAI, некомерційна дослідницька лабораторія штучного інтелекту, заснована у 2015 році, розширює межі розвитку автономних систем. Лабораторія займається розширенням можливостей штучного інтелекту (ШІ) в області автономних систем і зробила значний внесок у досягнення цієї мети [9].

OpenAI досягла багатьох віх у розвитку автономних систем: У 2017 році вона випустила OpenAI Gym, інструментарій з відкритим вихідним кодом, який дозволяє розробникам створювати віртуальне середовище для тестування і вдосконалення

алгоритмів навчання з підкріпленням. OpenAI також розробила ряд нових продуктів, починаючи від автономних транспортних засобів і закінчуючи автономними роботизованими системами для цілого ряду застосувань, від роботизованих маніпуляторів до робототехнічних комплексів; в 2019 році компанія представила першу в світі роботизовану систему людського рівня, яка може самостійно пересуватися і взаємодіяти з об'єктами в складному середовищі.

OpenAI також розширює межі досліджень штучного інтелекту. У 2019 році вона запустила програму OpenAI Scholars, яка надає стипендії та наставництво студентам та аспірантам, що працюють над дослідницькими проектами в галузі ШІ. Лабораторія також випустила кілька інструментів з відкритим вихідним кодом, зокрема OpenAI Five - інструментарій, який дозволяє розробникам навчати ШІ-моделі та грати в стратегічні ігри в реальному часі, а також OpenAI Cloth - бібліотеку алгоритмів навчання з підкріпленням для розробки систем маніпулювання роботами.

Досягнення OpenAI в області автономних систем розширюють межі можливого в розробці штучного інтелекту. OpenAI і надалі залишатиметься лідером у розробці автономних систем, і його внесок, безсумнівно, вплине на сферу ШІ.
Навчання з підкріпленням, галузь штучного інтелекту, привертає великий інтерес в останні роки завдяки своєму потенціалу революціонізувати різні галузі. Від робототехніки та автономних транспортних засобів до фінансів та охорони здоров'я - алгоритми навчання з підкріпленням можуть дозволити машинам вивчати складні завдання та приймати рішення в динамічному середовищі. Одним з ключових гравців у цій галузі є OpenAI - дослідницька організація, що займається поширенням і розвитком дружнього штучного інтелекту на благо людства. OpenAI розробила інструментарій під назвою OpenAI Gym, який надає платформу для дослідників і розробників для експериментів і тестування алгоритмів навчання з підкріпленням. OpenAI Gym.

OpenAI Gym - це бібліотека з відкритим вихідним кодом, яка надає широкий спектр середовищ і завдань, в яких агенти навчання з підкріпленням можуть взаємодіяти і навчатися. Ці середовища включають класичні задачі управління,

настільні ігри та навіть ігри Atari, і надають алгоритмам багатий набір завдань для вирішення. OpenAI Gym надає стандартизований інтерфейс для цих середовищ, що дозволяє дослідникам легко порівнювати продуктивність різних алгоритмів і ділитися результатами з широкою спільнотою.

OpenAI Gym дозволяє користувачам розвивати навички навчання з підкріпленням. Це тип завдань з машинного навчання, де надається середовище з певними правилами та агент, який може діяти в цьому середовищі. Наприклад, гра Маріо - це середовище, а фігурка італійського водопровідника, керована машинним алгоритмом, - агент.

OpenAI Gym пропонує користувачеві великий набір середовищ і агентів на вибір, включаючи симуляції старих комп'ютерних ігор (включаючи класичні ігри Atari), віртуальних роботів, промислових машин і примітивних середовищ з простими фізичними законами.

Всі ці складні та різноманітні середовища представлені у вигляді відкритих і стандартизованих числових моделей, якими можна легко і просто керувати за допомогою скриптів Python. Агентами також можна легко керувати за допомогою програм на Python.

OpenAI Gym відіграє ключову роль у сприянні цим розробкам, надаючи дослідникам платформу для експериментів і розробки алгоритмів. Заохочуючи співпрацю та обмін знаннями всередині спільноти фахівців з навчання з підкріпленням, OpenAI Gym допомагає прискорити розробку нових методів і додатків, які можуть змінити наш світ.

Отже, майбутнє навчання з підкріпленням багатообіцяюче, і OpenAI Gym відіграє ключову роль у його розвитку. Оскільки дослідники продовжують вивчати потенціал алгоритмів навчання з підкріпленням, очікується, що кількість застосувань у різних галузях буде зростати. Надаючи стандартизовану платформу для експериментів і бенчмаркінгу, OpenAI Gym сприяє інноваціям у цій захоплюючій галузі та наближає нас до повної реалізації потенціалу ШІ.

2.3.2 TensorFlow Agents

TensorFlow - це програмна бібліотека з відкритим вихідним кодом для чисельних обчислень з використанням графів потоків даних. Вузли графа представляються у вигляді математичних операцій, а ребра - у вигляді багатовимірних масивів даних (тензорів), що передаються між вузлами.

TensorFlow створений і підтримується командою Google Brain в рамках дослідницької організації Google з машинного інтелекту в галузі ML і DL. В даний час TensorFlow випускається під ліцензією Apache 2.0 з відкритим вихідним кодом, програмні інтерфейси включають Python і C++, з запланованими API Java, GO, R і Haskell, а також хмари Google і Amazon. TensorFlow призначений для великомасштабного розподіленого навчання з 200 стандартними операціями, написаними на C++, включаючи математичні операції, операції з масивами, управління потоком і операціями управління станами. На відміну від інших DL-бібліотек (наприклад, Theano), які в першу чергу призначені для досліджень, TensorFlow призначена для використання як в дослідницьких системах, так і для розробки та виробництва програмного забезпечення. TensorFlow можна використовувати на CPU, GPU, мобільних пристроях і великих розподілених системах з сотнями вузлів. Він підтримується на системах. Крім того, існує TensorFlow Lite, полегшене рішення TensorFlow для мобільних і вбудованих пристроїв [TensorflowLite]. Це дозволяє публікувати рішення, орієнтовані на машинне навчання, на пристроях користувачів дуже швидко і з невеликим двійковим розміром, але підтримує лише обмежену кількість систем. Апаратне прискорення також підтримується за допомогою Android Neural Networks API.

Переваги:

- на сьогоднішній день найпопулярніший інструмент dl, з відкритим кодом, швидким залученням, який добре підтримується сильною промисловою компанією (google).
- потужна чисельна бібліотека для програмування потоків даних, яка є основою для досліджень та розробок dl.

- ефективно працює з математичними виразами, що включають багатовимірні масиви.
- дуже добре задокументована.
- обчислення на gpu / cpu, мобільні обчислення, висока масштабованість обчислень на різних машинах та величезні набори даних.
- вищий шар абстракції, ніж theano.

Недоліки:

- API нижчого рівня важко використовувати безпосередньо для створення моделей DL.
- кожен обчислювальний потік повинен бути побудований як статичний графік (хоча пакет Tensorflow Fold намагається полегшити цю проблему), а також відсутні символьни цикли.

Кожне обчислення в TensorFlow визначає орієнтовний граф, що складається з вузлів і ребер. Вузли - це операції/функції, а ребра - вхідні/вихідні дані та їх переповнення.

Входи і виходи TensorFlow називаються тензорами. Тензор - це не що інше, як багатовимірний масив, який визначає тип базових елементів при побудові графа.

Клієнтські програми, що використовують TensorFlow, створюють графіки потоків даних за допомогою затвердженої мови програмування (C або Python).

Операції - це функції, які мають імена і представляють абстрактні обчислення. Операції можуть мати атрибути, які повинні бути створені і передані в процесі побудови діаграми.

Одне з поширених застосувань атрибутів - створення поліморфних операцій.

Ядро - це реалізація операції на конкретному пристрої.

Клієнтський додаток взаємодіє з входом системи TensorFlow шляхом створення сеансу. Інтерфейс сеансу має високорівневий стиль для створення обчислювальних графів і буст-метод run(), який обчислює вихід одного вузла на графі трасування при подачі необхідних вхідних даних.

У більшості задач машинного навчання обчислювальні графи застосовуються ітеративно, а більшість звичайних тензорів не витримують роз'єднаних прогонів.

Змінні - це спеціальний тип операцій, який повертає ідентифікатори тензорних змінних, які можуть витримати кілька трас графа.

Параметри, що вивчаються, такі як ваги, переміщення тощо, переносяться до тензора, що зберігається у змінній.

Перший рівень TensorFlow складається з рівня пристроїв та мережевого рівня. Пристрійний рівень містить додатки для зв'язку з різними пристроями, такими як GPU, CPU і TPU операційної системи, на якій працює TensorFlow. Мережевий рівень містить додатки для зв'язку з різними машинами, що використовують різні мережеві протоколи в конфігурації Distributed Trainable.

Другий рівень TensorFlow містить основну реалізацію програми, яка в основному використовується для машинного навчання.

Третій рівень TensorFlow складається з розподілених майстрів і виконавців потоку даних. Розподілений майстер має можливість розподіляти робочі навантаження на різні пристрої в системі. Виконавець потоку даних, з іншого боку, виконує оптимізацію графіка потоку даних.

Наступний рівень розкриває всю функціональність у вигляді API, реалізованих на мові C. Мова C була обрана тому, що вона швидка, надійна і може працювати на будь-якій операційній системі.

П'ятий рівень забезпечує підтримку клієнтів на Python та C++.

Останній шар TensorFlow містить навчальні та вихідні бібліотеки, реалізовані на Python та C++.

Розширені можливості TensorFlow дозволяють розробникам створювати більш складні та потужні моделі машинного навчання. Ці можливості включають до себе:

- спеціальні операції : TensorFlow дозволяє розробникам визначати власні власні операції, які можна використовувати в графі обчислень, як і вбудовані операції. Це дозволяє розробникам додавати нові функції або оптимізувати наявні функції для конкретних випадків використання.

- спеціальні градієнти: TensorFlow також дозволяє розробникам визначати власні градієнти для своїх власних операцій. Це може бути корисним для реалізації складних або нестандартних алгоритмів зворотного поширення.

- TensorFlow Eager Execution : TensorFlow Eager Execution – це альтернативний режим виконання, який дозволяє розробникам запускати операції TensorFlow негайно, без попереднього створення графіка обчислень. Це може спростити розробку та налагодження, а також може бути корисним для інтерактивних сеансів і досліджень.
- TensorFlow Keras : TensorFlow Keras – це API високого рівня, який забезпечує зручний і послідовний інтерфейс для побудови та навчання моделей машинного навчання за допомогою TensorFlow. Він створений на основі TensorFlow і сумісний із TensorFlow Eager Execution і стандартним виконанням TensorFlow на основі сеансу.
- TensorFlow Probability : TensorFlow Probability – це бібліотека для імовірнісних міркувань і статистичного аналізу в TensorFlow. Він надає широкий спектр інструментів для імовірнісного моделювання, включаючи розподіли, біектори тощо.
- TensorFlow Model Optimization : інструментарій TensorFlow Model Optimization – це набір інструментів для оптимізації моделей машинного навчання для розгортання. Набір інструментів містить методи квантування, скорочення тощо.

Ці функції можуть допомогти розробникам створювати більш потужні та гнучкі моделі машинного навчання, а також можуть зробити процес розробки ефективнішим.

Ось список алгоритмів, які зараз підтримує TensorFlow:

- Класифікація – tf.estimator.LinearClassifier;
- Лінійна регресія – tf.estimator.LinearRegressor;
- Розширена класифікація дерев – tf.estimator.BoostedTreesClassifier;
- Регресія бустерного дерева – tf.estimator.BoostedTreesRegressor;
- Класифікація глибокого навчання – tf.estimator.DNNClassifier;
- Deep learning wipe і deep – tf.estimator.DNNLinearCombinedClassifier.

TensorFlow розроблений для полегшення розробки та виконання передових аналітичних додатків для таких користувачів, як аналітики даних, статистики та розробники моделей прогнозування.

Компанії всіх типів і розмірів широко використовують фреймворк для автоматизації процесів і розробки нових систем, що робить його особливо корисним для додатків з масовою паралельною обробкою даних, таких як нейронні мережі. Він також використовується для випробувань і тестування безпілотних автомобілів.

Як і слід було очікувати, материнська компанія Google також використовує TensorFlow для внутрішньої обробки, наприклад, для покращення пошукової функціональності своєї пошукової системи та забезпечення роботи таких додатків, як автоматична генерація відповідей на електронні листи, класифікація зображень та оптичне розпізнавання символів.

Однією з переваг TensorFlow є те, що він забезпечує абстракцію. Це означає, що розробники можуть зосередитися на загальній логіці програми, а фреймворк обробляє дрібні деталі. Це також корисно для розробників, яким потрібно налагоджувати і самоаналізувати свої TensorFlow-додатки.

TensorFlow є потужним інструментом для широкого спектру завдань машинного навчання і використовується в багатьох реальних додатках. Деякі приклади використання TensorFlow:

- класифікація зображень : TensorFlow використовувався для завдань класифікації зображень, таких як ідентифікація об’єктів на зображеннях або класифікація зображень за різними категоріями. Згорткові нейронні мережі (CNN) зазвичай використовуються в TensorFlow для завдань класифікації зображень;
- обробка природної мови (NLP): TensorFlow використовувався для широкого спектру завдань NLP, таких як переклад мови, підсумовування тексту та аналіз настроїв. Рекурентні нейронні мережі (RNN) і трансформатори зазвичай використовуються в TensorFlow для завдань NLP;
- аналіз часових рядів: TensorFlow можна використовувати для аналізу даних часових рядів, таких як ціни на акції, погодні дані або дані датчиків. Рекурентні нейронні мережі (RNN) і мережі довгострокової короткочасної пам’яті (LSTM) зазвичай використовуються в TensorFlow для аналізу часових рядів;
- генеративні моделі : TensorFlow використовувався для навчання генеративних моделей, таких як Generative Adversarial Networks (GANs) і Variation Autoencoders

(VAEs), які можна використовувати для таких завдань, як генерація зображень, передача стилів тощо;

- Reinforcement Learning: TensorFlow можна використовувати для навчання агентів Reinforcement Learning (RL), які можна використовувати для таких завдань, як ігри, робототехніка тощо;

- виявлення аномалій: TensorFlow можна використовувати для навчання моделей для виявлення аномалій, ідентифікації шаблонів або подій у даних, які відрізняються від нормальної поведінки, наприклад, у безпеці мережі чи виробництві .

Ці приклади дають уявлення про широкий спектр програм, для яких можна використовувати TensorFlow, і він постійно використовується в нових і захоплюючих способах.

TensorFlow - це потужна і гнучка платформа машинного навчання з відкритим вихідним кодом, розроблена компанією Google за допомогою багатьох учасників з відкритим вихідним кодом. TensorFlow представляє обчислення за допомогою орієнтованого графа, що складається з вузлів і ребер, де вузли - це процеси і функції, а ребра - вхідні і вихідні потоки; TensorFlow розроблений таким чином, щоб бути масштабованим і гнучким і може працювати на різних пристроях і в розподілених середовищах. Платформа включає розширені функції, такі як кастомні операції, кастомні градієнти та TensorFlow Eager Execution, які дозволяють розробникам створювати більш складні та потужні моделі машинного навчання. TensorFlow можна використовувати для класифікації зображень, обробки природної мови, аналізу часових рядів, генеративного моделювання, навчання з підкріпленням, виявлення аномалій і застосовується в широкому спектрі реальних додатків.

Важливо відзначити, що TensorFlow - це платформа, яка постійно розвивається, і регулярно випускаються нові функції та оновлення. Також важливо пам'ятати, що TensorFlow - це лише одна з багатьох доступних платформ машинного навчання, і найкращий вибір для конкретного проекту буде залежати від специфічних вимог і обмежень проекту .

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ АГЕНТІВ ДЛЯ УПРАВЛІННЯ МОДУЛЕМ ПОШУКУ, СПОСТЕРЕЖЕННЯ ТА НАВЕДЕННЯ НА ОБ’ЄКТ

3.1 Навчання штучного інтелекту для трекінгу об’єктів

У сучасному світі комп'ютерного зору, швидке та точне розпізнавання об'єктів у відеопотоках стає надзвичайно важливим завданням, яке знаходить застосування у різноманітних сферах - від систем безпеки до автономних транспортних засобів. В цій дипломній роботі я фокусуюся на розробці програмного забезпечення, що використовує алгоритм YOLO (You Only Look Once) для детектування та класифікації об'єктів у відеопотоці.

Алгоритм YOLO, що є однією з передових методик у галузі комп'ютерного зору, дозволяє виконувати детектування об'єктів майже в реальному часі, що є критично важливим для багатьох застосувань. Головна особливість YOLO полягає у його здатності обробляти весь зображення за один раз, що значно підвищує швидкість і ефективність порівняно з іншими методами, які аналізують окремі частини зображення.

Перед початком реалізації, важливо вибрати відповідний інструментарій та платформу. Вибір буде базуватися на кількох критеріях, включаючи сумісність з алгоритмом YOLO, здатність обробляти великі обсяги даних і підтримку в реальному часі. Після цього, наступним кроком є адаптація і оптимізація алгоритму для наших конкретних потреб. Це включає налаштування параметрів моделі, вибір підходящих датасетів для тренування, і регулювання алгоритму для максимізації його точності та швидкості.

Навчання моделі - це критичний етап. Важливо зібрати досить великий і різноманітний набір даних, що дозволить моделі ефективно вчитися і адаптуватися до різних сценаріїв детектування. Тестування та валідація системи на цьому етапі допоможуть виявити будь-які слабкі сторони та внести необхідні корективи.

Останнім кроком є інтеграція системи з реальним відеопотоком. Це включає розробку інтерфейсу для взаємодії з відеокамерами, а також налаштування системи

для обробки відеоданих в реальному часі. Ефективна інтеграція є ключовою для успішного застосування розробленої системи в практичних сценаріях.

3.1.1 Формування дата сету

У розвитку та вдосконаленні нейронних мереж ключовим аспектом є якість та різноманітність даних, на яких ці мережі навчаються. Це особливо важливо для алгоритмів комп'ютерного зору, таких як YOLO, які призначені для швидкого та точного детектування об'єктів у реальному часі. У рамках мого проекту, де метою є виявлення мішеней у відеопотоках, створення ефективного датасету є вирішальним. Такий датасет не тільки забезпечує навчання нейронної мережі, але й впливає на її здатність точно розпізнавати об'єкти у різних умовах.

У рамках мого дипломного проекту, який зосереджений на розробці системи детектування мішеней за допомогою алгоритму YOLO, я взяв на себе завдання зібрати великий та різноманітний набір фотографій. Моя мета полягала у створенні датасету з 1000 унікальних зображень, який би містив мішені у різних умовах, щоб модель ефективно навчилася розпізнавати їх у реальних сценаріях.

Рис.3.1 Приклади фотографій для формування датасету

При формуванні датасету для навчання алгоритму YOLO важливо дотримуватися правила різноманітності умов зйомки. Оптимальний підхід полягає у виборі локацій, які можуть демонструвати широкий спектр умов - від зйомок на відкритому повітрі до

зйомок в закритих приміщеннях. Це необхідно для того, щоб забезпечити різноманітність контекстів, в яких мішені можуть з'являтися. Такий вибір локацій забезпечує, що навчальна модель буде здатна адекватно функціонувати в широкому діапазоні реальних умов та середовищ, збільшуючи її універсальність та ефективність.

Рис.3.2 Приклади фотографій в різних локаціяї

Після визначення локацій, наступним важливим кроком у процесі створення датасету є забезпечення різноманітності умов освітлення. Ефективний підхід полягає у проведенні фотозйомок у різний час доби, щоб відобразити мішені в умовах від яскравого сонячного світла до штучного освітлення. Важливо включити фотографії, зроблені під час різних погодних умов, таких як ясний день, хмарно або навіть дощова погода. Це додає різноманітності контрастності та яскравості зображень в датасеті.

Таке різноманіття освітлення та погодних умов є ключовим для створення датасету, що відображає реальні умови, з якими нейронна мережа може зіткнутися під час використання. Включення зображень із широким діапазоном освітлення забезпечує, що модель буде здатна адекватно розпізнавати мішені в різних освітлювальних умовах, підвищуючи її універсальність та надійність.

Рис.3.3 Приклади фотографій для формування датасету з різним освітленням

Після забезпечення різноманітності освітлення, наступним важливим аспектом у створенні датасету є різноманітність кутів та відстаней зйомки. Необхідно варіювати перспективи, включаючи зйомку з близької відстані, з середньої відстані, та здалеку, а також з різних висот. Така різноманітність не тільки додає глибини датасету, але й є критично важливою для навчання моделі адекватно розпізнавати мішені незалежно від їх розміру та положення у кадрі.

Різноманітність кутів зйомки дозволяє моделі розуміти та адаптуватися до різних перспектив, з яких мішені можуть бути зафіксовані. Це особливо важливо в реальних умовах, де мішені можуть з'являтися під різними кутами та на різних відстанях від камери. Ефективне тренування моделі на таких зображеннях гарантує, що система буде здатна точно ідентифікувати цілі в різних ситуаціях.

Рис.3.4 Приклади фотографій для формування датасету з різними відстанню

Після встановлення різноманітності кутів та відстаней зйомки, наступним критичним аспектом у формуванні датасету є уважний вибір фону для фотографій. Різноманітність фонів, від простих до складних, є необхідною для створення фотографій з різним рівнем контрастності між мішенню та фоном. Це дозволяє навчити модель ефективно розрізняти мішені від фону та ігнорувати нерелевантні об'єкти.

Вибір фону, який забезпечує різні рівні контрастності, важливий для забезпечення, що нейронна мережа не тільки розпізнає мішені, але й ефективно відокремлює їх від фонових елементів у різних умовах. Фотографії, які включають мішені на фоні складних текстур або взаємодіючих кольорів, допомагають моделі розвивати здатність до глибшого аналізу зображень та точнішого виявлення об'єктів.

Одним з ключових аспектів у процесі створення датасету для навчання нейронної мережі є забезпечення різноманітності кількості мішеней, які зображені на кожній фотографії. Це необхідно для того, щоб модель могла адекватно впоратися з різними сценаріями, які можуть виникнути в реальних умовах використання системи.

Фотографії з однією мішенню є базовими і надають моделі можливість зосередитися на деталях окремого об'єкта. Однак, важливо також включити зображення з кількома мішенями. Це може включати фотографії, на яких мішені знаходяться на різних відстанях від камери або розташовані під різними кутами. Такий підхід дозволяє навчити систему одночасно розпізнавати та розрізняти кілька мішеней, що є критично важливим для багатьох практичних застосувань.

Включення зображень з різною кількістю мішеней забезпечує, що модель буде здатна точно працювати в умовах, де можуть з'являтися кілька об'єктів одночасно. Це підвищує гнучкість та адаптивність системи, готуючи її до ефективного використання в різноманітних ситуаціях.

Рис.3.5 Приклади фотографій для формування датасету з різною кількістю цілей

Завершальним етапом у процесі формування датасету є ретельний контроль якості зібраних фотографій. Це необхідно для забезпечення, що всі зображення, які входять до датасету, відповідають високим стандартам якості. Контроль якості включає перевірку кожного зображення на чіткість, адекватність освітлення та видимість мішені. Важливо, щоб кожне фото було чітким, добре освітленим і щоб мішені були легко розрізненні.

Фотографії, які не відповідають цим критеріям, повинні бути відхилені. Це може означати відкидання зображень із розмитими або погано освітленими мішенями, а також фотографій, де мішені важко відрізнити від фону. Цей крок забезпечує, що датасет містить тільки якісні зображення, що підвищує ефективність навчання нейронної мережі.

3.1.1 Розмітка навчальних даних

Розмітка зображень є вирішальним етапом у підготовці навчальних даних для комп'ютерного зору та машинного навчання. Вона забезпечує необхідну основу, на якій базується навчання моделей для точного виявлення та класифікації об'єктів.

Процес розмітки вимагає детального та уважного підходу, оскільки якість розмітки безпосередньо впливає на ефективність і точність майбутніх моделей машинного навчання.

У процесі розмітки дані анотуються вручну, що включає ідентифікацію об'єктів на зображенні та відзначення їх меж. Найпоширенішим методом анотації є створення так званих "bounding boxes" - прямокутників, які окреслюють контури об'єктів. Цей метод дозволяє легко ідентифікувати та класифікувати об'єкти на зображенні. Для кожного об'єкта анотація містить інформацію про його розташування та розміри, а також може включати додаткові метадані, такі як клас об'єкта або унікальні ідентифікатори.

Після визначення об'єктів на зображенні та їх розмітки, наступним етапом є перевірка анотацій. Цей процес включає детальний огляд кожного зображення для забезпечення точності та консистентності анотацій. Помилки в анотаціях, такі як неточне визначення меж об'єктів або невірна їх класифікація, можуть призвести до неправильного навчання моделей, що в свою чергу вплине на їх здатність ефективно виконувати завдання виявлення та класифікації об'єктів.

В контексті використання програмного забезпечення для розмітки зображень, makesense.ai є одним із провідних інструментів у цій галузі. Завдяки своїй інтуїтивно зрозумілій інтерфейсній панелі та широкому спектру функціональних можливостей, makesense.ai дозволяє користувачам ефективно та точно анотувати зображення.

Процес роботи в makesense.ai починається з завантаження зображень. Користувачі можуть легко імпортувати зображення через інтерфейс перетягування або вибору файлів. Після завантаження зображень, вони відображаються у головному робочому просторі, де можна приступити до процесу анотації.

Makesense.ai пропонує різні інструменти для анотації, включаючи інструменти для створення bounding boxes, полігонів, ліній та точок. Користувачі можуть вибирати відповідний інструмент в залежності від типу об'єктів, які потребують анотації на конкретному зображенні. Наприклад, для анотації автомобілів або пішоходів на зображенні вулиці зазвичай використовуються bounding boxes, в той

час як для більш складних форм, таких як дерева або різноманітні предмети природи, можуть бути використані полігональні анотації.

Під час процесу анотації користувачі вручну визначають межі кожного об'єкта, використовуючи обраний інструмент, і присвоюють кожному об'єкту відповідну категорію або клас. Цей процес може бути часомістким, але він є критично важливим для забезпечення високої якості навчальних даних.

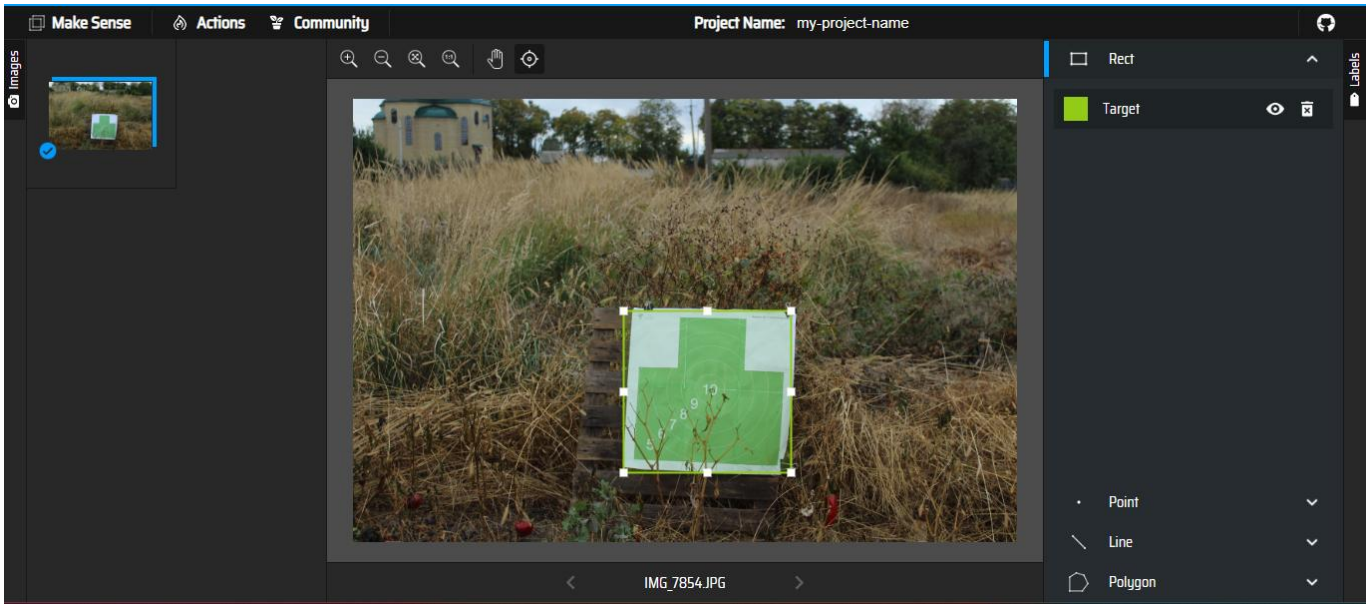


Рис.3.6 Інтерфейс Makesense.ai

Після завершення процесу анотації, користувачі переходять до етапу перевірки та виправлення анотацій. Це включає перегляд кожного зображення та перевірку точності анотацій. У разі виявлення помилок або неточностей, анотації можуть бути відкориговані. Цей етап є важливим для забезпечення консистентності та надійності навчальних даних.

Нарешті, після завершення розмітки та перевірки, анотовані дані експортуються з makesense.ai для подальшого використання в тренуванні моделей машинного навчання. Інструмент дозволяє експортувати дані у різних форматах, включаючи популярні формати, які використовуються в галузі машинного навчання. Зазвичай експортовані дані містять важливу інформацію про координати анотацій і класифікацію об'єктів на кожному зображенні. Ці дані стають ключовими для навчання та розвитку моделей комп'ютерного зору, забезпечуючи їм необхідну

інформацію для визначення та класифікації об'єктів у різноманітних сценаріях та умовах.

Процес експорту в makesense.ai включає кілька кроків, що дозволяють забезпечити правильне та ефективне використання анотованих даних. Користувачі вибирають відповідний формат файлу для експорту, в залежності від вимог їхнього проекту та використовуваної системи машинного навчання. Це може включати формати, такі як YOLO, XML або CSV, кожен з яких має свої особливості та переваги для різних типів проектів.

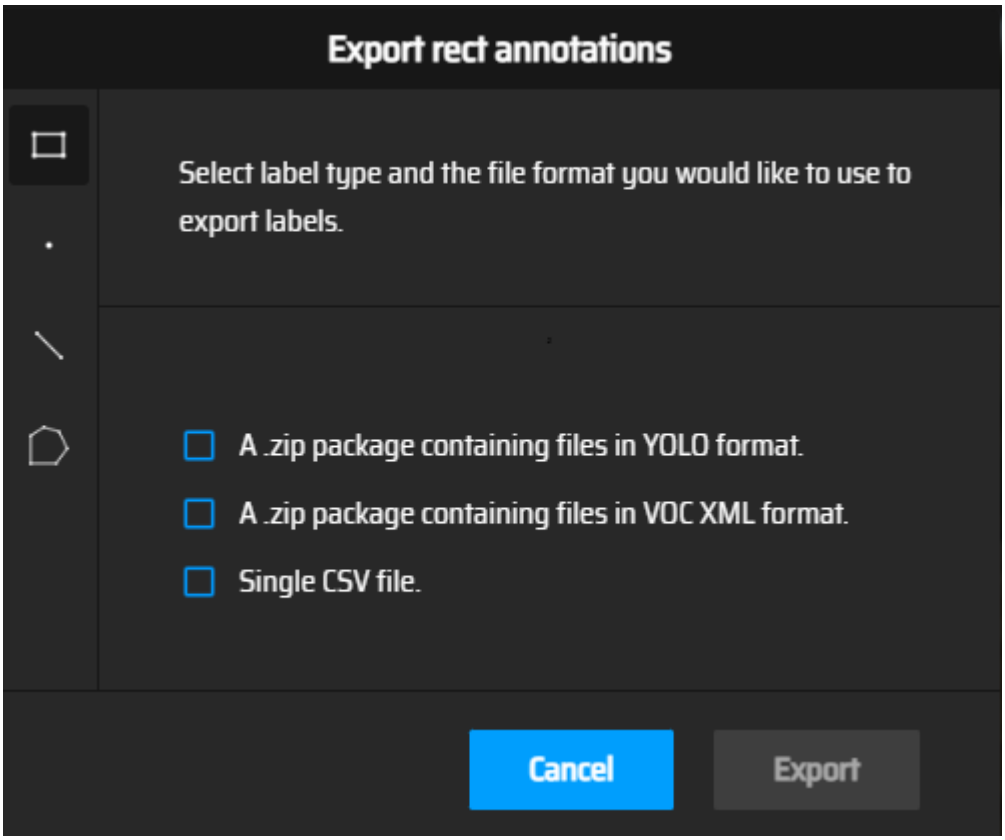


Рис.3.7 Інтерфейс експорту makesense.ai

Важливість розмітки даних у процесі розвитку та навчання моделей машинного навчання не може бути переоцінена. Якісно підготовлені та точно анотовані дані забезпечують міцну основу для створення надійних та ефективних систем комп'ютерного зору.

Навчання Моделі YOLO

Навчання моделі YOLO є фундаментальним процесом у розвитку ефективних систем комп'ютерного зору. Цей процес вимагає ретельного планування та виконання кроків, спрямованих на оптимізацію моделі для конкретних задач виявлення та класифікації об'єктів. Першим та важливим етапом у цьому процесі є ініціалізація ваг моделі.

Ініціалізація ваг моделі YOLO відіграє ключову роль у процесі її навчання, встановлюючи початкові умови, які значною мірою впливають на здатність моделі до адаптації та вивчення. Важливо зазначити, що вибір методу ініціалізації ваг залежить від конкретних цілей та умов задачі.

Підхід до випадкової ініціалізації полягає у присвоєнні вагам випадкових значень з певного розподілу, що дозволяє кожному нейрону в мережі розпочинати з унікального стану. Цей метод сприяє різноманітності відповідей нейронів та запобігає ситуації, коли всі нейрони навчаються однаково. Випадкова ініціалізація є особливо корисною у ситуаціях, де модель повинна розпізнавати унікальні або рідкісні особливості в даних.

З іншого боку, використання переднавчених ваг включає ініціалізацію моделі вагами, отриманими з інших, схожих задач. Цей метод може значно скоротити час навчання та покращити загальну ефективність, оскільки модель вже має базове розуміння певних візуальних патернів і структур. Переднавчені ваги особливо корисні у випадках, коли доступний датасет для навчання є обмеженим або коли задача вимагає великого обсягу даних для ефективного вивчення.

У кінцевому підсумку, вибір між випадковою ініціалізацією та використанням переднавчених ваг залежить від специфіки задачі та доступних ресурсів. Обидва методи мають свої переваги та можуть бути ефективно використані для досягнення оптимальних результатів у навчанні моделі YOLO.

YOLOv8 включає кілька варіантів моделей, кожна з яких має різні характеристики та оптимізована для певних задач. Моделі включають YOLOv8n (Nano), YOLOv8s (Small), YOLOv8m (Medium), YOLOv8l (Large) і YOLOv8x

(XLarge).

Модель	розмір (пікселі)	mAP val 50-95	Швидкість ЦП ONNX (мс)	Швидкість A100 TensorRT (мс)	параметри (М)	FLOPs (В)
YOLOv8n	640	37.3	80.4	0,99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
ЙОЛОв8м	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257,8

Рис.3.8 Преднавчені моделі YOLO

Основні параметри, які розрізняють ці моделі, включають:

Розмір моделі: Визначає обсяг пам'яті, який займає модель. Більші моделі мають більше параметрів та потенційно можуть бути точнішими, але вимагають більше ресурсів для обробки.

mAP (mean Average Precision) на валідації: Цей показник вимірює точність моделі у виявленні об'єктів. Вищий mAP означає кращу здатність моделі точно класифікувати та локалізувати об'єкти.

Швидкість: Швидкість, з якою модель обробляє зображення, важлива для реальних застосувань, таких як відеоспостереження або автономні транспортні засоби.

TensorRT: Це платформа оптимізації моделей глибокого навчання для вирішення задач в реальному часі, що забезпечує прискорення висновків.

Кількість параметрів: Це загальна кількість ваг у моделі, що впливає на її розмір та складність.

FLOPs (Floating Point Operations Per Second): Міра обчислювальної складності моделі. Більша кількість FLOPs вказує на більшу обчислювальну вимогливість моделі.

У моїй дипломній роботі було вирішено використовувати переднавчену модель YOLOv8s. Цей вибір був зумовлений перевагами, які переднавчена модель пропонує, включаючи зменшений час на навчання та покращену первинну точність.

YOLOv8s, як частина оновленої серії YOLO, забезпечує високу продуктивність при обробці відеопотоків, що є важливим для задач, де потрібне швидке виявлення об'єктів у реальному часі. Вибір переднавченої моделі дозволив ефективно використовувати доступні ресурси, забезпечуючи оптимальне співвідношення швидкості та точності для моєї дослідницької роботи.

Після успішної ініціалізації ваг моделі YOLOv8s, наступним важливим етапом є оптимізація та налаштування гіперпараметрів, що має критичне значення для ефективності процесу навчання. Цей етап включає в себе регулювання ключових параметрів, таких як швидкість навчання, розмір пакету (batch size), а також визначення кількості епох.

Швидкість навчання визначає, з якою швидкістю модель оновлює свої ваги у відповідь на помилки, знайдені під час навчання. Занадто висока швидкість може призвести до нестабільності навчання, тоді як занадто низька - до повільного прогресу. Розмір пакету впливає на кількість даних, оброблених за одну ітерацію, і є важливим для балансування між ефективністю обчислень та точністю навчання. Кількість епох визначає, скільки разів весь навчальний датасет буде пройдено під час тренування, що впливає на здатність моделі до узагальнення.

Ці гіперпараметри потрібно ретельно налаштувати, щоб забезпечити оптимальне навчання моделі, особливо в контексті виявлення об'єктів у відеопотоках. Вдале налаштування забезпечує ефективне використання ресурсів та високу точність моделі. У моїй роботі я встановив власні значення гіперпараметрів, які відповідають моїм конкретним вимогам та умовам:

```
# Load the model.
model = YOLO('C:\\Users\Dima\\Desktop\\studyFoldert\\nau\\Ochka\\diplo\\runs\\detect\\yolov8n_new_must_work\\weights\\last.pt')

# Training.
results = model.train(
    data='C:/Users/Dima/Desktop/studyFoldert/nau/Ochka/diplo/dataset.yaml',
    imgsz=1152,
    epochs=25,
    batch=6,
    name='yolov8n_new_must_work_new_29.11_top')
```

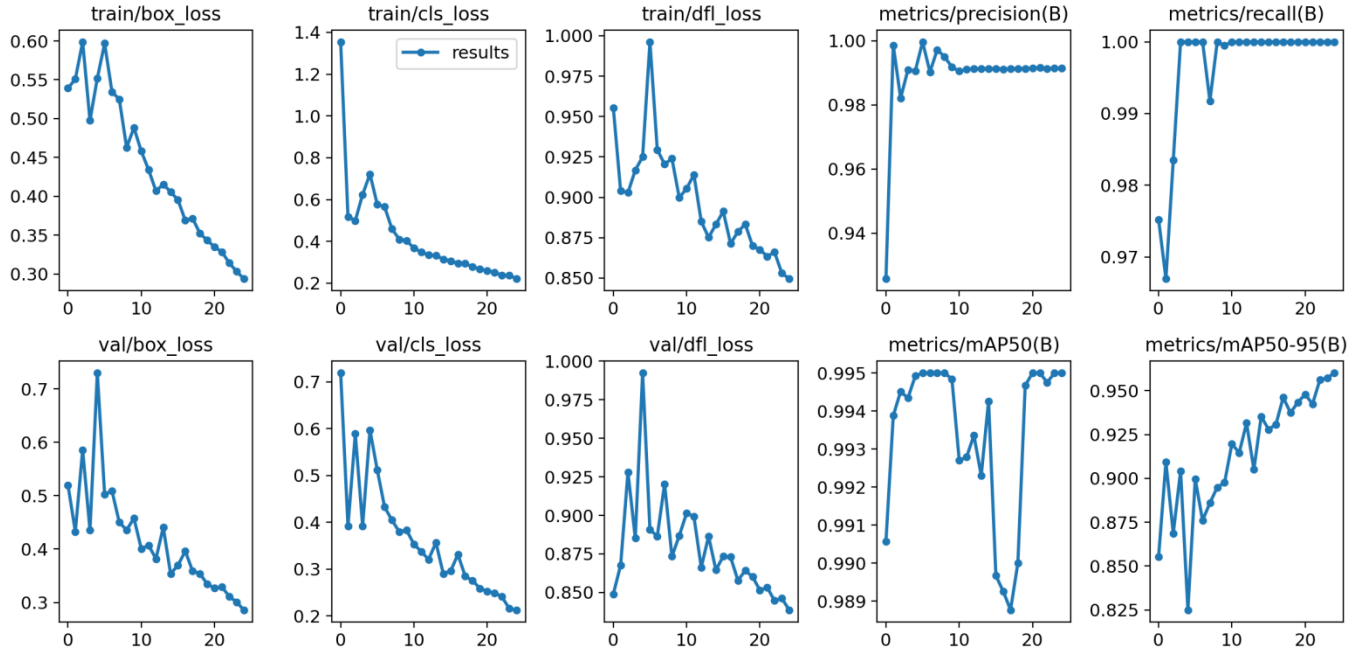
Рис.3.9 Ініціалізація гіперпараметрів

Процес навчання моделі YOLOv8s, з визначеними параметрами,

включає декілька ключових етапів. Після налаштування гіперпараметрів, модель піддається ітеративному процесу навчання, де вона обробляє навчальні дані у пакетах визначеним. Протягом кожної епохи, модель оптимізує свої ваги, покращуючи здатність точно виявляти та класифікувати об'єкти на зображеннях.

На кожному кроці тренування, використовуючи функції втрат, визначається, наскільки добре модель виконує задачу виявлення. Після завершення епох, модель проходить оцінку на валідаційному наборі даних, щоб перевірити її здатність до узагальнення та виявлення об'єктів на нових даних.

Після завершення навчання моделі YOLOv8s важливо проаналізувати результати, щоб зрозуміти її ефективність та здатність до виявлення об'єктів на зображеннях. Результати навчання можуть бути оцінені за допомогою різних метрик та графіків. Ось метрики які оцінюють навчання моделі:



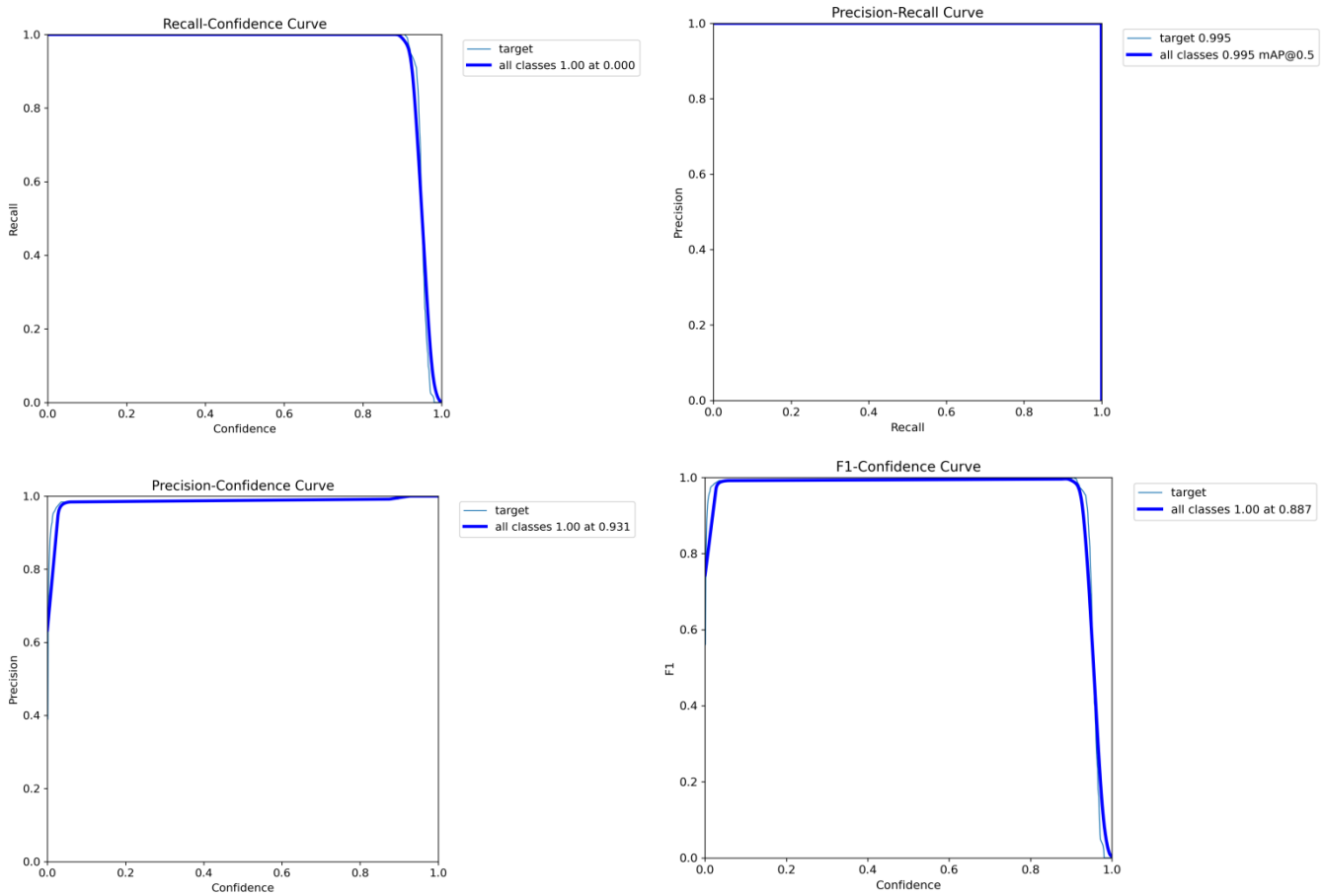


Рис.3.10 Результати навчання моделі YOLO

Розглянемо кожну з наведених метрик:

box_loss (Втрати локалізації об'єктів): Ця метрика вимірює втрати, пов'язані з точністю визначення положення об'єктів на зображеннях. Вона визначає рівень неточності в локалізації об'єктів. Зменшення значення box_loss свідчить про поліпшення у точності визначення положення об'єктів моделлю.

cls_loss (Втрати класифікації): Ця метрика вимірює втрати, пов'язані з правильністю класифікації об'єктів. Вона відображає, наскільки точно модель класифікує об'єкти на зображеннях. Зменшення значення cls_loss свідчить про покращення у правильності класифікації.

dfl_loss (Дефокус-втрати): Ця метрика вимірює якість прогнозів щодо дефокусу об'єктів. Вона дозволяє визначити, наскільки модель вірно передбачає ступінь дефокусу об'єктів. Зменшення значення dfl_loss вказує на поліпшення у визначенні фокусу об'єктів.

precision (B) (Точність виявлення об'єктів): Ця метрика визначає точність виявлення об'єктів (B) моделлю. Вона вказує, який відсоток об'єктів, виявлених моделлю, був правильно класифікований. Високе значення precision (B) вказує на високу точність виявлення об'єктів.

recall (B) (Відновлення виявлення об'єктів): Ця метрика визначає, який відсоток об'єктів (B) був виявлений моделлю серед усіх існуючих об'єктів цього класу. Вона вказує на здатність моделі виявляти всі наявні об'єкти даного класу. Високе значення recall (B) вказує на високу повність виявлення об'єктів.

Map50 (B) (Середня точність виявлення на 50% порозі впевненості): Ця метрика визначає середню точність виявлення об'єктів (B) при 50% порозі впевненості. Вона показує, наскільки правильно модель виявляє об'єкти при цьому рівні впевненості.

Map50 (B)-95 (Середня точність виявлення на 50%-95% порозі впевненості): Ця метрика вимірює середню точність виявлення об'єктів (B) на діапазоні від 50% до 95% порогу впевненості. Вона вказує на стабільність та надійність виявлення об'єктів у широкому діапазоні впевненості моделлю.

Recall-Confidence Curve (Графік Відновлення-Впевненості): Цей графік відображає залежність відновлення (recall) від рівня впевненості моделі у її прогнозах. Він допомагає визначити, на якому рівні впевненості модель досягає високого відновлення.

Precision-Recall Curve (Графік Точність-Відновлення): Цей графік показує залежність точності (precision) від відновлення (recall) при різних порогових значеннях впевненості. Він дозволяє знайти оптимальний баланс між точністю та повністю в залежності від конкретних потреб задачі.

Precision - Confidence Curve (Графік Точність-Впевненість): Цей графік відображає, як точність моделі залежить від рівня впевненості у її прогнозах. Він допомагає визначити, як змінюється точність залежно від впевненості моделі.

F-1 Curve (Графік F-1 міри): Цей графік відображає зв'язок між F-1 мірою та рівнем впевненості моделі у її прогнозах. F-1 міра є гармонічним середнім між точністю та відновленням і використовується для оцінки збалансованості між цими

двома метриками. Графік F-1 міри дозволяє визначити, на якому рівні впевненості модель досягає оптимального балансу між точністю та повністю в залежності від конкретних потреб задачі.

Аналізуючи наведені метрики продуктивності моделі YOLOv8s в контексті завдання виявлення об'єктів , можемо зробити наступні висновки:

Модель виявляється досить точною та надійною у більшості метрик, що були оцінені. Зниження значень box_loss та cls_loss свідчить про поліпшення точності локалізації об'єктів та класифікації. Дефокус-втрати (dfl_loss) також показують, що модель вивчила правильно передбачати ступінь дефокусу об'єктів.

Precision (B) та recall (B) для виявлення об'єктів (B) показують високу точність та повність, що важливо для завдань, де важлива якість виявлення об'єктів. Метрика Map50 (B) вказує на добру середню точність виявлення при 50% порозі впевненості.

Map50 (B)-95 підкреслює стабільність та надійність моделі на широкому спектрі порогів впевненості, що важливо для різних умов та потреб застосування.

Графіки Recall-Confidence та Precision-Recall допомагають визначити оптимальний баланс між точністю та повністю, а графік Precision-Confidence дозволяє зрозуміти, як точність моделі залежить від рівня впевненості у її прогнозах.

З урахуванням цих результатів можна зазначити, що модель має високу потенціальну ефективність у виявленні об'єктів на зображеннях.

Рис.3.11 Результат роботи моделі YOLO

3.2 Навчання штучного інтелекту для управління модулем

Основна мета цього розділу полягає у дослідженні та реалізації алгоритму Proximal Policy Optimization (PPO) для управління модулем в середовищі, адаптованому з OpenAI Gym. PPO, який визнаний одним з найбільш ефективних методів навчання з підкріпленням, буде адаптовано для виконання специфічних завдань управління модулем у цьому унікальному середовищі.

Використання власного середовища, створеного на базі OpenAI Gym, дозволяє забезпечити максимальну контрольованість та гнучкість у налаштуванні параметрів та сценаріїв, що є критично важливим для точного аналізу та вдосконалення алгоритму PPO. Це середовище дозволить детально вивчати поведінку алгоритму, його реакцію на зміни умов та ефективність в різних оперативних сценаріях.

Зосередження на PPO в контексті управління модулем дозволить вивчати, як цей алгоритм може бути оптимізований для вирішення конкретних задач, пов'язаних з динамічним вибором та відстеженням цілей. Це включає аналіз його здатності

адаптуватися до змінних умов середовища та ефективно виконувати завдання з високою точністю та надійністю.

У цьому контексті, ключовим аспектом є розробка та налаштування сценаріїв тестування у власному середовищі, що дозволить ефективно вимірювати та оцінювати продуктивність PPO. Це включає створення різних варіантів завдань, з різним рівнем складності та динаміки, щоб забезпечити всебічне тестування алгоритму.

3.2.1 Стратегії прийняття рішень

Модуль, оснащений алгоритмом Proximal Policy Optimization (PPO), здійснює вибір цілей, керуючись їх координатами та дальністю. Ефективність цього процесу полягає у здатності адаптуватися до динамічних змін в розташуванні цілей. Основна функція модуля — ідентифікувати найбільш підходящу ціль, виходячи із поточної ситуації та змін у координатах потенційних об'єктів.

1.**Аналіз Координат та Дальності до Цілей:** Модуль використовує PPO для аналізу координат потенційних цілей та їх дальності. Цей аналіз допомагає визначити, які цілі знаходяться у зоні досяжності та які з них мають пріоритет на основі їхнього розташування.

2.Динамічний Вибір Цілей: З огляду на, що цілі можуть рухатися, важливо, щоб модуль міг швидко адаптуватися до змін у їх координатах. PPO дозволяє модулю періодично переоцінювати цілі та змінювати свій вибір у відповідь на нову інформацію про їх розташування та дальність.

3.Уникнення Фокусування на Одній Цілі: Щоб запобігти зацикленню на одній цілі, PPO використовує механізм, який оцінює ефективність взаємодії з кожною ціллю. Наприклад, якщо ціль залишається недосяжною протягом певного часу або виявляється неможливою для взаємодії, алгоритм буде перенаправляти фокус на інші, більш доступні цілі.

4.Стратегія "Зміни Цілі": Важливим аспектом у прийнятті рішень є стратегія зміни цілі. Наприклад, якщо визначена ціль не зникає чи виявляється неможливою

для взаємодії, і водночас інші потенційні цілі продовжують рух, PPO пристосовує свою стратегію, переходячи до інших, більш досяжних цілей.

5.Оптимізація Вибору з Плином Часу: Завдяки неперервному навчанню та оптимізації, PPO поступово покращує свою здатність ефективно вибирати цілі. Це включає вивчення з попередніх взаємодій та коригування стратегій на основі отриманих даних про ефективність попередніх виборів цілей.

3.2.2 Створення середовища opengym

Створення середовища OpenAI Gym для алгоритму Proximal Policy Optimization (PPO) починається з ретельного теоретичного планування, яке включає визначення основних параметрів та умов, необхідних для ефективного тренування модуля. На цьому етапі, ключовим завданням є створення віртуального простору, який імітує реальні умови, з якими модуль може зіткнутися. Важливо врахувати такі аспекти, як розмір простору, властивості об'єктів, які будуть використовуватися в середовищі, та правила взаємодії між модулем та цими об'єктами.

На практиці, процес конфігурації середовища включає кілька кроків. Першим кроком є ініціалізація та встановлення OpenAI Gym. Це можна зробити за допомогою певних команд у терміналі або в середовищі розробки, яке використовується для проекту. Наприклад, встановлення OpenAI Gym виглядає так:

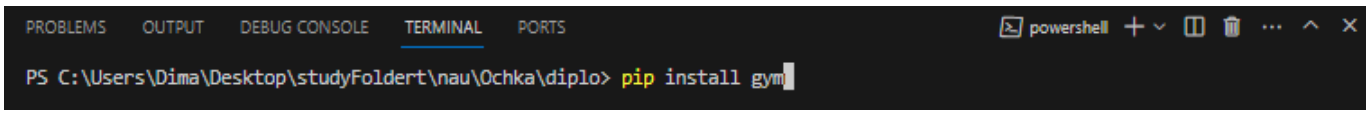
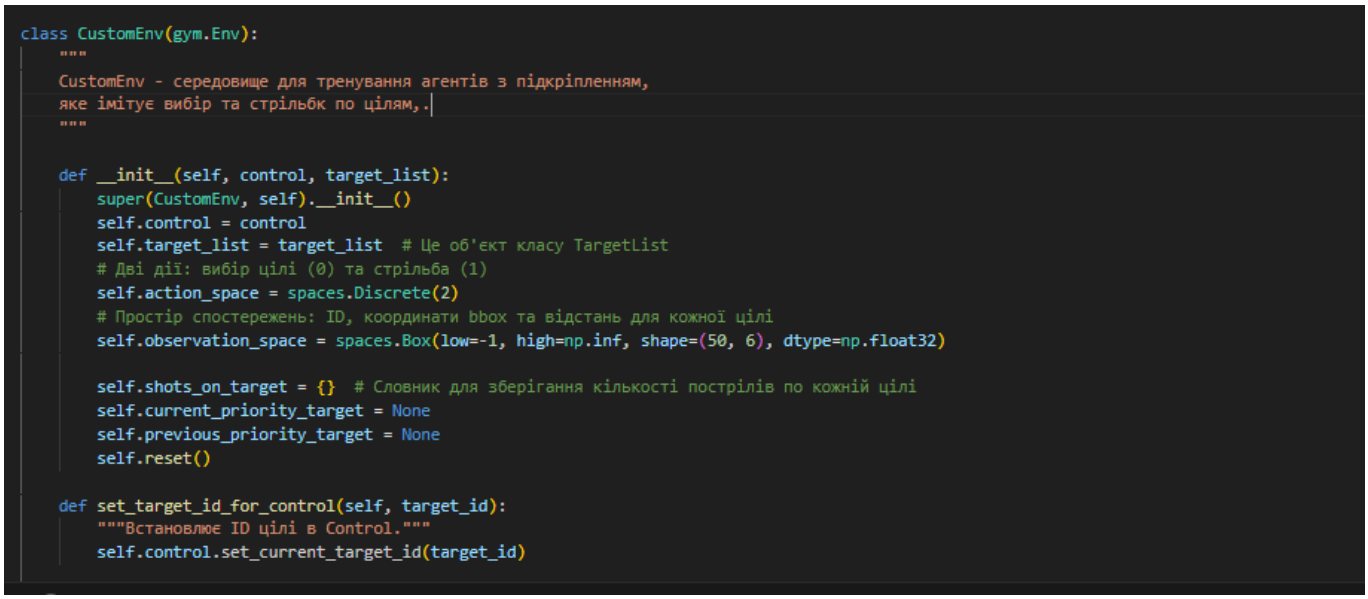


Рис.3.12 Встановлення OpenAI Gym

Після успішного встановлення OpenAI Gym, наступним кроком є створення специфічного середовища для тренування PPO. Це включає написання коду для ініціалізації середовища з використанням функцій та класів, які надає OpenAI Gym. Визначаються параметри середовища, такі як розміри, межі, об'єкти, які включаються до середовища, та правила їх взаємодії.



```python
class CustomEnv(gym.Env):
    """
    CustomEnv - середовище для тренування агентів з підкріпленням,
    яке імітує вибір та стрільбк по цілям,.
    """

    def __init__(self, control, target_list):
        super(CustomEnv, self).__init__()
        self.control = control
        self.target_list = target_list  # Це об'єкт класу TargetList
        # Дві дії: вибір цілі (0) та стрільба (1)
        self.action_space = spaces.Discrete(2)
        # Простір спостережень: ID, координати bbox та відстань для кожної цілі
        self.observation_space = spaces.Box(low=-1, high=np.inf, shape=(50, 6), dtype=np.float32)

        self.shots_on_target = {}  # Словник для зберігання кількості пострілів по кожній цілі
        self.current_priority_target = None
        self.previous_priority_target = None
        self.reset()

    def set_target_id_for_control(self, target_id):
        """Встановлює ID цілі в Control."""
        self.control.set_current_target_id(target_id)
```

Рис.3.13 Ініціалізація власного середовища OpenAI Gym

Після ініціалізації середовища важливим етапом є налаштування характеристик та динаміки об'єктів у цьому середовищі. Це включає розробку та конфігурацію віртуальних об'єктів із зазначенням їхніх особливих властивостей та моделей поведінки. Наприклад, у сценаріях, де необхідно імітувати динамічні об'єкти, як-от рухомі цілі, слід чітко визначити параметри їх руху, такі як швидкість, траєкторію та способи взаємодії цих об'єктів з агентом модуля. Це дозволяє створити умови, максимально наближені до реальних викликів, з якими агент буде стикатися при виконанні завдань управління модулем.

Завершальним етапом налаштування середовища є розробка механізмів взаємодії між модулем управління та об'єктами у віртуальному середовищі. Цей процес передбачає детальне визначення реакцій модуля на зміни, що відбуваються з об'єктами середовища, та оцінку його поведінки в різних сценаріях. Особлива увага приділяється встановленню критеріїв, за якими оцінюється ефективність модуля, відповідно до специфіки поставлених завдань. Визначення цих правил є ключовим для забезпечення адекватного та ефективного взаємодії модуля з динамічно змінюваним середовищем.

Останнім кроком у налаштуванні середовища є розробка правил і механізмів взаємодії між управлінським модулем та елементами в середовищі OpenAI Gym.

Цей етап включає глибокий аналіз того, як модуль відреагує на різноманітні ситуації та зміни, що відбуваються з об'єктами у середовищі. Основна увага зосереджується на розробці критеріїв оцінки ефективності дій модуля, з урахуванням особливостей та вимог конкретних задач. Встановлення чітких правил взаємодії є вирішальним для забезпечення адекватної та цілеспрямованої взаємодії модуля зі змінним віртуальним середовищем.

3.2.3 Симулятор для навчання PPO

Розробка симуляційного середовища для навчання PPO. В центрі цього процесу стоїть клас MovingObject, який втілює основні характеристики динамічних об'єктів у віртуальному середовищі. Об'єкти, представлені в цьому класі, оснащені атрибутами, які дозволяють їм наслідувати рух та зміну розмірів, імітуючи реальні цілі.

```python
class MovingObject:
    def __init__(self, video_size):
        global global_id_counter
        self.video_size = video_size
        self.position = np.array([random.uniform(0, video_size[0]), random.uniform(0, video_size[1])])
        self.size = random.uniform(3, 10)  # Розмір об'єкта
        self.distance = random.uniform(5, 100)
        self.movement_vector = np.random.rand(2) * 2 - 1  # Вектор руху
        self.size_change = random.uniform(-0.1, 0.1)  # Зміна розміру
        self.lives = random.randint(1, 4)  # Кількість життів
        self.track_id = global_id_counter
        global_id_counter += 1
        self.bbox = self.get_bbox()
```

```python
    def to_track_format(self):
        # Конвертація у формат, сумісний з TargetList
        return {
            'track_id': self.track_id,
            'bbox': self.get_bbox(),
            'distance': self.distance
        }

    def get_bbox(self):
        # Обчислення координат "ліва верхня точка - права нижня точка"
        x1 = self.position[0] - self.size / 2
        y1 = self.position[1] - self.size / 2
        x2 = self.position[0] + self.size / 2
        y2 = self.position[1] + self.size / 2
        return [x1, y1, x2, y2]
```

```python
    def update(self):
        # Оновлення положення та розміру об'єкта
        old_size = self.size
        self.position += self.movement_vector
        self.size += self.size_change
        self.size = max(1, min(self.size, 40))
        self.bbox = self.get_bbox()
        if self.size > old_size:
            # Якщо розмір збільшився, зменшуємо дистанцію
            self.distance -= np.abs(self.size_change)
        else:
            # Якщо розмір зменшився, збільшуємо дистанцію
            self.distance += np.abs(self.size_change)

        # Обмеження координат в межах відеопотоку
        self.position[0] = np.clip(self.position[0], 0, self.video_size[0])
        self.position[1] = np.clip(self.position[1], 0, self.video_size[1])

    def hit(self):
        # Обробка "пострілу" по об'єкту
        self.lives -= 1
        return self.lives <= 0  # Повертає True, якщо об'єкт "знищений"
```

Рис.3.14 Клас MovingObject

Клас SimulationEnvironment відповідає за створення та управління цілим середовищем, де відбувається взаємодія між об'єктами та агентом PPO. В цьому середовищі відтворюються різні сценарії, що дозволяє агенту вчитися адаптуватися та реагувати на зміни. Регулярні оновлення стану об'єктів та відповідні дії агента, такі як вибір цілі та постріл.

```python
class SimulationEnvironment(threading.Thread):
    def __init__(self, video_size, num_objects,TargetList):
        super().__init__()
        self.video_size = video_size
        self.num_objects = num_objects
        self.objects = [MovingObject(video_size) for _ in range(num_objects)]
        self.running = False
        self.target_list = TargetList


    def run(self):
        self.running = True
        while self.running:
            self.step()
            self.update_Target()
            print(self.target_list.print_targets())
            time.sleep(1)  # Частота оновлення

    def stop(self):
        self.running = False

    def update_Target(self):
        self.target_list.update_targets(self.objects)

```

```python
    def shoot_at_target(self,target_id):
        """Обробляє постріл по цілі."""
        target = next((obj for obj in self.objects if obj.track_id == target_id), None)
        if target:
            is_destroyed = target.hit()  # Зменшує кількість життів і перевіряє, чи ціль знищена
            return is_destroyed
        return False

    def step(self):
        # Оновлення стану об'єктів
        for obj in self.objects:
            obj.update()
            if obj.lives <= 0:
                print("ціль знищено")
                # Заміна об'єкта новим, якщо він "знищений"
                self.objects[self.objects.index(obj)] = MovingObject(self.video_size)

    def get_state(self):
        # Отримання поточного стану симуляції
        return [{"position": obj.position.tolist(), "size": obj.size, "lives": obj.lives} for obj in self.objects]
```

Рис.3.15 Клас SimulationEnvironment

У процесі тренування PPO агент вчиться аналізувати середовище, оцінювати цілі та розробляти стратегії для максимально ефективних дій. Агент використовує дані про положення, рух та інші параметри об'єктів для прийняття обґрунтованих рішень. Цей процес вимагає від алгоритму адаптації до постійно змінних умов середовища.

В цілому, розроблений симулятор стає ключовим інструментом для вивчення поведінки алгоритмів штучного інтелекту у складних умовах, відкриваючи шлях до їх застосування у реальному світі.

3.2.4 Навчання PPO

Навчання PPO в розробленому середовищі OpenAI Gym відіграє ключову роль у вивченні та розвитку ефективних стратегій управління. Використання PPO, реалізованого через бібліотеку stable_baselines3, забезпечує можливість тренувати алгоритм в контрольованому, але реалістичному середовищі, де він може вчитися взаємодіяти з динамічно змінюваними об'єктами.

Першим кроком у процесі навчання є інтеграція PPO з розробленим симуляційним середовищем. Використовуючи функції та інструменти, надані stable_baselines3, налаштовується взаємодія між агентом та середовищем. Агенту PPO надаються дані про стан середовища, що включають інформацію про положення, розміри та інші характеристики об'єктів.

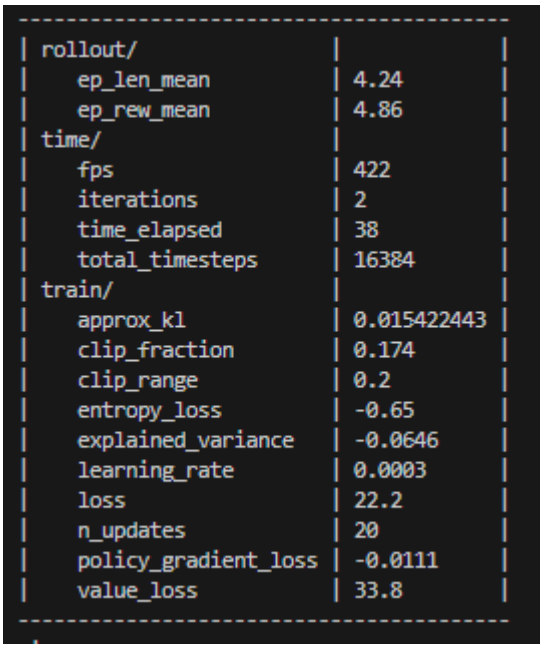
```python
from Simulation import SimulationEnvironment
from TargetListForDiploSimulator import TargetList
from MyCustomENVForDiplo import CustomEnv
from control import Control
from stable_baselines3 import PPO
from stable_baselines3.common.env_util import make_vec_env
import threading
# Функція для запуску симуляції
TargList = TargetList()
simulation = SimulationEnvironment((640, 480), 3, TargList)
control = Control(TargList,simulation, 0.5)
def run_simulation():
    simulation.start()

# Функція для тренування моделі
def train_model():
    env = CustomEnv(control, TargList)
    vec_env = make_vec_env(lambda: env, n_envs=4)
    model = PPO('MlpPolicy', vec_env, verbose=1)
    model.learn(total_timesteps=100000)
    model.save("ppo_custom_env_model")
# Створення та запуск потоків
simulation_thread = threading.Thread(target=run_simulation)
model_thread = threading.Thread(target=train_model)
simulation_thread.start()
model_thread.start()
simulation_thread.join()
model_thread.join()
```

Рис.3.16 Ініціалізація Навчання PPO

Процес навчання PPO розподіляється на кілька етапів. Спочатку агент здійснює випробувальні дії в середовищі, збираючи дані та спостереження. Ці дані використовуються для оновлення політик агента, де алгоритм вчиться визначати, які дії призводять до більш ефективних результатів.

Важливою частиною навчання є адаптація стратегії агента до складності середовища. Завдяки гнучкості PPO, агент здатен оптимізувати свої дії з урахуванням різних сценаріїв та обмежень. Цей процес включає постійне оцінювання результатів дій та коригування політики на основі зібраного досвіду.

Після тривалого процесу навчання, агент PPO досягає стадії, де він здатен ефективно визначати цілі та виконувати оптимальні дії в різних сценаріях середовища. Завдяки адаптивності та глибокому навчанню, модель здатна продемонструвати високу ефективність та точність у виборі стратегій.



rollout/	
ep_len_mean	4.24
ep_rew_mean	4.86
time/	
fps	422
iterations	2
time_elapsed	38
total_timesteps	16384
train/	
approx_kl	0.015422443
clip_fraction	0.174
clip_range	0.2
entropy_loss	-0.65
explained_variance	-0.0646
learning_rate	0.0003
loss	22.2
n_updates	20
policy_gradient_loss	-0.0111
value_loss	33.8

Рис.3.17 Результат навчання

Завершення цього процесу навчання відкриває нові перспективи для застосування PPO в реальних умовах, де алгоритм здатен адаптуватися до складних ситуацій та ефективно управляти визначеними задачами.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ МОДУЛЕМ

Система управління бойовим модулем інтегрує методики штучного інтелекту, зосереджуючись на автоматизованому виявленні, ідентифікації та слідкуванні за об'єктами. Розробка такого прототипу здійснена для вивчення можливостей впровадження сучасних технологій у військову практику, з метою покращення ефективності бойових дій. Завдяки впровадженню алгоритмів машинного навчання та комп'ютерного зору, прототип відкриває перспективи для оптимізації процесів управління та ведення бою, що є значним внеском у сфері військових інновацій.

4.1 Функціональні вимоги

Розроблена система задовольняє визначені функціональні вимоги, які є вирішальними для її ефективності та надійності. Кожна з цих вимог специфічно спрямована на забезпечення ключових аспектів роботи системи, відповідаючи на потреби заданих оперативних цілей.

1.AI-підтримка безперервного трекінгу цілей

Система активно використовує алгоритми штучного інтелекту для безперервного відстеження та ідентифікації цілей у відеопотоці, забезпечуючи високу точність і швидкість реакції.

2.Автономне патрулювання AI

Автономне патрулювання заданого сектору вогню ефективно реалізоване за допомогою алгоритмів штучного інтелекту, що забезпечує постійне спостереження та адаптивну реакцію на зміни в середовищі.

3.Інтегроване ручне та автоматичне управління

Система успішно інтегрує ручний вибір цілей з автоматичним націленням, підвищуючи точність та швидкість стрільби завдяки співпраці оператора та автоматизованих функцій.

4.2 Функціональна Схема

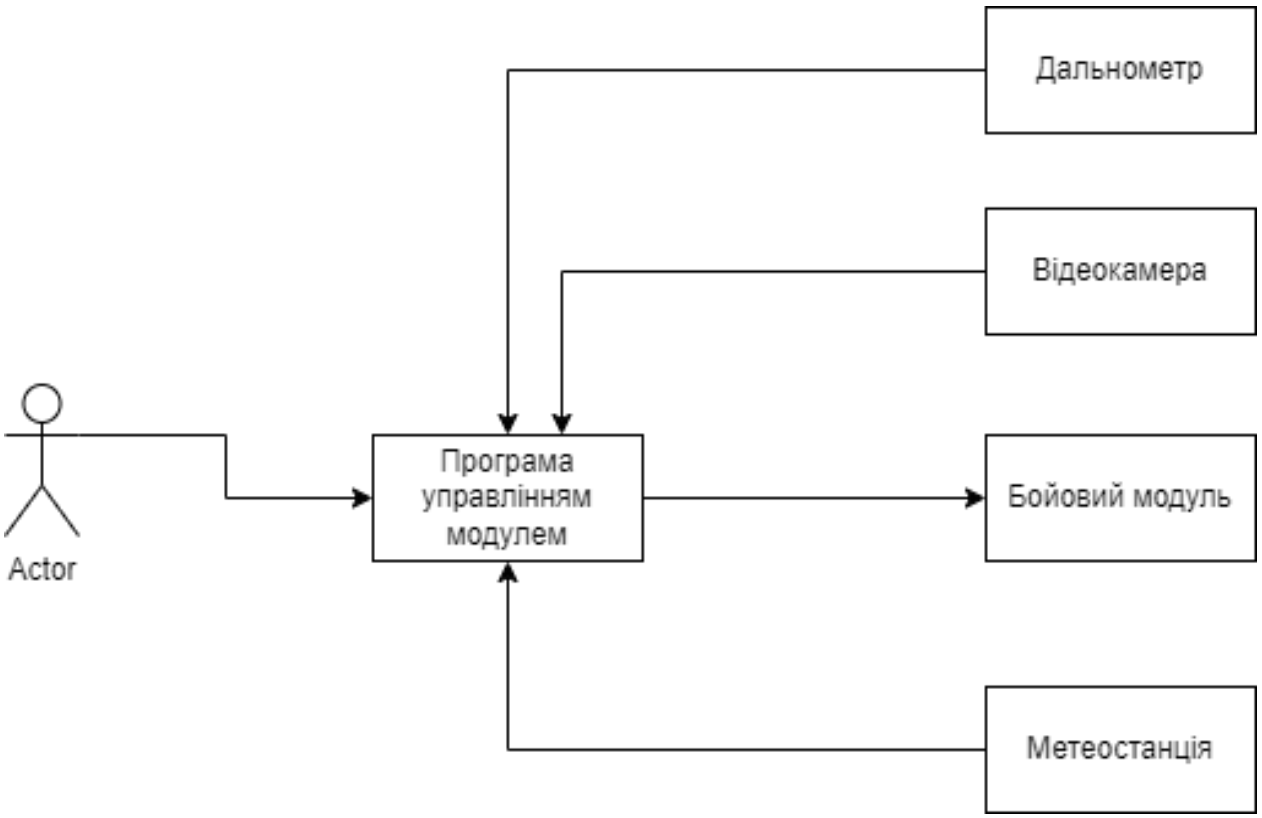


Рис.4.1 Функціональна схема

Згідно функціональної схеми, програмне забезпечення для системи управління бойовим модулем розроблене з урахуванням певних допусків, які оптимізують його функціонування та інтеграцію з апаратною частиною.

Ключовий аспект реалізації ПЗ полягає у тому, що воно допускає, і відповідно до цього програмується, що центр поля зору відеокамери є ідентичним центру прицілу зброї, встановленої на бойовому модулі. Це забезпечує, що будь-які рухи модуля, які здійснюються під час націлювання або слідкування за цілями, точно відтворюються в відеозображенні, що значно підвищує точність ведення вогню та ефективність реагування на рухомі цілі.

Дальніше, ПЗ враховує, що метеостанція розташована поруч із бойовим модулем, що дозволяє збирати метеодані без значних затримок, та інтегрує цю інформацію в реальному часі для корекції стрільби відповідно до зовнішніх умов.

Також ПЗ передбачає, що дальнометр встановлено безпосередньо на бойовому модулі, що дозволяє здійснювати точні виміри відстаней до цілей відносно

положення модуля. Ця інтеграція критично важлива для точного розрахунку параметрів для націлення та стрільби.

Ці допуски відіграють вирішальну роль у реалізації програмного забезпечення, яке повинно гарантувати, що всі компоненти системи працюють злагоджено та ефективно, забезпечуючи високу точність та оперативність у управлінні бойовим модулем.

4.3 Дизайн ПЗ

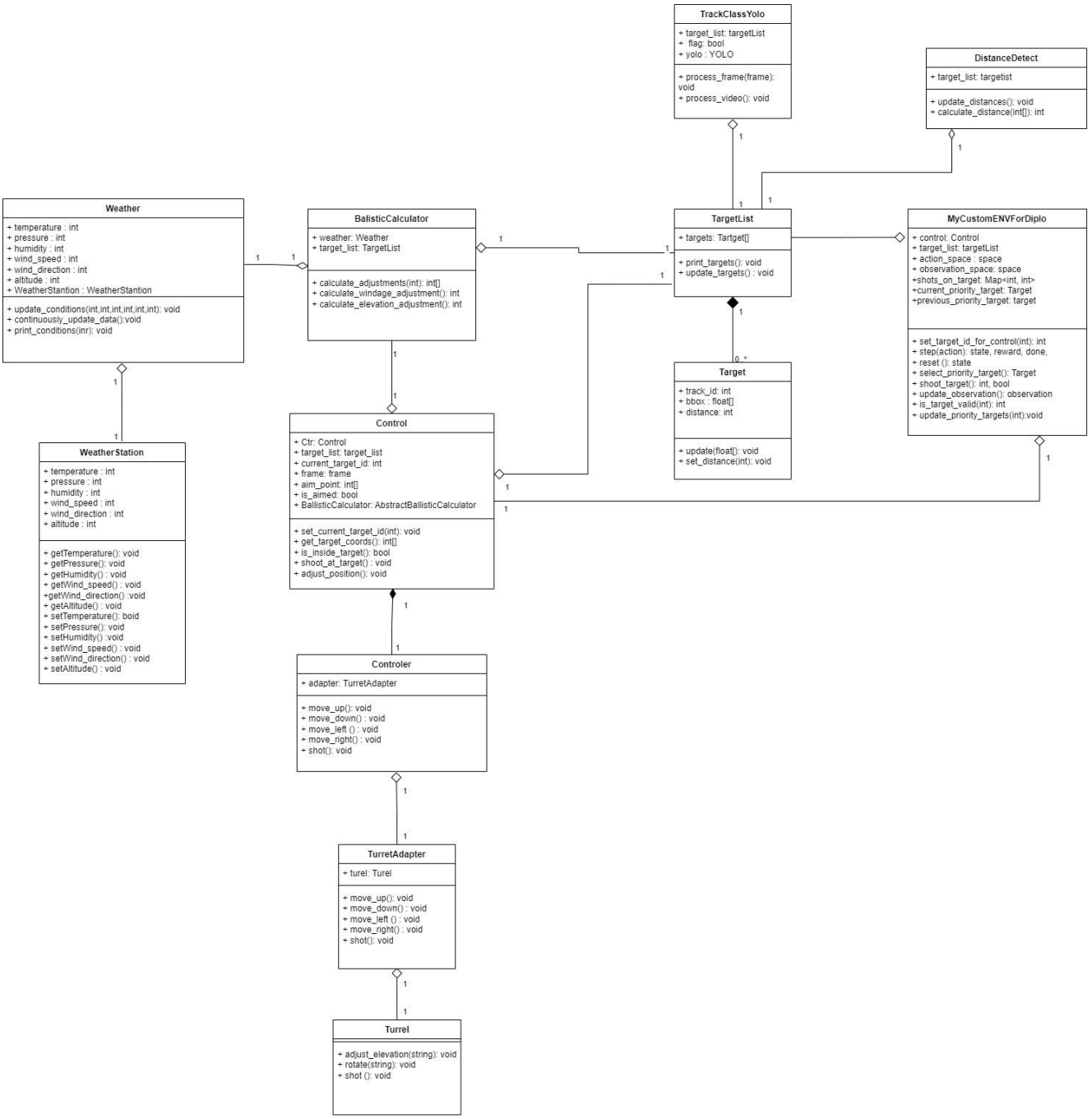


Рис.4.2 UML діаграма класів

4.4 Мова програмування

Python, мова програмування високого рівня, була створена Гвідо ван Россумом у 1991 році. Основна ідея створення Python полягала у розробці мови, яка була б одночасно потужною та легкою для читання та написання. Python швидко набув популярності у розробників завдяки своїй чистоті, ефективності та універсальності. Від початку Python був задуманий як мова, що дозволяє вирішувати різноманітні завдання - від розробки веб-додатків до складних наукових обчислень. Ця універсальність, поєднана з легкістю освоєння, сприяла його широкому прийняттю у науковій спільноті, особливо у сфері штучного інтелекту.

Основна характеристика дизайну Python - це його синтаксис, заснований на принципах читабельності та простоти. Використання відступів для визначення структури коду замість складного використання дужок або ключових слів сприяє легкості читання та розуміння програм. Ця простота не впливає на потужність мови; Python підтримує динамічну типізацію, роботу з різними типами даних, та включає велику стандартну бібліотеку, яка охоплює широкий спектр функціоналу від веб-розробки до наукових обчислень.

Python має вирішальне значення у сферах, де використовується штучний інтелект та машинне навчання. Його бібліотеки, такі як TensorFlow, Keras та PyTorch, надають потужні інструменти для розробки та тренування моделей машинного навчання, від простих алгоритмів класифікації до складних нейронних мереж. Легкість інтеграції Python з іншими мовами та системами, а також його універсальність, робить його ідеальним вибором для розробки та дослідження у галузі ШІ.

Управління пакетами в Python відіграє важливу роль у спрощенні та оптимізації процесу розробки. В основі цього процесу лежить pip, офіційний менеджер пакетів Python, який використовується для встановлення, оновлення та видалення бібліотек та інструментів з Python Package Index (PyPI). PyPI служить централізованим сховищем для тисяч пакетів, що розширюють функціональність Python у різних областях, від веб-розробки до наукових обчислень.

4.5 Середовище програмування

Visual Studio Code, розроблене компанією Microsoft, представляє собою одне з найпопулярніших середовищ розробки, яке використовується програмістами по всьому світу. Основною відмітною особливістю Visual Studio Code є його універсальність і підтримка широкого спектру мов програмування, включаючи, але не обмежуючись, JavaScript, Python, C++, Java. Ця можливість робить Visual Studio Code інструментом, який може бути адаптований до різних проектів в області програмування.

Однією з ключових характеристик Visual Studio Code є його розширюваність. Користувачі мають можливість встановлювати додатки та плагіни, що розширюють функціональність середовища. Це включає підтримку додаткових мов програмування, інтеграцію з іншими сервісами та інструментами, а також налаштування інтерфейсу і робочого процесу згідно індивідуальних потреб користувача.

Ще однією важливою особливістю Visual Studio Code є його вбудовані інструменти для спрощення розробки та відлагодження коду. Середовище надає розширені можливості для відлагодження, що дозволяє розробникам легко знаходити та виправляти помилки в коді. Крім того, Visual Studio Code містить інструменти для управління версіями коду, що є важливим аспектом в сучасній розробці програмного забезпечення.

Інтеграція з системами контролю версій, зокрема з Git, є ще однією значущою перевагою Visual Studio Code. Ця інтеграція забезпечує зручний доступ до функцій контролю версій безпосередньо з середовища розробки. Така можливість сприяє більш ефективній та організованій роботі над проектами.

Visual Studio Code також відомий своїм легким та швидким виконанням порівняно з іншими інтегрованими середовищами розробки. Це робить його відмінним вибором для розробників, які шукають ефективне та водночас потужне середовище для програмування.

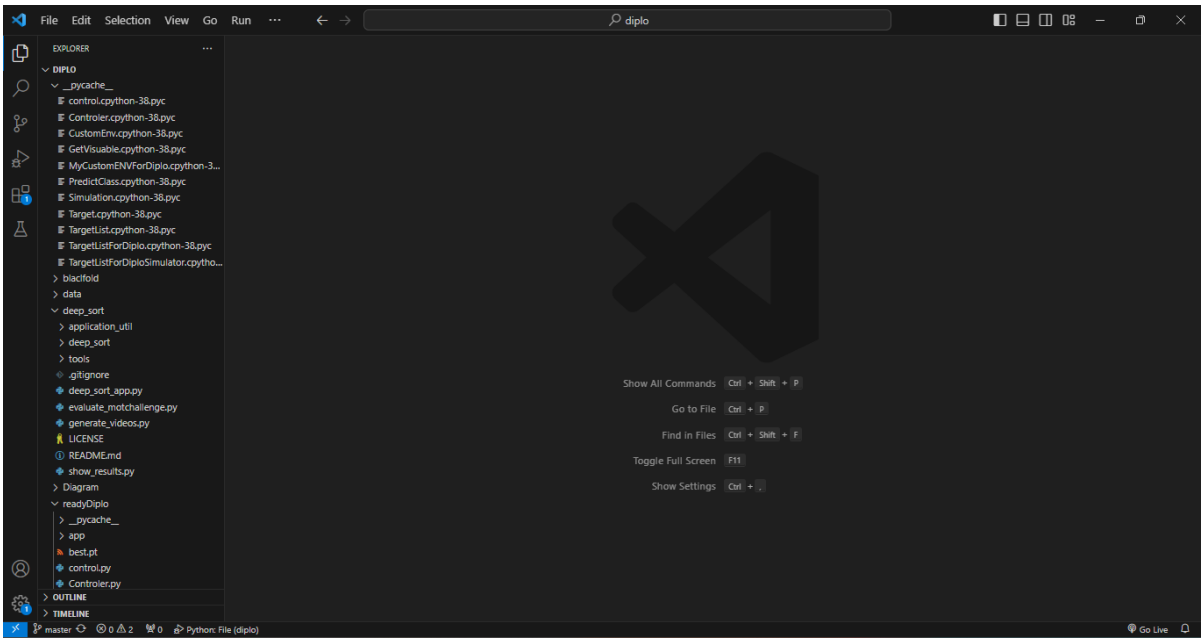


Рис.4.3 Інтерфейс Visual Studio Code

Нарешті, Visual Studio Code підтримує спільноту розробників шляхом надання великої кількості документації, посібників та ресурсів для навчання. Це не тільки сприяє швидкому вивченню середовища новими користувачами, але й підтримує розвиток навичок і знань досвідчених розробників.

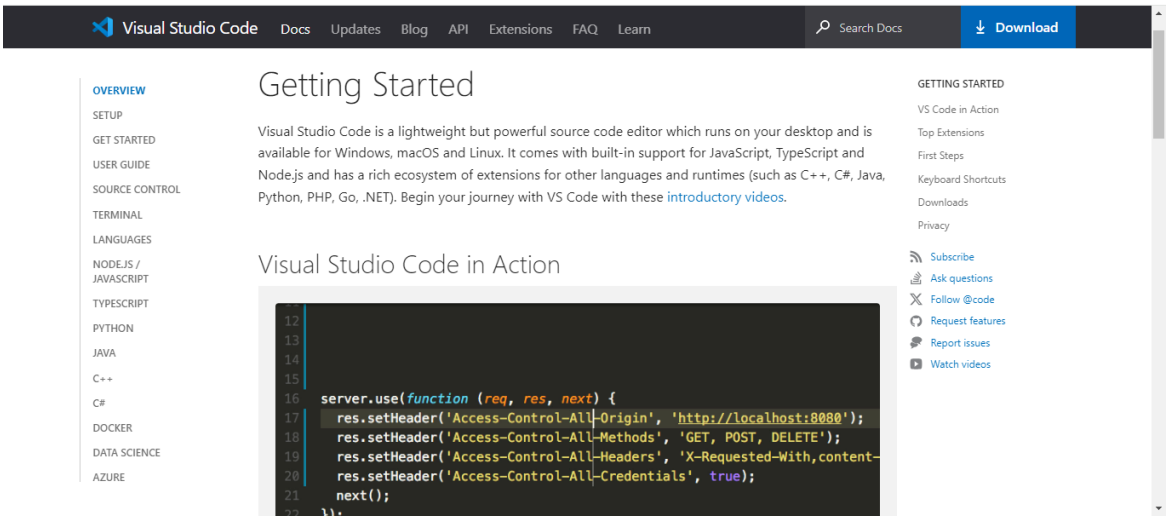


Рис.4.4 Документація Visual Studio Code

Враховуючи ці особливості, Visual Studio Code можна вважати одним з найбільш гнучких і потужних інструментів для розробки програмного забезпечення,

що робить його популярним вибором серед професіоналів різних напрямків у сфері IT.

4.5 Використані бібліотеки

При роботі з Python, особливо у великих проектах або при роботі в команді, важливою є спроможність управління залежностями та забезпечення консистентності середовища розробки. pip вирішує цю проблему, дозволяючи розробникам визначати конкретні версії пакетів, необхідних для проекту. Це здійснюється через використання файлів вимог, зазвичай називаних requirements.txt, в яких перераховуються всі необхідні пакети та їх версії. Такий підхід дозволяє легко відтворити однакове середовище на різних машинах, спрощуючи співпрацю та розгортання проектів.

Використання pip як основного інструмента для управління пакетами в проектах на Python має безпосередній вплив на ефективність розробки, забезпечуючи легкий доступ до широкого спектру інструментів та бібліотек. Це, в свою чергу, відкриває двері для широких можливостей у різних областях застосування Python, підкреслюючи його міцну позицію як універсальної, гнучкої та потужної мови програмування.

З огляду на значення управління пакетами у Python та роль менеджера пакетів pip, слід звернути увагу на специфічні бібліотеки, які були використані в даній дипломній роботі.

PyQt5 - бібліотека для мови програмування Python, створена для розробки графічних користувацьких інтерфейсів (GUI) на базі бібліотеки Qt. Вона дозволяє розробникам легко створювати і керувати елементами інтерфейсу, такими як вікна, кнопки, меню і багато інших. PyQt5 об'єднує можливості Python з Qt, роблячи його ідеальним для розробки інтерактивних програм.

Ця бібліотека підтримує різні операційні системи та платформи, що робить її універсальним інструментом для розробки програм. Вона знаходить застосування в різних областях, включаючи наукові дослідження, розробку десктопних додатків та веб-програмування. PyQt5 - це потужний і зручний інструмент для створення інтерфейсів і програм на Python.

OpenCV, або Open Source Computer Vision Library, є бібліотекою для обробки зображень та обчислювального зору. Вона надає розробникам та дослідникам інструменти для виконання різноманітних завдань у галузі обробки зображень, таких як читання та запис зображень, аналіз зображень і відео, а також їх змінення.

OpenCV дозволяє працювати з різними форматами зображень і відео, виконувати такі завдання, як обрізка, фільтрація, виокремлення об'єктів, визначення обличчя, відстеження об'єктів, вимірювання і багато інших операцій у галузі комп'ютерного зору.

OpenCV є потужним інструментом для розробників і дослідників, які працюють з обробкою зображень та відео в різних областях застосування.

Ultralytics - це потужна інструментальна бібліотека, призначена для розробки та навчання глибоких нейронних мереж з використанням машинного навчання та комп'ютерного зору. Ця бібліотека забезпечує широкий спектр функціональних можливостей і дозволяє дослідникам та розробникам ефективно працювати над завданнями у сфері обробки зображень і відео.

Однією з ключових особливостей Ultralytics є його інтегрований інтерфейс зі стандартними фреймворками машинного навчання, такими як PyTorch. Це спрощує розробку та навчання моделей штучного інтелекту та дозволяє використовувати потужність цих фреймворків в повній мірі.

Бібліотека Ultralytics також має багатий набір функцій для роботи з даними, а також забезпечує зручний інтерфейс для навчання моделей на власних даних. Вона включає в себе інструменти для обробки зображень та відео, а також можливість візуалізації результатів роботи моделей

OpenAI Gym - це програмна бібліотека, розроблена OpenAI, призначена для роботи з алгоритмами і штучним інтелектом, зокрема в галузі підсиленого навчання. Головною метою бібліотеки є забезпечення науковців та розробників доступом до стандартних віртуальних середовищ, у яких можна навчати і тестувати агентів, що приймають рішення.

OpenAI Gym надає широкий спектр стандартних завдань і сценаріїв для навчання агентів. Це включає симуляції фізичних процесів, ігри та інші віртуальні

середовища, в яких агентам потрібно вчитися взаємодіяти з оточенням та приймати оптимальні рішення для досягнення поставлених цілей.

Однією з ключових переваг OpenAI Gym є його зручний інтерфейс, який спрощує розробку та тестування алгоритмів. Він також надає можливість обміну дослідженнями та методами між спільнотою науковців у галузі штучного інтелекту.

Бібліотека Stable Baselines3 є програмним інструментом для реалізації та навчання алгоритмів з підсиленою вченістю в мові програмування Python. Вона представляє собою нащадок і покращену версію попередніх бібліотек "Stable Baselines" та "Stable Baselines2" і спеціалізується на забезпеченні інтерфейсу та інструментів для навчання та оцінки різних алгоритмів глибокого навчання, спрямованих на розв'язання задач з посиленням.

Ця бібліотека зосереджена на наданні засобів для розв'язання завдань з посиленням в різних областях, включаючи автоматизоване управління, робототехніку, гральну індустрію та інші сфери, де необхідне прийняття рішень в умовах невизначеності. Вона базується на сучасних технологіях глибокого навчання та надає гнучкий інтерфейс для навчання агентів, які можуть взаємодіяти з динамічними середовищами та вирішувати задачі з максимальною ефективністю.

4.6 Реалізація програми управління

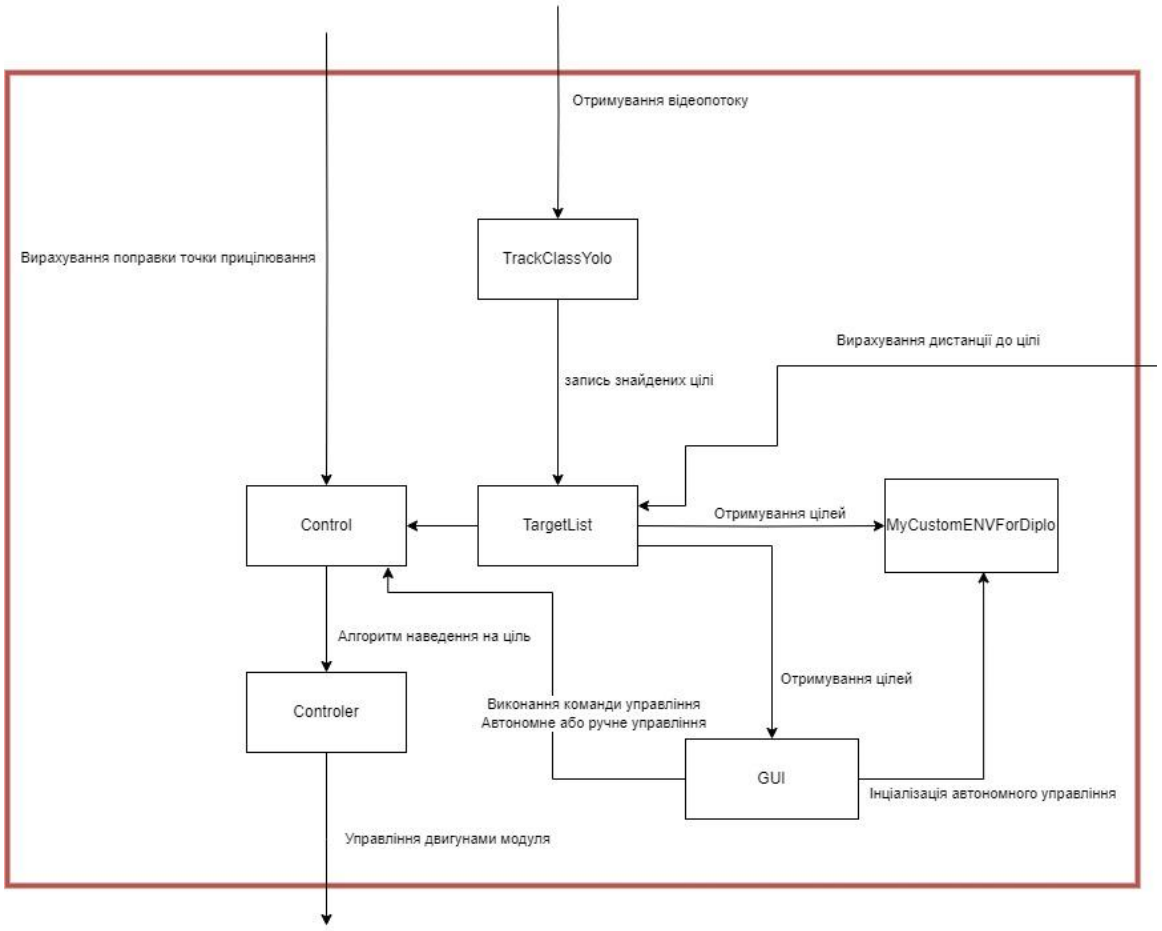


Рис.4.5 Структурна схема

Розроблена програма представляє собою важливий інструмент, який може бути інтегрований у складні системи управління. Ця програма розпочинає свою роботу з активації класу TrackClassYolo, який відіграє ключову роль у пошуку та спостереженні за цілями, здійснюючи трекінг об'єктів у відеопотоці. Використовуючи алгоритм YOLO, цей клас ефективно ідентифікує та відстежує об'єкти в полі зору камери, реєструючи їх у класі TargetListForDiplo з точними координатами та унікальними ідентифікаторами.

Вибір конкретних цілей може здійснюватися через інтеграцію з MyCustomENVForDiplo, використовуючи модель PPO, або вручну оператором через клас GUI. Клас Control спеціалізується на точному націленні, забезпечуючи націлення модуля наведення на вибрані координати цілі.

Ця програма ідеально підходить для інтеграції в складні системи, особливо у сферах, де потрібна висока точність та адаптивність, наприклад у військових або оборонних застосуваннях

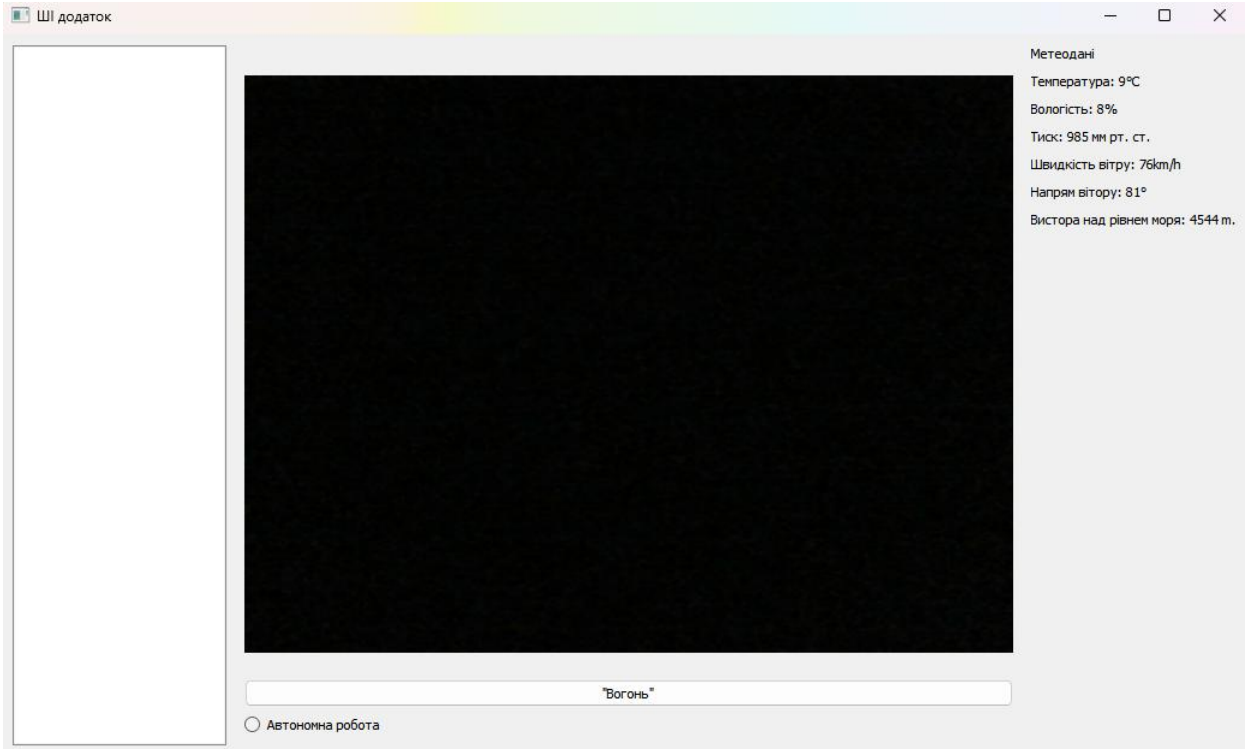


Рис.4.6 GUI

ВИСНОВКИ

Ця робота представляє глибокий аналіз та практичне застосування штучного інтелекту в конкретній сфері - управлінні модулем пошуку та наведення. Вона включає детальний огляд основ нейронних мереж, їх історії, структури та методів навчання, а також комп’ютерного зору, з акцентом на згорткові нейронні мережі (CNN), які відіграють ключову роль у процесі ідентифікації та відстеження об'єктів.

Особливий акцент було зроблено на аналізі сучасних алгоритмів штучного інтелекту, зокрема YOLO, DEEPSORT та SORT для виявлення та відстеження об’єктів, а також алгоритмів навчання з підкріпленням, таких як DQN, PPO та A3C. Важливим аспектом стала також оцінка імітаційних середовищ для навчання штучного інтелекту, зокрема OpenAI Gym та TensorFlow Agents.

На основі проведеного аналізу було розроблено практичний проект системи агентів. Проект включає етапи навчання штучного інтелекту для трекінгу об’єктів за допомогою моделі YOLO, а також для управління модулем з використанням середовища OpenAI Gym та алгоритму PPO. Це дозволило створити систему, здатну самостійно ідентифікувати та відстежувати цілі, а також приймати рішення щодо наведення на них.

Результати даної роботи вказують на величезний потенціал штучного інтелекту у розробці інноваційних систем управління, які можуть знайти застосування в різних галузях, включаючи оборонну сферу. Аналіз та розробка, виконані в рамках цієї роботи, можуть слугувати фундаментом для подальших досліджень та розвитку в даній області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016.

2. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Т. 521, № 7553. – С. 436-444.

3. JavaTpoint. Artificial Neural Network [Електронний ресурс]. – Режим доступу: https://www.javatpoint.com/artificial-neural-network.

4. Schmidhuber J. Deep learning in neural networks: An overview // Neural networks. – 2015. – Т. 61. – С. 85-117.

5. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. – O'Reilly Media, 2019.

6. Karpathy A. The Unreasonable Effectiveness of Recurrent Neural Networks. – 2015.

7. Brownlee J. Deep Learning for Computer Vision. – Machine Learning Mastery, 2016.

8. NVIDIA. What’s the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning? [Електронний ресурс]. – 2020. – Режим доступу: (URL не надано).

9. Raschka S., Mirjalili V. Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2. – Packt Publishing Ltd, 2019.

10. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE conference on computer vision and pattern recognition. – 2016.

11. Zhang C., Bengio S., Hardt M., Recht B., Vinyals O. Understanding deep learning requires rethinking generalization // arXiv preprint arXiv:1611.03530. – 2017.

12. Mnih V., Kavukcuoglu K., Silver D., Rusu A. A., Veness J., Bellemare M. G., Petersen S. Human-level control through deep reinforcement learning // Nature. – 2015. – Т. 518, № 7540. – С. 529-533.

13. Roboflow Blog. What is Semi-Supervised Learning? [Електронний ресурс]. – Режим доступу: https://blog.roboflow.com/what-is-semi-supervised-learning/.

14. Synopsys. What is Reinforcement Learning? [Електронний ресурс]. – Режим доступу: https://www.synopsys.com/ai/what-is-reinforcement-learning.html.

15. IBM. What is Unsupervised Learning? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/unsupervised-learning.

16. IBM. What is Supervised Learning? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/supervised-learning.

17. Тегмарк М. Життя 3.0.: доба штучного інтелекту. Київ: Наш формат, 2019. 432 с.

18. Bengio Y., Lecun, Y. Convolutional Networks for Images, Speech, and Time-Series. 1997.

19. Khandelwal R. Convolutional Neural Network (CNN) Simplified. 2018. URL : https://medium.com/datadriveninvestor/convolutional-neuralnetwork-cnn-simplified-ecafd4ee52c5

20. Stockman G. C., Shapiro L. G. Computer Vision. Prentice Hall, February 2001. 609 p

21. Rao D., McMahan B. Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning. O'Reilly Media, 2019

22. Springenberg J. T., Dosovitskiy A., Brox T., Riedmiller M. Striving for implicity: The All Convolutional Net. arXiv preprint: 1412.6806. 2014

23. Scherer, Dominik; Müller, Andreas C.; Behnke, Sven (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. / 20th International Conference on Artificial Neural Networks (ICANN), Thessaloniki, Greece: Springer, 2010.

24. Graham B. Fractional max-pooling. arXiv preprint: 1412.6071. 2014.

25. Khan S., Rahmani H., Ali Shakh S. A., Bennamoun M. A Guide to Convolutional Neural Networks for Computer Vision. Morgan & Claypool Publishers, 2018

26. Гітіс В. Б., Гудкова К.Ю. Методи штучного інтелекту: навч. посіб. Краматорськ, 2018. 136 с.

27. Faty L. A General-Purpose Machine Reasoning Engine. *International Conference on Artificial General Intelligence*. 2023. №15. P. 3–13.

28. Навчання з підкріпленням. Вікіпедія: веб-сайт. URL: http://surl.li/loxen. (дата звернення: 07.11.2023).

29. Савченко А. С., Синельніков О.О. Методи та системи штучного інтелекту. Київ: НАУ-друк, 2017. 190 с.

30. Asynchronous Advantage Actor-Critic (A3C). ActiveLoop: веб-сайт. URL: https://www.activeloop.ai/resources/glossary/proximal-policy-optimization-ppo/. (дата звернення: 07.11.2023).

31. Proximal Policy Optimization (PPO). ActiveLoop: веб-сайт. URL: https://www.activeloop.ai/resources/glossary/proximal-policy-optimization-ppo/. (дата звернення: 07.11.2023).

32. Deep Q-Networks (DQN). ActiveLoop: веб-сайт. URL: https://www.activeloop.ai/resources/glossary/deep-q-networks-dqn/. (дата звернення: 07.11.2023).

33. OpenAI. Вікіпедія: веб-сайт. URL:: https://uk.wikipedia.org/wiki/OpenAI. (дата звернення: 07.11.2023).

34. Фронцкевич М. Майбутнє навчання закріплення з OpenAI Gym. TS2: веб-сайт. URL: http://surl.li/mvqjx. (дата зверення: 07.11.2023.).

35. TensorFlow. NOSC-UA Hub: веб-сайт. URL: http://cloud-5.bitp.kiev.ua/?page_id=598(дата зверення: 07.11.2023).

36. Гак П. Що таке Tensorflow. OKSIM: веб-сайт. URL: https://www.oksim.ua/2023/08/07/shho-take-tensorflow/. (дата зверення: 07.11.2023).

https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/

Додатки

TrackClassYolo.py - головний файл для детекції і трекінгу об’єктів

```
import cv2
from ultralytics import YOLO
from TargetListForDiplo import TargetList

class Traker:
    def __init__(self, target_list):
        self.yolo =
YOLO("C:/Users/Dima/Desktop/studyFoldert/nau/Ochka/diplo/readyDiplo/best.pt")
        self.target_list = target_list  # Зберігання списку цілей
        self.flag = True

    def process_frame(self, frame):
        # Виявлення та трекінг об'єктів за допомогою YOLO
        results = self.yolo.track(source=frame, show=False)
        self.target_list.update_targets(results)

    def process_video(self):
        cap = cv2.VideoCapture(0)

        while self.flag == True:
            ret, frame = cap.read()
            if not ret:
                break
            self.process_frame(frame)

        cap.release()
```

MyCustomENVForDiplo.py - головний файл для ініціалізації імітації середовища модуля

```
import time
import gym
from gym import spaces
import numpy as np
import time


class CustomEnv(gym.Env):
    """
    CustomEnv - середовище для тренування агентів з підкріпленням,
    яке імітує вибір та стрільби по цілям,.
    """

    def __init__(self, control, target_list):
        super(CustomEnv, self).__init__()
        self.control = control
        self.target_list = target_list  # Це об'єкт класу TargetList
        # Дві дії: вибір цілі (0) та стрільба (1)
```

```
        self.action_space = spaces.Discrete(2)
        # Простір спостережень: ID, координати bbox та відстань для кожної цілі
        self.observation_space = spaces.Box(low=-1, high=np.inf, shape=(50, 6),
dtype=np.float32)

        self.shots_on_target = {}  # Словник для зберігання кількості пострілів по
кожній цілі
        self.current_priority_target = None
        self.previous_priority_target = None
        self.reset()

    def set_target_id_for_control(self, target_id):
        """Встановлює ID цілі в Control."""
        self.control.set_current_target_id(target_id)




    def step(self, action):
        reward = 0
        done = False

        if action == 0:  # Вибір цілі
            self.selected_target = self.select_priority_target()
            if self.selected_target is not None:
                print(f"Вибрана ціль: ID {self.selected_target.track_id}")
                self.set_target_id_for_control(self.selected_target.track_id)
                print("вибыр цілі")
            reward = -1  # Пенальті за вибір цілі
        elif action == 1:  # Стрілянина
            if self.selected_target is not None:
                self.control.adjust_position()
                if self.control.is_aimed:
                    reward, done = self.shoot_target(self.selected_target.track_id)
                else:
                    print("Ціль ще не націлена.")
            else:
                print("Ціль не вибрана.")
                reward = -2  # Пенальті за спробу стрільбу без вибраної цілі

        # Оновлення стану спостереження
        self.state = self.update_observation()
        time.sleep(1)

        return self.state, reward, done, {}

    def reset(self):
        self.state = self.update_observation()
        self.selected_target = None
        self.shots_on_target.clear()
        self.current_priority_target = None
        self.previous_priority_target = None
```

```
        return self.state

    def select_priority_target(self):
        # Вибір цілі з найменшою відстанню з урахуванням кількості вистрілів
        valid_targets = [target for target in self.target_list.targets if
self.is_target_valid(target.track_id)]
        if not valid_targets:
            return None
        selected_target = min(valid_targets, key=lambda t: t.distance)
        self.update_priority_targets(selected_target.track_id)
        return selected_target

    def shoot_target(self, target_id):
        print("Ціль для стрільби ід:")
        print(target_id)
        print("Початок списку:")
        for target in self.target_list.targets:
            print(target.track_id)
        print("Кінець списку:")

        target = next((t for t in self.target_list.targets if t.track_id == target_id),
None)
        print(" Тут працюэш ?")
        if target:
                if self.control.is_inside_target():
                    self.shots_on_target[target_id] = self.shots_on_target.get(target_id,
0) + 1
                    time.sleep(1)
                    for targ in self.target_list:
                        if target_id not in targ.target_id:
                            is_destroyed = True
                    print("Стрільба: постріл здійснено")
                    reward = 10 if is_destroyed else -1
                    return reward, is_destroyed
                else:
                    print("не прицілено")
        else:
            print("Стрілянина: ціль не знайдена")
        return -10, False

    def update_observation(self):
        # Оновлення спостереження з поточними даними про цілі
        observation = np.full((50, 6), -1.0)  # Заповнення "пустими" значеннями
        for i, target in enumerate(self.target_list.targets):
            if i < 50:
                # Формат: [ID, x_min, y_min, x_max, y_max]
                observation[i] = [target.track_id, *target.bbox, target.distance]
        return observation

    def seed(self, seed=None):
        # Цей метод встановлює "зерно" генератора випадкових чисел
```

```
        pass

    def is_target_valid(self, target_id):
        # Перевірка, чи валідна ціль для стрілянини
        return (target_id != self.previous_priority_target or
                self.shots_on_target.get(target_id, 0) < 3)

    def update_priority_targets(self, selected_target_id):
        # Оновлення інформації про пріоритетні цілі та обнулення лічильника пострілів
        if selected_target_id != self.current_priority_target:
            if self.current_priority_target and
self.shots_on_target.get(self.current_priority_target, 0) >= 3:
                self.shots_on_target[self.current_priority_target] = 0
            self.previous_priority_target = self.current_priority_target
            self.current_priority_target = selected_target_id
```

control.py - файл для викинання націлювання

```
from Controler import Controler

class Control:
    def __init__(self, target_list, frame,AbstractBallisticCalculator,ctr):
        self.Ctr = Controler(ctr)
        self.target_list = target_list  # Посилання на список цілей
        self.current_target_id = None
        self.frame = frame
        self.aim_point = frame.shape[1] // 2, frame.shape[0] // 2
        self.is_aimed = False
        self.BallisticCalculator = AbstractBallisticCalculator



    def set_current_target_id(self, target_id):
        """Встановлює ID поточної цілі."""
        self.current_target_id = target_id

    def get_target_coords(self):
        """Отримує координати поточної цілі."""
        target = next((t for t in self.target_list.targets if t.track_id ==
self.current_target_id), None)
        return target.bbox if target else None

    def is_inside_target(self):
        """Перевіряє, чи aim_point знаходиться всередині квадрата поточної цілі."""
        target_coords = self.get_target_coords()
        if not target_coords:
            return False
```

```
        x1, y1, x2, y2 = target_coords
        return x1 <= self.aim_point[0] <= x2 and y1 <= self.aim_point[1] <= y2

    def shoot_at_target(self):
        """Ініціює постріл по цілі."""
        self.is_aimed = False
        self.Ctr.shot()

    def adjust_position(self):
        """Регулює позицію, намагаючись центрувати aim_point всередині квадрата цілі."""
        target_coords = self.get_target_coords()
        if not target_coords:
            return  # Ціль відсутня

        max_iterations = 500  # Максимальна кількість спроб прицілювання
        iterations = 0

        while not self.is_inside_target() and iterations < max_iterations:
            print("Йде прицілювання")
            x1, y1, x2, y2 = target_coords

            # Перевірка позиції aim_point відносно координат цілі та регулювання
положення
            if self.aim_point[0] < ((x2 + x1)/2) +
self.BallisticCalculator.calculate_adjustments(self.current_target_id)[0]:
                self.Ctr.move_right()
                self.get_target_coords()
            elif self.aim_point[0] >  ((x2 + x1)/2) +
self.BallisticCalculator.calculate_adjustments(self.current_target_id)[0]:
                self.Ctr.move_left()
                self.get_target_coords()

            if self.aim_point[1] < ((y2 + y1)/2) +
self.BallisticCalculator.calculate_adjustments(self.current_target_id)[1]:
                self.Ctr.move_up()
                self.get_target_coords()
            elif self.aim_point[1] > ((y2 + y1)/2) +
self.BallisticCalculator.calculate_adjustments(self.current_target_id)[1]:
                self.Ctr.move_down()
                self.get_target_coords()

            iterations += 1


        print(self.aim_point)
        if iterations >= max_iterations:
            print("Прицілювання не вдалося.")
        else:
            print(f"Ціль націлена: {self.aim_point}")

        self.is_aimed = self.is_inside_target()
```

Controler.py :

```
class Controler:

    def __init__(self,adapter):
        self.turret_adapter = adapter



    def move_up(self):
        self.turret_adapter.move_up()

    def move_down(self):
        self.turret_adapter.move_down()

    def move_left(self):
        self.turret_adapter.move_left()

    def move_right(self):
        self.turret_adapter.move_right()

    def shot(self):
        self.turret_adapter.shot()
```

TargetListForDiplo.py - файл для який описує зберігання знайдених об’єктів

```
class Target:
    def __init__(self, track_id, bbox):
        self.track_id = track_id  # Унікальний ідентифікатор об'єкта
        self.bbox = bbox          # Координати обрамлення об'єкта в форматі "ліва верхня
точка - права нижня точка"
        self.distance = 0        # Дистанція (початково нульова)

    def update(self, bbox):
        # Оновлення координат об'єкта
        self.bbox = bbox

    def set_distance(self, distance):
        # Встановлення дистанції
        self.distance = distance

from ultralytics import YOLO
class TargetList:
    def __init__(self):
        self.targets = []  # Список об'єктів Target

    def print_targets(self):
        print("Текущие цели:")
```

```
        for target in self.targets:
            print(f"ID: {target.track_id}, BBox: {target.bbox}, distance:
{target.distance}")

    def update_targets(self, yolo_tracks):
        # Оновлення та додавання нових об'єктів
        self.print_targets()
        if yolo_tracks is not None:
            for track in yolo_tracks:
                if track is not None and track.boxes.id is not None:
                    track_id = track.boxes.id.int().cpu().tolist()[0]
                    xywh = track.boxes.xywh.cpu()[0]
                    xywh_list = xywh.tolist()
                    x, y, w, h = xywh
                    x1 = int(x - w / 2)  # Обчислення лівої верхньої точки (x1, y1)
                    y1 = int(y - h / 2)
                    x2 = int(x + w / 2)  # Обчислення правої нижньої точки (x2, y2)
                    y2 = int(y + h / 2)
                    bbox = x1,y1,x2,y2
                    found = False

                    for target in self.targets:
                        if target.track_id == track_id:
                            target.update(bbox)
                            found = True
                            break

                    if not found:
                        new_target = Target(track_id, bbox)
                        self.targets.append(new_target)


        # Видалення об'єктів, які втратилися
                    lost_track_ids = [track.boxes.id.int().cpu().tolist()[0] for track
in yolo_tracks]
                    self.targets = [target for target in self.targets if target.track_id
in lost_track_ids]
                else:
                    self.targets = []
```

GUI - основний клас в програми ініціює роботу програми

```
import sys
import cv2
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
QPushButton, QRadioButton, QListWidget, QLabel, QListWidgetItem, QSizePolicy
from PyQt5.QtCore import QTimer, Qt, QThread, pyqtSignal
from PyQt5.QtGui import QImage, QPixmap
import gym
```

```python
from stable_baselines3 import PPO
from MyCustomENVForDiplo import CustomEnv
from TrackClassYolo import Traker
from TargetListForDiplo import TargetList
from Weather import Weather
from WeatherStantion import WeatherStation
from control import Control
from BalisticCalculator import AbstractBallisticCalculator
from TurellAdpater import TurretAdapter
from Turel import Turrel

class VideoThread(QThread):
    finished_signal = pyqtSignal()

    def __init__(self, tracker):
        super().__init__()
        self.tracker = tracker

    def run(self):
        while not self.isInterruptionRequested():
            self.tracker.process_video()
        self.finished_signal.emit()



class GymThread(QThread):
    def __init__(self, env, model):
        super().__init__()
        self.env = env
        self.model = model
        self.running = True  # Прапорець для контролю стану потоку

    def run(self):
        obs = self.env.reset()
        while self.running:
            print("Я працюю")
            action, _states = self.model.predict(obs, deterministic=True)
            obs, reward, done, info = self.env.step(action)
            if done:
                obs = self.env.reset()

    def stop(self):
        self.running = False

class VideoStreamWidget(QWidget):
    def __init__(self, targetList, parent=None):
        super().__init__(parent)
        self.video_label = QLabel("Відеопотік")
        self.layout = QVBoxLayout()
        self.targetList = targetList
        self.layout.addWidget(self.video_label)
```

```
        self.setLayout(self.layout)

        # Налаштування відеопотоку
        self.capture = cv2.VideoCapture(0)
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_frame)
        self.timer.timeout.connect(self.set_squares)

        self.timer.start(20)
        self.id_list = []
        self.squares = []

    def set_squares(self):
        self.squares = []
        self.id_list = []
        for tar in self.targetList.targets:
            bbox_list = tar.bbox
            id = tar.track_id
            x1, y1, x2, y2 = map(int, bbox_list)
            self.squares.append([x1, y1, x2, y2])
            self.id_list.append(id)

    def update_frame(self):
        ret, frame = self.capture.read()
        if ret:
            for (x, y, w, h) in self.squares:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

            for id in self.id_list:
                cv2.putText(frame, f"ID: {id}", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255, 255), 2)

            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image = QImage(frame, frame.shape[1], frame.shape[0], QImage.Format_RGB888)
            self.video_label.setPixmap(QPixmap.fromImage(image))


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("ШІ додаток")
        self.setGeometry(100, 100, 900, 600)
        self.target_List = TargetList()
        self.WeatherStation = WeatherStation()
        self.Weather = Weather(self.WeatherStation)
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        main_layout = QHBoxLayout(central_widget)
        self.cap = cv2.VideoCapture(0)
        _, self.frm = self.cap.read()
```

```
        self.Turrel = Turrel()
        self.TurretAdapter = TurretAdapter(self.Turrel)
        self.BallisticCalculator =
AbstractBallisticCalculator(self.Weather,self.target_List)
        self.Control =
Control(self.target_List,self.frm,self.BallisticCalculator,self.TurretAdapter)
        self.CustomEnv = CustomEnv(self.Control,self.target_List)
        self.list_widget = QListWidget()
        main_layout.addWidget(self.list_widget)
        self.list_widget.itemClicked.connect(self.on_item_clicked)


        self.stream_widget = VideoStreamWidget(self.target_List)
        main_layout.addWidget(self.stream_widget)

        control_layout = QVBoxLayout()
        self.shoot_button = QPushButton("Стріляти")
        self.shoot_button.clicked.connect(self.shoot)
        control_layout.addWidget(self.shoot_button)


        self.radio_button = QRadioButton("Автономна робота")
        control_layout.addWidget(self.radio_button)
        self.radio_button.toggled.connect(self.onRadioButtonToggled)


        self.stream_widget.layout.addLayout(control_layout)

        self.weather_data_layout = QVBoxLayout()
        self.weather_data_layout.setSpacing(10)  # Встановіть відстань між елементами

        self.weather_data_title = QLabel("Метеодані")
        self.weather_data_layout.addWidget(self.weather_data_title)

        # Приклад значень метеоданих

        self.temperature_label = QLabel("Температура: --°C")
        self.humidity_label = QLabel("Вологість: --%")
        self.pressure_label = QLabel("Тиск: --- мм рт. ст.")
        self.wind_speed = QLabel("Швидкість вітру: --м./год.")
        self.wind_direction = QLabel("Напрям вітру: --кут")
        self.altitude = QLabel("Висота над рівнем моря: --- м.")
        self.weather_data_layout.addWidget(self.temperature_label)
        self.weather_data_layout.addWidget(self.humidity_label)
        self.weather_data_layout.addWidget(self.pressure_label)
        self.weather_data_layout.addWidget(self.wind_speed)
        self.weather_data_layout.addWidget(self.wind_direction)
        self.weather_data_layout.addWidget(self.altitude)

        self.weather_data_layout.addStretch()  # Додаємо гнучке простір в кінці

        # Додавання метеоданих до правої частини головної компоновки
```

```
        main_layout.addLayout(self.weather_data_layout)

        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_weather_data)
        self.timer.start(10000)  # Оновлення кожні 10 секунд


        self.target_items = {}  # Словник для збереження відповідності між ID та
елементами списку
        self.update_target_list_timer = QTimer(self)
        self.update_target_list_timer.timeout.connect(self.update_target_list)
        self.update_target_list_timer.start(1000)  # Оновлення кожну секунду
        tracker = Traker(self.target_List)  # Створення нового екземпляру Tracker
        self.video_thread = VideoThread(tracker)
        self.video_thread.start()

    def onRadioButtonToggled(self, checked):
        if checked:
            env = self.CustomEnv
            model = PPO.load("ppo_custom_env_model.zip")
            self.gym_thread = GymThread(env, model)
            self.gym_thread.start()
        else:
            if hasattr(self, 'gym_thread'):
                self.gym_thread.stop()



    def shoot(self):
        print("кнопка нажата")
        self.Control.shoot_at_target()

    def update_target_list(self):
        current_ids = set([target.track_id for target in self.target_List.targets])
        existing_ids = set(self.target_items.keys())

        for target in self.target_List.targets:
            item_text = f"ID: {target.track_id}, Координати: {target.bbox}"
            if target.track_id in existing_ids:
                self.target_items[target.track_id].setText(item_text)
            else:
                item = QListWidgetItem(item_text)
                self.list_widget.addItem(item)
                self.target_items[target.track_id] = item

        for id_to_remove in existing_ids - current_ids:
            item = self.target_items.pop(id_to_remove)
            self.list_widget.takeItem(self.list_widget.row(item))



    def on_item_clicked(self, item):
```

```
        print(f"Вибрано: {item.text()}")
        if not self.radio_button.isChecked():
            print(f"Вибрано (радіо-батон не активовано): {item.text()}")
            item_text = item.text()
            item_parts = item_text.split(', ')
            id_part = item_parts[0]
            track_id = id_part.split(': ')[1]
            track_id = int(track_id)
            self.Control.set_current_target_id(track_id)
            self.Control.adjust_position()
        else:
            print(f"Вибрано: {item.text()}")


    def update_weather_data(self):
        # Отримання та відображення оновлених даних про погоду
        self.Weather.continuously_update_data(False)
        self.temperature_label.setText(f"Температура: {self.Weather.temperature}°C")
        self.humidity_label.setText(f"Вологість: {self.Weather.humidity}%")
        self.pressure_label.setText(f"Тиск: {self.Weather.pressure} мм рт. ст.")
        self.wind_speed.setText(f"Швидкість вітру: {self.Weather.wind_speed}km/h")
        self.wind_direction.setText(f"Напрям вітору: {self.Weather.wind_direction}°")
        self.altitude.setText(f"Вистора над рівнем моря: {self.Weather.altitude} m.")


app = QApplication(sys.argv)
main_window = MainWindow()
main_window.show()
sys.exit(app.exec_())
```