**Міністерство освіти і науки України**

**Чернівецький національний університет імені Юрія Федьковича**

**Інститут фізико-технічних та комп’ютерних наук**

(повна назва інституту/факультету)

**Кафедра комп’ютерних наук**

(повна назва кафедри)

ДИПЛОМНА РОБОТА

з дисципліни

«»

на тему: «Виявлення моделей персонажів гри та часткове керування на основі алгоритмів YOLOv8 та WindMouse»

Виконав: студент 4 курсу, групи 444 (СК)

Напряму підготовки 122 «Комп’ютерні науки»

(шифр і назва спеціальності)

Максимович Микола Юрійович .

(прізвище та ініціали)

Керівник \_к.б.н., асистент Дворжак В.В.\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Оцінка: ECTS\_\_\_\_\_\_\_

Чернівці – 2023

**Чернівецький національний університет імені Юрія Федьковича**

Факультет: Інститут фізико-технічних та комп’ютених наук

Кафедра: Комп’ютерних наук Спеціальність 122 «Комп’ютерні науки»

Освітній ступінь Бакалавр Форма навчання Денна Курс 4 Група 444ск

ЗАТВЕРДЖУЮ

Завідувач кафедри

|  |  |  |
| --- | --- | --- |
|  |  | Ушенко Ю.О. |

( *підпис*) (*ініціали, прізвище*)

« » 2022 року

**ЗАВДАННЯ**

**НА КУРСОВУ РОБОТУ СТУДЕНТА**

|  |
| --- |
| **Максимовича Миколи Юрійовича** |

(прізвище , ім’я, по батькові)

1. Тема роботи

|  |
| --- |
| Виявлення моделей персонажів гри та часткове керування на основі алгоритмів YOLOv8 та WindMouse |

затверджена наказом від « » 2022 року №

1. Термін подання студентом закінченої роботи: 14.12.2022
2. Вхідні дані до роботи

|  |
| --- |
| Набір зображень, мітки до зображень, натреновані ваги моделі Yolov8, |
| головний скрипт. |

1. Зміст розрахунково-пояснювальної записки (перелік питань, які треба

розробити)

|  |
| --- |
| Вступ |
| Розділ 1 ТЕОРЕТИЧНА ЧАСТИНА |
| Розділ 2 ПРОЕКТУВАННЯ |
| Розділ 3 ДЕМОНСТРАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ МОДУЛЮ |
| ВИСНОВКИ |
| ДОДАТКИ |

1. Перелік графічного, наочного матеріалу

|  |
| --- |
| Діаграма поведінки додатку; Скріншоти результатів навчання; |
| Скріншоти оцінки якості; графіки оцінки метриками: Precision, Recal, mAP, |
| F1; Лістинг програми; |

1. Консультант(и) курсової роботи

|  |
| --- |
|  |

Календарний план підготовки курсової роботи

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсової роботи | Термін виконання етапів роботи | Примітки |
| 1 | Отримання завдання на курсову роботу | 08.09.2022 | виконано |
| 2 | Аналіз предметної області, дослідження літератури та матеріалів на задану тему | 22.09.2022 | виконано |
| 3 | Аналіз існуючих аналогів програмного забезпечення | 01.10.2022 | виконано |
| 4 | Постановка задачі за темою курсової роботи | 06.10.2022 | виконано |
| 5 | Вибір інструментальних засобів розробки системи | 15.10.2022 | виконано |
| 6 | Проєктування структури та алгоритму роботи розроблюваної системи | 21.10.2022 | виконано |
| 7 | Розробка компонентів системи | 10.11.2022 | виконано |
| 8 | Розгортання та тестування розробленої системи | 25.11.2022 | виконано |
| 9 | Написання розділів пояснювальної записки | 01.12.2022 | виконано |
| 10 | Представлення закінченої роботи на перевірку | 13.12.2022 | виконано |
| 11 | Захист курсової роботи | 14.12.2022 | виконано |

**Студент** Максимович М.Ю

(*підпис*)

**Керівник**  Дворжак В.В.

(*підпис*)

« » 2022 року

**АНОТАЦІЯ**

У дипломній роботі розроблено інформаційну систему, що забезпечує аналіз освітньої діяльності закладів вищої освіти з використанням методів інтелектуального аналізу даних. Досліджено діяльність закладів вищої освіти в Україні. Проаналізовано вимоги до інформаційних систем аналізу, здійснено характеристику їх функціональних можливостей, обґрунтовано вибір технології Виявлення Об’єктів (Object Detection) для розпізнавання та виявлення позиції класу на зображенні. Обґрунтовано вибір потоку фотографій у якості джерела вхідних даних, продемонстровано підходи до їх отримання. Показано ефективність застосування алгоритмів train/val/test/split, а також функції сайту RoboFlow для створення міток на зображеннях та обробки масиву вхідних даних. Налаштування параметрів моделі проводилось вручну користуючись порадами від розробника. Проведена оцінка якості моделі показала 96%mAP(mean Average Precision). З врахуванням поставлених завдань та форматів отриманих даних були обрані наступні види візуалізацій:

* Real time, створено код для обробки нейромережею зображень в реальному часі з екрану монітора.
* Результати роботи мережі на картинках.
* confusion\_matrix
* F1 curve
* P curve
* R curve
* PR curve

Пояснювальна записка складається з трьох розділів загальним обсягом 31 сторінки, містить 0 таблиць, 3 додатки, 10 використаних джерел.

**Ключові слова**: Data Mining, Python, PyTorch, алгоритм навчання, Object Detection, Yolo, комп’ютерний зір, заклад вищої освіти

ЗМІСТ

[Вступ 3](#_Toc121999025)

[Розділ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, АЛГОРИТМІВ ТА ЗАСОБІВ РОЗРОБКИ 5](#_Toc121999026)

[1.1 Актуальність розробки програмного продукту 5](#_Toc121999027)

[1.2 Огляд та аналіз об’єктів дослідження 6](#_Toc121999028)

[1.3 Способи використання об’єктів дослідження в житті 7](#_Toc121999029)

[Розділ 2. Проектування програмного продукту 10](#_Toc121999030)

[2.1 Опис предметної області 10](#_Toc121999031)

[2.2 Засоби та технології розробки і причини їх використання 10](#_Toc121999032)

[2.3 Етапи створення продукту 13](#_Toc121999033)

[2.4 Опис вхідних та вихідних даних 14](#_Toc121999034)

[Формування набору навчальних зображень 14](#_Toc121999035)

[Архітектура нейромережі Yolov7 15](#_Toc121999036)

[Процес розробки головного скрипту 15](#_Toc121999037)

[Розділ 3. ДЕМОНСТРАЦІЯ результатів РОБОТИ МОДУЛЮ 17](#_Toc121999038)

[3.1 Результати обробки зображень 17](#_Toc121999039)

[3.2 Процес навчання нейромережі 17](#_Toc121999040)

[3.3 Оцінка якості навчання 18](#_Toc121999041)

[3.4 Інструкція використання 22](#_Toc121999042)

[Висновки 23](#_Toc121999043)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 24](#_Toc121999044)

[Додатки 25](#_Toc121999045)

[Додаток А 25](#_Toc121999046)

[Додаток Б 26](#_Toc121999047)

[Додаток В 29](#_Toc121999048)

# Вступ

Завданням курсової роботи є розробка додатку з використанням нейромережі для класифікації та виявлення позиції моделей персонажів в грі CSGO.

Об’єктом дослідження курсової роботи є нейронна мережа для виявлення моделей персонажів в грі CSGO. Предметом дослідження курсової роботи є тренування нейромережі типу Object Detection, покращення якості навчання та збільшення ефективності при використанні в реальному часі.

Метою курсової роботи є поглиблення знань в сфері нейромереж та їх здобуття в галузі «Object Detection» і комп’ютерного зору. Також удосконалення практичних навичок повного циклу обробки даних, таких як: підготовка та обробка інформації, навчання нейромережі і покращення її точності, оцінка якості, візуалізація результатів.

Основними задачами додатку є:

* Виявлення моделей гравців на екрані за допомогою комп’ютерного зору.
* Класифікація «свій, чужий» для уникнення стрільби по своїх.

До принципових задач відносять:

* Покращення якості, точності моделі.
* Збільшення ефективності комп’ютерного зору за рахунок зменшення часу обробки кадру.

До функціональних задач відносять:

* Створення набору картинок
* Створення міток класів
* Розбиття набору даних на 3 групи – навчальна, валідаційна, тестова
* Тренування нейромережі
* Перевірка якості нейромережі
* Підключення комп’ютерного зору до нейромережі
* Обробка даних отриманих з нейромережі
* Створення скрипту автонаведення

Вимоги надійності.

* Програмний продукт повинен надійно працювати;
* Забезпечити високу точність класифікації, понад 90%.
* Забезпечити високу швидкість реакції

Додаток працюватиме лише на операційній системі Windows. Однак код тренування нейромережі можна запускати там, де є встановлений python, pytorch та yolov8.

Для реалізації програмного продукту використовувалось середовище розробки Visual Studio Code, також для тренування нейромережі використовувався сервіс GoogleColab. Для створення міток на наборі даних використовувався веб сайт RoboFlow, з допомогою його функції було розподілено дані на тренувальну, валідаційну та тестову вибірку. Для реалізації основної задачі виявлення об’єктів використовувався алгоритм Yolov8. Для його функціювання стала у пригоді бібліотека PyTorch. А для захоплення екрану використовувалась бібліотека DxCam. Системою контролю версій було обрано Git.

## 1.1 Актуальність розробки програмного продукту

13 квітня 2019 року команда OG, формально найсильніша в багатокористувацькій онлайн-грі Dota 2, провела показовий матч проти "бітового" противника - алгоритму машинного навчання Open AI. Роботи перемогли з рахунком 2:0. Нейронна мережа вже грала проти людей, але в серпні 2018 року OpenAI Five програла всі матчі.

Машини стали сильнішими за людей у шашках після 1994 року, у шахах - з 1997-го, а Го остаточно підкорили у 2017-му. Того ж 2017-го алгоритм перевершив людей у грі в безлімітний техаський холдем (один із варіантів покеру). У ці ж роки розробники стали намагатися натаскати нейромережі на відеоігри: творці AlphaGo займалися розробкою "бітового гравця" в Starcraft 2, а OpenAI (серед засновників якої був Ілон Маск) взялися за "Доту".

У 2017 році бота випробували в обмеженому режимі "дзеркальної" дуелі (противники грали однаковими персонажами). Почавши з регулярних програшів любителям у березні, бот OpenAI закінчив тим, що в серпні переміг найкращих гравців на планеті: після матчів люди говорили, що бота просто неможливо перемогти.

У 2018-му OpenAI спробувала свої сили вже в нормальній командній грі (щоправда, теж із низкою обмежень), але зазнала поразки. На якийсь час люди відстояли свою перевагу над штучним інтелектом.

13 квітня 2019 року роботи взяли реванш, обігравши команду OG, яка раніше стала чемпіоном The International, найбільшого турніру з Dota 2, що щорічно збирає найкращі команди з усієї планети. Матч проходив у Каліфорнії, його запис можна подивитися на платформі Twitch.

Після матчу розробники повідомили, що, готуючись до нього, алгоритм грав і з іншими професійними командами, також вигравши із рахунком 2:0.

Саме тому, натхненний цією історією, моєю темою було обрано: розробити нейромережу для виявлення персонажів в грі CSGO. А в подальшому поліпшити її до повноцінного бота, який буде самостійно грати проти справжніх людей.

# Розділ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, АЛГОРИТМІВ ТА ЗАСОБІВ РОЗРОБКИ

## 1.2 Огляд та аналіз об’єктів дослідження

**Штучні нейронні мережі** (ШНМ, англ. artificial neural networks, ANN), або конективістські системи (англ. connectionist systems) — це обчислювальні системи, натхненні біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони обробляють.

ШНМ ґрунтується на сукупності з'єднаних вузлів, що називають штучними нейронами (аналогічно до біологічних нейронів у головному мозку тварин). Кожне з'єднання (аналогічне синапсові) між штучними нейронами може передавати сигнал від одного до іншого. Штучний нейрон, що отримує сигнал, може обробляти його й потім сигналізувати штучним нейронам, приєднаним до нього [1].

**Виявляння об'єктів** (англ. object detection) — це комп'ютерна технологія, пов'язана з комп'ютерним баченням та обробкою зображень, яка має справу з виявлянням примірників семантичних об'єктів певного класу (таких як люди, будівлі чи автомобілі) у цифрових зображеннях та відео. До добре досліджених областей виявляння об'єктів належать виявляння облич та виявляння пішоходів. Виявляння об'єктів має застосування у багатьох сферах комп'ютерного бачення, включно з пошуком зображень та відеоспостереженням [2].

**Комп'ютерний зір або Комп'ютерне бачення** — теорія та технологія створення машин, які можуть проводити виявляння, відстежування та визначення об'єктів.

Як наукова дисципліна комп'ютерний зір належить до теорії та технології створення штучних систем, які отримують інформацію у вигляді зображень. Відеодані можуть бути представлені у вигляді багатьох форм, таких як відеопослідовність, зображення з різних камер або тривимірними даними з медичного сканера.

Як технологічна дисципліна комп'ютерний зір прагне застосувати теорії та моделі комп'ютерного зору до створення систем комп'ютерного зору. Прикладами таких систем можуть бути:

1. системи керування процесами (промислові роботи, автономні транспортні засоби)
2. системи відеоспостереження
3. системи організації інформації (наприклад, для індексації баз даних зображень)
4. системи моделювання об'єктів або навколишнього середовища (аналіз медичних зображень, топографічне моделювання)
5. системи взаємодії (наприклад, пристрої введення для систем людино-машинної взаємодії).

Комп'ютерний зір також може бути описаний як доповнення (але не обов'язково протилежність) біологічному зору. У біології вивчається зорове сприйняття людини і різноманітних тварин, в підсумку створюються моделі роботи таких систем в термінах фізіологічних процесів. Комп'ютерний зір, з іншого боку, вивчає і описує системи комп'ютерного зору, які виконано апаратно або програмно. Міждисциплінарний обмін між біологічним та комп'ютерним зором виявився досить продуктивним для обох наукових галузей.

Підрозділи комп'ютерного зору охоплюють відтворення дій, виявлення подій, стеження, розпізнавання образів, відновлення зображень [3].

## 1.3 Способи використання об’єктів дослідження в житті

Через свою здатність відтворювати та моделювати нелінійні процеси, ШНМ знайшли застосування в широкому діапазоні дисциплін.

До областей застосування належать: ідентифікація систем та керування (керування транспортними засобами, передбачування траєкторії, автоматизація виробничих процесів, природокористування), квантова хімія, гра в ігри та ухвалення рішень (короткі нарди, шахи, покер), розпізнавання образів (радарні системи, ідентифікація облич, класифікація сигналів, розпізнавання об'єктів та ін.), розпізнавання послідовностей (жестів, мовлення, рукописного тексту), медична діагностика, фінанси (наприклад, автоматизовані системи торгівлі), добування даних, унаочнення, машинний переклад, соціально-мережеве фільтрування та фільтрування спаму електронної пошти.

ШНМ застосовували в діагностуванні раку, включно з раком легені, простати, колоректальним раком, а також щоби відрізняти лінії ракових клітин, сильно схильні до розповсюдження, від менш схильних до розповсюдження ліній, із застосуванням лише інформації про форму клітин.

ШНМ також використовували для побудови чорноскринькових моделей в геонауках: гідрологія, моделювання океану та прибережна інженерія та геоморфологія є лише деякими з прикладів такого роду.

**Виявляння об'єктів** широко використовують у задачах комп'ютерного бачення, таких як анотування зображень, підрахунок транспортних засобів, розпізнавання діяльності, виявляння облич, розпізнавання облич, співсегментування об'єктів у відео. Його також використовують у відстежуванні об'єктів, наприклад відстежуванні м'яча під час футбольного матчу, відстежуванні руху біти для крикету, або відстежуванні особи на відео.

Приклади застосування комп'ютерного зору:

Одним з найбільш важливих застосувань є обробка зображень в медицині. Ця область характеризується отриманням інформації з відеоданих для визначення медичного діагнозу пацієнту.

Іншою прикладною галуззю комп'ютерного зору є промисловість. Тут інформацію отримують для підтримки виробничого процесу. Прикладом може слугувати контроль якості, коли деталі чи кінцевий продукт автоматично перевіряють на наявність дефектів. Іншим прикладом є вимірювання положення та орієнтація деталей, які піднімає рука робота.

Військове застосування є, мабуть, найбільшою областю комп'ютерного зору. Очевидним прикладом є виявлення ворожих солдатів і транспортних засобів та керування ракетами. Найбільш досконалі системи керування ракетами відправляють ракету в задану область, замість конкретної цілі, а визначення цілей відбувається тоді, коли ракета досягає заданої області, базуючись на відеоданих, що надходять.

Одними з нових галузей застосування є автономні транспортні засоби: підводні, наземні (роботи, машини), повітряні. Рівень автономності вимірюється від повністю автономних (безпілотних) до транспортних засобів, де системи, що базуються на комп'ютерному баченні, підтримують водія чи пілота в різноманітних подіях. Повністю автономні транспортні засоби використовують комп'ютерне бачення для навігації, тобто для отримання інформації про місце свого положення, для створення мапи навколишнього оточення, для визначення перешкод. Вони також можуть бути використані, наприклад, для визначених завдань знаходження лісових пожеж. Прикладом таких систем, можуть бути: система попереджувальної сигналізації про перешкоди на машинах і системи автономної посадки літаків.

Інші області застосування охоплюють:

* підтримку створення відео ефектів для кіно та телебачення;
* спостереження.

# Розділ 2. Проектування програмного продукту

## 2.1 Опис предметної області

Основним завданням реалізації програмного продукту було створення набору зображень, їх маркування та навчання нейромережі. Основною її перевагою є швидкість обробки зображення.

Окрім навчання було створено скрипт для захоплення зображення екрану та його передачі до нейромережі на обробку, класифікацію та виявлення позицій об’єктів. З використанням цих даних алгоритм буде наводитись на ворога та проводити вистріл.

Заплановані можливості користувачів:

* При певному рівні знань користувач має змогу запустити тренування мережі на вже готовому наборі даних. Якщо в нього є власний набір, можна тренувати на ньому за умови, що дані розмічені.
* Основна можливість - це перегляд вихідних даних мережі в окремому вікні. В ньому буде візуалізовано інформацію про розміщення персонажів на екрані та їх відповідність до певного класу.

## 2.2 Засоби та технології розробки і причини їх використання

Для реалізації програмного продукту обрано середовище розробки Visual Studio Code.

**Visual Studio Code** — засіб для створення, редагування та налагодження програми сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Компанія Microsoft представила Visual Studio Code у квітні 2015р на конференції Build 2015. Це середовище розробки стало першим кросплатформовим продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проєкту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовує браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проєкту Atom і на сайті Visual Studio Code, і в пресрелізі, і в офіційному блозі не згадується.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js і позиціюється як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Підтримуваних мов і технологій досить велика кількість.

Основною моделлю нейромережі в курсовій роботі є **YOLOv8**. Про цей алгоритми чув кожен, хто має середні знання про комп'ютерний зір. YOLO розшифровується як You Only Look Once, а v8 означає восьму версію алгоритму.

**Object Detection - Чому саме YOLO?**

**YOLO** - це алгоритм виявлення об'єктів, який використовує PyTorch як основу для кодування. Він відомий тим, що виявляє об'єкти в режимі реального часу.

YOLO - це чудовий алгоритм, який дає рішення для багатьох реальних проблем комп'ютерного зору. YOLO використовувався для розпізнавання сигналів світлофора, тестування на іспитах, ігрових аімботів та різних інструментів промислової автоматизації [4].

YOLO залишається однією з найкращих мереж виявлення об'єктів з моменту її створення з двох основних причин: точність, відносно низька вартість і простота використання. Ці риси разом зробили YOLO, безсумнівно, однією з найвідоміших моделей DL за межами спільноти науки про дані, в цілому завдяки цьому корисному поєднанню. Пройшовши кілька ітерацій розвитку, YOLOv8 є останньою версією популярного алгоритму і значно покращує свої попередники [5].

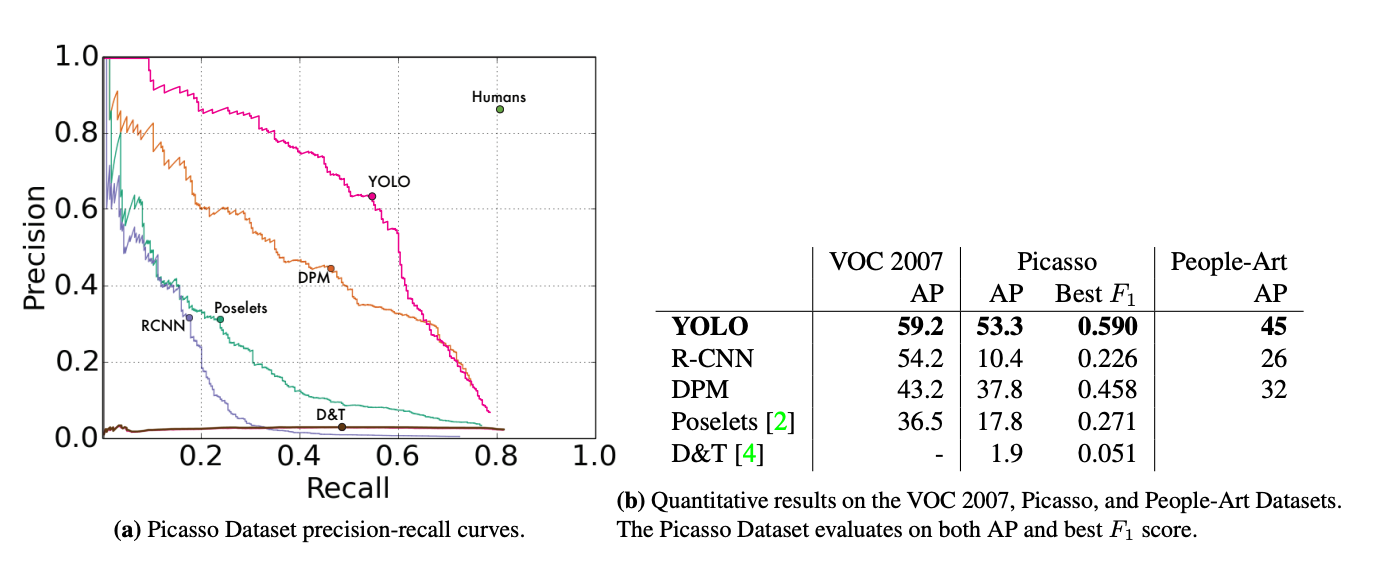


Рисунок 2.1 – Узагальнення результатів щодо наборів даних Picasso та People-Art з оригінальної статті YOLO [6].

Підсумовуючи, можна сказати що Yolo найкращий варіант для виявлення об’єктів на екрані в режимі реального часу, завдяки високій точності та швидкості.

**PyTorch** — відкрита бібліотека машинного навчання на основі бібліотеки Torch, що використовують для таких застосувань, як комп'ютерне бачення та обробка природної мови. Розробляє її переважно група дослідження штучного інтелекту компанії Facebook. Вона є вільним та відкритим програмним забезпеченням, що випускають під ліцензією Modified BSD. І хоча інтерфейс Python є більш відшліфованим і головним зосередженням розробки, PyTorch також має зовнішній інтерфейс і для C++. Крім того, програмне забезпечення ймовірнісної мови програмування Pyro компанії Uber використовує PyTorch як внутрішній інтерфейс [7].

PyTorch забезпечує дві високорівневі функціональності:

* Тензорні обчислення (як NumPy) із сильним прискоренням через графічні процесори (ГП)
* Глибинні нейронні мережі, побудовані на системі автоматичного диференціювання на основі плівки.

Для захоплення зображення з екрану потрібна була максимальна продуктивність, тому було обрано найшвидший варіант – DxCam.

**DXcam** – високопродуктивна бібліотека скріншотів для Windows на мові Python, що використовує Desktop Duplication API, яка здатна робити скріншоти з частотою 240 Гц+. Спочатку вона була створена як частина конвеєра глибокого навчання для FPS ігор, щоб працювати краще, ніж існуючі рішення на python (python-mss, D3DShot) [8].

У порівнянні з цими існуючими рішеннями DXcam забезпечує:

* Набагато більшу швидкість захоплення екрану (> 240 Гц)
* Захоплення ексклюзивних Direct3D повноекранних додатків без переривання, навіть при натисканні клавіш alt+tab.
* Автоматичну обробку масштабованої / розтягнутої роздільної здатності.
* Точне націлювання FPS в режимі захоплення, що робить його придатним для виведення відео.
* Безшовна інтеграція з NumPy, OpenCV, PyTorch і т.д.

**OpenCV** (англ. Open Source Computer Vision Library, бібліотека комп'ютерного зору з відкритим кодом) — бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом.

Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях [9,10].

**Алгоритм WindMouse** натхненний шкільним курсом фізики. Головне що потрібно знати, це те, що курсор моделюється як об'єкт з певною інерцією (масою), на який діють дві сили:

Гравітація, яка є постійною за величиною (параметр, що налаштовується) і завжди спрямована в бік кінцевого пункту призначення.

Вітер, який діє з випадковою силою у випадковому напрямку і плавно змінює як величину, так і напрямок з плином часу.

## 2.3 Етапи створення продукту

Створення продукту відбулось в декілька етапів:

1. Підготовка набору даних зображень.

Зображення спочатку було обрано з інтернету. Але для ефективного навчання потрібна велика кількість зображень. Тому було розроблено python скрипт для створення картинок при натисканні на попередньо задану клавішу.

1. Створення міток класів на зображення.

Це потрібно для того, щоб мережа могла не лише класифікувати об’єкт, а ще і вказувати його розташування. Одне із найдовших та найбільш нудних занять. На обробку 160 картинок витрачено приблизно 2 години.

1. Тренування моделі Yolov8 на підготовленому наборі даних. Для цього використовувалось хмарне середовище розробки «GoogleColab». З його допомогою можна застосовувати надану потужну відеокарту для тренування моделі.
2. Оцінка якості моделі.
3. Покращення моделі.
4. Розробка головного скрипту. Його ціль - захоплення екрану, передача картинки в нейромережу, отримання результатів, обробка, а саме малювання обмежуючих рамок навколо об’єктів та вивід картинки на екран в окреме вікно, а наприкінці, з допомогою координат із нейромережі скрипт виконуватиме автоматичне наведення на об’єкт та постріл.
5. Виправлення помилок.

## 2.4 Опис вхідних та вихідних даних

### Формування набору навчальних зображень

Спочатку потрібно було створити набір даних із зображеннями моделей спецназу і терористів. Спершу було взято фото з інтернету, але не вистачало різноманіття і це було дуже довго та нудно.

Тому було вирішено створити скрипт «набивка бази». При натисканні на попередньо задану кнопку, скрипт робив скріншот та зберігав в задану папку. Використовувалося це для того, щоб робити скріншоти зображення прямо з гри. Це дозволило пришвидшити створення бази в декілька разів.

Наступним, та можливо найдовшим і найнуднішим пунктом, було створення міток класів на зображеннях. Мітки записувались в окремому файлі і містили інформацію про розташування і імена класів на картинці. Кожне зображення мало свій власний файл з мітками з розширенням .txt, а назвою такою ж як і у зображення.

Для спрощення підготовки та обробки зображень було використано інтернет сервіс RoboFlow. Розібратись на сайті було не проблемою, так як при його запуску користувачу зразу пропонують коротке та зрозуміле керівництво з використанням його функціоналу. Але навіть з використанням цього сервісу, створення 100 міток зайняло 1 годину. Загалом було розташовано мітки на близько 160 зображень.

За допомогою функцій сайту було виділено 10% на валідаційну вибірку та 10% на тестову. Також було застосовано обробку зображень. Після чого алгоритм потроїв кількість зображень, використовуючи наступні параметри:

* Масштабування зображення до 1024х1024, з чорними полосами на менших краях.
* Нахил в сторони 15 градусів
* Зміна яскравості на 30%
* Зміна насиченості кольорів на 50%
* Розмиття картинки максимум до 1.5px

### Архітектура нейромережі Yolov8

Оскільки код дуже громіздкий - була висвітлена лише його частина в додатку В. Можна замітити, що модель складається повністю зі згорткових шарів та шарів підвибірки.

### Процес розробки головного скрипту

Скрипт розроблявся і модифікувався під час написання курсової. Основним його завданням було: запис екрану, передача кожного кадру в нейромережу, обробка результату та виведення його в окреме вікно.

На рисунку 2.1 зображено модель поведінки скрипту:

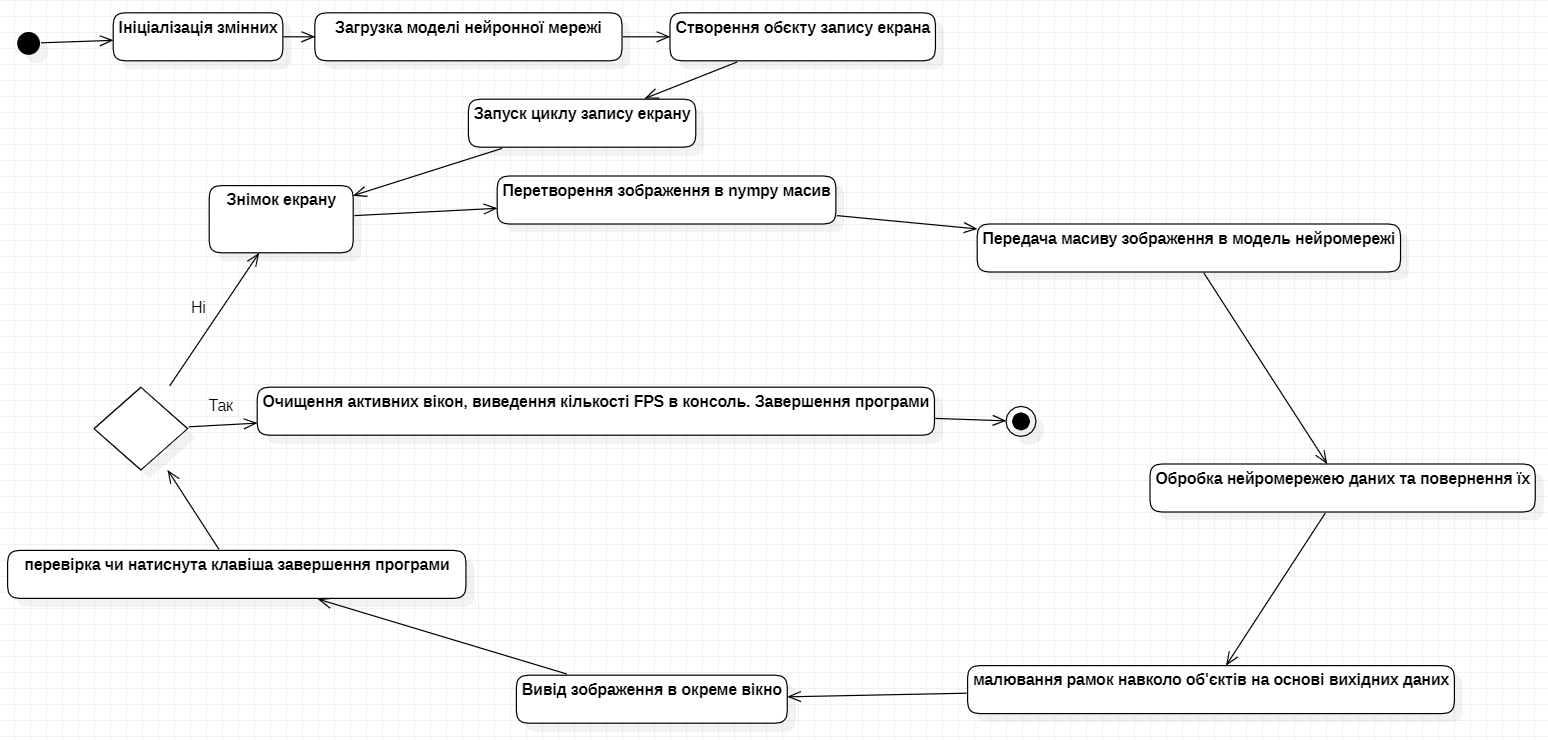


Рисунок 2.2 – Модель поведінки скрипту

Під час розробки було випробувано декілька рішень виконання певної задачі, відкинуті найгірші та обрані найкращі. Приклад: захоплення екрану. Спочатку планувалось використовувати функцію grab() з бібліотеки pyautogui. Але ефективність була дуже низькою, адже це була бібліотека для створення скірншотів екрану, а не його запису.

Наступною бібліотекою було використано mss. Її функція захоплення екрану виявилась кращою, оскільки можна було вказувати розмір захоплення екрану, а чим менший розмір – тим швидше виконується програма. Програма записувала екран зі швидкістю ~50 кадрів в секунду, що було непоганим результатом.

Але це був лише запис екрану, а скрипт має працювати паралельно із включеною грою та передавати зображення з екрану в нейромережу, обробляти результат та малювати їх в окремому вікні. Отож, при повному навантаженні mss показувала результат близько 20 FPS. Потрібно шукати заміну.

Було вирішено зупинитися на бібліотеці DxCam. Як описує її розробник: це високо-швидкісна бібліотека для скріншоту та захоплення екрану. Чудово підходить для запису ігор жанру FPS. Перевага бібліотеки у грамотній реалізацій функцій багатопотоковості. Саме через це функція запису екрану намагається записувати екран в початково заданій швидкості(60FPS) [8]. При великому розширенні та повному навантаженні, у вигляді працюючої нейромережі та відкритої гри, DxCam демонструє результат ~57 стабільних FPS.

Для перевірки продуктивності створено скрип для підрахування FPS(frame per second). Нище приведено декілька замірів ефективності:

* Pyautogui: середній час для 1-го скріншоту 0.05 секунд (~20 FPS) при будь-якому розширенні. Його проблема в тому, що вона робить скріншот всього екрану зразу.
* Mss: в розширенні (600x500), середній час для 1-го скріншоту: 0.02 секунд (50+-5 FPS)
* Mss, з використанням багатопотоковості не показав кращого результату
* DxCam: при будь-якому розширенні оптимізує використання потоків для отримання заданого користувачем результату. При повному навантаженні ~57FPS.

### Реалізація алгоритму WindMouse в коді

# Розділ 3. ДЕМОНСТРАЦІЯ результатів РОБОТИ МОДУЛЮ

## 3.1 Результати обробки зображень

На рисунку 3.1 наведений результат розставлення міток на зображеннях.



Рисунок 3.1 – Приклад міток на зображеннях

## 3.2 Процес навчання нейромережі

При навчанні нейромережі Yolov8, результат дивитись на рисунку 3.2, використовувались конфігураційні параметри, які були рекомендовані в різних керівництвах [5,11]. А саме:

* Epochs – 100. Кількість епох, значення вибране для запобіганню перенавчання.
* img\_size – 1024. Розмір зображення було взято більше ніж стандартний, це аргументовано потребою в кращій точності моделі.
* Batch\_size, розмір підвибірки – 16.
* Weights – це шлях до файлу з вагами. Було використано попередньо натреновані ваги, які залишив розробник на сайті GitHub [4]. Це дозволило значно зменшити час навчання.
* Data – шлях до даних для навчання.

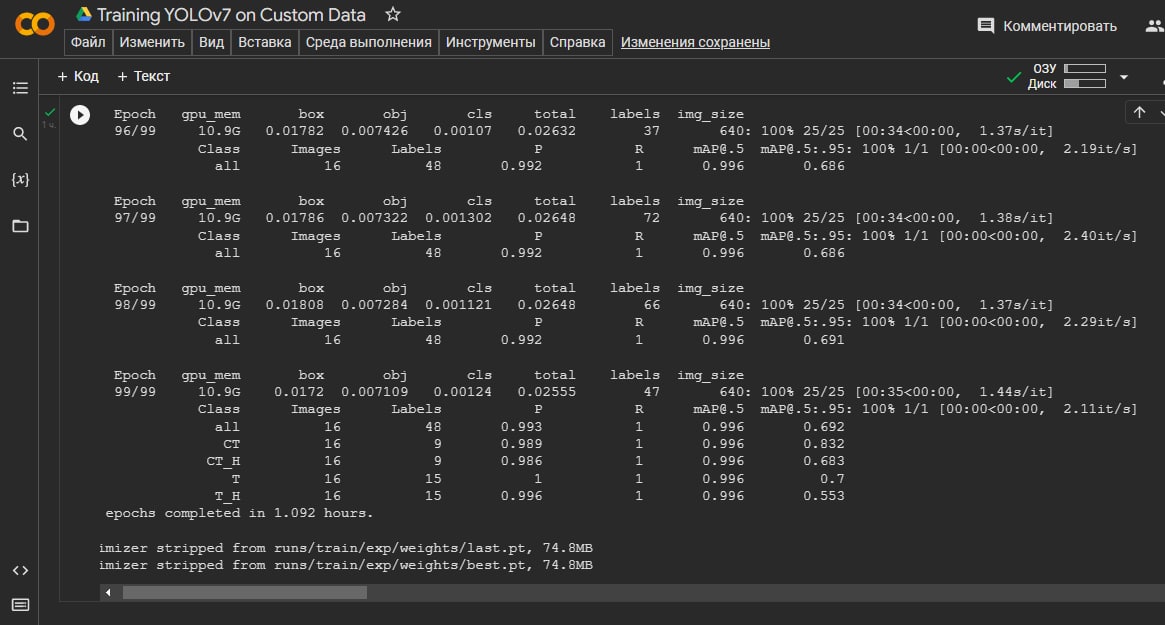


Рисунок 3.2 – Результат навчання нейромережі

При повному обсягу набору даних в 164 зображення, нейромережа навчалась лише на 80% всього обсягу тому, що інша частина була виділена під валідаційний(10% - 16 зображень) та тестовий (10% - 17 зображень) набір.

## 3.3 Оцінка якості навчання

Була проведена перевірка якості навчання на валідаційній та тестовій вибірці. На рисунках 3.3 та 3.4 висвітлено mAP для всіх класів – 95-96%.

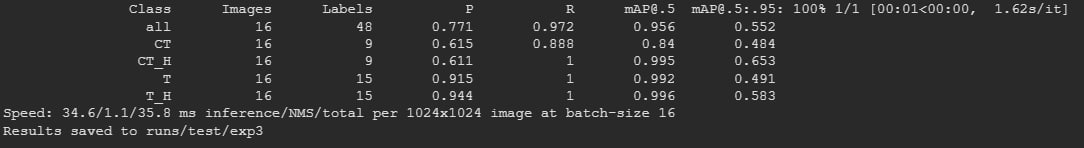


Рисунок 3.3 – Оцінка якості на валідаційній вибірці

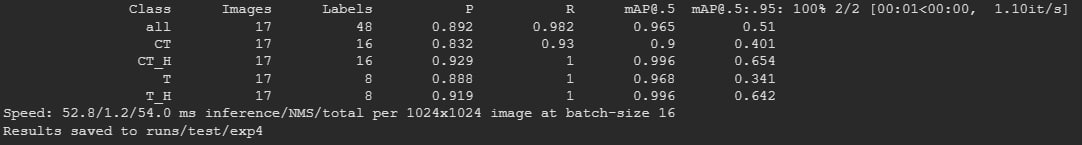


Рисунок 3.4 – Оцінка якості на тестовій вибірці

При оцінці якості навчання використовувались метрики, результати яких можна візуалізувати на графіках. Першою із них є Confusion Matrix, дивитись рисунок 3.5.

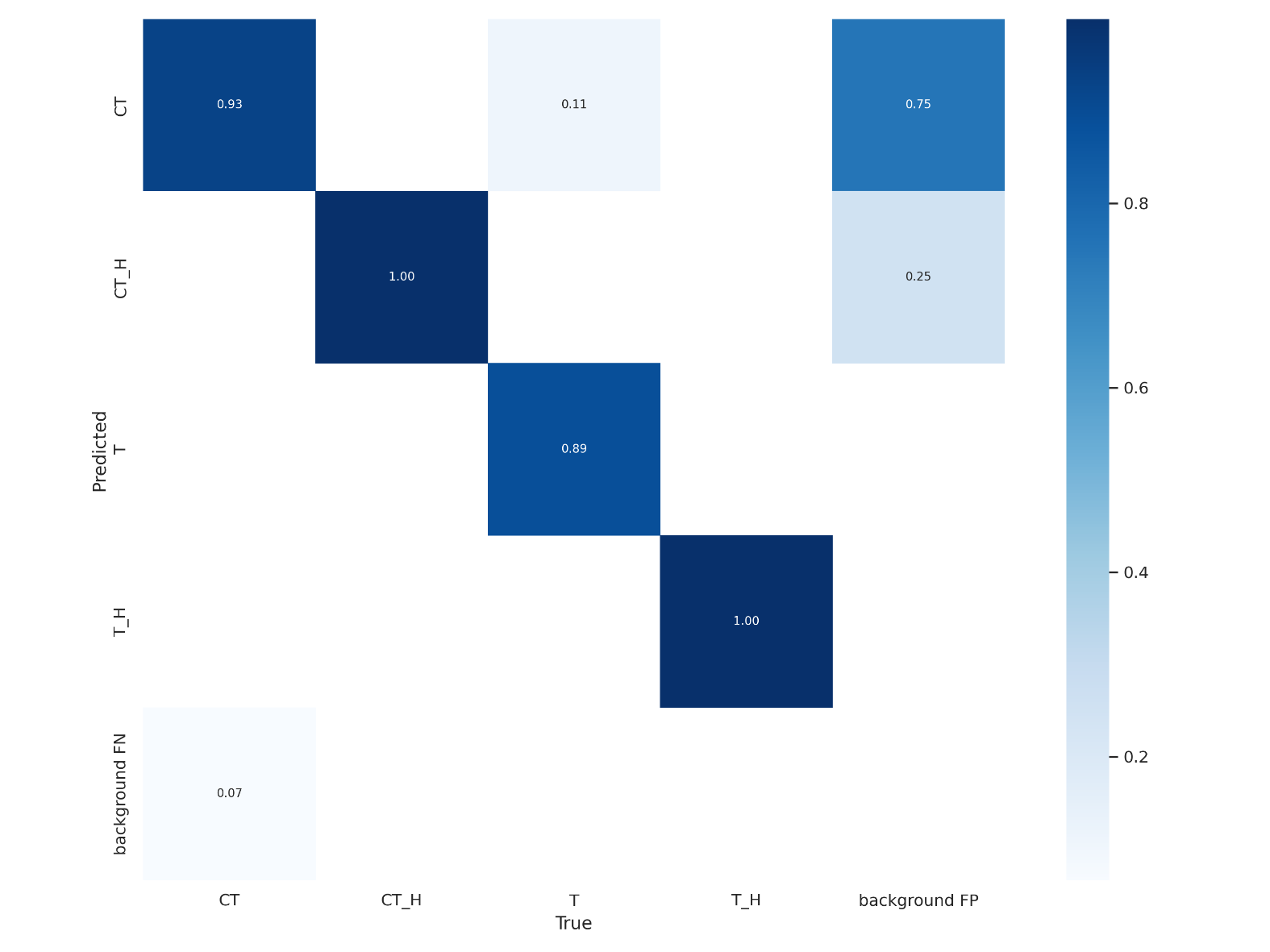


Рисунок 3.5 – Confusion Matrix

При аналізі спостерігається аномальне виявлення класу CT та CT\_H на задньому фоні.

Провівши дослідження та прочитавши статті, пов’язані з цією проблемою, дійшлося висновку, що в навчальний набір даних мережі можна і потрібно добавляти зображення фону, на якому немає міток класу. Це дозволить мережі якісніше навчитись та запобігти хибному спрацюванню класифікації.

Графік 3.6 - це крива відношення влучності до впевненості мережі. Доволі непоганий середній результат за виключенням класу CT. На графіку видно провал влучності в регіоні 70% впевненості. Бажано отримувати кращі результати.

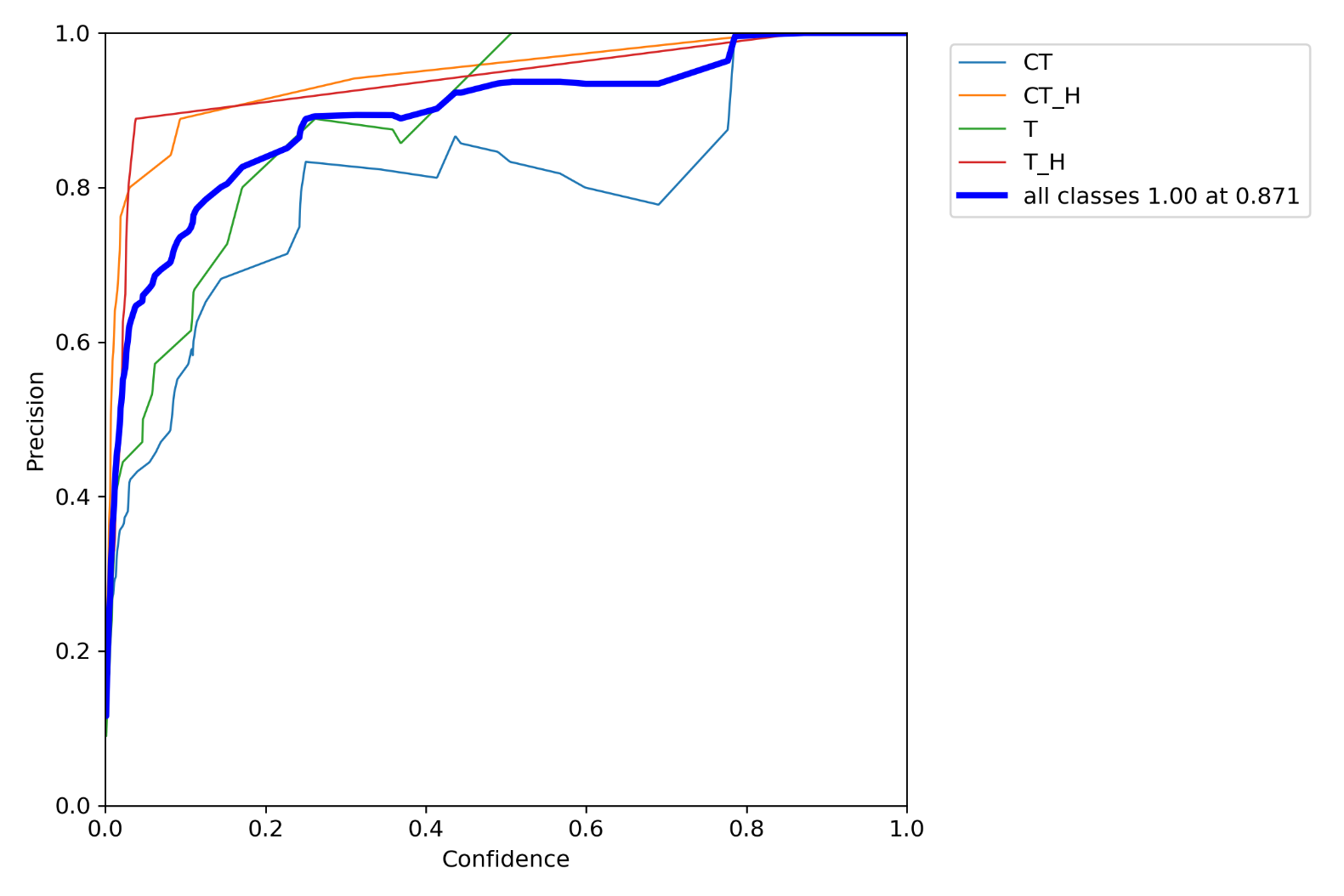


Рисунок 3.6 – Крива влучності

На рисунку 3.7 зображений графік показує нам криву повноти. Аналізуючи, можна сказати, що середній результат не поганий, але його псують класи CT та T.

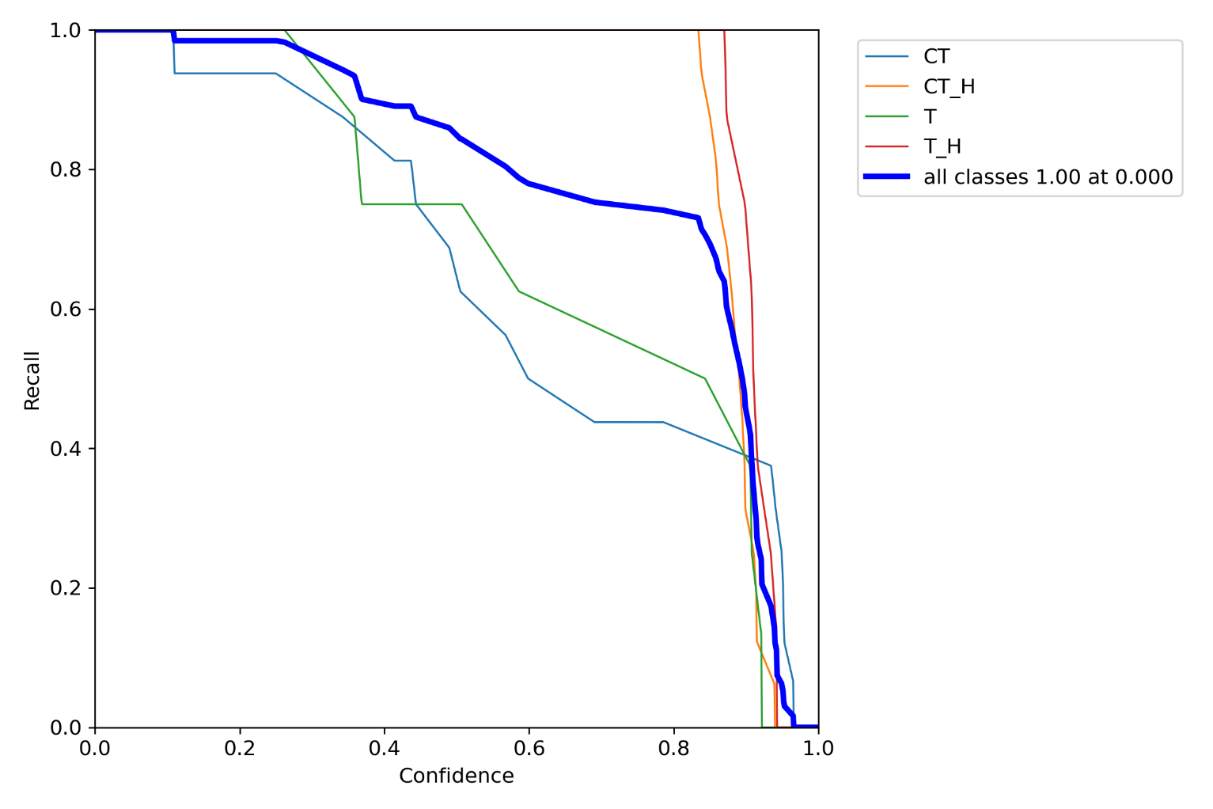


Рисунок 3.7 – Крива повноти

На рисунку 3.8 зображена середня точність (mAP) - це метрика, яка використовується для оцінки моделей виявлення об'єктів, таких як Fast R-CNN, YOLO, Mask R-CNN тощо. Середнє значення середньої точності (AP) розраховується за значеннями відгуків від 0 до 1.

Формула mAP базується на наступних підпоказниках: Матриця плутанини, Перетин над об'єднанням (IoU), Відкликання, Точність.

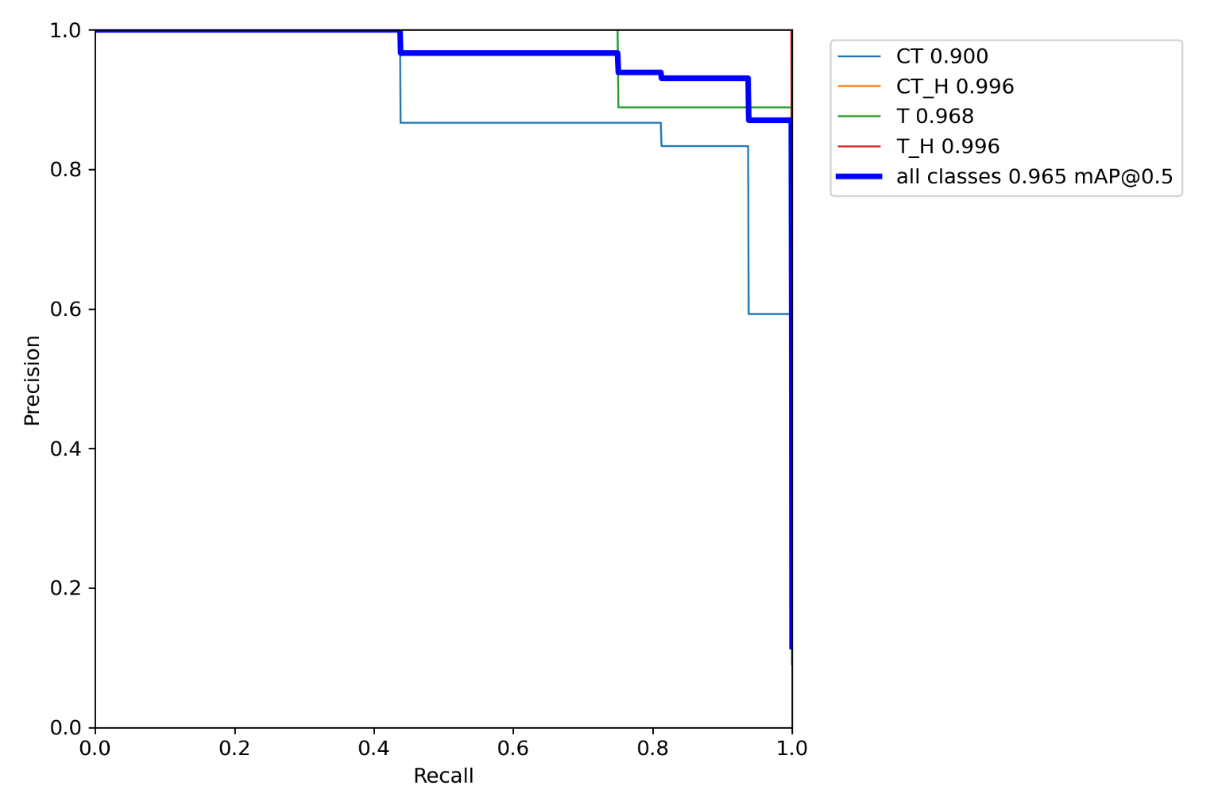


Рисунок 3.8 – mAP(mean Average Precision)

На рисунку 3.9 зображена F міра, яка поєднує влучність та повноту, є їх середнім гармонійним. При аналізі кривої було вибрано довірчу ймовірність: 85%, яка згодом була використана в розробці головного скрипта для виведення результатів із впевненістю вище заданого порогу.



Рисунок 3.9 – Крива F1

## 3.4 Приклад виявлення об’єктів

Нижче приведу кілька скріншотів які показують результат виводу програми.

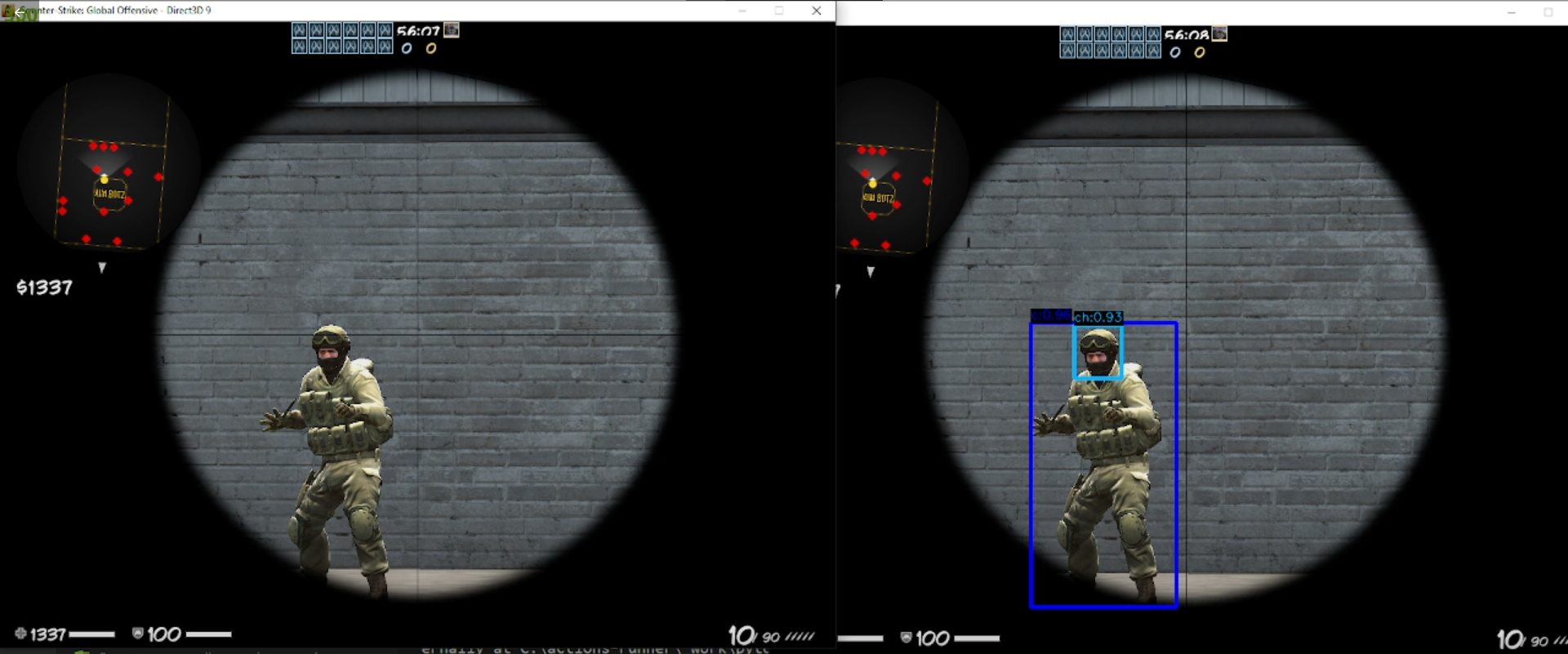


Рисунок 3.10 – приклад виявлення моделі спецназу

На рисунку 3.10 видно 2 вікна: ліве це вікно з грою а праве це результат оброблений нейромережею, який повертається із програми. В ньому можна побачити темно синю рамку навколо моделі гравця спецназу, та світло синю навколо його голови.

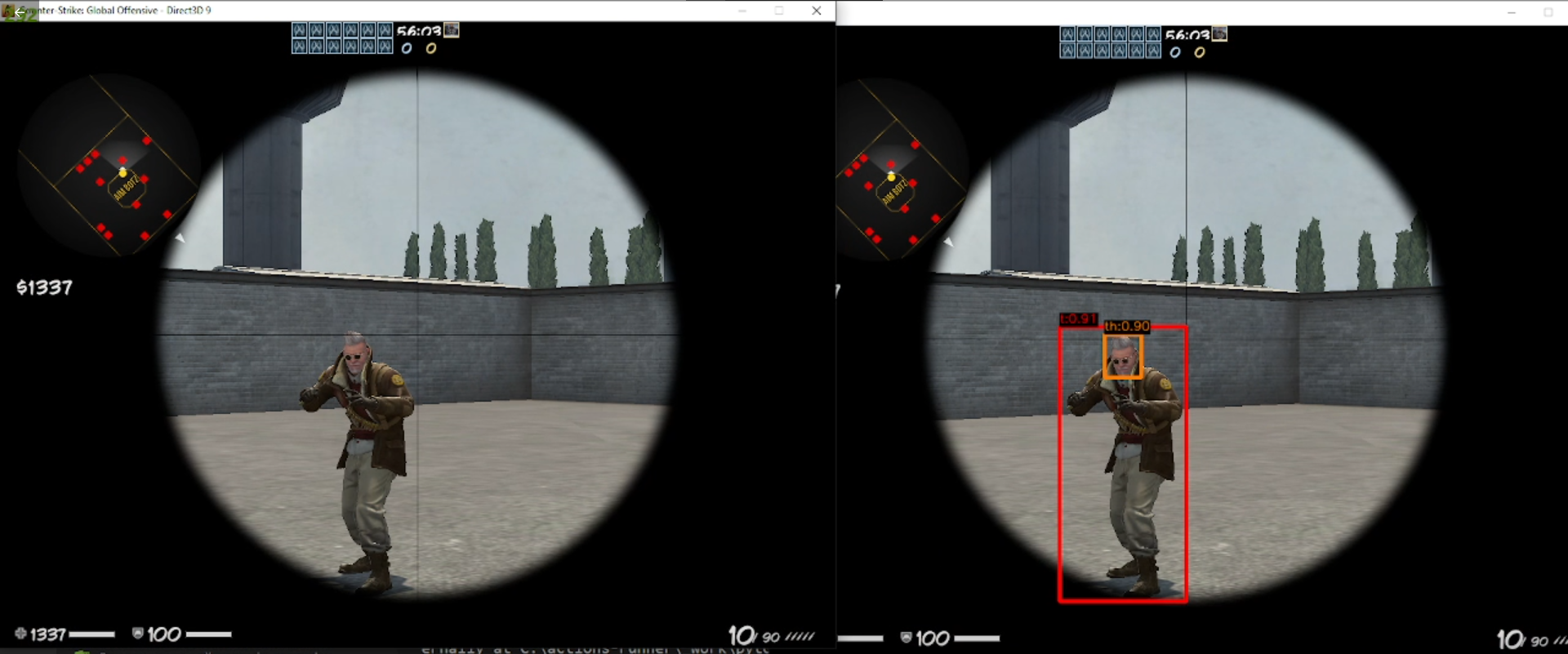


Рисунок 3.11 – приклад виявлення моделі терориста

На рисунку 3.11 аналогічний приклад, тільки вже з моделями терориста. Червоною рамкою обводиться його тіло, оранжевою – голова.

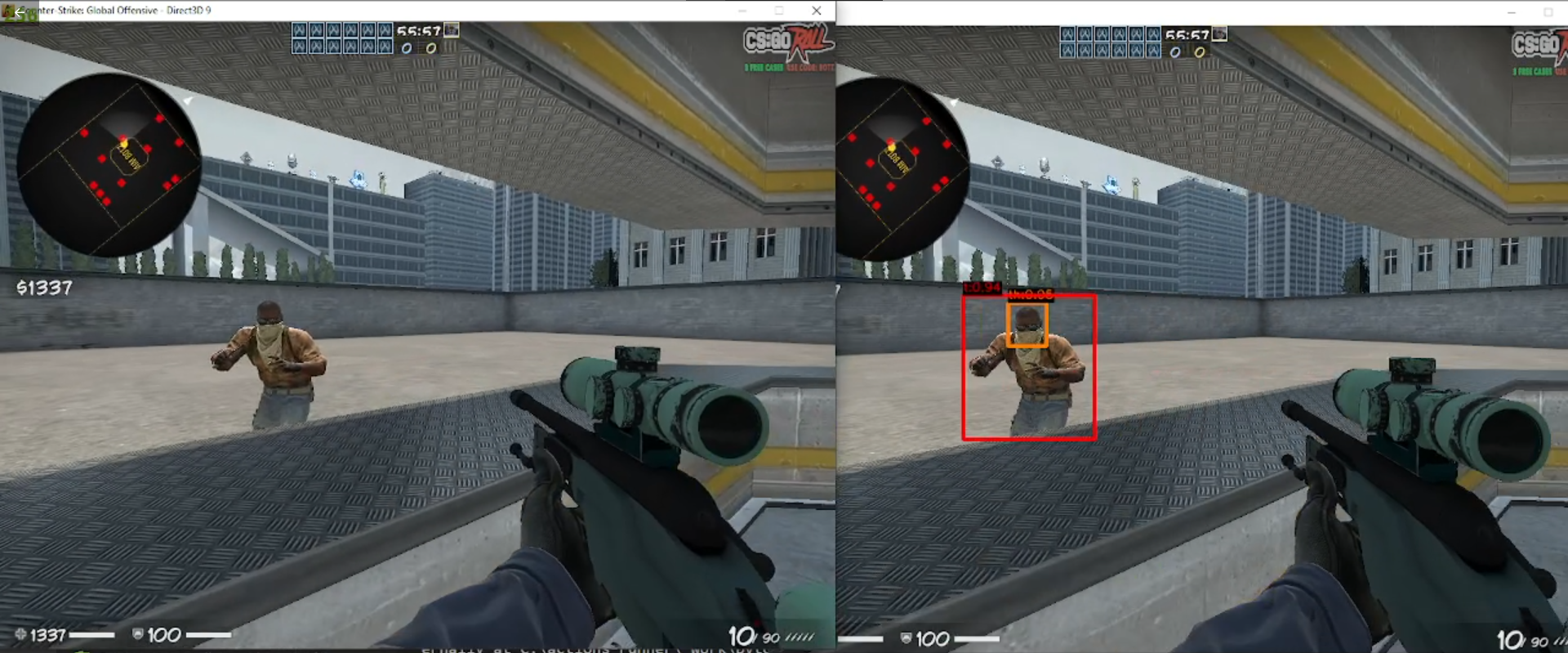


Рисунок 3.12 – приклад виявлення моделі терориста без прицілу

На рисунку 3.12 візуалізує можливість виявлення без використання прицілу.

## 3.5 Інструкція використання

Для навчання своєї нейромережі користувач може використати свій набір даних або загрузити той, який використовувався в цій курсовій роботі. Друге та третє посилання в додатку A.

Також можна скористатись готовим блокнотом, скопіювавши його з GoogleColab. Четверте посилання в додатку A.

Якщо є потреба скористатись вже натренованою моделлю, то можна скачати ваги за п’ятим посиланням в додатку А.

Коли вже є навчена модель(файл із розширенням .pt), можна запустити головний скрипт. Важливо запускати його в одній директорії з файлом розширення “.pt”. Лістинг скрипту знаходиться в додатку Б.

# Висновки

Працюючи над курсовою, було закріплено теоретичні знання з дисципліни «Інтелектуальний аналіз засобами Python/R». Сформовано та закріплено навички та уміння застосування інструментальних засобів, алгоритмів та методів Data Mining під час розв’язання задач аналітичного дослідження для виявлення прихованих закономірностей, необхідних для прийняття рішень у заданій предметній області.

Під час реалізації курсової роботи було виконано всі поставлені задачі. Виконано повний цикл обробки даних, такий як:

* створення набору даних, де було власноруч та з допомогою скрипту створено набір із 160 зображень
* пре\пост обробка набору даних, в результаті якої було нанесено мітки класів на кожне зображення та з допомогою алгоритмів потроєно кількість зображень
* навчання моделі, для якої використовувався попередньо підготовлений набір даних. Для навчання використовувались рекомендовані розробником та іншими спеціалістами параметри
* оцінка та візуалізація результатів на графіках за допомогою різних метрик, де добре висвітлюються сильні та слабкі сторони підготовленої моделі.

Обґрунтовано вибір алгоритму Data Mining для розв’язання поставленої задачі. Здійснено програмну реалізацію розв’язання задачі, а саме: створення скрипту для формування набору картинок, навчання моделі Yolov8 для класифікації та виявлення розташування об’єкта, а також скрипту комп’ютерного зору, який в режимі реального часу записував зображення на екрані, передавав його на обробку нейромережі, після чого проводилась обробка та вивід результатів в окреме вікно. Проведена робота над помилками.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття «Штучні нейронні мережі» [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Штучна_нейронна_мережа>
2. Стаття «Виявляння об`єктів» [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Виявляння_об%27єктів>
3. Стаття «Комп'ютерний зір» [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Комп%27ютерний_зір>
4. Документація розробника Yolov7 [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/WongKinYiu/yolov7>
5. Керівництво з використання Yolov7 [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.paperspace.com/yolov7/>
6. Стаття Yolo [Електронний ресурс] – Режим доступу до ресурсу: <https://arxiv.org/pdf/1506.02640.pdf?ref=blog.paperspace.com>
7. Керівництво з використання Yolov7 разом з бібліотекою PyTorch [Електронний ресурс] – Режим доступу до ресурсу: <https://inside-machinelearning.com/en/use-yolov7/>
8. Документація бібліотеки DxCam [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/ra1nty/DXcam>
9. Документація бібліотеки OpenCV [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.opencv.org/4.x/>
10. Керівництво з використання Yolov5 та OpenCV [Електронний ресурс] – Режим доступу до ресурсу: <https://learnopencv.com/object-detection-using-yolov5-and-opencv-dnn-in-c-and-python/>
11. Керівництво з використання Yolov7 та підготовки власного набору даних для нього [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.roboflow.com/yolov7-custom-dataset-training-tutorial/>

# Додатки

## Додаток А

**Інтернет посилання для завантаження даних**

1. Вся розробка велась із використанням GitHub, весь код можна знайти у вільному доступі: <https://github.com/Kolyanuss/Course_work_4_2/releases/tag/V1>
2. Набір даних із зображеннями можна загрузити за посиланням: <https://drive.google.com/drive/folders/1q4TnwMNvCqW-MHG8G4JbIOvUB7XSnM_D?usp=sharing>
3. Набір даних із зображеннями та мітками: <https://drive.google.com/file/d/1PQ1kVYaiXf0Xo7x67n8gXiWo2hkfFvB4/view?usp=sharing>
4. Блокнот GoogleColab для тренування нейромережі: <https://colab.research.google.com/drive/15FucEdt8nKVX4GZv8Mp0w6CehROpWNkX?usp=sharing>
5. Натреновані ваги: <https://drive.google.com/file/d/1yMl9jUhqS9xfyBcZhjIXB1MoRcZ1mhfe/view?usp=sharing>

## Додаток Б

**Лістинги власного коду**

**yolo\_torch.py**

import cv2

import numpy as np

import torch

import dxcam

INPUT\_WIDTH = 1024

INPUT\_HEIGHT = 1024

CONFIDENCE\_THRESHOLD = 0.85

# Text parameters.

FONT\_FACE = cv2.FONT\_HERSHEY\_SIMPLEX

FONT\_SCALE = 0.5

THICKNESS = 1

def draw\_label(im, label, x, y, color):

"""Draw text onto image at location."""

# Get text size.

text\_size = cv2.getTextSize(label, FONT\_FACE, FONT\_SCALE, THICKNESS)

dim, baseline = text\_size[0], text\_size[1]

# Use text size to create a BLACK rectangle.

cv2.rectangle(im, (x,y - dim[1] - baseline), (x + dim[0], y), (0,0,0), cv2.FILLED)

# Display text inside the rectangle.

cv2.putText(im, label, (x, y - baseline), FONT\_FACE, FONT\_SCALE, color, THICKNESS, cv2.LINE\_AA)

def draw\_wrap\_detection(input\_image, outputs\_2darr):

image\_width, image\_height, \_ = input\_image.shape

x\_factor = image\_width / INPUT\_WIDTH

y\_factor = image\_height / INPUT\_HEIGHT

for row in outputs\_2darr:

confidence = row[4]

if confidence >= CONFIDENCE\_THRESHOLD:

clas\_id = int(row[5])

x, y, w, h = row[0], row[1], row[2], row[3]

left = int(x \* x\_factor)

top = int(y \* y\_factor)

width = int(w \* x\_factor)

height = int(h \* y\_factor)

color = colors[int(clas\_id) % len(colors)]

# Draw bounding box.

**Продовження додатку**

cv2.rectangle(frame, (left, top), (width, height), color, THICKNESS\*3)

# Class label.

label = "{}:{:.2f}".format(class\_list[clas\_id], confidence)

# Draw label.

draw\_label(input\_image, label, left, top, color)

def format\_yolo(frame):

row, col, \_ = frame.shape

\_max = max(col, row)

result = np.zeros((\_max, \_max, 3), np.uint8)

result[0:row, 0:col] = frame[:, :, :3]

return result

# link to model: https://drive.google.com/file/d/1yMl9jUhqS9xfyBcZhjIXB1MoRcZ1mhfe/view?usp=sharing

model = torch.hub.load("WongKinYiu/yolov7", 'custom', 'config\_files/yolov7\_csgo\_v1.pt') #download custom model

class\_list = ["c","ch","t","th"]

colors = [(0, 0, 255), (0, 180, 255), (255, 0, 0), (255, 125, 0)]

monitor = (0, 26, 1024, 768+26)

camera = dxcam.create()

camera.start(region=monitor)

while True:

frame = camera.get\_latest\_frame()

frame = format\_yolo(frame)

results = model(frame) # includes NMS

# draw boxes

draw\_wrap\_detection(frame, results.xyxy[0].cpu().numpy())

# results.print() # info in console

cv2.imshow("OpenCV", cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

if cv2.waitKey(1) & 0xFF == ord("q"):

break

camera.stop()

cv2.destroyAllWindows()

**creater\_dataset.py**

import pyautogui

import cv2

import numpy as np

import time

from pynput import keyboard

path\_to\_save = "temp\_img"

def foo():

screen = pyautogui.screenshot()

screen\_arr = np.array(screen)

screen\_arr = screen\_arr[27:890, 768:, :]

screen\_arr = cv2.cvtColor(screen\_arr, cv2.COLOR\_RGB2BGR)

cv2.imwrite(path\_to\_save+"/screen\_" + str(time.time()) + ".png", screen\_arr)

def checker(key):

# print(f'{key} released')

if key == keyboard.Key.alt\_l:

foo()

if key == keyboard.Key.esc:

return False

with keyboard.Listener(on\_press=checker) as listener:

listener.join()

## Додаток В

**Модель мережі Yolov7**

# parameters

nc: 80 # number of classes

depth\_multiple: 1.0 # model depth multiple

width\_multiple: 1.0 # layer channel multiple

# anchors

anchors:

- [12,16, 19,36, 40,28] # P3/8

- [36,75, 76,55, 72,146] # P4/16

- [142,110, 192,243, 459,401] # P5/32

# yolov7 backbone

backbone:

# [from, number, module, args]

[[-1, 1, Conv, [32, 3, 1]], # 0

[-1, 1, Conv, [64, 3, 2]], # 1-P1/2

[-1, 1, Conv, [64, 3, 1]],

[-1, 1, Conv, [128, 3, 2]], # 3-P2/4

[-1, 1, Conv, [64, 1, 1]],

[-2, 1, Conv, [64, 1, 1]],

[-1, 1, Conv, [64, 3, 1]],

[-1, 1, Conv, [64, 3, 1]],

[-1, 1, Conv, [64, 3, 1]],

[-1, 1, Conv, [64, 3, 1]],

[[-1, -3, -5, -6], 1, Concat, [1]],

[-1, 1, Conv, [256, 1, 1]], # 11

[-1, 1, MP, []],

[-1, 1, Conv, [128, 1, 1]],

[-3, 1, Conv, [128, 1, 1]],

[-1, 1, Conv, [128, 3, 2]],

[[-1, -3], 1, Concat, [1]], # 16-P3/8

[-1, 1, Conv, [128, 1, 1]],

[-2, 1, Conv, [128, 1, 1]],

[-1, 1, Conv, [128, 3, 1]],

[-1, 1, Conv, [128, 3, 1]],

[-1, 1, Conv, [128, 3, 1]],

[-1, 1, Conv, [128, 3, 1]],

[[-1, -3, -5, -6], 1, Concat, [1]],

[-1, 1, Conv, [512, 1, 1]], # 24

[-1, 1, MP, []],

[-1, 1, Conv, [256, 1, 1]],

[-3, 1, Conv, [256, 1, 1]],

[-1, 1, Conv, [256, 3, 2]],

[[-1, -3], 1, Concat, [1]], # 29-P4/16

[-1, 1, Conv, [256, 1, 1]],

[-2, 1, Conv, [256, 1, 1]],

[-1, 1, Conv, [256, 3, 1]],

[-1, 1, Conv, [256, 3, 1]],

[-1, 1, Conv, [256, 3, 1]],

[-1, 1, Conv, [256, 3, 1]],

...