МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОЛОСОВА КАТЕРИНА КОСТЯНТИНІВНА

Допускається до захисту:
в. о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____О. В. Зелінська
«____»_____ 2024 р.

ВИЯВЛЕННЯ ТА ВІДСТЕЖЕННЯ ОБ'ЄКТІВ БЕЗПІЛОТНИМИ ЛІТАЛЬНИМИ АПАРАТАМИ У РЕЖИМІ РЕАЛЬНОГО ЧАСУ НА ОСНОВІ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Т. В. Січко, доцент кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____
(підпис)

Оцінка: ___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Колосова К. К. Виявлення та відстеження об'єктів безпілотними літальними апаратами у режимі реального часу на основі згорткових нейронних мереж. Спеціальність 122 «Комп’ютерні науки», Освітня програма «Комп’ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (магістерській) роботі досліджено процеси виявлення та відстеження об'єктів у режимі реального часу за допомогою згорткових нейронних мереж. У ході виконання кваліфікаційної (магістерської) роботи було створено ефективну систему виявлення та відстеження об'єктів у реальному часі для використання на безпілотних літальних апаратах.

Ключові слова: БПЛА, згорткова нейронна мережа, виявлення та відстеження об’єктів, машинний зір, реальний час, обробка зображень.

93 сторінки, 28 рисунків, 1 таблиця, 35 джерел.

ABSTRACT

Kolosova K. K. Real-Time Object Detection and Tracking by Unmanned Aerial Vehicles Based on Convolutional Neural Network. Specialty 122 "Computer Science", educational program " Computer technologies of data processing (Data Science)". Vasyl` Stus Donetsk National University, Vinnytsia 2024.

In the qualification (master's) thesis, the processes of real-time object detection and tracking using convolutional neural networks were studied. During the performance of the qualification (master's) work, an efficient real-time object detection and tracking system was developed for use on unmanned aerial vehicles.

Keywords: Unmanned Aerial Vehicles (UAVs), Convolutional Neural Network, Object Detection, Object Tracking, Machine Vision, Real-Time, Image Processing.

93 pages, 28 figures, 1 table, 35 sources.

ЗМІСТ

ВСТУП

В останні десятиліття технології безпілотних літальних апаратів (БПЛА) набули значного поширення та розвитку, що відкрило нові можливості для їх застосування у різних сферах, включаючи військову справу, сільське господарство, моніторинг навколишнього середовища та рятувальні операції. Однією з ключових задач, що стоять перед БПЛА, є виявлення та відстеження об'єктів у реальному часі. Це завдання є надзвичайно складним, оскільки вимагає високої точності, швидкодії та здатності до адаптації в умовах мінливого освітлення, погодних умов та інших факторів зовнішнього середовища.

З розвитком комп'ютерного зору та машинного навчання з'явилися нові інструменти для розв'язання цієї проблеми. Зокрема, згорткові нейронні мережі (Convolutional Neural Networks, CNN) показали високу ефективність у задачах класифікації зображень, розпізнавання об'єктів та сегментації. Використання CNN у поєднанні з сучасними апаратними засобами дозволяє створювати системи, що здатні виконувати складні обчислення в реальному часі, що є критично важливим для роботи БПЛА.

Мета дослідження. Створення програмного модуля виявлення та відстеження об'єктів у реальному часі для використання на безпілотних літальних апаратах.

Об’єкт дослідження. Процеси виявлення та відстеження об'єктів у режимі реального часу за допомогою згорткових нейронних мереж.

Предмет дослідження. Методи та алгоритми виявлення та відстеження об'єктів у режимі реального часу, що реалізовані на основі згорткових нейронних мереж, та їх застосування у безпілотних літальних апаратах.

Завдання:

- дослідити сучасні методи виявлення та відстеження об’єктів із використанням згорткових нейронних мереж(CNN);

- обрати оптимальні алгоритми згорткових нейронних мереж для реалізації в реальному часі;
- створити набір даних для навчання та тестування згорткової нейронної мережі;
- виконати навчання моделі на основі зібраного набору даних;
- інтегрувати модель виявлення та відстеження об’єктів із системою керування БПЛА.

Актуальність:

Безпілотні літальні апарати (БПЛА) є ключовим інструментом сучасних технологій для моніторингу, аналізу та безпеки. Розробка програмного модуля для виявлення і відстеження об’єктів у реальному часі з використанням згорткових нейронних мереж є актуальною через потребу у підвищенні точності, швидкодії та адаптивності систем.

На відміну від традиційних методів, нейронні мережі забезпечують ефективність і гнучкість роботи в складних умовах, роблячи їх незамінними для виконання критичних завдань, таких як моніторинг, рятувальні операції та забезпечення безпеки.

Апробація результатів дослідження:

Результати кваліфікаційної (магістерської) роботи апробовано на:

1. Міжнародній науково практичній конференції «Відновлення України у повоєнні часи: виклики, стратегічні пріоритети, ресурсне забезпечення, потенціал майбутнього розвитку» , м. Вінниця, 10-11 жовтня 2024 р. Вінниця: ДонНУ імені Василя Стуса, 2024. Тези на тему «Виявлення та відстеження об’єктів БПЛА у реальному часі на основі CNN» (Прийнято до друку).

2. Вісник Хмельницького національного університету. Серія: Технічні науки, м. Хмельницький. Стаття на тему «Згорткові нейронні мережі у програмуванні безпілотних літальних апаратів» (Прийнято до друку).

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ І ПІДХОДІВ РОЗПІЗНАВАННЯ ОБРАЗІВ

1.1 Аналіз проблематики області дослідження

Виявлення та відстеження об'єктів у реальному часі є однією з ключових задач в області комп'ютерного зору та машинного навчання. Особливо важливим це завдання є для безпілотних літальних апаратів (БПЛА), які повинні виконувати свої функції у динамічному та непередбачуваному середовищі. Успішне розв'язання цієї проблеми має широкий спектр застосувань, включаючи військові операції, цивільну авіацію, моніторинг довкілля, сільське господарство та рятувальні операції.

Основні виклики та проблеми:

1. Реалізація в реальному часі:

Обробка зображень та відео в режимі реального часу потребує високої обчислювальної потужності. Більшість БПЛА мають обмежені ресурси, що ускладнює використання складних алгоритмів машинного навчання.

Затримка обробки може призвести до критичних помилок у роботі системи, що особливо небезпечно у ситуаціях, де потрібна миттєва реакція, наприклад, при уникненні перешкод або виявленні рухомих цілей.

2. Точність виявлення та відстеження:

Висока точність необхідна для коректного ідентифікування об'єктів у різних умовах освітлення, відстані та ракурсу. Низька точність може призвести до помилкових виявлень або пропусків важливих об'єктів.

Невизначеність зовнішнього середовища (зміни погоди, рухомі об'єкти, перешкоди) додає додаткової складності у забезпеченні стабільної роботи алгоритмів.

3. Робота в умовах обмежених ресурсів:

Більшість сучасних моделей згорткових нейронних мереж (CNN) вимагають значних обчислювальних ресурсів для навчання та виконання.

Оптимізація моделей для використання на пристроях з обмеженими ресурсами є критично важливою.

Врахування енергоспоживання є також важливим фактором, оскільки БПЛА зазвичай мають обмежену ємність батарей [1].

4. Різноманітність об'єктів та умов:

Система повинна бути здатна розпізнавати широкий спектр об'єктів, незалежно від їх форми, кольору, текстури та руху.

Зміни в умовах зовнішнього середовища, такі як різні рівні освітлення, тіні, атмосферні явища, можуть значно впливати на якість виявлення та відстеження.

Поточні підходи та технології:

1. Класичні методи комп'ютерного зору:

Використання методів сегментації, фільтрів, детекторів країв та контурів. Ці методи зазвичай менш вимогливі до ресурсів, але мають обмежену точність та надійність в умовах реального світу.

2. Глибоке навчання та згорткові нейронні мережі (CNN):

Застосування CNN для виявлення та класифікації об'єктів значно покращило точність та надійність систем виявлення та відстеження.

Архітектури, такі як YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), Faster R-CNN, є популярними завдяки їх здатності виконувати виявлення об'єктів у реальному часі з високою точністю.

3. Оптимізація та компресія моделей:

Методи оптимізації, такі як прунинг (pruning), квантування (quantization), використання полегшених архітектур (наприклад, MobileNet), дозволяють зменшити обчислювальні вимоги та покращити швидкодію без значних втрат у точності.

Використання апаратних прискорювачів, таких як графічні процесори (GPU) та спеціалізовані процесори для нейронних мереж (TPU), також сприяє покращенню продуктивності [2].

1.2 Застосування нейронних мереж у розпізнаванні образів

Нейронна мережа — це модель машинного навчання або програма, яка приймає рішення за принципом, схожим на роботу людського мозку. Вона імітує взаємодію біологічних нейронів, що спільно аналізують явища, оцінюють варіанти та роблять висновки.

Структура нейронної мережі складається з шарів вузлів, які також називають штучними нейронами. Вони поділяються на вхідний шар, один або кілька прихованих шарів і вихідний шар. Кожен вузол має зв’язки з іншими, а також власну вагу та поріг активації. Якщо значення виходу вузла перевищує цей поріг, вузол активується та передає дані на наступний шар. Якщо ж поріг не досягнуто, дані далі не передаються.

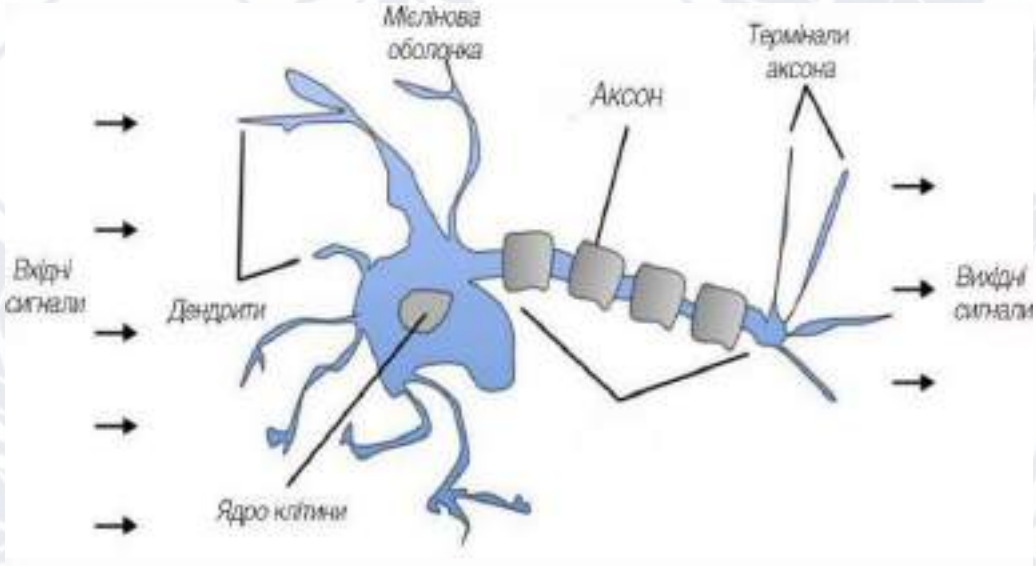

Рис. 1.1 – Зображення нейрона

Нейронні мережі покладаються на навчальні дані, щоб навчатися та підвищувати свою точність з часом. Після їх точного налаштування на точність вони стають потужними інструментами в інформатиці та штучному інтелекті , що дозволяє класифікувати та кластеризувати дані з високою швидкістю. Завдання з розпізнавання мовлення або розпізнавання зображень можуть тривати хвилини чи години порівняно з ручною ідентифікацією

експертів-людей. Одним із найвідоміших прикладів нейронної мережі є пошуковий алгоритм Google [3].

Нейронна мережа є фундаментальним компонентом глибокого навчання, підгалузі штучного інтелекту. Це обчислювальна модель, натхненна структурою та функціонуванням людського мозку. Нейронні мережі складаються з кількох ключових компонентів, кожен із яких відіграє свою особливу роль у процесі навчання [4].

Штучна нейронна мережа складається з кількох блоків обробки, які називаються вузлами або нейронами, які організовані в шари. Ці шари з'єднані один з одним за допомогою вантажів.

Кількість вузлів у кожному шарі частково залежить від місця розташування рівня в мережі, частково від даних, які зрештою будуть передані на рівень, і частково від вибору проекту архітектора мережі для рівня.

Кожен вузол має вихідне значення, яке є результатом певної операції, що відбувається між даними, які передаються йому як вхідні дані, разом із вагами, які з’єднують вхідні дані з вузлом. Операція залежить від типу шару, частиною якого є вузол.

Вихідні дані з вузлів одного шару передаються як вхідні дані вузлам наступного рівня [5].

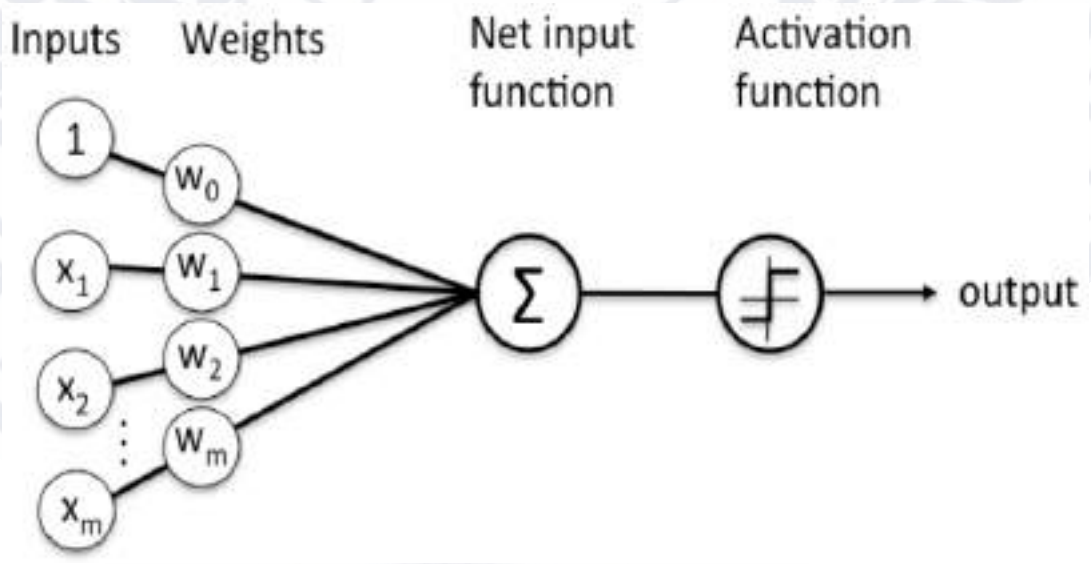

Рис. 1.2 – Схема зображення одного вузла

Шар вузла складається з набору нейронних комутаторів, які активуються або деактивуються при подачі вхідних даних через мережу. Результат роботи кожного шару стає вхідними даними для наступного, починаючи з першого шару, який отримує початкову інформацію.

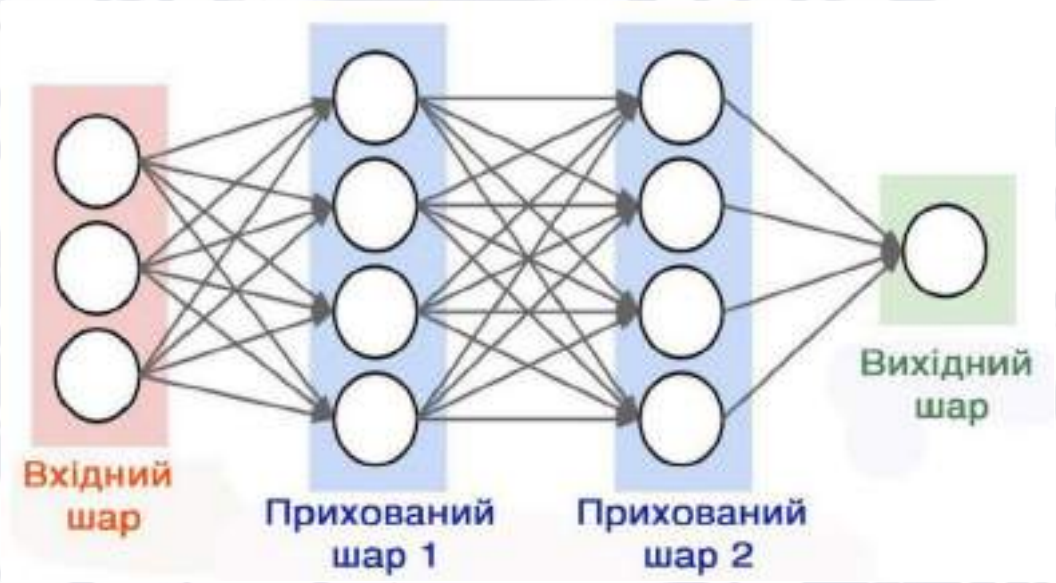

Рис. 1.3 – Приклад простої нейромережі

Сполучення регульованих ваг моделі з функціями введення полягає в тому, як ми надаємо значення цим характеристикам щодо того, як класифікує та кластеризує нейромережа.

Згорткові нейронні мережі (CNN) — це спеціальний тип нейронних мереж, розроблений для ефективного оброблення даних із просторовою структурою, таких як зображення. CNN складаються з кількох шарів, кожен з яких виконує специфічні обчислення, спрямовані на виявлення та виділення певних особливостей у вхідних даних. Основна операція в таких мережах — згортка, яка застосовується для виділення низькорівневих особливостей, таких як краї або текстури, з подальшим зростанням рівня абстракції на наступних шарах. Цей процес відомий як пряме поширення, де інформація передається через мережу від вхідного шару до вихідного для класифікації або прогнозування.

Вхідний і вихідний шари в CNN, як і в інших нейронних мережах, називаються видимими шарами. Вхідний шар — це місце, де модель отримує необроблені дані (наприклад, пікселі зображення), а вихідний шар забезпечує остаточний результат у вигляді передбачення або розпізнаної категорії. Під час проходження через шари CNN кожен наступний шар вивчає більш абстрактні особливості, агрегуючи інформацію з попередніх шарів. Таким чином, CNN здатна розпізнавати складні патерни, що дозволяє їй виділяти структури об’єктів у зображеннях.

CNN зазвичай складається з декількох типів шарів, включаючи згорткові, шари підвибірки (пулінгу) та шари повного з'єднання. Кожен з них виконує свою унікальну функцію: згорткові шари виділяють специфічні ознаки, шари підвибірки зменшують розмір даних, зберігаючи ключову інформацію, а шари повного з'єднання відповідають за остаточне передбачення. На рис. 1.4 показано структуру згорткової нейронної мережі [6].

Завдяки своїй архітектурі CNN особливо підходять для роботи з неструктурованими даними, такими як зображення, відео та інші мультимедійні дані. Це дозволяє моделі виділяти приховану структуру в неструктурованих даних, автоматично знаходячи шаблони та аномалії.

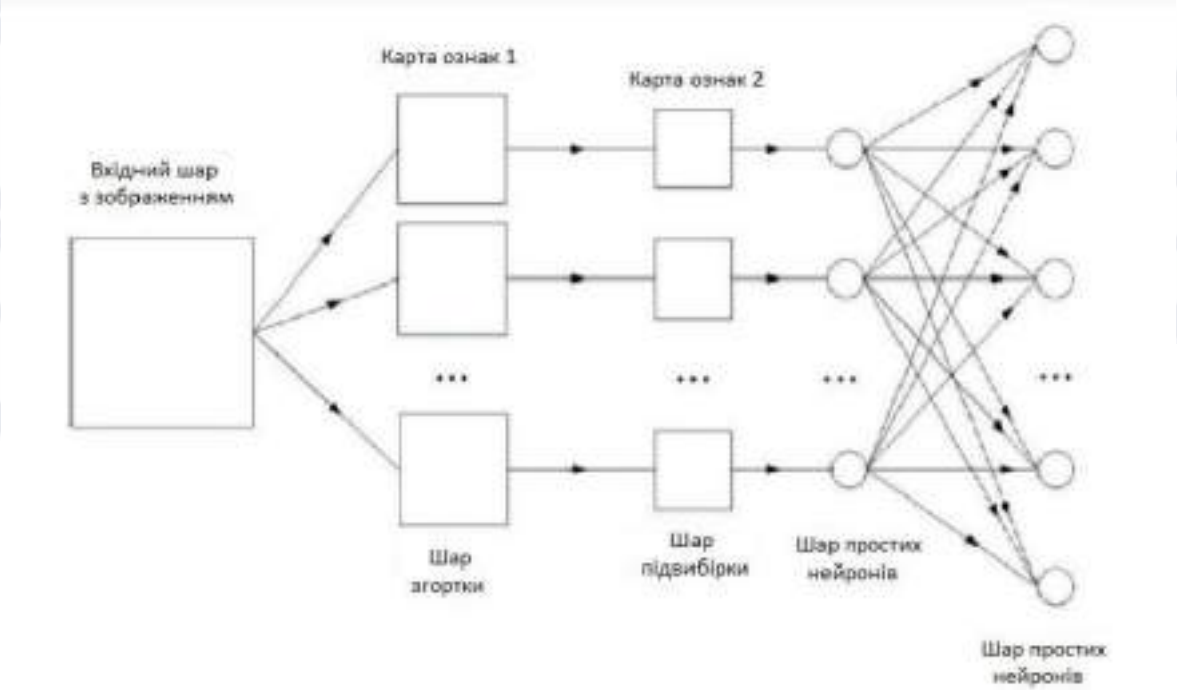

Рис. 1.4 – Структура згорткової нейронної мережі

1.3 Методи розпізнавання образів

Для сприйняття образу необхідно виділити набір релевантних інформативних ознак і сформувати вектор-реалізацію образу, вимірявши їх значення. У практиці зазвичай виникає потреба в додаткових процедурах попередньої обробки, фільтрації та відновлення образів.

Важливу роль у розв'язанні задачі відіграє апріорна інформація про об'єкти розпізнавання, яка потрібна для формування алфавіту класів розпізнавання, словника ознак і навчальної матриці. Крім того, для побудови високодостовірних вирішальних правил на етапі формування вхідного математичного опису системи розпізнавання необхідно проводити так званий розвідувальний аналіз для забезпечення статистичної стійкості та однорідності реалізацій образу [7].

Методи розпізнавання поділяють на дві основні групи:

- методи, які для формування вирішальних правил використовують класифіковану навчальну матрицю (методи навчання з учителем);
- методи автоматичної класифікації, що здатні обробляти некласифіковані дані (методи навчання без учителя).

Кожна з цих основних груп ділиться на підгрупи залежно від апріорної інформації, необхідної для їх застосування [8].

Розглянемо в рамках геометричного підходу узагальнену постановку задачі інформаційного синтезу системи розпізнавання образів, здатної до навчання.

Нехай дано алфавіт класів розпізнавання $\{Xmo|m = 1. . M\}$, де M – потужність алфавіту.

На етапі навчання необхідно:

1. Побудувати деяким оптимальним (в інформаційному розумінні) способом розбиття R простору ознак Ω|N| на класи розпізнавання;

2. На основі геометричних параметрів оптимального розбиття R* простору ознак на класи розпізнавання необхідно створити вирішальні правила, що гарантують безпомилковість за навчальною матрицею.

На етапі розпізнавання (екзамену) слід приймати максимально достовірне рішення щодо належності образу до одного з класів заданого алфавіту. У топологічному сенсі клас розпізнавання — це область у просторі ознак, елементи якої еквівалентні в контексті задачі.

Вирішальні правила призначені для визначення приналежності образу до конкретного класу. Вони можуть базуватися на математичних формулах, геометричних моделях, лінгвістичних підходах тощо.

Розглянемо побудову вирішальних правил у межах геометричного підходу, який відрізняється універсальністю та наочністю:

1. Вимірювання ознак для розпізнавання.
2. Нормалізація значень ознак, яка передбачає приведення їх до вигляду, зручного для обчислення, і формування векторів-реалізацій образів для кожного спостереження.
3. Створення навчальної матриці.
4. Допустимі перетворення навчальної матриці в субпросторі ознак.
5. Оптимальне розбиття простору ознак на класи розпізнавання.
6. Побудова вирішальних правил, заснованих на геометричних параметрах цього розбиття.

Таким чином, для створення вирішальних правил (або класифікатора) необхідно оптимізувати параметри функціонування системи розпізнавання. Параметри функціонування – це характеристики системи розпізнавання, які впливають на її функціональну ефективність. У системах розпізнавання, що навчаються, ці параметри часто називаються параметрами навчання.

Розглянемо кілька прикладів формування вирішальних правил. Нехай $X1$ і $X2$ – два класи розпізнавання, розподіл реалізацій яких показано на рис. 1.5.

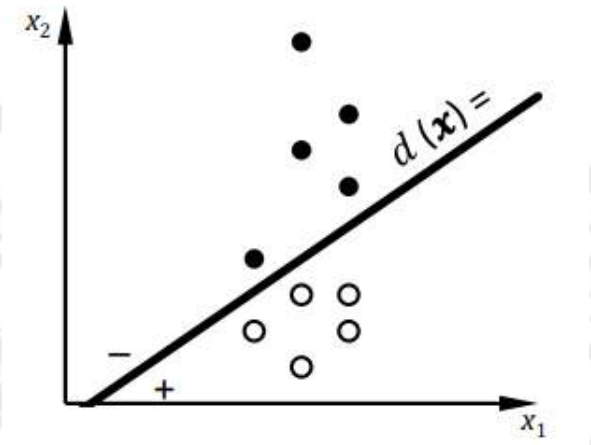

Рис. 1.5 – Розподіл реалізацій двох класів розпізнавання

Кожна реалізація є вектором ознак розпізнавання $x = (x_1, x_2)^T$, заданим у просторі ознак $x_1 - x_2$. На рис. 1.5 реалізації, що належать до класу X_1, позначено світлими кружечками, а які належать до класу X_2 – темними.

Реалізації показаних на рис. 1.5 класів розпізнавання можна розділити прямою: $d(x) = w_0 + w_1x_1 + w_2x_2 = 0$.

При цьому значення коефіцієнтів w_0, w_1 та w_2 визначено так, що для всіх реалізацій класу X_1 виконується нерівність $d(x) > 0$, а для всіх реалізацій класу X_2 – $d(x) < 0$. Крім того, для будь-якої реалізації x, для якої апріорно відомо, що вона належить одному з класів X_1 або X_2, можна обчислити значення $d(x)$ і прийняти рішення $x \in X_1$, якщо $d(x) > 0$, або $x \in X_2$, якщо $d(x) < 0$. Сформовану в такий спосіб роздільну функцію $d(x)$ називають лінійним вирішальним правилом класу X_1.

Для N-вимірного випадку $x = (x_1, x_2, \dots, x_N)^T$ пряма перетворюється на гіперплощину:

$$d(x) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Nx_N = w_0 + w\,T_x = 0,$$

де $w = (w_1, w_2, \dots, w_N)$ – вектор коефіцієнтів, до значень яких ставляться вимоги.

Зазвичай вектор-реалізація x розширюється до $x = (1, x_1, x_2, \dots, x_N)^T$, а вектор коефіцієнтів – до $w = (w_0, w_1, w_2, \dots, w_N)$.

Тоді лінійні вирішальні правила набирають вигляду:

$d(\text{x}) = w\ \text{T}_\text{x} = 0.$

1.4 Критерії якості розпізнавання образів

Якість класифікації об’єктів істотно залежить від вибору міри їх близькості (подібності).

Мірою близькості називається величина, що має граничне значення і здатна збільшуватися із зменшенням відстані між реалізаціями класів розпізнавання. Розглянемо основні способи визначення близькості між об’єктами для кількісних шкал вимірювань [9].

Найбільш поширеною мірою близькості в теорії розпізнавання образів є евклідова відстань:

$$d_E\ (x_m, x_i) = \sqrt{\sum_{i=1}^{N}(x_m^i - \ x_i^i)^2} \qquad (1.1)$$

де x_m – реалізація класу x_m; x_l – реалізація класу x_l; $x_m{}^i$ – і-та ознака розпізнавання класу $X_m o$; x_{li} – і-та ознака розпізнавання класу X_{li}; N – кількість ознак розпізнавання; M – кількість класів розпізнавання.

Відстань відповідає інтуїтивному уявленню про геометричну близькість двох точок у багатовимірному просторі ознак і особливо ефективно розділяє вектори реалізації класів розпізнавання при обґрунтуванні гіпотези компактності їх розподілу. Ця міра широко використовується в методах кластер-аналізу, які ґрунтуються на детермінованих дистанційних критеріях близькості.

Для надання більшої ваги значно віддаленим одна від одної реалізаціям класів розпізнавання застосовують квадрат евклідової відстані

$$d_E^2\ (x_m, x_i) = \sum_{i=1}^{N}(x_m^i - \ x_i^i)^2 \qquad (1.2)$$

Міра дозволяє збільшувати внесок віддалених реалізацій, оскільки вони підносяться до квадрата. Для зменшення впливу великих різниць застосовують лінійну (манхеттенську) міру близькості:

$$d(x_m, x_i) = \sum_{i=1}^{N} \left| x_m^i - x_i^i \right|, m, l = \overline{1, M} \quad (1.3)$$

Застосування міри виправдано в ситуаціях, пов'язаних із багатокритеріальним оцінюванням (наприклад, якості продукції), де для порівняння об'єктів використовують модулі відхилень.

У разі, коли необхідно зменшити або збільшити вагу реалізацій образів, що сильно відрізняються один від одного, застосовують узагальнену степеневу відстань:

$$d\,(x_m, x_i) = \sqrt[r]{\sum_{i=1}^{N} (x_m^i - x_i^i)^p} \quad (1.4)$$

Де r – параметр, який відповідає за прогресивне зважження великих відстаней між об'єктами; p – параметр, що відповідає за поступове зважження різниць за окремими координатами.

Якщо у виразі обидва параметри дорівнюють двом, то ця відстань збігається з відстанню Евкліда.

На практиці має місце гіпотеза нечіткої компактності реалізацій образів, оскільки апріорно класи перетинаються і мають нечіткі (розмиті) межі. Тому застосування в задачах класифікації вищенаведених детермінованих дистанційних критеріїв близькості не дозволяє в загальному випадку побудувати чітке розбиття простору ознак на класи розпізнавання [10].

Висновки до розділу 1

Розділ 1 надає комплексний огляд ключових аспектів дослідження розпізнавання образів, включаючи аналіз проблематики, застосування нейромереж, методів розпізнавання та критеріїв якості.

Проблематика виявлення та відстеження об'єктів у реальному часі зосереджена на необхідності високої точності, швидкості обробки, стійкості до шумів та адаптивності системи до різних умов. Застосування згорткових нейронних мереж (CNN) значно покращує ефективність розпізнавання завдяки їх здатності автоматично витягувати інформативні ознаки з даних, що

робить їх ефективними у різних додатках, включаючи медичну діагностику та автоматичне керування транспортом.

Методи розпізнавання образів поділяються на ті, що використовують навчання з учителем, і ті, що працюють без нагляду. Вибір методу залежить від типу даних та вимог до системи. Якість розпізнавання оцінюється за різними критеріями, такими як точність, чутливість, специфічність, точність передбачення, F-міра, AUC, час обробки, стійкість до шумів та узагальнююча здатність. Ці критерії дозволяють об'єктивно оцінити ефективність та надійність системи.

Узагальнюючи, успішне розпізнавання образів вимагає комплексного підходу, що включає використання передових методів та нейромереж, а також всебічну оцінку якості системи за визначеними критеріями для її оптимізації та подальшого практичного застосування.

РОЗДІЛ 2
ОГЛЯД НЕЙРОННОЇ МЕРЕЖІ ТА СПОСІБ ЇЇ НАВЧАННЯ

2.1 Вибір моделі нейронної мережі для вирішення задачі розпізнавання образів

Існують різні парадигми штучних нейронних мереж, що відрізняються математичною моделлю штучного нейрона, способом організації нейронів, методами навчання та ін. Найбільше практичне використання знаходять штучні нейронні мережі, розроблені на основі багатошарового персептрона і згорткових нейронних мереж.

Багатошаровий персептрон (MLP) має три або більше шарів і використовується для класифікації даних, які не можна розділити лінійно. Це повністю пов'язаний тип штучної нейронної мережі, де кожен вузол у шарі з'єднаний з кожним вузлом у наступному шарі. MLP використовує нелінійну функцію активації, зазвичай гіперболічну дотичну або логістичну функцію.

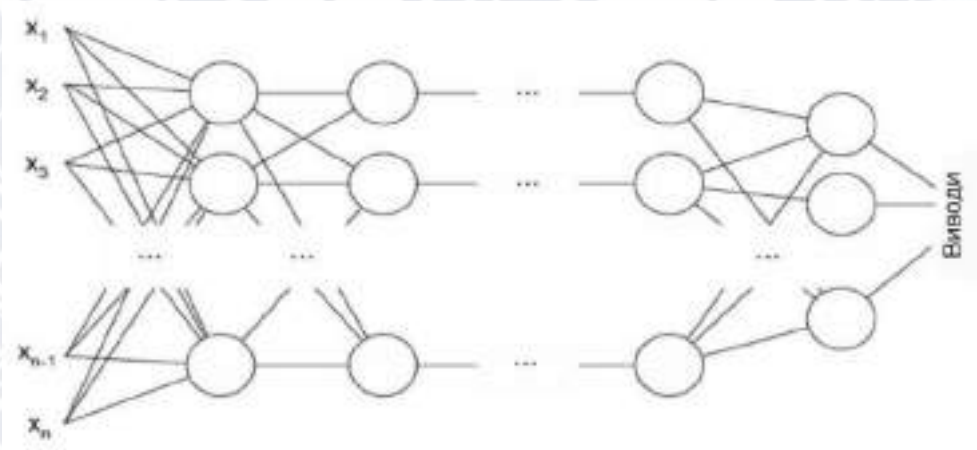

Рис. 2.1 – Багатошаровий перцептор

Цей тип нейронної мережі широко застосовується в технологіях розпізнавання мовлення та машинного перекладу. Багатошаровий персептрон або нейромережа прямого поширення з двома або більше шарами мають

більшу обчислювальну потужність і можуть обробляти нелінійні структури [11].

Згорткова нейронна мережа (Convolutional Neural Network, CNN) є варіацією багатошарових перцептронів, яка складається з одного або кількох згорткових шарів, що можуть бути повністю пов'язаними або об'єднаними. Згортковий шар виконує операцію згортки на вхідних даних, перш ніж передати результат наступному шару. Це дає змогу побудувати глибшу мережу з меншою кількістю параметрів.

Завдяки цій особливості CNN демонструють високу ефективність у завданнях розпізнавання зображень і відео, обробки природної мови та системах рекомендацій. Вони також досягають чудових результатів у семантичному аналізі, обробці сигналів та класифікації зображень.

CNN широко використовуються в аналізі зображень та розпізнаванні в аграрній сфері, де супутникові дані допомагають прогнозувати зростання та врожайність культур. На рис. 2.2 показано, як виглядає згорткова нейронна мережа [12].

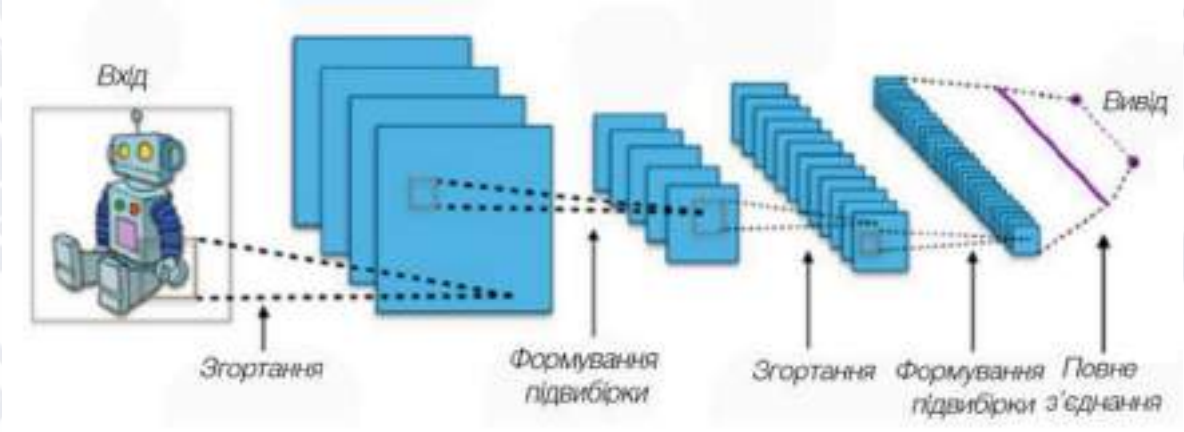

Рис. 2.2 – Згорткова нейронна мережа

В роботі розглядатиметься підхід до розпізнавання і класифікації об'єктів на основі згорткових нейронних мереж (convolutional neural network).

Згорткова нейромережа (CNN) добре витягує ознаки з зображень

завдяки своїй структурі. CNN використовує одну вагову маску для всього зображення, замість того щоб створювати ваги для кожного пікселя, як це робить персептрон. Це сприяє узагальненню інформації, а не запам'ятовуванню кожного пікселя. CNN також демонструє високу продуктивність для розпізнавання зображень, що є важливим для обчислення даних у реальному часі. Вони добре підходять для розпаралелення обчислень і стійкі до поворотів та зсувів зображень.

2.2 Вибір архітектури CNN для розпізнавання зображення

Архітектура згорткової нейронної мережі (CNN) є ключовим фактором у побудові системи для розпізнавання зображень. Вона визначає здатність системи до навчання, точність розпізнавання та ефективність роботи в реальному часі. Вибір відповідної архітектури має критичне значення особливо в умовах обмежених обчислювальних ресурсів, наприклад, при використанні на безпілотних літальних апаратах (БПЛА). У цьому розділі розглянемо основні архітектури CNN та їх відповідність завданням розпізнавання зображень на БПЛА.

Класичні архітектури CNN

Одними з перших моделей, що привели до прориву у сфері комп'ютерного зору, були LeNet, AlexNet та VGGNet.

- LeNet — це 7-рівнева згорточна мережа, розроблена LeCun у 1998 році, яка класифікує цифри та використовується кількома банками для розпізнавання рукописних чисел на чеках, оцифрованих у вхідних зображеннях у відтінках сірого 32x32 пікселя. Здатність обробляти зображення з вищою роздільною здатністю вимагає більших і більш згорнутих шарів, тому ця техніка обмежена доступністю обчислювальних ресурсів.

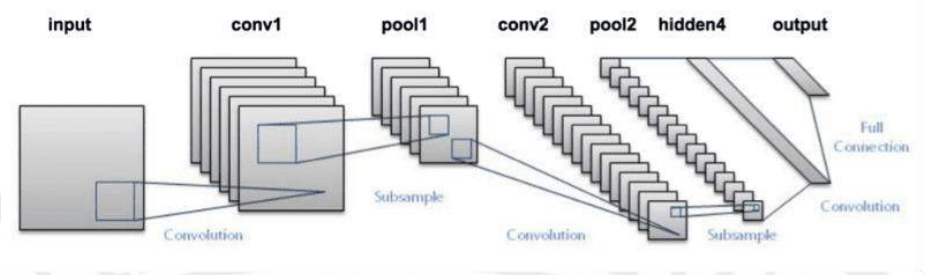

Рис. 2.3 – Архітектура LeNet

- AlexNet стала революційною завдяки використанню глибших шарів і апаратного прискорення на GPU, що дозволило значно покращити точність розпізнавання на змаганні ImageNet.

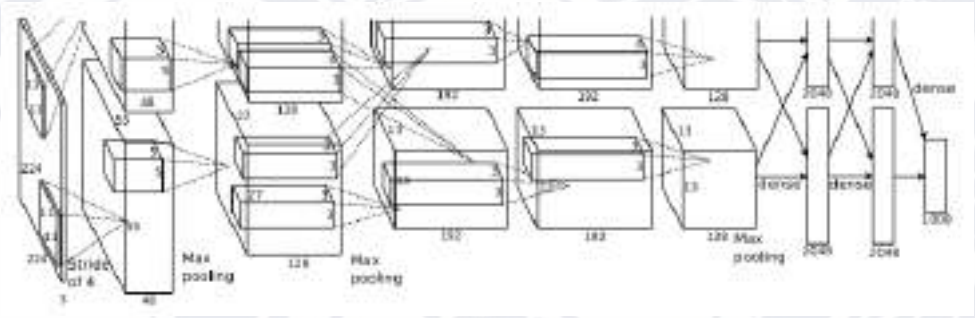

Рис. 2.4 – Архітектура AlexNet

- VGGNet відома своїми послідовними конволюційними шарами з малими ядрами (3x3), що дозволяє побудувати дуже глибоку мережу, хоча така архітектура потребує значних обчислювальних ресурсів.

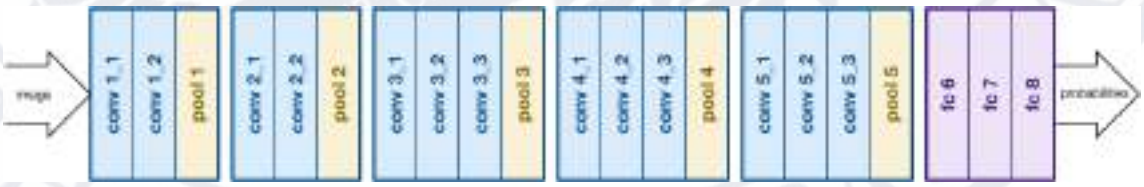

Рис. 2.5 – Архітектура VGGNet

Попри їх ефективність, ці класичні архітектури не завжди підходять для мобільних і вбудованих систем, таких як БПЛА, через велику кількість параметрів і високу вимогливість до ресурсів [13].

Архітектури для виявлення об'єктів у реальному часі

Для вирішення завдань виявлення об'єктів у реальному часі популярними є архітектури YOLO (You Only Look Once) і SSD (Single Shot Multibox Detector). Ці моделі спеціалізуються на швидкій обробці зображень і дозволяють виконувати розпізнавання об'єктів з мінімальною затримкою.

- YOLO обробляє все зображення за один прохід через мережу, забезпечуючи високу швидкість розпізнавання. Ця модель накладає на зображення сітку, розділяючи його на осередки. Кожна осередок намагається передбачити координати зони виявлення з оцінкою впевненості для цих полів і ймовірністю класів. Потім оцінка впевненості для кожної зони виявлення множиться на ймовірність класу, щоб отримати остаточну оцінку [14].

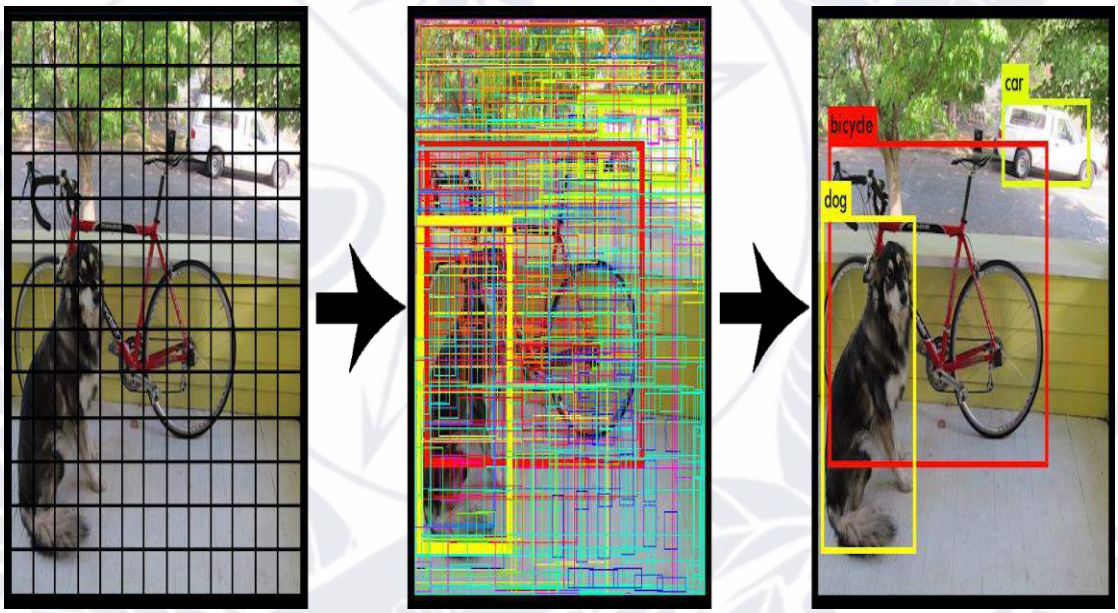

Рис. 2.6 – Приклад комірної сітки *YOLO* архітектури

- SSD (Single Shot MultiBox Detector) дозволяє виконувати виявлення декількох об'єктів на зображенні за один прохід, тоді як методи, засновані на регіональних мережах пропозицій (RPN), наприклад, серії R-CNN, вимагають двох етапів: спочатку створення пропозицій регіонів, а потім

виявлення об'єктів у кожній пропозиції. Завдяки цьому SSD працює значно швидше, ніж підходи, що використовують двоетапний процес RPN.

У SSD використовується архітектура VGG16 для створення карт ознак, а для виявлення об'єктів застосовується шар Conv4_3. Наприклад, якщо розглядати Conv4_3 з просторовим розміром 8×8 (він має розмір 38×38), кожна комірка (її також називають місцеположенням) генерує 4 прогнози об'єктів. Кожен прогноз включає межі об'єкта (bounding box) та 21 значення для класів (з урахуванням одного додаткового класу для відсутності об'єкта). Об'єкту присвоюється клас із найвищим значенням серед прогнозів. Для шару Conv4_3 загальна кількість прогнозів становить 38×38×4, тобто чотири прогнози на кожну комірку, незалежно від глибини карти ознак. Оскільки більшість прогнозів не містять об'єктів, SSD резервує клас "0" для позначення відсутності об'єкта [15].

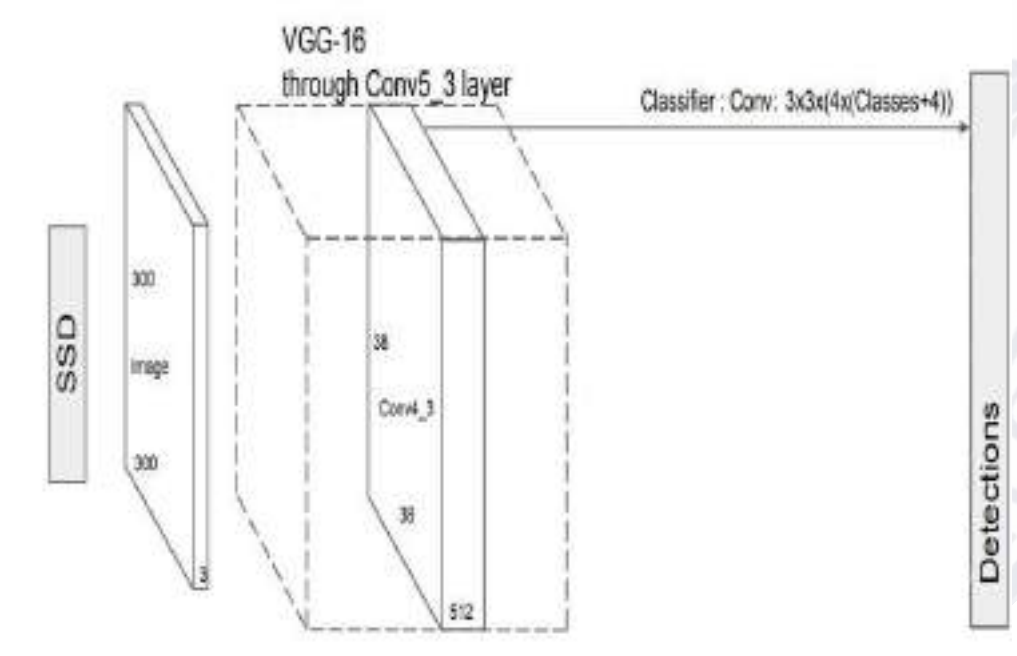

Рис. 2.7 – Принцип роботи SSD архітектури

При порівнянні для навчання нейронної мережі було обрано архітектуру YOLO (You Only Look Once), це обґрунтовано її високою швидкістю та ефективністю. Модель YOLO обробляє зображення за один прохід через

мережу, що дозволяє значно зменшити час виведення результатів у порівнянні з іншими підходами, такими як R-CNN. Його архітектура має глибоку згорткову мережу, яка генерує координати об’єктів і ймовірність їхнього класу в одній ітерації, що робить її надзвичайно придатною для застосувань у реальному часі, таких як відеоспостереження та автономні транспортні засоби.

2.3 Навчання нейронної мережі

Згорткові нейронні мережі (CNN) є основою багатьох сучасних методів розпізнавання та детекції об’єктів. Однією з найпопулярніших архітектур для швидкого й точного розпізнавання є YOLO (You Only Look Once), яка реалізована у фреймворку Darknet. Модель YOLO відзначається високою швидкістю, оскільки виконує детекцію за допомогою єдиного проходу через зображення, що робить її придатною для роботи в режимі реального часу.

Навчання згорткової нейронної мережі за допомогою YOLO Darknet є складним, але важливим етапом у розв’язанні задач розпізнавання та детекції об’єктів. Процес складається з кількох основних етапів, кожен з яких має важливе значення для отримання точної та ефективної моделі.

Першим етапом є підготовка даних, які будуть використовуватися для навчання моделі. Це включає збирання зображень, на яких необхідно визначити об'єкти, та їх маркування. Для цього використовують інструмент labelImg, який дозволяє створювати прямокутні мітки навколо об'єктів на зображеннях та зберігати їх у вигляді текстових файлів у форматі, сумісному з YOLO. Структура папок для тренувальних та тестових зображень організовується таким чином, щоб забезпечити зручність для подальшої обробки та навчання [16].

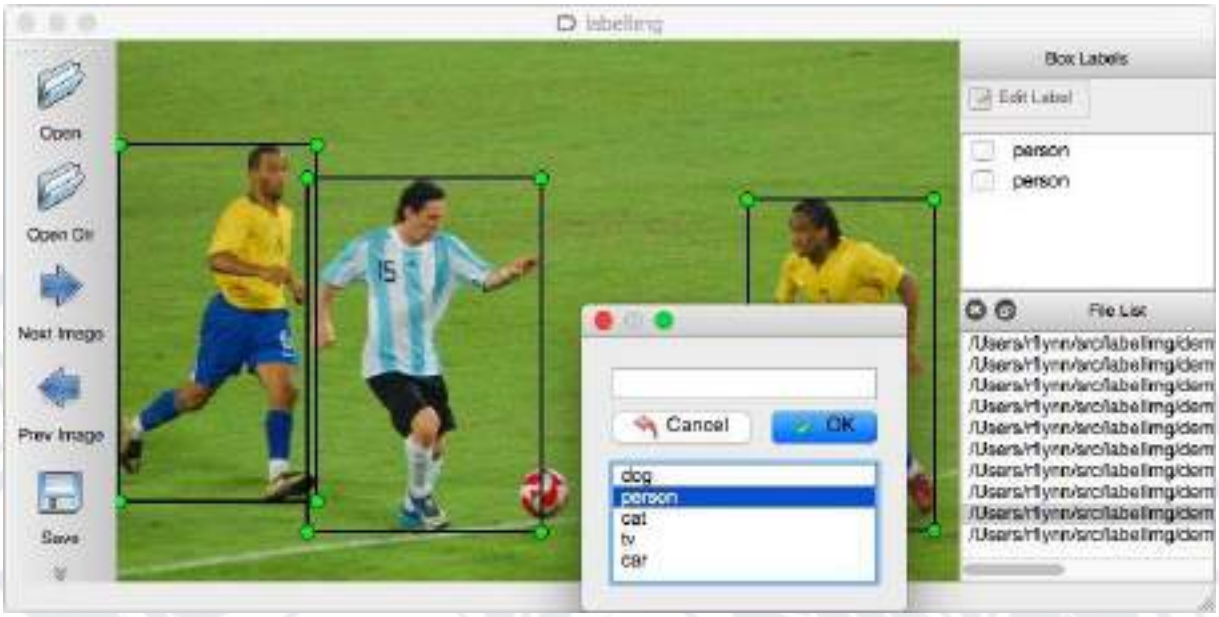

Рис. 2.8 – Програмне забезпечення для створення анотацій зображень LabelImg

Після підготовки даних, наступним етапом є налаштування конфігураційних файлів моделі YOLO, зокрема файлу yolov3.cfg, який визначає архітектуру нейронної мережі, кількість класів, розміри вхідних зображень, а також параметри навчання. Важливими параметрами є кількість епох (epochs), швидкість навчання (learning rate) та інші, які впливають на ефективність та швидкість навчання. Також необхідно налаштувати файл obj.data, в якому вказуються шляхи до тренувальних даних і кількість класів, що дозволяє моделі знати, з якими об'єктами вона працює.

Навчання YOLO Darknet передбачає використання графічних процесорів (GPU) для прискорення обчислень, що є надзвичайно важливим для великих датасетів, що містять тисячі зображень. Для цього використовуються бібліотеки CUDA та cuDNN, які значно пришвидшують обчислювальний процес за рахунок використання паралельних обчислень на GPU. Процес навчання передбачає кілька етапів, на яких мережа поступово оптимізує свої ваги, знижуючи функцію втрат і покращуючи точність прогнозування [17].

Під час навчання модель коригує свої параметри на основі функції втрат, яка вимірює різницю між передбаченими та реальними значеннями. Зазвичай використовуються метрики, такі як точність (precision) та повнота (recall), для оцінки продуктивності моделі. Чим більше епох пройшло, тим точніша модель, але з надмірним навчанням вона може почати втрачати здатність до узагальнення, тому важливо регулярно перевіряти її ефективність.

Після завершення навчання модель тестується на окремому наборі зображень, який не використовувався під час навчання. Це дозволяє оцінити здатність моделі до узагальнення — чи зможе вона правильно визначати об'єкти на нових, невідомих зображеннях. Для цього використовуються метрики, такі як точність (precision) і повнота (recall), що дають змогу визначити, як добре модель ідентифікує об'єкти та наскільки її прогнози відповідають реальним даним.

Таким чином, навчання згорткової нейронної мережі за допомогою YOLO Darknet є багатоступеневим процесом, що включає підготовку даних, налаштування параметрів мережі, навчання та тестування моделі. Використання технологій прискорення на GPU через CUDA та cuDNN дозволяє значно підвищити ефективність процесу, зменшивши час навчання при обробці великих обсягів даних. У результаті, ця методика є надзвичайно корисною для створення високопродуктивних систем для розпізнавання та детекції об’єктів.

Висновки до розділу 2

Навчання згорткової нейронної мережі за допомогою YOLO Darknet підтверджує ефективність цього підходу для вирішення завдань виявлення та класифікації об’єктів. Використання YOLO (You Only Look Once) дозволяє моделі виконувати детекцію за один прохід через мережу, завдяки чому досягається висока швидкість та точність у режимі реального часу. Це особливо важливо для завдань, які потребують швидкої обробки, таких як

відеоспостереження, аналіз потокових даних або управління безпілотними літальними апаратами.

Підхід YOLO базується на розділенні зображення на сітку, де кожна комірка відповідає за передбачення наявності об'єктів у ній. Це спрощує процес навчання та забезпечує модель можливістю швидко знаходити об'єкти з високою точністю, що значно скорочує обчислювальні витрати. Інтеграція з графічними процесорами (GPU) та використання бібліотек CUDA і cuDNN дозволяє досягти ще більшої продуктивності завдяки паралельним обчисленням.

Навчання YOLO Darknet включає кілька етапів, починаючи з підготовки даних і розмітки зображень, створення необхідної структури папок та конфігурації параметрів для моделі. Завдяки підтримці сучасних графічних процесорів модель може швидко оптимізувати свої ваги, знижуючи функцію втрат і покращуючи результати. У процесі тестування модель оцінюється за допомогою метрик точності та повноти, що дозволяє визначити її здатність до узагальнення та адаптації до нових зображень.

Завдяки своїй здатності до обробки великих обсягів даних і швидкому розпізнаванню, модель YOLO є одним із найбільш придатних варіантів для побудови систем виявлення об'єктів у реальному часі. Цей підхід, що поєднує ефективну архітектуру та апаратне прискорення, відкриває можливості для використання в різноманітних галузях, від автономного транспорту до сільського господарства, забезпечуючи точне розпізнавання об'єктів у складних умовах.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ

3.1 Вибір середовища розробки

Інтегроване середовище розробки (IDE) — це програмне забезпечення, що надає розробникам всі необхідні інструменти для створення програм в одному інтерфейсі. Це спрощує процес розробки, роблячи його більш організованим і ефективним. IDE зазвичай включає вбудований текстовий редактор для написання коду з функціями підсвітки синтаксису, автодоповнення та інших інструментів для полегшення кодування. Також середовище має компілятор або інтерпретатор, який перетворює написаний код на виконуваний формат або дозволяє виконувати скрипти безпосередньо під час написання.

Невід'ємною частиною IDE є налагоджувач, який допомагає розробникам знаходити й виправляти помилки, забезпечуючи можливість покрокового виконання програми та відстеження значень змінних під час виконання. Багато IDE інтегрують системи контролю версій, такі як Git, що дає змогу зручно відстежувати зміни в коді, керувати гілками проектів і працювати в команді. Також середовище часто підтримує інструменти для автоматичного тестування коду, що дозволяє перевіряти функціональність програм безпосередньо в процесі розробки [18].

Завдяки таким можливостям, IDE стає єдиним інструментом, у якому розробники можуть писати, компілювати, тестувати та налагоджувати свої проекти, не переходячи між різними програмами. Популярні приклади IDE — це Visual Studio, IntelliJ IDEA, PyCharm, Eclipse, що підтримують широкий спектр мов програмування і дозволяють працювати над проектами різної складності.

Visual Studio Code (VS Code) — це легке й потужне інтегроване середовище розробки, популярне серед розробників завдяки своїй гнучкості та широкому набору можливостей. Воно підтримує безліч мов програмування,

має багату екосистему розширень і підходить для різноманітних завдань, включно зі створенням модулів для розпізнавання зображень [19].

Ключові можливості:

1. Підтримка різних мов програмування: VS Code підтримує мови, які часто використовуються для розробки модулів комп'ютерного зору, зокрема Python, C++, і JavaScript. Завдяки розширенням, як-от Pylance або C/C++ Tools, можна легко працювати з популярними бібліотеками для машинного навчання, такими як TensorFlow, PyTorch або OpenCV.

2. Інтеграція з Jupyter Notebook: Завдяки розширенню Jupyter можна запускати, тестувати та експериментувати з кодом безпосередньо у VS Code. Це дуже зручно для тренування моделей машинного навчання або аналізу зображень, адже Jupyter забезпечує візуалізацію даних та швидку перевірку гіпотез.

3. Автодоповнення та підсвітка синтаксису: Функції автодоповнення та підсвітки синтаксису допомагають ефективніше писати код для розпізнавання зображень. За допомогою розширень для Python або C++ VS Code розпізнає бібліотеки, класи та функції, спрощуючи роботу з такими складними інструментами, як OpenCV чи TensorFlow [20].

4. Налагоджувач (debugger): VS Code має потужний вбудований налагоджувач, який дозволяє тестувати код крок за кроком, переглядати змінні та виконувати різні операції над ними під час виконання програми. Це допомагає шукати помилки в алгоритмах обробки зображень або нейронних мережах.

5. Інтеграція з Git: Вбудована підтримка Git дозволяє відстежувати зміни в коді, працювати з різними гілками та легко синхронізувати роботу з командою. Це важливо під час спільної розробки модулів розпізнавання зображень, коли кілька розробників працюють над одним проектом.

6. Інтеграція з Docker: За допомогою розширення Docker можна легко створювати та запускати контейнеризовані середовища для

розпізнавання зображень, що спрощує розгортання і тестування модулів у різних середовищах.

7. Розширення для підтримки штучного інтелекту: Є багато розширень для роботи зі штучним інтелектом та машинним навчанням, наприклад Python for Machine Learning. Вони полегшують роботу з моделями розпізнавання зображень, дозволяючи інтегрувати моделі безпосередньо у VS Code.

8. Термінал вбудований: VS Code містить вбудований термінал, що дозволяє запускати скрипти та командні утиліти для роботи з моделями, їхньої обробки та тренування без необхідності переходити до зовнішнього терміналу.

Завдяки цим функціям, Visual Studio Code є чудовим інструментом для розробки програмних модулів, що займаються розпізнаванням зображень, забезпечуючи простоту роботи, гнучкість і потужні можливості для інтеграції сучасних технологій машинного навчання та комп'ютерного зору [21].

3.2 Список використаних технологій

Python — це інтерпретована об'єктно-орієнтована мова програмування високого рівня, що використовує строгу динамічну типізацію. Вона була створена в 1990 році Гвідо ван Россумом. Її відмінною рисою є поєднання високорівневих структур даних із динамічною семантикою та зв'язуванням, що робить мову ідеальною для швидкої розробки програм і інтеграції існуючих компонентів. Python підтримує модулі та пакети, що сприяє модульності й повторному використанню коду [22].

Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);
- відкритий код (можливість редагувати його іншими користувачами) [23].

Рис. 3.1 ілюструє структуру системи. Портативний комп’ютер (ПК) підключено до дрона Tello через Wi-Fi для спілкування. Дрон передає зображення з постійною частотою 30 Гц, яка попередньо налаштована в програмному забезпеченні драйвера дрона. Ці зображення обробляються на ПК за допомогою скороченої версії алгоритму YOLOv4 для виявлення об’єктів. Користувачі мають можливість вибирати обмежувальні рамки відповідно до своїх вимог.

Загальна блок-схема архітектури системи показана на рис. 3.1. Дрон надсилає зображення на ПК, де отримані зображення обробляються за допомогою YOLOv4-tiny для виявлення людини. Якщо на зображенні виявлено людину, на екрані відображається її обмежувальна рамка. Зелені рамки на малюнку 1 представляють результати виявлення людини. Якщо користувач вибирає певний об’єкт інтересу, клацнувши його обмежувальну рамку, система виділяє людину в межах цієї рамки як рамку шаблону для

мережі SiamMask, що дозволяє подальше відстеження. Алгоритм стеження обчислює похибку між цілі та центром кадру. Ця помилка служить вхідним сигналом для ПІД-контролера, який генерує команди польоту для повороту, крену та висоти. Що стосується четвертої команди польоту, кроку, вона розраховується на основі відносної відстані відстежуваного об’єкта з використанням даних про його місцезнаходження. Якщо ціль на зображенні не виявлена, дрон зберігає свою позицію, доки ціль не з’явиться.

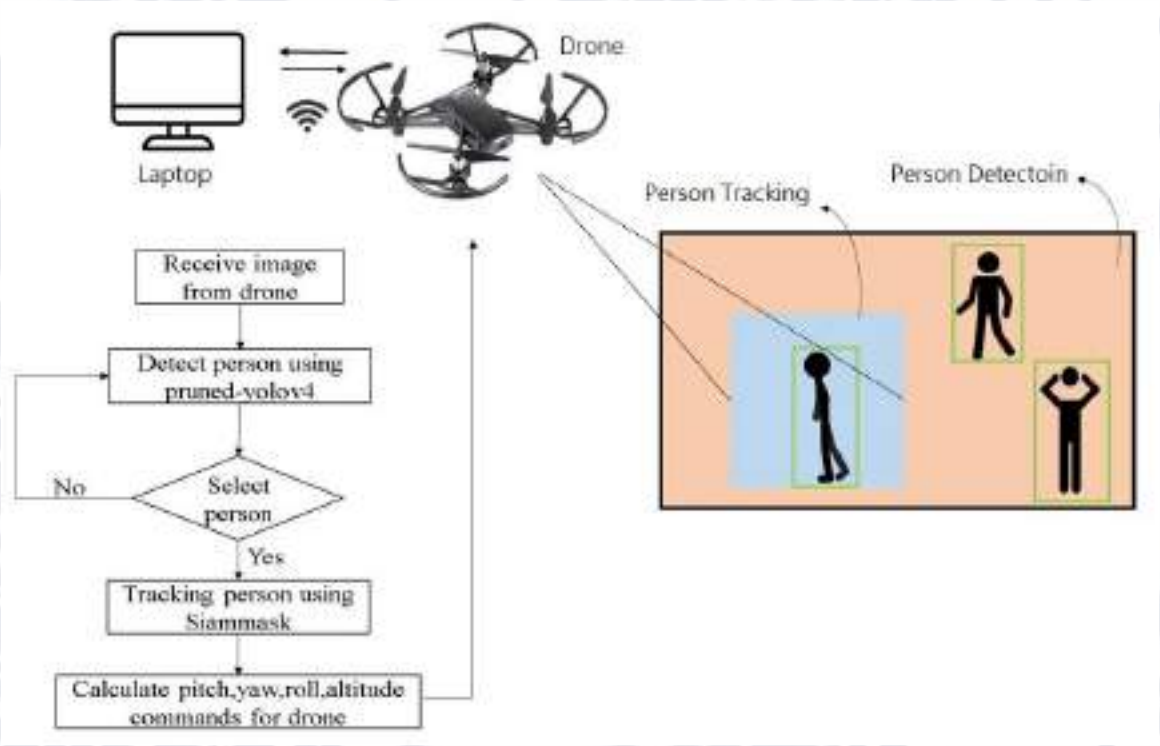

Рис. 3.1 – Структура системи

3.3 Відсікання моделі виявлення об’єктів

Під час процесу виявлення об’єктів нам спочатку потрібно навчити модель. Наш навчальний процес показано на рис. 3.2. Жовта частина на рис. 3.2 представляє модулі для структури Darknet. Світло-блакитна частина на рис. 3.2 представляє модулі для етапу обрізки. По-перше, ми використовуємо структуру Darknet для навчання базової моделі YOLOv4. Потім, під час етапу скорочення, ми виконуємо розріджене навчання, скорочення каналів, скорочення шарів і точне налаштування базової моделі за допомогою

структури Darknet. Після завершення етапу скорочення модель проходить навчання тонкого налаштування на фреймворку Darknet. Нарешті, ми розгортаємо модель на ноутбуці для виявлення.

A. Навчання Darknet

Ми використовуємо структуру Darknet для навчання моделі YOLOv4 [24] і налаштовуємо кілька гіперпараметрів на початковому етапі навчання, щоб підвищити точність і продуктивність моделі. Одним із перших гіперпараметрів, які потрібно налаштувати, є вхідний розмір мережі. Збільшення розміру вхідних даних допомагає виявляти маленькі об’єкти, хоча це також може сповільнити швидкість обчислення моделі та споживати більше пам’яті GPU. Важливо зазначити, що мережа YOLOv4 власноруч відбирає розмір вхідних даних із коефіцієнтом 32 як у вертикальному, так і в горизонтальному напрямках, тому ширина та висота вхідних даних мають бути кратними 32. Щоб досягти цього, ми вирішили використовувати 416 × 416 як розмір введення.

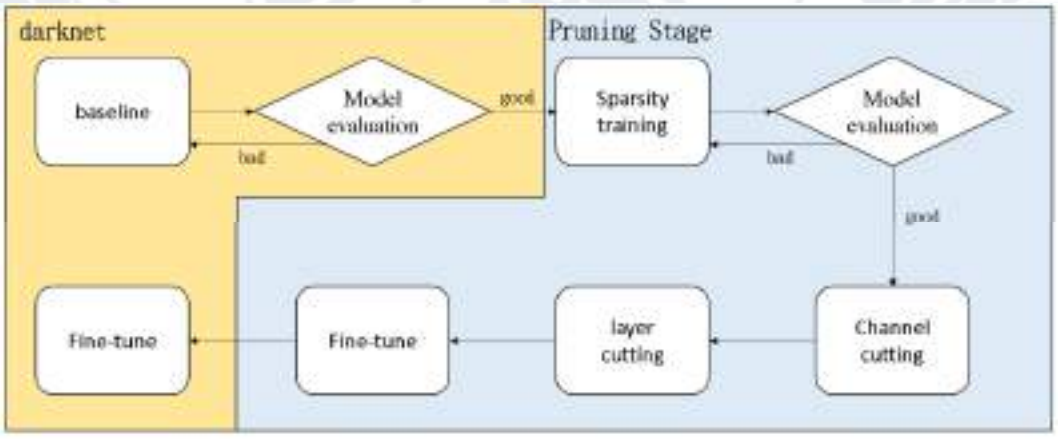

Рис. 3.2 – Етапи навчання для моделі виявлення.

Другим і третім гіперпараметрами, які потрібно налаштувати, є розмір партії та підрозділи. Ці параметри регулюються на основі продуктивності GPU. Гіперпараметр розміру партії представляє кількість зображень, які потрібно завантажити під час навчання, зі значенням за замовчуванням 64. Однак, якщо розмір пам’яті графічного процесора недостатній, він не зможе

завантажити 64 зображення одночасно. Щоб вирішити цю проблему, кожна партія додатково поділяється на кілька підпартій. Кожен суб-пакет подається в графічний процесор один за одним, доки пакет не буде завершено. У цьому дослідженні ми встановили розмір партії та підрозділи на 64 і 8 відповідно. Четвертий гіперпараметр, який потрібно налаштувати, — це кількість ітерацій (зауважте, що в рамці Darknet навчання вимірюється ітераціями, а не епохами). Згідно з інструкціями фреймворку Darknet, кожен клас об'єктів повинен мати принаймні 2000 ітерацій. Оскільки у нас лише один клас, кількість ітерацій має перевищувати 2000. Ми встановлюємо кількість ітерацій рівною 200 для досягнення вищої точності.

B. Стадія обрізки

Через апаратні обмеження ноутбука необхідно використовувати легку модель. Тому після навчання моделі за допомогою фреймворку Darknet її потрібно скоротити, щоб досягти мети полегшення. Ми використовуємо показники точності (mAP@0,5) і швидкості логічного висновку (BFLOPs), щоб оцінити скорочену модель. Однак важливо зазначити, що існує компроміс між точністю та швидкістю висновку. Якщо припустити, що апаратна конфігурація фіксована, коли модель скорочується до дуже малого розміру, її швидкість логічного висновку може зрости, але її точність зазвичай знижується. Перед обрізанням ваги з фреймворку Darknet проходять базовий процес навчання. Після завершення основного навчання отримана модель скорочується за допомогою стратегії скорочення з [25]. Ця стратегія передбачає спочатку проведення розрідженого навчання на моделі, де розрідженість каналу в глибоких моделях допомагає відрізати канал. Щоб полегшити скорочення каналу, кожен канал у згорткових шарах пов'язується з коефіцієнтом масштабування. Під час навчання регулярізація L1 застосовується до цих коефіцієнтів масштабування, щоб автоматично ідентифікувати неважливі канали. Канали з меншими значеннями коефіцієнта масштабування (помаранчевий колір) обрізаються (ліворуч). Після обрізки ми отримуємо компактну модель (права сторона), яка потім налаштовується для

досягнення порівнянної (або навіть вищої) точності з повністю навченою мережею. Процес обрізки показано на рис. 3.3.

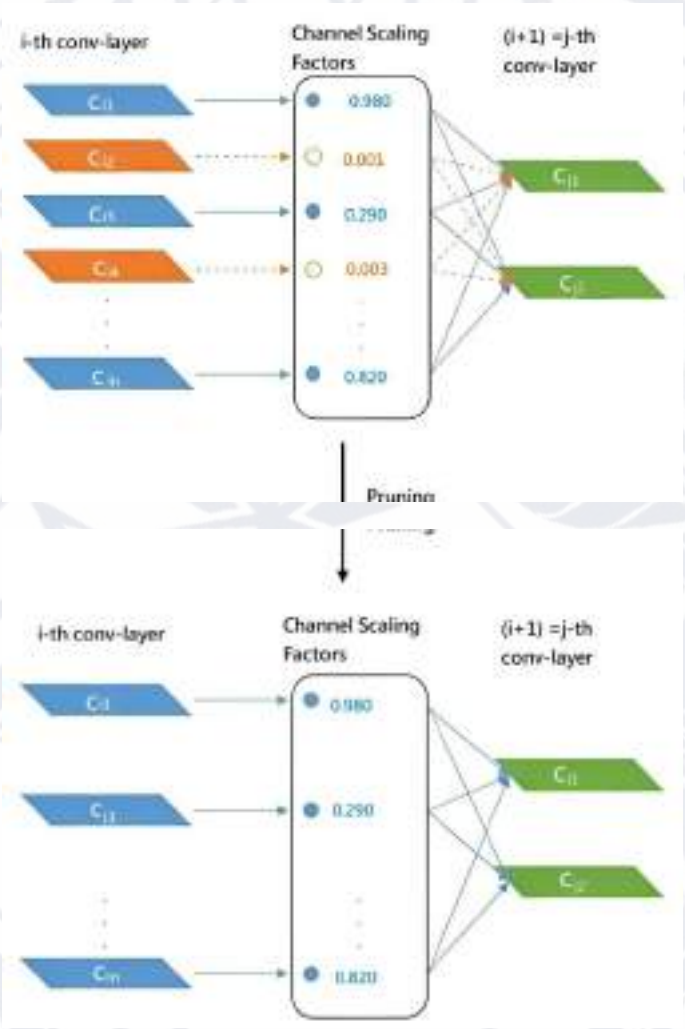

Рис. 3.3 – Спосіб обрізки

C. Розріджене навчання

Ми додаємо шар пакетної нормалізації після кожного згорткового шару в YOLOv4, щоб прискорити конвергенцію та покращити узагальнення. Рівень пакетної нормалізації використовує пакетну статистику для нормалізації згорткових функцій як:

$$\boldsymbol{y} = \boldsymbol{\gamma} * \frac{\boldsymbol{x} - \bar{\boldsymbol{x}}}{\sqrt{\boldsymbol{\sigma}^2 + \boldsymbol{\varepsilon}}} + \boldsymbol{\beta} \tag{3.1}$$

Тут x і σ 2 представляють середнє значення та дисперсію вхідних ознак у міні-серії відповідно. γ і β представляють тренований масштабний коефіцієнт і зміщення в шарі BN. У цьому дослідженні ми безпосередньо використовуємо коефіцієнт масштабування в шарі BN як показник важливості каналу. Щоб ефективно розрізнити важливі та неважливі канали, ми застосовуємо регулярізацію L1 до γ, забезпечуючи розріджене навчання на рівні каналу. На рис. 3.3 показано функцію втрат для навчання розрідженого аналізу:

$$\boldsymbol{L = loss_{yolo} + \alpha \sum_{\gamma \epsilon \Gamma} f(\gamma)} \quad (3.2)$$

D. Врізання каналу

Після завершення тренування розрідженого каналу можна виконувати різання каналу. Ось пояснення того, як продовжити різання каналу. Спочатку обчислюється загальна кількість каналів у магістралі. Після визначення кількості каналів відповідні значення γ зберігаються в змінній і сортуються в порядку зростання. Наступний крок — вирішити, які канали залишити, а які обрізати. Цього можна досягти, встановивши швидкість скорочення, яка представляє частку каналів, які потрібно скоротити. Швидкість обрізання зазвичай становить значення від 0 до 1, де більше значення вказує на вищий ступінь обрізання. Виконуючи ці кроки, можна виконати процес вирізання каналів, щоб вибірково зберегти або видалити канали на основі вказаної швидкості скорочення.

E. Розрізання шару

У магістралі YOLOv4 є кілька модулів CSPX, де кожен модуль CSPX складається з трьох рівнів CBL і модулів X ResUnit. Отримані функції цих модулів об’єднані разом, як показано на рис. 3.4a. Для вирізання шару ми в основному обрізаємо ResUnit у YOLOv4. Архітектура ResUnit проілюстрована на рис. 3.4b, яка складається з двох рівнів CBL і швидкого підключення. Рівень CBL складається з рівня Conv, рівня BN і функції активації Leaky ReLU, як показано на рис. 3.4c. Під час розрізання шару спочатку сортуються середні значення γ для кожного шару, і, оцінюючи попередній шар CBL кожного

скорочення, можна вибрати мінімальне значення для обрізання шару. Щоб забезпечити структурну цілісність YOLOv4, під час скорочення одного ResUnit одночасно скорочуються як ярлик, так і попередній шар CBL, що призводить до скорочення всього трьох шарів [26].

F. Тонка настройка

Різні стратегії скорочення та порогові параметри дають різний вплив на скорочену модель. Іноді точність скороченої моделі може навіть підвищитися, хоча в більшості випадків скорочення може мати негативний вплив на точність моделі. У таких випадках необхідно виконати точне налаштування обрізаної моделі, щоб компенсувати втрату точності, викликану обрізанням. Точне налаштування має вирішальне значення для відновлення точності обрізаної моделі. У наших експериментах ми безпосередньо перенавчали Pruned-YOLOv4, використовуючи ті самі гіперпараметри навчання, що й у звичайному процесі навчання для YOLOv4 [27].

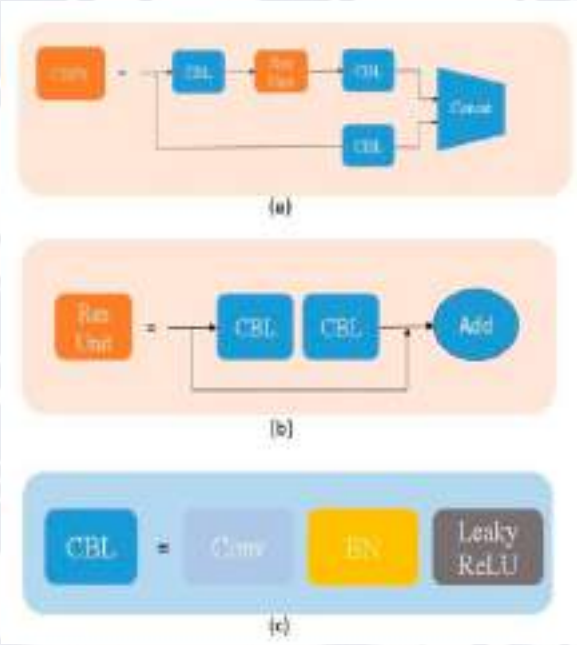

Рис. 3.4 – (a) CSPX, (b) ResUnit і (c) рівень CBL в YOLOv4

3.4 Використання дрона DJI Tello

Дрон Tello є універсальною платформою для досліджень і навчання, розробленою компанією Ryze Tech у співпраці з DJI та Intel. Завдяки

компактності та зручності використання, Tello є популярним вибором для освітніх і наукових проєктів. Оснащений HD-камерою з роздільною здатністю 5 Мп, дрон може транслювати відео у форматі 720p з електронною стабілізацією, що забезпечує якісне та чітке зображення. Використовуючи процесор Intel, Tello здатний обробляти базові алгоритми комп'ютерного зору та обробки зображень безпосередньо на борту.

Рис. 3.5 – Дрон Tello

Програмованість є важливою перевагою Tello. За допомогою Tello SDK можна налаштовувати автоматизовані польоти, створювати алгоритми навігації й обробки даних, а також розробляти нові рішення для виконання різноманітних завдань. Підтримка мови Python дозволяє легко інтегрувати його з популярними бібліотеками для комп’ютерного зору, такими як OpenCV. Додатково Tello підтримує Scratch — візуальну платформу для початкового навчання програмуванню, що робить його ідеальним для школярів і початківців.

Дрон обладнаний сенсорами, які сприяють стабілізації під час польоту. Барометр і оптичні сенсори дозволяють йому зберігати стабільність висоти та положення, навіть без використання GPS, що робить його особливо корисним для використання в приміщеннях. Ці функції розширюють можливості для автономного польоту та дослідження алгоритмів навігації, а також для експериментів з комп'ютерним зором.

Завдяки таким характеристикам, Tello широко застосовується для навчання робототехніці, обробці зображень та алгоритмам комп'ютерного зору. Його можливості особливо цінні для роботи з моделями розпізнавання об’єктів і автоматизації польотів, що робить його відмінним інструментом для початкових досліджень у сфері штучного інтелекту й дронів [28].

3.5 Відстеження об'єктів і управління дронами

Ноутбук виконує виявлення дронів у режимі реального часу, дозволяючи користувачам відстежувати виявлені цілі, доки відстеження не буде завершено. Об’єкти, виявлені за допомогою Pruned-YOLOv4, представлені обмежувальними рамками, кожна з яких містить чотири координати (x, y, w, h). Тут x і y представляють координати верхнього лівого кута обмежувальної рамки, а w і h представляють ширину та висоту обмежувальної рамки відповідно. Після отримання координат ми постійно визначаємо положення миші користувача. Якщо клацання миші потрапляє в обмежувальну рамку, чотири координати обмежувальної рамки передаються в модуль відстеження об’єктів, який використовує SiamMask [29]. SiamMask — це алгоритм відстеження цілей, заснований на сіамських нейронних мережах [30]. Спочатку були запропоновані сіамські нейронні мережі Бромлі та ЛеКун для вирішення проблем перевірки підписів [31] і з тих пір широко застосовуються в різних сферах, таких як зіставлення зображень і відстеження цілей.

У задачі відстеження цілі сіамські нейронні мережі використовують дві ідентичні підмережі зі спільними параметрами та вагами. Шаблон відстеження подається в мережу, і виходять вихідні ваги. Ці вагові коефіцієнти потім зіставляються з вихідними ваговими показниками регіону пошуку для обчислення оцінки подібності. Розташування цілі, яку необхідно відстежувати, визначається шляхом обчислення карти балів відповіді. Спираючись на традиційну сіамську мережу, SiamMask включає обчислення

сегментації цілі, що дозволяє виділити контур цілі. Це допомагає пом'якшити вплив варіацій цільових функцій, викликаних обертанням і деформацією.

Під час відстеження SiamMask одночасно повертає зображення з обмежувальною рамкою відстежуваного об'єкта. Ця рамка містить інформацію про положення предмета на зображенні. Обмежувальна рамка відстежуваного об'єкта складається з чотирьох точок:

$(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max})$

Ми можемо використовувати ці чотири точки для обчислення центру положення об'єкта, яке обчислюється як:

$$(\boldsymbol{x_{center}, y_{center}}) = \frac{\boldsymbol{x_{min}+x_{max}}}{\boldsymbol{2}}, \frac{\boldsymbol{y_{min}+y_{max}}}{\boldsymbol{2}} \quad (3.3)$$

Щоб точно відслідковувати об'єкт, необхідно знати точне центральне положення екрана дрона. Це пов'язано з тим, що центр виявленого об'єкта завжди повинен збігатися з центром екрана дрона для правильного відстеження. Розрахунок невідповідності між центром екрана дрона та центром об'єкта виконується як:

$$\boldsymbol{e_x = imgx_{center} - x_{center}} \quad (3.4)$$

$$\boldsymbol{e_y = imgy_{center} - y_{center}} \quad (3.5)$$

Щоб досягти ефективного відстеження e_x і e_y завжди мають дорівнювати нулю або бути близькими до нього.

Всього дрон має чотири параметри керування: нахил, поворот, висота та крок. Нахил контролює бічний рух дрона, поворот контролює обертання дрона за або проти годинникової стрілки, висота контролює вертикальний рух дрона, а крок контролює рух дрона вперед або назад. На рис. 3.6 показані основні маневри польоту дрона [32].

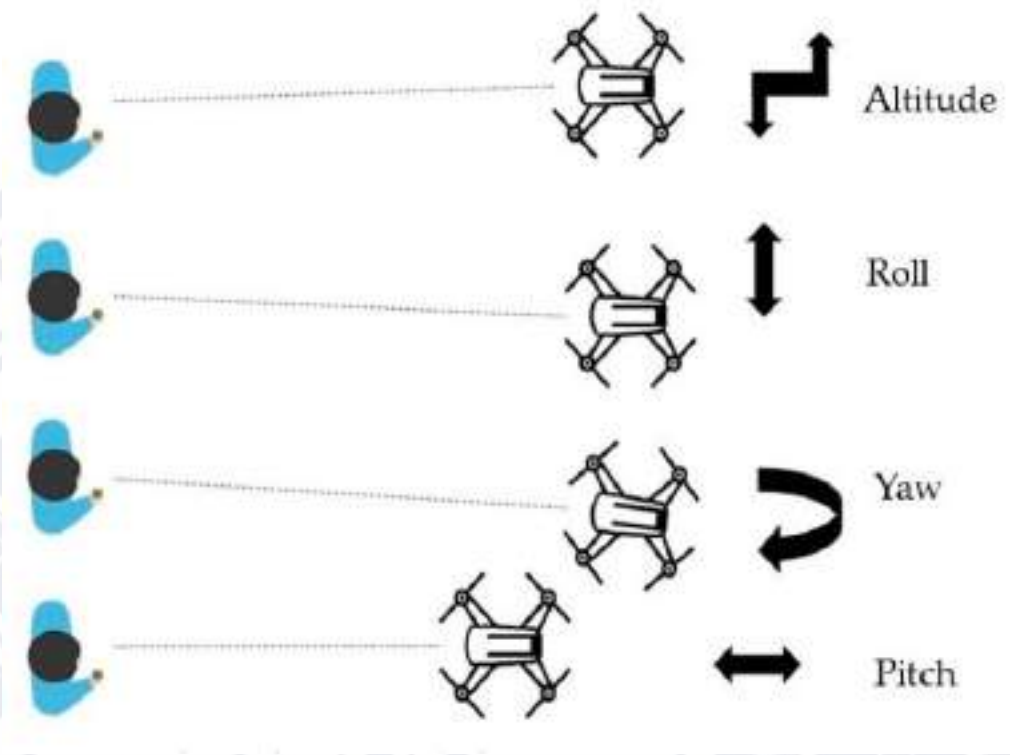

Рис. 3.6 – Основні маневри дрона

Далі розглянемо, як PID контролює політ дрона. Очевидно, що використовуючи центральну точку відстежуваного об’єкта та центральну точку екрана, ми можемо отримати похибку по осі X. Ця помилка пов’язана з креном дрона для бічного руху та поворотом за годинниковою стрілкою або проти неї. Якщо дрон виявляє, що об’єкт рухається ліворуч або праворуч, ми можемо змінити напрямок дрона, щоб повернути його обличчям до об’єкта, або виконати бічні рухи, щоб не відставати від нього. Крім того, існує вісь висоти, яка передбачає рух вперед і назад. Віднімаючи від бажаної ідеальної відстані відстань між дроном і реальним об’єктом, ми можемо обчислити похибку відстані та відповідно керувати рухом дрона вперед або назад. Нарешті, щодо висоти, шляхом віднімання Y-координати відстежуваного об’єкта з Y-координати центру екрана, ми можемо отримати похибку по осі Y. Це дозволяє розрахувати необхідну висоту для вертикального підйому або зниження дрона. Конкретні методи контролю проілюстровано на рис. 3.7.

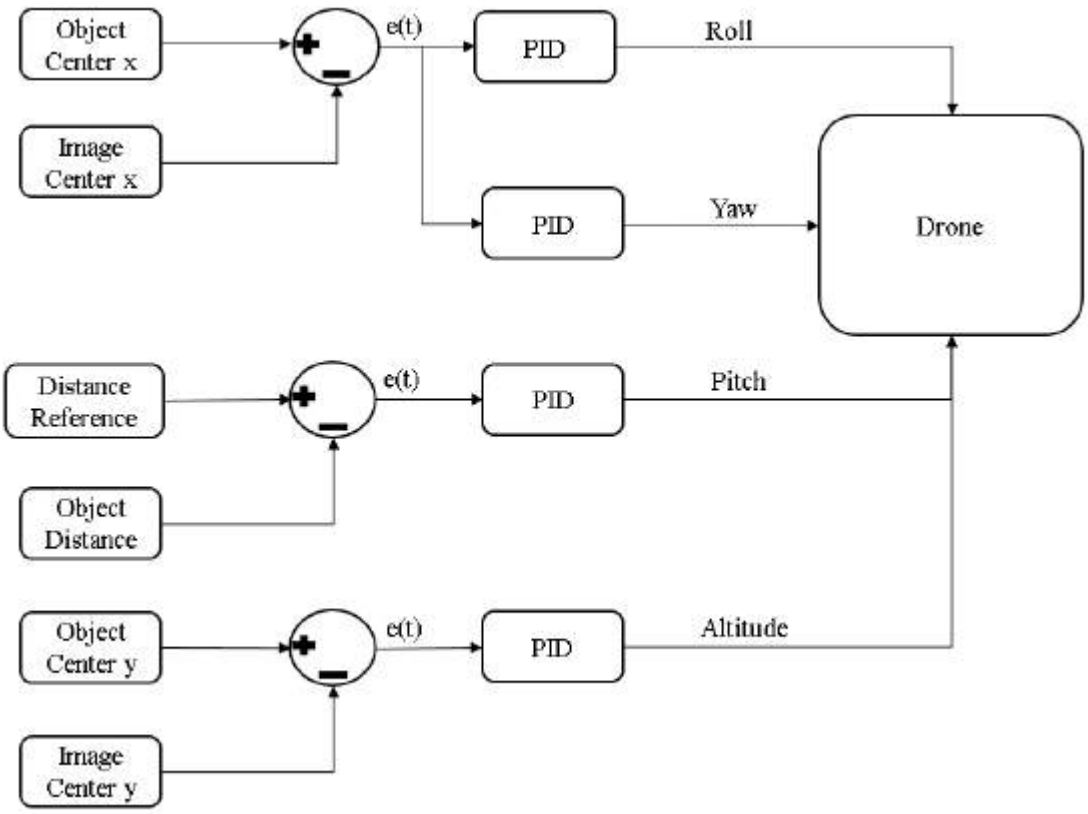

Рис. 3.7 – Схема управління дроном

Коли справа доходить до похибки осі X, вибір між креном і ухилом розроблений так, щоб залежати від похідної (D) PID-регулятора. Термін D представляє швидкість зміни помилки. Якщо помилка швидко змінюється, дрону потрібно збільшити потужність, щоб не відставати від рухів об’єкта. Однак, якщо об’єкт продовжує рухатися вздовж осі X, як показано на рис. 3.8a, для швидкого відстеження цілі потрібен сильніший контроль. Червоний прямокутник позначає обмежувальну рамку об’єкта. Тому ми обираємо варіант нахилу, що означає виконання бічних рухів праворуч, щоб слідувати за об’єктом, представленим зеленою стрілкою, зображеною на рис. 3.8a. З іншого боку, якщо відстежуваний об’єкт не демонструє значного руху, вибирається варіант повороту, який вимагає лише коригування курсу, щоб слідувати за ціллю, представленою зеленою стрілкою, зображеною на рис. 3.8b.

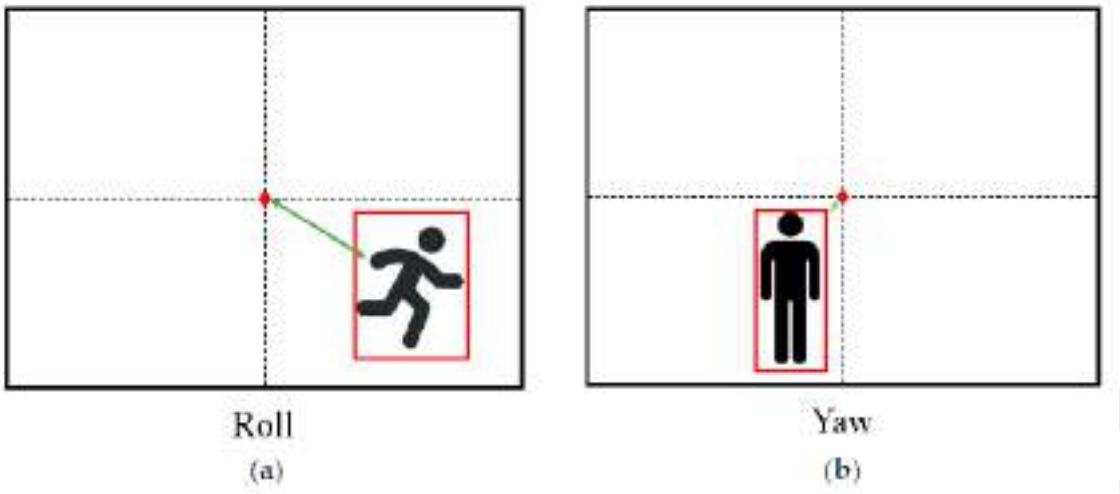

Рис. 3.8 – Приклади руху об’єкта (а) Нахил, (б) Поворот.

3.6 Навчання моделі та розробка програмних модулів

Розробка програмного модуля для управління квадрокоптером DJI Tello починається з визначення логіки системи та інтеграції її складових компонентів. Для цього було створено структурну схему, яка ілюструє взаємодію ключових модулів, зокрема машинного навчання, сенсорних даних, відеопотоку та команд управління, що реалізовані на основі Python Backend. Нижче наведено схему, яка демонструє архітектуру програмного модуля та її зв’язки.

Рис. 3.9 – Структурна схема програмного модуля управління дроном DJI Tello

У цій роботі ми використовуємо ROS [33] (роботизовану операційну систему) для реалізації виявлення зображення та відстеження для керування БПЛА. Навчання проводилось на такому апаратному та програмному забезпеченні:

- операційна система Ubuntu 24.04,
- процесор Intel core i3-9100F - 3,60 ГГц,
- відеокарта Nvidia GTX-1060 - 2 ГБ.

Через апаратні обмеження ноутбука потрібні легкі моделі. Тому для детектора об’єктів ми навчаємо згорткову нейронну мережу на основі архітектури YOLOv4. Darknet YOLOv4 швидше та точніше, ніж real-time нейронні мережі Google, TensorFlow, EfficientDet та FaceBook, Pytorch/Detectron, RetinaNet/MaskRCNN. YOLOv4-tiny використовуэться для спрощення структури мережі та зменшення параметрів, що робить його придатним для розробки на мобільних і вбудованих пристроях. Також, YOLOV4- оптимальна для використання в real-time, так, як. мережа лежить на кривій оптимальності за Парето на графіку AP(accuracy)/FPS(speed), який зображено на рис. 3.10 [34].

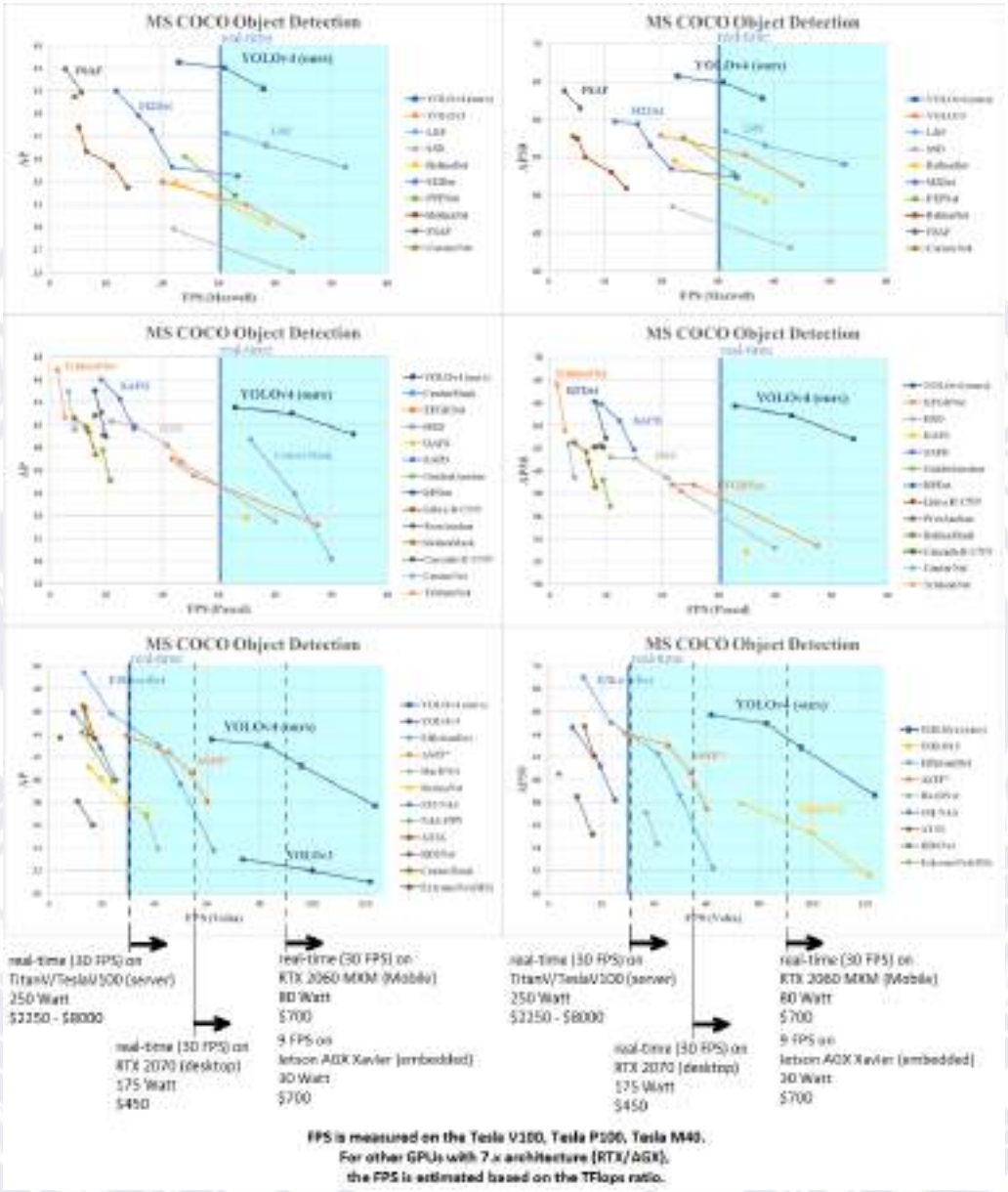

Рис. 3.10 – Графіки точності (AP) і швидкості (FPS) безлічі нейронних мереж для виявлення об'єктів заміряних на відеокартах GPU TitanV/TeslaV100, TitanXP/TeslaP100, TitanX/TeslaM40 для двох основних індикаторів точності AP50:95

Щоб покращити виявлення об’єктів у реальному часі, пропонується метод виявлення об’єктів на основі YOLOv4-tiny, який має 3 шари, для більшої точності. Він більше підходить для виявлення об'єктів у реальному часі. Також YOLOV4 дозволяє виявити відразу кілька об'єктів, що належать різним класам, що дозволяє звести всю операцію знаходження координат до одного

прогону мережі. Детектор для розпізнавання сегментів був навчений за допомогою репозиторію Darknet [35]. Навчання проходило порівняно швидко, оскільки проводилося відразу з усіма сегментами. У цій роботі цільовим об'єктом для виявлення є лице людини, але в майбутніх реалізацій керування дроном за допомогою жестів руки, було обрано таких 6 варіантів жестів. В результаті, ми отримали модель, яка складається із 7 класів.

Таблиця 3.1 – Класи моделі

Об’єкт	Завдання
Лице людини	Відстеження
Up	Летіти доверху
Down	Летіти донизу
Left	Летіти вліво
Right	Летіти вправо
Fist	Розпочати відстеження
Ok	Закінчити відстеження

У конфігураційному файлі були задані параметри: кількість класів, розмір партій та шари YOLO. Перед початком навчання я завантажила попередньо навчені ваги, що допомогло прискорити процес. Для навчання моделі YOLO я створила власний датасет із 2500 зображень. За допомогою інструмента LabelImg було розмічене обличчя, а результати розмітки збережені у форматі YOLO. Було створено файли train.txt і test.txt, щоб визначити зображення для навчання і тестування. Структура папок була організована таким чином, щоб усі дані, конфігураційні файли та результати тренувань зберігалися в одному місці. Я налаштувала модель для навчання, використовуючи 80% датасету для тренування і 20% для тестування. Під час тренування результати регулярно зберігалися у вигляді чекпоінтів у спеціальній папці backup.

Загалом, час навчання моделі становив 9 годин, так як для 7 класів моделі необхідно пройти 14 000 ітерацій (classes*2000). В результаті ми отримали 7 файлів weights (так звані ваги), які створювались в результаті навчання моделі, після кожної 2 000 ітерації. На рис. 3.11 зображено mAP-діаграму (червона лінія) у вікні збитків. mAP обчислювався для кожних 4 епох за допомогою valid = test.txt файлу, вказаного у data файлі (1 Epoch = images_in_train_txt / batch ітерацій).

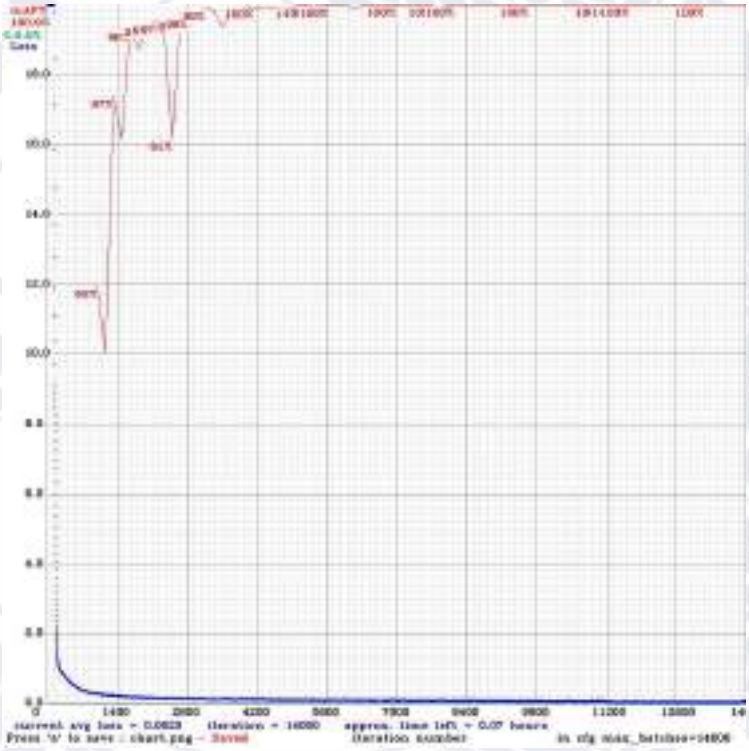

Рис. 3.11 – mAP-діаграма

Де, IoU (перетин через об’єднання) – середнє перетин через об’єднання об’єктів і виявлень для певного порогу = 0,24 та mAP (середня точність) - середнє значення average precisions для кожного класу, де average precision середнє значення 11 точок на PR-кривій для кожного можливого порогу (кожної ймовірності виявлення) для того самого класу.

Результати виявлення об’єктів навченої моделі YOLOv4-tiny, яка має 7 класів (Lena, up, down, right, left, fist, ok) зображено на рис. 3.12.

Рис. 3.12 – Результати виявлення об`єктів

Після завершення навчання моделі для виявлення та розпізнавання об'єктів ми переходимо до етапу інтеграції програмного забезпечення, яке забезпечить взаємодію між моделлю машинного навчання та квадрокоптером DJI Tello.

Розглянемо етап розробки програмного модуля.

На цьому етапі розробляється Python Backend, який включає реалізацію основних компонентів програмного модуля:

– обробки сенсорних даних,

– передавання відеопотоку,

- виконання команд управління,
- забезпечення логіки роботи всієї системи.

Модуль DnnObjectDetect, використовується для виявлення об'єктів за допомогою нейронної мережі, зокрема моделі DarkNet. Ось основні функціональні елементи цього модуля:

1. Ініціалізація (__init__):

Завантажує конфігурацію та ваги моделі, задає класи, поріг впевненості, а також тип об'єкта для відстеження (наприклад, "Lena").

2. Попередня обробка зображень (preprocess_image):

Масштабує та перетворює зображення для подачі в нейронну мережу.

3. Виявлення об'єктів (detect):

Виконує:

- Обчислення характеристик зображення.
- Визначення ймовірностей класів для кожного об'єкта.
- Виділення координат об'єктів із використанням немаксимального придушення (NMS) для уникнення перекриття блоків.

У випадку типу об'єкта Lena оцінюються справжні позитивні результати, включно з розміром області та центральною точкою.

4. Відображення результатів (draw_detections):

Накладає межі та мітки на зображення для кожного виявленого об'єкта.

5. Виявлення та візуалізація (detect_and_draw):

Об'єднує виявлення об'єктів та їхнє відображення на зображенні.

Цей клас ідеально підходить для завдань комп'ютерного зору, таких як відстеження об'єктів у реальному часі, використовуючи модель DarkNet.

Модуль FollowObject надає функціональність для відстеження об'єкта за допомогою дрону, зокрема дрону Tello. Він інтегрує комп'ютерне зір, фільтрацію Калмана та управління дроном для відстеження та слідування за об'єктами, такими як обличчя або інші виявлені об'єкти. Опис основних компонентів і функцій коду:

Клас FollowObject керує відстеженням об'єктів (наприклад, облич) в реальному часі за допомогою дрону. Він використовує різні технології, такі як виявлення об'єктів на основі DNN, фільтрацію Калмана для оцінки стану та управління потоками для паралельної обробки.

Ініціалізація (__init__)

Залежності: Клас залежить від OpenCV (cv2), threading, NumPy та власних модулів (safethread, kalman, і dnnobjectdetect).

Аргументи:

- tello: Об'єкт дрону Tello, який буде отримувати команди.
- model_configuration, model_weights: Шляхи до конфігураційного файлу та ваг для моделі виявлення об'єктів на основі DNN.
- CLASSES: Список класів об'єктів, які може виявити модель.
- CONFIDENCE: Мінімальний поріг для впевненості виявлення (за замовчуванням 0.8).
- DETECT: Визначає об'єкт для виявлення (наприклад, обличчя або будь-який об'єкт).
- DEBUG: Прапор для виведення відлагоджувальних повідомлень.

Внутрішні компоненти:

- Виявлення об'єктів на основі DNN (self.dnnfacedetect).
- Фільтри Калмана (self.kf, self.kfarea) для відстеження позиції та площі виявленого об'єкта.
- Потоки (self.wt) для фонової обробки циклу відстеження об'єкта.

Методи

- set_default_distance() та set_default_area():

Встановлюють стандартні значення для відстеження відстані та площі.

- set_tracking():

Налаштовує, які осі (горизонталь, вертикаль, відстань, обертання) будуть використовуватися для відстеження.

- set_image_to_process():

Приймає зображення для обробки з метою виявлення об'єктів.

- set_detection_periodicity():

Встановлює частоту, з якою повинна відбуватися виявлення об'єктів (в циклах).

- safety_limiter():

Забезпечує, щоб команди руху для дрону не перевищували безпечні межі, обмежуючи значення.

- __worker():

Основний фоновий процес, який обробляє зображення, виявляє об'єкти та надсилає команди для дрону.

Використовує DNN для виявлення об'єктів, фільтрацію Калмана для згладжування рухів об'єкта та надсилає команди для корекції позиції дрону по горизонталі, вертикалі, обертанням та відстані.

- draw_detections():

Візуалізує результати виявлення на зображенні, включаючи обведення об'єктів прямокутниками, відображення HUD (наприклад, заряд батареї, висота) та поточної позиції об'єкта щодо центру зображення.

Основні функції

- Виявлення об'єктів на основі DNN: Використовує модель на основі глибоких нейронних мереж (наприклад, попередньо натреновану модель, таку як YOLO або MobileNet) для виявлення об'єктів в реальному часі.
- Фільтр Калмана: Два окремі фільтри Калмана використовуються для згладжування рухів об'єкта та більш точної оцінки його позиції. Один відповідає за відстеження позиції (горизонталь/вертикаль), а інший — за відстеження площі об'єкта (для оцінки відстані).
- Режими відстеження: Дрон може відстежувати об'єкт по горизонталі, вертикалі, відстані та обертанням незалежно або в комбінації, залежно від налаштувань.
- Безпека: Функція безпечного обмеження гарантує, що команди для руху дрону не перевищують певні пороги, щоб уникнути небажаної поведінки.

• Відлагодження та візуалізація: Включає можливість виведення відладкової інформації та малювання обведення навколо виявлених об'єктів, а також іншої телеметрії, наприклад, рівень заряду батареї та висота.

Цей клас призначений для застосувань, де потрібно, щоб дрон слідував за рухомим об'єктом (наприклад, за людиною або предметом). Дрон використовує виявлення об'єктів для ідентифікації цілі, а потім коригує свою позицію та рух на основі місцезнаходження виявленого об'єкта та змін у його положенні.

Основні концепції

• Фільтр Калмана: Алгоритм, який використовується для оцінки стану динамічної системи з шумними вимірюваннями. У цьому випадку використовується для згладжування відстеження позиції об'єкта та його площі.

• Потоки: Метод __worker() працює в окремому потоці для обробки зображення в реальному часі, що дозволяє забезпечити плавну роботу під час руху дрону.

• Виявлення об'єктів на основі DNN: Модель для виявлення об'єктів завантажується та використовується для виявлення об'єктів (облич або будь-яких заданих об'єктів) на вхідних зображеннях.

Цей код є основою для створення автономної системи для дрону, здатної слідувати за вказаним об'єктом, використовуючи передові методи, такі як фільтрація Калмана та виявлення об'єктів на основі DNN.

Модуль TelloConnect, є класом для управління дроном Tello за допомогою UDP-з'єднання. Ось його основні компоненти та функціональність:

Імпорти:

• Модулі socket, cv2, Queue використовуються для роботи з сокетами, обробкою відеопотоку та зберіганням кадрів.

• Модуль safethread використовується для забезпечення багатопотоковості та безпеки потоків.

Конструктор (__init__):

- Ініціалізує з'єднання з дроном Tello через UDP-сокети для команд та стану.
- Створюються потоки для обробки відео, періодичних команд, отримання стану та інших функцій.
- Додатково налаштовуються параметри для відео, розміри зображення, і параметри для періодичних команд.

Методи:

- set_image_size: задає розміри зображення для відео.
- get_frame: отримує кадр із відеопотоку.
- add_periodic_event: додає періодичну команду для відправлення.
- send_cmd_return: надсилає команду до дрону та чекає на відповідь.
- send_cmd: надсилає команду до дрону без отримання відповіді.
- wait_till_connected: чекає, поки дрон не підключиться і не перейде в командний режим.
- start_communication і stop_communication: запускає та зупиняє отримання даних і команд.
- start_video і stop_video: запускає та зупиняє відеопотік.
- __receive, __state_receive, __periodic_cmd: допоміжні методи для обробки команд, стану і періодичних подій.

Періодичні команди:

Клас підтримує механізм періодичного надсилання команд, що дозволяє підтримувати активність підключення та отримувати важливі дані від дрону.

Мультипоточність:

Модуль використовує декілька потоків для паралельної обробки відео, команд, стану та періодичних подій, забезпечуючи ефективну роботу без блокувань.

Використання сокетів:

Для зв'язку з дроном використовуються UDP-сокети для надсилання команд та отримання стану. Це дозволяє ефективно управляти дроном та отримувати з нього дані.

Приклад використання:

1. Ініціалізація:

```
tello = TelloConnect(TELLOIP='192.168.10.1', UDPPORT=8889, VIDEO_SOURCE="udp://@0.0.0.0:11111", UDPSTATEPORT=8890, DEBUG=False)
```

2. Підключення до дрону:

```
tello.wait_till_connected()
```

3. Запуск відео:

```
tello.start_video()
```

4. Отримання кадру:

```
frame = tello.get_frame()
```

5. Надсилання команд:

```
tello.send_cmd('command')
```

6. Остановка зв'язку:

```
tello.stop_communication()
```

Клас розрахований на використання з дроном Tello, тому налаштування для команд і стану специфічні для цього пристрою.

Для зручності роботи з відео, використовується потокова обробка через OpenCV.

Модуль SafeThread, розширює стандартний клас threading.Thread і додає функціональність для безпечного завершення роботи потоку. Основні компоненти:

1. Конструктор (__init__):

- Ініціалізує потік, встановлюючи параметри:
  - daemon = True: потік буде завершений при завершенні основної програми.
  - target: функція, яку потрібно виконати в потоці.

- stop_ev: подія, що використовується для зупинки потоку.

2. Метод stop:

Встановлює подію stop_ev, що сигналізує потоку про необхідність завершення роботи.

3. Метод run:

Цикл, який виконує функцію target() до того, як подія stop_ev буде встановлена.

- SafeThread дозволяє легко створювати та керувати потоками, надаючи додаткову можливість безпечного завершення роботи потоку через подію stop_ev.
- Потік виконує свою задачу в циклі до отримання сигналу про зупинку. Це дозволяє мати контроль над виконанням і забезпечує правильне завершення роботи потоку.

Цей клас зручний для довготривалих або нескінченних операцій у фонових потоках, де важливо правильно зупиняти потік.

Модуль clKalman, реалізує 2D фільтр Калмана для прогнозування та корекції руху об'єкта. Основні частини цього модуля:

1. Конструктор (__init__):

- Ініціалізує фільтр Калмана з параметрами:
  - kalman: об'єкт cv2.KalmanFilter, який реалізує фільтр Калмана.
  - measurementMatrix: матриця вимірювань, яка визначає, як вимірювання зіставляються зі станом системи.
  - transitionMatrix: матриця переходу, яка визначає, як стан змінюється з часом.
  - processNoiseCov: коваріаційна матриця процесу, яка визначає невизначеність в процесі системи.
- Змінні стану:
  - last_measurement, current_measurement: вимірювання на попередньому та поточному етапах.

- last_prediction, current_prediction: прогнози стану на попередньому та поточному етапах.
- x_i, y_i: зсуви для корекції координат.

2. Метод predictAndUpdate(x, y, correct=True):

- Прогнозує нове місцезнаходження об'єкта за допомогою фільтра Калмана.
- Оновлює поточне вимірювання, виправляє прогноз (якщо параметр correct=True), та повертає попередній та поточний прогнози.
- Параметри:
  - x, y: нові координати об'єкта, які використовуються для корекції стану.
  - correct: якщо True, фільтр Калмана коригує прогноз на основі нових вимірювань.

3. Метод getStateVariables():

Повертає останні вимірювання та прогнози:

- last_measurement, current_measurement: попереднє та поточне вимірювання.
- last_prediction, current_prediction: попереднє та поточне передбачення.

4. Метод init(x, y):

Ініціалізує початкові координати об'єкта, встановлюючи значення для x_i та y_i.

Фільтр Калмана використовується для передбачення положення об'єкта (наприклад, у відео трекінгу) на основі попередніх даних та нових вимірювань. Кожного разу, коли надходить нове вимірювання, фільтр коригує поточний стан системи для досягнення більш точного прогнозу, враховуючи помилки вимірювань та шум у процесі.

МодульTello_Keyboard, реалізує просту програму для керування дроном Tello, використовуючи модуль TelloConnect для взаємодії з дроном. Програма дозволяє управляти дроном через командний рядок, відправляючи різні команди за допомогою клавіш. Основні компоненти і функціональність:

1. Імпорти:

- TelloConnect: імпортується з модуля telloconnect, цей клас відповідає за підключення до дрону Tello та виконання команд через UDP.
- signal: використовується для обробки сигналів завершення програми, наприклад, при натисканні Ctrl+C або отриманні сигналу завершення процесу.

2. Обробник сигналів:

Визначений обробник сигналів для перехоплення сигналів завершення роботи програми (SIGTERM та SIGINT), який викидає виняток, що дозволяє коректно завершити роботу програми.

3. Основна логіка програми:

Програма починається з підключення до дрону через TelloConnect. Використовується метод wait_till_connected(), щоб дочекатися підключення до дрону, після чого запускається зв'язок за допомогою start_communication().

У циклі програма чекає на введення команд від користувача через командний рядок (input):

t: команда для зльоту (відправляє команду 'takeoff').

l: команда для посадки (відправляє команду 'land').

w: підйом вгору (відправляє команду 'up 20').

s: спуск вниз (відправляє команду 'down 20').

a: обертання по годинниковій стрілці (відправляє команду 'cw 20').

d: обертання проти годинникової стрілки (відправляє команду 'ccw 20').

q: вихід з програми, що зупиняє комунікацію та завершує цикл.

4. Обробка виключень:

Якщо під час роботи виникає помилка або якщо користувач натискає Ctrl+C або програма отримує сигнал завершення, комунікація з дроном зупиняється за допомогою stop_communication().

Цей модуль забезпечує базову можливість керувати дроном Tello через командний рядок, дозволяючи відправляти команди для зльоту, посадки, руху та обертання. Він використовує модуль TelloConnect для обробки команд і зв'язку з дроном. Система також підтримує коректне завершення роботи через перехоплення сигналів завершення.

Основний модуль Tello_Object_Tracking, реалізує систему для відстеження об'єктів за допомогою дрону Tello з використанням бібліотеки OpenCV та моделі Yolo для детекції об'єктів. Основні частини і функціональність коду:

Опис основних компонентів:

1. Імпорти:

- TelloConnect - модуль для зв'язку з дроном Tello та обробки відео.
- FollowObject - клас, що реалізує алгоритм відстеження об'єкта за допомогою детекції на зображенні.
- cv2 та numpy - бібліотеки для обробки зображень.
- argparse - бібліотека для обробки аргументів командного рядка.
- signal - для обробки сигналів (наприклад, для коректного завершення програми).

2. Аргументи командного рядка:

Користувач може передати кілька параметрів:

- -model_configuration та -model_weights: шляхи до конфігураційного файлу та ваг моделі Yolo.
- -obj: об'єкт, який потрібно відстежувати (за замовчуванням "Lena").
- -debug: увімкнення/вимкнення режиму відладки.
- -video: використання відеофайлу замість дрону.
- -vsize: розмір відео.

- -th, -tv, -td, -tr: параметри для налаштування, чи потрібно відстежувати горизонтальне, вертикальне положення, дистанцію та обертання об'єкта.

3. Ініціалізація класів:

- TelloConnect: об'єкт для підключення до дрону Tello та отримання відео.
- FollowObject: об'єкт для відстеження об'єкта на зображенні за допомогою Yolo.

4. Налаштування відео та відстеження:

- Налаштовується розмір відео, зберігання відеофайлів.
- Встановлюється обробник сигналів для коректного завершення програми (наприклад, при натисканні Ctrl+C або SIGTERM).

5. Основний цикл:

- В циклі програма чекає кадр від камери дрону.
- Визначається, чи потрібно робити відеозапис або чи потрібно змінити команди (наприклад, підняти дрон, приземлити, повертати і т.д.).
- Виводиться кадр з результатами детекції на екран (через cv2.imshow).
- Команди на керування дроном (зліт, посадка, рух вгору/вниз, обертання) виконуються при натисканні відповідних клавіш.

6. Клавіші для керування:

t: зліт дрону.

l: посадка дрону.

w: підйом вгору.

s: спуск вниз.

a: обертання по годинниковій стрілці.

d: обертання проти годинникової стрілки.

v: почати/зупинити відеозапис.

q: вихід з програми.

Після запуску програма підключається до дрону Tello або використовує відеофайл, якщо передано параметр -video.

Ініціалізується об'єкт FollowObject, який використовує модель Yolo для виявлення та відстеження об'єкта.

В основному циклі програма отримує кадри з камери дрону, обробляє їх через Yolo для детекції об'єктів і відображає результат на екрані.

За допомогою клавіш можна керувати дроном: злітати, приземлятися, обертатися, а також записувати відео.

Якщо режим відладки увімкнено, програма може виводити додаткові повідомлення в консоль для відстеження процесів.

Цей модуль дозволяє відстежувати об'єкти за допомогою дрону Tello, використовуючи модель Yolo для виявлення та аналізу об'єктів на відео. Програма дозволяє знімати відео, а також контролювати рухи дрону через клавіші, забезпечуючи інтерактивний інтерфейс для користувача.

Висновок до розділу 3

У результаті проведеного дослідження було розроблено програмний модуль для автоматизованого відстеження об’єктів за допомогою дрону DJI Tello, що використовує сучасні технології комп’ютерного зору та оптимізованих нейронних мереж. Система реалізована із застосуванням моделей YOLOv4-tiny, бібліотеки OpenCV та інструментів Python, що забезпечує ефективність і гнучкість її використання.

Навчена модель була інтегрована в систему управління Tello. Камера дрона аналізувала відеопотік у реальному часі, знаходила об'єкти (обличчя) і визначала їхнє положення щодо центру кадру. На основі цього розраховувалися відхилення, і дрон автоматично коригував свій рух для утримання об'єкта в центрі. Модель забезпечила достатню швидкість і точність для роботи в реальному часі. Це підтвердило можливість використання YOLO Darknet для задач комп’ютерного зору на компактних

дронах. Усі скрипти та конфігураційні файли були зібрані в одному проєкті, що дозволяє легко відтворити систему на інших пристроях.

Основними перевагами розробленого підходу є:

1. Швидкість і точність виявлення об’єктів завдяки використанню нейронних мереж YOLOv4-tiny, оптимізованих для роботи на апаратних платформах із обмеженими ресурсами.

2. Ефективне керування рухом дрону із використанням PID-регуляторів, що забезпечують стабільність та точність навігації у тривимірному просторі.

3. Можливість навчання моделі на власному наборі даних, адаптованих до специфіки задачі, що підвищує якість детекції об’єктів.

4. Інтеграція відеопотоку та інтерфейсу керування для роботи у режимі реального часу, що дозволяє виконувати завдання відстеження, розпізнавання об’єктів, жестів або аналізу оточення.

Розроблена система демонструє універсальність у вирішенні завдань автоматизації, моніторингу та аналізу даних у реальних умовах. Вона може бути застосована у різних галузях, включаючи навчання, наукові дослідження, промислову автоматизацію, а також інтерактивні проєкти.

Таким чином, отримані результати підтверджують доцільність застосування запропонованого підходу для розробки систем автоматизованого відстеження об’єктів з використанням дронів, що відкриває перспективи для подальшого вдосконалення та адаптації системи до нових умов і завдань.

ВИСНОВКИ

У ході виконання роботи було проведено комплексне дослідження, спрямоване на розробку та впровадження програмного модуля для розпізнавання образів із застосуванням нейронних мереж, зокрема згорткових нейронних мереж (CNN).

У першому розділі проведено аналіз методів і підходів до розпізнавання образів. Було визначено ключові проблеми в цій галузі, зокрема необхідність високої точності та ефективності розпізнавання при обмежених обчислювальних ресурсах. Розглянуто основи побудови нейронних мереж, їх елементи та особливості згорткових нейронних мереж, які є найбільш ефективними для задач розпізнавання зображень.

Другий розділ був присвячений вибору відповідної моделі нейронної мережі та її навчання. Обґрунтовано вибір архітектури CNN, розроблено підхід до навчання моделі, що забезпечує високу точність і адаптивність до різних наборів даних.

У третьому розділі здійснено розробку програмного модуля для реалізації системи розпізнавання образів. Визначено оптимальне середовище розробки, перелік технологій, а також запропоновано методи інтеграції алгоритмів розпізнавання в програмний комплекс. Особливу увагу приділено використанню дрона DJI Tello для відслідковування об’єктів, що розширює функціонал системи. Тестування показало, що розроблений модуль відповідає критеріям точності, продуктивності та зручності використання.

У цій роботі також запропоновано метод реалізації системи виявлення об’єктів і супроводу цілей на основі операційної системи робота (ROS) із застосуванням дрона Tello. Система забезпечує ефективне виявлення об’єктів і відстеження цілей у реальному часі. Як модель виявлення використано скорочену архітектуру YOLOv4, що дозволяє зменшити кількість параметрів і обчислень, забезпечуючи високу швидкість виконання та точність. Для

відстеження об’єктів застосовано OpenCV і фільтр Калмана, що гарантує стабільність і точність в реальних умовах.

Крім того, було впроваджено модуль PID для коригування орієнтації дрона, який дозволяє стабільно відстежувати цільові об’єкти шляхом розрахунку похибок та налаштування керуючих сигналів на основі їх зміни. Проведені експерименти підтвердили доцільність і ефективність розробленої системи для використання в реальних умовах.

Результати роботи демонструють ефективність застосування нейронних мереж у розпізнаванні образів, інтеграції алгоритмів виявлення і відстеження у програмні модулі, а також можливість реалізації комплексних систем автоматизації для різноманітних прикладних задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. X. Zhang, X. Zhou, M. Lin and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 684-685
2. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788
3. What is a neural network? URL: https://www.ibm.com/topics/neural-networks
4. What are the key components of a neural network and what is their role? URL: http://surl.li/voactp
5. Deep Learning Dictionary - Lightweight Crash Course. URL: https://deeplizard.com/lesson/ddd1riadlz
6. Convolutional Neural Networks: A Comprehensive Guide. URL: https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175
7. Копча-Горячкіна Г. Е. Теорія розпізнавання образів. Частина І: Навчально-методичний посібник для студентів факультету інформаційних технологій напрямів «Комп'ютерні науки» та «Програмна інженерія». – Ужгород: Видавництво ДВНЗ «Ужгородського національного університету», 2016 р.
8. Основні поняття теорії розпізнавання образів. URL: http://surl.li/yvbugn
9. Відстань між об'єктами (кластерами) і міри близькості груп об'єктів. URL:https://stud.com.ua/93356/statistika/vidstan_obyektami_klasterami_miri_blizkosti_grup_obyektiv
10. Довбиш А. С. Основи теорії розпізнавання образів – Суми : Сумський держ. університет, 2015. Ч.1. С. 26-28.

11. Що таке багатошаровий персептрон (Multilayer Perceptron, MLP) у машинному навчанні? URL: https://thetransmitted.com/adlucem/shho-take-mlp-u-mashynnomu-navchanni/
12. Quoc V. Le. A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks. – 1600 Amphitheatre Pkwy, Mountain View, CA 94043, 2015.
13. Evolution of CNN Architectures: LeNet, AlexNet, ZFNet, GoogleNet, VGG and ResNet. URL: https://iq.opengenus.org/evolution-of-cnn-architectures/#google_vignette
14. YOLO: Real-Time Object Detection. URL: https://pjreddie.com/darknet/yolov2/
15. SSD object detection: Single Shot MultiBox Detector for real-time processing. URL: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
16. LabelImg for Image Annotation. URL: https://viso.ai/computer-vision/labelimg-for-image-annotation/
17. Object detection with darknet (YOLOv3). URL: https://thedatafrog.com/en/articles/object-detection-darknet/
18. Інтегроване середовище розробки. URL: https://w3schoolsua.github.io/hyperskill/ide.html#gsc.tab=0
19. Visual Studio Code. URL: https://uk.wikipedia.org/wiki/Visual_Studio_Code
20. Florian Rappl, "Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers", Packt Publishing, 2019.
21. Pradeepta Mishra, "Hands-On Deep Learning Algorithms with Python", Packt Publishing, 2019.
22. Python. URL: https://umity.in.ua/concept/?id=849
23. Опис Python. URL: https://uk.wikipedia.org/wiki/Python
24. YOLOv4 Baseline Training. URL: https://github.com/AlexeyAB/Darknet
25. Zhang, P.; Zhong, Y.; Li, X. SlimYolov3: Narrower, Faster, and Better for Real-Time UAV Applications. In Proceedings of the IEEE/CVF International

Conference on Computer Vision Workshop (ICCVW 2019), Seoul, Republic of Korea, 27–28 October 2019; pp. 37–45.

26. C. -Y. Wang, H. -Y. Mark Liao, Y. -H. Wu, P. -Y. Chen, J. -W. Hsieh and I. -H. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 1571-1580
27. Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, John Guttag. Proceedings of Machine Learning and Systems. *What is the State of Neural Network Pruning?*, March 2020, pp. 129–142.
28. Дрон DJI Tello. URL: https://airunit.com.ua/dron-dji-tello/?srsltid=AfmBOoq7uqwzMFOegDwQcHqyvOGvNNjpQJZEmlop8wTg4jETK8qlZqKU
29. Wang, Q.; Zhang, L.; Bertinetto, L.; Hu, W.; Torr, P. Fast Online Object Tracking and Segmentation: A Unifying Approach. In Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 15–20 June 2019; pp. 132–133.
30. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese Neural Networks for One-Shot Image Recognition. In Proceedings of the International Conference on Machine Learning Deep Learning Workshop (ICML 2015), Lille, France, 6–11 July 2015; pp. 1–8.
31. Bromley, J.; LeCun, Y. Signature Verification Using a “Siamese” Time Delay Neural Network. In Proceedings of the Advances in the 6th Neural Information Processing Systems, Denver, CO, USA, November 2015; pp. 737–744.
32. Вчимося керувати FPV-дронами. URL: https://www.rozhkov.me/how-to-fpv-drones/
33. Robot Operating System (ROS). URL: https://www.ros.org/
34. Yolo v4, v3 and v2 for Windows and Linux. URL: https://github.com/AlexeyAB/darknet?tab=readme-ov-file#how-to-train-to-detect-your-custom-objects
35. Репозиторій DarkNet. URL: https://github.com/pjreddie/darknet

ДОДАТКИ

ДОДАТОК А
TELLO_KEYBOARD

```
from utils.telloconnect import TelloConnect
import signal


if __name__=="__main__":

    # обробник сигналів
    def signal_handler(sig, frame):
        raise Exception

    # захоплення сигналів
    signal.signal(signal.SIGTERM, signal_handler)
    signal.signal(signal.SIGINT, signal_handler)

    tello = TelloConnect(DEBUG=False)

    # зачекайте до підключення, а потім продовжуйте
    tello.wait_till_connected()
    tello.start_communication()


    while True:

        try:
            k = input()

            # вихід
            if k == 'q':
                tello.stop_communication()
```

```
            break

        if k == 't':
            tello.send_cmd('takeoff')

        if k == 'l':
            tello.send_cmd('land')

        if k == 'w':
            tello.send_cmd('up 20')

        if k == 's':
            tello.send_cmd('down 20')

        if k == 'a':
            tello.send_cmd('cw 20')

        if k == 'd':
            tello.send_cmd('ccw 20')

    except Exception:
        tello.stop_communication()
        break
```

ДОДАТОК Б

TELLO_OBJECT_TRACKING

```
from utils.telloconnect import TelloConnect
from utils.followobject import FollowObject
import signal
import cv2
import argparse
import numpy as np
```

```
if __name__=="__main__":

    # вхідні аргументи
    parser = argparse.ArgumentParser(description='Tello Object tracker. keys: t-takeoff, l-land, v-video, q-quit w-up, s-down, a-ccw rotate, d-cw rotate\n')
    parser.add_argument('-model_configuration', type=str, help='Yolo config file, see data folder', default='/home/lena/Documents/Drone-Object-Detection/tello_object_tracking/data/yolov4-tiny-tello-3l.cfg')
    parser.add_argument('-model_weights', type=str, help='Yolo weights file, see data folder', default='/home/lena/Documents/Drone-Object-Detection/tello_object_tracking/data/yolov4-tiny-tello-3l_best.weights')
    parser.add_argument('-obj', type=str, help='Type of object to track. [Lena], default = Face', default='Lena')
    parser.add_argument('-dconf', type=float, help='Detection confidence, default = 0.7', default=0.7)
    parser.add_argument('-debug', type=bool, help='Enable debug, lists messages in console', default=False)
    parser.add_argument('-video', type=str, help='Use as inputs a video file, no tello needed, debug must be True', default="")
    parser.add_argument('-vsize', type=list, help='Video size received from tello', default=(640,480))
    parser.add_argument('-th', type=bool, help='Horizontal tracking', default=False)
    parser.add_argument('-tv', type=bool, help='Vertical tracking', default=True)
    parser.add_argument('-td', type=bool, help='Distance tracking', default=True)
    parser.add_argument('-tr', type=bool, help='Rotation tracking', default=True)


    args = parser.parse_args()
    classes = []
    classes_path="/home/lena/Documents/Drone-Object-Detection/tello_object_tracking/data/tello.names"

    with open(classes_path, "r") as file_object:
        for class_name in file_object.readlines():
```

```
        class_name = class_name.strip()
        classes.append(class_name)



print(classes)

#швидкість обробки
pspeed = 1

#розмір зображення
imgsize=args.vsize

#збереження відео
writevideo = False

#обробник сигналів
def signal_handler(sig, frame):
    raise Exception

#захоплення сигналів
signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

if args.debug and args.video is not None:
    tello = TelloConnect(DEBUG=True, VIDEO_SOURCE=args.video)
else:
    tello = TelloConnect(DEBUG=False)
tello.set_image_size(imgsize)

videow = cv2.VideoWriter('out.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 30, (imgsize))

if tello.debug == True: pspeed = 30

#періодично запитувати статистику
```

```
tello.add_periodic_event('wifi?',40,'Wifi')

# зачекайте до підключення, а потім продовжуйте
tello.wait_till_connected()
tello.start_communication()
tello.start_video()

fobj = FollowObject(tello, model_configuration = args.model_configuration, model_weights = args.model_weights, CLASSES = classes, CONFIDENCE=args.dconf, DEBUG=False, DETECT=args.obj)
fobj.set_tracking( HORIZONTAL=args.th, VERTICAL=args.tv,DISTANCE=args.td, ROTATION=args.tr)

while True:

    try:
        img = tello.get_frame()

        # дочекатися дійсного кадру
        if img is None: continue

        imghud = img.copy()

        fobj.set_image_to_process(img)

        k = cv2.waitKey(pspeed)

    except Exception:
        tello.stop_video()
        tello.stop_communication()
        break

    fobj.draw_detections(imghud, ANONIMUS=False)
    cv2.imshow("TelloCamera",imghud)
```

```
    # запис відео
    if k ==ord('v'):
        if writevideo == False : writevideo = True
        else: writevideo = False

    if writevideo == True:
        videow.write(img)

    # вихід
    if k == ord('q'):
        tello.stop_communication()
        break

    if k == ord('t'):
        tello.send_cmd('takeoff')

    if k == ord('l'):
        tello.send_cmd('land')

    if k == ord('w'):
        tello.send_cmd('up 20')

    if k == ord('s'):
        tello.send_cmd('down 20')

    if k == ord('a'):
        tello.send_cmd('cw 20')

    if k == ord('d'):
        tello.send_cmd('ccw 20')

cv2.destroyAllWindows()
```

ДОДАТОК В

DNNOBJECTDETECT

```
import cv2
import numpy as np

class DnnObjectDetect():

    def __init__(self, model_configuration, model_weights, CLASSES, CONFIDENCE=0.8, object_type='general'):

        net = cv2.dnn.readNetFromDarknet(model_configuration, model_weights)

        self.network = net
        self.confidence_threshold = CONFIDENCE
        self.type = object_type
        self.classes = CLASSES


    def preprocess_image(self, img, size=(608, 608)):
        """Попередня обробка зображення для мережі."""
        return cv2.dnn.blobFromImage(cv2.resize(img, size), 1 / 255, [0, 0, 0], swapRB=True, crop=False)

    def detect(self,img, size=(416, 416)):

        height, width = img.shape[:2]
        blob = self.preprocess_image(img, size)
        self.network.setInput(blob)

        # Отримайте назви вихідного шару та виконайте перехід вперед
        layer_names = self.network.getLayerNames()
        output_layer_names = [layer_names[i - 1] for i in self.network.getUnconnectedOutLayers()]
        outputs = self.network.forward(output_layer_names)
```

```
boxes, confidences, class_ids = [], [], []

for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        if confidence > self.confidence_threshold:
            box = detection[:4] * np.array([width, height, width, height])
            (center_x, center_y, w, h) = box.astype("int")
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)
    # Немаксимальне придушення для видалення блоків, що перекриваються
    indices = cv2.dnn.NMSBoxes(boxes, confidences, self.confidence_threshold, .4)
    detections = []
    true_positive_coords = []

    for i in indices:
        i = i  # Отримати індекс із масиву
        box = boxes[i]
        (x, y, w, h) = box[0], box[1], box[2], box[3]
        detections.append({
        "box": (x, y, w, h),
        "class_id": class_ids[i],
        "confidence": confidences[i]
        })

        # Обчислити справжні позитивні результати на основі типу об’єкта
        if self.type == 'Lena' and class_ids[i] == 0:  # Приклад перевірки для конкретного типу об'єкта
```

```
                area_frame = width * height
                area_det = w * h
                area_ratio = int((area_det / area_frame) * 100)

                # Central point
                mid_x = x + w // 2
                mid_y = y + h // 2
                tp = [mid_x, mid_y, area_ratio]
                true_positive_coords.append(tp)
                print(true_positive_coords)
                print(detections)
        return true_positive_coords, detections

    def draw_detections(self,det,img,COLOR=[0,255,0]):

        for detection in det:
            (x, y, w, h) = detection['box']
            class_id = detection['class_id']
            confidence = detection['confidence']

            label = f"{self.classes[class_id]}: {confidence:.2f}"
            cv2.rectangle(img, (x, y), (x + w, y + h), COLOR, 2)
            cv2.putText(img, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, .5, COLOR, 2)
        return img

    def detect_and_draw(self,img):

        tp, det = self.detect(img)
        self.draw_detections(det,img)
        return img

if __name__ == "__main__":

    # Ініціалізація моделі
```

```
    classesFile = "/home/lena/Documents/Drone-Object-
Detection/tello_object_tracking/data/tello.names"
    classNames = []

    with open(classesFile, "r") as file_object:
        for class_name in file_object.readlines():
            class_name = class_name.strip()
            classNames.append(class_name)
    print(classNames)
    # Створення об'єкта детектора
    detector = DnnObjectDetect('/home/lena/Documents/Drone-Object-
Detection/tello_object_tracking/data/yolov4-tiny-tello-3l.cfg', '/home/lena/Documents/Drone-
Object-Detection/tello_object_tracking/data/yolov4-tiny-tello-3l_best.weights',
classNames,CONFIDENCE=0.7, object_type='Lena')


    image = cv2.imread("/home/lena/Documents/Drone-Object-
Detection/my_project_tello/1.jpg")  # Load an image
    # detections = detector.detect(image)
    output_image = detector.detect_and_draw(image)

    cv2.imshow("Detections", output_image)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

ДОДАТОК Г

FOLLOWOBJECT

```
import cv2
import threading
import numpy as np
from . import safethread
from . import kalman
from . import dnnobjectdetect
```

```
class FollowObject():

    def __init__(self, tello, model_configuration, model_weights, CLASSES ,CONFIDENCE=0.8, DETECT='Lena', DEBUG=False) -> None:

        # детектор обличчя
        self.classes = CLASSES

        if model_configuration !='' and model_weights!='':
                self.dnnfacedetect = dnnobjectdetect.DnnObjectDetect(model_configuration, model_weights, CLASSES= CLASSES,CONFIDENCE=CONFIDENCE, object_type=DETECT)
        else:
                self.dnnfacedetect = dnnobjectdetect.DnnObjectDetect(CONFIDENCE=CONFIDENCE,DETECT=DETECT)

        self.tello = tello

        # Оцінювачі Калмана
        self.kf = kalman.clKalman()
        self.kfarea= kalman.clKalman()

        # тікер для часової бази
        self.ticker = threading.Event()

        # ініціали
        self.track = False

        self.img = None
        self.det = None
        self.tp = None

        # параметри відстеження
        self.use_vertical_tracking = True
        self.use_rotation_tracking = True
```

```
        self.use_horizontal_tracking = True
        self.use_distance_tracking = True

        # відстань між об'єктом і дроном
        self.dist_setpoint = 100
        self.area_setpoint = 13

        self.cx = 0
        self.cy = 0

        # використовуйте цей параметр для друку даних налагодження
        self.debug = DEBUG

        # частота обробки (для економії процесорного часу)
        self.cycle_counter = 1
        self.cycle_activation = 10

        # Масштабні коефіцієнти оцінки Калмана
        self.kvscale = 6
        self.khscale = 4
        self.distscale = 3

        self.wt = safethread.SafeThread(target=self.__worker).start()

    def set_default_distance(self,DISTANCE=100):

        self.dist_setpoint = DISTANCE

    def set_default_area(self,DISTANCE=13):

        self.area_setpoint = DISTANCE


    def set_tracking(self, HORIZONTAL=False, VERTICAL=True, DISTANCE=True,
ROTATION=True):
```

```
        self.use_vertical_tracking = VERTICAL
        self.use_horizontal_tracking = HORIZONTAL
        self.use_distance_tracking = DISTANCE
        self.use_rotation_tracking = ROTATION

    def set_image_to_process(self, img):

        self.img = img

    def set_detection_periodicity(self,PERIOD=10):

        self.cycle_activation = PERIOD

    def safety_limiter(self,leftright,fwdbackw,updown,yaw, SAFETYLIMIT=30):

        val = np.array([leftright,fwdbackw,updown,yaw])

        # test uppler lover levels
        val[val>=SAFETYLIMIT] = SAFETYLIMIT
        val[val<=-SAFETYLIMIT] = -SAFETYLIMIT

        return val[0],val[1],val[2],val[3]


    def __worker(self):

        # база часу
        self.ticker.wait(0.005)

        # обробляти зображення, команда tello
        if self.img is not None and self.cycle_counter % self.cycle_activation == 0:

            dist = 0
```

```
vy = 0
vx,rx = 0,0
# робота над локальною копією
img = self.img.copy()

# виявити обличчя
tp,det = self.dnnfacedetect.detect(img)

if  len(det) > 0:
    self.det = det
    self.tp = tp

    # початкові оцінки
    if self.track == False:
        h,w = img.shape[:2]
        self.cx = w//2
        self.cy = h//2
        self.kf.init(self.cx,self.cy)

        # обчислити init 'area', ігнорувати розмір x
        self.kfarea.init(1,tp[1])
        self.track = True

    # корекції процесу, обчислення дельти між двома об'єктами
    _,cp = self.kf.predictAndUpdate(self.cx,self.cy,True)

    # обчислити дельту по 2 осі
    mvx = -int((cp[0]-tp[0])//self.kvscale)
    mvy = int((cp[1]-tp[1])//self.khscale)

    if self.use_distance_tracking:
        # використовуйте значення y для визначення відстані до об’єкта
        obj_y = tp[2]

        _, ocp = self.kfarea.predictAndUpdate(1, obj_y, True)
```

```
            dist = int((ocp[1]-self.dist_setpoint)//self.distscale)

        # Заповніть змінні, які потрібно надіслати в команді tello
        # не поєднуйте горизонталь і обертання
        if self.use_horizontal_tracking:
            rx = 0
            vx = mvx
        if self.use_rotation_tracking:
            vx = 0
            rx = mvx

        if self.use_vertical_tracking:
            vy = mvy

        # обмеження сигналів, якщо це так, може врятувати ваш Tello
        vx,dist,vy,rx = self.safety_limiter(vx,dist,vy,rx,SAFETYLIMIT=40)

        cmd = "rc {leftright} {fwdbackw} {updown} {yaw}".format(leftright=vx,fwdbackw=-
dist,updown=vy,yaw=rx)
        self.tello.send_cmd(cmd)

        if self.debug:
            print (cmd, str(self.cycle_counter))

    else:
        # не виявлено, зберігайте позицію
                                cmd = "rc {leftright} {fwdbackw} {updown}
{yaw}".format(leftright=0,fwdbackw=0,updown=0,yaw=0)
        self.tello.send_cmd(cmd)
        self.det = None

    self.cycle_counter +=1
```

```
def draw_detections(self,img, HUD=True, ANONIMUS=False):

    sizef = 0.5
    typef = cv2.FONT_HERSHEY_SIMPLEX
    color = [0,255,255]
    sizeb = 2
    if img is not None:

      h,w = img.shape[:2]

      if HUD and self.tello.state_value is not None and len(self.tello.state_value)>0:
        hud =self.tello.state_value
       cv2.putText(img,str('Battery') + ": " + str(hud[21]),(w//2-100,20),typef,sizef,color,sizeb)
        cv2.putText(img,str('Height') + ": " + str(hud[19]),(30,20),typef,sizef,color,sizeb)
        cv2.putText(img,str('Tof') + ": " + str(hud[17]),(30,40),typef,sizef,color,sizeb)
        cv2.putText(img,str('Temp') + ": " + str(hud[15]),(w//2+100,20),typef,sizef,color,sizeb)
        cv2.putText(img,str('Baro') + ": " + str(hud[23]),(w//2+100,40),typef,sizef,color,sizeb)
       cv2.putText(img,str('Acceleration') + ": " + 'agx'+ " "+ str(hud[-6]) + ' agy'+ " "+ str(hud[-
4]) + ' agz'+ " "+ str(hud[-2]),(30,h-30),typef,sizef,color,sizeb)

        for ev in self.tello.eventlist:
          ret = ev['cmd']
          if ret is not None and ret == 'wifi?':
                              cv2.putText(img,str(ev['info'])  +  ":  "  +  str(ev['val']),(w//2-
100,40),typef,sizef,color,sizeb)

      # handle detection visualization
      if self.det is not None:
        for val in self.det:
          (x, y, w, h) = val['box']
          class_id = val['class_id']
          confidence = val['confidence']

          label = f"{self.classes[class_id]}: {confidence:.2f}"
          if ANONIMUS:
```

```
                cv2.rectangle(img,(x, y), (x + w, y + h),[0,255,0],-1)
            else:
                cv2.rectangle(img,(x, y), (x + w, y + h),[0,255,0],2)
                cv2.putText(img, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, .5, [0,255,0],
2)
        cv2.circle(img,(self.tp[0],self.tp[1]),3,[0,0,255],-1)
        cv2.circle(img,(int(w/2),int(h/2)),4,[0,255,0],1)
        cv2.line(img,(int(w/2),int(h/2)),(self.tp[0],self.tp[1]),[0,255,0],2)
```

ДОДАТОК Д

KALMAN

```
import cv2
import numpy as np
class clKalman():
  def __init__(self):

    # 2d Kalman
    self.kalman = cv2.KalmanFilter(4, 2)

    self.kalman.measurementMatrix = np.array([[1, 0, 0, 0], [0, 1, 0, 0]], np.float32)
      self.kalman.transitionMatrix = np.array([[1, 0, 1, 0],[0, 1, 0, 1],[0, 0, 1, 0], [0, 0, 0,
1]],np.float32)

      self.kalman.processNoiseCov = np.array([[1, 0, 0 ,0],[0, 1, 0, 0],[0, 0, 1, 0],[0, 0, 0,
1]],np.float32) * 0.01

    # змінні стану
    self.last_measurement = np.array((2, 1), np.float32)
    self.current_measurement = np.array((2, 1), np.float32)
    self.last_prediction = np.zeros((2, 1), np.float32)
    self.current_prediction = np.zeros((2, 1), np.float32)

    # поправка => взяти з вимірювання, додати до поправки
```

```
        self.xi = 0
        self.yi = 0

    def predictAndUpdate(self,x,y,correct=True):

        self.last_prediction = self.current_prediction
        self.last_measurement = self.current_measurement
        self.current_measurement = np.array([[np.float32(x-self.xi)], [np.float32(y-self.yi)]])

        # виправити і передбачити, якщо це так
        if correct:
            self.kalman.correct(self.current_measurement)
        self.current_prediction = self.kalman.predict()

                                self.current_prediction = [self.current_prediction[0]+self.xi,
self.current_prediction[1]+self.yi]

        return self.last_prediction, self.current_prediction

    def getStateVariables(self):

                return self.last_measurement, self.current_measurement, self.last_prediction,
self.current_prediction

    def init(self,x,y):

        self.xi = x
        self.yi = y
```

ДОДАТОК Е

SAFETHREAD

```
import threading
```

```
class SafeThread(threading.Thread):

    def __init__(self, target):
        threading.Thread.__init__(self)
        self.daemon = True
        self.target = target
        self.stop_ev = threading.Event()

    def stop(self):
        self.stop_ev.set()

    def run(self):
        while not self.stop_ev.is_set():
            self.target()
```

ДОДАТОК Ж

TELLOCONNECT

```
import threading
from . import safethread

class TelloConnect:
    import socket
    import cv2
    from queue import Queue

    def __init__(self,TELLOIP='192.168.10.1', UDPPORT=8889,
VIDEO_SOURCE="udp://@0.0.0.0:11111",UDPSTATEPORT=8890, DEBUG=False) ->
None:

        self.localaddr = ('',UDPPORT)
        self.telloaddr = (TELLOIP,UDPPORT)
        self.video_source = VIDEO_SOURCE
        self.stateaddr = ('',UDPSTATEPORT)
```

```
self.debug = DEBUG

# запис значення satate
self.state_value = []

# розмір зображення
self.image_size = (640,480)

# повернути повідомлення з UDP
self.udp_cmd_ret = ''

# зберігати одне зображення
self.q = self.Queue()
self.q.maxsize = 1


# зберігати одне зображення
self.frame = None

# лічильник планувальника
self.count = 1

# команда отримана подія
self.cmd_recv_ev = threading.Event()

# подія таймера
self.timer_ev = threading.Event()

# обробник періодичних команд
self.eventlist = list()

# додати першу періодичну команду для надсилання, підтримувати активність
self.eventlist.append({'cmd':'command','period':100,'info':''})
```

```
    # # створити UDP-пакет для команд
    self.sock_cmd = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_DGRAM)
    self.sock_cmd.bind(self.localaddr)

    # стан Tello
    self.sock_state = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_DGRAM)
    self.sock_state.bind(self.stateaddr)

    # почати отримувати потік
    self.receiverThread = safethread.SafeThread(target=self.__receive)
    self.receiverThread.daemon = True

    # надсилати періодичні команди
    self.eventThread = safethread.SafeThread(target=self.__periodic_cmd)

    # розпочати відеопотік
    self.videoThread = safethread.SafeThread(target=self.__video)

    # розпочати відеопотік
    self.stateThread = safethread.SafeThread(target=self.__state_receive)

def set_image_size(self, image_size=(960,720)):

    self.image_size = image_size

def get_frame(self):

    # повернути self.frame
    return self.q.get()

def __video(self):

    # обробка потоку
    self.video = self.cv2.VideoCapture(self.video_source)
    while True:
```

```
        try:
            # кадр із потоку
            ret, frame = self.video.read()

            if ret:
                frame = self.cv2.resize(frame,self.image_size)
                self.frame = frame
                self.q.put(frame)

        except Exception:
            pass

def add_periodic_event(self,cmd,period,info=''):

    self.eventlist.append({'cmd':str(cmd),'period':int(period),'info':str(info), 'val':str("")})

def __periodic_cmd(self):

    try:

        for ev in self.eventlist:
            period = ev['period']

            if self.count % int(period) == 0:

                cmd = ev['cmd']
                info = ev['info']
                ret = self.send_cmd_return(cmd).rstrip()

                ev['val'] = str(ret)

        # базовий час планувальника ~ 100 мс
        self.timer_ev.wait(0.1)

        self.count +=1
```

```
    except Exception:
        pass

def __receive(self):

    try:
        data, _ = self.sock_cmd.recvfrom(2048)
        self.udp_cmd_ret = data.decode(encoding="utf-8")
        self.cmd_recv_ev.set()
    except Exception:
        pass

def __state_receive(self):

    data, _ = self.sock_state.recvfrom(512)
    val = data.decode(encoding="utf-8").rstrip()

    # розділення даних
    self.state_value = val.replace(';',':').split(':')

def stop_communication(self):

    self.receiverThread.stop()
    self.stateThread.stop()
    self.eventThread.stop()

    # також закрийте розетку
    self.sock_cmd.close()

def start_communication(self):

    # почати спілкування / прослухати UDP
    if self.receiverThread.is_alive() != True:
        self.receiverThread.start()
```

```
    if self.eventThread.is_alive() != True:
        self.eventThread.start()

    if self.stateThread.is_alive() != True:
        self.stateThread.start()


def start_video(self):

    self.send_cmd('streamon')
    if self.videoThread.is_alive() != True:
        self.videoThread.start()
        # self.videoThread.join()


def stop_video(self):

    self.send_cmd('streamoff')
    self.videoThread.stop()

def wait_till_connected(self):

    self.receiverThread.start()

    while True:
        try:
            ret = self.send_cmd_return('command')

            # примусово перевести tello в режим «DEBUG».
            if self.debug== True: ret = "OK"

        except Exception:
            exit()
        if str(ret) != 'None':
            break
```

```
def send_cmd_return(self,cmd):

    # надіслати cmd через UDP
    self.udp_cmd_ret = None
    cmd = cmd.encode(encoding="utf-8")
    _ = self.sock_cmd.sendto(cmd, self.telloaddr)

    # зачекайте на відповідь через UDP
    self.cmd_recv_ev.wait(0.3)

    # підготуватися до наступного отриманого повідомлення
    self.cmd_recv_ev.clear()

    return self.udp_cmd_ret

def send_cmd(self,cmd):

    # надіслати cmd через UDP
    cmd = cmd.encode(encoding="utf-8")
    _ = self.sock_cmd.sendto(cmd, self.telloaddr)
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я,_____
_____
_____
_____
_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)