ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження алгоритмів обробки зображення та відео для навігації в безпілотних літальних апаратах»

на здобуття освітнього ступеня магістра

зі спеціальності ____122 Комп'ютерні науки ____

(код, найменування спеціальності)

освітньо-професійної програми _____ Комп'ютерні науки ____

(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Назар Білинський

(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої
освіти гр. КНДМ-61 Назар Білинський

Керівник: Сергій Сєрих
к.т.н., доцент

Рецензент:

Київ 2023

Додаток

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедрою _____
_____ Ім'я, ПРІЗВИЩЕ «____»
_____ 20__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Білинський Назар Володимирович_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження алгоритмів обробки зображення та відео для навігації в безпілотних літальних апаратах

керівник кваліфікаційної роботи _____Сергій Сєрих, к.т.н., доцент_____,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» _жовтня_ 2023 р. № 145

2. Строк подання кваліфікаційної роботи «29» ___ грудня 2023р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, вимоги до нейромережі для використання в безпілотних літальних апаратах, дослідження в області комп'ютерного зору, машинного навчання та існуючих алгоритмів виявлення та відстеження об'єктів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Огляд технологій комп'ютерного зору та обчислювальних систем, збір репрезентативних даних для навчання моделі, дослідження алгоритмів обробки зображень.
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «19» _жовтня_ 2023 р.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на здобуття освітнього ступеня магістра

Направляється здобувач БілинськийН.В. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 122 "Комп'ютерні науки"
(код, найменування спеціальності)
освітньо-професійної програми "Комп'ютерні науки"
(назва)
на тему: «Дослідження алгоритмів обробки зображення та відео для навігації в безпілотних літальних апаратах»

Кваліфікаційна робота і рецензія додаються.

Директор ННІ ІТ _____ Андрій Бондарчук
(підпис) (Ім'я, ПРІЗВИЩЕ)

Висновок керівника кваліфікаційної роботи

Здобувач Білинський Назар Володимирович за період написання кваліфікаційної роботи проявив вміння користуватися науковою та технічною літературою, показав відповідальне ставлення до роботи. За формою і змістом кваліфікаційна робота відповідає чинним вимогам, є самостійною роботою, у якій здобувач показав: знання засад спеціальності; знання щодо конкретного предмету своєї роботи; вміння одержувати інформацію за допомогою сучасних наукових методів; вміння осмислювати одержану інформацію і подавати її в прийнятній для даної галузі знань формі.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Білинського Н.В. на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з комп'ютерних наук.

Керівник кваліфікаційної роботи Сергій Сєрих
(підпис) (Ім'я, ПРІЗВИЩЕ)

"___"_____2023 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Білинський Н.В. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою Комп'ютерних наук

_____ Віктор Вишнівський
(підпис) (Ім'я, ПРІЗВИЩЕ)

ВІДГУК РЕЦЕНЗЕНТА
на здобуття освітнього ступеня магістра

здобувача(ки) вищої освіти *Білинського Назара Володимировича*
(прізвище, ім'я, по батькові)
на тему *«Дослідження алгоритмів обробки зображення та відео для навігації в безпілотних літальних апаратах»*

Актуальність.

Ця магістерська робота заслуговує уваги через зростаючий інтерес до безпілотних технологій. Автор зосереджується на глибокому аналізі фундаментальних аспектів підготовки та тренування моделей, та використанні передових нейронних мереж для обробки відеоданих. Цей підхід є важливим для підвищення ефективності застосування безпілотних літальних апаратів, відкриваючи нові горизонти у їх використанні.

Позитивні сторони.

1. Робота демонструє глибоке розуміння тематики та високий рівень аналітичних навичок.

2. У роботі ретельно розглянуто процес підготовки та навчання моделі з використанням як реальних, так і синтетичних даних, що свідчить про творчий підхід до розв'язання задач.

3. Автор вдало інтегрував технології комп'ютерного зору та машинного навчання для розпізнавання об'єктів.

4. Ефективне використання програмного забезпечення та обчислювальних платформ, підкреслює практичну спрямованість роботи.

Недоліки.

1. Хоча в роботі розглядаються багато аспектів обробки зображень, можна було б розширити дослідження, включивши детальніший аналіз потенційних проблем та обмежень в реальних умовах експлуатації.

2. Більш глибокий огляд суміжних досліджень міг би посилити наукову основу роботи.

Відзначені зауваження не впливають на загальну позитивну оцінку магістерської кваліфікаційної роботи

Висновок: *кваліфікаційна магістерська робота заслуговує оцінку «відмінно», а здобувач (ка) Білинський Н.В. заслуговує присвоєння кваліфікації: магістр з комп'ютерних наук.*

Рецензент: _____

(науковий ступінь, вчене звання) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка завдання	19 - 22 жовтня	
2	Пошук і аналіз літератури	22 - 27 жовтня	
3	Збір та обробка даних	27 жовтня - 06 листопада	
4	Розмітка датасету та навчання моделі	6 - 29 листопада	
5	Аналіз отриманих результатів	29 листопада - 07 грудня	
6	Реалізація алгоритму для виявлення об’єктів та навігації БПЛА	07 - 18 грудня	
7	Оформлення графічного та ілюстративного матеріалу	18 - 29 грудня	

Здобувач(ка) вищої освіти _____ _____
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ _____
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина магістерської роботи на здобуття освітнього ступеня магістра: 83 стор., 36 рис., 21 джерело.

Актуальність теми полягає у використанні комп’ютерного зору для управління БПЛА, що може значно підвищити ефективність навігації, особливо в умовах радіоелектронної боротьби. Такий підхід надає військовим нові інструменти, сприяючи точнішому виявленню та ідентифікації об'єктів.

Наукова новизна даного дослідження полягає в його фокусі на розробці та оптимізації алгоритмів обробки зображення та відео для навігації безпілотних літальних апаратів в умовах коли традиційні методи можуть бути неефективними. Додатковим аспектом є створення комплексних рішень, що спрямовані на вирішення викликів при створенні ефективної моделі.

Мета роботи полягає в створенні рішень, які забезпечать високу точність, та швидкість розпізнавання об’єктів з БПЛА.

Наукове завдання включає огляд технологій комп’ютерного зору та обчислювальних систем, збір репрезентативних даних для навчання моделі, розробку алгоритмів обробки зображень, адаптованих до обмежених обчислювальних можливостей БПЛА.

Об`єкт дослідження – використання нейронних мереж для виявлення об’єктів.

Предмет дослідження – формування набору даних, навчання нейронної мережі, створення та оптимізація алгоритмів для навігації та розпізнавання об’єктів.

Методи дослідження включають поєднання автоматизованої та ручної анотації даних, розробку синтетичних датасетів для навчання моделей, а також написання і впровадження алгоритмів для виявлення об'єктів та корекції напрямку руху БПЛА.

У рамках даної роботи досліджується процес навчання моделей, створення тренувальних наборів даних, вибір оптимальної архітектури та мікропроцесорного модуля, а також алгоритми обробки відео, спрямовані на виявлення об'єктів та навігацію безпілотних літальних апаратів (БПЛА).

Ключові слова: комп’ютерний зір, машинне навчання, формування датасету, БПЛА.

ABSTRACT

Text part of the Master's thesis for obtaining the Master's degree: 83 pages, 36 figures, 21 sources.

The relevance of the topic lies in the use of computer vision for controlling UAVs, which can significantly enhance navigation efficiency, especially in conditions of electronic warfare. This approach provides the military with new tools, contributing to more accurate detection and identification of objects.

The scientific novelty of this research lies in its focus on the development and optimization of image and video processing algorithms for UAV navigation in situations where traditional methods may be ineffective. An additional aspect is the creation of comprehensive solutions aimed at addressing challenges in developing an effective model.

The purpose of the work is to create solutions that will provide high accuracy and speed of object recognition from UAVs.

The scientific task includes a review of computer vision technologies and computing systems, gathering representative data for training the model, developing image processing algorithms adapted to the limited computing capabilities of UAVs.

The object of research is the use of neural networks for object detection.

The subject of the study is the formation of a dataset, training of the neural network, and creation and optimization of algorithms for navigation and object recognition.

Research methods include a combination of automated and manual data annotation, development of synthetic datasets for model training, and writing and implementing algorithms for object detection and UAV trajectory correction.

Within the scope of this work, the process of training models, creation of training datasets, selection of the optimal architecture and microprocessor module, and video processing algorithms aimed at object detection and navigation of unmanned aerial vehicles (UAVs) are investigated.

Keywords: computer vision, machine learning, dataset formation, UAVs.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

APC (armoured personnel carrier) – бронетранспортер

FPS (frames per second) – кадри на секунду

GFLOPs/TFLOPs (floating point operations per second) – кількість операцій з плаваючою комою за секунду

mAP (mean average precision) – середня точність

TOPS (trillion operations per second) – кількість операцій за секунду

YOLO (you only look once) – алгоритм об'єктного виявлення в реальному часі

БПЛА – безпілотний літальний апарат

РЕБ – радіоелектронна боротьба

ВСТУП

У сучасних умовах, освоєння передових технологій у сфері безпілотної авіації стає ключовим, особливо в контексті військових дій, де БПЛА можуть надавати велику перевагу.

Важливою проблемою є підтримка надійного зв'язку та забезпечення ефективної навігації БПЛА, з огляду на зростаюче використання радіоелектронних перешкод та обмеження, що накладаються специфікою установки, дальністю дії антени та компетенцією оператора. Тому, розробка автономних рішень для управління БПЛА стає пріоритетною задачею.

Актуальність теми полягає у використанні комп’ютерного зору для управління БПЛА, що може значно підвищити ефективність навігації, особливо в умовах радіоелектронної боротьби. Такий підхід надає військовим нові інструменти, сприяючи точнішому виявленню та ідентифікації об'єктів.

Наукова новизна даного дослідження полягає в його фокусі на розробці та оптимізації алгоритмів обробки зображення та відео для навігації безпілотних літальних апаратів в умовах коли традиційні методи можуть бути неефективними. Додатковим аспектом є створення комплексних рішень, що спрямовані на вирішення викликів при створенні ефективної моделі.

Використання обчислювальних модулів Nvidia Jetson для обробки відео та зображень, дозволяє ефективно використовувати потужні обчислювальні ресурси для реалізації алгоритмів обробки зображень та відео в реальному часі.

Мета роботи полягає в створенні рішень, які забезпечать високу точність, та швидкість розпізнавання об’єктів з БПЛА.

Наукове завдання включає огляд технологій комп’ютерного зору та обчислювальних систем, збір репрезентативних даних для навчання моделі, розробку алгоритмів обробки зображень, адаптованих до обмежених обчислювальних можливостей БПЛА.

Об`єкт дослідження – використання нейронних мереж для виявлення об’єктів.

Предмет дослідження – формування набору даних, навчання нейронної мережі, створення та оптимізація алгоритмів для навігації та розпізнавання об’єктів.

Методи дослідження включають поєднання автоматизованої та ручної анотації даних, розробку синтетичних датасетів для навчання моделей, а також написання і впровадження алгоритмів для виявлення об'єктів та корекції напрямку руху БПЛА.

У рамках даної роботи досліджується процес навчання моделей, створення тренувальних наборів даних, вибір оптимальної архітектури та мікропроцесорного модуля, а також алгоритми обробки відео, спрямовані на виявлення об'єктів та навігацію безпілотних літальних апаратів (БПЛА).

Результати дослідження можуть використовуватись для розвитку систем безпілотної авіації, відкриваючи нові можливості для ефективного впровадження технологій в безпілотних літальних апаратах.

1 БАЗОВІ ПРИНЦИПИ ТА ВИДИ НЕЙРОННИХ МЕРЕЖ

1.1 Загальна інформація

Нейромережі, або штучні нейронні мережі, є ключовим елементом галузі машинного навчання та глибокого навчання. Ці системи моделюють спосіб функціонування людського мозку, використовуючи велику кількість з'єднаних штучних нейронів, які вмикаються або вимикаються за допомогою функцій активації. Основні компоненти нейромережі включають входи, ваги, суматори, функції активації та виходи.

1.2 Класифікація нейронних мереж

Нейронні мережі класифікуються за різними критеріями, такими як архітектура, спосіб зв'язку між шарами, функції активації тощо. Далі описані основні класифікації та типи нейронних мереж.

1.2.1 Методи навчання нейромереж

У сфері машинного навчання завдання зазвичай розділяють на три основні категорії: навчання з вчителем, навчання без вчителя та навчання з підкріпленням. Ці категорії визначаються тим, як відбувається процес навчання і як системі, яку ми тренуємо, надається зворотний зв'язок.

Нейронні мережі, що базуються на методі навчання з вчителем, представляють собою клас алгоритмів машинного навчання, які використовують набір навчальних даних, де для кожного вхідного прикладу є відомий вихід або мітка. Головна ідея полягає в тому, щоб модель навчалася зіставляти вхідні дані з відповідними вихідними значеннями, щоб вона могла вивчити корисні абстракції та закономірності в даних.

Процес навчання з вчителем починається з передбачення моделі, яке порівнюється з вірними вихідними значеннями. Різниця між передбаченими та вірними результатами використовується для обчислення втрати або помилки моделі. Після цього використовується алгоритм оптимізації, такий як зворотнє

поширення помилки, для коригування параметрів моделі так, щоб мінімізувати цю втрату.

Дані, використані для навчання, зазвичай розділяються на дві частини: тренувальний та тестовий набори. Тренувальний набір використовується для самого процесу навчання, тоді як тестовий набір використовується для оцінки ефективності моделі на нових даних.

Цей підхід дозволяє нейронним мережам навчатися робити передбачення для нових вхідних даних, які не були використані під час навчання. Велика перевага навчання з вчителем полягає в тому, що воно може бути використане для різноманітних задач, таких як класифікація, регресія, або розпізнавання образів, роблячи його універсальним методом для різноманітних завдань машинного навчання.

Нейронні мережі, які використовують навчання без вчителя, представляють собою тип алгоритмів машинного навчання, де модель намагається вивчити природні структури та закономірності в наборі даних, не маючи явно визначених вихідних міток. Головна ідея полягає в тому, щоб модель самостійно виявляла характеристики та корисні структури в невизначених даних.

У цьому контексті навчання без вчителя часто використовує методи кластеризації, зменшення розмірності, чи автокодування. Кластеризація дозволяє групувати схожі елементи даних, тоді як зменшення розмірності спрощує представлення даних, зберігаючи важливу інформацію. Автокодування використовується для створення ефективних представлень даних, де модель намагається відновити вхідні дані, здатні вилучити важливі ознаки.

Важливим аспектом навчання без вчителя є здатність виявляти приховані закономірності в даних та використовувати ці відомості для подальшого аналізу чи вирішення завдань. Цей підхід особливо корисний у випадках, коли вихідні мітки або категорії відсутні, або коли завдання полягає в виявленні нових, раніше невідомих шаблонів у даних.

Нейронні мережі, які використовують навчання з підкріпленням, представляють собою тип машинного навчання, де модель взаємодіє з навколишнім середовищем і отримує позитивне або негативне підкріплення в залежності від власних дій. Головна ідея полягає в тому, щоб модель самостійно вивчала оптимальні стратегії для досягнення певних цілей, максимізуючи отриманий підкріплення.

У цьому контексті модель приймає деякі дії в середовищі і отримує числовий сигнал, який вказує, наскільки ці дії були ефективними для досягнення поставленої мети. Використовуючи це підкріплення, модель оновлює свої внутрішні параметри для покращення майбутніх вчинків. Цей цикл навчання з підкріпленням повторюється, дозволяючи моделі навчатися оптимальним стратегіям відтворення завдання.

Навчання з підкріпленням широко використовується в різних областях, включаючи гри, робототехніку, управління, та інші. Важливою частиною цього підходу є збалансованість між дослідженням та використанням вже вивчених стратегій для досягнення оптимальних результатів у нових ситуаціях.

1.2.2 Типи нейромереж за архітектурою

Перцептрон. Модель персептрону, запропонована Мінскі-Паперт, є однією з найпростіших і найстаріших моделей нейрона. Це найменша одиниця нейромережі, яка виконує певні обчислення для виявлення ознак або бізнес-інтелекту во вхідних даних. Вона приймає зважені вхідні дані та застосовує функцію активації для отримання вихідних даних як кінцевого результату. Він також відомий як TLU (блок логіки з пороговим значенням).

Персептрон – це алгоритм навчання, який класифікує дані на дві категорії, тому він є бінарним класифікатором.

Переваги персептронів полягають в тому, що вони можуть реалізовувати логічні елементи, такі як І, АБО чи НЕ. До недоліків відноситяся те, що персептрони можуть вивчати лише лінійно розділені проблеми.

Багатошаровий перцептрон (MLP) є одним з ключових елементів у світі нейронних мереж та глибокого навчання. Ця модель є розвитком простішого перцептрона, але з додаванням кількох прихованих шарів, що робить її значно потужнішою та гнучкішою.

У MLP, все починається з вхідного шару, де кожен нейрон відповідає за один атрибут вхідних даних. Ці дані потім передаються через один або декілька прихованих шарів. Ці приховані шари – справжнє серце MLP, де відбувається основна обробка інформації. Нейрони в цих шарах використовують ваговані з'єднання для трансформації вхідних даних, додаючи рівень абстракції та дозволяючи мережі вчитися складним залежностям.

Ключовим моментом у роботі MLP є використання активаційних функцій. Ці функції додають нелінійність до обробки даних, що є критично важливим для вивчення складних патернів. Популярні активаційні функції, такі як ReLU або сигмоїда, дозволяють MLP ефективно вирішувати різноманітні задачі, від класифікації зображень до прогнозування фінансових тенденцій.

Однією з унікальних особливостей MLP є її повністю з'єднана структура. Це означає, що кожен нейрон у певному шарі має з'єднання з усіма нейронами наступного шару. Така структура дозволяє мережі ефективно обробляти інформацію, роблячи її здатною виявляти складні залежності у великих наборах даних.

Процес навчання в MLP включає два основних етапи: пряме поширення та зворотне поширення. Під час прямого поширення, дані проходять через мережу від вхідного до вихідного шару. Після цього, під час зворотного поширення, мережа коригує свої ваги на основі помилок, зроблених у прогнозах, що дозволяє MLP поступово покращувати свою точність.

MLP знайшли своє застосування в різних сферах, від розпізнавання мови до аналізу тексту. Їх гнучкість та здатність вчитися складним залежностям роблять їх незамінними у багатьох областях глибокого навчання.

Прямі нейронні мережі (Feedforward Neural Networks, FNN) є основним і базовим типом нейромереж. Вони представляють собою систему з'єднаних нейронів, розташованих у вигляді шарів, де інформація переміщується вперед, від входу до виходу, без циклічних зв'язків.

Основними компонентами звичайної нейронної мережі є:

- Вхідний шар (Input Layer): Це перший шар нейронів, який приймає вхідні дані. Кількість нейронів у цьому шарі визначається кількістю вхідних ознак чи змінних.
- Приховані шари (Hidden Layers): Мережа може мати один або кілька прихованих шарів, де кожен шар містить нейрони, які обробляють інформацію. Кількість шарів і кількість нейронів у кожному з них – це параметри, які можна налаштовувати.
- Вихідний шар (Output Layer): Це останній шар, який видає результати обчислень. Кількість нейронів у вихідному шарі визначається кількістю класів чи значень, які ми намагаємося передбачити.

Кожен з'єднаний нейрон у мережі має вагу, яка регулює його вплив на вхідні дані. Ваги нейронів налаштовуються під час процесу навчання, де мережа намагається вивчити залежності вхідних даних і вихідних результатів.

Процес навчання включає в себе використання алгоритму оптимізації, такого як зворотній розповсюдження помилки (Backpropagation), для коригування ваг нейронів і досягнення оптимального виконання на тестових чи валідаційних даних.

Звичайні нейронні мережі застосовуються в різних областях, включаючи класифікацію, регресію, та інші завдання, і вони слугують основою для більш складних архітектур, таких як глибокі нейронні мережі.

Рекурентні нейронні мережі (RNN) – це тип нейромереж, який використовується для роботи з послідовністю даних. Їхні нейрони мають зворотний зв'язок, що означає, що вони можуть зберігати і використовувати

інформацію з попередніх кроків. Це дозволяє RNN ефективно обробляти послідовність даних, такі як текст, мова, часові ряди.

Одна з ключових особливостей RNN – це можливість враховувати контекст і взаємозв'язки між даними на різних часових кроках. Кожен нейрон у RNN приймає вхід із поточного кроку часу, а також вхід із попереднього кроку. Такий механізм дозволяє RNN враховувати попередні стани і приймати рішення на основі попередньої інформації.

Однак у класичних RNN є проблема зниклих градієнтів, яка виникає під час тривалого навчання. Для вирішення цієї проблеми були розроблені вдосконалені архітектури, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit). Вони мають спеціальні механізми для контролю потоку інформації і зберігання важливих залежностей.

RNN застосовуються в багатьох областях, таких як обробка природної мови, генерація тексту, машинний переклад, аналіз часових рядів та інше. Вони дозволяють ефективно враховувати контекст і роблять це зокрема завдяки своїй спроможності працювати з послідовністю даних.

Генеративні мережі суперництва (Generative Adversarial Networks, GAN) – це інноваційний клас нейромереж, який вперше був запропонований в 2014 році. Однією з головних особливостей GAN є їхній двоїстий характер, що включає генератор та дискримінатор.

Генератор відповідає за створення нових даних, наприклад, зображень або тексту, в той час як дискримінатор намагається визначити, чи є надані дані реальними чи згенерованими. Обидва ці компоненти взаємодіють у формі "гри", де генератор старається підвищити реалістичність своїх створених даних, а дискримінатор тримає крок, намагаючись розрізнити між реальним та згенерованим.

У результаті цієї взаємодії GAN може створювати дуже реалістичні дані, які часто не відрізняються від реальних. Це може застосовуватися для генерації

мистецьких зображень, реалістичних фотографій обличчя, або навіть для усунення шуму з фотографій.

Однак, важливо відзначити, що процес навчання GAN може бути важким і вимагає уваги до деталей, так як існує ризик, що генератор стане надто "вправним", або навпаки, дискримінатор не зможе розрізнити між справжніми та згенерованими даними.

GAN широко використовуються в галузі комп'ютерного зору, обробки зображень та генерації контенту. Вони представляють собою потужний інструмент для створення нових та цікавих візуальних елементів, що відкриває широкі перспективи в галузі творчості та розваг.

Мережі короткострокової та довгострокової пам'яті (LSTM, GRU) – це розширені варіанти рекурентних нейронних мереж (RNN), розроблені для вирішення проблеми зниклих градієнтів та здатні до зберігання та використання інформації на тривалий термін.

LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit) використовуються для обробки послідовних даних та викорінюють проблему зниклих градієнтів, яка може виникнути при тривалому навчанні RNN. Обидві архітектури використовують спеціальні механізми, щоб зберігати та передавати інформацію на різних часових кроках.

LSTM використовує комірки пам'яті, входи, виходи та ворота для керування потоком інформації. Це дозволяє їй ефективно визначати, коли забувати чи додавати нову інформацію до пам'яті. GRU, з іншого боку, має менше параметрів і використовує дві ворота – ворота оновлення та ворота скидання – для керування інформацією.

Обидві архітектури показують вражаючу ефективність у завданнях, де важлива обробка довгострокових залежностей та врахування контексту. Вони використовуються в різних застосуваннях, таких як машинний переклад, генерація тексту, обробка природної мови та інші, де обробка послідовних даних є ключовою.

Автоенкодери (Autoencoders) – це тип нейромережі, який використовується для безнагаданного навчання та стискання даних. Основна ідея полягає в тому, щоб автоматично вивчити представлення вхідних даних, створюючи компактне кодування, яке може бути використане для відтворення оригінальних даних.

Автоенкодер складається з двох частин: кодувальника (encoder) і декодувальника (decoder). Кодувальник перетворює вхідні дані в представлення меншого розміру, що називається кодом, а декодувальник відтворює оригінальні дані з цього коду.

Важливою особливістю автоенкодерів є те, що вони навчаються без участі вчителя, оскільки вихідні дані використовуються як цільові для навчання. Процес навчання включає в себе мінімізацію різниці між вхідними та відтвореними даними.

Існують різні варіації автоенкодерів, такі як варіаційні автоенкодери (Variational Autoencoders, VAE), які забезпечують генерацію нових, схожих на навчальні дані, і денойзерні автоенкодери, призначені для відновлення даних, обтяжених шумом чи пертурбаціями.

Застосування автоенкодерів розповсюджується в області стискання зображень, відновлення сигналів, генерації зображень, анамалійного виявлення та багатьох інших сферах, де важливо отримати ефективне та репрезентативне представлення даних.

Нейронні мережі з радіально-базисними функціями, або RBF-мережі, представляють собою цікавий тип штучних нейронних мереж, які застосовуються в різних областях. Основна особливість цих мереж полягає в тому, що вони використовують функції, які називаються радіально-базисними функціями (RBF), для обробки інформації та вирішення завдань.

У структурі RBF-мережі можна виділити кілька важливих компонентів. По-перше, є вхідний шар, який приймає вхідні дані. Це можуть бути характеристики об'єкта або будь-які числові дані, що потребують обробки.

Далі слідує основна особливість RBF-мереж – шар радіально-базисних функцій. Він складається з RBF-нейронів, кожен з яких володіє своїм центром. Ці центри виступають як точки відліку для визначення схожості між вхідними даними та центрами. Тобто, вони визначають, наскільки близькі вхідні дані до конкретного центру RBF-нейрона.

Після обчислення відстані між вхідними даними та центрами, використовуються RBF-функції для перетворення цієї відстані в активацію. Це важливий крок, оскільки він визначає, наскільки активованими стають RBF-нейрони в даному контексті.

З отриманими активаціями від RBF-нейронів вагові коефіцієнти використовуються для зваженого сумування. Це дозволяє RBF-мережі генерувати вихідні дані або прогнози. Зазвичай, у вихідному шарі мережі є один або кілька вузлів, кожен з яких відповідає певній категорії або класу, і вони генерують вихідні дані на основі обчислень.

Навчання RBF-мереж включає в себе налаштування центрів та ширин RBF-нейронів, а також вагових коефіцієнтів. Це важливо, оскільки від цих параметрів залежить якість та точність мережі у вирішенні завдань.

Застосування RBF-мереж можна знайти в різних областях, де важлива точність та апроксимація функцій. Вони застосовуються у прогнозуванні часових рядів, апроксимації функцій, а також у завданнях класифікації та асоціації даних. Ці мережі можуть бути потужним інструментом для вирішення задач, де інші типи нейронних мереж можуть бути менш ефективними.

Згорткові нейронні мережі (CNNs або ConvNets) є варіацією звичайних нейронних мереж, але з певними особливостями, що роблять їх особливо ефективними для обробки зображень.

Звичайні нейронні мережі отримують на вхід вектор і перетворюють його через серію прихованих шарів (рисунок 1.1). Кожен нейрон у цих шарах повністю з'єднаний з усіма нейронами попереднього шару. Останній повністю з'єднаний шар називається "вихідним шаром" і в налаштуваннях класифікації представляє класи.

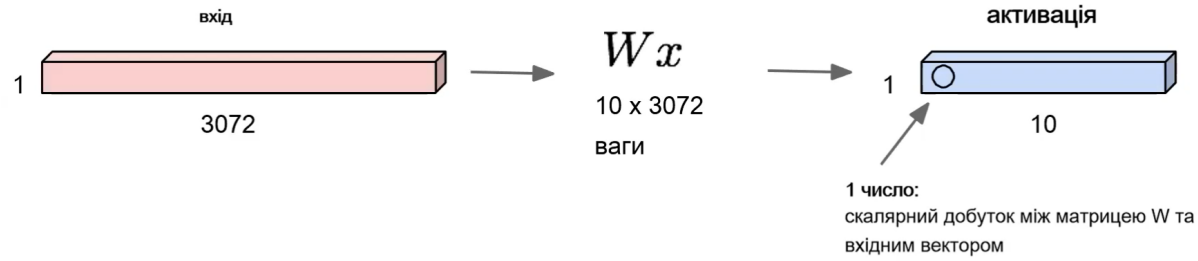

Рисунок 1.1 - Одношаровий перцептрон

Згорткові нейронні мережі використовують факт, що вхідні дані є зображеннями, що дозволяє кодувати певні властивості в архітектурі, роблячи функцію прямого розповсюдження більш ефективною та значно зменшуючи кількість параметрів у мережі.

CNN складаються з шарів нейронів, кожен з яких реагує на певні аспекти візуальних даних. На відміну від традиційних нейронних мереж, які сприймають вхідні дані як пласкі, двовимірні масиви, CNN зберігають просторову ієрархію в зображеннях завдяки унікальній архітектурі.

Архітектура CNN може бути різною, але типова включає декілька шарів згортки (з ReLU) і шарів пулінгу, за якими слідують повнозв'язні шари:

- Вхідний шар: містить сиру інформацію про пікселі зображення.
- Згортковий шар: основний компонент, де застосовуються фільтри для виявлення специфічних особливостей, таких як краї або текстури (рисунок 1.2).

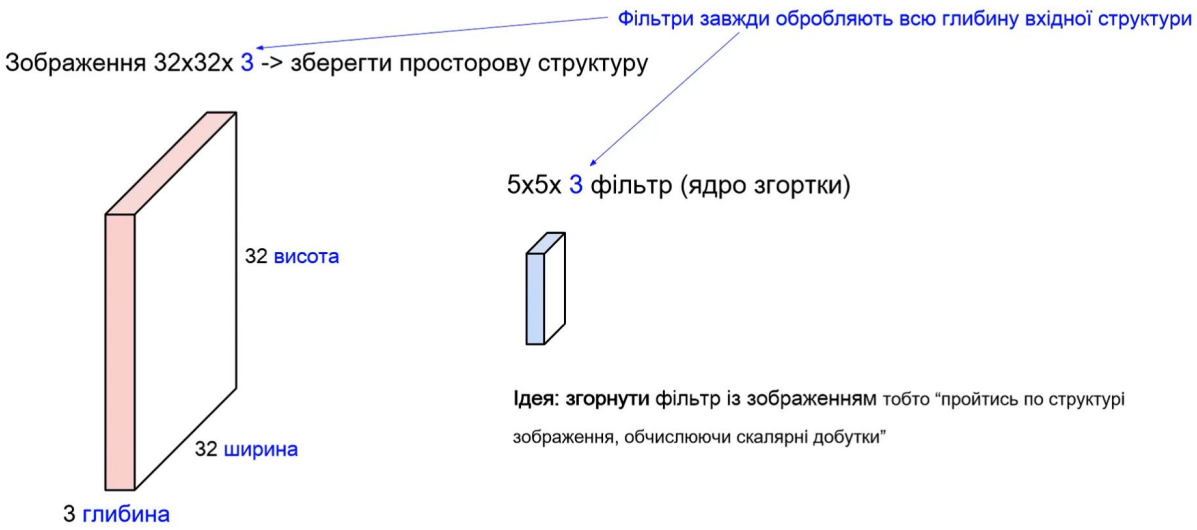

Рисунок 1.2 - Схема згортки у згортковій нейронній мережі

• Шар активації (ReLU): вводить нелінійність, дозволяючи мережі вирішувати складні задачі (рисунок 1.3).

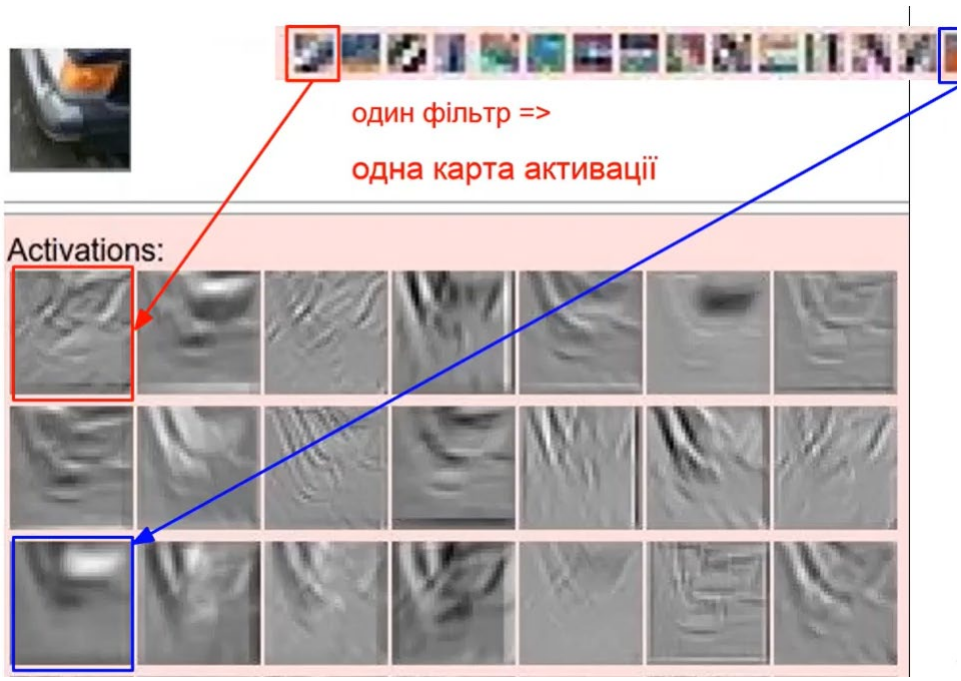

Рисунок 1.3 - Візуалізація карт активації у згортковій нейронній мережі

• Пулінг-шар: зменшує просторові розміри (ширину, висоту), щоб знизити обчислювальне навантаження, використання пам'яті та кількість параметрів (рисунок 1.4).

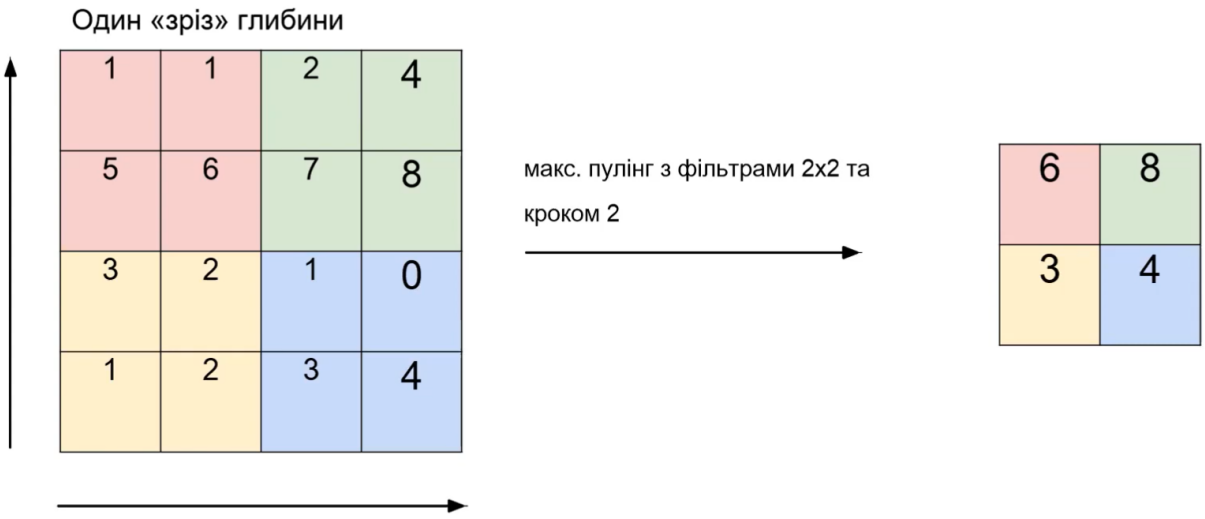

Рисунок 1.4 - Операція максимального згортання (max pooling) із кроком 2 на матриці

• Повнозв'язний шар: нейрони мають з'єднання з усіма активаціями попереднього шару і використовуються для класифікації зображень, вказаний на рисунок 1.5 з правого боку.

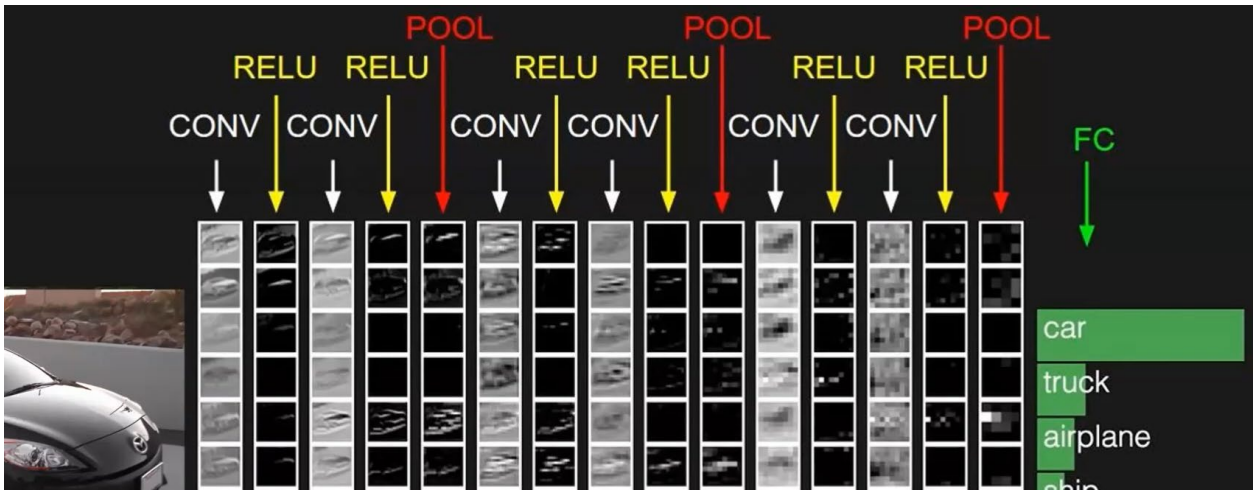

Рисунок 1.5 - Схема архітектури згорткової нейронної мережі

CNN перетворює вхідне зображення шар за шаром від первинних пікселів до остаточних класифікаційних оцінок. Деякі шари містять параметри, інші – ні. Наприклад, шари CONV/FC містять параметри (ваги та зміщення нейронів), тоді як шари RELU/POOL виконують фіксовану функцію.

Параметри:

- Глибина вихідного об'єму: Відповідає кількості фільтрів.
- Крок (Stride): Визначає, наскільки фільтр переміщується по входу.
- Нульове доповнення (Zero-padding): Додає нулі навколо входу, щоб контролювати розмір вихідного об'єму.

Після огляду найбільш поширених архітектур нейронних мереж можна зробити висновок, що для завдань детекції об'єктів на відео найбільш оптимальним вибором є використання згорткових нейронних мереж. Цей вибір обумовлений широким спектром застосування цих архітектур у галузі комп'ютерного зору, включаючи розпізнавання образів, класифікацію зображень та обробкою відеоматеріалів. Висока ефективність цих нейронних мереж обумовлена їхньою здатністю виявляти локальні особливості на зображеннях і зменшувати кількість параметрів завдяки спільному використанню ваг.

1.3 Основні задачі графічних нейромереж

Класифікація зображень – це процес визначення, до якої категорії або класу належить конкретне зображення. Ця задача має велике значення у розпізнаванні образів та машинному навчанні. Наприклад, нейромережі можуть бути навчені розпізнавати різні види тварин на фотографіях або класифікувати медичні зображення для діагностики захворювань. Ефективність таких систем оцінюється за точністю та швидкістю виконання класифікації, а також за їхньою здатністю правильно працювати з неочікуваними чи новими зображеннями (рисунок 1.6, ліворуч).

Визначення об'єктів в зображеннях – це завдання, яке полягає не лише у визначенні присутності об'єктів у зображенні, але й у точному визначенні їх розташування і розмірів. Це може включати в себе ідентифікацію кількох об'єктів та їх взаєморозташування. Наприклад, в системах автономного водіння, детекція об'єктів використовується для ідентифікації пішоходів, інших транспортних

засобів та дорожніх знаків. Такі системи повинні бути високо точними та надійними, оскільки помилки можуть мати серйозні наслідки (рисунок 1.6, посередині).

Сегментація зображень є однією з найскладніших задач у галузі комп'ютерного зору. Цей процес включає розділення зображення на різні сегменти, які представляють різні об'єкти або регіони. Сегментація може бути особливо складною, коли об'єкти на зображенні перекривають один одного або коли вони мають складні контури. В медичних застосуваннях, наприклад, сегментація використовується для відділення патологічних тканин від здорових, що має важливе значення для планування лікування. В геоінформаційних системах сегментація використовується для аналізу супутникових зображень, дозволяючи ідентифікувати різні ландшафтні елементи, такі як ріки, дороги, міські агломерації тощо (рисунок 1.6, праворуч).

Рисунок 1.6 - Приклад задач комп'ютерного зору: класифікація, виявлення об'єктів та сегментація зображень

1.4 Проблема перенавчання (Overfitting).

Перенавчання або Overfitting у сфері машинного навчання та нейронних мереж, це ситуація, коли модель стає надто специфічною для тренувального набору даних, на яких вона була навчена. Це означає, що хоча модель може показувати

високу точність під час тренування, вона втрачає здатність до узагальнення і тому не ефективно працює на нових, раніше невідомих даних.

Суть перенавчання полягає в тому, що модель, замість того щоб вивчити загальні шаблони та тенденції, "запам'ятовує" специфічні характеристики тренувального набору, включаючи випадкові шуми та відхилення. Це явище часто відбувається, коли модель має велику кількість параметрів та складну структуру, але обмежений або не повністю репрезентативний набір даних для навчання.

Основною проблемою перенавчання є те, що воно робить модель менш ефективною у практичному застосуванні. Модель, яка ідеально працює на даних, на яких вона була тренована, але не може адекватно прогнозувати або аналізувати нові дані, не є корисною у реальних умовах. Це важливо усвідомлювати при розробці та тренуванні нейронних мереж, оскільки мета будь-якої моделі машинного навчання – це здатність ефективно працювати не тільки на відомих даних, а й узагальнювати висновки на нових.

Причини перенавчання:

- Надмірна складність моделі: Якщо модель має занадто багато параметрів (наприклад, у випадку глибоких нейронних мереж), вона може "вчити" навіть найдрібніші деталі тренувального набору даних.
- Обмежений або неякісний набір даних: Якщо тренувальний набір даних малий або не повністю представляє проблему, що має бути вирішена, модель може адаптуватися до цих обмежених чи специфічних умов.
- Недостатня регуляризація: Регуляризація допомагає запобігати перенавчанню шляхом покарання моделі за надто складну структуру. Відсутність або недостатня регуляризація може сприяти перенавчанню.

До способів подолання перенавчання належать:

1. Розділення даних на набори.

Цей підхід передбачає створення окремих груп даних, кожна з яких має специфічну роль у процесі навчання та оцінювання моделі.

На початку процесу навчання великий набір даних ділиться на, як мінімум, три окремі частини: тренувальний набір, валідаційний набір та тестовий набір. Кожен з цих наборів виконує унікальну функцію:

- Тренувальний набір: Це основна частина даних, яка використовується для навчання моделі. Алгоритми машинного навчання "вчаться" на цих даних, налаштовуючи свої параметри (ваги в нейронних мережах) для мінімізації помилок.

- Валідаційний набір: Ця частина даних використовується для періодичної оцінки продуктивності моделі під час навчання. Валідаційний набір дозволяє налаштувати гіперпараметри моделі (наприклад, розмір пакету або швидкість навчання) та визначити, коли слід зупинити навчання для запобігання перенавчанню.

- Тестовий набір: Після завершення навчання моделі, цей набір даних використовується для остаточної оцінки її продуктивності. Оскільки тестовий набір не використовувався ні у процесі тренування, ні для налаштування гіперпараметрів, він дозволяє зрозуміти, наскільки добре модель здатна узагальнювати свої висновки на нових, невідомих даних.

2. Регуляризація.

Основна ідея регуляризації полягає в тому, що замість того, щоб просто мінімізувати помилку на тренувальному наборі даних, модель також "карається" за великі ваги (параметри). Це робить модель менш чутливою до окремих, особливо складних шаблонів у тренувальному наборі, що допомагає уникнути перенавчання.

Існують два основних типи регуляризації, які часто використовуються в машинному навчанні:

• L1 регуляризація (лассо регуляризація): Цей тип додає штраф, що дорівнює абсолютній сумі ваг моделі до функції втрат. Це може призвести до того, що деякі ваги стануть нульовими, що робить модель простішою та зменшує її залежність від даних.

• L2 регуляризація (регуляризація Тихонова): Вона додає штраф, рівний квадрату величини ваг. Цей метод менше схильний до встановлення ваг на нуль, але зменшує величину ваг, роблячи модель менш чутливою до невеликих флуктуацій у даних.

3. Використання Dropout.

Суть методу полягає у випадковому "відключенні" деяких нейронів мережі під час процесу тренування. Це робиться для того, щоб запобігти занадто сильній залежності від конкретних шаблонів чи характеристик даних, що може привести до перенавчання.

Коли використовується Dropout, на кожному кроці тренування випадково обрані нейрони ігноруються. Це означає, що вони не беруть участі у прямому або зворотному розповсюдженні сигналу. Процент "відключених" нейронів є гіперпараметром, який можна налаштовувати. Зазвичай цей відсоток варіюється від 20% до 50%.

Основна перевага використання Dropout полягає у зменшенні залежності від конкретних особливостей тренувального набору даних. Оскільки нейрони не можуть покладатися на наявність певних інших нейронів, вони "змушені" вчитися більш корисних та узагальнених шаблонів у даних. Такий підхід допомагає підвищити здатність моделі до узагальнення.

4. Раннє припинення (Early Stopping)

Ідея полягає в моніторингу продуктивності моделі на валідаційному наборі даних і припиненні тренування, якщо продуктивність перестає покращуватися або навіть починає погіршуватися.

Під час тренування моделі, зазвичай відстежується помилка або точність на валідаційному наборі даних після кожної епохи (проходу через весь тренувальний набір). Якщо помилка на валідації не зменшується протягом певної кількості епох (часто називається "терпінням"), тренування припиняється. Це допомагає уникнути ситуації, коли модель продовжує навчатися на тренувальних даних, стаючи все більш специфічною для них і втрачаючи здатність узагальнювати.

Раннє припинення діє як форма регуляризації, оскільки воно запобігає моделі від надмірного "запам'ятовування" або "переоснащення" під тренувальні дані.

5. Пошук оптимального розміру пакету (Batch Size)

Розмір пакету (Batch Size) в машинному навчанні визначає кількість зразків даних, які обробляються перед оновленням ваг моделі. Вибір оптимального розміру пакету є важливим рішенням, яке впливає на ефективність і якість тренування моделі.

Малий розмір пакету означає, що ваги моделі оновлюються частіше, що може призвести до більш швидкого навчання і допомогти моделі уникнути локальних мінімумів. Однак, це також може спричинити більшу варіативність у процесі навчання і потребує більше обчислювальних ресурсів.

З іншого боку, великий розмір пакету забезпечує стабільніші оновлення ваг, але може потребувати більше пам'яті та обчислювальних ресурсів. Крім того, занадто великий розмір пакету може призвести до того, що модель застрягне в локальному мінімумі, що погіршує її здатність узагальнювати.

Таким чином, вибір розміру пакету є компромісом між швидкістю тренування, стабільністю процесу навчання та якістю узагальнення. Ідеальний розмір пакету може варіюватися в залежності від конкретної задачі, набору даних і обчислювальних можливостей. Експериментування з різними розмірами пакетів часто є необхідним кроком для знаходження оптимального балансу для конкретної задачі машинного навчання.

6. Підвищення обсягу даних (Data Augmentation).

Data Augmentation, є важливою технікою в процесі тренування нейронних мереж, особливо в задачах комп'ютерного зору. Ця техніка полягає у штучному збільшенні розміру та різноманітності тренувального набору даних шляхом застосування ряду трансформацій до існуючих даних.

Основна ідея Data Augmentation полягає в тому, що модифікація даних може допомогти моделі краще узагальнювати і покращувати її продуктивність на нових, невідомих даних. Наприклад, у задачах зі зображеннями це може включати обертання, масштабування, переклад, зміну кольорової схеми або навіть введення шуму. Ці трансформації створюють нові версії існуючих зображень, які зберігають основну інформацію, але виглядають дещо інакше.

Шляхом тренування на модифікованих даних, модель навчається бути більш стійкою до невеликих варіацій у вхідних даних, що покращує її здатність до узагальнення. Цей метод особливо ефективний у галузях, де збір додаткових даних є складним або дорогим.

1.5 Формування датасету для навчання моделі

Підготовка даних це фундамент для будь-якої моделі, оскільки якість та точність моделі безпосередньо залежать від якості та обробки вхідних даних. Незважаючи на зростаючу потужність обчислювальних систем та алгоритмів, навіть найпросунутіші моделі машинного навчання не зможуть демонструвати високу продуктивність без добре підготовлених даних.

На самому початку стоїть збір даних. Цей етап включає в себе вибір та збір відповідних датасетів, які відображають різноманіття ситуацій та об'єктів, які потрібно ідентифікувати. Це може включати в себе публічні датасети, використання веб-скрапінгу для збору зображень та відео з Інтернету. Важливо, щоб ці джерела були різноманітними та відображали широкий спектр сценаріїв, що підвищить універсальність та адаптивність моделі.

Далі слідує очищення даних, процес, який вимагає видалення помилкових, відсутніх або нерелевантних даних. Цей етап є критичним, адже некоректні дані

можуть значно спотворити результати, що отримує модель машинного навчання. Також велику роль відіграє якість зображень та відеоданих. Низька якість, шум або спотворення в зображеннях можуть ускладнити процес детекції об'єктів, знижуючи точність моделі.

Після очищення даних наступає етап розмітки. Тут дані маркуються або з метою забезпечення моделі інформацією про те, що саме в датасеті є об'єктом інтересу. Цей процес вимагає ручної роботи та великої уваги до деталей.

Для ручної розмітки даних, використовуються як програмні рішення, так і веб– платформи, що надають можливість розмітки даних. Одні з найбільш популярних:

- LabelImg – це відкритий інструмент для ручної анотації зображень. Він дозволяє легко обводити об'єкти на зображеннях за допомогою рамок (bounding boxes) та зберігати ці анотації у форматах, таких як XML.
- Roboflow – це платформа, яка надає інструменти розробникам для створення додатків комп'ютерного зору.

На наступному етапі підготовки даних може використовуватися техніка аугментації даних (data augmentation). Вона дозволяє збільшити об’єм та різноманітність даних шляхом застосування різних трансформацій, таких як обертання, масштабування або зміна освітленості зображень. Такий підхід сприяє підвищенню узагальнюючої здатності моделі.

Важливість підготовки даних також виявляється у контексті репрезентативності даних. Нерепрезентативні набори даних можуть призвести до виникнення упередженості в моделях, що робить їх нездатними адекватно працювати в реальному світі. Тому, належна підготовка даних включає в себе не тільки технічні аспекти, а й розуміння контексту задачі, для якої буде використовуватися модель.

Для оцінки ефективності та здатності моделі до узагальнення використовується поділ датасету на навчальний, тестовий та валідаційний набори:

Навчальний сет (Training set) – це основний набір даних, який використовується для навчання моделі. Він містить приклади, на основі яких алгоритм вчиться розпізнавати закономірності та зв'язки.

Валідаційний сет (Validation set) – використовується для оцінки моделі в процесі навчання, зазвичай після кожної епохи (проходу через навчальний набір). Це дозволяє регулювати гіперпараметри моделі (наприклад, швидкість навчання) і визначити, коли припинити навчання, щоб уникнути перенавчання.

Тестовий сет (Test set) – використовується після завершення навчання моделі, для остаточної оцінки її продуктивності. Ці дані не використовуються під час навчання і мають бути репрезентативними для реального використання моделі. Оцінка на тестовому наборі дає змогу зрозуміти, наскільки добре модель зможе справлятися з невідомими даними у реальному світі.

1.6 Особливості створення нейронної моделі для БПЛА

Створення нейронної моделі для детекції об'єктів з використанням БПЛА включає ряд особливостей та викликів, які потрібно врахувати:

- Якість та роздільна здатність: так як БПЛА можуть літати на значних висотах, модель повинна розпізнавати найменші ознаки об’єкта, для цього потрібна модель, яка змогла б працювати з достатньою роздільною здатністю відеопотоку.
- Обробка в реальному часі: модель для повинна бути здатна обробляти дані в реальному часі для швидкої реакції на зміни в оточенні.
- Робота в різних умовах: модель повинна бути стійкою до різних умов освітлення, погоди та ландшафту.
- Мінімізація помилок: точність детекції критично важлива, для розпізнавання об’єктів інтересу.
- Обмеження пам'яті та обчислювальних ресурсів: нейронна мережа повинна бути оптимізована, щоб працювати з обмеженими обчислювальними ресурсами та пам'яттю.
- Адаптація до специфічних завдань: модель може вимагати налаштування під конкретні завдання.

- Збір та розмітка даних: для тренування ефективних моделей потрібно великий набір даних, який правильно розмічений та представляє різноманітні сценарії.
- Інтеграція з іншими системами БПЛА: Нейронна модель повинна інтегруватися з системами навігації та управління.

1.7 Вибір архітектури

YOLOv8 є однією з найновіших та найефективніших версій архітектури YOLO, яка використовує згорткові та інші шари для виявлення об'єктів на відео в реальному часі.

Обрання YOLOv8 для детекції об'єктів з БПЛА має кілька важливих переваг:

1. Висока швидкість обробки: YOLOv8 це одна з найшвидких архітектур для об'єктної детекції. Це критично важливо для БПЛА, де швидкість обробки має вирішальне значення для реакції на динамічні умови середовища.
2. Висока точність: YOLOv8, як остання ітерація у серії YOLO, має поліпшення в точності детекції об'єктів. Використання цієї архітектури допоможе зменшити помилки та підвищити надійність системи детекції об'єктів.
3. YOLOv8 оптимізована для роботи в режимі реального часу і була спроектована з урахуванням ефективного використання обчислювальних ресурсів. Це робить її ідеальним варіантом для обладнання з обмеженими обчислювальними можливостями.
4. Здатність обробляти відео з високою роздільною здатністю: YOLOv8 ефективно обробляє великі зображення, що є перевагою для детекції об'єктів.
5. Гнучкість у навчанні та адаптації: YOLOv8 дозволяє гнучко налаштовувати модель під конкретні вимоги, , для різноманітних умов та різних задач.
6. Зручність в розробці та налаштуванні: оскільки YOLO є популярною архітектурою у спільноті комп'ютерного зору, існує багато ресурсів та підручників, які можуть бути корисними при розробці та налаштуванні моделі.

1.8 Вибір середовища для навчання моделі

Для середовища навчання моделі, був вибраний Google Colab. Це безкоштовний інструмент від Google, який дозволяє навчати модель прямо у браузері.

Однією з ключових особливостей Google Colab є те, що він надає безкоштовний доступ до обчислювальних ресурсів, включаючи графічні та тензорні процесори. Це робить його ідеальним для тренування складних моделей машинного навчання. Також, Colab легко інтегрується з Google Drive, що полегшує зберігання та обмін даними. Google Colab дозволяє використовувати популярні бібліотеки, такі як TensorFlow, PyTorch, Keras та інші.

2 КОМПЛЕКСНІ РІШЕННЯ ДЛЯ ФОРМУВАННЯ ДАТАСЕТІВ ТА НАВЧАННЯ НЕЙРОМЕРЕЖІ ДЛЯ БПЛА

Одним із важливих завдань у розробці моделей для БПЛА є формування ефективних наборів даних. Основні виклики, які необхідно вирішити, включають недостатність даних, що може призвести до недостатньої ефективності моделі, а також затратну процедуру ручної розмітки.

Для подолання цих викликів були розроблені наступні підходи:

Генерація синтетичного набору даних – цей підхід передбачає створення наборів даних у 3D середовищі з автоматичною розміткою, що дозволяє генерувати великі обсяги даних без необхідності ручної розмітки.

Напівавтоматична мануальна розмітка – цей метод поєднує в собі мануальну розмітку та використання нейромережі для підвищення ефективності процесу.

Розмітка за допомогою великих загальних моделей – використання передових моделей машинного навчання для попередньої розмітки даних, що зменшує потребу в ручній роботі.

Для навчання моделей БПЛА було визначено два ключові класи об'єктів: Tank та APC.

2.1 Збір матеріалів для навчання нейромережі

Перший етап включав в себе пошук та аналіз відеоматеріалів на YouTube, зокрема відео зафільмованих з дрону, в різних погодних умовах, для досягнення максимальної репрезентативності.

Після завантаження знайдених відео вони пройшли процес монтажу, під час якого були вирізані всі непотрібні фрагменти, і залишені лише ті, на яких присутні об'єкти інтересу (рисунок 2.1).

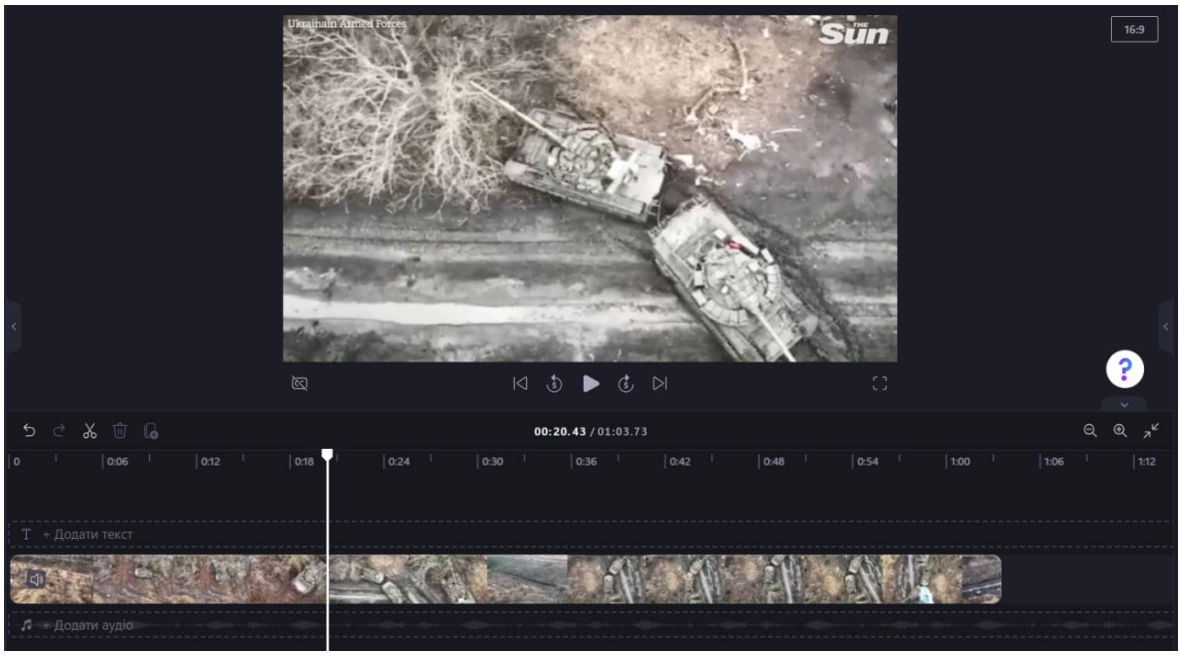

Рисунок 2.1 - Вікно відео редактору, вирізання нерепрезентативних частин відео

Після завершення вищезгаданого етапу було сформовано набір зображень із відеофайлу:

```
from moviepy.editor import VideoFileClip
import numpy as np
import os
from datetime import timedelta
from google.colab import files

framesPerSecond = 2

def format_timedelta(td):
    result = str(td)
    try:
        result, ms = result.split(".")
    except ValueError:
        return result + ".00".replace(":", "-")
```

```
        ms = round(int(ms) / 10000)
        return f"{result}.{ms:02}".replace(":", "-")

    def main(video_file):
        global framesPerSecond
        video_clip = VideoFileClip(video_file)
        filename, _ = os.path.splitext(video_file)

        if not os.path.isdir(filename):
            os.mkdir(filename)

        framesPerSecond = min(video_clip.fps, framesPerSecond)
        step = 1 / video_clip.fps if framesPerSecond == 0 else 1 / framesPerSecond

        for current_duration in np.arange(0, video_clip.duration, step):
                                                    frame_duration_formatted =
format_timedelta(timedelta(seconds=current_duration)).replace(":", "-")
                                            frame_filename = os.path.join(filename,
f"frame{frame_duration_formatted}.jpg")

            video_clip.save_frame(frame_filename, current_duration)

    uploaded_files = files.upload()
    for video_file in uploaded_files.keys():
        main(video_file)
```

Для досягнення оптимального співвідношення між кількістю та якістю, було експериментально вибрано значення FPS=2. В рамках дослідження було використано близько 20 відео.

2.2 Аугментація

Для покращення загальної здатності моделі до виявлення об'єктів, була застосована аугментація даних (рисунок 2.2). Перетворення, які були застосовані до зображень:

- Ротація (Rotation): включає в себе обертання зображень на різні кути. Це допомагає моделі адаптуватися до різних орієнтацій об'єктів, які можуть зустрічатися в реальному світі, наприклад, коли камера знаходиться під різними кутами до об'єкта.
- Чорно-біле перетворення (Grayscale): допомагає моделі концентруватися на текстурі та формі об'єктів, ігноруючи кольорову інформацію. Це може бути корисним у сценаріях, де кольорова інформація не є критичною.
- Розмиття (Blur): застосування випадкового гауссового розмиття допомагає моделі вчитися ідентифікувати об'єкти, навіть коли зображення не є чітким. Це може відтворювати умови, коли фокус камери не ідеальний.
- Яскравість (Brightness): Варіювання яскравості зображення допомагає моделі адаптуватися до різних умов освітленості, що є особливо важливим для роботи у реальному світі, де рівень освітлення може сильно варіюватися.
- Шум (Noise): Додавання шуму до зображень може допомогти запобігти перенавчанню моделі, змушуючи її вчитися виявляти корисні патерни, ігноруючи випадкові коливання в даних.
- Вирізання (Cutout): Ця техніка включає видалення частин зображення, імітуючи часткове закриття об'єктів. Це допомагає моделі вчитися розпізнавати об'єкти навіть тоді, коли вони частково закриті іншими об'єктами.

Аугментація виконувалась перед навчанням моделі для кожної вибірки:

```
import cv2
import albumentations as A
import os
import numpy as np
```

```
def augmentations():
    # Визначення окремих аугментацій з власними ймовірностями застосування
    transform = A.Compose([
        A.Rotate(limit=25, p=0.3),
        A.ToGray(p=0.2),
        A.GaussianBlur(blur_limit=(3, 7), p=0.2),
        A.RandomBrightnessContrast(p=0.3),
        A.GaussNoise(p=0.2),
        A.Cutout(num_holes=9, max_h_size=6, max_w_size=6, fill_value=0, p=0.2)
    ])
    return transform

def apply_augmentations(transform, image):
    augmented_image = transform(image=image)['image']
    return augmented_image

def load_and_augment_images(image_folder, transform):
    for filename in os.listdir(image_folder):
        if filename.endswith((".png", ".jpg", ".jpeg")):
            image_path = os.path.join(image_folder, filename)
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            # Застосування аугментацій до зображення
            augmented_image = apply_augmentations(transform, image)

            # Зберігання аугментованого зображення
            cv2.imwrite(os.path.join(image_folder, f"augmented_{filename}"), cv2.cvtColor(augmented_image, cv2.COLOR_RGB2BGR))
```

```
transform = augmentations()
image_folder = '/content/dataset'
load_and_augment_images(image_folder, transform)
```

Після проходження процедури аугментації, вихідні зображення поповнюються їх аугментованими варіантами (рисунок 2.2).

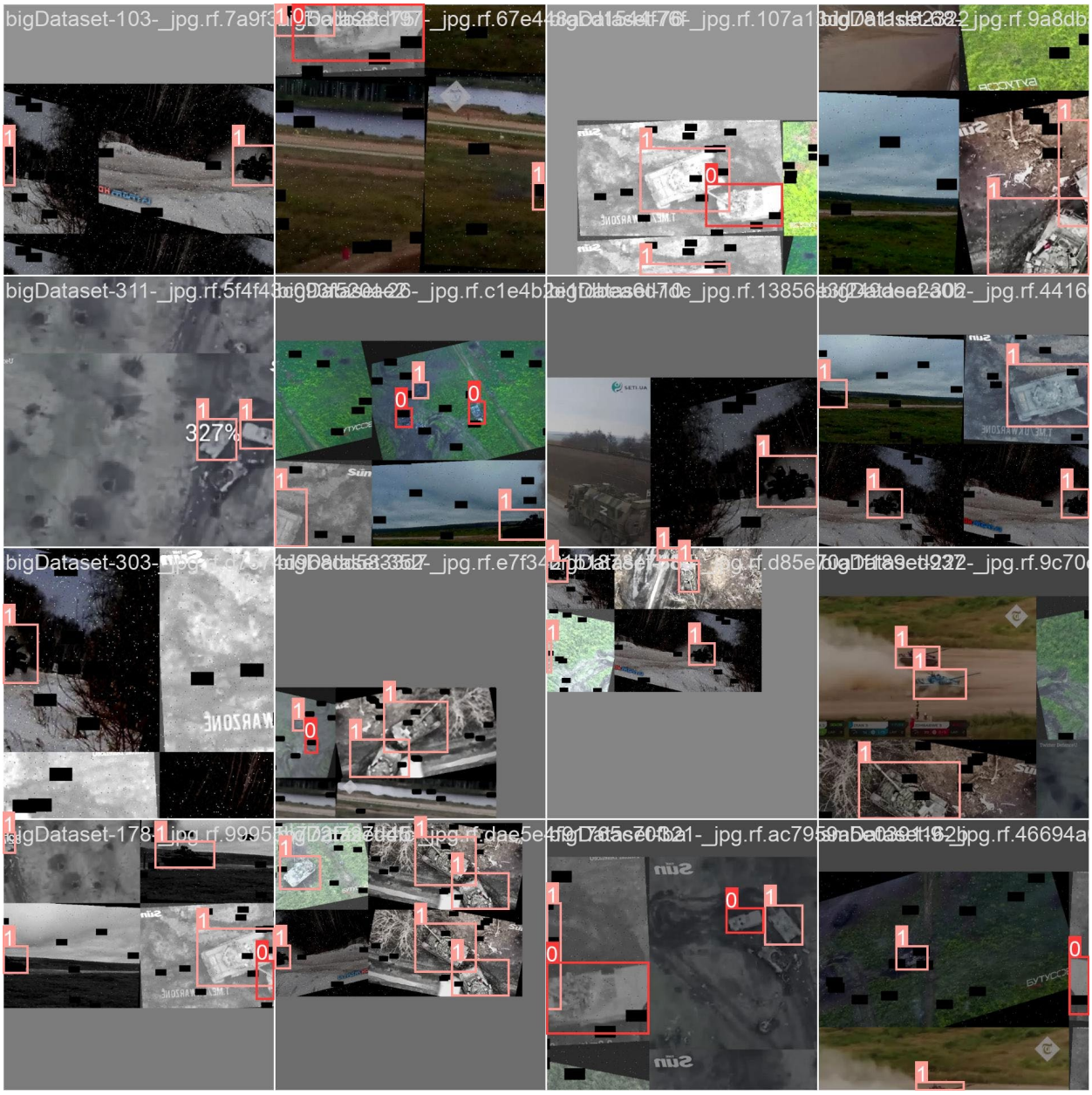

Рисунок 2.2 - Приклад аугментованих та розмічених зображень, де 0, 1 – це класи об'єкту

2.3 Вибір моделі для навчання

Навчання моделі здійснюється відповідно до принципів Transfer Learning. Використовується YOLOv8, побудована на основі фреймворку PyTorch, і надає можливість вибору з 5 моделей, що відрізняються розміром, швидкістю та точністю (рисунок 2.3).

Серед цих моделей було вибрано YOLOv8s, яка має оптимальні характеристики точності та швидкодії.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Рисунок 2.3 - Таблиця характеристик моделей YOLOv8

Архітектура YOLOv8 розроблена з трьох ключових компонентів: 'спини', 'шиї' та 'голови', що разом формують сильну та гнучку структуру для ефективного виявлення та класифікації об'єктів на зображеннях. YOLOv8 аналізує зображення через різні рівні абстракції, об'єднує особливості з різних рівнів для точного виявлення об'єктів, та використовує функції втрат для виправлення помилок і поліпшення здатності виявлення та класифікації (рисунок 2.4).

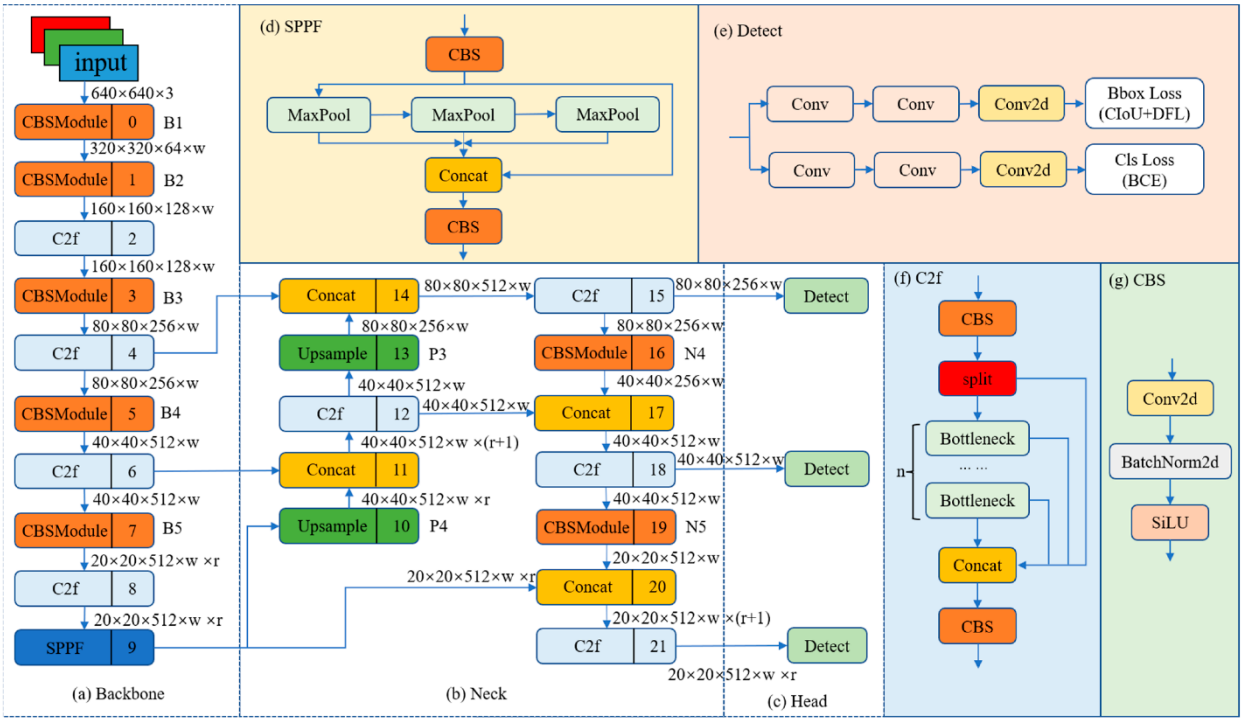

Рисунок 2.4 - Структура YOLOv8

Спина (backbone) YOLOv8 є основою, на якій будується весь процес виявлення. Вона складається з вхідних зображень розміром 640x640 пікселів, зазвичай у форматі RGB, та серії конволюційних блоків (CBSModules). Ці блоки виконують важливу роль у фільтрації та абстрагуванні особливостей зображення на різних рівнях, забезпечуючи збереження ключових особливостей при зменшенні розмірності. Крім того, спина включає спатіальний пірамідальний блок пулінгу (SPPF), який поліпшує здатність мережі розпізнавати об'єкти різних масштабів.

Шия (neck) мережі YOLOv8 забезпечує інтеграцію та уточнення особливостей, отриманих від спини. Вона складається з набору операторів, які включають об'єднання (Concat), збільшення розміру (Upsample) та додаткові CBSModules. Ця частина відіграє ключову роль у поєднанні особливостей з різних рівнів спини для покращення точності виявлення об'єктів різних розмірів.

Голова (head) є вирішальним компонентом YOLOv8, відповідальним за остаточне визначення класу та положення об'єктів. Вона використовує набуті особливості для передбачення обмежувальних рамок та класів об'єктів. Ключові елементи голови включають блоки Detect, які займаються класифікацією та

локалізацією, а також функції втрат Bbox Loss та Cls Loss для вимірювання розбіжностей між передбаченнями та фактичними даними, що сприяє поліпшенню тренування моделі.

Додаткові компоненти архітектури, такі як Cross stage partial connections (C2f) та CBS (Convolution, Batch normalization, SiLU activation), істотно покращують здатність мережі до ефективної обробки та аналізу зображень.

У рамках боротьби з перенавчанням, YOLOv8 впроваджує кілька механізмів. Це включає Ранню Зупинку (Early Stopping), що дозволяє припинити процес навчання у разі відсутності прогресу на валідаційному наборі, а також постійний моніторинг втрат та точності на тренувальних та валідаційних наборах. Регуляризація, дропаут та аугментація даних також є важливими інструментами для запобігання перенавчання.

2.4 Напівавтоматична мануальна розмітка

Суть напівавтоматичної розмітки полягає в тому, що замість ручного розмічення всього датасету, який може складатися з тисяч зображень і бути часом витратним і не дуже гнучким, створюються дві вибірки з розподілом зображень у співвідношенні 20% до 80%. Менший з цих наборів розмічається вручну і використовується для навчання моделі. Після цього, ця модель автоматично розмічує решту датасету.

В малу вибірку потрапили файли, які були вибрані вручну, по критеріях найбільшої репрезентативності та різноманітності, решта – залишились у великій.

Ручна розмітка виконувалась за допомогою LabelImg (рисунок 2.5).

Рисунок 2.5 - Приклад розмітки за допомогою LabelImg

Початок навчання моделі на малій вибірці:

```
!nvidia-smi
%pip install ultralytics
import ultralytics
ultralytics.checks()

from ultralytics import YOLO
import os
from IPython.display import display, Image
from IPython import display
display.clear_output()

!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="**********")
project = rf.workspace("nazar-ixgs8").project("c-czkft")
dataset = project.version(1).download("yolov8")
```

!yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml epochs=30 imgsz=640

Model summary: 225 layers, 11136374 parameters, 11136358 gradients, 28.6 GFLOPs

!yolo task=detect mode=val model=/content/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

Розмітка великої вибірки за допомогою моделі навченої на малій вибірці:

```
import cv2
import os
from pathlib import Path
from ultralytics import YOLO
model = YOLO('/content/runs/detect/train/weights/best.pt')
def detectAndSaveTxt(filename, output_folder):
    img = cv2.imread(filename)
    results = model(img)

    # Перевірка наявності детекції об'єкту
    if len(results) == 0 or not results[0].boxes or len(results[0].boxes.xyxy[0]) == 0:
        print(f"No detections for {filename}")
        return

    # Створення шляху до файлу для збереження результатів
    txt_filename = Path(output_folder) / (Path(filename).stem + '.txt')
    results[0].save_txt(txt_filename)

# Папка з зображеннями
```

```
source_folder = '/content/rawDataset'

# Папка для зберігання результатів
output_folder = '/content/annotated'
os.makedirs(output_folder, exist_ok=True)

# Обробка кожного зображення у папці
for filename in os.listdir(source_folder):
    filepath = os.path.join(source_folder, filename)
    if os.path.isfile(filepath) and filepath.lower().endswith(('.png', '.jpg', '.jpeg')):
        detectAndSaveTxt(filepath, output_folder)
```

Результати розмітки представлені на рисунку 2.6.

Рисунок 2.6 - Колаж розмітки великої вибірки, моделлю навченій на малій вибірці

З наведених зображень, можемо зробити висновок, що модель добре розпізнає різницю в класах (наприклад на 9, 11 зображеннях), не захоплює схожі об’єкти, такі як авто або вантажівки (на 1, 2 зображеннях), але в той же час подекуди пропускає об’єкти для детекції (12 зображення), інколи накладає 2 рамки на один об’єкт (5, 7 зображення), або невірно інтерпретує місцевість як об’єкт класу (3 зображення), внаслідок невеликої вибірки зображень для навчання.

2.5 Формування синтетичного датасету з автоматичною розміткою за допомогою ПЗ Blender

Формування синтетичного датасету дозволяє:

- Створювати різноманітні тренувальні випадки без потреби у фізичному зборі даних, який часто є часозатратним і може бути обмежений.
- Економити час та ресурси з можливістю швидко створювати великі датасети без необхідності ручної розмітки.
- Точно контролювати умови освітлення, перспективу, фон та інші аспекти сцени, що дає можливість створювати датасети, які враховують певні умови використання нейронних мереж.

Генерація синтетичного датасету проводилася з використанням програмного забезпечення Blender. Це відкрите програмне забезпечення, яке надає широкий спектр інструментів для 3D моделювання, анімації, рендерингу, пост-обробки та візуалізації. Його важливою особливістю є підтримка скриптів на Python, що дало можливість автоматизувати процес розмітки та рендерингу кадрів. Це значно скоротило час, необхідний для підготовки та розмітки великого обсягу зображень.

Алгоритм рендерингу та розмітки об'єктів у Blender був розроблений з метою автоматизації процесу генерації даних для навчання нейронної мережі. Основна ідея полягає у створенні різноманітних 3D сцен, де об'єкти розміщуються в різних конфігураціях і під різними кутами, а потім виконується їх автоматизована розмітка.

Процес розмітки полягав в наступному:

1. Спочатку в Blender створюється 3D сцена, в якій розміщуються об'єкти, що підлягають розмітці. 3D моделі для цього було взято з відкритих джерел.

2. Налаштовується початкове положення камери: кут нахилу, відстань, лінза. Камера прив’язується до об’єкту розмітки.

3. Виконується рендеринг сцени для отримання фінальних зображень. Кожне зображення відображає унікальну конфігурацію сцени та об'єктів.

4. Для кожного зображення виконується автоматизована розмітка. Координати зберігаються разом з зображенням у форматі, придатному для навчання YOLOv8.

Для досягнення високої точності розмітки, оскільки будь-які помилки могли має негативний вплив на навчання моделі, була використана тактика створення невидимих кубів. Ця потреба виникла через те, що форма об'єкта не надавала можливості використовувати її як безпосередню основу для рамки розмітки. Дані невидимі куби були створені таким чином, щоб максимально точно відповідати формі об'єкта і гарантувати найкращу можливу апроксимацію та точність в процесі розмітки (рисунок 2.7).

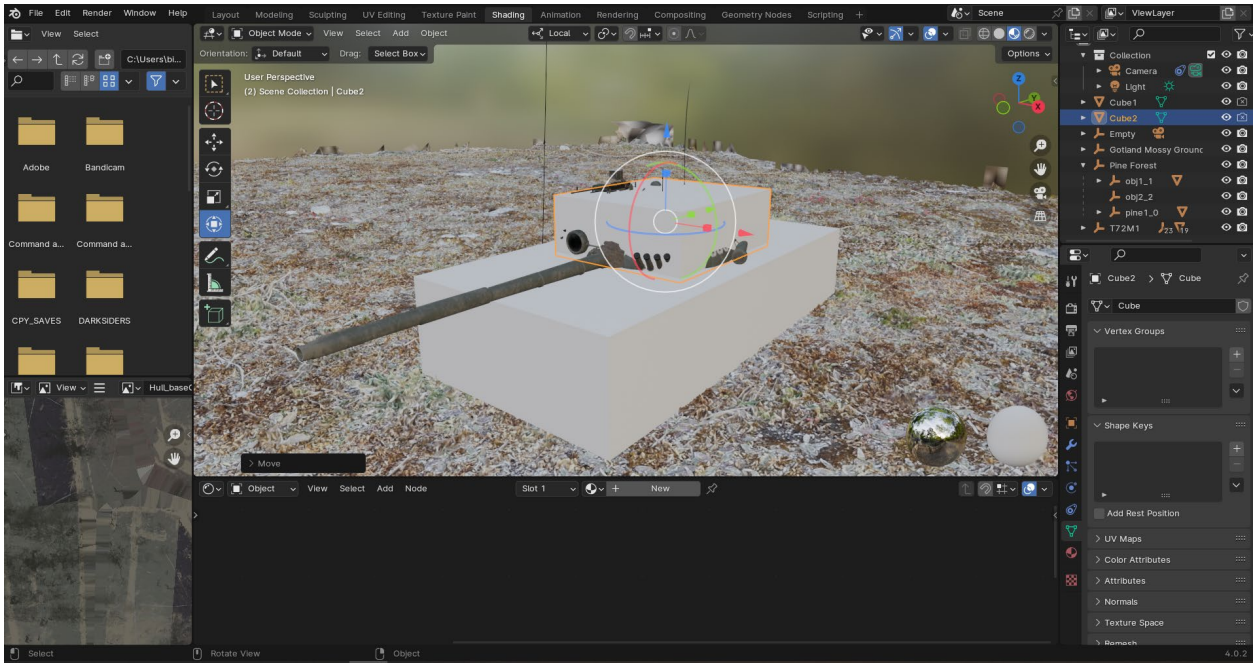

Рисунок 2.7 - Створення невидимих кубів в Blender

Ще одним важливим аспектом було створення достатньої різноманітності сцен і об'єктів, щоб датасет відображав наближений до реального спектр умов, з якими могла б стикатися нейронна мережа. Це вимагало творчого підходу до дизайну сцен, а також ретельного вибору об'єктів і налаштувань камери (рисунок 2.8).

Оскільки процес рендерингу і розмітки був часомістким, існувала необхідність його оптимізації. Рішення полягало у підборі оптимальних параметрів рендерингу для прискорення процесу без значної втрати якості.

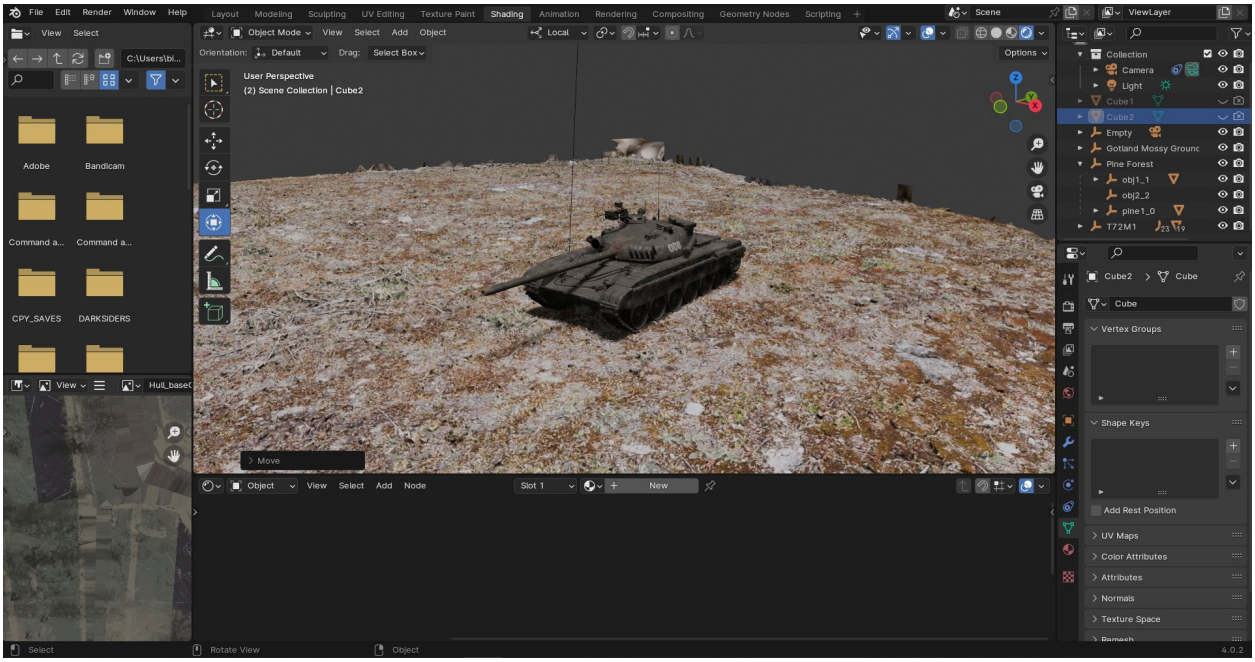

Рисунок 2.8 - 3D сцена з моделлю в Blender

Python код, для автоматизації рендерингу та автоматичнгої розмітки датасету:

```
import bpy
import math
import bpy_extras
from mathutils import Vector

def convert_to_camera_space(cam, obj):
    scene = bpy.context.scene
```

```
    mat_world = obj.matrix_world
    bbox_corners = [mat_world @ Vector(corner) for corner in obj.bound_box]
    bbox_coords_2d = [bpy_extras.object_utils.world_to_camera_view(scene, cam, corner) for corner in bbox_corners]
    return bbox_coords_2d

def get_normalized_bbox(bbox_coords_2d):
    # Отримання 2D границі рамки в нормалізованих координатах
    min_x = min(coord.x for coord in bbox_coords_2d)
    max_x = max(coord.x for coord in bbox_coords_2d)
    min_y = min(coord.y for coord in bbox_coords_2d)
    max_y = max(coord.y for coord in bbox_coords_2d)

    # Обчислення центру та розмірів границі рамки
    bbox_x = (min_x + max_x) / 2
    bbox_y = (min_y + max_y) / 2
    bbox_width = max_x – min_x
    bbox_height = max_y – min_y

    # Обернення координати y, щоб відповідати системі координат зображення
    bbox_y = 1 – bbox_y

    return [bbox_x, bbox_y, bbox_width, bbox_height]

# Параметри
horizontal_angle_step = 90
vertical_angle_step = 90
output_path = "C:\\Users\\bilin\\Downloads\\AI\\syntheticGeneration\\"
annotations_path = "C:\\Users\\bilin\\Downloads\\AI\\syntheticGeneration\\"
```

```
# Встановлення порожнього об'єкта як батьківського для камери
cam = bpy.context.scene.camera
empty = bpy.data.objects.get("Empty")
if empty is None:
    raise ValueError("Порожній об'єкт не був знайдений")
cam.parent = empty

# Назви об'єктів-цілей для обчислення границі рамки
target_object_names = ["Cube1", "Cube2"]  # Замініть на ваші фактичні назви об'єктів

# Пошук об'єктів-цілей
target_objects = [bpy.data.objects.get(name) for name in target_object_names]
if any(obj is None for obj in target_objects):
    raise ValueError("Один або кілька об'єктів-цілей не були знайдені")

for vertical_angle in range(0, 91, vertical_angle_step):
    for horizontal_angle in range(0, 360, horizontal_angle_step):
        # Обертання порожнього об'єкта до бажаного кута
        empty.rotation_euler[0] = math.radians(vertical_angle)
        empty.rotation_euler[2] = math.radians(horizontal_angle)
        bpy.context.view_layer.update()

        # Рендеринг зображення
        file_name = f"render_{vertical_angle:03d}_{horizontal_angle:03d}.png"
        bpy.context.scene.render.filepath = output_path + file_name
        bpy.ops.render.render(write_still=True)

        # Обчислення рамки для всіх об'єктів-цілей
        combined_bbox_coords_2d = []
        for obj in target_objects:
```

```
        combined_bbox_coords_2d.extend(convert_to_camera_space(cam, obj))

    # Отримання нормалізованої рамки
    normalized_bbox = get_normalized_bbox(combined_bbox_coords_2d)

    # Запис розмітки у файл
    annotation_file = f"render_{vertical_angle:03d}_{horizontal_angle:03d}.txt"
    with open(annotations_path + annotation_file, 'w') as file:
        file.write(f'0 {normalized_bbox[0]} {normalized_bbox[1]} {normalized_bbox[2]} {normalized_bbox[3]}\n')
```

Одне зі згенерованих зображень розмічених цим способом представлено на рисунку 2.9.

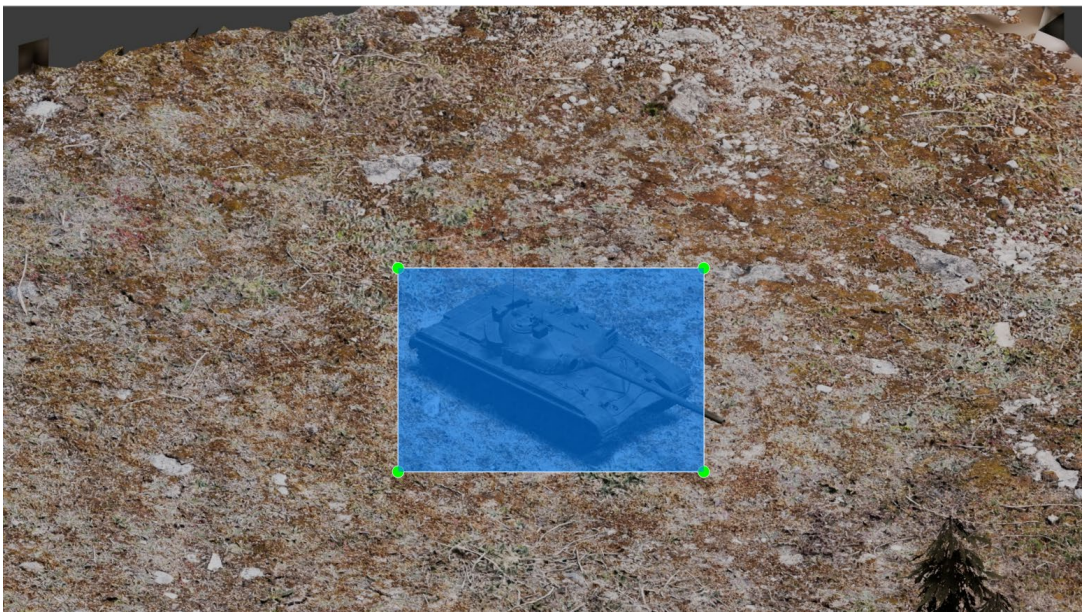

Рисунок 2.9 - Згенероване та розмічене зображення за допомогою скрипта в Blender

Цей датасет призначений для розширення датасету реальних зображень, оскільки він містить всі можливі проекції моделі з різними характеристиками. Однак через певні специфічні патерни, модель, навчена на ньому, може бути упереджена до реальних даних. Для подальшого покращення цього датасету можна

використовувати різноманітні моделі цільового об'єкта та його оточення, що вимагає значних зусиль і ресурсів.

2.6 Розмітка за допомогою великої базової моделі Grounded SAM.

Великі базові моделі виявляються дуже корисними під час розмітки зображень. Вони мають кілька переваг, які допомагають у створенні навчальних датасетів для спеціалізованих моделей. Завдяки своїм обширним навчальним даним і вдосконаленим алгоритмам, ці моделі забезпечують високу точність і якість розмітки, що є критично важливими аспектами.

Однією з ключових особливостей цих моделей є їх глибоке розуміння контексту, що дозволяє їм виявляти складні зв'язки на зображеннях. Це формує більш складні і інформативні мітки, що стає важливим для подальшого навчання моделей, особливо в ситуаціях, де важлива висока точність.

Слід також відзначити, що автоматизація процесу розмітки відкриває можливість значного заощадження часу та ресурсів порівняно з ручною розміткою, роблячи процес більш ефективним та доступним. Зводячи участь людини лише до перевірки та корегування результатів розмітки.

Проте, не дивлячись на всі ці переваги, великі базові моделі не підходять для безпосереднього використання у спеціалізованих завданнях з обмеженими обчислювальними ресурсами, що робить їх неефективними для сценаріїв, де потрібна швидка реакція.

Базові моделі також не завжди оптимізовані для конкретних завдань, таких як детекція або розпізнавання. У таких випадках спеціалізовані менші моделі, які можна налаштувати під конкретні вимоги та умови, будуть показувати кращі результати. Їх швидкість обробки даних набагато вища, що є критично важливим для сценаріїв, де потрібна швидка реакція в режимі реального часу.

Для розмітки, використовувався фреймворк Autodistill разом з базовою моделлю Grounded SAM. Він автоматизує деякі аспекти процесу машинного навчання та інтегрується з існуючими моделями, покращуючи їхні можливості.

```
!nvidia-smi
!pip install -q \
autodistill \
autodistill-grounded-sam \
autodistill-yolov8 \
supervision==0.9.0
!pip install roboflow
!pip install matplotlib-venn
!apt-get -qq install -y libfluidsynth1
# https://pypi.python.org/pypi/pydot
!apt-get -qq install -y graphviz && pip install pydot
import pydot
!pip install cartopy
import cartopy

import os
HOME = os.getcwd()
print(HOME)

!mkdir {HOME}/images

import supervision as sv

#Підрахунок зображень
image_paths = sv.list_files_with_extensions(
    directory= "/content/images",
    extensions=["png", "jpg", "jpg"])

print('image count:', len(image_paths))
```

```
IMAGE_DIR_PATH = f"{HOME}/images"
SAMPLE_SIZE = 16
SAMPLE_GRID_SIZE = (4, 4)
SAMPLE_PLOT_SIZE = (16, 16)

import cv2
import supervision as sv

titles = [
    image_path.stem
    for image_path
    in image_paths[:SAMPLE_SIZE]]
images = [
    cv2.imread(str(image_path))
    for image_path
    in image_paths[:SAMPLE_SIZE]]

sv.plot_images_grid(images=images, titles=titles,
grid_size=SAMPLE_GRID_SIZE, size=SAMPLE_PLOT_SIZE)

#Встановлення класів
from autodistill.detection import CaptionOntology

ontology=CaptionOntology({
    "tank": "Tank",
    "armoured_personnel_carrier": "APC"
})

DATASET_DIR_PATH = f"{HOME}/dataset"
```

```
from autodistill_grounded_sam import GroundedSAM

#Запуск базової моделі
base_model = GroundedSAM(ontology=ontology)
dataset = base_model.label(
    input_folder=IMAGE_DIR_PATH,
    extension=".jpg",
    output_folder=DATASET_DIR_PATH)

ANNOTATIONS_DIRECTORY_PATH = f"{HOME}/dataset/train/labels"
IMAGES_DIRECTORY_PATH = f"{HOME}/dataset/train/images"
DATA_YAML_PATH = f"{HOME}/dataset/data.yaml"

import supervision as sv

dataset = sv.DetectionDataset.from_yolo(
    images_directory_path=IMAGES_DIRECTORY_PATH,
    annotations_directory_path=ANNOTATIONS_DIRECTORY_PATH,
    data_yaml_path=DATA_YAML_PATH)

len(dataset)

#Перебір зображень та їх анотація:
import supervision as sv

image_names = list(dataset.images.keys())[:SAMPLE_SIZE]

box_annotator = sv.BoxAnnotator()

images = []
```

```
for image_name in image_names:
    image = dataset.images[image_name]
    annotations = dataset.annotations[image_name]
    labels = [dataset.classes[class_id] for class_id in annotations.class_id]

    annotates_image = box_annotator.annotate(
        scene=image.copy(),
        detections=annotations,
        labels=labels)
    images.append(annotates_image)
sv.plot_images_grid(
    images=images,
    titles=image_names,
    grid_size=SAMPLE_GRID_SIZE,
    size=SAMPLE_PLOT_SIZE)
```

Результати виявлення об’єктів представлені на рисунку 2.10.

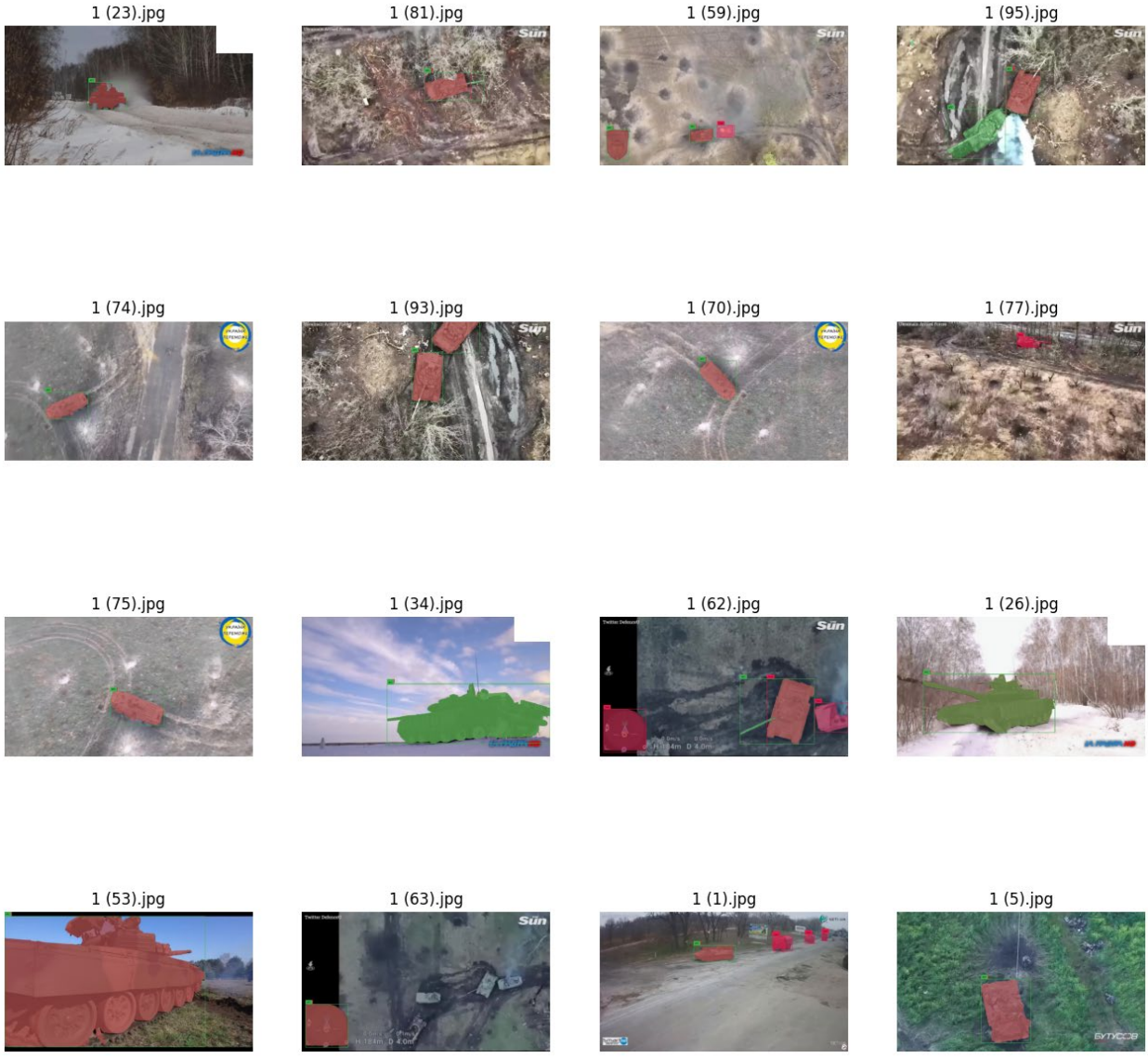

Рисунок 2.10 - Результати виявлення об’єктів базовою моделлю Grounded SAM на колажі

Якщо взяти для розгляду 16 випадкових розмічених зображень, можемо помітити, що 1 з них було розмічено невірно, інші 4 невірно в межах класу, але слід зазначити що для деякої техніки ця границя умовна, лише одну з них – вантажівку, модель помилково розмітила як БТР. З цього можна зробити висновок, що для первинної розмітки класів особливо які між собою відрізняються, цей спосіб показує хороші результати, деякі проблеми можуть виникнути, коли на зображенні зустрічаються декілька схожих об’єктів, які потрібно розмістити в різні класи.

3. АНАЛІЗ ЕФЕКТИВНОСТІ ТА РЕАЛІЗАЦІЯ АЛГОРИТМУ ДЛЯ НАВІГАЦІЇ БПЛА

3.1 Порівняння результатів навчання

Для скорочення, будемо вважати модель навчену лише на реальних даних (за допомогою напівавтоматичної розмітки) першою моделлю, а модель навчену на комбінованому датасеті (реальні дані + синтетичні) – другою.

На графіках (рисунок 3.1, 3.2) вказані криві залежності F1 оцінки від впевненості, для двох класів об'єктів – APC та Tank.

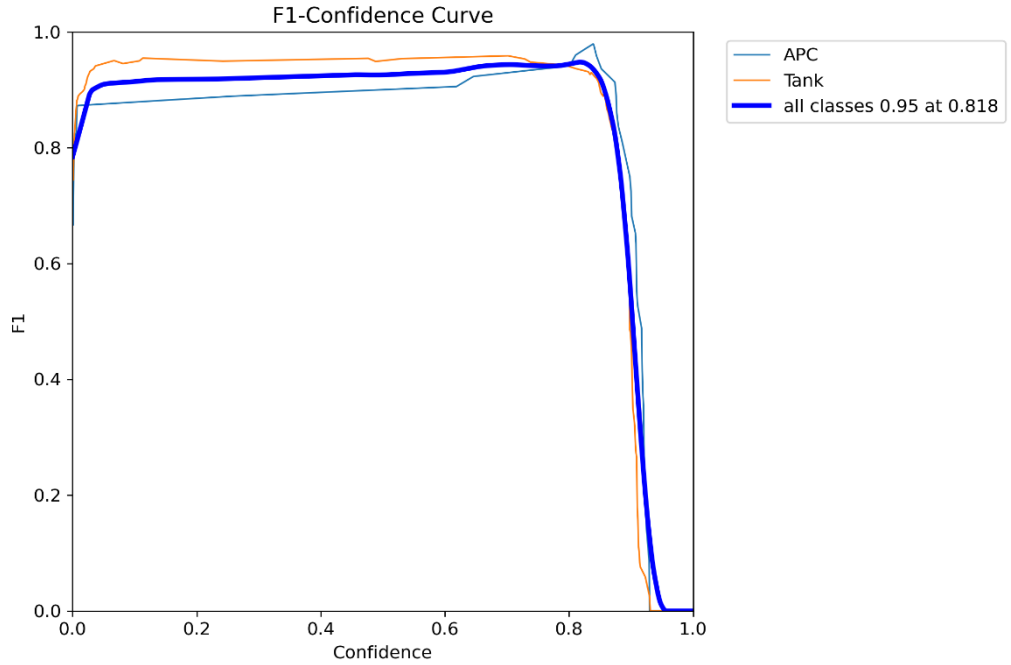

Рисунок 3.1 - Графік F1 впевненості для першої моделі

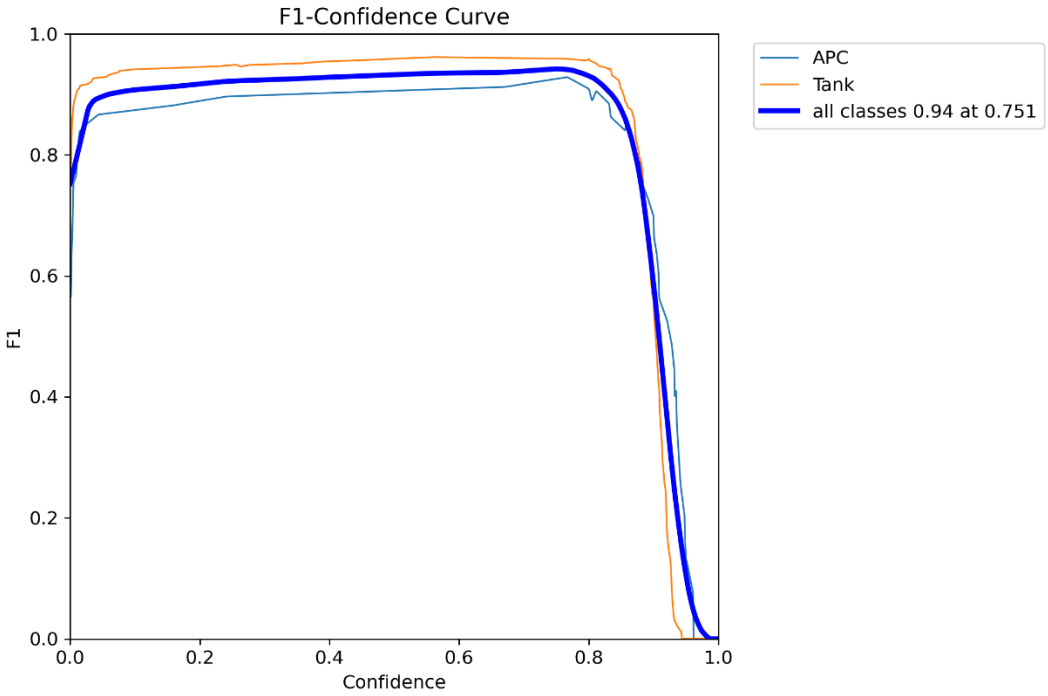

Рисунок 3.2 - Графік F1 впевненості для другої моделі

Перший графік (модель тренована тільки на реальному датасеті) показує високу F1-оцінку, яка досягає плато біля позначки впевненості 0.8 та тримається стабільно до самого кінця. Це вказує на те, що модель добре розпізнає класи "APC" та "Tank", маючи високу впевненість у своїх передбаченнях. Агрегована крива для всіх класів (синій колір) також показує високу F1-оцінку, досягаючи 0.95 при впевненості 0.818.

Другий графік (модель тренована на комбінованому датасеті з реальними та синтетичними даними) має дуже схожу форму, але можна помітити, що агрегована F1-оцінка для всіх класів трохи нижча (0.94 при конфіденційності 0.751). Це може свідчити про те, що додавання синтетичних даних дозволило моделі бути трохи більш загальною, але при цьому можливо знизило точність у певних ситуаціях.

Загалом, обидві моделі показують високі F1-оцінки.

На наступних графіках (рисунок 3.3, 3.4) представлені дві криві точності-повнота (Precision-Recall curves) для класів "APC" та "Tank", а також середнє значення точності (mAP) для всіх класів при порозі 0.5. Криві точності-повнота

важливі для оцінювання якості класифікаторів, особливо в задачах з незбалансованими класами, де кількість прикладів одного класу значно переважає інші.

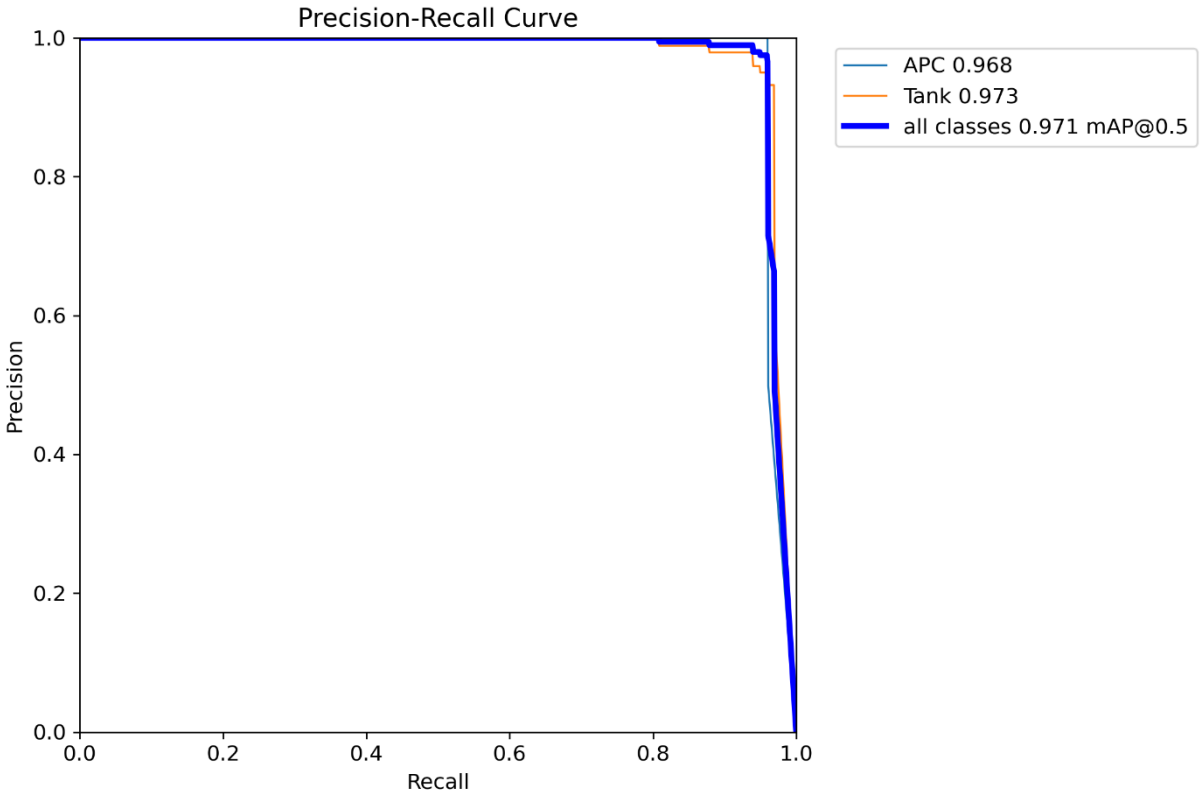

Рисунок 3.3 - Графік Precision-Recall для першої моделі

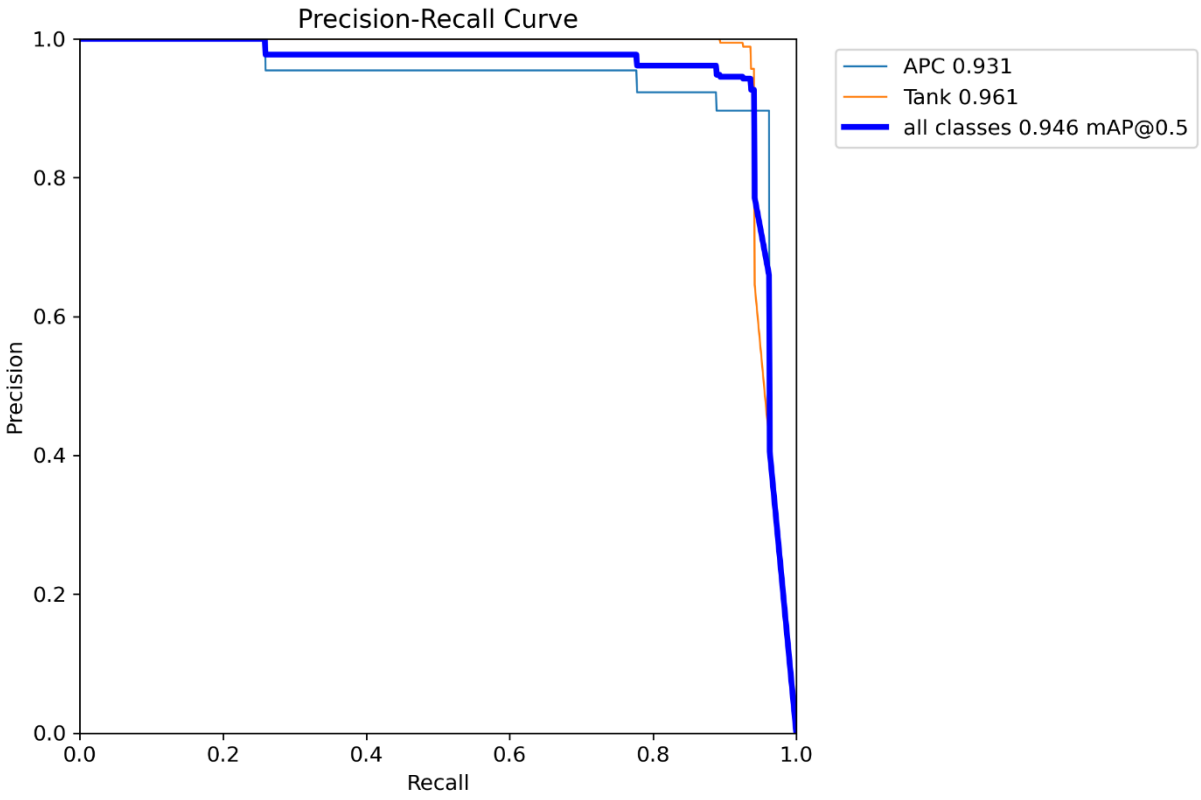

Рисунок 3.4 - Графік Precision-Recall для другої моделі

У першому випадку, де модель була навчена на основі лише реальних даних, ми бачимо дуже високі значення точності для обох класів, що наближаються до 1, з середньою точністю для всіх класів (mAP) 0.971. Це вказує на те, що модель дуже добре розпізнає і правильно класифікує об'єкти обох класів.

У другому випадку, де модель була навчена з використанням як реальних, так і синтетичних даних, точність для обох класів трохи нижча, з mAP 0.946. Це може свідчити про те, що додавання синтетичних даних дало змогу моделі бачити більше варіативності, але це також могло внести деякий шум, що призвело до зниження точності.

Порівнюючи два графіка, можна зробити висновок, що обидві моделі показують хороші результати.

На зображеннях нижче (рисунок 3.5, 3.6) представлені дві нормалізовані матриці помилок для моделей. Нормалізація у матриці помилок означає, що значення представлені у формі пропорцій, що дозволяє легше порівняти продуктивність моделей між собою.

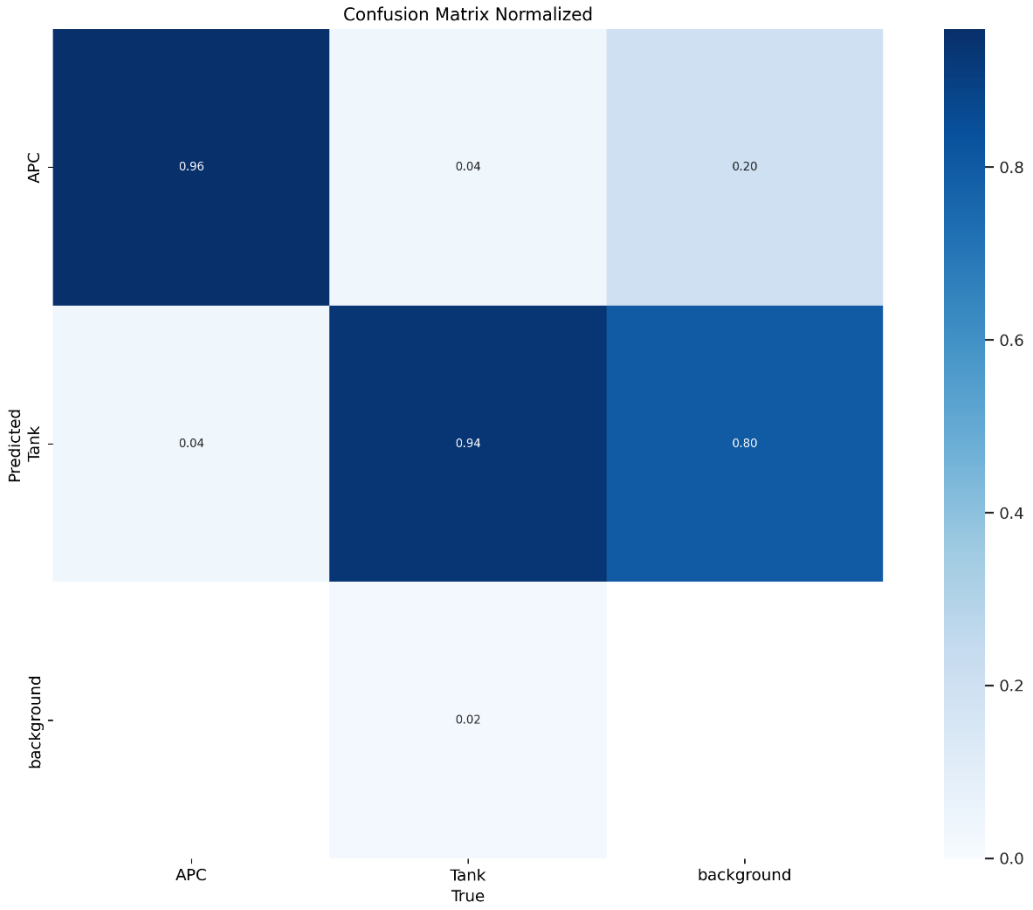

Рисунок 3.5 - Нормалізована матриця помилок для першої моделі

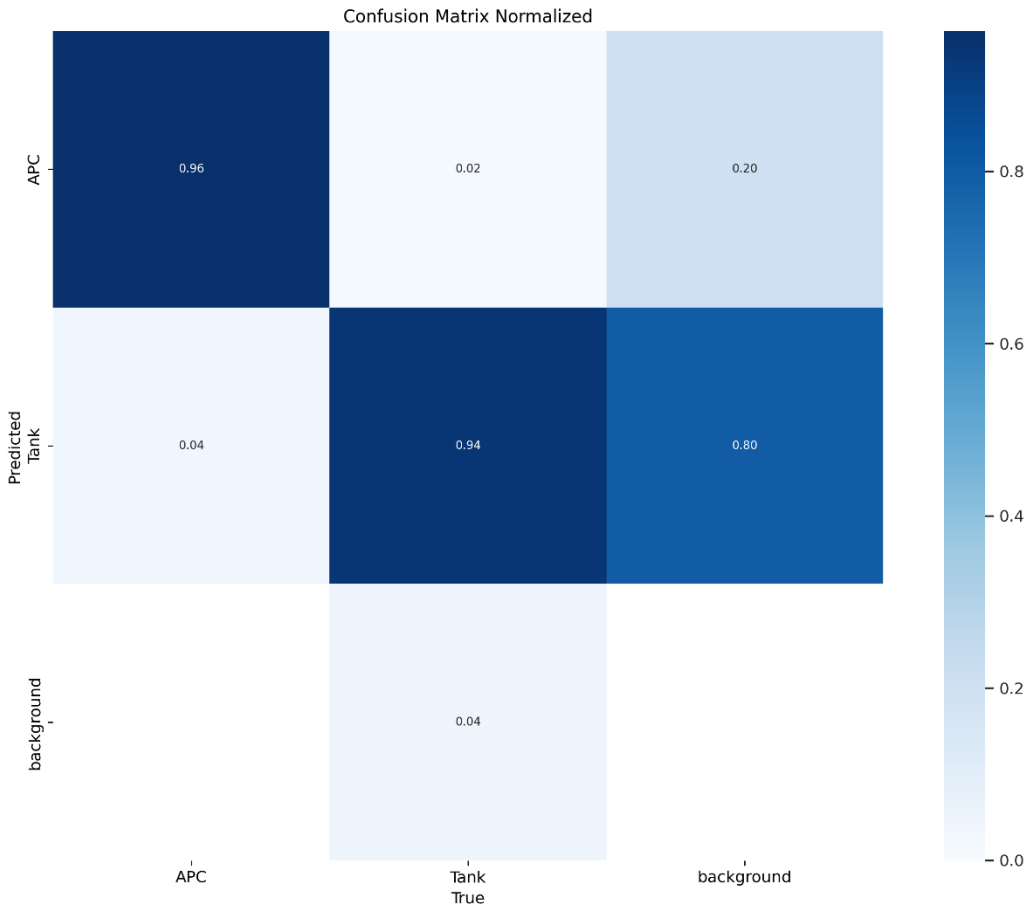

Рисунок 3.6 - Нормалізована матриця помилок для другої моделі

Перша матриця помилок, яка відповідає моделі, навченій тільки на реальному датасеті, показує високі значення діагоналі для класів "APC" та "Tank" (0.96 та 0.94 відповідно), що означає високу точність класифікації цих класів. Також помітно, що модель рідко плутає "APC" з "Tank" і навпаки (0.04 в обох випадках). Проте, з "background" ситуація трохи інша, оскільки модель частіше помилково класифікує "background" як "Tank".

Друга матриця помилок, для моделі, навченої на комбінованому датасеті (реальні + синтетичні дані), також показує високу точність для "APC" та "Tank" (знову ж 0.96 та 0.94), і зниження плутанини між цими класами (0.04 порівняно з 0.20 для "APC" у першій моделі). З класифікацією "background" такі ж помилки.

Порівнюючи дві матриці, можна зробити висновок, що внесення синтетичних даних допомогло зменшити плутанину між класами "APC" та "Tank". В обох випадках можна спостерігати досить високу точність класифікації для "APC" та "Tank", що свідчить про ефективність обох навчальних датасетів.

На наступних графіках представлені результати навчання обох моделей (рисунок 3.7, 3.8).

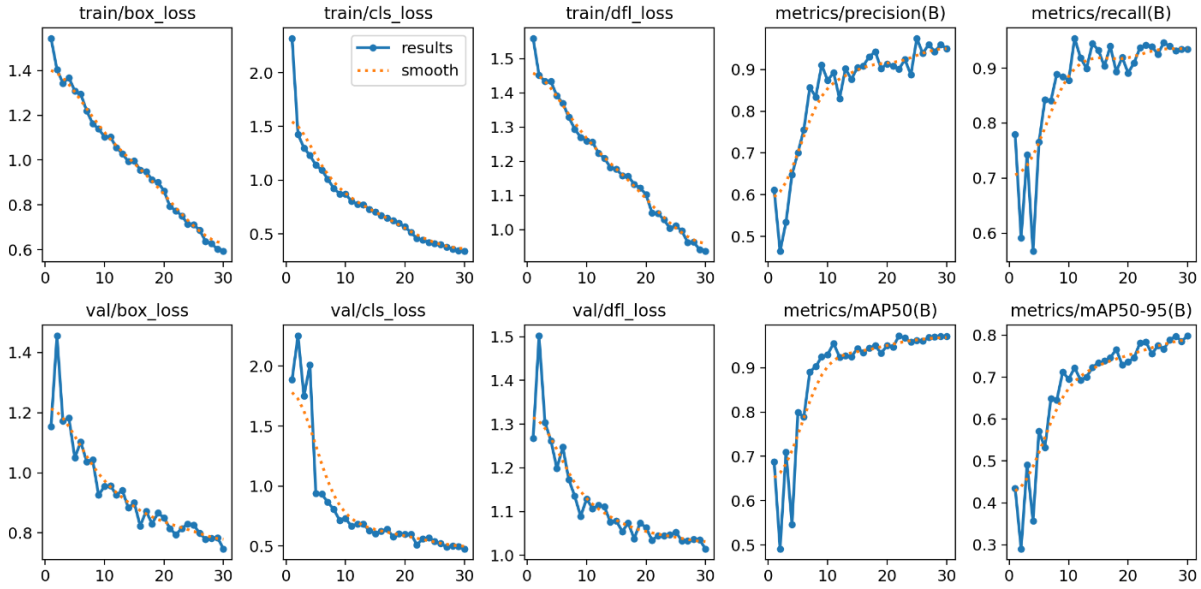

Рисунок 3.7 - Ключові метрики навчання для першої моделі

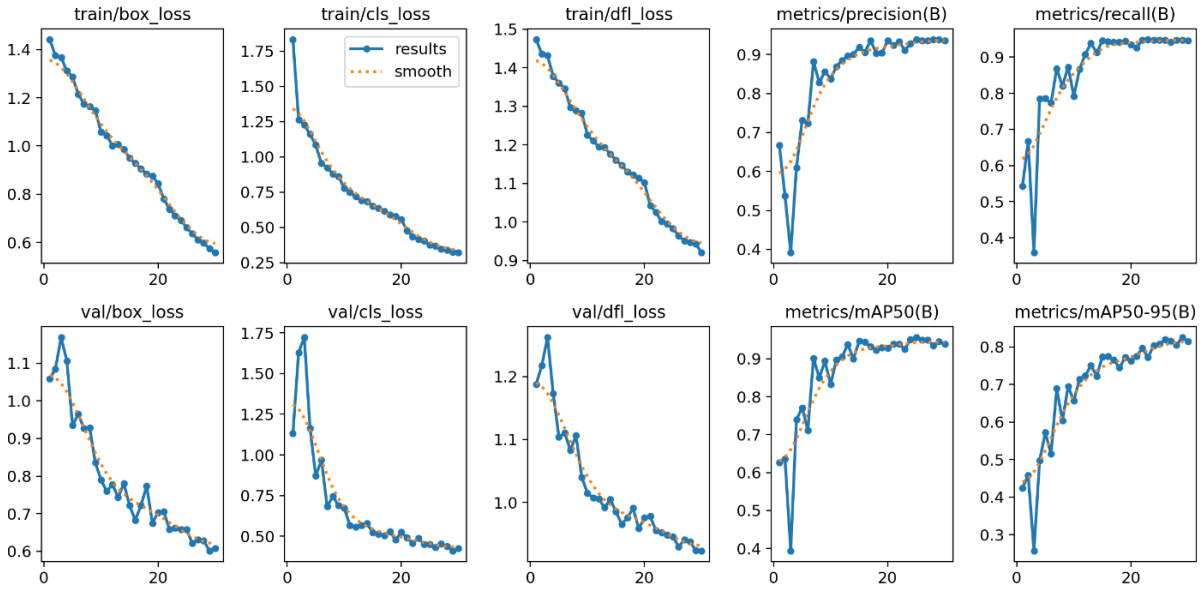

Рисунок 3.8 - Ключові метрики навчання для другої моделі

В обох випадках втрати знижуються з тренуванням, що є хорошим знаком. Однак, модель, навчена на комбінованих даних, показує трохи нижчі втрати на валідації (val/box_loss, val/cls_loss, val/dfl_loss), що може вказувати на кращу узагальнюючу здатність.

Точність та повнота (Precision & Recall) для обох моделей показують позитивну динаміку, але знову ж таки, модель з синтетичними даними демонструє трохи кращі результати на валідації. Особливо це помітно на графіку metrics/recall(B), де друга модель має менший розкид значень, що свідчить про більшу стабільність.

Щодо метрик mAP50 та mAP50-95, модель з синтетичними даними знову показує кращі результати, особливо це видно на початкових етапах тренування. Це може свідчити про те, що синтетичні дані допомогли моделі швидше адаптуватись до різноманітності задач.

Можемо зробити висновок, що модель, навчена з використанням синтетичних даних, показує трохи кращу продуктивність за більшістю метрик, особливо на етапі валідації. Це може свідчити про те, що синтетичні дані допомагають моделі краще узагальнювати та бути менш схильною до перенавчання. Також, це може вказувати на те, що синтетичні дані надають моделі додаткові приклади для тренування, які можуть бути не представлені в реальному датасеті.

3.2 Перевірка двох моделей на тестовому датасеті.

Результати порівняння розпізнавання об'єктів двох моделей, на тестовому наборі зображень, який не використовувався для навчання моделей. Перша модель була навчена виключно на реальних даних і розташована ліворуч, тоді як друга модель була навчена на комбінованому наборі даних, що включав як реальні, так і синтетичні дані, і розташована праворуч.

Результати порівняння розпізнавання об'єктів двох моделей на тестовому наборі зображень, який не використовувався для навчання моделей, представлені на рисунках 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15 нижче. Перша модель (ліворуч),

була навчена виключно на реальних даних. Друга модель (праворуч), була навчена на комбінованому наборі даних, що включав як реальні, так і синтетичні дані.

Рисунок 3.9 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

На 3 зображенні перша модель не змогла розпізнати танк, друга модель з цим впоралась з впевненістю 0,78. З рештою зображень моделі впорались приблизно однаково.

Рисунок 3.10 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

На перших двох зображеннях, друга модель визначила танк більш впевнено, в той час як перша помітила об’єкт двома рамками. На двох останніх, навпаки, краще себе показала перша модель, яка позначила об’єкт з більшою впевненість та змогла розпізнати танк на останньому зображенні.

Рисунок 3.11 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

На 3 зображенні, друга модель змогла точніше визначити клас об’єкту, та краще себе показала на останньому зображенні. Обидві моделі відзначили схожість обрізаного об’єкту як на БМП так і на танк, в перевагу останньому.

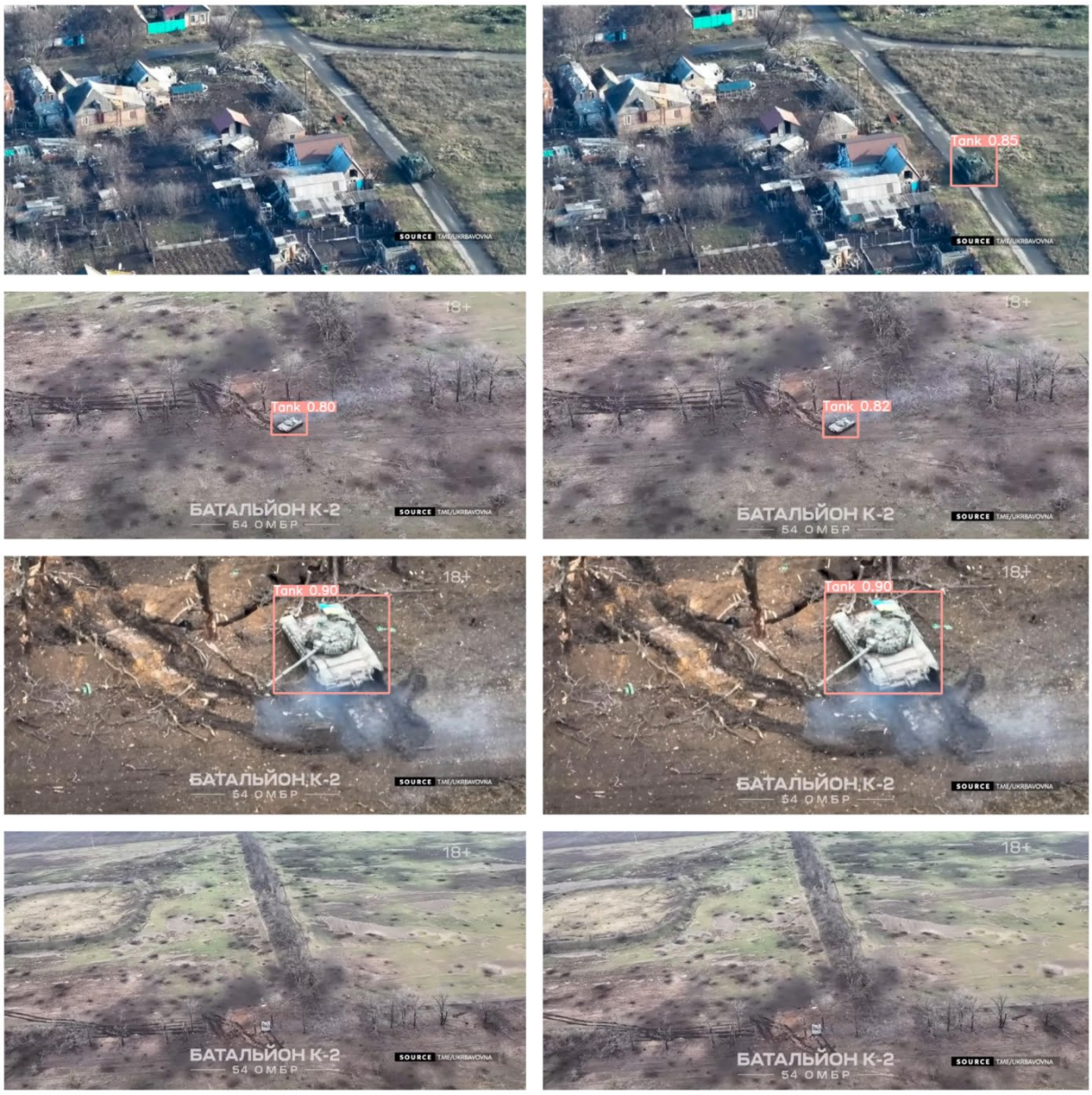

Рисунок 3.12 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

На першому фото, друга модель змогла розпізнати об’єкт з впевненістю 0.85, в той час як перша, його не побачила взагалі. На останньому зображенні, жодна з моделей не змогла розпізнати невеликий об’єкт, схоже, що для таких масштабів, моделі в повній мірі не адаптовані, через невелику кількість схожих зображень в навчальній вибірці.

Рисунок 3.13 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

З невеликою перевагою краще спрацювала друга модель, на останньому фото впевненість 0.78 проти 0.89 відповідно. На другому зображенні через нехарактерний масштаб жодна з моделей також нічого не побачила.

Рисунок 3.14 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

Тут в загальному краще справилась друга модель, зумівши правильно розпізнати клас об’єкту на 3 зображенні. На першому і останньому зображенні є попеременний успіх в обох моделей. На 2, ймовірно через дим, жодна з моделей не виявила об’єкт.

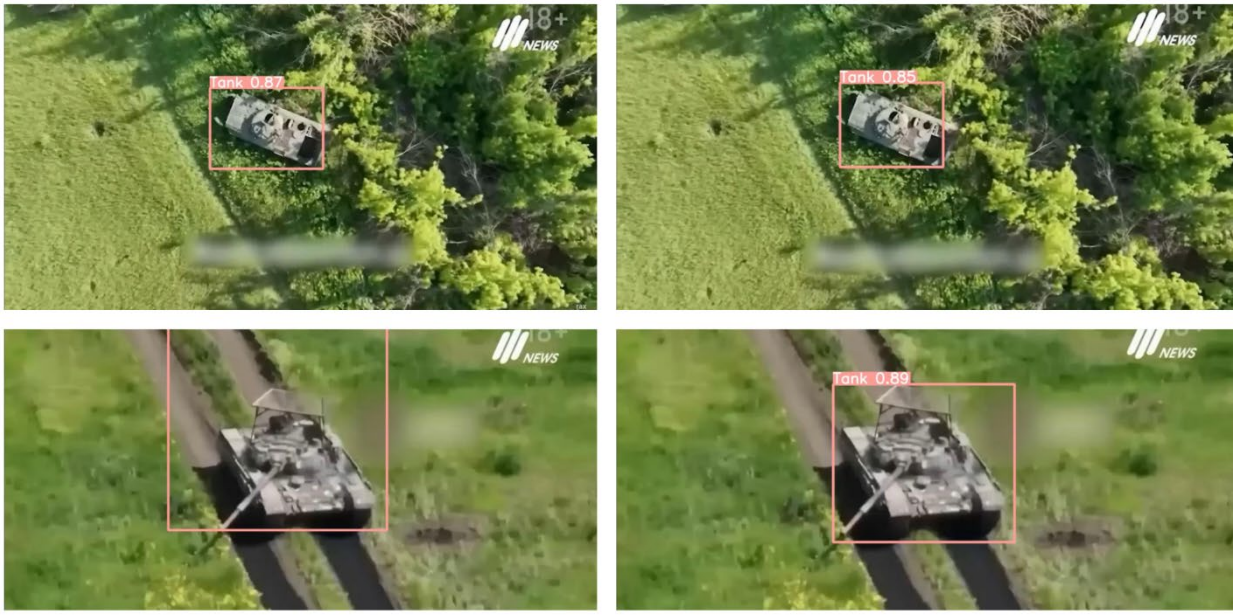

Рисунок 3.15 - Порівняння результатів розпізнавання об'єктів. Перша модель знаходиться ліворуч, друга модель праворуч

В цьому випадку також краще справилась друга модель, зумівши точніше розпізнати об’єкт та зробивши це з більшою впевненістю, 0.69 проти 0.89.

3.3 Процес навчання моделі YOLOv8

В даній роботі, процес навчання моделі YOLOv8 було реалізовано з використанням принципу Transfer Learning, який вважається ефективним підходом в глибокому навчанні. Архітектура моделі YOLOv8 (рисунок 3.16) включає декілька ключових шарів, кожен з яких виконує специфічні функції:

Convolutional Layers (Conv): Ці шари відіграють критичну роль у виявленні характеристик на різних рівнях зображення. Вони дозволяють моделі "вчитися" зі зображення, виявляючи різноманітні візуальні ознаки.

Bottleneck Layers (C2f): Використовуються для зменшення розмірності вхідних даних, що сприяє підвищенню ефективності обчислень, при цьому зберігаючи важливі характеристики об'єктів.

SPPF (Spatial Pyramid Pooling – Fast): Цей шар агрегує характеристики на різних масштабах і забезпечує моделі інваріантність до розміру об'єктів.

Upsample: Шари, що збільшують розмір просторових даних, використовуються для відновлення деталей зображення, які могли бути втрачені в попередніх шарах.

Concat: Ці шари об'єднують характеристики, отримані з різних джерел або шарів, що є важливим для синтезу характеристик на різних рівнях абстракції.

Шар ultralytics.nn.modules.head.Detect: Це фінальний шар, який відповідає за виявлення об'єктів, трансформуючи отримані характеристики у конкретні координати об'єктів, їх класи та рівні впевненості.

Процес навчання моделі було визначено на 30 епох (рисунок 3.16). Вхідні зображення масштабувалися до розміру 640x640 пікселів, з дотриманням пропорцій та заповненням вільного простору фоном або іншими зображеннями, для забезпечення консистентності вхідних даних.

Модель також включала автоматичні аугментації (рисунок 3.17), такі як Blur та MedianBlur, які додають розмиття до зображення; ToGray, що перетворює кольорові зображення в чорно-білі; CLAHE, що забезпечує адаптивне вирівнювання гістограми для підвищення контрастності зображення.

Модель використовує три види втрат - box_loss, cls_loss, і dfl_loss, що дозволяють оцінити помилки моделі у визначенні границь об'єктів (box), класифікації (cls), та дискретному перетворенні Фур'є (dfl).

Ключові показники ефективності моделі - mAP50 та mAP50-95 - це середня точність виявлення об'єктів (mean Average Precision) при порозі IoU (Intersection over Union) 0.5, а також середні значення в діапазоні від 0.5 до 0.95 відповідно. Ці метрики надають цінну інформацію про загальну точність моделі у різних умовах.

Модель (рисунок 3.18) на виході має 168 шарів, 11,126,358 параметрів, і 28.4 GFLOPs обчислювальної потужності.

```
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=30 imgsz=640

Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8s.pt to 'yolov8s.pt'...
100% 21.5M/21.5M [00:00<00:00, 47.5MB/s]
Ultralytics YOLOv8.0.227 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla V100-SXM2-16GB, 16151MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=/content/Combine-1/data.yaml, epochs=30, pa
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 5.50MB/s]
2023-12-19 22:26:32.881055: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to regis
2023-12-19 22:26:32.881103: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to regist
2023-12-19 22:26:32.882394: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to regi
Overriding model.yaml nc=80 with nc=2

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2116822  ultralytics.nn.modules.head.Detect           [2, [128, 256, 512]]
Model summary: 225 layers, 11136374 parameters, 11136358 gradients, 28.6 GFLOPs
```

Рисунок 3.16 - Консольний вивід процесу машинного навчання

```
Transferred 349/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100% 6.23M/6.23M [00:00<00:00, 23.3MB/s]
WARNING ⚠ NMS time limit 0.550s exceeded
AMP: checks passed ✅
train: Scanning /content/Combine-1/train/labels... 1896 images, 36 backgrounds, 0 corrupt: 100% 1896/1896 [00:01<00:00, 1333.03it/s]
train: New cache created: /content/Combine-1/train/labels.cache
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit=(1,
val: Scanning /content/Combine-1/valid/labels... 180 images, 0 backgrounds, 0 corrupt: 100% 180/180 [00:00<00:00, 1499.80it/s]
val: New cache created: /content/Combine-1/valid/labels.cache
Plotting labels to runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' a
optimizer: AdamW(lr=0.001667, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs/detect/train
Starting training for 30 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/30      4.52G      1.443      1.834      1.474         23        640: 100% 119/119 [00:21<00:00,  5.53it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.57it/s]
                   all        180        215      0.668      0.543      0.626      0.423

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/30      4.58G      1.375      1.263      1.435         22        640: 100% 119/119 [00:17<00:00,  6.97it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  4.91it/s]
                   all        180        215      0.538      0.668      0.635      0.458

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/30      4.59G      1.367      1.228      1.431         23        640: 100% 119/119 [00:16<00:00,  7.12it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  5.69it/s]
                   all        180        215      0.392      0.359      0.394      0.257

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/30       4.6G      1.313      1.161      1.378         26        640: 100% 119/119 [00:17<00:00,  7.00it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  5.03it/s]
```

Рисунок 3.17 - Консольний вивід процесу машинного навчання

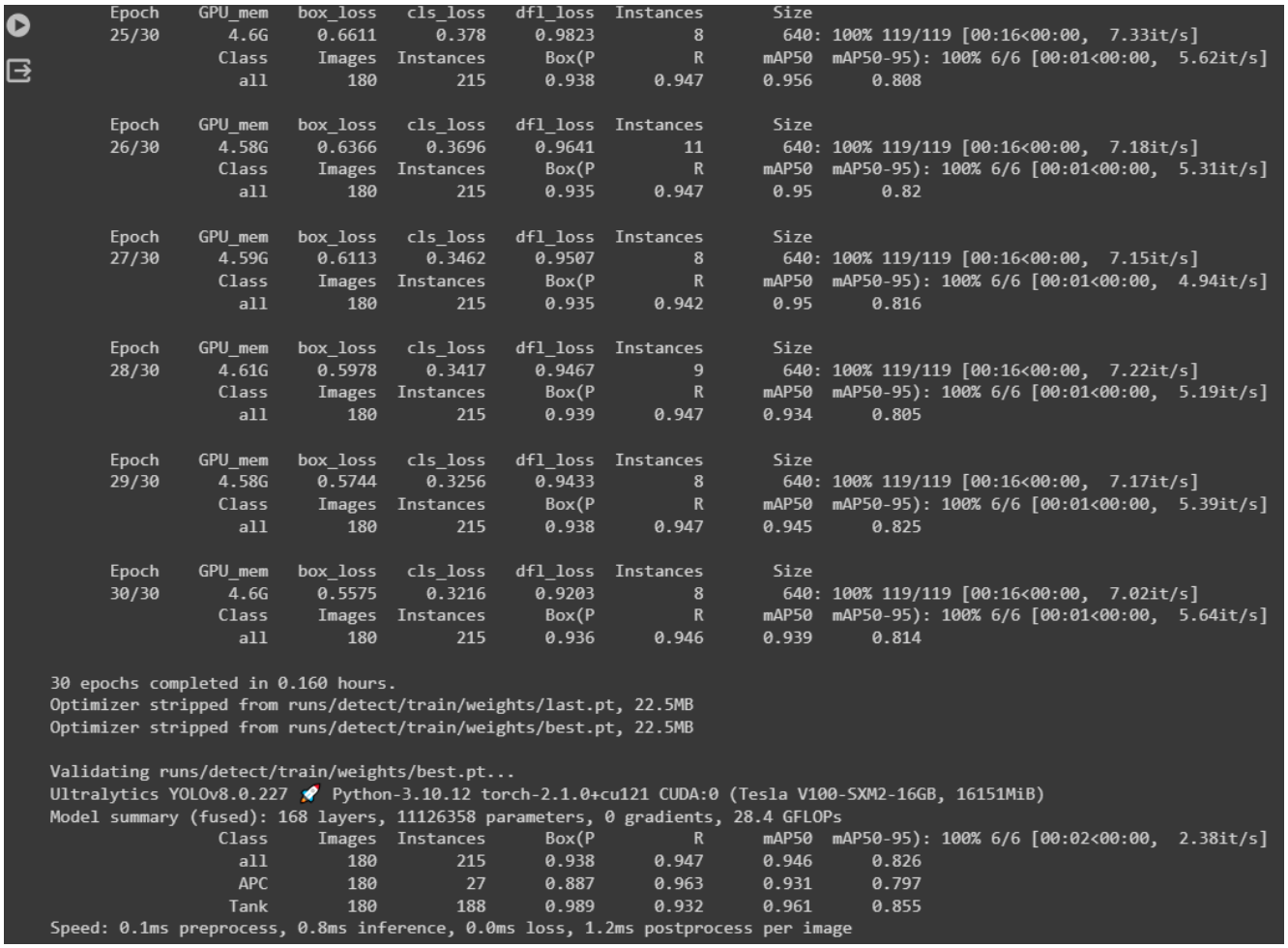

Рисунок 3.18 - Консольний вивід процесу машинного навчання

Зменшення значень втрат та поліпшення метрик точності (Precision) та відгуку (Recall) вказують на ефективне навчання моделі.

Остання епоха показує високі значення mAP50 (0.939) та mAP50-95 (0.814), свідчачи про високу ефективність моделі у виявленні об'єктів.

Зменшення втрат та покращення метрик mAP50 та mAP50-95 в останніх епохах, свідчить про доцільність збільшення кількості епох для навчання для збільшення ефективності моделі.

3.4 Сумісність моделі з модулями NVIDIA Jetson

Модулі Jetson підтримують всі ключові бібліотеки, необхідні для запуску YOLOv8, а для найбільшої продуктивності на модулях NVIDIA, YOLOv8 підтримує розгортку на TensorRT.

Обчислювальна потужність моделі підбиралась таким чином, щоб розгортатись на бюджетних модулях NVIDIA Jetson (рисунок 3.19).

Technical Specifications

Jetson Orin Nano Series		Jetson AGX Xavier Series			Jetson Xavier NX Series		Jetson TX2 Series				
Jetson Orin Nano 8GB	Jetson Orin Nano 4GB	Jetson AGX Xavier Industrial	Jetson AGX Xavier 64GB	Jetson AGX Xavier	Jetson Xavier NX 16GB	Jetson Xavier NX	TX2i	TX2	TX2 4GB	TX2 NX	Jetson Nano
40 TOPs	20 TOPs	30 TOPS	32 TOPS		21 TOPS		1.26 TFLOPS	1.33 TFLOPS			472 GFLOPS
1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores	512-core NVIDIA Ampere architecture GPU with 16 Tensor Cores	512-core NVIDIA Volta architecture GPU with 64 Tensor Cores			384-core NVIDIA Volta™ architecture GPU with 48 Tensor Cores		256-core NVIDIA Pascal™ architecture GPU				128-core NVIDIA Maxwell™ architecture GPU

Рисунок 3.19 - Технічні специфікації модулів NVIDIA Jetson

За результатами логів навчання, можемо відзначити, що найбільш ефективна модель продемонструвала обчислювальну потужність на рівні 28.4 GFLOPs.

Формула для розрахунку мінімальної продуктивності модуля з врахуванням бажаної кількості кадрів 30 FPS виглядає наступним чином:

$$\text{Продуктивність пристрою} = \text{Швидкодія моделі (в GFLOPs на кадр)} * minFPS$$

Підставивши значення, отримуємо:

$$28{,}4 * 30 = 852 \text{ GFLOPs}$$

З формули бачимо мінімальну необхідну потужність 852 GFLOPs. Враховуючи отримані результати, можемо зробити висновок, що модель має потенціал для розгортання на Jetson TX2, який пропонує обчислювальну потужність 1330 GFLOPs (рисунок 3.19).

Ще одна перевага використання YOLOv8 в комплексі з модулями NVIDIA Jetson, полягає в гнучкому підборі моделі та модуля в залежності від задачі. Наприклад YOLOv8n з використанням Jetson Nano або Jetson TX2 де необхідні недорогі рішення, або вища швидкість обробки кадрів. YOLOv8m-l з Jetson Orin Nano для задач які потребують більшої точності (Рисунок 2.3, Рисунок 3.19).

3.5 Побудова алгоритму для навігації БПЛА

Мета алгоритму полягає в коригуванні напрямку руху БПЛА, на основі аналізу положення об'єктів, виявлених на відео. Окрім використання навченої моделі, алгоритм включає декілька ключових компонентів:

1. Багатооб'єктне відстеження (Multi-Object Tracking)
2. Вибір пріоритетної цілі
3. Коригування напрямку руху для наведення на ціль

Багатооб'єктне відстеження (Multi-Object Tracking) реалізоване за допомогою трекера BoT-SORT у рамках архітектури YOLOv8. BoT-SORT - це алгоритм відстеження, який використовує комбінацію перекриття обмежувальних рамок і тимчасової асоціації, що робить його ефективним у відстеженні кількох об'єктів. Цей трекер стійкий до оклюзій і хаосу, що є звичайними проблемами в багатоцільовому відстеженні. Використання YOLOv8 забезпечує швидке та точне виявлення об'єктів, а інтеграція з BoT-SORT покращує точність відстеження та знижує помилкові спрацьовування.

Вибір пріоритетної цілі здійснюється на основі узагальненого показника впевненості, який розраховується для кожного об'єкта, використовуючи дані з останніх кадрів. Об'єкт із найвищим середнім значенням впевненості вважається

пріоритетним. Переключення між об'єктами відбувається, якщо різниця в показниках впевненості перевищує порогове значення.

Для коригування напрямку руху використовується методика розрахунку відхилення між центром обмежувальної рамки цільового об'єкта та центром зображення. Відхилення вказує на необхідність та ступінь необхідної корекції курсу БПЛА. Ця інформація візуалізується за допомогою червоних стрілок на зображенні, довжина яких відповідно відображає ступінь необхідної корекції (рисунок 3.20).

```
# Завантаження моделі YOLOv8
model = YOLO('/content/best.pt')


# Open the video file
video_path = "/content/video_input.mp4"
cap = cv2.VideoCapture(video_path)


# Отримання розмірів кадру та частоти кадрів від джерела відео
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)


# Створення об'єкту VideoWriter для збереження відео
output_path = '/content/processed_video.mp4'
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_path, fourcc, fps, (frame_width, frame_height))
```

```
        # Параметри для відстеження впевненості
        num_frames_to_consider = 5
        max_confidences_to_store = 10
        confidence_history = {}
        target_id = None  # Спочатку ціль не вибрана

        # Базова довжина та ширина стрілок
        base_arrow_length = 15
        max_arrow_length = 400  # Максимальна довжина, до якої може
розтягнутися стрілка
        arrow_width = 20
        arrow_color = (0, 0, 255)
        arrow_start_offset = 250  # Початковий зсув стрілок від лівого верхнього кута

        # Запуск циклу
        while cap.isOpened():
            success, frame = cap.read()
            if success:
                results = model.track(frame, persist=True)

                # Оновлення списку активних ID
                current_ids = set()
```

```
            if results and hasattr(results[0], 'plot'):
                for result in results:
                    if hasattr(result, 'boxes'):
                        for box in result.boxes:
                            if box.id is not None:
                                current_ids.add(box.id.item())

                # Очищення історії впевненості для ID, які більше не існують
                ids_to_remove = [box_id for box_id in confidence_history if box_id not
in current_ids]
                for box_id in ids_to_remove:
                    del confidence_history[box_id]

            if results and hasattr(results[0], 'plot'):
                frame_with_boxes = results[0].plot()
                h, w, _ = frame_with_boxes.shape
                center_image = (w // 2, h // 2)

                # Оновлення і розрахунок середньої впевненості
                for result in results:
                    if hasattr(result, 'boxes'):
                        for box in result.boxes:
                            if box.id is not None and box.conf is not None:
```

```
                box_id = box.id.item()
                confidence = box.conf.item()
                if box_id not in confidence_history:
                    confidence_history[box_id] =
deque(maxlen=max_confidences_to_store)
                confidence_history[box_id].append(confidence)
            avg_confidences = {box_id: np.mean(list(history)[-
num_frames_to_consider:])
                               for box_id, history in confidence_history.items()}

            # Вибір або оновлення цілі на основі впевненості
            if avg_confidences:
                if target_id is None or (target_id not in avg_confidences or
                    any(conf > avg_confidences[target_id] * 1.1 for conf in
avg_confidences.values())):
                    target_id = max(avg_confidences, key=avg_confidences.get)

            # Обробка та відображення інформації для цільового об'єкта
            if target_id is not None:
                target_box = next((box for box in results[0].boxes if box.id is not None
and box.id.item() == target_id), None)
                if target_box:
                    x1, y1, x2, y2 = target_box.xyxy[0]
```

```
center_box = (int((x1 + x2) / 2), int((y1 + y2) / 2))

# Розрахунок відхилення для цільового об'єкта
dx_percent = (center_box[0] - center_image[0]) / w
dy_percent = (center_box[1] - center_image[1]) / h

# Малювання стрілок у лівому верхньому куті
arrow_start = (arrow_start_offset, arrow_start_offset)  # Початкова точка стрілок зі зсувом

# Налаштування довжини стрілок з врахуванням відхилення від центру
right_arrow_length = base_arrow_length + int((max_arrow_length - base_arrow_length) * max(dx_percent, 0))
left_arrow_length = base_arrow_length + int((max_arrow_length - base_arrow_length) * max(-dx_percent, 0))
down_arrow_length = base_arrow_length + int((max_arrow_length - base_arrow_length) * max(dy_percent, 0))
up_arrow_length = base_arrow_length + int((max_arrow_length - base_arrow_length) * max(-dy_percent, 0))

# Стрілка вправо
cv2.arrowedLine(frame_with_boxes, arrow_start,
                (arrow_start[0] + right_arrow_length, arrow_start[1]),
```

```
                    arrow_color, arrow_width)
                if dx_percent > 0:  # Показувати мітку тільки для позитивного
відхилення
                    cv2.putText(frame_with_boxes, f"Right: {round(dx_percent *
100, 1)}%",
                                (arrow_start[0] + right_arrow_length + 10, arrow_start[1]),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
                # Стрілка вліво
                cv2.arrowedLine(frame_with_boxes, arrow_start,
                                (arrow_start[0] - left_arrow_length, arrow_start[1]),
                                arrow_color, arrow_width)
                if dx_percent < 0:  # Показувати мітку тільки для негативного
відхилення
                    cv2.putText(frame_with_boxes, f"Left: {round(-dx_percent * 100,
1)}%",
                                (arrow_start[0] - left_arrow_length - 60, arrow_start[1]),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
                # Стрілка вниз
                cv2.arrowedLine(frame_with_boxes, arrow_start,
                                (arrow_start[0], arrow_start[1] + down_arrow_length),
                                arrow_color, arrow_width)
```

```
                if dy_percent > 0:  # Показувати мітку тільки для позитивного
відхилення
                    cv2.putText(frame_with_boxes, f"Down: {round(dy_percent *
100, 1)}%",
                                (arrow_start[0], arrow_start[1] + down_arrow_length +
10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
                # Стрілка вверх
                cv2.arrowedLine(frame_with_boxes, arrow_start,
                                (arrow_start[0], arrow_start[1] - up_arrow_length),
                                arrow_color, arrow_width)
                if dy_percent < 0:  # Показувати мітку тільки для негативного
відхилення
                    cv2.putText(frame_with_boxes, f"Up: {round(-dy_percent * 100,
1)}%",
                                (arrow_start[0], arrow_start[1] - up_arrow_length - 20),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
                # Лінія
                if target_id is not None:
                    target_box = next((box for box in results[0].boxes if box.id is not
None and box.id.item() == target_id), None)
                    if target_box:
```

```
                x1, y1, x2, y2 = target_box.xyxy[0]
                center_box = (int((x1 + x2) / 2), int((y1 + y2) / 2))

                # Малювання лінії від центру до цільового об'єкта
                cv2.line(frame_with_boxes, center_image, center_box, (0, 255,
0), 4)  # Колір лінії

            # Відображення ступеня відхилення від центру
            deviation_texts = []
            if dx_percent > 0:
                deviation_texts.append(f"Right: {round(dx_percent * 100, 1)}%")
            if dx_percent < 0:
                deviation_texts.append(f"Left: {round(-dx_percent * 100, 1)}%")
            if dy_percent > 0:
                deviation_texts.append(f"Down: {round(dy_percent * 100, 1)}%")
            if dy_percent < 0:
                deviation_texts.append(f"Up: {round(-dy_percent * 100, 1)}%")

            for i, text in enumerate(deviation_texts):
                cv2.putText(frame_with_boxes, text, (10, h - 10 - i * 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255),
2)
```

```
            # Вивід кадру
            if frame_with_boxes is not None:
                out.write(frame_with_boxes)

                # Вивід кадру в консоль
                # cv2_imshow(frame_with_boxes)

        # Закінчення циклу при натисканні 'q'
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    else:
        # Закінчення циклу, якщо досягнуто кінця відео
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

Рисунок 3.20 - Результати роботи алгоритму

3.6 Покращення алгоритму

Одним із перспективних напрямків майбутнього дослідження є врахування траекторії руху об'єктів, яке може бути реалізоване за допомогою алгоритмів відстеження руху. Хоча у межах цієї роботи не вдалося повноцінно реалізувати цю функціональність, було зроблено кроки в напрямку використання алгоритму Лукаса-Канаде для визначення відносної швидкості руху об'єктів та алгоритму Калмана для прогнозування їх подальшого переміщення. Цей підхід може значно покращити точність та ефективність виявлення та навігації БПЛА для рухомих об’єктів. Реалізація алгоритму може бути представлена наступним чином:

```
# Клас для оцінки швидкості

class SpeedEstimator:

    def __init__(self):

        self.prev_frame = None

        self.prev_center = None
```

```
        self.lk_params = dict(winSize=(15, 15), maxLevel=2,
criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

    def estimate_speed(self, frame, center):
        # Визначення швидкості фону
        background_speed = self._calculate_background_speed(frame)

        # Визначення швидкості об'єкта
        object_speed = self._calculate_object_speed(center) if self.prev_center is not
None else np.array([0, 0])

        # Розрахунок відносної швидкості
        relative_speed = object_speed - background_speed

        # Оновлення попередніх значень
        self.prev_frame = frame.copy()
        self.prev_center = center

        return relative_speed

    def _calculate_background_speed(self, frame):
        if self.prev_frame is not None:
            # Конвертація кадрів до чорно білого
```

```
            prev_gray = cv2.cvtColor(self.prev_frame, cv2.COLOR_BGR2GRAY)
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Параметри для goodFeaturesToTrack
            feature_params = dict(maxCorners=100, qualityLevel=0.3,
minDistance=7, blockSize=7)

            p0 = cv2.goodFeaturesToTrack(prev_gray, mask=None,
**feature_params)
            if p0 is not None:
                p1, st, err = cv2.calcOpticalFlowPyrLK(prev_gray, gray, p0, None,
**self.lk_params)
                good_old = p0[st == 1]
                good_new = p1[st == 1]
                flow = good_new - good_old
                average_flow = np.mean(flow, axis=0)
                return average_flow
        return np.array([0, 0])

    def _calculate_object_speed(self, center):
        speed = np.array(center) - np.array(self.prev_center)
        return speed
```

```
# Клас KalmanFilter
class KalmanFilter:
    def __init__(self, dt=1):
        self.kf = cv2.KalmanFilter(4, 2)  # 4 станові змінні (x, y, vx, vy), 2 вимірювання (vx, vy)
        self.dt = dt  # часовий інтервал між кадрами

        # Матриця переходу (для рівномірного руху)
        self.kf.transitionMatrix = np.array([[1, 0, self.dt, 0],
                                             [0, 1, 0, self.dt],
                                             [0, 0, 1, 0],
                                             [0, 0, 0, 1]], np.float32)

        # Матриця вимірювань (тільки для швидкості)
        self.kf.measurementMatrix = np.array([[0, 0, 1, 0],
                                              [0, 0, 0, 1]], np.float32)

    def predict(self):
        prediction = self.kf.predict()
        return int(prediction[0]), int(prediction[1])

    def update(self, speed):
        # Оновлення фільтра за швидкістю
```

```
            self.kf.correct(np.array([[np.float32(speed[0])], [np.float32(speed[1])]]))


    # Ініціалізація KalmanFilter
    kf = KalmanFilter()
    speed_estimator = SpeedEstimator()


    # В циклі обробки кадрів
    # Оцінка відносної швидкості
                relative_speed = speed_estimator.estimate_speed(frame, center_box)
                print(f"Relative speed: {relative_speed}")


                # Використання відносної швидкості для корегування прогнозу фільтра Калмана
                kf.update(relative_speed)
                predictedX, predictedY = kf.predict()
                print(f"Kalman prediction: X={predictedX}, Y={predictedY}")


                # Візуалізація прогнозу
                cv2.circle(frame_with_boxes, (predictedX, predictedY), 10, (0, 0, 255), 2)
```

ВИСНОВКИ

Серед основних проблем застосування БПЛА є підтримка якісного зв’язку та необхідність високої кваліфікації оператора. Таким чином, в роботі було розглянуто використання нейромереж, для мінімізації людського втручання в управління польотом.

Розглянуті в роботі аспекти вибору, навчання моделі та підготовки даних демонструють підхід до вирішення проблем, що виникають під час розробки моделі для виявлення військових об'єктів. Використання базових моделей, таких як Grounded SAM, та напівавтоматичної розмітки дозволяє значно скоротити час та ресурси, необхідні для мануальної розмітки. Для вирішення проблеми обмеженості даних у відкритих джерелах, було запропоновано інтеграцію синтетичних даних разом з реальними, що підвищило загальну якість моделі. Використання ПЗ Blender для генерації цих даних дозволило розширити можливості для тренування моделі в різноманітних умовах.

Зіставлення результатів двох моделей вказує на перевагу моделі, навченої з використанням синтетичних даних, особливо у складних сценаріях, демонструючи її кращу здатність до узагальнення та меншу схильність до перенавчання.

Для використання результатів виявлення моделі для корекції курсу, було створено алгоритм який виконує багатооб'єктне відстеження, вибір пріоритетної цілі та коригування напрямку руху.

Наведені результати вказують на високу ефективність використання комплексних підходів до підготовки датасетів і застосування передової архітектури YOLOv8, у поєднанні з мікропроцесорними модулями Nvidia Jetson для задач детекції об'єктів та подальшого коригування напрямку руху БПЛА.

Використання нейромереж дозволяє зменшити людське втручання в управління польотом, оскільки БПЛА мають потенціал самостійно вибирати цілі та адаптувати завдання польоту на основі аналізу отриманих даних.

Результати цієї роботи мають потенціал для використання в подальших розробках алгоритмів, які можуть застосовуватися як у сценаріях автономного

пошуку та наведення на цілі, так і для допомоги операторам. Такі алгоритми здатні значно підвищити гнучкість та ефективність БПЛА, у ситуаціях одноразового та багаторазового застосування.

Подальші дослідження можуть бути спрямовані на розширення різноманітності генерації навчальних даних, розробку складніших алгоритмів, інтеграцію з польотними контролерами та вдосконалення взаємодії з навігаційними системами, що відкриває широкі перспективи для подальшого розвитку технологій в сфері безпілотної авіації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. https://medium.com/@The_Mad_Zaafa/machine-learning-an-explanation-for-the-novice-eb22d6515a32
2. https://www.mygreatlearning.com/blog/types-of-neural-networks/
3. Нейронні мережі : теорія та практика: навч. посіб., С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.
4. https://cs231n.github.io/convolutional-networks/#conv
5. https://www.cs.cornell.edu/courses/cs5670/2021sp/lectures/lec22_cnns2_web.pdf
6. https://bpb-us-e1.wpmucdn.com/blogs.rice.edu/dist/1/7022/files/2018/09/ELEC576-Lec-04-xr2cds.pdf
7. Data Preparation for Machine Learning Data Cleaning, Feature Selection, and Data Transforms in Python, Jason Brownlee, 2020
8. https://github.com/ultralytics/ultralytics/issues/1658
9. https://docs.ultralytics.com/tasks/detect/#models
10. https://www.mdpi.com/1424-8220/23/16/7190
11. https://github.com/autodistill/autodistill
12. https://docs.autodistill.com/
13. https://learnopencv.com/train-yolov8-on-custom-dataset/
14. https://docs.ultralytics.com/modes/train/
15. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
16. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville. – Cambridge, Massachusetts: The MIT Press, 2016. – 775 p
17. Neural Networks and Deep Learning: A Textbook by Charu Aggarwal. – Cham, Switzerland: Springer, 2018. – 497 p.
18. https://docs.ultralytics.com/guides/model-deployment-options/#tfjs
19. https://paperswithcode.com/sota/multi-object-tracking-on-mot20-1?p=bot-sort-robust-associations-multi-pedestrian
20. https://arxiv.org/abs/2206.14651

21. https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12754/2684256/Pedestrian-multi-object-tracking-based-on-YOLOv7-and-BoT-SORT/10.1117/12.2684256.full