НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Приладобудівний факультет
Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

До захисту допущено:
Завідувач кафедри _____Надія БУРАУ
«___»____20___р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерно - інтегровані системи та технології в приладобудуванні»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Розробка програмного забезпечення для системи технічного зору на основі нейронних мереж»

Виконав:
студент IV курсу, групи ПО-01
Старушик Вадим Миколайович _____

Керівник:
к.т.н., старший викладач
Мамута Марина Сергіївна _____

Консультант: _____

Рецензент:
к.т.н., доцент
Вонсевич Костянтин Петрович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторівбез відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Приладобудівний факультет

Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології

Освітня програма - Комп'ютерно - інтегровані системи та технології в приладобудуванні

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Надія БУРАУ

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Старушику Вадиму Миколайовичу

1. Тема проекту «Розробка програмного забезпечення для системи технічного зору на основі нейронних мереж», керівник роботи Мамута Марина Сергіївна, к.т.н., старший викладач, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи 5 червня 2024 р.

3. Вихідні дані до проекту: модель має забезпечувати виявлення об'єкта, ефективність на нових даних не нижче 0,5.

4. Зміст роботи: огляд інформаційних джерел з теорії нейронних мереж та систем технічного зору; тренування нейромережі YOLO для виявлення об'єкту; тестування моделі.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): модель та підготовка даних, тренування моделі, результати роботи моделі.

6. Консультанти розділів проекту*

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 16 квітня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів роботи	Примітка
1.	Дослідження систем технічного зору	20.05.2024	
2.	Дослідження нейронних мереж	25.05.2024	
3.	Підготовка даних	31.05.2024	
4.	Тренування моделі	03.06.2024	
5.	Тестування моделі	06.06.2024	
6.	Аналіз результатів	08.06.2024	
7.	Оформлення пояснювальної записки	10.06.2024	
8.	Подача дипломної роботи до захисту	12.06.2024	

Студент Руслан ТЕПЛЕНКО

Керівник Марина МАМУТА

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

Дана дипломна робота присвячена розробці програмного забезпечення на основі нейронної мережі для системи технічного зору дрону, щоб вона могла виявляти та розпізнавати об’єкти типу танк. У роботі було проаналізовано різні види нейронних мереж та їх актуальність на даний момент. Також проаналізовано системи технічного зору, їхній рівень розвитку та області використання. В роботі було здійснено підготовку власного набору даних за допомогою інструменту CVAT (Computer Vision Annotation Tool), який було використано для навчання нейронної мережі YOLOv8, щоб вона могла виявляти та розпізнавати об’єкти типу танк. Було здійснено тестування моделі, що показало гарні результати, в тому числі й на нових даних. Тестування проводились на конкретних відео, в яких нейронна мережа вдало розпізнавала танки.

ANOTATIONS

This thesis is devoted to the development of software based on a neural network for a drone's vision system so that it can recognize a specific object, in our case, a tank. The work analyzed: different neural networks, what they are and their relevance at the moment; different vision systems, their level of development at the moment and where they are used. In this work, we prepared our own dataset using the CVAT (Computer Vision Annotation Tool) tool, which was used to train the YOLOv8 neural network to detect and recognise objects such as a tank. The model was tested and showed good results, including on new data. Testing was conducted on specific videos in which the neural network successfully recognised tanks.

ЗМІСТ

ВСТУП

З появою комп’ютерів і цифрових технологій наш світ докорінно змінився, принісши з собою багато нових можливостей у багатьох сферах життя. Одним з найбільш важливих і перспективних напрямків, де досягнуто значного прогресу, є використання систем технічного зору для обробки та аналізу зображень. Завдяки поєднанню методів обробки зображень і штучних нейронних мереж ми сьогодні маємо можливість розробляти потужні програмні засоби для вирішення різноманітних завдань у сфері комп’ютерного зору. Системи технічного зору пропонують можливість автоматичного аналізу зображення для розпізнавання об'єктів, вимірювання параметрів, виявлення аномалій і багатьох інших завдань.

У сучасному світі, коли обсяги даних продовжують зростати, системи технічного зору стають все більш важливим інструментом для автоматизації процесів у багатьох галузях промисловості (наприклад, медицина, транспорт, виробництво, безпека тощо). Ці системи можуть не лише забезпечити значні економічні вигоди, але й покращити якість життя та безпеку людей.

Одним із ключових елементів сучасних систем технічного зору є використання нейронних мереж. Нейронні мережі — це математичні моделі, натхненні нейронною структурою людського мозку, які можуть автоматично вивчати корисні функції з великих обсягів даних і вирішувати різноманітні завдання аналізу даних. Завдяки своїм унікальним властивостям нейронні мережі стали надзвичайно популярними інструментами в галузі комп’ютерного зору та штучного інтелекту.

Біотехнологічні візуальні системи, такі як очі людей, тварин, комах, риб тощо, високоефективні та здатні адаптуватися до різноманітних умов

середовища. Наприклад, очі комах надзвичайно чутливі до рухомих об’єктів, а мітки у формі ходунів на клітинах тіла риби реагують на найменші зміни світла. Стимуляція таких біологічних систем при створенні технічних пристроїв для обробки й аналізу зображень може призвести до розробки більш ефективних і адаптивних систем технічного зору.

Так як тема пов’язана одночасно із технічним зором та нейронною мережею, доречно зауважити, що природна складна система очей (compound eye) має багато видатних властивостей, таких як більше компактний розмір, ширший кут огляду, краща здатність виявляти рухомі об’єкти та вища чутливість до інтенсивність світла в порівнянні з інтенсивністю світла системи зору з однією апертурою. Завдяки розвитку мікро- та нанотехніки виробництва, багато штучних комплексних систем зображення очей було проаналізовано, вивчено та виготовлено для успадкування захоплюючих оптичних особливостей природного складного ока, щоб взяти всі його переваги та виключити недоліки.

Важливою частиною будь-якої інноваційної розробки є її можливе застосування в різних сферах. Програмне забезпечення системи технічного зору, засноване на нейронних мережах, має потенціал для значного впливу на різноманітні галузі. У медицині цю технологію можна використовувати для автоматичного аналізу зображень для діагностики захворювань, їх виявлення на медичних зображеннях та моніторингу стану пацієнтів. У сфері автономної навігації програмне забезпечення для систем технічного зору допомагає розпізнавати об’єкти на шляху транспортного засобу та автоматично розпізнавати дорожні знаки та світлофори, щоб уникнути зіткнень. У робототехніці такі системи можна використовувати для автоматизації виробничих процесів, контролю якості продукції та створення роботів,

здатних самостійно взаємодіяти з навколишнім середовищем. Це лише маленький приклад того, в яких сферах може використовуватись дана технологія. Різноманітні можливості застосування демонструють великий інноваційний потенціал цієї розробки та підкреслюють її важливість для різних галузей промисловості.

Враховуючи той факт, що на сьогоднішній день першочерговою задачею є захист країни, то актуальність роботи не викликає сумнівів, адже робота присвячена вирішенню задачі автономного виявлення об’єктів дроном, що при подальшому вдосконаленні забезпечить автономне виконання місії (без участі оператора). Актуальність підтверджується запитами з підприємств нашої країни, зокрема НТК «Електронприлад». Робота виконана в межах ініціативної науково-дослідної роботи «Дослідження автоматизованих оптико-електронних систем різного призначення», номер державної реєстрації 0122U200981.

Збільшення обчислювальної потужності та доступності швидких і ефективних алгоритмів обробки даних ще більше підвищить точність, швидкість і надійність систем технічного зору. Очікується, що нейронні мережі й надалі відіграватимуть важливу роль у цьому процесі, дозволяючи системам адаптуватися до різних умов і завдань. Дане дослідження спрямоване на розробку програмного забезпечення на основі нейронних мереж для автоматизованих оптико-електронних систем дронів і має на меті вирішити одну із задач для створення повністю автономного дрону.

Поставлена мета вирішується шляхом виконання наступних завдань:

1. Здійснити огляд літератури, обґрунтувати вибір нейронної мережі для виявлення об’єктів;

2. Навчити нейронну мережу виявляти і розпізнавати об’єкт типу танк;
3. Проаналізувати ефективність навчання нейронної мережі та основні метрики;
4. Оцінити ефективність роботи програми.

РОЗДІЛ 1. Огляд інформаційних джерел з теорії нейронних мереж та систем технічного зору

Перед тим, як детально проаналізувати системи технічного зору та відповідне програмне забезпечення для них, доречно роз’яснити, що ж таке нейронні мережі.

1.1 Поняття нейронної мережі

Нейронна мережа, тип паралельного обчислення, у якому обчислювальні елементи моделюються на основі мережі нейронів, які складають нервову систему. Ця модель, призначена для імітації того, як мозок обробляє інформацію, дозволяє комп’ютеру певною мірою «навчатися». Нейронна мережа зазвичай складається з ряду взаємопов’язаних процесорів або вузлів. Кожен обробляє визначену сферу знань і має кілька входів і один вихід до мережі. На основі вхідних даних, які він отримує, вузол може «дізнаватися» про зв’язки між наборами даних, іноді використовуючи принципи нечіткої логіки.

Отже, даний розділ присвячений тому, щоб зрозуміти як працює нейромережа, з чого вона складається, як вона «навчається» та найголовніше – де вона використовується.

Штучна нейронна мережа - це комп'ютерна програма, яка працює на основі принципів природних нейронних мереж мозку. Метою штучних нейронних мереж є виконання когнітивних функцій, таких як вирішення проблем або машинне навчання. Основною властивістю нейронних мереж є їхня здатність імітувати здатність мозку до розпізнавання образів. Комерційні застосування цієї здатності включають інвестиційні рішення, розпізнавання почерку та різних об'єктів і навіть виявлення бомб.

Особливістю нейронних мереж є те, що знання про предметну область не записуються в програму в явному вигляді, а поширюються через саму мережу. Ці знання моделюються у вигляді зв'язків між елементами обробки (штучними нейронами) та адаптивних ваг для кожного зв'язку. І мережа навчається з різних ситуацій. Нейронні мережі досягають цього шляхом регулювання ваг зв'язків між взаємодіючими нейронами, згрупованими в шари. Вхідний шар штучних нейронів отримує інформацію з навколишнього середовища, а вихідний - дає відповідь. Між цими шарами знаходяться один або кілька «прихованих» шарів (які не мають прямого контакту з навколишнім середовищем), де відбувається більша частина обробки інформації. Вихід нейронної мережі залежить від вагових коефіцієнтів зв'язків між нейронами в різних шарах. Кожна вага відображає відносну важливість певного зв'язку. Коли сума зважених входів, отриманих певним нейроном, перевищує певний поріг, він надсилає сигнал кожному з нейронів, підключених до наступного шару. Наприклад, при обробці кредитної заявки вхідним сигналом може бути профіль заявника, а вихідним - інформація про те, чи було схвалено кредит. [1, 11, 16]

Додавання двох удосконалень до цієї простої нейронної мережі прямого поширення дозволило розширити сферу її застосування, наприклад, розпізнавання об'єктів. По-перше, мережа може бути оснащена механізмом зворотного зв'язку, відомим як алгоритм зворотного поширення. Цей механізм дозволяє коригувати ваги зв'язків у зворотному напрямку через мережу, навчаючи мережу у відповідь на репрезентативні приклади. По-друге, розробка рекурентної нейронної мережі дозволяє приймати сигнали в обох напрямках, а не тільки внутрішньо- або міжшарові сигнали. На практиці, для великих мереж дуже важко відстежити, як було визначено вихід.

Навчання нейронних мереж зазвичай включає контрольоване навчання, де кожен навчальний приклад містить значення як вхідних даних, так і бажаних вихідних даних. Як тільки мережа зможе достатньо добре працювати на додаткових тестових випадках, її можна буде застосувати до нових випадків. І навпаки, певні нейронні мережі навчаються за допомогою неконтрольованого навчання, у якому мережа отримує набір вхідних даних і ставить собі за мету виявлення закономірностей — без вказівок, на що конкретно звертати увагу. Таку нейронну мережу можна використовувати для аналізу даних.

Нейронні мережі знаходяться на передовій когнітивних обчислень, покликаних дозволити інформаційним технологіям виконувати деякі з передових розумових функцій людини. Системи глибокого навчання базуються на багатоступеневих нейронних мережах і потужності. З експоненціальним зростанням обчислювальних потужностей і величезних масивів даних нейронні мережі глибокого навчання впливають на розподіл праці між людьми і машинами.

1.2 Актуальність нейронних мереж на даний час

Одним з найважливіших досягнень 21 століття стало створення і поширення штучного інтелекту. Як не дивно, ШІ тісно пов'язаний з мережею. Штучний інтелект (ШІ) - це здатність цифрових комп'ютерів або керованих комп'ютером роботів виконувати завдання, які зазвичай асоціюються з розумним життям. Цей термін часто застосовується до проектів з розробки систем з інтелектуальними процесами, подібними до людських, такими як здатність міркувати, знаходити сенс, узагальнювати і вчитися на минулому досвіді. З розвитком цифрових комп'ютерів було продемонстровано, що комп'ютери можна запрограмувати на виконання надскладних завдань, таких

як пошук доказів математичних теорем або гра в шахи, з високим ступенем майстерності. Однак, незважаючи на постійно зростаючу швидкість обробки даних і обсяг пам'яті комп'ютерів, жодна програма ще не змогла повністю зрівнятися з людською гнучкістю в більш широкому діапазоні областей або в завданнях, які вимагають великого обсягу знань на щоденній основі. З іншого боку, деякі програми досягли рівня, коли вони можуть виконувати певні завдання не гірше за експертів або професіоналів, і ШІ в цьому обмеженому розумінні можна побачити в різних сферах застосування, включаючи медичну діагностику, комп'ютерні пошукові системи, розпізнавання мови і рукописного тексту, а також чат-боти. Застосування. Вирішення проблем, особливо в галузі штучного інтелекту, - це систематичне дослідження діапазону можливих дій для досягнення заздалегідь визначеної мети і рішення. Методи вирішення проблем можна розділити на спеціалізовані та універсальні. Спеціалізовані методи розроблені спеціально для конкретної проблеми, часто використовуючи дуже специфічні особливості контексту, в який вбудована проблема. З іншого боку, методи загального призначення можуть бути застосовані до широкого кола проблем. Одним з методів загального призначення, що використовується в ШІ, є аналіз засобів і цілей. Аналіз засобів і цілей - це метод зменшення різниці між поточним станом і кінцевою метою поетапно або інкрементно. [11]

Отже, роблячи висновки, можна сказати, що нейронні мережі є потужними інструментами в сучасній обчислювальній техніці, оскільки вони дозволяють моделювати складні зв’язки в даних і вирішувати різноманітні завдання від розпізнавання об’єктів до прийняття рішень. Вони працюють за принципом передачі сигналу між штучними нейронами та регулюють ваги з’єднань, що дозволяє їм навчатися через взаємодію з навколишнім

середовищем. Дедалі більше використання нейронних мереж у глибокому навчанні змінює спосіб обробки інформації та розуміння інтелекту. Такі мережі допомагають вирішувати складні завдання, які раніше були виключно прерогативою людей. Впровадження нейронних мереж змінило підхід до технічного зору, зробивши можливим автоматичну обробку інформації та ідентифікацію об’єктів у різних умовах. Штучний інтелект і нейронні мережі відкривають широкі перспективи для автоматизації та оптимізації різних сфер людської діяльності.

1.3 Системи технічного зору, їх актуальність

Якщо розглядати системи технічного зору на основі нейромереж, тобто з використанням якогось штучного інтелекту, можна сказати, що при сприйнятті навколишнє середовище сканується за допомогою різних органів чуття, реальних чи штучних, а сцена розкладається на окремі об’єкти в різних просторових співвідношеннях. Аналіз ускладнюється тим фактом, що об’єкт може виглядати по-різному залежно від кута, під яким на нього дивляться, напрямку та інтенсивності освітлення в сцені та того, наскільки об’єкт контрастує з навколишнім полем.

Актуальність систем технічного зору є досить великою в наш час. Тільки за останній рік ця галузь стрімко розвивалась та довела свій великий вплив на інші галузі промисловості. Від медичного зображення до творчого мистецтва, ці розробки готують основу для майбутніх проривів і застосувань у комп’ютерному зорі. Можна назвати багато прикладів використання систем технічного зору:

- Класифікація зображень за допомогою якого система з використання нейронної мережі бачить зображення та може його

класифікувати (собака, яблуко, обличчя людини). Точніше, він здатний точно передбачити, що дане зображення належить до певного класу. Наприклад, компанія соціальних медіа може використовувати його для автоматичної ідентифікації та відокремлення небажаних зображень, завантажених користувачами;

- Виявлення об’єктів за допомогою якого система може використовувати класифікацію зображень для ідентифікації певного класу зображень, а потім виявити їх появу на зображенні чи відео. Приклади включають виявлення пошкоджень на складальній лінії або ідентифікацію обладнання, яке потребує технічного обслуговування.
- Пошук зображень на основі вмісту використовує комп’ютерний зір для перегляду, пошуку та отримання зображень із великих сховищ даних на основі вмісту зображень, а не пов’язаних із ними тегів метаданих. Це завдання може включати автоматичне анотування зображення, яке замінює ручне тегування зображення. Ці завдання можна використовувати для систем управління цифровими активами та можуть підвищити точність пошуку та вилучення. [13]

Не зважаючи на вагомий прорив у даній сфері, є багато факторів, які потребують покращення. Наприклад:

- Етичний аспект: важливо розробити стандарти та правила для забезпечення етичного використання технологій технічного зору, включаючи захист конфіденційності та недискримінацію;

- Надійність та точність: хоча сучасні моделі мають високу точність, вони все ще не сприйнятливі до змін умов навколишнього середовища, таких як освітлення, кути зйомки тощо;
- Зменшення навантаження: незважаючи на потужне апаратне забезпечення, оптимізація моделей для зменшення споживання енергії та обчислювальних ресурсів є актуальним завданням;
- Розвиток: удосконалити методи навчання для ефективнішого використання нейронної мережі. [13]

Також дана тема має великі перспективи в майбутньому. Дивлячись уперед на 2024-2025 рік, є низка очікуваних тенденцій, які спрямовані на подальшу революцію в цій динамічній сфері. Наприклад:

- Комп'ютерне бачення в режимі реального часу: можливість аналізу відео в реальному часі та вжиття негайних заходів буде розширено за допомогою програм безпеки, моніторингу натовпу та промислової безпеки. Ці системи реального часу підвищать швидкість реагування та безпеку експлуатації;
- Удосконалене супутникове бачення: прогрес у супутникових зображеннях, які за допомогою систему технічного зору дозволить більш детально контролювати земні явища, такі як вирубка лісів, розростання міст і морське середовище. Покращена роздільна здатність, яку забезпечують ці технології, матиме вирішальне значення для моніторингу та управління навколишнім середовищем.
- Виявлення дипфейків: у міру того, як дипфейки, створені штучним інтелектом, стають більш реалістичними, система технічного зору відіграватиме вирішальну роль у боротьбі з дезінформацією. Її

здатність аналізувати зображення та виявляти ознаки маніпуляцій буде життєво важливою для підтримки цілісності інформації.

У зв'язку з безперервним зростанням повітряного руху та все частішого використання безпілотних апаратів збільшується кількість авіатранспортних подій. Системи технічного зору в дронах мають вирішальне значення для їх автономності та ефективності. Вони забезпечують безпеку польотів, точність місії та здатність адаптуватися до змін навколишнього середовища. Ось деякі ключові аспекти використання технічного зору в безпілотних літальних апаратах:

- Навігація: системи технічного зору в безпілотних літальних апаратах допомагатиме дрону виявляти перешкоди в просторі та оперативно їх уникати, також це допоможе визначити місце розташування самого дрону;
- Моніторинг: за допомогою системи технічного зору безпілотні літальні апарати можуть проводити моніторинг важливих об’єктів та швидко реагувати на проблеми в них;
- Зручне керування: системи технічного зору значно полегшують керування дроном, в такому випадку безпілотні літальні апарати зможуть самостійно реалізувати посадку та зліт, без втручання людини та уникати можливих помилок через людські фактори.

1.4 Нейронні мережі для систем технічного зору

Якщо об’єднувати ці дві великі галузі (нейронні мережі та системи технічного зору), то можна сказати, що вони прекрасно доповнюють одна одну.

Сучасні візуальні системи використовують різноманітні нейромережеві моделі для виявлення об'єктів, кожна з яких має свої унікальні особливості та переваги. Найпоширеніші моделі включають згорткові нейронні мережі (CNN), згорткові нейронні мережі на основі регіонів (R-CNN, Fast R-CNN, Faster R-CNN), однократні багатоелементні детектори (SSD) і метод «You Only Look Once» (YOLO).

Наприклад, нейронна мережа CNN є базовою архітектурою для багатьох інших моделей. Вона використовується для автоматичного вилучення ознак із зображень, що робить її придатною для задач класифікації та виявлення об'єктів. R-CNN сегментує зображення на кілька областей і використовує CNN для класифікації кожної області. Цей метод є точним, але трудомістким через велику кількість регіонів, які потрібно обробити.

You Only Look Once (YOLO) – це алгоритм виявлення об'єктів, представлений у 2015 році в дослідницькій роботі Джозефа Редмона, Сантоша Діввала, Росса Гіршика та Алі Фархаді. Архітектура YOLO стала значною революцією у сфері виявлення об'єктів у реальному часі, перевершивши свого попередника - регіональну згорткову нейронну мережу (R-CNN).

YOLO - це алгоритм, який безпосередньо класифікує об'єкт за один прохід, використовуючи лише одну нейронну мережу для прогнозування обмежувальних рамок та ймовірностей класів, використовуючи повне зображення в якості вхідних даних.

Сімейство моделей YOLO постійно розвивається. З того часу кілька дослідницьких груп випустили різні версії YOLO, остання з яких - YOLOv8 - є останньою ітерацією.

Особливості роботи моделі YOLO показано на рис. 1.1.

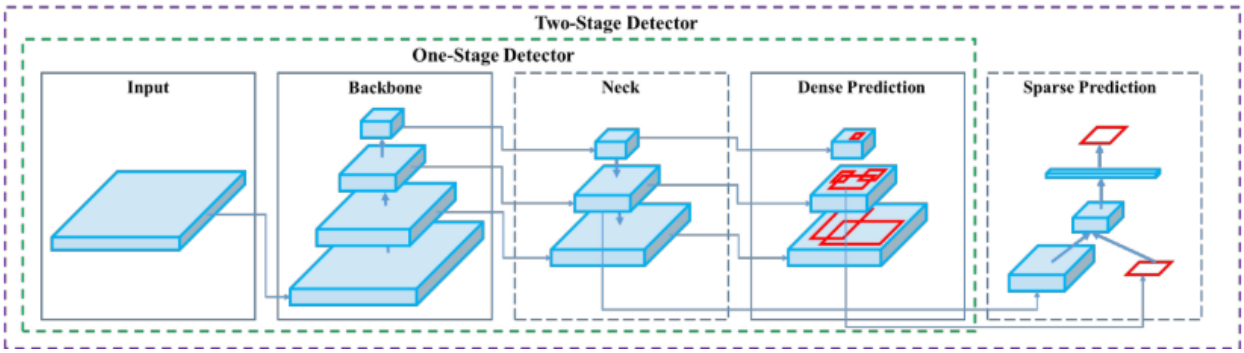


Рис. 1.1 Алгоритм виявлення об’єктів у системі YOLO [2]

Архітектура складається з кістяка (backbone), шиї (neck) та голови (head). Кістяк - це попередньо навчена згорткова нейронна мережа (CNN), яка витягує з вхідного зображення карти ознак низького, середнього та високого рівнів. Шия об'єднує ці карти ознак за допомогою блоків агрегації шляхів, таких як мережа піраміди ознак (Feature Pyramid Network, FPN). Вона передає їх голові, класифікуючи об'єкти і прогнозуючи обмежувальні рамки. Голова може складатися з одноетапних або щільних моделей прогнозування, таких як YOLO або Single-shot Detector (SSD). Крім того, вона може містити двоетапні або розріджені алгоритми прогнозування, такі як серія R-CNN [2].

З існуючих моделей виявлення об'єктів було обрано модель YOLO. Ця модель має кілька важливих переваг, які роблять її перспективною для використання у зоровій системі.

Унікальність моделі YOLO полягає в її швидкості. Оскільки вона обробляє зображення в режимі реального часу, це важливо для багатьох застосувань, де час відгуку є критичним фактором. Це пояснюється тим, що YOLO виявляє об'єкти за один прохід через мережу, на відміну від інших моделей, які потребують декількох проходів.

YOLO може виявляти кілька об'єктів на відео одночасно, що робить її більш корисною у складних сценах з кількома об'єктами.

Швидкість - одна з головних переваг YOLO, але також і точність виявлення об'єктів. Вона допускає менше помилок у позиціонуванні об'єктів, що важливо для точного аналізу зображень.

Оскільки модель YOLO відносно проста у впровадженні та використанні, її можна швидко інтегрувати в різноманітні системи технічного зору. Крім того, її зручний інтерфейс і багата документація дозволяють легко адаптувати модель до конкретних потреб.

1.5 Актуальність системи технічного зору для виявлення об’єктів

Новітні технології технічного зору та нейронні мережі відкривають нові можливості в різних галузях, в тому числі в оборонній промисловості. Зокрема, зі зростанням ролі безпілотних систем у військових операціях, розробка програмного забезпечення для дронів, здатного забезпечити автономне виконання місії при втраті зв’язку з оператором є надзвичайно важливою задачею.

Безпілотники з візуальними можливостями можуть значно підвищити ефективність розвідувальної діяльності за рахунок точного і швидкого виявлення та ідентифікації ворожих цілей. Це дозволяє проводити розвідувальні операції безпосередньо в зоні бойових дій з мінімальним втручанням людини, зменшуючи ризик для особового складу. [8, 9]

Використання нейронних мереж, особливо моделей YOLO, забезпечує високу точність і швидкість обробки даних. Це має вирішальне значення в реальних бойових ситуаціях, коли час реагування обмежений секундами.

Використання безпілотників з можливостями автономного виявлення, розпізнавання та знищення цілей зробить значний вплив на тактику і стратегію ведення бойових дій. Можливість швидко і точно ідентифікувати важку техніку противника дозволяє командуванню оперативно розподіляти сили і засоби, підвищуючи шанси на успіх у виконанні бойових завдань. [7, 10, 13]

Іншим важливим аспектом є постійне вдосконалення алгоритмів розпізнавання об'єктів. Останні досягнення в галузі машинного навчання та глибоких нейронних мереж можуть бути використані для постійного підвищення точності та швидкості роботи системи, а також адаптації до нових типів загроз та бойових ситуацій. Тому розробка програмного забезпечення для безпілотників є актуальним і перспективним завданням з потенціалом для подальшого розвитку та вдосконалення.

1.6 Висновки до розділу

Отже, роблячи висновки, можна сказати, впровадження нейронних мереж змінило підхід до технічного зору, зробивши можливим автоматичну обробку інформації та ідентифікацію об’єктів у різних умовах. Штучний інтелект і нейронні мережі відкривають широкі перспективи для автоматизації та оптимізації різних сфер людської діяльності. Системи технічного зору на основі нейронних мереж є ключовим напрямком розвитку штучного інтелекту, що дозволяє автоматично аналізувати та інтерпретувати візуальні дані. Їхня актуальність у сучасному світі значно зросла завдяки широкому спектру застосувань, що охоплює медицину, промисловість, безпеку та творчість. Системи технічного зору здатні вирішувати складні завдання, такі як класифікація зображень, виявлення об'єктів та пошук зображень на основі вмісту, що робить їх незамінними інструментами в багатьох сферах. Незважаючи на досягнутий прогрес, існує потреба у подальшому

вдосконаленні цих систем. Зокрема, важливо розробити етичні стандарти для забезпечення конфіденційності та недискримінації, підвищити надійність і точність роботи моделей у змінних умовах, а також оптимізувати їх для зниження енергоспоживання та обчислювальних витрат. Крім того, розвиток методів навчання нейронних мереж на обмежених даних сприятиме їх більш ефективному застосуванню.

Керуючись перевагами моделі YOLO, саме вона була обрана для розробки програмного забезпечення для систем технічного зору, оскільки через свою швидкість, точність та простоту, вона ідеально підходить для широкого спектру завдань у цій галузі.

Розробка нейромережевого програмного забезпечення для безпілотників, здатного виявляти та розпізнавати танки, є не лише технічно цікавим завданням, але й має важливе практичне значення для національної безпеки та підвищення ефективності військових операцій. Це може врятувати життя солдатів і допомогти їм виконувати критичні завдання більш точно і швидко в умовах сучасної війни.

РОЗДІЛ 2. Навчання нейронної мережі

Для навчання нейронної мережі використовується декілька підходів. Найбільш робастний, але і найбільш трудомісткий – це навчання мережі «з нуля». Більш швидкий та добре себе зарекомендувавший – перенесення навчання або transfer learning.

2.1 Transfer learning

Transfer learning – це ідея використання знань, отриманих при виконанні одного завдання, і застосування їх до подібної задачі. Наприклад, при спробі розробити систему для виявлення/класифікації об'єктів, модель може вимагати тисячі зображень на об'єкт, щоб досягти прийнятних результатів для одного об'єкта. Кожне з цих зображень потрібно буде позначити, щоб модель могла вивчити зв'язок між зображенням і класифікацією. Маркування даних може зайняти значну кількість часу навіть для простих додатків; саме тут в гру вступає навчання з перенесенням. Навчання з перенесенням навчання дозволяє використовувати знання і ваги моделі, навченої на попередньо розміченому наборі даних. Використання вагових коефіцієнтів попередньо навченої моделі часто може призвести до скорочення часу навчання, отримання більш точних результатів і меншої кількості навчальних даних. Даний метод є надзвичайно ефективним, особливо якщо навчена мережа застосовується для подібної задачі і тренувалась на схожих даних. [5, 6, 8, 12, 15]

Саме тому була обрана нейронна мережа YOLO, оскільки вона вже вміє виявляти такі об’єкти, як машини, велосипеди, люди тощо з високою ефективністю в режимі реального часу.

2.2 Модель YOLOv8

YOLOv8 значно перевершує своїх попередників за точністю і продуктивністю, YOLOv1 і YOLOv2 зосередилися на швидкості виявлення, але з відносно низькою точністю, YOLOv3 зробив великий крок вперед, покращивши виявлення малих об'єктів за допомогою резистивних блоків і триступеневого виявлення, але ці вдосконалення вимагали більшої обчислювальної потужності. YOLOv4 і YOLOv5 продовжили покращувати точність і швидкість, використовуючи нові архітектурні блоки і методи, такі як мозаїчне навчання і гіперпараметричне автоналаштування, тоді як YOLOv6 і YOLOv7 об'єднали всі ці досягнення і впровадили нові інновації в архітектуру мережі для досягнення більш високої точності виявлення при збереженні високої швидкості, з новими методами обробки функцій і поліпшенням обробки великих об'єктів.

Архітектурні вдосконалення в YOLOv8 також знаменують собою значний відхід від попередніх версій YOLO: попередні версії YOLO (YOLOv1 - YOLOv3) були засновані на даркнеті, тоді як пізніші версії (YOLOv4 - YOLOv5) запровадили CSPDarknet та інші оптимізації. Незважаючи на нововведення, YOLOv8 використовує найновіші архітектурні блоки, такі як трансформатори та вдосконалені згорткові шари. Це ще більше розширює можливості узагальнення та підвищує стійкість до варіацій даних, що є ключовим для ефективного виявлення об'єктів у різноманітних середовищах.

Ранні версії YOLO (YOLOv1 - YOLOv3) були відносно швидкими, але вимагали значних обчислювальних ресурсів, тоді як YOLOv4 і YOLOv5 зробили крок вперед в оптимізації для обмежених ресурсів, але все ще ставили

високі вимоги до продуктивності обладнання. YOLOv6 і YOLOv7 продовжували зменшувати необхідні ресурси, але YOLOv8 зробив наступний крок, оптимізувавши модель для роботи на різних платформах, включаючи мобільні пристрої та вбудовані системи. Це було досягнуто завдяки новим способам зниження обчислювальних витрат і оптимізації використання пам'яті.

Нарешті, з виходом YOLOv8 було значно покращено підтримку та інструменти для розробників. Якщо попередні версії (YOLOv1 - YOLOv3) надавали базову підтримку та інструменти для навчання та розгортання моделей, то в YOLOv4 та YOLOv5 було покращено документацію та доступні інструменти, YOLOv6 та YOLOv7 продовжили цю тенденцію, а YOLOv8 включає найсучасніші інструменти для навчання, валідації та розгортання моделей. Активна спільнота та підтримка роблять його ще більш привабливим для розробників, надаючи доступ до багатьох ресурсів для навчання та розгортання моделей. [3, 4, 14]

Таким чином, YOLOv8 є найкращою версією для сучасних програм виявлення об'єктів зі значними покращеннями в точності, швидкості, ефективності використання ресурсів, гнучкості налаштування та підтримці розробників.

Для подальшої роботи із зрозумілих причин було обрано найновішу версію – YOLOv8.

Ultralytics YOLOv8 - це не просто ще одна модель виявлення об'єктів; це універсальний фреймворк, розроблений для охоплення всього життєвого циклу моделей машинного навчання - від отримання даних і навчання моделі до валідації, розгортання та відстеження в реальному світі. Кожен режим

слугує певній меті і розроблений таким чином, щоб запропонувати гнучкість та ефективність, необхідну для різних завдань і варіантів використання.

2.3 Навчання моделі YOLOv8

Тренування моделі глибокого навчання передбачає надання їй даних і налаштування її параметрів таким чином, щоб вона могла робити точні прогнози. Режим навчання в Ultralytics YOLOv8 призначений для ефективного та результативного навчання моделей виявлення об'єктів, повністю використовуючи сучасні апаратні можливості.

Навчання мережі YOLOv8 виявляти та розпізнавати об’єкт типу танк було реалізоване в декілька кроків:

- ✓ підготовка власного набору даних для навчання мережі;
- ✓ власне навчання.

2.3.1 Підготовка власного набору даних для навчання мережі

Підготовка власного набору даних здійснювалась в декілька етапів:

- ✓ підбір відео-роликів з об’єктом;
- ✓ попередня підготовка відео для подальшої обробки: в разі необхідності зміна розширення; розбиття на кадри;
- ✓ розмітка кадрів з об’єктом.

Підбір відео здійснювався в мережі інтернет та інформаційних каналах. В результаті було підібрано 3 відео-ролики з об’єктом типу танк.

Для розбиття підібраних відео на кадри в середовищі Python було створено два прості коди, які швидко це зробили (див. додаток А, додаток Б).

Наступним кроком була розмітка отриманих кадрів.

Для того, щоб правильно розмітити об’єкти та зберегти в потрібному форматі було використано таке середовище, як CVAT (Computer Vision Annotation Tool). CVAT підтримує керовані завдання машинного навчання, що стосуються виявлення об'єктів, класифікації зображень, сегментації зображень та анотування 3D-даних. Він дозволяє користувачам анотувати зображення за допомогою різних інструментів (квадрати, багатокутники, кубоїди, кола, тощо).

У даному середовищі було створено клас “tank” та завантажено всі зображення, які були отриманні під час розбиття відео. На кожному кадрі було розмічено об’єкт (танк) (рис 2.2) та збережено у потрібному форматі (COCO) для подальшого тренування (рис 2.3).

Рис. 2.2 Приклад розмітки в середовищі CVAT

```
0 0.435523 0.571052 0.294859 0.212104
```

Рис. 2.3 Приклад розмітки в текстовому форматі

На першому місці знаходиться номер класу, який використовувався, далі прописані нормалізовані координати самої розмітки, які будуть однаковими незалежно від формату розширення.

2.3.2 Власне навчання

Після того, як були розмічені всі необхідні об’єкти на всіх кадрах, було використано ряд скриптів, за допомогою яких і відбувалось навчання нейронної мережі.

На самому початку було використано скрипт, за допомогою якого відбувалась конвертація формату COCO до формату YOLO (див. додаток В).

Далі було використано ще один скрипт, в якому задається шлях до всіх зображень і анотацій для того, щоб створилась папка із всіма необхідними директоріями для кожного кадру (див. додаток Г).

Потім було використано скрипт, за допомогою якого і проводилось тренування моделі (рис. 2.4).

```
from ultralytics import YOLO


model = YOLO('yolov8n.pt')

results = model.train(data='config.yaml', epochs=150, imgsz=640, name="train_detection", patience=30,
                      device=[0, 1], batch=16, iou=0.3)
```

Рис. 2.4 Скрипт для тренування моделі YOLOv8

Даний скрипт включає в себе попередньо обрану модель тренування, у нашому випадку це YOLOv8n. Була обрана саме ця модель, тому що вона має достатньо маленький розмір і кількість даних, які будуть оброблятись, також небагато.

У параметрі “config” прописано всі шляхи, де знаходяться необхідні дані для обробки та тренування моделі.

Параметр “epochs” вказує на кількість на кількість епох.

Параметр “imgsz” - розмір зображення, що використовується як вхідні дані для моделі YOLO. За замовчуванням це 640.

Параметр “patience” значно пришвидшує тренування моделі. У даному випадку якщо оцінка моделі не покращується (на основі 'Patience=30') протягом останніх 30 епох, навчальний екземпляр переривається після завершення епохи.

Параметр “batch” вказує на розмір батчу.

Параметр “iou” використовується для оцінки алгоритму виявлення об'єктів. Це перекриття між базовою істиною та прогнозованою областю, тобто він обчислює, наскільки прогнозована область схожа на базову істину.

При запуску даного скрипту на сервері створюється віртуальне середовище і встановлюється відповідна бібліотека для YOLOv8.

Далі прописано всі параметри, які використовувалися під час тренування моделі. Час тренування моделі залежить від параметрів комп’ютера, на якому проводиться тренування (рис. 2.5).

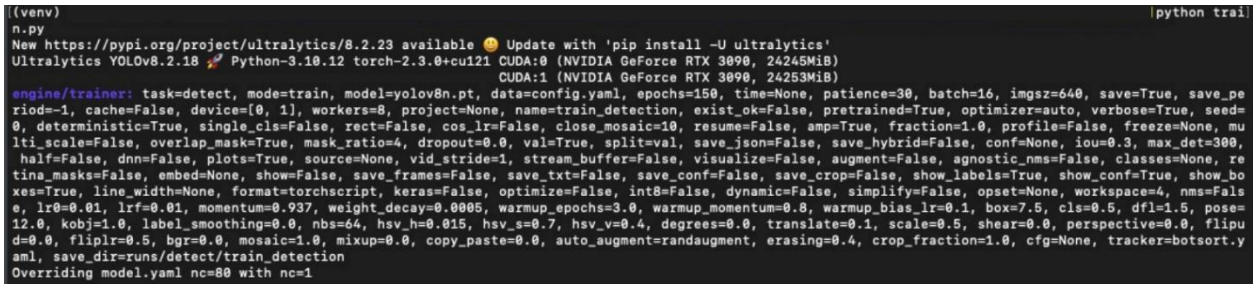


Рис. 2.5 Параметри тренування моделі на сервері

Аугментація є важливим аспектом навчання зображень для задач класифікації, виявлення та сегментації. Вони допомагають додати значущі доповнення до набору даних, застосовуючи візуальні перетворення до існуючих зображень, таким чином збільшуючи набір даних з мінімальними витратами.

Через те, що було використано стандартний скрипт для тренування моделі YOLO, процес аугментації воно провело самостійно (рис. 2.6). Тобто воно робить різні перетворення, об'єднуючи та обрізаючи кадри, на яких проводилось тренування. При цьому розмітка сама адаптується під ці кадри. Із рисунку бачимо, що не зважаючи на те, що кадр висвітлено, затемнено чи розмито, розмітка спрацювала добре та виділа той об'єкт, який потрібно.

Рис. 2.6 Аугментація YOLO

Стандартні шари, які використовувались під час тренування моделі, вказано на рис. 2.7.

```
                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 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    148224  ultralytics.nn.modules.block.C2f             [384, 128, 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     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22        [15, 18, 21]  1    751507  ultralytics.nn.modules.head.Detect           [1, [64, 128, 256]]
Model summary: 225 layers, 3011043 parameters, 3011027 gradients, 8.2 GFLOPs
```

Рис. 2.7 Стандартні шари, які використовувались під час тренування моделі

Власне процес навчання моделі показано на рис. 2.8.

```
 Got processor for bboxes, but no transform to process it.
  warnings.warn(f"Got processor for {proc.default_data_name}, but no transform to process it.")
Plotting labels to runs/detect/train_detection/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.000714, 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 16 dataloader workers
Logging results to runs/detect/train_detection
Starting training for 150 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  0%|          | 0/40 [00:00<?, ?it/s]                                                 /venv/lib/python3.10/site-packages/torch/nn/modul
es/conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: C
UDNN_STATUS_NOT_SUPPORTED (Triggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
                                                       /venv/lib/python3.10/site-packages/torch/nn/modules/conv.py:456: UserWarning: Plan fail
ed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered i
nternally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
      1/150      1.37G      1.107      2.132       1.29         18        640: 100%|██████████| 40/40 [00:02<00:00, 15.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:00<00:00, 21.04it/s]
                   all         70         68          1      0.851      0.995      0.855

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/150      1.33G     0.7868      1.168      1.035         12        640: 100%|██████████| 40/40 [00:01<00:00, 22.06it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:00<00:00, 41.48it/s]
                   all         70         68      0.983          1      0.994      0.851

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/150      1.33G     0.7178       1.04     0.9996         16        640: 100%|██████████| 40/40 [00:01<00:00, 22.26it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:00<00:00, 43.74it/s]
                   all         70         68      0.981          1      0.987      0.824
```

Рис. 2.8 Процес навчання моделі YOLOv8

Через те, що дана модель донавчалась, вона вже на перших епохах показала високий результат. Загальні результати навчання наведено на рис. 2.9.

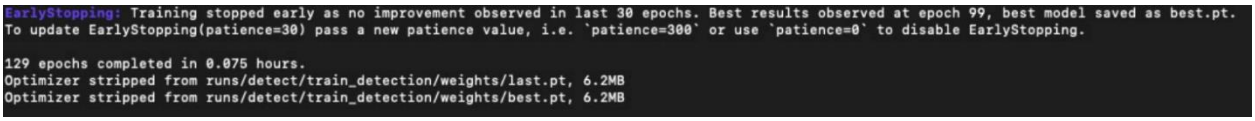


Рис. 2.9 Результат навчання

Із рисунку бачимо, що на 99 епосі було показано найкращі результати, які і будуть використовуватись в моделі. Також бачимо, що час тренування склав 4.5 хвилин, через те, що під час тренування використовувались дві потужні відеокарти.

2.4 Висновки до розділу

Отже, спираючись на наявні в галузі нейронних мереж методи та прийоми, було обрано та успішно реалізовано метод навчання з перенесенням, який дозволяє ефективно використовувати знання, отримані під час виконання завдання, для вирішення подібних задач. Застосування методу transfer learning дозволило скоротити час навчання моделі та зменшило потребу у великих обсягах навчальних даних.

Для підготовки власного набору даних було використано середовище CVAT, яке забезпечило ефективну розмітку цифрового відео.

Також використовувалась аугментація, яка включає в себе широкий спектр методів, таких як зміна яскравості, контрастності, обертання, масштабування та інші перетворення зображень. Ці методи допомогли забезпечити більшу різноманітність вхідних даних, що дозволило моделі

навчитися розпізнавати танки за різних умов освітлення та перспективи, а також за наявності шуму та інших артефактів.

Таким чином, аналізуючи результати, отримані під час тренування моделі, використання YOLOv8n і transfer learning є розумним вибором для ефективного і точного виявлення об'єктів нового класу, адаптованого до конкретних потреб завдання.

РОЗДІЛ 3. Тестування моделі

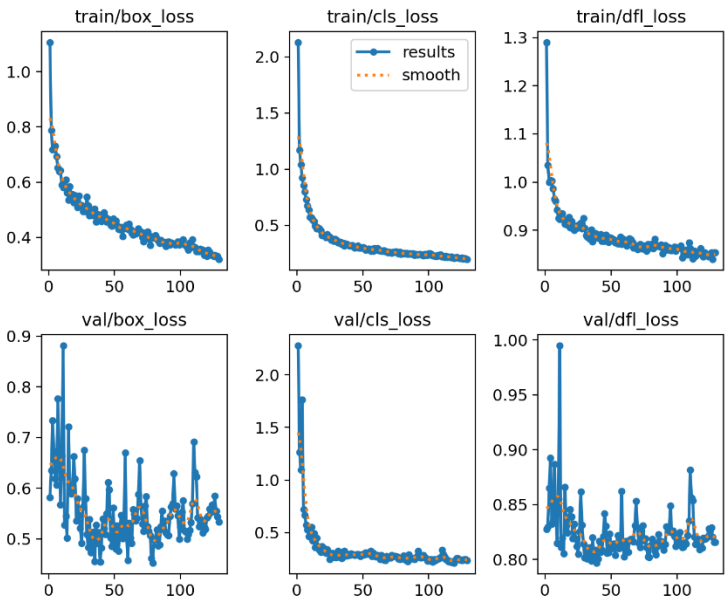
Після навчання моделі важливим кроком є аналіз результатів навчання та перевірка ефективності моделі. Оцінка результатів навчання проводиться на основі аналізу різних метрик. Тестування навченої моделі дозволяє оцінити її здатність точно ідентифікувати об'єкти за новими та невідомими даними, а також виявити можливі недоліки та вдосконалення. Необхідно перевірити, наскільки добре модель працює в реальних умовах, і зробити необхідні висновки для подальшого вдосконалення алгоритму.

3.1 Аналіз метрик

Як було сказано вище, тренування моделі глибокого навчання передбачає надання їй даних і налаштування її параметрів так, щоб вона могла робити точні прогнози. Режим навчання в Ultralytics YOLOv8 призначений для ефективного та результативного навчання моделей виявлення об'єктів, повністю використовуючи сучасні апаратні можливості.

Валідація – це важливий крок у конвеєрі машинного навчання, який дозволяє оцінити якість навчених моделей. Режим валідації в Ultralytics YOLOv8 надає надійний набір інструментів і метрик для оцінки продуктивності моделей виявлення об'єктів.

Спершу проаналізуємо метрики втрат, які були отримані окремо для тренувальної та валідаційної вибірки (рис. 3.1).



а) б) в)

Рис. 3.1 Графіки втрат під час тренування

Горизонтальна вісь графіка – це кількість епох. Епоха - це повна ітерація навчального набору даних, що проходить через нейронну мережу під час навчання. Більша кількість епох зазвичай призводить до кращого навчання моделі, але також може призвести до перенавчання, якщо їх робити надмірно.

Вертикальна вісь відображає значення втрат. Втрати - це міра того, наскільки добре працює модель під час навчання. Менші значення втрат свідчать про кращу роботу моделі, а більші - про гіршу. Метою навчання є мінімізація значення втрат.

Графіки «box_loss» (рис. 3.1 а): це втрата регресії граничної області, яка вимірює похибку в прогнозованих координатах і розмірах граничної області порівняно з істинними даними. Чим менше значення box_loss, тим точнішими є прогнозовані граничні області. Із графіку бачимо, що поступово це значення зменшується, а отже результати достатньо точні.

Графіки «cls_loss» (рис. 3.1 б): це втрата класифікації, яка вимірює похибку в прогнозованій ймовірності класу для кожного об'єкта на зображенні порівняно з істиною. Чим менше значення cls_loss, тим точніше модель прогнозує клас об'єктів. Із графіку помічаємо зменшення цього значення, а отже модель точно прогнозує клас об’єктів.

Графіки «dfl_loss» (рис. 3.1 в): це втрата деформівного шару згортки. Ця втрата вимірює похибку деформівних шарів згортки, які призначені для покращення здатності моделі виявляти об'єкти з різними масштабами та співвідношенням сторін. Нижчий показник dfl_loss означає, що модель краще справляється з деформаціями об'єктів та змінами їхнього зовнішнього вигляду. З графіку видно зменшення цього параметру, а отже модель досить точно розпізнає об’єкти з можливими дефектами.

Загальне значення всіх цих втрат, як правило, є зваженою сумою цих окремих втрат. Конкретні одиниці вертикальної осі залежать від реалізації, але загалом вони відображають величину помилки або різницю між прогнозованим і фактичним значеннями.

Проте, найбільшу увагу звертають на графіки «precision» та «recall» (рис. 3.2). «Precision» - це точність виявлених об'єктів, що вказує на те, скільки виявлених об'єктів були правильними. «Recall» - це здатність моделі ідентифікувати всі екземпляри об'єктів на зображеннях. Із цих графіків помітно, що модель швидко почала процес перенавчання (від 70% приблизно через 20 епох перейшла до 90%).

Показник F1 - це міра точності моделі, яка враховує як точність, так і відтворення. Це середнє гармонійне значення точності та пригадування.

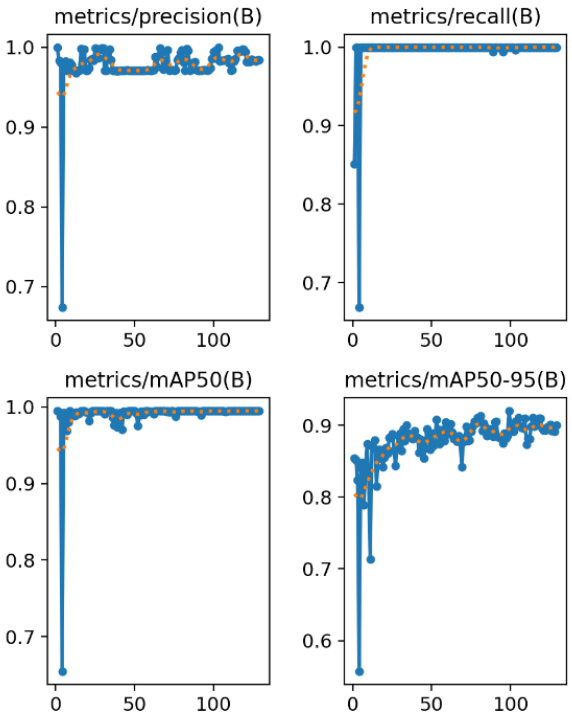


Рис. 3.2 Графіки «precision» та «recall»

Крива довіри F1 відображає оцінку F1 в залежності від різних довірчих порогів. Вищий пік свідчить про кращу продуктивність моделі. Із графіку (рис. 3.3) помічаємо, як робота моделі стабільно тримається на 90%.

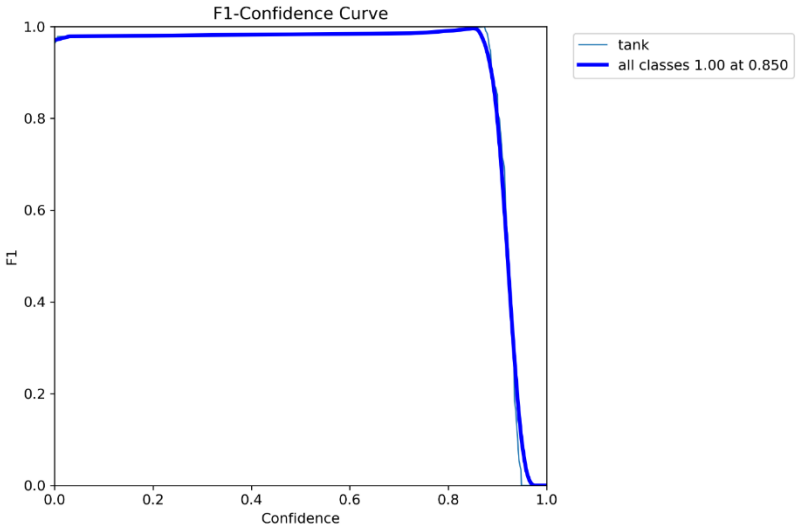


Рис. 3.3 Графік «F1 - Confidence»

Крива «точність - довірча ймовірність» показує значення точності при різних довірчих інтервалах. Починаючи з високого довірчого порогу, точність буде відносно високою, оскільки тільки достовірні прогнози вважаються дійсними. Графік показує, як змінюється точність з різними рівнями довіри. В ідеалі потрібна висока точність на всіх рівнях довіри, що і помітно із отрманого графіку (рис. 3.4).

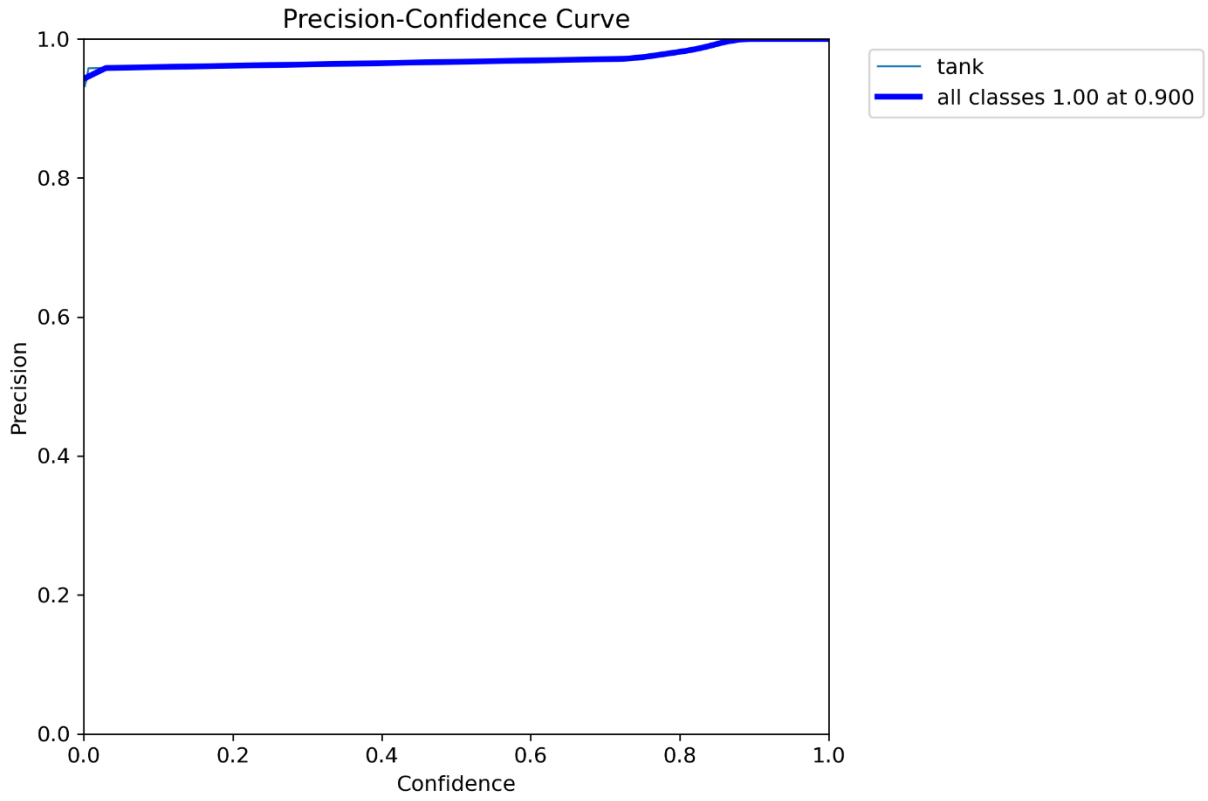


Рис. 3.4 Графік «Precision-Confidence»

Крива довіри до згадування показує відкликання при різних довірчих інтервалах. В іделі потрібен високий рівень пригадування в усіх сферах. Із графіку помічаємо, що модель працює на 90% (рис. 3.5).

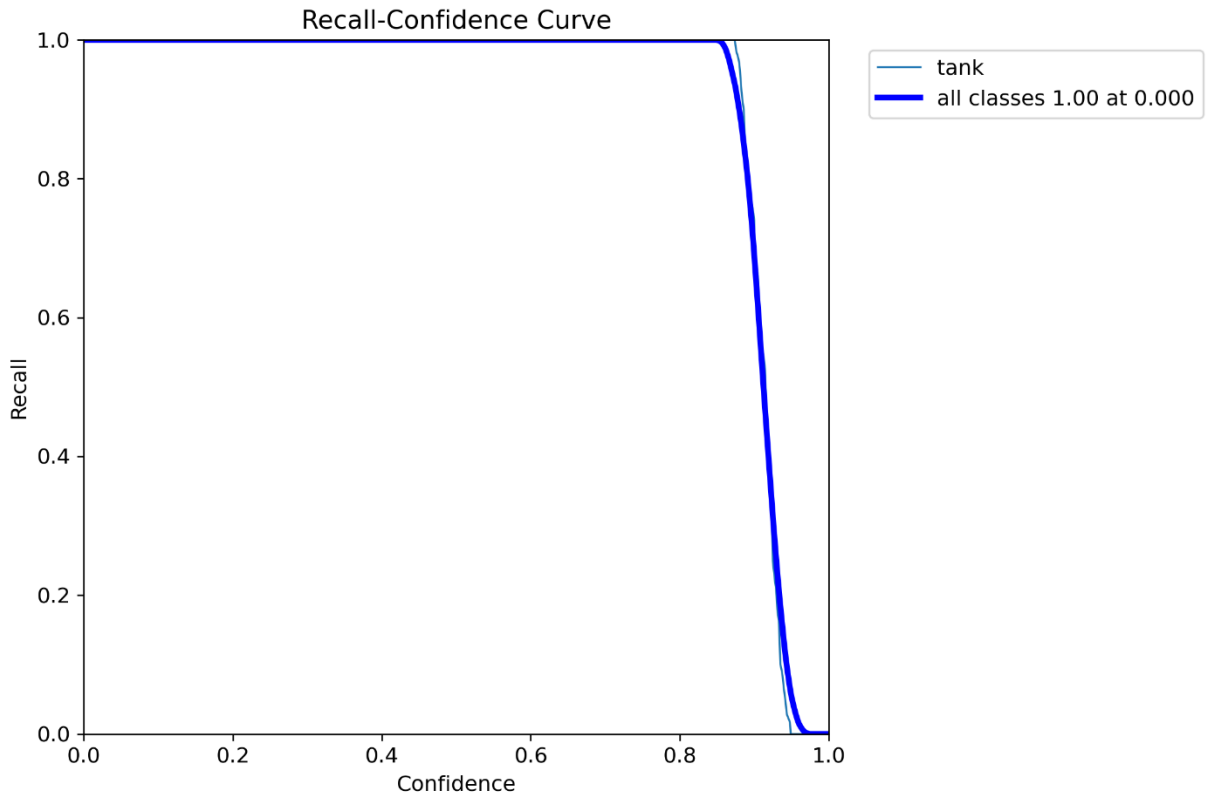


Рис. 3.5 Графік «Recall-Confidence»

Крива «точність-відтворення» показує компроміс між точністю та відтворенням для різних порогових значень. Висока область під кривою відображає як високу пригадування, так і високу точність, де висока точність пов'язана з низьким рівнем помилкових спрацьовувань, а висока пригадування пов'язана з низьким рівнем помилкових негативних спрацьовувань. Вона ілюструє компроміс між точністю та пригадуванням для різних порогових значень. Модель, яка досягає ближче до верхнього правого кута, є кращою. Навчена модель має гарний результат за цією метрикою (рис. 3.6).

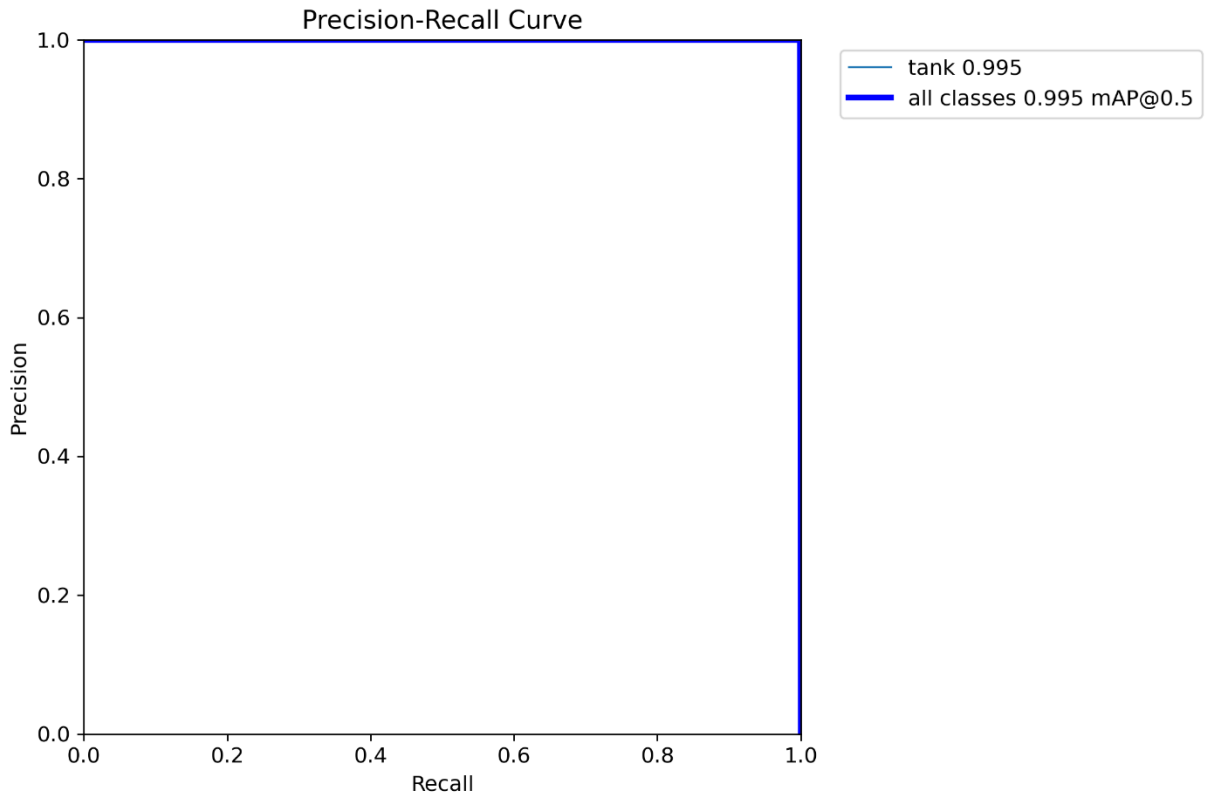


Рис. 3.6 Графік «Precision-Recall»

Окрім графіків та кривих, одним із важливих метрик після тренування моделі є «Confusion matrix» (матриця помилок).

Дана матриця – це фактично таблиця, що показує продуктивність класифікатора при заданих значеннях істинності. Це візуалізація, яка показує, як дана модель працює на класах, на яких вона була навчена. Так як у даній роботі використовувався лише один клас «tank», помічаємо, що модель добре справляється із своїми обов’язками, добре розпізнає потрібний об’єкт та не плутає його із іншими об’єктами на фоні (рис. 3.7).

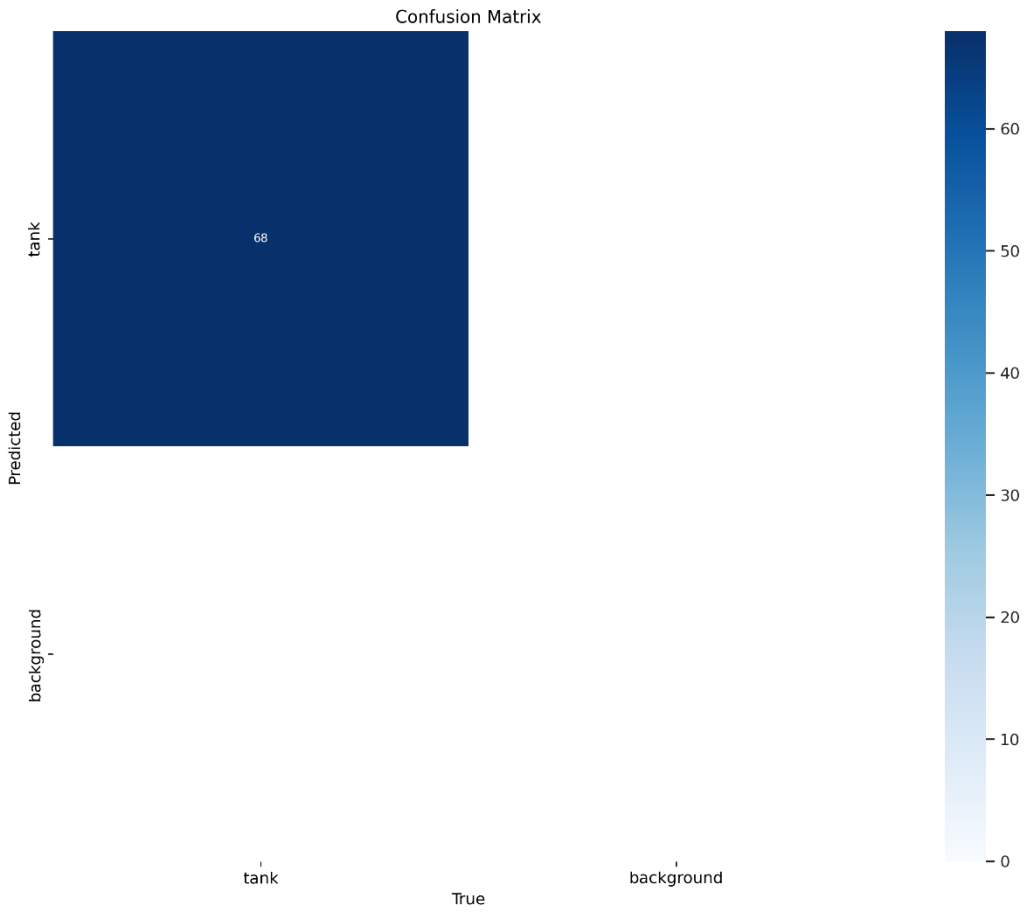


Рис. 3.7 Confusion matrix

3.2 Тестування моделі на відео

Далі тренування проводилось на реальних відео. Це дозволяє оцінювати моделі в більш природних і змінних умовах. Такі тести допоможуть зрозуміти, як модель працює на динамічних зображеннях, таких як рух об'єкта або зміна освітлення, що може вплинути на точність розпізнавання.

Тестування на даних, на яких тренувалась модель, показало непогані результати (рис. 3.8). Із рисунку бачимо, що розмітка спрацювала практично ідеально. Число «0.9» говорить про те, що модель спрацювала фактично на 100%, що підтверджує правильність проведеного тренування.

Рис. 3.8 Тестування на відео

Для максимально точного аналізу та перевірки справності проведеного тренування, модель також була перевірена на інших відео, які не використовувались під час навчання (рис. 3.9)

Рис. 3.9 Тестування на відео

У деяких випадках розмітка спрацювала практично на 90%, проте поставлену задачу дана модель виконує, а саме розпізнає об’єкти з довірністю більше 0,5.

3.3 Висновки до розділу

В даному розділі проаналізовано результати навчання моделі та досліджено ефективність її роботи.

Під час оцінки навчання моделі аналізувалися різні метрики, включаючи точність, повноту, значення F1, матрицю плутанини, які дозволили зрозуміти сильні та слабкі сторони моделі. Графічне представлення результатів дозволило виявити конкретні аспекти, які потребують подальшого вдосконалення.

Дослідження ефективності роботи моделі проводилося на реальних відеоматеріалах, що дозволило протестувати моделі в умовах максимально наближених до реальних сценаріїв.

Загалом, результати роботи моделі повністю відповідають вимогам завдання.

Однак аналіз метрик виявив кілька областей, які можна покращити для подальшого підвищення ефективності моделі. До них відносяться: подальше налаштування гіперпараметрів, збільшення кількості навчальних даних та застосування більш складних методів аугментації.

Результати не тільки дають впевненість у тому, що модель може бути ефективно застосована в реальних умовах, але й демонструють потенціал для подальших досліджень та оптимізації.

ВИСНОВКИ

У ході виконання дипломної роботи на тему "Програмне забезпечення для систем технічного зору на основі нейронної мережі" було досягнуто значних результатів, що підтвердили ефективність розробленої системи.

У першому розділі здійснено аналіз сучасного стану нейронних мереж та технічних візуальних систем. Показано, що нейронні мережі наразі є одним з найперспективніших інструментів для обробки та аналізу візуальної інформації. Також досліджується їх актуальність для систем технічного зору, їх використання в різних галузях та перспективи розвитку. Важливим елементом цього розділу є висвітлення практичного значення цього дослідження шляхом визначення доцільності використання дронів для розпізнаваннявиявлення танків.

У другому розділі описано навчання нейронної мережі для виявлення танків. Метод, який було обрано - це навчання з перенесенням. Трансферне навчання використовує попередньо навчені моделі для досягнення високих результатів за короткий час. Трансферне навчання - це дуже ефективний спосіб швидко адаптувати існуючу модель, навчену на великому наборі даних, до конкретного завдання (в нашому випадку - виявлення танків). Цей метод дозволяє значно скоротити час і обчислювальні ресурси, необхідні для навчання нової моделі з нуля, зберігаючи при цьому високу точність і надійність. Було аргументовано вибір моделі YOLOv8 для автоматизації системи технічного зору. Детально описано кожен крок процесу навчання моделі. Значну увагу приділено етапу підготовки вхідних даних, який включає відбір відеоматеріалу, розбиття відео потоку на кадри та маркування об’єктів за допомогою платформи CVAT. Важливу роль у цьому розділі відіграла

аугментація даних. Використання аугментації зробило модель більш стійкою до варіацій даних і покращило загальну ефективність.

Третій розділ дипломної роботи присвячений аналізу результатів навчання моделі та тестуванню її ефективності. З цією метою були детально розглянуті та проаналізовані основні метрики. Тестування ефективності моделі проводились на реальному відеоматеріалі.

В цілому, результати роботи моделі повністю відповідають вимогам завдання. Крім того, результати роботи підтверджують можливість ефективного використання нейронних мереж у візуальній системі розпізнавання об'єктів, в тому числі танків, за допомогою дронів. Розроблене програмне забезпечення демонструє високу точність та надійність, що робить його перспективним рішенням для військових та цивільних застосувань.

Існує кілька важливих напрямків для подальших досліджень. По-перше, вдосконалення алгоритму виявлення, зокрема експерименти з різними архітектурами нейронних мереж та оптимізація гіперпараметрів моделі. Це дозволить ще більше підвищити точність і швидкість роботи системи. По-друге, розширення навчальної бази та збір більшої кількості відеоматеріалів та зображень з різних умов та ракурсів зйомки. Це дозволить моделі краще реагувати на різні сценарії в реальному світі. По-третє, інтегрувати систему в реальні умови експлуатації, включаючи безпілотники, та протестувати її в польових умовах. Це допоможе виявити можливі недоліки та покращити адаптивність моделі до реальних завдань. Також необхідно, розширити можливості та сферу застосування розробленої системи, застосувавши її для інших типів об'єктів. Наприклад, систему можна налаштувати на розпізнавання інших військових об'єктів, цивільних транспортних засобів або навіть об'єктів у сільськогосподарському секторі.

Таким чином, це дослідження має потенціал зробити значний внесок у різні галузі завдяки ефективному використанню методів комп’ютерного зору та нейронних мереж.

СПИСОК ЛІТЕРАТУРИ

1. C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook. Cham, Switzerland: Springer, 2018.
2. I.J. Intelligent Systems and Applications, 2022, 6, 50-62.
3. J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv preprint arXiv:1804.02767, Apr. 2018.
4. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 2016, pp. 779-788.
5. K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," Journal of Big Data, vol. 3, no. 1, p. 9, May 2016.
6. L. Torrey and J. Shavlik, "Transfer learning," in Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, I. Global, Ed. Hershey, PA, USA: IGI Global, 2010, pp. 242-264.
7. R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2018.
8. S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.
9. S. Raschka and V. Mirjalili, Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd ed. Birmingham, UK: Packt Publishing Ltd, 2019.
10. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, May 2015.

11. А. В. Буряк, В. І. Буряк, Нейронні мережі: теорія та практика. Київ, Україна: Вид. дім "Кондор", 2014.
12. А. О. Ковальов і С. В. Мартинюк, "Методи transfer learning для обробки зображень у медичних діагностичних системах," Журнал НАН України, вип. 10, с. 145-152, жовт. 2018.
13. В. П. Потоцький, "Розробка систем технічного зору для автоматизації виробничих процесів," Вісник Харківського національного університету радіоелектроніки, вип. 84, с. 85-92, 2017.
14. І. С. Ковальчук, "Застосування алгоритму YOLO для виявлення об'єктів на відео," Вісник Київського національного університету технологій та дизайну, вип. 3, с. 45-50, 2019.
15. О. О. Кириченко, Машинне навчання і великі дані: методи та алгоритми. Харків, Україна: Видавництво Харківського національного університету, 2018.
16. Ю. В. Хоменко, "Застосування нейронних мереж в прогнозуванні фінансових показників," Науковий вісник НЛТУ України, вип. 24, № 7, с. 134-140, лип. 2014.

ДОДАТОК А

Код для зміни розширення відео

```
import cv2
import os
def crop_video_to_center(video_path, output_path, width=640, height=480):
    # Перевірка, чи існує відео файл
    if not os.path.isfile(video_path):
        raise FileNotFoundError(f"Відео файл за шляхом {video_path} не
знайдено.")

    # Відкриття відео для читання
    video_capture = cv2.VideoCapture(video_path)

    # Отримання характеристик вихідного відео
    original_width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = video_capture.get(cv2.CAP_PROP_FPS)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Використання кодека для
запису

    # Перевірка, чи вихідні розміри більші за бажані
    if original_width < width or original_height < height:
        raise ValueError("Розмір вихідного відео менший за 640x480")

    # Відкриття відео для запису
    video_writer = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    # Визначення координат для обрізки по центру
    x_start = (original_width - width) // 2
    y_start = (original_height - height) // 2

    while True:
        success, frame = video_capture.read()
        if not success:
            break

        # Обрізка кадру по центру
        cropped_frame = frame[y_start:y_start + height, x_start:x_start +
width]

        # Запис кадру у новий файл
        video_writer.write(cropped_frame)

    # Звільнення ресурсів
    video_capture.release()
    video_writer.release()
    print(f"Відео збережено за шляхом {output_path}")


# Приклад виклику функції
video_path = '/Users/bzetr/Downloads/Video1.MP4'
output_path = '/Users/bzetr/Downloads/frames1.MP4'
crop_video_to_center(video_path, output_path)
```

ДОДАТОК Б

Код для розбиття відео на кадри

```
import cv2
import os
def split_video_into_frames(video_path, output_folder, frame_interval=5):
    # Перевірка, чи існує відео файл
    if not os.path.isfile(video_path):
        raise FileNotFoundError(f"Відео файл за шляхом {video_path} не
знайдено.")

    # Перевірка, чи існує папка для збереження кадрів, і створення її, якщо
не існує
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Завантаження відео
    video_capture = cv2.VideoCapture(video_path)

    frame_number = 0
    saved_frame_number = 0
    while True:
        # Зчитування кадру
        success, frame = video_capture.read()
        if not success:
            break

        # Збереження кожного третього кадру
        if frame_number % frame_interval == 0:
            frame_filename = os.path.join(output_folder,
f"frame_{saved_frame_number:04d}.png")
            cv2.imwrite(frame_filename, frame)
            saved_frame_number += 1

        frame_number += 1

    # Звільнення відео файлу
    video_capture.release()
    print(f"Завершено! Кількість збережених кадрів: {saved_frame_number}")

# Приклад виклику функції
video_path = '/Users/bzetr/Downloads/frames1.MP4'
output_folder = '/Users/bzetr/Downloads/frames'
split_video_into_frames(video_path, output_folder)
```

ДОДАТОК В

Код для зміни формату COCO в YOLO

```
import json
import os
import argparse
class COCO2YOLO:
    def __init__(self, json_file, output_folder):
        self.json_file = json_file
        self.output_folder = output_folder
        self._check_file_and_dir(self.json_file, self.output_folder)
        self.labels = json.load(open(self.json_file, 'r', encoding='utf-8'))
        self.coco_id_name_map = self._categories()
        self.coco_name_list = list(self.coco_id_name_map.values())
        print("total images", len(self.labels['images']))
        print("total categories", len(self.labels['categories']))
        print("total labels", len(self.labels['annotations']))

    def _check_file_and_dir(self, file_path, dir_path):
        if not os.path.exists(file_path):
            raise ValueError("File not found")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)

    def _categories(self):
        categories = {}
        for cls in self.labels['categories']:
            categories[cls['id']] = cls['name']
        return categories

    def _load_images_info(self):
        images_info = {}
        for image in self.labels['images']:
            id = image['id']
            file_name = image['file_name']
            if file_name.find('\\') > -1:
                file_name = file_name[file_name.index('\\')+1:]
            w = image['width']
            h = image['height']
            images_info[id] = (file_name, w, h)
        return images_info

    def _bbox_2_yolo(self, bbox, img_w, img_h):
        x, y, w, h = bbox[0], bbox[1], bbox[2], bbox[3]
        centerx = bbox[0] + w / 2
        centery = bbox[1] + h / 2
        dw = 1 / img_w
        dh = 1 / img_h
        centerx *= dw
        w *= dw
        centery *= dh
        h *= dh
        return centerx, centery, w, h
```

```
    def _convert_anno(self, images_info):
        anno_dict = dict()
        for anno in self.labels['annotations']:
            bbox = anno['bbox']
            image_id = anno['image_id']
            category_id = anno['category_id']

            image_info = images_info.get(image_id)
            image_name = image_info[0]
            img_w = image_info[1]
            img_h = image_info[2]
            yolo_box = self._bbox_2_yolo(bbox, img_w, img_h)

            anno_info = (image_name, category_id, yolo_box)
            anno_infos = anno_dict.get(image_id)
            if not anno_infos:
                anno_dict[image_id] = [anno_info]
            else:
                anno_infos.append(anno_info)
                anno_dict[image_id] = anno_infos
        return anno_dict

    def save_classes(self):
        sorted_classes = list(map(lambda x: x['name'],
sorted(self.labels['categories'], key=lambda x: x['id'])))
        print('coco names', sorted_classes)
        with open('coco.names', 'w', encoding='utf-8') as f:
            for cls in sorted_classes:
                f.write(cls + '\n')

    def coco2yolo(self):
        print("loading image info...")
        images_info = self._load_images_info()
        print("loading done, total images", len(images_info))

        print("start converting...")
        anno_dict = self._convert_anno(images_info)
        print("converting done, total labels", len(anno_dict))

        print("saving txt file...")
        self._save_txt(anno_dict)
        print("saving done")

    def _save_txt(self, anno_dict):
        for k, v in anno_dict.items():
            file_name = os.path.splitext(v[0][0])[0] + ".txt"
            with open(os.path.join(self.output_folder, file_name), 'w',
encoding='utf-8') as f:
                print(k, v)
                for obj in v:
                    cat_name = self.coco_id_name_map.get(obj[1])
                    category_id = self.coco_name_list.index(cat_name)
                    box = ['{:.6f}'.format(x) for x in obj[2]]
                    box = ' '.join(box)
                    line = str(category_id) + ' ' + box
                    f.write(line + '\n')
```

```
if __name__ == '__main__':
    # parser = argparse.ArgumentParser(description='Test yolo data.')
    # parser.add_argument('-j', help='JSON file', dest='json', required=True)
    # parser.add_argument('-o', help='path to output folder', dest='out',
required=True)
    # args = parser.parse_args()
    #
    # c2y = COCO2YOLO(args.json, args.out)
    # c2y.coco2yolo()
    coco_ann_path = None
    yolo_ann_path =
"/Volumes/Kingston2tb/work_flow/etc/trackNet_data/test_2/data/yolo_annotation
"
    os.makedirs(yolo_ann_path, exist_ok=True)
    c2y = COCO2YOLO(coco_ann_path, yolo_ann_path)
    c2y.coco2yolo()
```

ДОДАТОК Г

Код для створення директорій

```
import os
import shutil


def sort_files_by_index(file_list):
    # Function to sort the file list by index
    return sorted(file_list, key=lambda x:
int((x.split('_')[1]).split(".")[0]))


def split_images_and_annotations(images_path, yolo_annotation, paths,
percentage, start_index):
    # Get a list of all images in the images_path directory and sort them by
index
    images_list = sort_files_by_index(os.listdir(images_path))

    # Calculate the number of images to move to the validation set
    total_images_count = len(images_list)
    val_images_count = int(total_images_count * percentage / 100)

    # Select images for the validation set starting from the start_index
    val_images = images_list[start_index:start_index + val_images_count]

    # Move selected images and annotations to the validation set
    for image in val_images:
        image_path = os.path.join(images_path, image)
        annotation_path = os.path.join(yolo_annotation, image.split('.')[0] +
'.txt')
        shutil.move(image_path, paths['val_images'])
        shutil.move(annotation_path, paths['val_labels'])

    # Move the rest of the images and annotations to the training set
    for image in images_list[start_index + val_images_count:]:
        image_path = os.path.join(images_path, image)
        annotation_path = os.path.join(yolo_annotation, image.split('.')[0] +
'.txt')
        shutil.move(image_path, paths['train_images'])
        shutil.move(annotation_path, paths['train_labels'])

    print("Images and annotations splitted successfully!")


def create_yolo_structure(yolo_structure):
    # Create a 'data' folder in the yolo_structure directory
    data_folder = os.path.join(yolo_structure, 'data')
    if not os.path.exists(data_folder):
        os.makedirs(data_folder)

    # Create 'images' and 'labels' folders within the 'data' folder
    images_folder = os.path.join(data_folder, 'images')
    labels_folder = os.path.join(data_folder, 'labels')
```

```
    if not os.path.exists(images_folder):
        os.makedirs(images_folder)
    if not os.path.exists(labels_folder):
        os.makedirs(labels_folder)

    # Create 'train' and 'val' folders within 'images' and 'labels' folders
    train_images_folder = os.path.join(images_folder, 'train')
    val_images_folder = os.path.join(images_folder, 'val')
    train_labels_folder = os.path.join(labels_folder, 'train')
    val_labels_folder = os.path.join(labels_folder, 'val')
    if not os.path.exists(train_images_folder):
        os.makedirs(train_images_folder)
    if not os.path.exists(val_images_folder):
        os.makedirs(val_images_folder)
    if not os.path.exists(train_labels_folder):
        os.makedirs(train_labels_folder)
    if not os.path.exists(val_labels_folder):
        os.makedirs(val_labels_folder)

    # Return paths to 'train' and 'val' folders in 'images' and 'labels'
    return {
        'train_images': train_images_folder,
        'val_images': val_images_folder,
        'train_labels': train_labels_folder,
        'val_labels': val_labels_folder
    }
```