

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний університет кораблебудування імені адмірала Макарова**

Навчально-науковий інститут комп'ютерних наук та управління проектами  
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" \_\_\_\_ " \_\_\_\_\_ 2024 р.

***КВАЛІФІКАЦІЙНА РОБОТА***

**на здобуття ступеня вищої освіти "магістр"**

**на тему: Розробка та впровадження системи відеоспостереження на основі  
нейронних мереж YOLO та методів трекінгу для детекції та відстеження  
об'єктів у реальному часі.**

Виконав: студент групи 6141 м

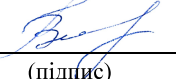
  
(підпис)

**Зубар Ю.В.**

Керівник роботи:

доцент кафедри ІУСТ, PhD,

(посада, науковий ступінь вчене звання)

  
(підпис)

**Ворона М.В.**

м. Миколаїв – 2024 р.

# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами

Кафедра інформаційних управляючих систем та технологій

Спеціальність 122 "Комп'ютерні науки"

Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

В. Ажищев

к.т.н, доц. Ажищев В.Ф.

"\_\_\_" \_\_\_\_\_ 2024 р.

## **ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"**

Студенту Зубарю Юрію Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка та впровадження системи відеоспостереження на основі нейронних мереж YOLO та методів трекінгу для детекції та відстеження об'єктів у реальному часі.

Керівник роботи Ворона Михайло Владиславович

Затверджені наказом ректора № 1055-уч. від "08" жовтня 2024 року.

2. Термін подання роботи: 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів)

Теоретичні основи нейронних мереж та їх сучасні архітектури.

Алгоритми детекції об'єктів та комп'ютерний зір: сучасні підходи та застосування.

Еволюція, розвиток та основні характеристики YOLO: від YOLOv3 до YOLOv11 у завданнях комп'ютерного зору.

Розробка програмного забезпечення для відеоспостереження з використанням нейронних мереж YOLO та методів трекінгу DEEPSORT.

5. Перелік презентаційних матеріалів \_\_\_\_\_



## 6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата   |                  |
|--------|-------------------------------------------|----------------|------------------|
|        |                                           | завдання видав | завдання прийняв |
| 1      | Ворона М.В.                               | 05.09.2023     | 22.12.2023       |
| 2      | Ворона М.В.                               | 23.12.2023     | 15.06.2024       |
| 3      | Ворона М.В.                               | 15.06.2024     | 25.09.2024       |
| 4      | Ворона М.В.                               | 25.09.2024     | 14.11.2024       |
| 5      | Ворона М.В.                               | 25.09.2024     | 14.11.2024       |
| 6      | Ворона М.В.                               | 25.09.2024     | 14.11.2024       |

## 7. Дата видачі завдання 05 вересня 2023 р.

### КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи)                       | Строк виконання етапів роботи | Примітка |
|-------|----------------------------------------------------------------|-------------------------------|----------|
| 1     | Дослідження та аналіз предметної галузі                        | 22.12.2024                    |          |
| 2     | Розробка концепції                                             | 15.03.2024                    |          |
| 3     | Розділ з розробки проектних рішень по системі (та її частинах) | 15.06.2024                    |          |
| 4     | Реалізація проекту та розробка робочої документації            | 25.09.2024                    |          |
| 5     | Обчислення економічної ефективності                            | 14.11.2024                    |          |
| 6     | Вирішення питань охорони праці та навколишнього середовища     | 14.11.2024                    |          |
| 7     | Оформлення пояснювальної записки                               | 14.11.2024                    |          |
| 8     | Подання магістерської роботи на перевірку на плагіат           | 16.11.2024                    |          |
| 9     | Подання магістерської роботи рецензенту                        | 20.11.2024                    |          |
| 10    | Подання магістерської роботи на державний захист               | 15.12.2024                    |          |

Студент

  
(підпис)

**Зубар Ю. В.**

(ПІБ)

Керівник роботи

  
(підпис)

**Ворона М.В.**

(ПІБ)

## АНОТАЦІЯ

Сучасні технології штучного інтелекту та комп'ютерного зору широко застосовуються в різних сферах, таких як безпека, медицина та автономний транспорт. Серед найефективніших методів для розпізнавання та класифікації об'єктів виділяються алгоритми глибокого навчання, зокрема YOLO та Faster R-CNN. Це дослідження присвячене розробці інтелектуальної системи відеоспостереження, що використовує алгоритм YOLOv8 для виявлення об'єктів та метод Deep Tracking для їхнього відстеження в режимі реального часу. Основна мета роботи — створення високопродуктивної системи, здатної оперативно аналізувати відеопотік у реальних умовах. Запропонований підхід сприяє підвищенню точності та швидкості обробки відеоданих, що є важливим для безпекових і автоматизованих рішень.

**Ключові слова:** система відео спостереження; нейронні мережі; YOLO (You Only Look Once); детекція об'єктів; відстеження об'єктів; інтелектуальні системи відео спостереження; обробка зображень; системи автоматичного моніторингу; аналіз відеоданих; інтеграція систем відеоспостереження.

## ABSTRACT

Modern technologies of artificial intelligence and computer vision are widely used in various fields, such as security, medicine and autonomous transportation. Among the most effective methods for object recognition and classification, deep learning algorithms stand out, in particular YOLO and Faster R-CNN. This research is devoted to the development of an intelligent video surveillance system that uses the YOLOv8 algorithm for object detection and the Deep Tracking method for their real-time tracking. The main goal of the work is to create a high-performance system capable of quickly analyzing a video stream in real conditions. The proposed approach helps to increase the accuracy and speed of video data processing, which is important for security and automated solutions.

**Keywords:** video surveillance system; neural networks; YOLO (You Only Look Once); object detection; object tracking; intelligent video surveillance systems; image processing; automatic monitoring systems; video data analysis; video surveillance system integration.

## ЗМІСТ

|                                                                                                                   |    |
|-------------------------------------------------------------------------------------------------------------------|----|
| ВСТУП.....                                                                                                        | 5  |
| РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ ТА ЇХ СУЧАСНІ АРХІТЕКТУРИ                                             |    |
| 1.1 Загальна характеристика нейронних мереж та історія розвитку нейронних мереж.....                              | 8  |
| 1.2 Аналіз і порівняння функцій активації в нейронних мережах. Принципи, переваги та обмеження.....               | 11 |
| 1.3 Етапи навчання нейронних мереж: методи та алгоритми.....                                                      | 16 |
| 1.4 Алгоритми оптимізації: Принципи, різновиди та застосування у нейронних мережах.....                           | 17 |
| 1.5 Перенавчання в нейронних мережах: проблеми, причини та методи подолання.....                                  | 22 |
| 1.6 Огляд архітектур основних нейронних мереж (CNN, RNN, GNN).....                                                | 25 |
| РОЗДІЛ 2. АЛГОРИТМИ ДЕТЕКЦІЇ ОБ’ЄКТІВ ТА КОМП’ЮТЕРНИЙ ЗІР: СУЧАСНІ ПІДХОДИ ТА ЗАСТОСУВАННЯ                        |    |
| 2.1 Комп’ютерний зір: історія, основні компоненти, методології та виклики.....                                    | 36 |
| 2.2 Огляд сучасних алгоритмів детекції об’єктів: традиційні алгоритми, глибоке навчання та трансформери.....      | 41 |
| 2.3 Реальні застосування алгоритмів детекції об’єктів.....                                                        | 50 |
| 2.4 Тенденції та майбутнє розвитку алгоритмів детекції об’єктів.....                                              | 54 |
| РОЗДІЛ 3. ЕВОЛЮЦІЯ, РОЗВИТОК ТА ОСНОВНІ ХАРАКТЕРИСТИКИ YOLO: ВІД YOLOv3 ДО YOLOv11 У ЗАВДАННЯХ КОМП’ЮТЕРНОГО ЗОРУ |    |
| 3.1 YOLOv3: важливий етап у розвитку YOLO.....                                                                    | 57 |
| 3.2 YOLOv4: оптимізація для продуктивності.....                                                                   | 58 |
| 3.3 YOLOv5: простота використання і інтеграції.....                                                               | 60 |
| 3.4 YOLOv6: промислові застосування та оптимізація.....                                                           | 62 |
| 3.5 YOLOv7: продуктивність та оптимізація.....                                                                    | 63 |
| 3.6 YOLOv8: покращенні роботи на пристроях з обмеженими                                                           |    |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| ресурсами.....                                                          | 65 |
| 3.7 YOLOv9: у пошуку ідеального алгоритму.....                          | 66 |
| 3.8 YOLOv10: інновації в реальному часі для детекції об'єктів.....      | 67 |
| 3.9 YOLOv11: новітня ітерація детекторів об'єктів у реальному часі..... | 68 |
| РОЗДІЛ 4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ                          |    |
| ВІДЕОСПОСТЕРЕЖЕННЯ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ                      |    |
| YOLO ТА МЕТОДІВ ТРЕКІНГУ DEEPSORT                                       |    |
| 4.1 Технологічне забезпечення та архітектура системи                    |    |
| відеоспостереження.....                                                 | 71 |
| 4.2 Основні функції застосунку.....                                     | 73 |
| ВИСНОВКИ.....                                                           | 79 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....                                         | 81 |
| ДОДАТКИ.....                                                            | 83 |

## **Вступ**

У сучасному світі технології штучного інтелекту (ШІ) та комп'ютерного зору стали невід'ємною частиною багатьох інноваційних галузей, таких як автомобільна промисловість, медицина, безпека, промисловість і навіть розваги. Зростання обчислювальних можливостей і розвиток алгоритмів глибокого навчання призвели до революційних змін у тому, як машини можуть аналізувати та інтерпретувати візуальні дані. Глибоке навчання стало одним із найбільш потужних методів для аналізу складних структур даних та створення інтелектуальних систем. Особливої важливості набули завдання детекції та класифікації об'єктів, оскільки вони забезпечують автоматизацію процесів та підвищують їхню ефективність. Саме ці технології є критичними для створення систем, здатних працювати в реальному часі, наприклад, для автономних транспортних засобів, відеоспостереження чи медичної діагностики.

### **Актуальність теми.**

Сучасні алгоритми детекції та класифікації об'єктів застосовуються для розв'язання реальних завдань у різних сферах, де традиційні методи машинного навчання виявилися малоефективними або надто обмеженими. Однією з найбільш ефективних моделей для обробки зображень і відео в реальному часі є YOLO (You Only Look Once), що пропонує уніфікований підхід до детекції об'єктів. Наприклад, у сфері безпеки системи відеоспостереження повинні не лише знімати відео, а й автоматично аналізувати його для виявлення підозрілих об'єктів або дій, таких як вторгнення на територію або виявлення зброї. У цих випадках швидкість і точність роботи систем відіграють вирішальну роль, і завдяки глибокому навчанню можна досягти значного прогресу.

Інший важливий приклад — це сфера автономних транспортних засобів. Технології детекції об'єктів дозволяють транспортним засобам розпізнавати інші машини, пішоходів, дорожні знаки та перешкоди, забезпечуючи безпечний рух у реальному часі. Алгоритми, такі як Faster R-CNN та YOLO, забезпечують високу швидкість і точність детекції об'єктів, що робить їх важливими для автономних систем. Автономні автомобілі використовують складні системи, які

постійно опрацьовують візуальну інформацію і приймають рішення менше ніж секунду.

У галузі охорони здоров'я комп'ютерний зір став важливим інструментом для медичної діагностики. Застосування нейронних мереж до аналізу медичних зображень допомагає виявляти аномалії на ранніх стадіях. Нейронні мережі на базі CNN (Convolutional Neural Networks) можуть ідентифікувати патології та надавати лікарям додаткову інформацію для діагностики, що підвищує точність і швидкість медичних рішень.

### **Мета і завдання дослідження.**

Мета дослідження – розробка системи відеоспостереження, що поєднує алгоритми YOLO для детекції та Deep Tracking для відстеження об'єктів у реальному часі.

Основні завдання:

- Аналіз існуючих алгоритмів детекції та трекінгу об'єктів.
- Оцінка продуктивності алгоритмів YOLOv8 і Deep Tracking.
- Реалізація системи, яка здатна працювати в реальному часі.
- Тестування системи у реальних умовах для оцінки її ефективності.

**Об'єктом дослідження** є сучасні методи автоматичної детекції та класифікації об'єктів, що використовуються в задачах комп'ютерного зору.

**Предметом дослідження** є алгоритми глибокого навчання, зокрема згорткові нейронні мережі (CNN), що лежать в основі методів детекції та класифікації об'єктів, таких як YOLO, SSD та Faster R-CNN.

### **Методи дослідження.**

Для проведення дослідження використовуються методи глибокого навчання, зокрема архітектури нейронних мереж, що спеціалізуються на застосуванні комп'ютерного зору. Методи глибокого навчання дозволяють досягти високої точності в задачах комп'ютерного зору, особливо в поєднанні з великими наборами даних. В процесі дослідження здійснюється програмна реалізація алгоритмів із застосуванням бібліотек Python (PyTorch, OpenCV), що дозволяє протестувати моделі на реальних наборах даних та провести їхню оцінку.

### **Наукова новизна.**

Новизна дослідження полягає в інтеграції YOLOv8 і Deep Tracking для створення системи відеоспостереження, здатної працювати в реальному часі. Оцінка ефективності цих алгоритмів дозволяє знайти оптимальне співвідношення між точністю та швидкістю в умовах реального світу. Це сприяє розвитку технологій автоматичного відеоспостереження для складних сценаріїв.



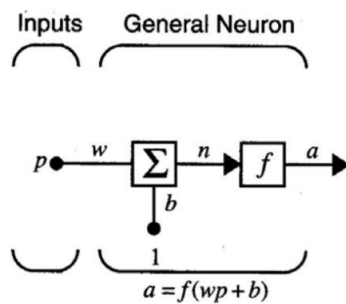
# **РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ ТА ЇХ СУЧАСНІ АРХІТЕКТУРИ.**

## **1.1 Загальна характеристика нейронних мереж та історія розвитку нейронних мереж.**

Нейронні мережі (НМ) є основою сучасного машинного навчання, забезпечуючи ефективне рішення для багатьох складних завдань, таких як класифікація, регресія та сегментація зображень. В цьому розділі розглянуті основні принципи роботи нейронних мереж, зокрема їхню структуру, функції активації, методи навчання, алгоритми оптимізації та питання, пов'язані з перенавчанням.

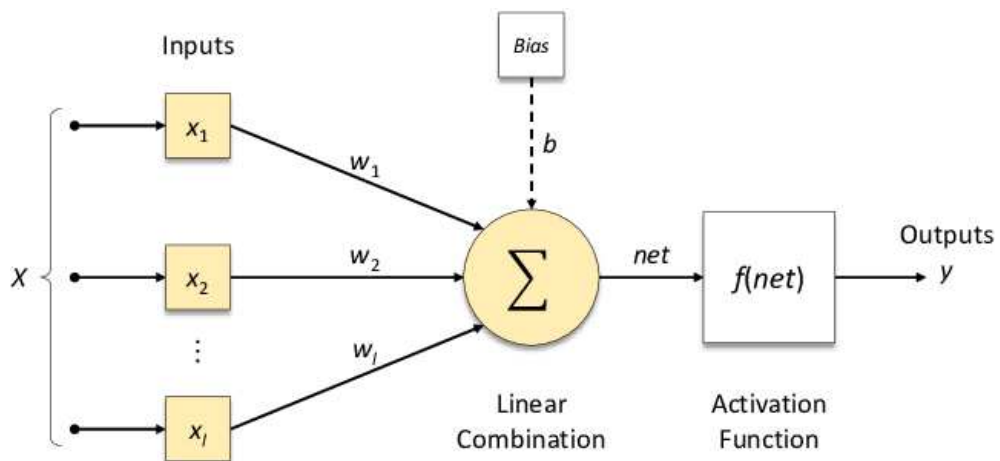
Штучна нейронна мережа – це математична модель, натхненна біологічними нейронними мережами. У певному сенсі вона побудована за образом та подобою мереж нервових клітин живого організму. Штучні нейронні мережі дозволяють моделювати складні залежності у даних, і їх успіх у значній мірі залежить від правильного налаштування їхньої структури та функцій”(LeCun, Y., Bengio, Y., & Hinton, G. E., 2015). Фактично, штучна нейронна мережа є сукупністю штучних нейронів, що взаємодіють між собою. Фактично, штучна нейронна мережа є сукупністю штучних нейронів, що взаємодіють між собою. Основні принципи роботи нейронних мереж були описані ще в 1943 Уорреном Мак-Каллоком і Уолтером Піттсом. В 1957 нейрофізіолог Френк Розенблатт розробив першу нейронну мережу, а в 2010 великі обсяги даних для навчання відкрили можливість використовувати нейронні мережі для машинного навчання. На даний момент нейронні мережі використовуються у численних галузях машинного навчання та вирішують проблеми різної складності.

На представленому малюнку 1 зображено перцептрон - найпростіша нейронна мережа, запропонована нейрофізіологом Френком Розенблаттом. В її основі лежить математична модель сприйняття інформації мозком, що складається із сенсорів, асоціативних та реагуючих елементів.



Малюнок 1: Схематичне зображення перцептрона.

Ця нейронна мережа складається з одного шару, в якому знаходиться єдиний нейрон. Розглянемо її структуру докладніше на малюнку 2.

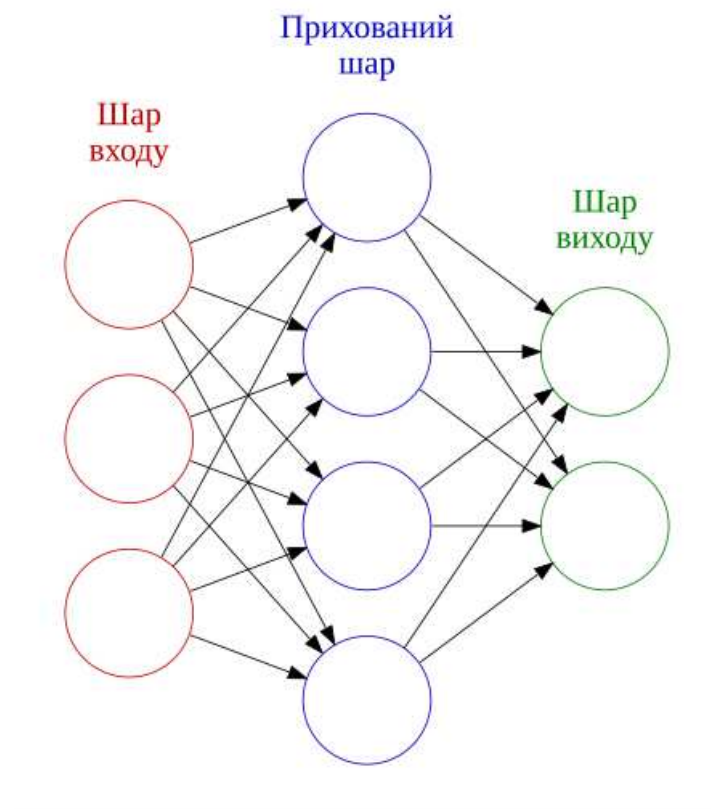


Малюнок 2.

У нас є деякий вектор  $X$ , що складається з  $l$  компонент. Важливість кожної компоненти визначається вагою  $\omega$ . Цей коефіцієнт керує "силою" синаптичного зв'язку. Функція  $f$  називається функцією активації, як аргумент вона приймає лінійну комбінацію компонент вектора  $X$ .

Нейронні мережі складаються з кількох типів шарів, які взаємодіють між собою, щоб обробляти дані. Основними компонентами є:

- Вхідний шар (Input Layer) - це перший шар мережі, який отримує вхідні дані. Кожен нейрон у цьому шарі відповідає за певну характеристику або ознаку даних. Наприклад, у мережі для розпізнавання зображень кожен нейрон може представляти піксель зображення.
- Приховані шари (Hidden Layers). Між вхідним і вихідним шарами можуть бути кілька прихованих шарів, які здійснюють обробку даних. Кількість прихованих шарів і нейронів у кожному з них впливає на потужність і здатність мережі вчитися. Чим більше прихованих шарів, тим складніші шаблони мережа може виявляти.
- Вихідний шар (Output Layer) - останій шар мережі, який видає результати обробки. Кількість нейронів у вихідному шарі визначається завданням: для бінарної класифікації достатньо одного нейрона, тоді як для багатокласової класифікації кількість нейронів відповідає кількості класів.



Малюнок 3: Схематичне зображення нейронної мережі.

## 1.2 Аналіз і порівняння функцій активації в нейронних мережах.

**Принципи, переваги та обмеження.**

Функції активації відіграють ключову роль у нейронних мережах, забезпечуючи їм нелінійність, що дозволяє моделі засвоювати складні залежності у даних. Розглянемо декілька основних функцій активації:

- Східчаста функція:

$$f(x) = \begin{cases} 1, & \text{якщо } x \geq \theta \\ 0, & \text{якщо } x < \theta \end{cases}$$

$x$  — вхід нейрона

$\theta$  — порогове значення

- Бінарна функція, одна із найпростіших функцій активації.

Нейрон активується, якщо значення  $Y$  більше за певне порогове значення, тобто вихід дорівнює одиниці. Якщо  $Y$  менший за це порогове значення, то нейрон не буде активовано і його значення дорівнює нулю. Її часто використовують у задачах бінарної класифікації, де метою є класифікація вхідних даних за однією з двох категорій.

Переваги: ця функція легка для розуміння і реалізації, тому її часто використовують в освітніх цілях для вивчення принципів роботи нейронних мереж.

Недоліки: не підходить для складних моделей, що потребують нелінійних зв'язків.

Якщо треба створити нейромережу, яка визначає приналежність об'єкта до певного типу, але класифікація не обмежується лише двома класами, застосування східчастої функції активації буде проблематичним. Коли більш як один нейрон прийматиме значення 1, нейромережа не зможе визначити, до якого класу належить об'єкт.

- Лінійна функція:

$$f(x) = x$$

або

$$f(x) = w * x + b$$

Також називають функцією без активації, оскільки за використання такої функції вихід завжди пропорційний входу. Таким чином, така функція надає зважену суму вхідних даних і повертає значення, подане в мережу, формуючи прямі зв'язки між входом і виходом. Вихід лінійної функції не обмежений кордонами. Одним із практичних прикладів застосування лінійної функції активації є прогнозування зміни цін за допомогою регресійної моделі на основі попередніх даних.

Переваги: так само, як і східчаста, лінійна функція активації проста для сприйняття і використання у простих моделях або навчальних цілях для демонстрації роботи нейромереж.

Недоліки: оскільки ця функція також не вводить нелінійність в модель, вона не може бути ефективною для складних патернів.

- Сигмоїд:

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

Ця функція перетворює вихід нейрона в діапазон (0, 1), що корисно для бінарної класифікації. Тому цю функцію особливо часто використовують для моделей, де потрібно передбачити ймовірність як результат. Оскільки ймовірність будь-чого існує лише в діапазоні від 0 до 1, sigmoid є правильним вибором. Однак, функція має недолік — при великих або малих значеннях  $x$  вона "насихується", що призводить до повільного навчання.

Переваги: як нелінійна функція активації, сигмоїда може працювати зі складними закономірностями.

Недоліки: діапазон від 0 до 1 не завжди актуальний для багатьох задач, тому використання цієї функції є відносно обмеженим. Також навчання мережі з функцією сигмоїдної активації є складним і нестабільним, бо вихідні дані такої функції не є симетричними навколо нуля, тому всі нейрони мають однаковий знак на виході.

- Тангенс гіперболічний (tanh):

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

Це скоригована версія сигмоїди. Однак, на відміну від sigmoid, вихідний сигнал якого становить від 0 до 1, вихід tanh варіюється від -1 до 1. Більший вхідний сигнал ближче до 1, менший — ближче до -1.

Його зазвичай застосовують у прихованих шарах нейронної мережі, оскільки його значення перебуває в діапазоні від -1 до 1. Тому середнє значення для прихованого шару виходить рівним 0 або дуже близьким до нього. Отже, це допомагає центрувати дані, наближаючи середнє значення до 0.

Градiєнт tanh у чотири рази більший за градієнт сигмоїдної функції. Це означає, що використання функції активації tanh призводить до вищих значень градієнта під час навчання та більших оновлень вагових коефіцієнтів мережі.

Переваги: має нульовий центр, що дозволяє градієнту рухатися в обох напрямках. Може легко відтворити вихідні дані великого позитивного або негативного значення.

Недоліки: ця функція має ту саму проблему градієнта, що зникає, як і сигмоїдна функція, але з градієнтом з більшою амплітудою.

- ReLU (Rectified Linear Unit):

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

Це одна з найпоширеніших функцій активації в сучасних нейронних мережах, особливо якщо йдеться про глибокі нейронні мережі (DNNs), які мають велику кількість шарів між входом та виходом. Функцію ReLU можна легко вважати майже лінійною функцією, і в певному сенсі це кусково-лінійна функція з двома лінійними частинами. Однак, оскільки вона має функцію похідної, її вважають нелінійною функцією активації й вона допускає зворотне розповсюдження помилок. Ця функція замінює всі від'ємні вхідні значення на 0 та не змінює позитивні значення, оскільки вона сприяє швидкому навчальному

процесу. Однак, існує проблема "мертвих" нейронів, коли деякі нейрони ніколи не активуються (подають 0 на виході).

Переваги: ця функція дозволяє мережі бути обчислювально ефективною, утримуючи нейрони деактивованими, коли вхід менший за 0. Завдяки цьому модель також швидше навчається.

Недоліки: проблема полягає в тому, що всі від'ємні значення відразу стають нульовими, що зменшує здатність моделі навчатися на основі даних. Це означає, що будь-який негативний вхід, наданий функції активації ReLU, одразу створює нульовий градієнт, що впливає на кінцевий графік, не відтворюючи від'ємні значення належним чином.

- Leaky ReLU:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Повертає ненульовий результат, невід'ємні значення, що перетворюються на нуль (як у випадку з ReLU). Це допомагає збільшити діапазон функції ReLU. Зазвичай значення  $a$  становить 0,01 або близько того. Таким чином, замість попередньої горизонтальної лінії існує негоризонтальна похила лінія для негативної частини. Такий підхід розповсюджує зворотне поширення помилок для негативних вхідних значень, оминаючи проблему «мертвого нейрона».

- Softmax:

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

Це нелінійна функція, яку переважно використовують для класифікаційних задач. Вона перетворює набори чисел (логітів) на ймовірності.

Softmax дає оцінку для вихідних даних для кожного класу між 0 і 1, а потім ділить їх на суму, щоб визначити ймовірність потрапляння вхідного значення до певних категорій. Ця функція найбільш поширена на вихідному рівні, зокрема для нейронних мереж, які потребують класифікації вхідних даних у кількох категоріях. Наприклад, Softmax зручно використовувати для визначення

приналежності контенту до якогось типу (об'єкта на зображенні, жанру музики, тексту або відео, виявлення теми тощо).

Переваги: імовірності легші для розуміння та інтерпретації в порівнянні з вихідними значеннями. Softmax забезпечує чітко визначений розподіл імовірностей для кожного класу, що дає змогу оцінити роботу нейромережі та її впевненість у прогнозах. Функція softmax також відносно чисельно стабільна, що робить її ефективною для навчання нейронних мереж.

Недоліки: підходить тільки для класифікації об'єктів, які належать тільки до одного з наявних класів, що обмежує застосування функції, якщо об'єкт представляє декілька категорій одночасно. Також модель може бути неточною під час роботи з великою кількістю класів, оскільки в такому разі всі значення будуть дуже малими.

Вибір функції активації залежить від типу задачі, яку потрібно розв'язати:

1. Східчасту функцію рідко використовують у сучасних нейронних мережах. Можна застосувати на вихідному рівні для бінарної класифікації.
2. Лінійна функція - для задач регресії, де результат може приймати будь-яке значення.
3. ReLU підходить для прихованих шарів.
4. Leaky ReLu - якщо є проблеми із вмиранням нейронів у ReLU.
5. Сигмоїда - на вихідному шарі для бінарної класифікації та прихованих шарах, коли потрібне перетворення даних на ймовірності.
6. Tanh - замість сигмоїди, коли треба функція активації з нульовим центром і вхідні дані зосереджені навколо нуля.
7. Softmax підходить для класифікаційних задач.

### **1.3. Етапи навчання нейронних мереж: методи та алгоритми.**

Нейронні мережі навчаються за допомогою алгоритмів, які регулюють ваги нейронів на основі вхідних даних і очікуваних виходів. Цей процес складається з декількох етапів, кожен із яких відіграє критично важливу роль у забезпеченні ефективності та точності моделі.



Основні етапи навчання включають:

- Пряме поширення (Forward Propagation). На цьому етапі дані проходять через нейронну мережу. Кожен нейрон обчислює зважену суму вхідних сигналів, застосовує функцію активації та передає результат наступному шару. У результаті на виході мережі генерується прогноз.

Формула обчислення:

$$y = f(Wx + b)$$

Де  $W$  — ваги,  $x$  — вхідні дані,  $b$  — зсув,  $f$  — функція активації..

- Обчислення втрат (Loss Calculation). Після отримання прогнозу необхідно визначити, наскільки він відрізняється від фактичного значення. Це досягається за допомогою функції втрат (loss function), такої як категорійна крос-ентропія для класифікаційних задач. Функція втрат вимірює відстань між прогнозом мережі та фактичним значенням та є критичним компонентом навчання, оскільки вони визначають, як алгоритм оцінює якість свого прогнозу [2].

Популярні функції втрат:

- MSE (Mean Squared Error): Для задач регресії.
- Крос-ентропія (Categorical Cross-Entropy): Для багатокласових задач класифікації.

Функція втрат визначає, наскільки добре мережа адаптується до вхідних даних, і слугує орієнтиром для коригування ваг.

- Зворотне поширення помилки (Backpropagation) - це алгоритм, який обчислює градієнти функції втрат по відношенню до ваг мережі. Основна ідея полягає в тому, щоб зменшити втрати, регулюючи ваги. Градієнти обчислюються за допомогою правила ланцюга, починаючи з виходу мережі і рухаючись назад до вхідного шару.

- Оновлення ваг. Ваги оновлюються за допомогою алгоритму оптимізації. Змінюючи ваги, мережа намагається зменшити помилки.

## **1.4. Алгоритми оптимізації: Принципи, різновиди та застосування у нейронних мережах.**

Алгоритми оптимізації є важливими для ефективного навчання нейронних мереж, оскільки вони дозволяють налаштовувати параметри (ваги та зсуви) таким чином, щоб зменшити функцію втрат і, як результат, покращити роботу моделі. Алгоритми оптимізації відіграють ключову роль у швидкому та стабільному навчанні моделей глибокого навчання, особливо при роботі з великими обсягами даних [3]. Нижче розглянуті основні алгоритми, що використовуються для оптимізації нейронних мереж.

### **Гرادієнтний спуск (Gradient Descent)**

Градiєнтний спуск є найпоширенішим алгоритмом оптимізації, який базується на принципі пошуку мінімуму функції втрат. Вона полягає в тому, щоб поступово коригувати ваги нейронної мережі у напрямку зменшення функції втрат. Крок коригування визначається на основі обчисленого градієнта (похідної) функції втрат відносно ваг.

### **Основні варіанти градієнтного спуску:**

- **Пакетний градієнтний спуск (Batch Gradient Descent).** Оновлення ваги відбувається після обробки всього навчального набору. Цей метод є дуже стабільним, однак для великих обсягів даних може бути дуже повільним, оскільки кожне оновлення потребує обчислення середнього значення градієнтів для всіх навчальних прикладів.
- **Стадійний градієнтний спуск (Stochastic Gradient Descent, SGD).** Оновлення ваг відбувається після кожного окремого навчального прикладу. Хоча цей підхід дає швидше оновлення, він може бути нестабільним і мати коливання, оскільки градієнти можуть сильно змінюватися між окремими прикладами.
- **Міні-пакетний градієнтний спуск (Mini-batch Gradient Descent).** Компроміс між пакетним та стадійним методами. Оновлення ваг відбувається

після обробки невеликої групи прикладів (міні-пакету). Цей метод зазвичай забезпечує найкраще поєднання стабільності та швидкості.

Адаптивні алгоритми оптимізації.

Адаптивні алгоритми оптимізації автоматично регулюють швидкість навчання для кожного параметра, що дозволяє підвищити ефективність навчання та зменшити час на налаштування гіперпараметрів.

- **Momentum (Моментум).** Цей метод використовує ідею з фізики, додаючи “момент” до зміни ваги на основі попередніх оновлень. Це дозволяє зменшити коливання при оновленні ваг і прискорити процес сходження. Формула оновлення виглядає наступним чином:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla L(w_t)$$

де  $v_t$  — швидкість,  $\beta$  — коефіцієнт моментуму,  $\nabla L(w_t)$  — градієнт функції втрат.

- **Adagrad (Adaptive Gradient Algorithm).** Цей алгоритм змінює швидкість навчання для кожного параметра залежно від того, наскільки часто той чи інший параметр змінюється. Параметри, які часто змінюються, отримують менші швидкості навчання, щоб уникнути перенавчання, в той час як параметри, що змінюються рідше, отримують більші швидкості навчання.

- **RMSprop (Root Mean Square Propagation).** Цей алгоритм є модифікацією Adagrad і дозволяє краще працювати з даними, які мають великі коливання. Він використовує середнє квадратів попередніх градієнтів, щоб адаптувати швидкість навчання для кожного параметра. В результаті зменшується занадто велике зменшення швидкості навчання, що спостерігається в Adagrad.

- **Adam (Adaptive Moment Estimation).** Adam є одним з найпопулярніших алгоритмів оптимізації. Він комбінує ідеї як з градієнтним спуском, так і з моментумом. Алгоритм обчислює середнє значення перших (градієнт) та других (квадрати градієнтів) моментів і використовує їх для

адаптації швидкості навчання. Він дає хороші результати і є менш чутливим до вибору початкових значень параметрів.

Формула для Adam виглядає наступним чином:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(w_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L(w_t))^2$$

$$m_t = \frac{m_t}{1 - \beta_1^t}, v_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \alpha \frac{m_t}{\sqrt{v_t + \epsilon}}$$

де  $\alpha$  — швидкість навчання,  $\beta_1$  та  $\beta_2$  — коефіцієнти моментуму для першого та другого моменту, а  $\epsilon$  — маленька константа для запобігання ділення на нуль.

Вибір алгоритму оптимізації для навчання нейронної мережі залежить від кількох факторів, таких як тип задачі, архітектура мережі, характер даних, а також бажаний компроміс між швидкістю навчання та точністю.

#### 1. Тип задачі (регресія, класифікація, тощо).

- Для задач класифікації та регресії: Якщо задача включає багатокласову класифікацію чи регресію, зазвичай добре працюють такі адаптивні алгоритми як Adam, RMSprop, або Adagrad, оскільки вони можуть швидко адаптуватися до різних характеристик даних і функцій втрат.

- Для складних задач, таких як генерація зображень (наприклад, GAN): Можна використовувати Adam, оскільки він добре працює з задачами, де потрібно забезпечити швидку та стабільну оптимізацію.

#### 2. Архітектура нейронної мережі.

- Глибокі мережі (Deep Networks). Глибокі нейронні мережі, особливо ті, що мають багато шарів, потребують стабільної та ефективної оптимізації. У таких випадках зазвичай застосовують Adam або RMSprop, оскільки ці алгоритми допомагають підтримувати стабільність і швидкість навчання навіть при великій кількості параметрів.

- Рекурентні нейронні мережі (RNN). Для задач з послідовними даними, наприклад, при обробці тексту чи часового ряду, добре підходять алгоритми, що адаптують швидкість навчання, такі як Adam або RMSprop, оскільки вони здатні компенсувати коливання у навчанні, які можуть виникати через залежність між етапами послідовності.

### 3. Розмір та характер даних.

- Малі та середні набори даних. Якщо набір даних невеликий або середній, можна використовувати Batch Gradient Descent для стабільності, оскільки він дає точніші оцінки градієнтів, але може бути повільним на великих наборах даних.

- Великі набори даних. Для великих наборів даних або коли час навчання є критичним, краще використовувати Mini-batch Gradient Descent. Оскільки він комбінує переваги пакетного та стохастичного спуску, дозволяючи швидше оновлювати ваги при великій кількості прикладів.

### 4. Стабільність та швидкість навчання.

- Для швидкої збіжності та стабільного навчання. Якщо потрібно, щоб модель швидко збігалася до мінімуму і стабільно працювала, Adam буде хорошим вибором. Цей алгоритм комбінує швидкість стохастичного спуску з перевагами моментуму та адаптивних кроків навчання.

- Для більш стабільного сходження. Якщо справа з даними, що змінюються з часом або мають високий рівень шуму, RMSprop або Adagrad можуть бути кращими, оскільки вони адаптують швидкість навчання для кожного параметра окремо.

### 5. Управління перенавчанням.

Якщо вам потрібно уникнути перенавчання:

- Моментум (Momentum) або Nesterov Accelerated Gradient (NAG) можуть допомогти, оскільки вони знижують коливання градієнта і покращують загальну стабільність навчання.

- Dropout або L2-регуляризація в поєднанні з адаптивними алгоритмами, як Adam, можуть допомогти уникнути перенавчання, особливо на великих наборах даних.

## 6. Налаштування гіперпараметрів.

Якщо є потреба мінімізувати кількість ручних налаштувань:

- Adam є дуже популярним через те, що він має автоматичне налаштування гіперпараметрів (наприклад, швидкості навчання), і зазвичай він дає добрі результати без потреби в складній настройці.
- Якщо потрібно більше контролю над параметрами, SGD з моментумом може бути корисним, але тоді вам доведеться ретельно налаштовувати швидкість навчання.

## 7. Вибір алгоритму в залежності від проблеми.

- Для невеликих нейронних мереж або простих задач SGD може бути достатньо, але він часто вимагає гарної налаштування початкової швидкості навчання.
- Для складних, великих або глибоких мереж використовуються адаптивні методи, такі як Adam або RMSprop, які дозволяють досягти кращих результатів за менший час.

Підсумок:

- Adam — універсальний вибір для більшості задач, особливо якщо у вас є великі або складні набори даних, або якщо ви хочете зменшити потребу в ручній налаштуванні параметрів.
- SGD з моментумом може бути корисним, якщо ви хочете отримати стабільні результати, але готові налаштовувати гіперпараметри.
- RMSprop або Adagrad — хороший вибір для задач, де дані мають великий шум або змінюються в часі.
- Batch Gradient Descent — найкращий для невеликих наборів даних або коли ви хочете отримати точні результати без швидких оновлень.

Алгоритми оптимізації нейронних мереж є критичними для досягнення хороших результатів у навчанні. Застосування адаптивних алгоритмів, таких як Adam, дозволяє значно покращити ефективність навчання та швидкість сходження до оптимальних рішень. Проте, правильний вибір алгоритму та його

налаштування є важливими чинниками для отримання високої продуктивності моделі.

### **1.5 Перенавчання в нейронних мережах: проблеми, причини та методи подолання.**

Перенавчання — це одна з основних проблем при навчанні нейронних мереж, яка зменшує їх здатність до узагальнення та ефективності на нових, невідомих даних. Це відбувається, коли модель “запам’ятовує” тренувальні дані замість того, щоб виявляти загальні закономірності. Як наслідок, модель показує хороші результати на тренувальному наборі, але значно гірші на валідаційному чи тестовому наборі. Перенавчання виникає через надмірну складність моделі відносно обсягу доступних даних, що обмежує її здатність узагальнювати нову інформацію [2].

Опис основних методів боротьби з перенавчанням:

#### **1. Регуляризація.**

Регуляризація допомагає зменшити складність моделі, що зменшує її схильність до перенавчання.

Два основних методи регуляризації:

- L1 регуляризація (Lasso) додає штрафний термін, пропорційний абсолютному значенню коефіцієнтів ознак. Він заохочує розрідженість, доводячи деякі коефіцієнти до точного нуля, ефективно виконуючи вибір функцій, усуваючи менш важливі функції. Ця можливість вибору функцій робить регуляризацію рівня L1 особливо корисною при роботі з наборами даних із великою кількістю функцій, оскільки вона допомагає спростити моделі, зосереджуючись на найбільш відповідних функціях. Отримане спрощення моделі зменшує переобладнання.

- L2 регуляризація (Ridge) додає штрафний термін, пропорційний квадрату коефіцієнтів ознак. Він не змушує коефіцієнти ставати рівними нулю, а натомість зменшує їх до нуля, завдяки чому всі функції певною мірою сприяють моделі. Регуляризація L2 є ефективною для обробки

мультиколінеарності та, як правило, призводить до більш стабільних, але менш розріджених моделей порівняно з регуляризацією L1.

Обидва методи сприяють більшій стабільності моделі, але L2 регуляризація зазвичай більш ефективна для більшості задач. Але на практиці комбінація регуляризації L1 і L2, відома як регуляризація Elastic Net, може бути використана, щоб отримати переваги від обох методів, використовуючи розрідженість L1 і стабільність L2

## 2. Dropout.

Dropout — це метод, при якому під час навчання випадковим чином “вимикаються” деякі нейрони, тобто їх активація встановлюється в нуль. Це змушує модель не покладатися на конкретні нейрони і змушує її навчатися більш загальним, стійким ознакам, а не конкретним патернам у тренувальних даних. Dropout:

- Випадково вимикає певний відсоток нейронів під час кожної ітерації навчання, що забезпечує певну варіативність.
- Допомагає зменшити ризик того, що мережа буде надмірно залежати від конкретних характеристик даних.

Зазвичай Dropout застосовується в глибоких нейронних мережах, особливо в багат шарових мережах для боротьби з перенавчанням.

## 3. Раннє зупинення (Early Stopping).

Раннє зупинення — це техніка, яка полягає в тому, щоб припинити навчання моделі до того, як вона почне перенавчатися. Коли модель тренується, її продуктивність на валідаційних даних зазвичай покращується на початкових етапах навчання, а потім стабілізується або навіть погіршується.

- Раннє зупинення включає моніторинг помилки на валідаційному наборі і зупинку навчання, коли продуктивність перестає покращуватися або починає погіршуватися.
- Зупинка навчання в момент досягнення найкращих результатів на валідаційних даних дозволяє зберегти найкращу модель без перенавчання.



Цей метод особливо ефективний, коли дані мають високий рівень шуму або коли навчання є надто тривалим.

#### 4. Аугментація даних (Data Augmentation).

Аугментація даних полягає в створенні нових варіантів навчальних прикладів шляхом трансформацій оригінальних даних. Це може включати:

- Повороти, зміщення, зміни масштабу зображень (для комп'ютерного зору).
- Використання синонімів, заміна слів (для обробки текстів).
- Зміна темпу, частоти, додавання шуму в аудіо.

Аугментація допомагає збільшити розмір тренувального набору даних, що робить модель менш схильною до перенавчання.

#### 5. Зменшення складності моделі.

Інший метод боротьби з перенавчанням полягає у зменшенні складності моделі, зокрема шляхом:

- Використання меншої кількості параметрів (менше шарів, менше нейронів на шар).
- Простіші моделі, які мають менше можливостей “запам'ятовувати” тренувальні дані.

Цей підхід особливо корисний, коли є занадто малий обсяг даних для складних моделей, таких як глибокі нейронні мережі.

#### 6. Крос-валідація.

Крос-валідація — це метод перевірки моделі на різних підмножинах даних, що дозволяє отримати більш точну оцінку її здатності узагальнювати. Замість того, щоб тренувати модель на всіх даних, їх ділять на кілька частин, і модель тренується на кожній з них, а потім перевіряється на решті даних.

Цей метод допомагає зменшити ризик перенавчання, оскільки модель перевіряється на більшій кількості різних даних.

Для боротьби з перенавчанням важливо застосовувати кілька методів разом. Наприклад, Dropout в поєднанні з регуляризацією L2 і раннім зупиненням зазвичай дає хороші результати. Вибір конкретних методів залежить від специфіки задачі, кількості даних і складності моделі.

## **1.6 Огляд архітектур основних нейронних мереж (CNN, RNN, GNN).**

Нейронні мережі активно використовуються в різних сферах машинного навчання для вирішення складних задач, і існує кілька архітектур, які спеціалізуються на різних типах даних і проблем. Ось огляд трьох основних архітектур нейронних мереж: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) та Graph Neural Networks (GNN)..

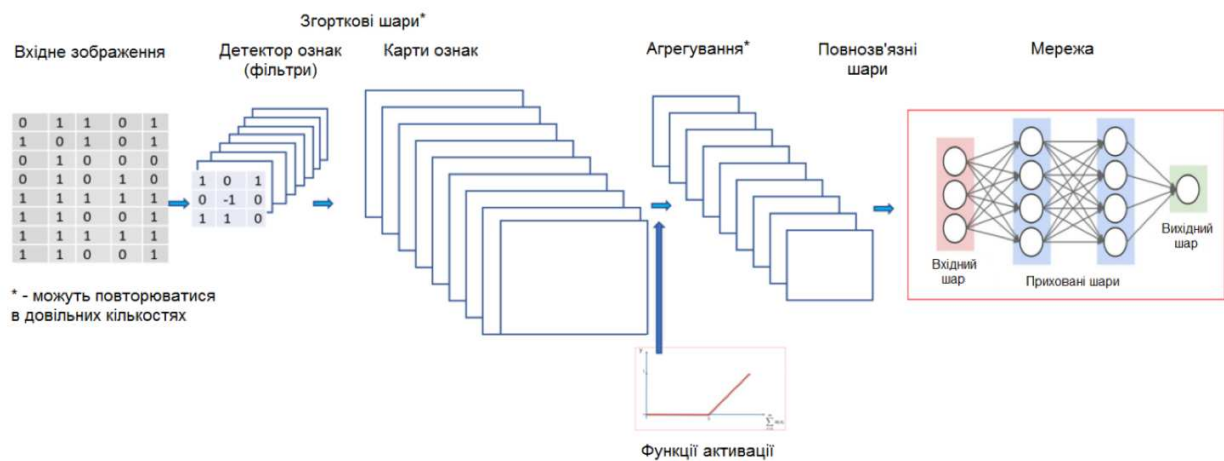
### **1. Згорткові нейронні мережі (CNN)**

#### **Історія та розвиток CNN**

Згорткові нейронні мережі (CNN) стали популярними після успіху моделі LeNet-5, запропонованої Яном ЛеКуном у 1998 році для розпізнавання рукописних цифр. Проте справжній прорив CNN стався у 2012 році, коли команда, очолювана Алексом Кризевським, Геоффом Хінтоном та Іллею Сутскевером, виграла конкурс ImageNet за допомогою архітектури AlexNet, досягнувши значного зменшення помилки розпізнавання зображень.

CNN за своєю структурою розроблені для обробки багатовимірних даних, таких як зображення та відео. Вони здатні вивчати складні ієрархічні ознаки даних на основі багаторівневих згорток, що дає їм перевагу у порівнянні з

традиційними багатoshаровими перцептронами (MLP), які не мають цієї спеціалізації.



Малюнок 4. Загальна структура побудови згорткової нейронної мережі.

#### Основні компоненти CNN.

Згорткові нейронні мережі складаються з декількох основних компонентів, які дозволяють ефективно витягувати ознаки з вхідних даних:

1. Згортковий шар (Convolutional Layer). Основний компонент CNN, який відповідає за згортку вхідного зображення фільтрами (також відомими як ядра або kernel). Кожен фільтр має власні параметри і виявляє певні характеристики на зображенні, такі як краї, текстури або кольори. Результатом згортки є так звані карти ознак (feature maps), які є основою для подальшого аналізу зображення.

2. Шар активації (Activation Layer). Найбільш часто використовується функція активації ReLU (Rectified Linear Unit), яка робить модель більш нелінійною і запобігає насиченню (saturation), як це може траплятися з сигмоїдними функціями активації. ReLU активує лише позитивні значення, що підвищує ефективність роботи нейронної мережі.

3. Шар вибірки або шар зниження розмірності (Pooling Layer). Цей шар використовується для зменшення розміру вхідних даних, знижуючи розмірність карти ознак і зменшуючи обчислювальні ресурси, необхідні для навчання моделі. Найбільш поширеною технікою є max pooling, який вибирає найбільше значення в кожному фрагменті карти ознак.

4. Повнозв'язний шар (Fully Connected Layer). Після кількох згорткових та пулінгових шарів дані подаються на повнозв'язні шари, які поєднують всі

отримані ознаки для прийняття остаточного рішення, наприклад, для класифікації.

Сучасні архітектури CNN.

CNN продовжують еволюціонувати, і з кожним роком з'являються нові архітектури, які досягають кращих результатів у задачах комп'ютерного зору.

Ось кілька найбільш популярних архітектур:

- AlexNet. Ця архітектура мала величезний вплив на розвиток глибокого навчання, зокрема завдяки використанню великих згорткових шарів і методу нормалізації (Local Response Normalization), що допомогло значно покращити точність розпізнавання на конкурсах з великими даними.

- VGGNet. Запропонована дослідниками Оксфордського університету, VGGNet використовує невеликі згорткові ядра (3x3), але збільшує кількість шарів, що дозволяє більш точно виявляти деталі в зображеннях. Це призводить до значного покращення точності, але також підвищує обчислювальну складність моделі.

- ResNet (Residual Networks). Проблема зникання градієнтів стала ключовою при тренуванні глибоких мереж. ResNet вводить концепцію залишкових з'єднань, які дозволяють передавати інформацію через кілька шарів без втрати градієнтів. Це дозволило будувати мережі з глибиною понад 1000 шарів.

- EfficientNet - це одна з найсучасніших архітектур CNN, яка використовує збалансовану стратегію масштабування розміру моделі залежно від кількості даних, що робить її ефективною як з точки зору ресурсів, так і точності.

Застосування CNN.

CNN широко використовуються у задачах комп'ютерного зору, таких як:

- Класифікація зображень. Проблема ідентифікації об'єктів на зображеннях на основі заздалегідь визначених категорій. Один із прикладів — розпізнавання об'єктів у базі даних ImageNet.

- Сегментація зображень. Поділ зображення на кілька частин або об'єктів для кращого аналізу, наприклад, для медичних зображень.
- Відеоаналіз. CNN також використовуються для обробки відео, що дозволяє автоматично аналізувати контент відеозаписів для виявлення певних об'єктів або подій.

## 2. Рекурентних нейронних мереж (RNN).

### Історія та розвиток RNN.

Рекурентні нейронні мережі (RNN) були запропоновані для роботи з послідовними даними, такими як текст, мовлення або часова послідовність. Традиційні нейронні мережі не підходять для таких задач, оскільки не мають механізму збереження контексту. RNN вирішують цю проблему шляхом збереження інформації про попередні стани у спеціальних внутрішніх змінних, що дозволяє враховувати послідовність вхідних даних.

### 1. Ранні етапи (1980-ті роки).

Перші дослідження в галузі рекурентних нейронних мереж почалися в середині 1980-х років. Ідея створення мереж з пам'яттю надихалася на моделях, що спробували імітувати людське сприйняття, де кожне нове сприйняття має значення для контексту попереднього досвіду.

- 1986 рік. Одним із основних етапів у розвитку RNN стало відкриття алгоритму “Backpropagation Through Time” (BPTT). Це дозволило навчати нейронні мережі на основі алгоритмів зворотного поширення помилки, але з урахуванням часових залежностей.
- 1987 рік. Вчені Джон Хопфілд і Девід Румельгарт зробили важливий внесок, розробивши теоретичну модель, в якій нейрони можуть мати зв'язки з іншими нейронами в мережі, що дозволяє зберігати попередню інформацію.

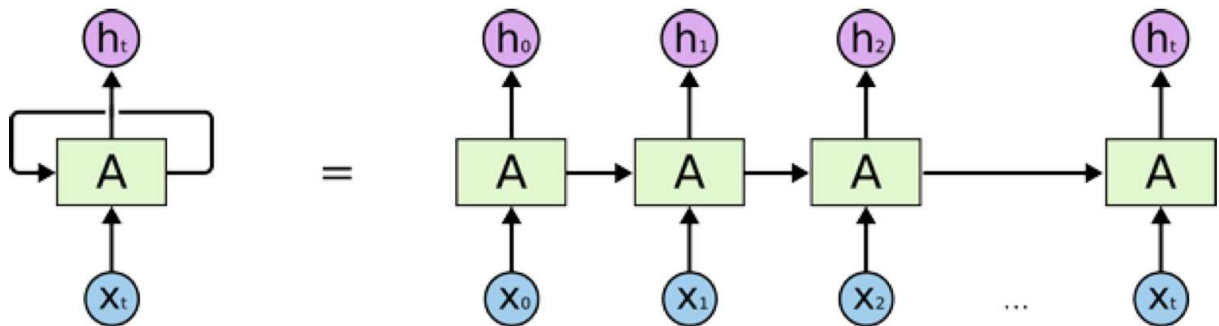
### 2. Проблеми ранніх RNN (1990-ті роки).

- Випадіння градієнтів. Однак, на початкових етапах використання RNN виявилось, що вони мають серйозні проблеми зі зникненням та вибухом градієнтів під час навчання, що значно обмежувало їхню ефективність на довгих послідовностях.
- Ранній етап досліджень також був обмежений в обчислювальних потужностях, і лише прості архітектури нейронних мереж могли бути реалізовані в рамках наявних ресурсів.

### 3. Прогрес в RNN та розширення можливостей (2000-ті роки)

У 2000-х роках почали з'являтися важливі поліпшення в алгоритмах навчання та обчислювальних потужностях, що дозволило RNN стати більш практичними.

Малюнок 5: Загальна структурно-функціональна схема рекурентної нейронної мережі.



#### Основні компоненти RNN.

1. Рекурентний шар (Recurrent Layer). На відміну від CNN, в яких нейрони з'єднані тільки між шарами, в RNN кожен нейрон має зворотні зв'язки. Це дозволяє мережі запам'ятовувати попередні значення і використовувати їх для прогнозування наступних елементів послідовності.

2. Використання внутрішньої пам'яті. Завдяки внутрішній пам'яті RNN можуть моделювати залежності між елементами послідовності, зберігаючи

контекст протягом довгих відрізків часу. Однак, зі збільшенням довжини послідовностей, традиційні RNN мають тенденцію забувати важливу інформацію, що стало відомо як проблема зникання градієнтів.

### LSTM і GRU.

Для вирішення проблеми зникання градієнтів були розроблені архітектури LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які використовують спеціальні «вентилі» (gates) для керування потоком інформації через мережу.

1. LSTM. Ця архітектура вводить три основні елементи: вхідний вентиль, забування та вихідний вентиль. Це дозволяє LSTM зберігати інформацію про важливі етапи послідовності, а також контролювати, яку інформацію потрібно зберегти, а яку — забути.

2. GRU. GRU є спрощеною версією LSTM, що зменшує обчислювальні витрати, використовуючи два вентилі замість трьох. Це робить GRU менш чутливими до налаштувань, але все ще ефективними в багатьох застосуваннях.

### Застосування RNN.

RNN, а також їх модифікації (LSTM, GRU) широко використовуються в задачах обробки природної мови (Natural language processing), таких як:

- Генерація тексту. Використовуючи RNN, можна створювати автоматизовані системи, які генерують текст на основі заданого контексту або теми.
- Переклад мов. RNN також активно використовуються для машинного перекладу, де мережі навчаються переводити речення з однієї мови на іншу.
- Аналіз настроїв. Системи, що використовують RNN, можуть аналізувати текст для визначення емоційного забарвлення або ставлення автора до теми.

### 3. Довільні мережі (Graph Neural Networks)

Історія та розвиток Graph Neural Networks (GNN)

Graph Neural Networks (GNN) — це клас нейронних мереж, призначених для роботи з графовими даними, де важливими є як окремі елементи (ноди), так і їхні взаємозв'язки (ребра). Оскільки графи є загальним способом представлення структурованих даних у багатьох галузях, таких як соціальні мережі, молекулярна хімія, логістика та навіть штучний інтелект, GNN стали важливою частиною сучасного машинного навчання.

Етапи розвитку:

1. До 2000-х років. Класичні методи аналізу графів.
  - До появи GNN основні підходи до аналізу графів ґрунтувалися на класичних методах, таких як пошук у глибину чи в ширину, алгоритми кластеризації (наприклад, методи на основі модульності, такі як алгоритм Лувен), та інші техніки, що не використовували нейронні мережі.
  - В середині 1990-х років з'явилися перші роботи з використанням нейронних мереж для аналізу графів, але ці підходи мали обмеження у здатності працювати з графами різної структури або з великою кількістю даних.
2. 2000-2010 рр. Початок використання нейронних мереж для графів.
  - У цей період з'являються перші спроби адаптувати класичні нейронні мережі до графових даних. В основному йшлося про пошук способів моделювати взаємозв'язки між елементами графа.
  - Однією з перших спроб був Graph Neural Networks (GNN), запропоновані Gori et al. в 2005 році. Вони описали ідею обробки графів за допомогою нейронних мереж, де вузли в графі могли взаємодіяти з їхніми сусідами, щоб отримати більш складні представлення.
3. 2010-2014 рр. Розвиток базових ідей
  - З 2010 року з'являються нові моделі, які вдосконалюють GNN, вводячи ідею рекурентного поширення інформації між вузлами графа. Наприклад, в роботах Scarselli et al. (2009) було запропоновано модель, яка використовує рекурентні нейронні мережі для побудови представлення графа, де кожен вузол отримує інформацію не тільки від своїх сусідів, але й від їхніх сусідів, що дозволяє більш глибоко розуміти структуру графа.



#### 4. 2015-2020: Переломний момент та основні досягнення.

##### 1. Графові згорткові мережі (GCN)

- Важливим кроком став розвиток Graph Convolutional Networks (GCN), запропонованих у 2016 році Thomas Kipf та Max Welling. GCN стали одним з найбільш значних досягнень у цій області, оскільки вони запровадили поняття згортки в графах, що дозволяє ефективно проводити обробку графів з використанням локальних патернів.

- В основі GCN лежить ідея обробки сусідніх вузлів з використанням згортки, схожих на ті, що застосовуються в Convolutional Neural Networks (CNN) для зображень, але адаптованих для графів.

##### 2. Розвиток і вдосконалення.

- У 2017 році був представлений Graph Attention Network (GAT), який використовує механізм уваги для зважування внеску сусідів у процес обробки графа. Це дозволяє мережам адаптувати важливість різних сусідів в залежності від їхнього контексту.

- У той же час були розроблені й інші модифікації GNN, такі як GraphSAGE (Graph Sample and Aggregation), які вводять нові способи агрегування сусідніх вузлів для побудови більш ефективних моделей.

##### 3. Поширення застосувань.

- До 2020 року GNN почали активно застосовувати в різних галузях, таких як:

- Соціальні мережі: Моделювання відносин між користувачами, прогнозування зв'язків або рекомендації.

- Молекулярна хімія: Створення моделей для прогнозування властивостей молекул або біологічних молекул.

- Мережі зв'язків: Аналіз транспортних і логістичних мереж.

- Технології блокчейнів: Використання графів для аналізу та оптимізації транзакцій.

#### 4. 2020-2024: Сучасний стан та нові горизонти.

### 1. Поглиблені моделі GNN.

- В останні роки дослідження в галузі GNN зосереджуються на створенні більш глибоких і потужних архітектур, які можуть обробляти графи з мільйонами вузлів і ребер. Наприклад, Graph Neural Networks with Attention Mechanisms стали дуже популярними через їх здатність адаптувати увагу до важливих частин графа.

- Однією з важливих розробок стали багатошарові GNN, що дозволяють обробляти графи з різними рівнями глибин.

### 2. Техніки для великих графів.

- Оскільки GNN зазвичай мають високі обчислювальні витрати для великих графів, розроблено кілька технік для покращення ефективності, таких як міні-пакетне навчання (mini-batching), мікроскопічне навчання (graph sampling) та інші стратегії, які дозволяють зменшити витрати на обчислення.

### 3. Широкий спектр застосувань.

- Окрім традиційних сфер застосування, GNN також використовуються для нових задач, таких як обробка текстових даних з графовою структурою, застосування в області штучного інтелекту для обробки складних даних, таких як 3D моделі.

Основна ідея GNN полягає в тому, що вона може ефективно агрегувати та поширювати інформацію на графових структурах, використовуючи локальні зв'язки між вузлами. Кожен вузол графа отримує інформацію з його сусідніх вузлів та використовує цю інформацію для свого власного оновлення. Цей процес повторюється для кожного вузла, що дозволяє моделі аналізувати та розуміти взаємодії в графі.

Основні компоненти GNN включають вузлові представлення (node embeddings), агрегаційні функції (aggregation functions) та оновлювальні функції (update functions). Вузлові представлення визначають векторне подання кожного вузла в графі, яке може залежати від його стану та властивостей. Агрегаційні функції об'єднують інформацію з сусідніх вузлів, щоб отримати загальне

представлення для кожного вузла. Функції оновлення змінюють вузлові представлення, використовуючи інформацію, отриману в результаті агрегації.

GNN можуть бути багатошаровими та містити декілька послідовних шарів для більш складних аналізів графів. Кожен шар GNN виконує агрегацію та оновлення представлення вузлів на графі, допомагаючи моделі збирати інформацію з різних віддалених частин графа.

Застосування GNN включають класифікацію вузлів та графів, прогнозування властивостей вузлів, рекомендаційні системи, виявлення спільнот та багато інших задач, пов'язаних з аналізом графів. Вони є потужним інструментом для роботи з незваженими та структурованими даними, які можуть бути представлені у вигляді графів.

Основні принципи роботи нейронних мереж базуються на їхній архітектурі, функціях активації, алгоритмах навчання, методах оптимізації та стратегіях боротьби з перенавчанням. Розуміння цих принципів є ключовим для ефективної розробки та застосування нейронних мереж у різних сферах, таких як комп'ютерний зір, обробка природної мови та багато інших. Ці знання дозволяють дослідникам та інженерам створювати більш складні та ефективні моделі, здатні вирішувати складні завдання з високою точністю.

Згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та графові нейронні мережі (GNN) є одними з найбільш впливових і широко використовуваних архітектур у галузі машинного навчання та глибокого навчання. Кожна з цих архітектур має свої унікальні можливості: CNN відзначаються ефективністю в обробці просторових даних, RNN — у роботі з послідовними даними, а GNN відкривають нові горизонти для обробки та аналізу даних, представлених у вигляді графів, що є важливими для задач, пов'язаних із взаємозв'язками і структурованими даними.

Разом ці архітектури розширюють можливості для вирішення складних завдань у різних галузях, таких як комп'ютерний зір, обробка природної мови, аналіз соціальних мереж, біоінформатика, а також розв'язання проблем у сфері автономних систем. Розвиток нових технологій, таких як трансформери, і вдосконалення існуючих моделей роблять ці мережі ще потужнішими.

Подальший розвиток алгоритмів навчання та оптимізації, а також інтеграція різних типів нейронних мереж, сприятиме новим досягненням у галузі штучного інтелекту. Це відкриває великі можливості для інновацій в таких сферах, як медицина, робототехніка, автономне керування та аналіз великих даних, що веде до важливих практичних досягнень у багатьох сферах людської діяльності.

## **РОЗДІЛ 2 АЛГОРИТМИ ДЕТЕКЦІЇ ОБ'ЄКТІВ ТА КОМП'ЮТЕРНИЙ ЗІР: СУЧАСНІ ПІДХОДИ ТА ЗАСТОСУВАННЯ**

### **2.1 Комп'ютерний зір: історія, основні компоненти, методології та виклики.**

Комп'ютерний зір (Computer Vision) — це наука і технологія, що займається автоматизацією аналізу візуальної інформації, отриманої з навколишнього середовища, яка об'єднує елементи інформатики, математики, фізики та когнітивних наук. Вона націлена на імітацію людського зору за допомогою алгоритмів і систем, здатних сприймати, аналізувати та розуміти візуальну інформацію у вигляді зображень, відео або навіть тривимірних сцен. Як зазначає Fei-Fei Li, один із провідних дослідників у цій галузі, «Комп'ютерний зір — це наука, що дозволяє комп'ютерам розуміти зображення і відео так само, як це роблять люди» [4]. Розвиток комп'ютерного зору обумовлений прогресом у таких галузях, як машинне навчання, нейронні мережі та обробка зображень. Ця галузь має великий потенціал для застосування у багатьох сферах, включаючи медицину, автомобілебудування, безпеку та розваги.

Історія комп'ютерного зору.

Комп'ютерний зір як наукова дисципліна бере свій початок з 1960-х років. У цей час дослідники почали вивчати, як комп'ютери можуть обробляти зображення. Основні етапи розвитку комп'ютерного зору включають:

- 1960-ті роки. Одними з перших завдань, які були поставлені в цій галузі, були прості задачі з розпізнавання форм і контурів. Наприклад, у 1966 році в рамках проекту "Штучний інтелект" в МІТ було проведено дослідження, в якому комп'ютер намагався розпізнавати об'єкти на зображеннях.

- 1970-ті — 1980-ті роки. В цей час вчені почали розробляти алгоритми для виявлення країв, фільтрації та сегментації зображень. Методи, такі як детектор Канні і фільтри Собеля, стали основою для подальшого аналізу зображень. Згідно з дослідженням, методи виявлення країв та сегментації залишаються важливими етапами у більшості комп'ютерних зорових задач [5].

- 1990-ті роки. У цей період з'явилися перші методи для розпізнавання обличч, а також техніки, засновані на статистичних методах. Створення бази даних з зображеннями обличчя, таких як Yale Face Database, сприяло розвитку алгоритмів розпізнавання.

- 2010-ті роки. Поява глибокого навчання стала революцією в комп'ютерному зорі. Глибокі нейронні мережі, зокрема згорткові нейронні мережі (CNN), продемонстрували вражаючі результати в розпізнаванні зображень на змаганнях, таких як ImageNet [6]. Це призвело до значного зростання інтересу до цієї галузі та її практичних застосувань.

Основні компоненти комп'ютерного зору.

Комп'ютерний зір складається з кількох ключових етапів, які допомагають системі аналізувати візуальну інформацію:

- Збір даних. На першому етапі відбувається захоплення зображень або відео за допомогою камер, сканерів або інших пристроїв. Якість отриманих зображень безпосередньо впливає на результати подальшого аналізу. Сучасні камери з високою роздільною здатністю, а також системи тривимірного сканування забезпечують точне захоплення деталей навколишнього середовища.
- Обробка зображення. Цей етап включає в себе підготовку зображення до аналізу, що може включати зменшення шуму, корекцію яскравості,

контрасту та інших параметрів. Як зазначає Szeliski, "обробка зображень є важливим етапом для поліпшення якості даних перед їх аналізом" [7].

- Сегментація. На цьому етапі зображення розділяється на різні сегменти, що дозволяє виділити об'єкти та регіони, які потребують подальшого аналізу. Це може включати виділення об'єктів, країв або текстур. Популярними алгоритмами сегментації є K-means, Watershed та згорткові нейронні мережі.
- Аналіз та розпізнавання. На цьому етапі зображення аналізується для виявлення та класифікації об'єктів. Використовуються різноманітні алгоритми машинного навчання, нейронні мережі та інші методи для розпізнавання об'єктів. Наприклад, згорткові нейронні мережі (CNN) активно використовуються для розпізнавання об'єктів у зображеннях. Yann LeCun, один із піонерів CNN, зазначає, що згорткові мережі ефективні у виявленні локальних структур у зображеннях [8].
- Інтерпретація результатів. Після аналізу отримані дані інтерпретуються для прийняття рішень. Це може включати в себе візуалізацію даних, видачу результатів у текстовій формі або передачу їх до інших систем для подальшої обробки. Системи можуть приймати рішення на основі отриманих даних, що є важливим аспектом в реальних застосуваннях, таких як автономні транспортні засоби.

Методології комп'ютерного зору.

Комп'ютерний зір використовує різноманітні методології та алгоритми для досягнення своїх цілей:

- Машинне навчання. Використання алгоритмів машинного навчання дозволяє комп'ютерним системам автоматично навчатися на основі даних. Моделі, такі як дерева рішень, SVM (Support Vector Machines), та нейронні мережі, широко використовуються в комп'ютерному зорі. Машинне навчання є потужним інструментом для аналізу даних і виявлення шаблонів [9].

- Глибоке навчання. Згорткові нейронні мережі (CNN) стали популярними завдяки своїй здатності ефективно розпізнавати зображення і класифікувати їх. Глибоке навчання забезпечує нейронні мережі, які можуть автоматично видобувати особливості з необроблених даних [2]. Глибоке навчання дозволяє досягати високої точності у складних завданнях, таких як детекція об'єктів та сегментація.
- Візуальний трекінг. Ця методологія дозволяє системам слідкувати за об'єктами в реальному часі, використовуючи методи, такі як оптичний потік та детекція ключових точок. Трекінг є критично важливим для завдань, де потрібно виявити і відслідковувати рухомі об'єкти [10].

Основні завдання глибоких нейронних мереж для комп'ютерного зору:

- Класифікація зображень.

Завдання класифікації зображень полягає в присвоєнні певної категорії (класу) кожному зображенню. Архітектури, такі як ResNet та EfficientNet, широко використовуються для класифікації зображень. ResNet відомий своєю здатністю працювати з дуже глибокими мережами завдяки використанню залишкових блоків. EfficientNet є більш оптимізованим варіантом, що досягає високої точності при меншій кількості параметрів, що робить її ефективною для застосування в реальних умовах.

- Детекція об'єктів.

Детекція об'єктів полягає у визначенні не лише наявності об'єкта на зображенні, але й його місцезнаходження (координат) на цьому зображенні. Серед популярних архітектур для цього завдання — YOLO (You Only Look Once), SSD (Single Shot Multibox Detector) та Faster R-CNN. YOLO дозволяє здійснювати детекцію об'єктів в реальному часі, SSD працює швидко, навіть на менших пристроях, а Faster R-CNN досягає високої точності на складних зображеннях.

- Генерація зображень.

Генерація зображень передбачає створення нових зображень, які можуть бути схожі на ті, що були використані для навчання. Популярними архітектурами для цієї задачі є Generative Adversarial Networks (GANs) та Diffusion Models. GANs складаються з двох моделей (генератора і дискримінатора), які змагаються між собою, щоб генерувати реалістичні зображення. Diffusion Models працюють за принципом поступового додавання шуму до зображення, а потім відновлення його, що дозволяє генерувати високоякісні зображення.

Виклики комп'ютерного зору.

Незважаючи на досягнення, комп'ютерний зір стикається з численними викликами:

- Непередбачуваність навколишнього середовища. Зовнішні умови, такі як освітлення, погодні умови та наявність шуму, можуть суттєво вплинути на якість зображення та результати аналізу. Як зазначає Geiger, "Умови, в яких системи комп'ютерного зору працюють, можуть бути непередбачуваними" [11].
- Складність обробки даних. Великий обсяг даних, який потрібно обробити, може призвести до високих витрат на обчислювальні ресурси та час обробки. Це вимагає ефективних алгоритмів та оптимізації процесів. Обробка великих даних вимагає нових підходів для зменшення навантаження на системи [12].
- Етика та безпека. Використання технологій комп'ютерного зору в деяких сферах викликає питання щодо етики, конфіденційності та безпеки. Наприклад, системи розпізнавання обличчя можуть порушувати права особистості, якщо їх використовують без належного контролю. Етичні питання щодо використання комп'ютерного зору стають дедалі актуальнішими [13].
- Узагальнення моделей. Одним із викликів у комп'ютерному зорі є забезпечення того, щоб моделі, навчені на одному наборі даних, ефективно функціонували на інших. Це вимагає створення методів, спрямованих на підвищення здатності моделей до узагальнення. "Здатність до



узагальнення моделей є надзвичайно важливою для їх успішного використання” [14].

Роль комп’ютерного зору в сучасних системах.

Комп’ютерний зір є однією з ключових технологій, яка дедалі більше інтегрується у різні галузі завдяки своїй здатності аналізувати та інтерпретувати візуальну інформацію. Її вплив поширюється на численні сфери, де автоматизація, ефективність і точність відіграють вирішальну роль. Комп’ютерний зір є технологією, яка значно підвищує ефективність та автоматизацію систем у багатьох сферах. З подальшим розвитком алгоритмів та апаратних засобів очікується, що його застосування стане ще ширшим, трансформуючи традиційні підходи до виконання багатьох завдань.

## **2.2 Огляд сучасних алгоритмів детекції об’єктів: традиційні алгоритми, глибоке навчання та трансформери.**

Детекція об’єктів — це одна з основних задач комп’ютерного зору, що полягає у визначенні об’єктів на зображеннях або відео та їх локалізації (визначення координат об’єктів). Існує кілька сучасних підходів і алгоритмів для вирішення цієї задачі, які досягли значних успіхів завдяки використанню глибокого навчання.

### **1. Алгоритми на основі традиційних методів комп’ютерного зору.**

Традиційні методи комп’ютерного зору були основою для багатьох систем детекції об’єктів до появи глибоких нейронних мереж. Хоча сьогодні алгоритми на основі глибокого навчання значно переважають за точністю та здатністю працювати в складних умовах, традиційні методи все ще використовуються в специфічних випадках, де ресурси обмежені або коли швидкість обробки є критично важливою.

- Методи на основі градієнтів (HOG, SIFT, SURF).

Методи на основі градієнтів орієнтовані на виявлення важливих ознак в зображеннях, які дозволяють легко відрізнити різні об'єкти. Вони зазвичай фокусуються на характерних змінах інтенсивності пікселів, що можуть бути корисними для розпізнавання форм, контурів та текстур.

- HOG (Histogram of Oriented Gradients).

Принцип роботи: HOG — це метод, що використовує інформацію про орієнтацію та амплітуду градієнтів, тобто, як змінюється яскравість пікселів по всьому зображенню. Система розбиває зображення на малесенькі клітинки та для кожної з них обчислює гістограму напрямків градієнтів, що дає змогу захопити структуру об'єктів. Вона є дуже ефективною для задач детекції людей, особливо для пішоходів у відео [15].

Переваги: Простота та швидкість обчислень, хороша ефективність у детекції людей і інших об'єктів з чіткими контурними ознаками.

Недоліки: Чутливість до змін в освітленні та положенні об'єкта.

- SIFT (Scale-Invariant Feature Transform)

Принцип роботи: SIFT — це метод для виявлення та опису локальних характеристик зображень, таких як точки інтересу, що не залежать від масштабу, обертання та освітлення. Він шукає так звані ключові точки на зображенні (сегменти з найбільшими градієнтами) і створює для кожної з них унікальний опис, заснований на оточуючих пікселях. Цей метод широко використовується для задач, що потребують зіставлення зображень [16].

Переваги: Надійність щодо зміщення, масштабування та обертання, добре працює в умовах поганого освітлення та шуму.

Недоліки: Висока обчислювальна складність і час обробки.

- SURF (Speeded Up Robust Features).

Принцип роботи: SURF — це вдосконалення методу SIFT, яке дозволяє швидше і точніше виявляти ключові точки на зображеннях. Основна різниця полягає в оптимізації обчислень, зокрема використання апроксимації гауссових

фільтрів для прискорення обробки. SURF також робить систему більш стійкою до змін в освітленні та зображеннях з різним фоном [17].

Переваги: Швидше за SIFT, добре працює в реальних умовах, зберігаючи високу точність виявлення об'єктів.

Недоліки: Може бути менш ефективним при серйозних деформаціях об'єкта.

## 2. Алгоритми на основі контурів та текстур (Canny, LBP).

Алгоритми, які базуються на контурних ознаках і текстах, спрямовані на виділення важливих структурних елементів зображення, таких як контури об'єктів або текстурні особливості, які характеризують різні типи матеріалів або структуру поверхонь.

- Canny (Canny Edge Detection).

Принцип роботи: Алгоритм Кенні — це метод виявлення контурів на зображенні, який працює в кілька етапів: первинне фільтрування для зменшення шуму (часто використовується гауссов фільтр), виявлення градієнтів для підкреслення країв, а також подальша процедура подвійного порогу для визначення реальних контурів. Алгоритм Кенні є одним з найпопулярніших для виявлення об'єктів по краях [18].

Переваги: Висока точність виявлення контурів, особливо при чітко визначених межах об'єктів.

Недоліки: Вразливість до шуму і труднощі в умовах складних фонових текстур або відсутності чітких меж.

- LBP (Local Binary Patterns).

Принцип роботи: LBP — це метод аналізу текстур, що зосереджується на відносній інтенсивності пікселів у локальних областях зображення. Метод порівнює значення пікселів із середнім значенням їхніх сусідів, формуючи бінарні шаблони, які використовуються для опису текстурної інформації. Цей підхід є особливо ефективним для задач розпізнавання обличь, де текстурні ознаки відіграють ключову роль [19].

Переваги: Простота, швидкість обчислень, висока ефективність для розпізнавання текстур, особливо в реальному часі.

Недоліки: Відсутність інформації про геометричні структури об'єктів, чутливість до шуму.

Традиційні методи комп'ютерного зору, такі як HOG, SIFT, SURF, Canny, і LBP, мають величезне значення в задачах детекції об'єктів, особливо в умовах, де глибокі нейронні мережі можуть бути непотрібно складними або ресурсоемними. Кожен з цих методів має свої переваги та недоліки, що визначає їх ефективність в залежності від специфіки застосування. У поєднанні з іншими методами або в простіших системах вони можуть забезпечити достатній рівень точності та швидкості для багатьох реальних задач.

### 3. Алгоритми на основі глибокого навчання.

- YOLO (You Only Look Once).

Принцип роботи: YOLO — це одна з найбільш популярних моделей для детекції об'єктів, яка працює за принципом одноразового проходу через нейронну мережу для отримання результатів. Основна концепція алгоритму YOLO полягає в тому, що мережа одночасно прогнозує обмежувальні прямокутники (bounding boxes) і ймовірності класу для кожного з цих прямокутників. Весь процес детекції виконується за один прохід через мережу, що забезпечує високу швидкість обробки.

Переваги: Завдяки високій швидкості роботи, YOLO є відмінним вибором для реального часу. Він може працювати з відео і вимагає менших обчислювальних ресурсів порівняно з іншими методами, що робить його ідеальним для мобільних пристроїв та відео.

Недоліки: Втрачає точність при детекції малих об'єктів або в складних сценах, де є багато перекриттів об'єктів. Модель може не добре справлятися з дуже дрібними або складними об'єктами [20].

- SSD (Single Shot MultiBox Detector).

Принцип роботи: SSD — це алгоритм детекції об'єктів, який використовує різні масштаби для виявлення об'єктів різних розмірів. Мережа має кілька рівнів (multi-scale feature maps), на яких генерується набір пропозицій для об'єктів. Алгоритм SSD розбиває зображення на численні області, кожна з яких одночасно проходить через єдину згорткову нейронну мережу. Далі координати та ймовірність класу, розраховані моделлю, порівнюються з рамками за замовчуванням, і на основі цього порівняння створюється остаточна обмежувальна рамка.

SSD використовує різні карти ознак для комбінування передбачень, що дозволяє збільшити кількість виявлених об'єктів кожного класу на зображенні. Завдяки різній роздільній здатності цих карт ознак метод ефективно виявляє об'єкти різних розмірів.

Переваги: SSD працює швидко завдяки своїй архітектурі одноразового проходу через мережу. Це робить його ідеальним для реального часу, особливо для пристроїв з обмеженими ресурсами.

Недоліки: Точність може знижуватися при детекції дрібних об'єктів, оскільки модель має обмежену здатність до точного визначення малих об'єктів в складних сценах (Liu et al., 2016, SSD: Single Shot MultiBox Detector).

- Faster R-CNN (Region-based Convolutional Neural Networks).

Принцип роботи: Faster R-CNN є розширенням класичного R-CNN, що використовує Region Proposal Network (RPN) для генерації пропозицій об'єктів. Зображення подається на вхід згорткової нейронної мережі, яка формує карту ознак. Ця карта ознак потім обробляється через шар Region Proposal Network. У шарі RPN використовується ковзаюче вікно, яке рухається по карті ознак. Центр цього вікна пов'язаний з центром якорів — областей, що мають різні співвідношення сторін і розміри. У моделі використовуються 3 співвідношення сторін і 3 розміри для якорів.

Для ухвалення рішення щодо наявності об'єкта в поточній області застосовується метрика Intersection-over-Foreground (IoF), яка оцінює ступінь перекриття між якорями та істинно розміченими прямокутниками. Якщо

перекриття є достатньо великим, визначається, що в цій області знаходиться об'єкт. Після цього використовується алгоритм Fast R-CNN: карта ознак із знайденими об'єктами передається в шар RoI (Region of Interest), де здійснюється подальша обробка, включаючи повнозв'язні шари для класифікації об'єктів та визначення зміщення для точного визначення меж потенційних об'єктів.. Мережа поєднує дві основні компоненти: RPN для генерації регіонів та CNN для класифікації і локалізації.

Переваги: Це один із найбільш точних алгоритмів для детекції об'єктів, що добре працює навіть в складних сценах, де об'єкти мають різні розміри або перекриваються.

Недоліки: Повільніший, ніж YOLO та SSD, і вимагає великих обчислювальних ресурсів для тренування та тестування [22].

- RetinaNet.

Принцип роботи RetinaNet полягає в поєднанні двох основних компонентів: Feature Pyramid Network (FPN) та ResNet, що разом формують багаторівневу архітектуру для детекції об'єктів різних розмірів. FPN дозволяє мережі ефективно обробляти зображення на різних масштабах, що робить її зручною для виявлення об'єктів як малого, так і великого розміру. ResNet в свою чергу використовує глибокі згорткові шари для створення більш детальних ознак зображення, що покращує точність детекції.

Ключовою особливістю RetinaNet є використання спеціальної функції втрат Focal Loss, яка модифікує традиційну крос-ентропію для підвищення уваги до складних, малочисельних об'єктів. Focal Loss зменшує вагу для легко класифікованих прикладів і збільшує увагу до тих, які складно виявити (наприклад, рідкісні або малочисельні класи), що зменшує дисбаланс класів і підвищує точність детекції, особливо в складних умовах, таких як сцени з великою кількістю об'єктів різних розмірів. Це дає змогу моделі більш ефективно працювати з рідкісними об'єктами та знижує кількість помилок при детекції.

Переваги: Висока точність при наявності великої кількості фонових об'єктів. Focal Loss допомагає вирішити проблему дисбалансу класів (об'єкти фонового зображення часто переважають).

Недоліки: Може бути менш швидким порівняно з YOLO і SSD через більш складні механізми обчислення втрат [23].

- EfficientDet.

Принципи роботи: EfficientDet використовує модифікацію EfficientNet, яка оптимізує глибину, ширину і роздільну здатність нейронної мережі для зниження обчислювальних витрат без втрати точності, BiFPN (Bidirectional Feature Pyramid Network) покращує обробку ознак різних масштабів, застосовуючи двонаправлену фузію для кращого виявлення об'єктів різних розмірів. Стратегія одночасного масштабування глибини, ширини і роздільної здатності мережі, що дозволяє досягти оптимального балансу між точністю та ефективністю. Легкість конструкції забезпечує високу продуктивність при низьких вимогах до обчислювальних ресурсів, що робить модель підходящою для реального часу, навіть на пристроях з обмеженими ресурсами. Цей підхід дозволяє EfficientDet бути високоефективним, швидким і легким, що робить його оптимальним вибором для мобільних і вбудованих систем [24].

Переваги: Висока ефективність за рахунок оптимізації обчислювальних ресурсів. Підходить для пристроїв з обмеженими можливостями, таких як мобільні телефони або інші вбудовані системи.

Недоліки: Може бути менш точним порівняно з складнішими моделями, такими як Faster R-CNN.

- CenterNet.

Принцип роботи: CenterNet — це метод для детекції об'єктів, який фокусується на передбаченні центру об'єкта і його габаритів без необхідності шукати потенційні області (пропозиції регіонів). Це дозволяє значно спростити процес детекції порівняно з традиційними методами, такими як Faster R-CNN або SSD та зменшує складність і обчислювальні витрати, оскільки модель не потребує складного етапу генерації пропозицій. Модель передбачає координати

центру і масштаби об'єктів через згорткові шари, що дозволяє точніше локалізувати об'єкти в порівнянні з іншими методами, які розраховують лише межі.

Переваги: Простота і швидкість - завдяки тому, що пропозиції регіонів більше не генеруються, цей метод забезпечує значну економію ресурсів і прискорює процес виявлення об'єктів та висока точність на складних сценах - CenterNet добре справляється з виявленням об'єктів в умовах перекриття та високої складності сцен.

Недоліки: Потрібні великі дані для тренування для досягнення високої точності в задачах з великими та різноманітними об'єктами та проблеми з малими об'єктами.

4. Сучасні підходи до детекції: Трансформери та їх застосування в задачах детекції (DETR, Vision Transformers).

Останнім часом значну популярність у задачах детекції об'єктів здобули моделі на основі трансформерів, які спочатку були розроблені для обробки природної мови. Завдяки своїй здатності моделювати довгострокові залежності та обробляти глобальну інформацію про зображення, трансформери стали потужним інструментом в області комп'ютерного зору.

Трансформери — це архітектури глибоких нейронних мереж, що спочатку були розроблені для задач обробки природної мови, зокрема для перекладу текстів. Вони застосовують механізм уваги (self-attention), який дозволяє моделі вивчати взаємозв'язки між усіма елементами входу, незалежно від їх позицій. Це дозволяє ефективно працювати з великими даними та вирішувати завдання, що потребують врахування глобальних контекстів, таких як розпізнавання об'єктів на складних зображеннях.

Self-attention дозволяє кожному елементу вхідного вектора взаємодіяти з усіма іншими елементами, присвоюючи ваги залежно від їх важливості для поточного завдання. Це дає змогу ефективно працювати з різними частинами



зображення без потреби використовувати локальні операції, які характерні для традиційних методів, таких як згортки в CNN [25].

- DETR (DEtection TRansformers).

DETR є одним з перших значних кроків в адаптації трансформерів для задач детекції об'єктів. Це модель, яка замінює традиційні етапи, такі як пропозиції регіонів та застосування CNN, одним етапом, де трансформер безпосередньо здійснює детекцію об'єктів на зображенні.

Принцип роботи: DETR використовує стандартну архітектуру трансформера, включаючи механізм самоуваги, для обробки зображення. Зображення спочатку розбивається на піксельні патчі, кожен з яких представлений як вектор, після чого ці вектори обробляються через мережу трансформера. На виході трансформера модель генерує список об'єктів, кожен з яких містить координати обмежувальної рамки, клас об'єкта та ймовірність. Після цього застосовується алгоритм для видалення дублюючих рамок і вибору найбільш релевантних результатів [26].

Переваги: DETR значно спрощує задачу детекції об'єктів, позбавляючись необхідності в таких кроках, як генерація пропозицій для об'єктів (Region Proposal Networks, RPN), які використовуються в традиційних моделях. Завдяки механізму самоуваги трансформер може враховувати відносини між різними частинами зображення на глобальному рівні, що дозволяє моделі краще розпізнавати об'єкти навіть у складних сценах.

Недоліки: DETR потребує значно більше часу для навчання порівняно з традиційними методами, оскільки модель повинна навчитися правильно витягувати глобальну інформацію з зображення. Порівняно з іншими методами, DETR може демонструвати менш точні результати при роботі з малими об'єктами.

- Vision Transformers (ViT).

Vision Transformers (ViT) — це ще одна архітектура, яка застосовує принципи трансформерів до задач комп'ютерного зору. Вона була розроблена

для класифікації зображень, але з часом знайшла застосування й у задачах детекції об'єктів.

Принцип роботи: Зображення розбивається на невеликі патчі (наприклад, 16x16 пікселів), і кожен патч представлений як лінійний вектор, що проходить через трансформер. Кожен патч передає свою інформацію про локальні ознаки в глобальний контекст через механізм самоуваги. Модель обробляє ці патчі одночасно, дозволяючи трансформеру ефективно виявляти складні залежності між різними частинами зображення.

Переваги: Трансформери здатні більш ефективно обробляти довгострокові залежності в зображеннях, що є важливим для розпізнавання складних об'єктів і контекстів. ViT легко масштабується під різні розміри зображень і здатен адаптуватися до різних задач (класифікація, детекція, сегментація).

Недоліки: Vision Transformers часто потребують великих наборів даних для досягнення високої точності, оскільки вони мають велику кількість параметрів. Високі вимоги до обчислювальних ресурсів

### **2.3 Реальні застосування алгоритмів детекції об'єктів.**

Алгоритми детекції об'єктів широко використовуються в різних галузях завдяки їх здатності швидко та точно виявляти об'єкти на зображеннях чи в відео. Ось кілька основних застосувань:

#### **1. Безпека та спостереження.**

- Моніторинг відео. Алгоритми детекції об'єктів використовуються для автоматичного виявлення людей, транспортних засобів або підозрілих дій у відео з камер спостереження. Це допомагає у швидкому реагуванні на події або виявленні аномальних ситуацій.

- Виявлення небезпечних предметів. В аеропортах чи на кордонах алгоритми детекції об'єктів застосовуються для виявлення небезпечних предметів, таких як зброя або вибухові пристрої, у багажі або на підозрілих об'єктах.

## 2. Автономні транспортні засоби.

- Самокеровані автомобілі. Автономні транспортні засоби використовують детекцію об'єктів для виявлення пішоходів, інших автомобілів, дорожніх знаків і перешкод на дорозі. Це дозволяє їм уникати зіткнень і безпечно рухатися.

- Безпілотні літаки (дрони). Дрони з алгоритмами детекції об'єктів використовуються для картографування місцевості, моніторингу стану об'єктів (наприклад, ліній електропередач) або навіть доставки товарів, завдяки здатності точно ідентифікувати об'єкти на землі.

## 3. Медицина.

- Аналіз медичних зображень. Детекція об'єктів широко застосовується в аналізі медичних зображень, таких як рентгенівські знімки, МРТ або КТ. Алгоритми можуть виявляти пухлини, пошкодження або інші аномалії, що допомагає лікарям у діагностиці.

- Автоматичне розпізнавання органів і структур. У хірургії та плануванні операцій, детекція об'єктів використовується для автоматичного визначення органів і анатомічних структур в зображеннях, щоб полегшити планування лікування.

## 4. Промисловість і виробництво.

- Контроль якості на виробничих лініях. Детекція об'єктів дозволяє автоматично перевіряти дефекти в продукції (наприклад, наявність пошкоджень на деталях чи упаковці). Це допомагає підвищити якість продукції та зменшити людський фактор.

- Роботизовані системи. Алгоритми детекції об'єктів використовуються в роботизованих системах для маніпулювання об'єктами, таких як складання, сортування та пакування товарів.

## 5. Торгівля та реклама.

- Аналіз поведінки споживачів. В магазинах і торгових центрах детекція об'єктів використовується для відстеження покупців, аналізу їхнього поведінки (наприклад, скільки часу вони проводять біля конкретних товарів). Це допомагає в поліпшенні викладки товарів та маркетингових кампаній.

- Розпізнавання продуктів. Алгоритми детекції об'єктів можуть використовуватися в автоматичних системах покупок, де вони ідентифікують товари на касах самообслуговування.

## 6. Спортивний аналіз.

- Аналіз відео змагань. У спорті детекція об'єктів використовується для автоматичної обробки відео, що дозволяє точно відслідковувати рухи гравців або м'ячів, аналізувати стратегії, оцінювати техніку виконання та приймати рішення (наприклад, автоматичні повтори в тенісі).

- Підрахунок очок. У деяких видах спорту детекція об'єктів застосовується для підрахунку очок (наприклад, у боксі чи фехтуванні) або фіксації порушень.

## 7. Сільське господарство.

- Моніторинг посівів і врожаю. Датчики з камерою, що використовують алгоритми детекції об'єктів, допомагають фермерам відслідковувати стан сільськогосподарських культур, виявляти захворювання рослин або навіть моніторити урожай у реальному часі.

- Автономні сільськогосподарські машини. Алгоритми використовуються в тракторах і комбайнах для точного виявлення рядків рослин та виконання обробки, наприклад, внесення добрив або обробки від шкідників.

## 8. Роботи в обслуговуванні.

- Роботи-помічники. Алгоритми детекції об'єктів застосовуються для роботи домашніх і промислових роботів, які можуть очищати поверхні, організовувати простір або допомагати в конкретних завданнях (наприклад, перенесення предметів або упорядкування предметів).

- Роботи для обслуговування клієнтів. В готелях або ресторанах роботи можуть використовувати алгоритми для розпізнавання відвідувачів, орієнтування в просторі та виконання завдань, таких як доставка замовлень.

#### 9. Розпізнавання осіб.

- Біометричні системи. Детекція об'єктів і, зокрема, розпізнавання обличчя використовуються для доступу до закритих приміщень, банківських транзакцій або навіть на мобільних пристроях для забезпечення безпеки.
- Контроль доступу. Алгоритми детекції осіб застосовуються для автоматичного розпізнавання та автентифікації користувачів у системах безпеки.

#### 10. Мистецтво та творчість.

- Автоматизоване створення контенту. В деяких творчих сферах, таких як відео і графіка, алгоритми детекції об'єктів використовуються для автоматичного визначення важливих елементів на зображеннях або відео (наприклад, створення відео з автоматичним додаванням спецефектів на основі розпізнаваних об'єктів).

Алгоритми детекції об'єктів мають величезний потенціал для застосування в різних сферах, від безпеки до медицини та сільського господарства. Оскільки технології в цій галузі постійно розвиваються, можна очікувати ще більших досягнень в автоматизації та поліпшенні точності систем.

### **2.4 Тенденції та майбутнє розвитку алгоритмів детекції об'єктів.**

Детекція об'єктів є однією з найважливіших складових сучасного комп'ютерного зору, і її розвиток продовжує рухатися в напрямку інтеграції з іншими технологіями, оптимізації для реального часу та поліпшення адаптивності до різних умов. Зокрема, є кілька основних тенденцій розвитку, які визначатимуть майбутнє алгоритмів детекції об'єктів.

#### 1. Інтеграція детекції об'єктів з іншими технологіями комп'ютерного зору.

Інтеграція алгоритмів детекції об'єктів з іншими аспектами комп'ютерного зору, такими як сегментація та трекінг, дозволяє створити більш ефективні та багатофункціональні системи.

- Сегментація. Комбінація детекції об'єктів і сегментації зображень дозволяє точніше розрізняти та виділяти об'єкти в складних сценах. Сегментація допомагає визначити точні контури об'єктів, що є важливим для таких додатків, як автономне водіння та медичне зображення. Згідно з дослідженням “Panoptic Segmentation” (Kirillov et al., 2019), інтеграція детекції і сегментації дозволяє вирішувати складні завдання, де точність виділення об'єктів важлива для подальшої обробки.

- Трекінг. Трекінг об'єктів у відео дозволяє системам не тільки виявляти об'єкти, але й стежити за їх рухом у реальному часі. Це важливо для багатьох застосувань, від відеоаналітики до робототехніки. Наприклад, алгоритми, які поєднують детекцію і трекінг, як у роботах “DeepSORT” (Wojke et al., 2017) “taTrackR-CNN” (Choi et al., 2018), можуть ефективно відстежувати об'єкти, покращуючи точність і зменшуючи помилки у відео.

2. Розвиток методів детекції в реальному часі для мобільних та вбудованих систем.

Сучасні методи детекції об'єктів стають все більш пристосованими для використання на мобільних та вбудованих пристроях, що вимагає оптимізації як за швидкістю, так і за споживаними ресурсами.

- Оптимізація для мобільних пристроїв. Алгоритми, такі як YOLOv5 і MobileNet, були розроблені для роботи в реальному часі навіть на пристроях з обмеженими ресурсами. Як зазначають в роботі “Efficient Object Detection on Mobile Devices” (Howard et al., 2017), для мобільних пристроїв необхідно розробляти легкі архітектури, які поєднують високу точність з швидкістю обробки і низьким споживанням енергії.

- Вбудовані системи. Вбудовані системи, такі як автомобільні камери для автономних транспортних засобів, вимагають особливої уваги до швидкості і точності. Це включає не тільки алгоритми детекції об'єктів, але й адаптацію до специфічних вимог, як у роботах “Real-Time Object Detection for Autonomous

Vehicles” (Chen et al., 2020), де алгоритми детекції об’єктів інтегруються з іншими сенсорами для забезпечення високої надійності.

### 3. Підвищення адаптивності алгоритмів до різноманітних умов.

Алгоритми детекції об’єктів повинні ставати більш адаптивними до різноманітних умов навколишнього середовища, включаючи динамічні сцени, зміни освітлення та різні типи об’єктів.

- Динамічні сцени. В реальних умовах сцени часто змінюються (наприклад, переміщення об’єктів або зміна кадрів), що вимагає адаптації алгоритмів до таких змін. Методи, засновані на глибокому навчанні, все більше адаптуються до таких ситуацій завдяки використанню трансформерів, як у “DETR” (Carion et al., 2020), де мережа навчається виділяти об’єкти в динамічних сценах.

- Зміни освітлення. Одним із ключових викликів є стабільна робота при змінах освітлення, що може суттєво вплинути на якість детекції. Для покращення адаптації до таких змін використовуються новітні техніки, які враховують фактори освітлення в процесі навчання. У роботі “Lighting Invariant Object Detection” (Yin et al., 2019) представлені методи, що дозволяють зменшити вплив варіацій освітлення на точність детекції.

- Різні типи об’єктів. Детекція різноманітних об’єктів, включаючи маленькі, великі або перекриваються, вимагає постійного розвитку моделей для точнішого розпізнавання. Наприклад, методи, такі як FPN (Feature Pyramid Networks) або вищезгадані трансформери, дозволяють працювати з об’єктами різного розміру та складності.

### 4. Використання мульти-датчикових систем для поліпшення точності та надійності детекції.

Інтеграція даних з різних датчиків дозволяє значно покращити точність та надійність алгоритмів детекції об’єктів, що особливо важливо для складних середовищ, де один тип сенсора може мати обмеження.

- Мульти-датчикові системи. Використання декількох сенсорів (камери, радар, LIDAR) дає можливість покращити детекцію об’єктів, забезпечуючи більш точну інформацію, особливо в умовах поганої видимості

або вночі, поєднання даних з різних датчиків дозволяє значно знизити ймовірність помилок і підвищити надійність систем в складних ситуаціях.

- Надійність і точність. Важливим аспектом є також покращення надійності систем, де одні датчики можуть бути ненадійними в певних умовах. Наприклад, камери можуть мати проблеми при поганому освітленні, а LIDAR може не дати чітких результатів в умовах сильного дощу або туману. У таких випадках мульти-датчикові підходи дозволяють компенсувати недоліки кожного окремого сенсора.

Тенденції розвитку алгоритмів детекції об'єктів зосереджуються на інтеграції з іншими технологіями комп'ютерного зору, оптимізації для мобільних і вбудованих систем, підвищенні адаптивності до різноманітних умов і використанні мульти-датчикових систем для досягнення високої точності та надійності. Ці напрямки сприятимуть розвитку більш ефективних, універсальних і стійких систем детекції об'єктів, що стане основою для нових технологій в галузях автономних транспортних засобів, робототехніки, безпеки та багато інших.

### **РОЗДІЛ 3. ЕВОЛЮЦІЯ, РОЗВИТОК ТА ОСНОВНІ ХАРАКТЕРИСТИКИ YOLO: ВІД YOLOV3 ДО YOLOV11 У ЗАВДАННЯХ КОМП'ЮТЕРНОГО ЗОРУ.**

YOLO (You Only Look Once) — це один з найбільш ефективних алгоритмів для детекції об'єктів, що відрізняється швидкістю та точністю роботи в реальному часі. Його основна ідея полягає в тому, що модель обробляє все зображення за один прохід через нейронну мережу, що значно підвищує швидкість. На відміну від підходів, які використовують регіональні пропозиції, таких як R-CNN, YOLO перетворює задачу детекції в задачу регресії, прогножуючи межі об'єктів і їхні класи одночасно [31]. Саме це дозволяє YOLO



працювати значно швидше за інші методи, такі як Faster R-CNN або SSD, які потребують багатокрокової обробки.

Кожна нова версія YOLO приносить суттєві покращення в продуктивності, швидкості та точності. У цьому розділі ми розглянемо основні зміни в алгоритмах YOLO від версії YOLOv3 до YOLOv8 і їхні ключові відмінності.

### **3.1 YOLOv3 - важливий етап у розвитку YOLO.**

YOLOv3 стала важливим етапом у розвитку YOLO, коли вона була представлена в 2018 році Джозефом Редмоном та його командою. Однією з ключових особливостей YOLOv3 є використання багаторівневої карт ознаки для детекції об'єктів різних розмірів. YOLOv3 використовує архітектуру Darknet-53, що складається з 53-х шарів конволюційної мережі, забезпечуючи обробку на трьох різних масштабах [31]. Це дозволяє знаходити як великі, так і маленькі об'єкти в межах одного зображення, що значно підвищує точність детекції.

Модель працює на основі трьох рівнів карт ознак, що дозволяє виявляти об'єкти на різних масштабах зображення. Це стало можливим завдяки використанню глибокої архітектури та введенню багаторівневої обробки. Використання трьох масштабів для детекції дозволяє YOLOv3 справлятися з об'єктами різного розміру, підвищуючи точність на великих та малих об'єктах [31].

Основні характеристики YOLOv3:

- Три рівні карт ознак для детекції об'єктів різних розмірів.
- Архітектура Darknet-53, що складається з 53-х шарів для глибокого навчання.
- Підтримка багатокласової класифікації, що робить YOLOv3 придатним для складних завдань детекції.

На практиці, YOLOv3 показує чудові результати на багатьох популярних датасетах, таких як COCO, де модель демонструє високу точність і швидкість.

### 3.2 YOLOv4: Оптимізація для продуктивності.

YOLOv4 вийшла у 2020 році і стала ще одним кроком у розвитку архітектури YOLO, зосереджуючись на покращенні продуктивності та точності. Мета цієї версії — запропонувати модель, яка може бути легко розгорнута на стандартних GPU, таких як NVIDIA GTX 1080 Ti, і працювати швидко навіть на менш потужних пристроях.

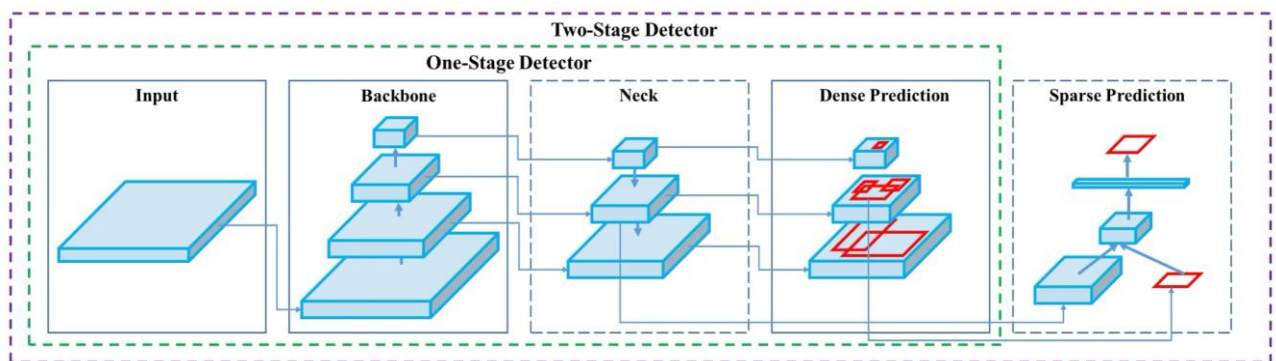
YOLOv4 інтегрує низку функцій, які працюють разом, щоб оптимізувати продуктивність алгоритму. До цих функцій належать:

- Зважені резидуальні з'єднання (Weighted-Residual-Connections, WRC) підвищують ефективність обчислень, забезпечуючи збалансований розподіл інформації.
- Перехресні часткові з'єднання (Cross-Stage-Partial-connections, CSP) зменшують надмірність обчислень і покращують узагальнення моделі.
- Перехресна міні-пакетна нормалізація (CmBN) знижує вплив варіацій у міні-пакетах, сприяючи більш стабільному навчанню.
- Самоадверсаріальне навчання (SAT) покращує стійкість моделі до атак і змін у даних.
- Mish-активація забезпечує гладкіші градієнти, що сприяє кращому навчанню нейронних мереж.
- Аугментація даних Mosaic дозволяє комбінувати кілька зображень в одне, підвищуючи різноманітність навчального набору.
- Регуляризація DropBlock запобігає перенавчанню, видаляючи випадкові блоки ознак.
- Втрати CIoU (Complete Intersection over Union) забезпечують точніше вирівнювання передбачуваних рамок із фактичними.

Архітектура YOLOv4.

Типовий детектор об'єктів складається з кількох компонентів: вхід, магістраль (backbone), шия (neck) та голова (head). YOLOv4 використовує цю структуру для покращення детекції.

1. Магістраль (Backbone):
  - Попередньо навчена на ImageNet для прогнозування класів і рамок об'єктів.
  - У якості основи можуть використовуватись різні моделі: VGG, ResNet, ResNeXt або DenseNet.
2. Шия (Neck):
  - Відповідає за збирання карт ознак із різних етапів мережі.
  - Містить кілька висхідних та низхідних шляхів для агрегації багатомасштабних ознак.
3. Голова (Head):
  - Використовується для остаточного виявлення та класифікації об'єктів.



Малюнок 6: Архітектура YOLOv4

Переваги YOLOv4:

- Інноваційні функції. Використання таких методів, як Mosaic і SAT, забезпечує високу стійкість до шумів і змін даних.

- Універсальність. Підтримка різних моделей магістралі дозволяє адаптувати YOLOv4 до різних потреб.
- Продуктивність. Поєднання передових технологій, таких як CIOU та DropBlock, дозволяє досягати сучасних результатів із меншими ресурсами. YOLOv4 залишається одним із найефективніших алгоритмів детекції об'єктів, завдяки своєму балансу між точністю, швидкістю та універсальністю.

YOLOv4 також продовжує використовувати методи, які дозволяють збалансувати точність і швидкість на стандартних графічних процесорах, роблячи цю модель популярним вибором для завдань реального часу, таких як системи відеоспостереження та детекція на транспортних засобах.

### **3.3 YOLOv5: Простота використання і інтеграції.**

YOLOv5 вийшла у 2020 році і здобула велику популярність завдяки простоті впровадження та інтеграції в різні проекти. YOLOv5 було написано на основі PyTorch, що значно полегшило його інтеграцію з Python-проектами. Ця версія не пов'язана з оригінальними розробниками YOLO, але швидко стала улюбленою завдяки простому інтерфейсу, широкій підтримці та можливості налаштування. YOLOv5 представляє собою сучасний підхід до детекції об'єктів, який поєднує інновації та продуктивність, роблячи його придатним для широкого спектра задач комп'ютерного зору.

Основні характеристики YOLOv5:

#### **1. Без'якорна спліт-голова Ultralytics (Anchor-Free Split Head).**

Традиційні моделі детекції об'єктів використовують наперед визначені якорі для прогнозування місцезнаходження об'єктів. YOLOv5u модернізує цей підхід, впроваджуючи без'якорну спліт-голову Ultralytics, яка забезпечує більш гнучкий та адаптивний механізм детекції. Це дозволяє досягти вищої продуктивності у різних сценаріях, де традиційні методи можуть бути менш ефективними.

#### **2. Оптимізований баланс між точністю та швидкістю.**

Швидкість і точність зазвичай знаходяться у протиріччі. Проте YOLOv5u вирішує цю проблему, пропонуючи виважений компроміс, що забезпечує

детекцію об'єктів у реальному часі без втрати точності. Це особливо важливо для застосувань, які потребують швидкого реагування, таких як:

- Автономні транспортні засоби.
- Робототехніка.
- Відеоаналітика у реальному часі.

### 3. Різноманіття попередньо навчених моделей:

Розуміючи, що різні задачі потребують різних інструментів, YOLOv5u пропонує широкий вибір попередньо навчених моделей. Незалежно від того, чи це інференс, перевірка або навчання, YOLOv5u надає модель, яка оптимально підходить для конкретного завдання. Це гарантує, що замість універсального підходу ви отримаєте модель, налаштовану для вирішення вашої унікальної задачі.

#### Переваги YOLOv5:

- Гнучкість. Впровадження без'якорної архітектури покращує здатність моделі адаптуватися до різноманітних сценаріїв.
- Ефективність. Забезпечує швидке та точне виявлення об'єктів у реальному часі, знижуючи обчислювальні витрати.
- Універсальність. Надає моделі, які можуть бути легко інтегровані у широкий спектр застосувань, таких як транспорт, робототехніка та системи безпеки.

### **3.4 YOLOv6: Промислові застосування та оптимізація.**

YOLOv6 була створена з урахуванням потреб промислових застосувань, де продуктивність на різних пристроях, включаючи мобільні, має ключове значення. Основні покращення YOLOv6 стосуються продуктивності на CPU і менш потужних GPU. Завдяки використанню RepVGG, що забезпечує більш ефективну обробку даних під час інференсу, YOLOv6 здатна забезпечувати вражаючі результати навіть на менш потужних пристроях. RepVGG дозволяє

YOLOv6 забезпечувати високу швидкість інференсу на CPU без втрати точності, що робить її ідеальною для застосувань з обмеженими ресурсами.

YOLOv6 пропонує інноваційні рішення, які роблять її потужним інструментом для сучасних задач комп'ютерного зору, забезпечуючи баланс між точністю, швидкістю та ефективністю.

#### Основні характеристики YOLOv6:

##### 1. Модуль двонаправленої конкатенації (BiC).

В YOLOv6 впроваджено модуль BiC у “шийці” детектора, який покращує сигнали локалізації та забезпечує приріст продуктивності з мінімальним впливом на швидкість. Цей модуль сприяє точнішому визначенню об'єктів, особливо в складних сценах.

##### 2. Стратегія навчання з використанням якорів (AAT).

YOLOv6 інтегрує стратегію AAT, яка дозволяє одночасно використовувати переваги підходів із якорями та без них. Це забезпечує високу ефективність навчання без шкоди для швидкості інференсу.

##### 3. Покращена конструкція магістралі та шийки.

У YOLOv6 додано ще один етап у “магістраль” і “шийку”, що дозволяє моделі досягати передової продуктивності на датасеті COCO навіть за високої роздільної здатності вхідних даних.

##### 4. Стратегія самодистиляції.

Для підвищення продуктивності компактних моделей YOLOv6 реалізує нову стратегію самодистиляції. Вона посилює допоміжну гілку регресії під час навчання та видаляє її під час інференсу, уникаючи помітного зниження швидкості роботи.

#### Переваги YOLOv6:

- Точність. Модуль BiC і вдосконалена конструкція магістралі підвищують точність детекції.

- Гнучкість. Стратегія ААТ дозволяє моделі адаптуватися до різних сценаріїв навчання та інференсу.
- Продуктивність. Навіть за високої роздільної здатності вхідних даних YOLOv6 зберігає високу швидкість.
- Ефективність. Стратегія самодистиляції оптимізує компактні моделі для реальних умов застосування.

### **3.5 YOLOv7: Продуктивність та оптимізація.**

YOLOv7 представила подальші покращення в оптимізації для досягнення ще кращих результатів на більш складних зображеннях. YOLOv7 використовує нові техніки нормалізації та покращену архітектуру, що дозволяє досягати більшої точності при обробці великих об'єктів [33].

Одним з головних інноваційних моментів стало використання Extended Efficient Layer Aggregation Network (E-ELAN), що дозволяє ефективно використовувати ресурси при одночасному підвищенні точності.

Об'єктна детекція в реальному часі: Особливості YOLOv7

На відміну від традиційних детекторів об'єктів, що фокусуються переважно на оптимізації архітектури, YOLOv7 акцентує увагу на оптимізації процесу навчання. Цей підхід включає впровадження модулів і методів, які підвищують точність детекції без збільшення витрат на інференс — концепція, відома як “навчальний мішок безкоштовних даних” (Trainable Bag of Freebies).

Основні характеристики YOLOv7:

#### **1. Перепараметризація моделі.**

YOLOv7 впроваджує заплановану перепараметризовану модель, що є стратегією для шарів у різних мережах з урахуванням концепції шляху поширення градієнта. Цей підхід сприяє підвищенню ефективності навчання та точності прогнозування.

#### **2. Динамічне призначення міток.**

Під час навчання моделі з кількома вихідними шарами виникає питання: “Як правильно призначити динамічні мітки для виходів різних гілок?”. YOLOv7 вирішує цю проблему завдяки новому методу призначення міток, який називається грубе та точне призначення міток (Coarse and Fine Label Assignment).

### 3. Розширене та складене масштабування.

YOLOv7 пропонує методи розширеного та складеного масштабування, які дозволяють ефективно використовувати параметри та обчислення. Це сприяє оптимальному використанню ресурсів при високій точності.

### 4. Ефективність.

Завдяки інноваційному підходу, YOLOv7 скорочує приблизно 40% параметрів і 50% обчислень порівняно з сучасними детекторами об'єктів у реальному часі. Це забезпечує:

- Більшу швидкість інференсу.
- Вищу точність детекції.

На даний момент YOLOv7 більше не підтримується Ultralytics.

## **3.6 YOLOv8: покращенні роботи на пристроях з обмеженими ресурсами.**

YOLOv8 став одним із етапів у розвитку YOLO, зосередившись на покращенні роботи на пристроях з обмеженими ресурсами та підтримці нових стандартів обчислювальних платформ. YOLOv8 використовує нові шари для оптимізації роботи на мобільних пристроях та підтримує останні технології, такі як TensorRT та ONNX .

### Основні характеристики YOLOv8:

#### 1. Передові архітектури магістралі та шийки.

YOLOv8 використовує сучасні архітектури магістралі (backbone) та шийки (neck), які підвищують ефективність витягання ознак і покращують точність



детекції об'єктів. Це дозволяє моделі справлятися навіть зі складними задачами комп'ютерного зору.

## 2. Без'якорна головка Ultralytics.

YOLOv8 впроваджує без'якорну головку (anchor-free head) від Ultralytics, яка покращує точність та ефективність детекції порівняно з традиційними підходами, що базуються на якорях.

## 3. Оптимізація співвідношення точності та швидкості.

Завдяки оптимальному балансу між точністю та швидкістю, YOLOv8 є ідеальним вибором для задач детекції об'єктів у реальному часі. Це робить її придатною для застосувань у багатьох галузях, таких як автономні транспортні системи, робототехніка та відеоаналітика.

## 4. Різноманіття попередньо навчених моделей.

YOLOv8 пропонує широкий вибір попередньо навчених моделей, які можна адаптувати для вирішення конкретних задач. Це забезпечує гнучкість і полегшує вибір оптимальної моделі залежно від вимог до продуктивності та точності.

Основні покращення YOLOv8:

- Оптимізовані шари для роботи на мобільних пристроях.
- Підтримка останніх стандартів обчислювальних платформ.
- Поліпшена робота з малою кількістю навчальних даних.

Завдяки цим покращенням YOLOv8 стала найкращою версією для використання в умовах обмежених ресурсів і для вирішення складних завдань детекції об'єктів.

## **3.7 YOLOv9: у пошуку ідеального алгоритму.**

Від попередніх версій YOLOv9 виділяється своїм інноваційним підходом до вирішення проблем втрати інформації, характерної для глибоких нейронних мереж. Завдяки інтеграції PGI (Програмованої градієнтної інформації) та універсальної архітектури GELAN (Узагальненої ефективної мережі агрегації

рівнів), YOLOv9 не лише покращує здатність моделі до навчання, але й забезпечує збереження важливої інформації протягом усього процесу виявлення, досягаючи виняткової точності та продуктивності. Удосконалення YOLOv9 спрямовані на вирішення проблем втрати інформації в глибоких нейронних мережах. Принцип “інформаційного вузького місця”, що висвітлює фундаментальну проблему глибокого навчання: у процесі проходження даних через послідовні шари мережі ймовірність втрати інформації зростає та інноваційне використання оборотних функцій є центральними в його конструкції, забезпечуючи високу ефективність і точність.

GELAN є стратегічним досягненням в архітектурі YOLOv9, яке дозволяє досягти чудового співвідношення між кількістю параметрів і обчислювальною ефективністю. Її дизайн забезпечує гнучку інтеграцію різних обчислювальних блоків, що робить YOLOv9 адаптивним до широкого спектра застосувань без втрати швидкості та точності.

Ключові інновації YOLOv9:

- Принцип інформаційного вузького місця.
- Програмована градієнтна інформація (PGI).
- Узагальнена ефективна мережа агрегації рівнів (GELAN).

### **3.8 YOLOv10: Інновації в реальному часі для детекції об’єктів.**

YOLOv10, розроблений на основі Ultralytics Python Package дослідниками з Університету Цінхуа, представляє новий підхід до детекції об’єктів у реальному часі. Він усуває недоліки постобробки та архітектури моделі, які були виявлені в попередніх версіях YOLO. Завдяки усуненню методу не-максимального придушення (NMS) та оптимізації компонентів моделі, YOLOv10 досягає передової продуктивності при значному зменшенні обчислювальних витрат. Експерименти продемонстрували чудовий компроміс між точністю та затримкою для моделей різного масштабу.

Детекція об’єктів у реальному часі спрямована на точне прогнозування категорій та положення об’єктів на зображеннях із низькою

затримкою. Серія YOLO залишалася лідером у цій галузі завдяки збалансованому підходу між продуктивністю та ефективністю. Проте залежність від NMS та неефективність архітектури в минулих версіях стримували її оптимальну продуктивність. YOLOv10 вирішує ці проблеми завдяки двом ключовим інноваціям:

1. Послідовне подвійне призначення. Удосконалює навчання без використання NMS.

2. Цілісна стратегія моделювання. Зосереджується на підвищенні точності та ефективності.

Архітектура YOLOv10 розвиває сильні сторони попередніх моделей YOLO, додаючи ключові інновації. Вона включає наступні компоненти:

1. Магістраль (Backbone).

Використовує вдосконалену версію CSPNet (Cross Stage Partial Network) для покращення градієнтного потоку та зменшення обчислювальної надмірності.

2. Шия (Neck).

Поєднує ознаки з різних масштабів і передає їх у голову моделі. Використовує шари PAN (Path Aggregation Network) для ефективної агрегації багатомасштабних ознак.

3. Голова One-to-Many.

Генерує декілька прогнозів для кожного об'єкта під час навчання, забезпечуючи багаті навчальні сигнали та покращуючи точність навчання.

4. Голова One-to-One.

Генерує один найкращий прогноз для кожного об'єкта під час інференсу, усуваючи необхідність у NMS, що знижує затримку та підвищує ефективність.

YOLOv10 є революційним кроком у розвитку алгоритмів детекції об'єктів, забезпечуючи чудовий баланс між точністю, ефективністю та затримкою. Це робить його потужним інструментом для широкого спектра застосувань — від мобільних пристроїв до великих обчислювальних кластерів.

Основні характеристики YOLOv10

- Навчання без NMS:
- Полегшені класифікаційні голови.
- Просторово-канальна декорельована вибірка.
- Ранжування блоків.
- Великі згортки ядра.
- Модулі часткового самонавчання для підвищення продуктивності без значного збільшення обчислювальних витрат.

### **3.9 YOLOv11: Новітня ітерація детекторів об'єктів у реальному часі.**

YOLOv11 є останньою ітерацією в серії Ultralytics YOLO, яка переосмислює можливості комп'ютерного зору завдяки передовій точності, швидкості та ефективності. Спираючись на досягнення попередніх версій YOLO, YOLOv11 приносить суттєві покращення в архітектурі та методах навчання, що робить її універсальним вибором для широкого спектра задач комп'ютерного зору.

#### **Основні характеристики YOLOv11:**

##### **1. Удосконалене витягання ознак.**

Використання покращеної архітектури “магістралі” та “шийки” (backbone і neck), що розширює можливості витягання ознак. Це дозволяє досягти більш точної детекції об'єктів і вирішення складних завдань.

##### **2. Оптимізація для ефективності та швидкості.**

YOLOv11 інтегрує покращені архітектурні рішення та оптимізовані конвеєри навчання, забезпечуючи високу швидкість обробки даних та оптимальний баланс між точністю і продуктивністю.

##### **3. Вища точність при меншій кількості параметрів.**

Удосконалена конструкція моделі дозволяє YOLOv11m досягати більш високої середньої точності (mAP) на датасеті COCO, використовуючи на 22% менше параметрів, ніж YOLOv8m. Це робить модель ефективною з точки зору обчислень без втрати точності.

#### 4. Адаптованість до різних середовищ.

YOLOv11 легко розгортається в різних середовищах, таких як прикордонні пристрої, хмарні платформи та системи з підтримкою графічних процесорів NVIDIA, забезпечуючи максимальну гнучкість.

#### 5. Широкий спектр підтримуваних задач.

YOLOv11 розроблено для розв'язання різноманітних задач комп'ютерного зору, включаючи:

- Детекцію об'єктів.
- Сегментацію екземплярів.
- Класифікацію зображень.
- Оцінку пози.
- Орієнтовану детекцію об'єктів (OBB).

YOLOv11 є потужним інструментом для сучасних задач комп'ютерного зору, що поєднує інноваційні підходи з високою ефективністю, роблячи її ідеальним вибором для дослідників та інженерів.

#### Переваги YOLOv11:

- Покращена продуктивність навіть на обмежених ресурсах завдяки оптимізації кількості параметрів.
- Універсальність використання в різних сценаріях, від мобільних пристроїв до хмарних платформ.
- Висока швидкість та точність для завдань реального часу.

#### Висновок.

Кожна нова версія YOLO принесла важливі покращення, орієнтовані на підвищення продуктивності, точності та швидкості детекції об'єктів. Починаючи з YOLOv3, що забезпечив значний прорив у швидкості роботи алгоритму, і до останньої версії YOLOv11, яка забезпечує ще кращі результати, кожна з цих версій відповідає вимогам сучасних завдань комп'ютерного зору та машинного навчання.

## **РОЗДІЛ 4**

### **РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВІДЕОСПОСТЕРЕЖЕННЯ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ YOLO ТА МЕТОДІВ ТРЕКІНГУ DEEPSORT**

Розроблений застосунок для відеоспостереження є інтегрованою системою, яка поєднує сучасні методи комп'ютерного зору та алгоритми нейронних мереж для ефективного аналізу відеопотоку в реальному часі. Використовуючи модель YOLO (You Only Look Once), а саме «yolov8n», для детекції об'єктів та алгоритми трекінгу (DeepSort), система забезпечує високоточне виявлення і відстеження об'єктів. Додаткові функції, такі як запис відео, збереження скріншотів, логування та створення звітів, роблять застосунок універсальним рішенням для різноманітних задач відеоспостереження.

Для реалізації системи відеоспостереження використано Python — високорівневу мову програмування, яка забезпечує простоту написання коду та має багатий екосистему.

#### **4.1 Технологічне забезпечення та архітектура системи відеоспостереження.**

Бібліотеки та фреймворки:

1. OpenCV - використовується для захоплення відеопотоку, обробки кадрів та відображення результатів. OpenCV надає інструменти для роботи з відео, зображеннями, виявленням та відстеженням об'єктів.
2. PyTorch - головний фреймворк для глибокого навчання, використовується для роботи з нейронною мережею YOLO. PyTorch дозволяє запускати передтреновані моделі та швидко адаптувати їх до нових задач.
3. Ultralytics YOLO - модель YOLOv8 для детекції об'єктів у реальному часі. Ця модель забезпечує високу точність та продуктивність.
4. DeepSort - використовується для трекінгу об'єктів. DeepSort базується на методі асоціації об'єктів у послідовності кадрів та дозволяє відстежувати траєкторії.

5. PyQt5 - бібліотека для створення графічного інтерфейсу користувача. Забезпечує інтерактивне управління системою, включаючи зміну параметрів детекції, перегляд відеопотоку та збереження скріншотів.

Інструменти та середовище розробки:

1. PyCharm - основне середовище розробки коду з підтримкою автодоповнення, налагодження та інтеграції з Git.
2. Git - використовується для управління версіями проєкту.
3. Python Virtual Environment - віртуальне середовище для ізоляції залежностей проєкту.
4. Hardware:
  - Apple M1/M2 для прискорення обчислень

Використані формати даних

- MP4: Формат збереження відео.
- PNG/JPEG: Формат для скріншотів.
- TXT: Логи детекції об'єктів.

Загальна архітектура:

Розроблений застосунок складається з таких основних модулів:

1. Обробка відеопотоку (video\_recorder.py): реалізована у класі VideoProcessor. Цей модуль відповідає за захоплення відео, корекцію якості кадрів та передачу їх для подальшої обробки.
2. Детекція об'єктів (object\_detection.py): реалізована у класі ObjectDetector, що інтегрує модель YOLO. Клас підтримує фільтрацію об'єктів за класами та порогами впевненості.
3. Трекінг об'єктів (object\_tracking.py): використовується клас ObjectTracker, який інтегрує DeepSort. Цей модуль дозволяє відстежувати рух об'єктів у кадрі, навіть якщо вони тимчасово зникають із поля зору.
4. Логування та статистика (logger.py): запис інформації про виявлені об'єкти та активність системи реалізовано у класі Logger.

5. Графічний інтерфейс (object\_detection\_gui.py): Інтерфейс для взаємодії з користувачем реалізований за допомогою PyQt5. Він забезпечує керування системою (запуск запису, збереження скріншотів) та відображення відеопотоку.

## 4.2 Основні функції застосунку:

### 1.Виявлення об'єктів.

Використовується модель YOLO для виявлення об'єктів на кожному кадрі відео, з підписом класу та ID

```
# Завантаження моделі YOLO
self.model = YOLO(model_path)
self.model.to(self.device)
```

```
frame_tensor = torch.from_numpy(frame_resized).permute(2, 0, 1).unsqueeze(0).float().to(self.device)

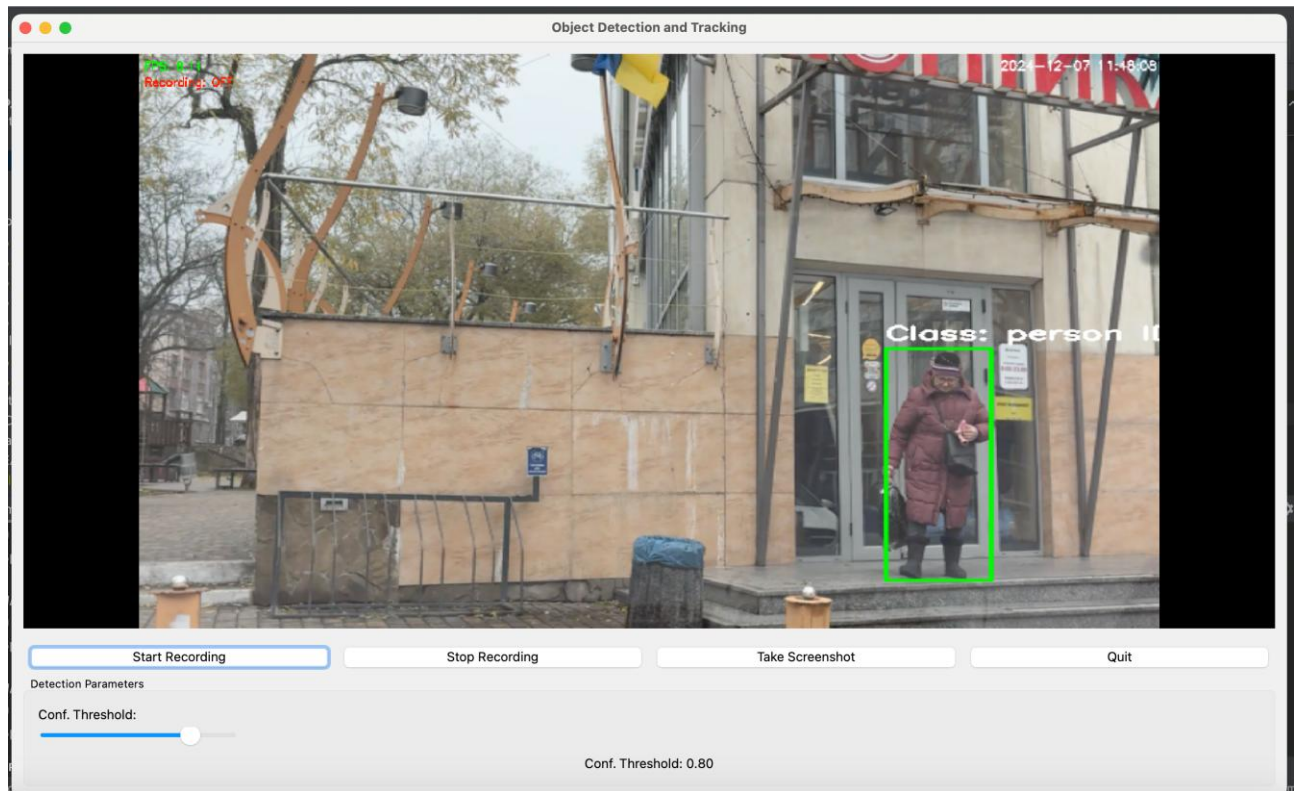
with torch.no_grad():
    detections = self.model(frame_tensor)[0]

if detections is None or len(detections.bboxes) == 0:
    return frame # Повернення без змін, якщо немає детекцій

results = self._extract_results(detections)
```



Приклад виявлення об'єкта :

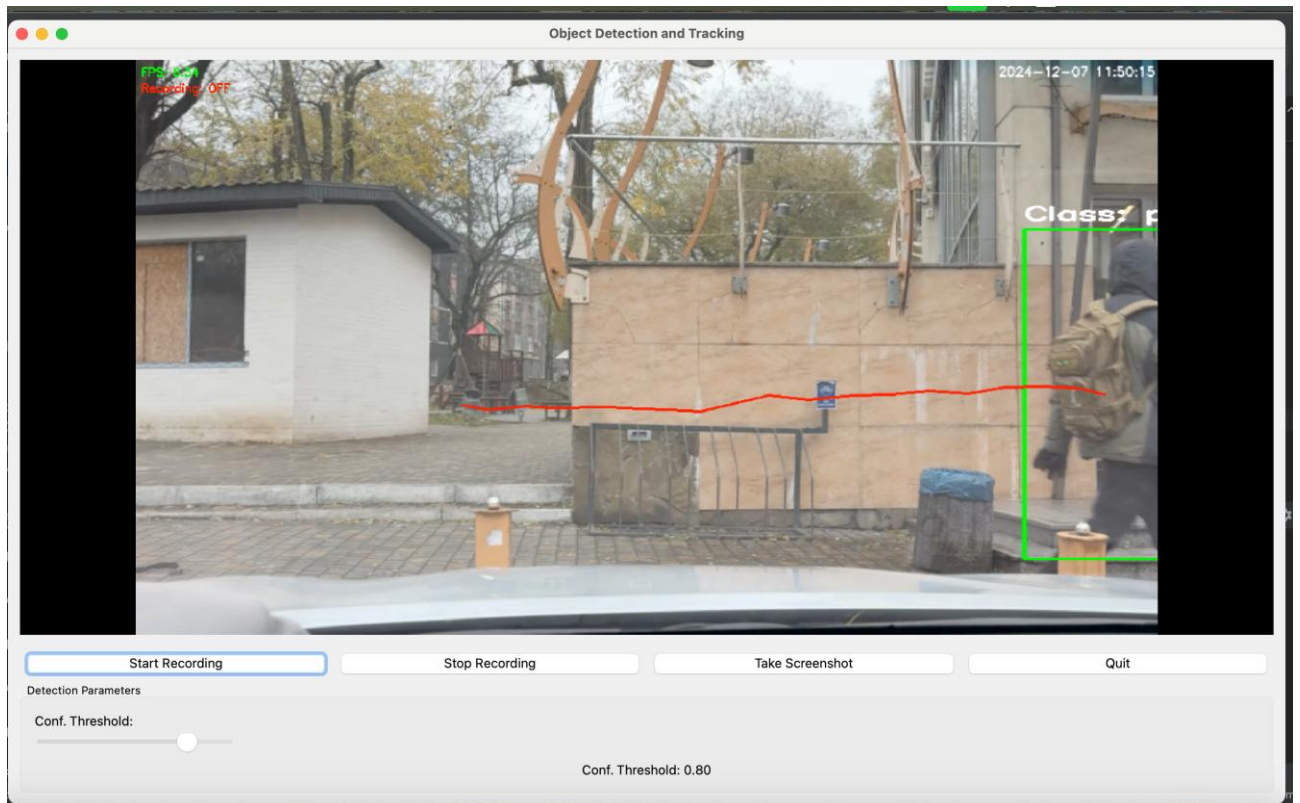


## 2. Відстеження об'єктів.

Для кожного виявленого об'єкта застосовується трекінг, що дозволяє зберігати унікальні ідентифікатори об'єктів протягом усього відеопотоку з відстеженням траєкторії руху (тільки для класу «person»).

```
"""Оновлення треків на основі результатів детекції."""  
tracks = self.tracker.update_tracks(results, frame=frame)  
self._update_trajectories(tracks) # Оновлення траєкторій  
self._draw_tracks(frame, tracks) # Виклик малювання треків  
return tracks
```

Приклад з траєкторією об'єкта:

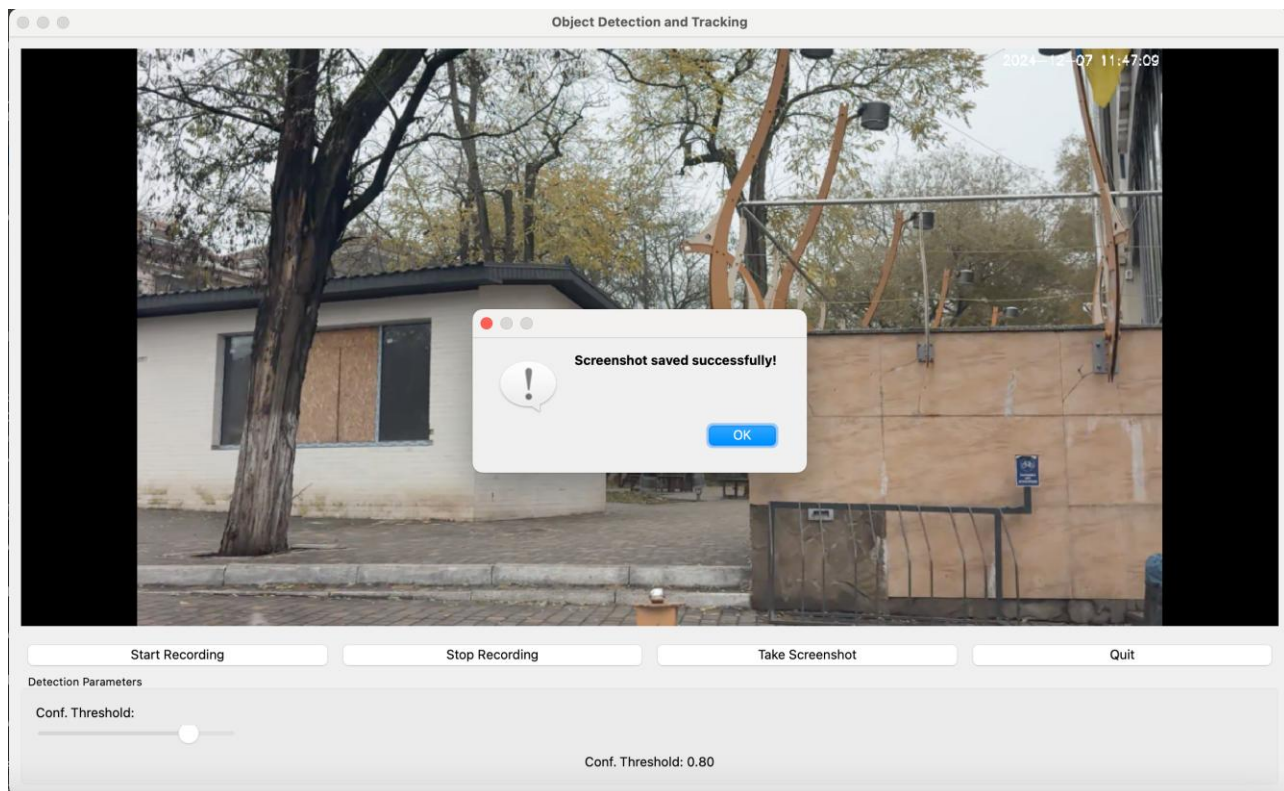


### 3. Запис відео та створення знімку екрану.

Користувач може ввімкнути або вимкнути запис відео, а також зробити знімок екрану у будь-який момент роботи.

```
def toggle_recording(self):  
    self.video_recorder.toggle_recording()  
  
#  
2 usages (2 dynamic)  Yuri Zubar  
def save_screenshot(self, frame):  
    self.video_recorder.save_screenshot(frame)
```

Під час створення знімку екрану, початку або зупинки відео з'являється відповідне сповіщення:



При створення відео в верхньому куті екрану напис Recording: ON



та відлік часу з початку відео



Після зупинки відео: Recording: OFF



#### 4. Логування та створення звітів.

Кожен виявлений і відстежений об'єкт записується до файлу із зазначенням часу, ідентифікатора та класу об'єкта.

```
def _log_detections(self, tracks):
    """Логування виявлених об'єктів через Logger."""
    detected_objects = [f"Time: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}, "
                        f"Class: {self.model.names[track.det_class]}, ID: {track.track_id}"
                        for track in tracks if track.is_confirmed()]
    self.logger.log_detections(detected_objects)
```

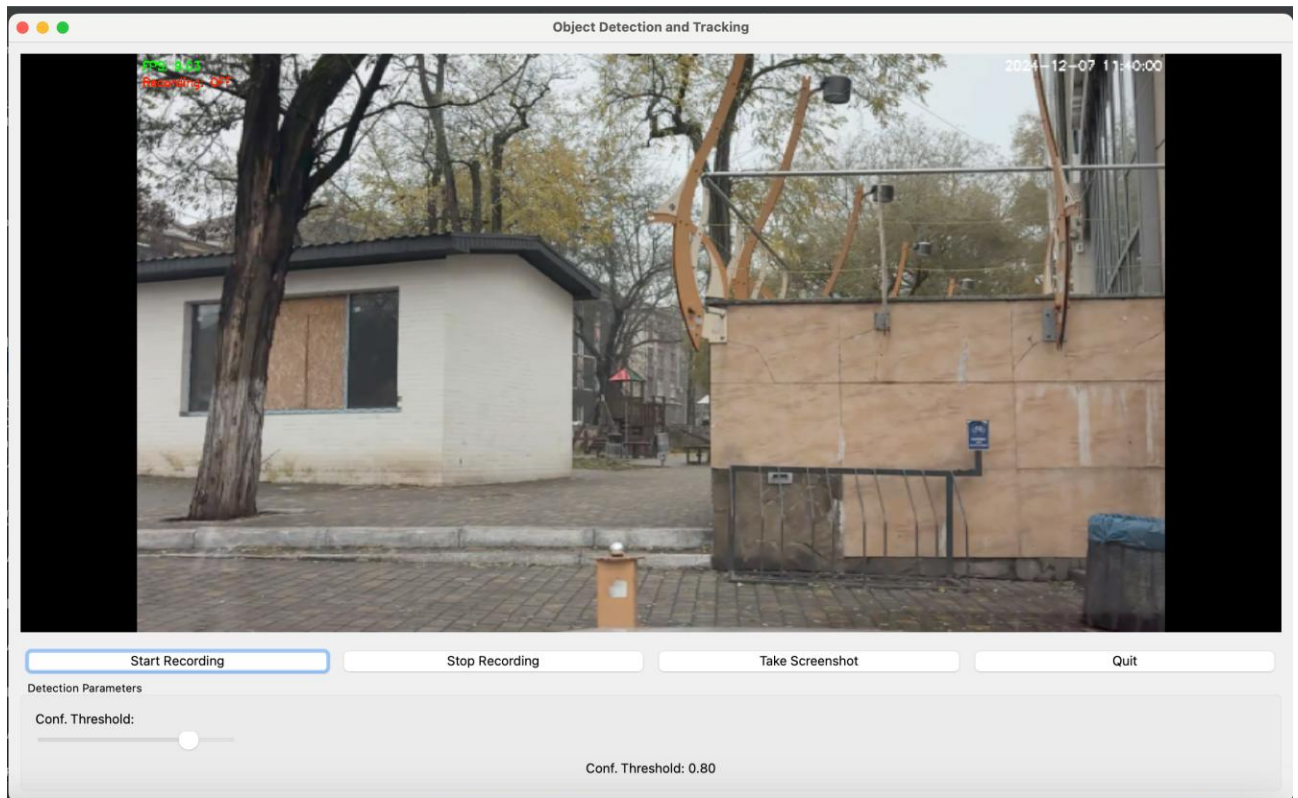
Приклад звіту:

```
Object Detection and Tracking Log
Time: 2024-12-07 11:42:54, Class: person, ID: 1
Time: 2024-12-07 11:42:54, Class: person, ID: 1
Time: 2024-12-07 11:42:54, Class: person, ID: 1
Time: 2024-12-07 11:42:54, Class: person, ID: 1
Time: 2024-12-07 11:42:54, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:55, Class: person, ID: 1
Time: 2024-12-07 11:42:56, Class: person, ID: 1
Time: 2024-12-07 11:42:56, Class: person, ID: 1
Time: 2024-12-07 11:42:56, Class: person, ID: 1
Time: 2024-12-07 11:42:56, Class: person, ID: 1
Time: 2024-12-07 11:42:56, Class: person, ID: 1
```

#### 5. Гнучке управління параметрами.



Через графічний інтерфейс користувач може змінювати параметри, такі як поріг довіри (confidence threshold), робити запис або зупинити систему.



Розроблений застосунок демонструє практичну реалізацію сучасних алгоритмів комп'ютерного зору, таких як детекція об'єктів на основі нейронної мережі YOLO та трекінг із використанням DeepSort. Поєднання цих методів забезпечує точне виявлення та відстеження об'єктів у реальному часі, що є важливим кроком у створенні інтелектуальних систем відеоспостереження.

## ВИСНОВКИ

У процесі виконання дипломної роботи на тему “Розробка та впровадження системи відеоспостереження на основі нейронних мереж YOLO та методів трекінгу для детекції та відстеження об’єктів у реальному часі” було досягнуто поставленої мети та реалізовано всі завдання дослідження.

Основні результати роботи:

### 1. Теоретичні аспекти:

- Проведено огляд сучасних методів детекції та трекінгу об’єктів, що базуються на нейронних мережах.
- Досліджено переваги й недоліки алгоритмів YOLO для детекції та DeepSort для трекінгу об’єктів у реальному часі.
- Обґрунтовано доцільність використання інтегрованої моделі YOLO та DeepSort у системах відеоспостереження.

### 2. Практичні результати:

- Розроблено модульну програмну систему для відеоспостереження, яка дозволяє здійснювати високоточну детекцію та трекінг об’єктів у реальному часі.
- Реалізовано гнучкий графічний інтерфейс, що забезпечує зручне керування системою та налаштування параметрів (поріг довіри, вибір області інтересу тощо).
- Впроваджено можливість запису відео, збереження знімків екрану, а також візуалізації ідентифікаторів об’єктів і їхніх траєкторій руху.
- Оптимізовано алгоритми для роботи на комп’ютерах із середнім рівнем продуктивності завдяки підтримці обчислень на GPU.

### 3. Наукова новизна та практична цінність:

- Запропоновано інтеграцію алгоритмів YOLO та DeepSort у контексті гнучкої системи відеоспостереження з налаштуванням параметрів.

Рекомендації щодо використання отриманих результатів:

- Розроблений застосунок може бути використаний для навчальних цілей у галузі комп'ютерного зору.
- Система підходить для впровадження у побутових умовах, де потрібен ефективний контроль за об'єктами.
- Подальше вдосконалення алгоритмів детекції та трекінгу, інтеграція з хмарними платформами та додавання сповіщень можуть значно розширити можливості системи.

Перспективи подальших досліджень:

- Розширення функціоналу системи шляхом впровадження аналізу поведінки об'єктів, прогнозування їхніх траєкторій та інтеграції методів розпізнавання обличь.
- Дослідження можливостей використання інших нейронних мереж для поліпшення точності та швидкості обробки відео.
- Впровадження алгоритмів обробки великих даних для аналізу масових потоків об'єктів у реальному часі.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Li, F.-F., Karpathy, A., & Johnson, J. "Convolutional Neural Networks for Visual Recognition".
2. Goodfellow, I., Bengio, Y., & Courville, A. "Deep Learning".
3. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization.
4. LeCun, Y., Bengio, Y., & Hinton, G. E. (2015). Deep learning. Nature.
5. Szeliski, R. "Computer Vision: Algorithms and Applications".
6. Krizhevsky, A., Sutskever, I., & Hinton, G. E. "ImageNet Classification with Deep Convolutional Neural Networks".
7. Szeliski, R. "Computer Vision: Algorithms and Applications".
8. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. "Gradient-Based Learning Applied to Document Recognition".
9. Murphy, K. P. "Machine Learning: A Probabilistic Perspective".
10. Khan, S. & Shah, M. "A Survey of Approaches and Applications of Optical Flow".
11. Geiger, A. et al. "Vision Meets Robotics: The KITTI Dataset".
12. Zikopoulos, P. & Eaton, C. "Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data".
13. O'Neil, C. "Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy".
14. Torralba, A. & Efros, A. A. "Unbiased Look at Dataset Bias".
15. Dalal & Triggs, 2005, Histograms of Oriented Gradients for Human Detection.
16. Lowe, 2004, Distinctive Image Features from Scale-Invariant Keypoints.
17. Bay et al., 2006, SURF: Speeded Up Robust Features.
18. Canny, 1986, A Computational Approach to Edge Detection.
19. Ojala et al., 2002, Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns.
20. Redmon et al., 2016, You Only Look Once: Unified, Real-Time Object Detection.



21. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. European Conference on Computer Vision (ECCV).
22. Ren et al., 2015, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
23. Lin et al., 2017, Focal Loss for Dense Object Detection.
24. Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and Efficient Object Detection.
25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. Advances in Neural Information Processing Systems (NeurIPS).
26. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. European Conference on Computer Vision (ECCV).
27. DeepSORT (Wojke et al., 2017) TrackR-CNN” (Choi et al., 2018).
28. “Efficient Object Detection on Mobile Devices” (Howard et al., 2017).
29. “Real-Time Object Detection for Autonomous Vehicles” (Chen et al., 2020).
30. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. European Conference on Computer Vision (ECCV).
31. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement.
32. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.
33. Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2022). YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors.

## ДОДАТКИ

### 1. object\_detection.py

```
import datetime
import cv2
import torch
from ultralytics import YOLO

from utils.helper import calculate_fps, draw_text,
filter_image, draw_datetime
from utils.config import classes, colors

class ObjectDetectionAndTracking:
    def __init__(self, model_path,
confidence_threshold=0.7, allowed_classes=classes,
logger=None, tracker=None,
video_recorder=None, output_dir=None):
        self.allowed_classes = allowed_classes
        self.output_dir = output_dir
        self.logger = logger
        self.confidence_threshold = confidence_threshold
        self.RED = colors.get('RED')
        self.GREEN = colors.get('GREEN')

        # Ініціалізація пристрою для обчислень
        self.device = torch.device("mps" if
torch.backends.mps.is_available() else "cpu")
        print(f"Using device: {self.device}")

        # Ініціалізація відеозахоплення
        self.video_recorder = video_recorder
        self.video_cap = self.video_recorder.video_cap

        # Завантаження моделі YOLO
        self.model = YOLO(model_path)
        self.model.to(self.device)

        # Ініціалізація трекара DeepSort
        self.tracker = tracker

        # Час про скріншот
        self.screenshot_notification_time = None

        # Ініціалізація FPS
        self.prev_time = datetime.datetime.now()
```

```

        self.fps = 0.0

    def process_frame(self):
        """Обробка одного кадру для детекції та
трекінгу."""
        # Фільтрація зображення
        frame =
filter_image(self.video_recorder.read_frame())

        # Додавання часу та дати
        draw_datetime(frame)

        original_size = frame.shape[1::-1] # Оригінальні
(ширина, висота)
        frame_resized = cv2.resize(frame, (640, 640))

        # Підготовка кадру для моделі
        frame_tensor =
torch.from_numpy(frame_resized).permute(2, 0, 1).unsqueeze(0).float().to(self.device)

        with torch.no_grad():
            detections = self.model(frame_tensor)[0]

        if detections is None or len(detections.bboxes) ==
0:
            return frame # Повернення без змін, якщо немає
детекцій

        results = self._extract_results(detections)

        # Оновлення трекара
        tracks = self.tracker.update_tracks(results,
frame=frame_resized, )

        # Логування інформації
        self._log_detections(tracks)

        # Масштабування до оригінальних розмірів
        frame_output = cv2.resize(frame_resized,
original_size)

        # Додавання FPS, статусу та сповіщень
        self.fps = calculate_fps(self.prev_time)
        self.prev_time = datetime.datetime.now()
        draw_text(frame_output, f"FPS: {self.fps:.2f}",
(10, 30), (0, 255, 0))

```

```

        self._draw_status(frame_output)

    return frame_output

def _extract_results(self, detections):
    """Обробка результатів детекції YOLO."""
    results = []
    for det in detections.bboxes.data.cpu().numpy():
        confidence = det[4]

        if confidence < self.confidence_threshold:
            continue
        class_id = int(det[5])
        class_name = self.model.names.get(class_id,
"Unknown")

        if class_name not in self.allowed_classes: #
Фільтр класів
            continue
        x_min, y_min, x_max, y_max = map(int, det[:4])
        width, height = x_max - x_min, y_max - y_min
        results.append([x_min, y_min, width, height],
confidence, class_id])
    return results

def _log_detections(self, tracks):
    """Логування виявлених об'єктів через Logger."""
    detected_objects = [f"Time:
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')},
"
                        f"Class:
{self.model.names[track.det_class]}, ID:
{track.track_id}"]
    for track in tracks if
track.is_confirmed()]
        self.logger.log_detections(detected_objects)

def _draw_status(self, frame):
    """Малювання статусу запису."""
    status_text = f"Recording: {'ON' if
self.video_recorder.is_recording else 'OFF'}"
    status_color = self.GREEN if
self.video_recorder.is_recording else self.RED
    draw_text(frame, status_text, (10, 60),
status_color)

def toggle_recording(self):

```

```

        self.video_recorder.toggle_recording()

#
def save_screenshot(self, frame):
    self.video_recorder.save_screenshot(frame)

def run(self):
    """Основний цикл програми для обробки відеопотоку."""
    try:
        while True:
            ret, frame = self.video_recorder.read_frame()
            if not ret:
                print("Не вдалося отримати кадр. Завершення роботи.")
                break

            # Обробка кадру
            processed_frame = self.process_frame(frame)

            # Виведення на екран
            cv2.imshow("Video Feed", processed_frame)

            # Обробка натиснень клавіш
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'): # Вихід за запитом
                break

    except Exception as e:
        print(f"Сталася помилка: {e}")
    finally:
        self.video_recorder.release()
        cv2.destroyAllWindows()
        print("Роботу завершено.")

```

## 2. object\_tracking.py

```

import cv2
from deep_sort_realtime.deepsort_tracker import DeepSort
from utils.helper import draw_text
from utils.config import deep_sort_params, classes, colors

class ObjectTracking:
    def __init__(self):

```

```

        self.tracker = DeepSort(**deep_sort_params)
        self.trajectories = {}
        self.GREEN = colors.get('GREEN')    # Колір для
"person"
        self.BLUE = colors.get('BLUE')    # Колір для інших
об'єктів
        self.WHITE = colors.get('WHITE')
        self.RED = colors.get('RED')
        self.classes = classes

    def update_tracks(self, results, frame):
        """Оновлення треків на основі результатів
детекції."""
        tracks = self.tracker.update_tracks(results,
frame=frame)
        self._update_trajectories(tracks)    # Оновлення
траєкторій
        self._draw_tracks(frame, tracks)    # Виклик
малювання треків
        return tracks

    def _update_trajectories(self, tracks):
        """Оновлення траєкторій для активних треків."""
        for track in tracks:
            if not track.is_confirmed():
                continue
            track_id = track.track_id
            center_x = int((track.to_tlbr()[0] +
track.to_tlbr()[2]) / 2)
            center_y = int((track.to_tlbr()[1] +
track.to_tlbr()[3]) / 2)
            if track_id not in self.trajectories:
                self.trajectories[track_id] = []
            self.trajectories[track_id].append((center_x,
center_y))

    def _draw_tracks(self, frame, tracks):
        """Малювання треків на кадрі."""
        for track in tracks:
            if not track.is_confirmed():
                continue
            track_id = track.track_id
            class_id = track.det_class
            x_min, y_min, x_max, y_max = map(int,
track.to_ltrb())
            color = self.GREEN if class_id == 0 else
self.BLUE

```

```

        cv2.rectangle(frame, (x_min, y_min), (x_max,
y_max), color, 2)
        text = f"Class: {self.classes[class_id]} ID:
{track_id}"
        draw_text(frame, text, (x_min, y_min - 10),
self.WHITE)

        if class_id == 0:
            # Малювання траєкторій
            if track_id in self.trajectories:
                points = self.trajectories[track_id]
                for i in range(1, len(points)):
                    cv2.line(frame, points[i - 1],
points[i], self.RED, 2)

```

### 3. video\_recorder.py

```

import cv2
import os
import datetime

from utils.helper import create_video_writer,
save_screenshot, draw_recording_timer

class VideoRecorder:
    def __init__(self, video_source, output_dir):

        self.video_cap = cv2.VideoCapture(video_source)
        if not self.video_cap.isOpened():
            raise RuntimeError(f"Error: Cannot open video
source {video_source}")

        self.is_recording = False
        self.writer = None
        self.record_start_time = None
        self.record_end_time = None
        self.output_dir = output_dir
        self.output_file = None
        self.movies_dir = os.path.join(output_dir,
"movies")
        self.screenshot_dir = os.path.join(output_dir,
"screenshots")
        os.makedirs(self.movies_dir, exist_ok=True)
        os.makedirs(self.screenshot_dir, exist_ok=True)

```

```

def toggle_recording(self):
    """Перемикання запису."""
    if self.is_recording:
        self.is_recording = False
        self.record_end_time =
datetime.datetime.now()
        if self.writer:
            self.writer.release()
            self.writer = None
    else:
        self.is_recording = True
        self.record_start_time =
datetime.datetime.now()
        timestamp =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        self.output_file =
os.path.join(self.movies_dir, f"output_{timestamp}.mp4")
        self.writer =
create_video_writer(self.video_cap, self.output_file)

def save_screenshot(self, frame):
    """Збереження скріншоту."""
    save_screenshot(frame, self.screenshot_dir)

def read_frame(self):
    ret, frame = self.video_cap.read()

    # Додавання інформації про запис із секундоміром
    if self.is_recording:
        draw_recording_timer(frame,
self.record_start_time)

    if not ret:
        raise RuntimeError("Failed to read frame from
video source")
    return frame

def release(self):
    self.video_cap.release()
    if self.writer:
        self.writer.release()

```

#### 4. logger.py

```
import os
```



```

class Logger:
    def __init__(self, output_dir):
        self.logs_dir = os.path.join(output_dir, "logs")
        os.makedirs(self.logs_dir, exist_ok=True)
        self.log_file = os.path.join(self.logs_dir,
"activity_log.txt")
        self._initialize_log()

    def _initialize_log(self):
        """Ініціалізація лог-файлу."""
        with open(self.log_file, "w") as f:
            f.write("Object Detection and Tracking Log\n")

    def log_detections(self, detected_objects):
        """Записує інформацію про виявлені об'єкти."""
        if not detected_objects:
            return
        with open(self.log_file, "a") as f:
            for obj in detected_objects:
                f.write(f"{obj}\n")

```

## 5. object\_detection\_gui.py

```

from PyQt5.QtWidgets import QMainWindow, QLabel,
QPushButton, QVBoxLayout, QWidget, QHBoxLayout,
QMessageBox, \
    QFormLayout, QGroupBox, QSpinBox, QSlider
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtCore import QTimer, Qt
import cv2

class ObjectDetectionGUI(QMainWindow):
    def __init__(self, app):
        super().__init__()
        self.app = app
        self.running = True

        # Налаштування вікна
        self.setWindowTitle("Object Detection and
Tracking")
        self.setGeometry(100, 100, 1280, 720)

        # Центральний віджет
        self.central_widget = QWidget(self)
        self.setCentralWidget(self.central_widget)

```

```

        # Відео віджет
        self.video_label = QLabel(self)
        self.video_label.setStyleSheet("background-color:
black;")
        self.video_label.setAlignment(Qt.AlignCenter)
        self.video_label.setMinimumSize(1024, 576)

        # Кнопки
        self.start_button = QPushButton("Start Recording")

self.start_button.clicked.connect(self.start_recording)

        self.stop_button = QPushButton("Stop Recording")

self.stop_button.clicked.connect(self.stop_recording)

        self.screenshot_button = QPushButton("Take
Screenshot")

self.screenshot_button.clicked.connect(self.take_screensh
ot)

        self.quit_button = QPushButton("Quit")

self.quit_button.clicked.connect(self.quit_application)

        # Розташування кнопок
        buttons_layout = QHBoxLayout()
        buttons_layout.addWidget(self.start_button)
        buttons_layout.addWidget(self.stop_button)
        buttons_layout.addWidget(self.screenshot_button)
        buttons_layout.addWidget(self.quit_button)

        # Параметри управління
        self.confidence_slider = QSlider(Qt.Horizontal)
        self.confidence_slider.setMinimum(1)
        self.confidence_slider.setMaximum(100)

self.confidence_slider.setValue(int(self.app.confidence_t
hreshold * 100))

self.confidence_slider.valueChanged.connect(self.update_c
onfidence_threshold)
        self.confidence_slider.setFixedSize(200, 20)      #
Ширина: 200 пікселів, висота: 20 пікселів

```

```

        self.confidence_label = QLabel(f"Conf. Threshold:
{self.app.confidence_threshold:.2f}")

self.confidence_label.setAlignment(Qt.AlignCenter)

        # Макет управління параметрами
        parameter_layout = QVBoxLayout()
        parameter_layout.addWidget(QLabel("Conf.
Threshold:"))

parameter_layout.addWidget(self.confidence_slider)
        parameter_layout.addWidget(self.confidence_label)

        parameter_group = QGroupBox("Detection
Parameters")
        parameter_group.setLayout(parameter_layout)

        # Основний макет
        layout = QVBoxLayout(self.central_widget)
        layout.addWidget(self.video_label)
        layout.addLayout(buttons_layout)
        layout.addWidget(parameter_group)

        # Таймер для оновлення відео
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_video)
        self.timer.start(30)

def update_video(self):
    """Оновлення відеопотоку."""
    if not self.running:
        return

    ret, frame = self.app.video_cap.read()
    if ret:
        frame = self.app.process_frame()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Перетворення у QImage
        height, width, channel = frame.shape
        bytes_per_line = channel * width
        qimg = QImage(frame.data, width, height,
bytes_per_line, QImage.Format_RGB888)

        # Відображення на QLabel
        pixmap = QPixmap.fromImage(qimg)
        self.video_label.setPixmap(pixmap.scaled(

```

```

        self.video_label.width(),
        self.video_label.height(),
        Qt.KeepAspectRatio
    ))

    def start_recording(self):
        if self.app.video_recorder.is_recording:
            QMessageBox.warning(self, "Recording",
"Recording is already in progress!")
        else:
            self.app.toggle_recording()
            QMessageBox.information(self, "Recording",
"Recording started.")

    def stop_recording(self):
        if not self.app.video_recorder.is_recording:
            QMessageBox.warning(self, "Recording",
"Recording is not active!")
        else:
            self.app.toggle_recording()
            QMessageBox.information(self, "Recording",
"Recording stopped.")

    def take_screenshot(self):
        ret, frame = self.app.video_cap.read()
        if ret:
            self.app.save_screenshot(frame)
            QMessageBox.information(self, "Screenshot",
"Screenshot saved successfully!")
        else:
            QMessageBox.warning(self, "Screenshot",
"Failed to capture screenshot.")

    def update_confidence_threshold(self):
        """Оновлення порогу довіри."""
        new_threshold = self.confidence_slider.value() /
100.0
        self.app.confidence_threshold = new_threshold
        self.confidence_label.setText(f"Confidence
Threshold: {new_threshold:.2f}")

    def quit_application(self):
        self.running = False
        self.app.video_cap.release()
        if self.app.video_recorder.writer is not None:
            self.app.writer.release()
        self.close()

```

