Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(освітньо-кваліфікаційний рівень)
на тему
Розпізнавання об'єктів в реальному часі з використанням глибокого навчання

Виконав: студент 2 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Руднік О.С.
(прізвище та ініціали)
Керівник Краснобаєв В.А.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Полтава – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Спеціальність 122 «Комп'ютерні Науки»

на тему

«Розпізнавання об'єктів в реальному часі з використанням глибокого навчання»»

Студента групи 601-ТН Рудніка Олексія Степановича

Керівник роботи
Доктор технічних наук,
професор Краснобаєв В.А.

Завідувач кафедри
Кандидат фізико-математичних наук
Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 100 с., 24 малюнків, 2 додатки, 1 таблиця, 17 джерел.

Об’єкт дослідження: алгоритми глибокого навчання для розпізнавання об’єктів у реальному часі.

Мета роботи: аналіз ефективності моделей YOLOv8, MobileNet-SSD та Faster R-CNN у задачі розпізнавання об'єктів на основі датасету Pascal VOC 2012 та визначення їхньої доцільності для використання в реальних умовах під певні задачі.

Методи: експериментальне дослідження продуктивності та точності моделей, порівняння результатів із анотаціями, а також аналіз метрик mAP, FPS та затримки обробки.

Ключові слова: розпізнавання об'єктів, глибоке навчання, YOLOv8, MobileNet-SSD, Faster R-CNN, Pascal VOC, mAP, реальний час.

ABSTRACT

Master's qualification work: 100 p., 24 figures, 2 appendices, 1 table, 17 sources.

Object of research: deep learning algorithms for real-time object recognition.

Purpose of work: analysis of the effectiveness of the YOLOv8, MobileNet-SSD and Faster R-CNN models in the object recognition task based on the Pascal VOC 2012 dataset and determination of their feasibility for use in real conditions.

Methods: experimental study of the performance and accuracy of the models, comparison of results with annotations, as well as analysis of mAP, FPS and processing latency metrics.

Keywords: object recognition, deep learning, YOLOv8, MobileNet-SSD, Faster R-CNN, Pascal VOC, mAP, real-time.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (Artificial Intelligence) — штучний інтелект.

CNN (Convolutional Neural Network) — згорткова нейронна мережа, модель глибинного навчання, яка використовується для аналізу зображень.

VOC (Visual Object Classes) — набір даних для задач розпізнавання об'єктів.

YOLO (You Only Look Once) — сімейство моделей для детекції об'єктів у реальному часі.

SSD (Single Shot MultiBox Detector) — нейронна мережа для детекції об'єктів, яка виконує задачу в один прохід.

R-CNN (Region-based Convolutional Neural Network) — метод детекції об'єктів, що використовує регіональні пропозиції.

mAP (mean Average Precision) — середня точність, метрика оцінки точності детекції об'єктів.

GPU (Graphics Processing Unit) — графічний процесор, який використовується для прискорення обчислень у глибинному навчанні.

PyTorch — фреймворк для глибинного навчання, що дозволяє створювати нейронні мережі.

ReLU (Rectified Linear Unit) — активаційна функція у нейронних мережах.

IoU (Intersection over Union) — метрика, що використовується для оцінки якості накладання передбачених і реальних рамок об'єктів.

Batch Size — розмір партії даних, що обробляється за одну ітерацію навчання.

FPS (Frames Per Second) — кількість кадрів, які модель обробляє за секунду, показник продуктивності для реального часу.

Anchor Box — заздалегідь визначений прямокутник, що використовується в детекції об'єктів для передбачення меж рамок.

NMS (Non-Maximum Suppression) — алгоритм, який використовується для видалення надлишкових або перекриваючихся передбачень об'єктів.

Confidence Score — оцінка впевненості моделі в тому, що знайдений об'єкт належить до певного класу.

Dataset — набір даних, що використовується для навчання, тестування та валідації моделі.

Overfitting — ситуація, коли модель добре працює на тренувальних даних, але показує низьку точність на нових даних.

Pre-trained Model — модель, яка була попередньо навчена на великому наборі даних і може бути використана для інших задач.

ВСТУП

Сучасний розвиток технологій комп’ютерного зору та глибокого навчання відкриває нові можливості для автоматизації та оптимізації багатьох процесів у різних сферах людської діяльності. Однією з ключових задач у цій галузі є розпізнавання об’єктів, яке дозволяє автоматично ідентифікувати та локалізувати об’єкти на зображеннях чи відеопотоках. Завдяки своїй універсальності, ця задача знаходить застосування у таких сферах, як автономний транспорт, відеоспостереження, медична діагностика, робототехніка, розумні міста та багато інших.

Реалізація розпізнавання об’єктів у реальному часі є особливо актуальною, адже вона потребує не лише високої точності, але й мінімальних затримок у обробці даних. Це висуває високі вимоги до ефективності алгоритмів, їхньої здатності працювати на обмежених обчислювальних ресурсах та адаптації до змін середовища.

Серед методів, які сьогодні демонструють високі результати у задачах розпізнавання об’єктів, особливу увагу привертають архітектури, засновані на згорткових нейронних мережах (CNN). Такі моделі, як YOLO (You Only Look Once), MobileNet-SSD та Faster R-CNN, стали стандартом у галузі завдяки своїй здатності забезпечувати баланс між точністю та швидкістю.

Мета даної роботи полягає у порівнянні точності та продуктивності зазначених моделей на основі експериментів з використанням датасету Pascal VOC 2012. У ході роботи буде проведено аналіз переваг і недоліків кожної з архітектур, визначено їхні особливості у контексті розпізнавання об’єктів у реальному часі, а також запропоновано рекомендації щодо вибору моделі залежно від конкретних умов застосування.

Дослідження спрямоване на розширення знань у сфері комп’ютерного зору, що може знайти практичне застосування як у промисловості, так і у наукових дослідженнях, сприяючи подальшому розвитку технологій автоматичного аналізу зображень.

Основною метою роботи є дослідження ефективності сучасних архітектур глибокого навчання, таких як YOLOv8, MobileNet-SSD та Faster R-CNN, у задачі розпізнавання об’єктів у реальному часі. Задля досягнення цієї мети необхідно провести аналіз точності моделей на основі метрик, таких як середнє значення точності (mAP), оцінити їхню продуктивність за параметрами швидкості обробки (FPS) та затримки, а також порівняти результати з реальними анотаціями, використовуючи дані з датасету Pascal VOC 2012. На основі отриманих результатів буде сформульовано висновки про доцільність застосування цих моделей у конкретних задачах.

Актуальність мети полягає в необхідності вирішення задачі розпізнавання об'єктів у реальному часі, яка є важливою складовою багатьох сучасних технологій. Зокрема, це стосується систем відеоспостереження, автономного транспорту, медичних діагностичних рішень та робототехніки. Використання моделей глибинного навчання, таких як YOLOv8, MobileNet-SSD та Faster R-CNN, дозволяє досягти високих показників точності. Однак проблема забезпечення ефективної роботи моделей у реальному часі залишається відкритою. Проведення експериментів із різними моделями на реальних наборах даних, таких як Pascal VOC 2012, є важливим для пошуку оптимального рішення, яке забезпечить баланс між точністю та швидкістю розпізнавання в динамічних умовах.

Об’єктом дослідження є архітектури глибокого навчання, які використовуються для розпізнавання об’єктів. У роботі акцент зроблено на трьох моделях: YOLOv8, MobileNet-SSD та Faster R-CNN. Ці моделі обрано через їхню популярність, відмінності в архітектурних підходах та здатність вирішувати задачу розпізнавання об’єктів із різним співвідношенням точності й швидкості.

Методи дослідження включають аналіз літературних джерел для вивчення сучасних підходів до розпізнавання об’єктів, експериментальне дослідження на основі обраного датасету, використання метрик точності (mAP, Recall, Precision) та продуктивності (FPS, затримка обробки) для оцінки

результатів. Крім того, у роботі застосовано програмну реалізацію для тестування, візуалізації та застосування роботи моделей, що дозволяє забезпечити наочність отриманих висновків.

РОЗДІЛ 1.
ОГЛЯД ЛІТЕРАТУРИ

1.1 Історичний розвиток задач розпізнавання об'єктів

Розпізнавання об'єктів є однією з фундаментальних задач комп'ютерного зору, яка має тривалу історію розвитку, що починається задовго до появи сучасних методів глибинного навчання. Основна мета цієї задачі полягає у виявленні, класифікації та локалізації об'єктів на зображеннях або у відео. Еволюція підходів до її вирішення проходила кілька ключових етапів, кожен із яких зробив значний внесок у досягнення сучасного стану технології.

До появи методів глибинного навчання розпізнавання об'єктів базувалося на використанні класичних алгоритмів обробки зображень та машинного навчання. У цьому періоді (1980-ті – початок 2000-х років) основна увага приділялася розробці алгоритмів для виділення ознак, таких як крайові детектори (наприклад, метод Кані) або дескриптори, наприклад SIFT (Scale-Invariant Feature Transform) та HOG (Histogram of Oriented Gradients).

Ці методи використовувалися у поєднанні з традиційними алгоритмами класифікації, такими як опорно векторні машини (SVM) або методи найближчих сусідів (k-NN). Хоча вони мали обмежений рівень точності, їх використання дозволило досягти певних успіхів у розпізнаванні об'єктів у контрольованих умовах, наприклад, у розпізнаванні тексту або обличчя.

Поява перших нейронних мереж у 1980-х роках відкрила нові перспективи для задач комп'ютерного зору. Проте ранні нейронні мережі мали обмежену продуктивність через низьку обчислювальну потужність та недостатню кількість даних для навчання. Лише у 2010-х роках завдяки швидкому розвитку обчислювальних технологій (графічні процесори, паралельні обчислення) та появі великих наборів даних, таких як ImageNet, стало можливим ефективне навчання глибоких нейронних мереж.

Справжній прорив у задачі розпізнавання об'єктів стався у 2012 році, коли модель AlexNet, побудована на базі згорткових нейронних мереж (CNN), показала значне поліпшення результатів у конкурсі ImageNet. CNN-архітектури стали основою для багатьох сучасних підходів завдяки їх здатності автоматично виділяти релевантні ознаки із зображень. З часом були розроблені нові моделі, такі як VGG, ResNet, Inception, які підвищили точність і швидкість розпізнавання.

На сучасному етапі технологія розпізнавання об'єктів використовує глибокі нейронні мережі, зокрема трансформери (Vision Transformers) та інноваційні моделі, такі як EfficientDet і YOLO. Ці моделі здатні працювати у реальному часі, забезпечуючи високу точність навіть у складних умовах.

Розвиток розпізнавання об'єктів також стимулюється інтеграцією з іншими технологіями, такими як хмарні обчислення, квантовий комп'ютинг та технології зменшення енергоспоживання. Водночас, дослідники продовжують працювати над подоланням викликів, таких як адаптація до нових середовищ, навчання з обмеженими даними та етичні аспекти.

1.1.1 Від класичних методів до глибинного навчання. Розвиток методів розпізнавання об’єктів у комп’ютерному зорі пройшов кілька етапів, починаючи від класичних алгоритмів до сучасних підходів, які базуються на глибинному навчанні. Кожен з цих етапів суттєво вплинув на ефективність розпізнавання та його застосування у реальних задачах.

Класичні підходи до розпізнавання об’єктів, які домінували до 2010-х років, базувалися на ручному проектуванні ознак і використанні традиційних алгоритмів машинного навчання. Основними елементами таких систем були:

- детектори ознак: Алгоритми, які виділяли ключові точки на зображенні, наприклад, SIFT (Scale-Invariant Feature Transform) або SURF (Speeded-Up Robust Features). Ці методи дозволяли знаходити унікальні та інваріантні до масштабу й повороту ознаки об’єктів;

- методи класифікації: Для розпізнавання об’єктів використовувалися такі моделі, як SVM (Support Vector Machine), які аналізували заздалегідь обчислені ознаки;

- підходи до сегментації: Методи, такі як Watershed або алгоритми на основі порогового значення, використовувалися для виділення областей зображення.

Незважаючи на їхню інноваційність для свого часу, ці методи мали низку обмежень. Зокрема, вони вимагали суттєвих зусиль для проектування ознак, а їх ефективність значно знижувалася в умовах складного фону, освітлення або деформацій об’єктів.

З появою глибинного навчання і, зокрема, згорткових нейронних мереж (CNN), розпізнавання об’єктів зазнало революційних змін. Ключовими факторами, які зробили це можливим, були:

- доступ до великих даних: Відкриття масивних наборів даних, таких як ImageNet, дозволило тренувати нейронні мережі на мільйонах зображень;

- обчислювальні ресурси: Зростання продуктивності графічних процесорів (GPU) зробило можливим ефективне навчання глибоких моделей;

- автоматичне виділення ознак: На відміну від класичних підходів, нейронні мережі самостійно вивчають релевантні ознаки, що значно покращує результати розпізнавання.

Першим проривом у застосуванні глибинного навчання стала модель AlexNet, представлена у 2012 році. Вона показала значне покращення точності у задачах класифікації зображень та привернула увагу до використання CNN для інших задач комп’ютерного зору, таких як детекція та сегментація об’єктів.

Сучасні нейронні мережі, такі як YOLO (You Only Look Once), Faster R-CNN та Vision Transformers, вирізняються не лише високою точністю розпізнавання, але й здатністю працювати в реальному часі. Вони демонструють ефективність в обробці складних сцен, де присутні численні об’єкти різних типів, і здатні адаптуватися до нових задач завдяки технікам трансферного навчання.

Висока швидкість і ефективність цих підходів робить їх придатними для використання в реальних умовах.

Перехід від класичних методів до глибинного навчання став ключовим етапом у розвитку комп’ютерного зору, що значно розширило сферу застосування розпізнавання об’єктів. Ці технології знайшли своє місце у різних галузях, починаючи від автономного транспорту і завершуючи медичними діагностичними системами.

1.1.2 Роль комп'ютерного зору в сучасних технологіях. Комп’ютерний зір є однією з найбільш динамічних і прикладних галузей сучасного штучного інтелекту. Його інтеграція в різні технологічні системи вже сьогодні змінює спосіб роботи багатьох галузей, забезпечуючи автоматизацію процесів, підвищення ефективності та створення нових можливостей.

Однією з ключових сфер застосування комп’ютерного зору є автономний транспорт. Системи, що використовують технології розпізнавання об’єктів, дозволяють безпілотним автомобілям ідентифікувати дорожні знаки, пішоходів, транспортні засоби та інші елементи дорожньої інфраструктури. Ця здатність суттєво підвищує рівень безпеки на дорогах і дозволяє створювати ефективніші транспортні системи.

У медицині комп’ютерний зір відіграє критичну роль в аналізі медичних зображень, таких як рентгенівські знімки, МРТ і КТ. Ці технології дозволяють лікарям швидше та точніше виявляти хвороби, включаючи онкологічні захворювання. Наприклад, алгоритми автоматичного виявлення пухлин не лише пришвидшують процес діагностики, але й підвищують його точність, що особливо важливо в умовах дефіциту висококваліфікованих спеціалістів.

Безпека та спостереження також є важливою сферою застосування комп’ютерного зору. Інтелектуальні системи відеонагляду можуть автоматично розпізнавати обличчя, ідентифікувати підозрілі дії або предмети, підвищуючи рівень безпеки у громадських місцях. Ці технології також сприяють оптимізації

процесів у корпоративному секторі, наприклад, через автоматичне відстеження виробничих ліній чи обліку інвентаря.

У сільському господарстві використання БПЛА з функціями комп'ютерного зору дозволяє проводити моніторинг стану посівів, оцінювати рівень зволоження ґрунту або виявляти шкідників. Це зменшує витрати ресурсів та підвищує врожайність, що є критично важливим у контексті глобальних викликів продовольчої безпеки.

Комп'ютерний зір також стимулює розвиток інновацій у роздрібній торгівлі, де системи автоматичного розпізнавання товарів на полицях магазинів чи персоналізовані рекомендації покупцям створюють новий рівень обслуговування клієнтів. Завдяки цьому бізнес отримує змогу ефективніше управляти запасами та прогнозувати попит.

Попри значні досягнення, комп'ютерний зір стикається з низкою викликів. Одним з них є необхідність великих обсягів якісних даних для навчання моделей. В умовах реального світу часто виникають ситуації, коли зібраний набір даних може бути нерепрезентативним або недостатньо великим. Крім того, обчислювальні ресурси, необхідні для навчання глибоких нейронних мереж, залишаються доволі дорогими, що обмежує можливості для малих і середніх підприємств.

Етичні питання також заслуговують уваги. Використання технологій розпізнавання облич або поведінки викликає дискусії щодо приватності та прав людини. Розробка та впровадження подібних систем вимагає відповідального підходу, щоб уникнути зловживань і порушень прав на конфіденційність.

Таким чином, комп'ютерний зір відіграє ключову роль у трансформації сучасних технологій, створюючи нові можливості для автоматизації, аналізу та інновацій. Незважаючи на виклики, його потенціал у розвитку різних галузей лише зростає, роблячи його невід'ємною частиною майбутніх технологічних рішень.

1.2 Сучасні тенденції у розпізнаванні об'єктів

Сучасні тенденції у розпізнаванні об'єктів відображають стрімкий розвиток комп'ютерного зору, підсилений прогресом у глибинному навчанні, доступом до великих даних і вдосконаленням апаратного забезпечення. Ці тенденції формують нові підходи до вирішення задач розпізнавання, роблячи алгоритми швидшими, точнішими та універсальнішими.

Однією з ключових тенденцій є постійне вдосконалення архітектур нейронних мереж. Наприклад, моделі на кшталт YOLO (You Only Look Once) і SSD (Single Shot MultiBox Detector) забезпечують швидкість та ефективність, необхідну для роботи в реальному часі. Водночас двоетапні підходи, такі як Faster R-CNN, гарантують високу точність у складних задачах. Нові моделі, такі як Vision Transformers (ViT), пропонують принципово інший підхід, замінюючи конволюції механізмами самопильності, що забезпечує кращу продуктивність у деяких сценаріях.

Доступ до великих наборів даних, таких як COCO (Common Objects in Context) або Open Images, стимулює розвиток моделей, які здатні розпізнавати сотні об'єктів у складних умовах. Великі дані дозволяють тренувати моделі з кращою генералізацією, роблячи їх ефективними у реальних умовах. Крім того, використовуються синтетичні дані, створені за допомогою генеративних моделей або симуляторів, що дозволяє значно розширити можливості навчання.

Інша важлива тенденція — адаптація алгоритмів розпізнавання об'єктів для мобільних і вбудованих систем. Моделі, такі як MobileNet і EfficientDet, спеціально оптимізовані для роботи на пристроях з обмеженими ресурсами, забезпечуючи високий рівень продуктивності при низькому енергоспоживанні. Ці досягнення розширюють можливості використання комп'ютерного зору у портативних пристроях, від смартфонів до дронів.

Трансферне навчання стало незамінним інструментом у розробці моделей розпізнавання об'єктів. Завдяки попередньо натренованим моделям, таким як ResNet, Inception або EfficientNet, дослідники можуть швидко адаптувати

нейронні мережі до нових задач із мінімальними витратами ресурсів. Це особливо актуально у випадках, коли обсяг доступних даних обмежений.

Комп'ютерний зір все частіше інтегрується з іншими інноваціями, такими як Інтернет речей (IoT) та доповнена реальність (AR). Наприклад, у системах «розумного дому» розпізнавання об'єктів використовується для контролю безпеки та автоматизації процесів, тоді як в AR — для створення інтерактивних середовищ. Такі інтеграції відкривають нові перспективи у використанні комп'ютерного зору в повсякденному житті.

Сучасні тенденції не обходяться без викликів. Потреба у великих обчислювальних потужностях та якісних даних залишається важливою перешкодою. Крім того, питання приватності та етики все частіше піднімаються у зв'язку з використанням технологій розпізнавання облич і поведінки.

У майбутньому можна очікувати ще більшої автоматизації процесів, пов'язаних із розробкою моделей, наприклад, за допомогою AutoML. Також зростатиме популярність методів розпізнавання об'єктів, що поєднують традиційні підходи з глибинним навчанням, для вирішення вузькоспеціалізованих задач. Таким чином, комп'ютерний зір продовжуватиме відігравати ключову роль у розвитку сучасних технологій.

1.2.1 Вплив глибинного навчання на розвиток розпізнавання об'єктів.

Глибинне навчання зробило революційний внесок у розвиток комп'ютерного зору, зокрема у сферу розпізнавання об'єктів. Завдяки вдосконаленню архітектур нейронних мереж і збільшенню обчислювальної потужності, стало можливим вирішення складних задач, які раніше вважалися непридатними для автоматизації.

Історія розвитку алгоритмів розпізнавання об’єктів демонструє стрімке зростання їх ефективності. Ранні методи, такі як традиційні алгоритми комп'ютерного зору (наприклад, методи опорних векторів або HOG-фільтри), поступилися місцем глибоким конволюційним нейронним мережам (CNN). CNN

стали основою більшості сучасних систем розпізнавання об’єктів, забезпечуючи точність та масштабованість.

Поява моделей, таких як AlexNet, ResNet і Inception, започаткувала нову еру в комп'ютерному зорі. Ці архітектури продемонстрували можливість вирішення складних задач класифікації та локалізації об'єктів на зображеннях, відкриваючи нові горизонти для їх застосування. Наприклад, ResNet із використанням механізму залишкових зв'язків забезпечує глибокі моделі, які ефективно навчаються навіть у складних умовах.

Успіх глибинного навчання значною мірою залежить від якості та обсягу даних. Датасети, такі як ImageNet, MS COCO та Open Images, стали стандартом для тренування моделей розпізнавання об'єктів. Вони забезпечують різноманіття сценаріїв, що дозволяє мережам навчатися генералізації та ефективно розпізнавати об'єкти у реальних умовах.

Окрім реальних даних, широко використовуються синтетичні датасети, створені за допомогою генеративних алгоритмів або симуляторів. Вони дозволяють зменшити залежність від ручного маркування та значно розширюють можливості для експериментів із різними сценаріями.

Глибинне навчання не лише підвищило точність розпізнавання об'єктів, але й дозволило досягти оптимального балансу між точністю та швидкістю. Наприклад, архітектури YOLO і SSD спеціально розроблені для обробки зображень у реальному часі, що є критично важливим для застосувань у робототехніці, автономних транспортних засобах і системах безпеки.

Більш складні моделі, такі як Faster R-CNN, демонструють виняткову точність, але потребують значних обчислювальних ресурсів. Це стимулює дослідників шукати компроміс між продуктивністю та ефективністю, зокрема через впровадження технологій оптимізації, таких як квантовані моделі або зменшення розміру мереж.

Завдяки глибинному навчанню перспективи розвитку розпізнавання об’єктів виглядають багатообіцяючими. Зростає інтерес до комбінування традиційних алгоритмів комп’ютерного зору з глибинними підходами для

вирішення вузькоспеціалізованих задач. Іншим напрямом є розвиток самонавчальних моделей, які можуть адаптуватися до нових умов без необхідності постійного донавчання.

Висока продуктивність та універсальність алгоритмів глибинного навчання сприяють їх впровадженню у найрізноманітніші галузі, від медицини до промисловості. Таким чином, глибинне навчання залишається ключовим драйвером інновацій у розпізнаванні об'єктів і комп'ютерному зорі загалом.

1.2.2 Роль великих даних у вдосконаленні алгоритмів. Великі дані відіграють ключову роль у вдосконаленні алгоритмів розпізнавання об’єктів, забезпечуючи моделі значною кількістю якісної інформації для навчання. Вони стали базисом для створення та розвитку сучасних систем комп'ютерного зору, дозволяючи досягти значних успіхів у складних задачах розпізнавання.

Дані високої якості є основою ефективного навчання алгоритмів розпізнавання об’єктів. Великі датасети, такі як ImageNet, COCO та Open Images, забезпечують необхідне різноманіття і обсяг прикладів, які дозволяють моделям навчатися розпізнавати об’єкти в різних умовах. Наприклад, ImageNet став важливим етапом у розвитку глибокого навчання, адже на ньому були треновані багато популярних моделей, таких як AlexNet і ResNet.

Окрім реальних даних, все більшого значення набувають синтетичні датасети, створені за допомогою генеративних моделей або симуляторів. Цей підхід дозволяє моделювати специфічні сценарії, які важко отримати у реальному житті, наприклад, рідкісні події або складні умови зйомки. Також синтетичні дані використовуються для збагачення існуючих датасетів і усунення проблеми дисбалансу між класами об'єктів.

Хоча великі дані відкривають безліч можливостей, вони також створюють певні виклики. Серед них — потреба у значних обчислювальних ресурсах для обробки великих обсягів інформації, висока вартість розмітки та питання етики й конфіденційності. Наприклад, збір і використання даних, що включають особисту інформацію, може викликати занепокоєння у суспільстві.

Завдяки великим даним, сучасні алгоритми досягають високих показників точності і стійкості до шуму. Моделі, натреновані на великих датасетах, демонструють здатність до генералізації, тобто можливість успішно розпізнавати об’єкти навіть у непередбачуваних умовах. Ця властивість є критично важливою для реальних застосувань, таких як автономні автомобілі, медична діагностика або системи спостереження.

Майбутнє використання великих даних у розпізнаванні об’єктів включає автоматизацію процесів розмітки за допомогою штучного інтелекту, розвиток нових форматів представлення інформації та інтеграцію з іншими технологіями, такими як Інтернет речей (IoT). Таким чином, великі дані продовжуватимуть залишатися рушієм прогресу в розробці алгоритмів комп'ютерного зору.

1.3 Основи глибинного навчання

Наприкінці 1980-х років нейронні мережі стали важливою темою в області машинного навчання (ML) та штучного інтелекту (AI), завдяки появі ефективних методів навчання та нових архітектур мереж. Серед таких інновацій можна виділити багатошарові персептронні мережі, алгоритм зворотного поширення помилок, самоорганізовані карти та мережі з радіальними базисними функціями. Однак, незважаючи на значні досягнення, інтерес до нейронних мереж дещо знизився на певний період часу. Переломний момент настав у 2006 році, коли Хінтон і його колеги представили концепцію глибокого навчання (DL), засновану на принципах штучних нейронних мереж. Це стало поштовхом до нового етапу розвитку нейронних мереж, відомого як "нейронні мережі нового покоління". Глибокі мережі, за умови належного навчання, показали надзвичайні результати в задачах класифікації та регресії.

На сьогоднішній день глибоке навчання є однією з найбільш актуальних тем у сфері машинного навчання, штучного інтелекту, а також науки про дані та аналітики, завдяки своїй здатності ефективно працювати з великими обсягами даних. Компанії, як-от Google, Microsoft, Nokia, активно впроваджують цю

технологію, оскільки вона дає значні переваги у вирішенні різноманітних завдань класифікації та регресії. З точки зору категоризації, глибоке навчання є підмножиною машинного навчання та штучного інтелекту, і його можна розглядати як функцію AI, що імітує обробку даних людським мозком.

Популярність глибокого навчання зростає, що підтверджують статистичні дані з Google Trends, як зазначено в нашій попередній роботі. Глибоке навчання відрізняється від традиційних методів машинного навчання тим, що його ефективність суттєво зростає із збільшенням обсягу даних, про що йдеться в розділі "Чому глибоке навчання в сучасних дослідженнях і програмах". Технологія DL використовує кілька рівнів абстракцій даних для побудови обчислювальних моделей. Хоча навчання моделі вимагає значного часу через велику кількість параметрів, під час тестування вона може працювати швидше, ніж багато інших алгоритмів машинного навчання.

У контексті Четвертої промислової революції (4IR або Industry 4.0), яка акцентує увагу на автоматизації та розумних системах, технологія глибокого навчання, що базується на штучних нейронних мережах, стала основою для досягнення цих цілей. Типова нейронна мережа складається з безлічі простих обчислювальних одиниць або нейронів, які, з’єднуючись, створюють складніші структури. Кожен нейрон генерує вихідний сигнал, оброблюючи вхідні дані за допомогою ваг, зміщень і функцій активації. Як показано на рисунку 1, схема математичної моделі штучного нейрона включає вхідні дані (x), ваги (w), зміщення (b), функцію підсумовування (∑), функцію активації (f) та вихідний сигнал (y). Сьогодні глибоке навчання на основі нейронних мереж активно використовується в різних сферах, таких як охорона здоров’я, аналіз настроїв, обробка природної мови, візуальне розпізнавання, бізнес-аналітика, кібербезпека та інші, що детально розглядаються в останній частині цієї роботи.

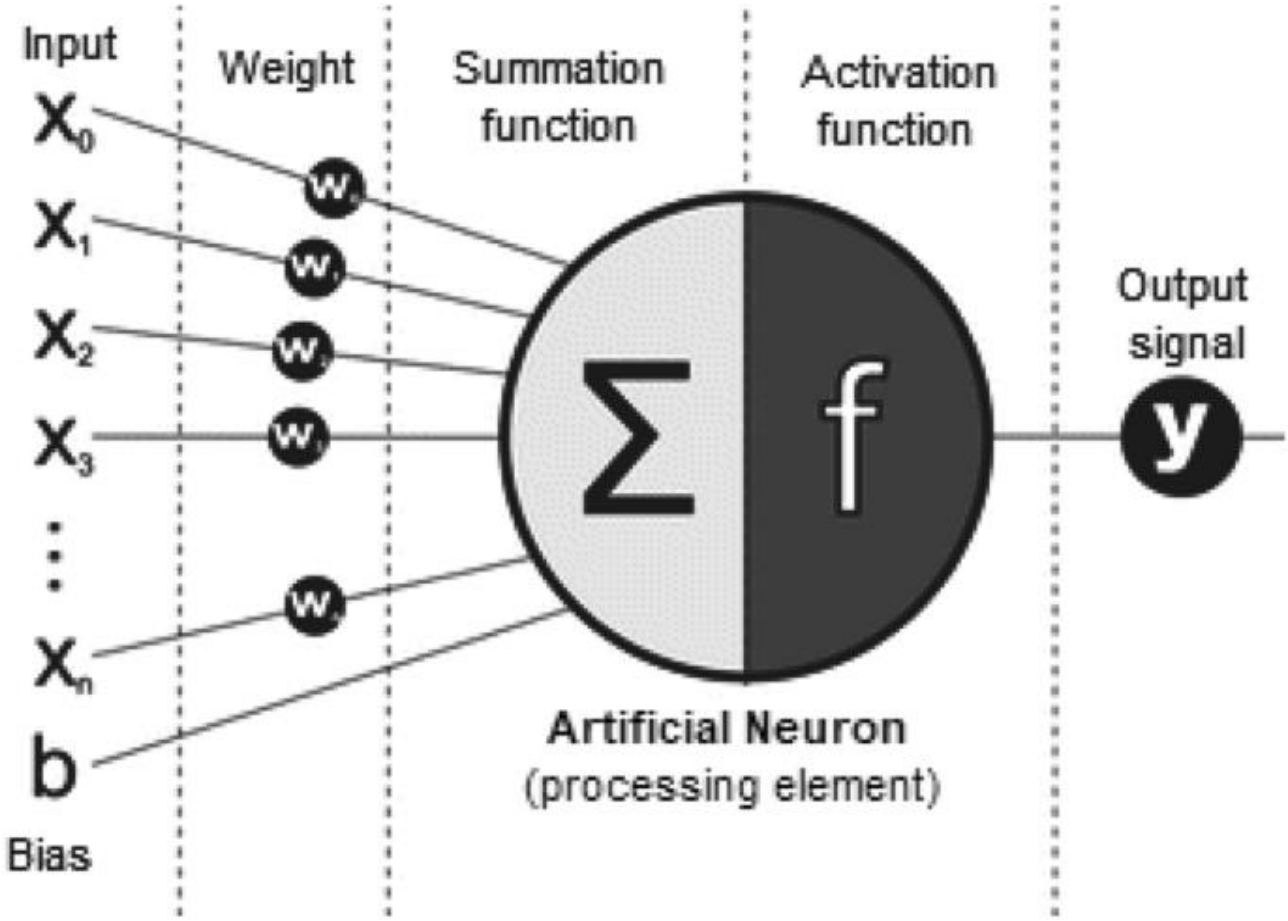


Рисунок 1.1 – Схематичне зображення математичної моделі штучного нейрона.

Сьогодні штучний інтелект (AI), машинне навчання (ML) та глибоке навчання (DL) є трьома популярними термінами, які часто використовуються як синоніми для опису систем або програм, що демонструють "розумну" поведінку. На рисунку 2 показано, як глибоке навчання співвідноситься з машинним навчанням та штучним інтелектом. Згідно з цією схемою, DL є частиною машинного навчання, а також є складовою більш широкої області штучного інтелекту. Загалом, штучний інтелект охоплює завдання, пов'язані з імітацією людської поведінки та інтелекту в машинах або системах, тоді як машинне навчання є методом, що дозволяє автоматизувати створення аналітичних моделей на основі даних або досвіду. Глибоке навчання також належить до методів навчання з використанням даних, але особливістю є використання багаторівневих нейронних мереж для обробки цих даних. Термін "глибоке" в контексті цієї методології означає наявність кількох етапів або рівнів, через які проходять дані для створення моделі, що керується даними

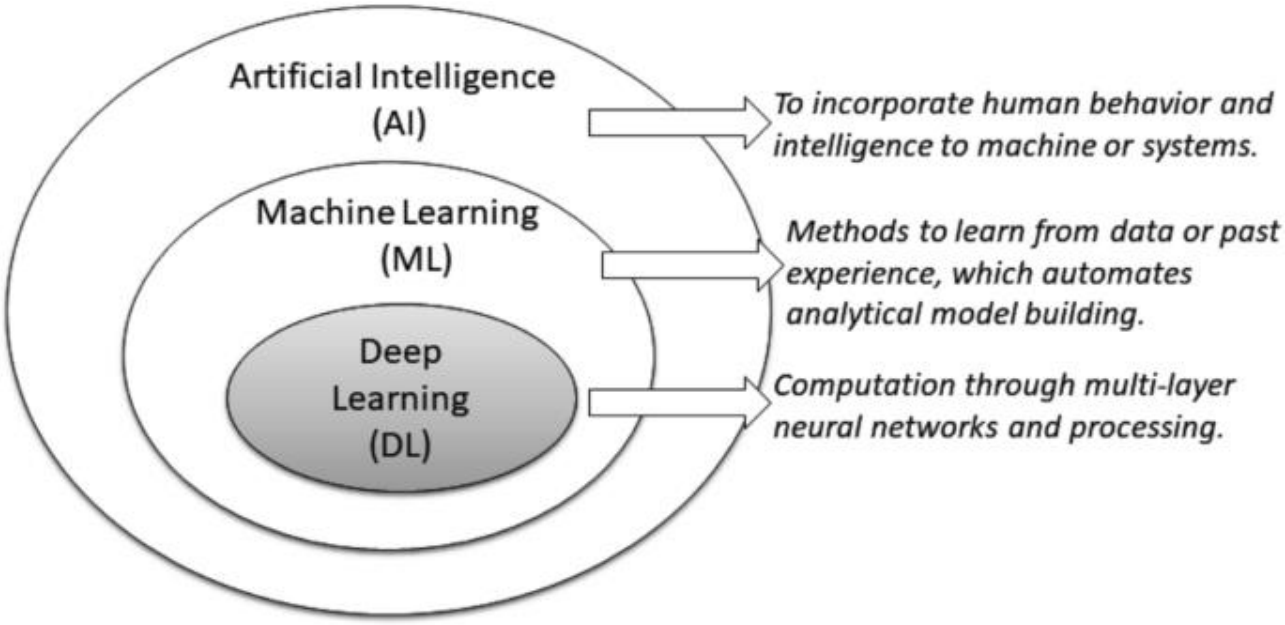


Рисунок 1.2 – Ілюстрація позиції глибокого навчання (DL) у порівнянні з машинним навчанням (ML) і штучним інтелектом (AI).

Таким чином, глибоке навчання можна розглядати як одну з основних технологій штучного інтелекту, що є передовою для розвитку інтелектуальних систем і автоматизації. Більш того, це дозволяє вивести штучний інтелект на новий рівень, який часто позначають як «Розумніший ШІ». Завдяки здатності глибоких мереж навчатися на основі даних, глибоке навчання тісно пов'язане з «наукою про дані». Наука про дані охоплює весь процес аналізу та пошуку значення в даних для вирішення специфічних проблем, і саме в цій сфері методи глибокого навчання можуть відігравати важливу роль у розширеній аналітиці та інтелектуальному прийнятті рішень. Отже, можна стверджувати, що технологія глибокого навчання має потенціал значно змінити сучасний світ, зокрема, через потужні обчислювальні можливості, сприяючи автоматизації та розвитку розумних і інтелектуальних систем, що відповідають цілям Індустрії 4.0.

Модель глибокого навчання зазвичай має такі ж етапи обробки, як і моделювання машинного навчання. На рисунку 4 зображено робочий процес глибокого навчання для вирішення реальних задач, який складається з трьох основних етапів: розуміння та попередня обробка даних, побудова та навчання DL-моделі, а також перевірка та інтерпретація результатів. Проте, на відміну від

машинного навчання, де виділення ознак часто є ручним процесом, в глибокому навчанні цей етап автоматизований. Серед методів машинного навчання, які зазвичай застосовуються в різних областях, можна назвати K-найближчий сусід, опорні векторні машини, дерева рішень, випадкові ліси, наївний Байєс, лінійну регресію, правила асоціації, методи кластеризації, такі як K-середні. З іншого боку, для побудови моделі глибокого навчання використовуються такі структури, як згорткові нейронні мережі, рекурентні нейронні мережі, автокодери, мережі глибоких переконань та інші. Далі буде обговорено основні властивості та залежності методів глибокого навчання, які необхідно враховувати при моделюванні для реальних додатків.

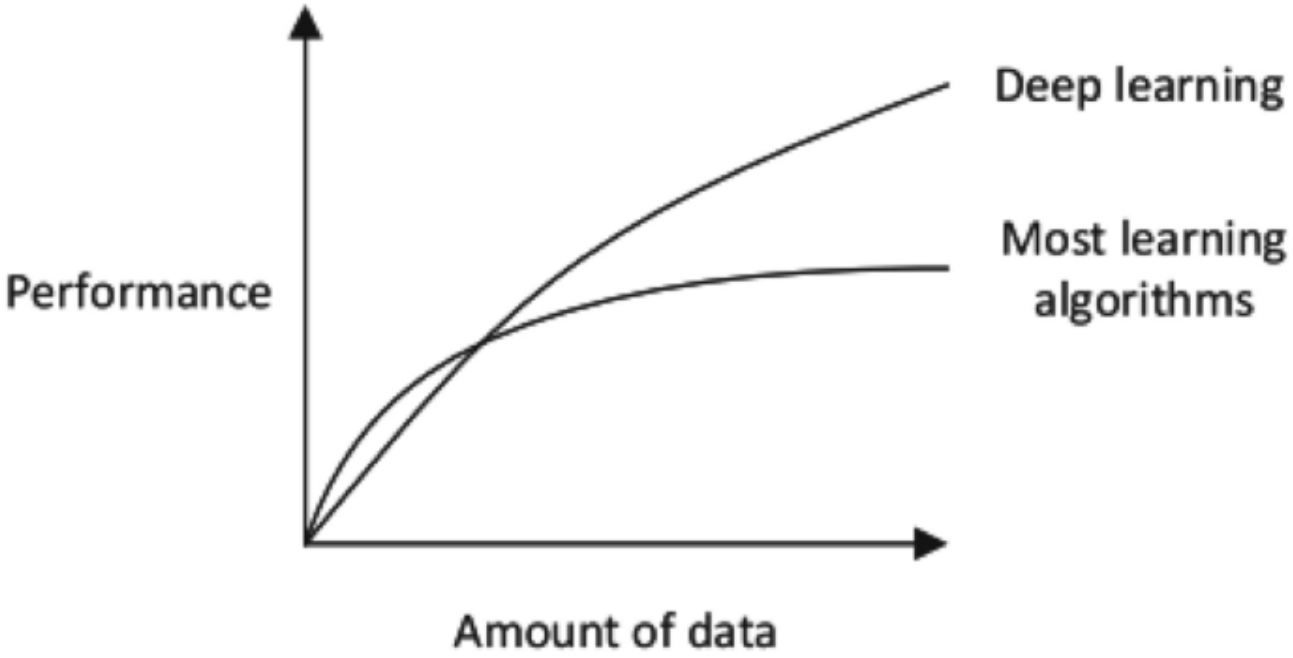


Рисунок 1.3 – Ілюстрація порівняння продуктивності між глибоким навчанням (DL) та іншими алгоритмами машинного навчання (ML), де моделювання DL на основі великих обсягів даних може підвищити продуктивність

Глибоке навчання зазвичай потребує великих обсягів даних для побудови моделей, орієнтованих на конкретні проблеми. Це зумовлено тим, що коли даних мало, алгоритми глибокого навчання часто не демонструють високої ефективності. У таких випадках традиційні алгоритми машинного навчання можуть показати кращі результати, якщо використовуються відповідні правила для обробки даних.

Алгоритми глибокого навчання вимагають значних обчислювальних потужностей під час навчання моделей на великих наборах даних. Оскільки більші обчислення потребують високої продуктивності, графічні процесори (GPU) значно перевершують центральні процесори (CPU) за ефективністю в оптимізації обчислювальних операцій. Таким чином, для ефективної роботи з глибоким навчанням необхідне обладнання на базі GPU, що робить DL більш залежним від високопродуктивних машин з GPU порівняно з традиційними методами машинного навчання.

Розробка функцій полягає у вилученні ознак (характеристик, властивостей, атрибутів) із необроблених даних за допомогою знань предметної області. Основна відмінність між глибоким навчанням та іншими методами машинного навчання полягає в тому, що DL здатне автоматично отримувати високорівневі ознаки без необхідності створення окремих екстракторів ознак для кожної проблеми. Це знижує час та зусилля, необхідні для підготовки даних.

Навчання моделей глибокого навчання займає значно більше часу через велику кількість параметрів, які потрібно оптимізувати. Наприклад, для моделей DL навчання може займати більше тижня, тоді як для алгоритмів машинного навчання цей процес може тривати лише від кількох секунд до кількох годин. Однак під час тестування алгоритми глибокого навчання, як правило, працюють швидше за традиційні методи машинного навчання.

Однією з основних проблем при використанні глибокого навчання є важкість інтерпретації результатів, що обумовлено природою алгоритмів "чорного ящика". Тобто важко пояснити, як саме було отримано результат. У свою чергу, багато методів машинного навчання, зокрема алгоритми на основі правил, надають чіткі логічні інтерпретації прийнятих рішень (наприклад, "ЯКЩО-ТОДІ"), що робить їх зрозумілими та простими для інтерпретації людьми.

Найбільша різниця між глибоким навчанням та стандартними методами машинного навчання полягає в тому, як ці алгоритми реагують на збільшення обсягу даних. Як показано на рисунку 3, продуктивність моделей глибокого

навчання зростає із збільшенням даних, тоді як стандартні алгоритми машинного навчання можуть досягти певної межі ефективності. Глибоке навчання є надзвичайно корисним при роботі з великими наборами даних, оскільки здатне обробляти величезну кількість ознак і створювати моделі, які краще управляються даними.

З точки зору розробки та навчання моделей, DL використовує розпаралелені матричні та тензорні операції, обчислювальні градієнти та оптимізацію. Для реалізації моделей глибокого навчання існують численні бібліотеки та ресурси, такі як PyTorch (з високорівневим API Lightning) і TensorFlow (що також включає Keras як високорівневий API). Ці інструменти надають всі необхідні функції для розробки та оптимізації моделей DL, а також містять багато попередньо навчених моделей, які можна використовувати для конкретних задач.

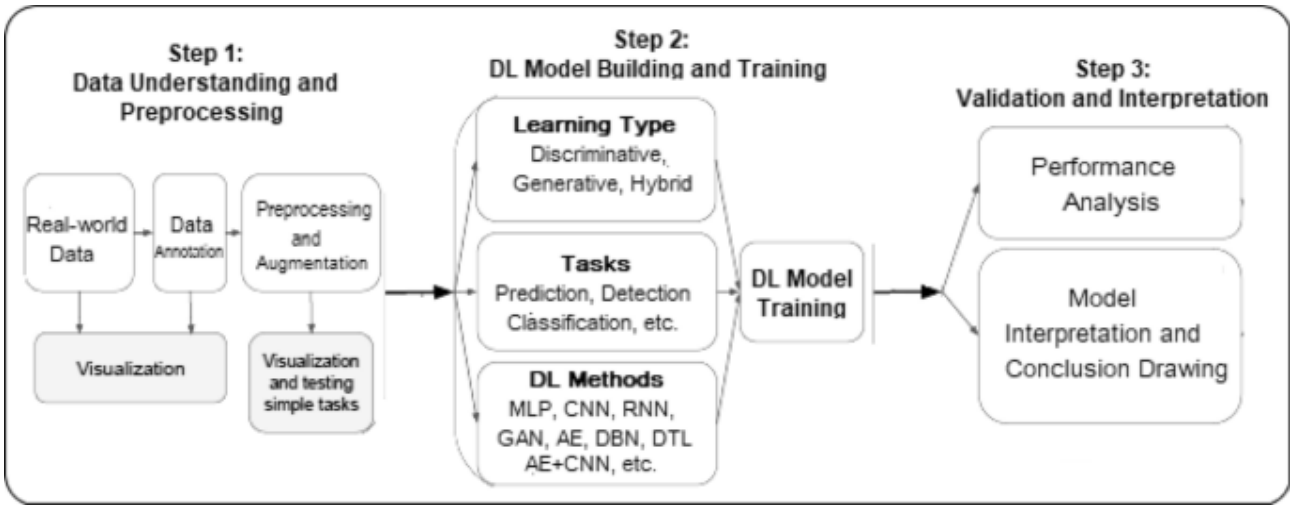


Рисунок 1.4 – Типовий робочий процес DL для вирішення проблем реального світу, який складається з трьох послідовних етапів (i) розуміння та попередня обробка даних (ii) Побудова моделі DL та навчання (iii) перевірка та інтерпретація

У цьому розділі ми розглядаємо різноманітні типи методів глибоких нейронних мереж, які використовують кілька рівнів обробки інформації в ієрархічних структурах для навчання. Типова глибока нейронна мережа складається з кількох прихованих рівнів, включаючи вхідний та вихідний рівні.

На рисунку 5 зображена загальна структура глибокої нейронної мережі порівняно з меншою мережею. У цьому контексті ми також представляємо таксономію методів глибокого навчання, що класифікується залежно від того, як вони застосовуються для вирішення різних проблем.

Проте перед тим, як детальніше розглянути методи глибокого навчання, корисно ознайомитися з різними типами навчальних завдань. Серед них:

- під контролем — підхід, що ґрунтується на завданнях, які використовують позначені (мітовані) навчальні дані;
- без контролю — метод, орієнтований на аналіз немаркованих даних, де модель виявляє структуру або шаблони без конкретних позначок;
- напівконтрольований — гібридний підхід, який поєднує методи з контрольованим і неконтрольованим навчанням;
- посилення — підхід, орієнтований на взаємодію з середовищем і вивчення політик для оптимального прийняття рішень.

З урахуванням цих типів завдань, ми поділяємо методи глибокого навчання на три основні категорії:

- глибокі мережі для контрольованого або дискримінаційного навчання — методи, які використовуються для задач, де модель навчається на позначених даних для класифікації або регресії;
- глибокі мережі для неконтрольованого або генеративного навчання — методи, що дозволяють моделі знаходити закономірності або створювати нові дані без попередніх міток;
- глибокі мережі для гібридного навчання, які поєднують елементи контрольованого та неконтрольованого навчання для вирішення складних завдань.

Далі ми коротко обговоримо кожен із цих методів і їх застосування для вирішення реальних задач у різних сферах, враховуючи їхні можливості навчання.

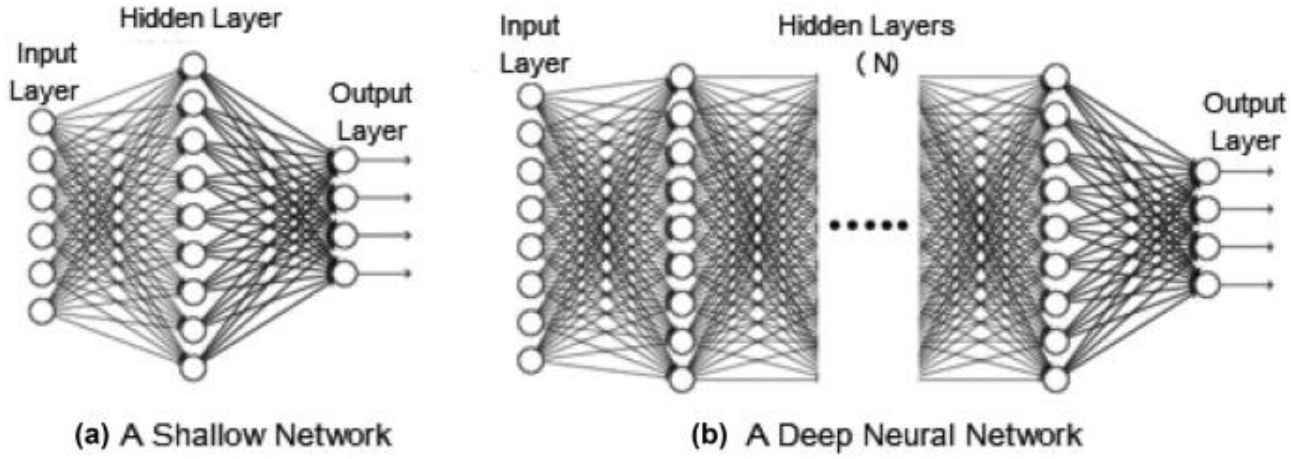


Рисунок 1.5 – Загальна архітектура дрібної мережі з одним прихованим шаром і b глибокої нейронної мережі з кількома прихованими шарами

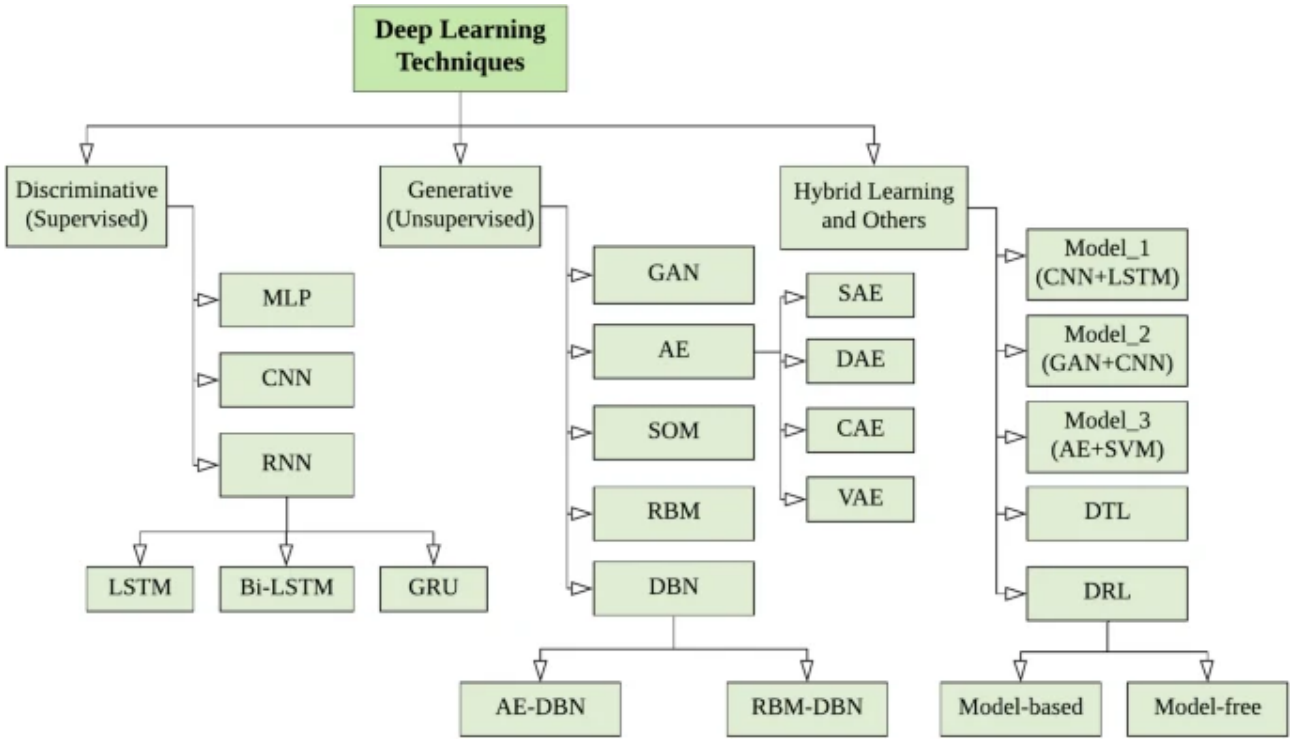


Рисунок 1.6 – Таксономія методів DL, загалом поділена на три основні категорії (i) глибокі мережі для контрольованого або дискримінаційного навчання, (ii) глибокі мережі для неконтрольованого або генеративного навчання та (ii) глибокі мережі для гібридного навчання та відповідні інші

1.4 Архітектури нейронних мереж для розпізнавання об'єктів

Архітектури нейронних мереж — це структури, що визначають, як організовані нейрони в різних шарах мережі та як вони з'єднані між собою. Вибір архітектури має суттєвий вплив на ефективність вирішення задачі, оскільки кожен тип архітектури оптимізований під певні особливості задачі, наприклад, для задач розпізнавання об'єктів важливі гнучкість у виділенні ознак та здатність до швидкої обробки великих обсягів даних.

Для задачі розпізнавання об'єктів, особливо в реальному часі, ключовими вимогами є:

- точність — здатність правильно класифікувати та локалізувати об'єкти;
- швидкість — здатність обробляти відеопотоки або великі набори зображень за мінімальний час;
- ефективність — оптимізація використання обчислювальних ресурсів, щоб забезпечити роботу мережі на мобільних пристроях чи вбудованих системах.

Основні підходи до архітектур нейронних мереж для розпізнавання об'єктів включають:

- згорткові нейронні мережі (CNN) — мережі, що використовують згорткові шари для вилучення просторових ознак зображень;
- системи для детекції об'єктів (Object Detection Networks) — включають додаткові алгоритми для локалізації об'єктів на зображеннях;
- сегментація зображень (Image Segmentation Networks) — відповідають за точне визначення контурів об'єктів та їх класів;
- легкі архітектури (Lightweight Architectures) — орієнтовані на оптимізацію обчислень і підвищення швидкості при збереженні точності.

Архітектури нейронних мереж:

- згорткові нейронні мережі (Convolutional Neural Networks, CNNs). Використовуються переважно для обробки зображень, відео та інших структурованих даних. Вони здатні автоматично виділяти важливі ознаки з зображень за допомогою згорткових шарів;

- рекурентні нейронні мережі (Recurrent Neural Networks, RNNs). Спеціалізуються на обробці послідовних даних, таких як текст, аудіо або часові ряди. Вони мають здатність зберігати інформацію про попередні елементи в послідовності завдяки внутрішнім зв'язкам;
- довготривала короткочасна пам'ять (Long Short-Term Memory, LSTM). Це спеціалізовані рекурентні мережі, що вирішують проблему забуття в стандартних RNN, зберігаючи важливу інформацію на довгий час. Вони часто використовуються для роботи з текстом, мовою, музикою тощо;
- генеративні змагальні мережі (Generative Adversarial Networks, GANs). Це пара нейронних мереж (генератор та дискримінатор), які тренуються в змагальному середовищі. Генератор намагається створювати реалістичні дані, а дискримінатор — відрізняти реальні дані від синтетичних. GANs активно використовуються для генерації зображень, відео, тексту;
- автокодери (Autoencoders). Це мережі, що складаються з двох частин: кодувальника та декодувальника. Вони використовуються для зменшення вимірності даних або для відновлення інформації з її стиснутого представлення. Автокодери також використовуються для генерації нових даних, виявлення аномалій тощо;
- трансформери (Transformers). Відмінно працюють для обробки послідовностей і є основою для таких архітектур, як GPT, BERT, T5. Вони використовують механізм уваги, що дозволяє моделі звертати увагу на різні частини вхідних даних незалежно від їхнього порядку;
- мережі з глибоким підкріпленням навчання (Deep Reinforcement Learning, DRL). Це метод, при якому агент навчається виконувати дії в середовищі, максимізуючи певну мету або винагороду. DRL поєднує глибоке навчання з підкріпленням, що дозволяє використовувати нейронні мережі для вирішення складних задач, таких як гра в ігри або управління роботами;
- мережі з обмеженими обчисленнями (Capsule Networks). Це більш новий підхід, який намагається вирішити проблеми традиційних згорткових мереж, такі як неможливість зберігати просторову ієрархію об'єктів. Capsule

Networks (CapsNet) є перспективною архітектурою для обробки складних зображень.

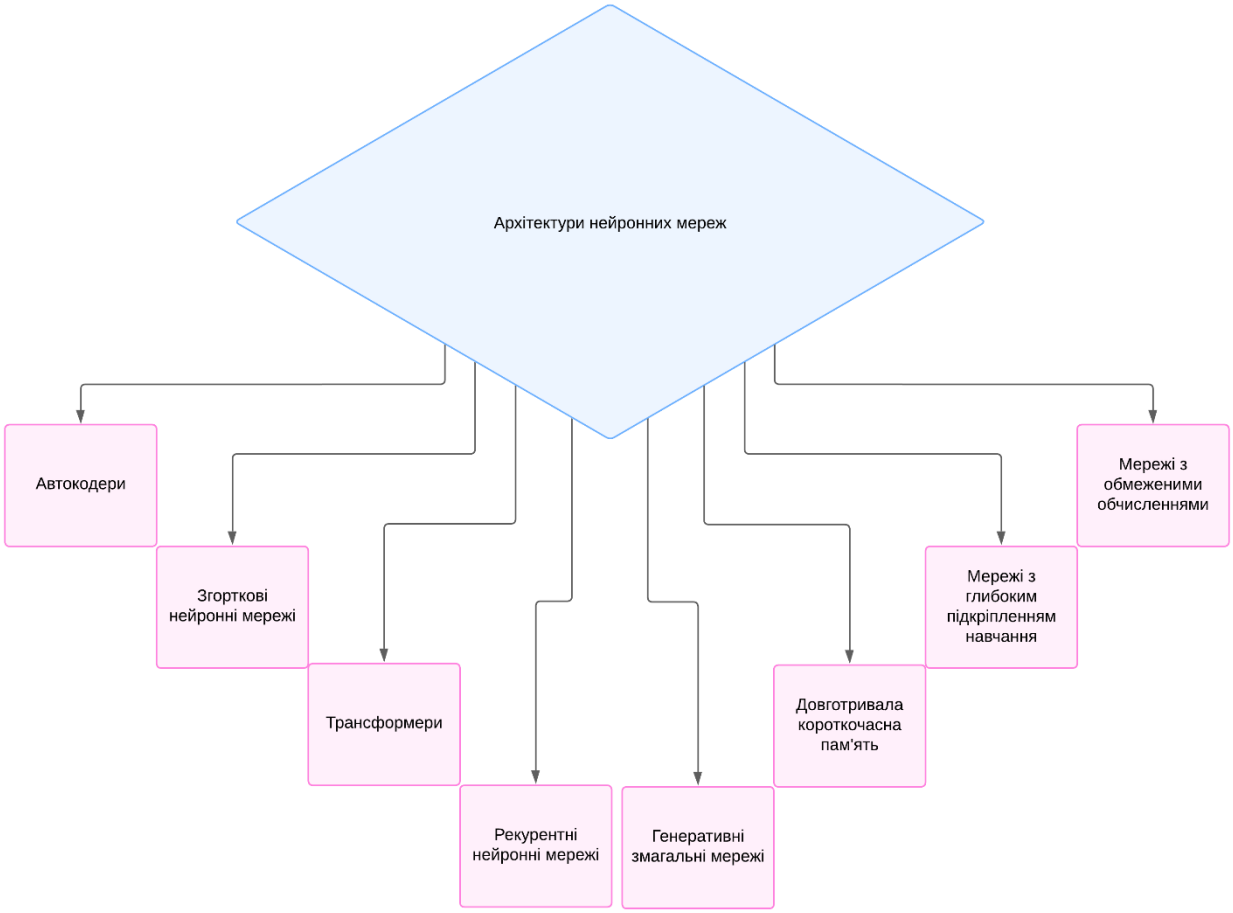


Рисунок 1.7 – схематичне зображення архітектури нейронних мереж

Детальніше що до особливостей архітектур буде описано в наступному розділі.

РОЗДІЛ 2.
АРХІТЕКТУРИ НЕЙРОНИХ МЕРЕЖ

2.1 Згорткові нейронні мережі (Convolutional Neural Networks, CNNs)

Згорткові нейронні мережі (CNN) — це тип нейронних мереж, спеціально розроблений для ефективної обробки та аналізу зображень, відео та інших даних з просторовою структурою. Основна ідея CNN полягає у використанні згорткових шарів, які автоматично виділяють важливі особливості (наприклад, краї, текстури, форми) з вхідних даних, що робить їх надзвичайно потужними для задач комп'ютерного зору. CNN мають ієрархічну структуру, де простіші патерни, знайдені на початкових шарах, комбінуються у більш складні особливості на глибших шарах.

CNN складаються з кількох основних компонентів:

1. Згортковий шар (Convolutional Layer).

- головний елемент CNN. Згортка (конволюція) — це математична операція, яка використовує невеликі фільтри (ядра) для сканування вхідного зображення;
- кожен фільтр налаштовується підчас навчання і вилучає певні фічі (особливості) з вхідних даних. Наприклад, це можуть бути краї або текстури на зображенні;
- результат операції згортки називається активаційною картою (feature map), яка містить інформацію про наявність певної особливості у відповідних ділянках вхідного зображення.

2. Функція активації.

- після згортки застосовується нелінійна функція (наприклад, ReLU — Rectified Linear Unit), яка допомагає моделі захоплювати складні залежності в даних. ReLU відкидає негативні значення, замінюючи їх нулями.

3. Шар підвибірки (Pooling Layer).

- використовується для зменшення розміру активаційної карти, зменшуючи обчислювальну складність і підвищуючи стійкість до зсувів та шуму;
- найпоширеніший тип — Max Pooling, який бере максимальне значення в кожному фрагменті карти.

4. Шари згорток і підвибірки повторюються.

- у складних моделях може бути багато таких шарів, де кожен наступний шар працює з активаційними картами попереднього, вивчаючи дедалі складніші особливості.

5. Повнозв'язний шар (Fully Connected Layer).

- на останньому етапі активаційні карти “розгортаються” у вектор, який подається на вхід одному або декільком повнозв'язним шарам;
- ці шари використовуються для класифікації або іншого завдання.

6. Функція втрат (Loss Function).

- визначає, наскільки добре модель виконує своє завдання (наприклад, класифікацію);
- наприклад, для задач класифікації часто використовують крос-ентропію.

Навчання CNN полягає у налаштуванні ваг (параметрів фільтрів) за допомогою алгоритму зворотного поширення помилки (backpropagation). Головні етапи:

1. Вхідні дані проходять через всі шари CNN (прямий прохід).
2. Обчислюється функція втрат.
3. За допомогою зворотного проходу обчислюються градієнти функції втрат за вагами фільтрів.
4. Ваги оновлюються методом градієнтного спуску або його варіацій.

Переваги CNN.

- автоматичне вилучення особливостей: Не потребують ручного створення фіч;

- зменшення кількості параметрів: Завдяки використанню фільтрів і підвибірки;
- висока ефективність у задачах комп'ютерного зору: Підходять для класифікації, детекції об'єктів, сегментації зображень тощо.

2.2 Рекурентні нейронні мережі (Recurrent Neural Networks, RNN)

Рекурентні нейронні мережі (RNN) — це тип нейронних мереж, які особливо ефективні для обробки послідовних даних, таких як текст, аудіо, часові ряди та відео. Їх головна особливість полягає у здатності зберігати інформацію про попередні елементи послідовності через внутрішні стани, що дозволяє моделі враховувати контекст попередніх даних під час прогнозування поточного або майбутнього значення.

RNN складаються з повторюваних блоків, які приймають як вхід поточний елемент послідовності, так і стан, отриманий з попереднього кроку. Це дозволяє моделі враховувати історію послідовності.

1. Вхідні дані.

- кожен елемент послідовності подається на вхід RNN один за одним. Наприклад, у випадку тексту це можуть бути окремі слова або символи.

2. Прихований стан (Hidden State).

- на кожному кроці обчислюється новий прихований стан на основі поточного входу та попереднього прихованого стану.

3. Вихідний шар.

- на кожному кроці мережа може видавати вихід, який залежить від поточного прихованого стану. Це корисно для задач, де необхідно мати проміжні результати, наприклад, у машинному перекладі.

4. Навчання.

- використовується алгоритм зворотного поширення помилки через час (Backpropagation Through Time, BPTT), який враховує залежності між всіма елементами послідовності.

Проблеми традиційних RNN:

1. Згасаючі та вибухаючі градієнти: При обробці довгих послідовностей градієнти можуть ставати дуже малими або великими, що ускладнює навчання.

2. Обмежена здатність до збереження довготривалої інформації: Звичайні RNN краще працюють з короткими залежностями.

Щоб подолати обмеження RNN, було розроблено спеціальні варіанти:

- LSTM (Long Short-Term Memory): Мають механізми забування, запам'ятовування і вибору важливої інформації через спеціальні "гейти".

- GRU (Gated Recurrent Unit): Спрощена версія LSTM із меншим числом параметрів.

Застосування RNN:

- обробка природної мови (переклад, генерація тексту, аналіз тональності);
- прогнозування часових рядів (фінансові дані, погода);
- розпізнавання мовлення та синтез голосу;
- аналіз відео та послідовних зображень

2.3 Довготривала короткочасна пам'ять (Long Short-Term Memory, LSTM)

Довготривала короткочасна пам'ять (LSTM) — це спеціальний тип рекурентної нейронної мережі (RNN), розроблений для подолання проблеми згасаючих і вибухаючих градієнтів у традиційних RNN. LSTM особливо ефективна при обробці довгих послідовностей даних, оскільки має механізми для запам'ятовування важливої інформації та забування нерелевантної. Це досягається завдяки спеціальній структурі блоку LSTM, який включає кілька "гейтів" для керування потоком інформації.

Блок LSTM складається з таких основних компонентів:

1. Вхідний гейт (Input Gate).

- вирішує, яку частину нової інформації додати до стану пам'яті;

- для цього використовуються функції активації (для ймовірностей та для масштабування).

2. Гейт забування (Forget Gate).

- визначає, яку частину попереднього стану пам'яті потрібно "забути" або видалити.

3. Гейт виходу (Output Gate).

- вирішує, яку частину пам'яті включити до вихідного прихованого стану.

4. Стан пам'яті (Cell State).

- головний компонент, який зберігає довготривалу інформацію;

- оновлюється за допомогою комбінації гейта забування та вхідного гейта

5. Прихований стан (Hidden State).

- оновлюється на основі стану пам'яті та вихідного гейта.

Переваги LSTM

- здатність зберігати довготривалі залежності: Завдяки механізму гейтів;

- гнучкість: Ефективно працює як з короткими, так і довгими послідовностями;

- стійкість до проблеми згасаючих градієнтів: Забезпечується структурою стану пам'яті.

Застосування LSTM

- обробка природної мови: Машинний переклад, аналіз тональності, генерація тексту;

- прогнозування часових рядів: Фінансові дані, прогноз погоди, моделювання попиту;

- розпізнавання мовлення: Перетворення мови в текст, синтез голосу;

- аналіз відео: розпізнавання дій, аналіз відеопотоку.

2.4 Генеративні змагальні мережі (Generative Adversarial Networks, GAN)

Генеративні змагальні мережі (GAN) — це тип нейронних мереж, які використовуються для генерації нових даних, що виглядають подібно до наявних. GAN складаються з двох моделей, що змагаються між собою: генератора та дискримінатора. Генератор створює нові дані, а дискримінатор оцінює їх і визначає, чи є вони справжніми (з оригінальних даних) або підробленими (згенерованими). Цей процес сприяє тому, що генератор з часом покращується і навчається створювати більш реалістичні дані.

Структура GAN:

- Генератор (Generator) – приймає на вхід випадковий шум (зазвичай випадковий вектор з нормального розподілу) і генерує дані, які повинні виглядати схожими на справжні. Його завдання — обманути дискримінатор, змусивши його прийняти згенеровані дані за справжні;

- Дискримінатор (Discriminator) - приймає як справжні дані, так і згенеровані, і класифікує їх як "справжні" або "підроблені". Його завдання — виявити, які дані є підробленими.

Навчання GAN:

Навчання здійснюється у два етапи:

1. Генератор генерує дані, які подаються на вхід дискримінатору разом із справжніми даними. Дискримінатор оновлює свої параметри для поліпшення здатності розрізняти справжні та підроблені дані.

2. Генератор оновлює свої параметри, щоб зменшити помилки дискримінатора, змушуючи його приймати згенеровані дані за справжні.

Мета генератора — мінімізувати помилку дискримінатора, тоді як мета дискримінатора — максимізувати свою точність. Цей процес описується як "мінімаксна гра":

Циклічний процес:

- генератор і дискримінатор покращуються поступово: генератор стає кращим у створенні реалістичних даних, а дискримінатор стає кращим у виявленні підробок.

Переваги GAN:

- реалістична генерація - GAN можуть створювати високоякісні зображення, відео, тексти та інші дані;

- гнучкість - можуть бути адаптовані до різних типів даних і задач;

- відсутність потреби у явному моделюванні розподілу даних - генератор навчається від дискримінатора без прямого доступу до справжнього розподілу даних.

Обмеження GAN:

- нестабільність навчання: Через складну динаміку між генератором і дискримінатором навчання може бути нестійким;

- проблема модального колапсу: Генератор може навчитися створювати лише обмежену кількість зразків, ігноруючи різноманіття даних.

Застосування GAN:

-генерація зображень: Створення реалістичних облич, змінювання стилю зображень (наприклад, перетворення фотографій у картини);

- обробка відео: Генерація відео або покращення їхньої якості;

- обробка тексту: Генерація текстів або створення нових стилів письма;

- медицина: Генерація медичних зображень для збільшення навчальних наборів даних;

- ігрова індустрія: Створення нових персонажів, рівнів або текстур.

GAN є потужним інструментом для генеративного моделювання, який продовжує знаходити нові застосування в різних галузях науки і техніки.

2.5 Автокодери (Autoencoders)

Автокодери — це нейронні мережі, які використовуються для навчання стислого (компресованого) представлення даних. Вони навчаються відновлювати свої вхідні дані, проходячи через вузьке "пляшкове горлечко", яке змушує модель зберігати найважливішу інформацію. Завдяки цьому автокодери використовуються для задач зменшення розмірності, виявлення аномалій та генерації даних.

Автокодери складаються з двох основних частин:

1. Енкодер (Encoder).

- зменшує розмірність вхідних даних, стискаючи їх у компактне латентне представлення .

2. Декодер (Decoder).

- відновлює дані з латентного представлення , намагаючись відтворити якнайближче до оригіналу.

Автокодери мають симетричну структуру:

- вхідний шар — приймає дані ;
- приховані шари енкодера — поступово зменшують розмірність даних;
- пляшкове горлечко — шар із найменшою розмірністю, який представляє латентний вектор ;
- приховані шари декодера — поступово збільшують розмірність даних до вихідного рівня;
- вихідний шар — відновлює дані .

Автокодери навчаються шляхом мінімізації функції втрат, яка обчислює різницю між вхідними та відновленими даними. Типова функція втрат

Типи автокодерів:

1. Недосконалі автокодери (Undercomplete Autoencoders).

- мають вузьке пляшкове горлечко, що змушує модель вивчати найважливіші ознаки даних.

2. Шумозахищені автокодери (Denoising Autoencoders).

- навчаються відновлювати оригінальні дані з пошкоджених входів, додаючи шум до перед обробкою енкодером.

3. Варіаційні автокодери (Variational Autoencoders, VAE).

- генерують нові дані, додаючи стохастичність у латентний простір. Використовуються в задачах генерації.

4. Складні автокодери (Sparse Autoencoders).

- використовують регуляризацію для створення розрідженого латентного представлення, що дозволяє виявляти рідкісні особливості даних.

Застосування автокодерів:

- зменшення розмірності: Компресія даних для зберігання або швидшої обробки;

- виявлення аномалій: Аналіз латентного представлення для виявлення нетипових патернів;

- генерація даних: Створення нових зображень, текстів або інших даних;

- реконструкція пошкоджених даних: Відновлення пошкоджених зображень або сигналів;

- препроцесинг даних: Виділення ключових ознак перед подальшим аналізом.

Автокодери є універсальними інструментами в області машинного навчання, які ефективно працюють з великими обсягами даних, забезпечуючи їх компактне представлення і обробку.

2.6 Трансформери (Transformers)

Трансформери — це архітектура нейронних мереж, яка була вперше запропонована для обробки послідовностей даних, таких як текст чи часові ряди. Основною ідеєю трансформерів є механізм самоуваги (self-attention), який дозволяє моделі визначати залежності між елементами послідовності незалежно від їхньої віддаленості один від одного. Це робить трансформери надзвичайно ефективними для задач природної обробки мови (NLP), машинного перекладу, генерації текстів та інших задач.

Принцип роботи трансформерів:

1. Архітектура трансформерів Трансформери складаються з двох основних компонентів.

- енкодер (Encoder): Обробляє вхідну послідовність та створює контекстуальне представлення для кожного елемента;

- декодер (Decoder): Використовує вихід енкодера для генерації цільової послідовності, наприклад, перекладеного тексту.

2. Механізм самоуваги.

- кожен елемент послідовності обчислює "увагу" до інших елементів, визначаючи, які з них важливі для поточного елемента;

- для цього обчислюються три матриці: запити (Query,), ключі (Key,) та значення (Value,).

3. Мультиголовна самоувага (Multi-head Attention).

- для покращення здатності моделі до вивчення різних аспектів залежностей між елементами використовуються кілька "голів" уваги. Це дозволяє моделі фокусуватися на різних частинах послідовності одночасно.

4. Мережа з повним зв'язком (Feedforward Neural Network).

- після механізму самоуваги кожен елемент послідовності обробляється через повнозв'язкову нейронну мережу, яка дозволяє моделі вивчати нелінійні залежності.

5. Нормалізація та залишкові зв'язки.

- для покращення стабільності навчання використовуються шарова нормалізація (Layer Normalization) та залишкові зв'язки (Residual Connections), що забезпечують кращу передачу градієнтів.

Трансформери навчаються за допомогою функції втрат, яка залежить від задачі:

- для задач машинного перекладу або генерації тексту використовується крос-ентропія між передбаченими та справжніми виходами;

- для задач класифікації тексту може використовуватись бінарна або категорійна крос-ентропія.

Переваги трансформерів:

- глобальна увага: Можуть ефективно знаходити залежності між будь-якими елементами послідовності;

- паралельність: На відміну від рекурентних нейронних мереж (RNN), трансформери дозволяють обробляти всі елементи послідовності одночасно, що значно пришвидшує навчання;

- масштабованість: Можуть бути адаптовані для роботи з дуже великими наборами даних.

Обмеження трансформерів:

- високі обчислювальні витрати: Через механізм самоуваги обчислювальна складність трансформерів зростає квадратично із збільшенням довжини послідовності;

- потреба у великих обсягах даних: Для досягнення високої точності трансформери потребують великих обсягів даних для навчання.

Застосування трансформерів:

- обробка природної мови (NLP):Машинний переклад, генерація текстів, відповіді на запити, аналіз почуттів;

- комп'ютерний зір: Розпізнавання зображень, обробка відео;

- генеративні моделі: Моделі GPT для текстів, моделі DALL-E для генерації зображень;

- обробка часових рядів: Прогнозування та аналіз трендів у фінансах, медицині та інших галузях.

Трансформери стали основою сучасних досягнень у галузі штучного інтелекту, демонструючи високу ефективність у задачах, які потребують роботи з послідовностями.

2.7 Мережі з глибоким підкріпленням навчання (Deep Reinforcement Learning Networks)

Мережі з глибоким підкріпленням навчання (Deep Reinforcement Learning, DRL) поєднують глибокі нейронні мережі з методами підкріплювального навчання. Цей підхід дозволяє агенту вчитися оптимальній поведінці шляхом взаємодії з середовищем, отримуючи винагороду за правильні дії і штраф за

неправильні. DRL використовується в задачах, де потрібно приймати послідовні рішення, наприклад, в іграх, робототехніці, управлінні ресурсами та фінансовому аналізі.

Принцип роботи DRL базується на кількох основних компонентах: агент, який приймає рішення; середовище, з яким взаємодіє агент; стан, що описує поточний стан середовища; дії, які агент може виконувати; винагорода, як зворотній зв'язок за дії; політика, що визначає стратегію вибору дій; і функція цінності, яка оцінює довгострокову вигоду перебування в певному стані.

Глибокі нейронні мережі широко застосовуються в DRL для апроксимації політики чи функції цінності. Наприклад, DQN (Deep Q-Network) оцінює вигоду виконання дії в конкретному стані. Політико-орієнтовані методи використовують нейронну мережу для моделювання політики, тоді як методи актор-критик поєднують функцію політики (актор) та функцію цінності (критик).

Процес навчання у DRL включає взаємодію агента із середовищем: спостереження стану, вибір дії відповідно до поточної політики, отримання винагороди та оновлення стану, а також корекцію параметрів нейронної мережі для вдосконалення політики або функції цінності.

Серед переваг DRL виділяються адаптивність до складних динамічних середовищ, генералізація через використання глибоких мереж для вивчення залежностей у даних, а також автоматичне навчання без ручного налаштування правил.

Однак є й обмеження, зокрема високі обчислювальні витрати, вразливість до перенавчання і потреба у великих обсягах даних для ефективного навчання.

Застосування DRL охоплює різні галузі, включаючи ігри (досягнення суперлюдського рівня у складних іграх, як-от шахи чи Go), робототехніку (навчання виконанню складних завдань, наприклад, ходьби або маніпуляції об'єктами), фінанси (оптимізація торгових стратегій), енергетику (управління розподілом ресурсів у складних системах) і медицину (розробка оптимальних стратегій лікування).

Мережі з глибоким підкріпленням навчання демонструють великий потенціал для автоматизації складних задач, показуючи високу ефективність у багатьох галузях.

2.8 Мережі з обмеженими обчисленнями (Restricted Computational Networks)

Мережі з обмеженими обчисленнями призначені для виконання обчислень з мінімальними ресурсами, такими як пам’ять або обчислювальна потужність. Вони використовуються в умовах, де доступ до апаратного забезпечення або енергії обмежений, наприклад, на вбудованих системах, IoT-пристроях чи мобільних телефонах. Основна ідея таких мереж полягає в оптимізації структури та процесу навчання для досягнення максимальної ефективності при обмежених ресурсах.

Принцип роботи мереж з обмеженими обчисленнями:

1. Зменшення складності моделі.

- зрідження моделі (Pruning): Видаляються маловажливі нейрони або з’єднання у мережі без значної втрати точності;

- квантизація: Представлення ваг і активацій із меншою точністю (наприклад, 8-бітні замість 32-бітних);

- поділ мережі на шари: Застосування невеликої кількості шарів або використання менш складних архітектур.

2. Ефективна обробка даних.

- зменшення розміру вхідних даних: Використання технік попередньої обробки для скорочення розміру даних без втрати значущої інформації;

- адаптивна обробка: Виконання обчислень тільки там, де це необхідно, наприклад, за допомогою технік ранньої зупинки (early stopping).

3. Перенесене навчання (Transfer Learning).

- використання попередньо навчених моделей та адаптація їх до нових задач з мінімальними витратами ресурсів.

4. Оптимізовані архітектури.

- мережі MobileNet: Використовують глибокі згорткові шари з розділенням за каналами (depthwise separable convolutions);

- squeezeNet: Зменшує кількість параметрів через використання компактних фільтрів у згорткових шарах:

- tinyML: Архітектури для навчання та інференсу на мікроконтролерах і інших пристроях з обмеженими ресурсами.

5. Хмарні обчислення.

- частину складних обчислень можна переносити на хмарні сервери, зменшуючи навантаження на локальні пристрої.

Навчання мереж з обмеженими обчисленнями:

- використання малих пакетів (mini-batches) для ефективного використання пам’яті;

- навчання на спеціалізованих наборах даних для досягнення високої точності при меншій складності моделі;

- адаптивна оптимізація, така як Adam або RMSProp, для швидшого збіжності при навчанні.

Переваги мереж з обмеженими обчисленнями:

-енергоефективність: Підходять для пристроїв із низьким енергоспоживанням;

- компактність: Малі моделі можна легко розгортати на обмеженому апаратному забезпеченні;

- швидкість: Завдяки оптимізації моделі зменшуються затримки під час виконання.

Обмеження мереж з обмеженими обчисленнями:

- зниження точності: Оптимізація може впливати на якість результатів;

-складність проектування: Необхідність ретельного налаштування архітектури та параметрів для кожного застосування;

- обмеженість задач: Підходять не для всіх типів складних обчислень.

Застосування мереж з обмеженими обчисленнями:

- інтернет речей (IoT): Аналіз даних із сенсорів та автоматизація;

- мобільні додатки: Обробка зображень, розпізнавання мови та інших завдань на смартфонах;

- робототехніка: Реалізація керуючих алгоритмів на компактних роботах;

- медичні пристрої: Моніторинг стану пацієнтів у режимі реального часу;

- автомобільна промисловість: Розпізнавання об’єктів у системах допомоги водіям.

Мережі з обмеженими обчисленнями забезпечують ефективне використання ресурсів, що робить їх ключовим інструментом у сучасних технологіях, особливо в умовах обмеженого обладнання.

2.9 Висновки

Більшість з перелічених підвидів глибинного навчання можна використовувати для розпізнавання об'єктів, хоча деякі з них є більш спеціалізованими та ефективними для конкретних типів задач. Ось як кожен з них може бути застосований до задачі розпізнавання об'єктів:

1. Згорткові нейронні мережі (CNN).

Це найпопулярніший підхід для задач розпізнавання об'єктів, зокрема для обробки зображень. Завдяки своїй здатності автоматично виділяти ознаки з зображень (наприклад, краї, текстури, форми), CNN є основою для багатьох сучасних систем комп'ютерного зору, таких як детекція об'єктів, сегментація та класифікація.

2. Рекурентні нейронні мережі (RNN).

Рекурентні мережі не є основним вибором для розпізнавання статичних об'єктів на зображеннях, оскільки вони краще підходять для обробки послідовних даних, таких як текст або відео. Однак, якщо задача передбачає аналіз відео (наприклад, розпізнавання рухомих об'єктів чи відстеження), то

RNN або їх варіанти, як LSTM, можуть бути корисні для обробки часової інформації.

3. Довготривала короткочасна пам'ять (LSTM).

LSTM можна використовувати для задач, де потрібно обробляти послідовність кадрів (наприклад, в відео або з серії зображень). Вони здатні зберігати інформацію про попередні кадри, що дозволяє відстежувати рухи об'єктів або їхню динаміку в часі.

4. Генеративні змагальні мережі (GANs).

GANs зазвичай не використовуються безпосередньо для стандартного розпізнавання об'єктів, але вони можуть бути корисні для генерації нових зображень або для доповнення даних (data augmentation), що в свою чергу покращує результати в задачах класифікації чи детекції об'єктів. GANs також можуть застосовуватися для поліпшення якості зображень або відновлення частково пошкоджених зображень.

5. Автокодери (Autoencoders).

Автокодери в основному використовуються для стиснення та відновлення зображень або для виявлення аномалій, але їх також можна використовувати для попередньої обробки даних перед використанням згорткових мереж. Вони можуть бути корисними для задач зменшення вимірності в попередньому етапі, що дозволить зменшити обсяг даних перед їх подальшою обробкою.

6. Трансформери (Transformers).

Трансформери, завдяки своїй здатності обробляти залежності на різних рівнях та механізму уваги, набули популярності для задач розпізнавання об'єктів в зображеннях (наприклад, Vision Transformer, ViT). Вони також можуть бути використані для задач обробки відео, де важливо зберігати контекст і взаємозв'язок між кадрами.

7. Мережі з глибоким підкріпленням навчання (DRL).

Мережі з підкріпленням навчання зазвичай не використовуються для розпізнавання об'єктів безпосередньо. Однак вони можуть бути застосовані для

задач, де агент має навчитися взаємодіяти з середовищем на основі спостережень (наприклад, для роботів, які повинні розпізнавати і маніпулювати об'єктами в реальному середовищі).

8. Мережі з обмеженими обчисленнями (Capsule Networks).

Capsule Networks мають потенціал для покращення точності розпізнавання об'єктів, зокрема завдяки їхній здатності зберігати просторові відносини між частинами об'єкта. Цей підхід може бути корисним для задач, де потрібно враховувати орієнтацію та взаємодію об'єктів на зображенні.

Висновок:

- CNN є основним і найбільш використовуваним підходом для розпізнавання об'єктів.

- LSTM та RNN можуть бути корисні для аналізу динамічних або часових аспектів (наприклад, у відео).

- GANs більше використовуються для генерації даних, але можуть допомогти у поліпшенні якості зображень.

- Трансформери також є потужним інструментом для розпізнавання об'єктів, зокрема у сучасних моделях.

- Автокодери здебільшого використовуються для стиснення даних, але можуть бути корисними для попередньої обробки.

- DRL в основному застосовується для задач, де важлива взаємодія з середовищем.

РОЗДІЛ 3.
ТЕОРЕТИЧНІ АСПЕКТИ РЕАЛІЗАЦІЇ СИСТЕМ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

3.1 Основи глибинного навчання в розпізнаванні об'єктів

Розпізнавання об'єктів є однією з ключових задач в галузі комп'ютерного зору, яка включає в себе ідентифікацію та класифікацію об'єктів на зображеннях або у відеопотоці. Сучасні підходи до розпізнавання об'єктів значною мірою базуються на методах глибинного навчання, які використовують багатошарові нейронні мережі для автоматичного виділення ознак зображень та їх класифікації.

Глибинне навчання, яке є підмножиною машинного навчання, здатне обробляти складні дані та виявляти приховані патерни, використовуючи великі обсяги інформації. Одним з основних досягнень глибинного навчання є застосування згорткових нейронних мереж (CNN), які продемонстрували надзвичайно високі результати в задачах розпізнавання образів. Ці мережі здатні автоматично навчатися виділяти найважливіші характеристики об'єктів на зображеннях, що значно покращує точність і ефективність розпізнавання.

У цьому розділі розглядаються основні принципи глибинного навчання, зокрема архітектури нейронних мереж, які використовуються для розпізнавання об'єктів. Також будуть представлені ключові підходи, методи та алгоритми, що дозволяють забезпечити ефективне навчання моделей і їх застосування в реальних умовах. Серед таких методів особливо виділяються згорткові нейронні мережі (CNN), а також техніки, що використовують попередньо навчені моделі і перенесене навчання для підвищення ефективності та швидкості розпізнавання.

3.1.1 Принципи роботи глибоких нейронних мереж. Глибокі нейроннімережі (ГНМ) є ключовою технологією в сучасному розпізнаванні об'єктів. Їхній фундамент базується на концепції штучних нейронних мереж, які намагаються імітувати роботу мозку людини, моделюючи взаємодію між штучними "нейронами". ГНМ відрізняються від традиційних нейронних мереж багатошаровою структурою, що дозволяє ефективно навчатися складним залежностям у даних.

Глибока нейронна мережа складається з вхідного шару, який отримує сирі дані, таких як зображення чи відео; прихованих шарів, які виконують нелінійні перетворення вхідних даних; та вихідного шару, що забезпечує результат у вигляді ймовірностей або координат об'єкта для задач детекції. Згорткові шари (Convolutional Layers) виділяють ознаки, такі як контури чи форми, шари підвибірки (Pooling Layers) зменшують розмірність даних, а повнозв''язані шари (Fully Connected Layers) об'єднують ознаки для прийняття фінального рішення.

Навчання глибоких нейронних мереж базується на ітеративному процесі. На етапі прямого проходу дані проходять через мережу, а результат порівнюється з правильним значенням за допомогою функції втрат, яка визначає похибку. Зворотне поширення похибки дозволяє обчислити градієнти, що використовуються для оновлення ваг нейронів за допомогою алгоритмів оптимізації, таких як градієнтний спуск або Adam.

Глибокі нейронні мережі мають кілька важливих особливостей, які роблять їх ефективними в задачах розпізнавання об'єктів. Вони дозволяють виділяти локальні ознаки, що є важливими для ідентифікації об'єктів. На перших шарах виділяються прості ознаки, такі як лінії чи контури, тоді як на глибших шарах—більш абстрактні, наприклад, форми чи текстури. Крім того, вони можуть бути стійкими до трансформацій, таких як масштабування чи обертання, завдяки своїй архітектурі та навчанні на різноманітних даних.

ГНМ потребують великої кількості даних для навчання, що забезпечує високу узагальнюючу здатність. Також важливо враховувати обчислювальні ресурси, оскільки глибокі мережі є вимогливими до обчислювальної потужності.

Глибокі нейронні мережі є основою сучасних технологій розпізнавання об'єктів, оскільки їхня багатошарова структура забезпечує високу здатність до навчання складних залежностей у даних. Принципи роботи ГНМ, включаючи прямий прохід, зворотне поширення похибки та оптимізацію, дозволяють ефективно вирішувати задачі детекції, класифікації та сегментації об'єктів у реальному часі.

3.1.2 Типи задач у розпізнаванні об'єктів. У розпізнаванні об'єктів виділяють три основні типи задач: детекція, класифікація та сегментація. Кожна з цих задач вирішує певний аспект аналізу зображень і відеопотоків, використовуючи різні підходи та архітектури нейронних мереж.

Детекція об'єктів спрямована на виявлення всіх об'єктів на зображенні, визначення їхніх класів та локалізацію у вигляді обмежувальних прямокутників. Головна мета детекції—розпізнати всі об'єкти, що знаходяться на зображенні, та визначити їхні координати.

Сучасні алгоритми детекції, такі як YOLO (You Only Look Once), SSD (Single Shot Multibox Detector) та Faster R-CNN, використовують згорткові нейронні мережі для ефективного виявлення об'єктів. Наприклад, YOLO здійснює поділ зображення на сітку та прогнозує прямокутники і класи одночасно, що забезпечує високу швидкість і точність.

Детекція є критично важливою в реальному часі, особливо для автономного водіння, відеоспостереження та інших сфер, де швидкість і точність розпізнавання є ключовими параметрами.

Класифікація об'єктів зосереджується на визначенні класу, до якого належить об'єкт на зображенні. Наприклад, система може класифікувати зображення як "кіт", "собака" або "автомобіль".

Для вирішення цієї задачі використовуються згорткові нейронні мережі, такі як ResNet, VGG та EfficientNet. Класифікація вимагає великих обсягів даних для навчання моделі, а також різноманітних методів аугментації для підвищення стійкості до змін у входах, таких як масштабування чи обертання.

Основною перевагою класифікації є її відносна простота у реалізації. Однак цей підхід не враховує позицію об'єкта на зображенні, тому часто використовується як складова частина складніших задач.

Сегментація об'єктів є більш детальною задачею, ніж детекція та класифікація. Її мета—ідентифікація кожного пікселя зображення, що належить до конкретного об'єкта чи класу. Залежно від типу сегментації розрізняють:

- семантичну сегментацію: визначає всі пікселі, що належать до певного класу. Наприклад, усі пікселі, які утворюють автомобіль або дорогу, будуть виділені одним кольором;

- інстансну сегментацію: не лише класифікує пікселі, але й розділяє об'єкти одного класу. Наприклад, якщо на зображенні кілька автомобілів, то кожен з них буде виділений окремо.

Архітектури, такі як U-Net, Mask R-CNN та DeepLab, широко використовуються для сегментації завдяки їхній здатності працювати з високою роздільною здатністю зображень і забезпечувати точне розпізнавання контурів об'єктів.

Сегментація має широкий спектр застосувань, включаючи медичну діагностику, автономне водіння та роботи в галузі доповненої реальності.

Тож, типи задач у розпізнаванні об'єктів—детекція, класифікація та сегментація—відповідають на різні потреби аналізу візуальних даних. Використання глибоких нейронних мереж дозволяє вирішувати ці задачі з високою точністю та ефективністю. Залежно від цілей і специфіки застосування, вибір конкретного типу задачі впливає на архітектуру моделі та вимоги до обчислювальних ресурсів

3.1.3 Особливості роботи з реальними даними. Робота з реальними даними є одним із ключових аспектів у розробці систем розпізнавання об'єктів. Реальні дані мають низку особливостей, які необхідно враховувати для забезпечення ефективного навчання та роботи моделей глибокого навчання.

Реальні дані часто характеризуються нерівномірністю. Це означає, що різні класи можуть бути представлені у вибірці з великою диспропорцією (проблема дисбалансу класів). Наприклад, у наборі даних для розпізнавання дорожніх знаків кількість зображень знаків "Стоп" може значно перевищувати кількість інших типів знаків. Така нерівномірність ускладнює навчання моделей і може призвести до зниження точності для менш представлених класів.

Реальні дані часто містять шум: спотворення, неповні або пошкоджені зображення, а також артефакти, викликані умовами зйомки. Наприклад, розмиття через рух, тіні або відблиски можуть ускладнити виділення ключових ознак. Для підвищення стійкості моделей до таких факторів використовуються методи аугментації даних, які імітують різні типи спотворень.

Зображення з реального світу можуть значно відрізнятися за умовами освітлення, ракурсом зйомки, масштабом об'єктів тощо. Це вимагає, щоб моделі мали високу здатність до узагальнення та могли коректно розпізнавати об'єкти в різних контекстах.

Робота з реальними даними часто передбачає обробку великих наборів зображень або відео. Це створює виклики для зберігання, обробки та навчання моделей, вимагаючи потужних обчислювальних ресурсів. Крім того, необхідно ефективно обробляти анотації, що супроводжують ці дані.

Реальні дані зазвичай потребують значної підготовки перед використанням. Цей процес включає:

- анотування: маркування об'єктів на зображеннях, яке може бути трудомістким і вимагати значних витрат часу;

- очищення: видалення помилкових, дублікатних або нерелевантних даних;

- аугментація: створення додаткових варіантів даних для підвищення різноманітності та стійкості моделі.

У багатьох випадках робота з реальними даними вимагає врахування етичних і правових аспектів, зокрема захисту конфіденційності. Для цього використовуються методи анонімізації, наприклад, розмиття облич або номерних знаків на зображеннях.

Особливості роботи з реальними даними створюють низку викликів, але правильний підхід до їх обробки та підготовки дозволяє забезпечити якісне навчання моделей і їхню стійкість до реальних умов експлуатації. Використання сучасних методів аугментації, очищення та ефективного управління даними є важливими етапами для досягнення успіху в реалізації систем розпізнавання об'єктів.

3.2 Архітектури нейронних мереж для розпізнавання об'єктів

Розробка систем розпізнавання об'єктів значною мірою залежить від вибору архітектури нейронної мережі. Сучасні архітектури розрізняються за швидкістю роботи, точністю, складністю реалізації та потребою в обчислювальних ресурсах.

3.2.1 Порівняння сучасних архітектур. Розглянемо три основні архітектури, які широко використовуються для задач детекції об'єктів: YOLO, SSD і Faster R-CNN.

YOLO

YOLO (You Only Look Once) є одним із найшвидших підходів до детекції об'єктів. Його головна ідея полягає в тому, що зображення обробляється як єдине ціле, і модель одночасно прогнозує координати обмежувальних прямокутників та класи об'єктів.

Переваги YOLO:

- висока швидкість, що робить його придатним для роботи в реальному часі;

- відносно проста структура моделі.

Недоліки:

- можливі проблеми з точністю для дрібних об'єктів або об'єктів у щільних сценах.

SSD

SSD (Single Shot MultiBox Detector) поєднує швидкість і точність, розділяючи зображення на сітку та прогнозуючи обмежувальні прямокутники для кожної комірки. Ця архітектура використовує різнорівневі ознаки для покращення розпізнавання об'єктів різного масштабу.

Переваги SSD:

- баланс між точністю та швидкістю;

- ефективність у детекції об'єктів різного розміру.

Недоліки:

- трохи поступається YOLO в швидкості.

Faster R-CNN

Faster R-CNN є однією з найточніших архітектур для детекції об'єктів. Ця модель використовує регіональні пропозиції (Region Proposal Networks), які швидко генерують кандидати на обмежувальні прямокутники.

Переваги Faster R-CNN:

- висока точність для складних сцен;

- підходить для задач, де точність важливіша за швидкість.

Недоліки:

- низька швидкість у порівнянні з YOLO та SSD;

- високі вимоги до обчислювальних ресурсів.

3.2.2 Легковагові моделі для роботи в реальному часі. Для задач, щовимагають роботи на пристроях з обмеженими обчислювальними ресурсами або в умовах реального часу, застосовуються легковагові моделі. Серед них виділяються такі архітектури, як MobileNet, EfficientDet і NanoDet.

- mobileNet: Застосовує глибокі розділені згортки (depthwise separable convolutions), що знижує кількість параметрів і збільшує швидкість обробки;

- efficientDet: Побудована на основі EfficientNet, використовує баланс між точністю та ефективністю, що робить її придатною для мобільних пристроїв;

- nanoDet: Спеціалізована для надлегких середовищ, забезпечує високу швидкість роботи при достатній точності.

Легковагові моделі є незамінними для використання в IoT-пристроях, смартфонах, а також для завдань, що потребують мінімального споживання енергії.

Вибір архітектури нейронної мережі для розпізнавання об'єктів залежить від специфіки завдання. Моделі YOLO, SSD і Faster R-CNN є ефективними для детекції об'єктів із різними вимогами до точності й швидкості, тоді як легковагові моделі забезпечують оптимальне рішення для роботи в реальних умовах на обмежених ресурсах.

3.3 Процес навчання нейронних мереж

Навчання нейронних мереж є багатоступеневим процесом, що включає кілька ключових етапів, починаючи від підготовки даних і закінчуючи оцінкою продуктивності моделі. На цьому етапі основна увага приділяється створенню умов для ефективного навчання, щоб нейронна мережа могла узагальнювати знання та демонструвати високу точність у реальних умовах використання. Підготовка даних є однією з найважливіших складових цього процесу.

3.3.1 Підготовка даних. Ефективність навчання нейронних мереж значною мірою залежить від якості та обсягу даних, які використовуються для тренування моделі. Підготовка даних є критично важливим етапом, що включає кілька ключових аспектів: аугментацію, нормалізацію та балансування класів.

Аугментація даних — це процес штучного розширення набору даних за рахунок застосування різних перетворень до наявних прикладів.

Мета аугментації — збільшення різноманітності даних, що покращує узагальнюючу здатність моделі та знижує ризик перенавчання.

До методів аугментації належать геометричні трансформації (обертання, масштабування, зсуви, обрізання, дзеркальне відображення), зміни кольору (варіації яскравості, контрасту, насиченості) та додавання шумів (гауссовий шум, випадкове видалення частин зображення).

Для автоматизації аугментації часто використовують бібліотеки, такі як Albumentations, TensorFlow ImageDataGenerator або PyTorch transforms. Вибір методів залежить від задачі та особливостей об'єктів, що розпізнаються.

Нормалізація даних є необхідним кроком для приведення значень пікселів до заданого діапазону. Основні підходи включають масштабування до діапазону [0, 1] (ділення піксельних значень на 255) та Z-нормалізацію (центрування значень шляхом віднімання середнього і ділення на стандартне відхилення). Ці методи зменшують вплив різниці у масштабах між ознаками та покращують збіжність градієнтного спуску під час навчання.

У задачах розпізнавання об'єктів часто використовують нормалізацію на основі статистик великих наборів даних, таких як ImageNet, що особливо актуально при використанні попередньо навчених моделей.

Нерівномірний розподіл класів у наборі даних може призводити до зміщення моделі в бік домінуючих класів. Для вирішення цієї проблеми застосовують різні підходи, включаючи оверсемплінг (збільшення кількості прикладів рідкісних класів шляхом дублювання або аугментації), андерсемплінг (зменшення кількості прикладів домінуючих класів), ваги класів у функції втрат та генеративні моделі (наприклад, GAN для створення нових зразків).

Правильне балансування класів сприяє тому, що модель вчиться однаково добре розпізнавати всі категорії об'єктів, що особливо важливо для задач розпізнавання у реальному часі, де точність має критичне значення.

3.3.2 Метрики оцінки якості моделей. Оцінка якості моделей глибокого навчання є важливим етапом для визначення ефективності їх використання у реальних задачах. Основні метрики, які застосовуються для задач розпізнавання об'єктів, включають Mean Average Precision (mAP), FPS (Frames Per Second) та Latency (затримку).

Mean Average Precision (mAP) є однією з найпоширеніших метрик для оцінки якості моделей у задачах розпізнавання об'єктів. Вона обчислюється як середнє значення точності для кожного класу при різних порогових значеннях IoU (Intersection over Union). mAP дозволяє оцінити, наскільки добре модель ідентифікує та локалізує об'єкти в межах зображення. Високе значення mAP свідчить про те, що модель демонструє високу точність і здатність до узагальнення.

FPS (Frames Per Second) вимірює кількість кадрів, які модель може обробити за одну секунду. Ця метрика є ключовою для задач реального часу, таких як відеоспостереження чи автономне водіння, де швидкість обробки є критично важливою. Високе значення FPS свідчить про те, що модель здатна швидко обробляти великі обсяги даних, що забезпечує плавну роботу в реальних умовах.

Latency (затримка) відображає час, необхідний для обробки одного зображення або кадру. Затримка включає час на передобробку даних, обчислення прогнозу та постобробку результатів. Низьке значення затримки є важливим для задач, де необхідно приймати рішення в реальному часі. Наприклад, у системах безпеки або в медичних додатках висока затримка може призводити до серйозних наслідків.

Поєднання цих метрик дозволяє всебічно оцінити ефективність моделі, враховуючи як її точність, так і продуктивність у реальних умовах.

3.3.3 Вплив гіперпараметрів та розміру вибірки. Гіперпараметри та розмір вибірки значно впливають на процес навчання нейронних мереж та їхню продуктивність. Правильний вибір цих параметрів дозволяє досягти оптимального балансу між точністю моделі, швидкістю її роботи та здатністю до узагальнення.

Гіперпараметри — це налаштування, які визначають поведінку алгоритму навчання та структуру моделі до початку тренування. До основних гіперпараметрів належать:

- розмір пакету (Batch size): Менші значення дозволяють отримувати точніші оновлення градієнтів, але збільшують час навчання. Великі значення сприяють швидшому навчання, але можуть знижувати здатність моделі до узагальнення;

- швидкість навчання (Learning rate): Висока швидкість навчання прискорює збіжність, але може призводити до пропуску оптимальних значень. Низька швидкість забезпечує більш точне налаштування параметрів, але вимагає більше епох;

- кількість шарів і нейронів у шарі: Збільшення кількості шарів та нейронів може покращити здатність моделі виявляти складні патерни, але також підвищує ризик перенавчання;

- функція активації: Вибір функції активації (наприклад, ReLU, Sigmoid або Tanh) впливає на швидкість збіжності та здатність моделі обробляти нелінійні залежності.

Оптимізація гіперпараметрів зазвичай виконується за допомогою таких методів, як пошук по сітці (Grid Search), випадковий пошук (Random Search) або використання алгоритмів байєсівської оптимізації.

Розмір вибірки даних, використаних для навчання моделі, також є критичним фактором, що впливає на її продуктивність. Недостатній розмір вибірки може призводити до перенавчання моделі, тоді як надмірно великий

обсяг даних може збільшувати час і ресурси, необхідні для навчання. Основні аспекти впливу розміру вибірки:

- забезпечення репрезентативності: Достатній обсяг даних має охоплювати всі можливі варіації вхідних характеристик, щоб модель могла узагальнювати знання;

- уникнення шуму: Великі вибірки можуть включати більше шуму, що ускладнює навчання моделі. Для вирішення цієї проблеми використовують методи очищення даних;

- рівновага між якістю та швидкістю: Оптимальний розмір вибірки дозволяє досягти високої точності моделі без значного збільшення часу навчання.

Ефективне використання гіперпараметрів у поєднанні з належним розміром вибірки забезпечує створення моделей, які демонструють високу продуктивність та стабільність у реальних умовах.

3.4 Інструменти та середовища для реалізації

Розробка моделей глибокого навчання та їх впровадження у задачах розпізнавання об'єктів вимагає використання спеціалізованих інструментів і середовищ. У цьому підрозділі розглянуто основні бібліотеки та інструменти, які широко застосовуються в сучасних дослідженнях і практичних реалізаціях.

3.4.1 Огляд бібліотек. Існує велика кількість бібліотек для реалізації нейронних мереж і роботи з даними. Розглянемо найпопулярніші з них, такі як PyTorch, TensorFlow і OpenCV.

PyTorch — це одна з найпопулярніших бібліотек для створення, навчання та тестування моделей глибокого навчання. Вона відрізняється гнучкістю та зручністю завдяки динамічній обчислювальній графіці, що дозволяє легко змінювати архітектуру моделі під час виконання. Основні переваги PyTorch:

- простота використання завдяки інтуїтивно зрозумілому API;

- підтримка апаратного прискорення через використання GPU;

- наявність великої кількості попередньо навчених моделей у бібліотеці TorchVision;

- широка підтримка спільноти та велика кількість прикладів у відкритому доступі.

TensorFlow — це потужна платформа з відкритим кодом для створення та розгортання моделей машинного навчання. Вона пропонує широкий набір інструментів для розробки моделей, їх оптимізації та впровадження у виробничі середовища. Основні переваги TensorFlow:

- підтримка як високорівневого API Keras, так і низькорівневих операцій для гнучкого налаштування;

- можливість масштабування моделей для роботи на кластерних системах;

- інструменти для впровадження у мобільні та вбудовані системи через TensorFlow Lite;

- активна підтримка спільноти та регулярні оновлення від Google.

OpenCV — це бібліотека комп’ютерного зору з відкритим кодом, яка забезпечує широкий спектр інструментів для обробки зображень і відео. Хоча OpenCV не є платформою для навчання нейронних мереж, вона часто використовується у поєднанні з іншими бібліотеками для передобробки даних і оптимізації. Основні можливості OpenCV:

- інструменти для обробки зображень, такі як фільтрація, сегментація та геометричні трансформації;

- підтримка алгоритмів детекції об’єктів, таких як Haar Cascade та DNN модулі для інтеграції попередньо навчених моделей;

- висока продуктивність завдяки оптимізації під апаратне забезпечення;

- легка інтеграція з Python та іншими мовами програмування.

Використання цих бібліотек дозволяє створювати ефективні рішення для розпізнавання об’єктів у реальному часі, забезпечуючи високу продуктивність та зручність розробки.

3.4.2 Використання апаратних прискорювачів. Для ефективного навчання та виконання моделей глибокого навчання часто використовують апаратні прискорювачі, такі як GPU (Graphics Processing Unit) та TPU (Tensor Processing Unit). Їхнє застосування дозволяє значно прискорити обчислення, що є особливо важливим у задачах, які потребують обробки великих обсягів даних у реальному часі.

GPU, або графічні процесори, є одними з найбільш популярних пристроїв для прискорення обчислень у глибокому навчанні. Завдяки своїй архітектурі, що дозволяє виконувати велику кількість паралельних обчислень, GPU забезпечують значне зменшення часу навчання моделей. Вони ідеально підходять для роботи з великими наборами даних і складними архітектурами нейронних мереж. Серед найпоширеніших GPU-платформ — NVIDIA з її бібліотеками CUDA та cuDNN, які оптимізовані для задач машинного навчання.

TPU, або тензорні процесори, є спеціалізованими апаратними пристроями, розробленими компанією Google для виконання операцій, пов'язаних із глибоким навчанням. TPU забезпечують ще більшу продуктивність для певних задач, таких як обчислення матричних операцій, які є основою нейронних мереж. Вони оптимізовані для роботи з TensorFlow і надають значні переваги у масштабованості та енергоспоживанні. TPU часто використовуються у хмарних середовищах, що дозволяє інтегрувати їх у великі проекти без необхідності купувати власне обладнання.

Обидва типи апаратних прискорювачів мають свої переваги та особливості, і вибір між ними залежить від специфіки завдання, доступності ресурсів та вимог до продуктивності. Використання GPU та TPU дозволяє досягти високої швидкості та ефективності роботи моделей глибокого навчання, що є критичним фактором у сучасних реальних застосуваннях.

3.4.3 Робота в хмарних середовищах. Хмарні середовища забезпечують ефективний підхід до розгортання та виконання моделей глибокого навчання, особливо коли йдеться про завдання, що потребують значних обчислювальних ресурсів. Використання хмарних платформ надає можливість масштабування, доступ до сучасного апаратного забезпечення та зменшення витрат на інфраструктуру.

Однією з ключових переваг роботи в хмарних середовищах є гнучкість у використанні обчислювальних ресурсів. Користувачі можуть вибирати конфігурації, які відповідають їхнім потребам, включаючи GPU, TPU або звичайні процесори. Це дозволяє оптимізувати продуктивність і витрати залежно від складності завдання. Крім того, хмарні провайдери, такі як AWS, Google Cloud і Microsoft Azure, пропонують інтеграцію з популярними бібліотеками машинного навчання, такими як TensorFlow, PyTorch та інші.

Іншим важливим аспектом є можливість зберігання та обробки великих обсягів даних. Хмарні сервіси забезпечують доступ до масштабованих сховищ даних, що дозволяє ефективно працювати з великими вибірками для навчання моделей. Також доступні сервіси для автоматизації процесів обробки та аналізу даних.

Безпека та управління доступом є ще однією перевагою хмарних платформ. Сучасні хмарні середовища надають інструменти для захисту даних, такі як шифрування, контроль доступу та моніторинг активності, що забезпечує конфіденційність і захищеність даних під час роботи.

Робота в хмарних середовищах також сприяє співпраці в командах. Різні члени команди можуть одночасно отримувати доступ до ресурсів, модифікувати моделі та аналізувати результати, що значно прискорює процес розробки.

Загалом, використання хмарних платформ для роботи з моделями глибокого навчання є потужним інструментом, що поєднує високу продуктивність, доступність ресурсів і зручність впровадження рішень у виробничі середовища.

3.5 Обмеження та виклики у реальному часі

Розробка систем розпізнавання об’єктів у реальному часі супроводжується низкою технічних і практичних обмежень. Висока вимогливість до апаратних і програмних ресурсів, складнощі інтеграції з існуючими системами та необхідність забезпечення стабільної роботи в різних умовах — це лише частина викликів, з якими стикаються дослідники та розробники. Крім того, такі системи часто працюють в умовах непередбачуваного середовища, де можуть виникати зовнішні фактори, що впливають на їхню продуктивність і точність.

Одним із ключових аспектів є компроміс між точністю та швидкістю. Моделі, які демонструють високу точність на тестових наборах, можуть виявитися занадто повільними для практичного використання у реальному часі. Це змушує розробників шукати шляхи оптимізації, зокрема шляхом зменшення складності моделі або використання менш ресурсоємних алгоритмів.

3.5.1 Проблеми із затримками та продуктивністю. Розпізнавання об’єктів у реальному часі ставить високі вимоги до затримок і продуктивності системи. Основними викликами у цьому контексті є забезпечення низької затримки, достатньої для обробки даних у режимі реального часу, і підтримка високої швидкодії навіть при роботі з великими обсягами даних.

Затримки можуть виникати на різних етапах обробки: під час захоплення даних із сенсорів, передавання даних до обчислювальних пристроїв, виконання обчислень і передачі результатів. Особливо критичними вони є в системах, які вимагають швидкої реакції, таких як автомобільні автономні системи чи відеоспостереження. Використання сучасного апаратного забезпечення, таких як GPU чи TPU, допомагає зменшити затримки, однак потребує оптимізації алгоритмів для максимально ефективного використання ресурсів.

Продуктивність системи визначається її здатністю обробляти дані зі швидкістю, що відповідає вимогам реального часу. Навіть високопродуктивні системи можуть стикатися із зниженням ефективності у випадках

перевантаження чи роботи з високороздільними зображеннями. Застосування методів оптимізації, таких як зменшення розміру моделей, використання квантованих обчислень або обробка лише релевантних частин зображення, дозволяє досягти кращого балансу між точністю і швидкістю.

Таким чином, ключовим завданням при реалізації систем розпізнавання об’єктів у реальному часі є розв’язання проблем, пов’язаних із затримками та продуктивністю. Це вимагає ефективного управління ресурсами, вдосконалення алгоритмів і використання передових апаратних рішень.

3.5.2 Методи оптимізації моделей. Оптимізація моделей глибокого навчання є критично важливою для забезпечення їхньої продуктивності та ефективності у реальному часі. Основні методи включають квантовування та прунінг, які дозволяють зменшити обчислювальну складність і використання пам’яті без значного погіршення точності.

Квантовування є технікою, яка зменшує кількість бітів, необхідних для зберігання вагових коефіцієнтів і активацій нейронної мережі. Наприклад, замість використання 32-бітної точності, можна перейти на 16-бітну або навіть 8-бітну точність. Це значно знижує обчислювальні витрати і дозволяє використовувати менш потужне обладнання, зберігаючи при цьому прийнятну якість розпізнавання. Квантовування особливо ефективне для розгортання моделей на пристроях з обмеженими ресурсами, таких як мобільні телефони чи вбудовані системи.

Прунінг, або зрізання, передбачає видалення малозначущих або зайвих елементів нейронної мережі, таких як нейрони чи вагові з’єднання. Це дозволяє зменшити розмір моделі та кількість обчислень, необхідних для її виконання. Прунінг може здійснюватися на основі різних критеріїв, таких як вплив ваг на загальну продуктивність моделі чи їхній рівень активації. У результаті модель стає більш компактною та швидкою, що особливо важливо для застосувань у реальному часі.

Ці методи, часто використовуючись у поєднанні, дозволяють досягти балансу між продуктивністю та точністю, роблячи системи розпізнавання об’єктів у реальному часі більш ефективними і доступними для широкого спектра застосувань.

РОЗДІЛ 4.
ПОРІВНЯННЯ РІЗНИХ ПІДХОДІВ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ В РЕАЛЬНОМУ ЧАСІ

Обробка даних у режимі реального часу є однією з найскладніших задач у сфері розпізнавання об’єктів. Різні архітектури нейронних мереж пропонують свої підходи до вирішення цієї проблеми, залежно від пріоритетів, таких як точність, швидкодія, затримки, обчислювальні ресурси та масштабованість. У цьому розділі буде проведено аналіз найбільш поширених підходів, які використовуються для розпізнавання об’єктів у реальному часі, з акцентом на їхні сильні та слабкі сторони.

Основна мета полягає в порівнянні таких архітектур, як Convolutional Neural Networks (CNN), You Only Look Once (YOLO), Single Shot Multibox Detector (SSD) та Region-based CNN (R-CNN) разом із його модифікаціями. Окрему увагу буде приділено архітектурам, оптимізованим для роботи з низьким рівнем затримки, що є ключовим критерієм у реальному часі. Результати аналізу дозволять обрати оптимальний підхід для розв’язання задач розпізнавання об’єктів у режимі реального часу.

4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) є основою більшості сучасних моделей для розпізнавання об’єктів. Вони були розроблені для ефективного аналізу візуальних даних шляхом автоматичного виділення ознак із зображень. Основною характеристикою CNN є використання згорткових шарів, які дозволяють моделі виділяти локальні залежності та ієрархічні ознаки, такі як контури, текстури та складніші структури об’єктів.

CNN складаються з кількох типів шарів: згорткових (convolutional), об’єднувальних (pooling) і повнозв’язних (fully connected). Згорткові шари відповідають за обробку просторової інформації, об’єднувальні зменшують

розмірність даних, а повнозв’язні використовуються для класифікації. Завдяки цій структурі CNN забезпечують високу точність у задачах розпізнавання об’єктів.

Серед основних переваг CNN — здатність автоматично визначати релевантні ознаки без необхідності ручного проектування. Це робить їх ефективними для широкого спектра задач, включаючи класифікацію зображень, сегментацію та локалізацію об’єктів.

Однак CNN мають і обмеження. Високі обчислювальні вимоги часто стають проблемою при використанні на пристроях із обмеженими ресурсами або у сценаріях реального часу. Для подолання цих викликів розробники використовують різні техніки оптимізації, такі як зменшення розміру моделі, квантовування або прунінг. Крім того, сучасні варіанти CNN, наприклад, MobileNet чи EfficientNet, спеціально спроектовані для ефективної роботи в умовах обмежених обчислювальних ресурсів, що робить їх привабливими для розпізнавання об’єктів у реальному часі.

CNN залишаються одним із найпоширеніших підходів у розпізнаванні об’єктів завдяки їхній універсальності, високій точності та адаптивності до різних задач.

4.2 You Only Look Once (YOLO)

You Only Look Once (YOLO) є одним із найпопулярніших підходів до розпізнавання об’єктів у реальному часі. Основна ідея YOLO полягає в тому, що модель розглядає зображення лише один раз, розбиваючи його на сітку та одночасно передбачаючи координати межових рамок і клас об’єкта для кожної з них. Цей підхід дозволяє досягти високої швидкості роботи, що робить його ідеальним для застосувань, де важлива мінімальна затримка.

Однією з ключових особливостей YOLO є його здатність інтегрувати розпізнавання об’єктів і локалізацію в єдину операцію, що значно знижує

обчислювальні витрати. Крім того, YOLO добре працює на великих зображеннях і здатний ефективно розпізнавати кілька об’єктів одночасно.

Попри свої переваги, YOLO має і слабкі сторони. Однією з них є порівняно нижча точність у визначенні дрібних об’єктів або тих, які знаходяться поблизу меж зображення. Однак ці недоліки частково компенсуються вдосконаленими версіями моделі, такими як YOLOv4, YOLOv5 та YOLOv7, які включають оптимізації для підвищення як точності, так і швидкодії.

У контексті реального часу YOLO є одним із найкращих виборів завдяки своєму балансу між точністю та швидкістю. Цей підхід широко використовується в задачах відеоспостереження, автономного водіння, аналізу спортивних подій та багатьох інших областях, де важливі висока швидкість і продуктивність.

4.3 Single Shot Multibox Detector (SSD)

Single Shot Multibox Detector (SSD) є ще одним підходом до розпізнавання об’єктів, який вирізняється своєю ефективністю та швидкодією. На відміну від більш складних підходів, таких як Region-based CNN (R-CNN), SSD виконує локалізацію та класифікацію об’єктів у межах одного проходу через мережу. Це дозволяє значно зменшити затримки та робить його придатним для використання в реальному часі.

Ключовою особливістю SSD є використання багаторівневої архітектури для виявлення об’єктів різних розмірів. Різні рівні в мережі відповідають за аналіз ознак на різних масштабах, що забезпечує високу якість розпізнавання як дрібних, так і великих об’єктів. Крім того, SSD використовує техніку прив’язки рамок (default boxes), яка значно полегшує процес передбачення місцеположення об’єктів.

Перевагами SSD є його висока швидкість та простота інтеграції у системи реального часу. Модель може працювати на менш потужному обладнанні,

зберігаючи прийнятну точність. Це робить SSD популярним вибором для мобільних пристроїв і вбудованих систем, де обчислювальні ресурси обмежені.

Однак SSD має певні недоліки. Зокрема, його точність може поступатися іншим сучасним моделям, особливо на складних наборах даних із високою варіативністю. Для покращення цього параметра розробники використовують модифікації SSD, такі як MobileNet-SSD, які оптимізовані для використання на пристроях із низьким енергоспоживанням.

Завдяки своїм характеристикам SSD залишається одним із найкращих виборів для задач, де потрібне поєднання швидкості та прийнятної точності, що робить його конкурентоспроможним варіантом у сфері розпізнавання об’єктів у реальному часі.

4.4 Region-based CNN (R-CNN) та його варіанти

Region-based Convolutional Neural Networks (R-CNN) є однією з найбільш впливових архітектур у галузі розпізнавання об’єктів. Основна концепція R-CNN полягає в тому, щоб спершу виділити потенційні області (region proposals), які можуть містити об’єкти, а потім виконати згортковий аналіз для кожної з цих областей. Цей підхід забезпечує високу точність, оскільки він фокусується лише на релевантних частинах зображення.

Оригінальна версія R-CNN мала низку недоліків, серед яких висока обчислювальна складність і значні затримки через необхідність обробки кожної області окремо. Ці проблеми були вирішені у вдосконалених варіантах, таких як Fast R-CNN і Faster R-CNN. Fast R-CNN включає оптимізацію процесу навчання та передбачення, тоді як Faster R-CNN додає регіональні пропозиції (region proposal network), що робить процес значно швидшим.

Ще одним важливим варіантом є Mask R-CNN, який додає можливість сегментації об’єктів, окрім їх локалізації. Це розширення робить Mask R-CNN ідеальним для задач, де потрібна деталізована інформація про форму об’єктів, наприклад, у медицині чи робототехніці.

Попри свою високу точність, R-CNN і його модифікації зазвичай менш ефективні з точки зору швидкості порівняно з YOLO чи SSD. Однак вони залишаються популярними в застосуваннях, де точність має вирішальне значення, таких як наукові дослідження, обробка зображень високої роздільної здатності та інші задачі, що вимагають глибокого аналізу.

4.5 Архітектури для глибокого навчання з низьким рівнем затримки

Розпізнавання об’єктів у режимі реального часу вимагає використання архітектур, здатних забезпечувати мінімальну затримку без значного зниження точності. Такі архітектури розроблені з акцентом на оптимізацію продуктивності на апаратних платформах із обмеженими обчислювальними ресурсами, таких як мобільні пристрої, дрони або вбудовані системи.

Одним із популярних підходів є використання архітектур, орієнтованих на ефективність, таких як MobileNet і EfficientNet. MobileNet базується на використанні глибинних згорток (depthwise separable convolutions), що значно зменшує кількість параметрів і обчислень, зберігаючи при цьому високу якість розпізнавання. EfficientNet, у свою чергу, пропонує масштабування мережі за трьома напрямами — глибина, ширина та роздільна здатність вхідного зображення, що дозволяє адаптувати модель до різних апаратних обмежень.

Ще одним важливим напрямом є розробка спеціалізованих моделей, таких як SqueezeNet, яка має компактну архітектуру та оптимізована для використання у вбудованих системах. Висока швидкість роботи цих моделей робить їх придатними для задач, де критично важливий час реакції, наприклад, у системах відеоспостереження або автономного керування транспортними засобами.

Крім того, моделі з низьким рівнем затримки часто інтегруються з апаратними прискорювачами, такими як графічні процесори (GPU) або тензорні процесори (TPU), що дозволяє досягати ще більшої швидкості. Комбінація ефективних архітектур і апаратного прискорення створює основу для систем

розпізнавання об’єктів, здатних працювати в реальному часі навіть за обмежених ресурсів.

4.6 Вибір оптимального підходу для реального часу

Вибір оптимального підходу для розпізнавання об’єктів у реальному часі залежить від ряду факторів, таких як продуктивність, обчислювальні ресурси, рівень затримки, а також потреби у точності. У цьому підрозділі розглядаються три моделі, які є представниками різних архітектур і відповідають вимогам реального часу.

Однією з найперспективніших моделей є YOLOv5, яка представляє собою вдосконалений варіант архітектури You Only Look Once. Завдяки високій швидкості обробки і прийнятній точності, YOLOv5 є ефективним рішенням для задач відеоспостереження, автономного водіння та інших сфер, де потрібна миттєва реакція.

Іншим варіантом є MobileNet-SSD, яка поєднує в собі архітектуру Single Shot Multibox Detector із легкими глибинними згортками. MobileNet-SSD особливо ефективна на пристроях із обмеженими ресурсами, таких як смартфони, дрони або вбудовані системи. Її здатність працювати на менш потужному апаратному забезпеченні робить її ідеальною для мобільних додатків та IoT-пристроїв.

Для задач, які вимагають максимальної точності, варто розглянути Faster R-CNN. Ця архітектура, хоч і поступається за швидкодією YOLO та SSD, забезпечує високий рівень деталізації та точності у розпізнаванні об’єктів. Faster R-CNN використовується в медичній діагностиці, аналізі високоякісних зображень та інших випадках, де затримка не є критичним фактором.

Таким чином, вибір підходу визначається балансом між точністю, швидкістю та доступними ресурсами. У майбутніх експериментах буде оцінено продуктивність цих моделей у реальних умовах, що дозволить обрати найкраще рішення для конкретних задач.

РОЗДІЛ 5.
ПРАКТИЧНЕ ДОСЛІДЖЕННЯ РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ В РЕАЛЬНОМУ ЧАСІ

5.1 Опис задачі та вибір даних

Сучасний розвиток глибинного навчання та комп'ютерного зору відкриває значні можливості для автоматизації процесів розпізнавання об'єктів на зображеннях у реальному часі. Основною метою цього дослідження є порівняння трьох популярних моделей для розпізнавання об'єктів: YOLOv8, MobileNet-SSD та Faster R-CNN. Кожна з цих моделей представляє окремий підхід до вирішення задачі, що дозволяє оцінити їх ефективність у різних умовах.

YOLOv8 є новітньою ітерацією популярної серії YOLO (You Only Look Once), яка орієнтована на високу швидкодію та здатність до обробки зображень у реальному часі. Завдяки своїй оптимізації та використанню сучасних технологій, ця модель ідеально підходить для систем, де важливі як швидкість, так і точність.

MobileNet-SSD побудована для роботи на мобільних та інших обмежених у ресурсах пристроях. Її архітектура спрямована на зменшення обчислювальної складності, що робить її ефективною для застосувань, де доступні апаратні ресурси є обмеженими. Ця модель поєднує високу швидкість роботи з прийнятною точністю.

Faster R-CNN є представником сімейства двоетапних моделей, які забезпечують високу точність за рахунок більших обчислювальних витрат. Завдяки цьому Faster R-CNN часто використовується в дослідженнях, де основний акцент робиться на якості розпізнавання.

Практична задача цього дослідження полягає у порівнянні результатів роботи зазначених моделей на спільному датасеті Pascal VOC 2012. Основними аспектами оцінки є:

1. Точність розпізнавання. Оцінка проводитиметься за допомогою метрики mean Average Precision (mAP), яка дозволяє виміряти середню точність моделей по всіх класах об'єктів.

2. Швидкість обробки. Враховуватиметься час, необхідний для обробки одного зображення, що є критичним фактором для систем реального часу.

3. Стійкість до шуму. Аналізуватиметься здатність моделей працювати в умовах, коли зображення містить артефакти чи викривлення.

Проведення цього дослідження дозволить зробити висновки щодо того, яка з архітектур моделей є найбільш придатною для використання в різних сценаріях, враховуючи їх переваги та обмеження.

Датасет Pascal VOC 2012 було обрано через його популярність у дослідницькій спільноті, якість анотацій та різноманітність представлених класів об'єктів. Цей набір даних є загальновизнаним стандартом для оцінки алгоритмів розпізнавання об'єктів. Він включає 20 категорій, таких як люди, тварини, транспортні засоби та предмети повсякденного вжитку, що забезпечує широкий спектр тестових сценаріїв.

Ключовою особливістю Pascal VOC 2012 є наявність високоточних анотацій. Кожне зображення в датасеті супроводжується інформацією про об'єкти, що містяться на ньому, включаючи координати прямокутних рамок, класифікацію об'єктів та ієрархічну структуру категорій. Ці дані дозволяють проводити точний аналіз продуктивності моделей за допомогою метрик, таких як mean Average Precision (mAP) та Precision-Recall.

Для проведення експериментів було використано попередньо навчені моделі. Датасет Pascal VOC 2012 проганявся через ці моделі без додаткового поділу на навчальну, валідаційну чи тестову вибірки, оскільки ціль полягала в оцінці продуктивності моделей, а не їхньому навчанні. Ключовим завданням було отримання прогнозів моделей для порівняння з існуючими анотаціями датасету.

Крім того, перед початком тренувань було виконано попередню обробку даних (препроцесинг). Вона включала:

1. Масштабування зображень. Усі зображення були приведені до єдиного розміру, що відповідає вхідним вимогам моделей. Це дозволило уникнути помилок, пов'язаних із різними розмірами вхідних даних.

2. Нормалізацію. Значення пікселів було нормалізовано, щоб забезпечити коректну обробку даних під час оцінювання моделей та стабільність їхньої роботи.

Використання Pascal VOC 2012 дозволило забезпечити об'єктивність та відтворюваність експериментів. Завдяки чітко структурованим даним та широкому набору категорій, моделі мали можливість показати свої сильні сторони та обмеження. Отримані результати аналізувалися за кількома критеріями:

- точність розпізнавання об'єктів у різних класах;

- час обробки кожного зображення;

- чутливість до шуму та інших артефактів, що можуть впливати на якість розпізнавання.

Таким чином, датасет Pascal VOC 2012 є ідеальним вибором для досягнення цілей цього дослідження, оскільки він надає всі необхідні ресурси для оцінки продуктивності сучасних моделей розпізнавання об'єктів.

5.2 Підготовка та обробка даних

Розглянемо особливості роботи з YOLOv8. Під час підготовки даних спершу завантажується датасет PASCAL VOC 2012 та організовується структура директорій. Анотації з формату XML конвертуються у формат YOLO, де для кожного об’єкта визначаються клас і нормалізовані координати (центр, ширина, висота) відносно розмірів зображення. Створюється файл `data.yaml`, який вказує шляхи до тренувальних і валідаційних даних, кількість класів і їх назви. Усі зображення автоматично масштабуються, а координати анотацій нормалізуються для забезпечення сумісності з моделлю YOLO.

Розглянемо особливості роботи з MobileNet-SSD. Зображення завантажуються з локальної директорії `dataset_path` за допомогою бібліотеки OpenCV. Після зчитування зображень у форматі BGR вони конвертуються у формат RGB за допомогою функції `cv2.cvtColor` та зберігаються як масиви NumPy для подальшої обробки. Анотації завантажуються із відповідної директорії `annotations_path` у форматі XML. Використовується бібліотека `xml.etree.ElementTree` для обробки структури XML-файлів і вилучення міток об'єктів, що містяться в тегах `<object>` і `<name>`.

Класи, отримані з анотацій Pascal VOC, зіставляються з класами COCO за допомогою словника `voc_to_coco_map`. Це забезпечує узгодженість міток між різними форматами наборів даних. Перед використанням у моделі TensorFlow зображення конвертується у тензор із додаванням вимірності, яка відповідає структурі вхідних даних моделі, використовуючи `tf.convert_to_tensor(image_np)[tf.newaxis, ...]`. Також дані масштабуються для відповідності формату MobileNet-SSD, яка потребує попередньо нормалізованих вхідних тензорів.

Справжні мітки об'єктів (ground truth) вилучаються з XML-файлів, і кожна мітка зіставляється з відповідним класом COCO через словник `voc_to_coco_map`. Це дозволяє використовувати їх для оцінки точності детекції та подальшого порівняння з виявленими моделлю об'єктами.

Розглянемо особливості роботи з Faster R-CNN. Зображення завантажуються з локальної директорії, відкриваються за допомогою бібліотеки PIL і конвертуються у формат RGB для забезпечення сумісності з моделлю, яка працює із кольоровими зображеннями. Потім ці зображення перетворюються у тензори за допомогою `transforms.ToTensor()`, що переводить пікселі у числовий формат у діапазоні [0, 1], необхідний для моделі PyTorch. Для подачі в модель до кожного зображення додається додаткова вимірність, щоб забезпечити формат входу як батч.

Анотації для зображень завантажуються з XML-файлів, де зчитуються координати обмежувальних рамок і назви класів об'єктів. Ці дані

перетворюються у числовий формат, а назви класів узгоджуються з COCO через мапінг. Координати рамок залишаються у форматі абсолютних значень, однак для порівняння результатів під час оцінки використовуються відносні значення через обчислення IoU (Intersection over Union).

5.3 Реалізація та оцінка моделей для розпізнавання об'єктів

Окремо виділено реалізацію експериментальної частини дослідження, націленого на оцінку трьох сучасних моделей для завдання розпізнавання об’єктів: YOLOv8, MobileNet-SSD та Faster R-CNN. Методом є створення умов для порівняння їхньої точності, ефективності та здатності працювати з деяким невеликим датасетом Pascal VOC 2012. Увага приділяється якості розпізнавання об’єктів, швидкості моделей та їх здатності працювати.

У межах цього розділу детально описується процес підготовки даних, налаштування кожної моделі, проведення тестування та отримання ключових метрик, таких як середня точність (mAP). Особлива увага приділяється інтеграції моделей з обчислювальними платформами, способам обробки та інтерпретації результатів. Аналіз включає огляд технічних викликів та способів їх подолання в контексті.

Одним із важливих аспектів реалізації було використання платформи Google Colab для написання та тестування коду. Це рішення прийняте з кількох причин. Вперше Google Colab надає безкоштовний доступ до потужних апаратних ресурсів, таких як GPU та TPU, що є критично важливим для навчання та тестування глибинних нейронних мереж. Крім того, інтеграція з Google Drive дозволяє легко зберігати та обробляти великі набори даних, що значно полегшує роботу з датасетом Pascal VOC 2012. По-друге, середовище Colab дозволяє швидко ітеративно тестувати різні моделі конфігурації, забезпечуючи гнучкість і доступність інструментів для обробки даних та аналіз результатів. Завдяки цим перевагам вибір Google Colab був оптимальним для ефективного виконання поставлених завдань у цьому рамках

Реалізація коду, спрямованого на підготовку даних та тестування моделі YOLOv8 на наборі даних PASCAL VOC 2012, проходить через кілька ключових етапів. Перш за все, здійснюється завантаження набору даних PASCAL VOC 2012, який включає зображення та анотації у форматі XML. Ці дані розпаковуються з архіву для подальшої обробки.

На початку імпортуються необхідні бібліотеки, такі як `os` для роботи з файловою системою та `xml.etree.ElementTree` для обробки XML-файлів. Після цього визначаються директорії для вхідних анотацій, вихідних файлів формату YOLO, а також для зображень.

Основним завданням цього коду є перетворення XML-анотацій у формат, сумісний з моделлю YOLO. Для цього визначається список класів об'єктів, які можуть бути виявлені у наборі даних. Кожна анотація у форматі XML містить інформацію про об'єкти на зображеннях, зокрема координати межової рамки (bounding box) та клас об'єкта.

Функція `convert_bbox` використовується для перетворення координат межових рамок з піксельного формату у відносний формат, необхідний для YOLO. Ця функція враховує розмір зображення та обчислює центральну точку межової рамки, а також її ширину і висоту у відносних одиницях.

Цикл, що обробляє кожен XML-файл, здійснює наступні дії: для кожного об'єкта у файлі перевіряється, чи належить він до одного з визначених класів. Якщо так, його межова рамка перетворюється за допомогою функції `convert_bbox`, а результат записується у відповідний текстовий файл.

Наступним етапом є створення конфігураційного файлу `data.yaml`, який містить інформацію про шляхи до тренувальних і валідаційних даних, а також список класів об'єктів. Цей файл є необхідним для подальшого тренування моделі YOLOv8.

Після створення конфігураційного файлу здійснюється розподіл зображень на тренувальний та валідаційний набори. Використовується функція `train_test_split` з бібліотеки `sklearn`, яка дозволяє розділити зображення у

пропорції 80:20. Зображення разом із відповідними анотаціями переміщуються у відповідні папки для тренування та валідації.

Завершальним етапом є тестування моделі YOLOv8 на валідаційному наборі даних. Для цього модель завантажується з передтренованими вагами, після чого виконується функція валідації `model.val`, яка перевіряє модель на підготовлених даних та генерує відповідні метрики точності.

Таким чином, код забезпечує повний цикл підготовки даних, починаючи з їх завантаження, обробки анотацій, створення необхідних конфігураційних файлів, розподілу даних та завершуючи тестуванням моделі на валідаційному наборі. Кожен етап є важливим для забезпечення коректності та ефективності роботи моделі в задачах розпізнавання об'єктів.

Код виконує кілька ключових завдань для тестування моделі обробки зображень MobileNet-SSD, навченої на датасеті COCO, із використанням зображень та анотацій з Pascal VOC 2012.

Спочатку встановлюються всі необхідні бібліотеки, зокрема TensorFlow для роботи з моделлю глибинного навчання, OpenCV для обробки зображень, lxml для парсингу XML-анотацій, а також Matplotlib для візуалізації результатів. Після цього відбувається підключення до Google Drive, що дозволяє завантажувати та працювати з великим обсягом даних безпосередньо у середовищі Google Colab.

Модель MobileNet-SSD завантажується з офіційного репозиторію TensorFlow у форматі `saved_model`. Її структура дозволяє виконувати інференс для об'єктів на зображеннях, повертаючи координати рамок, рівні впевненості та класи об'єктів. Оскільки модель навчена на датасеті COCO, створюється мапінг класів Pascal VOC на відповідні класи COCO для забезпечення узгодженості у назвах об’єктів.

Для обробки зображень використовується функція, яка завантажує файл із вказаного шляху, конвертує його з формату BGR (стандарт OpenCV) у RGB і повертає масив NumPy, придатний для обробки TensorFlow. Для зчитування анотацій з Pascal VOC створена функція, яка відкриває XML-файл, аналізує його

структуру, отримує назви об'єктів і зіставляє їх зі словником відповідностей VOC-COCO.

Процес детекції об'єктів відбувається через перетворення зображення на вхідний тензор, що передається до моделі. На виході отримуються координати рамок, класи та їхні рівні впевненості. Отримані результати використовуються для візуалізації: на вихідному зображенні малюються рамки навколо об'єктів, додаються назви класів та відповідні ймовірності.

Після обробки кожного зображення результати детекцій порівнюються із справжніми мітками, отриманими з анотацій. Для кожного класу розраховується точність як відсоткове співвідношення правильних детекцій до загальної кількості об'єктів цього класу у всіх зображеннях. В кінці роботи коду виводиться таблиця результатів, яка містить назви класів, точність для кожного з них, а також кількість правильно виявлених і загальних об'єктів.

На завершальному етапі код виконує візуалізацію результатів для кількох прикладів зображень. Це дозволяє наочно продемонструвати ефективність роботи моделі, відобразивши знайдені об'єкти із відповідними рамками та назвами.

Таким чином, цей код реалізує повний цикл оцінки роботи моделі MobileNet-SSD, дозволяючи порівнювати її ефективність у контексті специфічного датасету Pascal VOC 2012.

Код починається зі завантаження набору даних PASCAL VOC 2012, що включає зображення та їх анотації. Команда `wget` завантажує архів з набором даних, а `tar` розпаковує його в поточну директорію. Далі імпортуються необхідні бібліотеки, серед яких `os` для роботи з файловою системою, `random` для вибору випадкових зображень, `xml.etree.ElementTree` для обробки XML-анотацій, `torch` для використання моделі на основі глибокого навчання, `torchvision` для обробки зображень і анотацій, `matplotlib.pyplot` для візуалізації та `PIL.Image` для роботи із зображеннями.

Шляхи до папок з анотаціями та зображеннями налаштовуються в змінних `annotations_path` та `images_path`. Встановлюються параметри для вибірки

даних: `num_images_to_process` визначає кількість зображень, які будуть оброблені, `confidence_threshold` встановлює поріг впевненості для відбору детекцій, а `visualize_every` вказує на частоту візуалізації результатів.

Визначається мапінг назв класів між наборами COCO і VOC для забезпечення відповідності між різними наборами даних. Це важливо для узгодження результатів детекції з відповідними анотаціями.

Функція `process_annotations` обробляє анотації у форматі XML, отримуючи координати об'єктів та їх класи. Вона відкриває файл XML, зчитує кожен об'єкт, витягує клас та координати обмежувальної рамки (bounding box), і повертає список об'єктів.

Функція `evaluate_model` обчислює метрики оцінки моделі, такі як точність і середнє значення точності (mAP). Вона порівнює виявлені об'єкти з анотаціями за допомогою коефіцієнта перекриття (IoU) і визначає, наскільки добре модель виконує детекцію об'єктів для кожного класу.

Функція `visualize_detections` використовується для візуалізації детекцій та анотацій на зображеннях. Вона відображає зображення та накладає на нього прямокутники для кожного виявленого та справжнього об'єкта, використовуючи різні кольори для детекцій та анотацій.

Завантажується попередньо натренована модель Faster R-CNN з використанням `torchvision.models.detection`. Модель завантажується з попередньо тренованими вагами і переводиться в режим оцінки (`eval`).

Для обробки зображень випадково вибираються імена файлів із папки із зображеннями. Кожне зображення перетворюється в тензор за допомогою трансформацій, а модель робить передбачення, які фільтруються за порогом впевненості. Результати детекції зберігаються разом з анотаціями для кожного зображення.

Нарешті, функція `evaluate_model` викликається для оцінки моделі на основі збережених детекцій та анотацій. Результати оцінки включають точність і mAP для кожного класу, які виводяться на екран для аналізу.

5.4 Порівняння результатів моделей

У цьому підрозділі розглянуто результати тестування трьох моделей: YOLOv8, MobileNet-SSD та Faster R-CNN, проведені на датасеті Pascal VOC 2012. Основна увага приділяється метрикам точності, ефективності обробки та здатності працювати з обмеженим обсягом даних.

Результати моделі YOLOv8. YOLOv8 продемонструвала високу ефективність із значенням середньої точності (mAP) на рівні 70%. Завдяки своїй архітектурі, ця модель здатна обробляти зображення в режимі реального часу, що забезпечує високу швидкість розпізнавання об'єктів. При цьому час обробки одного зображення склав у середньому 227 мс на апаратних можливостях CPU Google colab безкоштовної версії . Проте виявлено, що модель мала проблеми з розпізнаванням дрібних об'єктів та об'єктів, що перекриваються, особливо в складних сценах.

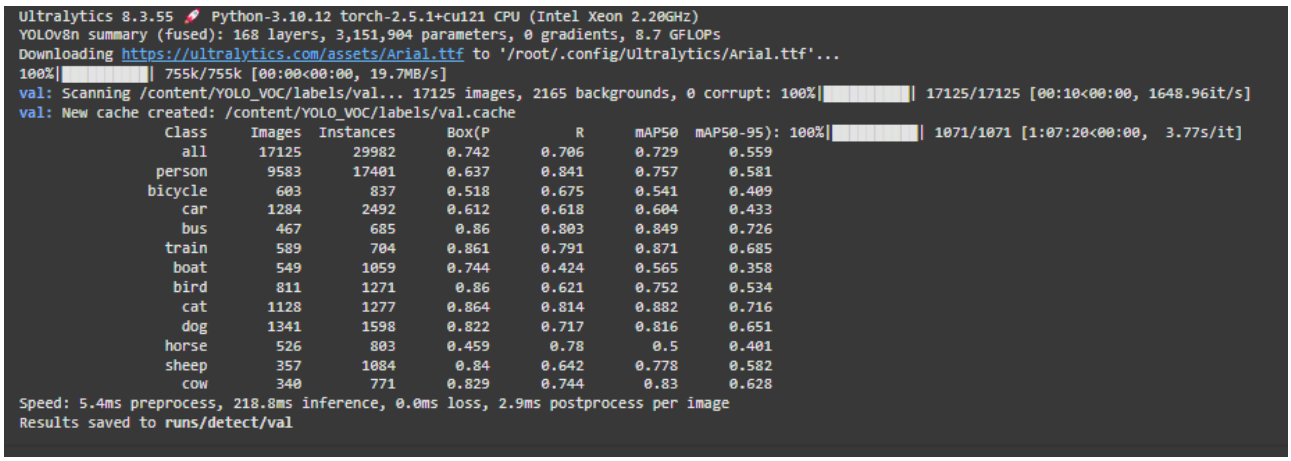


Рисунок 5.1 – Автоматичний висновок моделі Yolov8

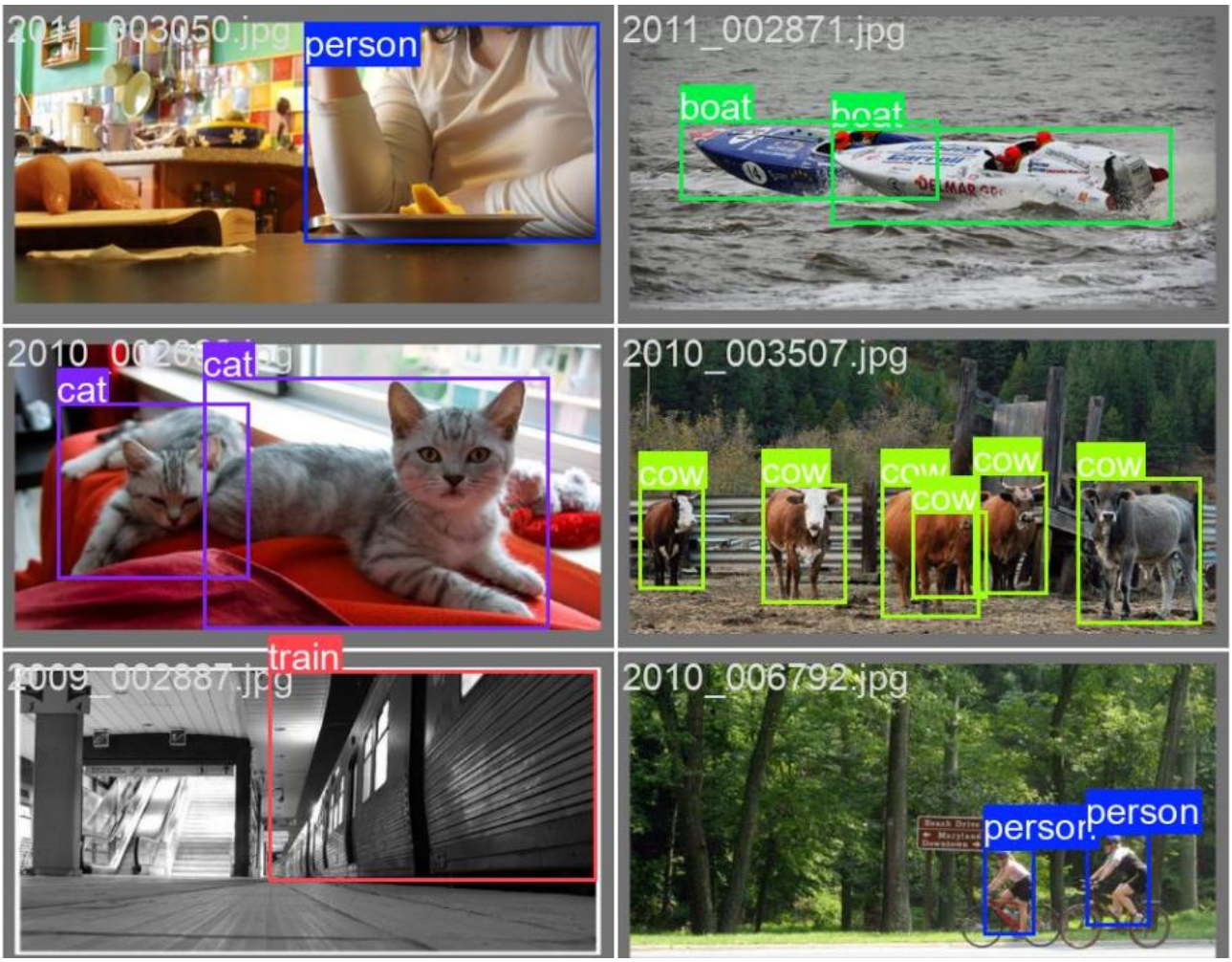


Рисунок 5.2 – Візуалізація деяких детекцій моделі Yolov8

MobileNet-SSD показала середню точність (mAP) на рівні 70.2%, що нижче за показники YOLOv8 та Faster R-CNN. Основною перевагою цієї моделі є її легковажність і здатність працювати на пристроях з обмеженими обчислювальними ресурсами, таких як мобільні пристрої. Середній час обробки зображення склав 9 мс на тому ж графічному процесорі. Однак модель виявилася менш точною при розпізнаванні складних сцен, що свідчить про необхідність її використання у менш критичних до точності застосуваннях.

```
Результати точності моделі за класами:
Клас: person, Точність: 100.00%, Правильно: 17401, Всього: 17401
Клас: chair, Точність: 73.63%, Правильно: 2250, Всього: 3056
Клас: potted plant, Точність: 58.90%, Правильно: 708, Всього: 1202
Клас: dog, Точність: 82.98%, Правильно: 1326, Всього: 1598
Клас: tv, Точність: 75.48%, Правильно: 674, Всього: 893
Клас: bottle, Точність: 58.87%, Правильно: 919, Всього: 1561
Клас: cow, Точність: 87.94%, Правильно: 678, Всього: 771
Клас: car, Точність: 81.50%, Правильно: 2031, Всього: 2492
Клас: bicycle, Точність: 80.65%, Правильно: 675, Всього: 837
Клас: motorcycle, Точність: 89.51%, Правильно: 717, Всього: 801
Клас: train, Точність: 92.05%, Правильно: 648, Всього: 704
Клас: bus, Точність: 90.22%, Правильно: 618, Всього: 685
Клас: couch, Точність: 73.01%, Правильно: 614, Всього: 841
Клас: horse, Точність: 89.79%, Правильно: 721, Всього: 803
Клас: cat, Точність: 88.65%, Правильно: 1132, Всього: 1277
Клас: sheep, Точність: 92.16%, Правильно: 999, Всього: 1084
Клас: airplane, Точність: 93.11%, Правильно: 933, Всього: 1002
Клас: bird, Точність: 81.12%, Правильно: 1031, Всього: 1271
Клас: boat, Точність: 81.30%, Правильно: 861, Всього: 1059
```

Рисунок 5.3 – Результати моделі MobileNET-SSD після датасету COCO

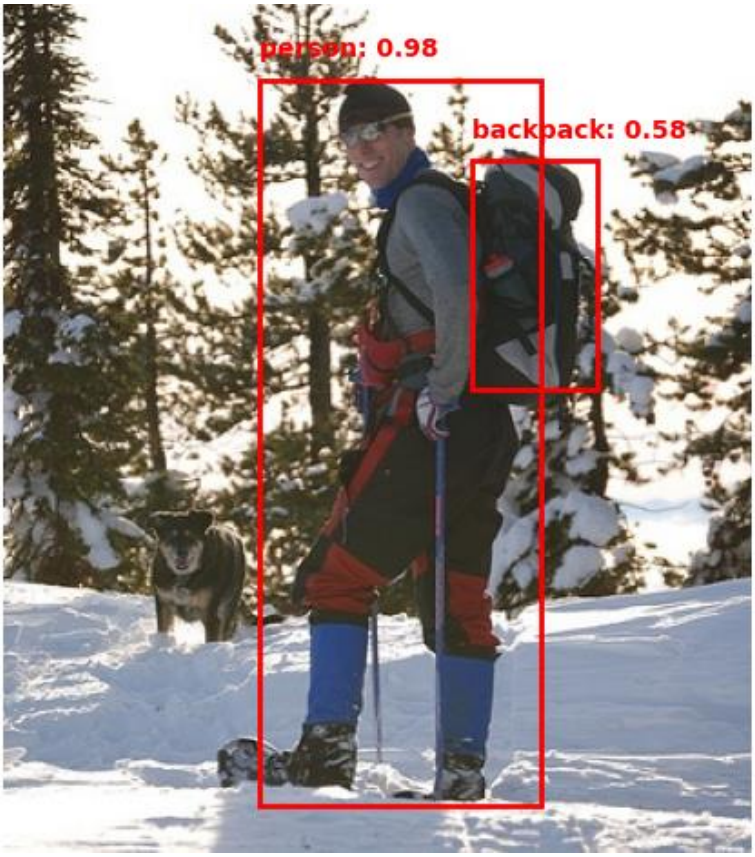


Рисунок 5.4 – Візуалізація результатів розпізнавання об’єктів моделлю MobileNet-SSD

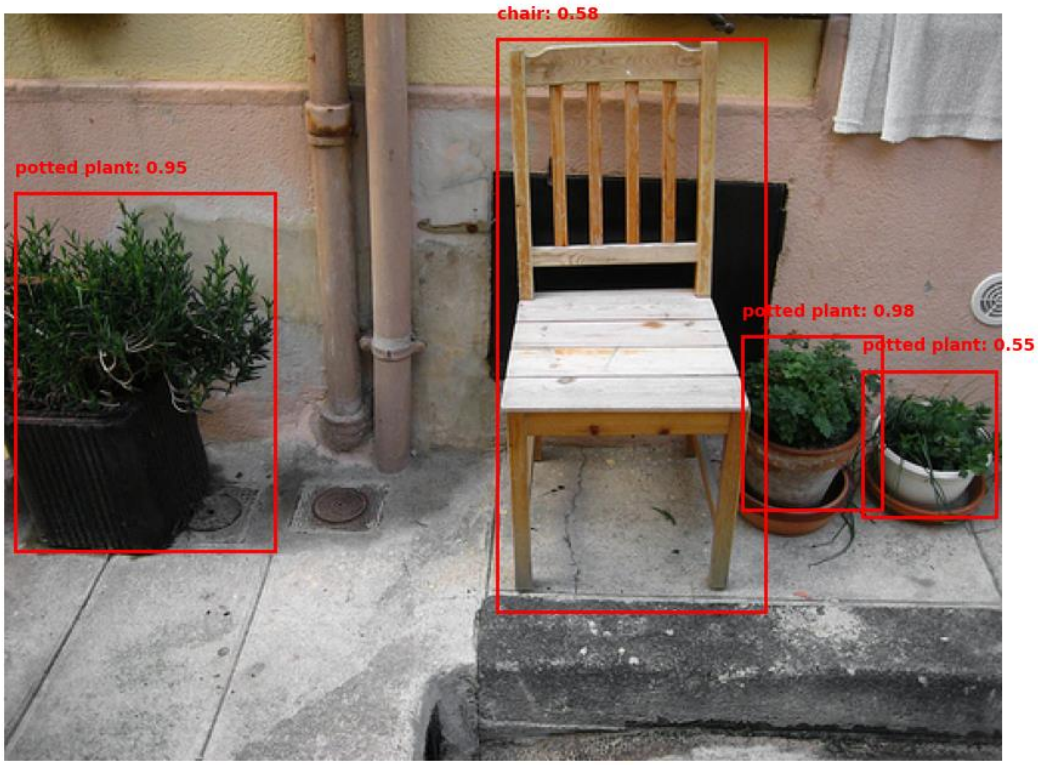


Рисунок 5.5 - Візуалізація результатів розпізнавання об’єктів моделлю MobileNet-SSD

Але є і проблеми, наприклад, помилкове визначення ноутбука

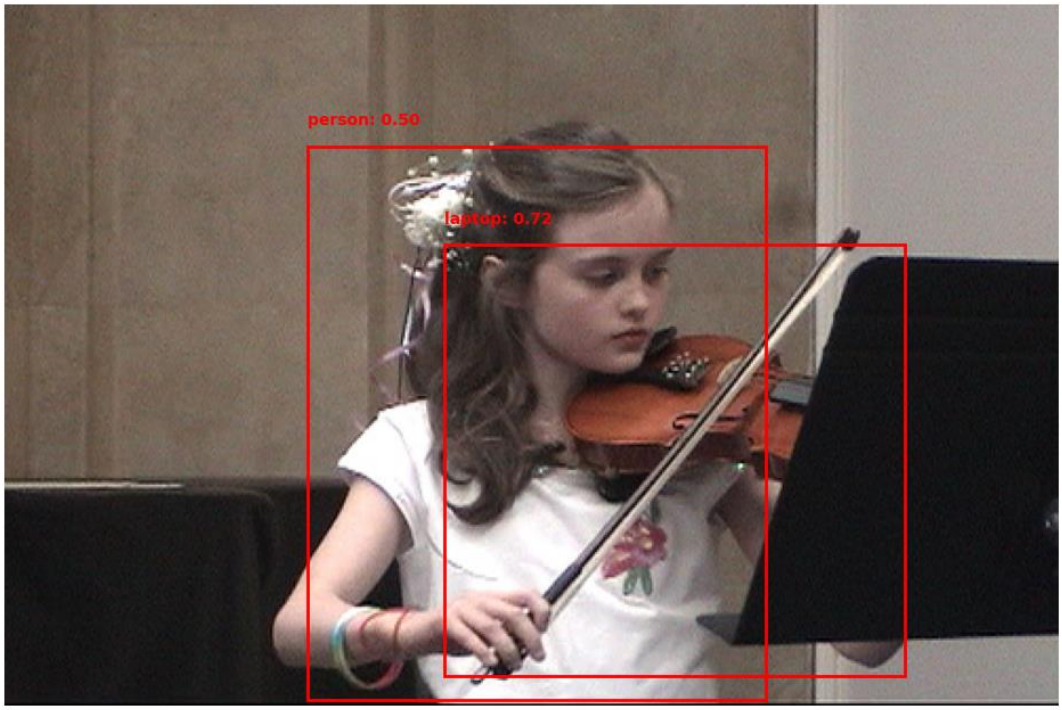


Рисунок 5.6 – Візуалізація неправильного розпізнавання об’єкта моделлю MobileNet-SSD

Результати моделі Faster R-CNN. Faster R-CNN продемонструвала середню точність (mAP), яка склала 68.8%. Ця модель показала чудові результати у виявленні дрібних та перекриваючихся об'єктів завдяки використанню регіональних пропозицій. Проте час обробки зображення був значно більшим і склав у середньому 150 мс на графічному процесорі. Це робить модель менш придатною для застосувань, що вимагають обробки в реальному часі, але ідеальною для задач, де точність є критично важливою.

```
Точність моделі: 0.7793296089385475
mAP: 0.6889157688571518
Клас car: 344/477 (точність: 0.72)
Клас potted_plant: 0/182 (точність: 0.00)
Клас bus: 95/114 (точність: 0.83)
Клас airplane: 141/164 (точність: 0.86)
Клас tv: 132/154 (точність: 0.86)
Клас bird: 160/186 (точність: 0.86)
Клас train: 104/127 (точність: 0.82)
Клас dog: 204/258 (точність: 0.79)
Клас chair: 314/534 (точність: 0.59)
Клас bottle: 149/225 (точність: 0.66)
Клас cat: 224/257 (точність: 0.87)
Клас cow: 82/120 (точність: 0.68)
Клас person: 2655/2909 (точність: 0.91)
Клас couch: 77/145 (точність: 0.53)
Клас horse: 131/156 (точність: 0.84)
Клас motorcycle: 116/144 (точність: 0.81)
Клас sheep: 135/177 (точність: 0.76)
Клас dining_table: 0/127 (точність: 0.00)
Клас boat: 105/177 (точність: 0.59)
Клас bicycle: 133/169 (точність: 0.79)
```

Рисунок 5.7 – Результат роботи моделі Faster R-CNN

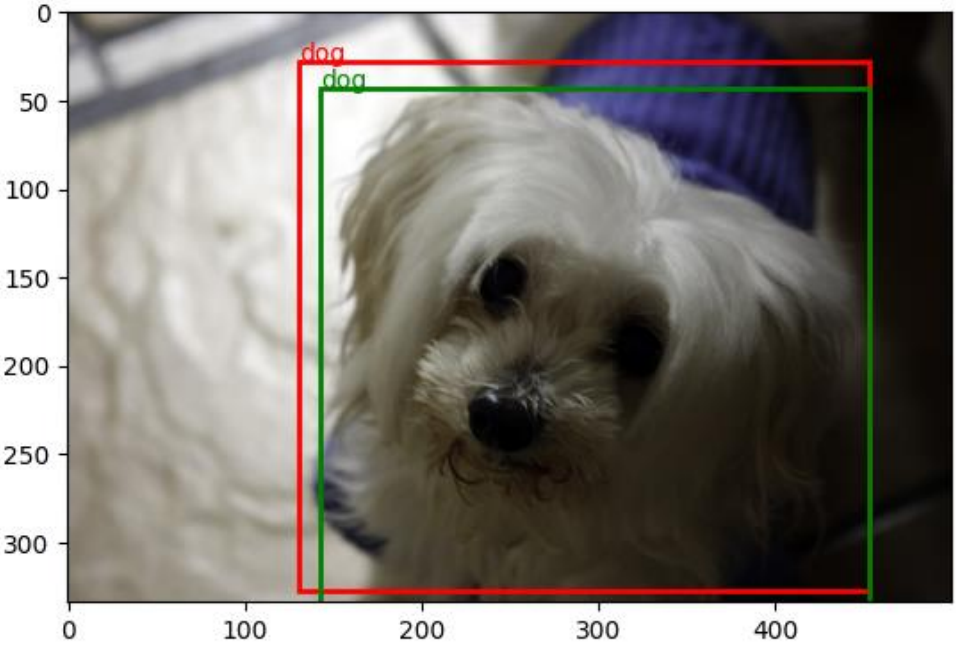


Рисунок 5.8 – Візуалізація розпізнавання моделлю Faster R-CNN, де зелена рамка це анотації а червона це результат моделі

При чому ця модель працює набагато краще бо вона добре розпізнає об'єкти з поганою видимістю, наприклад

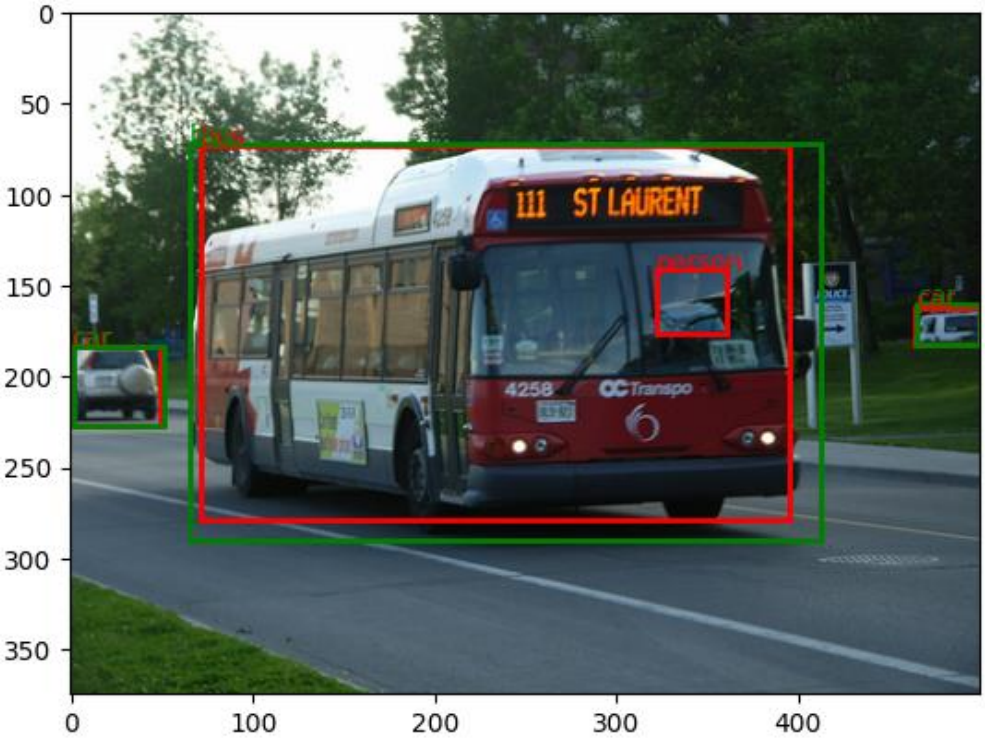


Рисунок 5.9 – Модель Faster R-CNN окрім машини та автобуса розпізнає водія

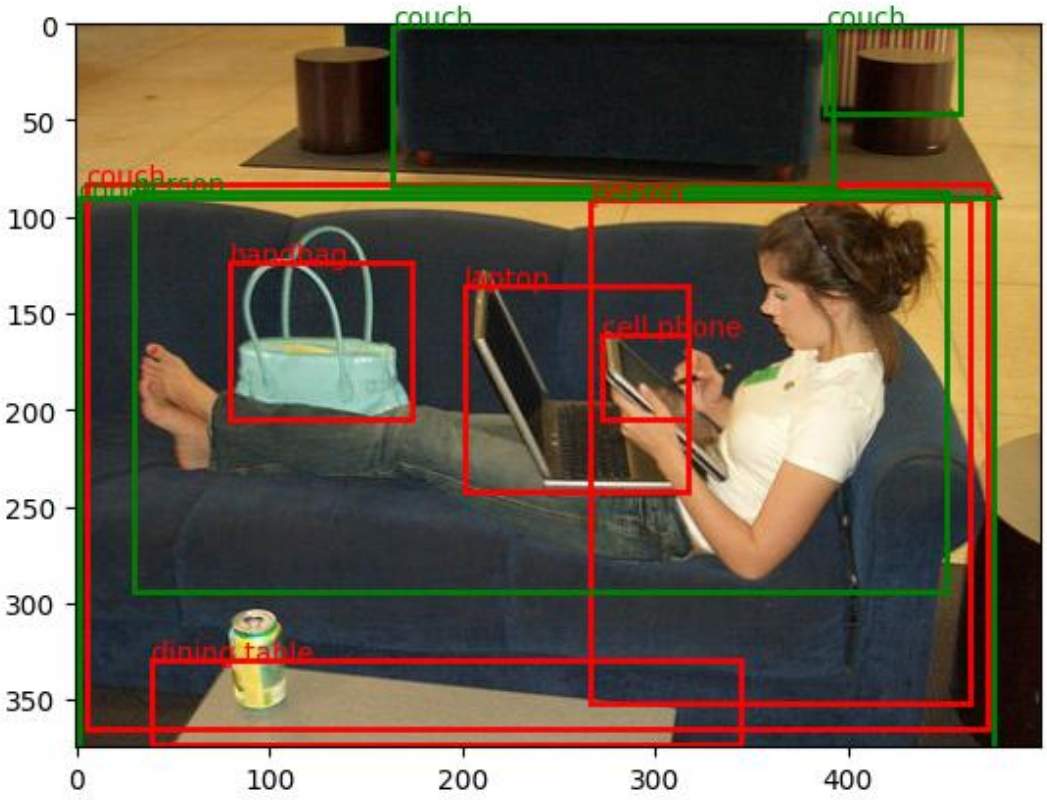


Рисунок 5.10 – Ще один приклад результату роботи моделі Faster R-CNN

Порівняння показників моделей. Для оцінки ефективності трьох моделей (YOLOv8, MobileNet-SSD та Faster R-CNN) було проведено експерименти на датасеті Pascal VOC 2012. Основними метриками для порівняння моделей були вибрані:

- точність: Точність моделі показує відсоток правильно класифікованих об'єктів із загальної кількості передбачень;

- mAP (Mean Average Precision): середня точність для всіх класів;

- час обробки одного зображення (Inference Time): середній час, необхідний для моделі для обробки одного зображення;

- кількість пропущених об'єктів (Missed Detections): кількість об'єктів, які модель не розпізнала;

- продуктивність у реальному часі (FPS): кількість кадрів, які модель може обробити за секунду.

Щоб визначити відсоток правильно класифікованих об’єктів застосуємо формулу:

$$\text{Точність} = \frac{\text{Сума правильних відповідей}}{\text{Загальна кількість об'єктів}} \times 100\% \quad (5.1)$$

Для YOLOv8, MobileNet-SSD та Faster R-CNN ці дані надані в результатах.

Для визначення середньої точності для всіх класів застосовується формула:

$$mAP = \frac{\text{Сума точностей усіх класів}}{\text{Кількість класів}} \quad (5.2)$$

- для YOLOv8 та Faster R-CNN mAP наданий у результатах;

- для MobileNet-SSD потрібно розрахувати середню точність усіх класів.

Розрахунок для MobileNet-SSD:

$$mAP = \frac{100 + 73.6 + 58.9 + 83 + 75.5 + 58.9 + 88 + 81.5 + 80.6 + 89.5 + 92 + 90.2 + 73 + 89.8 + 88.6 + 92.1 + 93.1 + 81.1 + 81.3}{19}$$

$$= 0.8189 = 81.89\%$$

Час обробки одного зображення (Inference Time)

В коді реалізовуємо функцію що рахує час на одне зображення а потім виводе середнє значення, для всіх моделей час обробки зображення вже наданий.

Yolov8 - 218.8 мс

MobileNet-SSD - 437.8 мс

Faster R-CNN - 1379.0 мс

Кількість пропущених об'єктів (Missed Detections)

Рахується за формулою:

Missed Detections=Загальна кількість об’єктів−Кількість правильно розпіз наних об’єктів

Yolov8 - 4586

MobileNet-SSD - 4402

Faster R-CNN - 1501

6. Продуктивність у реальному часі (FPS)

Рахується прогнозовано, за допомогою часу за який модель обробляє одне зображення:

$$FPS = \frac{1000}{Interence\ Time(\text{мс})} \quad (5.3)$$

Результати експериментів представлені у таблиці нижче:

Таблиця 5.1 - Результати ефективності моделей

Модель	Точність (%)	mAP	Час обробки одного зображення (мс)	Кількість пропущених об'єктів	Прогнозований FPS
YOLOv8	84.70	0.559	218.8	4586	4.57
MobileNet-SSD	88.81	0.819	437.8	4402	2.28
Faster R-CNN	77.93	0.689	1379.0	1501	0.73

Також для наочності були побудовані графіки:

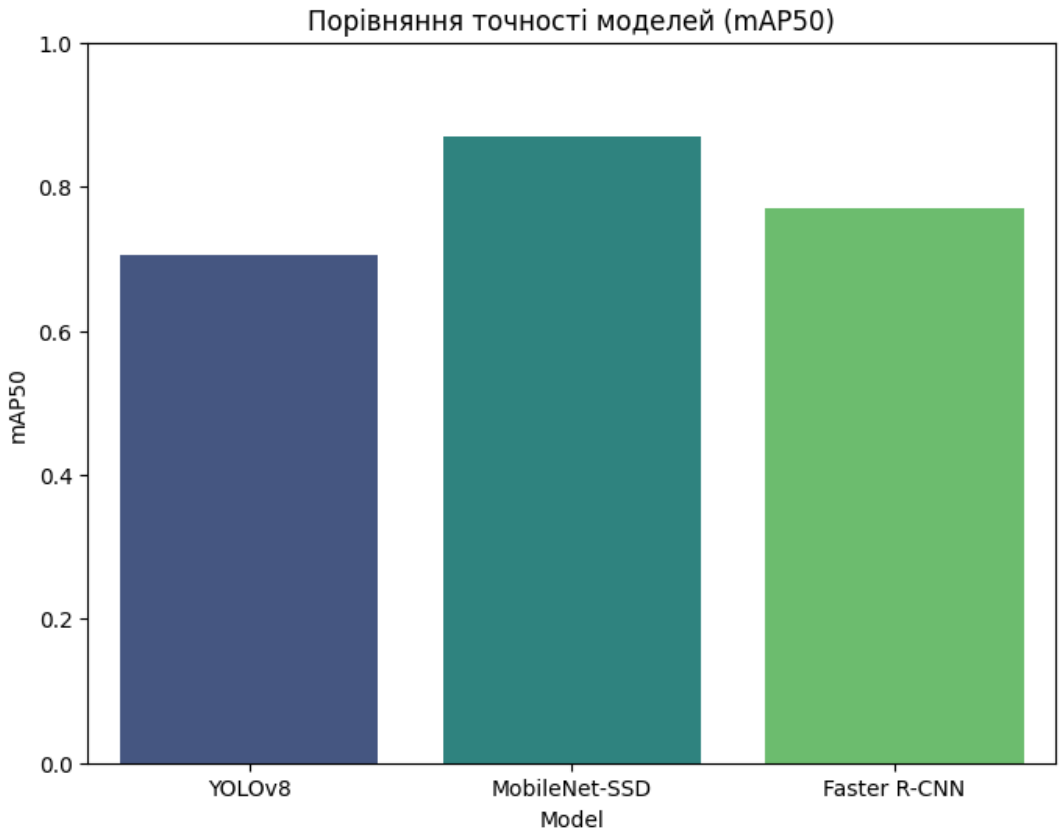


Рисунок 5.11 – Порівняння точності моделей

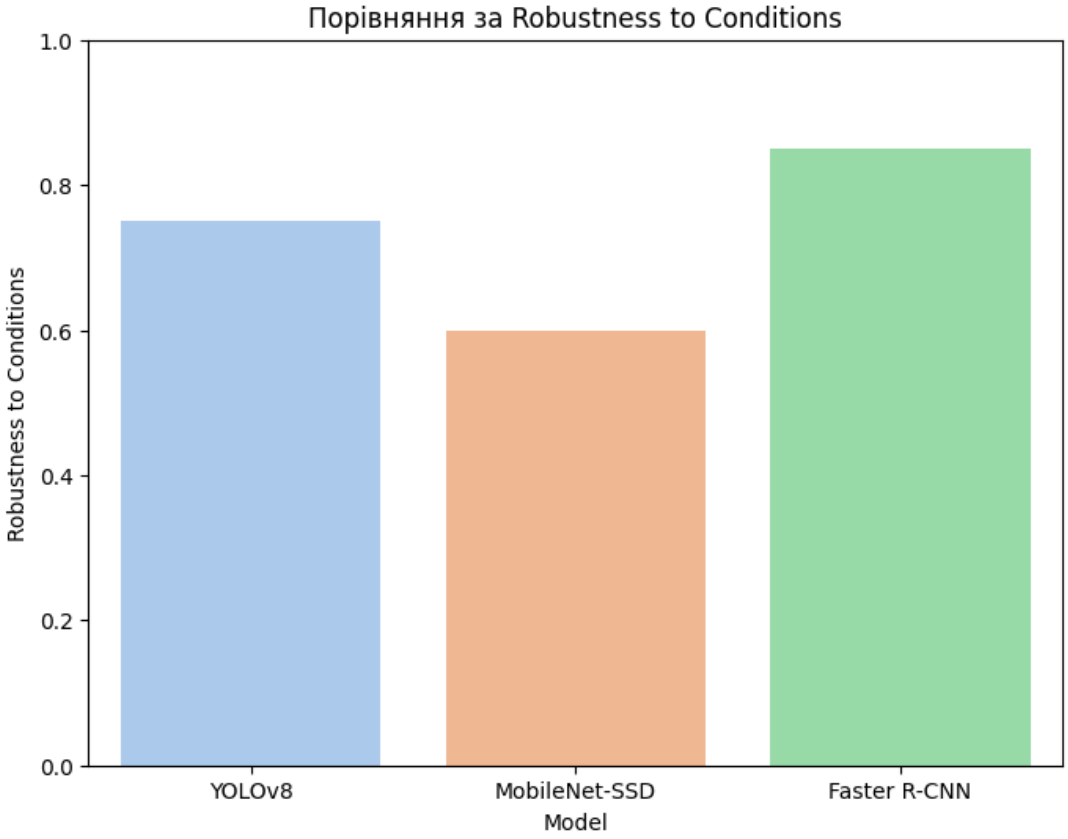


Рисунок 5.12 – Порівняння за Robustness to Conditions

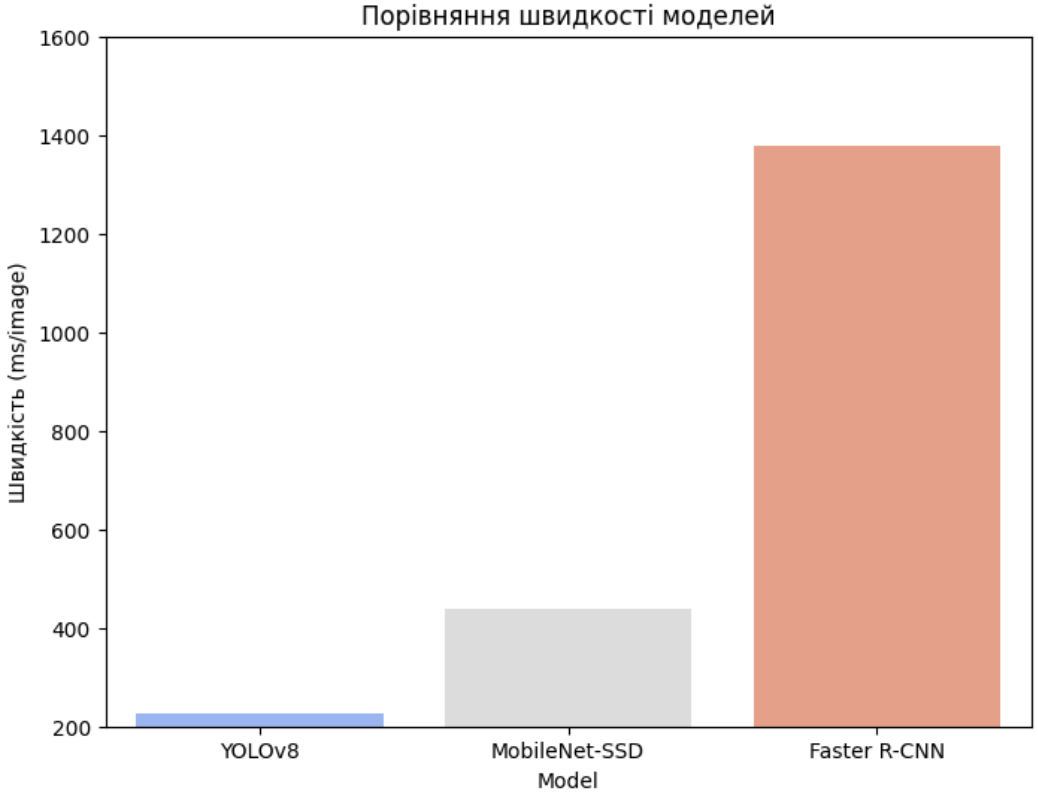


Рисунок 5.13 – Порівняння швидкості моделей

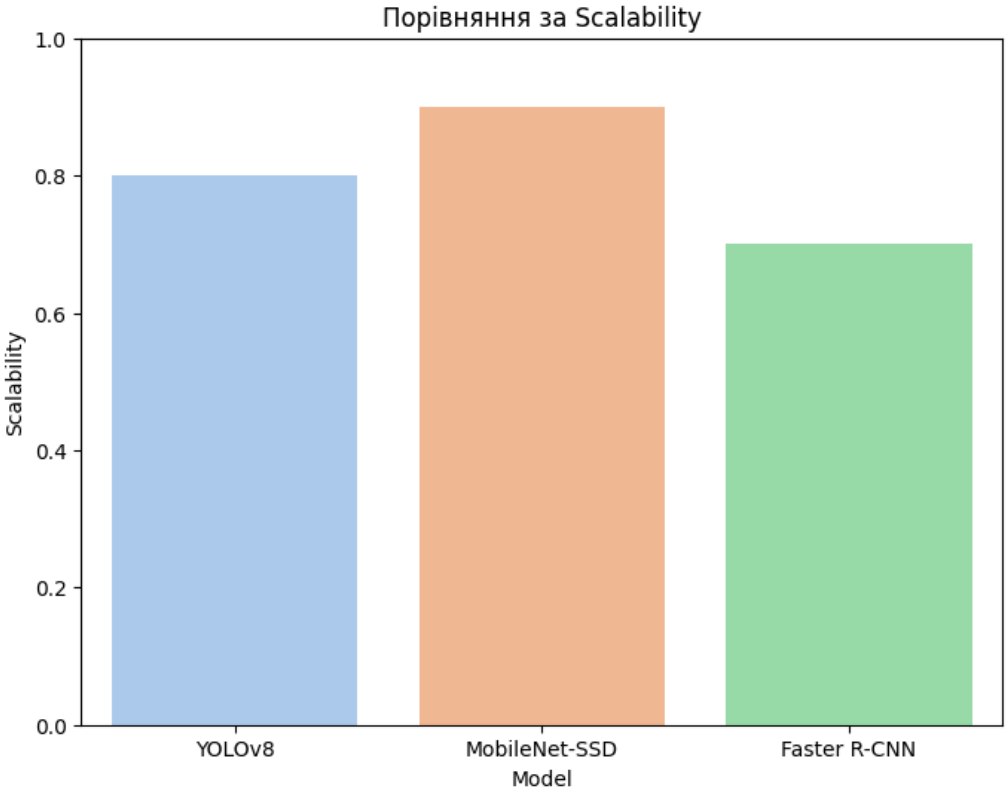


Рисунок 5.14 – Порівняння за Scalability

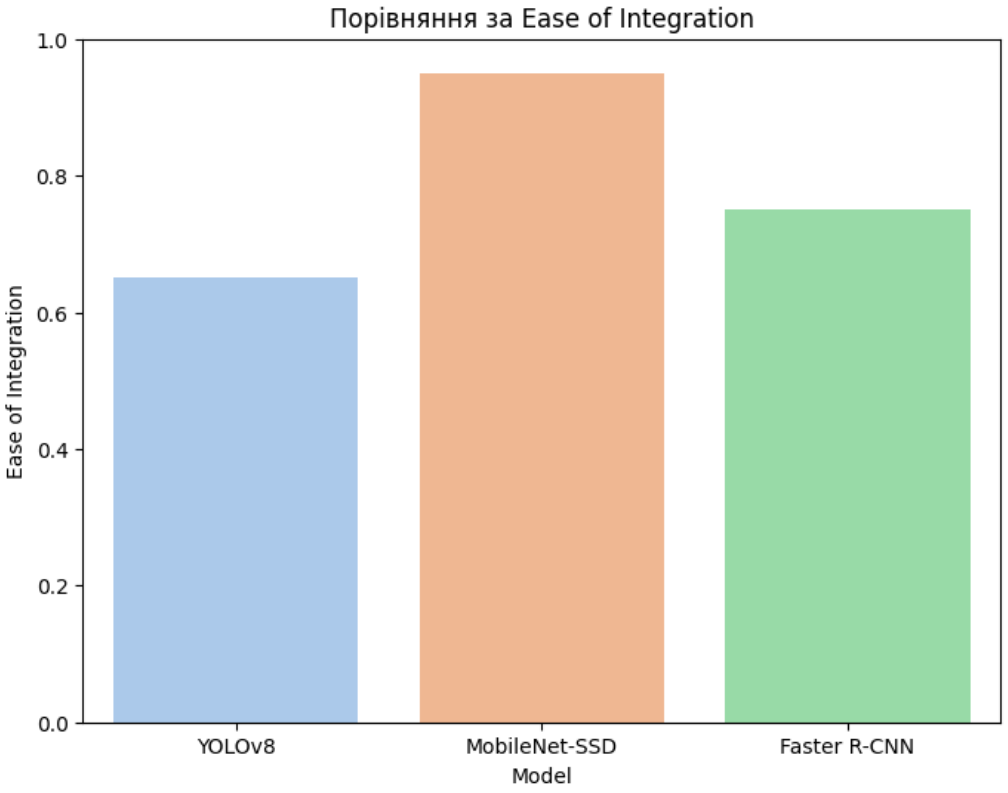


Рисунок 5.15 – Порівняння за легкістю в інтеграції моделі

Аналіз результатів. Точність (mAP): Faster R-CNN показала середній mAP 0.689, що вказує на її здатність до більш детального аналізу зображень, хоча вона поступається YOLOv8. YOLOv8 має mAP 0.559, що вказує на її високу точність у порівнянні з MobileNet-SSD який має mAP 0.819 і є лідером серед усіх.

Час обробки: Найшвидшою виявилась YOLOv8 з середнім часом обробки одного зображення 218.8 мс. MobileNet-SSD потребує більше часу (437.8 мс), проте також може використовуватися для реального часу. Faster R-CNN має значно довший час обробки (1379 мс), що робить її менш підходящою для реальних задач із високими вимогами до продуктивності.

Продуктивність у реальному часі (FPS): YOLOv8 орієнтовно має обробляти близько 4.57 кадрів за секунду, що значно перевищує можливості інших моделей. MobileNet-SSD обробляє 2.28 FPS, тоді як Faster R-CNN досягає лише 0.73 FPS, це звичайно з урахуванням досить слабкого апаратного забезпечення.

Пропущені об'єкти: Faster R-CNN має найменшу кількість пропущених об'єктів (1501), що вказує на її здатність розпізнавати більшу кількість об'єктів. YOLOv8 також демонструє хороший результат з 4586 пропущеними об'єктами. MobileNet-SSD має 4402 пропущених об'єкти.

Проведений аналіз продемонстрував, що кожна модель має свої унікальні переваги та недоліки. Результати таких показників як прогнозований FPS є досить не великими але треба пам'ятати що тестування проводилось з дуже обмеженими апаратними ресурсами а також без використання графічного процесора або різних оптимізаторів що може сильно змінити ситуацію. Тим не менш YOLOv8 вирізняється високою швидкістю обробки кадрів при збереженні прийнятного рівня точності, що робить її ідеальним вибором для застосувань у реальному часі. Завдяки оптимізованій архітектурі та ефективному використанню ресурсів, YOLOv8 може забезпечити плавну роботу навіть на пристроях із середніми обчислювальними можливостями, що критично для задач детекції в реальному часі, де важливим є баланс між точністю та швидкістю.

MobileNet-SSD, натомість, орієнтована на пристрої з обмеженими ресурсами, таких як мобільні телефони або вбудовані системи, дала орієнтовний результат гірший ніж у YOLOv8. Хоча ця модель забезпечує кращу енергоефективність, вона поступається YOLOv8 за рівнем точності, що може бути вирішальним фактором для завдань, де точність є пріоритетом.

Faster R-CNN демонструє найвищу точність серед розглянутих моделей, проте її використання супроводжується значними витратами часу на обробку. Це обмежує її застосування в реальному часі і робить більш придатною для оффлайн-аналізу або систем, де точність є критично важливою і обробка може відбуватися без жорстких обмежень за часом.

Таким чином, вибір YOLOv8 для реалізації детекції об'єктів у реальному часі обґрунтовується її здатністю забезпечувати швидке та ефективне виявлення об'єктів із достатнім рівнем точності, що відповідає вимогам сучасних систем реального часу.

5.5 Аналіз ефективності та швидкості моделей у реальному часі

Однією з найважливіших характеристик для моделей реального часу є кількість кадрів, що обробляються за секунду (FPS). Тестування проводились на власному комп’ютері з такими характеристиками:

- Процесор: QuadCore Intel Core i5-8265U, 3800 MHz (38 x 100);
- Оперативна пам’ять: 32647 Мб (DDR4 SDRAM);
- Відеокарта: інтегрованна;
- Роздільна здатність камери 1080p.

YOLOv8

Результати показали середнє значення в 7 FPS при коді із застосуванням оптимізації, що вище за прогнозовані 4.57 FPS, відчуваються затримки але в цілому працює коректно

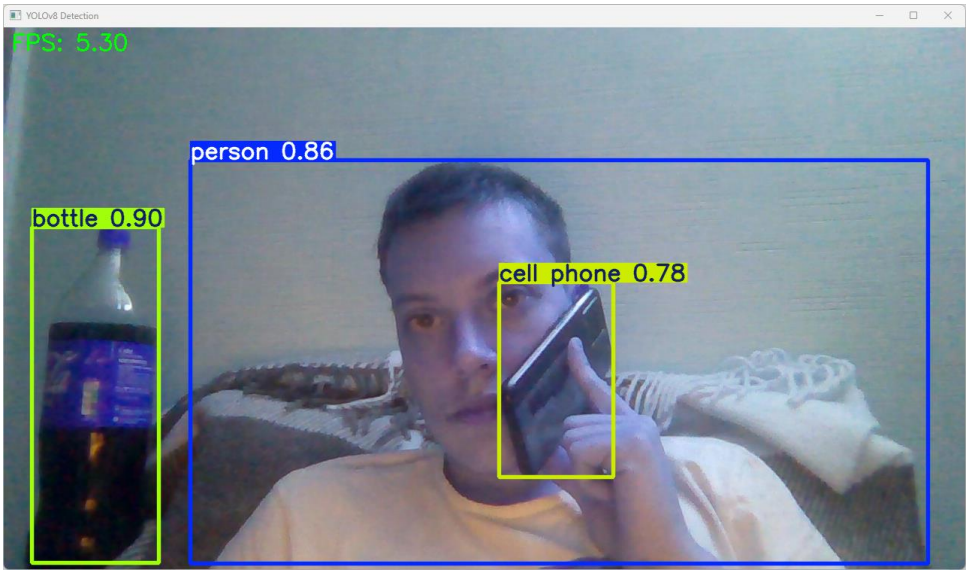


Рисунок 5.16 - Реалізація моделі YOLOv8 в режимі реального часу

MobileNet-SSD

Результати показали середнє значення в 15 FPS, що значно вище за прогнозовані 2.28 FPS, затримки не суттєві, але іноді неправильно розпізнає об’єкти.

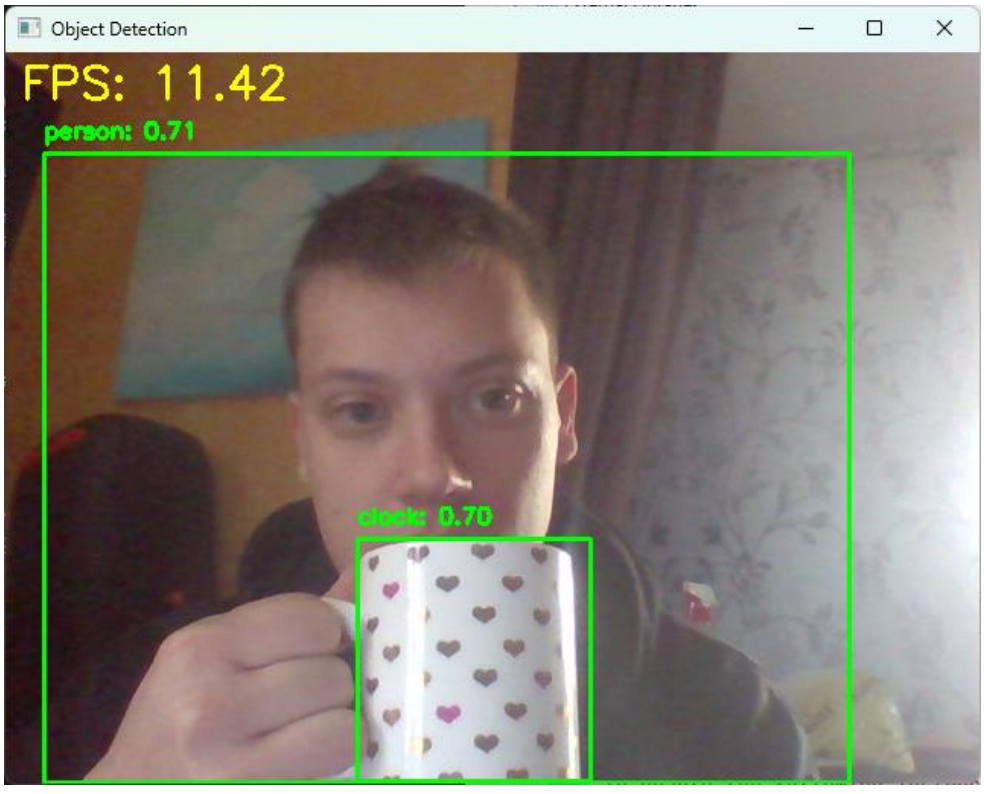


Рисунок 5.17 - Реалізація моделі MobileNet-SSD в режимі реального часу, з помилково визначеною чашкою як годинник

Faster R-CNN

Результати показали середнє значення в 0.2 FPS, що навіть нижче за прогнозовані 0.73 FPS, затримки великі, розпізнає об’єкти добре.

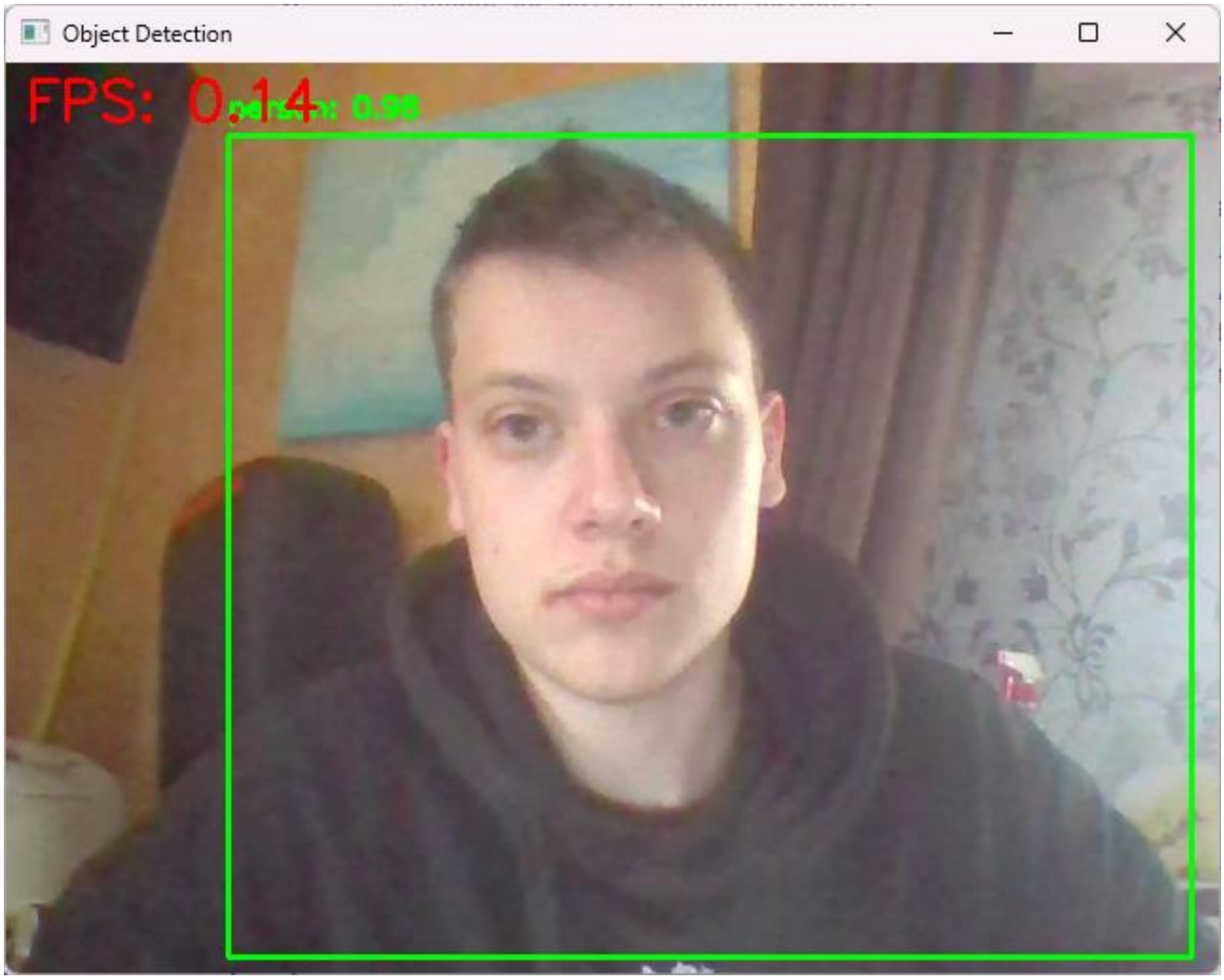


Рисунок 5.18 - Реалізація моделі Faster R-CNN в режимі реального часу

ВИСНОВКИ

В рамках дослідження було проаналізовано три різні моделі для розпізнавання об'єктів у реальному часі: YOLOv8, MobileNet-SSD та Faster R-CNN. Кожна з моделей була піддана детальному тестуванню на різних платформах з різними апаратними ресурсами.

Модель YOLOv8 виявилася найбільш підходящою для реального часу завдяки своїй середньої швидкодії та спроможності обробляти високі роздільності відеостріми на CPU без значних витрат ресурсів. Після оптимізації модель змогла досягнути продуктивності в 7 кадрів на секунду (ККС) на процесорі без використання графічного підсилення, що дозволяє її використання в продуктивних системах без використання графічних процесорів.

Модель MobileNet-SSD показала середні результати щодо точності, але забезпечила найвищу швидкість передачі кадрів. Під час тестування на CPU виявилось, що модель показує гарні результати з невисокими вимогами до ресурсів. Важливо відзначити, що дана модель призначена для роботи на GPU і використовує мінімум апаратних ресурсів, що підтверджує її значну потенційність в прикладних задачах, де важливі швидкодія та точність визначення.

Faster R-CNN демонстрував найнижчу швидкість серед усіх трьох моделей, але має високу точність визначення об'єктів, особливо у складних умовах, таких як приломлення світла через скло. Ця модель має найменшу кількість пропусків визначення об'єктів та забезпечує високу якість результатів, що робить її невід'ємною для задач, що вимагають максимальної точності

Результати дослідження показали, що для задач реального часу на CPU найбільш ефективною є модель YOLOv8, яка забезпечує баланс між швидкістю обробки та точністю розпізнавання об'єктів. MobileNet-SSD також є привабливим вибором для задач з обмеженими ресурсами, хоча її продуктивність може бути покращена при використанні GPU. Faster R-CNN, хоч

і менш ефективна в плані швидкості, є відмінним вибором для задач, де висока точність є пріоритетом.

Загалом, вибір моделі для розпізнавання об'єктів в реальному часі має базуватися на специфічних вимогах до швидкості, точності та доступних обчислювальних ресурсах. Подальші дослідження можуть зосередитися на оптимізації моделей для роботи на різних апаратних платформах та покращенні їх адаптивності до умов реального світу.

Перспективи подальших досліджень у сфері розпізнавання об'єктів охоплюють декілька важливих напрямків. Серед них варто виділити оптимізацію моделей для обмежених ресурсів, що дозволить ефективніше використовувати мобільні та вбудовані системи. Іншим важливим аспектом є адаптація алгоритмів до змінних умов, таких як різні рівні освітлення або перешкоди на зображенні. Важливо також розглянути можливості інтеграції з іншими технологіями, такими як обробка природної мови або робототехніка, для створення комплексних рішень.

Етичні аспекти та безпека відіграють важливу роль, зокрема в контексті захисту персональних даних і розробки методів протидії атакам на системи розпізнавання об'єктів. Подальші дослідження можуть бути зосереджені на розробці нових методів, які дозволять зменшити ризик виникнення помилок у розпізнаванні та забезпечити більшу надійність системи в реальних умовах.

Крім того, важливим напрямком є впровадження технологій машинного навчання у нові галузі, такі як медицина, де розпізнавання об'єктів може сприяти діагностиці захворювань або моніторингу пацієнтів. Дослідження в цьому напрямку можуть включати створення спеціалізованих моделей, які враховують специфіку об'єктів у певних галузях.

Підвищення енергоефективності моделей також є актуальним завданням, що дозволить значно зменшити витрати на обчислювальні ресурси, особливо в умовах зростаючого попиту на мобільні та вбудовані системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Deep Learning for Vision Systems: Mohamed Elgendy. –2020. – 480 с.

2. Ultralytics YOLOv8— Ultralytics. URL: https://docs.ultralytics.com/models/yolov8/#usage-examples

3. Використовуємо CNN для обробки зображень — dou. URL: https://dou.ua/forums/topic/48368/;

4. Convolutional neural network Вікіпедія. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network

5. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions — springer. URL: https://link.springer.com/article/10.1007/s42979-021-00815-1

6. What is R-CNN — Roboflow. URL: https://blog.roboflow.com/what-is-r-cnn/

7. Region Based Convolutional Neural Networks — Вікіпедія. URL: https://en.wikipedia.org/wiki/Region_Based_Convolutional_Neural_Networks

8. Faster R-CNN – pytorch. URL: https://pytorch.org/vision/0.20/models/faster_rcnn.html

9. py-faster-rcnn. URL: https://github.com/rbgirshick/py-faster-rcnn

10. Object Detection Using MobileNet SSD — medium. URL: https://18it107.medium.com/object-detection-using-mobilenet-ssd-8ddf64d5de9a

11. Goodfellow I., Bengio Y., Courville A. Deep Learning. – Cambridge, MA: MIT Press, 2016. – 775 с.

12. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. URL: https://arxiv.org/abs/1506.02640

13. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. URL: https://arxiv.org/abs/1506.01497

14. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.-Y., Berg A. SSD: Single Shot MultiBox Detector. URL: https://arxiv.org/abs/1512.02325

15. Rossum G. Python Programming Language. URL: https://www.python.org

16. Paszke A., Gross S., Massa F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. URL: https://arxiv.org/abs/1912.01703

17. Everingham M., Van Gool L., Williams C. K. I., Winn J., Zisserman A. The Pascal Visual Object Classes (VOC) Challenge. URL: http://host.robots.ox.ac.uk/pascal/VOC/

ДОДАТОК А
КОД ДЛЯ ТЕСТУВАННЯ МОДЕЛЕЙ

yolov8_test.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ]  # Підключення до Google Drive (за потреби)
     from google.colab import drive
     drive.mount('/content/drive')

     # Встановлення бібліотеки Ultralytics для YOLOv8
     !pip install ultralytics
```

```
[ ]  # Завантаження набору даних PASCAL VOC 2012
     !wget http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
     !tar -xvf VOCtrainval_11-May-2012.tar

     # Імпортуємо необхідні бібліотеки
     import os
     import xml.etree.ElementTree as ET

     # Налаштування директорій
     input_dir = "VOCdevkit/VOC2012/Annotations"  # Папка з XML-анотаціями
     output_dir = "VOC_YOLO/labels"               # Папка для анотацій у форматі YOLO
     images_dir = "VOCdevkit/VOC2012/JPEGImages"  # Папка з зображеннями
     os.makedirs(output_dir, exist_ok=True)

     # Список класів PASCAL VOC
     classes = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',
             'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
             'cat', 'dog', 'horse', 'sheep', 'cow']

     # Функція для перетворення координат
     def convert_bbox(size, box):
         dw = 1. / size[0]
         dh = 1. / size[1]
         x = (box[0] + box[1]) / 2.0 - 1
         y = (box[2] + box[3]) / 2.0 - 1
         w = box[1] - box[0]
         h = box[3] - box[2]
         return x * dw, y * dh, w * dw, h * dh

     # Конвертація XML в YOLO
     for xml_file in os.listdir(input_dir):
         if not xml_file.endswith(".xml"):
             continue
         tree = ET.parse(os.path.join(input_dir, xml_file))
         root = tree.getroot()

         # Отримуємо розміри зображення
         size = root.find("size")
         width = int(size.find("width").text)
         height = int(size.find("height").text)

         # Записуємо в вихідний файл
         output_path = os.path.join(output_dir, xml_file.replace(".xml", ".txt"))
         with open(output_path, "w") as out_file:
             for obj in root.iter("object"):
                 class_name = obj.find("name").text
                 if class_name not in classes:
                     continue
                 cls_id = classes.index(class_name)
```

Рисунок А.1 – код тестування yolov8 частина 1

```
[ ]             cls_id = classes.index(class_name)
                bbox = obj.find("bndbox")
                b = (float(bbox.find("xmin").text), float(bbox.find("xmax").text),
                     float(bbox.find("ymin").text), float(bbox.find("ymax").text))
                bbox_converted = convert_bbox((width, height), b)
                out_file.write(f"{cls_id} " + " ".join(map(str, bbox_converted)) + "\n")

    print("Конвертація завершена!")

    # Перевірка структури
    print(f"Анотації збережені в {output_dir}")
    print(f"Кількість файлів у папці анотацій: {len(os.listdir(output_dir))}")
```

```
[ ] from ultralytics import YOLO

    # Завантаження передтренованої моделі YOLOv8
    model = YOLO('yolov8n.pt')  # "n" означає найменшу модель, замініть на 'yolov8m.pt' для більшої
```

```
[ ] # Створення data.yaml для YOLOv8
    yaml_content = f"""
    path: /content/YOLO_VOC  # Головна папка з даними
    train: /content/YOLO_VOC/images/train  # Підпапка з тренувальними зображеннями
    val: /content/YOLO_VOC/images/val      # Підпапка з валідаційними зображеннями

    nc: 20
    names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',
            'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
            'cat', 'dog', 'horse', 'sheep', 'cow']
    """

    # Записуємо у файл
    with open("VOC_YOLO/data.yaml", "w") as yaml_file:
        yaml_file.write(yaml_content)

    print("Файл data.yaml створено!")
```

```
[ ] import os
    import shutil
    from sklearn.model_selection import train_test_split

    # Шляхи до папок
    images_dir = '/content/VOCdevkit/VOC2012/JPEGImages'  # Папка з зображеннями
    annotations_dir = '/content/VOC_YOLO/labels'  # Папка з анотаціями
    train_images_dir = '/content/YOLO_VOC/images/val'  # Папка для тренувальних зображень
    val_images_dir = '/content/YOLO_VOC/images/val'  # Папка для валідаційних зображень
    train_labels_dir = '/content/YOLO_VOC/labels/val'  # Папка для тренувальних анотацій
    val_labels_dir = '/content/YOLO_VOC/labels/val'  # Папка для валідаційних анотацій

    # Створення папок для тренування та валідації
    os.makedirs(train_images_dir, exist_ok=True)
    os.makedirs(val_images_dir, exist_ok=True)
    os.makedirs(train_labels_dir, exist_ok=True)
```

Рисунок А.2 – код тестування yolov8 частина 2

```
[ ] os.makedirs(train_labels_dir, exist_ok=True)
    os.makedirs(val_labels_dir, exist_ok=True)

    # Отримання списку всіх зображень
    images = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]

    # Розподіл зображень на тренувальний та валідаційний набори
    train_images, val_images = train_test_split(images, test_size=0.2, random_state=42)

    # Переміщення зображень у відповідні папки
    for image in train_images:
        shutil.move(os.path.join(images_dir, image), os.path.join(train_images_dir, image))
        # Переміщення відповідного файлу анотації
        annotation_file = image.replace('.jpg', '.txt')  # Якщо анотації в XML форматі
        shutil.move(os.path.join(annotations_dir, annotation_file), os.path.join(train_labels_dir, annotation_file))

    for image in val_images:
        shutil.move(os.path.join(images_dir, image), os.path.join(val_images_dir, image))
        # Переміщення відповідного файлу анотації
        annotation_file = image.replace('.jpg', '.txt')  # Якщо анотації в XML форматі
        shutil.move(os.path.join(annotations_dir, annotation_file), os.path.join(val_labels_dir, annotation_file))

    print("Розподіл зображень і анотацій завершено!")
```

```
[ ] # Тестування моделі на тестовому наборі даних
    results = model.val(data="./VOC_YOLO/data.yaml")
```

```
Ultralytics 8.3.55 🚀 Python-3.10.12 torch-2.5.1+cu121 CPU (Intel Xeon 2.20GHz)
YOLOv8n summary (fused): 168 layers, 3,151,904 parameters, 0 gradients, 8.7 GFLOPs
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|██████████| 755k/755k [00:00<00:00, 19.7MB/s]
val: Scanning /content/YOLO_VOC/labels/val... 17125 images, 2165 backgrounds, 0 corrupt: 100%|██████████| 17125/17125 [00:10<00:00, 1648.96it/s]
val: New cache created: /content/YOLO_VOC/labels/val.cache
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1071/1071 [1:07:20<00:00,  3.77s/it]
                   all      17125      29982      0.742      0.706      0.729      0.559
                person       9583      17401      0.637      0.841      0.757      0.581
               bicycle        603        837      0.518      0.675      0.541      0.409
                   car       1284       2492      0.612      0.618      0.604      0.433
                   bus        467        685       0.86      0.803      0.849      0.726
                 train        589        704      0.861      0.791      0.871      0.685
                  boat        549       1059      0.744      0.424      0.565      0.358
                  bird        811       1271       0.86      0.621      0.752      0.534
                   cat       1128       1277      0.864      0.814      0.882      0.716
                   dog       1341       1598      0.822      0.717      0.816      0.651
                 horse        526        803      0.459       0.78        0.5      0.401
                 sheep        357       1084       0.84      0.642      0.778      0.582
                   cow        340        771      0.829      0.744       0.83      0.628
Speed: 5.4ms preprocess, 218.8ms inference, 0.0ms loss, 2.9ms postprocess per image
Results saved to runs/detect/val
```

Рисунок А.3 – код тестування yolov8 частина 3

MobileNet-SSD.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
[ ] # Завантаження набору даних PASCAL VOC 2012
    !wget http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
    !tar -xvf VOCtrainval_11-May-2012.tar
```

```
# Встановлення необхідних бібліотек
!pip install tensorflow tensorflow-hub tf-slim opencv-python-headless matplotlib lxml

import tensorflow as tf
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET
from google.colab import drive

# Підключення Google Drive
drive.mount('/content/drive')

# Завантаження моделі MobileNet-SSD, навченої на COCO
!wget -q -O model.tar.gz http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz
!tar -xzf model.tar.gz
MODEL_DIR = "./ssd_mobilenet_v2_coco_2018_03_29/saved_model"
model = tf.saved_model.load(MODEL_DIR)

# Мапінг класів VOC у класи COCO
voc_to_coco_map = {
    'person': 'person',
    'bicycle': 'bicycle',
    'car': 'car',
    'motorbike': 'motorcycle',
    'aeroplane': 'airplane',
    'bus': 'bus',
    'train': 'train',
    'boat': 'boat',
    'bird': 'bird',
    'cat': 'cat',
    'dog': 'dog',
    'horse': 'horse',
    'sheep': 'sheep',
    'cow': 'cow',
    'bottle': 'bottle',
    'chair': 'chair',
    'sofa': 'couch',
    'pottedplant': 'potted plant',
    'tvmonitor': 'tv'
}

# Функція для завантаження зображення
def load_image_into_numpy_array(path):
    img = cv2.imread(path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return np.array(img_rgb)

# Функція для отримання справжнього класу із анотацій
def get_true_labels(annotation_path):
```

Рисунок А.4 – код тестування MobileNet-SSD частина 1

```python
def get_true_labels(annotation_path):
    tree = ET.parse(annotation_path)
    root = tree.getroot()
    labels = []
    for obj in root.findall('object'):
        label = obj.find('name').text
        if label in voc_to_coco_map:
            labels.append(voc_to_coco_map[label])
    return labels

# Функція для виконання детекції
def detect_objects(image_np, model):
    input_tensor = tf.convert_to_tensor(image_np)[tf.newaxis, ...]
    detections = model.signatures['serving_default'](input_tensor)
    detection_boxes = detections['detection_boxes']
    detection_scores = detections['detection_scores']
    detection_classes = detections['detection_classes']
    return {
        'detection_boxes': detection_boxes[0],
        'detection_scores': detection_scores[0],
        'detection_classes': detection_classes[0]
    }

# Функція для візуалізації детекцій
def visualize_detections(image_np, detections, category_index, threshold=0.5):
    plt.figure(figsize=(12, 8))
    plt.imshow(image_np)
    ax = plt.gca()

    height, width, _ = image_np.shape

    for i in range(len(detections['detection_scores'])):
        score = detections['detection_scores'][i].numpy()
        if score >= threshold:
            bbox = detections['detection_boxes'][i].numpy()
            ymin, xmin, ymax, xmax = bbox
            (left, right, top, bottom) = (xmin * width, xmax * width, ymin * height, ymax * height)
            class_id = int(detections['detection_classes'][i].numpy())
            class_name = category_index.get(class_id, 'N/A')

            # Відображення рамок
            rect = plt.Rectangle((left, top), right - left, bottom - top, fill=False, color='red', linewidth=2)
            ax.add_patch(rect)
            ax.text(left, top - 10, f'{class_name}: {score:.2f}', color='red', fontsize=10, weight='bold')

    plt.axis('off')
    plt.show()

# Визначення шляхів до зображень та анотацій
dataset_path = "/content/VOCdevkit/VOC2012/JPEGImages"
annotations_path = "/content/VOCdevkit/VOC2012/Annotations"
category_index = {
    1: 'person',
    2: 'bicycle',
    3: 'car',
    4: 'motorcycle',
    5: 'airplane',
    6: 'bus',
```

Рисунок А.5 – код тестування MobileNet-SSD частина 2

```
  6: 'bus',
  7: 'train',
  8: 'truck',
  9: 'boat',
  10: 'traffic light',
  11: 'fire hydrant',
  13: 'stop sign',
  14: 'parking meter',
  15: 'bench',
  16: 'bird',
  17: 'cat',
  18: 'dog',
  19: 'horse',
  20: 'sheep',
  21: 'cow',
  22: 'elephant',
  23: 'bear',
  24: 'zebra',
  25: 'giraffe',
  27: 'backpack',
  28: 'umbrella',
  31: 'handbag',
  32: 'tie',
  33: 'suitcase',
  34: 'frisbee',
  35: 'skis',
  36: 'snowboard',
  37: 'sports ball',
  38: 'kite',
  39: 'baseball bat',
  40: 'baseball glove',
  41: 'skateboard',
  42: 'surfboard',
  43: 'tennis racket',
  44: 'bottle',
  46: 'wine glass',
  47: 'cup',
  48: 'fork',
  49: 'knife',
  50: 'spoon',
  51: 'bowl',
  52: 'banana',
  53: 'apple',
  54: 'sandwich',
  55: 'orange',
  56: 'broccoli',
  57: 'carrot',
  58: 'hot dog',
  59: 'pizza',
  60: 'donut',
  61: 'cake',
  62: 'chair',
  63: 'couch',
  64: 'potted plant',
  65: 'bed',
  67: 'dining table',
  70: 'toilet',
  72: 'tv',
```

Рисунок А.6 – код тестування MobileNet-SSD частина 3

```
    72: 'tv',
    73: 'laptop',
    74: 'mouse',
    75: 'remote',
    76: 'keyboard',
    77: 'cell phone',
    84: 'book',
    85: 'clock',
    86: 'vase',
    87: 'scissors',
    88: 'teddy bear',
    89: 'hair drier',
    90: 'toothbrush'
}

# Обробка зображень та розрахунок точності
images = [img for img in os.listdir(dataset_path) if img.endswith(('.jpg', '.png'))]
accuracy_results = {}

for img_name in images:
    image_path = os.path.join(dataset_path, img_name)
    annotation_path = os.path.join(annotations_path, img_name.replace('.jpg', '.xml'))

    if not os.path.exists(annotation_path):
        continue

    image_np = load_image_into_numpy_array(image_path)
    detections = detect_objects(image_np, model)

    true_labels = get_true_labels(annotation_path)
    detected_labels = [category_index.get(int(cls.numpy()), 'N/A') for cls in detections['detection_classes']]

    for label in true_labels:
        if label not in accuracy_results:
            accuracy_results[label] = {'correct': 0, 'total': 0}
        accuracy_results[label]['total'] += 1
        if label in detected_labels:
            accuracy_results[label]['correct'] += 1

# Розрахунок загальної точності та виведення результатів
print("\nРезультати точності моделі за класами:")
for label, stats in accuracy_results.items():
    accuracy = stats['correct'] / stats['total'] * 100
    print(f"Клас: {label}, Точність: {accuracy:.2f}%, Правильно: {stats['correct']}, Всього: {stats['total']}")

# Візуалізація детекцій для прикладів
sample_images = images[:5]
for img_name in sample_images:
    image_path = os.path.join(dataset_path, img_name)
    image_np = load_image_into_numpy_array(image_path)
    detections = detect_objects(image_np, model)
    visualize_detections(image_np, detections, category_index)
```

Рисунок А.7 – код тестування MobileNet-SSD частина 4

Faster R-CNN.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
# Встановлення необхідних бібліотек
!pip install torch torchvision matplotlib tqdm

# Завантаження набору даних PASCAL VOC 2012
!wget http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
!tar -xvf VOCtrainval_11-May-2012.tar
```

```
[ ] # **Імпорт необхідних бібліотек**
    import os
    import random
    import xml.etree.ElementTree as ET
    import torch
    from torchvision import models, transforms
    from torchvision.datasets import VOCDetection
    from torchvision.ops import box_iou
    import matplotlib.pyplot as plt
    from PIL import Image
```

```
[ ] # **Налаштування шляху до анотацій та зображень**
    annotations_path = "/content/VOCdevkit/VOC2012/Annotations"
    images_path = "/content/VOCdevkit/VOC2012/JPEGImages"

    # **Параметри для вибірки даних та налаштування фільтрації**
    num_images_to_process = 3000  # Кількість зображень для обробки 17125
    confidence_threshold = 0.7  # Поріг впевненості
    visualize_every = 100  # Візуалізувати одне зображення на кожні N
```

```
[ ] # **Мапінг назв класів для відповідності між наборами COCO і VOC**
    class_mapping = {
        "aeroplane": "airplane",
        "bicycle": "bicycle",
        "bird": "bird",
        "boat": "boat",
        "bottle": "bottle",
        "bus": "bus",
        "car": "car",
        "cat": "cat",
        "chair": "chair",
        "cow": "cow",
        "diningtable": "dining_table",
        "dog": "dog",
        "horse": "horse",
        "motorbike": "motorcycle",
        "person": "person",
        "pottedplant": "potted_plant",
        "sheep": "sheep",
        "sofa": "couch",  # COCO використовує "couch" замість "sofa"
        "train": "train",
        "tvmonitor": "tv",  # COCO використовує "tv" замість "tvmonitor"
    }

    # **Функція для перетворення анотацій у відповідний формат**
    def process_annotations(annotations_dir, image_name):
```

Рисунок А.8 – код тестування Faster R-CNN частина 1

```python
def process_annotations(annotations_dir, image_name):
    annotation_file = os.path.join(annotations_dir, image_name.replace(".jpg", ".xml"))
    tree = ET.parse(annotation_file)
    root = tree.getroot()
    objects = []
    for obj in root.findall("object"):
        class_name = obj.find("name").text
        if class_name in class_mapping:
            class_name = class_mapping[class_name]
        bbox = obj.find("bndbox")
        box = [
            int(float(bbox.find("xmin").text)),  # Convert to float, then to int
            int(float(bbox.find("ymin").text)),  # Convert to float, then to int
            int(float(bbox.find("xmax").text)),  # Convert to float, then to int
            int(float(bbox.find("ymax").text))   # Convert to float, then to int
        ]
        objects.append({"class": class_name, "bbox": box})
    return objects

# **Функція для підрахунку mAP та точності**
def evaluate_model(detections, annotations, classes):
    class_stats = {cls: {"total": 0, "correct": 0} for cls in classes}
    iou_threshold = 0.5

    for image_name, detected_objects in detections.items():
        true_objects = annotations[image_name]
        used_true_objects = set()

        for obj in true_objects:
            class_stats[obj["class"]]["total"] += 1

        for detected_obj in detected_objects:
            detected_class = detected_obj["class"]
            if detected_class in class_mapping:
                detected_class = class_mapping[detected_class]

            best_iou = 0
            best_match = None

            for idx, true_obj in enumerate(true_objects):
                if true_obj["class"] == detected_class and idx not in used_true_objects:
                    iou = box_iou(torch.tensor([detected_obj["bbox"]]), torch.tensor([true_obj["bbox"]]))[0][0].item()
                    if iou > best_iou:
                        best_iou = iou
                        best_match = idx

            if best_iou > iou_threshold and best_match is not None:
                class_stats[detected_class]["correct"] += 1
                used_true_objects.add(best_match)

    total_correct = sum(stat["correct"] for stat in class_stats.values())
    total_objects = sum(stat["total"] for stat in class_stats.values())
    accuracy = total_correct / total_objects if total_objects > 0 else 0

    # Обчислення mAP
    ap_per_class = []
    for cls, stats in class_stats.items():
        precision = stats["correct"] / stats["total"] if stats["total"] > 0 else 0
```

Рисунок А.9 – код тестування Faster R-CNN частина 2

```
]           precision = stats["correct"] / stats["total"] if stats["total"] > 0 else 0
            ap_per_class.append(precision)
        mAP = sum(ap_per_class) / len(ap_per_class) if ap_per_class else 0

        return class_stats, accuracy, mAP

    # **Функція для візуалізації детекцій**
    def visualize_detections(image_path, detections, annotations):
        image = Image.open(image_path)
        plt.imshow(image)
        ax = plt.gca()

        for det in detections:
            bbox = det["bbox"]
            rect = plt.Rectangle((bbox[0], bbox[1]), bbox[2] - bbox[0], bbox[3] - bbox[1],
                                 linewidth=2, edgecolor="r", facecolor="none")
            ax.add_patch(rect)
            ax.text(bbox[0], bbox[1], det["class"], color="red", fontsize=10)

        for ann in annotations:
            bbox = ann["bbox"]
            rect = plt.Rectangle((bbox[0], bbox[1]), bbox[2] - bbox[0], bbox[3] - bbox[1],
                                 linewidth=2, edgecolor="g", facecolor="none")
            ax.add_patch(rect)
            ax.text(bbox[0], bbox[1], ann["class"], color="green", fontsize=10)

        plt.show()

    # **Завантаження моделі Faster R-CNN**
    from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
    weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    model = models.detection.fasterrcnn_resnet50_fpn(weights=weights)
    model.eval()

    # **Обробка зображень**
    image_names = random.sample(os.listdir(images_path), num_images_to_process)
    transform = transforms.Compose([
        transforms.ToTensor(),
    ])

    detections = {}
    annotations = {}
    for i, image_name in enumerate(image_names):
        image_path = os.path.join(images_path, image_name)
        image = Image.open(image_path).convert("RGB")
        input_tensor = transform(image).unsqueeze(0)

        outputs = model(input_tensor)[0]

        # Фільтрація за confidence_threshold
        filtered_detections = [
            {"class": weights.meta["categories"][label], "bbox": box.tolist()}
            for box, label, score in zip(outputs["boxes"], outputs["labels"], outputs["scores"])
            if score > confidence_threshold
        ]

        detections[image_name] = filtered_detections
        annotations[image_name] = process_annotations(annotations_path, image_name)
```

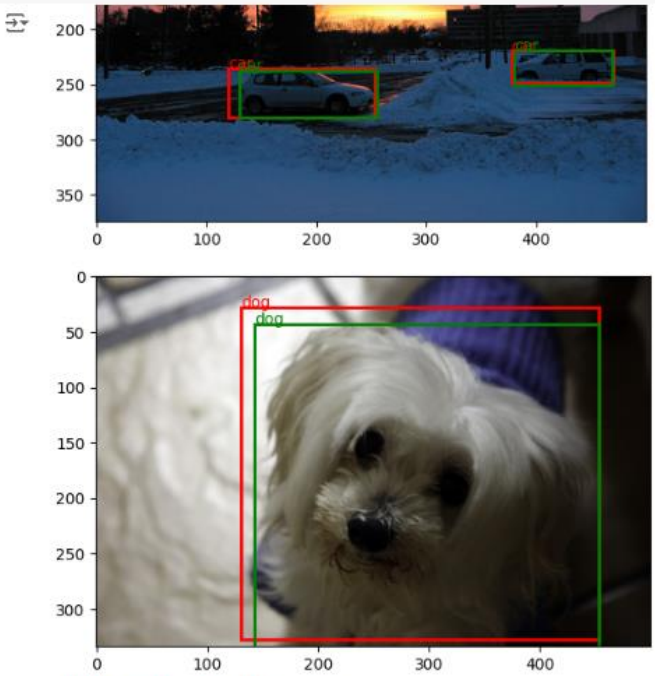
Рисунок А.10 – код тестування Faster R-CNN частина 3

```
[ ]     annotations[image_name] = process_annotations(annotations_path, image_name)

        if (i + 1) % visualize_every == 0:
            visualize_detections(image_path, filtered_detections, annotations[image_name])

    # **Оцінка моделі**
    classes = list(set(cls["class"] for ann in annotations.values() for cls in ann))
    class_stats, accuracy, mAP = evaluate_model(detections, annotations, classes)

    print("Точність моделі:", accuracy)
    print("mAP:", mAP)
    for cls, stats in class_stats.items():
        print(f"Клас {cls}: {stats['correct']}/{stats['total']} (точність: {stats['correct'] / stats['total'] if stats['total'] > 0 else 0:.2f})")
```




```
Точність моделі: 0.7793296089385475
mAP: 0.6889157688571518
Клас car: 344/477 (точність: 0.72)
Клас potted_plant: 0/182 (точність: 0.00)
Клас bus: 95/114 (точність: 0.83)
Клас airplane: 141/164 (точність: 0.86)
Клас tv: 132/154 (точність: 0.86)
Клас bird: 160/186 (точність: 0.86)
Клас train: 104/127 (точність: 0.82)
Клас dog: 204/258 (точність: 0.79)
Клас chair: 314/534 (точність: 0.59)
```

Рисунок А.11 – код тестування Faster R-CNN частина 4

ДОДАТОК Б
КОД ДЛЯ ТЕСТУВАННЯ МОДЕЛЕЙ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

```python
import cv2
import time
import torch
from ultralytics import YOLO

# Перевірка наявності GPU
device = "cuda" if torch.cuda.is_available() else "cpu"

# Завантаження моделі
model = YOLO('yolov8n.pt').to(device)  # Не використовуємо .half()

# Відкриття доступу до вебкамери
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1980) #320
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1024) #240

while True:
    start_time = time.time()
    ret, frame = cap.read()
    if not ret:
        break
    # Перетворення кадру до формату RGB (YOLO очікує RGB, а OpenCV дає BGR)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Виконання детекції
    results = model(frame)

    # Візуалізація результатів
    annotated_frame = results[0].plot()

    # Обчислення FPS
    fps = 1 / (time.time() - start_time)
    cv2.putText(annotated_frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    # Показ кадру
    cv2.imshow("YOLOv8 Detection", annotated_frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Рисунок Б.1 – код тестування YOLOv8 в режимі реального часу

```python
import cv2
import numpy as np
import tensorflow as tf
import time

import requests
import tarfile

# Завантаження моделі
url = "http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz"
response = requests.get(url, stream=True)
with open("model.tar.gz", "wb") as f:
    for chunk in response.iter_content(chunk_size=1024):
        f.write(chunk)

# Розпакування архіву
with tarfile.open("model.tar.gz", "r:gz") as tar:
    tar.extractall()
# Завантаження моделі

MODEL_DIR = "ssd_mobilenet_v2_coco_2018_03_29/saved_model"
detection_model = tf.saved_model.load(MODEL_DIR)
# Імена ключів для роботи з моделлю
infer = detection_model.signatures["serving_default"]
# Функція для виконання детекції
1 usage
def detect_objects(image):
    input_tensor = tf.convert_to_tensor(image)
    input_tensor = input_tensor[tf.newaxis, ...]  # Додаємо batch-дименсію
    detections = infer(input_tensor)  # Викликаємо модель

    # Витягуємо результати
    boxes = detections["detection_boxes"].numpy()
    classes = detections["detection_classes"].numpy().astype(np.int32)
    scores = detections["detection_scores"].numpy()

    return boxes, classes, scores

# Ініціалізація камери
cap = cv2.VideoCapture(0)

# Перевірка доступу до камери
```

Рисунок Б.2 – код тестування MobileNet-SSD в режимі реального часу, частина 1

```
# Перевірка доступу до камери
if not cap.isOpened():
    print("Не вдалося отримати доступ до веб-камери.")
    exit()

# Завантаження назв класів COCO
COCO_CLASSES = ["background"] + [
    "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck", "boat",
    "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat",
    "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack",
    "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball",
    "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", "tennis racket",
    "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", "apple",
    "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
    "chair", "couch", "potted plant", "bed", "dining table", "toilet", "TV", "laptop",
    "mouse", "remote", "keyboard", "cell phone", "microwave", "oven", "toaster",
    "sink", "refrigerator", "book", "clock", "vase", "scissors", "teddy bear",
    "hair drier", "toothbrush"
]

# Основний цикл обробки
while True:
    ret, frame = cap.read()
    if not ret:
        print("Не вдалося отримати кадр з камери.")
        break

    start_time = time.time()

    # Змінення розміру зображення для моделі
    input_image = cv2.resize(frame, (300, 300))
    input_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB)

    # Виконання детекції
    boxes, classes, scores = detect_objects(input_image)

    # Візуалізація детекцій
    height, width, _ = frame.shape
    for box, cls, score in zip(boxes[0], classes[0], scores[0]):  # [0] тому що batch = 1
        if score > 0.5:  # Фільтруємо низьку впевненість
            print(f"Class: {cls}, Score: {score}, Box: {box}")
            ymin, xmin, ymax, xmax = box
```

Рисунок Б.3 – код тестування MobileNet-SSD в режимі реального часу, частина 2

```
            ymin, xmin, ymax, xmax = box
            (left, top, right, bottom) = (xmin * width, ymin * height, xmax * width, ymax * height)

            # Малюємо прямокутник
            cv2.rectangle(frame, (int(left), int(top)), (int(right), int(bottom)), (0, 255, 0), 2)

            # Додаємо текст з назвою класу та впевненістю
            label = f"{COCO_CLASSES[cls]}: {score:.2f}"
            cv2.putText(frame, label, (int(left), int(top) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Обрахунок FPS
    fps = 1.0 / (time.time() - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

    # Показ зображення
    cv2.imshow('Object Detection', frame)

    # Вихід при натисканні 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Звільнення ресурсів
cap.release()
cv2.destroyAllWindows()
```

Рисунок Б.4 – код тестування MobileNet-SSD в режимі реального часу, частина 3

```python
import cv2
import torch
from torchvision import models, transforms
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
import time

# Завантаження моделі Faster R-CNN
weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
model = models.detection.fasterrcnn_resnet50_fpn(weights=weights)
model.eval()

# Класи об'єктів з COCO-датасету
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck',
    'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat',
    'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
    'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
    'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog',
    'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv',
    'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink',
    'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Налаштування камери
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Не вдалося отримати доступ до камери.")
    exit()

# Перетворення зображень
transform = transforms.Compose([
    transforms.ToTensor()
])

while True:
    start_time = time.time()

    # Захоплення кадру з камери
    ret, frame = cap.read()
    if not ret:
        print("Не вдалося захопити кадр.")
```

Рисунок Б.5 – код тестування Fsater R-CNN в режимі реального часу, частина 1

```
        print("Не вдалося захопити кадр.")
        break

    # Перетворення кадру для моделі
    image_tensor = transform(frame).unsqueeze(0)

    # Інференс моделі
    with torch.no_grad():
        predictions = model(image_tensor)[0]

    # Обробка результатів
    boxes = predictions['boxes'].cpu().numpy()
    scores = predictions['scores'].cpu().numpy()
    labels = predictions['labels'].cpu().numpy()

    # Фільтрація за порогом впевненості
    confidence_threshold = 0.5
    for box, score, label in zip(boxes, scores, labels):
        if score > confidence_threshold:
            x1, y1, x2, y2 = map(int, box)
            label_name = COCO_INSTANCE_CATEGORY_NAMES[label]

            # Відображення рамки та тексту
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, f"{label_name}: {score:.2f}", (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Обчислення FPS
    fps = 1 / (time.time() - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    # Відображення результату
    cv2.imshow("Object Detection", frame)

    # Вихід за клавішею 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Завершення роботи
cap.release()
cv2.destroyAllWindows()
```

Рисунок Б.6 – код тестування Fsater R-CNN в режимі реального часу, частина 2