МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет економіки і підприємництва

Кафедра інформаційних технологій

Допущено до захисту
«_____» _____ 2025 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ІНФОРМАЦІЙНОЇ ПІДСИСТЕМИ ДЛЯ ПІДРАХУНКУ НАЯВНОСТІ ТОВАРУ

Виконав – студент 2 курсу, групи 22к-кн
освітньої програми «Комп’ютерні науки»
спеціальності 122 Комп’ютерні науки
Левицький Микола Сергійович

Науковий керівник – д.т.н., професор
Кучерук Володимир Юрійович

Умань – 2025

Зміст

Вступ 4
РОЗДІЛ 1 Дослідження технологій розпізнавання об’єктів 5
1.1. Огляд сучасних програмних засобів підрахунку товарів. 5
1.2. Аналіз архітектур нейромереж. 8
1.3. Постановка задачі. 20
РОЗДІЛ 2 Огляд програмних засобів для виконання завдання 23
2.1.Аналіз середовища розробки. 23
2.2. Архітектура YOLO. 28
2.3. Візуальні алгоритми штучного інтелекту YOLO. 32
РОЗДІЛ 3 Підготовка даних та виконання завдання 35
3.1. Підготовка середовища та даних для нейронної мережі YOLO. 35
3.2. Тренування нейронної мережі YOLO. 39
3.3. Тестування підсистеми для підрахунку наявності товару. 43
Висновок 49
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 51
Додаток А ..... **Помилка! Закладку не визначено.**

Перелік скорочень

YOLO – You Only Look Once

CNN – Convolution Neural Network

GPU – Graphics Processing Unit

RNN – Recurrent Neural Networks

SVM – Support Vector Machine

ШНМ – Штучна нейронна мережа

ANN – Artificial Neural Network

DSP – Digital signal processor

FPGA – Field programmable gate array

AWR- Advantage-Weighted Regression

A2C- Advantage Actor-Critic

RL-Алгоритм - Reinforcement Learning

ВСТУП

У сучасних умовах автоматизація бізнес-процесів є ключовим чинником підвищення ефективності підприємств. Особливої актуальності набуває автоматизація обліку товарів, що дозволяє мінімізувати людський фактор, зменшити витрати часу та уникнути помилок при інвентаризації. Застосування технологій комп'ютерного зору та штучного інтелекту у сфері обліку відкриває нові можливості для точного та оперативного підрахунку товарно-матеріальних цінностей.

Мета дослідження – розробку інформаційної підсистеми, що забезпечує автоматизований підрахунок наявного товару на основі відео або зображень з використанням алгоритму YOLO,з метою підвищення ефективності товарного залишку.

Об'єктом дослідження кваліфікаційної роботи є процес обліку товарно-матеріальних цінностей на підприємстві із застосуванням систем комп'ютерного зору.

Предметом дослідження є методи автоматичного розпізнавання та підрахунку товарів за допомогою алгоритмів машинного навчання, зокрема технології розпізнавання об'єктів YOLO.

Методи дослідження включають в себе аналіз предметної області, методи комп'ютерного зору, алгоритми глибокого навчання розпізнавання об'єктів (YOLO, CNN, RNN), системний та структурний аналіз, моделювання, а також практична реалізація з використанням відповідного програмного забезпечення та бібліотек для машинного навчання.

Така підсистема стане складовою частиною системи розпізнавання об'єктів у відео та сприятиме підвищенню ефективності обліку товарів. На основі аналізу буде обрано найбільш доцільний підхід для реалізації запропонованої підсистеми підрахунку товару.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ

1.1. Огляд сучасних програмних засобів підрахунку товарів.

У сучасних умовах автоматизація процесів обліку та контролю товарів на складах і в торгових точках набуває все більшого значення. Зростаючий обсяг даних, потреба в оперативному аналізі залишків і мінімізація людських помилок стимулюють впровадження програмних рішень для підрахунку товарів.

Розглянемо найпоширеніші системи підрахунку товару:

1. Trax – це система, яка аналізує полиці магазинів за допомогою фото та AI для виявлення нестач, недоліків викладки тощо.
2. Scandit – це мобільна система на базі смартфона з комп'ютерним зором для інвентаризації та логістики.
3. Gather AI- це система інвентаризації, що використовує дрони з комп'ютерним зором для автоматичного підрахунку товарів на складах.

Trax - це наскрізна система для глибокого навчання, яка фокусується на чистому коді та швидкості. Вона активно використовується і підтримується в команді Google Brain. Trax включає базові моделі (такі як ResNet, Transformer) та алгоритми такі як REINFORCE, A2C. Він також активно використовується для досліджень і включає нові моделі, такі як Reformer, та нові RL-алгоритми, такі як AWR. Trax має прив'язки до великої кількості наборів даних глибокого навчання, включаючи Tensor2Tensor і TensorFlow. Ви можете використовувати Trax як бібліотеку з ваших власних python-скриптів і блокнотів або як бінарний файл з оболонки, що може бути зручніше для навчання великих моделей. Він працює без будь-яких змін на CPU, GPU та TPU.

Основні можливості Scandit включають програмне сканування штрих-кодів, сканування ідентифікаторів та розпізнавання тексту. Scandit описує свій продукт і екосистему підтримки як «Платформа інтелектуального збору даних». Платформа включає базовий продукт - програмну бібліотеку (SDK)

для сканування штрих-кодів і розпізнавання тексту корпоративного рівня, що працює на основі власних алгоритмів комп'ютерного зору. SDK можна інтегрувати в додатки сторонніх розробників, що дає змогу будь-якому смарт-пристрою з камерою, наприклад, смартфону, стати мобільним сканером штрих-кодів. SDK може бути доповнений більш складними функціями, такими як обробка зображень, аналітика та доповнена реальність. Окрім цих основних можливостей, Scandit також надає додаткову функціональність, включаючи MatrixScan, який сканує та декодує декілька штрих-кодів одночасно, та ShelfView, який використовує запатентовану технологію розпізнавання зображень та доповнену реальність для управління запасами в роздрібній торгівлі.

Gather AI - це система на базі глибокого навчання для автоматизованої інвентаризації, яка поєднує автономні дрони з комп’ютерним зором для сканування складів. Система оптимізована для точного розпізнавання об'єктів, зчитування штрихкодів і виявлення нестач товару в режимі реального часу. Gather AI активно використовується у великих логістичних центрах і складах, де необхідна масштабована, швидка та мінімально ручна інвентаризація. Gather AI ґрунтується на сучасних моделях глибокого навчання (CNN, OCR, Object Detection на базі YOLO/ResNet), які забезпечують високоточну обробку зображень, отриманих з дронів. Система включає в себе алгоритми SLAM (одночасна локалізація та побудова карти), що дозволяють дронам автономно орієнтуватись у просторі без GPS. Gather AI інтегрується з корпоративними ERP та WMS-системами, передаючи структуровані дані про розміщення та кількість товару. Рішення працює у хмарі, підтримує обробку даних на GPU і дозволяє масштабування під великі обсяги складів. Ви можете керувати Gather AI через зручний веб-інтерфейс або API. Система також адаптована до роботи в умовах різного освітлення, високих стелажів та складних конфігурацій складу.

Однак якщо впроваджувати ці системи можна зіткнутись з їхніми критичними мінусами. Ось декілька критичних мінусів які можна описати цим системам :

Trax - Обіцяє вам чарівне AI яке працює з фото, але якщо працівник зробив криво або з бликом - AI вже не чарівник. Людина мусить бігати по магазину, фоткати все вручну - це повільно і дорого при масштабі. При поганому освітленні або нестандартній викладці система плутає навіть базові товари. Якщо реальна викладка хоч трохи відрізняється — виникає хаос і помилки. Ідея хороша, але реалізація тримається на руках персоналу, як костилі на AI.

Scandit надає супер-смартфон у кишені кожного співробітника.

Але почавши роботу треба сканувати кожен товар власноруч - ні про яку автоматизацію не йдеться. Якщо смартфон старий або камера слабка - точність падає миттєво. Товар без чіткої мітки? Забудь. Немає ні AI-аналітики, ні глибокого розуміння. Великий склад чи магазин = втрачений робочий день на сканування всього вручну.

GatherAI – обіцяє автономні дрони, розумний AI і магію майбутнього.

А на ділі:

1. Якщо ви не Amazon - підготовте гаманець. Вартість дронів, серверів і налаштувань «кусається»
2. Без підготовки приміщення, навчання персоналу й адаптації - нічого не злетить (буквально).
3. Якщо зв'язок слабкий - дрони «гальмують» або взагалі не працюють.
4. Маленькі магазини чи складські куточки - не про Gather. Вони там просто не розвернуться.

Зручний як сканер, але точно не революція. Це як покласти касиру в руки калькулятор і сказати - «ось, AI».

Після аналізу всіх цих недоліків мною було прийняте рішення створити свою підсистему для підрахунку товару на базі нейронних мереж. Після огляду систем потрібно проаналізувати архітектури нейронних мереж.

1.2. Аналіз архітектур нейромереж.

Для ефективного підрахунку товарів у сучасних системах автоматизації широко застосовуються нейронні мережі. Різні архітектури нейромереж мають свої переваги і недоліки залежно від конкретного завдання, тому їх детальний аналіз є важливим кроком для вибору оптимального рішення.

Отже, розглянемо та проаналізуємо такі нейронні мережі:

Рекурентна нейронна мережа (RNN) - це модель глибокого навчання, яка навчена обробляти та перетворювати послідовні вхідні дані на певні послідовні вихідні дані. Послідовні дані - це дані, такі як слова, речення або часові ряди, де послідовні компоненти взаємопов'язані на основі складної семантики та синтаксичних правил. RNN - це програмна система, яка складається з багатьох взаємопов'язаних компонентів, що імітують процес перетворення послідовних даних людиною, наприклад, переклад тексту з однієї мови на іншу. RNN значною мірою замінюються штучним інтелектом на основі трансформаторів та великими мовними моделями, які є набагато ефективнішими в послідовній обробці даних. [1]

Вони складаються з нейронів - вузлів обробки даних, які працюють разом для виконання складних завдань. Нейрони організовані у вигляді вхідного, вихідного та прихованого шарів. Вхідний шар отримує інформацію для обробки, а вихідний шар надає результат. Обробка даних, аналіз і прогнозування відбуваються у прихованому шарі.

Прихований шар працює, передаючи послідовні дані, які вони отримують, прихованим шарам, крок за кроком. Однак вони також мають само зациклений або повторюваний робочий процес: прихований шар може запам'ятовувати і використовувати попередні вхідні дані для майбутніх прогнозів у компоненті короткострокової пам'яті. Він використовує поточні

вхідні дані та збережену пам'ять для прогнозування наступної послідовності (рис.1.1).

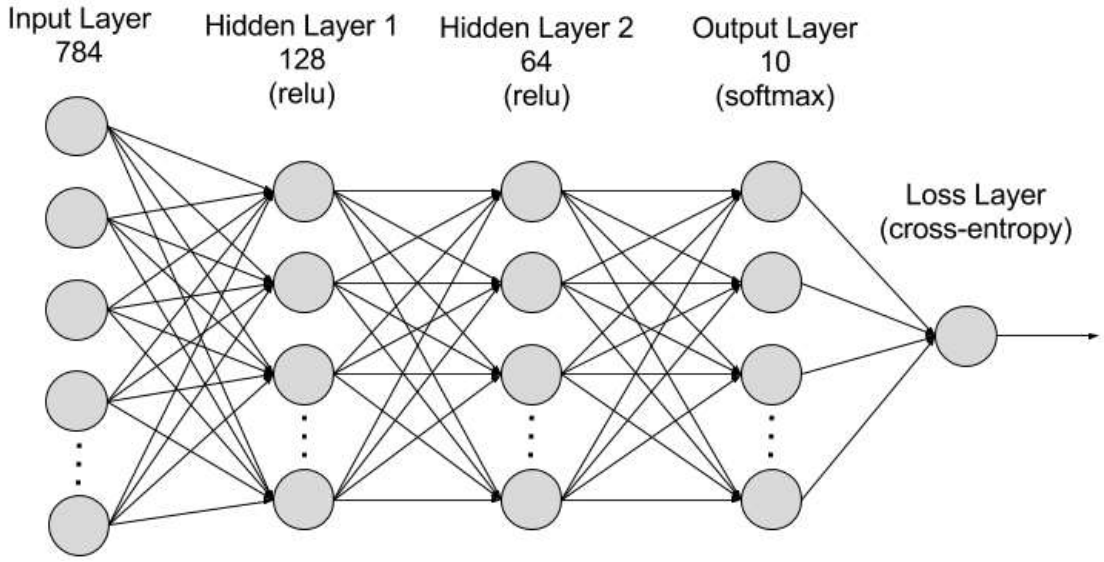


Рис 1.1 - Прихований шар

Рекурентні нейронні мережі (RNN) були введені для подолання обмежень традиційних нейронних мереж, таких як нейронні мережі прямого поширення , коли мова йде про обробку послідовних даних. CNN приймає вхідні дані і обробляє кожен вхід незалежно через ряд прихованих шарів, не враховуючи порядок і контекст інших вхідних даних. Через це він не може ефективно обробляти послідовні дані та вловлювати залежності між входами. Як наслідок, CNN не дуже добре підходять для задач послідовної обробки, таких як моделювання мови, машинний переклад, розпізнавання мови, аналіз часових рядів та багатьох інших додатків, що вимагають послідовної обробки. Для подолання обмежень традиційних нейронних мереж з'являються RNN. [2]

RNN долають ці обмеження, вводячи рекурентні зв'язки, які дозволяють інформації переходити від одного часового кроку до іншого. Цей рекурентний зв'язок дозволяє RNN підтримувати внутрішню пам'ять, де вихід кожного кроку подається як вхід для наступного кроку, що дозволяє мережі фіксувати інформацію з попередніх кроків і використовувати її на

поточному кроці, дозволяючи моделі вивчати часові залежності і обробляти вхідні дані змінної довжини.

Для більш чіткого розуміння концепції RNN, розглянемо Рисунок 1.2 .

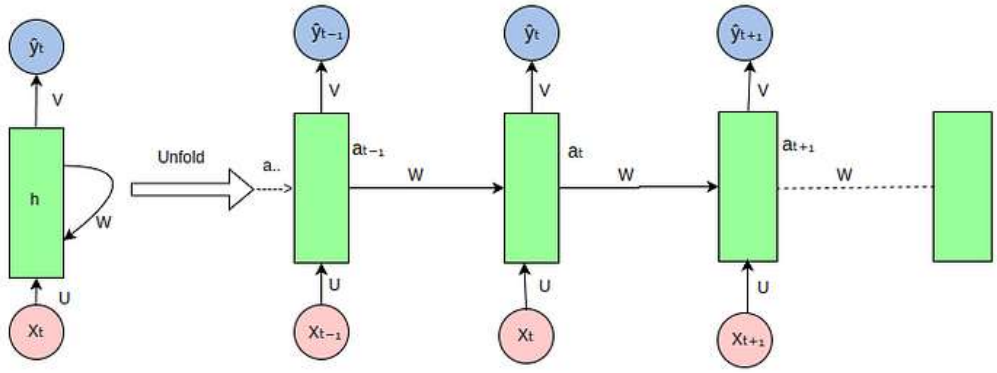


Рис 1.2 - Концепція роботи RNN

RNN отримує вхідний вектор X, і мережа генерує вихідний вектор у шляхом послідовного сканування даних зліва направо, з кожним кроком оновлюючи прихований стан і виробляючи вихідний вектор. Мережа має однакові параметри на всіх часових кроках. Це означає, що один і той самий набір параметрів, представлений U, V, W, використовується послідовно по всій мережі. U - ваговий параметр, що визначає зв'язок між вхідним шаром X і прихованим шаром h, W - вагу, пов'язану зі зв'язком між прихованими шарами, а V - зв'язок між прихованим шаром h і вихідним шаром y. Такий розподіл параметрів дозволяє RNN ефективно вловлювати часові залежності і більш ефективно обробляти послідовні дані, зберігаючи інформацію з попереднього входу в поточному прихованому стані. [3]

Зворотне розповсюдження передбачає коригування параметрів моделі (ваг та зміщень) на основі похибки між прогнозованим виходом та фактичним цільовим значенням. Метою зворотного розповсюдження є покращення продуктивності моделі шляхом мінімізації функції втрат. Поширення в часі - це спеціальний варіант зворотного поширення, який використовується для навчання RNN, де помилка поширюється назад у часі до початкового часового кроку t=1. Поширення включає два ключові кроки: прохід вперед і прохід назад.

Прохід вперед: Під час прямого поширення RNN обробляє вхідну послідовність у часі від t=1 до t=n, де n - довжина вхідної послідовності. При кожному прямому поширенні відбувається наступний розрахунок

Прохід назад: прохід назад у BPTT передбачає обчислення градієнтів функції втрат по відношенню до параметрів мережі (U, W, V та зсувів) на кожному часовому кроці. [4]

Під час зворотного поширення градієнти можуть стати занадто малими, що призводить до проблеми зникаючого градієнта, або занадто великими, що призводить до проблеми «вибухаючого» градієнта, коли вони поширюються назад у часі. У випадку зі зникаючими градієнтами проблема полягає в тому, що градієнт може стати занадто малим, коли мережа намагається ефективно вловити довгострокові залежності. Він все ще може зійтися під час навчання, але це може зайняти дуже і дуже багато часу. На противагу цьому, в задачі з вибуховим градієнтом великий градієнт може призвести до чисельної нестабільності під час навчання, що призведе до відхилення моделі від оптимального рішення і ускладнить збіжність мережі до глобального мінімуму(рис 1.3).

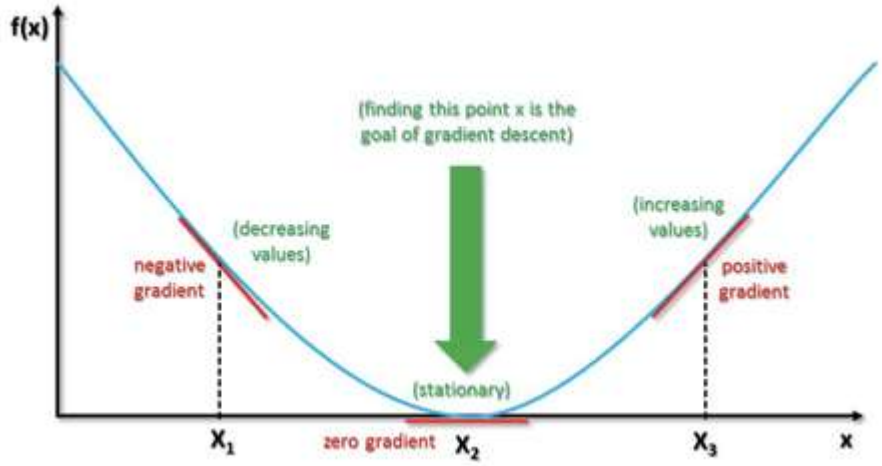


Рис 1.3 - Приклад «вибухаючого» градієнту

Отже, розглянувши архітектуру RNN перейдемо до розгляду архітектури CNN (рис 1.6).

CNN – це клас глибоких штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень.

CNN зазвичай складається з декількох шарів, які можна умовно поділити на три групи: згорнуті шари, об’єднані шари та повністю з’єднані шари. Коли дані проходять через ці шари, складність CNN зростає, що дозволяє CNN послідовно ідентифікувати більші частини зображення, а також більш абстрактні об’єкти. [5]

Згортковий шар є фундаментальною частиною CNN, і саме тут відбувається більшість обчислень. Цей шар використовує фільтр або ядро – невелику матрицю ваг – для переміщення по сприйнятливому полю вхідного зображення, щоб виявити наявність специфічних особливостей. Процес починається з переміщення ядра по ширині та висоті зображення, зрештою, за кілька ітерацій воно охоплює все зображення. У кожній позиції обчислюється точковий добуток між вагами ядра та значеннями пікселів зображення під ядром. Це перетворює вхідне зображення на набір карт ознак або згорнутих ознак, кожна з яких представляє наявність та інтенсивність певної ознаки в різних точках зображення. CNN часто включають декілька згорнутих шарів, що накладаються один на одного.

Завдяки такій багаторівневій архітектурі CNN поступово інтерпретує візуальну інформацію, що міститься у вихідних даних зображення. На ранніх шарах CNN ідентифікує основні характеристики, такі як краї, текстури або кольори. Більш глибокі шари отримують вхідні дані з карт особливостей попередніх шарів, що дозволяє їм виявляти більш складні патерни, об’єкти і сцени. [7]

Шар об’єднання в CNN є критично важливим компонентом, який слідує за згортковим шаром. Подібно до згорткового шару, операції шару об’єднання включають процес розгортання вхідного зображення, але його функція відрізняється. Мета шару об’єднання – зменшити розмірність вхідних даних, зберігаючи при цьому критично важливу інформацію, таким чином підвищуючи загальну ефективність мережі. Зазвичай це досягається шляхом зменшення вибірки, тобто кількості точок у вхідних даних. Для CNN це зазвичай означає зменшення кількості пікселів, що

використовуються для представлення зображення. Найпоширенішою формою об’єднання є максимальне об’єднання, яке зберігає максимальне значення в межах певного вікна – тобто розмір ядра – і відкидає інші значення. Інша поширена методика, відома як середнє об’єднання, використовує подібний підхід, але використовує середнє значення замість максимального. Зменшення дискретизації (Down sampling) значно зменшує загальну кількість параметрів і обчислень. На додаток до підвищення ефективності, це посилює здатність моделі до узагальнення. Менш складні моделі з високорівневими характеристиками зазвичай менш схильні до перенавчання – явища, коли модель навчається шуму та надто специфічним деталям у своїх навчальних даних, що негативно впливає на її здатність узагальнювати нову, небачену інформацію.

Повністю зв’язаний шар відіграє важливу роль на заключних етапах CNN, де він відповідає за класифікацію зображень на основі ознак, вилучених у попередніх шарах. Термін «повністю пов’язаний» означає, що кожен нейрон в одному шарі пов’язаний з кожним нейроном у наступному шарі. Повністю зв’язний шар інтегрує різні ознаки, отримані в попередніх згорткових і об’єднувальних шарах, і зіставляє їх з конкретними класами або результатами. Кожен вхід з попереднього шару з’єднується з кожною одиницею активації в повністю пов’язаному шарі, що дозволяє CNN одночасно враховувати всі ознаки при прийнятті остаточного рішення про класифікацію (рис1.4).

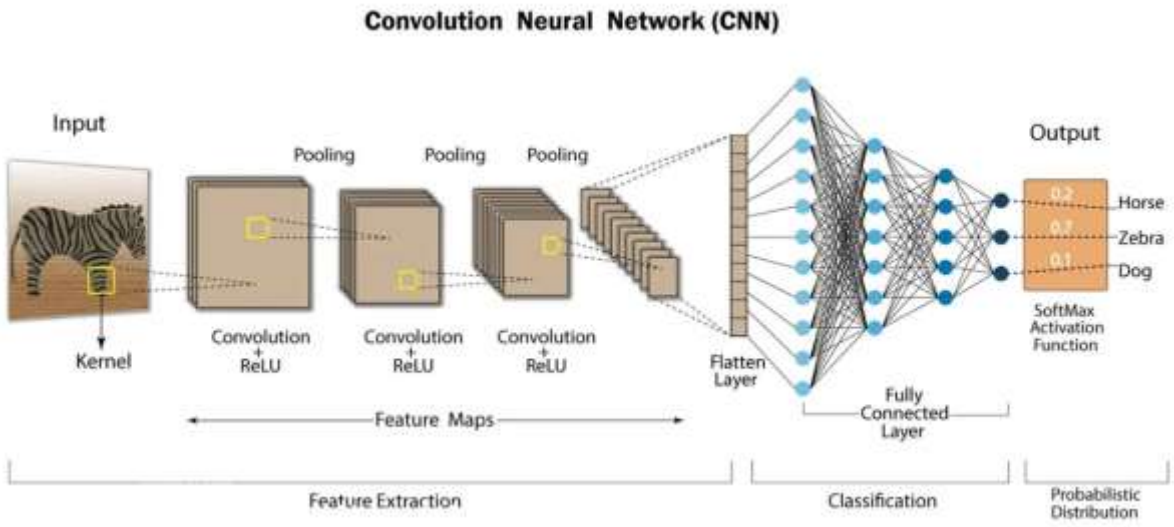


Рис 1.4 – Архітектура CNN

Архітектура CNN складається з двох ключових компонентів:

1. Виділення особливостей:

- Інструмент згортки, який відокремлює та ідентифікує відмінні риси зображення для аналізу в процесі.

2. Шар класифікації:

-Повністю пов'язаний шар, який отримує результат процесу згортки і прогнозує клас зображення на основі ознак, отриманих раніше. [9]

Існує кілька рівнів і важливих компонентів CNN. Я поясню ці шари щоб була можливість глибше зрозуміти CNN (рис 1.5).

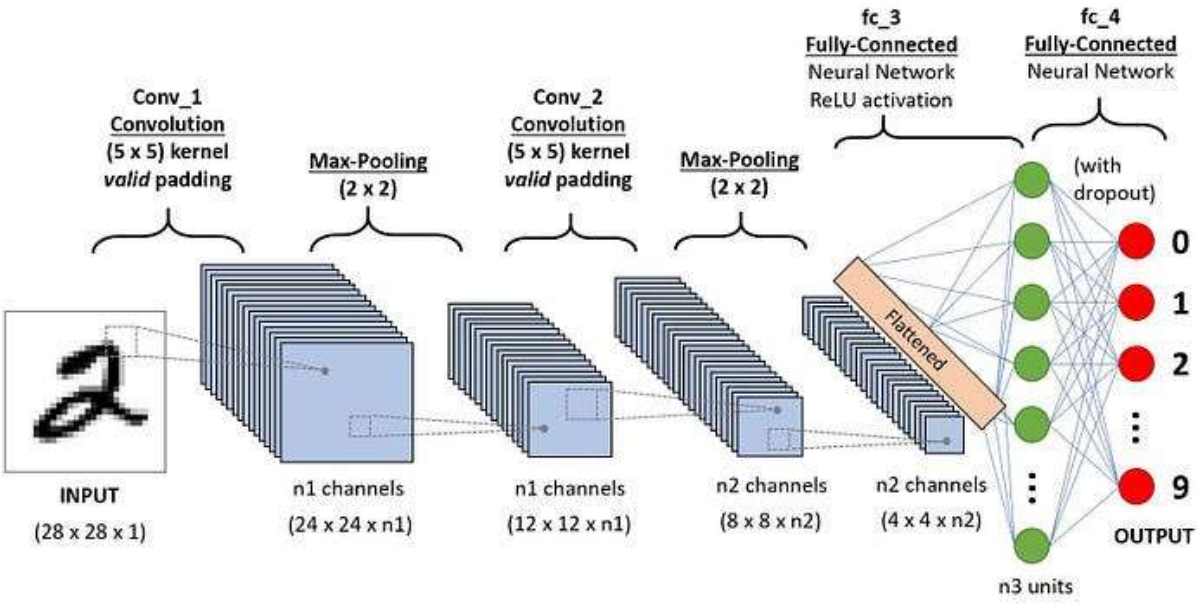


Рис 1.5 - Важливі компоненти CNN

Згорткові шари складаються з набору фільтрів (також званих ядрами), які застосовуються до вхідного зображення. Результатом згорткового шару є карта ознак, яка є представленням вхідного зображення із застосованими фільтрами. Згорткові шари можна об'єднувати для створення більш складних моделей, які здатні розпізнавати більш складні особливості зображень. [10]

Об'єднання шарів - це тип згорткового шару, який використовується в глибокому навчанні. Завдяки об'єднанню шарів зменшується просторовий розмір вхідних даних, що полегшує обробку і використовує менше пам'яті. Об'єднання пришвидшує навчання, а також сприяє зменшенню кількості параметрів. Максимальне об'єднання та середнє об'єднання - це два основні типи об'єднання. У той час як середнє об'єднання використовує середнє значення з кожної карти ознак, максимальне об'єднання використовує максимальне значення. Для того, щоб мінімізувати розмір вхідних даних перед тим, як вони потрапляють до повністю з'єднаного шару, шари об'єднання часто використовуються після шарів згортки. [11]

Одним з найбільш фундаментальних типів шарів у згортковій нейронній мережі є повністю зв'язаний шар . Кожен нейрон у повністю зв'язаному шарі повністю з'єднаний з кожним іншим нейроном у шарі нижче, як випливає з назви. Коли метою є використання особливостей, виявлених попередніми шарами, для створення прогнозів, повністю пов'язані шари часто застосовуються в кінці CNN.

Останній рівень згорткової нейронної мережі - це класифікатор (CNN). Його називають щільним шаром, і це просто класифікатор штучної нейронної мережі (ANN). Класифікатор ШНМ, як і будь-який інший класифікатор, вимагає певних властивостей. Це означає, що потрібен вектор ознак. [12]

Як наслідок, ви повинні перетворити вихід згорткового компонента CNN в одномірний вектор ознак, який може використовувати ANN. Ця

процедура відома як згладжування. Вона вирівнює всю структуру вихідних даних згорткових шарів в один довгий вектор ознак, який щільний шар може використовувати для класифікації.

Перейдемо до розгляду архітектури YOLO.

YOLO або You Only Look Once – популярна нейронна мережа розпізнавання об’єктів у реальному часі.

Віна розроблений для швидкої та ефективної роботи, що робить його ідеальним для застосування в реальному часі. Вона може розпізнавати об’єкти на зображеннях або відео потоках зі швидкістю до 45 кадрів на секунду, що значно швидше, ніж інші моделі виявлення об’єктів. Така швидкість і ефективність є критично важливими для таких застосувань, як відеоспостереження та охорона, де моніторинг у реальному часі є надзвичайно важливим. [13]

Незважаючи на свою швидкість, YOLO також відрізняється високою точністю. Її середня точність (mAP) досягає 78,6%, що можна порівняти з іншими найсучаснішими моделями розпізнавання об’єктів. Така висока точність досягається завдяки комбінації методів, зокрема, якірним ящикам, багато масштабному навчанню та виділенню ознак.

Також він є гнучким і налаштовуваним, що робить його придатним для різних застосувань. Він може розпізнати кілька об’єктів одночасно і навчатися на спеціальних наборах даних, що дозволяє йому розпізнавати конкретні об’єкти або класи. Така гнучкість і можливість кастомізації роблять його ідеальним для таких галузей, як роздрібна торгівля, де його можна використовувати для аналізу поведінки покупців і оптимізації планування магазинів.

Сьогодні YOLO – найточніша й найефективніша модель, яку повсюдно використовують для широкого спектра завдань комп’ютерного зору. Це одна з небагатьох архітектур глибокого навчання, яка через багато років після відкриття залишається улюбленою і застосовною.

Також слід зазначити що він є надійним алгоритмом розпізнавання об’єктів, який здійснив революцію у відео аналітиці. Швидкість, точність і простота використання зробили його основним алгоритмом для виявлення об’єктів у різних сферах застосування[14]. Оскільки сфера відео аналітики продовжує зростати, YOLO, ймовірно, залишиться на передньому краї алгоритмів розпізнавання об’єктів (рис 1.6).

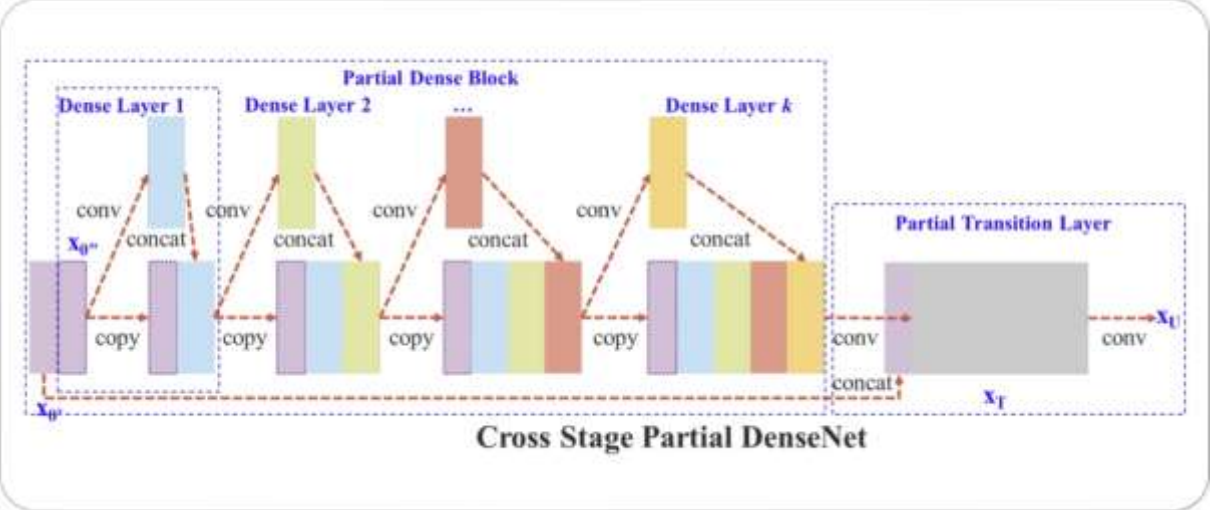


Рис 1.6 – Алгоритм YOLO

YOLO використовує ознаки з усього зображення і одночасно прогнозує граничні області. Зображення розбивається на сітку S X S, і кожна сітка генерує B обмежувальних рамок та їхні оцінки достовірності. Ці оцінки впевненості відображають, наскільки модель впевнена в тому, що область містить об'єкт, а також наскільки точно вона вважає, що область є такою, яку вона прогнозує. Ці оцінки впевненості відображають, наскільки модель впевнена в тому, що область містить об'єкт, а також наскільки точно вона вважає, що область відповідає прогнозованій. Кожна обмежувальна рамка складається з 5 прогнозів: x, y, w, h та достовірності. Координати (x, y) представляють центр рамки відносно меж комірки сітки. Ширина і висота прогнозуються відносно всього зображення (рис 1.7).

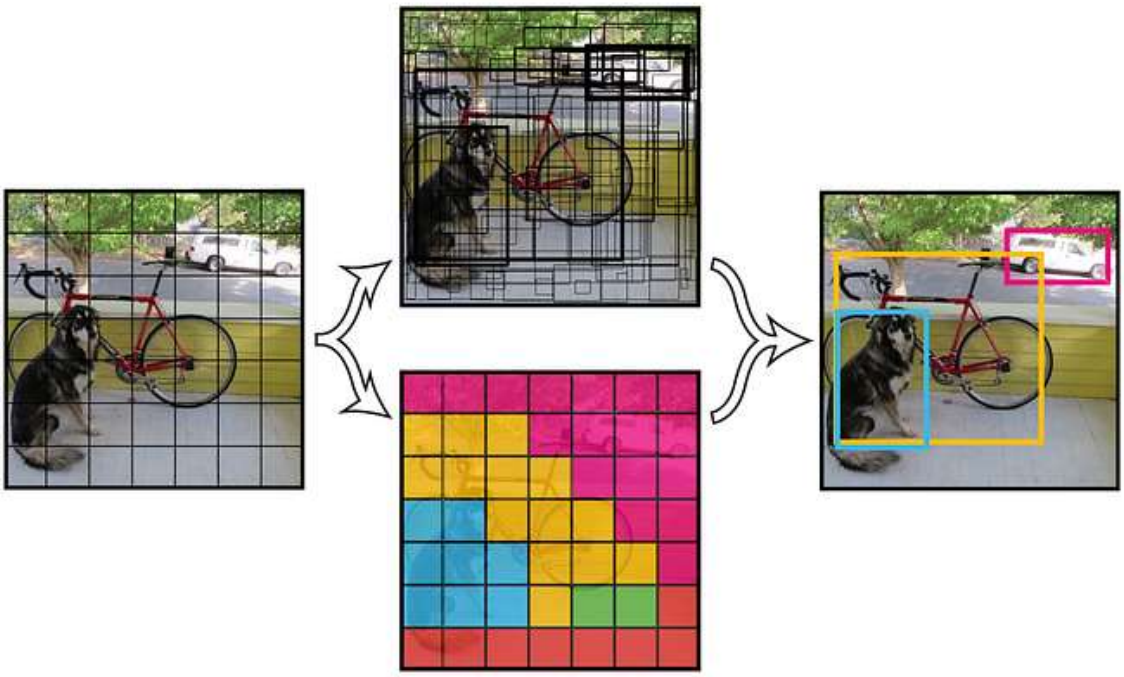


Рис 1.7 - Графічне представлення роботи ШНМ за архітектурою YOLO

Початкові згорткові шари мережі виділяють ознаки із зображення, тоді як повністю з'єднані шари прогнозують вихідні ймовірності та координати. Модель має 24 згорткових шари, за якими слідують 2 повністю з'єднаних шари.

YOLO накладає сильні просторові обмеження на прогнозування граничних областей, оскільки кожна комірка сітки прогнозує лише дві області і може мати лише один клас. Це просторове обмеження обмежує кількість найближчих об'єктів, які модель може передбачити. Модель не може впоратися з невеликими об'єктами, які з'являються групами, наприклад, зграями птахів.

Після розгляду цих архітектур зведемо таблицю порівнянь (таблиця 1)

Таблиця 1.1. Порівняння архітектур

Критерій порівнянь	CNN	RNN	YOLO
Тип задачі	Класифікація, сегментація зображень	Обробка послідовностей, мова	Об’єктне розпізнавання на відео\фото
Принцип роботи	Виділення ознак з зображень за допомогою згорток	Обробка послідовних даних зі збереженням стану	Одно етапне виявлення об'єктів
Переваги	Висока швидкість роботи	Висока точність класифікації	Працює з текстом, мовою, сигналами
	Підходить для реального часу	Врахування контексту та порядку	Можливість генерації послідовностей
	Добра точність виявлення об’єктів	Добра масштабованість	
Недоліки	Зниження точності при виявленні дрібних об'єктів	Відсутність пам’яті	Затухаючі/вибухаючі градієнти
	Чутливість до позиційного шуму	Складність навчання	Складна структура навчання
	Не працює з часовими залежностями	Відносно повільна робота	
Продуктивність у реальному часі	Висока	Середня	Низька

Зважаючи на порівняння архітектур RNN, CNN та YOLO, можна стверджувати, що кожна з них має свої переваги та недоліки залежно від умов використання. У процесі роботи було обрано YOLO -як ефективну та швидку модель для розпізнавання об’єктів. [16]

1.3. Постановка задачі.

В умовах, де потрібна оперативність, гнучкість та візуальний контроль (наприклад, у виробництві), більш перспективними є системи на основі комп’ютерного зору. Використання алгоритмів, розпізнавання, дозволяє побудувати ефективну, автономну підсистему для автоматичного підрахунку товару без прямої участі людини, що й стало основою поставленного завдання.

Для розв’язання задачі автоматизованого підрахунку товару на основі зображень або відео необхідно обрати відповідну архітектуру нейронної мережі, яка забезпечить ефективне виявлення об’єктів у кадрі з достатньою точністю та швидкістю.

Для реалізації поставленої задачі важливо вибрати таку архітектуру нейронної мережі, яка найкраще підходить для обробки зображень, виявлення об’єктів на них та визначення їхньої кількості. У цьому розділі проведено аналіз трьох популярних типів нейронних мереж - RNN ,CNN та YOLO - з точки зору їхньої придатності до задач комп’ютерного зору, а саме - виявлення та підрахунку товарів на зображеннях.

RNN - Рекурентні нейронні мережі розроблені для роботи з послідовними даними - такими як текст, аудіо або часові ряди. Їх основна особливість полягає у здатності "запам’ятовувати" попередні стани, що дозволяє враховувати контекст при обробці кожного нового елемента послідовності. Однак, у задачі розпізнавання товарів на статичних зображеннях RNN є малоефективними, оскільки вони не спеціалізовані на аналізі просторових структур, таких як пікселі, форми чи контури об’єктів.

Зробимо висновок що RNN не є доцільним вибором для розпізнавання об’єктів на зображеннях, оскільки вони не мають вбудованих механізмів для обробки візуальної інформації.

CNN - Згорткові нейронні мережі є класичним підходом до обробки зображень. Вони здатні автоматично виділяти ознаки з зображень (краї, текстури, форми) за допомогою згорткових фільтрів. CNN широко

застосовуються в задачах класифікації зображень, розпізнавання об’єктів, сегментації тощо. У контексті поставленої задачі CNN добре підходять для класифікації або локалізації одного об’єкта на зображенні. Проте для виявлення багатьох об’єктів (наприклад, кількох товарів різних типів) стандартна CNN не є достатньою. Для цього потрібні додаткові механізми, наприклад, регіональні пропозиції , що використовуються у таких мережах, як Faster R-CNN.

Маємо висновок що CNN - хороший базовий підхід до обробки зображень, проте для задачі розпізнавання кількох об’єктів на зображенні краще використовувати модифіковані або спеціалізовані архітектури.

YOLO - це сучасна нейронна мережа для детекції об’єктів у реальному часі. Її головна перевага полягає в тому, що вона виконує локалізацію та класифікацію всіх об’єктів на зображенні за один прохід, тобто дуже швидко. YOLO ділить зображення на сітку і прогнозує межі об’єктів та відповідні класи одночасно. YOLO є однією з найефективніших архітектур для задач, де важлива швидкість і точність, зокрема для реального застосування в магазинах, логістичних системах або інвентаризації. Мережа добре масштабується, підтримує навчання на власних датасетах і адаптується до різних категорій об’єктів, включаючи продукти харчування, напої, побутові товари тощо.

Оглянувши архітектури маємо висновок що YOLO є найоптимальнішим вибором для поставленої задачі, оскільки дозволяє розпізнавати та підраховувати кілька об’єктів на одному зображенні з високою точністю та швидкодією.

Оскільки більшість відкритих наборів даних не містять специфічних товарів, які необхідно розпізнавати в межах поставленої задачі, виникає потреба у створенні власного датасету. Для цього здійснюється зйомка товарів у різних умовах, розмітка зображень та формування структури датасету відповідно до вимог обраної моделі.

На наступному етапі виконується навчання нейронної мережі на підготовленому датасеті. Після завершення навчання проводиться тестування моделі на нових зображеннях. Основною метою є не лише розпізнавання товарів, а й точний підрахунок їх кількості, що особливо важливо для практичного застосування, наприклад, у сфері торгівлі або інвентаризації.

Після завершення розпізнавання та отримання результату потрібно буде модифікувати код для підрахунку товару та провести процес розпізнавання знову для отримання результату підрахунку товару.

На завершальному етапі результати роботи моделі перевіряються в реальних умовах, з використанням зображень, що були зроблені в магазині або на складі. Аналізується точність розпізнавання та на основі отриманих результатів робляться висновки щодо ефективності запропонованого підходу та можливостей його подальшого вдосконалення.

Отже, для реалізації поставленої задачі потрібно виконати такий ряд завдань:

1. Проаналізувати архітектури для розпізнавання (підрахунку кількості) товарів.
2. Створити набір даних для підрахунку кількості товарів.
3. Тренування нейронної мережі з використанням набору даних.
4. Проаналізувати результати розпізнавання та підрахунку кількості товарів в реальних умовах.

РОЗДІЛ 2 ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВИКОНАННЯ ЗАВДАННЯ

2.1.Аналіз середовища розробки.

Вибір відповідного середовища розробки є ключовим етапом у створенні програмних засобів для автоматизації підрахунку товарів. Від характеристик та функціональності середовища залежить ефективність розробки, швидкість впровадження та подальша підтримка системи.

При виконанні завдання було проаналізовано багато можливих середовищ таких як:

1. PyCharm
2. VS Code
3. OpenVX
4. Google Colab

PyCharm, створена компанією JetBrains, поєднує в собі безліч корисних функцій та можливостей, що роблять процес написання та налагодження коду зручнішим та ефективнішим. Ця IDE підходить як для новачків, так і для досвідчених розробників, пропонуючи різні інструменти, які прискорюють створення, тестування та розгортання програм Python

PyCharm – це інтегроване середовище розробки для мови Python. Надає способи для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веб-розробку. [18]

Це не просто текстовий редактор з підтримкою Python, а повноцінне інтегроване середовище розробки, що надає широкий набір інструментів для зручної та ефективної роботи. Розглянемо основні можливості PyCharm, які роблять її такою популярною серед Python-розробників.

1. Розумний код-редактор. Однією з ключових функцій PyCharm є її інтелектуальний редактор коду. Він не тільки підсвічує синтаксис, а й пропонує автодоповнення, аналізує код у реальному часі та вказує на помилки, попереджаючи про можливі проблеми ще до того, як програма

буде запущена. Це значно прискорює процес написання коду, а також підвищує його якість.

2. Налагодження та тестування. PyCharm пропонує потужний вбудований налагоджувач, який дозволяє запускати код по кроках, відстежувати значення змінних та бачити стек викликів. Це допомагає швидко знаходити помилки та розуміти, як працює програма. Крім того, PyCharm підтримує різні інструменти для тестування, як-от юніт-тести, що дозволяє розробникам переконатися у правильній роботі їхнього коду ще до його розгортання.

3. Робота з базами даних. Для розробників, що працюють з даними, PyCharm пропонує вбудовані засоби для взаємодії з базами даних. За допомогою цієї IDE можна підключатися до різних СУБД (як-от MySQL, PostgreSQL, SQLite та інших), виконувати SQL-запити та керувати даними, не залишаючи робочого середовища. Це особливо зручно при створенні програм, що працюють з більшими обсягами даних.

Як і будь-яке інше інтегроване середовище розробки, PyCharm має свої сильні сторони, які роблять його популярним серед Python-розробників. А також має деякі недоліки, які можуть виявитися важливими залежно від вимог проєкту та уподобань користувача. Розгляньмо основні плюси та мінуси використання PyCharm.

Microsoft Visual Code, який також називають VS Code - це редактор початкового коду, створений Microsoft із Electron Framework для Windows, Linux та macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність. [26]

Visual Studio Code містить базову підтримку для більшості поширених мов програмування. Ця базова підтримка включає підсвічування синтаксису, зіставлення дужок, згортання коду та настроювані фрагменти.

Visual Studio Code також постачається з IntelliSense для JavaScript, TypeScript, JSON, CSS і HTML, а також підтримує налагодження Node.js. Підтримка додаткових мов може бути забезпечена безоплатними розширеннями на VS Code Marketplace.

Основні можливості Visual Studio Code :

1. Підтримка автодоповнення коду (IntelliSense).
2. Вбудована підтримка Git і інших систем контролю версій.
3. Підтримка відлагодження (debugging) прямо з редактора.
4. Потужна система розширень (plugins).
5. Вбудований термінал.
6. Підтримка віддаленої розробки (Remote Development).
7. Інтеграція з хмарними середовищами (Azure, AWS через плагіни).
8. Підтримка штучного інтелекту через GitHub Copilot.

OpenVX - специфікація, котра визначає прикладний програмний інтерфейс (API) для розробки переносних, високопродуктивних і енергоефективних застосунків і бібліотек для вирішення задач комп'ютерного зору (виявлення, стеження і класифікація об'єктів на зображеннях і відеоданих). специфікації OpenVX розроблена групою Khronos, що відповідає за розробку стандартів сімейства OpenGL і OpenCL.

OpenVX надає можливість використання серії алгоритмів для розпізнавання і відстеження переміщення особи, тіла і жестів, автоматизації відеоспостереження, автоматичних систем допомоги водієві, реконструкції об'єктів та сцен, доповненої реальності, візуального огляду, робототехніки та багатьох інших застосувань. Обробка даних в OpenVX-застосунках проводиться через маніпуляцію з графом функціональних вузлів, робота з яким може бути прискорена за рахунок залучення центрального процесора, графічного процесора, процесора цифрових сигналів і додаткових апаратних пристроїв, а також роздільної обробки фрагментів (tiling).

Основні особливості OpenVX Графова модель обчислень: замість прямого виконання інструкцій розробник створює граф операцій, які потім оптимізуються й виконуються. Портативність: код може бути виконаний на різних апаратних платформах без змін. Оптимізація апаратного прискорення: використання GPU, DSP, FPGA для максимальної продуктивності. Енергетична ефективність: ідеально підходить для мобільних і вбудованих систем. Стандартизовані API: забезпечують стабільність і сумісність. [22]

Кожна з них має свої переваги та недоліки залежно від специфіки завдань та наявних ресурсів. Проте для реалізації обчислювально складних задач комп'ютерного зору та машинного навчання потрібно мати потужне апаратне забезпечення, зокрема продуктивну графічну карту (GPU). У зв'язку з відсутністю потужної GPU на моєму пристрої, для роботи було обрано Google Colab. Це хмарне середовище забезпечує безкоштовний доступ до хмарних обчислювальних ресурсів із використанням GPU та TPU, що дозволяє ефективно виконувати великі обчислення, працювати з великими об'ємами даних і реалізовувати сучасні алгоритми машинного навчання без потреби в додаткових інвестиціях в апаратне забезпечення.

Google Colab, також відомий як Colab, став важливим інструментом в арсеналі Python-розробників. Ця безоплатна хмарна платформа кодування, запропонована технологічним гігантом Google, надає кілька переваг, які виходять за рамки типового середовища розробки.

Суть Google Colab полягає у створенні інтерактивного середовища для експериментів, аналізу даних та навчання моделей. Розгляньмо, що саме ви можете робити за допомогою Colab:

1. Навчати нейронні мережі для класифікації зображень і відео за допомогою бібліотек TensorFlow або PyTorch;
2. Створювати та навчати моделі для обробки тексту (бібліотеки NLTK або SpaCy);

3. Досліджувати та візуалізувати зібрані дані за допомогою бібліотек Python (Pandas, Matplotlib та Seaborn);

4. Навчати моделі для визначення мови та аналізу тональності текстів;

5. Використовувати бібліотеки Gym для навчання моделей з підкріпленням у симуляційних середовищах (ігрові симулятори);

6. Застосовувати алгоритми кластеризації або зниження розмірності для аналізу даних без явних міток;

7. Аналізувати великі набори даних, зокрема дані з хмарних сервісів або BigQuery;

8. Обробляти потокові дані за допомогою Apache Spark.

Розглянувши всі вибрані середовища зведемо таблицю порівнянь(таблиця 2.1)

Таблиця 2.1 Порівняння середовищ

Критерії порівнянь	PyCharm	VS Code	OpenVX	Colab
Тип інструменту	Повноцінне IDE для Python	Редактор коду (легке IDE)	Фреймворк для комп'ютерного зору та обробки зображень	Хмарне інтерактивне середовище для запуску Python-коду
Мова реалізації	Java + Python	JavaScript, TypeScript, Electron	C / C++	Python
Підтримувані мови	Переважно Python, також HTML, JS, SQL	Багато мов (через розширення)	C, C++, з обгортками для Python	
Фокус використання	Розробка Python-додатків, Django, Flask	Універсальний редактор для розробки	Обробка зображень, відео, алгоритми комп'ютерного зору	Машинне навчання, обробка даних, візуалізація, навчання моделей
Переваги	Потужна функціональність для професійної розробки	Безкоштовність	Висока продуктивність	Безкоштовний доступ до GPU/TPU

Критерії порівнянь	PyCharm	VS Code	OpenVX	Colab
	Гнучкість налаштування під різні завдання	Кросплатформеність	Енергозбереження	Зручна інтеграція з Google Drive
	Широка підтримка фреймворків та технологій	Швидкість і легкість	Портативність коду	Підтримка бібліотек для ML/AI (TensorFlow, PyTorch тощо)
	Інтеграція з різними інструментами	Величезна кількість розширень	Оптимізація графа виконання	Підходить для командної роботи
		Інтеграція з хмарними середовищами	Широка підтримка	Не потребує встановлення
Недоліки	Високі системні вимоги	Не повноцінна IDE	Висока складність для новачків	Обмеження по ресурсах у безкоштовній версії
	Вартість професійної версії	Залежність від розширень	Менш популярний серед розробників	Залежність від підключення до Інтернету
	Іноді надто складний інтерфейс для новачків	Високе використання ресурсів	Обмежена гнучкість	Менш зручне для великих проєктів

Отже, після дослідження та визначення їх плюсів та мінусів було обране середовище. Проте через обмежені апаратні ресурси персонального комп’ютера виникли труднощі із запуском та навчанням моделі локально, тому для реалізації було прийнято рішення скористатися середовищем Google Colab, яке надало необхідні обчислювальні потужності.

2.2. Архітектура YOLO.

Архітектура YOLO -це одне з найвідоміших рішень для об'єктного розпізнавання в реальному часі. Порівнявши версії можна побачити ефективність серед інших методів(рис.2.1).

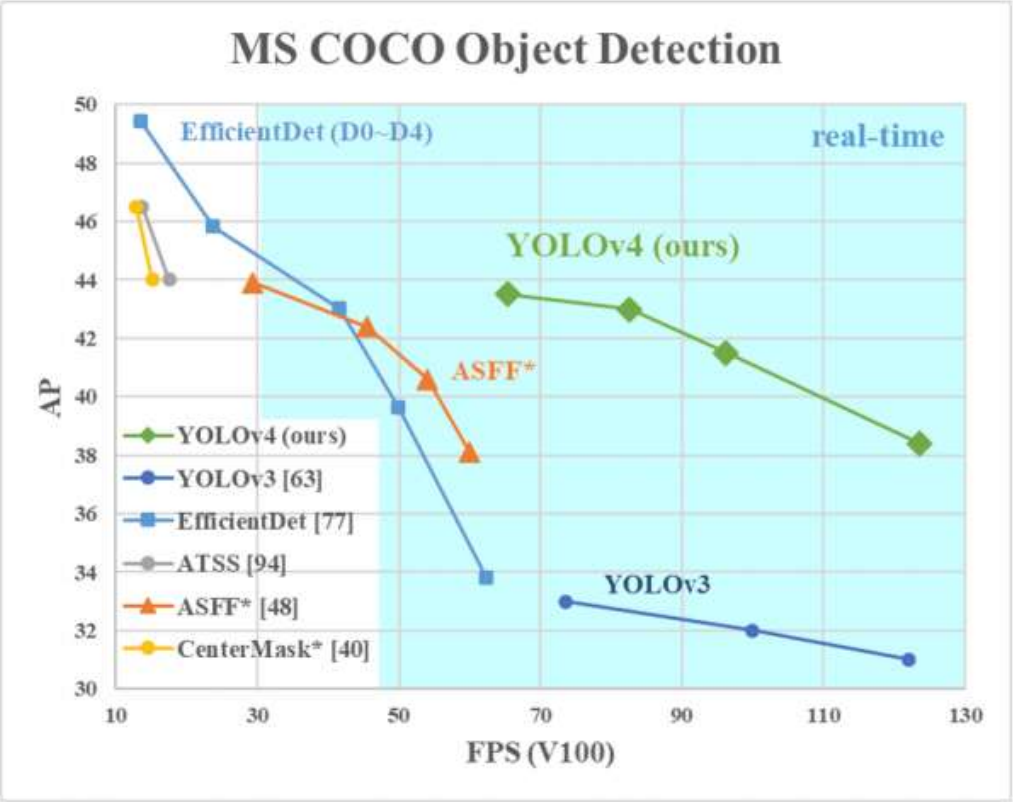


Рис 2.1 - Порівняння версій

Алгоритм навчання нейронної мережі YOLO:

Крок 1: Зазвичай картинки змінюють розмір під 416x416 перед початком навчання нейронної мережі, щоб можна було їх подавати розмірами партій (batch size) для прискорення навчання.

Крок 2: Ділимо картинку на клітини розміром SxS. У YOLO прийнято ділити картинку на клітини (рис 2.2).

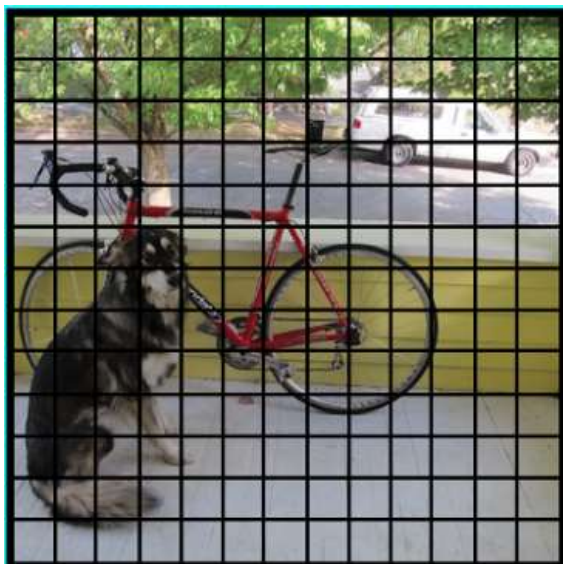


Рис 2.2 - Приклад поділу зображення

Крок 3: Картинка з датасету проганяється через нашу нейронну мережу.

Розглянемо вихід моделі

Для кожної клітинки i, модель повертає:

$$\mathcal{L} = \lambda_{\text{coord}} i = S^2 \sum_{j=0}^{B} 1_{ij}^{\text{oj}[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2]}, \quad (2.1.)$$

Функція втрат складається з кількох частин:

1. координатна втрата(λ_{coord}),
2. втрата впевненості (S),
3. втрата на класифікацію(B):

$$\lambda_{\text{coord}} i = S^2 \sum_{j=0}^{B} 1_{ij}^{\text{oj}\left[\left(\sqrt{w_i}-\sqrt{\widehat{w_i}}\right)^2+\left(\sqrt{h_i}-\sqrt{\widehat{h_i}}\right)^2\right]} \quad (2.2.)$$

$$i = S^2 \sum_{j=0}^{B} 1_{ij}^{\text{oj}(C_i-\widehat{C_i})^2} + \lambda_{\text{noobj}} i = S^2 \sum_{j=0}^{B} 1_{ij}^{\text{noj}(C_i-\widehat{C_i})^2} + i = 0 S^2 1_i^{\text{oj}} \sum_{c\in\text{classes}} \left(p_i(c)-\widehat{p_i}(c)\right)^2 \quad (2.3.)$$

YOLO має такий вигляд регресії координат

- x , y ∈ [0 , 1] x , y ∈ [0,1]: координати відносно клітинки (нормалізовані),

- w , h ∈ [0 , 1] w , h ∈ [0,1]: ширина і висота відносно зображення.

Щоб прибрати перекриті бокси, застосовується NMS:

Алгоритм:

1. Відсортувати всі бокси за ймовірністю (confidence).
2. Вибрати найкращий бокс.
3. Видалити всі бокси з IOU > threshold (зазвичай 0.5).
4. Повторити.

На відміну від CNN, YOLO працює на рівні всього зображення -це означає: Модель дивиться на глобальний контекст, а не лише на регіони. Ефективніша (особливо на відео).

Для розуміння технології глибокого сортування (DeepSORT), слід спочатку розібратися з парою математичних аспектів - відстань Махалонобіса та фільтр Калмана. [29]

Розглянемо дуже простий приклад, щоб інтуїтивно зрозуміти, що таке відстань Махалонобіса і навіщо вона потрібна. Багатьом, напевно, відомо, що таке евклідова відстань. Зазвичай, це відстань від однієї точки до іншої в евклідовому просторі(рис 2.3):

$$d(p,q) = \sqrt{\{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2\}} = \sqrt{\left\{\sum_{\{k=1\}}^{\{n\}(p_k - q_k)^2}\right\}} \quad (2.4.)$$

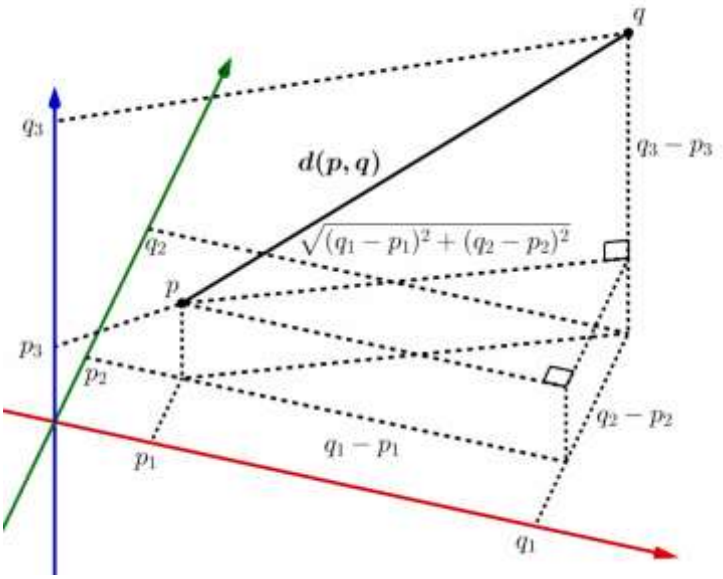


Рис 2.3 - Відстань Махалонобіса

Що насправді робить відстань Махалонобіса:

- Позбавляється коваріації змінних
- Робить дисперсію (variance) змінних рівною 1
- Після цього використовує звичайну евклідову відстань для трансформованих даних

Для подальшого розуміння технології DeepSORT потрібно розібрати фільтр Калмана. За своєю суттю фільтр Калмана рекурсивно оперує потоками зашумлених вхідних даних, щоб отримати статистично оптимальну оцінку основного стану системи. Він працює у двоетапному циклі:

1. Прогнозування: На основі попередньої оцінки стану та моделі того, як система змінюється з часом (модель процесу), фільтр прогнозує

наступний стан системи. Цей прогноз за своєю суттю містить певну невизначеність.

2. Оновлення: Фільтр включає нове вимірювання (яке також має шум і невизначеність), пов'язане з поточним станом. Фільтр порівнює вимірювання з прогнозованим станом і оновлює оцінку, надаючи більшу вагу або прогнозу, або вимірюванню, виходячи з їхньої відповідної невизначеності. Результатом є вдосконалена оцінка стану, яка є статистично кращою (ближчою до істинного стану), ніж або передбачення, або лише вимірювання.

Отже, ми тепер знаємо, що таке фільтр Калмана і відстань Махалонобіса. Технологія DeepSORT просто пов'язує ці два поняття між собою для того, щоб переносити інформацію від одного кадру до іншого, і додає нову метрику, під назвою вигляд. За допомогою розпізнавання визначаються позиція, розмір і клас одної обмежувальної рамки.

2.3. Візуальні алгоритми штучного інтелекту YOLO.

«Conv» - це математична операція, яка поєднує дві функції для створення третьої. У комп'ютерному зорі та обробці сигналів згортка часто використовується для застосування фільтрів до зображень або сигналів, виділяючи певні шаблони(pattern). У згорткових нейронних мережах (CNN) згортка використовується для вилучення ознак з вхідних даних, таких як зображення. Згортання структурується ядрами (K), стрічками (s) і прокладками (p). [31]

На рисунку 2.4 , взятому з сайту розробників Nvidia, представлено реальний випадок застосування згортки для вилучення ознаки.

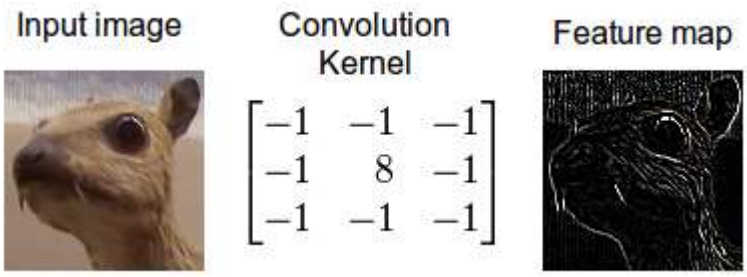


Рис 2.4 - Приклад застосування згорток(Kernel)

«Kernel», також відоме як фільтр, - це невеликий масив чисел, який ковзає по входу (зображенню або сигналу) під час операції згортки. Мета полягає в тому, щоб застосувати локальні операції до вхідних даних для виявлення певних характеристик. Кожен елемент в ядрі представляє вагу, яка множиться на відповідне значення на вході під час згортки. Нижче на Рисунку 2.5 наведено кілька прикладів.

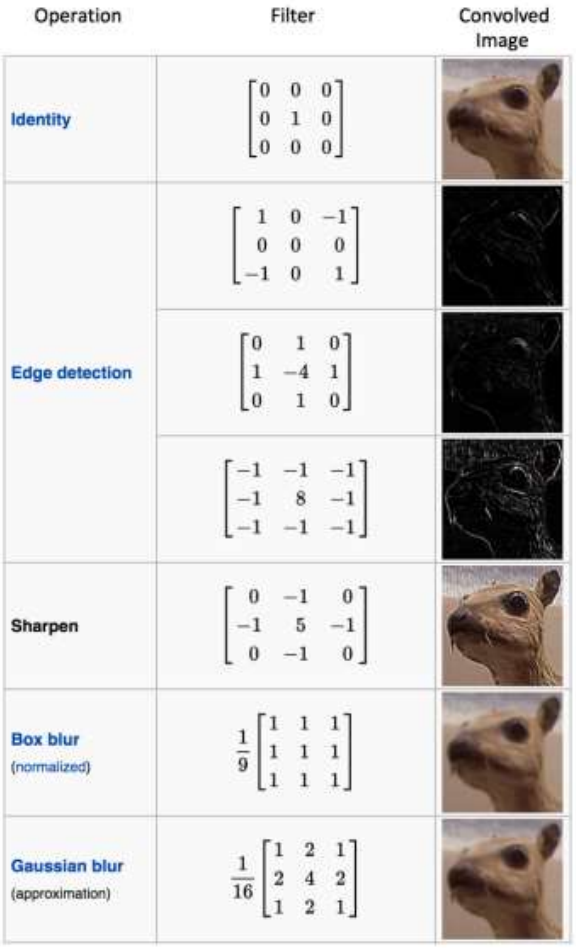


Рис 2.5 - Приклади фільтра

«Stride» - це величина зсуву, якого зазнає ядро, коли рухається по вхідним даним під час згортки. Крок 1 означає, що ядро переміщується на одну позицію за раз, тоді як крок 2 означає, що ядро пропускає дві позиції при кожному переміщенні. Крок безпосередньо впливає на просторові розміри результату згортки. Більший крок може зменшити розмірність результату, тоді як менший крок зберігає більше просторової інформації. Більші кроки зменшують обчислювальні зусилля, тим самим збільшуючи

швидкість операції, що може безпосередньо впливати на якість... На рисунку 2.6 нижче ядро, позначене червоним кольором, проходить карту пікселів зображення з кроком 1. [32]

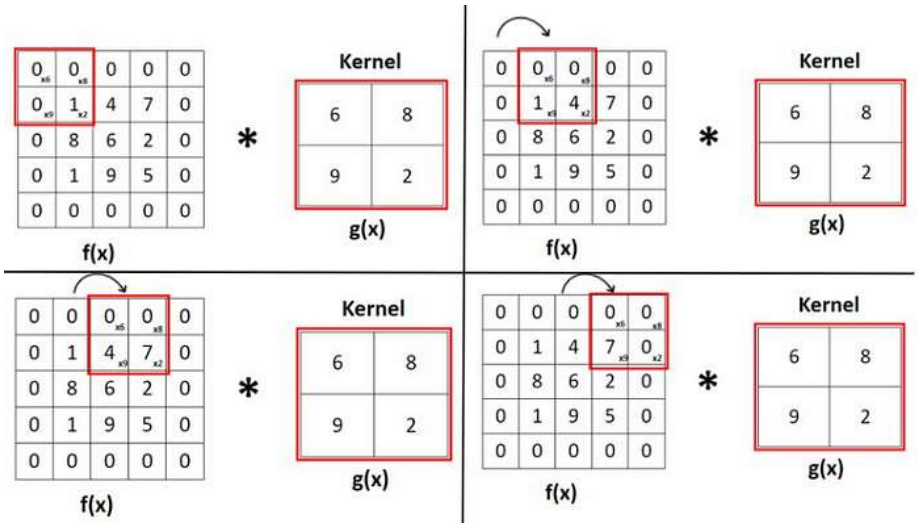


Рис 2.6 - Переміщення на 1 крок

«Padding» означає додавання додаткових пікселів по краях вхідного зображення (зазвичай нулів) перед застосуванням операцій згортки. Це робиться для того, щоб гарантувати, що інформація на краях зображення обробляється так само, як і інформація в центрі під час операцій згортки.

Коли фільтр (ядро) застосовується до зображення, він зазвичай проходить через зображення піксель за пікселем. Якщо не застосовано операцію «padding», пікселі на краях зображення мають менше сусідів, ніж пікселі в центрі, що може призвести до втрати інформації в цих регіонах. На рисунку 2.7 нижче наведено приклад застосування додавання додаткових пікселів по краях вхідного зображення. [33]

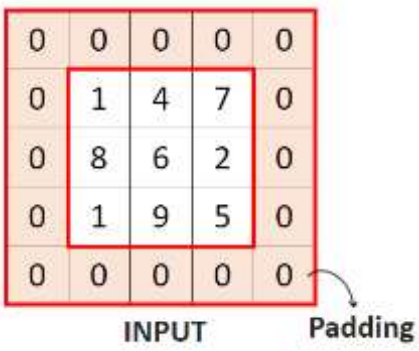


Рис 2.7 - Приклад застосування додавання додаткових пікселів по краях вхідного зображення

РОЗДІЛ 3 ПІДГОТОВКА ДАНИХ ТА ВИКОНАННЯ ЗАВДАННЯ

3.1. Підготовка середовища та даних для нейронної мережі YOLO.

Першим завданням при виконанні даної роботи є збір даних та підготовка середовища для її використання, в результаті, отримання ваг для можливості розпізнавання об’єктів.

Використовуючи свій створений набір даних (зображення для проведення навчання). Отже спочатку потрібно розкадрувати відео на фотознімки для проведення майбутніх робіт з ними.

Після створення набору зображень переходимо до їх анотування. Для цього після завантаження і підтвердження додавання зображень переходимо в LabelImg та починаємо анотувати наші зображення (рис. 3.1). Для навчання мережі рекомендується брати для навчання 100 зображень з 1000 анотованих об’єктів, але в даній роботі буде достатньо лише 50 зображень через роботу лише з одним відео з якого й будуть створюватися дані для навчання, а також визначення лише сімнадцяти класів при тренуванні.



Рис. 3.1 Вікно анотації зображень

Далі створюємо набір даних з оброблених зображень. Для цього необхідно буде обрати пропорції поділу між папкою val та train. До кожного зображення створюється файл з описом рамок (рис. 3.2).

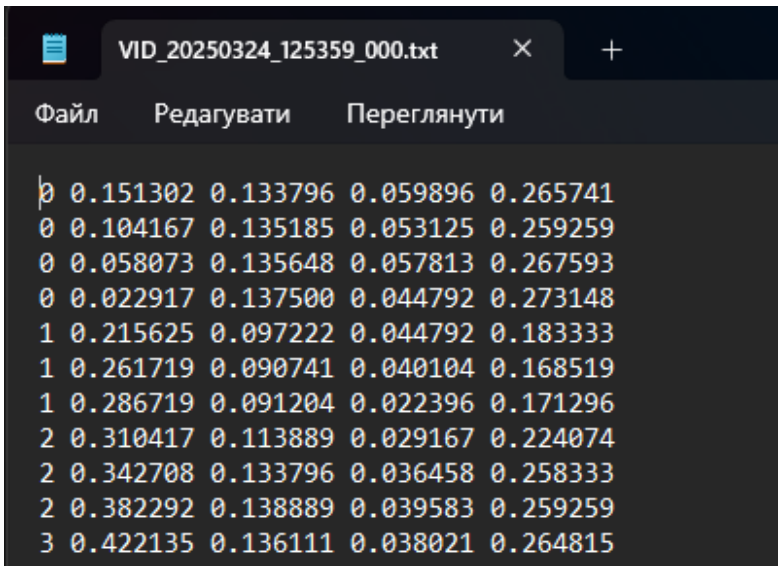


Рис. 3.2 - Текстовий опис рамок об’єкта

У роботі [1] описано що у задачах машинного навчання прийнято ділити датасет на 80/20, з таким розподілом:

80% -для тренування (train) -щоб модель побачила достатньо варіацій і навчилася загальним шаблонам.

20% -для валідації (val) -щоб відслідковувати якість на «незнайомих» прикладах.

Коротке пояснення чому саме 80/20, це прояв принципу Парето, також відомого як правило 80/20. Цей принцип, відкритий італійським економістом Вільфредо Парето на початку 20 століття, став потужним аналітичним інструментом у різних сферах, особливо в бізнесі та науці про дані.

В основі принципу Парето лежить теза про те, що в багатьох ситуаціях приблизно 80% наслідків зумовлені 20% причин. Хоча на практиці ці відсотки можуть дещо відрізнятися, принцип підкреслює існування нерівномірного розподілу в системах, де невелика частка входів часто непропорційно впливає на результати.

Цей принцип - не закон природи, а скоріше евристика - емпіричне правило, яке забезпечує цінну перспективу для прийняття рішень і

визначення пріоритетів. Він заохочує професіоналів зосереджувати свої зусилля на «життєво важливих кількох», а не на «тривіальних багатьох».

Наступним кроком постає завдання проведення тренування за допомогою YOLO. Для цього створюємо файл Google Colab в Google Drive (рис. 3.3) .

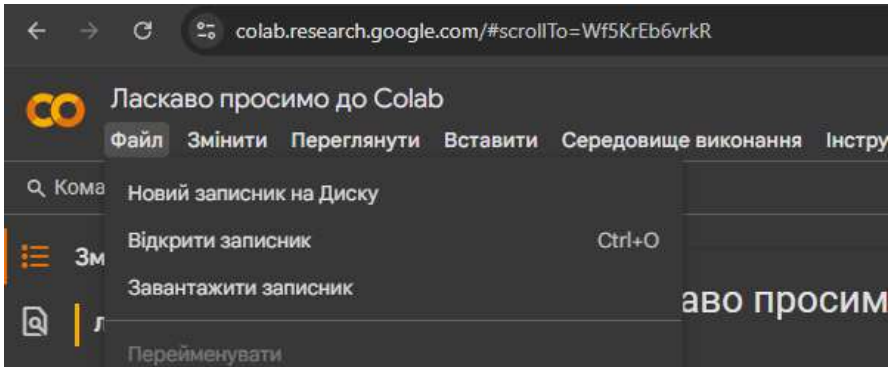


Рис. 3.3 Приклад створення файлу Colab в Google Drive

Змінюємо апаратне прискорення на T4 або A100.

Tesla T4 - це графічна карта на основі архітектури NVIDIA Turing, призначена для прискорення виведення моделей глибокого навчання.

Графічний процесор A100 Tensor Core - це потужна платформа для центрів обробки даних, призначена для штучного інтелекту, аналізу даних та високопродуктивних обчислень . Вона базується на архітектурі NVIDIA Ampere і пропонує значний приріст продуктивності порівняно з попередніми поколіннями.

Далі імпортуємо репозиторій YOLO з GitHub (на рис. 3.4 зображена рядок коду для виконання даного завдання). Після цього також проводиться імпорт даних, які використовуються для тренування мережі ,а також ваги yolov7 на основі яких буде проводитись тренування (посилання на завантаження можна знайти в репозиторії YOLOv7 на GitHub).



Рис. 3.4 - Рядок коду для імпорту репозиторію YOLOv7

Далі, для зручності подальшої роботи з файлами, заходимо в папку yolov7 (рис. 3.5). Потім встановлюємо необхідні бібліотеки за допомогою, наявного в папці yolov7 (рис. 3.6).

```
%cd /content/yolov7
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_training.pt
```

Рис. 3.5 - Рядок коду для переходу в папку yolov7

```
!git clone https://github.com/WongKinYiu/yolov7
%cd yolov7
!pip install -r requirements.txt
```

Рис. 3.6 - Рядок коду для встановлення додаткових бібліотек

Далі необхідно внести зміни до файлів для відповідності тренуванню, а саме: до вмісту файлу coco.yaml (назву змінюємо custom-data.yaml), місцезнаходження якого yolov7/data/, вносяться зміни, зображені на рис. 3.7; в файлі обраної моделі yolov7.yaml (шлях до файлу - /yolov7/cfg/training/) вноситься зміна до кількості класів (зображено на рис. 3.8).

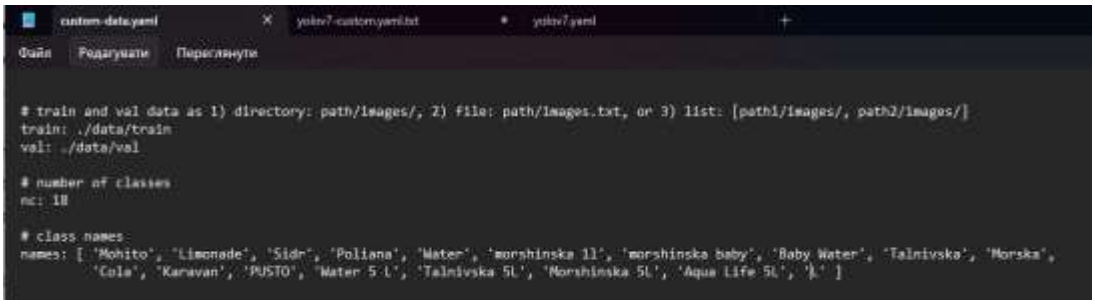


Рис. 3.7 Фінальний вигляд файлу custom-data.yaml



Рис. 3.8 Фінальний вигляд файлу yolov7.yaml

3.2. Тренування нейронної мережі YOLO.

Після підготовки даних маємо таке дерево-видну структуру папок(рис 3.9).

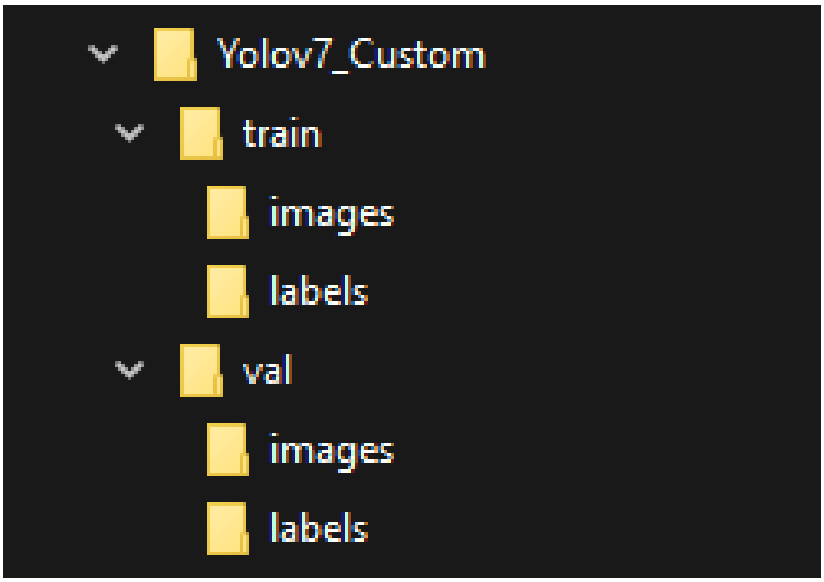


Рис 3.9 – Дерево-видна структура папок

Отже тепер можливо починати тренування. Для цього вводимо рядок !python train.py та в тому ж рядку визначаємо необхідні змінні, що передаються в цей файл, одними з яких є:

1. img – роздільна здатність наданих зображень;
2. workers – скільки під процесів розпаралелювати під час навчання;
3. batch_size – визначає кількість зразків оброблених до створення оновленої моделі;
4. epochs – кількість епох для тренування, а саме це значення відповідає загальній кількості проходів, яке модельне навчання пройде по всьому набору даних;
5. hyp – файл YAMP, який описує вибір гіперпараметрів для навчання проекту;
6. weights – попередньо натренеровані ваги, з яких ми хочемо знову почати навчання(у випадку відсутності ваг ставимо значення «’ ’»);
7. cfg – файл конфігурації архітектури моделі;

8. data – файл DATA YAMP містить інформацію про набір даних, щоб допомогти керувати YOLO.

Одним із ключових параметрів при навчанні нейронної мережі YOLO є кількість епох (epochs), яка визначає, скільки разів модель повністю пройде через увесь навчальний набір. У ході роботи з розпізнаванням товарів на відео було здійснено кілька спроб тренування з різною кількістю епох, щоб визначити найоптимальніший варіант для досягнення високої точності.

На початковому етапі було обрано значення 100 епох. Це стандартна кількість, яка зазвичай використовується для базового тренування, особливо коли обсяг даних є обмеженим або ще не до кінця оптимізовано структуру набору даних. Після завершення навчання й застосування отриманих ваг на тестовому відео з'ясувалося, що модель демонструє низький рівень точності розпізнавання.

Ці результати вказували на недостатню кількість епох, яка не дозволила моделі повноцінно навчитися розрізняти подібні між собою об’єкти.

Наступний етап передбачав збільшення кількості епох до 200. Після завершення навчання за оновленим параметром результати покращилися. Модель стала впевненіше розпізнавати товари на полицях, демонструючи вищу точність локалізації та класифікації.

Однак залишався помітний недолік помилкове злиття об’єктів. У випадках, коли два однакових товари знаходилися поруч і один частково перекривав інший, модель визначала їх як один об’єкт. Це спричиняло зниження точності підрахунку товарів, особливо у випадках, коли йшлося про аналіз залишків або перевірку наявності.

Останній етап передбачав подальше збільшення кількості епох до 300, що дозволило досягти найбільш стабільних і точних результатів. Модель змогла глибше навчитися наданому датасету, краще опрацьовувала складні сцени та коректніше розрізняла об'єкти навіть у випадках часткового перекриття.

Таким чином, аналіз навчання з різною кількістю епох продемонстрував, що збільшення епох до 300 суттєво впливає на якість розпізнавання. Модель отримала достатню кількість ітерацій для «вивчення» особливостей кожного товару, що дозволило досягти високої точності.

Також потрібно звернути увагу на формат виведення результатів тренування після кожної епохи(Рис. 3.10 – 3.12).

Epoch	gpu_mem	box	obj	cls	total	labels	img_size
99/99	13.4G	0.03715	0.0538	0.02827	0.1192	42	640: 100% 4/4 [00:03<00:00, 1.11it/s]
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95: 100% 1/1 [00:01<00:00, 1.93s/it]	
all	8	376	0.455	0.474	0.39	0.255	
Mohito	8	40	0.202	0.789	0.374	0.242	
Limonade	8	39	0.247	0.769	0.428	0.281	
Sidr	8	24	0.248	0.625	0.416	0.3	
Poliana	8	16	0.075	0.0281	0.1	0.0657	
Water	8	8	0	0	0	0	
morshinska 1l	8	8	1	0	0	0	
morshinska baby	8	46	0.809	0.978	0.988	0.607	
Baby Water	8	15	0.135	0.267	0.0841	0.0432	
Talnivska	8	16	0.539	0.25	0.368	0.245	
Morska	8	16	1	0	0	0	
Cola	8	16	0	0	0.0607	0.0386	
Karavan	8	47	0.229	1	0.97	0.615	
PUSTO	8	24	1	0.453	0.547	0.177	
Water 5 L	8	29	0.321	1	0.814	0.545	
Talnivska 5L	8	8	0.142	0.5	0.139	0.0951	
Morshinska 5L	8	8	0.972	0.875	0.944	0.783	
Aqua Life 5L	8	8	0.265	1	0.71	0.497	

Рис. 3.10 - Приклад результатів тренування на 100 епохах

Epoch	gpu_mem	box	obj	cls	total	labels	img_size
199/199	13.7G	0.02914	0.0717	0.02195	0.1228	127	640: 100% 4/4 [00:03<00:00, 1.26it/s]
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95: 100% 1/1 [00:02<00:00, 2.29s/it]	
all	8	376	0.814	0.631	0.766	0.551	
Mohito	8	40	0.555	0.95	0.915	0.612	
Limonade	8	39	0.525	0.897	0.81	0.582	
Sidr	8	24	0.538	1	0.939	0.614	
Poliana	8	16	1	0	0.501	0.348	
Water	8	8	1	0	0.126	0.0754	
morshinska 1l	8	8	1	0	0.146	0.0899	
morshinska baby	8	46	0.865	0.972	0.968	0.62	
Baby Water	8	15	0.475	0.733	0.818	0.587	
Talnivska	8	16	0.717	1	0.976	0.744	
Morska	8	16	0.92	0.875	0.938	0.661	
Cola	8	16	1	0	0.419	0.272	
Karavan	8	47	0.959	0.979	0.996	0.711	
PUSTO	8	24	0.837	0.856	0.941	0.628	
Water 5 L	8	29	0.933	1	0.996	0.74	
Talnivska 5L	8	8	0.83	0.625	0.768	0.609	
Morshinska 5L	8	8	0.491	1	0.996	0.868	
Aqua Life 5L	8	8	1	0.463	0.886	0.698	
L	8	8	1	0	0.642	0.539	

200 epochs completed in 0.465 hours.

Рис. 3.11 - Приклад результатів тренування на 200 епохах

```
      Epoch    gpu_mem       box       obj       cls     total    labels  img_size
    299/299      13.4G   0.01955    0.0434  0.009667   0.07262         0       640: 100% 4/4 [00:03<00:00,  1.19it/s]
                 Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100% 1/1 [00:03<00:00,  3.37s/it]
                   all          8        376      0.933      0.857      0.959      0.723
                Mohito          8         40       0.89        0.9      0.969      0.731
              Limonade          8         39          1       0.94      0.985      0.721
                  Sidr          8         24      0.976          1      0.996       0.72
               Poliana          8         16      0.933      0.872      0.955      0.744
                 Water          8          8          1          0      0.693      0.531
      morshinska 1l             8          8          1          0       0.78      0.536
    morshinska baby             8         46      0.952          1      0.997      0.723
         Baby Water             8         15      0.935          1      0.992      0.666
          Talnivska             8         16      0.982          1      0.996       0.81
             Morska             8         16          1      0.758      0.992       0.75
               Cola             8         16      0.966          1      0.996      0.696
            Karavan             8         47       0.96          1      0.997      0.734
               PUSTO            8         24      0.903      0.958      0.967      0.662
          Water 5 L             8         29      0.964          1      0.996       0.77
        Talnivska 5L            8          8        0.8          1      0.971      0.841
       Morshinska 5L            8          8       0.95          1      0.996      0.798
        Aqua Life 5L            8          8      0.683          1      0.982      0.779
                  L             8          8      0.901          1      0.995        0.8
300 epochs completed in 0.662 hours.
```

Рис. 3.12 - Приклад результатів тренування на 300 епохах

В цих результатах:

1. P (Precios - точність) – визначає наскільки точно модель класифікує об’єкти під час навчання;

2. R (Recall - повнота) – визначає наскільки ефективно модель виявляє всі наявні об’єкти в зображенні;

3. mAP (mean Average precision – середня точність) – вимірює якість детектування об’єктів і обчислюється як середнє значення показника точності для різних значень порогу впевненості моделі в присутності об’єкту.

Середнє значення	100 Епох	200 Епох	300 Епох
P	0.424	0.858	0.951
R	0.501	0.665	0.904
mAP	0.441	0.808	1

На наданих вище прикладах результатів видно що mAP є зі значенням 0.5 та 0.5:0.80. різниця між ними полягає в тому, що в першому випадку визначається середня точність для порогу 0.5, а в другому розглядається ширший діапазон порогів, тобто починаючи з 0.5 і збільшуючи його (з кроком 0.05) до 0.90, для кожного порогу обчислюється точність розпізнавання.

Потім обчислюється середнє значення точності детектування для всіх порогів у діапазоні 0.7-0.95. На рис. 3.13 зображено зміни mAP даних, що демонструє вплив епох на дані значення.

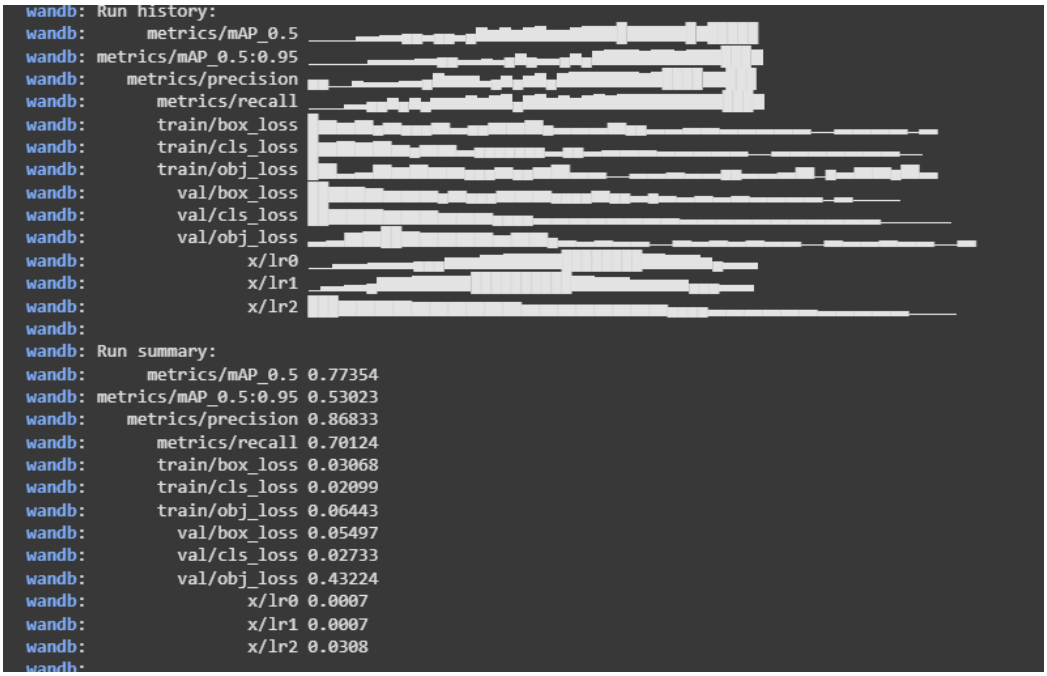


Рис. 3.13 - Зміни mAP

3.3. Тестування підсистеми для підрахунку наявності товару.

У межах реалізації розпізнавання товарів на відео було застосовано модифікований підхід до розпізнавання та підрахунку об’єктів на основі результатів роботи нейронної мережі YOLO. Основним завданням на цьому етапі було розпізнавання об’єктів на кожному кадрі відео з подальшим коректним підрахунком їх кількості.

Під час розпізнавання виконується поетапний прохід по кожному розпізнаному об’єкту на кадрі. Для кожного з них здійснюється порівняння з об’єктами на попередньому кадрі - визначається найближчий за координатами. У разі, якщо відстань між двома об’єктами не перевищує встановленого порогового значення, поточний об’єкт вважається тим самим, що й на попередньому кадрі, і записується до загального текстового файлу з його класом.

Цей механізм дозволяє ефективно відстежувати рух об’єктів у кадрі. Для реалізації підрахунку було впроваджено умовну лінію (counting line),

відносно якої визначається напрям руху товару. Кожному об’єкту призначається логічний статус залежно від його розташування: до перетину або після перетину лінії. У процесі наступного розпізнавання статус об’єкта успадковується, що дозволяє уникнути повторного врахування одного й того ж елемента.

У момент, коли об’єкт перетинає встановлену лінію, його статус змінюється на нейтральний, і в підсистемі збільшується лічильник розпізнаних товарів. Таким чином, кожен об’єкт враховується лише один раз, що забезпечує достовірність результатів. Далі запускається файл detect.py кодом.

В результаті отримуємо вивід на консолі (рис. 3.14).

```
[ ]  !python detect.py --weights best.pt --conf 0.5 --img-size 640 --source video.mp4 --no-trace

Namespace(weights=['best.pt'], source='video.mp4', img_size=640, conf_thres=0.5, iou_thres=0.45, device='', v
YOLOR 🚀 v0.1-128-ga207844 torch 2.0.0+cu117 CUDA:0 (Tesla T4, 15095.0625MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
IDetect.fuse
/usr/local/lib/python3.11/dist-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 314 layers, 36573470 parameters, 6194944 gradients, 103.4 GFLOPS
```

Рис. 3.14 Результат виконаної команди

Для самого підрахунку було задіяно такий модуль Ultralytics. За допомогою нього була проведена модернізація коду для підрахунку кількості товару(рис 3.15). Також був використаний модуль numpy для роботи з масивами.

```
model = YOLO('yolov8n.pt')
        video_path = 'video.mp4'
cap = cv2.VideoCapture(video_path)

while True:
    ret, frame = cap.read()
    if not ret:
        break
            results = model(frame)[0]
    total_objects = len(results.boxes)
           class_ids = results.boxes.cls.cpu().numpy()
    num_water = sum(class_ids == 0)
```

Рис. 3.15 – Код для підрахунку товару

Таблиця 3.1 Пояснення коду

Model=YOLO	Завантажує модель
cap = cv2.VideoCapture	Відкриває відео
cap.read()	Зчитує кадр з відео
model(frame)	Виконує розпізнавання об’єктів
class_ids	Містить класи виявлених об'єктів
sum(class_ids == X)	Рахує скільки об'єктів класу X

Детальний лістинг коду наведено в Додатку А.

На рис. 3.16 продемонстровано результат виконання на 100 епохах, а саме розпізнавання товару на полиці магазину.



Рис 3.16 – Результат із тренуванням в 100 епох

При всьому цьому, середня точність коливається але не є поганою, не враховуючи деякі малі об’єкти (точність коливається від 0.7 до 0.75 враховуючи що при розпізнаванні був встановлений поріг 0.70).

Рис. 3.17 демонструє результат виконання на 200 епохах.



Рис 3.17 - Результат із тренуванням в 200 епох

Тут вже середня точність є доволі не погана, не враховуючи об’єкти які знаходять в неосвітленому місця (точність коливається від 0.75 до 0.95).

Рис. 3.18 демонструє вдале виконання завдання.



Рис 3.18 - Результат із тренуванням в 300 епох

Отриманого результату розпізнавання об’єктів достатньо для виконання завдання. Отже порівняємо точність серед результатів.

Таблиця 3.2 Порівняння кінцевого результату

Результат	Кількість товарів при 100 епох (рис 3.16)	Кількість товарів при 200 епох (рис 3.17)	Кількість товарів при 300 епох (рис 3.18)
morshinska baby	3	10	6
Karavan	7	9	7
Water-5L	6	5	9
Mohito	0	2	6
Limonade	0	1	6
Baby Water	0	3	4
Sidr	0	0	4
Poliana	0	0	5
Morshinska 1L	0	0	2
Talnivska	0	0	4
Cola	0	0	2

У ході тестування інформаційної підсистеми для підрахунку кількості товарів було проведено порівняння результатів розпізнавання товарів за допомогою моделі нейронної мережі YOLO на трьох етапах навчання: після 100, 200 та 300 епох. Відповідні результати подано в таблиці 3.2.

Метою аналізу було визначити, як збільшення кількості епох впливає на точність виявлення об’єктів, представлених у вигляді різних категорій товарів.

У результаті аналізу можна помітити загальну позитивну динаміку: зі зростанням кількості епох модель демонструє кращу здатність до виявлення об’єктів. Зокрема, при 100 епохах модель розпізнавала лише обмежену кількість товарів - переважно ті, що були найбільш чітко видимі або представлені у великій кількості в навчальній вибірці.

Наприклад, товари «Karavan» і «Water-5L» були виявлені вже на цьому етапі (відповідно 7 і 6 одиниць), що свідчить про їх добру представленість або характерні візуальні ознаки. Найбільше покращення спостерігається на рівні 200 епох, де модель змогла значно краще розпізнати багато категорій.

Зокрема, кількість розпізнаних об’єктів «morshinska baby» зросла з 3 до 10, а «Baby Water» з 0 до 3. Це свідчить про те, що модель на цьому етапі почала краще узагальнювати та виділяти менш очевидні шаблони в даних. Варто зазначити, що на цьому етапі ще не всі класи товарів були впевнено розпізнані, що вказує на потребу в подальшому навчанні. Після 300 епох результати стали ще ширшими — модель почала виявляти нові категорії, які не були розпізнані раніше.

Наприклад, товари «Sidr», «Poliana», «Talnivska», «Morshinska 1L» та «Cola» не були виявлені ні при 100, ні при 200 епохах, однак після 300 епох кількість їх розпізнань становила від 2 до 5 одиниць. Це свідчить про те, що глибше навчання дозволило моделі краще диференціювати об'єкти, навіть якщо вони мають складніші або менш виражені візуальні характеристики. Проте слід зауважити, що не в усіх випадках результати 300 епох були кращими за результати 200. Наприклад, кількість розпізнаних об’єктів «morshinska baby» зменшилась із 10 до 6, а «Karavan» - з 9 до 7. Це може бути наслідком перенавчання моделі або статистичних коливань, викликаних особливостями самої тестової вибірки. Така поведінка є типовою для моделей глибокого навчання і вказує на необхідність впровадження регуляризації або використання механізмів ранньої зупинки під час тренування.

Отже, за результатами аналізу можна зробити висновок, що збільшення кількості епох у загальному позитивно впливає на точність розпізнавання, особливо для менш поширених або складних для виявлення об’єктів. Найкращий баланс між точністю та загальною здатністю до генералізації спостерігався на рівні 300 епох.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи було досліджено та реалізовано один з найсучасніших та найбільш ефективних підходів до задач комп’ютерного зору - моделі YOLO. Цей підхід є особливо актуальним у контексті розпізнавання товарів на полицях, де критично важливими є як висока точність, так і швидкість роботи алгоритму в режимі реального часу.

Було проаналізувано архітектуру нейронної мережі YOLO, звертаючи особливу увагу на математичні принципи, що лежать в її основі. Одним із ключових елементів цього аналізу стало вивчення функції втрат, яка включає в себе декілька важливих складових: координатну (для точного визначення місця розташування об'єкта), розмірну (що дозволяє правильно масштабувати виявлені об'єкти), класифікаційну (що відповідає за правильне розпізнавання категорії товару), а також впевненісну (confidence score), яка є критичною для фільтрації хибнопозитивних виявлень.

З практичного боку, було створено набір даних, що відповідає завданням розпізнавання об’єктів на полицях магазинів. Датасет було ретельно анотовано, після чого виконано його поділ на тренувальну та валідаційну вибірки у пропорції 80/20, як це рекомендується у більшості наукових публікацій та практичних гайдлайнів з машинного навчання. Далі було проведено навчання нейронної мережі YOLO на зібраних даних, ретельно налаштовуючи параметри та оптимізуючи процес тренування з метою досягнення найкращих результатів.

Після завершення етапу навчання виконавши візуалізацію отриманих результатів отримали відповіді, що дозволили наочно оцінити здатність моделі до точного розпізнавання товарів. Precision (точність) складає 0.951, Recall (повнота) –0.904 та mAP (mean Average Precision — середнє значення точності по всіх класах) – 1 .

Проведений аналіз показав, що після 100 епох модель змогла розпізнати всього 16 товарів. Це досить обмежений результат, при якому

виявленими були лише окремі, найбільш помітні класи. Після 200 епох кількість розпізнаних товарів зросла до 43. Це означає, що модель значно краще навчилась і почала ідентифікувати більше об’єктів на зображеннях. Після 300 епох результат досягнув 59 розпізнаних товарів, що є найвищим показником серед трьох варіантів навчання. Також простежується позитивна динаміка щодо охоплення класів. Якщо після 100 епох модель не розпізнала жодного товару у 7 категоріях із 11, то після 200 епох кількість таких категорій зменшилась до 5. Після 300 епох модель змогла виявити принаймні один об’єкт у кожному класі. Проведений аналіз показав, що реалізована модель демонструє високий рівень ефективності, а також є придатною для використання в умовах реального часу – там, де швидкість обробки відеопотоку чи зображень є важливою.

Реалізована підсистема може бути застосована у реальних комерційних та дослідницьких проєктах, що пов’язані з автоматизацією торгівлі, моніторингом розкладки товарів, логістикою або іншими сферами, де є потреба в ефективному розпізнаванні та підрахунку об’єктів на зображеннях.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. https://www.nb-data.com/p/pareto-principle-in-machine-learning
2. Piotr Dollar, Mannat Singh, and Ross Girshick. Fast and ´ accurate model scaling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 924–932, 2021.
3. Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021. arXiv preprint arXiv:2107.08430, 2021.
4. Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10886–10895, 2021.
5. Mu Hu, Junyi Feng, Jiashen Hua, Baisheng Lai, Jianqiang Huang, Xiaojin Gong, and Xiansheng Hua. Online convolutional re-parameterization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.
6. Shuvo Kumar Paul, Muhammed Tawfiq Chowdhury, Mircea Nicolescu, Monica Nicolescu, and David FeilSeifer. Object detection and pose estimation from rgb and depth data for real-time, adaptive robotic grasping. In Advances in Computer Vision and Computational Biology, pages 121–142. 2021.
7. Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. BYTETrack: Multi-object tracking by associating every detection box. arXiv preprint arXiv:2110.06864, 2021.
8. Benjin Zhu, Jianfeng Wang, Zhengkai Jiang, Fuhang Zong, Songtao Liu, Zeming Li, and Jian Sun. AutoAssign: Differentiable label assignment for dense object detection. arXiv preprint arXiv:2007.03496, 2020.
9. Zetong Yang, Yin Zhou, Zhifeng Chen, and Jiquan Ngiam. 3D-MAN: 3D multi-frame attention network for object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 1863–1872, 2021.

10. Mingxing Tan and Quoc Le. EfficientNetv2: Smaller models and faster training. In International Conference on Machine Learning (ICML), pages 10096–10106, 2021.

11. X. Chen, R. B. Girshick, K. He, and P. Doll&39;ar. Tensormask: A foundation for denseobject segmentation. CoRR, abs/1903.12174, 2019

12. E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmen-tation. IEEE Trans. Pattern Anal. Mach. Intell., 39(4):640–651, 2017.

13. R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. CoRR, abs/1603.08511,2016.

14. F. Sultana, A. Sufian, and P. Dutta. A review of object detection models based on convo-lutional neural network. CoRR, abs/1905.01614, 2019.

15. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale imagerecognition. CoRR, abs/1409.1556, 2014.

16. Chung, Junyoung, et al. "Gated feedback recurrent neural networks." CoRR, abs/1502.02367 2019.

17. Shah, Rajiv, and Rob Romijnders. "Applying Deep Learning to Basketball Trajectories." arXiv preprint arXiv:1608.03793 2019.

18. Chu, Hang, Raquel Urtasun, and Sanja Fidler. "Song From PI: A Musically Plausible Network for Pop Music Generation." arXiv preprint arXiv:1611.03477 2019.

19. Zheng, Shuai, et al. "Conditional random fields as recurrent neural networks." Proceedings of the IEEE International Conference on Computer Vision. 2019.

20. Baccouche, Moez, et al. "Action classification in soccer videos with long short-term memory recurrent neural networks." International Conference on Artificial Neural Networks. Springer Berlin Heidelberg, 2019.

21. OpenCV, Open source Computer Vision library. In http://opencv.willowgarage.com/wiki/, 2019.

22. Facial Recognition using OpenCV, Shervin EMAMI1, Valentin Petruț SUCIU2.,www.jmeds.eu.

23. Ammar Anuar, Khairul Muzzammil Saipullah, Nurul Atiqah Ismail, Yewguan Soo "OpenCV Based Real-Time Video Processing Using Android Smartphone" , IJCTEE, Volume 1, Issue 3

24. Microsoft. Code of Conduct. Microsoft Open Source. url: https://opensource.microsoft.com/ codeofconduct/ 2020.

25. Stack Overflow Team. Stack Overflow Developer Survey 2019. Stack Overflow. 2019.

26. VSCodium. Vscodium. VSCodium, Nov. 24, 2020.

27. Erich Gamma, Asim Hussain, and Simona Cotin. The History of Visual Studio Code. Microsoft Channel 9. May 26, 2020.

28. Mary Jo Foley. Microsoft's Browser-Based Dev Toolbox: How 'Monaco' Came to Be. ZDNet. Nov. 18, 2023.

29. The VS Code Team. Visual Studio Code 1.0 Release. Visual Studio Code. Apr. 14, 2016.

30. Microsoft Developer, director. Ask the Team: Visual Studio Code — COM01. May 25, 2020.

31. Bird, S., Ellison, N. B., & Klein, D. (2020). The rise of Python: A survey of recent research. ACM Computing Surveys, 53(5), 1-36.

32. Géron, A. (2019). Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

33. McKinney, W. (2017). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

34. VanderPlas, J. (2020). Python data science handbook: Essential tools for working with data. O'Reilly Media.

35. A. Griewank. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2020.

36. Eric Battenberg, Sander Dieleman, Daniel Nouri, Eben Olson, Aaron van den Oord, Colin Raffel, Jan Schluter, and Soren Kaae Sonderby. Lasagne: Lightweight library to build and train neural networks in theano, 2016.

37. Fran¸cois Chollet. Keras: Deep learning library for theano and tensorflow, 2016.

38. R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2016.

39. Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and Tyler Hunt. caret: Classification and Regression Training, 2016.

40. Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pages 7–10. ACM, 2016.