Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконала: студентка 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Новіцька О. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: к.т.н., проф. каф. КСУ
Биков М. М.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри _____ Яровий А.А
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Новіцькій Оксані Вікторівні

1. Тема роботи: Програмний модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі.
керівник роботи: Паночишин Юрій Миколайович, к.т.н., доц.
затверджені наказом вищого навчального закладу "11" 03 2024 року №80

2. Термін подання студентом роботи 27. 05. 2024

3. Вихідні дані до роботи :
кількість класів продуктів харчування – не менше 100; обсяг набору даних для навчання та тестування – не менше 10000 зображень; використання об'єктноорієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи розпізнавання продуктів харчування та вибрати найбільш ефективний;
вибрати нейронну мережу, спроектувати її структуру та метод навчання
розробити алгоритм роботи модуля розпізнавання продуктів харчування;
спроектувати та реалізувати модуль розпізнавання продуктів харчування за допомогою згорткової нейронної мережі;
протестувати роботу програми розпізнавання продуктів харчування на основі згорткової нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Схема алгоритму роботи програми
Структура нейронної мережі
Характеристики набору даних
Результати роботи програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Паночишин Ю.М., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07.03.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	6.05.24	10.05.24	
2	Обґрунтування методу розв'язання задачі, проектування програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі	11.05.24	15.05.24	
3	Програмна реалізація модуля розпізнавання продуктів харчування	16.05.24	20.05.24	
4	Аналіз результатів тестування модуля розпізнавання продуктів харчування	21.05.24	24.05.24	
5	Розробка інструкції користувача	25.05.24	27.05.24	
6	Оформлення матеріалів до захисту БДР	28.05.24	6.06.24	

Студентка _____ **Новіцька О. В.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Паночишин Ю. М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Новіцька О. В. Програмний модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі. Бакалаврська дипломна робота складається з 68 сторінок формату А4, на яких є 15 рисунків, 2 таблиці, список використаних джерел містить 13 найменувань.

Метою роботи є підвищення якості розпізнавання продуктів харчування за рахунок використання згорткової штучної нейронної мережі.

У роботі було розроблено метод розпізнавання продуктів харчування на основі архітектури згорткової нейронної мережі Mask R-CNN. Навчання проводилось за допомогою стохастичного градієнтного спуску по мінібатчам на швидкостях навчання 1E-4, 1E-5 та 1E-6 протягом 12 000, 3 000 та 7 000 ітерацій відповідно. Програмна реалізація здійснена на мові програмування Python з використанням бібліотек NumPy, Pandas та Matplotlib. Набір даних для навчання та тестування модуля містив приблизно 19 000 зображень, на яких присутні 186 класів продуктів харчування. Розроблений програмний модуль розпізнавання продуктів харчування має порівняно з аналогом збільшену на 7% влучність розпізнавання сегмента та збільшену на 3% влучність розпізнавання рамки.

Ключові слова: розпізнавання, продукти харчування, згорткова нейронна мережа.

ABSTRACT

Novitska O.V. Food product recognition software module based on a convolutional neural network. The bachelor thesis consists of 68 pages of A4 format, on which there are 15 figures, 2 tables, the list of used sources contains 13 names.

The aim of the work is to improve the quality of recognition of food products through the use of a convolutional artificial neural network.

In the paper, a food recognition method based on the Mask R-CNN convolutional neural network architecture was developed. Training was carried out using minibatch stochastic gradient descent at learning rates 1E-4, 1E-5, and 1E-6 for 12,000, 3,000, and 7,000 iterations, respectively. The software was implemented in the Python programming language using the NumPy, Pandas, and Matplotlib libraries. . The data set for training and testing the module contained approximately 19,000 images with 186 food classes. The developed food recognition software module has a 7% increase in segment recognition accuracy and a 3% increase in frame recognition accuracy compared to its counterpart.

Key words: recognition, food products, convolutional neural network.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Їжа безумовно є одним з найважливіших аспектів для людського виживання, культури та задоволення. Тому не дивно, що автоматизація у сфері харчування є одним з найцікавіших застосувань машинного навчання. Зокрема, здатність виявляти їжу за допомогою комп'ютерного зору може потенційно зменшити витрати на сільське господарство та переробку харчових продуктів та революціонізувати спосіб, яким ми робимо покупки продуктів.

Рішення щодо їжі часто приймаються вручну, що потребує часу та людських зусиль: перевірка торгівельного інвентаря вручну, щоб підтвердити наявність певних продуктів (і в якій кількості), або огляд ринкового прилавка, щоб побачити, які товари доступні (і за якими цінами), і прийняття рішення на основі цієї інформації. Утримувати велику кількість продуктів харчування в короткостроковій пам'яті, одночасно вилучаючи рецепти з довгострокової пам'яті, на кращий випадок є важким і затратним, а на гірший — просто ненадійним.

Автоматизуючи етап виявлення їжі, ми можемо оптимізувати цей процес, дозволяючи користувачеві зробити фотографію свого холодильника за допомогою мобільного телефону, відразу побачити наявні інгредієнти та рецепти, які можна приготувати. Готування та покупки є універсальними видами діяльності, і не існує культури у світі, де люди не скористалися б такою послугою.

У цій роботі розробляється програма, яка дозволить користувачам отримувати інформацію та рекомендації стосовно інгредієнтів з фотографій, зроблених їх мобільним телефоном. Це включає в себе надання рекомендацій стосовно інгредієнтів для готових продуктів, а також інгрідієнтів для сировини (помідори, борошно тощо). У цій програмі обмежено застосування до зображень їжі в кошиках у продуктових магазинах.

Завдання полягає в розробці детектора продуктів харчування за допомогою передових технологій комп'ютерного зору. Вхідними даними є фотографічне зображення продуктів харчування. Буде використано модель глибокого навчання для виведення передбаченої маски сегментації зображення, що описує наявні в зображенні продукти харчування.

Метою дослідження є підвищення якості розпізнавання продуктів харчування за рахунок використання згорткової штучної нейронної мережі. На відміну від аналогів, які використовують традиційні методи машинного навчання без використання нейромереж, в роботі пропонується застосувати згорткову штучну нейронну мережу для навчання розпізнаванню продуктів харчування на основі прикладів.

Об’єктом дослідження є процес комп’ютеризованого розпізнавання продуктів харчування на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розпізнавання продуктів харчування на основі згорткової нейронної мережі та якість їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи розпізнавання продуктів харчування та обрати напрямок досліджень;
2. Розробити структуру програмного модуля розпізнавання продуктів харчування на основі нейронної мережі;
3. Обгрунтувати вибір архітектури нейромережі;
4. Розробити алгоритм роботи програмного модуля розпізнавання продуктів харчування;
5. Здійснити програмну реалізацію модуля розпізнавання продуктів харчування;
6. Провести тестування програмного модуля розпізнавання продуктів харчування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ

1.1 Постановка задачі

У цій роботі розробляється програма, яка дозволить користувачам розпізнавати продукти харчування з тим, щоб в подальших розробках отримувати рецепти та рекомендації стосовно продуктів з фотографій, зроблених їх мобільним телефоном. Це включає в себе надання рекомендацій стосовно інгредієнтів для готових продуктів, а також інгрідієнтів для сировини (помідори, борошно тощо).

Завдання полягає в розробці детектора продуктів харчування за допомогою передових технологій комп'ютерного зору. Вхідними даними є фотографічне зображення продуктів харчування. Буде використано модель глибокого навчання для виведення передбаченої маски сегментації зображення, що описує наявні в зображенні продукти харчування.

На вході програми – зображення у форматі jpeg, яке генерується камерою, а на виході – додавання до зображення рамок, що обмежують продукт і виведення назви продукту харчування.

Проект розбито на два основних етапи. Етап 1 полягає в навчанні нейромережі для подальшого розпізнавання елементів. Етап 2 полягає розробці детектора продуктів харчування за допомогою передових технік комп'ютерного зору. Вхідними даними для етапу 1 є фотографічне зображення, на якому є продукти харчування та інші предмети. Ми використовуємо модель глибокого навчання для виведення передбачуваної маски сегментації зображення, що описує наявні в зображенні продукти харчування. Це легко перетворюється в список продуктів харчування, які можуть бути передані як вхідні дані для етапу 2.

1.2 Огляд відомих методів розпізнавання продуктів харчування

Завдання аналізу зображень та розпізнавання на них конкретних об’єктів в наш час є одними з основних завдань систем машинного зору. Велика зацікавленість в галузі обробки зображень, штучного інтелекту, інформаційно-аналітичних систем викликає необхідність розробки методів автоматизованої обробки візуальної інформації. До таких методів обробки зображень можна віднести методи автоматичного пошуку, виявлення та локалізації специфічних ділянок зображення, які являють інтерес для певної предметної області [1].

За останній період було запропоновано множину різних алгоритмів обробки, локалізації та розпізнавання візуальних об'єктів: нейронні мережі, ланцюги Маркова, дерева рішень і т. п. При обробці світлин на якість розпізнавання впливають умови освітлення, розташування об'єкта на зображенні, чіткість, роздільна здатність та шуми.

Розпізнавання образів - це процес віднесення вхідних даних до відповідного класу через виділення суттєвих ознак, які характеризують ці дані, із загальної маси неістотних даних [1].

Розглянемо класичну постановку задачі розпізнавання образів.

Дано множину об'єктів. Відносно цих об'єктів потрібно виконати класифікацію. Множину об'єктів представляють підмножинами, які носять назву класів. Задано: інформація про класи, опис усієї множини об'єктів і опис інформації про об'єкт, належність якого до певного класу невідома. Потрібно за наявною інформацією про класи і описи об'єкта встановити, до якого класу він належить.

Найчастіше у задачах розпізнавання образів розглядають чорно-білі зображення, що надає можливість розглядати зображення як функцію на площині. Якщо розглянути множину точок T на площині, де функція $f(x,y)$ відображає в кожній точці зображення якусь його характеристику - яскравість,

оптичну щільність, прозорість, то така функція є формальним описом зображення.

Множина всіх можливих функцій *f(x,y)* на площині *T* - це модель множини усіх зображень X. Вводячи поняття подібності між образами, можна поставити задачу розпізнавання. Конкретний вид такої постановки задачі сильно залежить від таких етапів при розпізнаванні відповідно до того або іншого підходу.

Методи, покладені в основу логічного, математичного і програмного забезпечення систем розпізнавання образів, класифікуються на:

1. Детерміновані методи – для побудови алгоритмів розпізнавання застосовують “геометричні” міри близькості, засновані на вимірюванні відстаней між числовими ознаками об’єкта та числовими ознаками еталонів класів.

2. Ймовірнісні методи – для будування алгоритмів розпізнавання використовують ймовірнісні методи теорії статистичних рішень. Для цього треба мати ймовірнісні залежності між ознаками об’єктів і класами.

3. Логічні методи - засновані на дискретному аналізі й обчисленні висловлювань. Потрібно вирішити систему булевих рівнянь з використанням логічних ознак об’єктів і знайти невідомі величини – класи, до яких ці об’єкти відносяться.

4. Структурні (лінгвістичні) методи – для побудови алгоритмів розпізнавання використовується речення, кожне з яких описує структуру (будову) об’єкта з непохідних (“атомарних”) елементів. Ці речення складають спеціальну мову. Класифікація об’єкта виконується шляхом порівняння речення невідомого об’єкта з еталонними реченнями класів.

5. Нейромережеві методи – використовують моделі нейронних мереж для розпізнавання образів.

6. Методи потенціалів – ознака об’єкта розглядається як його електричний потенціал, який зменшується зі зростанням відстані від об’єкта. За рахунок цього області класів об’єктів мають велике цифрове значення

потенціалу і можуть розглядатися як “гори, які відокремлюються одна від одної “ярами”.

4.7. Експертні методи розпізнавання образів. Експертні системи застосовують там, де існують евристичні або інтуїтивні методи рішень і немає точно визначених алгоритмів або розрахунків. Експертна система – це набір комп’ютерних програм, який містить накопичені знання кількох експертів у даній предметній області і здатний в рамках цієї області класифікувати об’єкти, давати відповіді, поради, рекомендації, запитуючи при необхідності додаткову інформацію. Найчастіше використовуються правила, які мають форму: “ЯКЩО… ТОДІ… ІНАКШЕ…”. Твердження, зазвичай, мають ймовірнісну оцінку. Верхній рівень цих систем є логічним, але висновки в ньому робляться не методом порівняння (порівняння отриманої апостеріорної інформації з апріорною), а методом індукції, дедукції, аналогії, тобто методами, що властиві лише людині. При цьому отримані логічні висновки мають породжувати нові речення, нові знання, які поповнюють базу даних та знань. Класи об’єктів у таких експертних системах розпізнавання образів виглядають як нечіткі множини за рахунок нюансів, напівтонів (наприклад: об’єкт майже круглої форми, блідо-рожевого кольору, приблизно рівний по об’єму м`ячу для гольфу і т.п.). Належність об’єктів або явищ до класів грунтується на рівняннях, які застостовують функції еквівалентності з використанням імовірнісних характеристик. Машина виведення на основі отриманої інформації генерує гіпотезу про клас невідомого об’єкта, яка не суперечить отриманій про нього інформації.

Задача розпізнавання об'єкта є складною через такі основні причини:

− різні умови освітленості, які визначаються типом, кількістю і напрямком джерел світла;

− різна роздільна здатність зображення, на якому знаходиться об’єкт, а також розміри самого об’єкта відносно зображення;

– зашумленість зображення, яка обумовлюється його якістю, в першу чергу, а також можливі візуальні ефекти на зображенні, що зменшують видимість об’єкта;

– часткове перекриття об'єкта іншими об'єктами сцени; необхідність локалізації і розпізнавання об’єкта, який може мати будь-яке положення у просторі.

Однак, існуючі системи локалізації і розпізнавання об’єктів не завжди враховують перераховані особливості, що не дозволяє досягти прийнятного значення якості розпізнавання зображень [1].

Із розглянутих вище 7 груп методів розпізнавання образів найбільш перспективною є група нейромережевих методів. Це пояснюється тим, що штучні нейромережі працюють на основі навчання на прикладах, а тому не потрібно видумувати якісь складні алгоритми обробки та виділення корисної інформації із зображень, формулювати якісь правила або критерії прийняття рішень. Все це виконує сама нейромережа, яка була навчена на розпізнавання вдало підібраних прикладів тих чи інших зображень.

Відомі методи розпізнавання продуктів харчування в основному підсилені потужними моделями виявлення об'єктів та семантичної сегментації, багато з яких є відкритими для використання. Фреймворки, такі як Tensorflow та PyTorch, мають реалізації згорткових нейромережевих моделей, таких як RCNN, SSD, RPN, RetinaNet і т. д. Ці моделі з відкритим кодом можуть бути подальшим чином точніше навчені на більш специфічних даних для вирішення більш конкретних проблем машинного навчання. Для даної роботи було вирішено використовувати фреймворк Detectron2 [2] від Meta. Detectron2 реалізований на PyTorch і включає в себе високоякісні реалізації передових алгоритмів виявлення об'єктів та сегментації.

Ще одним важливим компонентом для виявлення продуктів харчування є наявність масивних наборів даних з виявлення об'єктів/семантичної сегментації. MS COCO [3] містить близько 80 міток об'єктів (як обмежувальні рамки, так і маски семантичної сегментації) та 200 тисяч позначених

зображень, що в результаті призводить до 1,5 мільйона позначених об'єктів на 200 тисячах зображень. LVIS [4] ґрунтується на тій же базі зображень з додатковими поліпшеннями анотацій (близько 1200 міток об'єктів). Оглядова стаття [5] надає набори даних з виявлення об'єктів/семантичної сегментації, які налаштовані на продукти харчування та роботу у роздрібних магазинах. MVTec D2S [6] містить 60 міток об'єктів (маски семантичної сегментації) та 21 тис. позначених зображень, на кожному з яких зображені об'єкти лише одного класу на однорідному фоні (нагадує автоматизовану касу). Набір даних Retail Product Checkout (RPC) [7] містить 200 категорій товарів та 83 739 зображень, включаючи зображення як одного, так і кількох продуктів одночасно.

Отже, метою цього проекту є інтеграція існуючих моделей виявлення об'єктів/семантичної сегментації з існуючими базами даних продуктів харчування.

1.3 Обгрунтування вибору способу виявлення переваг розробленого модуля

Опишемо деталі цього проекту: спочатку було зібрано загальний набір даних з продуктів харчування з наборів даних LVIS [4], щільно сегментованого набору даних супермаркетів [6] та RPC: великого набору даних для перевірки роздрібних товарів [7]. Потім було ітеративно підігнано моделі, надані Detectron 2 [2] від Meta, для виявлення та класифікації лише продуктів харчування.

Detectron2 — це платформа нового покоління для виявлення та сегментації об’єктів. Detectron2 був створений Facebook AI Research (FAIR) для підтримки швидкого впровадження та оцінки нових досліджень комп’ютерного зору. Він містить при реалізації таких алгоритмів виявлення об’єктів:

- Mask R-CNN,
- RetinaNet,
- Fast R-CNN,
- RPN,
- TensorMask,
- PointRend,
- DensePose,

та інші...

Було здійснено заміну базової моделі в Detectron2 нашою власною навченою моделлю. Тому як аналог для порівняння з розробленою моделлю було взято базову модель в Detectron2, але вона навчена розпізнавати різноманітні об`єкти, а не тільки продукти харчування. Тому ми додатково підготували її, виключивши з неї всі класи об`єктів, які не є продуктами харчування.

Нашою метою є більш точне розпізнавання продуктів харчування, ніж попередньо навчені моделі із зоопарку моделей Detectron. Для оцінки нашого детектора/класифікатора продуктів харчування на різних зображеннях, будемо використовувати середню влучність масок, як описано в наборі даних COCO [3]. Ми обираємо діапазон порогів перетину/об'єднання (IoU) (від 0,5 до 0,95 з кроком 0,05) та категорії: для кожної пари (поріг, категорія) ми розраховуватимемо середню влучність (проводячи обчислення IoU на масках сегментації, позначаючи виявлення на основі порогу IoU). Нашою метрикою продуктивності є середнє значення влучності по всіх (поріг, категорія) парах.

Головним показником якості програм розпізнавання продуктів харчування є така метрика як влучність розпізнавання, тому порівнювати різні методи розпізнавання продуктів харчування будемо у розділі 4 саме за параметром влучності розпізнавання.

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі розпізнавання продуктів харчування. Також розглядаються різні методи розв'язання задачі розпізнавання об`єктів і оцінюється їх застосовність до завдання розпізнавання продуктів харчування. Серед таких методів розпізнавання об`єктів як детерміновані методи, ймовірнісні методи, логічні методи, структурні (лінгвістичні) методи, методи потенціалів, експертні методи та нейромережеві методи як найбільш перспективний і застосовний до даної задачі було обрано нейромережевий метод. Було показано, що з різних типів нейронних мереж найбільше підходить для вирішення поставленої задачі саме згорткові нейронні мережі. Крім цього, було обгрунтовано вибір способу виявлення переваг розробленого модуля шляхом паралельного моделювання двох різних методів розпізнавання продуктів харчування.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Розробка методу розпізнавання продуктів харчування на основі згорткової нейронної мережі

2.1.1 Бібліотека Detectron2. Detectron2 є популярною бібліотекою модульних моделей комп'ютерного зору на основі PyTorch. Вона дозволяє використовувати передові моделі комп'ютерного зору для навчання наших власних наборів даних з метою виявлення об'єктів та сегментації. Вона базується на бенчмарку Mask R-CNN. Підтримується кілька завдань, таких як виявлення обмежувальних рамок, сегментація екземплярів, виявлення ключових точок, виявлення щільних сутностей і т. д. Багато розробленого коду базується на навчальному посібнику Detectron2.

2.1.2 Архітектура моделі. Сегментація екземплярів передбачає правильне виявлення всіх об'єктів на зображенні разом з точною сегментацією кожного екземпляра об'єкта. Це включає як виявлення об'єктів, так і семантичну сегментацію. У виявленні об'єктів основна мета полягає в класифікації окремих об'єктів і локалізації їх за допомогою обмежувальних рамок, тоді як у семантичній сегментації мета полягає в класифікації кожного пікселя у фіксований набір категорій без розрізнення екземплярів об'єктів.

Ми використовували фреймворк Mask R-CNN для сегментації екземплярів. Mask R-CNN можна розглядати як розширення Faster R-CNN. Faster R-CNN складається з двох етапів. Перший етап пропонує обмежувальні рамки для кандидатів на об'єкт, який називається мережею пропозицій регіонів (RPN). Другий етап виділяє ознаки за допомогою ROIPool з кожного кандидата та виконує класифікацію та регресію обмежувальної рамки. Mask R-CNN розширює Faster R-CNN, додаючи гілку для передбачення масок

сегментації на кожній ділянці інтересу паралельно з існуючою гілкою для класифікації та регресії обмежувальної рамки. Додатковий вихід маски відрізняється від вихідних класів та рамок і вимагає виділення набагато більш дрібного просторового розташування об'єкта. На рисунку 2.1 пояснено загальну ідею Mask R-CNN.

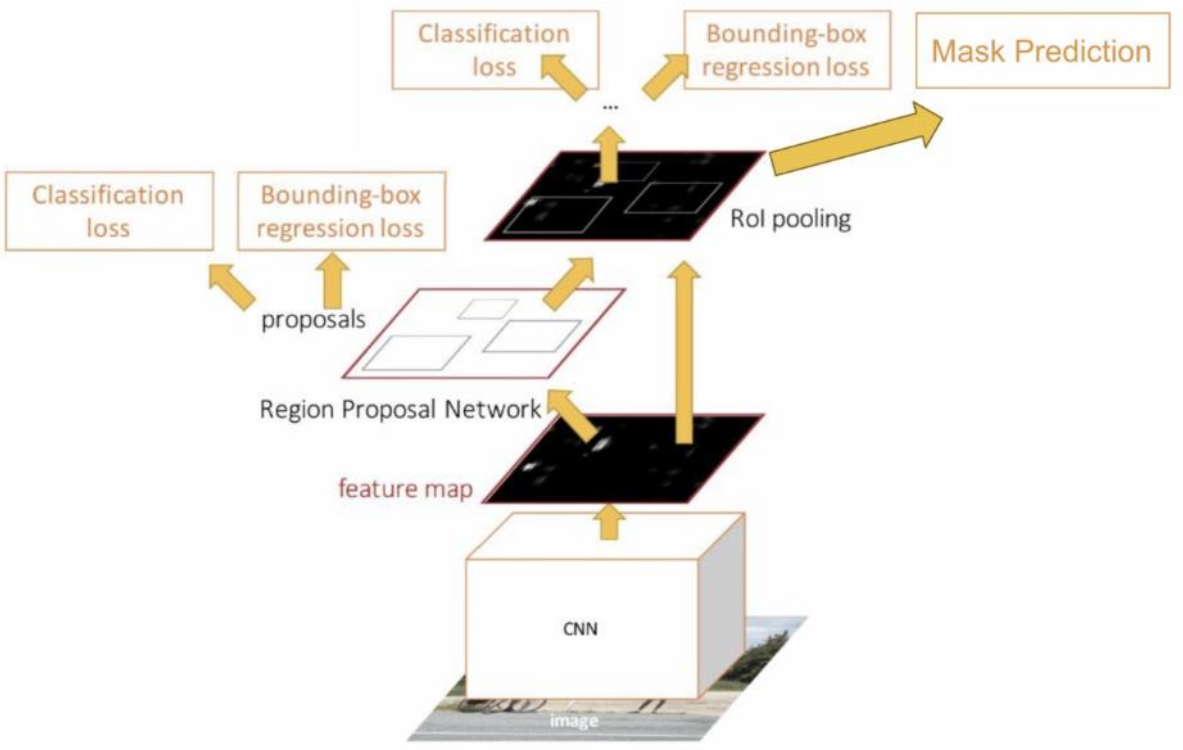

Рисунок 2.1 – Архітектура Mask R-CNN

Під час навчання ми можемо визначити багатозадачну втрату на кожному вибраному регіоні інтересу (RoI) як $L = L_{cls} + L_{box} + L_{mask}$. Тут L_{cls} представляє класифікаційні втрати, L_{box} представляє втрати при визначенні обмежувальної рамки, а L_{mask} представляє втрати при передбаченні маски.

Для сегментації екземплярів за допомогою Mask R-CNN піксель-до-пікселю необхідно, щоб ознаки об'єктів з областей інтересу були добре вирівняні, щоб зберегти просторову відповідність на рівні пікселів. Тому існує шар RoIAlign, який відіграє ключову роль у передбаченні маски. Шар RoIAlign усуває сувору квантизацію RoIPool та правильно вирівнює виділені ознаки з

вхідними даними. На рисунку 2.2 показана загальна структура для сегментації екземплярів за допомогою Mask R-CNN, включаючи RoIAlign.

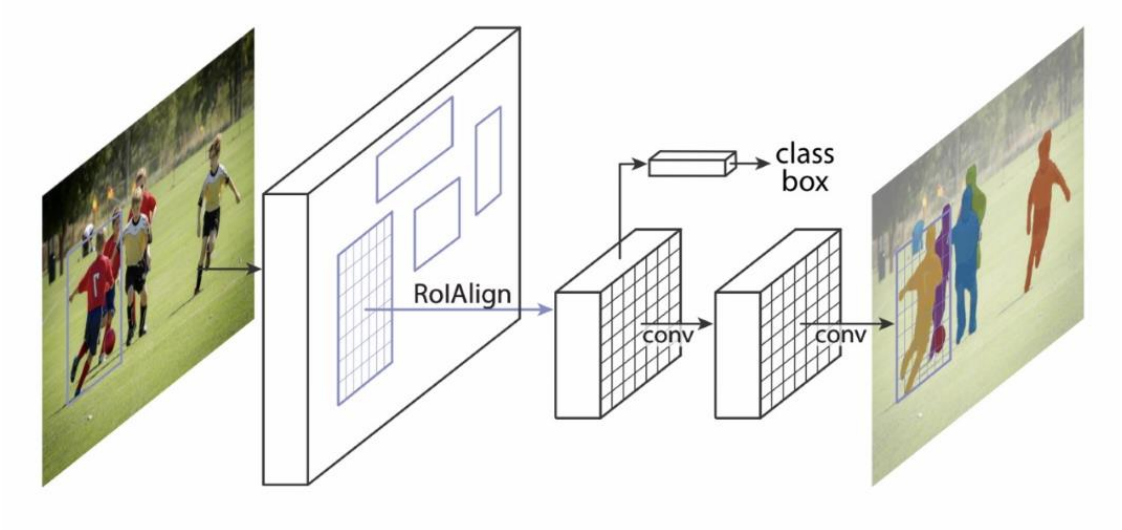

Рисунок 2.2 – Структура мережі Mask R-CNN

Архітектура мережі Mask R-CNN може бути розділена на дві частини: (1) архітектура згорткової основи, що використовується для вилучення ознак з усього зображення, та (2) головна мережа для класифікації, регресії та передбачення масок, яка застосовується окремо до кожної області інтересу. В роботі було використано модель mask_rcnn_R_50_FPN_3x із Detectron2 для навчання нашого набору даних. Ця модель має основу ResNet-50-FPN.

ResNet є однією з найпопулярніших архітектур згорткових нейронних мереж. Вона складається з дуже глибоких мереж, які використовують зв'язки зі зміною залишкових блоків. У ResNet є стеки залишкових блоків, кожен з яких складається з двох згорткових шарів розміром 3 X 3. На рисунку 2.3 показано залишковий блок ResNet.

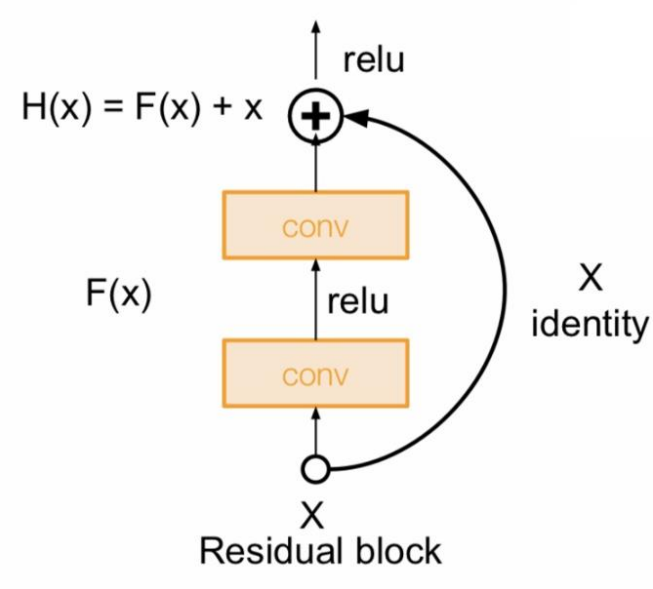

Рисунок 2.3 - Залишковий блок у ResNet

Системи пірамідальних ознак (FPN) [8] використовують пірамідальну ієрархію ознак ConvNet, яка має семантику від низького до високого рівня, і будує піраміду ознак з високорівневою семантикою по всьому. Вона використовує архітектуру зверху-вниз з боковими (латеральними) зв'язками для побудови внутрішньої піраміди ознак з одномасштабного введення. Основи ResNet-FPN для вилучення ознак з Mask R-CNN відомі своїми відмінними показниками якості та швидкості.

2.1.3 Базова модель: MS COCO. Detectron2 складається з кількох попередньо навчених моделей для сегментації екземплярів. Планується розпочати з моделей, навчених на наборі даних MS COCO, і порівняти їх. Є така думка, що моделі на основі MS COCO будуть хорошою відправною точкою для сегментації екземплярів. Для кожної моделі можна налаштовувати гіперпараметри, такі як швидкість навчання, момент оптимізатора, зменшення ваги, розмір пакету, кількість ітерацій та інше. Ці гіперпараметри можна змінити, використовуючи конфігураційний екземпляр, який ми створюємо для навчання даних.

2.2 Архітектура згорткової нейронної мережі Mask R-CNN

Mask RCNN — глибока нейронна мережа, призначена для вирішення проблеми сегментації екземплярів у машинному навчанні або комп'ютерному зорі. Іншими словами, вона може розділяти різні об'єкти на зображенні чи відео. На вхід мережі надається зображення, а на виході мережа видає обмежувальні рамки об'єкта, класи та маски.

Існує два етапи роботи Mask RCNN. По-перше, вона генерує пропозиції щодо регіонів, де може бути об'єкт на полі вхідного зображення. По-друге, вона передбачає клас об'єкта, уточнює обмежувальну рамку та генерує маску на рівні пікселів об'єкта на основі пропозиції першого етапу. Обидва етапи пов'язані з хребтовою структурою (хребтом).

Хребет — це глибока нейронна мережа в стилі FPN (Feature Pyramid Network). Вона складається з шляху «знизу-до-гори», шляху «зверху-до-низу» і бічних з'єднань (див. рис. 2.4). Шляхом « знизу-до-гори » може бути будь-який ConvNet, зазвичай ResNet або VGG, який витягує ознаки з необроблених зображень. Шлях «зверху-до-низу» генерує пірамідну карту ознак, подібну за розміром до шляху « знизу-до-гори ». Бічні зв'язки — це операції згортання та додавання між двома відповідними рівнями двох шляхів. FPN перевершує інші одиночні ConvNets головним чином через те, що він підтримує сильні семантичні характеристики в різних масштабах роздільної здатності.

Тепер розглянемо перший етап. Легка нейронна мережа, яка називається RPN (Region Proposal Network), сканує весь шлях «зверху-до-низу» FPN (далі – карта ознак) і пропонує області, які можуть містити об'єкти. Хоча сканування карти об'єктів є ефективним способом, нам потрібен метод прив'язування об'єктів до розташування необробленого зображення. У цьому допомагають прив'язки. Прив'язки — це набір прямокутників із заздалегідь визначеними розташуваннями та масштабами відносно зображень. Основні класи істинності (тільки двійковий файл об'єкта або фону, класифікований на цьому етапі) і обмежувальні прямокутники призначаються окремим

прив’язкам відповідно до певного значення IoU (перетин над об`єднанням). Оскільки прив’язки з різними масштабами прив’язуються до різних рівнів карти об’єктів, RPN використовує ці прив’язки, щоб визначити, де на карті об’єктів «повинен» отриматись об’єкт і який розмір його обмежувальної рамки. Тут ми можемо погодитися, що згортання, зменшення та підвищення дискретизації збережуть об’єкти в тих самих відносних розташуваннях, що й об’єкти на вхідному зображенні, і не зіпсують їх.

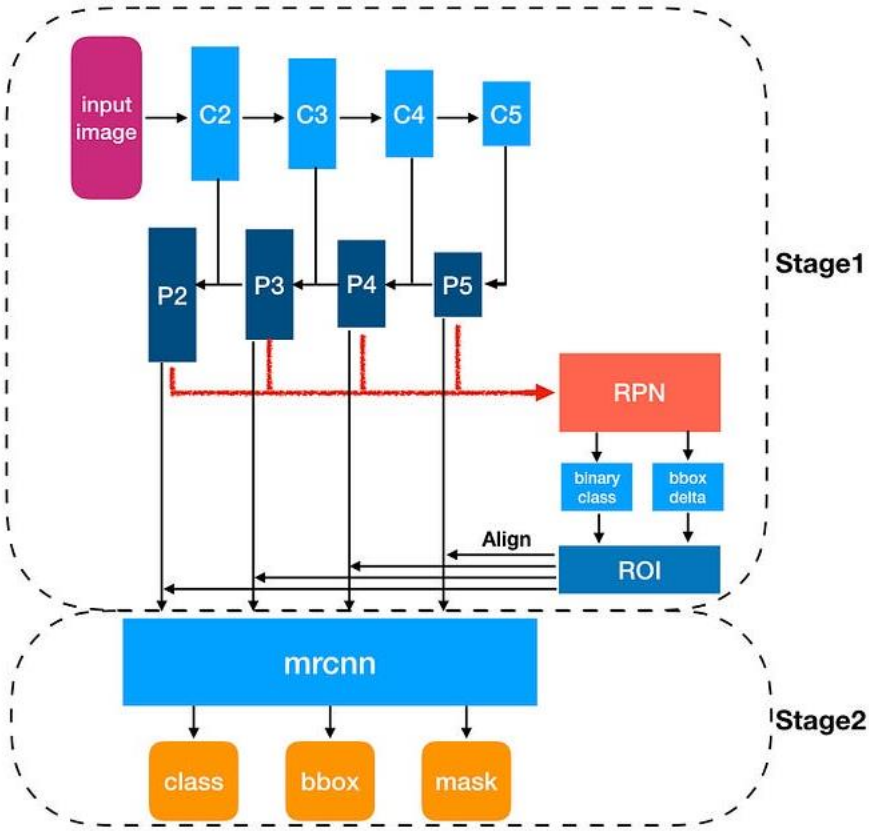

Рисунок 2.4 – Логічна структура нейронної мережі Mask RCNN

На другому етапі інша нейронна мережа бере запропоновані на першому етапі регіони та призначає їх кільком конкретним областям рівня карти ознак, сканує ці області та генерує класи об’єктів (багатокатегорійну класифікацію), обмежувальні рамки та маски. Процедура схожа на RPN. Відмінності полягають у тому, що без допомоги прив’язок другий етап використовував

трюк під назвою ROIAlign, щоб знайти відповідні області карти об’єктів, і існує гілка, яка створює маски для кожного об’єкта на рівні пікселів.

Найцікавіше у Mask RCNN, це те, що можна було б змусити різні шари нейронної мережі вивчати функції з різними масштабами, так само як прив’язки та ROIAlign, замість того, щоб розглядати шари як чорний ящик.

Сітка об’єднання для області інтересу (ROI);

Використання мережі Feature Pyramid Network (FPN), яка розширює можливості Mask R-CNN, надаючи багатомасштабне представлення ознак, дозволяючи ефективне повторне використання функцій і вирішуючи коливання масштабу в об’єктах.

Що відрізняє Mask R-CNN, так це його здатність правильно сегментувати та ідентифікувати піксельні межі кожного елемента всередині зображення. Ця можливість дрібної сегментації досягається шляхом додавання додаткової гілки «голова маски» до моделі Faster R-CNN (рис. 2.5).

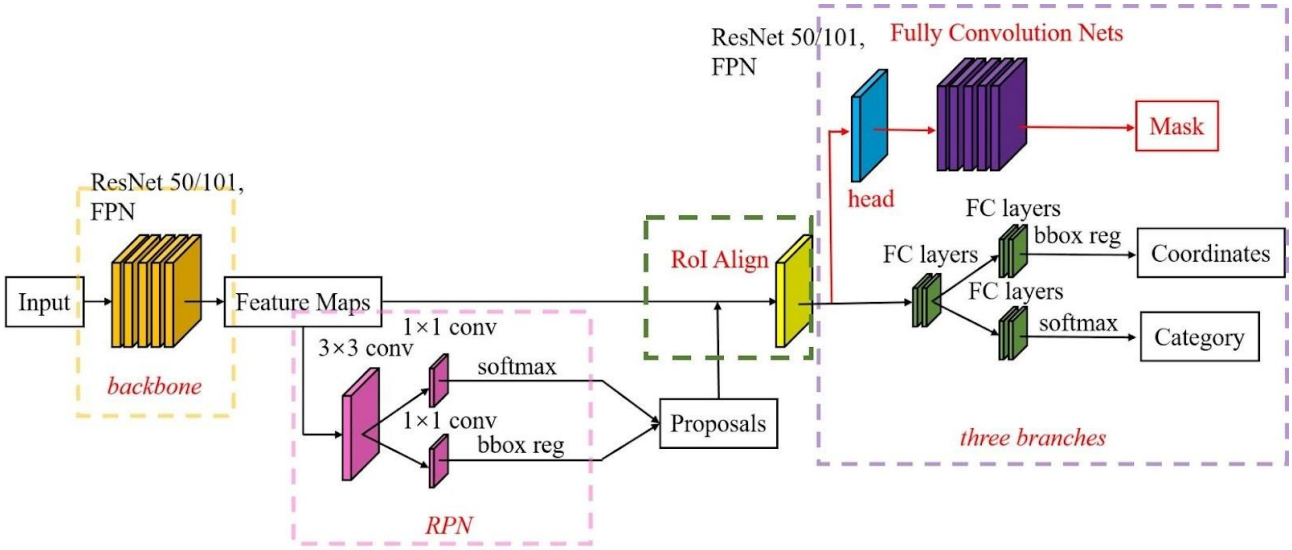

Рисунок 2.5 – Пошарова структура нейронної мережі Mask RCNN

Основною новинкою Mask R-CNN є його здатність робити попіксельну сегментацію екземплярів із розпізнаванням об’єктів. Це досягається шляхом включення додаткової гілки «mask head», яка створює точні маски сегментації

для кожного розпізнаного елемента. Увімкнено чіткі межі на рівні пікселів, що забезпечує точну та ретельну сегментацію екземплярів.

ROIAlign і Feature Pyramid Network (FPN) — дві важливі інновації, вбудовані в Mask R-CNN. ROIAlign долає обмеження класичного підходу до об'єднання ROI, включаючи білінійну інтерполяцію в процес об'єднання. Це зменшує труднощі зі зміщенням і забезпечує правильне отримання просторової інформації з вхідної карти об'єктів, що забезпечує кращу точність сегментації, особливо для крихітних об'єктів.

Створюючи багатомасштабну піраміду функцій, FPN відіграє вирішальну роль у вилученні функцій. Ця піраміда об'єднує інформацію з багатьох масштабів, дозволяючи моделі отримати більш повне розуміння контексту об'єкта та забезпечуючи покращене розпізнавання об'єктів і сегментацію в широкому діапазоні розмірів об'єктів.

Конструкція Mask R-CNN базується на архітектурі Faster R-CNN із додаванням додаткової гілки «mask head», що забезпечує попіксельну сегментацію. Загальна архітектура складається з багатьох основних компонентів (рис. 2.5).

Магістральна мережа (хребет).

Основна мережа Mask R-CNN часто є попередньо навченою згортковою нейронною мережею, такою як ResNet або ResNeXt. Ця магістраль відповідає за обробку вхідного зображення та вилучення інформації високого рівня. Потім на вершині цієї магістральної мережі будується FPN, щоб сформувати піраміду функцій.

FPN призначені для вирішення проблеми роботи з елементами різних розмірів і масштабів на зображенні. Шляхом об'єднання функцій з кількох рівнів магістральної мережі проект FPN утворює багатомасштабну піраміду функцій. Ця піраміда має функції з різною просторовою роздільною здатністю, починаючи від функцій високої роздільної здатності з великою семантичною інформацією до функцій низької роздільної здатності з дрібнішими просторовими деталями.

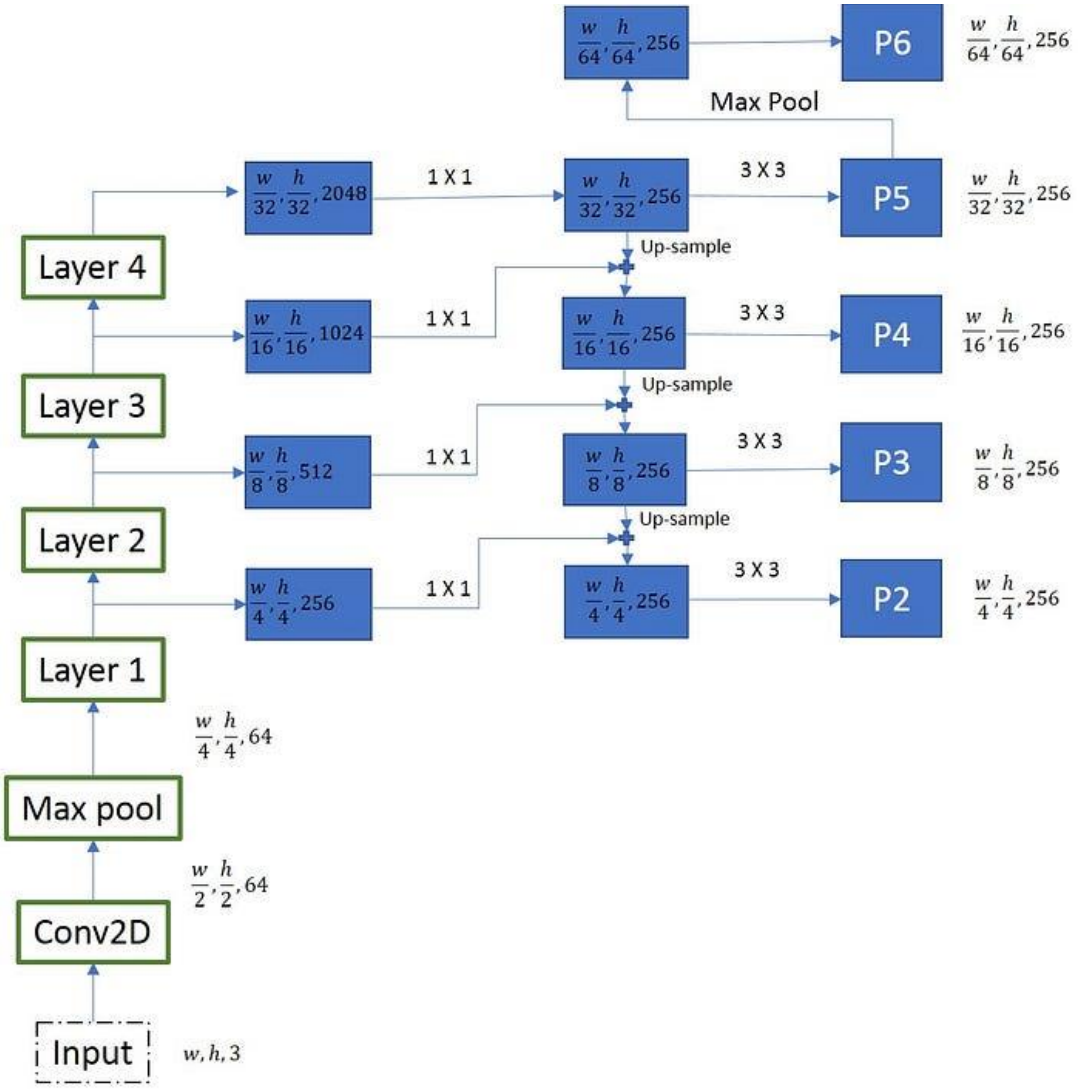

Рисунок 2.6 – Пірамідна мережа ознак (FPN, хребет)

У Mask R-CNN робота FPN складається з таких кроків:

1. Вилучення характеристик високого рівня: магістральна мережа отримує характеристики високого рівня з вхідного зображення.

2. Об’єднання функцій: щоб побудувати шлях зверху вниз, FPN з’єднує кілька рівнів магістральної мережі. Цей низхідний підхід інтегрує семантичну інформацію високого рівня з картами функцій нижчого рівня, що дозволяє моделі повторно використовувати функції різних розмірів.

3. Піраміда функцій: процес злиття формує багатомасштабну піраміду функцій, де кожен рівень піраміди представляє різну роздільну

здатність ознак. Об’єкти з найвищою роздільною здатністю знаходяться у верхній частині піраміди, а об’єкти з найнижчою – у нижній частині.

Mask R-CNN може успішно обробляти об'єкти різного розміру завдяки піраміді функцій, утвореній FPN. Завдяки багатомасштабному представленню модель може збирати контекстну інформацію та надійно розпізнавати елементи в кількох масштабах зображення.

Мережа регіональних пропозицій (RPN) – рис. 2.7.

RPN відповідає за створення регіональних пропозицій або потенційних обмежувальних рамок, які можуть містити об’єкти на зображенні. Він працює на карті функцій магістральної мережі та пропонує потенційні цікаві місця.

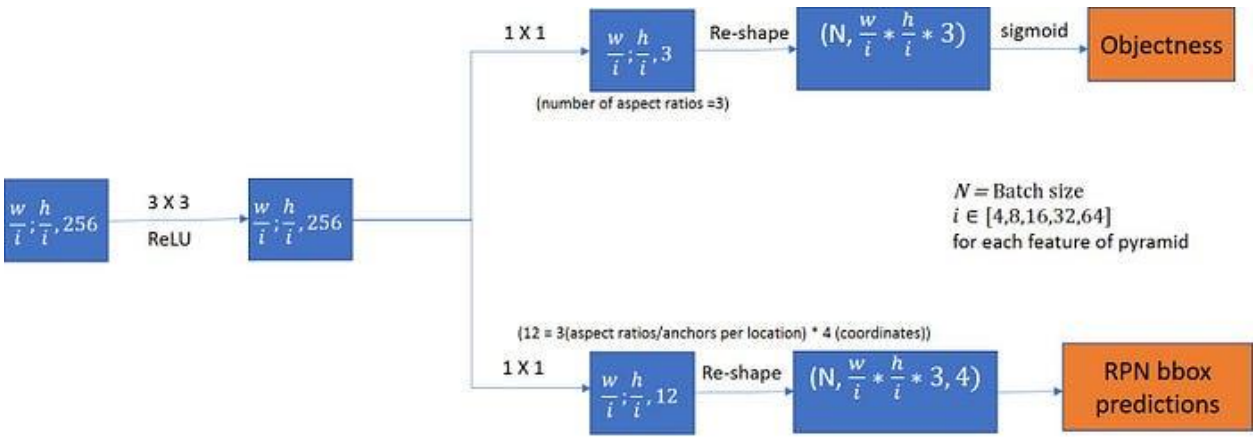

Рисунок 2.7 – Приклад RPN мережі (Region Proposal Network)

ROIAlign.

Рівень ROIAlign (Region of Interest Align) вставляється після того, як RPN створить пропозиції регіону. Цей крок допомагає подолати проблему неузгодженості в об’єднанні ROI.

ROIAlign має вирішальне значення для отримання ознак із вхідної карти ознак точно для кожної пропозиції області, гарантуючи ідеальну сегментацію по пікселях у завданнях сегментації екземплярів.

Основною метою ROIAlign є вирівнювання об’єктів у межах цікавої області (ROI) з просторовою сіткою вихідної карти об’єктів. Це вирівнювання має вирішальне значення для уникнення втрати інформації, яка може статися,

коли просторові координати досліджуваної області квантуються до найближчого цілого числа (як у об’єднанні досліджуваної області).

Процес ROIAlign включає такі кроки:

1. Карта вхідних ознак: Карта вхідних ознак, яка зазвичай збирається з магістральної мережі, є першим кроком у процесі. Ця карта ознак пропонує повну семантичну інформацію про все зображення.

2. Регіональні пропозиції: RPN створює регіональні пропозиції (кандидати, обмежувальні рамки), які можуть містити цікаві елементи всередині зображення.

3. Поділ на сітки: кожна пропозиція області розбивається на певну кількість просторових ящиків або сіток однакового розміру. Ці сітки використовуються для вилучення об’єктів, пов’язаних із регіоном інтересу, із вхідної карти об’єктів.

4. Білінійна інтерполяція: на відміну від об’єднання ROI, яке квантує просторові координати сіток до найближчого цілого числа, ROIAlign обчислює внески об’єднання для кожної сітки за допомогою білінійної інтерполяції. Ця інтерполяція гарантує точніше вирівнювання функцій у ROI.

5. Вихідні об’єкти: Об’єкти з вхідної карти об’єктів, вирівняні з кожною сіткою на вихідній карті об’єктів, служать репрезентативними об’єктами для кожної пропозиції території. Дрібнозерниста просторова інформація фіксується цими вирівняними функціями, що є критично важливим для ефективної сегментації.

ROIAlign значно підвищує точність виділення ознак для кожної запропонованої області, застосовуючи білінійну інтерполяцію під час фази об’єднання, зменшуючи проблеми з розбіжністю.

Завдяки цьому ідеальному вирівнюванню Mask R-CNN може створювати більш точні маски сегментації, що особливо корисно для крихітних об’єктів або областей, де необхідно зберегти дрібні особливості. Як наслідок, ROIAlign додає виняткову продуктивність Mask R-CNN у завданнях сегментації екземплярів.

Голова маски.

Mask Head — це нова гілка в Mask R-CNN, яка відповідає за створення масок сегментації для кожної пропозиції області. Вирівняні функції, отримані ROIAlign, використовуються керівником для прогнозування бінарної маски для кожного об’єкта, окреслюючи піксельні межі екземплярів. Як правило, голова маски складається з кількох згорткових шарів, за якими йдуть шари підвищення дискретизації (деконволюція або транспоновані згортки) – рис. 2.8.

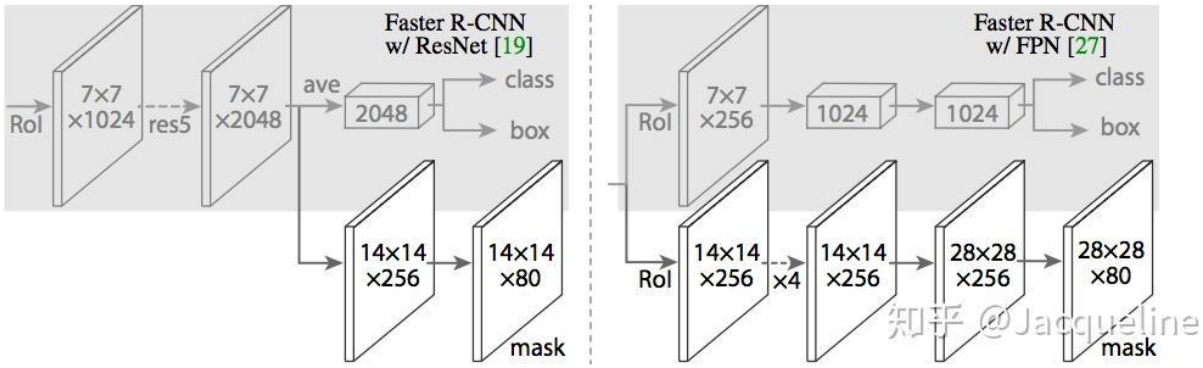

Рисунок 2.8 – Структура голови маски (Mask Head)

Модель спільно оптимізується під час навчання з використанням суміші втрат класифікації, втрат регресії обмежувальної рамки та втрат сегментації маски. Це дає змогу моделі навчитися розпізнавати об’єкти, а також уточнювати їх обмежувальні рамки та створювати точні маски сегментації.

Mask R-CNN чудово працює в різних областях, що робить її ефективною моделлю для широкого спектру програм комп’ютерного зору, таких як розпізнавання об’єктів, сегментація екземплярів, сегментація кількох об’єктів і складна обробка сцен.

Однак у Mask R-CNN є кілька недоліків, які слід враховувати:

1. Сегментація дрібних об’єктів: через недостатню інформацію про пікселі, Mask R-CNN може не відокремлювати дуже дрібні об’єкти.

2. Обчислювальна складність: Навчання та логічні висновки можуть вимагати обчислень, потребуючи значних ресурсів, особливо для зображень із високою роздільною здатністю або великих наборів даних.

3. Вимоги до даних: Навчання Mask R-CNN потребує величезної кількості анотованих даних, отримання яких може зайняти багато часу та дорого коштувати.

4. Обмежене узагальнення: здатність моделі до узагальнення категорій елементів, які зустрічалися раніше, обмежена, особливо коли дані розріджені.

Mask R-CNN поєднує ідентифікацію об’єктів і сегментацію екземплярів, дозволяючи розпізнавати об’єкти, а також точно окреслювати їхні межі на рівні пікселів. Mask R-CNN забезпечує високу продуктивність і точність завдяки поєднанню функції пірамідної мережі (FPN) із вирівнюванням регіону інтересу (ROIAlign).

Mask R-CNN має певні недоліки, такі як обчислювальна складність і споживання пам'яті під час навчання та логічного висновку. Можуть виникати труднощі з правильним сегментуванням дуже маленьких об’єктів або справлятися з ситуаціями з серйозними перешкодами. Отримання великої кількості анотованих навчальних даних також може бути складним, а для точного налаштування моделі для певних областей може знадобитися суворе налаштування параметрів.

2.3 Навчання згорткової нейронної мережі Mask R-CNN

В роботі було модифіковано вищезгадану модель **mask_rcnn_R_50_FPN_3x**, попередньо навчену на MS-COCO, для виявлення 186 класів продуктів харчування з нашого набору даних. Навчання проводиться за допомогою стохастичного градієнтного спуску по мінібатчам на швидкостях навчання 1E-4, 1E-5 та 1E-6 протягом 12 000, 3 000 та 7 000 ітерацій відповідно, що призводить до насичення графіка втрат, показаного на

рисунку 2.9. Використовується розмір пакета 8, який представляє верхню межу пам'яті GPU. Оскільки мережа швидко збігається до 2 епох, для навчання на 3 різних наборах даних застосовується частково-лінійний підхід.

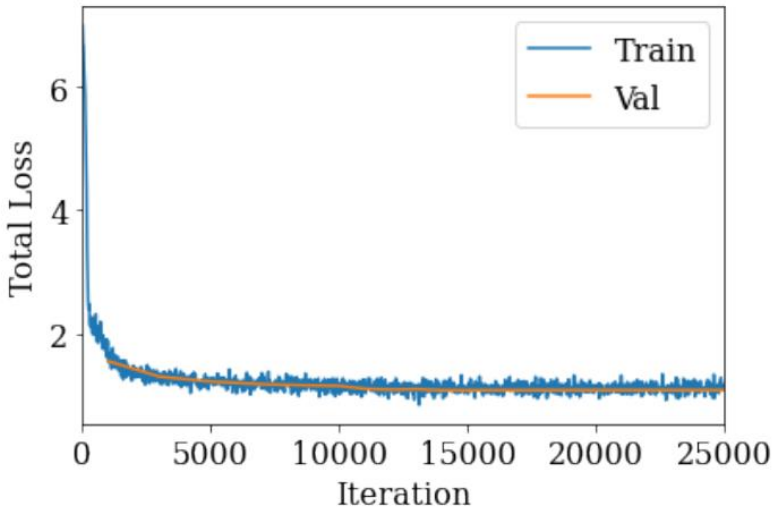

Рисунок 2.9- Крива навчання та перевірки

Зокрема, модель спочатку навчається на наборі даних продуктів харчування, витягнутому з LVIS, а потім навчається на мітках з класами, які не прогнозуються добре на наборах даних D2S та RPC. Ця робота досліджує лише результати з навчання мережі на наборі даних LVIS, а покращення з D2S та MVTEC залишається на майбутнє. Перед навчанням дані розділяються у співвідношенні 80:10:10 на тренування, валідацію та тестування відповідно, а також використовується аугментація даних у вигляді зміни розміру зображення для вхідного шару мережі та випадкового перевертання. Навчання проводилось на одному графічному процесорі Tensor Core T4 протягом 24 годин згідно годинника. Слід зазначити, що на рисунку 2.4 спостерігається хороша узгодженість між кривими втрат на тренуванні та валідації, що свідчить про відсутність перенавчання. Зауважимо, що перевірка виконувалася для всього валідаційного набору кожні 1000 кроків через обмеження в інтерфейсі Detectron2.

2.4 Розробка алгоритму роботи програмного модуля розпізнавання продуктів харчування

Відповідно до мети роботи та постановки задачі було розроблено алгоритм програмного модуля розпізнавання продуктів харчування, представлений на рис. 2.10.

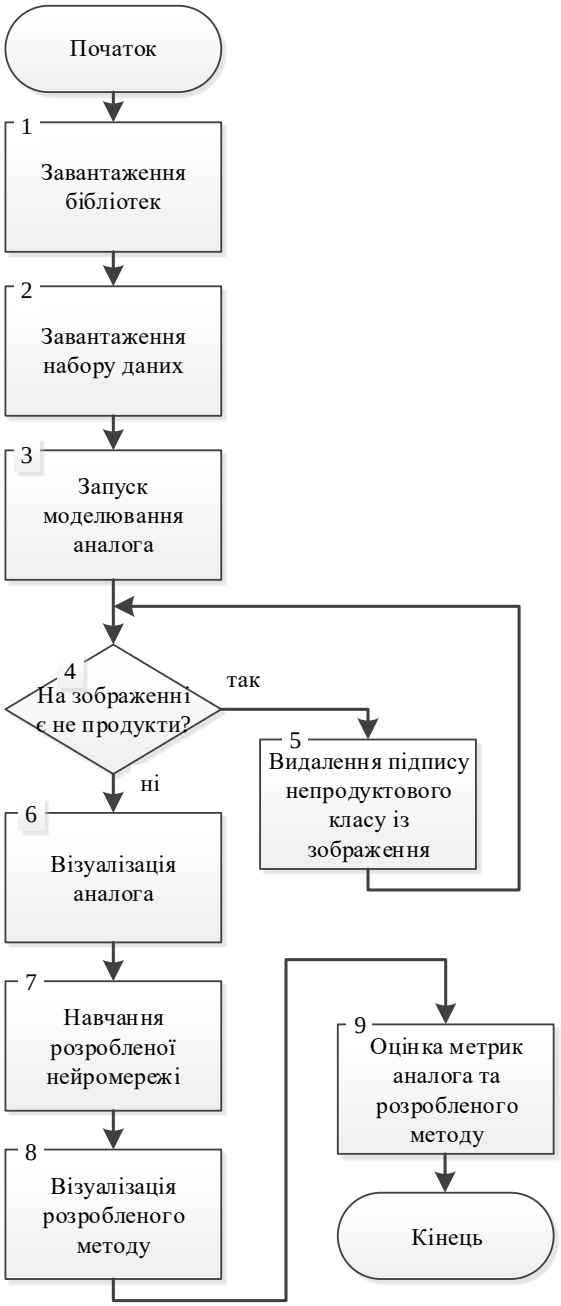

Рисунок 2.10 – Схема алгоритму роботи програмного модуля розпізнавання продуктів харчування

Першим кроком в програмному модулі розпізнавання продуктів харчування буде завантаження необхідних бібліотек (блок 1), а другим кроком буде завантаження набору даних зображень продуктів харчування (блок 2).

Далі переходимо до запуску роботи моделі аналогу (блок 3). Аналогом є переднавчена нейронна мережа, яка натренована розпізнавати не тільки продукти харчування, а різноманітні об`єкти із набору даних MS-COCO [3]. Тому ми для адекватного порівняння з нашою нейромережею (яка навчається розпізнавати тільки продукти харчування), видаляємо із аналога відображення міток класів і обмежувальних прямокутників для тих об`єктів, які не є продуктами харчування (блоки 4 і 5).

Потім переходимо до візуалізації результатів роботи аналога. Тобто відображується 6 зображень різноманітних сцен, на яких проставлено обмежувальні прямокутники та мітки класів продуктів харчування. Результат цієї візуалізації показано на рис. 3.2. А результат роботи аналогу з відображенням усіх назв об’єктів розпізнавання, включаючи нехарчові, показано на рис. 4.2.

Далі проводиться навчання розробленої нейронної мережі (блок 7). Навчання мережі відбувається на наборі даних LVIS, а покращення з D2S та MVTEC залишається на майбутнє. Перед навчанням дані розділяються у співвідношенні 80:10:10 на тренування, валідацію та тестування відповідно.

Потім переходимо до візуалізації результатів роботи розробленої нейромережі (блок 8) . Тобто відображується 6 зображень різноманітних сцен, на яких проставлено обмежувальні прямокутники та мітки класів продуктів харчування. Результат цієї візуалізації показано на рис. 4.3.

Далі переходимо до підрахунку показників якості (метрик) аналога та розробленого методу (блок 9). Основна метрика якості роботи модуля розпізнавання продуктів харчування - це середня влучність. Результати оцінювання метрик для аналога і розробленого модуля наведено відповідно у табл. 4.1 та 4.2.

2.5 Висновок до розділу 2

У розділі було розроблено метод розпізнавання продуктів харчування на основі згорткової нейронної мережі. Із всіх згорткових нейромереж, що реалізовані на популярній платформі Detectron2 модульних моделей комп'ютерного зору на основі PyTorch було обрано архітектуру нейронної мережі Mask R-CNN. Описано принципи роботи та навчання нейронної мережі Mask R-CNN для модуля розпізнавання продуктів харчування. Навчання проводиться за допомогою стохастичного градієнтного спуску по мінібатчам на швидкостях навчання 1E-4, 1E-5 та 1E-6 протягом 12 000, 3 000 та 7 000 ітерацій відповідно, що призводить до насичення графіка втрат. Використовується розмір пакета 8, який представляє верхню межу пам'яті GPU. Розроблено алгоритм роботи програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови програмування та спеціалізованих бібліотек

Для реалізації програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі було обрано мову програмування Python [9].

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Була розроблена у 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічним зв'язуванням та динамічною семантикою роблять її придатною для швидкої розробки програм, а також як засіб поєднання наявних можливостей та компонентів. Python підтримує модулі та пакети модулів, що сприяє повторному використанню коду. Інтерпретатор мови Python та стандартні бібліотеки доступні як у вихідній, так і у скомпільованій формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, аспектно-орієнтована, процедурна та функціональна.

Тобто, Python - це сучасна високорівнева мова програмування, яка привертає увагу розробників простотою, читабельністю та елегантністю. Завдяки простому синтаксису, Python є прекрасним вибором для програмістів-початківців, але він також має потужні функції, які задовольняють потреби досвідчених програмістів-розробників.

Основні позитивні риси Python:

– Легко читабельний код: Синтаксис Python легко зрозуміти і легко прочитати. Код на Python виглядає природньо і майже як псевдокод, який

спрощує процес розробки програм, підвищує зрозумілість та сприяє співпраці між розробниками.

– Легкість вивчення: Python має низький вступний поріг інтеграції. Новачки можуть швидко засвоїти основи і почати писати код. Багато різноманітних навчальних ресурсів і матеріалів доступні для допомоги початківцям.

– Розширюваність: У Python дозволено розширювати свої можливості за допомогою зовнішніх модулів та бібліотек, що значно розширює його функціональність. Python має велике число сторонніх пакетів, які допомагають вирішувати різноманітні задачі.

– Кросплатформність: Python підтримується у різних операційних системах, таких як Windows, Linux і macOS. Це дозволяє розробникам створювати програми, які працюють на різноманітих платформах без необхідності суттєвих змін у коді.

– Багатофункціональність: Python підтримує різноманітні парадигми програмування, включно з об'єктно-орієнтованим програмуванням, процедурним програмуванням та функціональним програмуванням. Це дає розробникам змогу обирати кращий підхід для вирішення конкретних складних завдань.

– Активна широка спільнота: Python має широку та активну спільноту розробників, які внесли суттєвий внесок у розширення функціональності мови та створення корисних інструментів. Це означає, що завжди є підтримка, допомога та великий обсяг ресурсів для розробників Python.

Python застосовується для різних задач, включно з веб-розробкою, науковими дослідженнями, аналізом даних, штучним інтелектом, робототехнікою та багато чим іншим. Він є потужним і гнучким інструментом для програмістів-розробників, які прагнуть простоти в поєднанні з штрокою функціональністю.

Оскільки нейронні мережі використовують багато операцій перемноження векторів та матриць, то для цієї роботи стане у пригоді бібліотека NumPy [10]. NumPy (скорочено від Numerical Python) — це бібліотека з відкритим кодом для мови Python. Вона має такі можливості:

- підтримка багатовимірних масивів (включаючи матриці);
- підтримка математичних функцій високого рівня, які призначені для роботи з багатовимірними масивами.

Математичні алгоритми, які реалізовані інтерпретованими мовами (напр., Python), часто працюють повільніше за ті алгоритми, які реалізовано компільованими мовами (наприклад, Фортран, C, Java). Бібліотека NumPy має реалізації обчислювальних алгоритмів (у виді функцій та операторів), які оптимізовано для роботи з багатомірними масивами. У результаті будь-який алгоритм, що може бути виражений як послідовність операцій над матрицями (масивами) та реалізований із застосуванням NumPy, працює так само швидко, як еквівалентний код у MATLAB.

Так як при проектуванні програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі ми використовуємо набір даних зображень із продуктами харчування для навчання та тестування згорткової нейронної мережі, то нам стане у пригоді бібліотека Pandas.

Pandas - це програмна бібліотека [11], написана для мови Python з метою обробки даних та їх аналізу. Вона, зокрема, пропонує операції та структури даних для маніпулювання таблицями чисел та часовими рядами. Pandas є вільним програмним забезпеченням, яке випускається за трипунктовою ліцензією BSD. Ця назва походить від англійського терміну «panel data» (панельні дані), який в економетрії означає багатовимірні структуровані набори даних.

Можливості бібліотеки Pandas:

- Інструменти для зчитування та запису даних між структурами даних у пам'яті тау різних форматах файлів.

- Об'єкт DataFrame із вбудованою індексацією для маніпулювання даними.

- Вирівнювання даних та вбудована підтримка відновлення пропущених даних.

- Отримання зрізів за мітками, індексування з розширеними функціональними можливостями та отримання піднаборів із великих наборів даних.

- Переформатовування для формування зведених наборів даних.

- Вставлення та вилучення стовпчиків у структурах даних.

- З'єднання та злиття наборів даних.

- Рушій групування, який дозволяє робити з наборами даних операції зміни-розділення-об'єднання (англ. apply-split-combine).

- Набір функцій для часових рядів: породження діапазонів дат та перетворення частот, статистика рухомого вікна, лінійні регресії рухомого вікна, зсув дат та запізнювання.

- Ієрархічне індексування осей для роботи з даними високої розмірності у структурі даних меншої вимірності.

Ця бібліотека суттєво оптимізована за продуктивністю, критичні ланцюги коду написано на мовах програмування C та Cython.

Оскільки при проектуванні програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі ми виконуємо побудову різних графіків та діаграм, то нам знадобится бібліотека Matplotlib [12].

Matplotlib - комплексна бібліотека для створення статичних, інтерактивних та анімованих візуалізацій на мові Python. Matplotlib має такі функціональні спроможності:

- Створення якісних сюжетів для публікацій.

- Створення інтерактивних фігур, котрі можна масштабувати, оновлювати, панорамувати.

- Налаштовування візуального макету і стилю.

- Експорт у велику кількість форматів файлів.
- Інтегрування у JupyterLab і у графічний користувацький інтерфейс.
- Використання широкого набору зовнішніх пакетів, побудованих на основі Matplotlib.

Matplotlib може створювати публікації цифрової якості у різних форматах друкованих копій та інтерактивних середовищах на різноманітних платформах. Matplotlib може бути використаний у сценаріях мови Python, серверах веб-додатків, оболонках Python/IPython та у різноманітних наборах інструментів графічного користувацького інтерфейсу.

3.2 Опис набору даних продуктів харчування для навчання та тестування модуля

Було сформовано комбінований набір даних, який включає набори даних LVIS [4], D2S [6] та RPC: великий набір даних для перевірки роздрібних товарів [7]. Набір даних LVIS містить приблизно 1000 міток, багато з яких включають не-харчові предмети, такі як лампи, чайники і т.д. Тому було здійснено передобробку даних, відкинуто нехарчові класи, і залишено лише 186 класів продуктів харчування. Це зменшило кількість зображень з приблизно 120 000 до приблизно 19 000. Ці 19 000 зображень мають загальну кількість приблизно 260 000 екземплярів об'єктів харчування (див. рис. 3.1).

category	#inst.	category	#inst.	category	#inst.
alcohol	207	almond	1700	apple	17451
applesauce	7	apricot	62	artichoke	293
asparagus	969	avocado	1048	bagel	372
banana	50552	batter_(food)	26	beef_(food)	1242
beer_bottle	1227	beer_can	203	bell_pepper	4369
blackberry	406	blueberry	2114	boiled_egg	125
bread	6550	broccoli	12166	brownie	217
brussels_sp..	590	bubble_gum	4	bun	1780
burrito	14	butter	308	cake	2297
candy_bar	29	candy_cane	107	cantaloup	193
carrot	18049	casserole	12	cauliflower	1035
cayenne_(sp..	49	celery	911	cherry	903
chickpea	265	chili_(vege..	354	crisp_(pota..	541
chocolate_bar	179	chocolate_c..	80	chocolate_m..	7
chocolate_m..	1	cider	38	clementine	108
cocoa_(beve..	4	coconut	273	coleslaw	72
cookie	1920	edible_corn	1883	cornbread	10
cornmeal	1	crab_(animal)	50	crabmeat	5
cracker	510	crape	12	cream_pitcher	10
crescent_roll	152	crouton	140	crumb	3021
cucumber	1533	cupcake	1628	date_(fruit)	103
doughnut	11911	egg	813	egg_roll	15
egg_yolk	90	eggplant	337	escargot	5
fig_(fruit)	147	fish	525	fish_(food)	96
French_toast	41	fruit_juice	37	garlic	487
gelatin	248	ginger	93	gourd	101
grape	6377	green_bean	2571	green_onion	1618
grits	3	ham	1765	hamburger	126
honey	90	hot_sauce	70	hummus	9
icecream	180	popsicle	1	ice_pack	4
jam	29	jelly_bean	116	keg	6
kiwi_fruit	702	lamb-chop	8	lasagna	7
legume	333	lemon	2168	lemonade	2
lettuce	5500	lime	1134	liquor	66
lollipop	59	mandarin_or..	401	martini	3
mashed_potato	58	meatball	174	melon	167
milk	227	milk_can	8	milkshake	1
mint_candy	27	muffin	352	mushroom	6257
nut	790	octopus_(fo..	5	olive_oil	36
omelet	10	onion	9779	orange_(fru..	13034
orange_juice	223	pancake	295	papaya	206
pastry	4972	patty_(food)	20	pea_(food)	1869
peach	1041	peanut_butter	50	pear	1069
pepper	697	pepper_mill	91	persimmon	22
pickle	632	pie	228	pineapple	1636
pita_(bread)	28	pizza	4103	pop_(soda)	951
potato	4393	prawn	779	pretzel	179
prune	8	pudding	2	quesadilla	6
quiche	33	radish	519	raspberry	778
rib_(food)	32	root_beer	3	salad	171
salami	290	salmon_(food)	14	salsa	22
saltshaker	543	sandwich	2315	smoothie	53
soup	193	sour_cream	49	soya_milk	2
crawfish	5	squid_(food)	6	steak_(food)	139
stew	7	strawberry	4386	string_cheese	1
sugar_bowl	10	sugarcane_(..	31	sushi	337
sweet_potato	137	Tabasco_sauce	5	taco	21
tea_bag	42	tequila	2	toast_(food)	125
tomato	12338	truffle_(ch..	4	turkey_(food)	120
turnip	109	vinegar	1	vodka	3
waffle	61	watermelon	814	whipped_cream	201
wine_bottle	4449	yogurt	116	zucchini	798
total	262491				

Рисунок 3.1 – Назви класів та кількість прикладів харчових продуктів у наборі даних

Для наборів даних D2S і RPC єдине, що потрібно було зробити, - це переконатися, що ідентифікатори категорій з цих наборів даних відповідають набору даних LVIS. На рисунку 3.2 показано шість знімків з нашого тестового набору, з мітками для областей та сегментації, де всі нехарчові категорії більше не розглядаються як мітки.

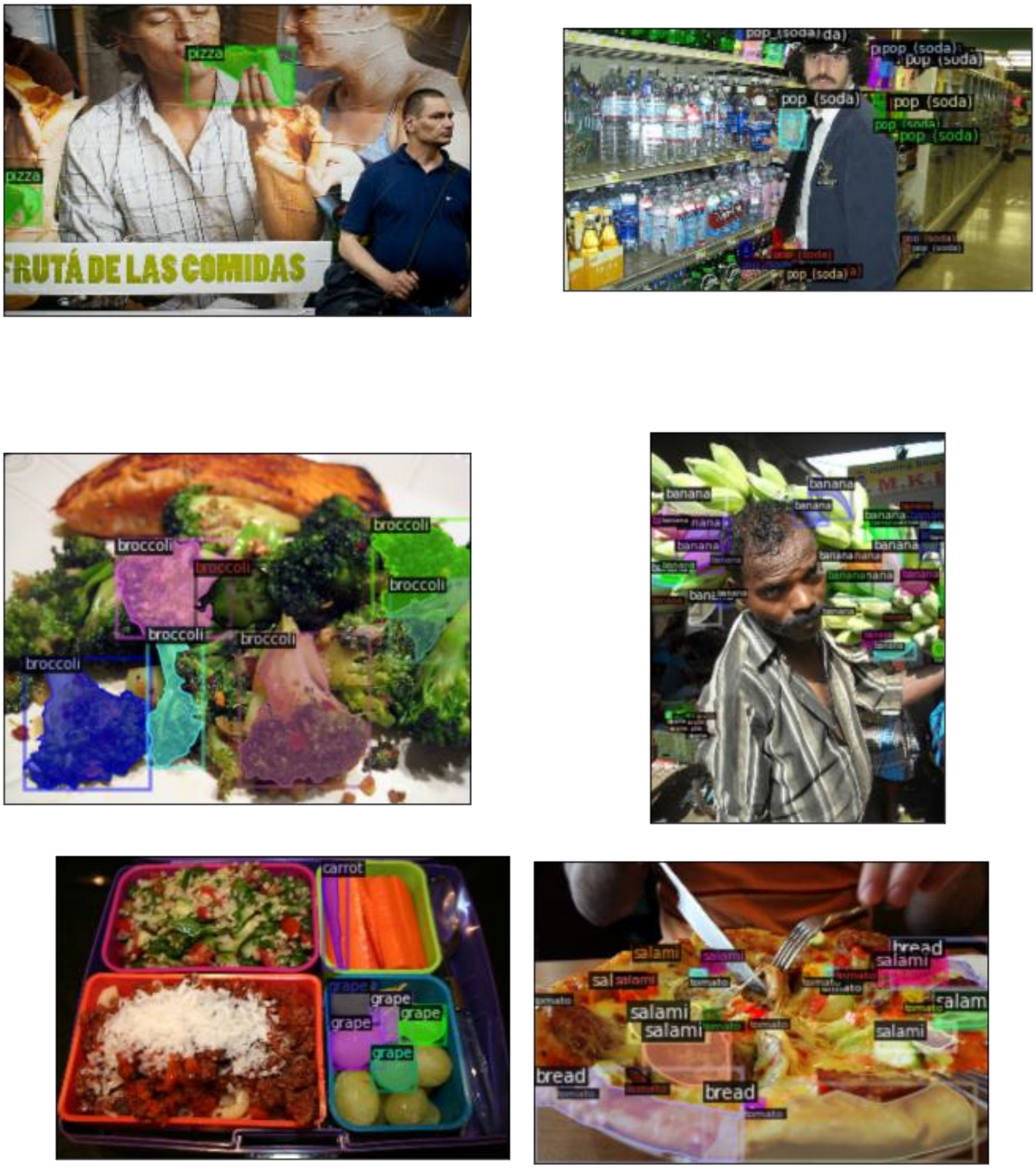

Рисунок 3.2 – Результати роботи програми-аналогу, де відображено назви тільки харчових категорій

3.3 Програмна реалізація модуля розпізнавання продуктів харчування

Першим кроком в нейромережевому модулі розпізнавання продуктів харчування буде завантаження бібліотек:

```
import torch
import sys
import multiprocessing
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import os, json, cv2, random

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

import matplotlib.pyplot as plt
import json
import pandas as pd
```

Далі потрібно завантажити набір даних:

```
lvis_train_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_train_data.jso
n', './data/train2017','lvis_train')

DatasetCatalog.register("lvis_train",lambda train='train': lvis_train_dict)

lvis_val_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_val_data.json'
, './data/train2017','lvis_val')

DatasetCatalog.register("lvis_val",lambda val='val': lvis_val_dict)

lvis_test_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_test_data.json
', './data/train2017','lvis_test')

DatasetCatalog.register("lvis_test",lambda test='test': lvis_test_dict)
```

Потім переходимо до візуалізації результатів аналізу набору даних.

```
truth_list = []
for d in lvis_test_dict[66:72]:
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1],
metadata=MetadataCatalog.get('lvis_test'), scale=0.5)
    out = visualizer.draw_dataset_dict(d)
    truth_list.append(out.get_image()[:, :, :])

x=2
y=3
fig, axs = plt.subplots(x,y, figsize=(18,10), sharey=False,sharex=False)

for i in range(x):
    for j in range(y):
        axs[i,j].imshow(truth_list[j+i*y],interpolation='gaussian')
        axs[i,j].set_yticks([])
        axs[i,j].set_xticks([])
# fig.savefig("./io.pdf", bbox_inches="tight")
fig.savefig("./true_vis.pdf", bbox_inches="tight")
```

Далі проведемо навчання нейронної мережі `mask_rcnn_R_50_FPN_3x`:

```
# epochs = 10
it= 25001
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("lvis_train",)
cfg.DATASETS.TEST = ("lvis_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")  # Let training initialize
from model zoo
cfg.SOLVER.IMS_PER_BATCH = 8  # This is the real "batch size" commonly known
to deep learning people
cfg.SOLVER.BASE_LR = 1e-5  # pick a good LR
cfg.SOLVER.MAX_ITER =
it#int(epochs*len(lvis_train_dict)/cfg.SOLVER.IMS_PER_BATCH)  # 300
iterations seems good enough for this toy dataset; you will need to train
longer for a practical dataset
cfg.SOLVER.GAMMA = 0.1 # The iteration number to decrease learning rate by
GAMMA.
cfg.SOLVER.STEPS = (12001,15001,18001)     #  decay learning rate
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512   # The "RoIHead batch size".
128 is faster, and good enough for this toy dataset (default: 512)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 186  # only has one class (ballon). (see
https://detectron2.readthedocs.io/tutorials/datasets.html#update-the-config-
for-new-datasets)
cfg.SOLVER.CHECKPOINT_PERIOD = 3000
cfg.TEST.EVAL_PERIOD = 1000
cfg.OUTPUT_DIR = './output_train'
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
```

Із фрагменту коду вище видно, що ми задали 10 епох навчання. Навчання проводимо із пакетною корекцією. Розмір пакету 8 зображень. Швидкість навчання задали 1e-5, коефіцієнт гамма методу навчання 0,1.

Використовуємо поступове зменшення швидкості навчання. Мережа навчається розпізнавати 186 класів продуктів харчування.

Параметри моделі нейронної мережі відображає наступний фрагмент коду:

```
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fals
e
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
```

Після кожної епохи навчання програма виводить середні значення влучності (Average Precision) по категоріям продуктів харчування як показано на наступному фрагменті

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.016
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.025
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.017
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.014
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.027
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.037
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.017
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.048
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.064
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.051
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.110
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.148
[05/29 16:06:47 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|  AP   |  AP50  |  AP75  |  APs  |  APm  |  APl  |
|:-----:|:------:|:------:|:-----:|:-----:|:-----:|
| 1.579 | 2.483  | 1.656  | 1.435 | 2.707 | 3.700 |
[05/29 16:06:47 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category         | AP     | category            | AP     | category            | AP     |
|:-----------------|:-------|:--------------------|:-------|:--------------------|:-------|
| alcohol          | 0.000  | almond              | 0.000  | apple               | 12.982 |
| applesauce       | 0.000  | apricot             | nan    | artichoke           | 0.000  |
| asparagus        | 0.000  | avocado             | 0.000  | bagel               | 0.000  |
| banana           | 14.225 | batter_(food)       | 0.000  | beef_(food)         | 0.045  |
| beer_bottle      | 3.430  | beer_can            | 0.000  | bell_pepper         | 0.918  |
| blackberry       | 0.000  | blueberry           | 2.325  | boiled_egg          | 0.000  |
| bread            | 4.436  | broccoli            | 16.165 | brownie             | 0.000  |
| brussels_sprouts | 0.000  | bubble_gum          | nan    | bun                 | 9.912  |
| burrito          | 0.000  | butter              | 0.000  | cake                | 11.321 |
| candy_bar        | 0.000  | candy_cane          | 0.000  | cantaloup           | 0.000  |
| carrot           | 12.953 | casserole           | 0.000  | cauliflower         | 0.170  |
```

Потім переходимо до візуалізації результатів роботи розробленої нейромережі. Тобто відображується 6 зображень різноманітних сцен, на яких проставлено обмежувальні прямокутники та мітки класів продуктів харчування. Результат цієї візуалізації показано на рис. 4.3.

Далі переходимо до підрахунку показників якості (метрик) аналога:

```
coco_test_dict = detectron2.data.datasets.load_coco_json('./data/coco_test_data.json', './data/train2017','coco_test')
```

```
[05/31 16:17:32 d2.data.datasets.coco]: Loaded 2383 images in COCO format from ./data/coco_test_data.json
```

```
DatasetCatalog.register("coco_test",lambda test='test': coco_test_dict)
```

```
cfg_baseline = get_cfg()
# add project-specific config (e.g., TensorMask) here if you're not running a model in detectron2's core library
cfg_baseline.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg_baseline.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model
# Find a model from detectron2's model zoo. You can use the https://dl.fbaipublicfiles... url as well
cfg_baseline.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
predictor_baseline = DefaultPredictor(cfg_baseline)
```

```
evaluator = COCOEvaluator("coco_test", output_dir="./output_train")
coco_loader = build_detection_test_loader(cfg_baseline, "coco_test")
print(inference_on_dataset(predictor_baseline.model, coco_loader, evaluator))
```

та розробленого методу:

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("lvis_train",)
cfg.DATASETS.TEST = ("lvis_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")  # Let training in
cfg.SOLVER.IMS_PER_BATCH = 8  # This is the real "batch size" commonly known to deep learning people
cfg.SOLVER.BASE_LR = 1e-5  # pick a good LR
cfg.SOLVER.MAX_ITER = 25001#int(epochs*len(lvis_train_dict)/cfg.SOLVER.IMS_PER_BATCH)  # 300 iterations seems good enough f
cfg.SOLVER.GAMMA = 0.1 # The iteration number to decrease learning rate by GAMMA.
cfg.SOLVER.STEPS = (12001,15001,18001)     #  decay learning rate
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512   # The "RoIHead batch size". 128 is faster, and good enough for this toy da
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 186  # only has one class (ballon). (see https://detectron2.readthedocs.io/tutorials/data
cfg.SOLVER.CHECKPOINT_PERIOD = 3000
cfg.TEST.EVAL_PERIOD = 1000
cfg.OUTPUT_DIR = './output_train'
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")  # path to the model we just trained
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.3   # set a custom testing threshold
predictor = DefaultPredictor(cfg)
from detectron2.utils.visualizer import ColorMode
dataset_dicts = lvis_test_dict
my_list = []
random.seed(69)
for d in dataset_dicts[66:72]:
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)  # format is documented at https://detectron2.readthedocs.io/tutorials/models.html#model-output
    v = Visualizer(im[:, :, ::-1],
                   metadata=MetadataCatalog.get('lvis_test'),
                   scale=0.5,
#                    instance_mode=ColorMode.IMAGE_BW   # remove the colors of unsegmented pixels. This option is only avail
    )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    my_list.append(v.draw_instance_predictions(outputs["instances"].to("cpu")))
```

Основна метрика якості роботи модуля розпізнавання продуктів харчування - це середня влучність.

Результати оцінювання метрик для аналога і розробленого модуля наведено відповідно у табл. 4.1 та 4.2.

3.3 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек NumPy, Pandas та Matplotlib для програмної реалізації модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі. Проведено аналіз набору даних продуктів харчування для навчання та тестування модуля, який містить приблизно 19 000 зображень , на яких присутні 186 класів продуктів харчування. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі, що складається з завантаження бібліотек, завантаження набору даних із зображеннями, запуску роботи моделі аналогу і запропонованої нейромережі, візуалізації результатів роботи аналогу і розробленої нейромережі, підрахунку показників якості (метрик) аналога та розробленої нейромережі.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Метрики для оцінювання якості роботи модуля розпізнавання продуктів харчування

Одна із метрик для множин A і B - перетин над об'єднанням (Intersection over Union, *IoU*) обчислюється як: $\frac{A \cap B}{A \cup B}$. Цю метрику можна застосувати як до обмежувальних рамок, так і до масок сегментації.

При застосуванні моделі виявлення об'єктів на зображенні: нехай об'єкти істинної інформації будуть $A_1 ... A_n$, а наша модель зробила прогнози (або обмежувальні рамки, або маски сегментації) $P_1 ... P_n$, кожен з впевненістю C_i. Припускаючи, що кожен прогноз P_i перекривається максимум одним об'єктом істинної інформації, A_{ki}, ми обчислюємо IoU для прогнозу P_i та A_{ki}, як:

$$IoU_i = \frac{|P_i \cap A_{k_i}|}{|P_i \cup A_{k_i}|}. \quad (4.1)$$

Для порогу T: ми класифікуємо прогноз P_i як **правильно позитивний** ($X_i(T) = 1$), коли $IoU_i \geq T$, і як **хибно позитивний** ($X_i(T) = 0$), коли $IoU_i < T$. Без втрати загальності, нехай C_i буде незростаючою послідовністю. Потім ми можемо обчислити влучність і повноту для кожного $i \leq n$:

$$Presicion_i = \frac{\sum_{j=0}^{i} X_i(T)}{i} \quad (4.2)$$

$$Recall_i = \frac{\sum_{j=0}^{i} X_i(T)}{k} \quad (4.3)$$

Більш просто визначення влучності та повноти показано на рис. 4.1.

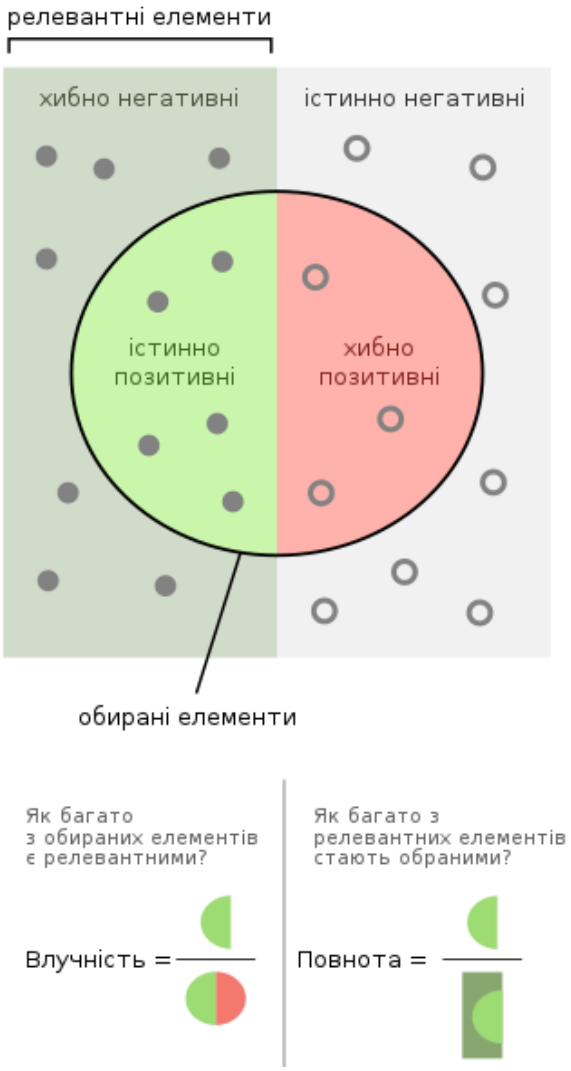

Рисунок 4.1 – Визначення влучності та повноти

Графік точок *(Recall$_i$, Precision$_i$)* дає **криву влучності-повноти**. Це можна зробити монотонно зростаючим, застосовуючи:

$$\widehat{Precision}(r) = max_{\hat{r} \geq r} Precision(\hat{r}) \tag{4.4}$$

Тоді ми визначаємо середню влучність для цього порогу як:

$$P(T) = \int_0^1 \widehat{Precision}\,(r)dr \tag{4.5}$$

(загалом це обчислюється чисельно на кількох рівномірно розподілених точках на інтервалі [0, 1], наприклад, COCO використовує 101 точку)

Наша основна метрика оцінки - це середня влучність маски (AP), як описано в наборі даних COCO: для класу *C* та порогу *T*, $AP_C(T)$ - це середня влучність для класу *C* (проведення обчислень *IoU* на масках сегментації, позначення виявлень на основі порогу *T*). AP_C - це середня з $AP_C(T)$, де *T* змінюється від 0.5 до 0.95 з кроком 0.05. *AP* - це середня AP_C для всіх класів *C*.

4.2 Оцінювання якості роботи модуля розпізнавання продуктів харчування

Як відзначалось у розділі 1, як аналог (або як базову модель) для порівняння з розробленою моделлю, було обрано програму розпізнавання продуктів харчування на основі MS-COCO [3]. Тому спочатку було оцінено кількісні результати у формі середньої влучності класів продуктів харчування (AP - average precision) базової моделі MS-COCO на тестовому наборі. Результати цього узагальнено в Таблиці 4.1.

Таблиця 4.1 – Середня влучність (AP) усіх класів продуктів харчування з базової моделі (аналога)

Клас	Банан	Яблуко	Сендвіч	Помаранч	Броколі	Морква	Піца	Пончик	Тістечко	Хотдог	**По всіх**
Середня влучність сегмента %	25,6	24,3	42,3	32,5	28,0	24,7	56,9	55,1	37,5	28,6	**35,6**
Середня влучність рамки %	29,7	25,1	42,2	32,9	30,7	27,9	58,6	55,1	37,6	31,7	**37,2**

Із табл. 4.1 видно, що базова модель має середню влучність обмежувальних рамок та сегментації від 20 до 60 для всіх 10 різних класів їжі у наборі даних MS-COCO.

Таблиця 4.2 показує середню влучність класів розробленої моделі на тестовому наборі.

Таблиця 4.2 –Середні влучності (AP) класів їжі розробленої моделі

Клас	Банан	Яблуко	Сендвіч	Помаранч	Броколі	Морква	Піца	Пончик	Тістечко	Хотдог
Середня влучність сегмента %	37,2	31,4	55,1	31,2	54,8	31,7	27,3	47,2	58,7	19,3
Середня влучність рамки %	33,7	28,2	51,9	30,5	49,6	28,2	24,5	49,3	56,1	23,8

Клас	Листя салату	Гриби	Горіхи	Цибуля	Паста	Картопля	Полуниця	Помідор	Вино
Середня влучність сегмента %	35,4	42,2	38,1	42,3	35,4	32,9	45,1	37,1	70,5
Середня влучність рамки %	33,9	37,7	35,9	45,6	37,3	34,8	39,4	38,5	69,7

Клас	Пиво	Болгарський перець	Чорниця	Панірувальна крихта	Кекс	Виноград	Лимон	По всіх
Середня влучність сегмента %	50,1	34,6	29,9	53,6	44,7	55,3	65,2	**42,55**
Середня влучність рамки %	45,4	29,7	25,1	47,8	38,4	47,2	61,3	**40,13**

Порівняно з результатами аналога із Таблиці 4.1, середня влучність аналога для сегмента склаає 35,6%, а розробленої програми 42,55% (збільшено майже на 7%); середня влучність аналога для рамки склаає 37,2%, а розробленої програми 40,13% (збільшено майже на 3%). Але по окремих класах розроблена програма має нижчі показники влучності. Наприклад, по таких класах як помаранч і пончик влучність є трохи нижчою, ніж у базової моделі. А по таких класах як піца і хот-дог влучність є суттєво нижчою, ніж у базової моделі. Це зниження ефективності може бути в значній мірі приписане складнішій задачі навчання. Зокрема, наша нова модель навчена розпізнавати 186 класів їжі, багато з яких мають схожість один з одним, наприклад, яблука та груші, торти та кекси, хліб та піца і т.д. Крім того, порівняння двох різних моделей, кожна з яких навчена на іншому наборі даних, може бути некоректним, оскільки мітки в наборі даних MS-COCO не повністю ідентичні міткам набору даних LVIS. Однак, зауважимо, що нова модель у середньому по всіх класах більш успішна, ніж базова модель, а також здатна виявляти 17 нових класів їжі з ненульовою середньою точністю. Є деякі класи продуктів харчування, які не можуть бути добре розпізнані. Це в основному зумовлено дисбалансом класів у наявному наборі даних LVIS. Цю проблему планується вирішити в майбутніх дослідженнях, доповнивши навчальний набір наборами даних RPC та D2S.

На рисунку 4.2 візуалізовані шість скріншотів результатів розпізнавання базової моделі (аналога). Зауважимо, що водночас із продуктовими класами з Таблиці 4.1 ця модель також визначає багато нехарчових предметів, таких як люди, миски, ножі тощо. Це робить базову модель непридатною для наших цілей створення програми виявлення саме продуктів харчування.

Рисунок 4.2 – Скріншоти роботи програми-аналога

На рис. 4.3 візуалізовані прогнози розробленої мережі. Тут модель була навчена розпізнавати лише продукти харчування. Модель успішно розпізнає більшу кількість продуктів харчування порівняно з кількістю продуктів харчування, що розпізнає аналог (див. рис. 3.2). Це чітко спостерігається за визначенням компонентів салату та моркви (знизу ліворуч), більшої кількості броколі (всередині ліворуч), більшої кількості бананів (всередині праворуч) та

овочевих начинок на піці (знизу праворуч). Однак модель плутає хліб з піцею, пляшки вина з пляшками води, та томати з салямі. Це зрозумілий вид помилки, оскільки перелічені продукти мають дуже схожі форми та кольори на свої помилкові аналоги.

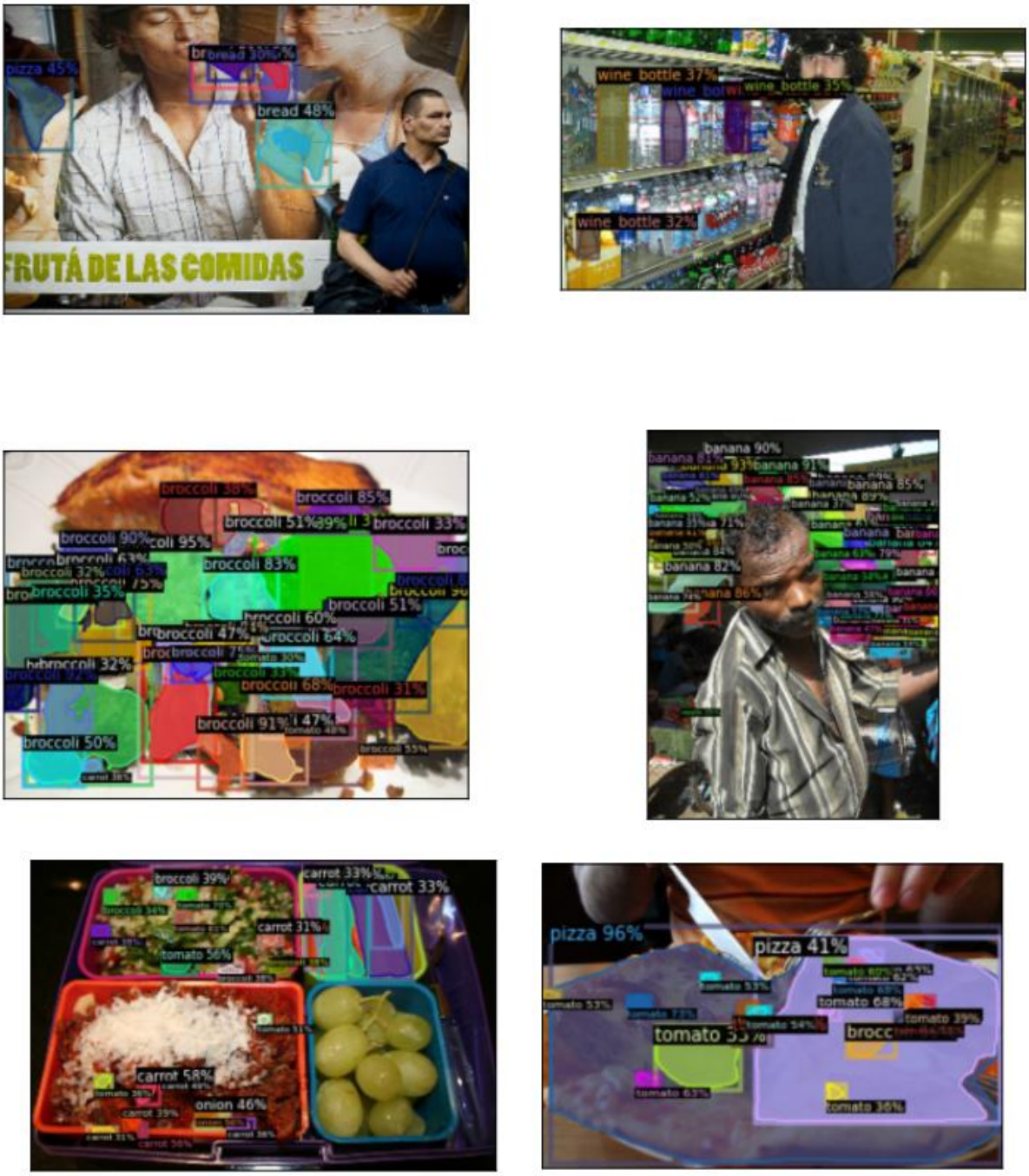

Рисунок 4.3 – Скріншоти роботи розробленої програми

Крім того, поточна модель не може розпізнавати пляшки газованої води, які бачимо на верхньому правому знімку, що пов'язано з дисбалансом класів, який бачимо на рис. 4.1. Ці помилкові розпізнавання можна виправити в майбутніх роботах, включивши більше зразків цих неузгоджених класів та схожих предметів харчування в навчальний набір. Однак це може бути складним, оскільки ручна анотація додаткових зразків продуктів харчування може бути затратною по часу.

Із табл. 4.1 і 4.2 видно, що середня влучність аналога для сегмента складає 35,6%, а розробленої програми 42,55%; середня влучність аналога для рамки складає 37,2%, а розробленої програми 40,13%.

Таким чином, можна зробити висновок, що розроблений модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі має порівняно з аналогом збільшену на 7% влучність розпізнавання сегмента та збільшену на 3% влучність розпізнавання рамки. Тобто мета роботи досягнута – якість розпізнавання продуктів харчування підвищена.

4.3 Висновок до розділу 4

У розділі в результаті тестування програми було доведено її працездатність та відповідність поставленому завданню. Для оцінки якості роботи модуля використовувалась така метрика як влучність розпізнавання. Розроблений програмний модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі має середню влучність для сегмента 42,55%, а аналог - 35,6%; середню влучність для рамки 40,13%, а аналог - 37,2%. Таким чином, розроблений модуль розпізнавання продуктів харчування має порівняно з аналогом збільшену на 7% влучність розпізнавання сегмента та збільшену на 3% влучність розпізнавання рамки. Тобто мета роботи досягнута – якість розпізнавання продуктів харчування підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу розпізнавання продуктів харчування на основі згорткової нейронної мережі. В ході аналізу предметної області було розглянуто детальну постановку задачі розпізнавання продуктів харчування. Також розглянуто різні методи розв'язання задачі розпізнавання об`єктів і оцінено їх застосовність до завдання розпізнавання продуктів харчування. Серед таких методів розпізнавання об`єктів як детерміновані методи, ймовірнісні методи, логічні методи, структурні (лінгвістичні) методи, методи потенціалів, експертні методи та нейромережеві методи як найбільш перспективний і застосовний до даної задачі було обрано нейромережевий метод. Було показано, що з різних типів нейронних мереж найбільше підходить для вирішення поставленої задачі саме згорткові нейронні мережі.

Також було розроблено метод розпізнавання продуктів харчування на основі згорткової нейронної мережі. Із всіх згорткових нейромереж, що реалізовані на популярній платформі Detectron2 модульних моделей комп'ютерного зору на основі PyTorch було обрано архітектуру нейронної мережі Mask R-CNN. Описано принципи роботи та навчання нейронної мережі Mask R-CNN для модуля розпізнавання продуктів харчування. Навчання проводилось за допомогою стохастичного градієнтного спуску по мінібатчам на швидкостях навчання 1E-4, 1E-5 та 1E-6 протягом 12 000, 3 000 та 7 000 ітерацій відповідно, що призводить до насичення графіка втрат. Використовується розмір пакета 8, який представляє верхню межу пам'яті GPU. Було розроблено алгоритм роботи програмного модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі.

Було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек NumPy, Pandas та Matplotlib для програмної реалізації модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі. Проведено аналіз набору даних продуктів харчування для навчання та тестування модуля, який містить приблизно 19 000 зображень, на яких

присутні 186 класів продуктів харчування.. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання продуктів харчування на основі згорткової нейронної мережі, що складається з завантаження бібліотек, завантаження набору даних із зображеннями, запуску роботи моделі аналогу і запропонованої нейромережі, візуалізації результатів роботи аналогу і розробленої нейромережі, підрахунку показників якості (метрик) аналога та розробленої нейромережі.

У результаті тестування програми було доведено її працездатність та відповідність поставленому завданню. Для оцінки якості роботи модуля використовувалась така метрика як влучність розпізнавання. Розроблений програмний модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі має середню влучність для сегмента 42,55%, а аналог - 35,6%; середню влучність для рамки 40,13%, а аналог - 37,2%. Таким чином, розроблений модуль розпізнавання продуктів харчування має порівняно з аналогом збільшену на 7% влучність розпізнавання сегмента та збільшену на 3% влучність розпізнавання рамки. Тобто мета роботи досягнута – якість розпізнавання продуктів харчування підвищена.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Куссуль Н. М. Інтелектуальні обчислення: навчальний посібник (із грифом МОН України) / Н. М. Куссуль, А. Ю. Шелестов, А. Н. Лавренюк. - К.: «Наукова думка», 2006. — 186 с. ISBN 966-00-0592-X.

2 Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. [Електронний ресурс] – Режим доступу: https://github.com/facebookresearch/detectron2, 2019

3 T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, Computer Vision – ECCV 2014, pages 740–755, Cham, 2014. Springer International Publishing.

4 A. Gupta, P. Dollar, and R. Girshick. Lvis: A dataset for large vocabulary instance segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

5 Y. Wei, S. Tran, S. Xu, B. Kang, and M. Springer. Deep learning for retail product recognition: Challenges and techniques. Computational Intelligence and Neuroscience,, page 8875910, 2020.

6 P. Follmann, T. Bottger, P. H ¨ artinger, R. K ¨ onig, and M. Ul- ¨ rich. Mvtec d2s: Densely segmented supermarket dataset. In European Conference on Computer Vision (ECCV), 2018.

7 X.-S. Wei, Q. Cui, L. Yang, P. Wang, and L. Liu. Rpc: A large-scale retail product checkout dataset, 2019.

8 T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2016.

9 Python [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

10 NumPy [Електронний ресурс] – Режим доступу: https://numpy.org/

11 Pandas - Python Data Analysis Library [Електронний ресурс] – Режим доступу: https://pandas.pydata.org/

12 Matplotlib: Visualization with Python [Електронний ресурс] – Режим доступу: https://matplotlib.org/

13 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп'ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль розпізнавання продуктів харчування на основі згорткової нейронної мережі

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 94,4% Схожість 5,6%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Новіцька О. В.

Керівник роботи Паночишин Ю.М.

Додаток Б (обов'язковий)

Лістинг програми

```
import torch
import sys
import multiprocessing
print('__Python VERSION:', sys.version)
print('__pyTorch VERSION:', torch.__version__)
print('__CUDA VERSION')
from subprocess import call
# call(["nvcc", "--version"]) does not work
! nvcc --version
print('__CUDNN VERSION:', torch.backends.cudnn.version())
print('__Number CUDA Devices:', torch.cuda.device_count())
print('__Devices')
call(["nvidia-smi", "--format=csv", "--query-
gpu=index,name,driver_version,memory.total,memory.used,memory.free"])
print('Active CUDA Device: GPU', torch.cuda.current_device())

print ('Available devices ', torch.cuda.device_count())
print ('Current cuda device ', torch.cuda.current_device())

print('CPUs:', multiprocessing.cpu_count())

# Some basic setup:
# Setup detectron2 logger

import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import os, json, cv2, random

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

import matplotlib.pyplot as plt
import json
import pandas as pd

from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import DatasetMapper, build_detection_test_loader

from detectron2.engine.hooks import HookBase
from detectron2.evaluation import inference_context
from detectron2.utils.logger import log_every_n_seconds

from detectron2.engine import DefaultTrainer
```

```
import detectron2.utils.comm as comm
import torch
import time
import datetime
import logging

class LossEvalHook(HookBase):
    def __init__(self, eval_period, model, data_loader):
        self._model = model
        self._period = eval_period
        self._data_loader = data_loader

    def _do_loss_eval(self):
        # Copying inference_on_dataset from evaluator.py
        total = len(self._data_loader)
        num_warmup = min(5, total - 1)

        start_time = time.perf_counter()
        total_compute_time = 0
        losses = []
        for idx, inputs in enumerate(self._data_loader):
            if idx == num_warmup:
                start_time = time.perf_counter()
                total_compute_time = 0
            start_compute_time = time.perf_counter()
            if torch.cuda.is_available():
                torch.cuda.synchronize()
            total_compute_time += time.perf_counter() - start_compute_time
            iters_after_start = idx + 1 - num_warmup * int(idx >= num_warmup)
            seconds_per_img = total_compute_time / iters_after_start
            if idx >= num_warmup * 2 or seconds_per_img > 5:
                total_seconds_per_img = (time.perf_counter() - start_time) /
iters_after_start
                eta = datetime.timedelta(seconds=int(total_seconds_per_img * (total -
idx - 1)))
                log_every_n_seconds(
                    logging.INFO,
                    "Loss on Validation  done {}/{}. {:.4f} s / img. ETA={}".format(
                        idx + 1, total, seconds_per_img, str(eta)
                    ),
                    n=5,
                )
            loss_batch = self._get_loss(inputs)
            losses.append(loss_batch)
        mean_loss = np.mean(losses)
        self.trainer.storage.put_scalar('validation_loss', mean_loss)
        comm.synchronize()

        return losses

    def _get_loss(self, data):
        # How loss is calculated on train_loop
        metrics_dict = self._model(data)
        metrics_dict = {
            k: v.detach().cpu().item() if isinstance(v, torch.Tensor) else float(v)
            for k, v in metrics_dict.items()
        }
```

```
        total_losses_reduced = sum(loss for loss in metrics_dict.values())
        return total_losses_reduced


    def after_step(self):
        next_iter = self.trainer.iter + 1
        is_final = next_iter == self.trainer.max_iter
        if is_final or (self._period > 0 and next_iter % self._period == 0):
            self._do_loss_eval()
        self.trainer.storage.put_scalars(timetest=12)

class MyTrainer(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        if output_folder is None:
            output_folder = os.path.join(cfg.OUTPUT_DIR, "inference")
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

    def build_hooks(self):
        hooks = super().build_hooks()
        hooks.insert(-1,LossEvalHook(
            cfg.TEST.EVAL_PERIOD,
            self.model,
            build_detection_test_loader(
                self.cfg,
                self.cfg.DATASETS.TEST[0],
                DatasetMapper(self.cfg,True)
            )
        ))
        return hooks

lvis_train_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_train_data.json',
'./data/train2017','lvis_train')

DatasetCatalog.register("lvis_train",lambda train='train': lvis_train_dict)

lvis_val_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_val_data.json',
'./data/train2017','lvis_val')

DatasetCatalog.register("lvis_val",lambda val='val': lvis_val_dict)

lvis_test_dict =
detectron2.data.datasets.load_coco_json('./data/detectron_lvis_test_data.json',
'./data/train2017','lvis_test')

DatasetCatalog.register("lvis_test",lambda test='test': lvis_test_dict)

truth_list = []
for d in lvis_test_dict[66:72]:
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1],
metadata=MetadataCatalog.get('lvis_test'), scale=0.5)
    out = visualizer.draw_dataset_dict(d)
    truth_list.append(out.get_image()[:, :, :])
x=2
y=3
```

```
fig, axs = plt.subplots(x,y, figsize=(18,10), sharey=False,sharex=False)

for i in range(x):
    for j in range(y):
        axs[i,j].imshow(truth_list[j+i*y],interpolation='gaussian')
        axs[i,j].set_yticks([])
        axs[i,j].set_xticks([])
# fig.savefig("./io.pdf", bbox_inches="tight")
fig.savefig("./true_vis.pdf", bbox_inches="tight")

# epochs = 10
it= 25001
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("lvis_train",)
cfg.DATASETS.TEST = ("lvis_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")  # Let training initialize from
model zoo
cfg.SOLVER.IMS_PER_BATCH = 8  # This is the real "batch size" commonly known to deep
learning people
cfg.SOLVER.BASE_LR = 1e-5  # pick a good LR
cfg.SOLVER.MAX_ITER = it#int(epochs*len(lvis_train_dict)/cfg.SOLVER.IMS_PER_BATCH)  #
300 iterations seems good enough for this toy dataset; you will need to train longer
for a practical dataset
cfg.SOLVER.GAMMA = 0.1 # The iteration number to decrease learning rate by GAMMA.
cfg.SOLVER.STEPS = (12001,15001,18001)     #  decay learning rate
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512   # The "RoIHead batch size". 128 is
faster, and good enough for this toy dataset (default: 512)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 186  # only has one class (ballon). (see
https://detectron2.readthedocs.io/tutorials/datasets.html#update-the-config-for-new-
datasets)
cfg.SOLVER.CHECKPOINT_PERIOD = 3000
cfg.TEST.EVAL_PERIOD = 1000
cfg.OUTPUT_DIR = './output_train'
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

trainer = MyTrainer(cfg)
# trainer.resume_or_load(resume=False) # For Iteration 0
trainer.resume_or_load(resume=True) #After first checkpoint
trainer.train()

coco_test_dict = detectron2.data.datasets.load_coco_json('./data/coco_test_data.json',
'./data/train2017','coco_test')

DatasetCatalog.register("coco_test",lambda test='test': coco_test_dict)

cfg_baseline = get_cfg()
# add project-specific config (e.g., TensorMask) here if you're not running a model in
detectron2's core library
cfg_baseline.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg_baseline.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model
# Find a model from detectron2's model zoo. You can use the
https://dl.fbaipublicfiles... url as well
```

```
cfg_baseline.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
predictor_baseline = DefaultPredictor(cfg_baseline)

evaluator = COCOEvaluator("coco_test", output_dir="./output_train")
coco_loader = build_detection_test_loader(cfg_baseline, "coco_test")
print(inference_on_dataset(predictor_baseline.model, coco_loader, evaluator))
# another equitestent way to etestuate the model is to use `trainer.test`

x=2
y=3
random.seed(69)
from detectron2.utils.visualizer import ColorMode
dataset_dicts = coco_test_dict
base_list = []
for d in dataset_dicts[66:72]:
    im = cv2.imread(d["file_name"])
    outputs = predictor_baseline(im)  # format is documented at
https://detectron2.readthedocs.io/tutorials/models.html#model-output-format
    v = Visualizer(im[:, :, ::-1],
                   metadata=MetadataCatalog.get('coco_test'),
                   scale=0.5,
                   instance_mode=ColorMode.IMAGE_BW   # remove the colors of
unsegmented pixels. This option is only available for segmentation models
    )
    base_list.append(v.draw_instance_predictions(outputs["instances"].to("cpu")))

fig, axs = plt.subplots(x,y, figsize=(18,10), sharey=False,sharex=False)

for i in range(x):
    for j in range(y):
        axs[i,j].imshow(base_list[j+i*y].get_image(),interpolation='gaussian')
        axs[i,j].set_yticks([])
        axs[i,j].set_xticks([])
# fig.savefig("./io.pdf", bbox_inches="tight")
fig.savefig("./base_vis.pdf", bbox_inches="tight")

# Inference should use the config with parameters that are used in training
# cfg now already contains everything we've set previously. We changed it a little bit
for inference:
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("lvis_train",)
cfg.DATASETS.TEST = ("lvis_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")  # Let training initialize from
model zoo
cfg.SOLVER.IMS_PER_BATCH = 8  # This is the real "batch size" commonly known to deep
learning people
cfg.SOLVER.BASE_LR = 1e-5  # pick a good LR
cfg.SOLVER.MAX_ITER = 25001#int(epochs*len(lvis_train_dict)/cfg.SOLVER.IMS_PER_BATCH)
# 300 iterations seems good enough for this toy dataset; you will need to train longer
for a practical dataset
cfg.SOLVER.GAMMA = 0.1 # The iteration number to decrease learning rate by GAMMA.
cfg.SOLVER.STEPS = (12001,15001,18001)     #  decay learning rate
```

```
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512   # The "RoIHead batch size". 128 is
faster, and good enough for this toy dataset (default: 512)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 186  # only has one class (ballon). (see
https://detectron2.readthedocs.io/tutorials/datasets.html#update-the-config-for-new-
datasets)
cfg.SOLVER.CHECKPOINT_PERIOD = 3000
cfg.TEST.EVAL_PERIOD = 1000
cfg.OUTPUT_DIR = './output_train'
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")  # path to the
model we just trained
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.3   # set a custom testing threshold
predictor = DefaultPredictor(cfg)
from detectron2.utils.visualizer import ColorMode
dataset_dicts = lvis_test_dict
my_list = []
random.seed(69)
for d in dataset_dicts[66:72]:
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)  # format is documented at
https://detectron2.readthedocs.io/tutorials/models.html#model-output-format
    v = Visualizer(im[:, :, ::-1],
                   metadata=MetadataCatalog.get('lvis_test'),
                   scale=0.5,
#                    instance_mode=ColorMode.IMAGE_BW   # remove the colors of
unsegmented pixels. This option is only available for segmentation models
    )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    my_list.append(v.draw_instance_predictions(outputs["instances"].to("cpu")))

fig, axs = plt.subplots(x,y, figsize=(18,10), sharey=False,sharex=False)

for i in range(x):
    for j in range(y):
        axs[i,j].imshow(my_list[j+i*y].get_image(),interpolation='gaussian')
        axs[i,j].set_yticks([])
        axs[i,j].set_xticks([])
fig.savefig("./my_net_vis.pdf", bbox_inches="tight")

# Inference should use the config with parameters that are used in training
# cfg now already contains everything we've set previously. We changed it a little bit
for inference:
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")  # path to the
model we just trained
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.3   # set a custom testing threshold
predictor = DefaultPredictor(cfg)
etestuator = COCOEvaluator("lvis_test", output_dir="./output_train")
test_loader = build_detection_test_loader(cfg, "lvis_test")
print(inference_on_dataset(predictor.model, test_loader, etestuator))
# another equitestent way to etestuate the model is to use `trainer.test`

def load_json_arr(json_path):
    lines = []
    with open(json_path, 'r') as f:
        for line in f:
            lines.append(json.loads(line))
    return lines

metrics= load_json_arr('./output_train/metrics.json')
```

```
my_loss = []
val_loss = []
for d in metrics:
    if('total_loss' in d.keys()):
        my_loss.append(d['total_loss'])
    if('validation_loss' in d.keys()):
        val_loss.append(d['validation_loss'])

#Setup figure
plt.rcParams["font.family"] = "Serif"
# plt.rcParams["text.usetex"] = True
plt.rcParams["font.size"] = 16
plt.plot(np.arange(len(my_loss))*20,my_loss,label='Train')
plt.plot((np.arange(len(val_loss))+1)*1000,val_loss,label='Val')
plt.xlabel('Iteration')
plt.ylabel('Total Loss')
plt.legend()
plt.xlim([0,25000])
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ПРОДУКТІВ ХАРЧУВАННЯ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ»

Виконала: студентка 4 курсу,групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Новіцька О. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

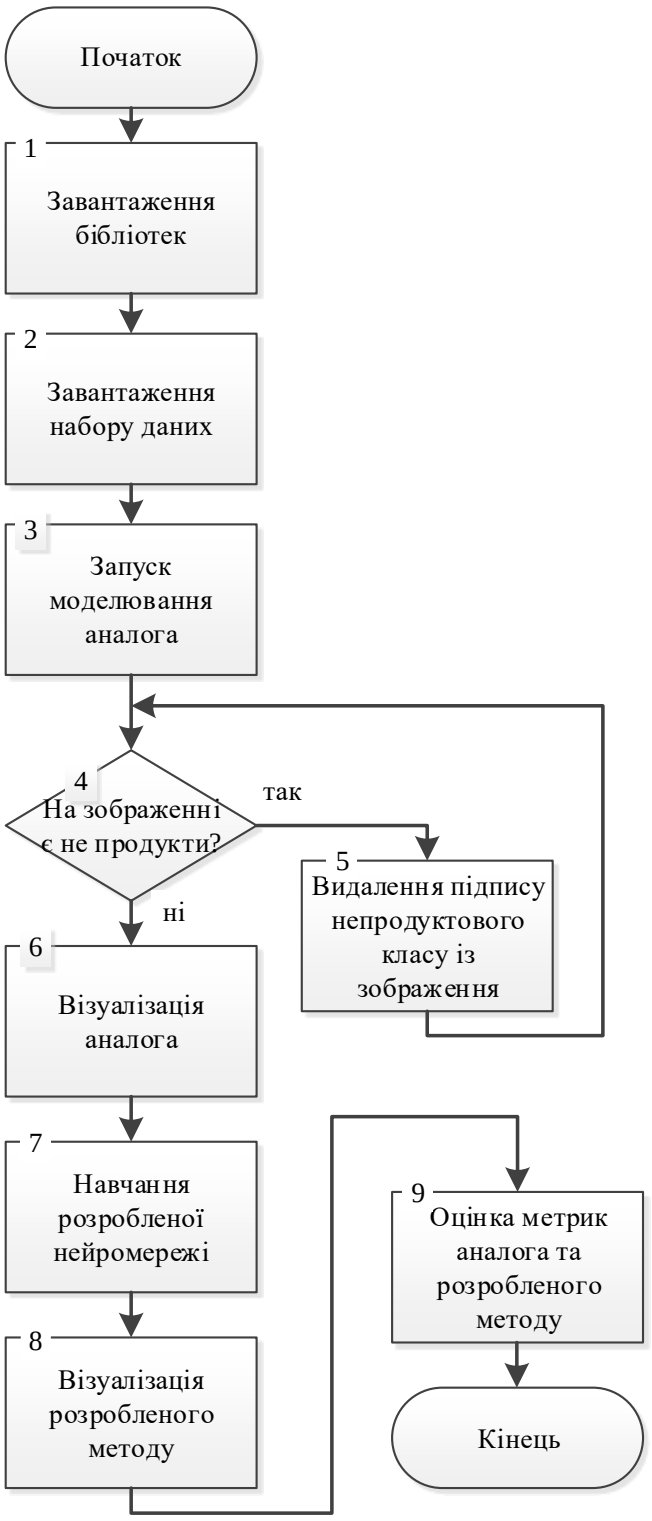

Рисунок В.1 – Схема алгоритму роботи програмного модуля розпізнавання продуктів харчування

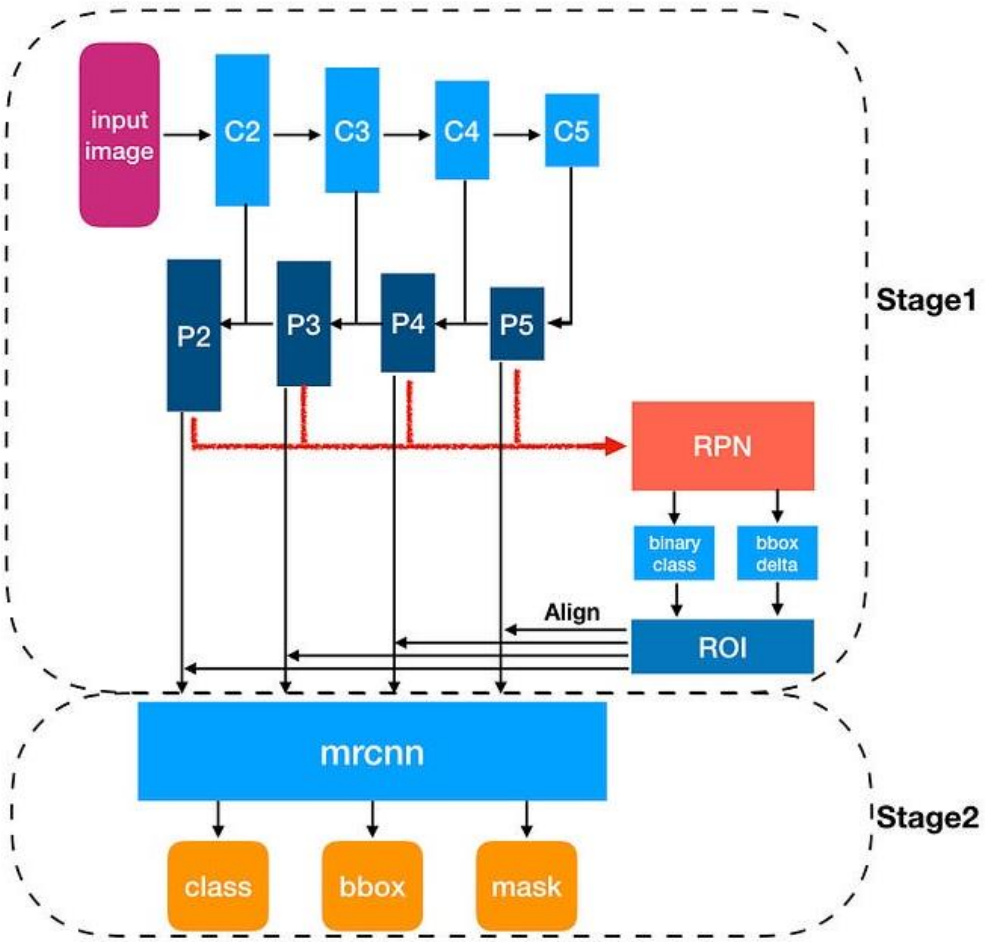

Рисунок В.2 – Логічна структура нейронної мережі Mask RCNN

category	#inst.	category	#inst.	category	#inst.
alcohol	207	almond	1700	apple	17451
applesauce	7	apricot	62	artichoke	293
asparagus	969	avocado	1048	bagel	372
banana	50552	batter_(food)	26	beef_(food)	1242
beer_bottle	1227	beer_can	203	bell_pepper	4369
blackberry	406	blueberry	2114	boiled_egg	125
bread	6550	broccoli	12166	brownie	217
brussels_sp..	590	bubble_gum	4	bun	1780
burrito	14	butter	308	cake	2297
candy_bar	29	candy_cane	107	cantaloup	193
carrot	18049	casserole	12	cauliflower	1035
cayenne_(sp..	49	celery	911	cherry	903
chickpea	265	chili_(vege..	354	crisp_(pota..	541
chocolate_bar	179	chocolate_c..	80	chocolate_m..	7
chocolate_m..	1	cider	38	clementine	108
cocoa_(beve..	4	coconut	273	coleslaw	72
cookie	1920	edible_corn	1883	cornbread	10
cornmeal	1	crab_(animal)	50	crabmeat	5
cracker	510	crape	12	cream_pitcher	10
crescent_roll	152	crouton	140	crumb	3021
cucumber	1533	cupcake	1628	date_(fruit)	103
doughnut	11911	egg	813	egg_roll	15
egg_yolk	90	eggplant	337	escargot	5
fig_(fruit)	147	fish	525	fish_(food)	96
French_toast	41	fruit_juice	37	garlic	487
gelatin	248	ginger	93	gourd	101
grape	6377	green_bean	2571	green_onion	1618
grits	3	ham	1765	hamburger	126
honey	90	hot_sauce	70	hummus	9
icecream	180	popsicle	1	ice_pack	4
jam	29	jelly_bean	116	keg	6
kiwi_fruit	702	lamb-chop	8	lasagna	7
legume	333	lemon	2168	lemonade	2
lettuce	5500	lime	1134	liquor	66
lollipop	59	mandarin_or..	401	martini	3
mashed_potato	58	meatball	174	melon	167
milk	227	milk_can	8	milkshake	1
mint_candy	27	muffin	352	mushroom	6257
nut	790	octopus_(fo..	5	olive_oil	36
omelet	10	onion	9779	orange_(fru..	13034
orange_juice	223	pancake	295	papaya	206
pastry	4972	patty_(food)	20	pea_(food)	1869
peach	1041	peanut_butter	50	pear	1069
pepper	697	pepper_mill	91	persimmon	22
pickle	632	pie	228	pineapple	1636
pita_(bread)	28	pizza	4103	pop_(soda)	951
potato	4393	prawn	779	pretzel	179
prune	8	pudding	2	quesadilla	6
quiche	33	radish	519	raspberry	778
rib_(food)	32	root_beer	3	salad	171
salami	290	salmon_(food)	14	salsa	22
saltshaker	543	sandwich	2315	smoothie	53
soup	193	sour_cream	49	soya_milk	2
crawfish	5	squid_(food)	6	steak_(food)	139
stew	7	strawberry	4386	string_cheese	1
sugar_bowl	10	sugarcane_(..	31	sushi	337
sweet_potato	137	Tabasco_sauce	5	taco	21
tea_bag	42	tequila	2	toast_(food)	125
tomato	12338	truffle_(ch..	4	turkey_(food)	120
turnip	109	vinegar	1	vodka	3
waffle	61	watermelon	814	whipped_cream	201
wine_bottle	4449	yogurt	116	zucchini	798
total	262491				

Рисунок В.3 – Назви класів та кількість прикладів харчових продуктів у наборі даних

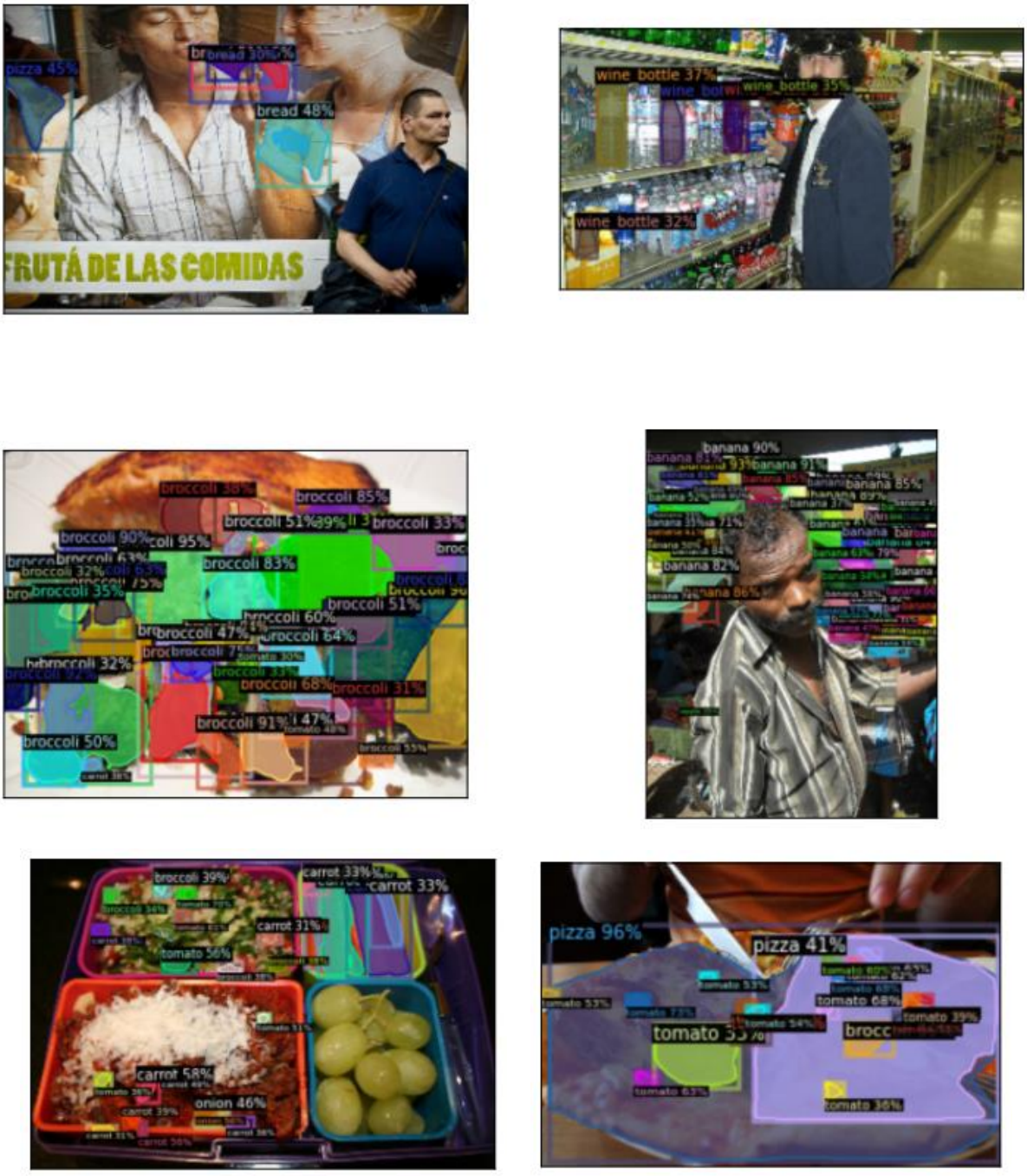

Рисунок В.4 – Скріншоти роботи розробленої програми

Клас	Банан	Яблуко	Сендвіч	Помаранч	Броколі	Морква	Піца	Пончик	Тістечко	Хотдог
Середня влучність сегмента %	37,2	31,4	55,1	31,2	54,8	31,7	27,3	47,2	58,7	19,3
Середня влучність рамки %	33,7	28,2	51,9	30,5	49,6	28,2	24,5	49,3	56,1	23,8

Клас	Листя салату	Гриби	Горіхи	Цибуля	Паста	Картопля	Полуниця	Помідор	Вино
Середня влучність сегмента %	35,4	42,2	38,1	42,3	35,4	32,9	45,1	37,1	70,5
Середня влучність рамки %	33,9	37,7	35,9	45,6	37,3	34,8	39,4	38,5	69,7

Клас	Пиво	Болгарський перець	Чорниця	Панірувальна крихта	Кекс	Виноград	Лимон	По всіх
Середня влучність сегмента %	50,1	34,6	29,9	53,6	44,7	55,3	65,2	**42,55**
Середня влучність рамки %	45,4	29,7	25,1	47,8	38,4	47,2	61,3	**40,13**

Рисунок В.5 – Середні влучності (AP) класів їжі розробленої моделі