Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

**ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ МЕТОДУ ДЕТЕКЦІЇ ОБ'ЄКТІВ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖ**
(тема)


Виконав:
студент 2 курсу, групи ІНФм-23-1
Стрельцов О.А.
(прізвище, ініціали)


Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Стрельцову Олександру Антоновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка методу детекції об'єктів з використанням нейромереж

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, бібліотека глибокого навчання з відкритим кодом Tensorflow, дані інтернет-мережі, мова програмування Python, середовище розробки Jupyter Notebook на платформі Kaggle, власний набір зображень харчових продуктів.

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд існуючих методів розпізнавання зображень та їхніх технічних аспектів.
2. Дослідження архітектур нейронних мереж, які використовуються для розпізнавання зображень харчових продуктів.
3. Обґрунтування вибору програмного забезпечення для реалізації задачі.
4. Реалізація трьох різних архітектур нейронних мереж для розпізнавання зображень.
5. Порівняння точності та продуктивності реалізованих моделей у середовищі розробки та у мобільному застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми, постановка задачі, схема роботи алгоритму розпізнавання харчових продуктів, тестові зображення, результати роботи розпізнавання зображень.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 25.11.2024 | |
| 2 | Аналіз завдання, підбір літератури | 25.11.24-26.11.24 | |
| 3 | Аналіз літератури з досліджуваної проблеми | 26.11.24-27.11.24 | |
| 4 | Аналіз програмних засобів та інструментів | 27.11.24-30.11.24 | |
| 5 | Розробка методу розпізнавання зображень | 30.11.24-02.12.24 | |
| 6 | Програмна реалізація | 02.12.24-11.12.24 | |
| 7 | Оформлення пояснювальної записки | 11.12.24-18.12.24 | |
| 8 | Перевірка на плагіат | 19.12.2024 | |
| 9 | Рецензування | 21.12.2024 | |
| 10 | Підготовка презентації та доповіді | 23.12.2024 | |
| 11 | Занесення роботи в електронний архів | 01.01.2025 | |
| 12 | Попередній захист кваліфікаційної роботи | 02.01.2025 | |

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Тітова О.В.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 79 с., 4 табл., 40 рис., 50 джерел.

КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, НЕЙРОННІ МЕРЕЖІ, КВАНТОВАНІ МОДЕЛІ, ТОЧНІСТЬ РОЗПІЗНАВАННЯ, ШВИДКІСТЬ ІНФЕРЕНСУ, ОПТИМІЗАЦІЯ МОДЕЛЕЙ.

Об'єктом дослідження є процес розпізнавання харчових продуктів у режимі реального часу з використанням нейронних мереж.

Метою дослідження є аналіз та оптимізація процесу розпізнавання харчових продуктів із застосуванням нейронних мереж, розробка та впровадження різних моделей для цього завдання, а також порівняння їхньої ефективності за допомогою власного набору даних.

В рамках дослідження розглядалися архітектури SSD-MobileNetV2-320, SSD-MobileNetV2-FPNLite320, EfficientDet-D0 у форматах SavedModel, TFLite і квантовані версії. Ефективність моделей оцінювали за метрикою mAP, кількістю правильно ідентифікованих об'єктів, часом обробки за офіційними бенчмарками TensorFlow, виконанням у Python-скриптах і мобільному застосунку для визначення їхньої придатності.

У результаті дослідження було розроблено метод розпізнавання харчових продуктів, створено датасет та мобільний застосунок.

COMPUTER VISION, IMAGE RECOGNITION, NEURAL NETWORKS, QUANTIZED MODELS, RECOGNITION ACCURACY, INFERENCE RATE, MODEL OPTIMIZATION.

The object of research is the process of real-time food recognition using neural networks.

The aim of the research is to analyze and optimize the process of food recognition using neural networks, develop and implement various models for this task, and compare their effectiveness using our own data set.

The research considered the SSD-MobileNetV2-320, SSD-MobileNetV2-FPNLite320, EfficientDet-D0 architectures in SavedModel, TFLite, and quantized versions. The models' performance was evaluated by the mAP metric, the number of correctly identified objects, processing time according to official TensorFlow benchmarks, and execution in Python scripts and a mobile application to determine their suitability.

The research resulted in the development of a food recognition method, a dataset, and a mobile application.

# ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів ..... 7
Вступ ..... 8
1 Аналіз літератури і сучасних методів розпізнавання зображень ..... 9
1.1 Актуальність та технічні аспекти застосування нейронних мереж для розпізнавання зображень ..... 9
1.2 Основні класичні методи розпізнавання зображень ..... 11
1.2.1 Метод Scale-Invariant Feature Transform ..... 11
1.2.2 Метод Speeded-Up Robust Features ..... 15
1.3 Основні сучасні методи розпізнавання зображень ..... 16
1.3.1 Метод Faster R-CNN (Regions with CNN) ..... 16
1.3.2 Метод SSD (Single Shot Multibox Detector) ..... 18
1.3.3 Метод EfficientDet ..... 20
1.4 Огляд літератури щодо методів та підходів для розпізнавання харчових продуктів ..... 21
1.5 Постановка задачі дослідження ..... 23
2 Застосування нейромереж для детекції об’єктів ..... 25
2.1 Архітектура та навчання SSD для розпізнавання об’єктів ..... 25
2.2 Компоненти та процес навчання EfficientDet ..... 28
2.3 Трансферне навчання ..... 32
2.4 Три популярні архітектури для розпізнавання зображень ..... 35
2.4.1 Архітектура SSD-MobileNetV2-320 ..... 35
2.4.2 Архітектура EfficientDet-D0 ..... 36
2.4.3 Архітектура SSD-MobileNetV2-FPNLite320 ..... 38
2.5 Опис наступних етапів та оцінка ефективності розпізнавання об’єктів ..... 39
3 Розробка та реалізація методу детекції об’єктів з використанням нейромереж ..... 45
3.1 Вибір програмного забезпечення ..... 45
3.2 Створення датасету харчових продуктів ..... 47
3.3 Програмна реалізація розпізнавання харчових продуктів ..... 51

3.4 Аналіз точності та продуктивності розпізнавання харчових продуктів ..... 57
Висновки ..... 72
Перелік джерел посилання ..... 74

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AP – Average Precision (середня точність)

BiFPN – Bidirectional Feature Pyramid Network (двонаправлена мережа піраміди ознак)

CNN – Convolutional Neural Network (згорткова нейронна мережа)

CRISP-DM – Cross-Industry Standard Process for Data Mining (стандартний процес для аналізу даних у різних галузях)

FPN – Feature Pyramid Network (мережа піраміди ознак)

GPU – Graphics Processing Unit (графічні процесори)

IoU – Intersection over Union (перетин на об’єднання)

mAP – mean Average Precision (середня точність)

NMS – Non-Maximum Suppression (пригнічення немаксимумів)

PR – Precision-Recall (точність та відзив)

R-CNN – Regions with CNN (регіони із згортковою нейронною мережею)

RGB-D – Red Green Blue and Depth (глибина та кольори RGB)

RPN – Region Proposal Network (мережа пропозицій регіонів)

SGD – Stochastic Gradient Descent (стохастичний градієнтний спуск)

SSD – Single Shot Multibox Detector (одноразовий багатоблоковий детектор)

SIFT – Scale-Invariant Feature Transform (масштабно-незалежне перетворення ознак)

TPU – Tensor Processing Unit (обчислювальні пристрої)

# ВСТУП

Протягом останніх десятиліть стрімкий прогрес у сфері технологій і інформаційних систем суттєво вплинув на щоденне існування людей, забезпечуючи можливості для автоматизації та спрощення багатьох рутинних процесів. Однією з найпопулярніших галузей, що отримала потужний поштовх завдяки цьому прогресу, є обробка зображень і комп'ютерний зір [1]. Ці інновації відкривають нові горизонти для розвитку в таких галузях, як спорт, медицина, освіта та харчова промисловість.

Різні методи машинного навчання використовуються для розпізнавання харчових продуктів. Зокрема, ефективними для аналізу зображень є згорткові нейронні мережі (CNN), оскільки вони здатні виявляти патерни на візуальних даних [2–4]. Ці моделі навчаються на значних масивах зображень їжі, що дозволяє їм точно розпізнавати продукти на нових кадрах.

Головна складність використання нейронних мереж полягає в тому, що їх ефективність залежить від багатьох факторів: обсягу та якості даних, специфіки завдання, вибору гіперпараметрів, структури мережі та методів оптимізації [5]. Навіть найсучасніші моделі можуть показувати низьку продуктивність без належного налаштування під конкретні умови. Тому дослідження нейронних мереж є важливим і спрямоване на пошук оптимальних рішень для різних задач, удосконалення процесу навчання та розробку методів, які підвищують ефективність і роблять моделі більш адаптивними до змін у даних та умовах їх використання.

Очікується, що результати дослідження підвищать точність розпізнавання харчових продуктів, виявлять переваги та недоліки архітектур, а також стануть корисними для реальних застосувань, зокрема для інтеграції в мобільні застосунки для розпізнавання харчових продуктів.

# 1 АНАЛІЗ ЛІТЕРАТУРИ І СУЧАСНИХ МЕТОДІВ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

## 1.1 Актуальність та технічні аспекти застосування нейронних мереж для розпізнавання зображень

Розпізнавання зображень – одна з ключових технологій, яка активно розвивається завдяки значним досягненням у галузі штучного інтелекту і комп’ютерного зору. Актуальність обумовлена зростаючою потребою в автоматизації обробки візуальної інформації, що стала невід’ємною частиною багатьох галузей сучасного життя. З кожним роком збільшується кількість цифрових зображень, відеоматеріалів та інших візуальних даних, що потребують ефективного аналізу [6]. Завдяки технологіям розпізнавання зображень можна автоматизувати рутинні процеси, знизити кількість помилок, покращити продуктивність, а також забезпечити високу точність у складних завданнях.

Однією з основних причин актуальності цієї технології є стрімкий розвиток обчислювальних потужностей. Завдяки появі потужних графічних процесорів (GPU) і спеціалізованих обчислювальних пристроїв TPU, стало можливим ефективно обробляти значні обсяги даних у реальному часі. Це забезпечує швидкий та точний аналіз зображень, що є необхідним у таких галузях, як безпека, медицина, автомобільна індустрія, роздрібна торгівля, агропромисловість та інші.

Однією з важливих сфер застосування технологій розпізнавання зображень є розпізнавання харчових продуктів (рис. 1.1). У ресторанній сфері, супермаркетах та застосунках для здорового харчування [7]. Ці технології дозволяють автоматично ідентифікувати продукти на основі зображень, аналізувати їх харчову цінність, а також автоматизувати процеси обслуговування клієнтів.

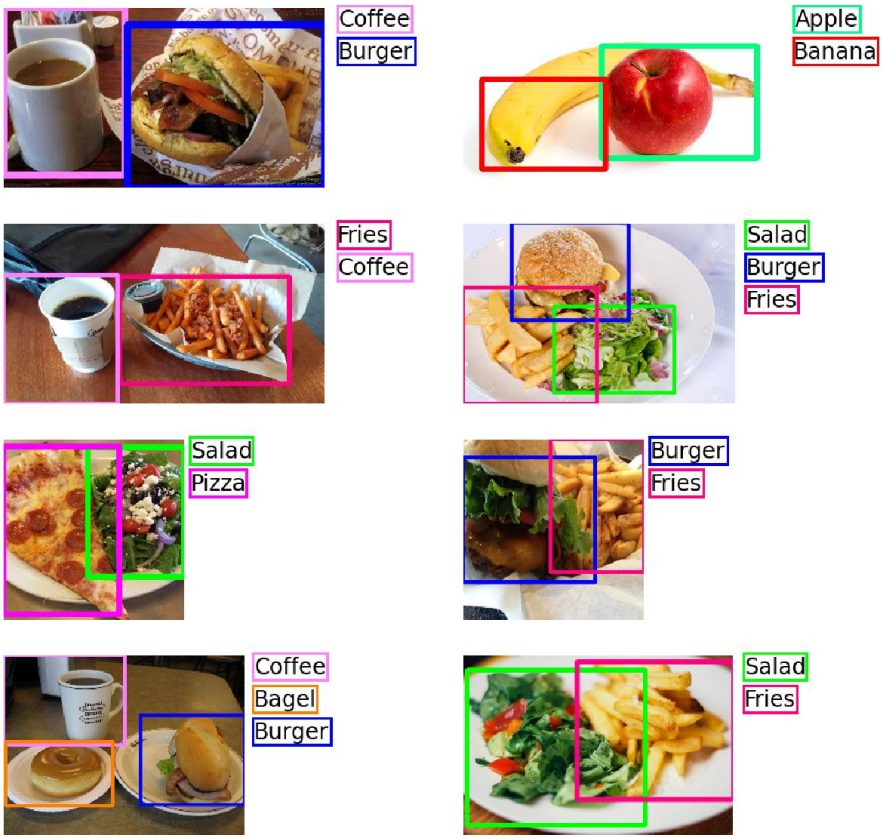


Рисунок 1.1 – Приклад розпізнавання харчових продуктів [8]

Для вирішення завдань розпізнавання харчових продуктів можуть використовуватися різні архітектури нейронних мереж. Наприклад, архітектура SSD-MobileNetV2-320 дозволяє ефективно виконувати детекцію об’єктів у реальному часі, що є важливим для автоматизованих кас у супермаркетах або безконтактних систем оплати, де продукти мають бути швидко і точно розпізнані [9]. Ця модель поєднує в собі швидкість та відносно невеликі вимоги до обчислювальних ресурсів, що робить її ідеальним вибором для мобільних та вбудованих систем.

Архітектура EfficientDet-D0 є ще однією популярною моделлю для задач детекції об’єктів для розпізнавання продуктів харчування [10]. Вона відома своїм ефективним співвідношенням між точністю та швидкістю, що дозволяє застосовувати її в різних системах, де потрібна висока продуктивність при мінімальних витратах ресурсів.

Ще одна потужна архітектура для задач детекції об’єктів – це SSD-MobileNetV2-FPNLite320. Вона демонструє відмінний баланс між швидкістю

інференсу та точністю, що є особливо корисним у високонавантажених системах, таких як обробка великого потоку зображень у реальному часі [11]. У сфері харчової індустрії ця модель може бути використана для автоматизованого контролю якості продуктів, сортування товарів на виробничих лініях або в касових системах.

Головна складність використання нейронних мереж полягає в тому, що їх ефективність залежить від багатьох факторів: обсягу та якості даних, специфіки завдання, вибору гіперпараметрів, структури мережі та методів оптимізації. Навіть найсучасніші моделі можуть показувати низьку продуктивність без належного налаштування під конкретні умови. Тому дослідження нейронних мереж є важливим і спрямоване на пошук оптимальних рішень для різних задач, удосконалення процесу навчання та розробку методів, які підвищують ефективність і роблять моделі більш адаптивними до змін у даних та умовах їх використання.

## 1.2 Основні класичні методи розпізнавання зображень

### 1.2.1 Метод Scale-Invariant Feature Transform

Scale-Invariant Feature Transform (SIFT) – це один із найбільш відомих та широко використовуваних алгоритмів для виділення характеристик на зображеннях [12], який був запропонований Девідом Лоу в 1999 році. Основна перевага цього методу полягає в його здатності залишатися стійким до змін повороту, масштабу, освітлення та якихось незначних змін перспективи. SIFT шукає ключові точки на зображеннях, які залишаються стабільними в умовах цих змін. Такий підхід робить цей метод незамінним для задач злиття зображень, пошуку об’єктів або розпізнавання.

Алгоритм методу SIFT складається з декількох етапів [13]. Блок-схема роботи методу SIFT представлено на рисунку 1.2. На першому етапі відбувається пошук особливих точок, які не залежать від масштабу. Для цього

зображення перетворюється на масштабно-просторове представлення за допомогою гауссового розмиття. Такий підхід дозволяє побудувати піраміду зображень на різних рівнях масштабування. Потім обчислюються різниці між сусідніми рівнями цієї піраміди, що утворює зображення різниці гауссіан. Локальні максимуми та мінімуми, знайдені на цих зображеннях, використовуються для визначення ключових точок.

На другому етапі відбувається точне визначення місцезнаходження ключових точок. Потенційні ключові точки перевіряються і ті точки, що мають слабкий контраст або вони розташовані на краях зображення, виключаються, бо вони можуть бути ненадійними для подальшого аналізу. Щоб точно визначити позиції цих точок, використовується процес інтерполяції в масштабно-просторовій області.

На третьому етапі для кожної ключової точки визначається її орієнтація. Для цього аналізуються напрямки градієнтів у її найближчому оточенні. Після цього кожній точці присвоюється одна або кілька орієнтацій. Це дозволяє алгоритму SIFT бути стійким до повороту зображення, бо всі наступні обчислення виконуються з урахуванням цієї орієнтації.

На четвертому етапі відбувається побудова дескрипторів ключових точок. Для кожної точки формується вектор дескриптора, який містить інформацію про напрямки градієнтів у найближчому оточенні ключової точки. Цей вектор залишається незмінним при змінні масштабу та повороті зображення, що дозволяє порівнювати ключові точки між різними зображеннями. Зазвичай довжина такого дескриптора становить 128 елементів.

На п’ятому етапі, після того як були обчислені дескриптори ключових точок, відбувається їхнє порівняння між зображеннями, яке аналізується та еталонним зображенням. Для цього використовуються дескриптори, які допомагають знайти відповідні точки на цих зображеннях. Після того як збіги між ключовими точками встановлені, виконується афінне перетворення зображень. Це дозволяє більш точно поєднати еталонне зображення з

зображенням, що аналізується, за допомогою побудови афінної матриці на основі сусідніх ключових точок.

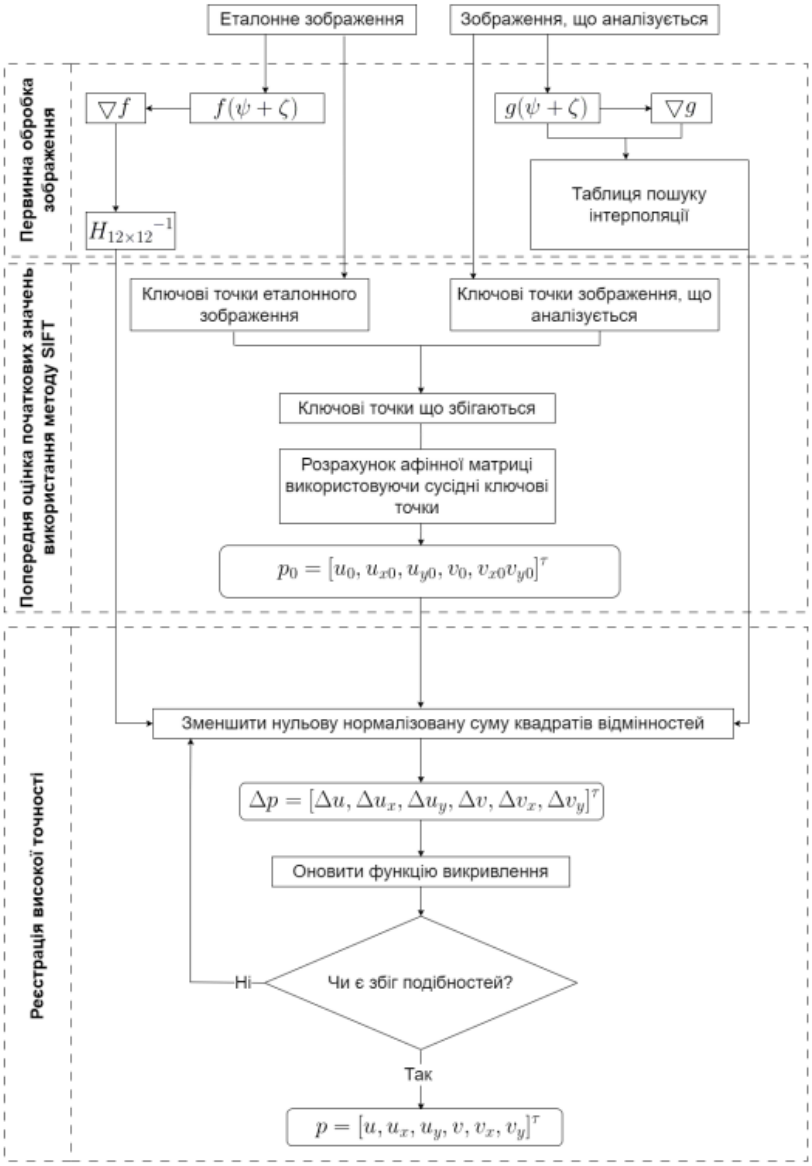


Рисунок 1.2 – Блок-схема роботи методу SIFT [13]

На шостому етапі виконується зменшення різниці між зображеннями шляхом мінімізації нормалізованої суми квадратів відмінностей. Після цього функція коригує викривлення зображення. Цей процес повторюється, поки не буде досягнуто точного співпадіння зображень або поки не буде виконано умови завершення.

Приклад зображення яблука з візуалізованими етапами алгоритму SIFT представлено на рисунку 1.3. Це демонструє всі основні кроки, такі як побудова піраміди зображень, визначення ключових точок, аналіз напрямків градієнтів, побудова дескрипторів та порівняння зображень за допомогою афінного перетворення [14].

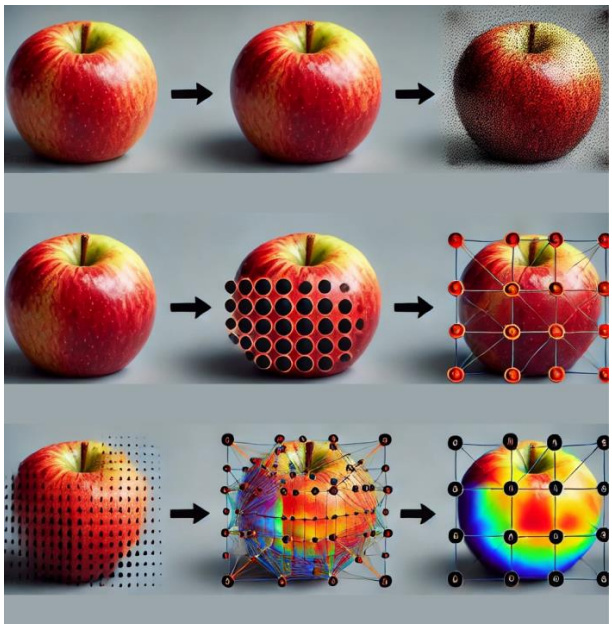

Рисунок 1.3 – Приклад зображення яблука з візуалізованими етапами алгоритму SIFT [14]

Отже, метод SIFT є ефективним інструментом для розпізнавання та порівняння харчових продуктів, забезпечуючи високу точність завдяки своїй стійкості до змін повороту, масштабу, наявності шуму та освітлення. Цей алгоритм добре підходить для вирішення завдань, в яких потрібно точно виявити ключові точки та порівнювати зображення в складних умовах.

Проте, через високу обчислювальну складність та великі вимоги до ресурсів, SIFT не завжди підходить для роботи в реальному часі або на пристроях з обмеженими можливостями. Крім того, багато дескрипторів можуть спричинити проблеми в системах з обмеженою пам’яттю. Також патентні обмеження можуть ускладнити використання цього методу, що й стимулювало розвиток альтернативних методів.

### 1.2.2 Метод Speeded-Up Robust Features

Метод Speeded-Up Robust Features (SURF) є одним із швидких методів для виявлення ключових точок та витягування ознак із зображень. Він був розроблений як альтернатива алгоритму SIFT, щоб пришвидшити процес порівняння зображень, зберігаючи при цьому хорошу точність. Цей алгоритм може бути придатним для використання в задачах, де необхідно порівнювати зображення в реальному часі.

Основна ідея SURF полягає у тому (рис. 1.4), щоб знайти ключові точки на зображенні, які залишаються стабільними при зміні масштабу, обертання або освітлення. Для цього алгоритм використовує спеціальні математичні операції, такі як другі похідні (гессіани). Тобто, SURF шукає екстремальні значення у цих гессіанах, що вказують на ключові області зображення, які є важливими для подальшого порівняння [15].

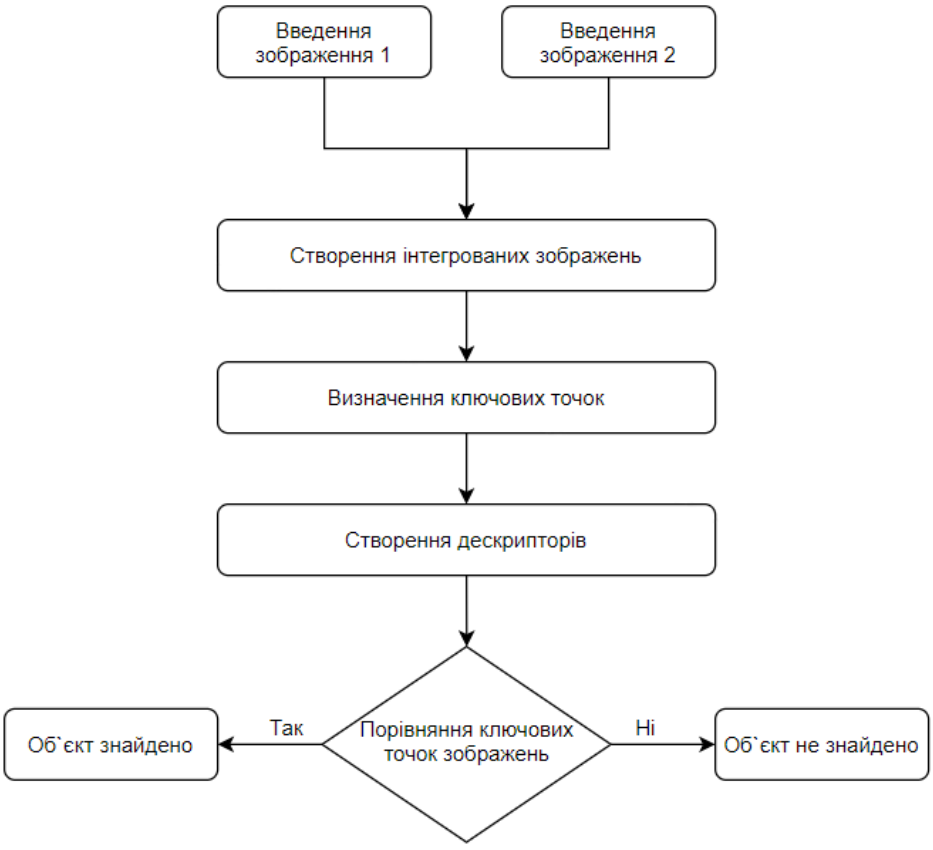


Рисунок 1.4 – Схема роботи методу SURF [15]

Також SURF використовує фільтри для того, щоб якомога швидко визначати напрямки змін пікселів навколо ключової точки. Це дозволить алгоритму ефективно визначати ознаки зображень та робити їх інваріантними до змін масштабу, освітлення або обертання [15]. Завдяки цьому метод SURF здатний працювати швидше за SIFT та є корисним для систем, де важлива швидкодія.

Отже, метод SURF є потужним інструментом для розпізнавання та порівняння зображень завдяки високої швидкості і стійкості до змін масштабу та орієнтації. Він особливо підійде для систем, де важлива швидка обробка даних, таких як аналіз відео або вбудовані системи комп'ютерного зору. Однак, попри переваги в швидкості, точність SURF може бути нижчою, ніж у SIFT, особливо коли важливі дрібні деталі або зображення спотворене через зміну ракурсу. Також потрібно враховувати, що метод SURF має ліцензійні обмеження, як і метод SIFT.

## 1.3 Основні сучасні методи розпізнавання зображень

### 1.3.1 Метод Faster R-CNN (Regions with CNN)

Метод Faster R-CNN є подальшим розвитком методів R-CNN і Fast R-CNN, усуваючи проблеми повільної роботи та низької ефективності попередніх версій. Він поєднує використання згорткових нейронних мереж для витягування ознак зображення та спеціальних мереж для визначення потенційних регіонів, де можуть бути об'єкти. Цей підхід вирішує два ключових завдання: знаходження об'єктів на зображенні та їх точну класифікацію.

Архітектура Faster R-CNN складається з кількох основних складових (рис. 1.5). Першою з них є згорткова нейронна мережа, яка відповідає за витяг ознак з зображення. Для цього може бути застосована будь-яка архітектура CNN, наприклад, ResNet чи VGG16. Після обробки зображення мережею

створюється карта ознак, яка містить інформацію про ключові елементи зображення [16, 17].

Основною інновацією методу Faster R-CNN є використання мережі пропозицій регіонів (RPN) спеціальної моделі, яка визначає області, де можуть знаходитись об’єкти [18]. RPN накладає на карту ознак набір попередньо визначених рамок (якорів) різних розмірів та пропорцій. Для кожного з цих якорів мережа оцінює ймовірність того, що в ньому є об’єкт та уточнює його межі. Завдяки цьому підходу, Faster R-CNN набагато швидше генерує пропозиції регіонів у порівнянні з попередніми методами, де для цієї задачі використовувались сторонні алгоритми.

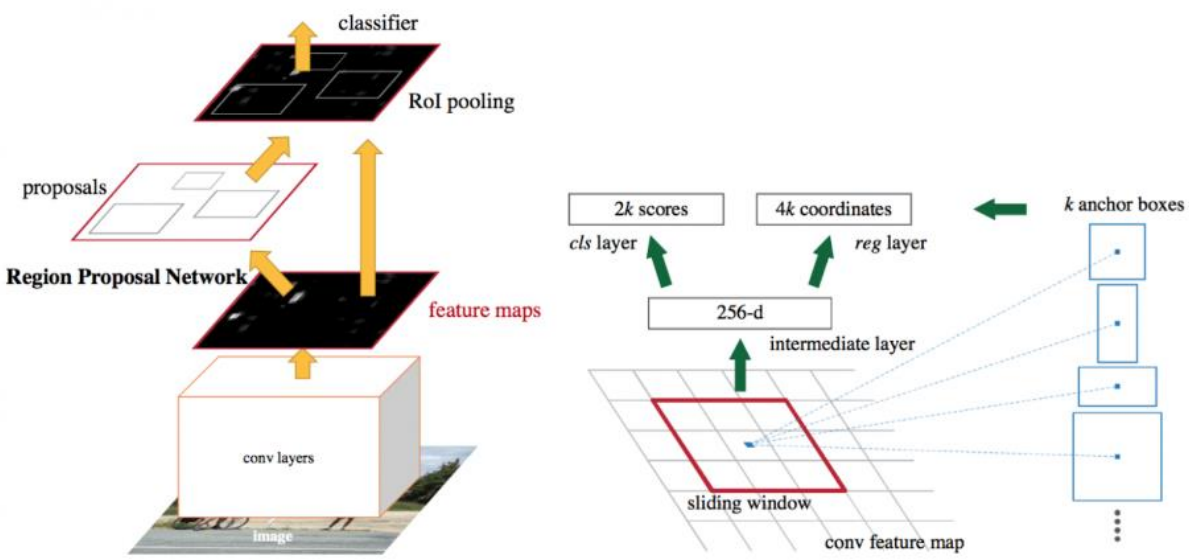


Рисунок 1.5 – Архітектура методу Faster R-CNN [19]

Після того як RPN визначає регіони інтересу, вони обробляються через RoI Pooling, який приводить їх до стандартного розміру. Це необхідно для подальшої зручної обробки регіонів на наступних етапах моделі. Потім для кожного регіону виконується класифікація об’єкта та уточнення його меж за допомогою окремого класифікатора [19].

Отже, основними перевагами Faster R-CNN є висока точність та покращена швидкість завдяки інтеграції мережі пропозицій регіонів. Оскільки

ця мережа навчається разом із CNN, вона здатна більш точно визначати області, що містять об’єкти. Крім того, метод Faster R-CNN є гнучким, оскільки дозволяє використовувати різні архітектури CNN для вилучення ознак, що робить його придатним для широкого спектра застосувань [20].

Однак, метод Faster R-CNN має певні недоліки. Він все ще залишається вимогливим до обчислювальних ресурсів та не підходить для завдань реального часу через його повільну роботу. Для таких завдань зазвичай обирають більш швидкі методи, такі як SSD або YOLO, хоча вони можуть бути менш точними.

### 1.3.2 Метод SSD (Single Shot Multibox Detector)

Метод SSD (Single Shot Multibox Detector) є одним із провідних сучасних методів розпізнавання об’єктів на зображеннях, що вирізняється своєю високою швидкістю та ефективністю. На відміну від таких методів, як R-CNN чи Faster R-CNN, SSD не вимагає окремого етапу визначення регіонів інтересу, що робить його процес виявлення об’єктів простішим та швидшим. Основна ідея SSD полягає в тому, що він одночасно виявляє об’єкти різних розмірів на кількох рівнях зображення під час одного проходу, що дозволяє використовувати його для задач в реальному часі [21].

Архітектура SSD базується на згортковій нейронній мережі для вилучення ознак із зображення, подібно до інших методів. Однак, після того як карта ознак побудована, SSD паралельно виконує розпізнавання об’єктів на різних рівнях цієї карти, використовуючи різні розміри якорів. Це дозволяє моделі виявляти об’єкти різного масштабу на різних частинах зображення (рис. 1.6). Такий підхід робить SSD ефективною при обробці сцен, де присутні об’єкти різного розміру.

SSD використовує підхід, при якому координати рамок та розпізнавання об’єктів здійснюються на кількох різних рівнях карти ознак. Кожен із цих

рівнів відповідає за виявлення об’єктів певного розміру. Завдяки цьому підходу можна уникнути потреби у попередньому етапі генерування регіонів, що значно прискорює процес. Наприклад, замість того, щоб виконувати класифікацію після генерації пропозицій регіонів, як у Faster R-CNN, SSD одночасно передбачає координати рамок та клас об’єкта на кожному рівні [21].

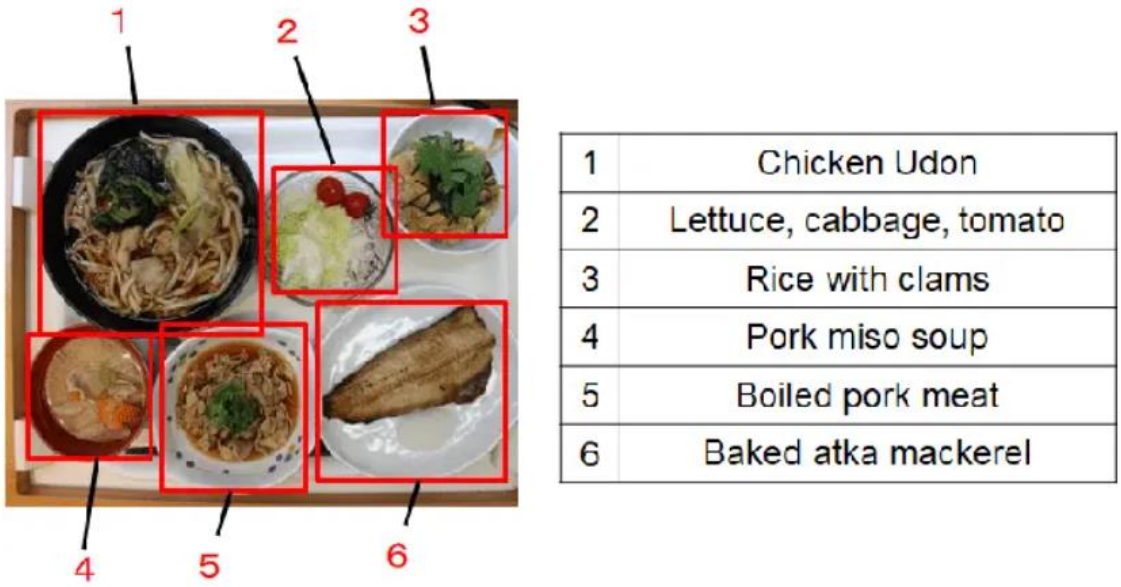


Рисунок 1.6 – Приклад використання методу SSD [21]

Отже, основними перевагами методу SSD є його висока швидкість та здатність працювати в режимі реального часу, завдяки чому він є придатним для застосувань, де важливий час обробки, як-от системи відеоспостереження, автономні транспортні засоби та мобільні застосунки. Завдяки використанню кількох рівнів ознак для детектування, SSD забезпечує досить високу точність при розпізнаванні об’єктів різних розмірів.

Однак, метод SSD має недоліки. Наприклад, його точність може бути нижчою, ніж у метода Faster R-CNN, особливо при виявленні дрібних об’єктів. Це пов’язано з тим, що на початкових рівнях ознак (на ранніх шарах CNN) модель може втрачати важливі деталі, які необхідні для точного окреслення меж маленьких об’єктів.

### 1.3.3 Метод EfficientDet

Метод EfficientDet є одним із сучасних підходів для детектування об’єктів, створеним на основі архітектури EfficientNet, яка була спеціально модифікована для цієї задачі. Головна мета цього методу полягала в досягненні високої точності розпізнавання за умови використання мінімуму обчислювальних ресурсів. EfficientDet відзначається тим, що поєднує в собі швидкість та точність завдяки унікальному підходу до масштабування моделі, що дозволяє оптимізувати її продуктивність для різних задач [22].

Масштабування в EfficientDet відбувається відразу в трьох напрямках: за шириною, глибиною та роздільною здатністю вхідних даних. Це дає можливість налаштовувати під різні вимоги, де важлива точність, швидкість та обмежені обчислювальні ресурси.

Однією з ключових переваг EfficientDet є застосування BiFPN, яка дозволяє об’єднувати ефективно інформацію з різних рівнів. Завдяки цьому мережа може краще розпізнавати об’єкти різного розміру та масштабу. Також метод EfficientDet відрізняється раціональним використанням ресурсів завдяки тому, що його масштабування одночасно збільшує розмір вхідних даних, глибину та кількість каналів, що дозволяє налаштувати архітектуру під конкретні потреби задачі [22].

Основним недоліком методу EfficientDet є його складність у налаштуванні. Цей метод забезпечує хороші результати, однак підбір оптимальної архітектури та налаштування параметрів для конкретних задач може вимагати значних зусиль та часу. Крім того, EfficientDet потребує потужного обладнання на етапі навчання. Хоча модель оптимізована для використання на мобільних пристроях, але сам процес тренування є доволі ресурсомістким.

Таким чином, метод EfficientDet подібний до методу SSD та може бути використаний для задач в реальному часі, що робить його придатним для мобільних пристроїв та інших систем, де обмежені ресурси.

## 1.4 Огляд літератури щодо методів та підходів для розпізнавання харчових продуктів

Огляд літератури спрямований на дослідження існуючих методів та підходів до розпізнавання харчових продуктів за допомогою CNN та інших сучасних алгоритмів комп’ютерного зору.

У джерелі [23] було досліджено алгоритм YOLO для вирішення проблеми автоматизації процесів у харчовій промисловості Японії, де є значний дефіцит робочої сили через демографічні зміни. Основною проблемою є необхідність налаштування роботів для кожного варіанту продукту. Автори пропонують рішення, що включає використання YOLO для спрощення програмування роботів. Удосконаленням є інтеграція методу Non-Maximum Suppression (NMS) для кращого розпізнавання перекриваючих об’єктів, використання даних з RGB-D сенсорів для точного вимірювання висоти та гравітаційного центру об’єктів. Тести з роботами підтвердили ефективність цього підходу для розташування реальних об’єктів, що свідчить про його потенціал для оптимізації виробничих процесів.

У джерелі [24] було проведено дослідження автоматичного визначення епізодів прийому їжі з використанням сенсора Automatic Ingestion Monitor v2 . Основна задача полягала у зменшенні кількості хибних позитивних спрацювань під час виявлення таких епізодів. Для цього застосували три підходи. Перший метод ґрунтувався на розпізнаванні їжі та напоїв на зображеннях, зроблених за допомогою сенсора Automatic Ingestion Monitor v2. Другий метод полягав в аналізі процесу жування за допомогою акселерометра цього сенсора, а третій – в ієрархічній класифікації, яка поєднувала результати перших двох методів. Завдяки поєднанню підходів вдалося досягти точності 94%, прецизійності 70% і F1-мітки 80%, що значно перевершує ефективність кожного методу окремо.

У джерелі [25] було проведено дослідження моделі глибокого навчання для ідентифікації трьох ароматичних рослин, таких як розмарин, м’ята та

лавровий лист за допомогою алгоритму YOLO. Дослідження базувалося на методології CRISP-DM, що включає етапи аналізу, підготовки даних, створення моделі, її оцінки та впровадження. Для навчання використовувалися зображення, які були отримані з різних пристроїв, на яких рослини були позначені обмежувальними рамками. Оцінка моделі проводилася за допомогою метрики середньої точності, що враховує точність та повноту. Модель досягла точності приблизно 70% для кожної рослини, проте високі показники recall свідчать про можливі випадки надмірного виявлення, що вказує на необхідність збільшення бази даних і вдосконалення методології. Це дослідження підкреслює потенціал використання глибокого навчання для ідентифікації рослин, що сприяє підвищенню ефективності та точності в аграрній сфері.

У джерелі [26] було представлено новий метод сегментації зображень їжі, який заснований на глибокому навчанні. Цей метод був розроблений для автоматичної обробки даних з унікальних наборів зображень. Використовуючи згорткові нейронні мережі з елементами Transfer Learning та механізмами уваги, модель продемонструвала високу точність сегментації. Експерименти підтвердили ефективність та стабільність цього підходу, що дозволяє точно аналізувати споживання їжі. Це рішення було інтегровано в користувацький інтерфейс для надання миттєвого аналізу харчових продуктів. Крім того, дослідження пропонує алгоритм розпізнавання їжі на основі CNN, який досяг середньої точності від 40% до 89% залежно від типу їжі.

У джерелі [27] було розглянуто застосування машинного зору та технологій обробки зображень у процесах харчової промисловості. Основна увага приділялася використанню класичних методів машинного навчання та глибокого навчання для ідентифікації типу і якості продуктів харчування. Представлено огляд поточних методів, таких як оцінка якості їжі, виявлення дефектів або сторонніх об’єктів та видалення домішок. Також проаналізовано проблеми, які можуть виникнути з цими підходами та наведено можливі

напрями подальшого розвитку технологій у сфері автоматизації харчових процесів.

## 1.5 Постановка задачі дослідження

Огляд літератури показав, що розпізнавання харчових продуктів у реальному часі є актуальною темою в галузі комп’ютерного зору, з низкою підходів, які зосереджені на підвищенні точності й ефективності моделей.

Проте є кілька обмежень у сучасних дослідженнях. Багато досліджень не приділяють достатньо уваги специфіці реальних мобільних пристроїв, що може впливати на швидкість роботи моделей у режимі реального часу [28]. Крім того, більшість існуючих рішень тестуються на стандартних наборах даних, що може зменшити їхню узагальнювальність при роботі з більш складними, нестандартними зображеннями, характерними для практичних умов. Окремо відзначається нестача порівняльного аналізу різних архітектур у контексті конкретних завдань розпізнавання їжі, що ускладнює вибір оптимальної моделі для використання на мобільних застосунках або в спеціалізованих застосунках. Тому виникає необхідність дослідження підходу розпізнавання харчових продуктів, який враховує специфіку мобільних пристроїв, зокрема оптимізацію моделей для забезпечення швидкої обробки зображень без значного зниження точності.

Об’єктом дослідження є процес розпізнавання харчових продуктів у режимі реального часу з використанням нейронних мереж.

Метою дослідження є аналіз та оптимізація процесу розпізнавання харчових продуктів із застосуванням нейронних мереж, розробка та впровадження різних моделей для цього завдання, а також порівняння їхньої ефективності за допомогою власного набору даних.

Щоб досягти зазначеної мети дослідження, слід вирішити такі завдання:

– провести аналіз наукової літератури та існуючих підходів до розпізнавання об’єктів у реальному часі, визначити ключові проблеми та обмеження;

– розглянути актуальні архітектури нейронних мереж для розпізнавання харчових продуктів, таких як SSD-MobileNetV2-320, EfficientDet-D0, SSD-MobileNetV2-FPNLite320;

– дослідити ключові аспекти процесу розпізнавання продуктів із використанням згорткових нейронних мереж та розглянути можливість використання Transfer Learning з метою покращення результатів навчання моделей;

– підготувати власний набір даних, який буде включати різні категорії харчових продуктів, які максимально наближені до умов реального використанню;

– налаштувати та провести тренування обраних архітектур нейронних мереж;

– виконати квантування та оптимізацію моделей;

– оцінити ефективність моделей за допомогою метрики mAP, кількості правильно ідентифікованих об’єктів, часу обробки моделей на основі офіційних бенчмарків TensorFlow, часу виконання моделей у середовищі Python-скриптів, а також часу виконання в умовах мобільного застосунку для визначення їхньої придатності;

– візуалізувати результати розпізнавання харчових продуктів та здійснити порівняльний аналіз архітектур.

# 2 ЗАСТОСУВАННЯ НЕЙРОМЕРЕЖ ДЛЯ ДЕТЕКЦІЇ ОБ'ЄКТІВ

## 2.1 Архітектура та навчання SSD для розпізнавання об'єктів

Метод SSD використовує згорткову нейронну мережу прямого поширення, яка створює фіксовану кількість обмежувальних рамок та визначає ймовірність наявності об'єктів у кожній із них. Після цього застосовується алгоритм не перевищувальної супресії для відбору остаточних детекцій. Початкові шари мережі ґрунтуються на архітектурі, яка часто використовується для розпізнавання зображень, без завершальних шарів класифікації. До цієї мережі додаються додаткові шари для реалізації процесу виявлення об'єктів з необхідними характеристиками [29].

Карти ознак з різними масштабами для детекції є важливим компонентом SSD. До завершення базової скороченої мережі додаються згорткові шари ознак, які поступово зменшують свій розмір. Це дозволяє здійснювати передбачення об'єктів на кількох рівнях масштабів. Кожен шар ознак використовує окрему згорткову модель для виявлення об'єктів. Використання карт ознак з різними масштабами може давати змогу моделі бути гнучкою та точною для детекції малих та великих об'єктів на одному зображенні.

Згорткові предиктори для виявлення об'єктів використовуються для створення фіксованого набору передбачень на кожному доданому або існуючому шарі ознак базової мережі. Ці шари ознак допомагають генерувати прогноз детекції за допомогою згорткових фільтрів, як показано в архітектурі SSD на рисунку 2.1. Для кожного шару з розмірами $m \times n$ та $p$ каналами використовується невелике згорткове ядро $3 \times 3$, яке визначає, чи належить об'єкт до певної категорії, або обчислює зміщення рамки відносно початкових координат. У кожному місці застосування ядра, тобто в точках $m \times n$, воно

генерує відповідні результати. Виведені зміщення для рамки обчислюються відносно координат кожної точки на карті ознак [29].

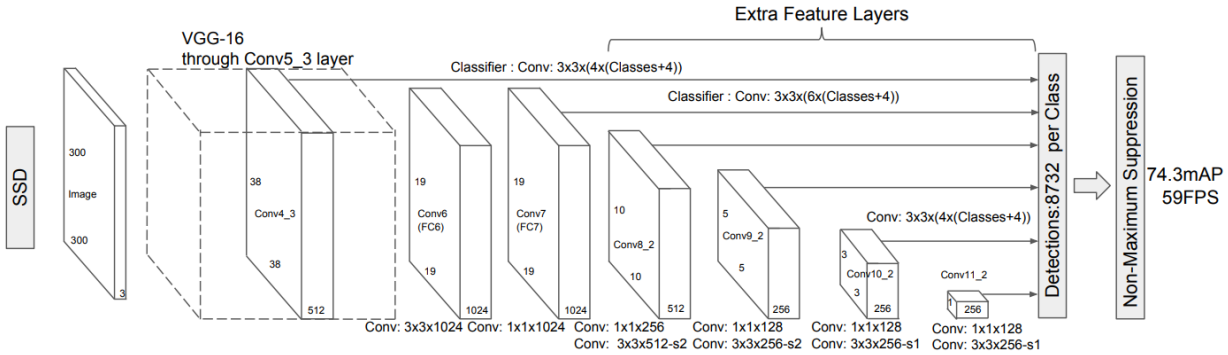


Рисунок 2.1 – Архітектура SSD [29]

В методі SSD використовується концепція обмежувальних рамок за замовчуванням (default boxes), що дозволяє ефективно обробляти об’єкти різного масштабу та співвідношення сторін. Кожен рівень карт ознак відповідає за передбачення об’єктів певного розміру, використовуючи різні обмежувальні рамки з різним співвідношенням сторін. Це дає змогу SSD бути ефективним у виявленні маленьких та великих об’єктів [30].

Метод SSD використовує комбінацію двох компонентів втрат, а саме класифікації та регресії рамок. Класифікаційна частина відповідає за правильне визначення класу об’єкта, а регресійна частина – за точне визначення координат рамок. Це дає змогу SSD бути точним у локалізації об’єктів на зображенні.

Неперевищувальна супресія використовується для уникнення дублювання детекцій. Після того як відбулося передбачення об’єктів на різних масштабах, алгоритм вибирає ті, які мають найвищу ймовірність та видаляє всі інші передбачення з меншою ймовірністю тієї ж рамки. Це дозволяє залишити лише найбільш релевантні рамки для остаточної детекції [30].

Навчання моделі SSD полягає в одночасній оптимізації розпізнавання об’єктів та регресії координат обмежувальних рамок. Для цього

застосовується крос-ентропійна втрата для розпізнавання та регресійна втрата для регресії координат рамок. Це дозволяє одночасно навчати модель правильно визначати класи об’єктів та точно локалізувати їх на зображені.

Крос-ентропійна втрата відповідає за передбачення ймовірності того, що об’єкт належить до певного класу або що у рамці немає об’єкта. Основна ідея полягає у тому, щоб мінімізувати втрату для тих рамок, які справді містять об’єкти (позитивні), і водночас забезпечити правильне передбачення фону для негативних рамок [30]. Крос-ентропійна втрата визначається наступним чином:

$$L_{conf}(x, c) = -\sum_{i \epsilon Pos} x_{ij} \log(\widehat{c_i}) - \sum_{i \epsilon Neg} \log(\widehat{c_i}), \quad (2.1)$$

де $x_{ij}$ – індикатор того, що передбачення рамки відповідає істинному класу;

$\widehat{c_i}$ – передбачена ймовірність для класу;

$Pos$ – позитивні рамки;

$Neg$ – негативні рамки.

Для точного передбачення координат обмежувальних рамок застосовується регресійна втрата, яка ефективно обробляє відхилення в координатах [30] та визначається наступним чином:

$$L_{loc}(x, l, g) = \sum_{i \epsilon Pos} x_{ij} smooth_{L1}(l_i - g_i), \quad (2.2)$$

де $l_i$ – передбачені координати обмежувальної рамки;

$g_i$ – істинні координати.

Функція $smooth_{L1}$ визначається наступним чином:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & if|x| < 1 \\ |x| - 0.5 & if|x| \geq 1 \end{cases}. \quad (2.3)$$

Ця функція дає змогу уникнути великих втрат, коли передбачення сильно відрізняються від істинних координат.

Загальна втрата для навчання моделі SSD є комбінацією крос-ентропійної та регресійної втрат і визначається наступним чином:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)), \quad (2.4)$$

Де $N$ – кількість позитивних рамок;

$\alpha$ – гіперпараметр, який визначає вагу втрати локалізації відносно втрати розпізнавання (зазвичай встановлюється рівним 1).

Функція оптимізується за допомогою градієнтного спуску та під час навчання модель намагається мінімізувати сумарну втрату, при цьому покращуючи розпізнавання об’єктів та регресію їх координат.

Оскільки обробка негативних рамок, фонів може значно перевищувати кількість позитивних рамок, то використовується механізм «hard negative mining» [30]. Він дозволяє вибирати тільки найважчі для розпізнавання негативні приклади, що значно покращує навчання. Наприклад, після кожної ітерації використовуються тільки негативні рамки, які мають найбільшу втрату для оновлення ваг мережі.

Таким чином, процес навчання SSD полягає в мінімізації загальної функції втрат, яка включає в себе втрати від крос-ентропії та регресії. Це дозволяє моделі успішно навчатися як правильно розпізнавати об’єкти на зображеннях, так і точно визначати їхні координати.

## 2.2 Компоненти та процес навчання EfficientDet

Архітектура EfficientDet (рис. 2.2) складається з декількох важливих компонентів, кожен з яких виконує свою роль у завданні розпізнавання

об’єктів. Завдяки використанню сучасних підходів, модель досягає високої продуктивності та точності, залишаючись при цьому ефективною з точки зору використання обчислювальних ресурсів.

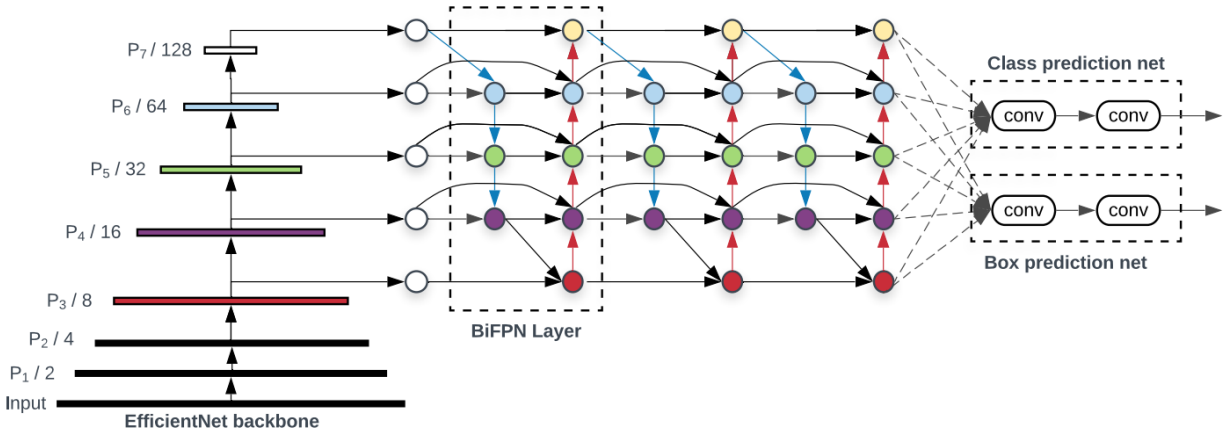


Рисунок 2.2 – Архітектура EfficientDet [31]

Першим елементом є EfficientNet Backbone, який відповідає за витягування важливих ознак із зображення. EfficientNet – це популярна архітектура CNN, оптимізована для збалансованого масштабування. Вона може одночасно змінювати глибину, ширину та розширення вхідного зображення, що дозволяє їй бути ефективною для різних розмірах вхідних даних. На кожному рівні цієї мережі, наприклад, $P_1, P_2, P_3$ відбувається витягування ознак з різними рівнями роздільної здатності. Це означає, що на нижчих рівнях модель працює з більшими деталями зображення, а на вищих рівнях модель здатна виявляти ознаки більших об’єктів або загальні структури. Такий підхід дозволяє моделі розпізнавати об’єкти різного розміру та складності, що робить її універсальною та придатною для завдань з різними вимогами до масштабу [31].

Наступним важливим елементом є BiFPN, який об’єднує ознаки з різних рівнів. Його головна особливість – це передавати інформацію в обох напрямках: зверху вниз та знизу вгору. Це допомагає ефективно поєднувати

ознаки різних масштабів, що дозволяє моделі більш точно розпізнавати об’єкти будь-яких розмірів.

Крім того, BiFPN використовує додаткові вагові коефіцієнти для різних шарів, що дає змогу динамічно коригувати важливість кожного рівня під час об’єднання ознак [31]. Це забезпечує гнучкість та адаптивність моделі під час роботи з даними, що дозволяє їй краще навчатися на складних зображеннях, де об’єкти можуть суттєво відрізнятися за розмірами, формою та розташуванням.

Наступним кроком після об’єднання ознак у BiFPN є обробка цих даних двома спеціалізованими підмережами: Class Prediction Net та Box Prediction Net. Кожна з цих підмереж виконує конкретне завдання, яке важливе для завершення процесу розпізнавання об’єктів.

Підмережа Class Prediction Net визначає, до якого класу належить кожен об’єкт, знайдений на зображенні. Наприклад, якщо модель аналізує зображення з кількома об’єктами, такими як яблука, банани та апельсини, то підмережа ідентифікує кожен з них, як яблуко, банан чи апельсин. Class Prediction Net складається з кількох згорткових шарів, які обробляють ознаки, згенеровані та злиті у BiFPN. Згорткові шари дозволяють моделі виявляти складні взаємозв’язки між ознаками, що допомагає здійснити більш точну класифікацію. Кінцевий результат роботи підмережі – це вектор класів, який вказує на ймовірність того, до якого класу належить кожен об’єкт на зображенні. Класи можуть бути різними, залежно від специфіки завдання, яке вирішує модель [31].

З іншого боку, підмережа Box Prediction Net відповідає за визначення місця розташування об’єктів на зображенні. Ця підмережа прогнозує точні координати кожного об’єкта у вигляді прямокутників. Ці прямокутники визначають, де саме на зображенні знаходиться об’єкт та скільки місця він займає. Також, підмережа Box Prediction Net використовує кілька згорткових шарів для обробки ознак, згенерованих у попередніх етапах. Однак замість

класифікації об’єктів, ці шари аналізують просторову інформацію про об’єкт, визначаючи його точні межі [31].

Ці дві підмережі працюють разом, щоб забезпечити повний процес розпізнавання об’єктів. Спершу модель визначає, що за об’єкт присутній на зображенні або відео, а потім знаходить точне місце розташування цього об’єкта. Наприклад, коли модель бачить яблуко на зображенні, спершу Class Prediction Net визначає, що це саме яблуко, а потім Box Prediction Net вказує, де саме це яблуко знаходиться, за допомогою координат, що окреслюють його межі.

Поєднання цих двох процесів: класифікації та визначення місця об’єкта, дозволяє EfficientDet виконувати завдання розпізнавання об’єктів швидко і з високою точністю.

Навчання моделі EfficientDet відбувається за допомогою оптимізаторів, таких як Adam або SGD. Оптимізатор оновлює ваги моделі на основі градієнтів, обчислених відносно функції втрат [32]. Формула оновлення ваг для кожного кроку навчання виглядає наступним чином:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t), \tag{2.5}$$

де $\theta_t$ – параметри моделі на кроці $t$;

$\eta$ – швидкість навчання;

$\nabla L(\theta_t)$ – градієнт функції втрат відносно параметрів моделі.

Цей процес повторюється до тих пір, поки функція втрат (2.2) не мінімізується до прийнятного рівня.

Таким чином, процес навчання EfficientDet є багатоступеневим та орієнтованим на одночасну оптимізацію двох задач: класифікації об’єктів та передбачення їхнього місця розташування на зображенні або відео. Це робить модель потужною та універсальною для завдань розпізнавання об’єктів у різних сценаріях.

## 2.3 Трансферне навчання

Трансферне навчання – це техніка в машинному навчанні, коли модель навчена для одного завдання, а використовується як стартова точка для моделі на другому завданні. Це може бути корисним, коли друге завдання схоже на перше або коли для другого завдання є обмежені дані. Використовуючи вивчені характеристики першого завдання як початкову точку, модель може навчатися ефективніше та швидше на другому завданні. Це також може допомогти запобігти перенавчанню, оскільки модель вже засвоїла загальні характеристики, які можуть бути корисними для другого завдання [33, 34].

Трансферне навчання є важливим інструментом у машинному навчанні, і його застосування має кілька суттєвих переваг. Однією з головних є можливість працювати з невеликою кількістю даних. Часто буває складно та дорого зібрати великий обсяг маркованих даних для створення нової моделі з нуля. Тому, використовуючи вже існуючі моделі, що були навчені на великих наборах даних, можна заощадити час та ресурси, зменшуючи потребу в додаткових даних.

Ще однією важливою перевагою є покращення продуктивності. Оскільки модель була навчена на значному обсязі даних та має певний рівень знань, то її можна швидко адаптувати для нових завдань, досягаючи високої точності [34].

Крім того, трансферне навчання є досить гнучким. Моделі, що були створені для одного завдання, можна переналаштувати для виконання схожих завдань. Завдяки цій здатності до адаптації, трансферне навчання широко використовується у багатьох сферах, що дозволяє ефективно переносити знання між різними завданнями.

Процес навчання у трансферному навчанні (рис. 2.3) починається з вибору архітектури вихідної моделі, яка була попередньо навчена на об’ємному наборі даних, наприклад, на ImageNet з 1000 класами. Така модель складається зі згорткових шарів, які навчилися витягувати ознаки зображень і

щільних шарів, що відповідають за класифікацію цих ознак. Це дозволяє моделі мати загальні знання, які корисні для багатьох подібних задач.

Наступним кроком є перенесення шарів. У цьому випадку вибираються згорткові шари, які вже навчилися розпізнавати низькорівневі ознаки на попередньому етапі. Ці шари переносяться в нову модель разом із «замороженими» вагами, тобто їх параметри не оновлюються під час подальшого навчання. Це дозволяє зберегти знання, які були накопичені на попередньому наборі даних, та використовувати їх для нового завдання [34].

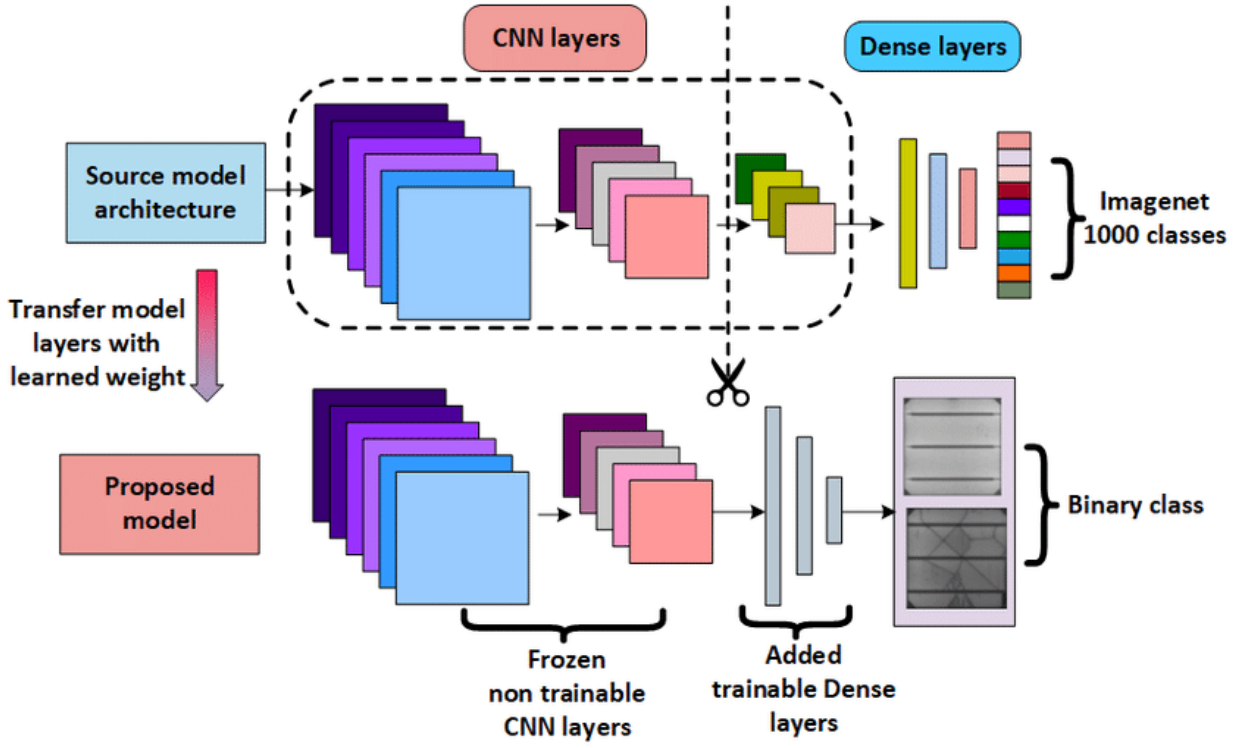


Рисунок 2.3 – Процес навчання у трансферному навчанні [34]

Для адаптації до нового завдання додаються нові щільні шари, які відповідають за класифікацію вже з новими класами. Наприклад, у даному випадку завдання змінилося з класифікації 1000 класів до бінарної класифікації, де потрібно моделі розпізнавати лише два класи. Нові шари є тренованими та налаштовуються під специфіку нового завдання.

Далі модель проходить етап навчання нових шарів на основі нового набору даних. Під час цього процесу нові шари адаптуються, але при цьому

попередньо перенесені згорткові шари залишаються незмінними. Це дозволяє зберегти продуктивність та точність моделі, використовуючи вже вивчені раніше ознаки, а також оптимізувати роботу нових шарів для специфічного завдання [34].

На заключному етапі модель може виконувати розпізнавання відповідно до нового завдання. У цьому випадку модель займається бінарною класифікацією, наприклад, визначаючи наявність або відсутність об’єкта на зображені. Трансферне навчання дозволяє використовувати наявні знання, що сприяють швидшому та ефективнішому навчальному процесу при вирішенні нових задач, навіть коли доступних даних мало.

Хоча трансферне навчання має багато переваг, важливо також звернути увагу на його недоліки. Один із основних мінусів полягає в тому, що цей підхід ефективний лише тоді, коли вихідне завдання та нове мають спільні ознаки. Якщо завдання буде сильно відрізнятися за своєю природою, то модель може не мати відповідних знань для перенесення, що робить трансферне навчання малоефективним у таких випадках [35].

Ще одним недоліком є процес тонкого налаштування. Якщо цей етап виконується неправильно, то існує ризик того, що буде перенавчання або погіршиться якість роботи моделі. Налаштування вимагає обережного підходу та правильного вибору гіперпараметрів, щоб модель могла адаптуватися до нових даних без втрати загальної продуктивності.

Крім того, великий розмір архітектури та складність обчислень можуть вимагати значних обчислювальних ресурсів, навіть при використанні попередньо навченої моделі. Це може бути проблематичним для застосування у реальних умовах, де ресурси можуть бути обмеженими [35].

Таким чином, трансферне навчання не є універсальним рішенням та має свої обмеження. Однак, при правильному застосуванні трансферне навчання може значно покращити продуктивність моделей у багатьох завданнях машинного навчання.

## 2.4 Три популярні архітектури для розпізнавання зображень

### 2.4.1 Архітектура SSD-MobileNetV2-320

Модель SSD-MobileNetV2-320 (рис. 2.4) є однією з популярних архітектур для задач розпізнавання об’єктів, яка об’єднує два ключових компонента: базову мережу MobileNetV2 та головку детектора SSD. Ця модель оптимізована для роботи в мобільних застосунках та пристроях з обмеженими ресурсами, забезпечуючи баланс між швидкістю та точністю [36].

Базова мережа слугує основою, що відповідає за обробку зображень та витягування ознак для розпізнавання об’єктів. Вона побудована з інвертованих залишкових блоків, які підвищують ефективність обчислень завдяки застосуванню глибинних згорток. Завершується мережа розміром $1 \times 1$ згорткою, яка зменшує кількість каналів, що допомагає скоротити розмір вихідних даних та збільшити швидкість роботи.

Головка SSD відповідає за визначення координат об’єктів та їх класифікацію. Вона включає глобальний пулінг для зведення просторових ознак до одного вектора, $1 \times 1$ згортку або повнозв’язний шар для розпізнавання об’єктів, а також функцію активації Softmax, яка прогнозує найбільш ймовірні категорії об’єктів на зображенні [36].

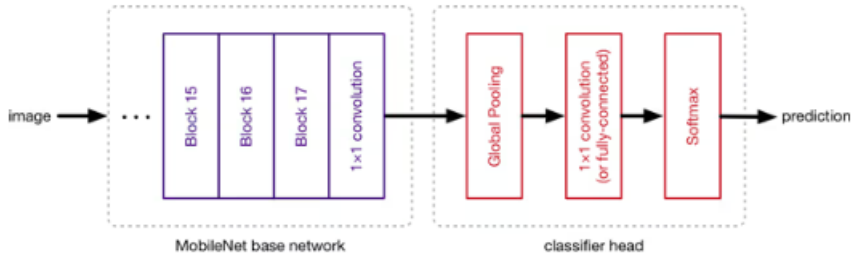


Рисунок 2.4 – Архітектура моделі SSD-MobileNetV2-320 [36]

Архітектура SSD-MobileNetV2-320 характеризується високою швидкістю, що робить її придатною для роботи в реальному часі на мобільних пристроях. Завдяки використанню глибинних згорток, модель має малу вагу

та споживає мінімум ресурсів, що важливо для пристроїв з обмеженими обчислювальними можливостями [37]. Модель також демонструє гнучкість у розпізнаванні об’єктів різного розміру.

Однак, модель SSD-MobileNetV2-320 має недоліки. Вона знижує точність при розпізнаванні малих об’єктів. Глибинні згортки можуть призвести до втрати деталей, що знижує ефективність для складних задач. SSD-MobileNetV2-320 потребує ретельної настройки гіперпараметрів для отримання кращих результатів [38].

Загалом, модель SSD-MobileNetV2-320 підходить для задач, де важливі швидкість та ефективність, проте для задач з високими вимогами до точності краще обрати іншу архітектуру.

### 2.4.2 Архітектура EfficientDet-D0

Модель EfficientDet-D0 (рис. 2.5) є однією з перших архітектур у серії EfficientDet, яка об’єднує високу ефективність та точність для вирішення задач розпізнавання об’єктів. Її основою є EfficientNet – потужна мережа для класифікації зображень, яка використовує методи нейронного архітектурного пошуку для оптимізації своєї структури. Це дозволяє досягти максимальної продуктивності при мінімальних витратах ресурсів [39].

Однією з характерних рис EfficientDet є BiFPN, що дозволяє ефективно обробляти інформацію з різних масштабів. BiFPN реалізує двосторонні зв’язки між різними рівнями, що покращує навчання на різних деталях зображення. Завдяки цьому підходу модель може краще виявляти об’єкти, незважаючи на їх розмір.

Модель EfficientDet-D0 має дві окремі мережі для передбачень. Одна для розпізнавання об’єктів, інша для визначення меж. Такий підхід дозволяє моделі більш точно виконувати кожну задачу, що підвищує загальну точність результатів [39].

Обробка вхідних зображень відбувається через кілька шарів, які включають згорткові шари та BiFPN, які дозволяють ефективно виділяти ознаки з різних масштабів. В результаті модель генерує передбачення класів об’єктів та координат їхніх меж, що є ключовим для точного розпізнавання.

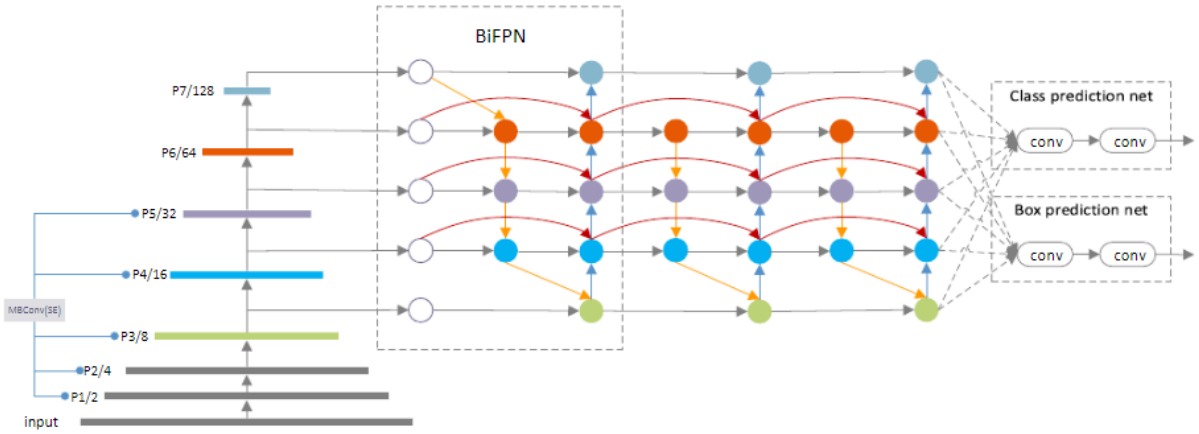


Рисунок 2.5 – Архітектура моделі EfficientDet-D0 [40]

Основними перевагами моделі EfficientDet-D0 є її ефективність та швидкість. Модель демонструє високу точність при низьких витратах обчислювальних ресурсів, що робить її придатною для використання на мобільних пристроях. Швидкість обробки зображень дозволяє ефективно застосувати модель в реальних умовах, таких як розпізнавання харчових продуктів, відеоспостереження, автономні автомобілі та інші системи, де необхідна швидка реакція на зміну обстановки [40].

Хоча модель EfficientDet-D0 має багато переваг, але вона також має свої недоліки. По-перше, вона є базовою версією в серії EfficientDet, тому для складніших задач може знадобитися більш потужна модель, яка забезпечить вищу точність, але за рахунок швидкості.

По-друге, EfficientDet-D0 чутлива до якості вхідних даних: погане освітлення або шум можуть суттєво знизити точність. Крім того, модель потребує значних обчислювальних ресурсів для навчання, що може бути перешкодою для користувачів з обмеженими можливостями [40].

Загалом, модель EfficientDet-D0 є потужною архітектурою для вирішення задач комп'ютерного зору, пропонуючи оптимальний баланс між точністю, швидкістю та гнучкістю в налаштуваннях для різноманітних сценаріїв використання.

### 2.4.3 Архітектура SSD-MobileNetV2-FPNLite320

Модель SSD-MobileNetV2-FPNLite320 – це сучасна архітектура для розпізнавання об'єктів, яка поєднує в собі переваги швидкості та точності. Основою цієї архітектури є SSD, вдосконалений за допомогою MobileNetV2, що робить його ефективним для використання на мобільних пристроях. Однією з основних особливостей моделі є використання Feature Pyramid Network (FPN), яка дозволяє обробляти об'єкти різних розмірів [41]. На рисунку 2.6 представлено структуру FPN, яка дозволяє обробляти об'єкти різних розмірів.

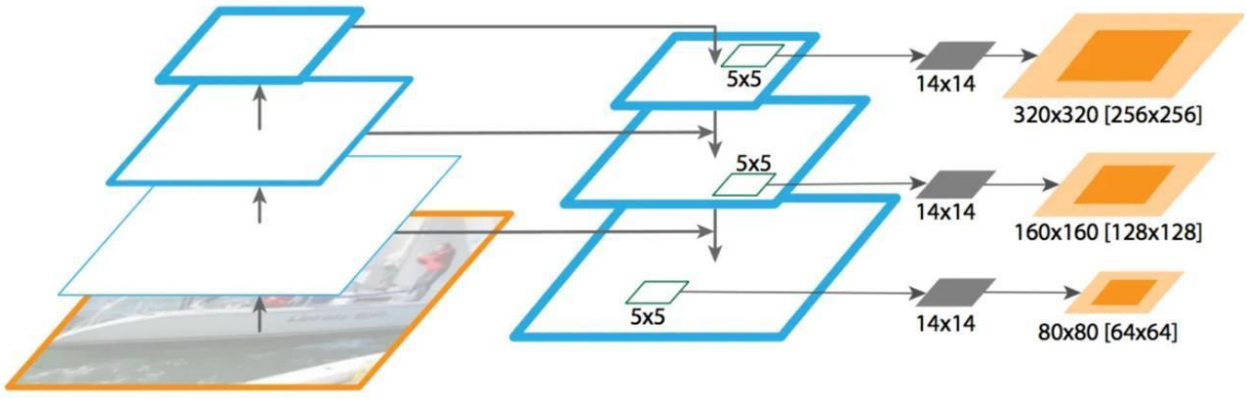


Рисунок 2.6 – Структура FPN, що поєднує інформацію з різних рівнів для поліпшення детектування об'єктів різних розмірів

Процес роботи починається з введення зображення, яке масштабується до розміру $320 \times 320$ пікселів. Далі модель аналізує зображення та витягує особливі ознаки, використовуючи багаторівневе представлення, яке надає

FPN. На основі цих ознак формуються бокси та класифікації для кожного виявленого об’єкта. Для покращення точності застосовуються методи постобробки, які видаляють дублюючі передбачення [42].

Отже, однією з ключових переваг моделі SSD-MobileNetV2-FPNLite320 є висока швидкість обробки, що дозволяє їй працювати в реальному часі. Крім того, модель можна використовувати на мобільних та вбудованих пристроях. Завдяки застосуванню FPN, архітектура ефективно виявляє об’єкти різних розмірів, покращуючи точність як для великих, так і для малих об’єктів.

Серед недоліків варто зазначити знижену точність при виявленні дуже малих об’єктів у порівнянні з більш складними архітектурами, такими як Faster R-CNN. Крім того, модель має обмежену гнучкість у налаштуванні для специфічних задач, що може знизити її ефективність у складних застосуваннях.

## 2.5 Опис наступних етапів та оцінка ефективності розпізнавання об’єктів

У цій кваліфікаційній роботі передбачається реалізація розпізнавання харчових продуктів за допомогою трьох популярних архітектур CNN: SSD-MobileNetV2-320, EfficientDet-D0 та SSD-MobileNetV2-FPNLite320 та оцінювання ефективності розпізнавання харчових продуктів.

Процес розпізнавання буде включати наступні кроки:

Крок 1. Вибір набору даних для навчання та тестування. В даному дослідженні застосовується власноруч створений датасет, що містить зображення різних категорій харчових продуктів. Крім того, можуть використовуватися загальнодоступні датасети, відповідні для задачі розпізнавання харчових продуктів.

Крок 2. Підготовка та обробка даних. На цьому етапі система збирає та організовує файли зображень і автоматично переміщує зображення в

відповідні папки для тренування, валідації та тестування. Після цього використовуються скрипти для конвертації даних у формат TFRecord та створення файлу pbtxt, необхідних для роботи з TensorFlow Object Detection API.

Крок 3. Вибір архітектури та налаштування конфігурації. На цьому етапі обирається модель, наприклад, SSD-MobileNetV2-320. Завантажуються попередньо натреновані ваги для застосування Transfer Learning та відповідний конфігураційний файл. Далі цей файл налаштовується: вказуються шляхи до тренувальних та валідаційних даних, шлях до міток, кількість класів для розпізнавання, розмір батча, кількість навчальних кроків та швидкість навчання для тренування моделі.

Крок 4. Навчання моделі. На цьому етапі модель навчається на тренувальних даних з використанням налаштованих параметрів, де ключовими аспектами є застосування fine_tune_checkpoint для продовження навчання з попередньо натренованих ваг, а також налаштування моделі для детекції продуктів з урахуванням конфігураційних параметрів, таких як швидкість навчання та кількість епох. Під час навчання модель коригує свої ваги на основі результатів кожної епохи, а метрика втрат допомагає відстежувати прогрес. Окрім цього, використовується валідаційний набір для оцінки якості навчання на кожному кроці, щоб уникнути перенавчання.

Крок 5. Тестування моделі та оцінка. На цьому етапі модель тестується на тестовій вибірці зображень харчових продуктів. Результати детекції зберігаються у вигляді файлів із прогнозами моделі. Для оцінки якості роботи моделі використовуються скрипти, що обчислюють метрики, зокрема mAP та Inference speed, які дозволяють оцінити точність та ефективність детекції об’єктів на нових даних.

Оцінка проводиться за допомогою метрики mAP, кількості правильно ідентифікованих об’єктів, часу обробки моделей на основі офіційних бенчмарків TensorFlow, часу виконання моделей у середовищі Python-

скриптів, а також часу виконання в умовах мобільного застосунку для визначення їхньої придатності.

Крок 6. Квантування моделі для подальшого використання на мобільних пристроях.

Крок 7. Виконання Кроків 2–6 з використанням інших архітектур нейронних мереж.

Крок 8. Аналіз ефективності різних архітектур на обраному наборі даних.

Метрика mAP – це узагальнена метрика, яка використовується для оцінки загальної ефективності моделей детекції об’єктів, комбінуючи метрики precision та recall [43]. Вона вимірює, наскільки добре модель виявляє об’єкти та наскільки точно вона робить це для кожного класу.

Метрика precision вимірює, наскільки точні передбачення моделі. Вона показує, яка частка передбачених об’єктів є правильними [44] та обчислюється за формулою:

$$Precision = \frac{TP}{TP+FP}, \tag{2.6}$$

де $TP$ – кількість правильно передбачених об’єктів;

$FP$ – кількість неправильно передбачених об’єктів.

Метрика recall – метрика, яка вимірює, наскільки повно модель знаходить всі об’єкти [44]. Вона показує, яка частка реальних об’єктів була правильно виявлена моделлю та обчислюється за формулою:

$$Recall = \frac{TP}{TP+FN}, \tag{2.7}$$

де $FN$ – кількість пропущених реальних об’єктів.

Зазвичай precision та recall перебувають у взаємозв’язку, тобто підвищення одного показника може призводити до зниження іншого.

Наприклад, якщо модель буде працювати більш «обережно» і передбачатиме тільки ті об’єкти, в яких вона впевнена, то точність буде високою, проте повнота може впасти, оскільки деякі об’єкти залишаться непоміченими. Та навпаки, якщо модель буде виявляти більше об’єктів, навіть з меншою впевненістю, повнота покращиться, але точність може постраждати через збільшення кількості хибних передбачень.

Для оцінки продуктивності моделі на різних порогах впевненості будується крива PR. Ця крива відображає співвідношення між точністю і повнотою моделі на різних рівнях порогу впевненості від 0 до 1.

Процес побудови PR-кривої включає декілька кроків. Спочатку модель робить передбачення об’єктів на зображенні з різним рівнем впевненості. В кожному випадку модель оцінює, наскільки вона впевнена в правильності свого передбачення. Для кожного порогу впевненості розраховуються показники precision та recall [44]. Ці показники відображаються на графіку (рис. 2.7), де по осі $x$ відкладається recall, а по осі $y$ – precision.

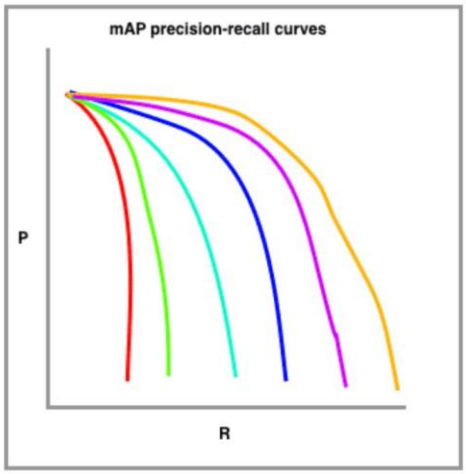


Рисунок 2.7 – Ескіз кривих точності-відтворення mAP [44]

Average Precision (AP) – метрика, яка вимірює точність моделі для одного класу [44] та обчислюється за формулою:

$$AP = \sum_n (R_n - R_{n-1}) P(R_n) \,, \tag{2.8}$$

де $P(R_n)$ – точність при відповідному рівні recall $R_n$;

$R_n - R_{n-1}$ – різниця між сусідніми значеннями recall.

Метрика IoU – метрика, яка визначає, наскільки точно модель може передбачити об'єкти. Вона вимірює ступінь перекриття між передбаченою рамкою об'єкта (bounding box) і істинною рамкою об'єкта [45]. IoU визначається як відношення перетину передбаченої та істинної областей до їх об'єднання:

$$IoU = \frac{Area_{pred} \cap Area_{true}}{Area_{pred} \cup Area_{true}}, \quad (2.9)$$

де $Area_{pred}$ – площа передбаченої рамки;

$Area_{true}$ – площа істинної рамки.

Отже, метрика mAP – це середня точність моделі по всіх класах та є усередненням всіх AP для кожного класу. Тобто, якщо є кілька класів об'єктів, то для кожного з них розраховується AP, а потім вже середнє значення цих AP дає mAP. Формула для розрахунку mAP виглядає так:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i, \quad (2.10)$$

де $N$ – кількість класів даної моделі;

$AP_i$ – середня точність для класу $i$.

Inference speed – це час, який необхідний для обробки вхідних даних нейронною мережею та генерації вихідних результатів [46]. Цей показник є важливим при порівнянні моделей машинного навчання, в задачах розпізнавання об'єктів, де швидкісна реакція системи може значно вплинути на ефективність роботи в реальному часі.

Основні чинники, які впливають на швидкість виведення: архітектура моделі, вхідні дані та обчислювальна платформа. По-перше, чим більше шарів і параметрів в архітектурі моделі, тим більше часу потрібно для її обробки.

Складніші моделі можуть бути точнішими, але водночас вимагають більше часу для виконання всіх необхідних обчислень.

По-друге, розмір та складність вхідних даних впливають на швидкість виведення. Наприклад, зображення з вищою роздільною здатністю вимагають більше обчислювальних ресурсів, що збільшує час обробки.

По-третє, потужність пристрою або сервера, на якому виконується модель можуть дуже вплинути на продуктивність. Використання GPU, спеціалізованих процесорів або мобільних платформ може значно підвищити ефективність.

Для вимірювання швидкості виведення моделей використовуються різні інструменти, наприклад TensorFlow Benchmark [46]. Цей інструмент дозволяє проводити багаторазові ітерації тестування для отримання точних результатів продуктивності моделі. Важливими етапами є проведення кількох ітерацій тестування для зменшення випадкових коливань, фіксація мінімального, максимального та середнього часу виведення, а також аналіз середнього часу для оцінки продуктивності моделі.

Для розрахунку середнього часу виведення моделі за результатами кількох ітерацій використовується така формула:

$$T_{avg} = \frac{1}{N}\sum_{i=1}^{N} T_i \ , \tag{2.11}$$

де $T_{avg}$ – середній час виведення моделі;

$T_i$ – час виведення на кожній ітерації;

$N$ – кількість ітерацій.

Отже, середнє значення часу виведення є хорошим показником того, як модель працюватиме за умови постійної експлуатації. Це важливо для завдань, де необхідна робота в реальному часі, оскільки затримки можуть негативно вплинути на кінцевий результат.

# 3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕТОДУ ДЕТЕКЦІЇ ОБ'ЄКТІВ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖ

## 3.1 Вибір програмного забезпечення

В рамках даної кваліфікаційної роботи було розроблено розпізнавання харчових продуктів за допомогою трьох популярних архітектур нейромереж: SSD-MobileNetV2-320, EfficientDet-D0, SSD-MobileNetV2-FPNLite320, а також проведено оцінку ефективності у задачах розпізнавання.

Для виконання завдання була обрана мова програмування Python, яка завдяки своїй широкій підтримці в науці є стандартом у галузі машинного навчання та комп'ютерного зору [47]. Python має багато різноманітних спеціалізованих бібліотек та фреймворків, таких як Keras, NumPy, Pandas, Matplotlib та TensorFlow, що створюють для розробників та дослідників потужну екосистему, дозволяючи їм ефективно обробляти дані, будувати та тренувати моделі, а також візуалізувати результати.

Бібліотека TensorFlow є однією з основних бібліотек, яка використовувалася для побудови, тренування та оптимізації моделей. TensorFlow забезпечує побудову моделей на рівні низькорівневих обчислень та спрощує створення нейронних мереж за допомогою Keras API. Використання бібліотеки TensorFlow дозволило ефективно працювати з різними архітектурами моделей, адаптуючи їх до завдань розпізнавання об'єктів, на прикладі харчових продуктів. Окрім цього, TensorFlow надає підтримку конвертації моделей у формат TFLite, що дозволяє інтегрувати їх у мобільні платформи, забезпечуючи високу продуктивність для реальних застосунків [47].

Для написання коду та візуального аналізу результатів було обрано середовище Jupyter Notebook (рис. 3.1). Воно забезпечує не тільки можливість написання коду, але й покрокове виконання з додаванням текстових та

графічних блоків [48]. Це інтерактивне середовище значно спрощує розробку моделей, полегшуючи тестування різних ідей, налаштування параметрів та моніторинг навчання в реальному часі. Такий підхід дозволяє аналізувати результати на кожному етапі і своєчасно коригувати можливі помилки.

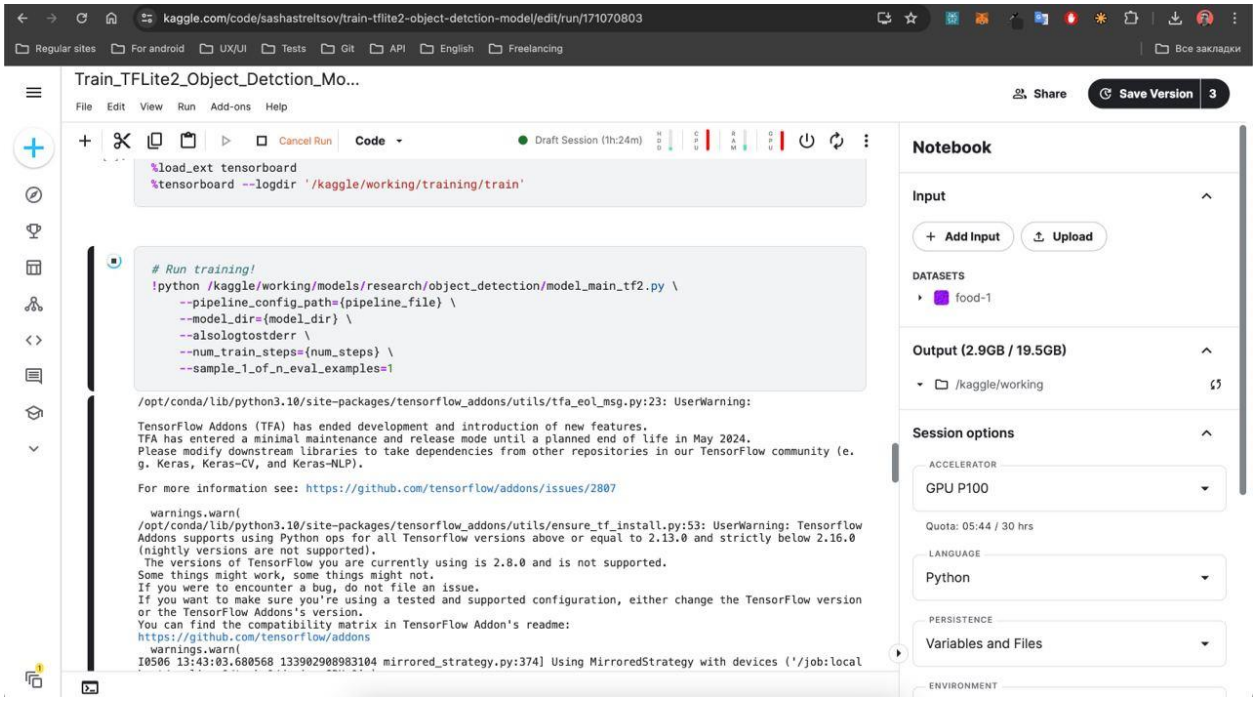


Рисунок 3.1 – Приклад роботи Jupyter Notebook в Kaggle

Для зберігання та обробки великих наборів даних використовувалась платформа Kaggle, яка надала необхідні обчислювальні ресурси, включаючи графічні процесори (GPU). Використання Kaggle дозволило суттєво прискорити процес навчання моделей, що дуже важливо при роботі з великими обсягами даних та складними нейронними мережами. Завдяки паралельній обробці обчислень на GPU, тренування моделі проходило набагато швидше, ніж при використанні звичайного процесора (CPU). Крім того, платформа Kaggle надає можливість безкоштовно використовувати віртуальні середовища з налаштованими бібліотеками, що полегшує та стабілізує процес розробки [49].

Отже, для виконання завдання використовувалася мова програмування Python разом з бібліотеками TensorFlow. Процес розробки проходив у

середовищі Jupyter Notebook, а обробку та зберігання зображень забезпечувала платформа Kaggle, яка надала можливість використовувати GPU для прискорення навчання моделей.

## 3.2 Створення датасету харчових продуктів

Процес навчання моделей для розпізнавання об’єктів починається зі збору даних. Хоча існує багато способів автоматизації цього процесу, в даному випадку збір даних виконується вручну, щоб відфільтрувати шумні зображення та забезпечити високу якість навчальної вибірки. Після збору даних необхідно виконати тегування зображень у наборі. Цей етап буде виконано за допомогою програмного забезпечення LabelImg, яке є реалізацією з відкритим кодом на основі Python і дозволяє створювати та зберігати рамки у файлах XML. Нижче на рисунку 3.2 наведено зображення, зроблене під час тегування.

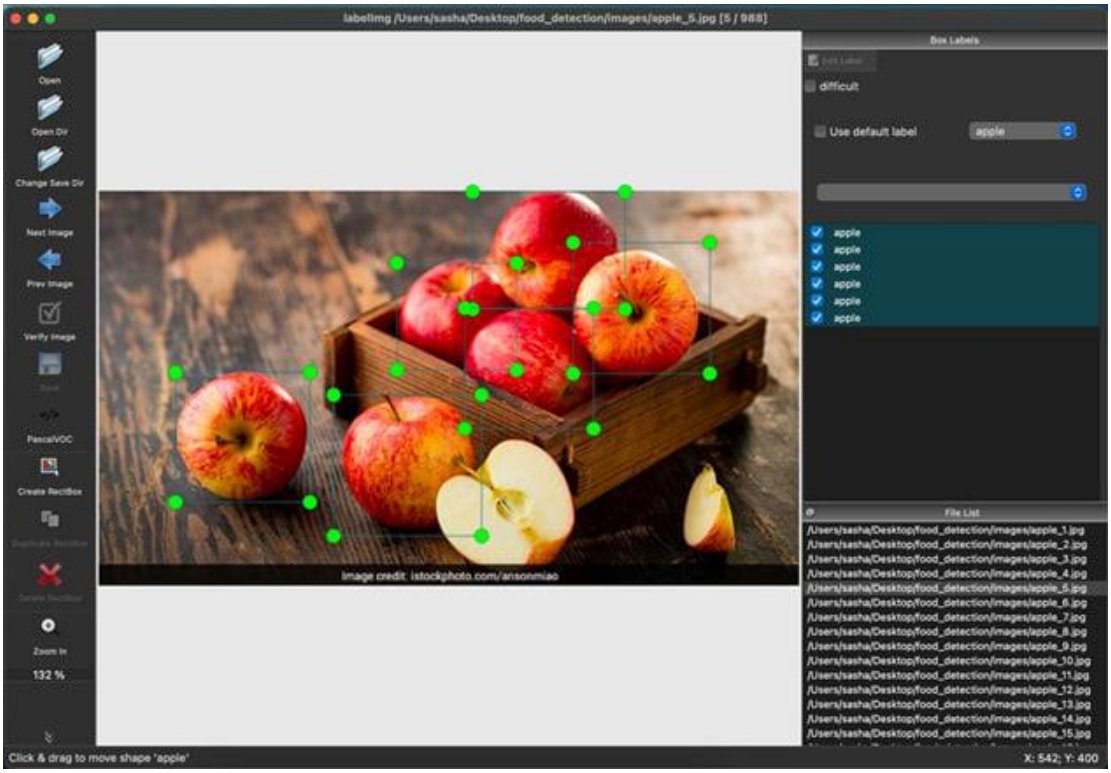


Рисунок 3.2 – Процес тегування зображень в LabelImg

Результатом роботи з тегування зображень є файл XML, який містить рамки тегованих зображень. Зразок XML-файлу показано на рисунку 3.3 для ознайомлення.

```
<annotation>
        <folder>images</folder>
        <filename>apple_55.jpg</filename>
        <path>/Users/sasha/Desktop/food_detection/images/apple_55.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>350</width>
                <height>232</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>apple</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>41</xmin>
                        <ymin>67</ymin>
                        <xmax>151</xmax>
                        <ymax>177</ymax>
                </bndbox>
        </object>
        <object>
                <name>apple</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>118</xmin>
                        <ymin>40</ymin>
                        <xmax>217</xmax>
                        <ymax>104</ymax>
                </bndbox>
        </object>
        <object>
                <name>apple</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>223</xmin>
                        <ymin>53</ymin>
                        <xmax>322</xmax>
                        <ymax>137</ymax>
                </bndbox>
        </object>
</annotation>
```

Рисунок 3.3 – Зразок XML-файлу

Щоб зручно зберігати зображення та мати швидкий доступ до них, доцільно використовувати хмарні сховища, наприклад, платформу Kaggle (рис. 3.4). Це рішення не лише дозволяє зберігати значні обсяги даних, а й полегшує управління ними, забезпечуючи можливість доступу з будь-якого пристрою та місця. Використання хмарних сервісів також підвищує безпеку даних і спрощує їх організацію в процесі роботи.

Крім того, платформа Kaggle надає можливості командної роботи, де кілька користувачів можуть мати доступ до одних і тих самих ресурсів, що робить спільні проєкти зручнішими й ефективнішими. Також доступні інструменти для аналізу та візуалізації даних, які можуть бути корисними на етапі попередньої обробки та підготовки даних для навчання моделі.

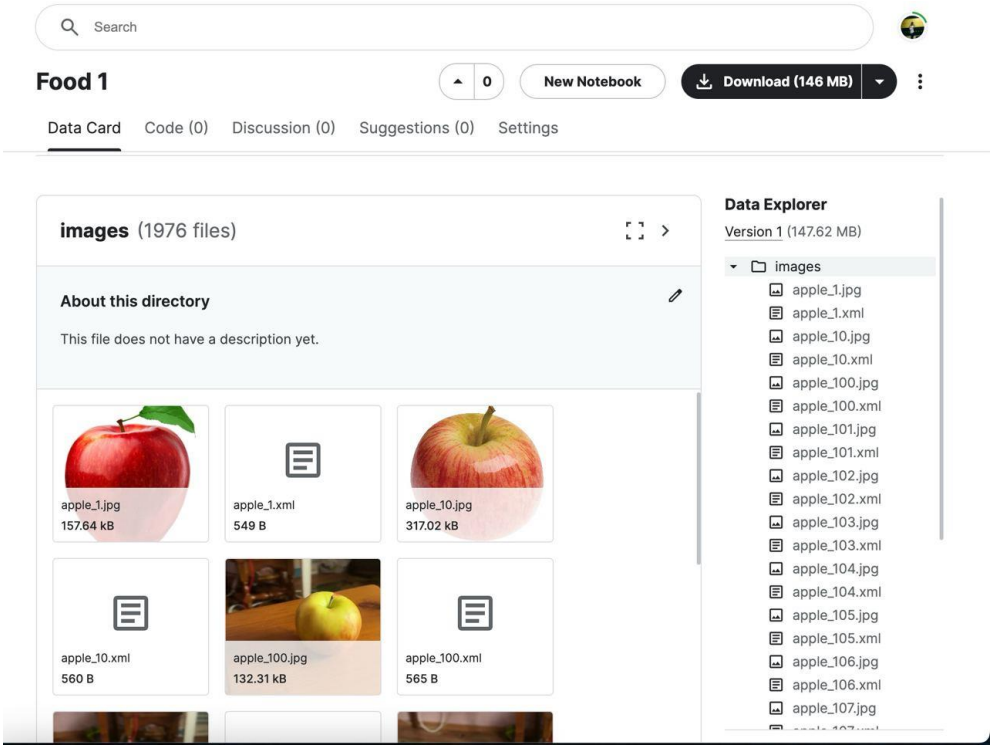


Рисунок 3.4 – Хмарне сховище для електронної колекції харчових продуктів

Нижче на рисунках 3.5–3.8 представлені приклади зображень харчових продуктів із даного датасету, які демонструють різноманіття категорій та візуальних особливостей. Зображення охоплюють різні ракурси, освітлення та

інші умови зйомки, що сприяє підвищенню стійкості моделі до змінних факторів у реальних умовах розпізнавання.

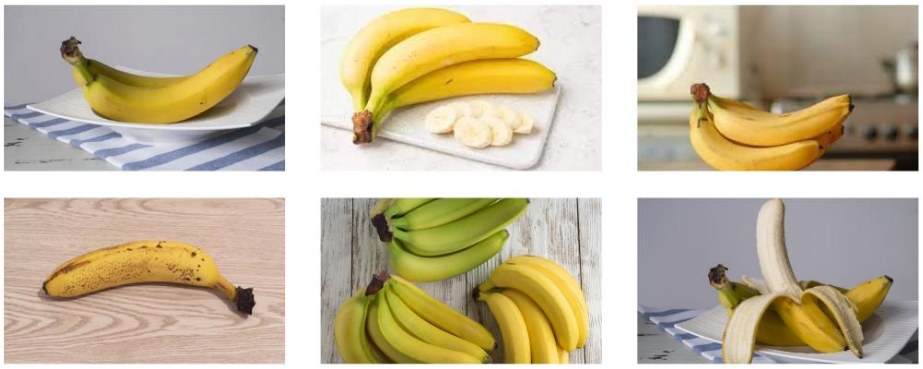

Рисунок 3.5 – Приклад датасету харчових продуктів: банани

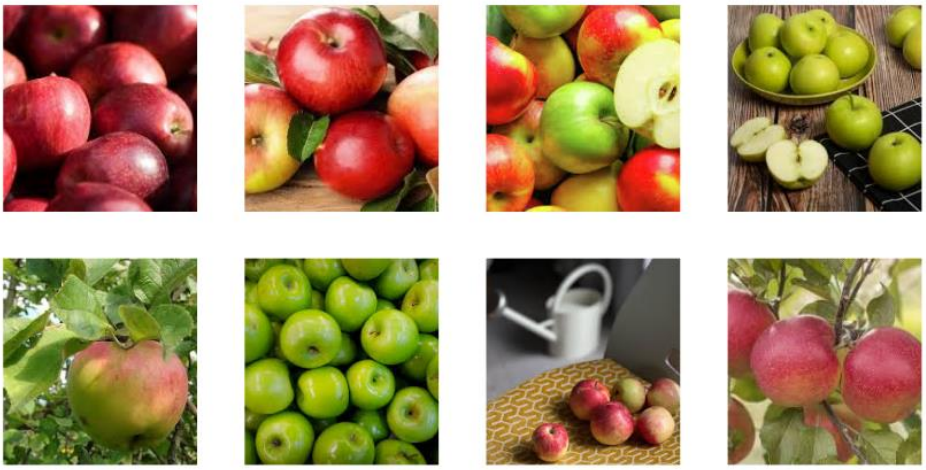

Рисунок 3.6 – Приклад датасету харчових продуктів: яблука

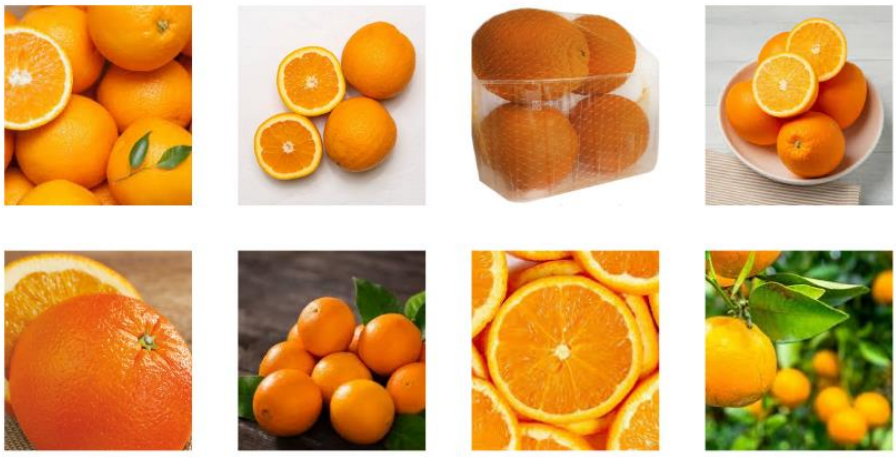

Рисунок 3.7 – Приклад датасету харчових продуктів: апельсини

Рисунок 3.8 – Приклад датасету харчових продуктів: мандарини

## 3.3 Програмна реалізація розпізнавання харчових продуктів

Процес розпізнавання харчових продуктів розпочався з того, що датасет був поділений на три частини: для навчання, валідації та тестування (рис. 3.9–3.10). Це дозволило оцінювати моделі на даних, яких вони раніше не бачили. Після цього використовувалися скрипти для конвертації даних у формат TFRecord та створили файл pbtxt, який необхідний для роботи з TensorFlow Object Detection API (рис. 3.11).

```
!mkdir -p /kaggle/working/images/all
!cp -r /kaggle/input/food-1/images/* /kaggle/working/images/all/
!mkdir -p /kaggle/working/images/train
!mkdir -p /kaggle/working/images/validation
!mkdir -p /kaggle/working/images/test

from pathlib import Path
import random
import os
import sys

# Define paths to image folders
image_path = '/kaggle/working/images/all'
train_path = '/kaggle/working/images/train'
val_path = '/kaggle/working/images/validation'
test_path = '/kaggle/working/images/test'
```

Рисунок 3.9 – Лістинг коду для створення папок

```
# Get list of all images
jpeg_file_list = [path for path in Path(image_path).rglob('*.jpeg')]
jpg_file_list = [path for path in Path(image_path).rglob('*.jpg')]
png_file_list = [path for path in Path(image_path).rglob('*.png')]
bmp_file_list = [path for path in Path(image_path).rglob('*.bmp')]

if sys.platform == 'linux':
    JPEG_file_list = [path for path in Path(image_path).rglob('*.JPEG')]
    JPG_file_list = [path for path in Path(image_path).rglob('*.JPG')]
    file_list = jpg_file_list + JPG_file_list + png_file_list + bmp_file_list + JPEG_file_list + jpeg_file_list
else:
    file_list = jpg_file_list + png_file_list + bmp_file_list + jpeg_file_list

file_num = len(file_list)
print('Total images: %d' % file_num)

# Determine number of files to move to each folder
train_percent = 0.8  # 80% of the files go to train
val_percent = 0.1 # 10% go to validation
test_percent = 0.1 # 10% go to test
train_num = int(file_num*train_percent)
val_num = int(file_num*val_percent)
test_num = file_num - train_num - val_num
print('Images moving to train: %d' % train_num)
print('Images moving to validation: %d' % val_num)
print('Images moving to test: %d' % test_num)

# Select 80% of files randomly and move them to train folder
for i in range(train_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, train_path+'/'+fn)
    os.rename(os.path.join(parent_path,xml_fn),os.path.join(train_path,xml_fn))
    file_list.remove(move_me)
# Select 10% of remaining files and move them to validation folder
for i in range(val_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, val_path+'/'+fn)
    os.rename(os.path.join(parent_path,xml_fn),os.path.join(val_path,xml_fn))
    file_list.remove(move_me)

# Move remaining files to test folder
for i in range(test_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, test_path+'/'+fn)
    os.rename(os.path.join(parent_path,xml_fn),os.path.join(test_path,xml_fn))
    file_list.remove(move_me)
```

Рисунок 3.10 – Лістинг коду для поділення датасету на три частини

```
# Download data conversion scripts
! wget https://raw.githubusercontent.com/Sasha071201/TensorFlow-Lite-Object-Detection/master/util_scripts/create_csv.py
! wget https://raw.githubusercontent.com/Sasha071201/TensorFlow-Lite-Object-Detection/master/util_scripts/create_tfrecord.py

# Create CSV data files and TFRecord files
!python3 create_csv.py
!python3 create_tfrecord.py --csv_input=images/train_labels.csv
--labelmap=labelmap.txt --image_dir=images/train --output_path=train.tfrecord
!python3 create_tfrecord.py --csv_input=images/validation_labels.csv
--labelmap=labelmap.txt --image_dir=images/validation --output_path=val.tfrecord

train_record_fname = '/kaggle/working/train.tfrecord'
val_record_fname = '/kaggle/working/val.tfrecord'
label_map_pbtxt_fname = '/kaggle/working/labelmap.pbtxt'
```

Рисунок 3.11 – Лістинг коду для конвертації даних та створення файлу pbtxt

Для оптимізації навчання використовувався метод Transfer Learning із завантаженням попередньо натренованих ваг, що скорочувало час навчання і покращувало результати (рис. 3.12). Було налаштовано конфігураційний файл, де вказані шляхи до тренувальних і валідаційних даних, кількість класів, розмір батча та швидкість навчання (рис. 3.13–3.14).

```
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2': {
        'model_name': 'ssd_mobilenet_v2_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz',
    },
    'efficientdet-d0': {
        'model_name': 'efficientdet_d0_coco17_tpu-32',
        'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
        'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
    },
    'ssd-mobilenet-v2-fpnlite-320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
    },
}

model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
%mkdir /kaggle/working/models/mymodel/
%cd /kaggle/working/models/mymodel/

import tarfile
download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint
!wget {download_tar}
tar = tarfile.open(pretrained_checkpoint)
tar.extractall()
tar.close()

download_config =
'https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/' + base_pipeline_file
!wget {download_config}
```

Рисунок 3.12 – Лістинг коду для використання Transfer Learning

```
num_steps = 40000

if chosen_model == 'efficientdet-d0':
  batch_size = 4
else:
  batch_size = 16

pipeline_fname = '/kaggle/working/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/kaggle/working/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())
num_classes = get_num_classes(label_map_pbtxt_fname)
```

Рисунок 3.13 – Лістинг коду для налаштування гіперпараметрів

```
import re

%cd /kaggle/working/models/mymodel
print('writing custom configuration file')

with open(pipeline_fname) as f:
    s = f.read()
with open('pipeline_file.config', 'w') as f:
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")', 'input_path: "{}"'.format(val_record_fname), s)

    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)
    s = re.sub(
        'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

    if chosen_model == 'ssd-mobilenet-v2':
      s = re.sub('learning_rate_base: .8',
                 'learning_rate_base: .08', s)

      s = re.sub('warmup_learning_rate: 0.13333',
                 'warmup_learning_rate: .026666', s)

    if chosen_model == 'efficientdet-d0':
      s = re.sub('learning_rate_base: 8e-2',
                 'learning_rate_base: 8e-3', s)

    if chosen_model == 'efficientdet-d0':
      s = re.sub('keep_aspect_ratio_resizer', 'fixed_shape_resizer', s)
      s = re.sub('pad_to_max_dimension: true', '', s)
      s = re.sub('min_dimension', 'height', s)
      s = re.sub('max_dimension', 'width', s)

    f.write(s)

# Set the path to the custom config file and the directory to store training checkpoints in
pipeline_file = '/kaggle/working/models/mymodel/pipeline_file.config'
model_dir = '/kaggle/working/training/'
```

Рисунок 3.14 – Лістинг коду для налаштування конфігураційного файлу

Навчання моделей проводилося за допомогою тренувального набору даних із застосуванням попередньо налаштованих параметрів. Під час навчання кожна модель коригувала свої ваги після кожної епохи на основі результатів тренувань, а показники втрат використовувалися для відстеження прогресу. Для запобігання перенавчанню використовувався валідаційний набір, який допомагав оцінювати якість моделі на кожному етапі (рис. 3.15).

Тестування проходило на окремій тестовій вибірці зображень харчових продуктів. Результати детекції були збережені у вигляді прогнозів, і для оцінки їхньої точності та ефективності використовувалися спеціальні скрипти. Було

обчислено метрики mAP, що дало змогу оцінити, наскільки добре моделі справляються з розпізнаванням об’єктів на нових даних (рис. 3.16).

```
!python /kaggle/working/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1
```

Рисунок 3.15 – Лістинг коду для навчання моделей

```
%%bash
git clone https://github.com/Cartucho/mAP /kaggle/working/mAP
cd /kaggle/working/mAP
rm input/detection-results/*
rm input/ground-truth/*
rm input/images-optional/*
wget https://raw.githubusercontent.com/Sasha071201/TensorFlow-Lite-Object-Detection/
    master/util_scripts/calculate_map_cartucho.py

!cp /kaggle/working/images/test/* /kaggle/working/mAP/input/images-optional
!mv /kaggle/working/mAP/input/images-optional/*.xml /kaggle/working/mAP/input/ground-truth/

!python /kaggle/working/mAP/scripts/extra/convert_gt_xml.py

import glob
import os
import tensorflow as tf

PATH_TO_IMAGES = '/kaggle/working/images/test'
saved_model_dir = '/kaggle/working/custom_model_saved/saved_model'
PATH_TO_LABELS = '/kaggle/working/labelmap.pbtxt'
PATH_TO_RESULTS = '/kaggle/working/mAP/input/detection-results'
min_conf_threshold = 0.1

image_list = glob.glob(PATH_TO_IMAGES + '/*.jpg') + glob.glob(PATH_TO_IMAGES + '/*.JPG')
+ glob.glob(PATH_TO_IMAGES + '/*.png') + glob.glob(PATH_TO_IMAGES + '/*.bmp')
images_to_test = min(500, len(image_list))

txt_only = True
model = tf.saved_model.load(saved_model_dir)
print(f'Starting inference on {images_to_test} images...')
detect_images(model, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold,
              images_to_test, savepath=PATH_TO_RESULTS, txt_only=txt_only)
print('Finished inferencing!')

%cd /kaggle/working/mAP
!python calculate_map_cartucho.py --labels=/kaggle/working/labelmap.txt
```

Рисунок 3.16 – Лістинг коду для тестування на окремій тестовій вибірці

Після тестування було виконано квантування моделей, що зменшило їх розмір і зробило придатними для використання на мобільних пристроях. Це дозволило оптимізувати модель для роботи за умов обмежених ресурсів без значної втрати точності (рис. 3.17).

```
# Make a directory to store the trained TFLite model
!mkdir /kaggle/working/custom_model_lite
output_directory = '/kaggle/working/custom_model_lite'

# Path to training directory (the conversion script automatically chooses the highest checkpoint file)
last_model_path = '/kaggle/working/training'

!python /kaggle/working/models/research/object_detection/export_tflite_graph_tf2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}
```

```
# Convert exported graph file into TFLite model file
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model('/kaggle/working/custom_model_lite/saved_model')
tflite_model = converter.convert()

with open('/kaggle/working/custom_model_lite/detect.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
image_path = '/kaggle/working/images/train'

jpg_file_list = glob.glob(image_path + '/*.jpg')
JPG_file_list = glob.glob(image_path + '/*.JPG')
png_file_list = glob.glob(image_path + '/*.png')
bmp_file_list = glob.glob(image_path + '/*.bmp')

quant_image_list = jpg_file_list + JPG_file_list + png_file_list + bmp_file_list
```

```
interpreter = Interpreter(model_path=PATH_TO_MODEL)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

import random

def representative_data_gen():
  dataset_list = quant_image_list
  quant_num = 300
  for i in range(quant_num):
    pick_me = random.choice(dataset_list)
    image = tf.io.read_file(pick_me)

    if pick_me.endswith('.jpg') or pick_me.endswith('.JPG'):
      image = tf.io.decode_jpeg(image, channels=3)
    elif pick_me.endswith('.png'):
      image = tf.io.decode_png(image, channels=3)
    elif pick_me.endswith('.bmp'):
      image = tf.io.decode_bmp(image, channels=3)

    image = tf.image.resize(image, [height,width])
    image = tf.cast(image / 255., tf.float32)
    image = tf.expand_dims(image, 0)
    yield [image]

converter = tf.lite.TFLiteConverter.from_saved_model('/kaggle/working/custom_model_lite/saved_model')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.target_spec.supported_types = [tf.int8]
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.float32
tflite_model = converter.convert()

with open('/kaggle/working/custom_model_lite/detect_quant.tflite', 'wb') as f:
  f.write(tflite_model)
```

Рисунок 3.17 – Лістинг коду квантування моделей

Завершальним етапом є порівняння ефективності та продуктивності різних архітектур нейронних мереж.

## 3.4 Аналіз точності та продуктивності розпізнавання харчових продуктів

Аналіз розпізнавання харчових продуктів спрямований на оцінку ефективності моделей розпізнавання в умовах реальних даних, що дозволить визначити, наскільки надійно й точно моделі можуть ідентифікувати харчові продукти у практичних застосунках. Для цього використовуватимуться показники, такі як метрика mAP, кількість правильно ідентифікованих об’єктів, час обробки моделей на основі офіційних бенчмарків TensorFlow, час виконання моделей у середовищі Python-скриптів, а також час виконання в умовах мобільного застосунку для визначення їхньої придатності.

Аналіз ефективності буде проводитися на трьох архітектурах у різних форматах: SavedModel, TFLite та Quantized TFLite. Кожен із цих форматів має свої особливості та призначення. SavedModel є форматом TensorFlow, оптимізованим для роботи у середовищах розробки та серверних застосунках, де забезпечується висока точність, проте обробка може бути більш ресурсоємною. TFLite спеціально розроблений для мобільних пристроїв і дозволяє зменшити навантаження на апаратне забезпечення, підтримуючи швидке виконання моделі з дещо меншою точністю. Quantized TFLite застосовує техніку квантування, яка ще більше знижує вимоги до ресурсів, що робить цей формат корисним для мобільних та вбудованих пристроїв, але з потенційною втратою точності.

Метрика mAP – одна з основних метрик для оцінки точності моделей. Під час тестування моделі виконують виведення на наборі тестових зображень, де результати передбачених класів та розташування об’єктів порівнюються з фактичними даними, тобто правильними класами і місцезнаходженням об’єктів. Значення mAP розраховується на основі точності прогнозів. Чим вище значення mAP, тим краща здатність моделей розпізнавати об’єкти на зображеннях.

Для розрахунку mAP використовували калькулятор з відкритим вихідним кодом від користувача GitHub Catchuro. Створили скрипт для взаємодії з цим калькулятором та розрахували метрику COCO для mAP @ 0,50:0,95 на тестовому наборі з 100 зображень. На рисунку 3.18 показано приклад роботи скрипту для архітектури SSD-MobileNetV2-320.

```
Calculating mAP at 0.50 IoU threshold...
98.21% = apple AP
99.72% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 99.48%
Calculating mAP at 0.55 IoU threshold...
98.21% = apple AP
99.72% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 99.48%
Calculating mAP at 0.60 IoU threshold...
98.21% = apple AP
99.72% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 99.48%
Calculating mAP at 0.65 IoU threshold...
98.21% = apple AP
99.72% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 99.48%
Calculating mAP at 0.70 IoU threshold...
94.51% = apple AP
99.72% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 98.56%
Calculating mAP at 0.75 IoU threshold...
94.51% = apple AP
90.10% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 96.15%
Calculating mAP at 0.80 IoU threshold...
94.51% = apple AP
90.10% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 96.15%
Calculating mAP at 0.85 IoU threshold...
92.76% = apple AP
78.98% = banana AP
100.00% = mandarin AP
100.00% = orange AP
mAP = 92.93%
Calculating mAP at 0.90 IoU threshold...
79.20% = apple AP
68.99% = banana AP
68.75% = mandarin AP
78.77% = orange AP
mAP = 73.93%
Calculating mAP at 0.95 IoU threshold...
29.21% = apple AP
39.27% = banana AP
25.00% = mandarin AP
46.00% = orange AP
mAP = 34.87%

*mAP Results*

Class  Average mAP @ 0.5:0.95
---------------------------------------
apple  87.75%
banana  86.60%
orange  92.48%
mandarin  89.38%

Overall  89.05%
```

Рисунок 3.18 – Приклад роботи скрипту для архітектури SSD-MobileNetV2-320

Хоча mAP є відмінною метрикою для кількісного порівняння моделей, вона не завжди надає інтуїтивне розуміння того, наскільки добре модель виявляє об’єкти на зображеннях. Для отримання більш якісної оцінки точності кожної моделі було також перевірено загальну кількість об’єктів, правильно виявлених із достовірністю не менше 80% у вибірці з 100 зображень харчових

продуктів. На рисунку 3.19 продемонстровано розпізнавання харчових продуктів за допомогою архітектури SSD-MobileNetV2-320.

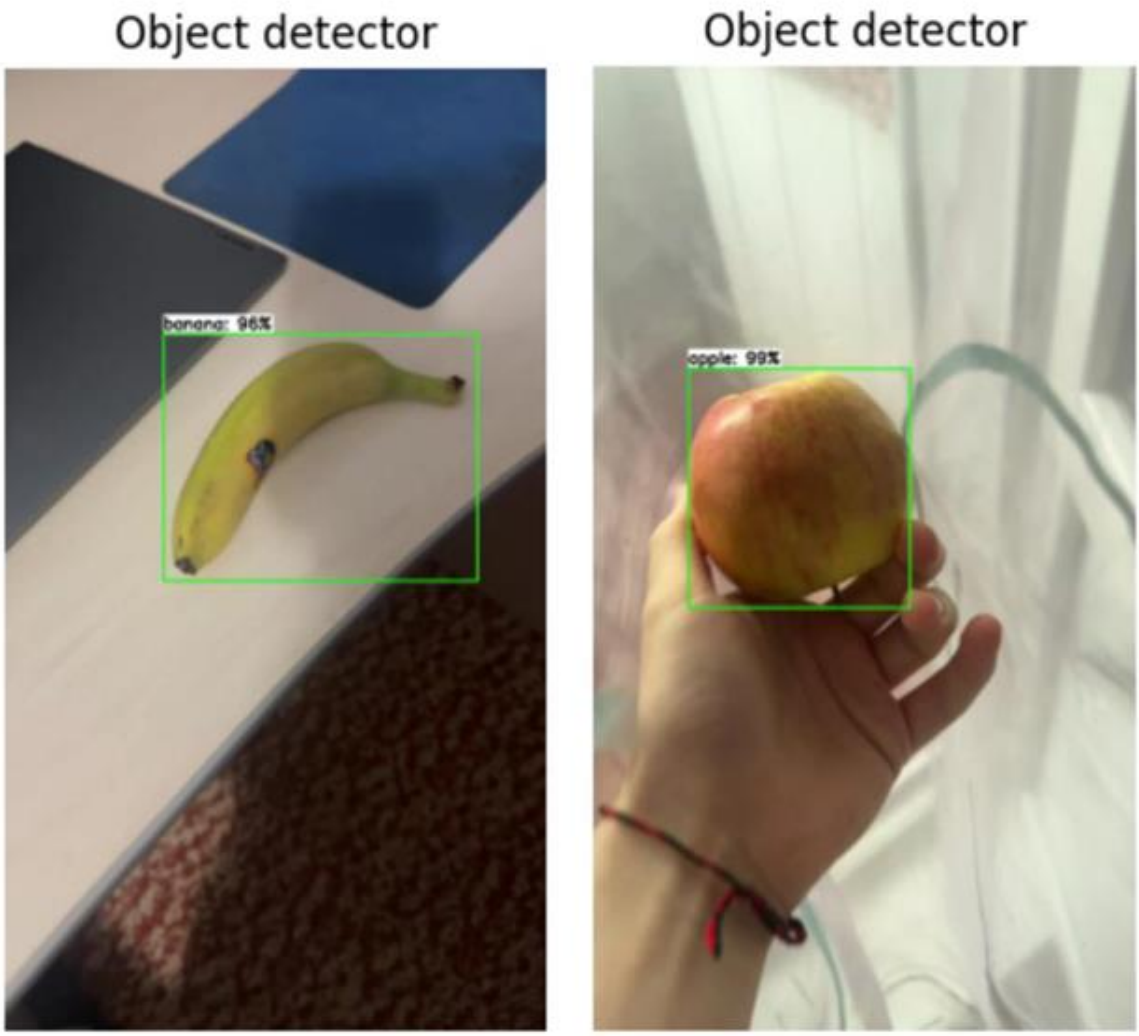


Рисунок 3.19 – Розпізнавання харчових продуктів на архітектурі SSD-MobileNetV2-320

Метрика COCO для mAP @ 0,50:0,95 дозволяє об’єктивно порівнювати різні архітектури. Для кращого розуміння того, наскільки моделі точно розпізнають об’єкти, було також проаналізовано кількість правильно виявлених об’єктів у тестовому наборі даних. Такий підхід дозволяє оцінити не тільки середню точність моделей, але й їхню продуктивність у реальних умовах. В таблиці 3.1 показано оцінку точності COCO (mAP @ IoU 0,5:0,95)

для кожної моделі та загальну кількість правильно позначених об’єктів у наборі даних.

Таблиця 3.1 – Оцінка точності моделей та загальна кількість правильно позначених об’єктів у наборі даних

| Модель | Оцінка точності | Кількість правильно позначених об’єктів (з 100) |
|---|---|---|
| 1 | 2 | 3 |
| SSD-MobileNetV2-320 (SavedModel) | 78,62% | 76 |
| SSD-MobileNetV2-320 (TFLite) | 78,01% | 75 |
| SSD-MobileNetV2-320 (Quantized TFLite) | 59,80% | 60 |
| SSD-MobileNetV2-FPNLite320 (SavedModel) | 88,01% | 85 |
| SSD-MobileNetV2-FPNLite320 (TFLite) | 84,93% | 82 |
| SSD-MobileNetV2-FPNLite320 (Quantized TFLite) | 77,98% | 74 |
| EfficientDet-D0 (SavedModel) | 57,03% | 55 |
| EfficientDet-D0 (TFLite) | 55,67% | 53 |

На рисунку 3.20 наведено порівняльну діаграму результатів тестування моделей, де оцінюється точність моделей у відсотках та кількість правильно позначених об’єктів із набору тестових зображень. На осі $X$ представлені різні архітектури моделей, а на осі $Y$ – значення точності і кількості правильно класифікованих об’єктів. Такий підхід дає змогу візуально оцінити ефективність кожної моделі на основі двох ключових показників: оцінки точності та кількості правильно позначених об’єктів.

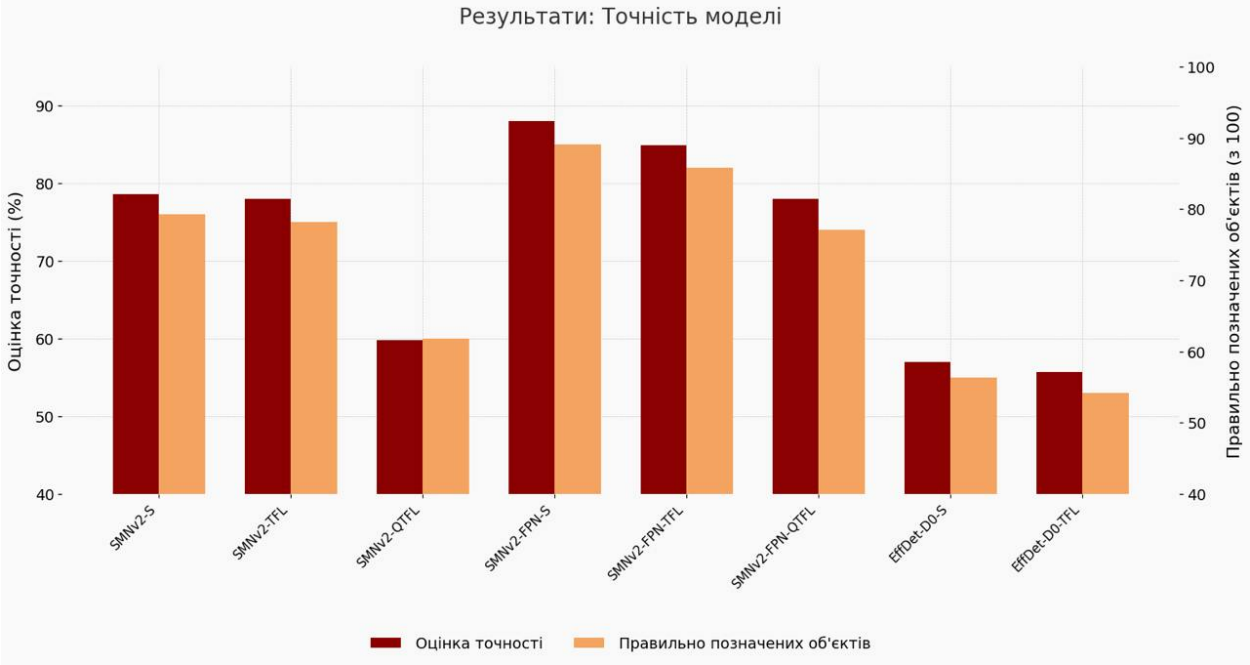


Рисунок 3.20 – Діаграма оцінки точності моделей та загальної кількості правильно позначених об’єктів

Тестування моделей також включало вимірювання середньої швидкості інференсу (average inference speed). Середня швидкість інференсу є показником продуктивності під час виконання завдань в умовах постійного навантаження. Вимірювання цього параметра дозволяє оцінити, наскільки ефективно модель виконує свої функції у тривалих сценаріях використання, таких як обробка потоків зображень або відео в реальному часі.

Було проведено аналіз середньої швидкості інференсу для моделей на основі офіційних бенчмарків TensorFlow (рис. 3.21), середовища Python-скриптів (рис. 3.22), а також у контексті мобільного застосунку (рис. 3.23). Порівняльний аналіз здійснено для кожного формату моделі (SavedModel, TFLite, Quantized TFLite) з метою виявлення найефективнішого варіанту для використання в мобільних застосунках. Це допоможе визначити, наскільки різні формати моделей впливають на продуктивність та використання ресурсів в умовах реального застосування. Крім того, виконано порівняння середньої швидкості інференсу для різних архітектур, що дозволить обрати оптимальну

модель з точки зору швидкості та точності для обробки зображень у мобільному середовищі.


Рисунок 3.21 – Результат швидкості інференсу на основі бенчмарків TensorFlow

```
Генерація випадкових даних для EfficientDet-D0-TFlite з формою [  1 512 512   3] з Input mean 127.5 та Input std 127.5

Бенчмаркінг моделі EfficientDet-D0-TFlite
Затримка запуску: 1545.26 мс
Inference:
  Кількість прогонів: 1000
  Середнє: 77.21 мс
  Мінімум: 69.58 мс
  Максимум: 137.10 мс
  Стандартне відхилення: 4.52 мс
Warmup:
  Кількість прогонів: 50
  Середнє: 76.87 мс
  Мінімум: 70.12 мс
  Максимум: 103.80 мс
  Стандартне відхилення: 6.43 мс
```

Рисунок 3.22 – Результат швидкості інференсу у середовищі Python-скриптів

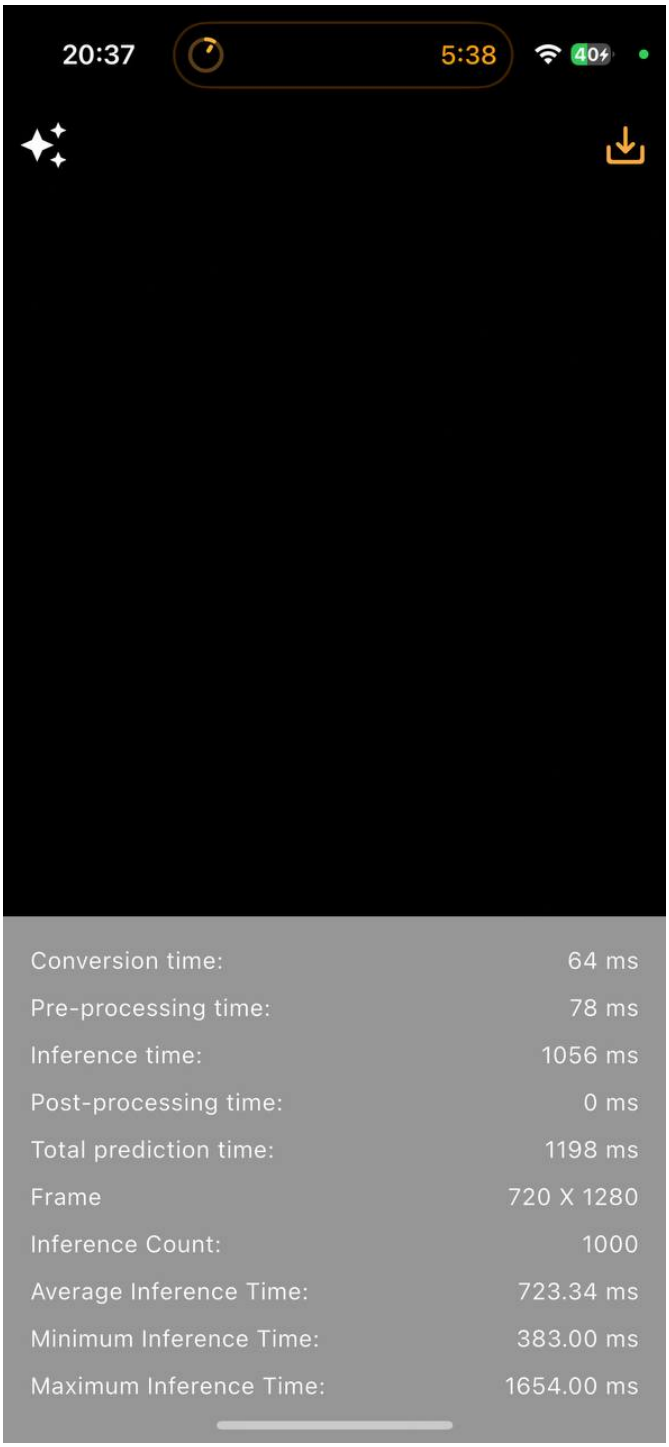


Рисунок 3.23 – Результат швидкості інференсу для мобільного застосунку

Таким чином, результати тестування середньої швидкості інференсу для кожного формату та архітектури, представлені в таблицях 3.2–3.4 та на рисунках 3.24–3.26, дозволять прийняти обґрунтоване рішення щодо оптимального вибору моделі.

Таблиця 3.2 – Середнє значення часу інференсу моделей у середовищі Python

| Модель | Час інференсу (Python) |
|---|---|
| 1 | 2 |
| SSD-MobileNetV2-320 (SavedModel) | 23,33 мс |
| SSD-MobileNetV2-320 (TFLite) | 7,90 мс |
| SSD-MobileNetV2-320 (Quantized TFLite) | 3,83 мс |
| SSD-MobileNetV2-FPNLite320 (SavedModel) | 39,62 мс |
| SSD-MobileNetV2-FPNLite320 (TFLite) | 11,68 мс |
| SSD-MobileNetV2-FPNLite320 (Quantized TFLite) | 5,53 мс |
| EfficientDet-D0 (SavedModel) | 158,8 мс |
| EfficientDet-D0 (TFLite) | 79,40 мс |

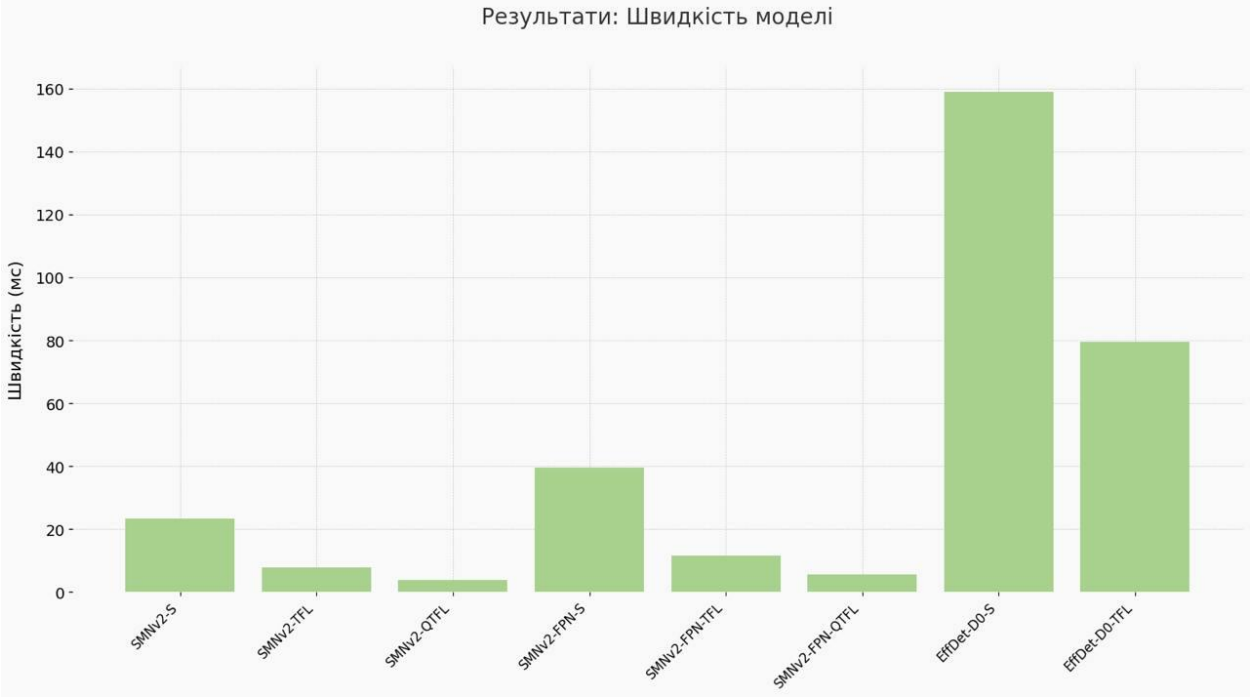


Рисунок 3.24 – Діаграма середнього часу інференсу моделей у середовищі Python

Отже, найшвидшою є модель SSD-MobileNetV2-320 (Quantized TFLite) із часом 3,83 мс, тоді як модель EfficientDet-D0 (SavedModel) показує

найбільший час інференсу – 158,8 мс. Загалом, квантовані версії моделей значно перевершують інші формати за швидкістю, забезпечуючи оптимальну продуктивність для реального застосування.

У таблиці 3.3 та 3.4 відсутні результати для формату SavedModel, оскільки час інференсу моделей оцінювався на мобільному застосунку, де формат SavedModel не використовується. SavedModel призначений переважно для серверного середовища, тоді як для мобільних платформ рекомендовано застосовувати формат TFLite, який забезпечує кращу продуктивність і сумісність із мобільними процесорами (CPU, GPU) та фреймворками, такими як CoreML на iOS.

Таблиця 3.3 – Середнє значення часу інференсу моделей на основі бенчмарків TensorFlow

| Модель | Час інференсу (CPU) | Час інференсу (GPU+CPU) | Час інференсу (CoreML + CPU) | Час інференсу (CoreML + GPU + CPU) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| SSD-MobileNetV2-320 (TFLite) | 9,61 мс | 10,58 мс | 2,34 мс | 5,93 мс |
| SSD-MobileNetV2-320 (Quantized TFLite) | 4,23 мс | 11,65 мс | 4,24 мс | 11,61 мс |
| SSD-MobileNetV2-FPNLite320 (TFLite) | 13,91 мс | 11,08 мс | 5,72 мс | 7,45 мс |
| SSD-MobileNetV2-FPNLite320 (Quantized TFLite) | 6,41 мс | 10,56 мс | 6,35 мс | 10,48 мс |
| EfficientDet-D0 (TFLite) | 151,29 мс | – | – | – |

Відсутність результатів для EfficientDet-D0 (TFLite) у інших налаштуваннях інференсу може бути спричинена обмеженою підтримкою цієї моделі в TensorFlow Lite для GPU та CoreML, технічними проблемами при конвертації або оптимізації моделі для цих платформ, а також складністю самої моделі, що ускладнює її використання з альтернативними апаратними прискорювачами.

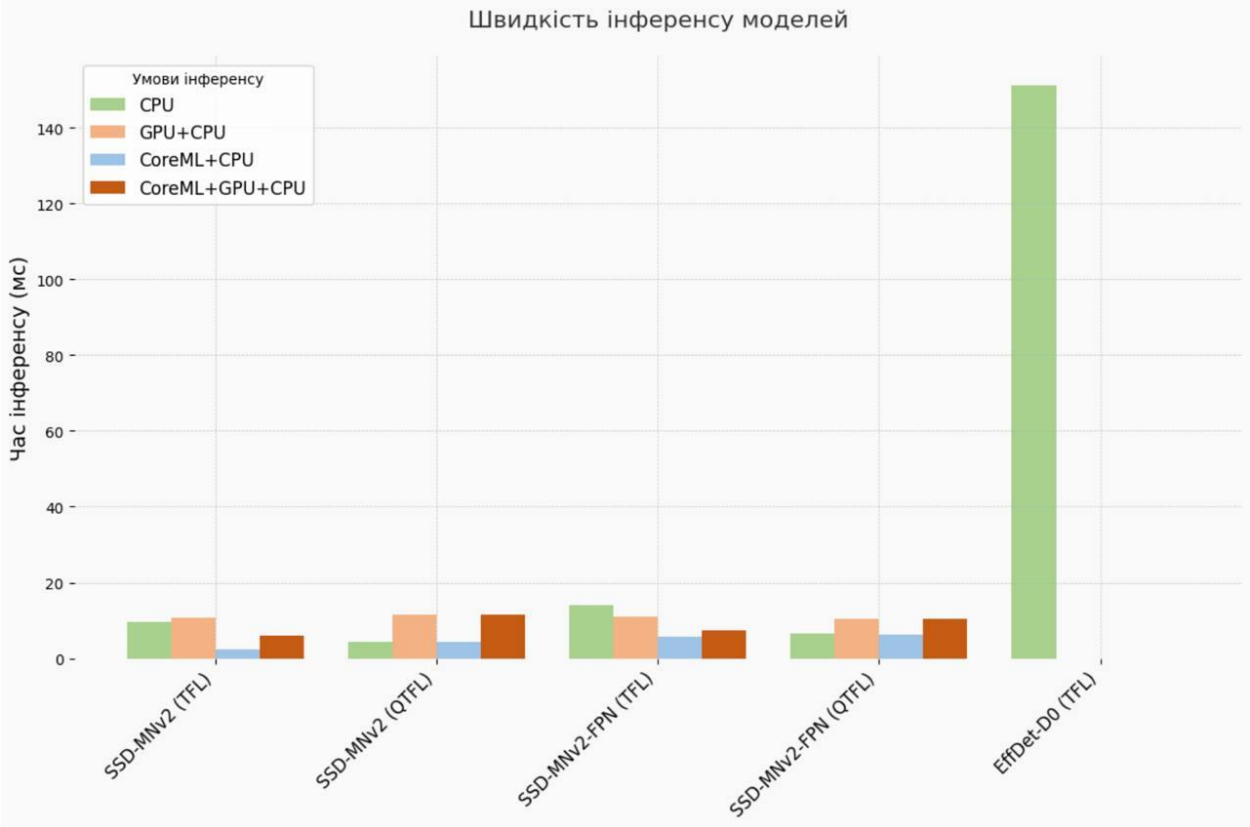


Рисунок 3.25 – Діаграма середнього часу інференсу моделей на основі бенчмарків TensorFlow

Отже, використання використання CoreML значно скорочує час інференсу моделей порівняно з TensorFlow Lite на CPU. Квантування моделей, зокрема SSD-MobileNetV2-320, суттєво знижує час обробки на CPU. Використання GPU разом із CPU не завжди покращує продуктивність у TFLite, тоді як CoreML ефективно використовує GPU для швидшого інференсу. Варто також відзначити, що більш складні моделі, як EfficientDet-D0, мають значно більший час інференсу на CPU.

Таблиця 3.4 – Середнє значення часу інференсу моделей для мобільного застосунку

| Модель | Час інференсу (CPU) | Час інференсу (GPU+CPU) | Час інференсу (CoreML + CPU) | Час інференсу (CoreML + GPU + CPU) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| SSD-MobileNetV2-320 (TFLite) | 224,82 мс | 158,61 мс | 151,76 мс | 196,75 мс |
| SSD-MobileNetV2-320 (Quantized TFLite) | 85,01 мс | 78,69 мс | 74,41 мс | 76,79 мс |
| SSD-MobileNetV2-FPNLite320 (TFLite) | 295,93 мс | – | 147,68 мс | – |
| SSD-MobileNetV2-FPNLite320 (Quantized TFLite) | 86,84 мс | – | 81,08 мс | – |
| EfficientDet-D0 (TFLite) | 723,34 мс | – | – | – |

Відсутність результатів для моделей SSD-MobileNetV2-FPNLite320 (TFLite), SSD-MobileNetV2-FPNLite320 (Quantized TFLite) та EfficientDet-D0 (TFLite) у деяких конфігураціях інференсу може бути пов’язана з тим, що моделі можуть бути несумісними з певними апаратними прискорювачами, такими як GPU, через специфічні шари або операції, які не підтримуються.

Отже, найкращі результати показала модель SSD-MobileNetV2-320 (Quantized TFLite) у комбінації з CoreML та CPU, забезпечуючи найнижчий час інференсу – 74,41 мс. Середні показники демонструють стандартні моделі SSD-MobileNetV2-320 з різними конфігураціями, де час інференсу

коливається від приблизно 150 мс до 224 мс. Найгірші результати спостерігаються у моделі EfficientDet-D0 (TFLite), яка має суттєво вищий час інференсу – 723,34 мс на CPU.

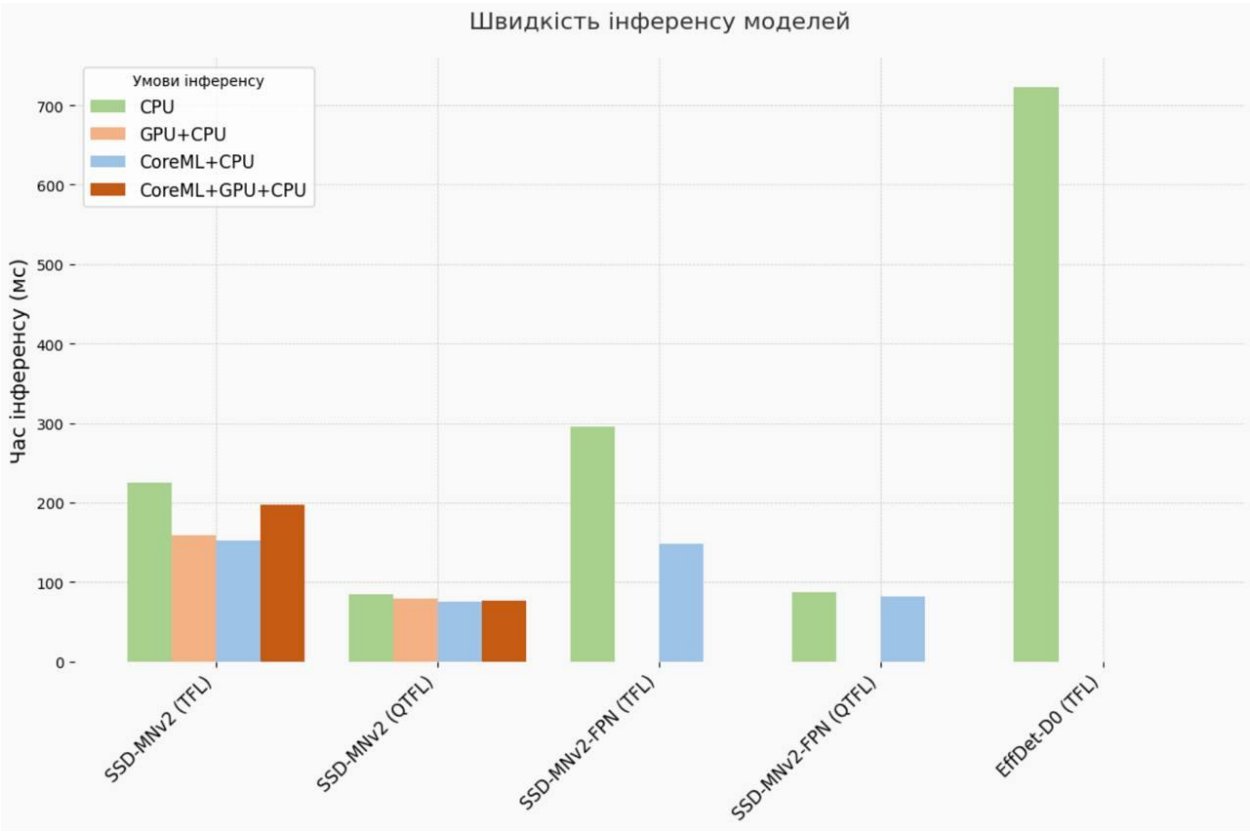


Рисунок 3.26 – Діаграма середнього часу інференсу моделей для мобільного застосунку

Таким чином, для оптимальної продуктивності рекомендується використовувати квантовані моделі в поєднанні з CoreML, тоді як складніші моделі, такі як EfficientDet-D0, можуть бути не ефективними через їх високі обчислювальні витрати.

На основі проведеного аналізу було виконано порівняння восьми моделей виявлення об’єктів. Для цього використовувались показники, такі як метрика точності (mAP), кількість правильно ідентифікованих об’єктів, час обробки моделей на основі офіційних бенчмарків TensorFlow, час виконання моделей у середовищі Python-скриптів, а також час виконання в умовах мобільного застосунку для визначення їхньої придатності. Під час ранжування

особлива увага приділялася тому, що невеликі відмінності в точності виявилися більш значущими, ніж незначні відхилення у швидкості роботи.

Результати аналізу показали, що серед протестованих моделей найкращий баланс між швидкістю і точністю був продемонстрований моделлю SSD-MobileNetV2-FPNLite320 (SavedModel). Ця модель зайняла перше місце завдяки найвищій точності 88,01% та найбільшій кількості правильно ідентифікованих об’єктів (85 з 100). Хоча її час інференсу в середовищі Python становив 39,62 мс, що є більшим порівняно з іншими моделями, високий рівень точності робить її оптимальним вибором для застосувань, де якість виявлення є критичною.

На другому місці опинилася модель SSD-MobileNetV2-FPNLite320 (TFLite), яка продемонструвала високу точність 84,93% та правильно ідентифікувала 82 з 100 об’єктів. Її перевагою є швидший час інференсу в середовищі Python (11,68 мс) та на мобільних пристроях (147,68 мс на CoreML + CPU). Це робить цю модель придатною для додатків, де необхідно поєднати високу точність із швидкістю обробки.

Третє місце дісталося моделі SSD-MobileNetV2-320 (SavedModel) з точністю 78,62% та 76 правильно ідентифікованими об’єктами. Час інференсу в середовищі Python становив 23,33 мс. Ця модель є хорошим компромісом між точністю та швидкістю і може бути використана в різних застосуваннях, де ці показники є важливими.

На четвертому місці знаходиться модель SSD-MobileNetV2-320 (TFLite), яка має точність 78,01% та правильно ідентифікувала 75 з 100 об’єктів. Її основною перевагою є швидкий час інференсу в середовищі Python (7,90 мс) та на мобільних пристроях (151,76 мс на CoreML + CPU). Це робить її ефективним вибором для застосувань з обмеженими ресурсами, де швидкість має пріоритет.

П’яте місце зайняла модель SSD-MobileNetV2-FPNLite320 (Quantized TFLite) з точністю 77,98% та 74 правильно ідентифікованими об’єктами. Її час інференсу в середовищі Python складає лише 5,53 мс, що є одним із

найшвидших показників. Ця модель може бути корисною у випадках, коли необхідна висока швидкість обробки з мінімальними втратами в точності.

Шосте місце посіла модель SSD-MobileNetV2-320 (Quantized TFLite) з точністю 59,80% та 60 правильно ідентифікованими об’єктами. Вона відзначається найшвидшим часом інференсу в середовищі Python (3,83 мс) та на мобільних пристроях (74,41 мс на CoreML + CPU). Однак, значне зниження точності порівняно з іншими моделями обмежує її застосування у випадках, де важлива якість виявлення.

На сьомому місці опинилася модель EfficientDet-D0 (SavedModel) з точністю 57,03% та 55 правильно ідентифікованими об’єктами. Її час інференсу в середовищі Python становив 158,8 мс, що є найбільшим серед усіх протестованих моделей. Низька точність та повільна швидкість обробки роблять її непридатною для більшості практичних застосувань.

Восьме місце зайняла модель EfficientDet-D0 (TFLite) з найнижчою точністю 55,67% та 53 правильно ідентифікованими об’єктами. Час інференсу в середовищі Python становив 79,40 мс, а на мобільному застосунку – 723,34 мс на CPU. Ці показники свідчать про те, що модель не відповідає вимогам швидкості та точності для використання в реальних умовах.

Отже, модель SSD-MobileNetV2-FPNLite320 (SavedModel) показала найкращий баланс між точністю та швидкістю, особливо у застосуваннях, де пріоритетом є висока якість виявлення об’єктів. Моделі EfficientDet-D0, як у SavedModel, так і у TFLite версіях, показали найгірші результати і не рекомендуються для використання в реальних умовах через низьку точність та повільну швидкість обробки. Вибір конкретної моделі залежить від вимог проєкту, але загалом моделі серії SSD-MobileNetV2-FPNLite320 демонструють найкращі показники для практичного застосування.

Після вибору оптимальної моделі для ефективного використання розглянемо її практичне застосування для розпізнавання харчових продуктів в реальних умовах. На рисунку 3.27 представлено мобільний застосунок, де обрана модель успішно розпізнає продукти, демонструючи точність з

відповідними ймовірностями, що підтверджує її ефективність в умовах реального часу.

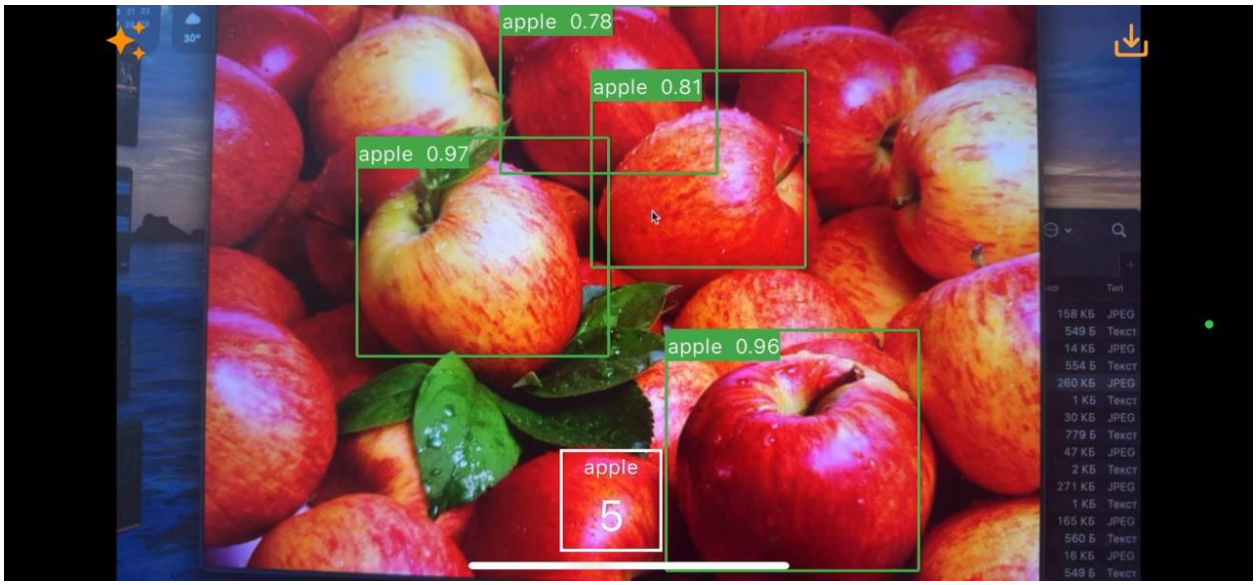


Рисунок 3.27 – Розпізнавання продуктів в мобільному застосунку

# ВИСНОВКИ

В рамках даної кваліфікаційної роботи було досліджено та розроблено метод розпізнавання харчових продуктів з використанням архітектур нейронних мереж. Основна мета роботи полягала в аналізі ефективності і оптимізації моделей для розпізнавання харчових продуктів на основі власних наборів даних. Окрім цього, був розроблений мобільний застосунок, де також проводився аналіз архітектур для визначення найбільш оптимального рішення для мобільного середовища.

Методологія дослідження включала налаштування трьох архітектур (SSD-MobileNetV2-320, EfficientDet-D0, SSD-MobileNetV2-FPNLite320), підготовку та розмітку даних, оптимізацію гіперпараметрів (розмір пакету, кількість кроків, швидкість навчання). Ефективність оцінювалася за метрикою mAP, кількістю правильно ідентифікованих об’єктів, часом обробки моделей на основі офіційних бенчмарків TensorFlow, часом виконання моделей у середовищі Python-скриптів, а також часом виконання в умовах мобільного застосунку для визначення їхньої придатності. Також проводилася візуалізація результатів оцінювання ефективності моделей у реальних умовах.

Дослідження показало, що для мобільних застосунків найкращим вибором є моделі серії SSD-MobileNetV2-FPNLite320, які забезпечують оптимальний баланс між точністю та швидкістю виконання. Зокрема, модель SSD-MobileNetV2-FPNLite320 (TFLite) демонструє високу якість виявлення об’єктів при прийнятній швидкості, що робить її особливо придатною для мобільних платформ. Квантована версія цієї моделі додатково прискорює процес обробки, зберігаючи точність на достатньому рівні для багатьох реальних сценаріїв.

Інші моделі, такі як EfficientDet-D0 у форматах SavedModel та TFLite, показали найнижчу ефективність через низьку точність і повільну швидкість роботи, що робить їх непридатними для застосувань з високими вимогами до

продуктивності. Модель SSD-MobileNetV2-320 має високу швидкість обробки, проте її точність нижча порівняно з моделлю SSD-MobileNetV2-FPNLite320, що робить її менш придатною для застосунків, де пріоритетом є точність.

Практична значущість дослідження полягає у визначенні оптимальних архітектур нейронних мереж для мобільних застосунків, орієнтованих на розпізнавання харчових продуктів, що дозволяє розробникам обирати моделі з найкращим балансом між точністю та швидкістю. Крім того, створений власний датасет харчових продуктів, який може використовуватися іншими дослідниками для подальших розробок і тестування моделей у сфері комп'ютерного зору.

Наукова новизна дослідження полягає у розробці методу розпізнавання харчових продуктів з використанням сучасних архітектур нейронних мереж, адаптованих до мобільного середовища. У ході роботи було створено унікальний датасет харчових продуктів, який може стати корисним для інших досліджень, та мобільний застосунок, що дозволив додатково протестувати продуктивність моделей на мобільних пристроях. Порівняння моделей у різних форматах (SavedModel, TFLite, квантовані версії) з оцінкою за метриками точності (mAP) та швидкості роботи допомогло визначити оптимальні архітектури для мобільного застосування.

Подальші перспективи дослідження включають розширення датасету з харчовими продуктами для охоплення більшої різноманітності категорій та умов зйомки, що підвищить точність моделей у реальних умовах.

Також перспективним є дослідження інших сучасних архітектур нейронних мереж, які можуть забезпечити ще вищу точність розпізнавання, зберігаючи оптимальний баланс між продуктивністю та ресурсомісткістю, особливо для мобільних та вбудованих пристроїв.

Результати дослідження апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [50].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Путятін, Є. П., Гороховатський, В. О., & Матат, О. О. (2006). Методи та алгоритми комп'ютерного зору: навч. посібник.

2. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

3. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

4. Тітов, С. В., Тітова, О. В., & Чорна, О. С. (2023). Метод знаходження апроксимацій приблизних множин з використанням систем числення. *Системи обробки інформації*, *2*(173), 58–62.

5. Гороховатський В.О., Творошенко І.С. (2022) Аналіз багатовимірних даних за описом у формі множини компонент: монографія. Харків: ХНУРЕ, 124 с.

6. Токарчук, Д. О., & Майданюк, В. П. (2023). *Застосування служби розпізнавання зображень Google Cloud Vision у процесі розробки програмного забезпечення* (Doctoral dissertation, НІКО/КЗВО «Вінницька академія безперервної освіти»).

7. Melnyk, V., Melnyk, K., & Koptiuk, J. (2019). Дослідження методів розпізнавання зображень на основі нейронних мереж. *COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION*, (35), 161-165.

8. Здітовецький, Ю. С., Бісікало, О. В., & Іванов, Ю. Ю. (2023). Інтелектуальна інформаційна система розпізнавання та аналізу складу продуктів харчування. *Вісник Вінницького політехнічного інституту.№ 2: 66-71*.

9. Aghaee, F., Fazl-Ersi, E., & Noori, H. (2024). MDSSD-MobV2: An embedded deconvolutional multispectral pedestrian detection based on SSD-MobileNetV2-320. *Multimedia Tools and Applications*, *83*(15), 43801-43829.

10. Muna, N., Ningsih, N., Syahroni, N., Syamlan, A. M., & Larasati, V. (2024). Implementasi Algoritma EfficientDet-D0 dan SSD-MobileNet-V2 FPNLite untuk Sistem Deteksi Gulma. *The Indonesian Journal of Computer Science*, *13*(1).

11. Muslim, H. A., Oktavianto, H., Widodo, R. T., Purwantini, E., & Sesulihatien, W. T. (2024, August). Implementation of Chicken Eggs Fertility Detection Device Using SSD MobileNet-V2 FPNLite. In *2024 International Electronics Symposium (IES)* (pp. 171-177). IEEE.

12. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С. 5-12.

13. Piran, Z., & Nitzan, M. (2024). SiFT: uncovering hidden biological processes by probabilistic filtering of single-cell data. *Nature Communications*, *15*(1), 760.

14. Zhang, J., & Xiu, Y. (2024). Image stitching based on human visual system and SIFT algorithm. The Visual Computer, 40(1), 427-439.

15. Arooj, S., Altaf, S., Ahmad, S., Mahmoud, H., & Mohamed, A. S. N. (2024). Enhancing sign language recognition using CNN and SIFT: A case study on Pakistan sign language. *Journal of King Saud University-Computer and Information Sciences*, *36*(2), 101934.

16. Mishra, M., Jain, N. K., & Kumar, A. (2024). Effective dense detection method for tampering detection on high quality images using PH-SIFT and RANSAC algorithm. *Multimedia Tools and Applications*, 1-26.

17. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, *Telecommunications and Radio Engineering*, 79(10), pp. 855-863.

18. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Cluster representation of the structural description of images for effective classification, *Computers, Materials & Continua*, 73(3), pp. 6069-6084.

19. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, *IEEE Access*, 9, pp. 92964-92973.

20. Syzonenko, O. D., & Bozhukha, L. M. (2023). Методи локалізації об'єктів на основі зображень із використанням комбінації алгоритмів та багатопоточної зв'язки Faster R-CNN. *Актуальні проблеми автоматизації та інформаційних технологій*, *27*.

21. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40-48.

22. Wang, L., Shoulin, Y., Alyami, H., Laghari, A. A., Rashid, M., Almotiri, J., ... & Alturise, F. (2024). A novel deep learning-based single shot multibox detector model for object detection in optical remote sensing images.

23. Koga, S., Hamamoto, K., Lu, H., & Nakatoh, Y. (2024). Optimizing food sample handling and placement pattern recognition with YOLO: Advanced techniques in robotic object detection. *Cognitive Robotics*.

24. Ghosh, T., Han, Y., Raju, V., Hossain, D., McCrory, M. A., Higgins, J., ... & Sazonov, E. (2024). Integrated image and sensor-based food intake detection in free-living. *Scientific Reports, 14*(1), 1665.

25. Antunes, S. N., Okano, M. T., Nääs, I. D. A., Lopes, W. A. C., Aguiar, F. P. L., Vendrametto, O., ... & Fernandes, M. E. (2024). Model development for identifying aromatic herbs using object detection algorithm. *AgriEngineering, 6*(3), 1924-1936.

26. Phadke, R., Chaurasia, A., Raj, H., & Kumari, N. (2024, April). Efficient food image segmentation using YOLOv5: A step towards automated food recognition. In *2024 Third International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)* (pp. 1-7). IEEE.

27. Zhu, L., Spachos, P., Pensini, E., & Plataniotis, K. N. (2021). Deep learning and machine vision for food processing: A survey. *Current Research in Food Science*, *4*, 233-249.

28. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, *Computers, Materials & Continua*, vol. 80, no. 2, pp. 3085-3106.

29. Ma, W., Wang, X., & Yu, J. (2020). A lightweight feature fusion single shot multibox detector for garbage detection. *IEEE Access*, *8*, 188577-188586.

30. Guo, S., Liu, Y., Ni, Y., & Ni, W. (2021). Lightweight SSD: Real-time Lightweight Single Shot Detector for Mobile Devices. In *VISIGRAPP (5: VISAPP)* (pp. 25-35).

31. Jia, J., Fu, M., Liu, X., & Zheng, B. (2022). Underwater object detection based on improved efficientdet. *Remote Sensing*, *14*(18), 4487.

32. Li, X., Liu, J., Tang, Z., Han, B., & Wu, Z. (2024). MEDMCN: a novel multi-modal EfficientDet with multi-scale CapsNet for object detection. *The Journal of Supercomputing*, 1-28.

33. Тітов, С. В., Тітова, О. В., & Чорна, О. С. (2022). Опис нескоротних наборів ознак в приблизних множинах з використанням систем числення. *Збірник наукових праць Харківського національного університету Повітряних Сил*, *1*(71), 106–110.

34. Ma, Y., Chen, S., Ermon, S., & Lobell, D. B. (2024). Transfer learning in environmental remote sensing. *Remote Sensing of Environment*, *301*, 113924.

35. Прус, Б. В., & Ракитянська, Г. Б. (2023). *Методи та програмні засоби трансферного навчання для агрегування медіа контенту у мобільних додатках* (Doctoral dissertation, ВНТУ).

36. Dhaif, Z. S., & El Abbadi, N. K. (2024). Road Signs Detection Using SSD MobileNetV2. *Karbala International Journal of Modern Science*, *10*(4), 1.

37. Aghaee, F., Fazl-Ersi, E., & Noori, H. (2024). MDSSD-MobV2: An embedded deconvolutional multispectral pedestrian detection based on SSD-MobileNetV2-320. *Multimedia Tools and Applications*, *83*(15), 43801-43829.

38. Chimakurthi, A. K., & Chimakurthy, L. P. (2024). Enhancing Real-Time Object Detection on Low-End Devices: A Comparative Study of Performance of YOLOv4 and SSD MobileNetv2.

39. Muna, N., Ningsih, N., Syahroni, N., Syamlan, A. M., & Larasati, V. (2024). Implementasi Algoritma EfficientDet-D0 dan SSD-MobileNet-V2 FPNLite untuk Sistem Deteksi Gulma. *The Indonesian Journal of Computer Science*, *13*(1).

40. Zhuang, X., Li, D., Wang, Y., & Li, K. (2024). Military target detection method based on EfficientDet and Generative Adversarial Network. *Engineering Applications of Artificial Intelligence*, *132*, 107896.

41. Muslim, H. A., Oktavianto, H., Widodo, R. T., Purwantini, E., & Sesulihatien, W. T. (2024, August). Implementation of Chicken Eggs Fertility Detection Device Using SSD MobileNet-V2 FPNLite. In *2024 International Electronics Symposium (IES)* (pp. 171-177). IEEE.

42. Abhijith, E., Thomas, A. M. C., Saji, I., Chiramal, J. J., & Jose, C. (2024, May). SEGRA-BOT: Autonomous Waste Pickup and Sorting Robot. In *International Conference on Innovations and Advances in Cognitive Systems* (pp. 139-149). Cham: Springer Nature Switzerland.

43. Minderer, M., Gritsenko, A., & Houlsby, N. (2024). Scaling open-vocabulary object detection. *Advances in Neural Information Processing Systems*, *36*.

44. Tian, J., Jin, Q., Wang, Y., Yang, J., Zhang, S., & Sun, D. (2024). Performance analysis of deep learning-based object detection algorithms on COCO benchmark: a comparative study. *Journal of Engineering and Applied Science*, *71*(1), 76.

45. Su, K., Cao, L., Zhao, B., Li, N., Wu, D., & Han, X. (2024). N-IoU: better IoU-based bounding box regression loss for object detection. *Neural Computing and Applications*, *36*(6), 3049-3063.

46. Xie, X., Cheng, G., Li, Q., Miao, S., Li, K., & Han, J. (2024). Fewer is more: Efficient object detection in large aerial images. *Science China Information Sciences*, *67*(1), 112106.

47. Landau, R. H., Páez, M. J., & Bordeianu, C. C. (2024). *Computational physics: Problem solving with Python*. John Wiley & Sons.

48. Samuel, S., & Mietchen, D. (2024). Computational reproducibility of Jupyter notebooks from biomedical publications. *GigaScience*, *13*, giad113.

49. Tai-Sung, H., & Suyoung, B. (2023). A Comparative Analysis of the Pre-Processing in the Kaggle Titanic Competition. *Journal of the Korea Society of Computer and Information*, *28*(3), 17-24.

50. Стрельцов, О. А. (2024). Застосування глибокого навчання до виявлення об’єктів.