МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ЛОКАЛІЗАЦІЯ ОБ'ЄКТІВ ЗОБРАЖЕННЯ ЗАСОБАМИ МОВИ ПРОГРАМУВАННЯ PYTHON»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.В. Левченко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Левченку Денису Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Локалізація об’єктів зображення засобами мови програмування Python

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 30.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Локалізація об’єктів зображення засобами мови програмування Python.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.05.2023	
2.	Збір вихідних даних.	05.06.2023	
3.	Обробка методичних та теоретичних	23.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2023	
5.	Розробка третього розділу.	30.10.2023	
6.	Оформлення та нормоконтроль	27.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ Д.В. Левченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Локалізація об’єктів зображення засобами мови програмування Python»: 62 с., 22 рис., 22 джерела, 4 додатки.

ЛОКАЛІЗАЦІЯ ОБ’ЄКТІВ ЗОБРАЖЕННЯ, НЕЙРОМЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ, OPEN CV, PYTHON, R-CNN, SSD, YOLO V3, YOLO V8.

Об’єкт дослідження – процес локалізації об’єкта зображення.

Предмет дослідження – локалізація об’єктів у межах цих зображень.

Мета роботи: вивчити сучасні підходи до локалізації об’єктів зображення та реалізувати алгоритм, який ефективно вирішує це завдання.

Метод дослідження – аналіз, вивчення та узагальнення, порівняння.

У магістерській кваліфікаційній роботі досліджувалися проблеми виявлення об’єктів на зображеннях та реалізації існуючих алгоритмів. У результаті було визначено та реалізовано найбільш оптимальну модель та алгоритм розв’язання задачі. Зокрема, було перевірено ефективність і точність алгоритму YOLOv3 у локалізації об’єктів на зображеннях. Було проведено порівняльний аналіз різних алгоритмів виявлення об’єктів, включаючи варіанти R-CNN, SSD і YOLO, де YOLOv8 став високоефективним і точним рішенням.

Крім того, розроблений алгоритм було реалізовано за допомогою мови програмування Python із використанням таких бібліотек, як OpenCV і numpy, для ефективної обробки зображень і локалізації об’єктів. Ефективність реалізованого алгоритму оцінювалася на основі підібраного набору даних, демонструючи значне покращення точності локалізації та часу обробки. Завдяки цій роботі було зроблено значний внесок у покращення методів локалізації об’єктів за допомогою інструментів програмування на Python, прокладаючи шлях для подальшого прогресу в обробці зображень і застосуваннях штучного інтелекту.

SUMMARY

Master’s qualifying paper «Localization of the Image Objects Based on the Python Programming Language»: 62 pages, 22 figures, 22 references, 4 supplements.

IMAGE OBJECT LOCALIZATION, NEURAL NETWORK, ARTIFICIAL INTELLIGENCE, OPEN CV, PYTHON, R-CNN, SSD, YOLO V3, YOLO V8.

The object of study is the process of image object localization.

The subject of research is the localization of objects within these images.

The aim of the work: to study modern approaches to image object localization, and to implement an algorithm that effectively addresses this task.

The methods of research are analysis, study and generalization, comparison.

In the Master’s qualification work, the problem of object detection within images and the implementation of existing algorithms were explored. As a result, the most optimal model and algorithm for solving the problem were identified and implemented. Specifically, the YOLOv3 algorithm was examined for its efficiency and accuracy in object localization within images. A comparative analysis was conducted among various object detection algorithms including R-CNN, SSD, and YOLO variants, where YOLOv8 emerged as a highly efficient and precise solution.

Moreover, the developed algorithm was implemented using the Python programming language, leveraging libraries such as OpenCV and numpy for effective image processing and object localization. The performance of the implemented algorithm was evaluated against a curated dataset, showcasing a significant improvement in localization accuracy and processing time. Through this work, a substantial contribution towards enhancing object localization techniques using Python programming tools has been made, paving the way for further advancements in image processing and artificial intelligence applications.

ЗМІСТ

ВСТУП

У сучасну цифрову епоху попит на автоматизацію вийшов на перший план. Компанії, організації та приватні особи постійно шукають ефективні способи автоматизації повсякденних завдань, щоб оптимізувати процеси та зменшити споживання ресурсів. В основі цієї революції автоматизації лежить сфера штучного інтелекту (ШІ).

Хоча ШІ здатен до трансформації, він не позбавлений своїх обмежень. Як і люди, ШІ покладається на знання, а знання є продуктом навчання. Щоб ШІ функціонував оптимально, йому потрібна база для навчання на конкретних прикладах. Ці приклади слугують планом, за яким ШІ виконує свої завдання.

Комп'ютерний зір – важлива галузь ШІ, міждисциплінарна сфера, яка дозволяє машинам інтерпретувати візуальні дані та приймати рішення. Розпізнавання об'єктів на зображеннях є особливо цікавою підгалуззю комп'ютерного зору, і саме їй присвячена ця стаття.

Ця стаття розпочинає дослідницьку подорож у галузі локалізації об'єктів на зображеннях. Використовуючи Python, мову програмування загального призначення, як інструмент, у статті розглядаються всі аспекти розробки програмного забезпечення, від аналізу та проєктування алгоритмів до оцінки їхньої ефективності та надійності.

Основними завданнями цього дослідження є:

- ретельно вивчити предметну галузь, ознайомившись із переважними рішеннями;
- визначити та пояснити найсучасніші та найефективніші алгоритми локалізації об'єктів;
- відібрати й обґрунтувати вибір бібліотек, сервісів і наборів даних, які допоможуть програмній реалізації завдання. Для ретельного тестування впровадженого рішення на підібраному наборі зображень, оцінки його продуктивності та надійності;

– протестувати реалізоване рішення на спеціально підібраному наборі зображень, оцінюючи його продуктивність і надійність.

1 ЗАГАЛЬНІ ВІДОМОСТІ РОЗПІЗНАВАННЯ ОБ’ЄКТІВ

1.1 Актуальність проблеми

У великій галузі комп’ютерних наук комп’ютерний зір посідає важливе місце, представляючи собою сплав математики, програмної інженерії та конкретних знань. Основна мета комп'ютерного зору – дати можливість машинам інтерпретувати візуальні дані і робити з них висновки, що лежить в основі штучного інтелекту.

Як підмножина комп’ютерного зору, область розпізнавання об’єктів на зображеннях має вирішальне значення. З точки зору програмної інженерії, існує багато викликів, включаючи потребу в ефективних алгоритмах, важливість обробки в реальному часі, роботу з різноманітними і великими наборами даних і забезпечення точності результатів. Розпізнавання об’єктів на зображеннях використовується в різних галузях:

- виявлення транспортних засобів: необхідне для розробки систем управління дорожнім рухом і безпілотних транспортних засобів;
- розпізнавання облич: необхідне для систем безпеки, персоналізованого маркетингу та платформ соціальних мереж;
- підрахунок пішоходів: необхідний для міського планування та заходів громадської безпеки;
- посилена безпека: корисна для виявлення аномалій та ідентифікації загроз на відео з камер спостереження;
- автономна навігація: дозволяє дронам і роботам орієнтуватися на складній місцевості.

Сфера розпізнавання об’єктів на зображеннях постійно розвивається завдяки технологічним стрибкам. Від R-CNN до його швидших варіантів, а пізніше RetinaNet, SSD і YOLO, методи характеризуються невпинним прагненням до ефективності і точності. Кожна ітерація вирішувала проблеми

попередніх поколінь програмної інженерії, впроваджуючи оптимізовані алгоритми та ощадливі процеси.

Це дослідження заглиблюється в цей шлях прогресу, розглядає різні підходи як в теорії, так і на практиці, і зосереджується на проблемах програмної інженерії, рішеннях і майбутній траєкторії розпізнавання об’єктів на зображеннях.

1.2 Розпізнавання об'єктів

Розпізнавання об’єктів – це фундаментальне завдання в широкій галузі комп'ютерного зору, спрямоване на ідентифікацію конкретних об’єктів на цифрових зображеннях. Цей складний процес включає багато підзадач, кожна з яких має свої проблеми та застосування:

- класифікація зображень: це завдання передбачає передбачення категорії окремого об’єкта на зображенні. З точки зору програмної інженерії, вона вимагає алгоритмів, здатних ефективно обробляти великі масиви даних і послідовно видавати точні прогнози;
- локалізація об’єктів: окрім класифікації об’єктів, локалізація звужує позицію об’єкта на зображенні, малюючи навколо нього обмежувальний прямокутник. Для цього потрібні алгоритми, які можуть обробляти просторові дані і точно визначати межі об’єктів на різних фонах і в різних умовах освітлення;
- виявлення об’єктів: це завдання поєднує класифікацію та локалізацію зображень. Воно ідентифікує, класифікує і визначає положення одного або декількох об’єктів на зображенні. Реалізація виявлення об’єктів передбачає розробку алгоритмів, здатних працювати в багатозадачному режимі, обробляти зображення в реальному часі і забезпечувати мінімальну кількість хибних спрацювань;
- сегментація об’єктів: це завдання, яке також називають «сегментацією

екземплярів» або «семантичною сегментацією», передбачає позначення конкретних пікселів, що відносяться до ідентифікованих об’єктів, а не малювання грубих обмежувальних прямокутників. Це вимагає алгоритмів, які можуть обробляти дані зображення на гранулярному рівні, розрізняючи близько розташовані об’єкти і точно визначаючи межі об’єктів.

Як показує цей розподіл, розпізнавання об’єктів охоплює складний набір завдань комп’ютерного зору. Від оптимізації алгоритмів і обробки даних до обробки в реальному часі та масштабування – кожна підзадача представляє унікальні проблеми програмної інженерії.

Більшість сучасних моделей розпізнавання об’єктів розробляються з використанням бібліотек глибокого навчання, таких як TensorFlow, PyTorch і Keras. Ці бібліотеки надають попередньо навчені моделі та архітектури, які можна точно налаштувати під конкретні завдання, скоротивши таким чином початковий час розробки. Однак вибір правильної бібліотеки зазвичай залежить від конкретного застосування, потреб у масштабуванні та досвіду команди.

Інтеграція моделей розпізнавання об’єктів у більші системи, такі як мобільні додатки, вебсервіси та вбудовані системи, пов’язана з низкою проблем. Важливо переконатися, що моделі працюють ефективно, можуть оновлюватися без простою системи і коректно взаємодіють з іншими компонентами системи.

Розпізнавання об’єктів належить до складних задач комп’ютерного зору [1]:

- класифікація зображень: продуктивність моделі вимірюється за допомогою середньої помилки класифікації для передбачуваних міток класів;
- локалізація об’єктів: ефективність моделі вимірюється на основі відстані між очікуваною та передбачуваною границею для передбачуваного класу;
- виявлення об’єктів: ефективність моделі оцінюється за допомогою метрик точності та відтворення для кожного з найкращих відповідних

обмежувальних рамок відносно відомих об’єктів на зображенні;

– проблеми навчання: навчання моделей розпізнавання об’єктів вимагає великої кількості мічених даних. Отримання цих даних саме по собі є проблемою. Існує додаткова проблема забезпечення різноманітності навчальних даних для запобігання упередженості моделі. Незбалансовані набори даних, де деякі категорії об’єктів недостатньо представлені, можуть спотворити прогнози моделі;
– розгортання та масштабування: після розробки модель потрібно розгорнути, що може бути на хмарних серверах, периферійних пристроях або вбудованих системах. Кожна з них пов’язана зі своїми проблемами. Хмарні розгортання вимагають ефективних рішень для масштабування, щоб впоратися з різними навантаженнями, в той час як периферійні або вбудовані розгортання вимагають оптимізації моделі відповідно до обмежень по ресурсах.

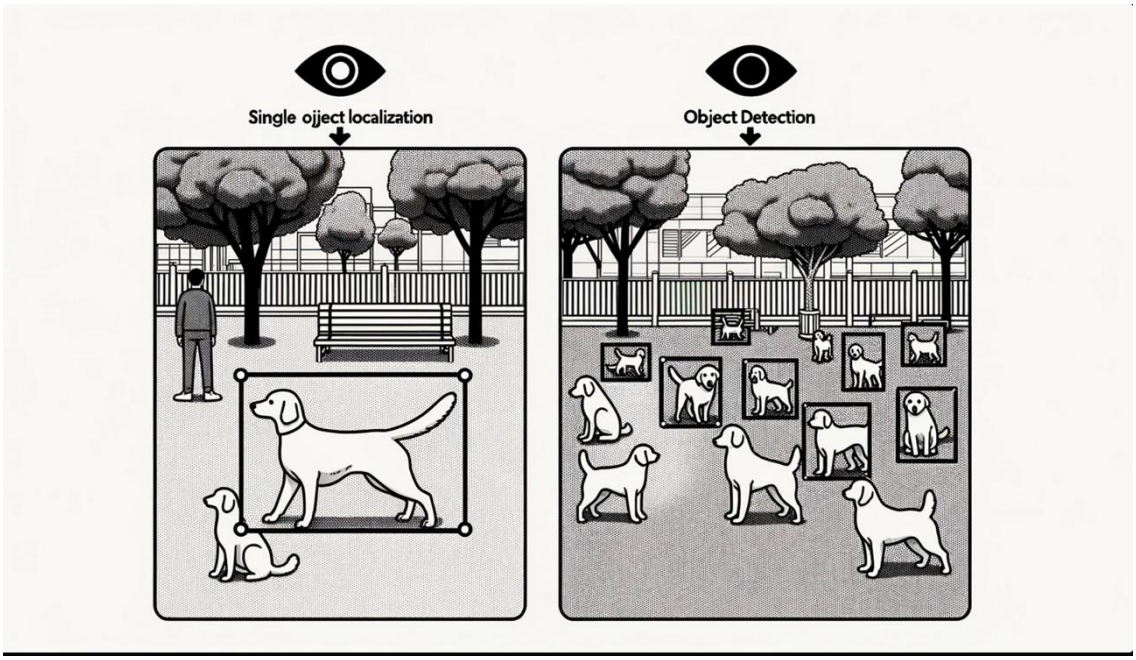


Рисунок 1.1 – Порівняння між «локалізацією одного об’єкта» та «виявленням об’єкта»

Розпізнавання об’єктів, будучи наріжним каменем комп’ютерного зору, тісно переплітається з принципами програмної інженерії. Незалежно від того,

чи це розробка моделі з використанням правильних інструментів, чи забезпечення її безперешкодної інтеграції в додатки, чи вирішення проблем розгортання, цілісний підхід, заснований на надійних практиках програмної інженерії, має першочергове значення.

1.3 Сімейство моделей R-CNN

Початкова модель R-CNN використовувала вибірковий пошук, щоб отримати близько 2000 пропозицій регіонів із зображення. Кожна з цих областей потім проходила через CNN (як правило, попередньо навчену мережу, як-от AlexNet), за якою слідували SVM для класифікації об’єкта та регресори для уточнення обмежувальної рамки. Незважаючи на ефективність, цей підхід був дорогим у плані обчислень через потребу обробляти кожну пропозицію регіону окремо.

R-CNN (Region-based Convolutional Neural Network) є значним прогресом у методології виявлення об’єктів. Спочатку розроблений Россом Гіршиком та його колегами, він є основоположним у поєднанні механізмів пропозицій регіонів із згортковий нейронними мережами для виявлення та класифікації об’єктів у зображенні [2].

1.3.1 R-CNN

R-CNN (Region-based Convolutional Neural Network) була, мабуть, одним із перших великих і успішних застосувань згорткових нейронних мереж (CNN) до проблеми локалізації об’єктів, виявлення та сегментації. Запропонована Россом Гіршиком та його командою модель R-CNN складається з трьох основних модулів:

– модуль регіональних пропозицій: цей модуль відповідає за створення

незалежних від категорії пропозицій регіонів, по суті, пропонуючи обмежувальні рамки-кандидати, які можуть містити об’єкти інтересу. Традиційні методи комп’ютерного зору, як-от «вибірковий пошук», використовуються для пропонування цих регіонів-кандидатів, але гнучка архітектура також дозволяє використовувати інші алгоритми пропонування регіонів;

– модуль вилучення функцій: для кожного запропонованого регіону цей модуль витягує функції. Це досягається за допомогою глибоких згорткових нейронних мереж. Зокрема, AlexNet, глибокий CNN, який виграв конкурс класифікації зображень ILSVRC-2012, використовувався як екстрактор функцій. Він створює 4096-елементний вектор, що описує вміст зображення;
– класифікатор: після виділення ознак об’єкти в цих областях класифікуються як один із відомих класів. Ця класифікація виконується за допомогою лінійної опорної векторної машини (SVM). Примітно, що для кожного відомого класу навчається окремий SVM.

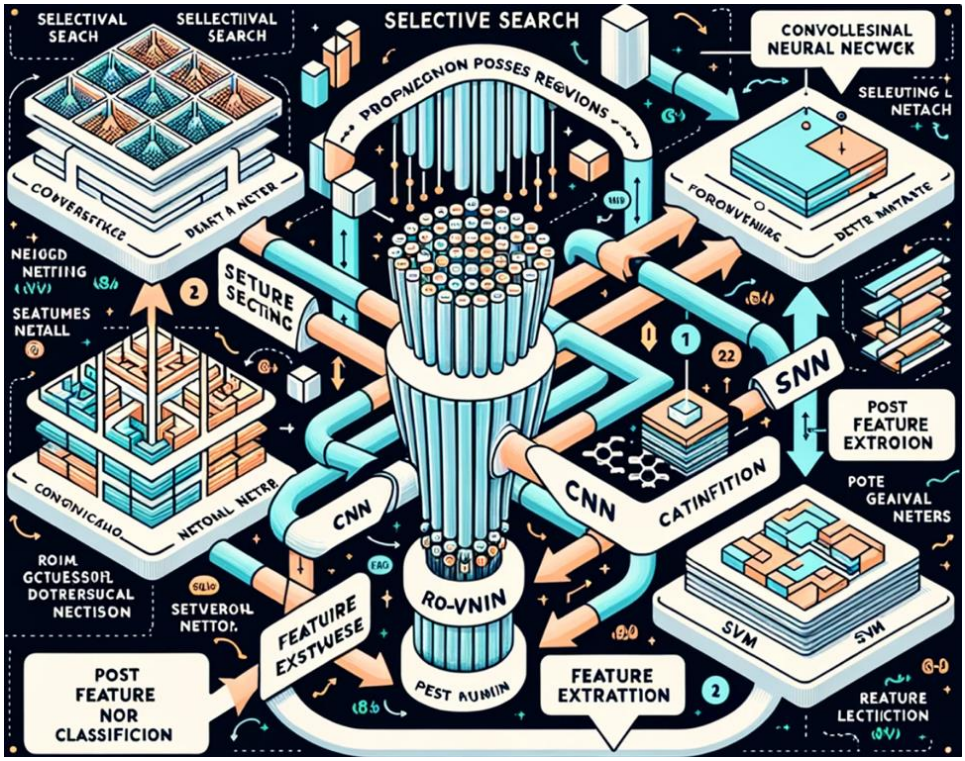


Рисунок 1.2 – Короткий опис архітектури моделі R-CNN

Значною перевагою R-CNN є його простий та інтуїтивно зрозумілий підхід до локалізації та розпізнавання об’єктів шляхом використання можливостей згорткових мереж. Однак основним недоліком цього підходу є його обчислювальна неефективність.

Вилучення ознак на основі CNN потрібно виконати для кожної з пропозицій регіону, згенерованого алгоритмом пропозиції регіону. Це стає вузьким місцем, оскільки під час тестування модель працює приблизно з 2000 запропонованими регіонами на зображення, що робить процес повільним і ресурсомістким.

1.3.2 Fast R-CNN

Визнаючи успіх R-CNN, Росс Гіршик, який тоді працював у Microsoft Research, вирішив усунути обмеження швидкості R-CNN. У своїй статті 2015 року під назвою «Fast R-CNN» [3] він представив удосконалення оригінальної моделі. Ось короткий перелік обмежень R-CNN:

- процес навчання був багатоетапним, спираючись на підготовку та роботу трьох різних моделей;
- навчання було дорогим як за часом, так і за ресурсами, що робило його помітно повільним;
- виявлення об’єктів було повільним через неодноразове використання глибоких CNN для пропозицій кількох регіонів;
- попередня стаття 2014 року під назвою «Об’єднання просторових пірамід у мережах глибокої згортки для візуального розпізнавання» [4] запропонувала техніку прискорення процесу з використанням мереж об’єднання просторових пірамід. Незважаючи на те, що це прискорило вилучення функцій, воно по суті покладається на форму прямого кешування.

Fast R-CNN було представлено як уніфіковану модель, яка спрощує

процес шляхом поєднання пропозицій регіонів і класифікацій. У його архітектурі зображення разом із набором пропозицій щодо регіонів беруться як вхідні дані та передаються через глибоку CNN, як-от VGG-16, для вилучення функцій. Кульмінацією глибокого CNN є спеціалізований рівень, який називається шаром «Об'єднання регіонів інтересів (RoI)». Цей шар виділяє характеристики, характерні для конкретної запропонованої області введення.

Відповідно до CNN, виходи інтерпретуються повністю пов'язаним шаром, після чого модель роздвоюється на два виходи: один прогнозує клас через шар softmax, а інший забезпечує лінійний вихід для обмежувальної рамки. Весь цей процес ітеративно застосовується для кожної запропонованої області на даному зображенні.

Хоча модель Fast R-CNN пропонує значні покращення у швидкості навчання та можливостях прогнозування порівняно з її попередницею, вона все ще потребує зовнішнього механізму, щоб пропонувати регіони-кандидати для подачі разом із кожним вхідним зображенням [5].

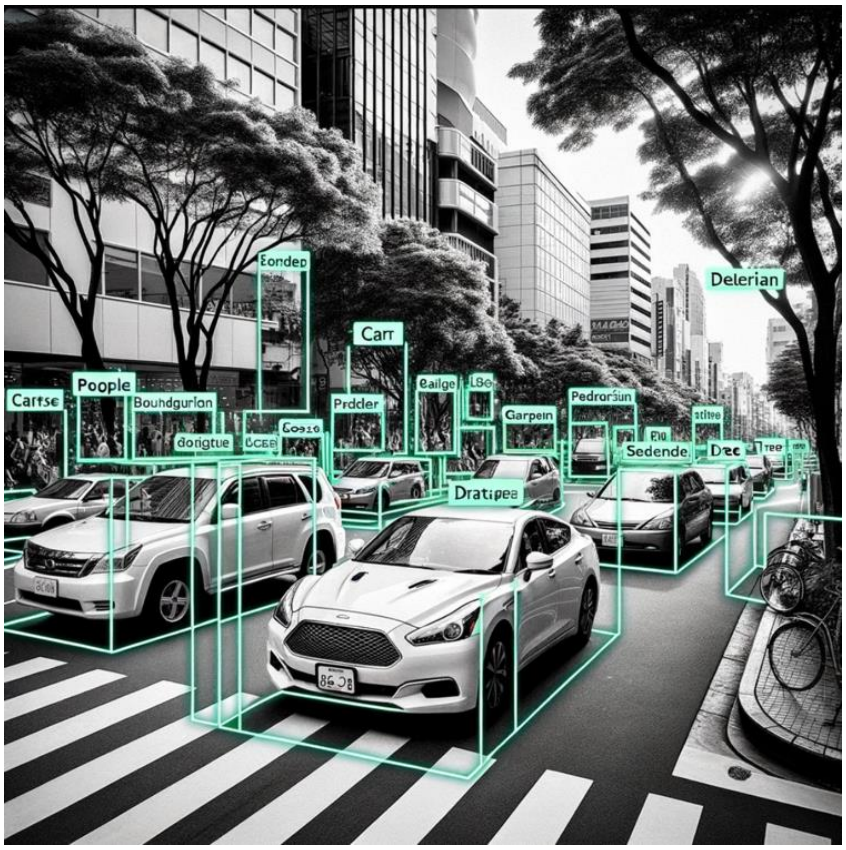


Рисунок 1.3 – Приклад локалізації засобами Fast R-CNN

1.4 Сімейство моделей YOLO

Серед величезного арсеналу моделей розпізнавання об’єктів сім’я YOLO (You Only Look Once), задумана Джозефом Редмоном у 2015 році, постає помітним суперником. Його початок ознаменував значний крок у виявленні об’єктів у режимі реального часу, встановивши швидкий, але ефективний підхід, який особливо відрізнявся від, можливо, більш точних, але повільніших моделей R-CNN.

Щоб окреслити суть доцільності YOLO, дуже важливо заглибитися в її базову архітектуру. Фреймворк YOLO використовує наскрізну навчену нейронну мережу, яка приймає зображення та негайно передбачає обмежувальні рамки та мітки класів для кожної обмежувальної рамки. Ця особливість в обробці дозволяє YOLO працювати з чудовою швидкістю від 45 до 155 кадрів на секунду, хоча й ціною точності передбачення, що проявляється у більших помилках локалізації.

Принцип роботи починається з поділу вхідного зображення на сітку клітинок, де кожна клітинка відповідає за прогнозування обмежувальної рамки, якщо центр обмежувальної рамки потрапляє в неї. Кожна комірка сітки прогнозує обмежувальну рамку, яка охоплює координати x, y, ширину, висоту та оцінку достовірності. Крім того, передбачення класу закріплюється на кожній комірці, інкапсулюючи зображення, наприклад, у сітку 7×7, кожна з яких здатна передбачити дві обмежувальні рамки, кульмінацією яких є 94 запропоновані обмежувальні рамки. Сукупність карт імовірностей класів і обмежувальних прямокутників з оцінкою достовірності об’єднується в остаточний набір обмежувальних прямокутників і міток класів (див. рис. 1.4).

У 2016 році Джозеф Редмон у співпраці з Алі Фархаді удосконалив модель, кульмінацією якої став YOLOv2, який також отримав назву «YOLO9000» за його здатність передбачати до 9000 класів об’єктів. Перегляд охопив безліч тренінгів і архітектурних модифікацій, таких як пакетна нормалізація та вхідні зображення з високою роздільною здатністю. Подібно до

Faster R-CNN [6], YOLOv2 використовує опорні рамки, попередньо визначені обмежувальні рамки з корисними формами та розмірами, вибраними під час навчання. Вибір обмежувальних рамок для зображення попередньо обробляється за допомогою аналізу k-середніх у навчальному наборі даних [7].

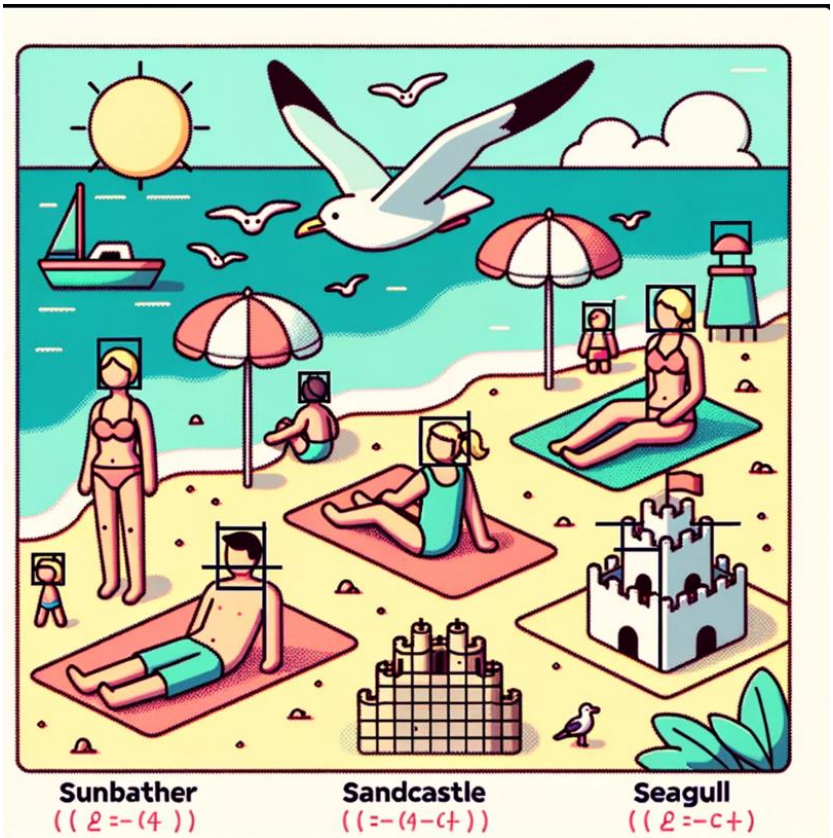


Рисунок 1.4 – Підсумок прогнозів, зроблених моделлю YOLO

Було внесено основну поправку до представлення прогнозованих обмежувальних рамок, щоб пом’якшити різкий вплив незначних змін на прогнози, створивши більш стабільну модель. Замість прямого прогнозування положення та розміру зміщення для переміщення та зміни форми попередньо визначених прив’язок відносно клітинки сітки прогнозуються та логістично пом’якшуються (див. рис. 1.5).

На цьому еволюція YOLO не закінчується. Стаття Редмона і Фархаді 2018 року під назвою «YOLOv3: Поступові вдосконалення» [8], яка показує, що подальші вдосконалення YOLO, хоча і відносно незначні, такі як більш глибока мережа функціональних детекторів і незначні зміни в презентації, були

додані [9].

10 січня 2023 року YOLO v8, або «Координована ультрааналітика», відкриє завісу. Цей авангардний продукт спирається на попереднє покоління, додаючи нові вдосконалення, спрямовані на підвищення продуктивності та універсальності; найбільшою силою YOLO v8 є його універсальність, про що свідчить постійний пошук вдосконалень у сфері ідентифікації місцезнаходження об’єктів з використанням широкого спектру інструментів Python.

Рисунок 1.5 – Приклад подання, вибраного під час прогнозування положення та форми обмежувальної рамки

1.5 Висновки до розділу 1

У цьому розділі описано загальні принципи розпізнавання об’єктів та його структурну основу. Крім того, обговорюються основні серії нейронних мереж, призначених для вирішення цієї задачі. Детальне дослідження показало,

що існують значні відмінності між сімейством R-CNN та сімейством YOLO. Виявилося, що сімейство R-CNN має вищу точність, тоді як сімейство YOLO є достатньо швидким для виявлення об’єктів у реальному часі. Це порівняння підкреслює важливий компроміс між точністю та своєчасністю і підтверджує необхідність збалансованого підходу до розпізнавання об’єктів. Зібрана тут інформація забезпечує міцну основу для наступних етапів аналізу та впровадження і демонструє необхідність розуміння тонкої динаміки різних архітектур нейронних мереж при використанні різних інструментів Python для ефективної локалізації об’єктів. Ця доповідь є не тільки інформативною, але й закладає основу для глибшого розуміння складнощів, пов'язаних з локалізацією об’єктів, і є важливим кроком на шляху до створення надійних та ефективних рішень для локалізації об’єктів на зображеннях.

2 ДЕТАЛЬНЕ ЗНАЙОМСТВО З ВИКОРИСТАНИМИ ТЕХНОЛОГІЯМИ

2.1 Python: основа локалізації об’єктів

Python – це не лише мова програмування, але й основа для наших досліджень у галузі локалізації об’єктів зображень. Її універсальність робить її незамінним інструментом для наших досліджень, особливо в галузі локалізації об’єктів зображень.

Цей розділ дає детальний огляд можливостей Python та їхнього зв’язку з завданнями нашого курсу.

Такі бібліотеки, як OpenCV, TensorFlow та PyTorch, необхідні для розробки складних алгоритмів, які можуть розпізнавати та локалізувати об’єкти на зображеннях. Легка інтеграція цих бібліотек та можливість використовувати їх функціональність значно прискорила прогрес нашого проєкту.

Крім того, синтаксис мови Python відомий своєю ясністю та читабельністю, що допомогло створити середовище для спільної роботи. Визначення місцезнаходження об’єктів є міждисциплінарним завданням, яке вимагає поєднання досвіду розробки програмного забезпечення, машинного навчання та аналізу даних, а статус Python як мови загального призначення полегшує співпрацю та прискорює перетворення теоретичних концепцій на практичні моделі визначення місцезнаходження об’єктів, це має велике значення.

Крім того, потужна підтримка спільноти Python є скарбницею знань і ресурсів. Невпинне прагнення спільноти до інновацій продовжує створювати нові інструменти, які вдосконалюють існуючі та покращують локалізацію об’єктів. Спільнота часто щедро ділиться заздалегідь підготовленими моделями та наборами даних. Це є благом для наших проєктів і дозволяє нам покладатися на колективну мудрість без необхідності винаходити велосипед.

2.1.1 Python в аналізі даних і машинному навчанні для локалізації об’єктів

Python став основним продуктом науки про дані, дозволяючи аналітикам даних та іншим професіоналам використовувати цю мову для проведення складних статистичних обчислень, створення візуалізацій даних, створення алгоритмів машинного навчання, обробки та аналізу даних, а також виконання інших завдань, пов’язаних із даними.

Python може створювати широкий спектр різних візуалізацій даних, як-от лінійні та стовпчасті діаграми, секторні діаграми, гістограми та 3D-графіки. Python також має низку бібліотек, які дозволяють програмістам писати програми для аналізу даних і машинного навчання швидше й ефективніше, наприклад TensorFlow і Keras.

2.1.2 Python у веброзробці для платформ локалізації об’єктів

Python постає стрижнем в області науки про дані, прокладаючи шлях до надійних рішень у нашій курсовій темі локалізації об’єктів. Його майстерність у виконанні складних статистичних обчислень, створенні візуалізації даних і створенні алгоритмів машинного навчання є центральною для розпізнавання та локалізації об’єктів на зображеннях. Такі бібліотеки, як TensorFlow і Keras, є факелоносцями в цій подорожі, рухаючи нас із царства абстрактних концепцій до реальних рішень у локалізації об’єктів.

Нюанси завдання локалізації об’єктів вимагають глибокого розуміння та ретельного аналізу даних. Python зі своїм комплексним набором бібліотек аналізу даних, таких як Pandas і NumPy, забезпечує благодатний грунт для просіювання даних, виявлення шаблонів і отримання безцінної інформації, яка є важливою для створення ефективних моделей локалізації об’єктів.

З іншого боку, Flask пропонує мінімалістичний і гнучкий підхід до

веброзробки. Його легка природа робить його ідеальним вибором для створення мікросервісів або коли потрібен більш практичний контроль над компонентами. Простота та гнучкість Flask робить його сприятливим середовищем для розробників для створення індивідуальних вебінтерфейсів, які задовольняють конкретні потреби програм локалізації об’єктів.

Крім того, машинне навчання має локалізації об’єктів, який знаходить зручне середовище в Python. Бібліотеки мови, TensorFlow і Keras, є маяками у бурхливому розвитку машинного навчання. Вони пропонують безліч функціональних можливостей, які спрощують впровадження складних алгоритмів машинного навчання, необхідних у навчальних моделях для точної ідентифікації та локалізації об’єктів на зображеннях.

Крім того, активна спільнота Python є резервуаром інноваційних рішень і спільних знань у сфері аналізу даних і машинного навчання. Доступність широкого спектру бібліотек із відкритим кодом і готових моделей прискорює процес досліджень і розробок, просуваючи зусилля з локалізації об’єктів.

2.1.3 Автоматизація сценаріїв у Python для завдань локалізації об’єктів

Автоматизація, сильна сторона Python, постає як лицар у блискучих обладунках, відлякувач демонів надмірності та неефективності наших завдань локалізації об’єктів. Сценарії Python – це невидимі робочі конячки, які невтомно обробляють дані, перевіряють помилки та готують ґрунт для процвітання наших моделей машинного навчання на арені локалізації об’єктів.

Автоматизація в Python прокладає шлях до оптимізації багатьох аспектів робочого процесу локалізації об’єктів. Від збору даних і попередньої обробки до навчання моделі та перевірки, сценарії Python виконують значну частину роботи. Вони автоматизують такі повсякденні завдання, як анотація зображень, доповнення даних і перевірка помилок, звільняючи дорогоцінний час для

зосередження на удосконаленні моделей локалізації об’єктів.

Багата екосистема бібліотек і фреймворків Python, таких як OpenCV і PIL, ще більше розширює його можливості автоматизації. Вони пропонують безліч функцій і інструментів для обробки зображень і маніпулювання ними, які мають вирішальне значення для підготовки даних для завдань локалізації об’єктів.

Крім того, майстерність сценаріїв Python поширюється на автоматизацію оцінки та порівняння різних моделей локалізації об’єктів. Завдяки написанню сценаріїв процесів оцінювання можна проводити послідовні та точні порівняння, що допомагає визначити моделі, які пропонують чудову продуктивність і точність.

У великій схемі локалізації об’єктів автоматизація за допомогою Python є не просто розкішшю, а необхідністю. Це прискорює темп розробки, підвищує точність моделей і наближає нас до мети ефективної та ефективної локалізації об’єктів.

2.1.4 Python у тестуванні програмного забезпечення та створенні прототипів для локалізації об’єктів

Такі інструменти, як Green і Requestium, є нашими союзниками в тому, щоб кожна ітерація нашої моделі локалізації об’єктів витримала випробування часом і логікою, готова до вдосконалення або розгортання.

У ітераційному процесі розробки моделі створення прототипів за допомогою Python забезпечує швидкий і ефективний спосіб перевірити нові ідеї та підходи. Простота мови та наявність великих бібліотек дозволяють швидко створювати прототипи алгоритмів локалізації об’єктів, забезпечуючи швидку перевірку гіпотез і коригування, якщо це необхідно.

Коли справа доходить до тестування програмного забезпечення, бібліотеки Python, такі як PyTest, і такі інструменти, як Green і Requestium,

сприяють ретельному тестуванню моделей локалізації об’єктів і пов’язаного коду. Вони допомагають виявляти помилки, забезпечувати якість коду та перевіряти, чи моделі працюють належним чином у різних сценаріях.

Ці інструменти також підтримують робочі процеси безперервної інтеграції та безперервної доставки (CI/CD), гарантуючи, що моделі та платформи локалізації об’єктів залишаються стійкими та надійними в міру їх розвитку.

Завдяки автоматизації процесу тестування вони допомагають підтримувати високий стандарт якості, який є обов’язковим у сфері локалізації об’єктів, де точність має першорядне значення.

2.1.5 Щоденний Python: погляд за межі локалізації об’єктів

Python виходить за межі, торкаючись життя не програмістів і професіоналів. Цей розділ, хоч і обхідний шлях, відзначає всюдисущість Python і його потенціал спрощувати життя, тим самим повторюючи ширшу розповідь нашої курсової роботи – роблячи технологію доступною та придатною для використання для всіх через локалізацію об’єктів.

Універсальність Python краще за все проявляється, коли він знаходить застосування в різних сферах, окрім локалізації об’єктів. Від автоматизації повсякденних завдань, допомоги в аналізі даних для журналістів до спрощення маркетингових зусиль у соціальних мережах, Python обіцяє зробити технологію стимулом, а не перешкодою.

Цей ширший вплив Python перегукується з духом нашої курсової роботи, яка спрямована на демократизацію технології через локалізацію об’єктів. Заглиблюючись у технічні особливості локалізації об’єктів, ми також визнаємо ширшу роль Python у подоланні цифрового розриву, підкреслюючи трансформаційний потенціал технології, коли вона стає доступною та придатною для використання всіма.

2.2 NumPy як основний інструмент у локалізації об’єктів зображення

NumPy, що розшифровується як Numerical Python, – це бібліотека з відкритим кодом, яка широко використовується в різних наукових та інженерних сферах. Він служить фундаментальним стандартом для обробки числових даних у Python і формує основу наукових екосистем Python, таких як PyData. База користувачів NumPy варіюється від початківців програмістів до досвідчених дослідників, які займаються передовими науковими та промисловими дослідженнями та розробками. Його прикладний програмний інтерфейс (API) широко застосовується в кількох інших наукових бібліотеках Python, таких як Pandas, SciPy, Matplotlib, scikit-learn і scikit-image, щоб назвати декілька.

Ядром NumPy є його багатовимірні масиви та матричні структури даних. Він забезпечує ndarray, однорідний n-вимірний об’єкт масиву, набором методів для активних операцій. NumPy дозволяє виконувати різноманітні математичні операції над масивами, розширюючи можливості Python, додаючи потужні структури даних, що забезпечує ефективні обчислення з масивами та матрицями. Крім того, він пропонує велику бібліотеку математичних функцій високого рівня, які працюють з цими масивами та матрицями.

NumPy відрізняє його здатність надавати безліч швидких і ефективних способів створення масивів і обробки числових даних у них. На відміну від списків Python, які можуть розміщувати різні типи даних в одному списку, усі елементи в масиві NumPy мають бути однорідними. Ця однорідність має вирішальне значення, оскільки інакше математичні операції, що виконуються над масивами, були б надзвичайно неефективними.

Масиви NumPy швидкі та компактні порівняно зі списками Python. Масив споживає менше пам'яті і зручніший у використанні. NumPy використовує значно менше пам’яті для зберігання даних і надає механізм для визначення типу даних, уможливлюючи подальшу оптимізацію коду.

У контексті локалізації об’єктів зображення використання NumPy може

значно прискорити обчислювальні процеси, що робить аналіз у реальному часі можливим. Завдяки дослідженню нових або менш відомих функцій NumPy або нових реалізацій у локалізації об’єктів зображення, це дослідження має на меті внести оригінальні ідеї та результати в цю сферу. У наступних розділах розглядатимуться конкретні інструменти та методології Python, демонструючи відмінні відкриття та досягнення, досягнуті у сфері локалізації об’єктів зображення.

2.3 OpenCV у локалізації об’єкта зображення в екосистемі Python

OpenCV (бібліотека комп’ютерного бачення з відкритим кодом) є основною бібліотекою для комп’ютерного зору та машинного навчання з ліцензуванням відкритого коду. Він був задуманий для створення надійної інфраструктури для додатків комп’ютерного зору та прискорення впровадження машинного сприйняття в комерційних продуктах. Завдяки ліцензії Apache 2 OpenCV дозволяє компаніям легко використовувати та змінювати код.

Бібліотека може похвалитися понад 2500 оптимізованими алгоритмами, що охоплюють повний набір як класичних, так і сучасних алгоритмів комп’ютерного зору та машинного навчання. Ці алгоритми забезпечують безліч функціональних можливостей, включаючи виявлення та розпізнавання облич, ідентифікацію об’єктів, класифікацію дій людини у відео, відстеження руху камери, відстеження рухомих об’єктів, вилучення моделей 3D-об’єктів, зшивання зображень високої роздільної здатності зі сцени, пошук подібних зображень у базі даних, усунення ефекту червоних очей із зображень зі спалахом, відстеження рухів очей, розпізнавання ландшафту для накладання доповненої реальності та багато іншого.

Спільнота користувачів OpenCV велика, нараховує понад 47 000 учасників, а бібліотеку завантажено понад 18 мільйонів разів, демонструючи її

широке визнання та використання в різних доменах.

Прийняття OpenCV поширюється не тільки на добре відомі корпорації, такі як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda або Toyota, але поширюється на численні стартапи, такі як Applied Minds, VideoSurf і Zeitera.

Застосування OpenCV охоплюють широкий діапазон: від зшивання зображень вулиць, виявлення вторгнень у системах відеоспостереження в Ізраїлі, моніторингу шахтного обладнання в Китаї, допомоги роботам у навігації та збиранні об’єктів у Willow Garage, виявлення випадків утоплення в басейнах у Європі, інтерактивне мистецтво установки в Іспанії та Нью-Йорку, перевірка злітно-посадкових смуг на предмет сміття в Туреччині, перевірка етикеток на продуктах на заводах по всьому світу, швидке виявлення облич у Японії.

OpenCV надає інтерфейси для C++, Python, Java і MATLAB, підтримуючи різні операційні системи, включаючи Windows, Linux, Android і Mac OS. В основному він зосереджений на програмах бачення в реальному часі, використовуючи переваги інструкцій MMX і SSE, якщо вони доступні.

Триває активна розробка повнофункціональних інтерфейсів CUDA та OpenCL. Бібліотека, написана мовою C++, має інтерфейс шаблону, який бездоганно взаємодіє з контейнерами STL.

У цій роботі пакет оболонки opencv-python використовувався для використання можливостей OpenCV для завдань локалізації об’єктів зображення.

Завдяки використанню прив’язок Python для OpenCV дослідження досліджує інноваційні підходи та впроваджує ефективні методи локалізації об’єктів на зображеннях.

Це дослідження не лише проливає світло на практичні застосування, але й прагне до глибшого розуміння та оптимізації базових алгоритмів, сприяючи оригінальним ідеям у цій галузі.

2.4 Алгоритм ефективної локалізації об’єктів YOLO

Люди можуть швидко сприймати зображення, ідентифікувати об’єкти, їх розташування та взаємодії в них. Це швидке й точне сприйняття полегшує виконання складних завдань, таких як водіння з мінімальними свідомими зусиллями. Швидкі та точні алгоритми виявлення об’єктів можуть оснастити комп’ютери для керування транспортними засобами в різних погодних умовах без спеціальних датчиків, дозволити допоміжним пристроям надавати користувачам інформацію про сцену в реальному часі та розкрити потенціал узагальнених роботизованих систем.

Традиційні системи виявлення об’єктів перепрофілюють класифікатори для завдань виявлення. Щоб знайти об’єкт, вони використовують класифікатор для цього об’єкта, оцінюючи його в різних положеннях і масштабах на тестовому зображенні. Такі системи, як моделі деформованих частин (DPM), використовують підхід ковзного вікна, коли класифікатор запускається в рівномірно розташованих місцях по всьому зображенню. Більш сучасні підходи, такі як R-CNN, використовують методи пропозиції регіону для початкового створення потенційних обмежувальних рамок на зображенні, згодом запускаючи класифікатор для цих запропонованих рамок. Пост Класифікація, постобробка уточнює обмежувальні прямокутники, усуває виявлення дублікатів і повторно оцінює коробки на основі інших об’єктів у сцені. Ці складні конвеєри є повільними та складними для оптимізації, оскільки кожен компонент потребує окремого навчання.

Алгоритм YOLO (You Only Look Once) перетворює виявлення об’єктів як єдине завдання регресії, переходячи безпосередньо від пікселів зображення до координат обмежувальної рамки та ймовірностей класу [10]. З YOLO ви дивитеся на зображення лише один раз, щоб передбачити наявність і розташування об’єктів.

Архітектура YOLO надзвичайно проста (див. рис. 2.1). Одна згорткова мережа одночасно передбачає кілька обмежувальних прямокутників і

ймовірності класів для цих рамок [11]. YOLO тренується на цілих зображеннях, безпосередньо оптимізуючи продуктивність виявлення. Ця уніфікована модель має кілька переваг перед традиційними методами виявлення об’єктів.

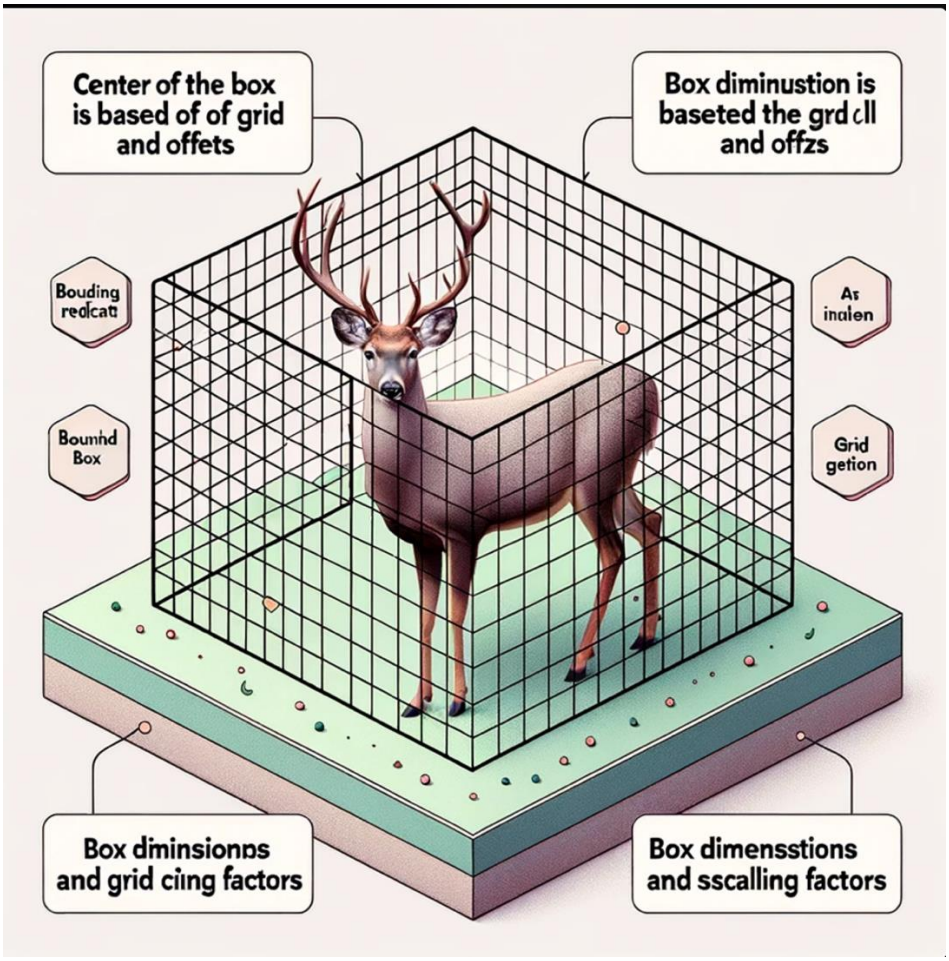


Рисунок 2.1 – Система виявлення YOLO

По-перше, YOLO надзвичайно швидкий. Перегляд виявлення як проблеми регресії усуває потребу в складних конвеєрах. Він просто запускає нейронну мережу на новому зображенні під час тестування, щоб передбачити виявлення. Ця базова мережа працює зі швидкістю 45 кадрів на секунду без пакетної обробки на графічному процесорі Titan X, а швидка версія працює зі швидкістю понад 150 кадрів на секунду. Це дозволяє обробляти відео в реальному часі із затримкою менше 25 мілісекунд. Крім того, YOLO досягає вдвічі більшої середньої точності, ніж інші системи реального часу.

По-друге, YOLO розуміє глобальний контекст зображення, роблячи

прогнози. На відміну від ковзного вікна та методів на основі пропозицій регіону, YOLO бачить все зображення під час навчання та тестування, кодуючи контекстну інформацію про класи та їх зовнішній вигляд. Швидкий R-CNN, провідний метод виявлення, неправильно інтерпретує фонові плями на зображенні як об’єкти, оскільки не може бачити ширший контекст. YOLO створює менше половини фонових помилок порівняно з Fast R-CNN.

По-третє, YOLO вивчає узагальнені представлення об’єктів. При навчанні на природних зображеннях і тестуванні на творах мистецтва YOLO значно перевершує такі провідні методи виявлення, як DPM і R-CNN. Його дуже узагальнений характер зменшує ймовірність збою при застосуванні до нових доменів або несподіваних вхідних даних.

У цій роботі прив’язка Python до YOLO використовується для дослідження інноваційних методологій локалізації об’єктів на зображеннях, демонструючи ефективність і результативність алгоритму у вирішенні проблем реального світу. Завдяки ретельному аналізу та оптимізації алгоритму надається нове уявлення про область локалізації об’єктів, що сприяє ширшому розумінню та розвитку цієї галузі.

2.4.1 Уніфіковане виявлення об’єктів

Інтеграція інструментів локалізації об’єктів у рамках програмування на Python суттєво розширена завдяки структурі YOLO (You Only Look Once), яка об’єднує різні компоненти виявлення об’єктів у єдину нейронну мережу. Ця мережа вправно використовує функції всього зображення для одночасного прогнозування кожної обмежувальної рамки, що передбачає глобальне споглядання всього зображення разом з усіма об’єктами, які містяться в ньому. Винахідливість конструкції YOLO сприяє безперебійному навчанню та швидкості в реальному часі, зберігаючи похвальну середню точність.

Операційний механізм YOLO розділяє вхідне зображення на сітку S × S.

Комірка сітки відповідає за виявлення об’єкта за умови, що центр об’єкта знаходиться всередині неї. Кожна комірка сітки виконує передбачення B обмежуючих прямокутників разом із пов’язаними з ними оцінками достовірності. Ці бали втілюють впевненість моделі щодо присутності об’єкта в рамці та передбачувану точність обмежувальної рамки, яку вона передбачає. Формально він визначає впевненість як Pr(Object) * IOU_pred_truth. У сценаріях, коли клітинка не інкапсулює жодного об’єкта, оцінка достовірності дорівнює нулю. Навпаки, він бажає, щоб оцінка достовірності прирівнювала перехрестя через з’єднання (IOU) між прогнозованою коробкою та основною правдою.

Кожне передбачення обмежувальної рамки містить п’ять очікувань: x, y, w, h і впевненість. Координати (x, y) окреслюють центр прямокутника по відношенню до меж комірки сітки, а ширина і висота прогнозуються по відношенню до всього зображення. Згодом передбачення впевненості є прикладом IOU між прогнозованим блоком і будь-яким базовим блоком правди.

Крім того, кожна клітинка сітки прогнозує умовні ймовірності класу, C, виражені як Pr(Class_i | Object). Ці ймовірності залежать від комірки сітки, в якій міститься об’єкт. Він передбачає єдиний набір ймовірностей класу на комірку сітки, незалежно від кількості обмежувальних рамок B.

$$\Pr(\text{Class}_i|\text{Object})*\Pr(\text{Object})*\text{IOU}_{\text{truth}}^{\text{pred}} = \Pr(\text{Class}_i)* \text{IOU}_{\text{truth}}^{\text{pred}}$$

Під час тестування YOLO об’єднує ймовірності умовного класу та прогнози достовірності окремих блоків, таким чином забезпечуючи оцінки достовірності для кожного блоку для конкретного класу. Ці показники кодують як ймовірність того, що певний клас проявиться всередині прямокутника, так і те, наскільки добре передбачуваний квадрат узгоджується з об’єктом.

Аналітичний опис ефективності коду та реалізованих методологій у рамках YOLO міг би додатково обґрунтувати дослідження. Це включає в себе

заглиблення в порівняльний аналіз з іншими інструментами локалізації об’єктів, дослідження нюансів коду та виявлення нових аспектів або додатків, які раніше не були описані в існуючих джерелах. Це прагнення до оригінальності досліджень і результатів сприятиме унікальному погляду на наявний обсяг знань щодо локалізації об’єктів за допомогою інструментів Python.

2.4.2 Архітектура проєктування

Реалізована модель сформульована як згортальна нейронна мережа (CNN) і оцінена за набором даних виявлення VOC PASCAL [12]. Основні згорткові шари мережі сконструйовані для вилучення особливостей із зображення, тоді як повністю з’єднані шари готові передбачити вихідні ймовірності та координати. Ця мережева архітектура черпає натхнення з моделі GoogLeNet, відомої своєю класифікацією зображень, що включає 24 згорткові шари, за якими слідують 2 повністю з’єднані шари [13]. Однак, відрізняючись від початкових модулів, використовуваних GoogLeNet, ця конструкція використовує шари скорочення 1 × 1, за якими слідують шари згортки 3 × 3.

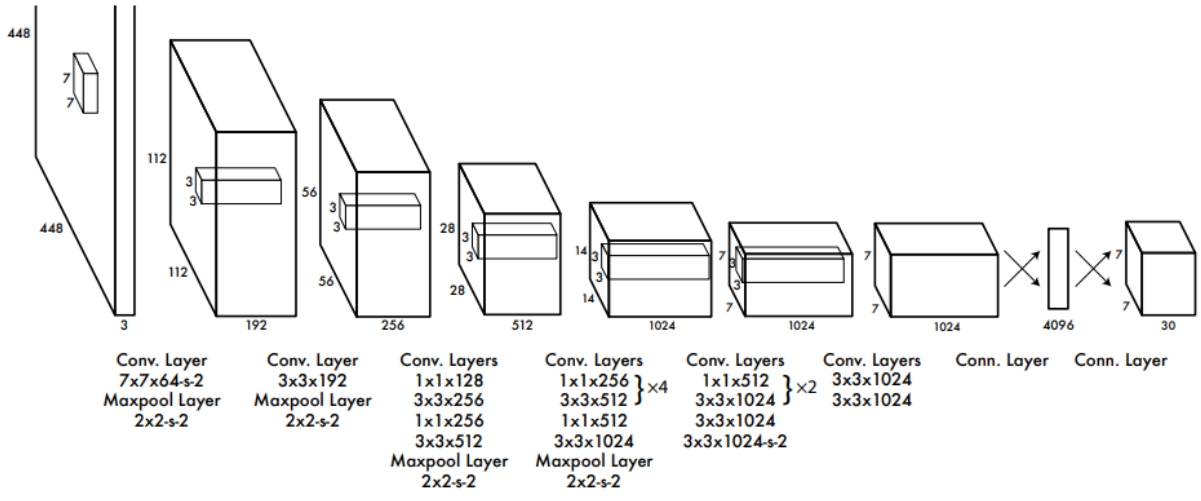


Рисунок 2.2 – Архітектура мережі

Крім того, досліджується гнучкіша версія YOLO, відома як Fast YOLO, щоб розширити межі швидкого виявлення об’єктів. Fast YOLO втілює нейронну мережу з меншою кількістю згорткових шарів (9 замість 24) і зменшеною кількістю фільтрів у цих шарах.

За винятком розміру мережі, усі параметри навчання та тестування залишаються узгодженими між YOLO та Fast YOLO. Кульмінацією роботи цієї мережі є тензор прогнозів $7 \times 7 \times 30$.

2.4.3 Обмеження YOLO

Алгоритм YOLO (You Only Look Once) накладає значні просторові обмеження на передбачення обмежувальної рамки, оскільки кожна клітинка сітки передбачає лише дві коробки та може мати лише один клас.

Це просторове обмеження обмежує кількість сусідніх об’єктів, які ця модель може передбачити. Зокрема, модель бореться з дрібними об’єктами, які з’являються в кластерах, як зграя птахів.

Навчаючись передбачати обмежувальні прямокутники з даних, модель знаходить труднощі в узагальненні об’єктів з новими або незвичними пропорціями чи конфігураціями. Він використовує відносно грубі функції для передбачення обмежувальної рамки через кілька рівнів архітектури зменшення вибірки вхідного зображення.

Крім того, хоча вона тренується на функції втрат, яка наближає продуктивність виявлення, ця функція втрат однаково розглядає помилки в малих і великих обмежуючих прямокутниках.

Невелика помилка у великому прямокутнику, як правило, є доброякісною, але невелика помилка у малому прямокутнику має значно більший вплив на показник перетину через з’єднання (IoU). Основним джерелом помилок є неправильна локалізація.

2.5 Порівняння Yolo з іншими системами виявлення

Виявлення об’єктів залишається фундаментальною проблемою комп’ютерного зору. Конвеєри виявлення зазвичай починаються з вилучення набору надійних функцій із вхідних зображень. Згодом для ідентифікації об’єктів у просторі ознак використовуються класифікатори або локалізатори. Ці класифікатори або локалізатори працюють у вигляді ковзного вікна по всьому зображенню або над підмножиною областей зображення.

2.5.1 Моделі деформованих частин (DPM)

Моделі деформованих частин (DPM) використовують підхід ковзного вікна для виявлення об’єктів. DPM використовує спеціалізований конвеєр для виділення статичних ознак, класифікації регіонів, прогнозування обмежувальної рамки для регіонів з високими балами тощо.

На відміну від цього, YOLO замінює всі ці різні компоненти єдиною згортковою нейронною мережею (CNN). Ця мережа одночасно виконує вилучення ознак, передбачення обмежувальної рамки, максимальне придушення та контекстне обґрунтування.

Замість того, щоб використовувати статичні функції, мережа тренує функції онлайн і оптимізує їх для завдання виявлення. Уніфікована архітектура YOLO забезпечує більш швидку та точну модель порівняно з DPM.

2.5.2 CNN (R-CNN)

R-CNN і його варіанти використовують пропозиції регіонів замість ковзних вікон для пошуку об’єктів на зображеннях. Вибірковий пошук [14] генерує потенційні обмежувальні прямокутники, згорткова мережа виділяє

функції, опорна векторна машина (SVM) оцінює блоки, лінійна модель уточнює обмежувальні прямокутники, а не максимальне придушення усуває виявлення дублікатів. Кожну стадію цього складного конвеєра потрібно ретельно налаштовувати незалежно, в результаті чого система працює досить повільно, займаючи більше 40 секунд для кожного зображення під час тестування. YOLO має деякі спільні риси з R-CNN.

Кожна клітинка сітки пропонує потенційні обмежувальні рамки та оцінює ці рамки за допомогою згорткових функцій. Однак YOLO накладає просторові обмеження на пропозиції комірок сітки, допомагаючи пом’якшити численні виявлення одного й того ж об’єкта. YOLO також пропонує набагато менше обмежувальних рамок, лише 98 на зображення порівняно з приблизно 2000 від вибіркового пошуку. Нарешті, YOLO об’єднує ці окремі компоненти в єдину, спільно оптимізовану модель.

2.5.3 Інші швидкі детектори

Fast and Faster R-CNN зосереджується на прискоренні структури R-CNN шляхом обміну обчисленнями та використання нейронних мереж для пропозицій регіону замість вибіркового пошуку.

Незважаючи на підвищення швидкості та точності порівняно з R-CNN, обидва все ще не досягають продуктивності в реальному часі. Багато дослідницьких зусиль спрямовані на прискорення конвеєра DPM, прискорення обчислень HOG, використання каскадів і перенесення обчислень на графічні процесори (GPU). Однак лише DPM 30 Гц справді працює в режимі реального часу. Замість того, щоб намагатися оптимізувати окремі компоненти великого конвеєра виявлення, YOLO повністю відкидає конвеєр і є швидким за своєю конструкцією.

Детектори для певних класів, як-от обличчя або люди, можуть бути дуже оптимізовані, оскільки вони мають справу з набагато меншою кількістю

варіацій. YOLO – універсальний детектор, який вчиться виявляти декілька об’єктів одночасно.

2.5.4 Deep MultiBox

На відміну від R-CNN, К. Сегеді навчив обгорточну нейронну мережу передбачати регіони інтересу [15], минаючи вибірковий пошук. MultiBox також може виконувати виявлення одного об’єкта, замінюючи впевнене передбачення прогнозом одного класу. Однак MultiBox не може виконувати загальне виявлення об’єктів і залишається частиною більшого конвеєра виявлення, який вимагає подальшої класифікації патчів зображень. І YOLO, і MultiBox використовують згорткову мережу для прогнозування обмежувальних рамок на зображенні, але YOLO є повною системою виявлення.

2.5.5 OverFeat

П. Серманет навчив обгорточну нейронну мережу виконувати локалізацію та адаптував цей локалізатор для завдань виявлення [16]. OverFeat ефективно виконує виявлення ковзного вікна, але залишається роз’єднаною системою. OverFeat оптимізує ефективність локалізації, а не виявлення.

Подібно до DPM, локалізатор бачить лише локальну інформацію під час передбачення. OverFeat не може міркувати про глобальний контекст і, отже, вимагає значної постобробки для створення узгоджених виявлень.

2.5.6 MultiGrasp

Дизайн YOLO нагадує роботу Дж. Редмона [17] над виявленням захоплення. Підхід сітки YOLO до передбачення обмежувальної рамки

базується на системі MultiGrasp для регресії до захоплень. Однак виявлення захоплення є набагато простішим завданням, ніж виявлення предметів. MultiGrasp потребує лише прогнозування однієї області захоплення для зображення, що містить один об’єкт. Йому не потрібно оцінювати розмір, розташування чи межі об’єкта або передбачати його клас, а лише знайти область, придатну для захоплення. YOLO передбачає як обмежувальні рамки, так і ймовірності класів для кількох об’єктів кількох класів на зображенні.

2.6 Висновки до розділу 2

У цьому розділі розглядаються методи, що використовуються для виконання поставленого завдання. Алгоритми були детально протестовані і на основі аналізу виявилися найбільш підходящими. Побудова моделі є простою і може бути навчена безпосередньо на повних зображеннях. На відміну від методів, заснованих на класифікаторах, метод You Only Look Once (YOLO) навчається за допомогою функції втрат, яка безпосередньо відповідає ефективності виявлення, і вся модель навчається разом; ця особливість YOLO робить його особливо придатним для завдань, які вимагають швидкого і надійного виявлення об’єктів. Крім того, високо цінується здатність YOLO до узагальнення в нових областях, що робить її ідеальною для застосувань, які вимагають швидкого і надійного виявлення об’єктів.

Крім того, порівняльний аналіз з іншими системами виявлення (наприклад, R-CNN, Fast R-CNN, Faster R-CNN, MultiBox, OverFeat, MultiGrasp) дає подальше розуміння унікальних переваг YOLO та її придатності для вирішення поставлених завдань. Ця дискусія також проливає світло на еволюційну траєкторію алгоритмів виявлення об’єктів і забезпечує всебічний контекст для оцінки внеску YOLO. Загальні висновки, зроблені в цьому розділі, важливі для закладання фундаменту для наступного етапу цього проєкту, де алгоритм YOLO буде впроваджений і оцінений на практиці для досягнення визначених цілей.

3 ЛОКАЛІЗАЦІЯ ОБ`ЄКТІВ ЗОБРАЖЕННЯ ЗАСОБАМИ МОВИ ПРОГРАМУВАННЯ PYTHON

3.1 Створення, ініціалізація та базове налаштування проєкту

Перш ніж зануритися в розробку сценарію, необхідно було створити, ініціалізувати та налаштувати проєкт, визначивши структуру та підготувавши базові класи для обробки даних. Процес розпочався з інсталяції Python на персональному комп'ютері, після чого було налаштовано менеджер пакунків Python, pip. Згодом у зручному місці була створена папка проєкту.

Враховуючи, що розробка сценарію передбачала роботу з пакетами Python – як вбудованими, так і тими, які планується інсталювати, – необхідно було створити середовище для локального зберігання всієї інформації, щоб уникнути захаращення глобального простору комп'ютера та уникнути потенційних конфліктів версій пакетів. Отже, віртуальне середовище було створено за допомогою спеціальної команди (див. рис. 3.1), яка, у свою чергу, створила для нас папку (див. рис. 3.2).

```
1. Windows: `python -m venv venv` | Linux: `python3 -m venv venv`
```

Рисунок 3.1 – Команда для створення віртуального оточення

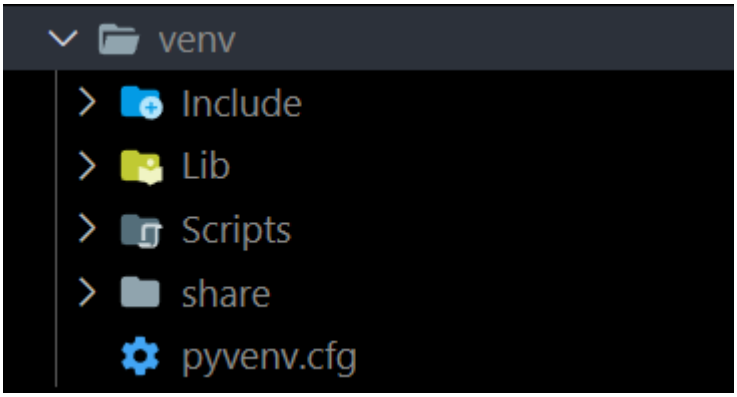


Рисунок 3.2 – Згенерована папка віртуального оточення

Наступний крок включав окреслення структури проєкту для полегшення майбутнього розширення. Папку під назвою «images» було створено для розміщення попередньо підготовленого набору зображень, а також папку «model», яка містить конфігурації, набір класів і попередньо навчені моделі. У кореневому каталозі було створено три файли «.py», які керували б роботою всієї програми.

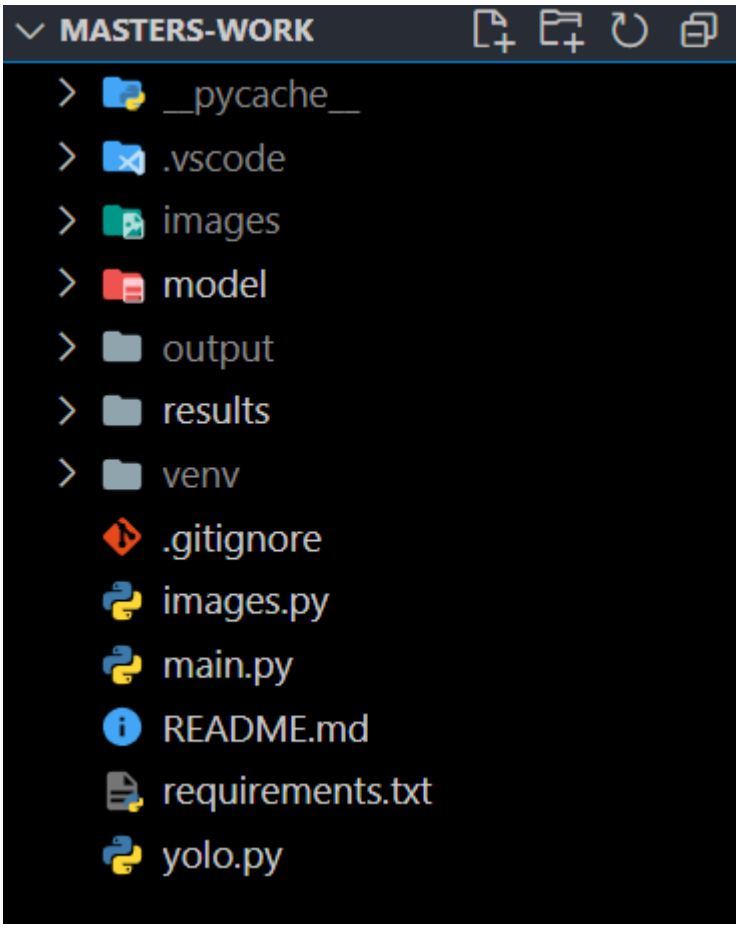


Рисунок 3.3 – Структура проєкту

Останнім кроком на цьому етапі було створення файлу «requirements.txt» для переліку необхідних пакетів разом із їхніми відповідними версіями. Після встановлення цих пакетів структура проєкту набула попередньої форми (див. рис. 3.4).

У подальшому було реалізовано клас для читання зображень із попередньо створеного та заповненого каталогу «images». Відповідний код

містився у файлі «images.py».

Згодом логіка ініціалізації та запуску проєкту була написана в основному файлі «main.py», який слугував початковою точкою. З цього моменту проєкт був створений і можна було розпочати реалізацію основної частини. Весь код буде доступний для перегляду в розділі додатків, а також на GitHub у вигляді репозиторію [18].

```
def get_images(self):
    return [(self.input_folder + '/' + image)
            for image in os.listdir(self.input_folder)]
```

Рисунок 3.4 – Метод для отримання картинок

3.2 Розробка методу розпізнавання об’єктів на зображенні

На початку було важливо підготувати конфігурацію та модель для використання з opencv-python. Вони зберігалися у визначених папках у попередньо створеному каталозі «модель». Для початкового прикладу yolov3.cfg і yolov3.weights були обрані з офіційного сайту YOLO [19].

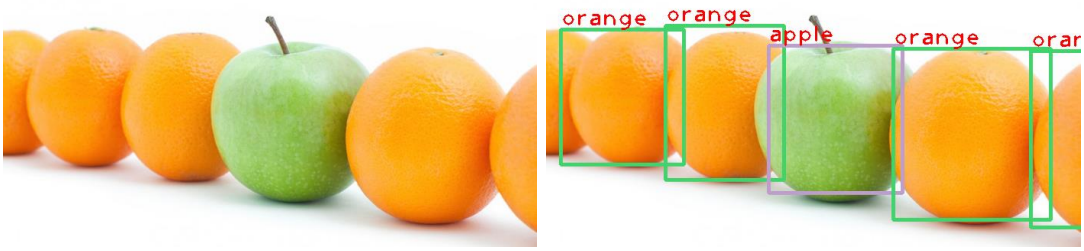
У методі новоствореного класу були згенеровані випадкові кольори, що відповідають розміру масиву імен класів. Ці імена класів були раніше отримані у файлі «main.py». Після цього зчитували зображення, що підлягало обробці, і отримували його розміри. Використовуючи один із методів opencv-python, надаючи йому зображення, конфігурацію та модель, багатовимірний масив із зразками [S,H,W,C] (розмір партії), висота, ширина, канали – фактичний розмір залежить від на архітектурі мережі, було закуплено.

На завершальному етапі, під час ітерації отриманого масиву, навколо кожного ідентифікованого об’єкта були намальовані обмежувальні рамки, які супроводжувалися мітками, що вказували на назву об’єкта. Отримані

зображення зберігаються в каталозі «output».

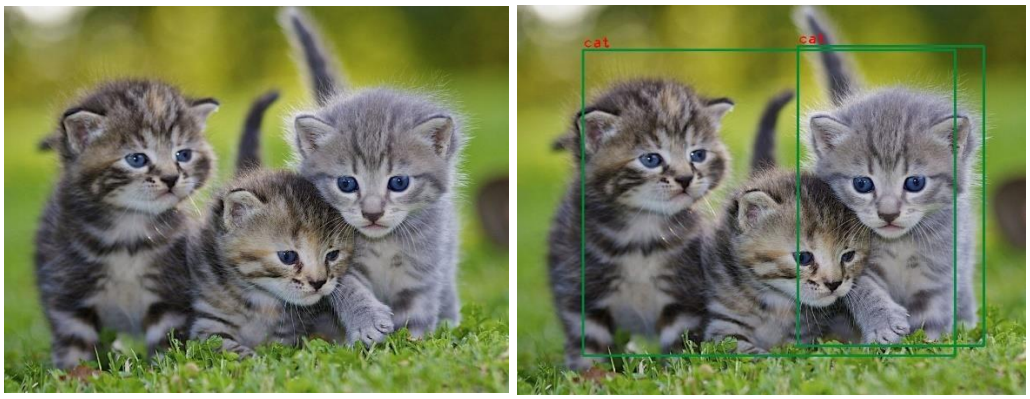
а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.5 – Приклад із моделлю yolov3



а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.6 – Приклад із моделлю yolov3



а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.7 – Приклад із моделлю yolov3

На рисунках 3.5–3.9 показано ефективність моделі в ідентифікації об’єктів на зображеннях. Однак при найближчому розгляді були помітні деякі

неточності, особливо коли кілька об’єктів були згруповані в певній області зображення (див. рис. 3.7). В якості експерименту ця ж модель була перевірена на зображеннях з високою щільністю об’єктів в одній «умовній» комірці.

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.8 – Приклад із моделлю yolov3

Ці приклади показали, що модель зіткнулася з труднощами з великою кількістю об’єктів. В обох сценаріях модель не змогла ідентифікувати всі передбачувані об’єкти, а в першому випадку вона неправильно ідентифікувала деякі.

Спостережувані проблеми можна віднести до якості зображення, точності моделі або кластеризації об’єктів (див. рис. 3.9).

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.9 – Приклад із моделлю yolov3

3.2.1 Модель yolov3-608

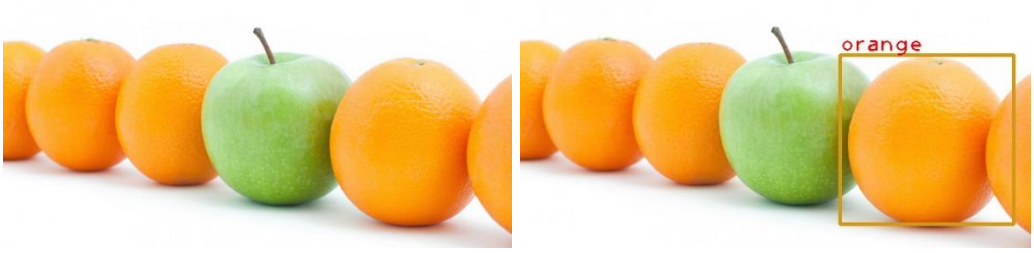
У спробі вирішити виявлені проблеми та підвищити точність розпізнавання об’єктів дослідження призвело до тестування альтернативної моделі. Офіційний вебсайт YOLO надає кілька варіантів моделей із різними середніми рівнями точності, серед яких модель yolov3-608 рекламується як найточніша згідно з їх тестами.

Щоб переконатися в правдивості цих тверджень, модель yolov3-608 була розгорнута на тому ж наборі зображень, які створювали проблеми для попередньої моделі (див. рис. 3.11). Результати, однак, не показали істотного покращення. Цей застій у продуктивності спонукав дослідити, чи справді модель була основною причиною помічених неточностей, чи вплинули інші фактори.

Під час подальшого дослідження було виявлено кілька факторів, які потенційно можуть вплинути на продуктивність розпізнавання об’єктів. Це охоплювало якість зображення, щільність об’єктів на зображенні та властиві обмеження самої моделі в обробці таких сценаріїв. Наприклад, було помічено, що зображення з нижчою роздільною здатністю або високим ступенем кластеризації об’єктів представляють складний ландшафт для моделі, щоб точно ідентифікувати та локалізувати кожен об’єкт (див. рис. 3.10).

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.10 – Приклад із моделлю yolov3-608



а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.11 – Приклад із моделлю yolov3-tiny

Більше того, глибше занурення в архітектуру та параметри моделі yolov3-608 показало, що хоча вона може похвалитися вищою середньою точністю, вона також вимагає більше обчислювальних ресурсів. Цей компроміс між точністю та обчислювальною ефективністю є типовою дилемою в галузі розпізнавання об’єктів. Було також виявлено, що продуктивність моделі можна потенційно покращити шляхом точного налаштування її параметрів або використання додаткових етапів попередньої та постобробки зображень.

Крім того, було розпочато дослідження альтернативних моделей і методологій розпізнавання об’єктів для пошуку кращих рішень для конкретних проблем, що виникають у цьому проєкті. Ця спроба була спрямована не лише на підвищення точності розпізнавання об’єктів, але й на отримання більш глибокого розуміння основної механіки та сучасного стану технології розпізнавання об’єктів.

3.2.2 Моделі yolov3-tiny та yolov3-spp

Щоб далі заглибитися в тонкощі розпізнавання об’єктів і підтвердити результати попередніх оцінок моделей, було визнано доцільним поекспериментувати з менш складними моделями, доступними на тому ж офіційному вебсайті YOLO. Дві такі моделі, yolov3-tiny і yolov3-spp, були обрані для цієї спроби з метою ретельного вивчення того, як відмінності в складності моделі впливають на точність і ефективність розпізнавання об’єктів.

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.12 – Приклад із моделлю yolov3-spp

Результати прояснили важливий аспект: рівень підготовки та властива архітектура моделі глибоко впливають на кінцевий результат. yolov3-tiny, будучи легшою моделлю, демонструє вищу швидкість обробки, хоча й ціною точності. Навпаки, yolov3-spp зі складною структурою продемонстрував кращу точність, але вимагає більших обчислювальних ресурсів.

Зіставлення цих моделей із yolov3-608 підкреслило важливий компроміс між точністю, швидкістю обробки та вимогами до обчислювальних ресурсів. Це пролило світло на той факт, що хоча більш складна модель потенційно може запропонувати вищу точність, вона також може створити проблеми в обробці в реальному часі або в середовищах з обмеженими обчислювальними ресурсами.

Крім того, спроба натякнула на можливість того, що альтернативний алгоритм або інший підхід до розпізнавання об’єктів може дати кращі результати. Це спонукало до дослідження інших найсучасніших алгоритмів і методів, які могли б потенційно подолати обмеження, спостережувані в моделях YOLO.

Після занурення глибше з’явилося кілька альтернативних алгоритмів і методологій, які можна було б вивчити. Наприклад, дослідження моделей машинного навчання з різними навчальними даними, використання методів ансамблю для поєднання сильних сторін декількох моделей або використання удосконалених методів попередньої обробки для покращення якості

зображення перед подачею його в модель були одними з міркувань.

Виходячи за межі моделей YOLO та розглядаючи поєднання різних алгоритмів і технологій, цей розділ закладає основу для більш вичерпного дослідження, спрямованого на досягнення підвищеної точності та ефективності локалізації об’єктів зображення за допомогою інструментів мови програмування Python.

3.3 Порівняння результатів тестування з іншими рішеннями

На етапі експерименту стало очевидно, що ступінь навчання моделі суттєво впливає на кінцеві результати в задачах розпізнавання об’єктів. Крім того, якість зображень також відігравала вирішальну роль у визначенні точності моделей. Виникло доречне запитання – чи можуть альтернативні алгоритми підвищити точність розпізнавання (див. рис. 3.13)? Щоб відповісти на це питання, було досліджено кілька готових до використання вебрішень [20].

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.13 – Приклад з алгоритмом SSD

Перше рішення передбачало тестування алгоритму Single Shot Multibox Detector (SSD) [20]. Хоча результати були не такими вражаючими, вони дали інший погляд на те, як різні алгоритми поводяться з тим самим набором

зображень.

Згодом було проведено порівняння з іншим вебрішенням, яке використовує той самий алгоритм (YOLOv3), що й у цій курсовій роботі, але зі спеціальною моделлю [21] (див. рис. 3.14).

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.14 – Приклад з кастомною моделлю

Користувальницький моделі вдалося виявити значно більше об’єктів на зображенні порівняно з попередніми спробами, але за помітним винятком – майже все, за винятком брокколі, було ідентифіковано неправильно. Це спостереження ще раз підтвердило важливість навчання моделі та її майстерності в узагальненні для різних типів об’єктів.

Дослідження також поширилося на тестування зображень за допомогою нової версії алгоритму YOLO, YOLOv8, який, за словами розробників, може похвалитися вищою точністю та швидкістю (див. рис. 3.15). З кількома рядками коду Python і наявними зображеннями YOLOv8 було випробувано [22].

Результати продемонстрували, що ця версія алгоритму справді була більш точною, ніж її попередники. Однак не обійшлося без недоліків. Не всі об’єкти були ідентифіковані, а один об’єкт у центрі було ідентифіковано неправильно – це була морква, а не апельсин.

Це підкреслює висновок про те, що на точність впливає поєднання різних факторів, а не лише окремий аспект.

а) Рисунок до обробки б) Рисунок після обробки

Рисунок 3.15 – Приклад з алгоритмом YOLOv8

У цій главі не лише окреслено порівняльну продуктивність різних моделей і алгоритмів, але й підкреслено багатогранний характер проблем, притаманних задачам розпізнавання об’єктів. Було підкреслено, що точність локалізації об’єктів є функцією багатьох елементів, включаючи архітектуру моделі, якість навчальних даних, роздільну здатність і чіткість зображень і, можливо, використовуваний алгоритмічний підхід [18].

Глибше розуміння цих факторів впливу та того, як вони взаємодіють, буде вирішальним у розробці більш надійних рішень для локалізації об’єктів за допомогою інструментів мови програмування Python. Крім того, розуміння, отримане в результаті цих порівнянь, прокладає шлях для подальших досліджень і оцінок нових або гібридних моделей і алгоритмів для підвищення точності та ефективності локалізації об’єктів на зображеннях.

3.4 Висновки до розділу 3

У нещодавній главі було успішно розроблено та впроваджено схему виявлення об’єктів на зображеннях. Схему оцінювали за допомогою різних попередньо навчених моделей, кожна з яких мала різний рівень кваліфікації для виконання завдання. Точність моделей широко варіювалася в різних сценаріях,

причому деякі моделі навіть досягали тривожних показників помилок.

Також було проведено порівняльний аналіз з іншими версіями алгоритму YOLO та з комерційно доступними мережевими рішеннями. Результати останніх виявилися невтішними і занадто «сирими», щоб їх можна було назвати готовими рішеннями.

Однак версія 8 алгоритму YOLO від ultralytics дала дуже хороші результати. Цікаво, що ці результати дещо відрізнялися залежно від використовуваної моделі.

Ця різниця підкреслює важливість вибору та навчання моделі для досягнення більшої точності в задачі виявлення об’єктів.

Дослідження в цій главі висвітлює широкий спектр можливостей і нюансів використання різних алгоритмів і моделей для виявлення об’єктів. Це створює міцну основу для поглибленого вивчення передових моделей та альтернативних підходів, які можуть допомогти підвищити точність локалізації об’єктів на зображеннях за допомогою засобів мови програмування Python.

ВИСНОВКИ

Результатом кваліфікаційної роботи стала успішна реалізація алгоритму локалізації об’єктів зображення за допомогою мови програмування Python. Для цієї роботи було обрано пакети numpy та opencv-python, а також мову Python та використано алгоритм YOLOv3. Було підготовлено набір зображень як основу для аналізу точності та ефективності моделі.

Ідентифікація включала детальне вивчення цільової області та порівняння існуючих алгоритмів для визначення найбільш ефективного. На етапі реалізації розроблені скрипти порівнювалися з результатами тестування комерційних рішень та вебдодатків.

Хоча розроблені скрипти готові до використання, важливо зазначити, що результати можуть бути не на 100% точними, оскільки обрана модель не є найточнішою моделлю в сімействі моделей YOLO. Виходячи зі змісту розділу 3 цієї роботи та порівняльного аналізу, найбільш ефективним рішенням на даний момент є YOLOv8.

У майбутньому модель може бути вдосконалена шляхом покращення набору даних, на яких навчається вхідна модель. Враховуючи, що якість вхідних даних також впливає на результати, доцільно додати категорії для обробки та відбору зображень. Це відкриє шлях до подальшого вдосконалення процесу локалізації об’єктів і забезпечить більш надійне і точне рішення для локалізації об’єктів на зображеннях за допомогою мови програмування Python.

ПЕРЕЛІК ПОСИЛАНЬ

1. ImageNet Large Scale Visual Recognition Challenge / O. Russakovsky, J. Deng and otc. *Cornell University*. 2014. 43 p. URL: https://arxiv.org/abs/1409.0575 (дата звернення: 05.07.2023).
2. Girshick R., Donahue J., Darrell T., Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. *Cornell University*. 2013. URL: https://arxiv.org/abs/1311.2524 (дата звернення: 07.07.2023).
3. Girshick R. B. Fast R-CNN. *Cornell University*. 2015. URL: https://arxiv.org/abs/1504.08083 (дата звернення: 07.07.2023).
4. He K., Zhang X., Ren S., Sun J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *Cornell University*. 2014. URL: https://arxiv.org/abs/1406.4729 (дата звернення: 08.07.2023).
5. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Cornell University*. 2015. URL: https://arxiv.org/abs/1506.01497 (дата звернення: 08.07.2023).
6. He K., Gkioxari G., Dollár P., Girshick R. Mask R-CNN. *Cornell University*. 2017. URL: https://arxiv.org/abs/1703.06870 (дата звернення: 09.07.2023).
7. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. *Cornell University*. 2015. URL: https://arxiv.org/abs/1506.02640 (дата звернення: 09.07.2023).
8. Redmon J., Farhadi A. YOLO9000: Better, Faster, Stronger. *Cornell University*. 2016. URL: https://arxiv.org/abs/1612.08242 (дата звернення: 09.07.2023).
9. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. *Cornell University*. 2018. URL: https://arxiv.org/abs/1804.02767 (дата звернення: 10.07.2023).
10. Felzenszwalb P. F., Girshick R. B., McAllester D., Ramanan D. Object detection with discriminatively trained part based models. *IEEE Transactions*

on Pattern Analysis and Machine Intelligence. 2010. 32(9) P. 1627–1645.

11. Girshick R., Donahue J., Darrell T., Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference*. 2014. P. 580–587.
12. The pascal visual object classes challenge / M. Everingham, S. M. A. Eslami and otc. *A retrospective. International Journal of Computer Vision*. 2015. 111(1). P. 98–136.
13. Going deeper with convolutions / C. Szegedy, W. Liu and otc. *CoRR*. 2014. URL: https://arxiv.org/abs/1409.4842 (дата звернення: 12.08.2023).
14. Uijlings J. R., K. E. van de Sande, Gevers T., Smeulders A. W. Selective search for object recognition. *International journal of computer vision*. 2013. 104(2). P. 154–171.
15. Erhan D., Szegedy C., Toshev A., Anguelov D. Scalable object detection using deep neural networks. *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference*. 2014. P. 2155–2162.
16. Overfeat: Integrated recognition, localization and detection using convolutional networks / P. Sermanet, D. Eigen and otc. *CoRR*. 2013. URL: https://arxiv.org/abs/1312.6229 (дата звернення: 13.08.2023).
17. Redmon J., Angelova A. Real-time grasp detection using convolutional neural networks. *CoRR*. 2014. URL: https://arxiv.org/abs/1412.3128 (дата звернення: 14.08.2023).
18. Levchenko D. Localization of the Image Objects Based on the Python Programming Language. *Github*. 2023. URL: https://github.com/xokcton/masters-work/tree/master (дата звернення: 15.08.2023).
19. YOLO: Real-Time Object Detection. *Pjreddie.com*. 2023. URL: https://pjreddie.com/darknet/yolo/ (дата звернення: 13.08.2023).
20. Free Online Object Detection. *ASPOSE*. 2023. URL: https://products.aspose.app/imaging/object-detection (дата звернення: 14.08.2023).

21. Iashin V. YOLO v3 Object Detector. 2023. URL: https://iashin.ai/detector (дата звернення: 14.08.2023).
22. Levchenko D. Yolov8. *Google Colab*. 2023. URL: https://colab.research.google.com/drive/1k4784TtkH11k4w6mgKh_YKSrldYUE7HY?usp=sharing (дата звернення: 13.08.2023).

ДОДАТОК А

Клас Images

```
import os
import random
import string

class Images:
    def __init__(self, input_folder, output_folder):
        self.input_folder = input_folder
        self.output_folder = output_folder

        if not os.path.exists(self.output_folder):
            os.makedirs(self.output_folder)

    def get_images(self):
        return [(self.input_folder + '/' + image)
                for image in os.listdir(self.input_folder)]

    def get_random_image_name(self, length):
        letters = string.ascii_lowercase
        return ''.join(random.choice(letters) for i in range(length))
```

ДОДАТОК Б

Клас Yolo

```
import cv2
import numpy as np

class Yolo:
    def __init__(self, classes, cfg_path, weights_path):
        self.classes = classes
        self.yolo = cv2.dnn.readNet(cfg_path, weights_path)

    def detect(self, input_image, output_image):
        COLORS = np.random.randint(0, 255, size=(
            len(self.classes), 3), dtype="uint8")
        layer_names = self.yolo.getLayerNames()
        output_layers = [layer_names[i - 1]
                         for i in self.yolo.getUnconnectedOutLayers()]

        redColor = (0, 0, 255)
        greenColor = (0, 255, 0)

        # Loading Images
        name = input_image
        img = cv2.imread(name)
        height, width, channels = img.shape

        # Detecting objects
        blob = cv2.dnn.blobFromImage(
            img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
```

```
self.yolo.setInput(blob)
outputs = self.yolo.forward(output_layers)

class_ids = []
confidences = []
boxes = []
for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
for i in range(len(boxes)):
    if i in indexes:
        color = [int(c) for c in COLORS[class_ids[i]]]
        x, y, w, h = boxes[i]
        label = str(self.classes[class_ids[i]])
```

```
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 3)
        cv2.putText(img, label, (x, y - 5),
                    cv2.FONT_HERSHEY_PLAIN, 2, redColor, 2)

cv2.imwrite(output_image, img)
```

ДОДАТОК В

Код запуску програми

```
from yolo import Yolo
from images import Images

import os

def main():
    cfg_path = os.path.join('.', 'model', 'cfg', 'yolov3-spp.cfg')
    weights_path = os.path.join('.', 'model', 'weights', 'yolov3-spp.weights')
    coco_names = os.path.join('.', 'model', 'coco.names')
    output_folder = 'output'
    classes = []

    with open(coco_names, "r") as file:
        classes = [line.strip() for line in file.readlines()]

    y = Yolo(classes=classes, cfg_path=cfg_path, weights_path=weights_path)
    i = Images('images', output_folder)
    imgs = i.get_images()

    for img in imgs:
        output_name = output_folder + '/' + \
            i.get_random_image_name(10) + '-' + img.split('/')[-1]
        print(output_name)
        y.detect(img, output_name)

if __name__ == "__main__":
```

```
main()
```

ДОДАТОК Г

Приклад коду для роботи з YOLOv8

```
!pip install ultralytics

from ultralytics import YOLO
import os

directory_path = './images/'
output_path = '../../output/'
directory_files = os.listdir(directory_path)

for file in directory_files:
  model = YOLO('yolov8n.pt') # yolov8n.pt | yolov8l.pt
  result = model(directory_path+file, save=True, name=output_path+file.split(".")[0])
```