Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Програмна система пошуку зображень з використанням нейронних мереж»

Виконав: студент II курсу
групи 1ПІ-22М спеціальності
121 – Інженерія програмного забезпечення
Бобко О.Л.

Керівник: к.т.н., доц. каф. ПЗ Рейда О.М.
«12» грудня 2023 р.

Опонент: к.т.н., доц. каф. Войцеховська О. В.
«12» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«12» 12 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____ Романюк О.Н.
« 19 » вересня 2023 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бобку Олексію Леонідовичу

1. Тема роботи – Програмна система пошуку зображень з використанням нейронних мереж.

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи
_____5 грудня 2023 р._____

3. Вихідні дані до роботи: Покращений алгоритм пошуку та класифікації об'єктів у зображенні; середовище розробки – Eclipse IDE; мова розробки – Python; операційна система – Windows 10; вихідні дані – програмна система пошуку зображень з використанням нейронних мереж.

4. Зміст текстової частини: дослідження існуючих способів побудови маршруту; постановка задачі дослідження; розробка структури системи; програмна реалізація системи; тестування системи; економічна частина.

5. Перелік ілюстративного матеріалу: контекстна діаграма системи; контейнерна діаграма системи; компонентна діаграма системи; скріншоти роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 4	Рейда О. М., к.т.н., доцент кафедри ПЗ	19.09.2023	05.12.2023
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	23.11.2023	01.12.2023

7. Дата видачі завдання 19 вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	При
1	Аналіз методів пошуку та розпізнавання зображень	20.09. 2023 – 09.10.2023	
2	Проектування системи	10.10. 2023 – 29.10.2023	
3	Програмна розробка системи	30.10. 2023 – 14.11.2023	
4	Тестування системи	15.11. 2023 – 24.11.2023	
5	Економічна частина	25.11. 2023 – 05.12.2023	

Студент _____ **Бобко О.**
(підпис) (прізвище та ініці

Керівник магістерської кваліфікаційної роботи _____ **Рейда О.М.**
(підпис) (прізвище та іні

АНОТАЦІЯ

Бобко О.Л. Програмна система пошуку зображень з використанням нейронних мереж: магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення – Вінниця: ВНТУ, 2023. 105с.

На укр. мові. Бібліогр. : 30 назв ; Рисунок : 34; табл. 9.

У магістерський кваліфікаційній роботі було розроблено систему для пошуку та розпізнавання зображень.

Для вирішення поставленої задачі було проведено аналіз методів для реалізації пошуку та розпізнавання зображень. Основну увагу приділено підходу використанню нейронних мереж.

В результаті реалізовано удосконалену нейронну мережу, яка здатна стійко розпізнавати зображення, що були піддані будь-якому спотворенню. Проведено порівняння з аналогами. Розроблено система може бути використана для пошуку зображень у великих базах даних. Вона може бути застосована в різних галузях, зокрема, в системах безпеки, візуального контролю та управління.

Ключові слова: нейронні мережі, бази даних, комп’ютерний зір, Python, SQLite.

ABSTRACT

Bobko O.L. Software system for image search using neural networks. Vinnitsa: VNTU, 2023. 105 p.

In Ukrainian language. Bibliographer: 30 titles ; fig. : 34; tabl. 9.

In the master’s qualification thesis, a system for image search and recognition was developed. To address the stated task, a detailed analysis of methods for implementing image search and recognition was conducted, with a primary focus on the use of neural networks.

As a result, an enhanced neural network was implemented, capable of robustly recognizing images subjected to various distortions. A comparison with existing analogs was performed. The developed system can be utilized for image search in large databases and finds applications in various fields, including security systems, visual control, and management.

Keywords: neural networks, databases, computer vision, Python, SQLite.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Комп'ютерні технології дозволяють автоматизувати широке коло справ і процесів, які раніше покладалися тільки на людину. Інформаційні технології використовуються всюди: в освіті, промисловості, транспорті та інших галузях. Програмісти всього світу розробляють нові та удосконалюють вже існуючі рішення для автоматизації.

Задачі розпізнавання образів набувають все більшої популярності завдяки необхідності автоматизації функцій контролю і управління складними динамічними об'єктами в реальному часі, а також образних процесів комунікації в інтелектуальних системах. Комп'ютерний зір – це область, яка набуває все більшої популярності та розвивається завдяки розвитку комп'ютерної техніки та штучного інтелекту. Один із актуальних завдань в комп'ютерного зору – порівняння зображень та знаходження зображень одного типу. Значна частина цього напрямку присвячена практичному застосуванню методів, таких як SIFT, SURF, BRIEF та SNN. Застосування комп'ютерного зору є дуже різноманітним, охоплюючи системи безпеки, контроль виготовлення об'єктів, системи візуального контролю та управління, контроль транспортних засобів. В товарно–орієнтованій промисловості системи машинного зору використовуються для пошуку схожих зображень та їх класифікації. Нейромережевих парадигм для вирішення задач розпізнавання образів запропоновано багато. Однак, значними складностями при розпізнаванні є образи, що були піддані будь-якому спотворенню: зсуву, повороту, масштабуванню.

Тому актуальними є питання розробка програмної системи пошуку зображень з використанням нейронних мереж

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Програмна система пошуку зображень з використанням нейронних мереж

Мета та завдання дослідження. Метою даної роботи є розробка програмної системи пошуку зображень з використанням нейронних мереж для підвищення швидкодії пошуку графічних зображень.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів розробки програмних систем пошуку зображень з використанням нейронних мереж;
- модифікувати такі методи:
  - метод визначення оптимальних вхідних параметрів нейронної мережі за допомогою Баєсової оптимізації;
  - метод оптимізації функції втрат за допомогою градієнтного спуску;
- розробити програмні компоненти та систему візуалізації на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів пошуку зображень з використанням нейронних мереж.

Об’єкт дослідження – процес пошуку зображень з використанням нейронних мереж.

Предмет дослідження – засоби та методи пошуку зображень з використанням нейронних мереж.

Методи дослідження – У процесі досліджень використовувались: теорія чисел та чисельних методів, теорія диференціально-інтегрального числення, лінійна алгебра, методи аналітичної геометрії для розробки моделей та методів, статистичні методи розпізнавання зображень, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод визначення оптимальних вхідних параметрів нейронної мережі за допомогою Баєсової оптимізації, що на відміну від існуючих, використовує характеристичні параметри вхідних об’єктів, для підвищення ефективності роботи моделі.

2. Подальшого розвитку отримав метод оптимізації функції втрат за допомогою градієнтного спуску, що на відміну від існуючих має меншу кількість ітерацій пошуку оптимальних значень для підвищення швидкодії процесу навчання.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано методи та розроблено програмні засоби пошуку зображень з використанням нейронних мереж.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: Використання тензорів у машинному навчанні [1]; Використання CPU, GPU та TPU для тренування нейронних мереж [2]; комп'ютерна програма для визначення, класифікації та пошуку графічних зображень.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на всеукраїнських конференціях: VI Всеукраїнській науково-практичній інтернет-конференції молодих вчених та студентів «Сучасні інформаційні системи та технології» (Херсонський національний технічний університет, 2023); Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії, секція програмного забезпечення (Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у матеріалах конференцій.

1 АНАЛІЗ МЕТОДІВ ПОШУКУ ТА РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

У машинному навчанні, розпізнаванні образів і обробці зображень, важливим етапом є виділення ознак з вихідних даних. Цей процес полягає в тому, щоб перетворити початковий набір вимірювань у більш інформативний та менший за розмірністю набір, який спрощує подальші кроки навчання та узагальнення. Окрім того, цей процес може зробити дані більш зрозумілими для людей. Важливо зазначити, що виділення ознак також пов’язане зі зменшенням розмірності даних.

Методи розпізнавання графічних образів можна умовно поділити на три групи, які складаються з попередньої фільтрації та підготовки зображення, логічної обробки результатів фільтрації і алгоритмів прийняття рішень на основі логічної обробки. Межі між цими групами можуть бути досить умовними, оскільки для вирішення конкретних завдань розпізнавання можуть знадобитися різні поєднання методів з цих груп. Іноді для досягнення бажаного результату досить використовувати методи лише з однієї з цих груп.

Перша група методів – це методи фільтрації. Вони спрямовані на виділення областей на зображенні без їх докладного аналізу. Багато з цих методів застосовують однакове перетворення до всіх точок зображення, і на цьому етапі аналіз зображення не проводиться. Однак точки, які успішно пройшли фільтрацію, можна розглядати як області з особливими характеристиками.

Друга група методів стосується пошуку особливих точок, які залишаються стабільними при незначних змінах освітлення та невеликих рухах об’єктів. Ці точки часто використовуються для навчання і класифікації типів об’єктів. Наприклад, класифікатори для розпізнавання пішоходів або людей часто будуються на основі таких точок. Деякі з відомих методів вейвлетів можуть служити основою для знаходження таких точок. До цієї групи можна віднести точки, знайдені за допомогою гістограм спрямованих градієнтів та інші методи.

Третя група методів стосується стабільних точок, які можна знаходити навіть при повороті зображення. Методи, такі як SURF і SIFT, здатні знаходити особливі

точки, які залишаються стабільними навіть при різних трансформаціях, таких як повороти та зміни масштабу.

Ці методи розпізнавання графічних образів різних груп можуть бути поєднані для вирішення конкретних завдань і використовуватися в залежності від характеристик зображення та потреб застосування.

Також потрібно не забувати, що коли вхідні дані для алгоритму занадто великі для ефективної обробки та вони мають підозру на надмірність або непотрібність, то ці дані можуть бути трансформовані в скорочений набір ознак. Цей процес називається виділенням ознак. Мета полягає в тому, щоб отримані ознаки містили важливу інформацію з вхідних даних, щоб задачу можна було вирішити з використанням цього спрощеного представлення замість повних початкових даних.

1.1 Класичні методи розпізнавання зображень

Існує декілька найбільш поширених методів розпізнавання зображень серед них – SIFT, SURF, ORB, та AKAZE. Це цілком різна алгоритми які використовуються у комп’ютерному баченні. Власне це алгоритми виявлення та опису ключових точок зображення. Дані алгоритми зазвичай використовуються у таких задачах як: розпізнавання, «зшивання» зображень, та виявлення особливостей. Розглянемо деякі з них [3].

Метод «SIFT» (Scale-Invariant Feature Transform) – це алгоритм комп’ютерного зору, призначений для виділення і опису ключових особливостей зображень. Розроблений Девідом Лоу в 1999 році, цей метод виявився дуже ефективним у вирішенні завдань розпізнавання об’єктів та вирішенні проблеми інваріантності до масштабу та орієнтації.

Один з головних плюсів методу SIFT полягає в тому, що він забезпечує інваріантність до змін масштабу та орієнтації, що робить його ефективним для розпізнавання об’єктів на зображеннях навіть при різних умовах.

Метод SIFT став одним із основних алгоритмів у комп’ютерному зорі і застосовується в багатьох видах застосувань, включаючи розпізнавання обличь, відстеження об’єктів, панорамне зшивання зображень та багато інших задач, де

необхідно робити висновки на основі вмісту зображень. Блок-схема принципу роботи алгоритму SIFT представлено на рисунку 1.1.

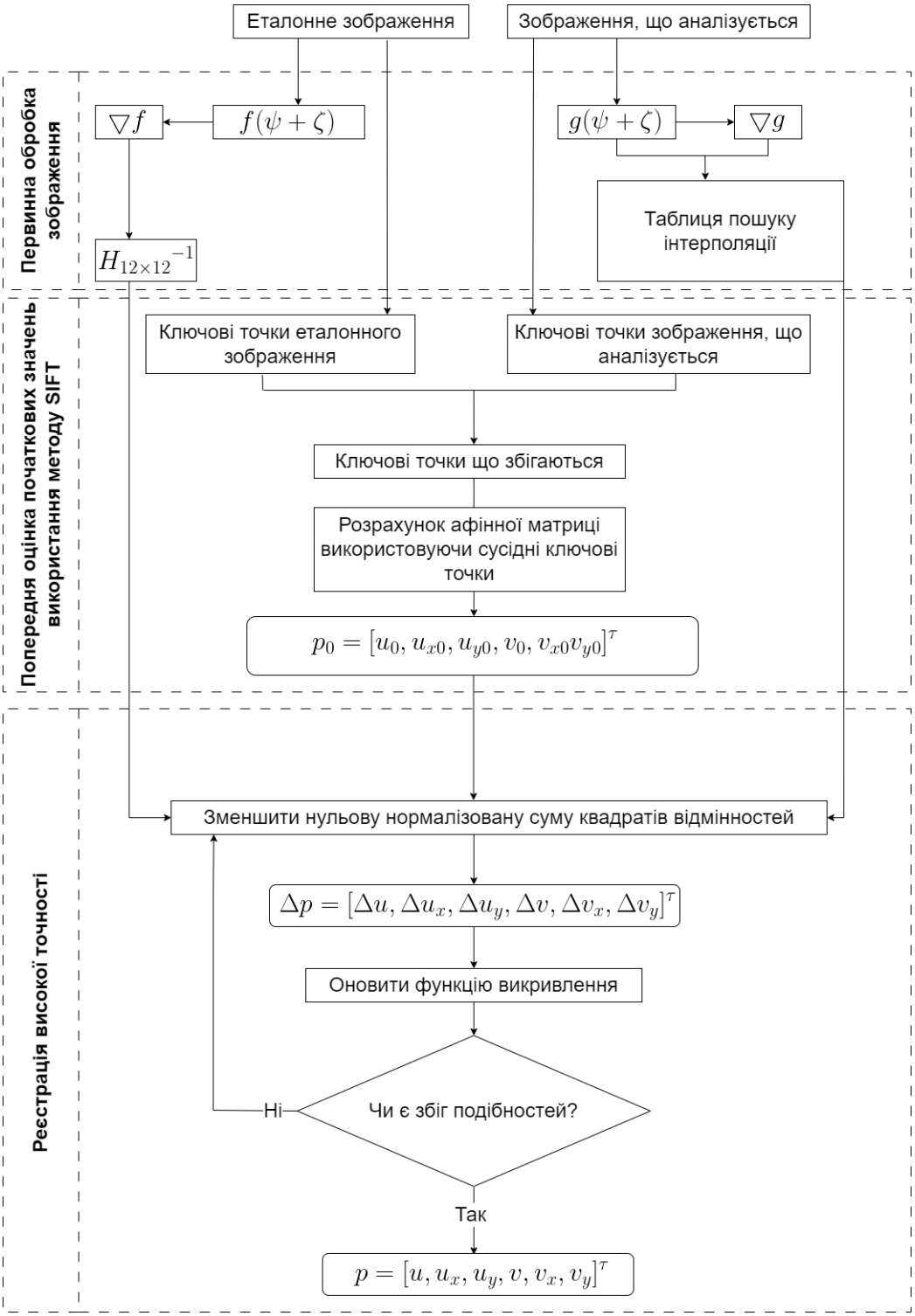

Рисунок 1.1 – Блок-схема роботи алгоритму SIFT

Опис алгоритму SIFT відповідно до блок-схеми:

1. Введення зображень – зчитування файлів зображень.
2. Створення різниці гаусіанів – створення гаусіанів дозволяє визначити ключові точки що залишаються стабільні при зміні масштабу зображення.
3. Визначення ключових точок – визначення позицій ознак на зображенні.
4. Визначення направлення – направлення точок ознак допомагає визначити не лише ключову точку а й локальну інформацію навколо точки.
5. Створення дескрипторів ключових точок – створення дескрипторів базується на інформації направлення ключових точок та являє собою закодовану інформацію про точку та локальний простір навколо неї.
6. Аналіз ступеня співпадіння – етап порівняння визначає степінь збігу оригінального зображення з зображенням, яке аналізується. Відповідно до визначеної функції порівняння приймається рішення чи є співпадіння чи ні.
7. Об’єкт знайдено – означає що оригінальне зображення містить достатню кількість співпадінь за ключовими точками, що і змінене зображення, тобто зображення яке аналізується.
8. Об’єкт не знайдено – відповідно до попереднього пункту порівняння зображень не виявило достатню кількість збігів за ключовими точками.

Метод SURF (Speeded-Up Robust Features) – це алгоритм обробки зображень та комп’ютерного зору, який спеціалізується на виділенні та описі ключових особливостей на зображеннях. Цей метод був розроблений Гербертом Байєм та Віктором Лепеттем у 2006 році та є покращеною версією методу SIFT (Scale-Invariant Feature Transform). Схема роботи SURF зображено на рисунку 1.2.

Опис схеми роботи:

1. Введення зображень – зчитування файлів зображень.
2. Створення інтегрованих зображень – побудова таблиці сум пікселів.
3. Визначення ключових точок – визначення позицій ознак на зображенні.
4. Створення дескрипторів – побудова структури даних що містить інформацію про точку та локальний простір навколо неї.

5. Порівняння ключових точок зображень – пошук збігів ключових точок між зображеннями

6. Об’єкт знайдено – означає що оригінальне зображення містить достатню кількість співпадінь за ключовими точками, що і змінене зображення, тобто зображення яке аналізується.

7. Об’єкт не знайдено – відповідно до попереднього пункту порівняння зображень не виявило достатню кількість збігів за ключовими точками.

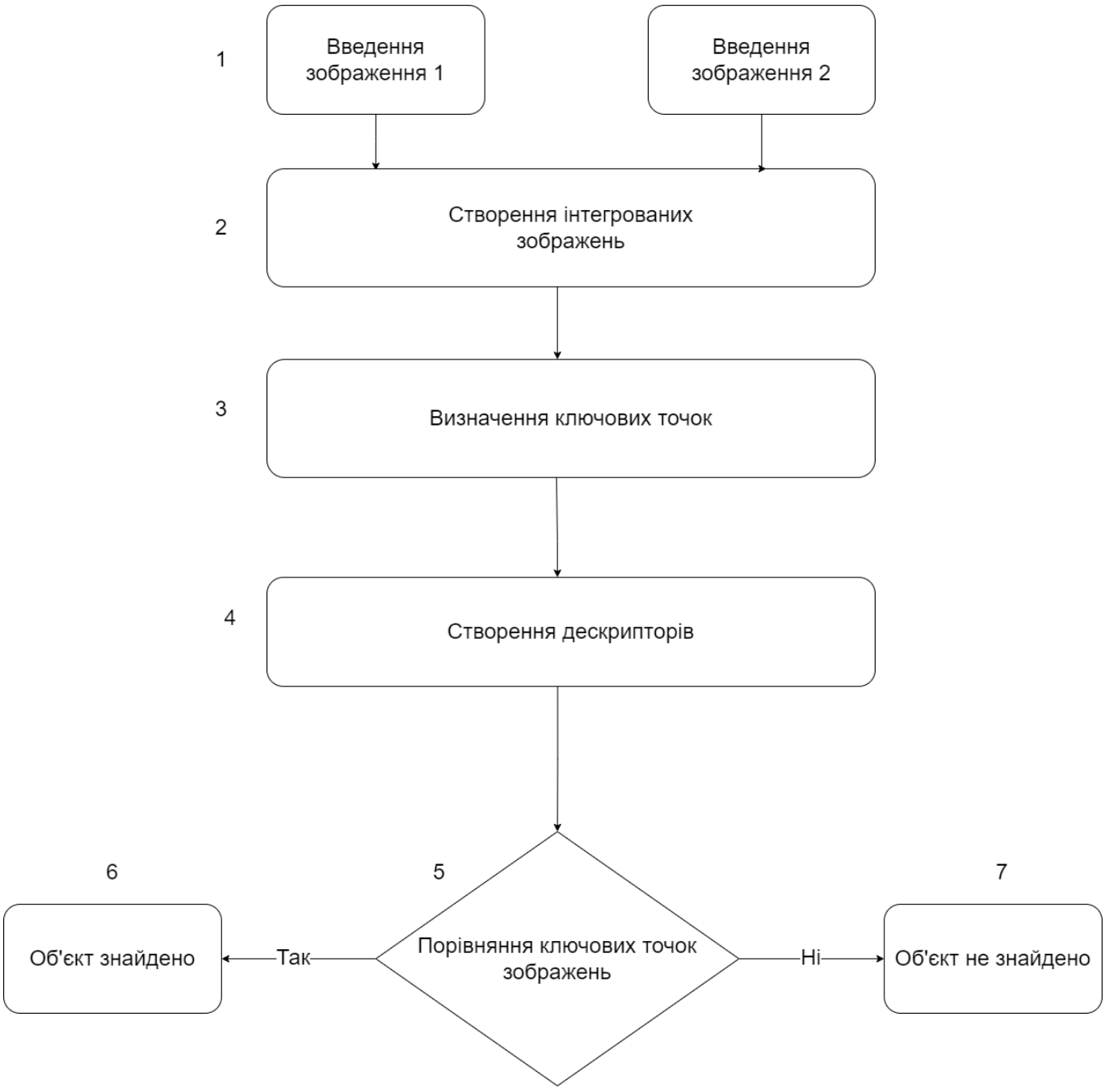

Рисунок 1.2 –Схема роботи SURF

Основні характеристики методу SURF включають:

- Швидкість обчислень: Однією з ключових переваг SURF є його висока швидкість обчислень. Це досягається завдяки використанню ефективних методів обчислення градієнтів та оцінки Гессе, які дозволяють досить швидко аналізувати зображення.
- Інваріантність до масштабу та орієнтації: SURF, подібно до методу SIFT, забезпечує інваріантність до змін масштабу та орієнтації об’єктів на зображенні. Це означає, що він може виявляти та описувати ключові особливості незалежно від їх розміру та орієнтації.
- Стійкість до змін контрастності: SURF демонструє стійкість до змін контрастності на зображеннях. Це важливо, оскільки об’єкти можуть змінювати освітлення або контраст, і SURF все одно може ефективно їх виявляти та описувати.
- Дескриптори ключових точок: SURF генерує дескриптори для кожної ключової точки, які дозволяють описати оточення цих точок на зображенні. Ці дескриптори є числовими векторами, які відображають структуру та особливості об’єктів у околицях ключових точок [4].

SURF знайшов широке застосування в комп’ютерному зорі та розпізнаванні образів. Він використовується для різних завдань, включаючи розпізнавання обличь, відстеження об’єктів на відео, виявлення об’єктів на зображеннях та багато інших застосувань. SURF є потужним і швидким інструментом для виділення ключових особливостей на зображеннях, який допомагає вирішувати різноманітні завдання в області комп’ютерного зору. Для більшого розуміння принципів роботи і відмінності деяких алгоритмів необхідно дослідити відмінності деяких алгоритмів. До прикладу порівняльна характеристика ефективності розпізнавання зображення зі зміненим масштабом за допомогою алгоритмів SIFT, SURF та ORB приведена у таблиці 1.1 [5]. На таблиці 1.2 приведено порівняльну характеристику ефективності розпізнавання зображення з нахиленим зображенням за допомогою алгоритмів SIFT, SURF та ORB

Таблиця 1.1 – Порівняльна характеристика ефективності розпізнавання зображення зі зміненим масштабом за допомогою алгоритмів SIFT, SURF та ORB

	Час (сек)	Ключових точок, оригінальне зображення. (тис. шт.)	Ключових точок, змінене зображення. (тис. шт).	Кількість збігів	Співпадіння. (%)
SIFT	0.25	248	1210	232	31.8
SURF	0.8	162	581	136	36.6
ORB	0.02	261	471	181	49.5

Таблиця 1.2 – Порівняльна характеристика ефективності розпізнавання зображення з нахиленим зображенням за допомогою алгоритмів SIFT, SURF та ORB

	Час (сек)	Ключових точок, оригінальне зображення. (тис. шт.)	Ключових точок, змінене зображення. (тис. шт).	Кількість збігів	Співпадіння. (%)
SIFT	0.133	248	229	150	62.89
SURF	0.049	162	214	111	59.04
ORB	0.026	261	298	145	51.88

Відповідно до наведених вище таблиць таблиці можна зробити висновок що SIFT більш надійний, однак згідно з описаними вище характеристиками SURF має вищу швидкодію та споживає менше ресурсів.

Варто зауважити, що такі методи як SIFT, SURF, ORB, та AKAZE не ґрунтуються на традиційних методах штучного інтелекту або машинного навчання. Замість цього, вони є класичними алгоритмами комп’ютерного зору, які використовують математичні та сигнальні методи для виявлення та опису ключових точок на зображеннях.

Ці алгоритми використовуються у таких задачах, як градієнти зображень, аналіз простору масштабів, локальні особливості зображення та різні дескриптори для виявлення характерних точок на зображеннях та опису їх локальних оточень. Вони призначені бути стійкими до змін масштабу, обертання, освітлення та точки огляду.

Проте, деякі з цих алгоритмів, зокрема SURF і AKAZE, використовують певні техніки машинного навчання у своїх конструкціях. Наприклад, SURF використовує концепції машинного навчання, такі як інтегральні зображення та хаарові хвилі, для прискорення обчислення особливостей зображення. Алгоритм AKAZE, подібно до цього, використовує нелінійну дифузію та локальні бінарні дескриптори, які можна розглядати як достатньо ефективні техніки, що використовують концепції машинного навчання.

Хоча ці алгоритми можуть використовувати елементи, деякі концепції МН, вони зазвичай не класифікуються як алгоритми, заснованими на штучному інтелекті в сучасному розумінні глибокого навчання або нейронних мереж. Вони належать до області класичних методів комп'ютерного зору і широко використовуються в завданнях комп'ютерного зору до появи глибокого навчання.

На відміну від цього, сучасні підходи до комп'ютерного зору, засновані на глибокому навчанні, такі як згорткові нейронні мережі, широко використовують принципи штучного інтелекту та машинного навчання. Вони навчаються ієрархічним ознакам безпосередньо з даних та досягають передових результатів у різних завданнях комп'ютерного зору, таких як класифікація зображень, виявлення об'єктів, сегментація та інші. У таблиці 1.3 зібрано порівняння принципів та особливостей роботи класичних алгоритмів з розпізнавання зображень таких як SIFT, SURF, та ORB до більш сучасних методів розпізнавання зображень за допомогою машинного навчання. Відповідно до даних наведених у таблиці можна зробити висновок що методи МН є більш ефективними. Основна складність полягає у тренуванні та налаштуванні нейронної мережі. У той час коли алгоритми вимагають високої кваліфікації для виконання покращення якості роботи або її швидкодії.

Таблиця 1.3 – Порівняння принципів роботи класичних алгоритмів з розпізнавання зображень до машинного навчання.

	SIFT, SURF, ORB	Машинне навчання
Призначення	Алгоритм для виявлення та опису ключових точок у зображеннях	Відповідно до призначення машинне навчання може виконувати одночасно виявлення та опис ключових точок та класифікацію зображень
Швидкодія	Класичні методами комп'ютерного зору, можуть бути відносно повільними на обробці великої кількості даних або в режимі реального часу	В залежності від моделі нейронної мережі швидкодія потенційно може бути значно вищою
Точність	Алгоритми мають високу точністю та стійкістю до змін масштабу та орієнтації	В залежності від рівня тренування та налаштувань нейронної мережі точність може бути як вищою так і нижчою
Складність реалізації	Дані алгоритми вимагають високого рівня експертизи для їх реалізації та оптимізації	Основна складність полягає у навчанні та налаштування нейронної мережі. Використання бібліотек таких як TensorFlow або PyTorch значно спрощує реалізацію
Робота з великими даними	Необхідно проводити оптимізацію алгоритмів при роботі з великими даними	Машинне навчання це галузі які розвиваються пліч опліч з «великими даними». Тому при використанні бібліотек таких як PyTorch або TensorFlow дозволить швидко інтегрувати можливість роботи з великими об'ємами даних.

1.2 Тензори як одиниця даних у розпізнаванні

Математична структура, що являє собою узагальнення матриць різної розмірності називається тензорами. Тензор може мати будь-яку розмірність. Поняття тензори використовується не лише у програмуванні а й у математиці та фізиці тензори використовуються для опису різноманітних об’єктів, що мають деяку кількість вимірів, наприклад тензори використовуються для опису поля у релятивістській фізиці [6].

Також тензори широко застосовуються у машинному навчанні. Тензори різної вимірності використовуються для представлення різних типів даних. Подання даних у тензорах може бути наступної вимірності:

- Скаляр –тензор 0-го порядку, являє собою число.
- Вектор – одновимірний тензор, подання у вигляді одновимірного масиву.
- Матриця –двовимірний тензор, відповідно двовимірний масив.

Також існують тензори більшої вимірності, масив з трьома або більше вимірами, приведено на рисунку 1.3.

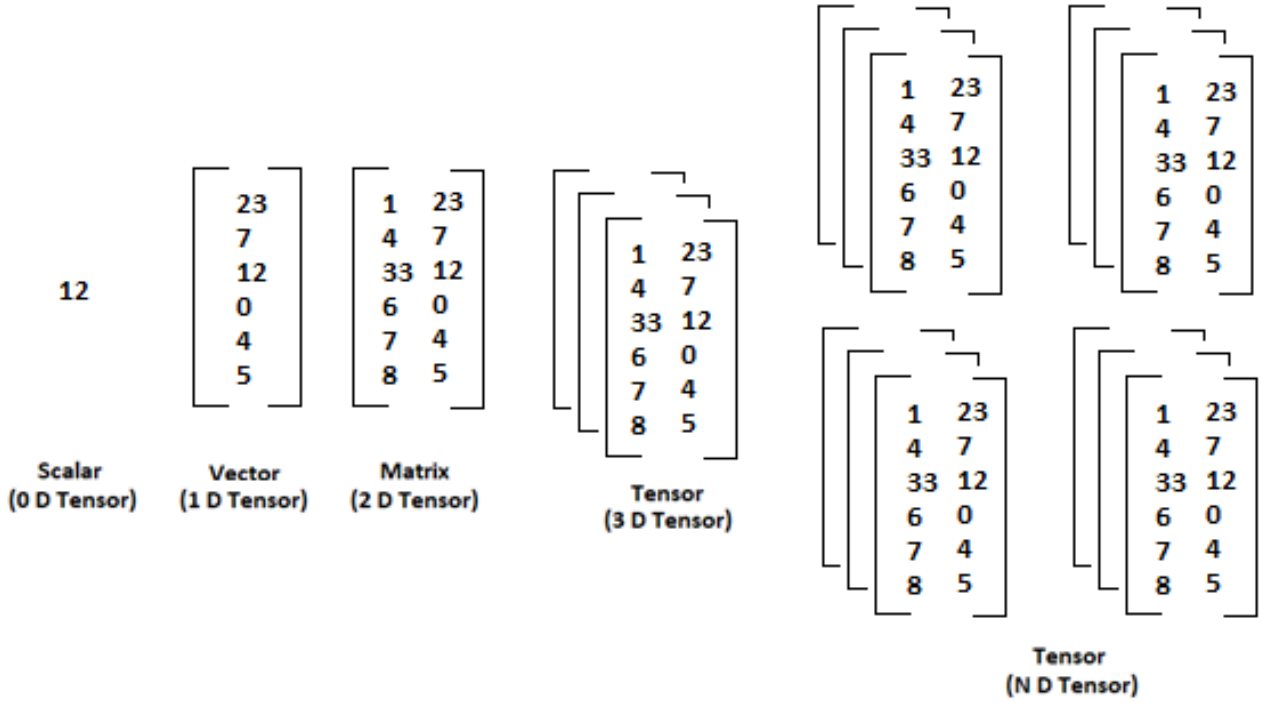

Рисунок 1.3 Представлення тензорів різного порядку

Відповідно до структури тензорів математичні операції, що використовуються для обчислення матриць також застосовуються і для роботі із тензорами. У машинному навчання їх використовують як структуру даних будь-якого типу, наприклад: звук, зображення, відео, текстові дані, тощо.

Навчальні набори даних що використовуються для тренування нейронних мереж -також подаються у вигляді багатовимірних матриць. Зсуви мережі, ваги також подані у вигляді тензорів. Операція зворотного поширення та оновлення ваг виконується за допомогою математичних операцій що такою використовуються для обчислення матриць у математиці.

Наприклад так як і матриці тензори можна додавати, віднімати. Такі арифметичні дії застосовуються послідовно відповідно до кожного елемента. Так як і матриці тензори можна множити та ділити на скаляр. Застосовувати матричне множення, транспонувати. Транспонування використовується для зміни розмірності. Разом з тим можна виконувати і складніші операції такі як піднесення до ступеня, тригонометричні функції так багато іншого. Важливо зауважити, що тензорами представлені не лише навчальні дані а й параметрів нейронної мережі у процесі навчання та оптимізації.

1.3 Нейронні мережі як модель розпізнавання та класифікації

Нейронна мережа – це математична модель, яка віддалено нагадує нейрони в головному мозку. Вона використовується для розв’язання завдань машинного навчання та штучного інтелекту. Нейронні мережі складаються з численних взаємопов’язаних «штучних нейронів», які обробляють інформацію та вчаться відповідати на різні вхідні сигнали. Основні компоненти нейронної мережі включають:

1. Вхідні шари: Це вхідні дані, які передаються мережі для обробки. Кількість вузлів у цьому шарі відповідає кількості вхідних функцій.
2. Приховані шари: Ці шари містять нейрони, які обчислюють проміжні значення між вхідним та вихідним шарами. Мережі можуть мати декілька прихованих

шарів.

3. Вихідний шар: Цей шар містить вихідні значення, які генеруються мережею після обчислень. Кількість вузлів у цьому шарі зазвичай відповідає кількості класів або категорій, які мережа намагається передбачити.

4. Ваги і зсуви: Кожен зв’язок між нейронами має асоційований ваговий коефіцієнт, який визначає вплив одного нейрона на інший. Зсуви – це додаткові параметри, що дозволяють зсувати ваги.

5. Функції активації: Кожний нейрон в мережі використовує функцію активації для обчислення виходу на основі ваг та вхідних даних.

6. Зворотне поширення помилки: Ця техніка використовується для навчання мережі, сповіщаючи її про помилку в прогнозу та адаптацію вагових коефіцієнтів для покращення точності.

Нейронні мережі використовуються для широкого спектру завдань, включаючи класифікацію, регресію, визнавання об’єктів у зображеннях, обробку природної мови, автономну навігацію роботів і багато інших. Вони стали ключовим інструментом у галузі штучного інтелекту та машинного навчання [7].

Найпростішим прикладом нейронної мережі є мережа побудована на базі перцептронів. Перцептрон – це проста модель штучного нейрону, яка використовується для задач класифікації та розпізнавання образів. Вона є фундаментальною одиницею у багатьох моделях нейронних мереж та машинного навчання. Перцептрон був розроблений Френком Розенблаттом в 1957 році і вважається однією з перших моделей нейрону, зображено на рисунку 1.4.

Основні характеристики перцептрону включають наступне:

- Входи і ваги – Перцептрон приймає вхідні сигнали та кожному з них призначає вагу. Вага визначає важливість конкретного входу для роботи перцептрону.
- Суматор – Ваги, присвоєні входам, використовуються для обчислення лінійної комбінації вхідних сигналів. Ця комбінація обчислюється у вигляді суми вагованих входів.

- Функція активації. Результат суматора передається через функцію активації, яка визначає, чи активується нейрон (видає вихідний сигнал) чи ні. Зазвичай використовуються порогова функція (якщо сума більше заданого порогу, то нейрон активний) або сигмоїдна функція.
- Бінарний вихід. Вихід перцептрону зазвичай бінарний – 0 або 1, що вказує на класифікацію вхідного об’єкта до одного з двох класів.

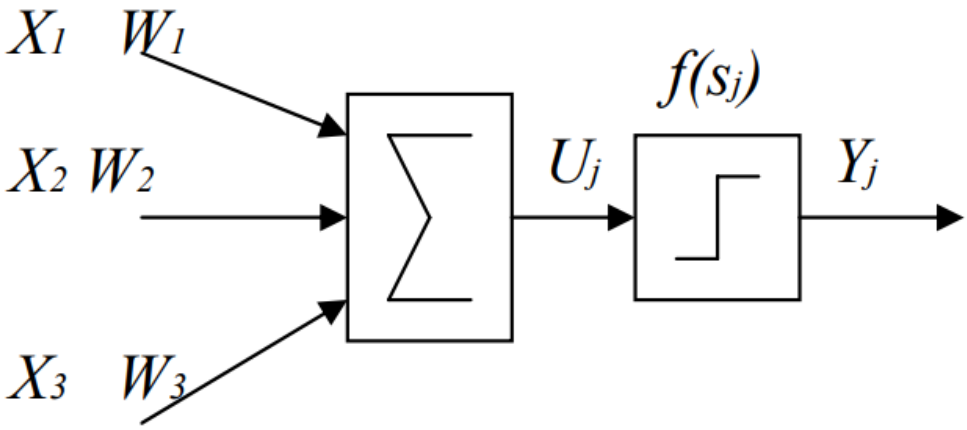

Рисунок 1.4 – Умовна схема перцептрона

Перцептрони можуть використовуватися для вирішення задач бінарної класифікації, таких як розпізнавання образів. Однак одиничний перцептрон має обмежену потужність та можливості. Для більш складних завдань використовують багатошарові нейронні мережі, які складаються з кількох шарів перцептронів і здатні навчатися більш складним залежностям у даних.

Перцептрони та нейронні мережі загалом використовуються в різних областях машинного навчання, включаючи розпізнавання образів, обробку природних мов, рекомендаційні системи та багато інших застосувань. Вони є важливим інструментом для вирішення завдань класифікації та розпізнавання [8].

Варто зауважити, що сучасні нейронні мережі мають дещо складнішу будову ніж просто мережа перцептронів.

1.4 Опис процесу класифікації зображень

Класифікація зображень за допомогою нейронних мереж – це процес призначення категорії або мітки до зображення на основі його вмісту. Цей процес вимагає використання нейронної мережі, яка навчилася розпізнавати об’єкти чи патерни на зображеннях.

Основні етапи класифікації зображень за допомогою нейронних мереж включають наступне:

- Підготовка даних: Спершу зображення повинні бути підготовлені для подачі на вхід нейронної мережі. Ця підготовка включає в себе зміну розміру зображення, нормалізацію значень пікселів, видалення шуму тощо.
- Вибір архітектури нейронної мережі: Вибір правильної архітектури нейронної мережі грає важливу роль у класифікації.
- Навчання: Навчання мережі включає в себе подачу набору даних з різними класами і мітками та оптимізацію вагових коефіцієнтів мережі з метою мінімізації помилки передбачень.
- Тестування та оцінка: Після навчання мережі, її ефективність оцінюється за допомогою незалежних тестових даних. Результати класифікації порівнюються зі справжніми мітками, і обчислюється точність моделі.
- Інференція: Після навчання та тестування модель може бути використана для класифікації нових зображень, які не брали участі у навчанні.

Класифікація зображень за допомогою нейронних мереж широко використовується у таких задачах, як визнавання об’єктів на фотографіях, медична діагностика, автономні автомобілі, фільтрація спаму та багато інших областях, де необхідно розпізнавати образи і приймати рішення на їх основі.

Для прикладу розглянемо процес класифікації зображень за допомогою бібліотеки PyTorch, це завдання, де ви використовуєте бібліотеку PyTorch для навчання нейронної мережі для розпізнавання та класифікації зображень. Ось загальна структура такого завдання:

Підготовка даних: Спершу вам потрібно підготувати набір даних для навчання та тестування. Зазвичай це включає в себе завантаження зображень, їхню попередню

обробку (зміну розміру, нормалізацію тощо) та розділення на тренувальний та тестовий набори. PyTorch надає зручний і гнучкий інтерфейс для роботи з нейронними мережами та глибоким навчанням. Ви можете використовувати засоби PyTorch для створення, навчання та використання нейронних мереж для класифікації зображень у вашому проекті.

- Вибір архітектури мережі: Виберіть тип нейронної мережі для класифікації. Зазвичай для класифікації зображень використовують згорткові нейронні мережі, такі як ResNet, VGG, або AlexNet.
- Створення моделі: Визначте архітектуру мережі у фреймворку PyTorch, створюючи власну модель або використовуючи попередньо навчену модель і адаптувати її до вашої задачі.
- Навчання моделі: Використовуйте тренувальний набір даних для навчання вашої моделі. Використовуйте функції втрат та методи оптимізації для покращення вагових коефіцієнтів мережі. Тут також важливо використовувати методи передавання градієнту, наприклад, зворотнє поширення помилки (backpropagation).
- Тестування та оцінка: Використовуйте тестовий набір даних для оцінки ефективності вашої моделі. Обчисліть метрики, такі як точність, відгук, точність та матрицю плутанини.
- Інференція: Після навчання та тестування ваша модель готова до використання для класифікації нових зображень. Підставляйте нові зображення у модель, і вона надасть передбачену категорію або мітку [9][10].

Більш класичним інструментом який здатний виконувати класифікації зображень є бібліотека OpenCV яка в першу чергу використовується для обробки та аналізу зображень, а не для навчання нейронних мереж. Тим не менш, ви можете використовувати OpenCV для виконання попереднього аналізу та обробки зображень перед їхньою подачею на класифікацію за допомогою інших бібліотек або моделей. Ось загальний підхід до класифікації зображень з використанням OpenCV:

- Завантаження та обробка зображень: Використовуйте OpenCV для завантаження зображень з джерела (наприклад, з файлу або відеопотоку) та виконайте

попередню обробку зображень, яка включає в себе зміну розміру, нормалізацію, видалення шуму та інші обробки, які можуть бути необхідними для підготовки зображень до подальшої обробки.

- Вибір моделі для класифікації: OpenCV не навчає нейронні мережі, але ви можете використовувати попередньо навчені моделі класифікації, такі як MobileNet, Inception, або ResNet. Зазвичай ці моделі доступні у фреймворках для машинного навчання, таких як TensorFlow чи PyTorch.
- Інференція: Після обробки зображень OpenCV використовує ці зображення для інференції з використанням обраної моделі. Модель дає передбачену категорію або мітку для кожного зображення.
- Оцінка та відображення результатів: Результати класифікації можна оцінити, порівнявши їх із справжніми мітками (якщо вони доступні) та обчислити метрики ефективності, такі як точність. Ви також можете відобразити результати на вихідному зображенні для подальшого аналізу.

Опираючись на вище наведені приклади двох бібліотек таких як OpenCV та PyTorch можна зробити висновок що етапи обробки подібні, проте, перша, не зважаючи на перевірену часом надійність, все ж не повністю використовує можливості машинного навчання, власне у сфері тренування та набору можливих типів нейронних мереж для використання [11].

На сьогоднішній день, для ефективного розпізнавання необхідно не лише мати зображення але і забезпечити захоплення об’єкта на зображенні, яких може бути декілька, різного розміру і у різних місцях, тому є необхідність більш детально розглянути CNN мережі. Тобто згорткові нейронні мережі, зображено на рисунку 1.5.

Мережі такого типу виявилися вкрай ефективними у обробці зображень порівняно з традиційними багатошаровими перцептронами (MLP). Їхній архітектурний підхід дозволяє автоматично визначати та використовувати важливі ознаки у вхідних даних, роблячи обробку зображень більш ефективною та точною.

Згорткова нейронна мережа зазвичай має три шари: згортковий шар, шар об’єднання та повністю з’єднаний шар, зображено на рисунку 1.6.

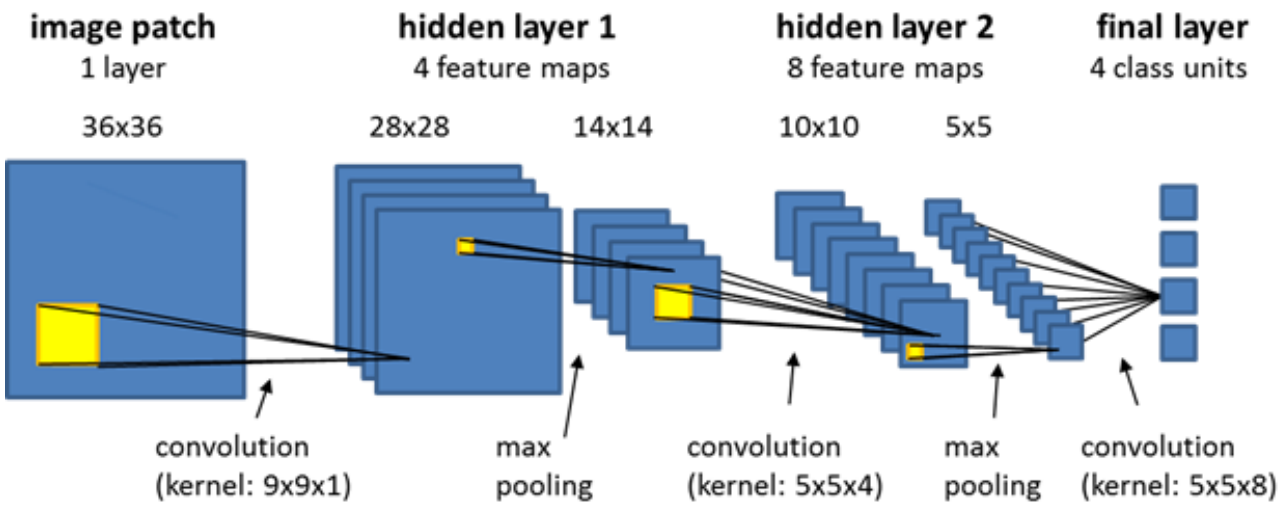

Рисунок 1.5 – Прицнип роботи згорткової нейронної мережі

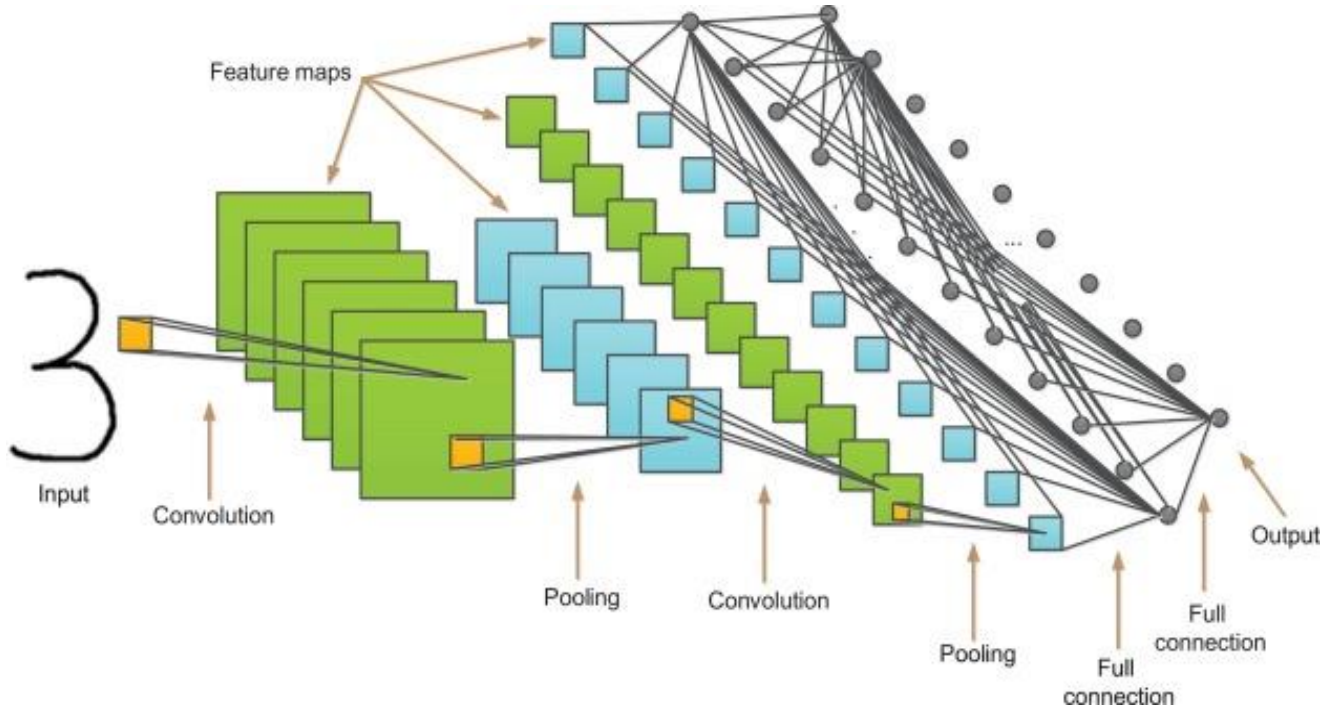

Рисунок 1.6 – Архітектура згорткових нейронних мереж

Основним і ключовим елементом у структурі згорткових нейронних мереж є шар згортки, який відіграє визначальну роль у обчислювальному процесі мережі. Основна ідея шару згортки полягає у використанні фільтрів (також відомих як ядра або фільтри згортки), які проводять операцію згортки зображення або вхідних даних. Фільтри згортки скользять по вхідному зображенню та виконують покомпонентне

множення пікселів у певній області (згортка) за вагами фільтра, а потім сумують отримані значення, щоб створити нове вихідне зображення, яке відображає особливості або певні візуальні атрибути.

Цей шар виконує точковий добуток між двома матрицями. Перша матриця представляє набір параметрів, які мережа вивчає, відомий як ядро. Друга матриця є обмеженою частиною рецептивного поля. Важливо відзначити, що ядро просторово менше за саме зображення, але має більшу глибину. Це означає, що, наприклад, для зображення з трьома каналами (RGB), висота і ширина ядра будуть просторово обмеженими, але його глибина розповсюджується на всі три канали зображення.

Узагальнюючи, шар згортки забезпечує важливий механізм виявлення та виділення ключових ознак у вхідних зображеннях, що робить його основною складовою частиною обробки зображень. Тобто, шари згортки допомагають автоматично виявляти та узагальнювати важливі ознаки у зображеннях, допомагаючи зменшити кількість параметрів та обчислювальний обсяг, а також покращуючи здатність моделі до розпізнавання об’єктів та ознак у вхідних даних.

Набагато кращі результати можна отримати, застосовуючи підхід машинного навчання, в якому великий набір правил або евристик для розрізнення цифр на основі форми штрихів, але на практиці такий підхід призводить до проліферації правил, винятків з правил і т.д., і незмінно дає поганий результат. Набагато кращі результати можна отримати, застосовуючи підхід машинного навчання, в якому великий набір N чисел{x1,...,xN} називається навчальною множиною використовується для налаштування параметрів адаптивної моделі. Категорії цифр у навчальній множині відомі заздалегідь, як правило, шляхом їх індивідуальної перевірки та ручного маркування. Ми можемо виразити категорію цифри за допомогою цільовий вектор t

Результат роботи алгоритму машинного навчання можна виразити у вигляді функції y(x), яка приймає нове зображення цифри x на вхід і генерує вихідний вектор y який кодується так само, як і цільові вектори. Точний вигляд функції y(x) визначається під час фази навчання фази, також відомої як фаза навчання на основі навчальних даних. Після того, як модель навчена, вона може визначати ідентичність нових зображень цифр, які, як кажуть, складають тестовий набір. Здатність правильно

класифікувати нові приклади, які відрізняються від тих, що використовувалися для навчання, називається узагальнення. У практичних застосуваннях мінливість вхідних векторів буде такою, що навчальні дані можуть включати лише малу частину всіх можливих вхідних векторів, і тому узагальнення є центральною метою розпізнавання образів. Для більшості практичних застосувань вихідні вхідні змінні, як правило, є попередньо обробляються щоб перетворити їх у деякий новий простір змінних, де, як сподіваються, проблему розпізнавання образів буде легше розв’язати. Наприклад, у задачі розпізнавання цифр зображення цифр зазвичай перекладають і масштабують так, щоб кожна цифра містилася в рамці фіксованого розміру.

Це значно зменшує варіабельність всередині кожного класу цифр, оскільки розташування і масштаб усіх цифр тепер однакові, що значно полегшує подальшому алгоритму розпізнавання образів розрізнення різних класів. Цей етап попередньої обробки іноді також називають вилученням ознак. Зауважте, що нові тестові дані мають бути попередньо оброблені за допомогою тих самих кроків, що й навчальні дані. Для прискорення обчислень також може виконуватися попередня обробка. Наприклад, якщо метою є розпізнавання обличь у реальному часі у відео-потоці високої роздільної здатності, комп’ютер повинен обробляти величезну кількість пікселів за секунду, і передача їх безпосередньо складному алгоритму розпізнавання образів може бути нездійсненною з обчислювальної точки зору. Натомість метою є пошук корисних ознак, які швидко обчислюються, але при цьому також зберігають корисну дискримінаційну інформацію, що дозволяє відрізняти обличчя від не-обличчя. Ці ознаки потім використовуються як вхідні дані для алгоритму розпізнавання образів. Наприклад, середнє значення інтенсивності зображення в прямокутній під області можна оцінити надзвичайно ефективно (Viola and Jones, 2004), і набір таких ознак може виявитися дуже ефективним для швидкого розпізнавання обличь. Оскільки кількість таких ознак менша за кількість пікселів, така попередня обробка являє собою певну форму зменшення розмірності

1.5 Висновки

Існує багато методів і алгоритмів пошуку зображень. Деякі з них базуються на математичних формулах, що не використовують машинне навчання. Інші методи – використовують нейронні мережі. Головною відмінністю алгоритмічних методів від методів, які застосовують машинне навчання полягає в тому, що модель нейронної мережі може покращуватися у результаті удосконалення та оптимізації тренувального алгоритму. Іншими словами, нейронна мережа може удосконалюватися безперервно. Тобто підміна більш досконалої нейронної мережі відбувається простим заміщенням, без зміни коду основної програми, що робить можливим ставати програмам «розумнішими» з кожним оновленням.

У свою чергу алгоритмічні методи розпізнавання хоч і піддаються оптимізації, однак обмежені математичним апаратом що застосовується. І хоча добре оптимізована бібліотека з розпізнавання та класифікації зображень може бути надзвичайно ефективною і малорозмірною, що дозволить її використання у мобільних пристроях, або інших відносно малопотужних обчислювальних системах, моделі машинного навчання також знаходяться на етапі оптимізації, що дозволить у майбутньому використовувати методи штучного інтелекту у багатьох пристроях.

Подання даних у машинному навчанні являє собою набір матриць; навчання та оптимізація – це ніщо інше як математичні операції над матрицями. Даний підхід дозволяє широким масам науковців використовувати штучний інтелект у дослідженнях, що значно сприяє науково технічному прогресу.

Процес класифікації зображень поділяється на два етапи. Перший це виділення меж зображення об’єкта на загальному зображенні. Далі приведення ділянки зображення з об’єктом до стандартної форми, тобто перетворення зображення на тензор визначеної розмірності. Після чого тензор подається на вхід нейронної мережі для аналізу.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Визначення методів оптимізації пошуку зображень

Аналіз існуючих методів прискорення пошуку графічних зображень включає в себе вивчення різних технік і алгоритмів, спрямованих на покращення швидкодії та точності пошуку великих обсягів графічної інформації. Ось деякі з підходів до прискорення пошуку:

1. Ключові функції пошуку:
    - Дескриптори: Вивчення різних дескрипторів для зображень є важливим етапом у покращенні ефективності пошуку. Дескриптори, такі як SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features) або CNN-засновані дескриптори, дозволяють ефективно вилучати інформацію з зображення, яка є унікальною та стійкою до змін масштабу, поворотів і освітлення. Використання цих дескрипторів дозволяє ефективно представляти та порівнювати зображення, що є ключовим для пошуку схожих об’єктів чи сцен у великих наборах графічної інформації.
    - Метрики схожості: Аналіз різних метрик схожості між зображеннями є необхідним для оцінки ступеня схожості між їхніми дескрипторами. Відстань між дескрипторами визначає, наскільки схожими є зображення. Використання цих метрик дозволяє визначити, наскільки добре зображення відповідає пошуковому запиту, що є критичним у задачах пошуку графічної інформації.
2. Алгоритми індексації: Алгоритми індексації відіграють ключову роль у вдосконаленні ефективності пошуку графічних зображень, дозволяючи швидко знаходити велику кількість зображень у великих наборах даних. Дослідження цих алгоритмів спрямоване на оптимізацію швидкості та точності пошуку. Ось деякі ключові аспекти:
    - Хешування Методи хешування використовують функції хешування для призначення кожному зображенню унікального коду (хешу), що дозволяє швидко виявляти дублікати або схожі зображення за їхніми хеш–кодами.

Використовується для прискорення пошуку схожих зображень за допомогою порівняння хеш–кодів.

- Індексація деревами:
  - k–d дерева: Координатні дерева, що розбивають простір даних на рекурсивні підпростори для швидшого пошуку.
  - R–дерева: Дерева, які групують об’єкти в області простору для ефективного пошуку.

Деревовидні структури даних дозволяють ефективно організовувати та шукати зображення в просторі дескрипторів. Більш загальне визначення дерева це граф.

Зображення представляються у вигляді графів, де вершини – це дескриптори, а ребра – схожість між ними. Графові методи дозволяють використовувати алгоритми глибинного пошуку для ефективного виявлення схожих зображень.

Дослідження та вдосконалення алгоритмів індексації спрямовані на створення більш ефективних та швидких систем пошуку графічних зображень в реальному часі.

3. Методи масштабованого пошуку:

- Розподілені системи. Розгляд можливостей використання розподілених систем для прискорення пошуку у великих наборах даних. Це включає в себе розподілені бази даних, які дозволяють паралельно обробляти та шукати зображення на різних вузлах системи. Розподілені системи забезпечують ефективне використання ресурсів та розподіл завдань для швидшого пошуку великих обсягів графічної інформації. Це особливо важливо для великих баз даних, де централізований підхід може стати обмеженням.
- GPU–прискорення. Вивчення використання графічних процесорів для прискорення обчислень у завданнях пошуку зображень. Графічні процесори (GPU) здатні оброблювати багато операцій паралельно, що робить їх ефективними для задач обробки зображень. Використання GPU дозволяє прискорити обчислення, пов’язані з обробкою зображень та виконанням алгоритмів пошуку. Це може бути важливим у випадках, коли велика кількість зображень потребує швидкого аналізу.

4. Аспекти обробки зображень:
    - Оптимізація обчислень. Розгляд методів оптимізації обчислень при роботі з графічними даними, таких як пакетна обробка зображень. Пакетна обробка зображень включає в себе групову обробку декількох зображень одночасно, що може покращити швидкодію завдань обробки. Застосування оптимізованих методів обчислень дозволяє швидше виконувати операції обробки, такі як фільтрація, зміна розміру, або витягання деяких характеристик зображення.
    - Паралельність. Дослідження можливостей використання паралельних обчислень для прискорення аналізу зображень. Паралельність в контексті обробки зображень може бути реалізована на рівні обчислень, фільтрів, чи операцій над декількома зображеннями одночасно. Використання паралельних обчислень дозволяє розділити завдання обробки між різними обчислювальними одиницями, таким чином прискорюючи аналіз зображень.
5. Застосування машинного навчання:
    - Нейронні мережі. Дослідження застосувань нейронних мереж для вдосконалення задач пошуку графічних зображень. Нейронні мережі, зокрема глибокі нейронні мережі (Deep Learning), можуть використовуватися для ефективного витягування признаків та автоматичного вивчення складних залежностей між зображеннями. Застосування нейронних мереж може покращити якість розпізнавання та класифікації зображень, а також забезпечити більш точні та адаптивні методи пошуку.
    - Застосування попередньої обробки.
    - Вивчення того, як перед обробка зображень за допомогою машинного навчання може поліпшити якість та швидкість пошуку. Вона може включати у себе використання алгоритмів машинного навчання для автоматичного визначення оптимальних параметрів обробки зображень. Застосування методів машинного навчання для передобробки може

допомогти у вилученні шуму, нормалізації зображень та підготовці їх до подальшого аналізу.

У сучасному інформаційному суспільстві, де завдання автоматизації та обробки великого обсягу графічної інформації стає все більш актуальним, пошук та класифікація графічних зображень відіграють важливу роль у забезпеченні ефективного доступу до збережених даних. Цей розділ присвячений вивченню особливостей пошуку графічних зображень та розробці ефективного механізму їхньої класифікації.

Дослідження спрямоване на розуміння та вирішення технічних аспектів, пов’язаних із завантаженням, зберіганням та обробкою графічної інформації на основі сучасних методів обробки зображень та машинного навчання. Проект передбачає розгортання класифікатора, який дозволить ефективно організовувати та здійснювати пошук зображень за їхнім змістом.

У цьому розділі розглянуто ключові аспекти, що формують фундамент для подальшого дослідження та визначено методологічний підхід до досягнення поставленої мети.

Інтерес до методів цифрової обробки зображень в даній роботі обумовлений двома важливими сферами застосування, які глибоко впливають на розвиток інформаційних технологій та взаємодію з графічною інформацією.

По–перше, використання цих методів направлене на покращення графічної інформації для інтерпретації людиною. В сучасному світі, де візуальна комунікація відіграє важливу роль, обробка зображень може значно полегшити розуміння та аналіз графічних даних. Наприклад, у медичній сфері це може допомогти вдосконалити діагностику за допомогою обробки та аналізу медичних зображень, а в галузі мистецтва та дизайну – створювати більш точні та естетичні візуальні елементи.

По-друге, обробка даних зображень грає ключову роль у сфері зберігання, передачі та представлення для автономного машинного сприйняття. Завдяки розвитку технологій автономних систем, таких як роботи та системи штучного

інтелекту, методи цифрової обробки зображень стають необхідним інструментом для розпізнавання об’єктів, навігації та прийняття рішень на основі візуальної інформації.

Ці дві сфери використання методів цифрової обробки зображень представляють собою великий потенціал для подальшого розвитку та інновацій в інформаційних технологіях та машинному навчанні.

Цифрова обробка зображень представляє собою широкий клас методів, які використовують комп’ютерні алгоритми для маніпулювання графічними даними. У в даному дослідженні, цей етап виявляється важливим у контексті розробки системи пошуку та класифікації графічних зображень. Цифрова обробка зображення є ключовим елементом роботи програми.

Розпізнавання образів бере свій початок в інженерії, тоді як машинне навчання виросло з комп’ютерних наук. Однак ці напрямки можна розглядати як дві складові частини однієї галузі, і вони разом зазнали значного розвитку за останні десять років. Зокрема, байєсівські методи перетворилися з вузькоспеціалізованої ніші на мейнстрім, а графічні моделі стали загальною основою для опису та застосування ймовірнісних моделей. Крім того, практичне застосування байєсівських методів значно розширилося завдяки розробці низки алгоритмів наближеного виведення, таких як варіаційний Байєс і прогнозування математичного сподівання. Аналогічно, нові моделі, засновані на ядрах, мали значний вплив як на алгоритми, так і на додатки

Сфера розпізнавання образів пов’язана з автоматичним виявленням закономірностей у даних за допомогою комп’ютерних алгоритмів і використанням цих закономірностей для виконання таких дій, як класифікація даних за різними категоріями.

Інноваційні методи обробки зображень включають в себе застосування передових технологій, таких як нейронні мережі. Глибоке навчання викликало справжню революцію у сфері комп’ютерного зору, розкриваючи можливості машин «бачити» та інтерпретувати оточуючий світ [12].

2.2 Вибір методу класифікації

Важливим етапом розробки системи є вибір типу реалізації нейронної мережі. Розглянемо деякі типи моделей які були створені раніше і відкриті до використання:

- Faster R-CNN: Ця модель об’єктного виявлення у два етапи пропонує регіони, а потім передбачає обмежувальні рамки та ймовірності класів у цих регіонах.
- Mask R-CNN: Розширення Faster R-CNN, яке додає гілку передбачення маски для сегментації форм об’єктів поруч з передбаченням обмежувальної рамки.
- RetinaNet – це модель одного етапу об’єктного виявлення, відома своєю фокусною втратою та мережею Feature Pyramid Network (FPN), які вирішують дисбаланс класів та ефективно виявляють об’єкти на різних масштабах [13].
- YOLO (You Only Look Once): Моделі YOLO, такі як YOLOv3 та YOLOv4, – це одноетапні детектори, які безпосередньо передбачають обмежувальні рамки та ймовірності класів по всьому зображенню за один прохід, відомі своєю швидкістю.
- SSD (Single Shot MultiBox Detector): Інша одноетапна модель об’єктного виявлення, яка передбачає обмежувальні рамки та ймовірності класів на кількох масштабах карт ознак за допомогою згорткових фільтрів.
- EfficientDet: Масштабована та ефективна родина моделей об’єктного виявлення, похідна від EfficientNet, що пропонує моделі від EfficientDet-D0 до EfficientDet-D7 з різною складністю.
- CenterNet: Ця модель безпосередньо передбачає центри та розміри об’єктів за тепловою картою, спрощуючи задачу об’єктного виявлення до задачі оцінки ключових точок.
- DETR (DEtection TRansformer): Використання трансформерів для об’єктного виявлення, що переформулює завдання виявлення як проблему передбачення набору без потреби у попередньо визначених ящиках–якорях.
- Cascade R-CNN: Ця модель покращує продуктивність, використовуючи каскадну архітектуру для покращення результатів об’єктного виявлення на кількох етапах.
- Sparse R-CNN: Введення розрідженості в обчислення Faster R–CNN, що прискорює інференс без значного втрати точності.

Як видно із опису практично усі типи наведених моделей є моделями зі згорткою, тобто. Варто зауважити, що модель DETR (DEtection TRansformer) використовує метод трансформації не опираючись лише на згортку. Однак враховуючи той факт, що більшість моделей повністю або майже повністю використовують згортку, було прийнято рішення використовувати молель із згорткою.

2.3 Розробка архітектури додатку.

Тришарова архітектура є структурним стилем для розробки програмного забезпечення, який розділяє систему на три основні компоненти або шари: інтерфейсний (представлення), бізнес–логіка та шар даних. Інтерфейсний шар відповідає за взаємодію з користувачем і представлення даних, бізнес–логіка реалізує бізнес–правила та обробку даних, а шар даних відповідає за доступ до сховища даних. Ця архітектура спрощує розробку та підтримку програмного забезпечення, а також сприяє виокремленню функцій та полегшує тестування та масштабування системи.

Щодо мікросервісної архітектури це підхід до розробки програмного забезпечення, де програма розбивається на невеликі незалежні компоненти, відомі як мікросервіси, кожен з яких відповідає за конкретну функцію чи послугу. Ці мікросервіси взаємодіють між собою через API, а їх розгортання і масштабування можуть проводитися окремо. Такий підхід дозволяє забезпечити більшу гнучкість та швидкість розробки, полегшує підтримку та масштабування системи, а також дозволяє використовувати різні технології для кожного мікросервіса [14].

Вибір між тришаровою архітектурою і мікросервісною архітектурою залежить від конкретних потреб та вимог вашого проекту:

- Використовуйте тришарову архітектуру, коли у вас є монолітичний проект або менший масштаб системи.
- Вона може бути практичною, коли бізнес–логіка досить проста і може бути впроваджена в одному шарі.
- Тришарова архітектура добре підходить для проектів з обмеженими ресурсами і бюджетом.

Мікросервісна архітектура:

- Розгляньте мікросервісну архітектуру, коли ваша система росте, і вам потрібно забезпечити більшу масштабованість та надійність.
- Цей підхід корисний, коли бізнес–логіка розділена на багато різних функцій чи модулів, які можуть бути розгорнуті та масштабовані окремо.
- Мікросервіси дозволяють використовувати різні технології для кожного сервісу та прискорюють розробку, тестування і впровадження нових функцій.

Комбінована архітектура:

- У багатьох випадках, реальний проект може використовувати комбінацію тришарової та мікросервісної архітектури.
- Наприклад, великий моноліт може бути розібраний на мікросервіси, якщо це дозволить покращити масштабованість та робити зміни у функціональності.
- Кожен випадок повинен оцінюватися окремо, враховуючи потреби та обмеження проекту.

Принципова схема додатку з мікросервісою архітектурою представлена на рисунку 2.1.

Рисунок 2.1 -Принципова схема додатку з мікросервісною архітектурою

Отже, вибір архітектурного стилю повинен відповідати конкретним цілям та вимогам вашого проекту, а також масштабу та обмеженням, з якими ви працюєте. Дане програмне забезпечення представляє собою, інформаційну систему, що аналізує зображення, та відповідно до результату роботи, відносить зображення до певної категорії та додає її до бази даних. Структуру проекту зображено на рисунку 2.1.

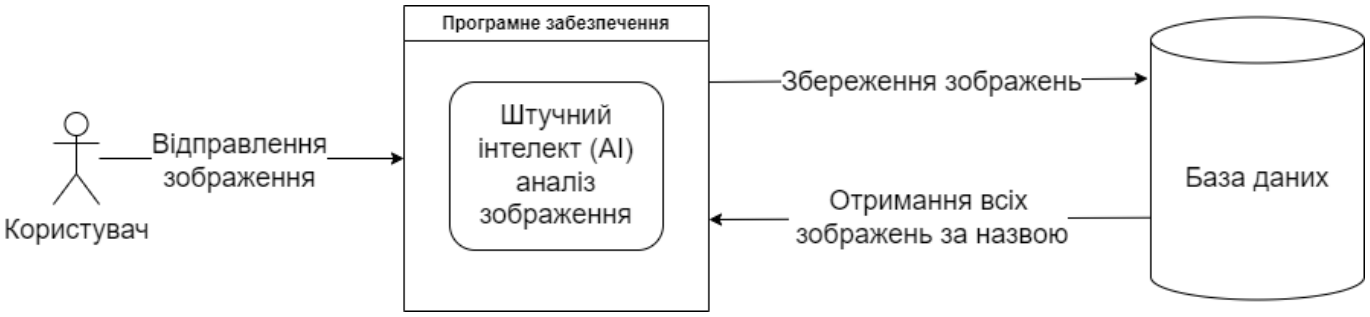

Рисунок 2.2 – Структура програмного продукту

Згідно даної структури, що зображена на рисунку 2.2, програма має 2 компоненти: це сама програма, яка має інструменти для аналізу зображення, та база даних, що зберігає зображення, після аналізу. Варто зауважити, що програму логічно можна розділити ще на 2 шари – це користувацький інтерфейс та бізнес-логіку. Розділення на шари сприяє модульності коду. Кожен шар відповідає за певну функціональність, що полегшує розвиток, тестування та зміни в програмі. Це дозволяє команді розробників працювати над різними частинами програми незалежно одна від одної. Шарування дозволяє легко вносити зміни або додавати новий функціонал, не впливаючи на інші частини програми. Це робить програмне забезпечення більш гнучким та легким для розширення. Чітке розділення на шари полегшує підтримку та обслуговування системи. Виправлення помилок або оптимізація можуть відбуватися на рівні конкретного шару, що спрощує вирішення проблем.

2.4 Висновки

В умовах, де візуальні дані стають неодмінною частиною нашого повсякденного життя, розробка ефективних та швидких методів обробки графічних зображень стає вельми важливою. Аналіз існуючих методів прискорення пошуку графічних зображень включає вивчення різних технік та алгоритмів, спрямованих на поліпшення швидкодії та точності пошуку у великих обсягах графічної інформації.

Описані вище підходи до прискорення пошуку надають вичерпний огляд ключових аспектів у цій області. Важливою є роль дескрипторів, таких як SIFT, SURF, і CNN-заснованих дескрипторів, для ефективного витягування та порівняння інформації на зображеннях. Метрики схожості грають критичну роль у визначенні ступеня подібності між зображеннями, що є ключовим у завданнях пошуку.

Алгоритми індексації, такі як хешування та різноманітні структури даних, включаючи k-d дерева, R-дерева, та графові структури, сприяють оптимізації швидкості та точності пошуку. Методи масштабованого пошуку, такі як розподілені системи та GPU або TPU прискорення, стають невід’ємними у вирішенні великомасштабних завдань аналізу графічних даних.

У контексті обробки зображень, оптимізація обчислень та біль потужні обчислювальні системи можуть значно покращити продуктивність. Підвищити швидкість обчислень можна «вертикально» або «горизонтально». Вертикальне пришвидшення це використання наприклад більш потужного процесора. Однак, на сьогодні процесори на базі напівпровідників досягли своєї максимальної потужності, тому при використанні найпотужнішого процесора обов’язково існує певний бар’єр продуктивності вище якого піднятися не вдасться. На противагу, горизонтальне пришвидшення обчислень полягає у паралелізації процесу. Особливо добре паралелізація підходить для алгоритмів обробки зображень, та процесу навчання моделей нейронних мереж. Також відсутні межі пришвидшення, тобто у систему можна додавати необмежену кількість процесорів і від того система буде працювати тільки швидше. Проведений аналіз дозволяє визначити сильні та слабкі сторони існуючих методів, а також виявити оптимальні рішення для різноманітних вимог та завдань у сфері пошуку графічної інформації.

3 ПРОГРАМНА РОЗРОБКА СИСТЕМИ

У світі, де візуальні дані стають невід'ємною частиною щоденного життя, пошук та аналіз графічних зображень викликають за собою потреби у розробці ефективних та швидких методів обробки великих обсягів даних. Зображення та фотографії стають мовчазними свідками нашого світу, але виявлення конкретних об'єктів чи подій у цьому безмежному океані вимагає нових технологій та підходів.

3.1 Визначення інструментів для побудови системи

На сьогодні існує безліч реалізацій різноманітних типів мереж призначених для розпізнавання практично будь-яких даних та інтеграцією з будь-якою мовою програмування. Найпоширенішими мовами програмування у сфері машинного навчання та штучного інтелекту є:

- Python є найбільш популярною мовою програмування машинному навчанні завдяки своїй простоті, зрозумілості, широкій підтримці спільноти та великому екосистемі бібліотек і фреймворків. Популярні бібліотеки, такі як TensorFlow, PyTorch, Scikit-learn, Keras та NLTK, широко використовуються для розробки у машинному навчанні [15].
- R є ще однією мовою, яку віддають перевагу статистики та вчені в області аналізу даних. Вона пропонує відмінні інструменти статистичного аналізу, можливості маніпулювання даними та бібліотеки візуалізації (наприклад, ggplot2). R часто використовується для статистичного моделювання та аналізу даних у машинному навчанні.
- Java залишається популярною у машинному навчанні завдяки своїй продуктивності, переносимості та зрілості бібліотек, таких як Deeplearning4j, Weka та MOA. Вона часто використовується в підприємницьких застосунках, але Python більш домінуюча в дослідженнях та прототипуванні.
- Відомий своєю швидкістю та ефективністю C++ використовується у вимогливих до продуктивності задачах машинного навчання, особливо для створення високопродуктивних бібліотек і фреймворків. Бібліотеки, такі як OpenCV,

використовують C++ для завдань комп'ютерного зору, а TensorFlow має API на C++ для використання у виробництві.

- JavaScript набирає популярність у машинному навчанні завдяки розвитку бібліотек, таких як TensorFlow.js та Brain.js, що дозволяють розробникам виконувати завдання машинного навчання безпосередньо в браузері або в середовищі Node.js.
- Julia є високорівневою, високопродуктивною мовою, спеціально призначеною для технічного обчислення. Вона набирає популярність через швидкість, зручність використання та функції, спрямовані на числове та наукове обчислення у штучного інтелекту.
- Scala використовується в штучного інтелекту завдяки функціональним можливостям та сумісності з Java. Бібліотеки, такі як Breeze та Spark MLlib, написані на Scala та надають інструменти для завдань машинного навчання.
- MATLAB пропонує комплексне середовище для числового обчислення, включаючи можливості штучного інтелекту з такими пакетами, як Neural Network Toolbox та Statistics and Machine Learning Toolbox. Це популярно у наукових та дослідницьких середовищах.

Вибір мови програмування може бути зумовленим не лише власними уподобаннями, але й наявними бібліотеками, фреймворками, популярність серед розробників, так зване community, і власне задачею яку необхідно вирішити. Серед готових бібліотек та фреймворків найбільш популярними є наступні продукти:

- TensorFlow – розроблений компанією Google, TensorFlow є одним з найпопулярніших та потужних фреймворків для роботи з нейронними мережами та глибоким навчанням. Він має широкий функціонал для розробки моделей, включаючи побудову нейронних мереж, обробку даних, навчання та інші завдання. Остання версія бібліотеки 2.14.0 [16].
- PyTorch – розроблений компанією Facebook, PyTorch набув значної популярності завдяки простоті використання та гнучкості. Він часто використовується в наукових дослідженнях та експериментах, має зручний інтерфейс та підтримує динамічні обчислення. Остання стабільна версія бібліотеки 2.1.0 [17].

- Scikit–learn – це популярний фреймворк з відкритим кодом для машинного навчання в Python, який надає широкий спектр алгоритмів для класифікації, регресії, кластеризації, а також інструменти для підготовки даних та оцінки моделей. Остання версія бібліотеки на час написання роботи 1.3.2.
- Keras – це високорівневий API для побудови та тренування нейронних мереж, який може використовувати TensorFlow, а також інші фреймворки. Він спрощує створення моделей та навчання, і дозволяє швидко прототипувати різні архітектури. Стабільна версія бібліотеки на момент написання 2.14.0.
- MXNet – це інша популярна бібліотека для глибокого навчання, яка має гнучкість та швидкість. MXNet підтримує різні мови програмування, такі як Python, Scala, R та інші. Стабільна версія бібліотеки на момент написання 1.9.1.
- Caffe: Це фреймворк для глибокого навчання, який використовується для швидкого створення та експериментування з нейронними мережами, особливо у сфері обробки зображень. Стабільна версія бібліотеки на момент написання 1.0.
- Theano: Хоча розвиток Theano припинено, він варто згадати, оскільки він був одним із перших фреймворків глибокого навчання, який надавав засоби для обчислення нейронних мереж у Python. Стабільна версія бібліотеки на момент написання 2.17.4.

Чи не найважливішими аспектом побудови системи є середовище тренування моделі. Як відомо тренування моделі вимагає надвеликих обчислень які практично неможливо виконати на звичайному персональному компютері. Якщо мова йде про тренування готової до використання у реальному житті моделі необхідно використовувати GPU або TPU. У крайньому разі якщо використовувати CPU у такому випадку процес навчання може тривати досить довго. Отже найбільш поширеними платформами що надають можливість використання GPU та TPU є:

- NVIDIA CUDA: CUDA – це паралельна платформа обчислень та модель програмування, розроблена компанією NVIDIA. Вона дозволяє розробникам використовувати обчислювальну потужність відеокарт NVIDIA для загальних обчислень, включаючи завдання машинного навчання. Багато фреймворків та

бібліотек машинного навчання, таких як TensorFlow, PyTorch та інші, підтримують CUDA для прискорення тренування на відеокартах NVIDIA.

- Відеокарти NVIDIA: NVIDIA пропонує ряд відеокарт, які підходять для завдань машинного навчання. Їх серії Tesla, Quadro та GeForce RTX часто використовуються у застосунках із машинним навчанням. Високопродуктивні відеокарти NVIDIA Tesla, зокрема ті з архітектурою NVIDIA Ampere (наприклад, A100 GPU), призначені спеціально для штучного інтелекту, пропонують збільшену продуктивність та обсяг пам’яті.
- AMD ROCm: Radeon Open Compute (ROCm) – це відкрита платформа для обчислень на відеокартах, розроблена компанією AMD. Хоча вона історично більше спрямована на наукові обчислення, вона розширюється для підтримки фреймворків глибокого навчання. Деякі фреймворки, такі як TensorFlow, працюють над підтримкою ROCm для навчання на відеокартах AMD.
- Google Cloud Platform GCP пропонує різні екземпляри відеокарт, такі як NVIDIA Tesla T4, P4, V100 та A100 через свій Compute Engine. Ці екземпляри підходять для прискорення роботи з машинним навчанням та тренування глибоких моделей з використанням фреймворків, таких як TensorFlow, PyTorch та інші.
- Amazon Web Services надає різноманітні екземпляри відеокарт у своєму сервісі EC2 (Elastic Compute Cloud), включаючи NVIDIA Tesla, такі як T4, P4, P3 (V100) та інші. Ці екземпляри часто використовуються для тренування моделей машинного навчання з використанням популярних фреймворків, таких як TensorFlow, PyTorch та MXNet.
- Azure пропонує екземпляри відеокарт у своїх віртуальних машинах, включаючи NVIDIA Tesla, такі як V100, P40, P4 та інші. Ці екземпляри можуть використовуватися для прискорення завдань машинного навчання та тренування моделей за допомогою різних фреймворків.

Обираючи платформу з використанням відеокарт для тренування моделей машинного навчання, важливо враховувати фактори, такі як вартість, доступність відеокарт, підтримка фреймворків, зручність використання та специфічні вимоги проекту з машинного навчання. Також важливо переконатися у сумісності обраної

платформи з використовуваними фреймворками чи бібліотеками машинного навчання.

Python вважається найбільш підходящою мовою для розробки в машинному навчанні завдяки своїй простоті, гнучкості та доступності великої кількості бібліотек та фреймворків. Також варто зазначити що мова програмування Python підтримується такими хмарними провайдерами як AWS та GCP. Якщо ж оберати між AWS та GCP більше привабливим виглядає саме GCP тому що має у своєму арсеналі такий інструмент як Google Colab що дозволяє писати сценарії тренувань мовою Python, одразу надає можливість використання GPU та TPU. Форма подання тренування у вигляді сценарію дозволяє писати код і коментарі до нього у вигляді виконуваної документації. Враховуючи усе вищевказане було прийнято рішення використовувати мову програмування Python у купі з бібліотекою PyTorch і для тренування використовувати Google Colab. Власне додаток буде написано також мовою Python з використанням PyTorch як інструмента для застосування натренованої моделі. Також для надання користувацького Web інтерфейсу буде використано бібліотеку Flask [18]. Для простоти збереження даних буде використано файлову SQL базу даних SQLite, що дозволить зберігати дані швидко. Для промислової реалізації можна просто перейти на будь-яку інше SQL базу даних.

Для розробки даного програмного забезпечення, використовується мова програмування Python. Дана мова програмування має широкий вибір бібліотек, які надають можливість використовувати штучний інтелект, створювати клієнт–серверні додатки, та використовувати бази даних, для зберігання різноманітної інформації.

База даних обрана SQLite – це база даних з відкритим вихідним кодом, написана на мові C і доступна для запитів за допомогою звичайного SQL. Має широке розповсюдження серед розробників, завдяки наступним перевагам:

- Порівняно з іншими базами даних, дуже легка та ненагромаджена, тому вивчання її проходить дуже легко, та підтримує більшу частину стандарту SQL.
- Має низькі системні вимоги, та споживає не велику частину ресурсів.

- Широко використовується, в різних типах програм, через те що вона забезпечує просту роботу, на відміну від інших баз даних, що потребують окремого сервера [19].

Для інтерфейсу, щоб взаємодіяти з системою використовується стандартний html, що дозволить створити відносно простий веб-додаток з мінімальним інтерфейсом.

Враховуючи достатньо широкий спектр технологій обраних для розробки застосунку також необхідно визначити інструменти, які дозволять полегшити розробку. Такі інструменти мають назву – інтегровані середовища розробки. На сьогодні існує ряд широко вживаних програмних пакетів, що включають у себе підпрограми, що надають такі можливості як: підсвітка тексту програми відповідно до мови програмування, здатні відтворювати програму із застосуванням точок зупину, що значно полегшує налагоджування програми у режимі роботи близькому до реального виконання, та багато іншого. Для того щоб обрати саме ту систему розробки яка оптимально підходить для написання програмного застосунку з пошуку зображень за допомогою нейронних мереж необхідно провести аналіз доступних варіантів:

- Visual Studio: зазвичай використовується для розробки програмного забезпечення під операційну систему Windows, підтримує різні мови програмування, такі як C#, C++, JavaScript та інші. Також ібснує Visual Studio Code – редактор коду, який підтримує багато мов програмування.
- Eclipse: це програмне забезпечення з відкритим вихідним кодом, що також підтримує багато мов програмування, але в основному використовується для Java. Має велику кількість плагінів що дозволяє значно розширити можливості з інтеграції з іншими системами та розробки майже усіх широко використовуваних мов програмування.

- IntelliJ IDEA: Використовується для розробки програмного забезпечення такими мовами як Java, Kotlin та інші. Має потужні можливості для редагування коду включаючи використання штучного інтелекту, що надає підказки розробнику щодо якості коду.

- PyCharm: Від компанії IntelliJ розроблений спеціально для роботи з мовою програмування Python. Має вбудовані інструменти для аналізу коду, відладки та керування середовищем виконання.
- Xcode: Середовище розробки для macOS, використовується для розробки програмного забезпечення для iOS, macOS, watchOS та tvOS, зокрема мобільних додатків та програм для Apple-пристроїв.
- Android Studio: Офіційне середовище розробки для платформи Android. Забезпечує повний спектр інструментів для розробки мобільних додатків для Android.
- NetBeans: Інша відкрита платформа, яка підтримує різні мови програмування, такі як Java, PHP, C++ та інші.

Проаналізувавши вище наведені програмні продукти можна зробити висновок, що майже всі вони здатні підтримувати будь-яку мову програмування та інтегруватися із будь-якою базою даних. Тому досить часто кожен фахівець обирає інструмент який подобається саме йому. Також вибір інтегрованих середовищ розробки часто зумовлений особистим досвідом. Однак, серед інших відомо, що середовище Eclipse це безкоштовний програмний продукт який має найбільшу кількість доступних плагінів (більше тисячі доступних плагінів), що надають практично необмежений функціонал який дозволяє проводити повний спектр розробки. На противагу IntelliJ IDEA має приблизно 700 плігінів. Тому, було прийнято рішення використовувати Eclipse як інструмент з розробки застосунку.

3.2 Створення, тренування та покращення моделі

Створення нейронної мережі включає кілька етапів, починаючи від визначення її архітектури, підготовка навчальних та тестових- даних до навчання та оцінки результатів. Існує правило: якщо задачу можна вирішити використовуючи алгоритмічне програмування – використання ШІ не виправдане. Однак, якщо задача дійсно не вирішується ніяким алгоритмом, існує ймовірність що вирішення можливе з використанням нейронної мережі.

Найпершим етапом буде вибір типу моделі. Відомо, що для обробки зображень зазвичай використовують моделі зі згорткою (convolutional neural network). До складу PyTorch входить багато типів мереж доступних для побудови, ось деякі з них:

- Класифікація зображень (Image Classification Models): AlexNet, VGG (VGG-11, VGG-13, VGG-16, VGG-19), ResNet (ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152), DenseNet (DenseNet-121, DenseNet-169, DenseNet-201, DenseNet-161), Inception (Inception v3), SqueezeNet (SqueezeNet 1.0, SqueezeNet 1.1), MobileNetV2 EfficientNet (EfficientNet-b0 to EfficientNet-b7), ResNeXt, Wide Residual Networks (WideResNet),
- Виявлення об’єктів (Object Detection Models): Faster R-CNN, Mask R-CNN, RetinaNet [20]
- Моделі симантичної сегментації: Semantic Segmentation Models: FCN (Fully Convolutional Networks), DeepLabV3, U-Net, PSPNet (Pyramid Scene Parsing Network)
- Генеруючі моделі (Generative Models): Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs).

Відповідно до поставленої задачі на перший погляд необхідно створити одну з моделей категорії: «класифікація зображень». Однак, імовірно, що тип «виявлення об’єктів» теж може підійти. Тож розглянемо різницю між класифікацією зображень та визначенням об’єктів. Отже:

Класифікація зображень – це завдання яке включає в себе категоризацію всього зображення на заздалегідь визначені класи або категорії. Мета такого аналізу полягає у призначенні однієї мітки або класу для всього зображення на основі його вмісту. Результатом роботи моделі буде одна мітка, що описує присутність усього зображення. Наприклад, визначення того, чи містить зображення кота, собаку, автомобіль або дерево.

Виявлення об’єктів, з іншого боку, включає локалізацію та класифікацію кількох об’єктів у межах зображення. Метою аналізу є визначення та локалізація окремих об’єктів, виявивши їх присутність, місцезнаходження (обмежуюча рамка) та класову мітку. Отже мережа відтворить кілька обмежуючих рамок навколо об’єктів

разом із відповідними класовими мітками. Застосовується для виявлення та локалізації різних об’єктів, таких як автомобілі, люди або велосипеди на зображенні. Тобто, модель вміє ідентифікувати конкретні об’єкти на зображенні, часто використовуючи методи пропозиції областей (детектори двох етапів) або пряме передбачення по всьому зображенню (детектори одного етапу).

Враховуючи характеристики двох типів більш ефективним у сенсі розширення функціоналу у майбутньому можна сказати що виявлення об’єктів не обмежується їх класифікацією. Окрім того модель має можливість класифікувати декілька об’єктів на одному зображенні.

Виявлення об’єктів як категорія включає у себе три типи нейронних мереж доступних для тренування. А саме:

RetinaNet: Це одноетапний детектор об’єктів, який призначений для виявлення об’єктів на зображеннях різних масштабів. Використовує Focal Loss та FPN. Проте, він не призначений для класифікації зображень як основне завдання.

Faster R-CNN: Це двоетапний детектор об’єктів, який пропонує регіони, де можуть знаходитися об’єкти, а потім класифікує ці регіони та локалізує об’єкти. Не є призначеним для класифікації зображень як таких.

Mask R-CNN: Ця модель є розширенням Faster R-CNN, яке додає можливість сегментації областей об’єктів (створення масок для виявлених об’єктів). Вона також не спеціалізується на класифікації зображень, але дозволяє отримати маску для кожного виявленого об’єкту.

З опису вище можна зробити висновок що RetinaNet найбільш підходячий тип мережі тому що окрім виявлення об’єктів також ефективно проводить їх класифікацію у той час коли Faster R-CNN та його модифікація Mask R-CNN призначені більше для локалізації а не для класифікації.

Предметом оптимізації системи може слугувати комбінація двох нейронних мереж, а саме використання Mask R-CNN для локалізації а затим використати іншу мережу безпосередньо створену для класифікації зображень. З іншого боку це значно ускладнить створення схеми бази даних для збереження зібраних даних для коджного зображення. Тому за для спрощення схеми бази даних було прийнято рішення

створити єдину модель яка спорможна ефективно виконувати локалізацію та класифікацію об’єктів а саме RetinaNet.

Наступний крок це визначення кількості шарів та їх конфігурація. Оптимальна кількість шарів для архітектури RetinaNet залежить від таких факторів як, складність набору даних, розмір і різноманіття об’єктів, обчислювальні ресурси та компроміс між точністю та швидкістю виведення.

Підбір оптимальних параметрів нейронної мережі являє собою важливий етап оптимізації. Для виконання даної дії можна використати метод Баєсівської оптимізації [21][22]. Вихідний код функції пошуку оптимальних вхідних параметрів нейронної мережі зображено на рисунку 3.1. На зображенні приведено функцію яку необхідно максимізувати. Необхідно знайти оптимальний розмір прихованого шару відповідно до швидкості навчання. Виведення роботи програми пошуку оптимальних параметрів зображено на рисунку 3.2. З чого видно, що оптимальний розмір прихованої мережі становить 50.86893414268804, а оптимальний час тренування має 0.0017144919599972192.

```
# Функція, яку необхідно максимізуватими хочемо максимізувати
def train_function(lr, hidden_size):
    # Задаємо параметри моделі та оптимізатора
    input_size = 13  # Припустимо розмір вхідних даних
    output_size = 4  # Припустимо розмір виходу
    model = SimpleNN(input_size, int(hidden_size), output_size)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    # Здійснюємо тренування моделі та повертаємо значення, яке хочемо максимізувати
    # Наприклад, точність або іншу метрику
    return torch.rand(1).item()


# Визначаємо простір параметрів, які хочемо оптимізувати
pbounds = {'lr': (0.001, 0.01), 'hidden_size': (16, 64)}

# Викликаємо байєсівську оптимізацію
optimizer = BayesianOptimization(
    f=train_function,
    pbounds=pbounds,
    random_state=44,
)

# Максимізуємо функцію втрати
optimizer.maximize(
    init_points=4,
    n_iter=11,
)

print(optimizer.max)

```

Рисунок 3.1 – Вихідний код функції пошуку оптимальних параметрів нейронної мережі

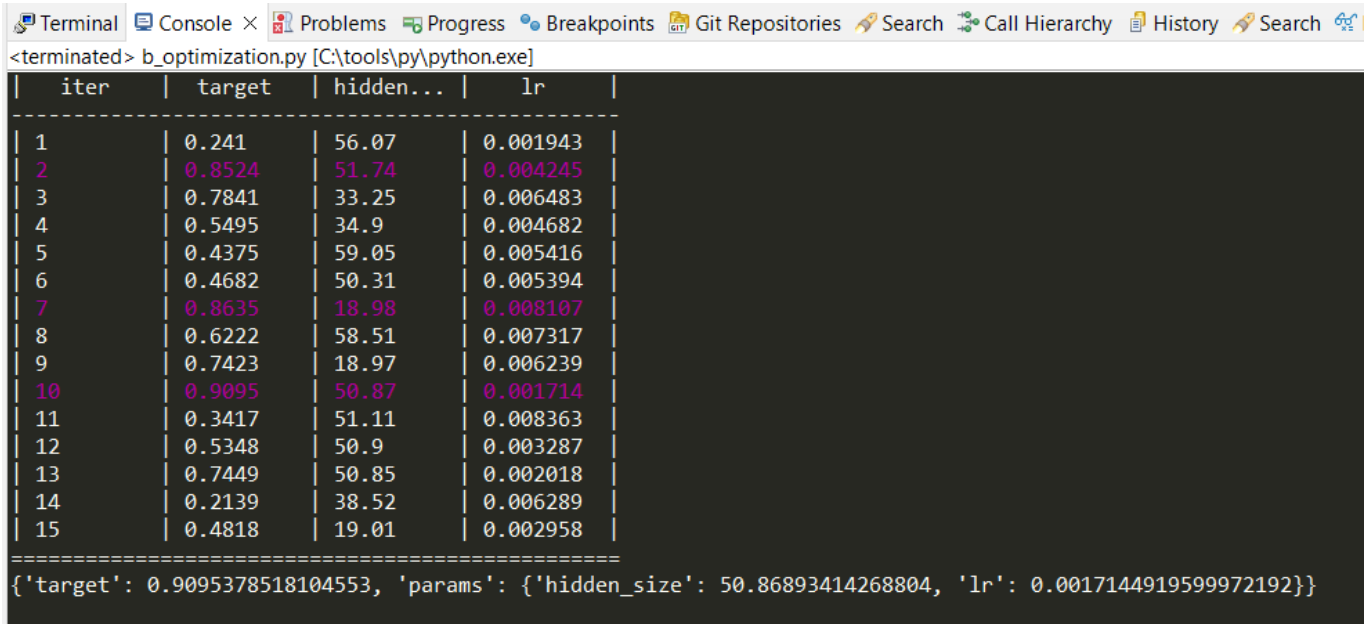

Рисунок 3.2 – Результат роботи пошуку оптимальних параметрів нейронної мережі

Власне RetinaNet побудована на основі «Feature Pyramid Network» (FPN) у поєднанні з підмережами класифікації та регресії обмежувальних рамок. Зазвичай вона використовує піраміду ознак з кількома рівнями, щоб виявляти об’єкти на різних масштабах. У свою чергу, глибокі мережі можуть захоплювати складніші ознаки, але можуть бути «важкими» у обчисленні та схильними до перенавчання, особливо якщо набір даних недостатньо великий. Менш глибокі мережі можуть працювати швидше, але не завжди здатні захопити складніші об’єкти.

Для ефективного тренування системи також важливо визначити функцію втрати. Функція втрати – це метрика, яка визначає, наскільки добре модель прогнозує цільові значення під час навчання. Основна ідея полягає в тому, щоб виміряти різницю між прогнозованими значеннями моделі та очікуваними значеннями. Вибір основної мережі (наприклад, ResNet-50, ResNet-101) також опосередковано впливає на кількість шарів, оскільки ці мережі мають різну глибину [23].

Під час навчання мережа поступово оновлює свої ваги, спробуючи зменшити значення функції втрати. Це виконується шляхом зменшення помилки між передбаченими значеннями та справжніми мітками у навчальних даних. Існують наступні стандартні функції:

Mean Squared Error (MSE) – середнє квадратичне відхилення між прогнозованими значеннями та справжніми.

Binary Cross Entropy Loss – використовується у випадку бінарної класифікації (вихід моделі - 0 або 1).

Categorical Cross Entropy Loss - використовується у випадку багатокласової класифікації (вихід моделі - ймовірності для кожного класу).

Cross Entropy Loss (Log Loss) – це одна з основних функцій втрати для багатокласової класифікації. Вона вимірює розбіжність між передбаченими ймовірностями класів та фактичними мітками класів. У разі бінарної класифікації використовується бінарний варіант цієї функції. Фактично, це метрика, яка використовується для вимірювання відстані між ймовірнісним розподілом передбачуваних класів та фактичними класами в задачах класифікації.

У задачах класифікації, де потрібно призначити об’єктам один або декілька класів, Cross Entropy Loss є популярною функцією втрат через свою здатність враховувати навчання моделі з ймовірнісною інтерпретацією.

1. Вибір конкретної функції втрати залежить від типу задачі, яку ви вирішуєте. Правильно обрана функція втрати допомагає моделі навчатися ефективно та досягати кращої продуктивності. Для класифікації зображень найбільш придатною вважається Cross Entropy Loss [24].

Важливою частиною тренування моделі являється оптимізація функції втрати. Власне оптимізація полягає у знаходженні її може бути оптимізована з метою знаходження її мінімуму.

Оптимізація функції втрати включає зміну параметрів моделі таким чином, щоб мінімізувати значення цієї функції. Найпоширеніший метод для цього – градієнтний спуск та його варіанти, приведено на рисунку 3.3.

```python
# Навчання моделі
epochs = 1000
for epoch in range(epochs):
    # Очищення градієнтів
    optimizer.zero_grad()

    # Прогноз моделі
    predictions = model(x_train)

    # Обчислення втрат
    loss = loss_function(predictions, y_train)

    # Обчислення градієнтів
    loss.backward()

    # Оновлення параметрів моделі
    optimizer.step()

    # Виведення втрат під час тренування
    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Отримання прогнозів після навчання
predicted = model(x_train)
print("Predicted values:", predicted.detach().numpy())
```

Рисунок 3.3 – Процес оптимізації функції втрати

Градієнтний спуск використовує обчислені градієнти для оновлення значень параметрів моделі так, щоб крок за кроком підходити до мінімуму функції втрати. Цей процес включає в себе вирахування нових значень параметрів, використовуючи поточні значення, градієнти та швидкість навчання [25].

Результат роботи програми з оптимізації функції втрати приведено на рисунку 3.4.

```
Console ×
<terminated> func_opt.py [C:\tools\py\python.exe]
Epoch [100/1000], Loss: 0.0354
Epoch [200/1000], Loss: 0.0194
Epoch [300/1000], Loss: 0.0107
Epoch [400/1000], Loss: 0.0059
Epoch [500/1000], Loss: 0.0032
Epoch [600/1000], Loss: 0.0018
Epoch [700/1000], Loss: 0.0010
Epoch [800/1000], Loss: 0.0005
Epoch [900/1000], Loss: 0.0003
Epoch [1000/1000], Loss: 0.0002
Predicted values: [[2.0203743]
 [4.009873 ]
 [5.9993715]
 [7.98887  ]]
```

Рисунок 3.4 – Оптимізація функції втрати

У цьому прикладі створюється проста модель лінійної регресії. Ми використовуємо середньоквадратичну помилку як функцію втрати та стохастичний градієнтний спуск як оптимізатор для мінімізації цієї функції втрати. Після тренування ми отримуємо прогнозовані значення моделі для вхідних даних.

Підготовка даних для навчання це найбільш складна із точки зору об’єму роботи частина. Особливо коли мова йде про навчання мережі по класифікації зображень.

Кожне зображення має набір унікальних параметрів таких як: розмір та кольорова палітра. Комп’ютер, у свою чергу, «бачить» зображення як набір числових значень, розташованих у визначених координатах матриці. Також зображення може містити декілька об’єктів різного розміру, наприклад в результаті віддалення об’єктів на зображенні одне від одного. Саме для цього застосовується механізм згортки, тобто об’єкти на зображенні спочатку визначаються (визначається прямокутна зона на зображенні де попередньо відомо щось знаходиться) а потім кожен визначений об’єкт класифікується окремо, як окреме зображення. Причому, перед тим як «згодувати» визначену зону нейронній мережі для класифікації, необхідно нормалізувати, тобто перевести частину зображення у вигляд матриці визначеної розмірності. Значення кожного елементу матриці вказує на ступінь яскравості чи кольору кожного пікселя. Використовуючи цю інформацію, алгоритми машинного навчання можуть аналізувати, класифікувати та взаємодіяти з графічними даними.

Кожне зображення інтерпретується як матриця, де кожен елемент представляє інтенсивність пікселя, зображено на рисунку 3.1. Матричне представлення це найбільш загальний спосіб представлення зображень у вигляді числових даних. Зображення розбивається на пікселі, кожен з яких представляється числовим значенням, яке відображає яскравість (для чорно-білих зображень) або значення каналів (для кольорових зображень - червоний, зелений, синій).

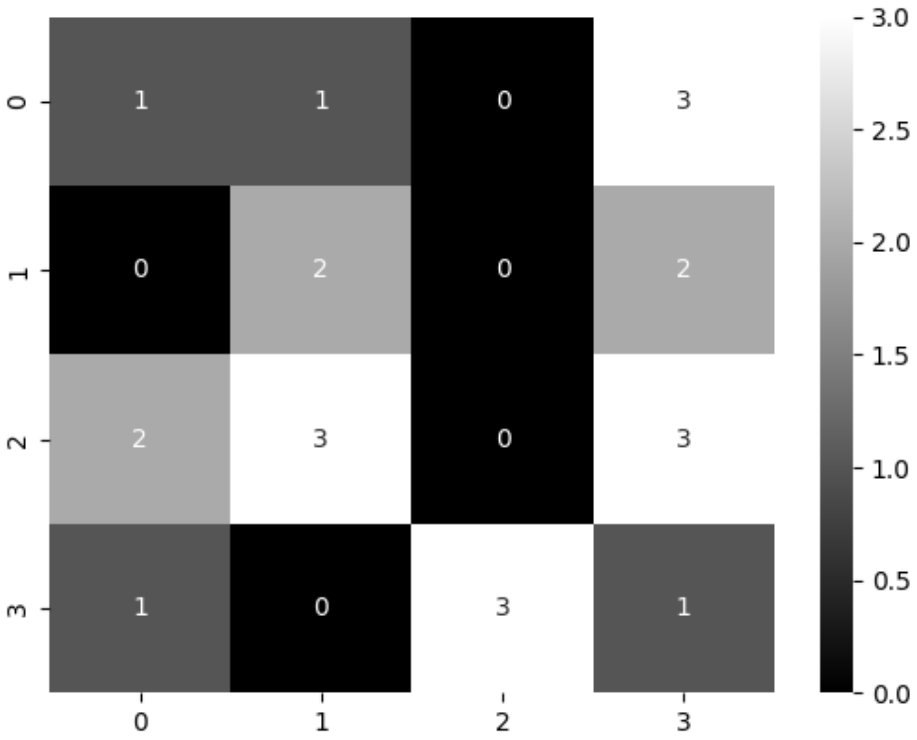

Рисунок 3.1 – Матриця відтінків (чорно–білий)

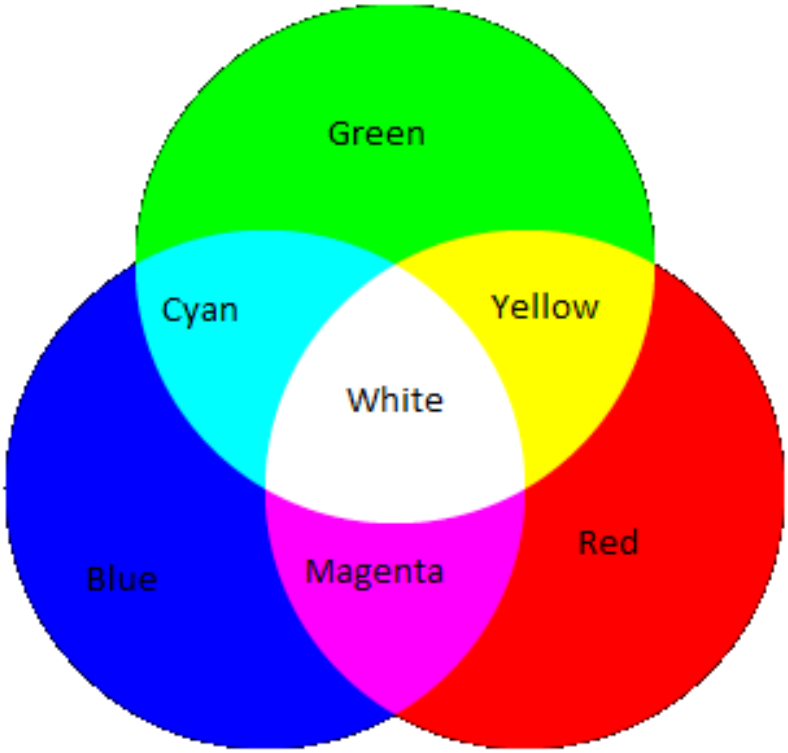

Рисунок 3.2 – Матриця створення нового кольору шляхом перетину 3 кольорів(червоного, зеленого, синього)

Для кольорових – три матриці для кожного каналу RGB: R(x, y), G(x, y), B(x, y), зображено на рисунку 3.2 [26].

Для прикладу, у бібліотеках таких як PyTorch та TenzorFlow дана матриця називається тензором, що являє собою абстракцію даних наданих для класифікації. Слід також зазначити, що подібним чином працює доставлення даних для аналізу будь-якого типу. Тобто будь-яка інформація, яку можна подати у вигляді матриці певної розмірності придатна для того щоб проводити аналіз цих даних за допомогою штучного інтелекту.

Умовний вигляд набору даних виглядає наступним чином. Зображено на рисунку 3.3. Відповідно у папці Images містяться зображення, а у папці Annotations знаходиться описання класу зображення з координатами відповідного об’єкта на відповідному зображенні із папки Images, зображено на рисунку 3.4.

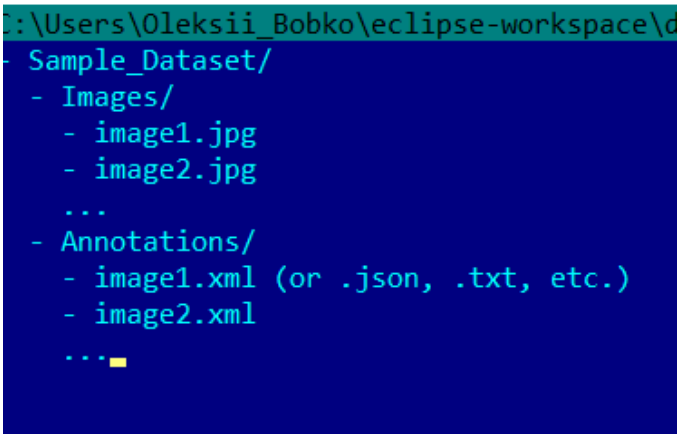

Рисунок 3.3 – Умовна структура розміщення файлів

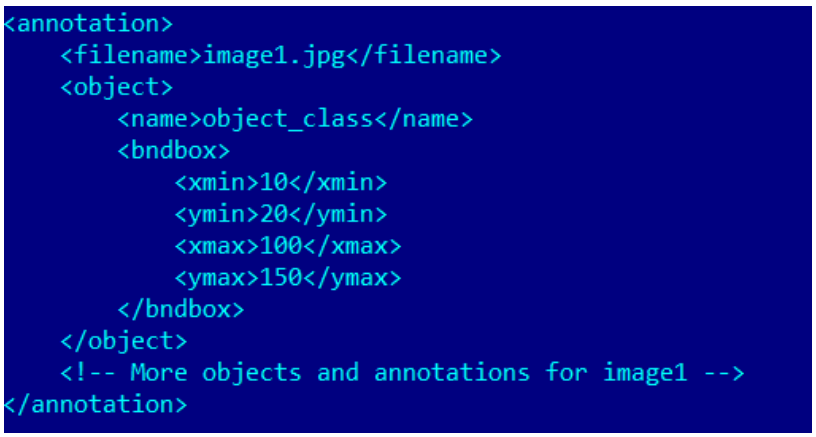

Рисунок 3.4 – Анотація зображення набору даних

Читання, обробка та підготовка даних для використання у моделі може включати масштабування, нормалізацію, розбиття на тренувальний та тестувальний набори тощо.

Існує декілька підходів навчання нейронних мереж, основні ж 3 основні типи: це навчання з учителем (Supervised learning), навчання без учителя (Unsupervised leaning), та навчання з передачею знань (Transfer learning). Розглянемо кожен тип більш детально:

- Навчання з учителем (Supervised Learning): Це тип навчання, при якому модель навчається на основі вхідних даних, які мають відповідні мітки або маркування. Модель намагається передбачити відповіді на основі навчального набору даних, а потім коригується на основі різниці між передбаченими та фактичними відповідями. Задачі, такі як класифікація та регресія, часто розв’язуються за допомогою навчання з учителем.
- Навчання без учителя (Unsupervised Learning): У цьому типі навчання модель намагається виявити приховані структури чи залежності у вхідних даних без будь-яких міток або маркувань. Зазвичай це включає кластеризацію, зменшення розмірності, асоціативне навчання та інші методи.
- Навчання з передачею (Transfer learning): Це метод машинного навчання, при якому модель, навчена на одній задачі, використовується або адаптується як стартова точка для нової, пов’язаної задачі. У transfer learning знання, отримане при розв’язанні однієї проблеми, передається та застосовується для іншої, але пов’язаної проблеми, зазвичай за допомогою попередньо навчених моделей, що були навчені на великих наборах даних. Основна ідея за transfer learning полягає в тому, що модель, навчена на великому і загальному наборі даних, може слугувати сильною стартовою точкою для іншої, але пов’язаної задачі. Замість навчання нової моделі з нуля, transfer learning включає використання знань або ознак, які були вивчені попередньо навченою моделлю, та їх доналаштування на меншому наборі даних, специфічному для нової задачі.

Наступний етап після завершення навчання це тестування моделі та оцінка та тестування моделі. Відбувається підготовка тестових даних, тобто завантаження

тестового набору даних або створення власного. Ці дані зазвичай не використовуються під час тренування моделі, і їх використання дозволяє оцінити загальну продуктивність моделі. Наступна фаза передбачає передачу тестових даних у модель для отримання результатів. Це може виконуватися частинами або цілими тестовими наборами даних. За тим відбувається оцінка продуктивності. Це обчислення метрик продуктивності, таких як точність (accuracy, precision). Аналіз результатів відбувається за допомогою оцінка результатів та визначення сильних і слабких сторін моделі. Це може включати аналіз помилкових передбачень, визначення сценаріїв, на яких модель працює краще або гірше, та ідентифікацію областей для покращення.

Подальше вдосконалення моделі залежить від результатів тестування, можуть бути запропоновані покращення для моделі, включаючи підвищення точності, швидкодії, оптимізацію параметрів або зміни архітектури.

У разі якщо характеристики моделі відповідають замовленим критеріям, модель використовують у системі. Зазвичай для подальшого тестування більш високого рівня. Наприклад тестування на обраній групі людей у певній географічній локації.

3.4 Розробка графічної та функціональної частини додатку.

Далі розглянемо основні етапи створення даного додатку та розглянемо структуру проекту та його програмну реалізацію. Як вже зазначено вище, для розробки додатку використовується мова програмування Python [27].

Python, дозволяє встановлювати бібліотеки відразу з одного файлу, рисунок 3.5 ввівши в термінал Windows команду pip install -r requirements.txt.

Наступним етапом розробки це виконання наступних завдань:

1. Функціонал, що надає можливість взаємодіяти з інтерфейсом.
2. Можливість аналізувати та класифікувати зображення.
3. Створення функцій для взаємодії з базою даних.

requirements.txt

```
attrs==23.1.0
blinker==1.6.3
certifi==2023.7.22
charset-normalizer==3.3.0
click==8.1.7
contourpy==1.1.1
cycler==0.12.1
Cython==3.0.3
filelock==3.12.4
Flask==3.0.0
fonttools==4.43.1
fsspec==2023.9.2
idna==3.4
imageai==3.0.3
iniconfig==2.0.0
itsdangerous==2.1.2
Jinja2==3.1.2
kiwisolver==1.4.5
MarkupSafe==2.1.3
matplotlib==3.8.0
mock==4.0.3
mpmath==1.3.0
networkx==3.1
numpy==1.26.1
opencv-python==4.8.1.78
packaging==23.2
Pillow==10.1.0
pluggy==1.3.0
py==1.11.0
pyparsing==3.1.1
pytest==7.1.3
python-dateutil==2.8.2
requests==2.31.0
scipy==1.11.3
six==1.16.0
sympy==1.12
tomli==2.0.1
torch==2.1.0
```

Рисунок 3.5 – Бібліотеки Python, використані в розробці

Для реалізації вище зазначеного функціоналу структура програми має наступний вигляд зображено на рисунку 3.6.

Розглянемо окремо кожен компонент структури, зображено на рисунку 3.7.Розпочнемо з головного компоненту web.py. Даний компонент відповідає за створення веб додатку використовуючи бібліотеку Flask. Дана бібліотека надає

можливість обробляти HTTP-запити, отримувати дані від користувачів і формувати відповіді. Для відображення веб сторінок Flask використовує шаблонізатор Jinja2. Flask має вбудований веб-сервер, який використовується для тестування та розробки вашого додатку на локальному комп'ютері. Таким чином спрощується робота з HTML-шаблонами та статичними файлами, дозволяючи організувати структуру веб-додатка.

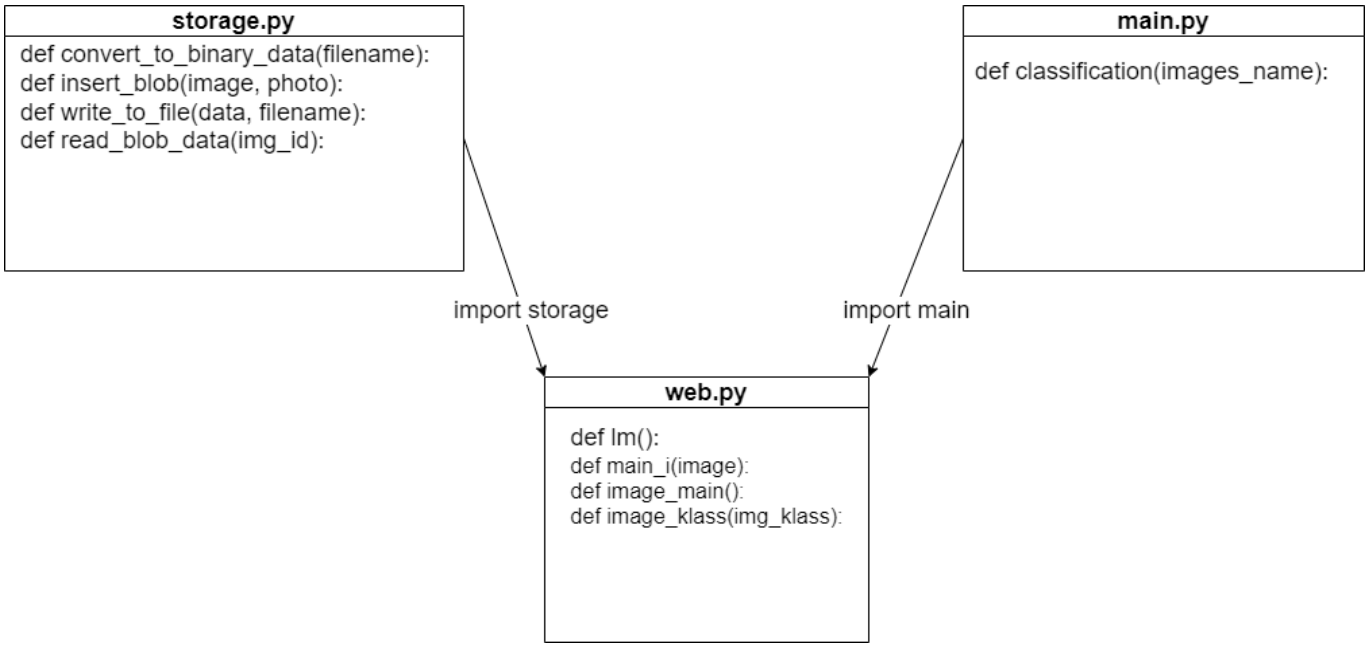

Рисунок 3.6 – Структура програмної реалізації застосунку

Варто зауважити, що за замовчуванням додаток Flask запускається у режимі розробки, задля розгортання додатку готового до великих навантажень необхідно виконати ряд додаткових дій. А саме, обрати WSGI сервер для розгортання Flask додатку у режимі штатної роботи. Зазвичай використовують такі WSGI (Web Server Gateway Interface) сервери: Gunicorn або uWSGI. Після вибору WSGI сервера, потрібно налаштувати сервер та його конфігурацію для роботи з Flask додатком. Це включає встановлення сервера, налаштування файлів конфігурації та створення скрипту для запуску додатку.

```python
from flask import Flask, render_template, url_for, redirect, request
import main
import storage


app = Flask(__name__)


@app.route('/local')
def lm():
    return render_template('index.html')

@app.route('/<name>', methods=['POST'])
def main_i(image):
    print(image)
    f = request.files['image']
    main.klasification(image)
    return render_template('index.html')
```

Рисунок 3.7 – Завантаження веб-додатку та його сторінки.

Також web.py використовує функції компонентів (storage, main) для зв’язку з базою даних та передачі фото для подальшого аналізу зображено на рисунку 3.8.

```python
@app.route('/main', methods=['POST'])
def image_main():
        if request.method == 'POST':
            f = request.files['image']
            f.save("static/images/" + f.filename)
            klass = main.classification(f.filename)
            return render_template('index.html', list=klass, image="/images/" + f.filename + "new.jpeg")

@app.route('/main/<string:img_klass>')
def image_klass(img_klass):
    ans = storage.read_blob_data(img_klass)
    print(ans)
    new_ans = list()
    for i in ans:
        ans = i.split('/')
        new_ans.append(ans[-1])
    return render_template('answer.html', list=new_ans)
```

Рисунок 3.8 – Функції image_main, image_klass

Наступний компонент – storage. Даний компонент використовується для взаємодією з базою даних image_kod та таблиці images, що використовується для зберігання зображень, для цього використовується бібліотека sqlite3, рисунок 3.9.

```python
def convert_to_binary_data(filename):
    with open(filename, 'rb') as file:
        blob_data = file.read()
    return blob_data

def insert_blob(image, photo):
    try:
        sqlite_connection = sqlite3.connect('image_kod')
        cursor = sqlite_connection.cursor()
        print("Connected к SQLite")

        sqlite_insert_blob_query = """INSERT INTO Images
                                  (image, kod) VALUES (?, ?)"""

        emp_photo = convert_to_binary_data(photo)
        data_tuple = (image, emp_photo)
        cursor.execute(sqlite_insert_blob_query, data_tuple)
        sqlite_connection.commit()
        cursor.close()
```

Рисунок 3.9 – Функція для підключення до бази даних

Також даний компонент може отримати всі фотографії певної категорії використовуючи наступну функцію зображено на рисунку 4.7.

```python
def read_blob_data(img_id):
    im = list()
    try:
        sqlite_connection = sqlite3.connect('image_kod')
        cursor = sqlite_connection.cursor()
        print("Connected to SQLite")

        sql_fetch_blob_query = """SELECT * from Images where image = ?"""
        cursor.execute(sql_fetch_blob_query, (img_id,))
        record = cursor.fetchall()
        for row in record:
            print("Id = ", row[0], "Name = ", row[1])
            name  = row[1]
            photo = row[2]

            print("save on disk \n")
            photo_path = "static/images/" + name + str(random.randint(1,10000)) + ".jpg"
            write_to_file(photo, photo_path)
            im.append(photo_path)
        cursor.close()
```

Рисунок 3.10 – Функція read_blob_data()

Останній компонент додатку – це main. Цей код використовує бібліотеку ImageAI яка у свою чергу використовує PyTorch для виявлення об’єктів на зображеннях. Він завантажує модель, використовуючи навчений файл, а потім використовує цю модель для виявлення об’єктів на зображенні, ім’я якого

передається як аргумент. Для кожного виявленого об’єкта код перевіряє, чи відповідає він певним категоріям (наприклад, ‘cat’, ‘car’, ‘dog’, ‘cow’ або ‘horse’), і якщо так, вставляє зображення в базу даних SQLite з відповідною міткою зображено на рисунку 3.11.

```
from imageai.Detection import ObjectDetection
import os
from storage import insert_blob

def classification(images_name):
    images_name1 = images_name

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath("coco_resnet_50_map_0_335_state_dict.pt")
    detector.loadModel()
    detections = detector.detectObjectsFromImage("static/images/" + images_name1, "static/images/" + images_name1 + "new.jpeg")

    klass = list()

    for eachObject in detections:
        insert_blob(eachObject["name"], "static\\images\\" + images_name1)
        klass.append(eachObject["name"] + " : " + str(eachObject["percentage_probability"]) + ", " + str(eachObject["box_points"]));
        print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObject["box_points"] )

    return klass
```

Рисунок 3.11 – Компонент main

Для створення інтерфейсу, як вже зазначалося використовуємо html, рисунок 3.12.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href = "{{ url_for('static', filename='css/main.css')}}">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    {% block body %}{% endblock %}
</body>
</html>
```

Рисунок 3.12 – Розмітка головної сторінки

Даний інтерфейс має доволі простий функціонал, 2 кнопки для додавання та аналізу зображення, рисунок 3.13.

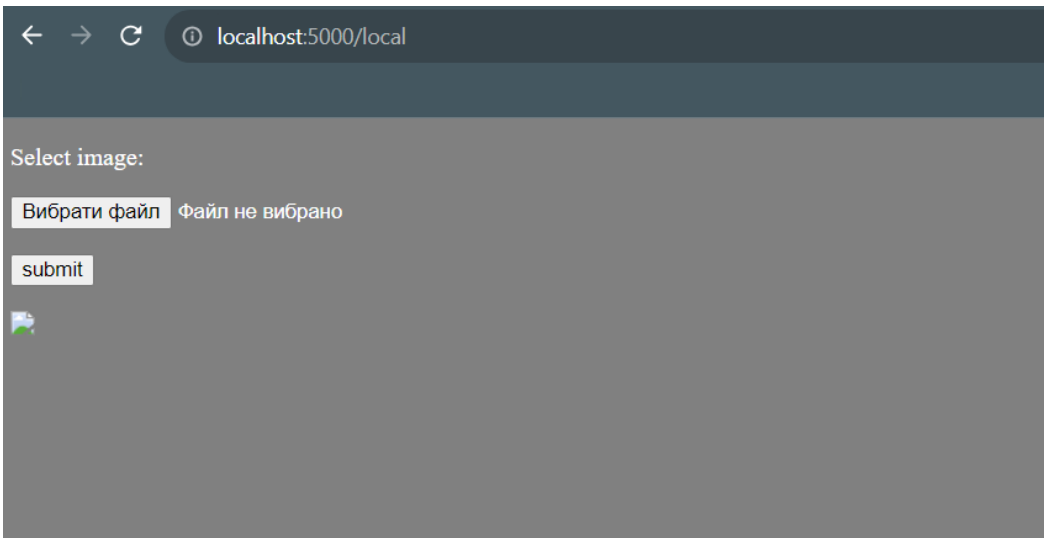

Рисунок 3.13 – Інтерфейс додатку

Остання частина розробки додатку є створення бази даних, та зберігання інформації в одну з таблиць цієї бази. Для цього було створено базу даних image_kod. Для зберігання картинок використовувалася таблиця images. Щоб зберегти зображення, потрібно перевести її в двійковий формат – BLOB, а вже потім додавати до таблиці, рисунок 3.14.

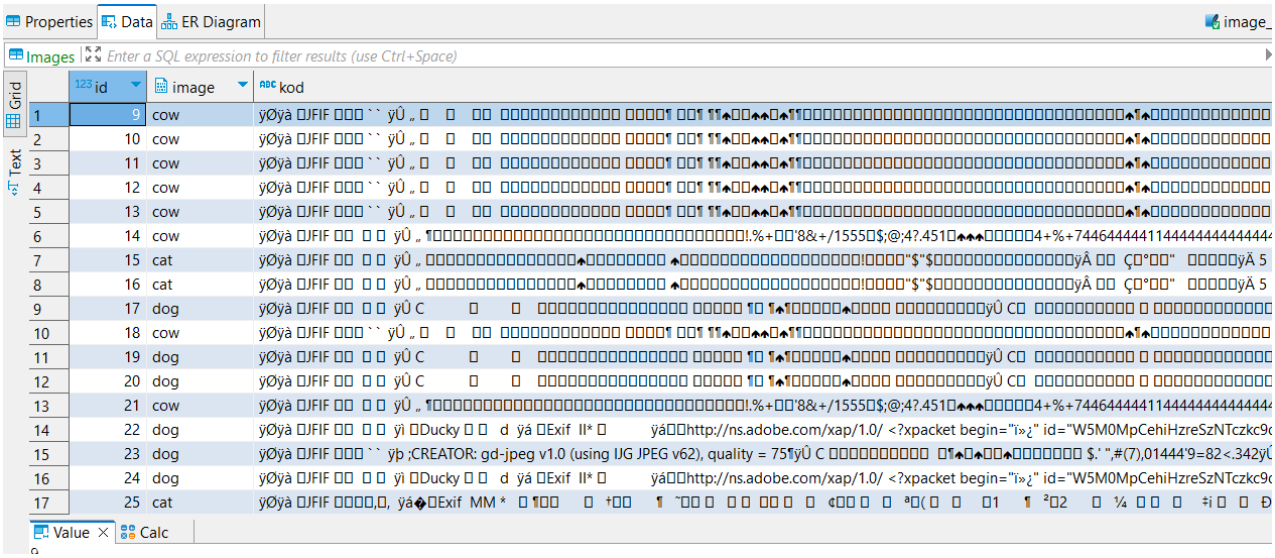

Рисунок 3.14 – Таблиця images бази даних image_kod

Отже в результаті розробки, отримано додаток, що дозволяє аналізувати зображення, визначати його категорію та зберігати в базу даних.

3.5 Висновки

У даному розділі було досліджено надзвичайно важливі області необхідні для побудови ефективної системи класифікації зображень. Було визначено оптимальну для виконання проекту мову програмування Python. Обрано бібліотеку яка містить необхідний функціонал для побудови та тренування моделі класифікації зображень. За тим було знайдено необхідне середовище розробки та тренування моделей Google Colab, який дозволяє зберігати великі набори даних для тренування, описувати архітектуру моделі, та проводити тренування. За допомогою інтегрованої підтримки GPU та TPU тренування моделі може відбуватись досить швидко використовуючи надзвичайно великі набори навчальних даних [28][29].

Подання зображення на обробку нейронною мережею відбувається за допомогою відображення даних у вигляді матриці визначеного розміру та кодування елементів матриці. Тобто кожен елемент матриці це число. У випадку із зображеннями значення числа може означати колір RGB або альфа канал. Однак, будь-які дані представлені у вигляді матриці підходять для використання їх як дані для класифікації нейронною мережею. Важливо розуміти, найголовніше це створити модель, тобто натренувати модель, нейронну мережу, класифікувати щось визначене. Іншими словами, ми можемо щось знайти, коли знаємо що шукаємо.

Саме механіка подання даних доводить перспективність напрямку машинного навчання. Класифікація даних можна використовувати у багатьох видах діяльності людини. Нейронні мережі здатні розпізнавати важкі хвороби на базі історії хвороби людини, або розпізнати шахрайське переведення коштів з одного рахунку на інший.

Ще одна велика перевага систем які використовують машинне навчання це те що модель можна перебудувати і покращити і далі замінити існуючу модель без жодного ризику що система буде працювати не вірно. Іншими словами, користувач буде все ще бачити систему до якої він звик, то самий звичний дизайн, а насправді, з кожним оновленням моделі система буде ставати все більш «розумною» і з рештою працюватиме краще і краще без необхідності переписувати існуючий код, який просто забезпечує графічний інтерфейс, та зберігання даних.

4 ТЕСТУВАННЯ СИСТЕМИ

Процес тестування програмного забезпечення – це послідовний набір дій, спрямованих на перевірку програмного продукту з метою виявлення помилок, переконливості у відповідності вимогам та забезпечення його якості. Цей процес включає в себе ряд етапів та дій, які зазвичай виконуються на різних стадіях розробки програмного забезпечення.

Першим кроком є ретельний аналіз вимог до програмного забезпечення. Це допомагає зрозуміти очікувані функції, функціональність та критерії успішності для програми.

Наступний етап це планування тестування, на цьому етапі визначаються стратегії, методи, ресурси, терміни та обсяги тестування. Розробляється план тестування, який описує загальну стратегію тестування та конкретні кроки для кожного етапу.

На основі вимог розробляються конкретні тестові сценарії або випадки, які охоплюють різні аспекти функціональності та властивостей програми.

Під час виконання тестів інженери-тестувальники виконують створені тестові випадки. Вони запускають програмне забезпечення, вводять дані, виконують дії та аналізують результати для виявлення помилок.

У випадку якщо під час тестування виявляються помилки або дефекти, вони реєструються в системі управління дефектами. Дефекти відстежуються, перевіряються та виправляються.

Після завершення тестування створюється звіт, який містить інформацію про виконані тести, виявлені дефекти, оцінки якості програмного забезпечення тощо. Цей звіт надає інформацію розробникам та замовникам щодо якості продукту.

У разі виправлення дефектів проводиться повторне тестування для перевірки виправлень та підтвердження їх коректності.

4.1 Методи та методики тестування програмного додатку

Існує багато різних методів та методик тестування програмного забезпечення, які використовуються для перевірки його якості та відповідності вимогам. Технологічно тестування можна поділити на загальних домени, це: мануальне тестування, автоматизоване тестування та тестування на рівні коду. Відповідно до назв – мануальне тестування це виконання послідовності дій для виявлення помилок у поведінці програмного забезпечення. Автоматизоване – так як і мануальне передбачає виконання послідовності дій але не мануальних, а за допомогою засобів автоматизації тестування таких як: Selenium, Cucumber, та інших у тому числі специфічних засобів написаних різними мовами програмування що призначені для тестування якогось конкретного програмного продукту.

Також методи та методики тестування можна поділити на такі групи. Дане розділення визначає загально відомі практики та підходи задля полегшення описання стратегії тестування відповідно до специфіки проекту. Наприклад, використання End-to-End може означати, що система яку необхідно тестувати використовує Веб-сервіси. Тож розглянемо деякі загально вживані методи та методики тестування програмного забезпечення більш детально.

Метод білого ящика (White Box Testing) – тестування на основі структури програми, де інженер-тестувальник має доступ до вихідного коду та виконує тести з врахуванням внутрішньої структури програми.

Метод чорного ящика (Black Box Testing) – тестування, яке проводиться без знання внутрішньої структури програми. Інженер-тестувальник перевіряє функціональність програми на основі специфікацій та вимог.

Метод сірого ящика (Grey Box Testing) – Комбінація методів білого та чорного ящика. Тестувальник має обмежений доступ до внутрішньої структури програми, наприклад, доступ до бази даних або інших деталей.

Модульне тестування (Unit Testing) – Перевірка окремих модулів або компонентів програмного забезпечення для переконливості у їх коректності. Зазвичай модульні тести розробляються програмістом відповідно до частини програми яку було розроблено та додано до репозиторію вихідного коду.

Інтеграційне тестування (Integration Testing) – Тестування взаємодії між різними модулями або компонентами програми. Даний вид тестів також зазвичай пишуть самі програмісти. Суть взаємодії між компонентами полягає у тому що інтеграційні тести дають змогу відтворити умови наближені до реальних задля того щоб протестувати взаємодію, тобто інтеграцію.

Системне тестування (System Testing) – Перевірка системи як єдиної одиниці для переконливості у відповідності вимогам та функціональності.

Приймальне тестування (Acceptance Testing): Тестування, що виконується замовником або кінцевим користувачем для переконливості у відповідності програми вимогам та придатності для використання. Зазвичай даний вид тестування виконується у середовищі максимально наближеному до промислового. Інколи приймальне тестування також виконується певною групою користувачів програми з обраної групи.

Навантажувальне тестування (Load Testing) – Випробування системи зі значним навантаженням, щоб переконатися, що вона може працювати ефективно в умовах реального використання. Власне ціль даного тестування полягає у тому щоб визначити максимальне навантаження за якої програма все ще продовжує працювати у штатному режимі тобто має заявлену пропускну здатність та затримку виконання запитів.

Стрес-тестування (Stress Testing) – Тестування системи на межі її можливостей, визначення точки перевантаження та оцінка поведінки в екстремальних умовах. На відміну від навантажувального тестування ціль стрес-тестування полягає у визначенні порогового навантаження коли система перестає працювати коректно а саме значенні пропускної здатності та затримки перевищує заявлені значення.

Тестування безпеки (Security Testing) – Перевірка програми на вразливість та забезпечення заходів для захисту від потенційних атак.

4.2 Тестування програмного засобу.

Тестування програмного засобу полягає у виконанні тестового плану. Тестовий план являє собою набір кроків які необхідно виконати після чого необхідно

перевірити чи програмний додаток працює вірно, результат роботи програми є ідентичним до того що описано у тестувальному плані. Тестувальний план розробленої системи наведено у таблиці 4.1. Варто зауважити що тестовий план це не доконаний список. У разі розширення функціоналу або виявлення програмних помилок інженер-тестувальник повинен додати відповідний запис до тестового плану. Також зазвичай визначають ряд критично важливих тестових сценаріїв що повинні виконуватися перед кожним випуском нової версії програми. Так зване регресійне тестування. Таким чином можна пересвідчитися що доданий функціонал не спричинив недоліків у функції системи що були додані раніше. Зазвичай регресійне тестування намагаються автоматизувати, що потенційно може пришвидшити процес випуску нової версії [30].

Після закінчення розробки програми, потрібно провести тестування, перевірити функціонал додатку на коректність її роботи та аналізу графічної інформації. Використовуючи визначений тестовий план виконаємо тестування системи.

Для початку запустимо програму, для цього запустимо головний скрипт web.py, зображено на рисунку 4.1.

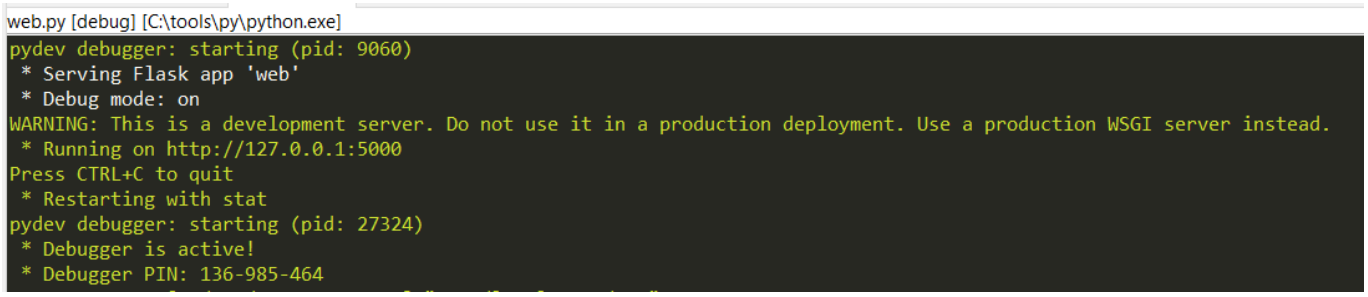

Рисунок 4.1 – Запуск web.py

Після запуску даного скрипта можна спостерігати, як консоль виводить адресу на якому знаходиться веб–додаток: http://127.0.0.1:5000.

Таблиця 4.1 – Тестувальний план програмного додатку

Тип сценарію	**Опис**	**Кроки для виконання**	**Очікуваний результат**
Функціональний	Завантаження графічного зображення у систему	1. Перейти за посиланням http://127.0.0.1:5000/local 2. Натиснути кнопку «Завантажити файл» 3. Обрати файл з зображенням. 4. Натиснути кнопку «Submit»	Файл завантажено та відображено на сторінці
Функціональний	Завантажене графічне зображення що містить об’єкти повинно бути розпізнаним та класифікованим	1. Завантажити зображення яке містить об’єкт. Наприклад автомобіль	Файл завантажено та відображено на сторінці з помітками у вигляді зелених прямокутників що розташовані навколо розпізнаних об’єктів
Функціональний	Необхідно мати можливість знайти усі завантажені та розпізнані графічні зображення за назвою визначеного об’єкта	1. Перейти за посиланням http://localhost:5000/main/<name>, де neme – це назва шуканих об’єктів, наприклад «car» Завантажити зображення яке містить об’єкти. Наприклад автомобіль	Переконатися, що усі зображення що мають розпізнаний об’єкт з назвою «car» виведені на сторінку
Функціональний	Застосунок повинен вміти розпізнавати та визначати більше одного об’єкта на зображенні.	1. Завантажити зображення, яке містить більше одного об’єкта для розпізнавання	Файл завантажено. Більше одного об’єкта на зображенні виділено зеленою лінією. Кожен об’єкт отримав відповідну класифікацію.
Функціональний	Застосунок повинен вміти знаходити завантажені зображення, які містять більше одного об’єкта	1. Завантажити зображення, яке містить більше одного об’єкта. 2. Ввести запит пошуку для одного з об’єктів.	Файл завантажено та проаналізовано. Запит з пошуку одного з об’єктів повертає зображення що також містить інші об’єкти.

Внаслідок цих дій завантажилася наступна сторінка, рисунок 4.3. У даному випадку веб-додаток на основі бібліотеки Flask було запущено у режимі розробки та

тестування. Цей режим роботи є зручним для розробки, оскільки він надає автоматичне оновлення веб-додатку при зміні коду, а також детальніші повідомлення про помилки. Запуску, веб-сервера Flask у даному режимі буде використовувати порт 5000. Також при розробці він буде автоматично перезавантажуватися при зміні коду, що дозволяє вам швидко бачити зміни під час розробки.

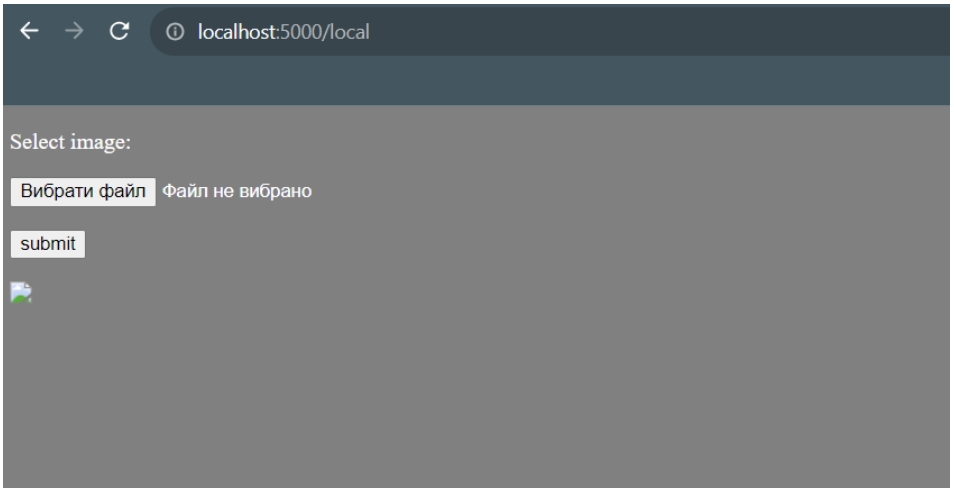

Рисунок 4.3 – Інтерфейс додатку

Далі потрібно обрати фото для перевірки, потрібно також звернути увагу, що модель може розрізняти наступні категорії cat, car, dog, cow. Для тестування пропонується наступне зображення, рисунок 4.4.

Рисунок 4.4 – Приклад зображення для тестування

Вибираємо дане зображення на сторінці, рисунок 4.5.

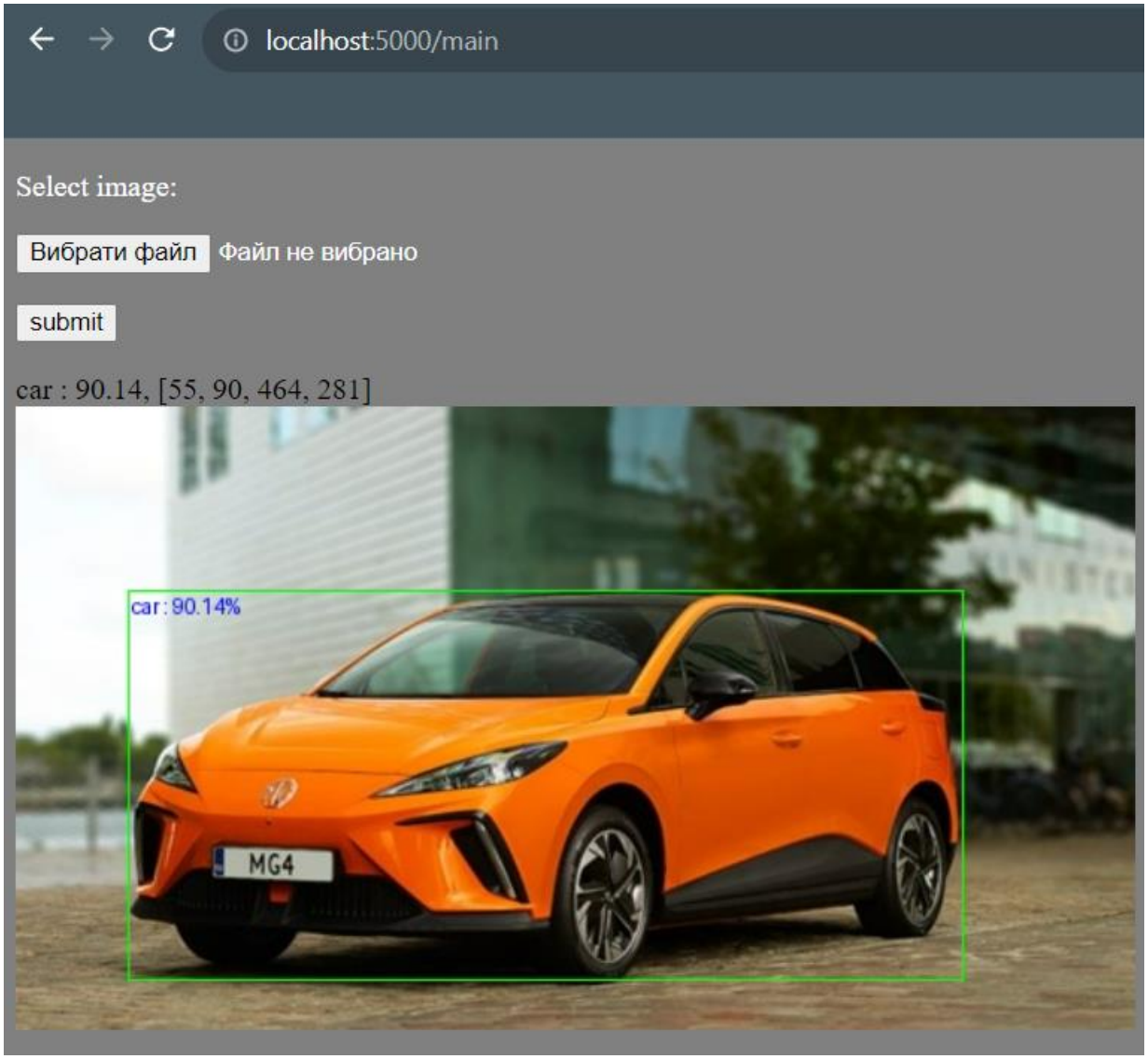

Рисунок 4.5 – Збережене зображення

Також, враховуючи те що використана нейронна мережа не являється лише мережею для класифікації а й мережею для захоплення об’єктів. Спробуємо завантажити зображення що містить 2 об’єкти, а саме кота і собаку. Результат роботи програми представлено на рисунку 4.6. Як видно на зображенні захоплені зони виділені лінією. Нейронна мережа визначила границі об’єктів та класифікувала їх із вірогідністю 89.25% для «dog» та 87.55% для «cat». Така сама механіка використовується при побудові систем з виявлення та відстеження пішоходів,

транспортних засобів та інших об’єктів на дорозі для автопілотів транспортних засобів.

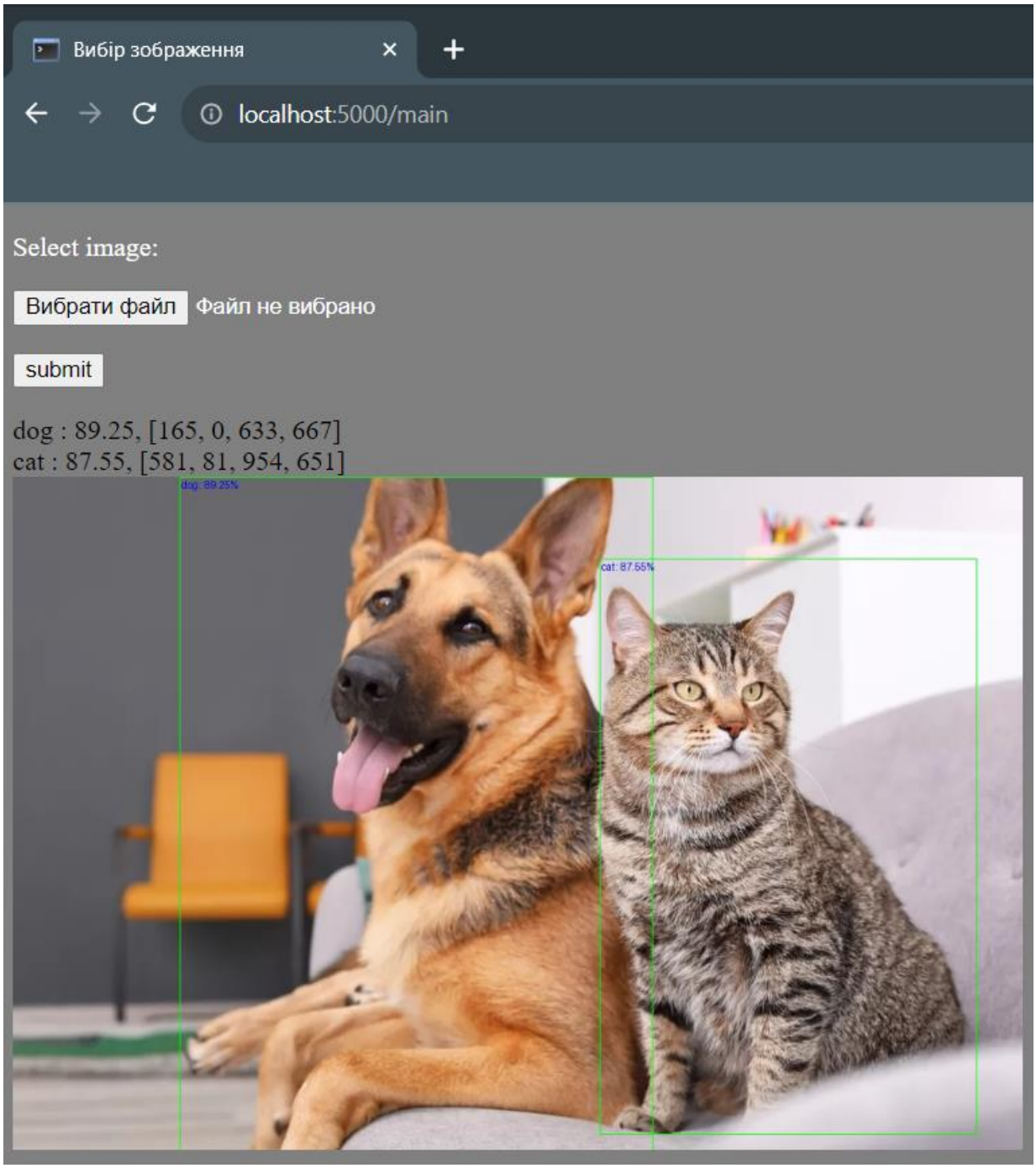

Рисунок 4.6 – Результат роботи програми по розпізнаванню декількох об’єктів на одному зображенні.

Після того як натиснули sumbit, програма вже проаналізувала та визначила її категорію, про що можемо впевнитися переглянувши базу даних, зображено на

рисунку 4.7. Дані зображень збережено у форматі BLOB що розшифровується як Binary Large Object та використовується для позначення двійкових даних, які представляють собою структуровані файли, зазвичай великого розміру.

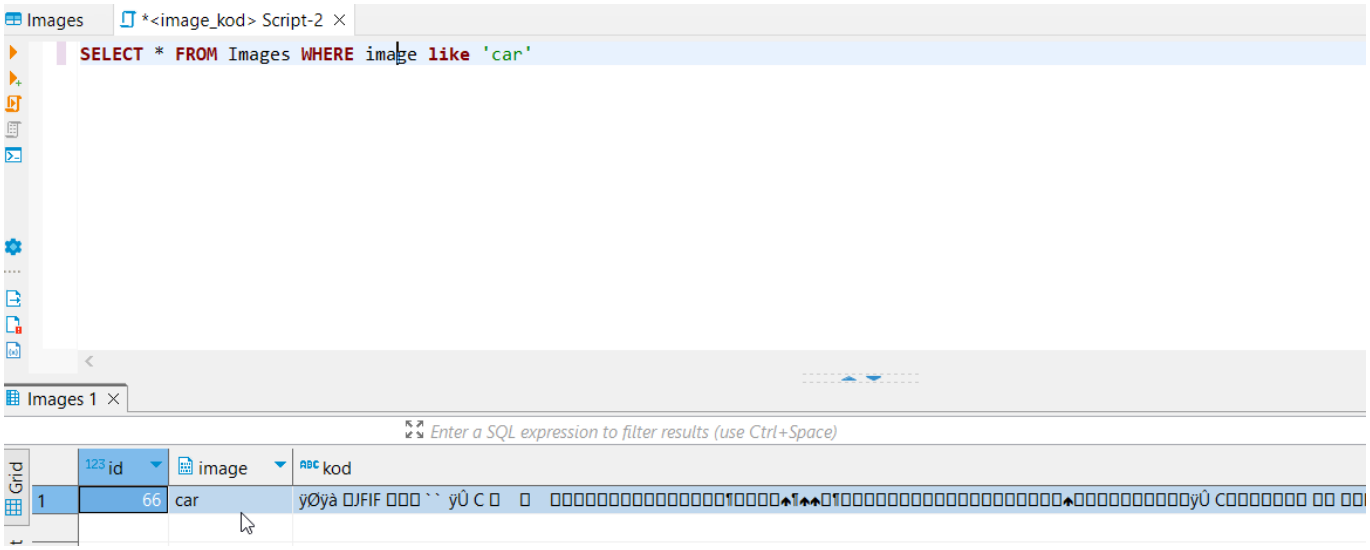

Рисунок 4.7 – Додавання нового фото в базу даних

Далі введемо в рядок наступну адресу: http://localhost:5000/main/car, представлено на рисунку 4.8.

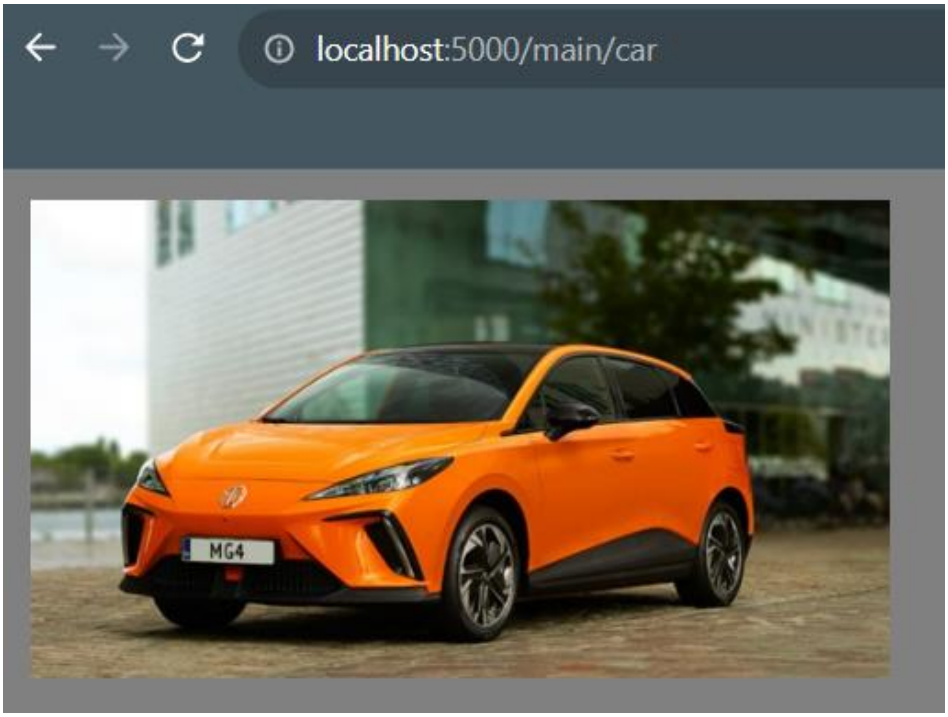

Рисунок 4.8 – Виведення всіх зображень категорії «машина».

Також можна спробувати інші запити. Наприклад http://localhost:5000/main/cow

Рисунок 4.9 – Зображення які система розпізнала як cow «корова»

На рисунку 4.9 на запит cow «корова» програма очевидно зробила помилкове виведення. Це гарний приклад того що штучний інтелект у деяких випадках не дає 100% гарантію коректності результатів. У цьому випадку необхідно враховувати ціну помилки. Дана неточність може бути обумовленою такими факторами як. нестача

репрезентативних прикладів для навчання, вона може показувати неточні результати. Не вистачає різноманітності та обсягу даних для вивчення різних варіантів та сценаріїв. З іншого боку іноді може виникати феномен перенавчання. Це ситуація, коли модель надто добре адаптується до навчальних даних, втрачаючи здатність узагальнювати на нових даних. В результаті вона може виявляти високу точність на навчальних даних, але погану точність на тестових або реальних даних.

Отже, внаслідок отримання зображення з відповідної категорії, можна впевнитися, що програма працює вірно та без помилок настільки наскільки добре натренована нейронна мережа.

4.3 Висновки

З метою, покращити технології обробки зображень та автоматизації її аналізу, було проведено дослідження внаслідок якого, було проаналізовано, основні методи обробки графічної інформації, використовуючи сучасні технології машинного навчання.

Результат дослідження є створений програмний засіб, що значно покращує можливості обробки зображень, та їх зберігання. Дане рішення є доволі актуальним в епоху інтернету. Різноманітні компанії намагаються створити ефективне рішення, для аналізу графічної інформації. Різноманітні графічні редактори такі як Photshop, вже впроваджують штучний інтелект, а компанії , що виробляють смартфони, активно впроваджують системи, що аналізують людське обличчя.

Програмне рішення дає можливість ефективно аналізувати зображення, та зберігати дані про нього в двійковому форматі, що облегшує взаємодію з базою даних. В результаті маємо доволі швидку та ефективну систему, як споживає не багато ресурсів.

Було проведено тестування додатку відповідно до заздалегідь визначеного тестувального плану.

5. ЕКОНОМІЧНА ЧАСТИНА

5.1. Проведення комерційного та технологічного аудиту науково-технологічної розробки

Комерційного та технологічного аудиту проводиться для оцінки науково-технічного рівня та комерційного потенціалу розробки. Для цього було залучено 3-х незалежних експертів: Гончар С.І., студент групи 1ПІ-22М, займає посаду провідного інженера програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ» (експерт 1), Нестерук В.О., студент групи 1ПІ-22М, займає посаду старшого інженера програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ» (експерт 2), Решетнік О.О., займає посаду провідного інженера-програміста у компанії ТОВ «ЕПАМ ДІДЖИТАЛ» (експерт 3).

Оцінювання здійснюється за 12-ма критеріями: технічна здійсненність концепції, ринкові переваги (наявність аналогів), ринкові переваги (ціна продукту), ринкові переваги (технічні властивості), ринкові переваги (експлуатаційні витрати), ринкові перспективи (розмір ринку), ринкові перспективи (конкуренція), практична здійсненність (наявність фахівців), практична здійсненність (наявність фінансів), практична здійсненність (потреба нових матеріалів), практична здійсненність (термін реалізації), практична здійсненність (розробка документів). Кожен критерій оцінюється з використанням 5-ти бальної шкали. Результати наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу експертами отриманої розробки

Критерії	Експерти		
	1	2	3
	Бали		
1. Технічна здійсненність концепції	2	2	2
2. Ринкові переваги (наявність аналогів)	2	2	2
3. Ринкові переваги (ціна продукту)	3	3	4
4. Ринкові переваги (технічні властивості)	3	2	3

5. Ринкові переваги (експлуатаційні витрати)	4	3	3
6. Ринкові перспективи (розмір ринку)	3	2	2
7. Ринкові перспективи (конкуренція)	2	3	2
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	$СБ_1 = 38$	$СБ_2 = 36$	$СБ_3 = 37$
Середньоарифметичне сума балів $СБ_с$	$СБ_с = \frac{СБ_1+СБ_2+СБ_3}{3} = 37$		

На основі результатів таблиці 5.1 можна зробити висновок, що розробка має вище середнього рівень комерційного потенціалу і науково-технічного рівня. Такий рівень потенціалу було досягнуто за рахунок розширення навчальних можливостей науково-технічної розробки, а саме завдяки новим методам обробки складної та комплексної інформації, а також використання нейронних мереж для пришвидшення розпізнавання зображень.

Отже, рівень веще середнього комерційного потенціалу та науково-технічної розробки в першу чергу пояснюється зростання ефективності вивчення шаблонів проєктування з рахунок комбінованого методу навчання, що поєднує практичні задачі та візуалізацію. По друге, додаток показує значно вищу якість в порівнянні з аналогами за рахунок удосконалених методів розпізнавання зображень. Як результат відбувається зменшення часу, необхідного для пошуку зображення.

5.2. Прогнозування витрат на виконання науково-дослідної роботи

Витрати на проведення науково-дослідної роботи розраховуються за різними складовими, такими як оплата праці, соціальні відрахування, закупівля матеріалів, витрати на паливо та енергію для наукових виробничих потреб, витрати на службові відрядження, придбання спеціального обладнання для наукових робіт, програмне забезпечення для наукових досліджень, витрати на послуги, надані сторонніми підприємствами, установами та організаціями, а також загальні накладні витрати та інші складові. Давайте розрахуємо витрати для кожної з цих категорій.

5.2.1 Витрати на оплату праці

До витрати на оплату праці входять витрати на виплату як основної, так і додаткової заробітної плати працівникам, що відносяться до різних рівнів управління, таких як керівники відділів, лабораторій, секторів і груп, а також наукові, інженерно-технічні працівники та інші спеціалісти, які безпосередньо зайняті виконанням конкретних дослідницьких завдань.

Основна заробітна плата розробників розраховується за формулою (5.1):

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість розробників;

$М_{пі}$– місячний посадовий оклад конкретного розробника, грн;

t_i– кількість днів роботи розробника, дн.;

T_p– кількість робочих днів у місяці (вважаємо, що $T_p = 21$ день).

Обчислені результати за цією формулою занесемо в таблицю 5.2

Таблиця 5.2 – Витрати на заробітну плату працівників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Керівник наукової роботи	11500	522,72	25	13068
Інженер програміст	8000	363,63	25	9090,9
Всього				22158,9

Додаткові витрати на оплату праці розраховуються за формулою (5.2):

$$З_{дод} = З_о \cdot \frac{Н_{дод}}{100\%} = 22158{,}9 \cdot 0{,}10 = 2215{,}89 \text{ (грн)}, \qquad (5.2)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати, що становить 10%.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою (5.3):

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} = (22158{,}9 + 2215{,}89) \cdot \frac{22\%}{100\%} = 5362{,}45 \text{ (грн)}, \qquad (5.3)$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

5.3.3 Спецустаткування для наукових робіт

Балансову вартість спецустаткування розраховують за формулою (5.4):

$$В_{спец} = \sum_{i=0}^{k} Ц_і \cdot С_{пр.і} \cdot К_і, \qquad (5.4)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$С_{пр.і}$ – кількість одиниць устаткування відповідного найменування, шт.;

$К_i$ – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування ($К_i$ = 1,1 … 1,12)

Візьмемо коефіцієнт додаткових витрат рівним 1,12. Проведені розрахунки зведено до таблиці 5.3.

Таблиця 5.3 – Витрати на устаткування для організації робочого процесу

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ноутбук	2	40000	80000
Мишка	2	3000	6000
Зарядний пристрій	2	2999	5998
Зовнішній HDD	1	2000	2000
Всього			93998

Отже, $В_{спец}$ з врахуванням додаткових витрат отримано 105277,76 (грн).

5.2.4 Програмне забезпечення для наукових робіт

Балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою (5.5):

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{прг.і} \cdot К_i, \quad (5.5)$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{прг.і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (Кі = 1,10…1,12).

Візьмемо коефіцієнт витрат на інсталяцію рівним 1,12. Проведені розрахунки зведено до таблиці 5.4.

Таблиця 5.4 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
PyCharm	2	895,9	1791,8
Office 365	2	449,75	899,5
GitHub	2	143,92	287,84
Всього			2979,14

Розрахуємо витрати на балансову вартості програмного забезпечення (5.6):

$$В_{прг} = 2979{,}14 \cdot 1{,}12 = 3336{,}64 \text{ (грн)} \quad (5.6)$$

5.2.5 Амортизація обладнання, програмних засобів та приміщень

Амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо розраховані з використанням прямолінійного методу амортизації за форулою (5.7):

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \quad (5.7)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$– термін використання обладнання та програмних засобів, 1 місяць;

$Т_в$ – строк корисного використання обладнання, 5 років

Проведемо розрахунки та знайдемо амортизацію в таблиці 5.5:

Таблиця 5.5 – Витрати на устаткування для організації робочого процесу

Найменування устаткування	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук	80000	5	1	1333,33
Мишка	6000	5	1	100
Зарядний пристрій	5998	5	1	99,97
Зовнішній HDD	2000	5	1	33,33
Всього				1566,33

5.2.6 Інші витрати

До інших витрати належать ті, що не відображені в попередніх статтях витрат і розраховуються як 50% від суми основної заробітної плати працівників за формулою (5.9):

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} = 22158{,}9 \cdot 0{,}5 = 11079{,}45 \text{ (грн)}, \quad (5.9)$$

5.2.7 Накладні (загальновиробничі) витрати

Витрати на проведення науково-дослідної роботи розраховуються за формулою (5.10) тобто як сума всіх попередніх статей витрат:

$$В_{заг} = З_о + З_{дод} + З_н + В_{спец} + В_{прг} + А_{обл} + І_в \quad (5.10)$$

$$В_{заг} = 22158{,}9 + 2215{,}89 + 5362{,}45 + 105277{,}76 + 3336{,}64 + 1566{,}33 + 11079{,}45 = 150997{,}42 \text{ (грн)} \quad (5.11)$$

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розрахуємо за формулою (5.12):

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.12)$$

де η – коефіцієнт, який характеризує етап(стадію) виконання наково-дослідної роботи.

$$ЗВ = \frac{150997{,}42}{0{,}9} = 167774{,}91 \text{ (грн)} \quad (5.13)$$

5.3. Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Зростання прибутку у потенційного інвестора протягом декількох років розраховується за формулою (5.14):

$$\Delta П_i = (\Delta Ц_о \cdot N + Ц_о \cdot \Delta N) \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \quad (5.14)$$

де $\Delta Ц_о$– зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_о = Ц_б \pm \Delta Ц_о$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). ρ = 0,2;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році становить 18%.

Отримані результати можна використати для покращення існуючих продуктів. Введені зміни приведуть до підняття якості певного продукту в порівнянні з конкурентами, що дозволить підвищити вартість 700 грн. Прогнозується збільшення кількості користувачів протягом 1 року на 1000, 2 рік – 700, 3 рік – 300, це при умові що в цей час не будуть внесені покращення та оптимізації продукту, що вимагатимуть додаткових витрат на підтримку.

Так як ринок доволі великий, то припустимо що в покращеного продукту була база користувачів у 10000, а її ціна 550 грн за 12 місяців.

Розрахуємо отриманий прибуток протягом 3 років відносно базового:

$$\Delta П_1 = (700 \cdot 10000 + (550 + 700) \cdot 1000) \cdot 0{,}8333 \cdot 0{,}2 \cdot \left(1 - \frac{18}{100}\right) = 1127454{,}9 \text{ (грн)}$$

$$\Delta П_2 = (700 \cdot 10000 + (550 + 700) \cdot 1700) \cdot 0{,}8333 \cdot 0{,}2 \cdot \left(1 - \frac{18}{100}\right) = 1247033{,}45 \text{ (грн)}$$

$$\Delta П_3 = (700 \cdot 10000 + (550 + 700) \cdot 2000) \cdot 0{,}8333 \cdot 0{,}2 \cdot \left(1 - \frac{18}{100}\right)$$
$$= 1298281{,}4 \text{ (грн)}$$

Далі розрахуємо приведену вартість збільшення чистих прибутків, що їх може отримати потенційний інвестор від впровадження розробки за формулою (5.15):

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.15)$$

де Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{1127454{,}9}{(1+0.15)^1} + \frac{1247033{,}45}{(1+0.15)^2} + \frac{1298281{,}4}{(1+0.15)^3} =$$
$$= 980395{,}57 + 1084376{,}91 + 1128940{,}35 = 3193712{,}83 \text{ (грн)}$$

Розрахуємо величину початкових інвестицій потенційного інвестору на впровадження оптимізацій та покращень отриманих в науково-технічній розробці за формулою (5.16):

$$PV = k_{\text{інв}} \cdot ЗВ, \quad (5.16)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. $k_{\text{інв}}$ = 3.

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

Тоді величина початкових інвестицій потенційного інвестору буде:

$$PV = 3 \cdot 167774{,}91 = 503324{,}73 \text{ (грн)}$$

Тоді абсолютний економічний ефект $\text{Е}_{\text{абс}}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки обраховується за формулою (5.16):

$$\text{Е}_{\text{абс}} = \text{ПП} - PV, \quad (5.16)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн

Підставимо значення та розрахуємо абсолютний економічний ефект:

$$\text{Е}_{\text{абс}} = 3193712{,}83 - 503324{,}73 = 2690388{,}1 \text{ (грн)}$$

Позитивний показник свідчить про потенційну зацікавленість інвесторів у впровадженні розробки, проте потребує додаткового дослідження. Для остаточного рішення потрібно розрахувати внутрішню економічну дохідність за формулою (5.17):

$$\text{Е}_{\text{в}} = \sqrt[\text{Т}_{\text{ж}}]{1 + \frac{\text{Е}_{\text{абс}}}{PV}} - 1, \quad (5.17)$$

де $\text{Е}_{\text{абс}}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$\text{Т}_{\text{ж}}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки. Отже, маємо таку внутрішню економічну дохідність:

$$\text{Е}_{\text{в}} = \sqrt[3]{1 + \frac{2084164{,}86}{1109547{,}97}} - 1 = 0{,}85$$

Далі визначають бар'єрну ставку дисконтування τ_{min}, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть. Мінімальна внутрішня економічна дохідність вкладених інвестицій τ_{min} визначається за формулою (5.18):

$$\tau_{min} = d + f, \tag{5.18}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках, у 2023 році становить 0,16;

f – показник, що характеризує ризикованість вкладення інвестицій, у 2023 році становить 0,4. Отже, маємо такі результати:

$$\tau_{min} = 0{,}16 + 0{,}4 = 0{,}56$$

Оскільки величина $\mathrm{E_в} > \tau_{min}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки. Далі розрахуємо період окупності інвестицій за формулою (5.19):

$$\mathrm{T_{ок}} = \frac{1}{\mathrm{E_в}}, \tag{5.19}$$

де $\mathrm{E_в}$ – внутрішня економічна дохідність вкладених інвестицій. Тоді період окупності буде наступний:

$$\mathrm{T_{ок}} = \frac{1}{0{,}85} = 1{,}18 \text{ року}$$

Отже маємо період окупності менше трьох років, що свідчить про комерційну привабливість науково-технічної розробки.

5.4. Висновки

Таким чином, в даному розділі проаналізовано економічну складову науково-дослідної роботи за темою «Програмна система пошуку зображень з використанням нейронних мереж». Проведено науковий аудит, за результатами досліджень створюване ПЗ за ступенем наукової новизни має характеристику «відносно нове», означає що в даній роботі присутні елементи новизни в методах пошуку зображень.

Проведено комерційний та технологічний аудит НДР. В результаті дослідження розробки трьома незалежними експертами сформовано таблицю оцінювання (таблиця 5.1) комерційного потенціалу. За результатами дослідження з’ясувалося, що розробка має вище середнього рівень комерційного потенціалу і науково-технічного рівня. Такий рівень потенціалу було досягнуто за рахунок розширення навчальних можливостей науково-технічної розробки, а саме завдяки новим методам обробки складної та комплексної інформації, а також використання нейронних мереж для пришвидшення розпізнавання зображень.

Здійснено прогнозування витрат на виконання НДР. До цієї категорії враховано наступні показники: витрати на оплату праці (основну та додаткову), відрахування на соціальні заходи, спец-устаткування та програмне забезпечення для наукових робіт, амортизація обладнання та інші витрати. На основі вище згаданих показників розраховано загальні витрати на завершення НДР, що становить 167774,91 грн.

Далі розраховано показник ефективності вкладених інвестицій. Величина цього показника складає 85%, що більше за 56%, тобто мінімальний поріг, що визначає потенційну можливість інвестування. Також розраховано період окупності, який дорівнює 1,18 року. Такий термін свідчить про комерційну привабливість проекту.

ВИСНОВКИ

У магістерській роботі розроблено програмну систему для пошуку зображень, яка використовує нейронні мережі для досягнення кращої точності та ефективності у порівнянні з традиційними методами обробки зображень.

Було розглянуто такі методи класифікації графічних зображень як розпізнавання об’єктів на зображенні за допомогою алгоритмів, що використовують підхід пошуку ключових точок та методів що використовують нейронні мережі. Проведено порівняльну характеристику роботи алгоритмів SIFT, SURF та ORB. У результаті дослідження було зроблено припущення, що методи машинного навчання надають більше можливостей у класифікації та визначенні об’єктів на графічних зображеннях ніж класичні алгоритми роботи з зображеннями що використовуються для комп’ютерного бачення. Ключовим аспектом у покращенні процесу обробки зображень є тренування нейронної мережі з оптимальними параметрами, та якісного набору навчальних даних. Проведений аналіз показав, що система з нейронною мережею забезпечує точніше виявлення об’єктів порівняно з іншими методами. Вона виявила об’єкти на зображеннях з високим рівнем впевненості та забезпечила стабільну продуктивність навіть на зображеннях в складних умовах.

Досліджено основні принципи та структуру даних у нейронних мережах за допомогою тензорів. Також було проведено аналіз та дослідження засобів тренування нейронних мереж, а саме використання CPU, GPU та TPU у процесі тренування. Було визначено, що для більш ефективного тренування нейронної мережі яка за якостями буде готова для використання у реальних проектах необхідно використовувати великі набори тренувальних даних у системах обладнаних потужнимb GPU.

У рамках розробки програмного продукту було досліджено ряд архітектурних шаблонів проектування, таких як мікро-сервісна архітектура, моноліт, та тришарова архітектура програмного додатку. Задля більшої ефективності розробки при відносно мінімальних витратах було прийнято рішення виконувати розробку програмного додатку з використанням тришарової архітектури, що передбачає логічне розділення

системи на три шари, а саме: шар користувацького інтерфейсу, шар бізнес-логіки та шар зображення даних.

Було модифіковано методи виявлення об’єктів на графічному зображенні шляхом оптимізації вхідних параметрів нейронної мережі за допомогою Баєсової оптимізації. Також було оптимізовано функцію втрати за допомогою градієнтних спусків. Це дозволило пришвидшити час навчання моделі та підвищити ефективність пошуку об’єктів на зображеннях низької якості.

Було запропоновано комбінування двох типів нейронних мереж, це Object Detection і Image classification. Даний метод показав, що комбінація дозволяє розподілити роботу між акторами, що спеціалізуються на окремих задачах, а саме виявлення а за тим класифікація. Варто зазначити, що даний метод також має недолік у швидкодії так як даний процес має послідовний характер.

Результатом проведених досліджень стала розробка програмної системи, що використовує архітектуру RetinaNet для об’єктного виявлення на зображеннях. Система відображає високу точність при виявленні об’єктів різних класів на тестових даних. Для реалізації даного програмного продукту було використано мову програмування Python, бібліотеку машинного навчання PyTorch, бібліотека для розробки веб додатків Flask та реляційна база даних SQLite.

Задля перевірки якості програмного продукту було складено базовий тест план та проведено процедуру мануального тестування відповідно до узгодженого плану.

З врахуванням результатів дослідження, існують можливості для подальшого вдосконалення системи. Подальші роботи можуть включати удосконалення архітектури нейронної мережі, оптимізацію алгоритмів для поліпшення швидкодії та розширення функціональності системи. Враховуючи сумісність нових версій мереж до попередніх версій, оновлення системи відбувається шляхом фізичної підміни моделі, що дозволяє удосконалювати якість розпізнавання, що не обмежується обчислювальною системою а лише якістю моделі.

У результаті дослідження підтверджено ефективність застосування нейронних мереж для пошуку зображень. Розроблена програмна система є кроком вперед у розвитку інструментів для обробки зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бобко О.Л., Рейда О.М. Використання CPU, GPU та TPU для тренування нейронних мереж // Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії – 2023 : матеріали Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) Вінниця, 20.03.23 р. - 23.03.23 р. [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023

2. Бобко О.Л., Рейда О.М. Використання тензорів у машинному навчанні // Сучасні комп’ютерні системи та мережі в управлінні – 2023 : матеріали VI Всеукраїнської науково-практичної інтернет-конференції молодих вчених та студентів «Сучасні інформаційні системи та технології» Херсон, 30.11.23 р. [Електронний ресурс] – Режим доступу: http://kntu.net.ua/ukr/Pro-universitet2/Novini-universitetu/VI-Vseukrayins-ka-naukovo-praktichna-konferenciya-studentiv-aspirantiv-ta-molodih-vchenih-z-avtomatichnogo-upravlinnya

3. A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. International Conference on Computing, Mathematics and Engineering Technologies, 2018. 10 p

4. A Comparison of SIFT, SURF and ORB on OpenCV: [Електронний ресурс] – Режим доступу: https://mikhail-kennerley.medium.com/a-comparison-of-sift-surf-and-orb-on-opencv-59119b9ec3d0

5. Karami, Ebrahim et al. “Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images.” ArXiv abs/1710.02726 (2017): n. pag.

6. Nazia Aslam. Data Representation in Neural Networks - Tensor. - [Електронний ресурс] – Режим доступу: https://www.analyticsvidhya.com/blog/2022/07/data-representation-in-neural-networks-tensor/

7. Y. Li Research and Application of Deep Learning in Image Recognition // 2nd International Conference on Power, Electronics and Computer Applications (ICPECA) – Shenyang, China, 2022

8. Lewis Tunstall, Leandro von Werra, Thomas Wolf. Natural Language Processing with Transformersx, 2022. – 406 p

9. Deep Learning with PyTorch: [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/deep-learning-with-pytorch-an-introduction/

10. Jean Ponce, 2021. Computer Vision. [Електронний ресурс] – Режим доступу: https://www.springer.com/journal/11263

11. Adrian Kaehler and Gary Bradski. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library 1st Edition, 2016. – 1022

12. Mohamed Elgendy. Deep Learning for Vision Systems, 2020. – 480 p

13. RetinaNet Explained and Demystified: [Електронний ресурс] – Режим доступу: https://blog.zenggyu.com/posts/en/2018-12-05-retinanet-explained-and-demystified/index.html

14. Sam Newman, Building Microservices, 2nd Edition, 2021. – 586 p

15. Eli Stevens, Luca Antiga, and Thomas Viehmann. Deep Learning with PyTorch, 2020. – 520 p

16. TensorFlow: [Електронний ресурс] – Режим доступу: https://www.tensorflow.org

17. Pytorch: [Електронний ресурс] – Режим доступу: https://pytorch.org/

18. Flask [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/3.0.x/

19. SQLite [Електронний ресурс] – Режим доступу: https://www.sqlite.org/doclist.html

20. ImageAI: [Електронний ресурс] – Режим доступу: https://imageai.readthedocs.io/en/latest/index.html

21. Step-by-Step Guide to Bayesian Optimization: A Python-based Approach: [Електронний ресурс] – Режим доступу: https://medium.com/@okanyenigun/step-by-step-guide-to-bayesian-optimization-a-python-based-approach-3558985c6818

22. Bayesian Optimization Concept Explained in Layman Terms: : [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f

23. Understanding Loss Functions to Maximize Machine Learning Model Performance (Updated 2023): [Електронний ресурс] – Режим доступу: https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

24. Contrastive and Triplet losses. [Електронний ресурс] – Режим доступу: https://gombru.github.io/2019/04/03/ranking_loss

25. В. П. Марценюк, докт. техн. наук, проф., Н. В. Мілян ОГЛЯД МЕТОДІВ ОПТИМІЗАЦІЇ В МАШИННОМУ НАВЧАННІ: ГРАДІЄНТНИЙ СПУСК ТА СТОХАСТИЧНИЙ ГРАДІЄНТНИЙ СПУСК // Матеріали IX Міжнародної науково-технічної конференції молодих учених та студентів. Актуальні задачі сучасних технологій – Тернопіль 25-26 листопада 2020

26. Gonzalez R. Digital image processing fourth edition / R. Gonzalez, R. Woods - M .: Technosphere, 2018. – 1022 c

27. Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects, 2019. – 308 p

28. Python [Електронний ресурс] – Режим доступу: https://docs.python.org/3/

29. CPU vs GPU vs TPU: When to Use For Your Machine Learning Models [Електронний ресурс] – Режим доступу: https://www.linkedin.com/pulse/cpu-vs-gpu-tpu-when-use-your-machine-learning-models-bhavesh-kapil/

30. Guide To Test Approach: Different Types With Examples[Електронний ресурс] – Режим доступу: https://www.lambdatest.com/learning-hub/test-approach

ДОДАТКИ

ДОДАТОК А. ТЕХНІЧНЕ ЗАВДАННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., професор Романюк О.Н.
"19" вересня 2022 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Методи оптимізації траєкторії руху транспортних засобів у програмній системі позиціювання в режимі реального часу» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доцент. Рейда О.М.
"19" 09 2023 р.

Виконав:
_____ студент гр.1ПІ-22М Бобко О.Л.
"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Програмна система пошуку зображень з використанням нейронних мереж».

Галузь застосування системи пошуку, класифікації та захоплення об’єктів на зображенні.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою даної роботи є покращення роботи нейронної мережі для захоплення та класифікації об’єктів на зображенні.

Призначення роботи – аналіз та розробка програмної системи на базі нейронної мережі для класифікації зображень.

3 Вихідні дані для проведення НДР

Покращена модель нейронної мережі; середовища розробки – Eclipse IDE; мова розробки - Python; операційна система – Windows 10; вихідні дані – Програмна система пошуку зображень з використанням нейронних мереж.

4. Технічні вимоги

Доступ до сучасного комп’ютера або мобільного пристрою; стабільне інтернет-з’єднання; останні версії веб-браузерів (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge); включений JavaScript у налаштуваннях браузера; актуальні версії операційних систем; знання основ принципів роботи нейронних мереж та обробки зображень.

5. Конструктивні вимоги.

Архітектура системи повинна відповідати галузевим стандартам без використання застарілих технологій.

6. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз методів пошуку та розпізнавання зображень	09.10.2023
2	Проектування системи	29.10.2023
3	Програмна розробка системи	14.11.2023
4	Тестування системи	24.11.2023
5	Економічна частина	05.12.2023

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б. ПРОТОКОЛ ПЕРЕВІРКИ РОБОТИ

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Програмна система пошуку зображень з використанням нейронних мереж

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 22м

Науковий керівник: ктн., доц. кафедри ПЗ Рейда О.М.

Unicheck	
Оригінальність	**95.0**
Схожість	5.0

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Бобко О.Л.

Керівник роботи _____ Рейда О.М.

ДОДАТОК В. ЛІСТИНГ ПРОГРАМИ

File: main.py

```
from imageai.Detection import ObjectDetection
import os
from storage import insert_blob

def classification(images_name):
    images_name1 = images_name

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath("coco_resnet_50_map_0_335_state_dict.pt")
    detector.loadModel()
    detections = detector.detectObjectsFromImage("static/images/" + images_name1,
"static/images/" + images_name1 + "new.jpeg")

    klass = list()

    for eachObject in detections:
        insert_blob(eachObject["name"], "static\\images\\" + images_name1)
        klass.append(eachObject["name"] + " : " + str(eachObject["percentage_probability"])
+ ", " + str(eachObject["box_points"]));
        print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ",
eachObject["box_points"] )

    return klass
```

File: storage.py

```
import sqlite3
import os
import random


def convert_to_binary_data(filename):
    with open(filename, ‘rb’) as file:
        blob_data = file.read()
    return blob_data

def insert_blob(image, photo):
    try:
        sqlite_connection = sqlite3.connect(‘image_kod’)
        cursor = sqlite_connection.cursor()
        print("Connected к SQLite")

        sqlite_insert_blob_query = """INSERT INTO Images
                                (image, kod) VALUES (?, ?)"""

        emp_photo = convert_to_binary_data(photo)
        data_tuple = (image, emp_photo)
        cursor.execute(sqlite_insert_blob_query, data_tuple)
        sqlite_connection.commit()
        cursor.close()

    except sqlite3.Error as error:
        print("SQLite error", error)
    finally:
```

```
        if sqlite_connection:
            sqlite_connection.close()
            print("Connection SQLite closed")

filename = "images"
def write_to_file(data, filename):
    with open(filename, ‘wb’) as file:
        file.write(data)
    print("blob saved: ", filename, "\n")

def read_blob_data(img_id):
    im = list()
    try:
        sqlite_connection = sqlite3.connect(‘image_kod’)
        cursor = sqlite_connection.cursor()
        print("Connected to SQLite")

        sql_fetch_blob_query = """SELECT * from Images where image = ?"""
        cursor.execute(sql_fetch_blob_query, (img_id,))
        record = cursor.fetchall()
        for row in record:
            print("Id = ", row[0], "Name = ", row[1])
            name  = row[1]
            photo = row[2]

            print("save on disk \n")
            photo_path = "static/images/" + name + str(random.randint(1,10000)) + ".jpg"
            write_to_file(photo, photo_path)
            im.append(photo_path)
        cursor.close()
```

```
except sqlite3.Error as error:
    print("can not connect to SQLite", error)
finally:
    if sqlite_connection:
        sqlite_connection.close()
        print("Connection closed")
return im
```

File: web.py

```
from flask import Flask, render_template, url_for, redirect, request
import main
import storage


app = Flask(__name__)


@app.route(‘/local’)
def lm():
    return render_template(‘index.html’)

@app.route(‘/<name>‘, methods=[‘POST’])
def main_i(image):
    print(image)
    f = request.files[‘image’]
    main.klasification(image)
    return render_template(‘index.html’)

@app.route(‘/main’, methods=[‘POST’])
def image_main():
        if request.method == ‘POST’:
            f = request.files[‘image’]
            f.save("static/images/" + f.filename)
            klass = main.classification(f.filename)
            return render_template(‘index.html’, list=klass, image="/images/" + f.filename +
"new.jpeg")
```

```
@app.route(‘/main/<string:img_klass>‘)
def image_klass(img_klass):
    ans = storage.read_blob_data(img_klass)
    print(ans)
    new_ans = list()
    for i in ans:
        ans = i.split(‘/’)
        new_ans.append(ans[-1])
    return render_template(‘answer.html’, list=new_ans)

if __name__ == "__main__":
    app.run(debug=True)
```

File: index.html

```
{% extends ‘base.html’%}


{% block title %}
Вибір зображення
{% endblock %}


{% block body %}
<form action = "http://localhost:5000/main" method = "post" enctype="multipart/form-data">
   <p color="white">Select image:</p>
   <p><input type = "file" name = "image" /></p>
   <p><input type = "submit" value = "submit" /></p>
</form>
{% for p in list %}
   {{p}}</br>
{% endfor %}

<img src="{{url_for(‘static’, filename=image)}}" style="width: 600px; height: auto;"/>

{% endblock %}
```

File: base.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href = "{{ url_for(‘static’, filename=‘css/main.css’)}}">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    {% block body %}{% endblock %}
</body>
</html>
```

File: answer.html

```
{% extends ‘base.html’%}


{% block title %}
Selected class
{% endblock %}

<style>
   img {
   width: 300px;
   height: auto;
}
</style>

{% block body %}
<table align="center">
      {% for p in list %}
<tr style="width: 300px;">
   <td>
      <img src="{{url_for(‘static’, filename=‘/images/’ + p)}}" style="width: 300px;
height: auto;"/>
   </td>
</tr>
      {% endfor %}
</table>

{% endblock %}
```

File: train_retinanet.py

```
import torch
import torchvision
import torch.nn as nn
from torchvision.models.detection import RetinaNet
from torchvision.datasets import CocoDetection
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torchvision.models.detection.backbone_utils import resnet_fpn_backbone

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

coco_train = CocoDetection(root='C:/tools/coco/coco2017/train2017',
annFile='C:/tools/coco/coco2017/annotations/instances_train2017.json',
transform=transform)

text_file = open("C:/tools/coco/coco2017/coco_classes.txt", "r")
classes = text_file.readlines()

backbone = resnet_fpn_backbone('resnet50', pretrained=True)
model = RetinaNet(backbone=backbone, num_classes=len(classes), out_channels=64,
pretrained=True)
```

```
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.9,
weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)



def collate_fn(batch):
    return tuple(zip(*batch))

train_loader = DataLoader(coco_train, batch_size=4, shuffle=True, collate_fn=collate_fn)

num_epochs = 3
for epoch in range(num_epochs):
    model.train()
    for images, targets in train_loader:
        images = list(image.to(device) for image in images)

        targets_on_device = []
        for target_dict in targets:

            new_target = {}

            for key in target_dict[0].keys():
                new_target[key] = target_dict[0][key]

            targets_on_device.append(new_target)

        optimizer.zero_grad()
```

```
        loss_dict = model(images, targets)

        losses = sum(loss for loss in loss_dict.values())
        losses.backward()
        optimizer.step()

        print(f"Epoch [{epoch+1}/{num_epochs}] Loss: {losses.item()}")

torch.save(model.state_dict(), 'retinanet_coco_model.pth')
```

File: b_optimization.py

```
import torch
import torch.nn as nn
from bayes_opt import BayesianOptimization



class SimpleNN(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

def train_function(lr, hidden_size):
    input_size = 13  # Припустимо розмір вхідних даних
```

```
    output_size = 4  # Припустимо розмір виходу
    model = SimpleNN(input_size, int(hidden_size), output_size)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    # Здійснюємо тренування моделі та повертаємо значення, яке хочемо максимізувати
    # Наприклад, точність або іншу метрику
    return torch.rand(1).item()



# Визначаємо простір параметрів, які хочемо оптимізувати
pbounds = {'lr': (0.001, 0.01), 'hidden_size': (16, 64)}

optimizer = BayesianOptimization(
    f=train_function,
    pbounds=pbounds,
    random_state=44,
)

optimizer.maximize(
    init_points=4,
    n_iter=11,
```

```
)


print(optimizer.max)
```

File requirements.txt

attrs==23.1.0
blinker==1.6.3
certifi==2023.7.22
charset-normalizer==3.3.0
click==8.1.7
contourpy==1.1.1
cycler==0.12.1
Cython==3.0.3
filelock==3.12.4
Flask==3.0.0
fonttools==4.43.1
fsspec==2023.9.2
idna==3.4
imageai==3.0.3
iniconfig==2.0.0
itsdangerous==2.1.2
Jinja2==3.1.2
kiwisolver==1.4.5
MarkupSafe==2.1.3
matplotlib==3.8.0
mock==4.0.3
mpmath==1.3.0
networkx==3.1
numpy==1.26.1
opencv-python==4.8.1.78
packaging==23.2
Pillow==10.1.0
pluggy==1.3.0

py==1.11.0

pyparsing==3.1.1

pytest==7.1.3

python-dateutil==2.8.2

requests==2.31.0

scipy==1.11.3

six==1.16.0

sympy==1.12

tomli==2.0.1

torch==2.1.0

torchvision==0.16.0

tqdm==4.64.1

typing_extensions==4.8.0

urllib3==2.0.6

Werkzeug==3.0.0

ДОДАТОК Г. ІЛЮСТРАТИВНА ЧАСТИНА

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмна система пошуку зображень з використанням нейронних мереж»

Виконав студент 2-го курсу, групи 1ПІ-22м,
Бобко Олексій Леонідович

Рисунок Г.1 – Слайд презентації №1

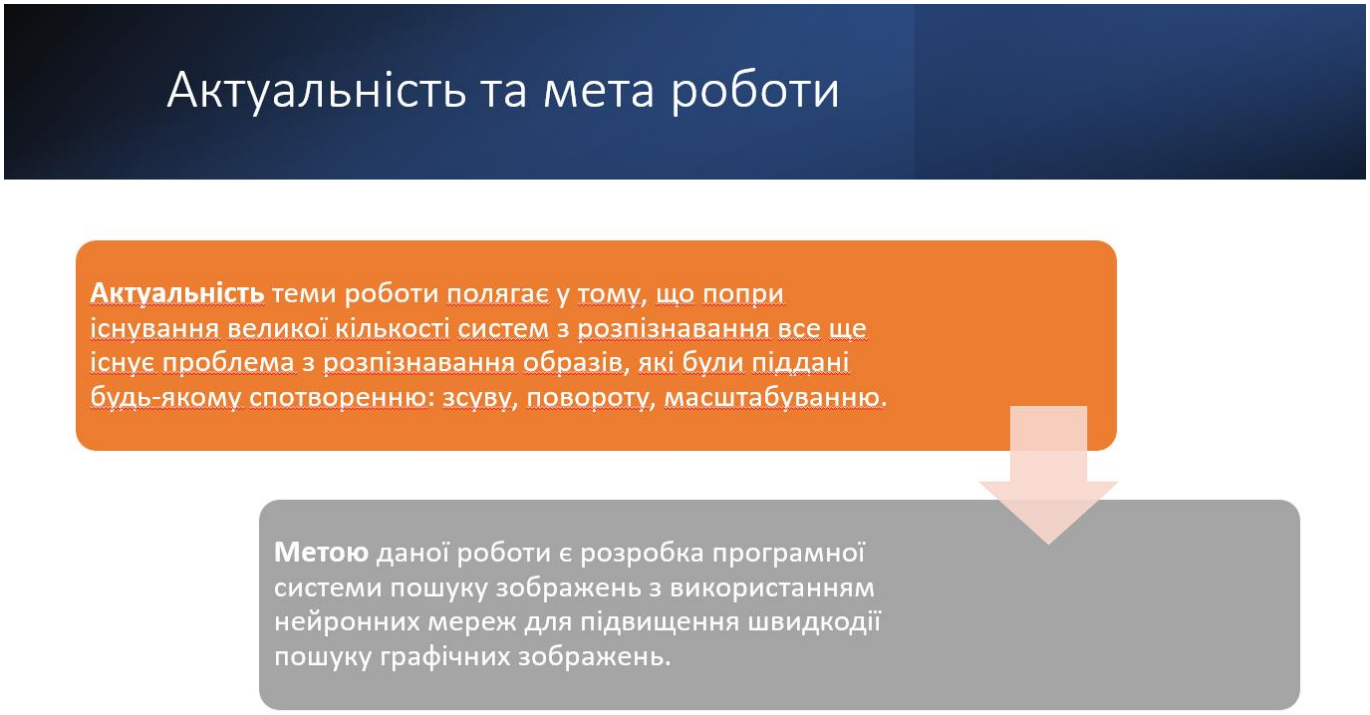

Рисунок Г.2 – Слайд презентації №2

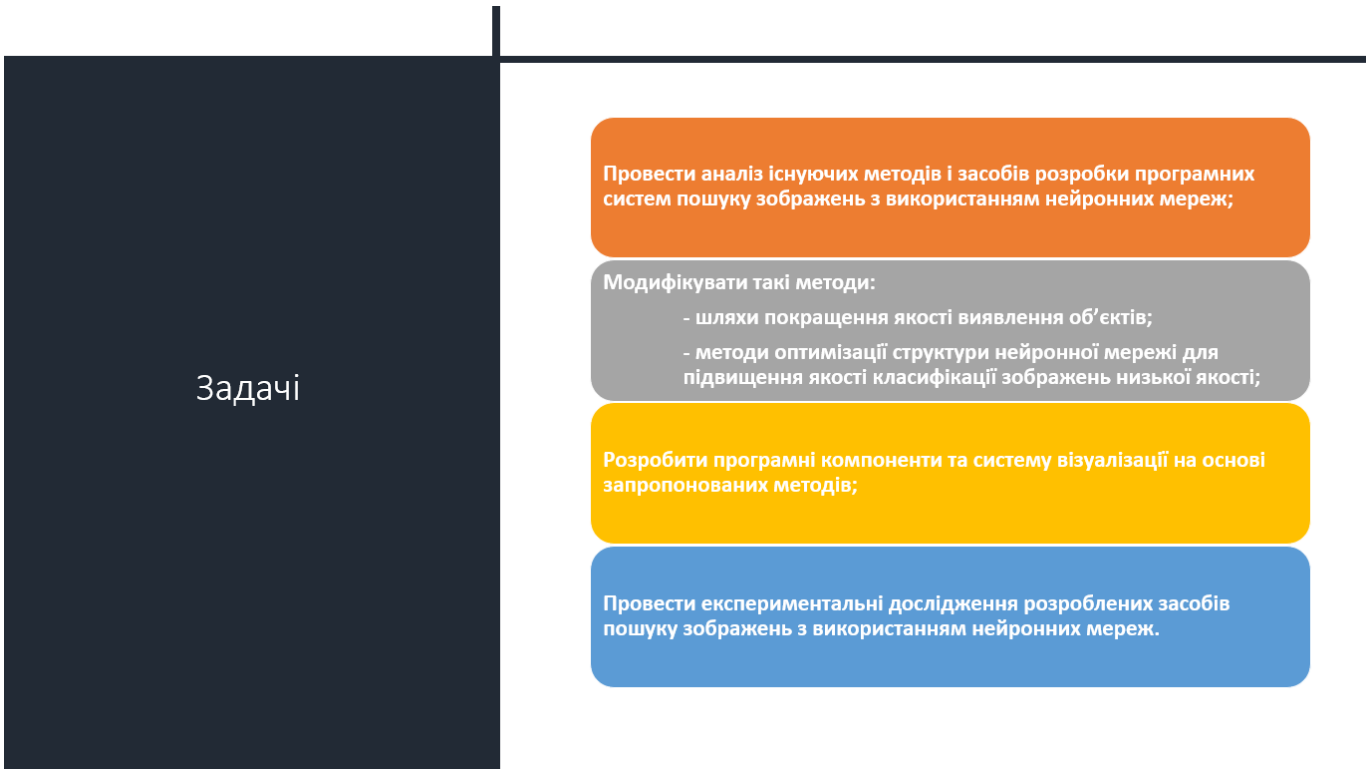

Рисунок Г.3 – Слайд презентації №3

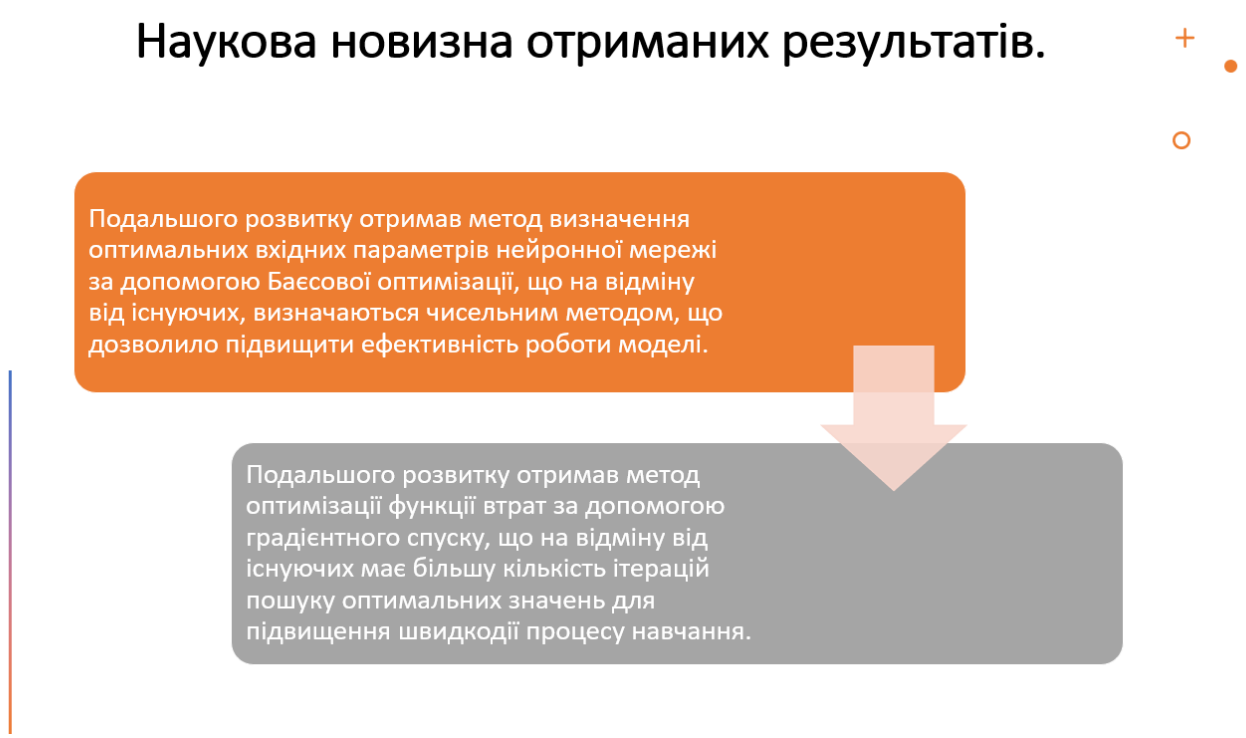

Рисунок Г.4 – Слайд презентації №4

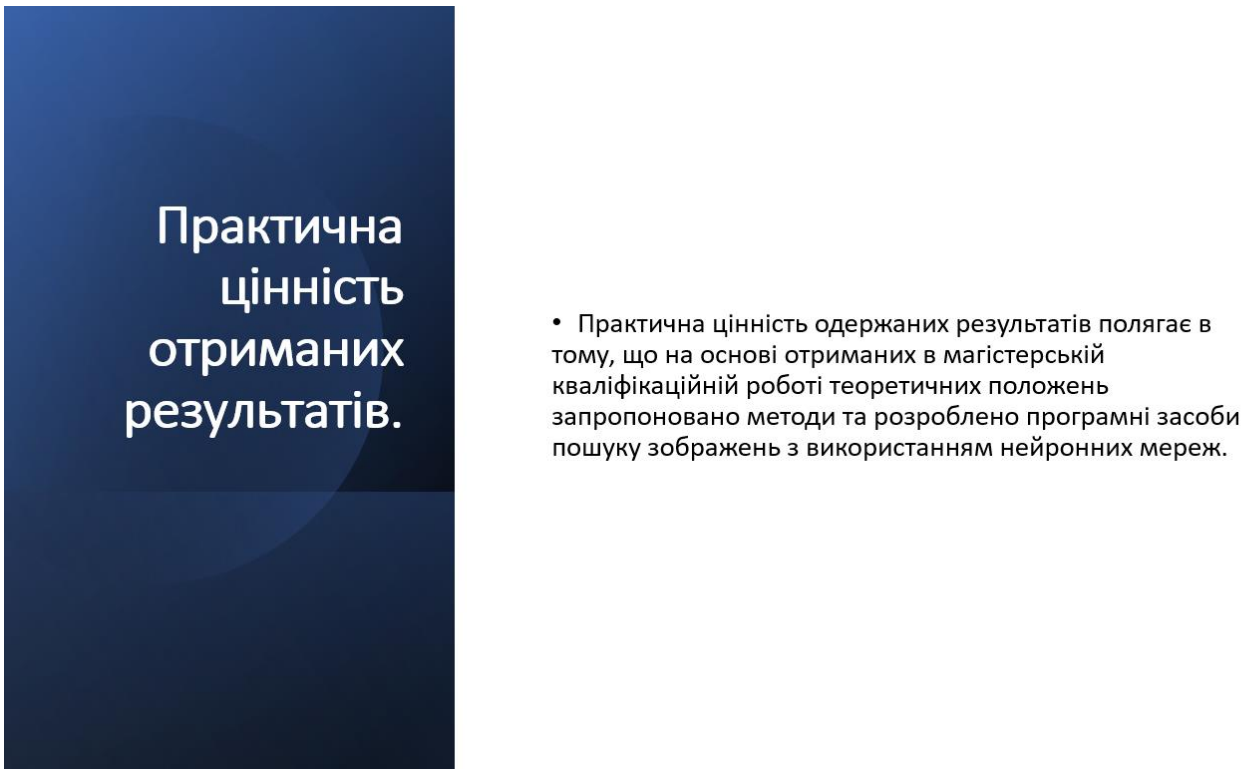

Рисунок Г.5 – Слайд презентації №5

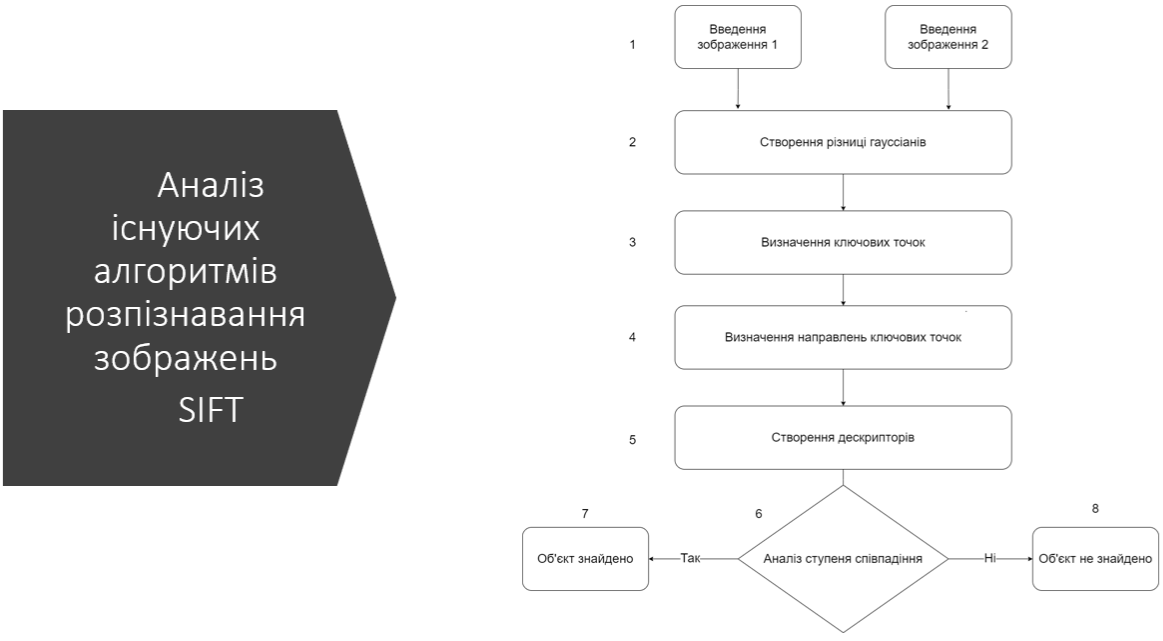

Рисунок Г.6 – Слайд презентації №6

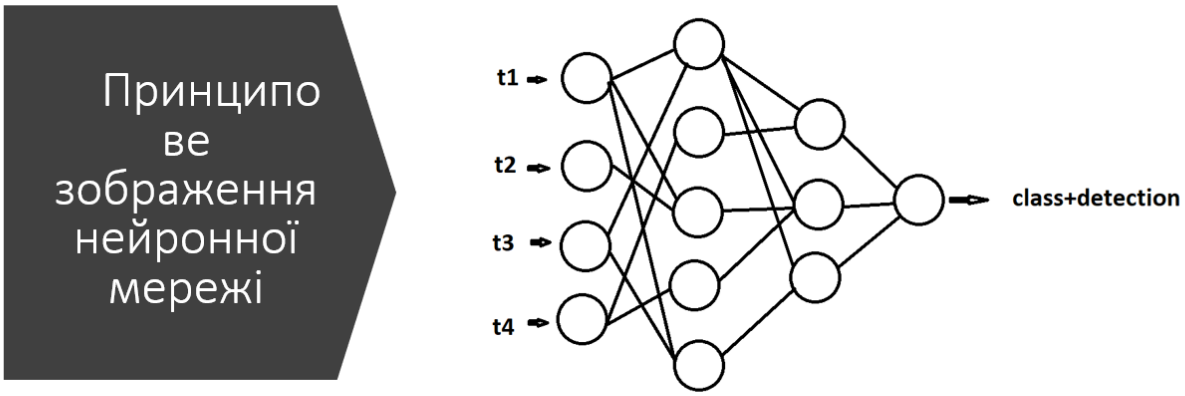

Рисунок Г.7 – Слайд презентації №7

Порівняння класичних алгоритмів з розпізнавання зображень та методу машинного навчання

	SIFT, SURF, ORB	Машинне навчання
Призначення	Алгоритм для виявлення та опису ключових точок у зображеннях	Відповідно до призначення машинне навчання може виконувати одночасно виявлення та опис ключових точок та класифікацію зображень
Швидкодія	Класичні методами комп'ютерного зору, можуть бути відносно повільними на обробці великої кількості даних або в режимі реального часу	В залежності від моделі нейронної мережі швидкодія потенційно може бути значно вищою
Точність	Алгоритми мають високу точністю та стійкістю до змін масштабу та орієнтації	В залежності від рівня тренування та налаштувань нейронної мережі точність може бути як вищою так і нижчою
Складність реалізації	Дані алгоритми вимагають високого рівня експертизи для їх реалізації та оптимізації	Основна складність полягає у навчанні та налаштування нейронної мережі. Використання бібліотек таких як TensorFlow або PyTorch значно спрощує реалізацію
Робота з великими даними	Необхідно проводити оптимізацію алгоритмів при роботі з великими даними	Машинне навчання це галузі які розвиваються пліч опліч з «великими даними». Тому при використанні бібліотек таких як PyTorch або TensorFlow дозволить швидко інтегрувати можливість роботи з великими об'ємами даних.

Рисунок Г.8 – Слайд презентації №8

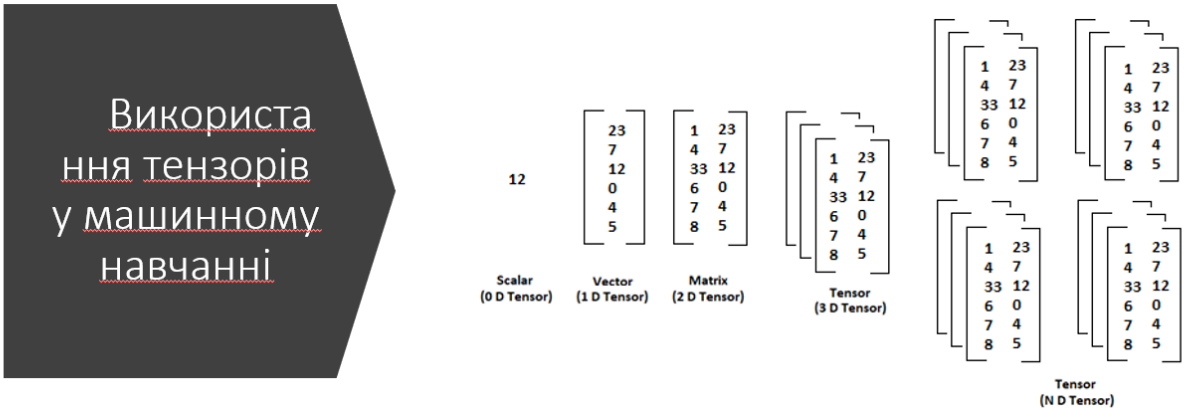

Рисунок Г.9 – Слайд презентації №9

Перетворення зображення у тензор

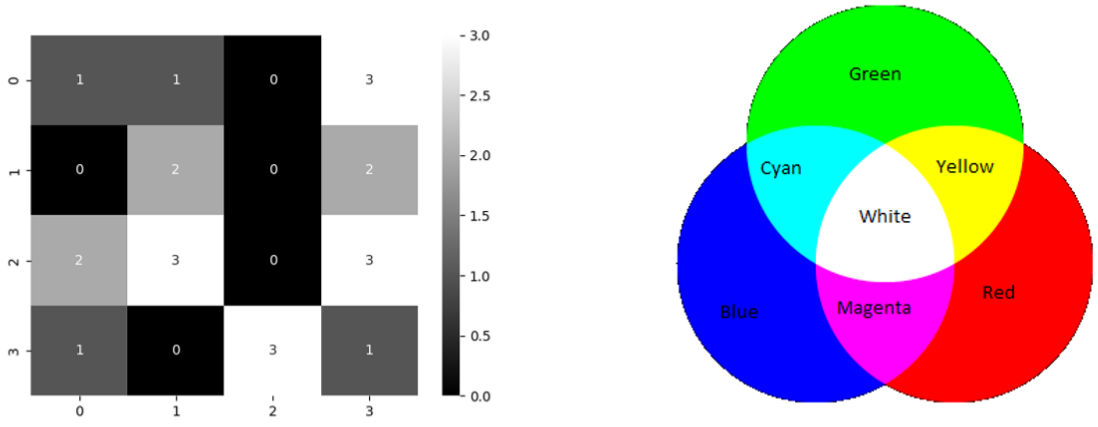

Рисунок Г.10 – Слайд презентації №10

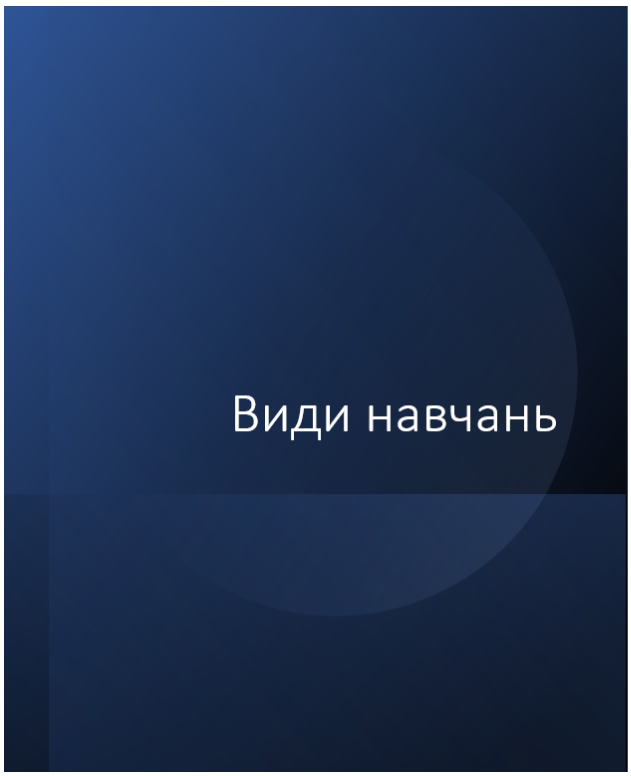

Рисунок Г.11 – Слайд презентації №11

Рисунок Г.12 – Слайд презентації №12

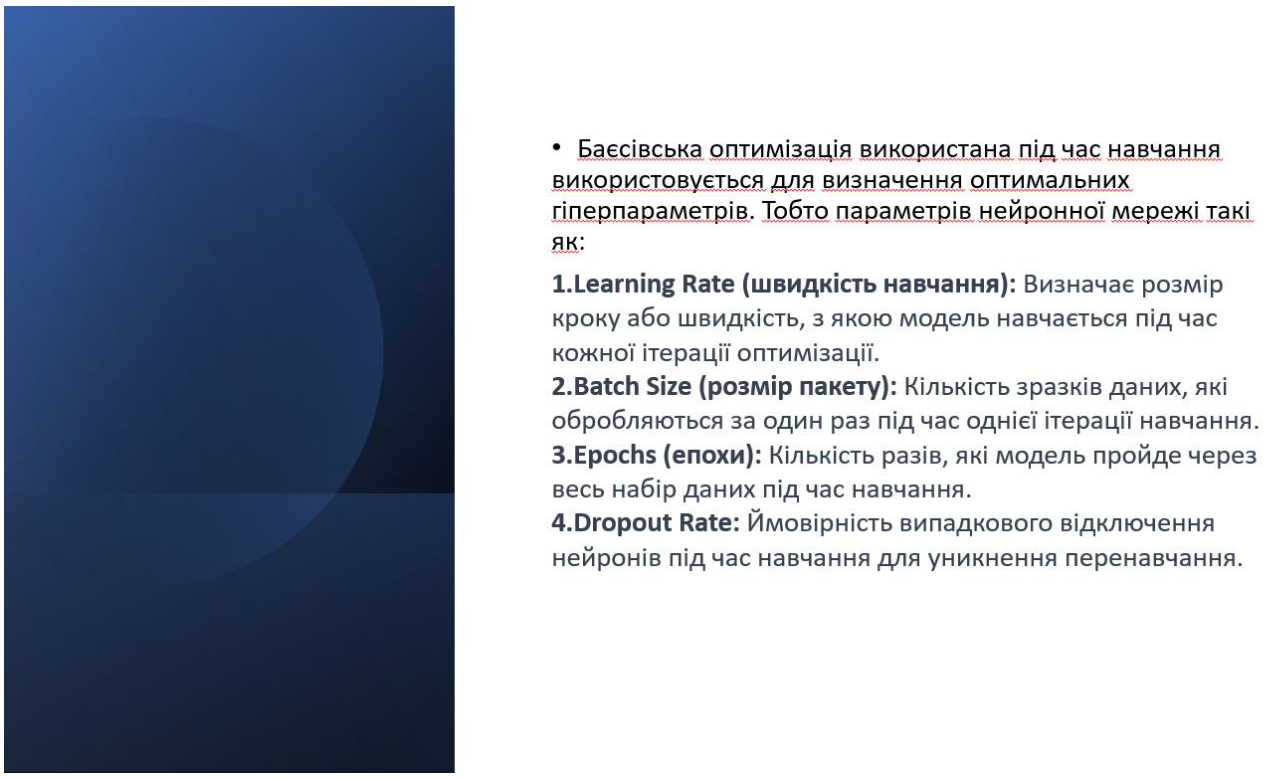

Рисунок Г.13 – Слайд презентації №13

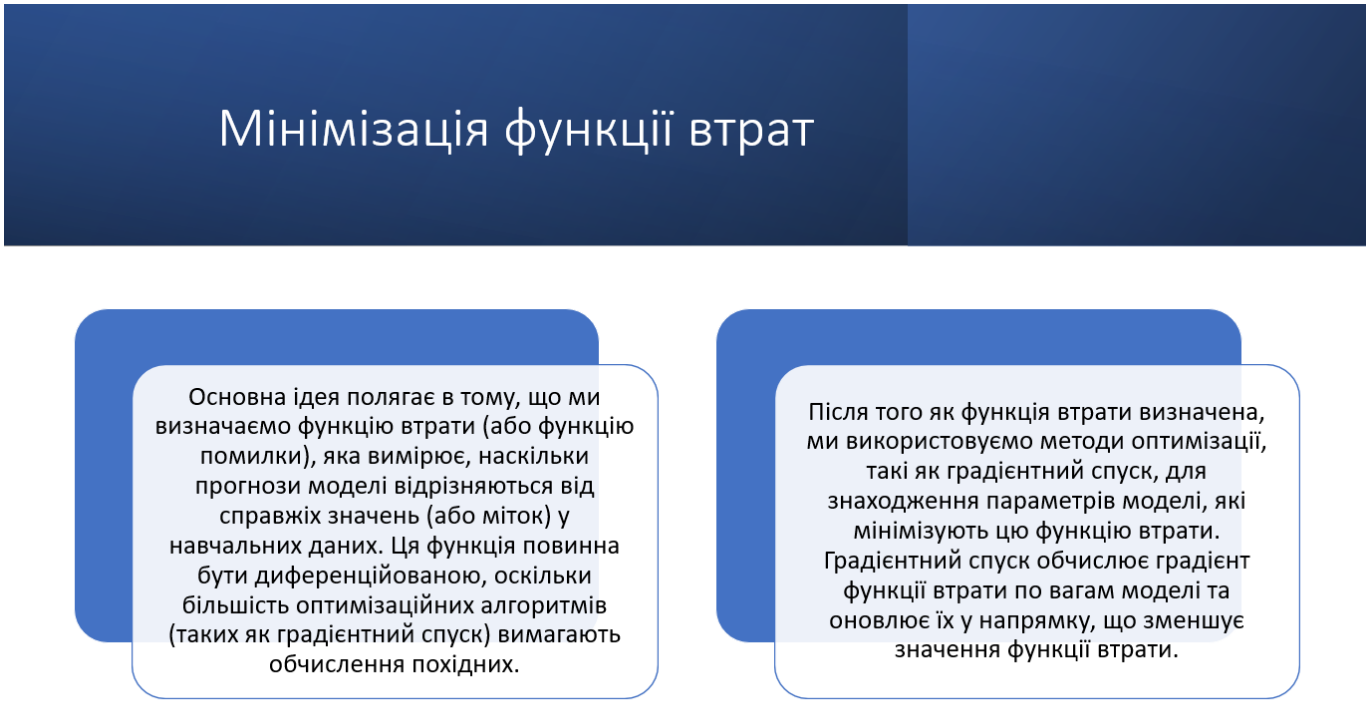

Рисунок Г.14 – Слайд презентації №14

Рисунок Г.15 – Слайд презентації №15

Тестування застосунку

Тип сценарію	Опис	Кроки для виконання	Очікуваний результат
Функціональний	Завантаження графічного зображення у систему	1. Перейти за посиланням http://127.0.0.1:5000/local 2. Натиснути кнопку «Завантажити файл» 3. Обрати файл з зображенням. 4. Натиснути кнопку «Submit»	Файл завантажено та відображено на сторінці
Функціональний	Завантажене графічне зображення що містить об'єкти повинно бути розпізнаним та класифікованим	1. Завантажити зображення яке містить об'єкт. Наприклад автомобіль	Файл завантажено та відображено на сторінці з помітками у вигляді зелених прямокутників що розташовані навколо розпізнаних об'єктів
Функціональний	Необхідно мати можливість знайти усі завантажені та розпізнані графічні зображення за назвою визначеного об'єкта	1. Перейти за посиланням http://localhost:5000/main/<name>, де name – це назва шуканих об'єктів, наприклад «car» 2. Завантажити зображення яке містить об'єкти. Наприклад автомобіль	Переконатися, що усі зображення що мають розпізнаний об'єкт з назвою «car» виведені на сторінку
Функціональний	Застосунок повинен вміти розпізнавати та визначати більше одного об'єкта на зображенні.	1. Завантажити зображення, яке містить більше одного об'єкта для розпізнавання	Файл завантажено. Більше одного об'єкта на зображенні виділено зеленою лінією. Кожен об'єкт отримав відповідну класифікацію.
Функціональний	Застосунок повинен вміти знаходити завантажені зображення, які містять більше одного об'єкта	1. Завантажити зображення, яке містить більше одного об'єкта. Ввести запит пошуку для одного з об'єктів.	Файл завантажено та проаналізовано. Запит з пошуку одного з об'єктів повертає зображення що також містить інші об'єкти.

Рисунок Г.16 – Слайд презентації №16

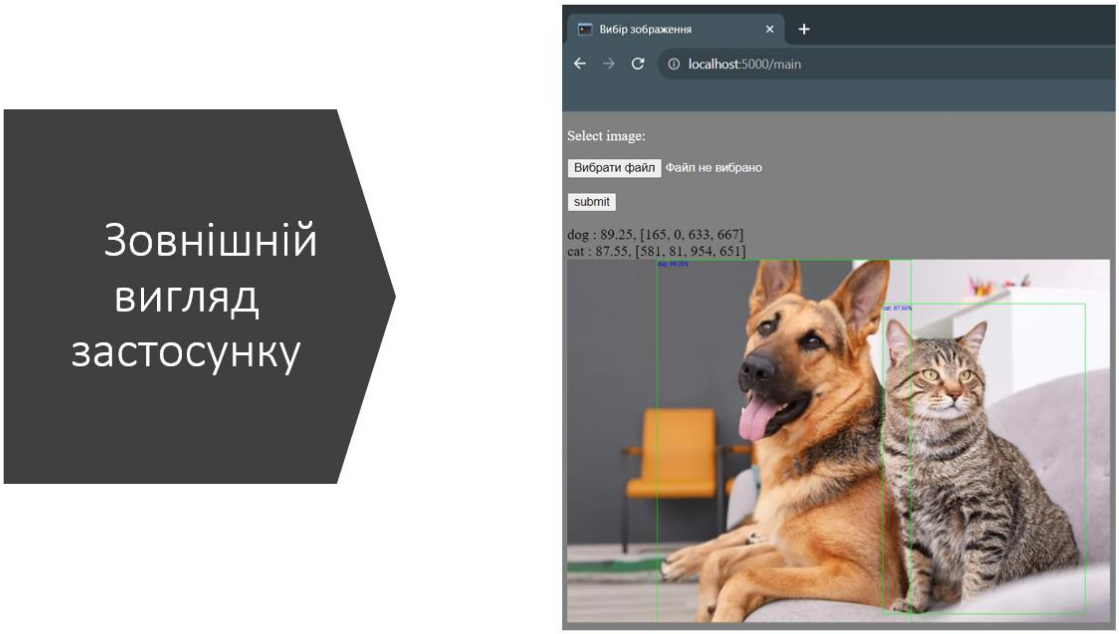

Рисунок Г.17 – Слайд презентації №17

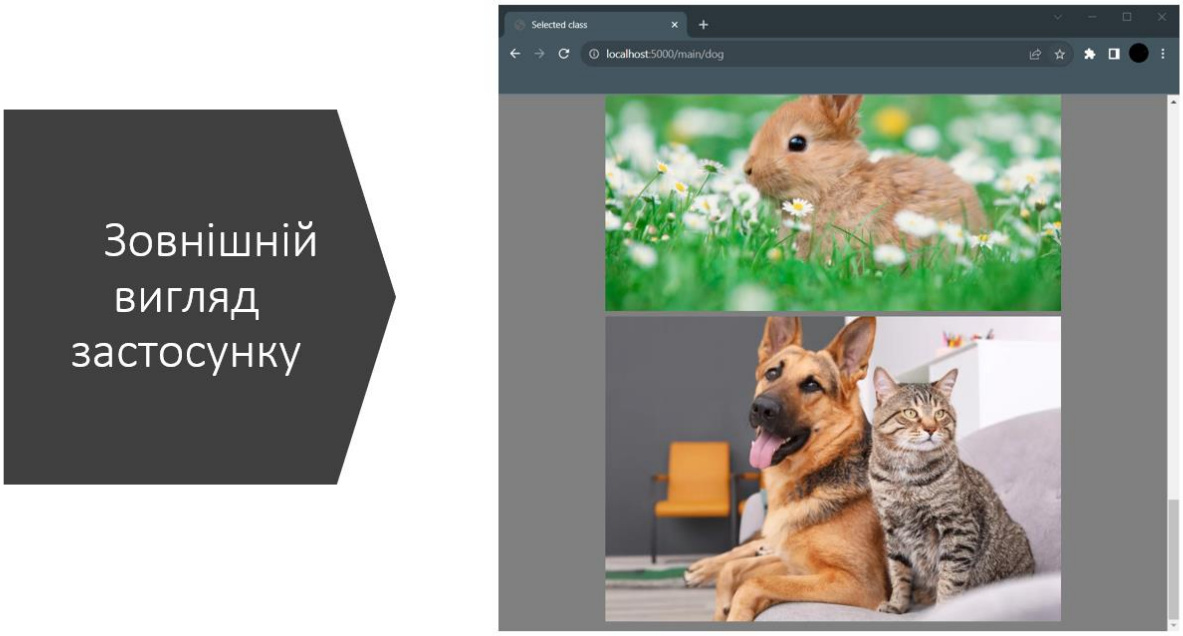

Рисунок Г.18 – Слайд презентації №18

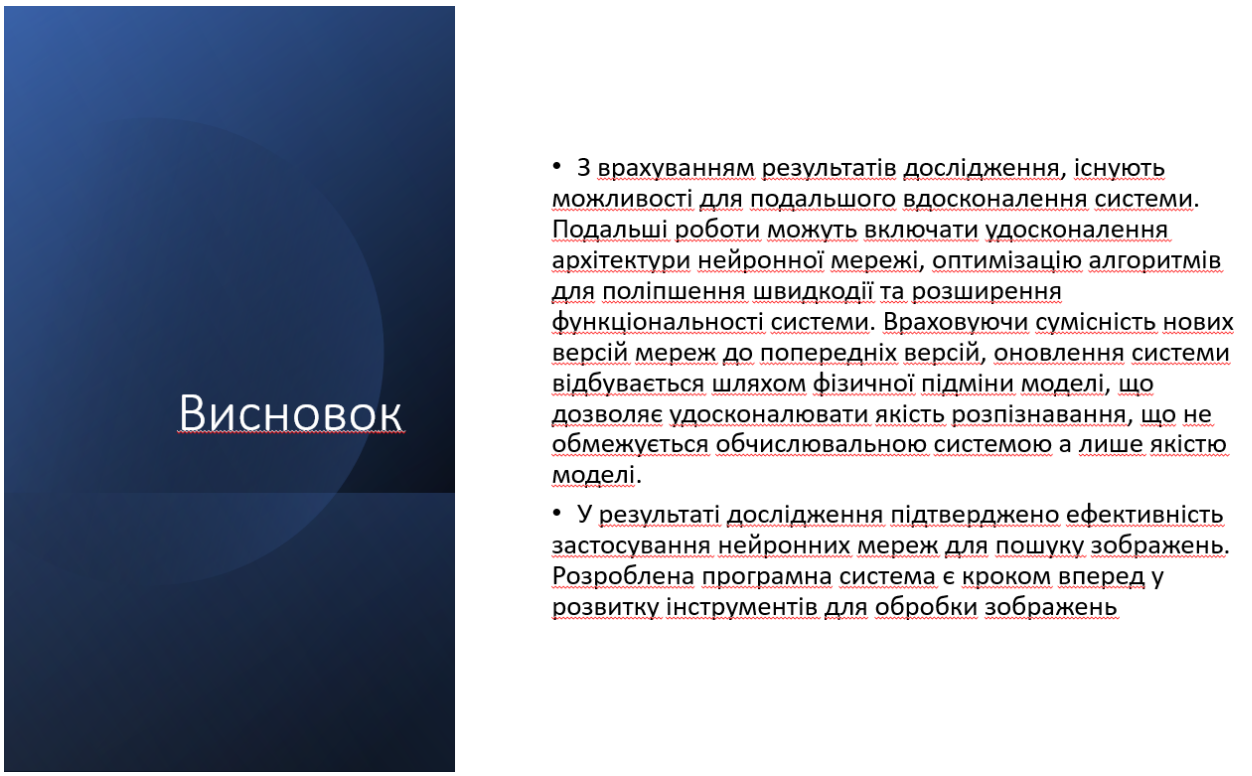

Рисунок Г.19 – Слайд презентації №19

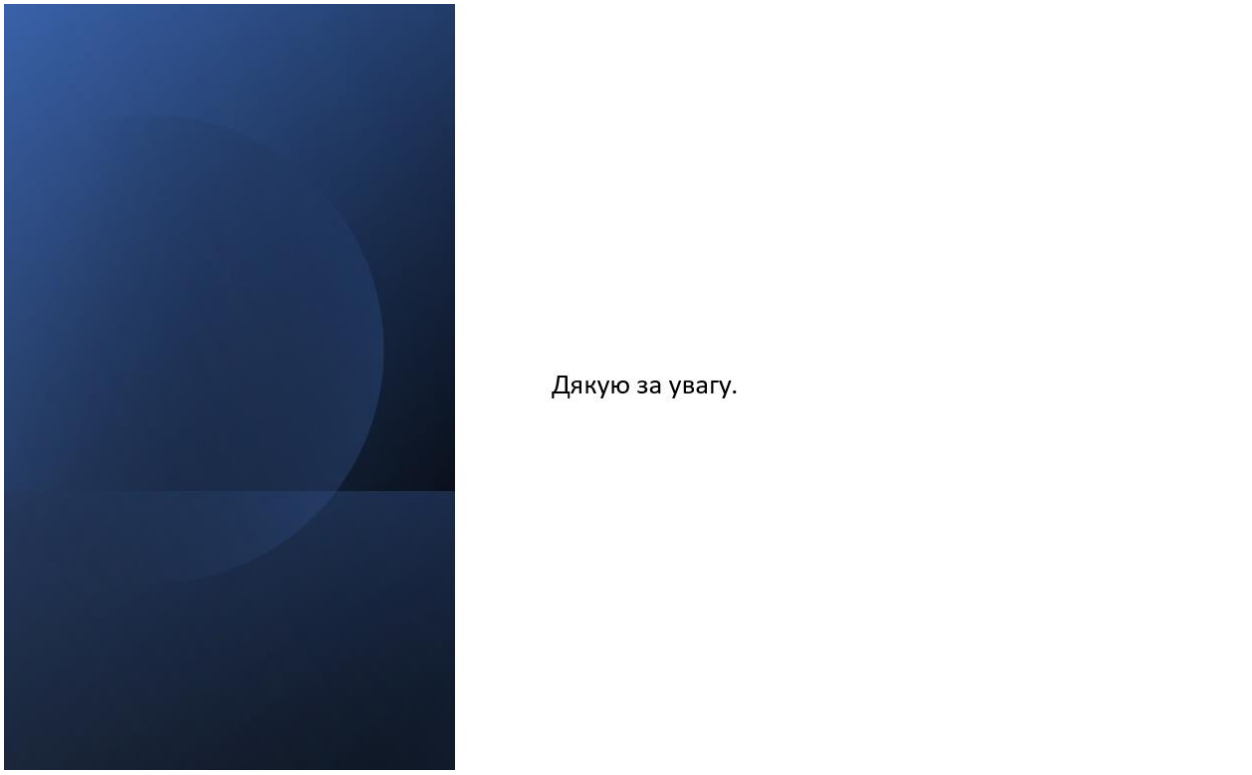

Рисунок Г.20 – Слайд презентації №20