

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ДЛЯ РОЗПІЗНАВАННЯ
ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ
DEVELOPMENT OF MODELS AND ALGORITHMS FOR OBJECT
RECOGNITION IN IMAGES

Виконала: студентка 4 курсу, групи КНТ-212

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ПАНЧЕНКО І.Д.

(ПРИЗВИЩЕ та ініціали)

Керівник ОЛІЙНИК А.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
код і найменування
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПАНЧЕНКО Ірини Дмитрівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка моделей та алгоритмів для розпізнавання об'єктів на зображеннях. Development of Models and Algorithms for Object Recognition in Images

керівник проєкту (роботи) к.т.н., професор, ОЛІЙНИК Андрій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 04 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Ірина ПАНЧЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Андрій ОЛІЙНИК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 71 с., 8 табл., 29 рис., 2 дод., 45 джерел.

КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ОБ'ЄКТІВ, ШТУЧНИЙ ІНТЕЛЕКТ, PYTHON, CNN, MLP, ПРОГРАМУВАННЯ, МАШИННЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ.

Об'єкт дослідження – процес розпізнавання об'єктів на цифрових зображеннях.

Предмет дослідження – методи машинного та глибокого навчання для класифікації та детекції об'єктів на зображеннях.

Мета роботи – зменшення кількості часу необхідного для розпізнавання об'єктів на наданому користувачем зображенні, шляхом використання нейромережових моделей за допомогою мови програмування Python та бібліотеки TensorFlow.

Матеріали, методи та технічні засоби: Fujitsu, Esprimo P7000, Windows7 x64, python 3.8.10, tensorflow 2.10.0, numpy 1.24.4, tkinter 8.6, pillow 10.2.0, ultralytics 8.4.14, CIFAR-10, Fashion MNIST, MNIST, CNN, MLP, YOLO

Результати: Застосунок з зрозумілим і зручним графічним інтерфейсом, що дозволяє тренувати моделі на наборах даних, включно з персональними, зберігати їх і використовувати для розпізнавання об'єктів на зображеннях.

Висновки: В результаті виконання завдання було створено програмне забезпечення, що дозволяє обрати модель для тренування серед MLP, YOLO та CNN, обрати датасет(CIFAR-10, Fashion MNIST, MNIST чи особистий набір даних), натренувати і зберегти модель у форматі keras та використати її відразу після тренування або завантажити пізніше. Використати модель для розпізнавання об'єктів на зображеннях формату jpg та png.

Галузь використання – медицина, сфера безпеки, освіта, промисловість.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 71 pages, 8 tables, 29 figures, 2 appendixes, 45 sources.

COMPUTER VISION, OBJECT RECOGNITION, ARTIFICIAL INTELLIGENCE, PYTHON, CNN, MLP, PROGRAMMING, MACHINE LEARNING, DEEP LEARNING.

Object of study the process of recognizing objects in digital images.

Subject of study machine and deep learning methods for classifying and recognition objects in images.

The purpose of the work is to reduce the amount of time required to recognize objects in a user-provided image by using neural network models using the Python programming language and the TensorFlow library

Materials, methods, and tools: Fujitsu, Esprimo P7000, Windows7 x64, python 3.8.10, tensorflow 2.10.0, numpy 1.24.4, tkinter 8.6, pillow 10.2.0, ultralytics 8.4.14, CIFAR-10, Fashion MNIST, MNIST, CNN, MLP, YOLO.

Results: An application with an intuitive and user-friendly graphical interface that allows you to train models on data sets, including personal ones, save them, and use them to recognize objects in images.

Conclusions: As a result of completing the task, software was created that allows you to select a model for training among MLP, YOLO, and CNN, select a dataset (CIFAR-10, Fashion MNIST, MNIST, or a personal dataset), train and save the model in Keras format, and use it immediately after training or download it later. Use the model to recognize objects in jpg and png images.

The field of use – medicine, security, education, industry.

ЗМІСТ

С.

Перелік скорочень та умовних познач	9
Вступ	10
1 Аналіз предметної області	12
1.1 Визначення предметної області	12
1.2 Проблематика предметної області	14
1.3 Використання в різних галузях	15
1.4 Наявні аналоги	15
1.5 Висновок до розділу	21
2 Матеріали і методи	22
2.1 Вибір мови програмування	22
2.2 Вибір середовища та інструментів розробки	24
2.3 Вибір нейромережових моделей для розпізнавання образів	26
2.4 Висновок до розділу	34
3 Опис програми	35
3.1 Структура програмного забезпечення	35
3.2 Функціонування програмного забезпечення розпізнавання об'єктів на зображеннях	36
3.3 Розробка інтерфейсу застосунку	38
3.4 Висновок до розділу	39
4 Експлуатація, тестування та експериментальне дослідження програми	40
4.1 Призначення й умови застосування програмного забезпечення	40

4.2	Характеристики програмного забезпечення	40
4.3	Інструкція з використання програми	41
4.3.1	Звернення до програми	41
4.3.2	Вхідні дані	41
4.3.3	Повідомлення	41
4.4	Виконання програмного забезпечення	42
4.5	Тестування програмного забезпечення	46
4.6	Висновок до розділу	46
	Висновки	47
	Перелік джерел посилання	49
	Додаток А Текст програми	53
	Додаток Б Слайди презентації	65

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	– Convolutional Neural Network;
DL	– Deep Learning;
GIL	– Global Interpreter Lock;
IoT	– Internet of Things;
IOU	– Intersection over Union;
ML	– Machine Learning;
MLP	– Multi-Layer Perceptron;
NMS	– Non-Maximum Suppression;
R-CNN	– Region-Based Convolutional Neural Network;
YOLO	– You Only Look Once;
ООП	– об'єктно-орієнтоване програмування;
ПЗ	– програмне забезпечення;
ШІ	– штучний інтелект.

ВСТУП

Людство живе в час коли штучний інтелект оточує з усіх сторін - в магазинах, в телефоні, в лікарнях чи навіть вдома. Ігнорувати можливості, які він надає неможливо. Одним з найбільш популярних методів використання ШІ є розпізнавання об'єктів на зображеннях.

Зростання обсягів цифрових зображень та відеоданих призводить до необхідності автоматизованого аналізу візуальної інформації. Традиційні алгоритми комп'ютерного зору мають обмежену ефективність при роботі зі складними сценами, тому все більшого поширення набувають методи машинного та глибокого навчання. Використання нейронних мереж дозволяє автоматично виділяти ознаки зображень та значно підвищувати точність класифікації і детекції об'єктів.

Перші спроби використання штучного інтелекту для розпізнавання об'єктів на зображеннях з'явилися ще у 1960-х роках у межах досліджень у галузі комп'ютерного зору. Тоді системи були дуже примітивними і працювали переважно з простими геометричними фігурами.

Справжній прорив стався у 2001 році, коли програма розпізнавання зображень навчилася ідентифікувати обличчя. Але це досягнення усе ще було експериментальним і не підходило для масового впровадження [1].

Наступна велика віха – 2009 рік. Google випустила додаток Goggles, який ошелешив користувачів можливостями візуального пошуку та доповненої реальності. На той час це був революційний досвід, дуже наближений до сучасних сервісів та додатків із розпізнаванням зображень.

Проте зараз розпізнавання об'єктів на зображеннях розвинулось до такого рівня, що розпізнавання об'єктів на зображеннях за допомогою ШІ допомагає у діагностиці хвороб, моніторингу врожаю, дотримання правил дорожнього руху та в комерції.

Метою роботи є створення програмного забезпечення для розпізнавання об'єктів на зображенні, що дозволить використовувати створене програмне

забезпечення для навчання, тестування та порівняння різних моделей штучного інтелекту для задач комп'ютерного зору.

Завданням роботи є:

- проаналізувати сучасні методи та алгоритми розпізнавання об'єктів на зображеннях;
- дослідити архітектури моделей машинного та глибокого навчання, зокрема MLP, CNN та YOLO;
- обрати набори даних для навчання та тестування моделей;
- розробити програмне забезпечення для навчання та тестування моделей розпізнавання зображень;
- реалізувати зручний графічний інтерфейс користувача для роботи з програмою;
- провести тестування точності та продуктивності реалізованих моделей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення предметної області

Розпізнавання об'єктів на зображеннях (Виявлення об'єктів) - це комп'ютерна технологія, пов'язана з комп'ютерним баченням та обробкою зображень, яка має справу з виявленням примірників семантичних об'єктів певного класу у цифрових зображеннях та відео [2].

Розпізнавання об'єктів на зображеннях є підгалуззю комп'ютерного зору.

Комп'ютерний зір (Computer Vision) – одна з областей штучного інтелекту, яка дає змогу комп'ютерам і системам отримувати корисну інформацію з цифрових зображень, відео, візуальних даних та виконувати дії або давати рекомендації на основі отриманої інформації [3].

Комп'ютерний зір, в свою чергу, входить до ширшої області - штучного інтелекту, адже реалізується методами машинного навчання та глибокого навчання.

Методи машинного навчання та глибокого навчання надають можливість реалізувати розпізнавання об'єктів на зображеннях. Порівняння навчань проведено у таблиці 1.1

Таблиця 1.1 – Порівняння машинного та глибокого навчання

	ML	DL
1	2	3
Категорія	Штучний інтелект	Штучний інтелект
Підмножина	Ні	Так, є підмножиною машинного навчання
Залежність	Від вхідних даних для навчання	Від шарів
Навчання	Дозволяє порівняно швидше навчання на основі даних; більше даних дорівнює кращим результатам.	Потрібні інтенсивні обчислення для навчання нейронних мереж з кількома неприємними та невидимими шарами.

Продовження таблиці 1.1

1	2	3
Продуктивність	Деякі форми досягли «плато у продуктивності».	Використання нейронних мереж та наявність надшвидких комп'ютерів прискорили зростання.
Ручне втручання	Щоразу, коли йдеться про нове навчання, розробники-люди повинні втручатися та адаптувати алгоритм для того, щоб навчання відбувалося.	Нейронні мережі сприяють багаторівневому навчанню, де розумні алгоритми можуть навчити машину використовувати знання, отримані з одного шару на наступний, для подальшого навчання без втручання людини.
Навчання	Розробник-людина вказує машині, який тип функції шукати. Вимагає формулювання проблеми, щоб розбити проблему на різні частини, які потрібно буде вирішити згодом, а потім об'єднати результати на завершальному етапі.	Процес вилучення ознак повністю автоматизований. Як результат, вилучення ознак у глибокому навчанні є точнішим та орієнтованим на результат. Вирішуйте проблему від початку до кінця, роблячи процес навчання швидшим та надійнішим.
Дані	Залежить від керованого дослідження вибірок даних, які все ще великі, але порівняно менші.	Оскільки нейронні мережі спираються на багаторівневі знання без втручання людини, для навчання потрібна велика кількість даних.
Точність	Помилки розробника можуть призвести до неправильних рішень та низької точності.	Можливості самонавчання дозволяють досягати швидших і точніших результатів[4].

Отже, після проведеного аналізу можна дійти висновку, що для розпізнавання об'єктів на зображеннях краще підійде глибоке навчання. Моделі глибокого навчання застосовні для будь-якого завдання машинного навчання, але вони особливо ефективні під час роботи з наборами даних, що містять великі числові масиви, такі як зображення [5]. Технологія працює шляхом виявлення

ключових та візуальних особливостей на зображеннях, які є важливими для точного пошуку та розпізнавання зображень на основі вмісту [6].

Існує два підходи до розпізнавання об'єктів за допомогою глибокого навчання:

- навчання моделі з нуля: щоб навчити глибоку мережу з нуля, необхідно зібрати дуже великий набір даних з позначками та проектує архітектуру мережі, яка вивчить функції та побудує модель. Результати можуть бути вражаючими, але цей підхід вимагає великої кількості навчальних даних, і є необхідність налаштувати шари та ваги в моделі розпізнавання;

- використання попередньо навченої моделі глибокого навчання: більшість програм глибокого навчання використовують підхід перенесення навчання, процес, який включає точне налаштування попередньо навченої моделі. Цей метод менш трудомісткий і може забезпечити швидший результат, оскільки модель вже навчена на тисячах або мільйонах зображень [7].

1.2 Проблематика предметної області

Виникають наступні проблеми розпізнавання зображень за допомогою штучного інтелекту:

- погана якість даних: моделі потребують великих та різноманітних наборів даних. Без достатньої різноманітності прогнози можуть бути упередженими або неточними;

- реальна складність: освітлення, кути та зашарашений фон ускладнюють точну ідентифікацію об'єктів штучним інтелектом;

- анотація, що займає багато часу: маркування зображень для навчання є повільним і дорогим процесом, але важливим для точних моделей;

- обмежена гнучкість: моделі штучного інтелекту, навчені для одного завдання, часто мають труднощі з адаптацією до нових застосувань;

- питання конфіденційності: занепокоєння щодо зловживань, таких як стеження та розпізнавання облич, викликають етичні питання;

- ризики безпеки: невеликі зміни в зображеннях можуть обдурити системи штучного інтелекту, що призведе до неправильних результатів;
- високі витрати: навчання ШІ вимагає потужного обладнання та значних енерговитрат, що може бути дорогим;
- відсутність прозорості: моделі штучного інтелекту часто працюють як «чорні ящики», що ускладнює розуміння їхніх рішень [6].

1.3 Використання в різних галузях

Розпізнавання об'єктів – це ключова технологія, що лежить в основі безпілотних автомобілів, яка дозволяє їм розпізнавати знак зупинки або відрізнити пішохода від ліхтарного стовпа. Вона також корисна в різних сферах застосування, таких як ідентифікація захворювань у біовізуалізації, промисловому інспектуванні та роботизованому зорі [7].

1.4 Наявні аналоги

Предметна область така як розпізнавання об'єктів на зображеннях доволі широко розвинена в наш час. Через це в даній сфері є чимало аналогів, найпопулярнішими за кількістю користувачів функціоналом є перелічені далі.

Google Teachable Machine - це веб-інструмент, який робить створення моделей машинного навчання швидким, простим та доступним для всіх[8]. Алгоритм роботи починається спочатку збираються та групуються особисті приклади в класи або категорії, які користувач хоче, щоб комп'ютер вивчив. Потім користувач навчає свою модель після чого миттєво тестує її, щоб побачити, чи може вона правильно класифікувати нові приклади. На завершення алгоритму йде можливість експорту моделі для її подальшого використання. Веб-інструмент зображений на рисунку 1.1.

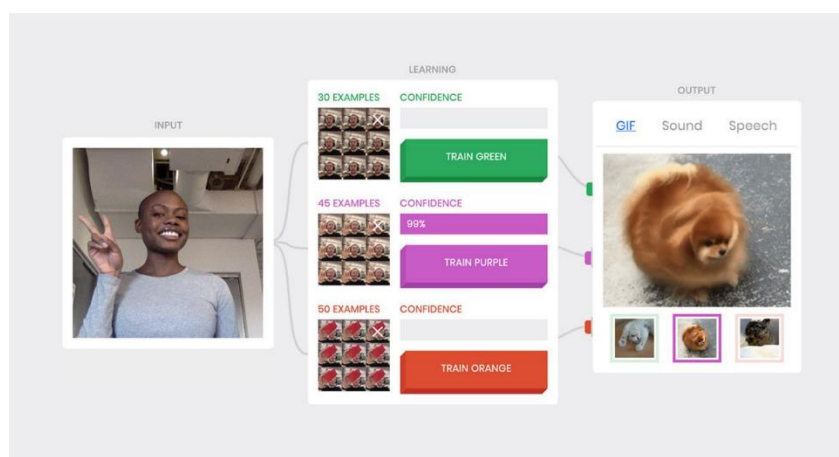


Рисунок 1.1 - Google Teachable Machine[9]

Edge Impulse Studio - було створено для консолідації та стандартизації робочого процесу створення моделі машинного навчання. Хмарна платформа дозволяє легко створювати, навчати та розгортати моделі машинного навчання для периферійних пристроїв в одному місці, забезпечуючи водночас прозорість та функції, які значно покращують процес MLOps(парадигма, метою якої є надійне та ефективне розгортання та підтримка моделей машинного навчання у виробництві[10]) та його можливості для користувача. Розроблена для інженерів та розробників, незалежно від того, чи створюють вони продукт, чи тільки вперше знайомляться з ML, вона спрощує весь робочий процес — від збору даних до розгортання — за допомогою інтуїтивно зрозумілого веб-інтерфейсу[11]. Детальніше можна побачити на рисунку 1.2.

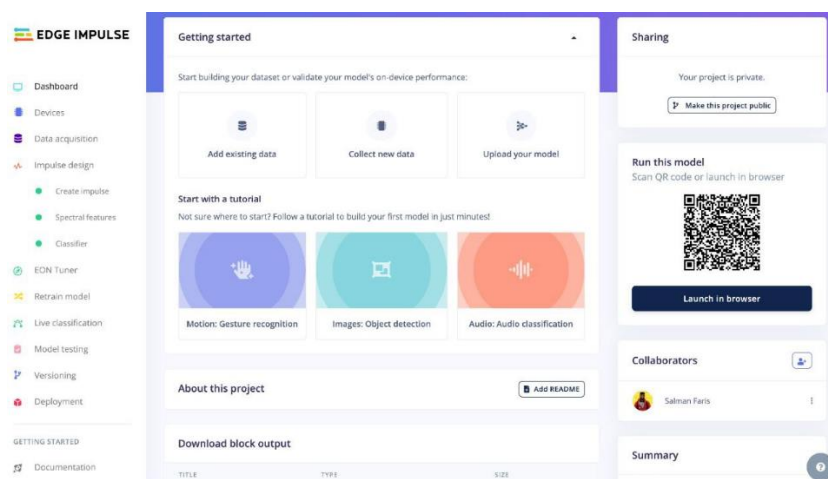


Рисунок 1.2 - Edge Impulse Studio[12]

Roboflow - це хмарна платформа для комп'ютерного зору, що надає розробникам і підприємствам повний набір інструментів для створення, навчання та розгортання моделей штучного інтелекту. Вона спрощує увесь цикл розробки — від збору та розмітки даних до інференсу на периферійних пристроях або в хмарі [13]. Платформа зображена на рисунку 1.3.

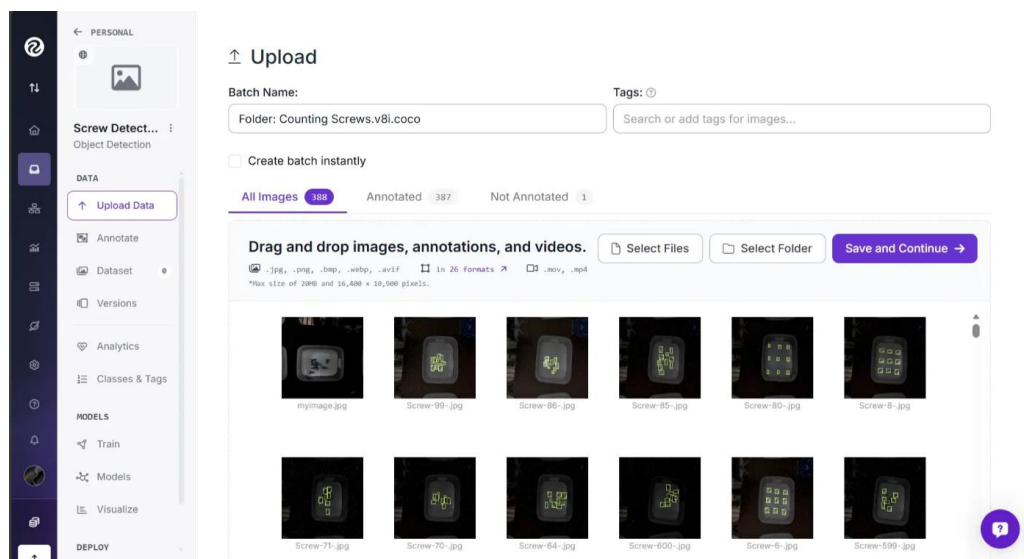


Рисунок 1.3 – Roboflow[14]

Labellmg - це легкий та простий у використанні інструмент для анотації зображень, який дозволяє позначати обмежувальні рамки об'єктів на зображеннях. У машинному навчанні та глибокому навчанні анотування зображень — це техніка позначення або категоризації окремих зображень за допомогою тексту анотації, програмних інструментів або обох методів для відображення ознак даних, які модель ML/DL користувача має самостійно ідентифікувати[15]. Сервіс зображено на рисунку 1.4.



Рисунок 1.4 – LabelImg[16]

TensorBoard – застосунок, що надає візуалізацію та інструменти, необхідні для експериментів з машинним навчанням такі як відстеження та візуалізація показників, таких як втрати та точність, візуалізація графа моделі (операції та шари), перегляд гістограм ваг, зміщень або інших тензорів у міру їх зміни з часом, проектування вбудовування у простір нижчої вимірності, відображення зображень, тексту та аудіоданих, профілювання програм TensorFlow і багато іншого[17]. Застосунок можна побачити на рисунку 1.5.

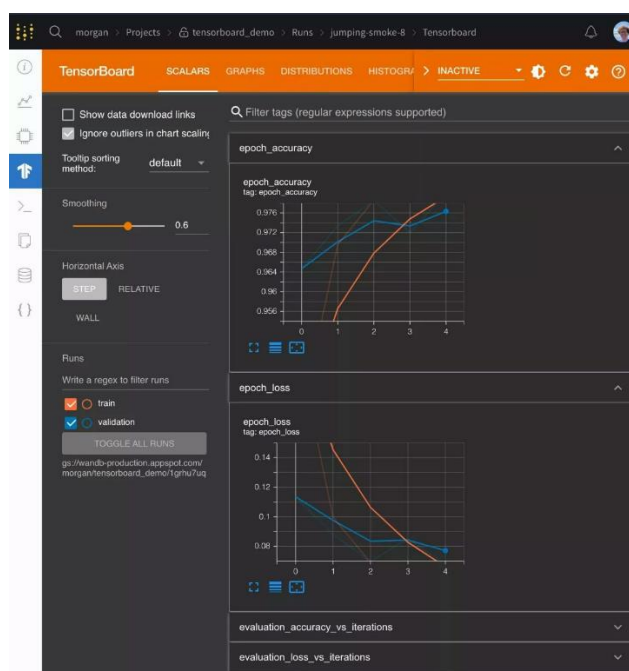


Рисунок 1.5 – TensorBoard[18]

Враховуючи всі вище перелічені інструменти, де частково або повністю наявне розпізнавання об'єктів на зображеннях, виникає потреба в їх порівнянні. Детальне порівняння наведено у таблиці 1.2

Таблиця 1.2 – Порівняння програм-аналогів

Критерій	Google Teachable Machine	Edge Impulse Studio	Roboflow	LabelImg	TensorBoard
1	2	3	4	5	6
Тип інструменту	Онлайн-платформа для навчання моделей	Платформа для розробки моделей ML для edge-пристроїв	Платформа для підготовки датасетів і навчання моделей	Інструмент для розмітки зображень	Інструмент для візуалізації процесу навчання
Основне призначення	Створення моделей класифікації без програмування	Розробка та оптимізація ML моделей для IoT	Підготовка даних, аугментація та навчання моделей	Розмітка об'єктів на зображеннях	Моніторинг та аналіз навчання нейронних мереж
Основне призначення	Створення моделей класифікації без програмування	Розробка та оптимізація ML моделей для IoT	Підготовка даних, аугментація та навчання моделей	Розмітка об'єктів на зображеннях	Моніторинг та аналіз навчання нейронних мереж
Тип задач	Класифікація зображень, звуку, поз	Класифікація та детекція	Детекція об'єктів, класифікація	Детекція об'єктів	Аналіз навчання моделей
Необхідність програмування	Ні	Частково	Частково	Ні	Так
Робота з датасетами	Обмежена	Повна підтримка	Розширені можливості	Лише розмітка	Не працює з датасетами
Підтримка нейронних мереж	TensorFlow	TensorFlow, PyTorch	YOLO, TensorFlow, PyTorch	Не підтримує	Підтримує TensorFlow

Продовження таблиці 1.2

1	2	3	4	5	6
Можливість тренування моделей	Так	Так	Так	Ні	Ні
Візуалізація результатів	Базова	Детальна	Детальна	Відсутня	Розширена
Робота без Інтернету	Ні	Частково	Ні	Так	Так
Основна перевага	Простота використання	Оптимізація для edge-пристроїв	Потужні інструменти роботи з даними	Простота розмітки	Детальний аналіз навчання
Основний недолік	Обмежена гнучкість	Потребує налаштування	Частина функцій платна	Немає тренування моделей	Не виконує навчання моделей [8]-[18]

Серед великої кількості застосунків в даній предметній області було розглянуто Google Teachable Machine, Edge Impulse Studio та Roboflow. Вони надають веб-інтерфейси для навчання моделей комп'ютерного бачення, проте не орієнтовані на використання з персонального комп'ютера з інтеграцією кількох моделей в єдину систему.

Переваги програмного забезпечення, що необхідно розробити в рамках завдання:

- автономна робота на персональному комп'ютері;
- навчання моделей CNN та MLP;
- використання моделі YOLO;
- збереження моделі як повністю натренованої так і частково, для подальшого продовження тренування;
- незалежність від мережі Інтернет.

1.5 Висновок до розділу

Ознайомившись з предметною областю, її проблемами та викликами завданням дипломної роботи є аналіз аналогів і розроблення програмного забезпечення, що б виконувало розпізнавання об'єктів на зображеннях за допомогою різних моделей глибокого навчання. Програмне забезпечення має містити графічну складову, декілька моделей, та можливості тренувати та зберігати моделі. Також воно має містити головну перевагу над розглянутими аналогами – не залежати від Інтернету.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

Головним інструментом в будь-якому написанні коду є мова програмування. Від вибору залежить наскільки багатофункціональним, якісним і точним буде створене програмне забезпечення. Для виконання задач комп'ютерного зору, необхідного для розпізнавання об'єктів на зображеннях, найбільш доцільно використовувати Python, Rust чи C++.

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Розроблена на початку 1990-х років Гвідо ван Россумом [19].

Rust - сучасна багатопарадигмова мова програмування загального призначення. Мова має сильну (строгу) типізацію і зосереджена на безпечній роботі з пам'яттю та забезпеченні високої рівночасності виконання задач (можливість породжувати тисячі й навіть мільйони підпроцесів) [20].

C++ - мова програмування загального призначення з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої, процедурної та ін. Б'ярн Страуструп почав створювати C++ в AT&T Bell Laboratories у 1979 році [21]. Порівняння мов наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мов для використання

	Python	Rust	C++
1	2	3	4
Тип мови	Інтерпретована високорівнева мова	Компілювана системна мова	Компілювана системна мова
Типізація	Динамічна	Статична	Статична
Парадигми програмування	ООП, процедурна, функціональна	Системна, функціональна, процедурна	ООП, процедурна, шаблонна
Швидкість виконання	Низька–середня	Висока	Дуже висока

Продовження таблиці 2.1

1	2	3	4
Управління пам'яттю	Garbage Collector	Ownership + Borrowing	Ручне або smart pointers
Безпека пам'яті	Висока, але з більшими витратами продуктивності	Дуже висока завдяки borrow checker	Нижча, можливі помилки типу use-after-free
Багатопоточність	Обмежена (GIL)	Потужна та безпечна	Потужна
Складність навчання	Легка	Складна	Складна
Основні сфери використання	Data Science, AI, Web, automation	Системне ПЗ, backend, embedded	Game dev, системне ПЗ, high-performance
Продуктивність	Нижча за компільовані мови	Близька до C++	Дуже висока
Кросплатформність	Висока	Висока	Висока [22]-[26]

Важливим фактором при виборі мови програмування є наявність бібліотек для реалізації алгоритмів комп'ютерного зору та машинного навчання. Наведені вище мови мають бібліотеки для реалізації задач комп'ютерного зору, що наведені в таблиці 2.2

Таблиця 2.2 – Бібліотеки обраних мов

Мова	Бібліотеки для CV та ML
Python	OpenCV, TensorFlow, PyTorch, Keras, Ultralytics YOLO
Rust	tch-rs, burn, opencv-rust
C++	OpenCV, Caffe, TensorRT

Проаналізувавши переваги і недоліки мов програмування найбільш доцільних до використання було обрано високорівневу мову програмування Python. Мова має такі переваги як:

- велика кількість бібліотек для комп'ютерного зору та машинного навчання;
- простота та швидкість розробки програмного забезпечення;

- активна спільнота розробників та велика кількість навчальних матеріалів;
- простий та зрозумілий синтаксис;
- підтримка сучасних моделей глибокого навчання.

2.2 Вибір середовища та інструментів розробки

Програмне забезпечення, що надає можливість розпізнавати об'єкти на зображеннях за допомогою високорівневої мови програмування Python не вимагає великого інструментарію для розробки. Перше, що необхідно – середовище розробки, що надасть можливість використовувати необхідні бібліотеки та створити мінімальний зручний і зрозумілий інтерфейс. Серед наявних середовищ найбільш вживаними є IDLE, PyCharm та Visual Studio Code. Порівняння даних середовищ наведено в таблиці 2.3

Таблиця 2.3 – Порівняння середовищ розробки

Критерій	IDLE	PyCharm	Visual Studio Code
Тип середовища	Просте IDE	Професійне IDE	Редактор коду з розширеннями
Встановлення	Встановлюється разом з Python	Встановлюється окремо	Встановлюється окремо
Споживання ресурсів	Низьке	Високе	Середнє
Складність освоєння	Низька	Середня	Середня
Налагодження	Базове	Розширене	Розширене
Підходить для ML	Так (через бібліотеки)	Так	Так
Використання у великих проєктах	Так але обмежене	Так	Так[27]-[29]

Відповідно до вимог програмного забезпечення та властивостей програм було обрано IDLE як середовищ розробки. Наступним кроком є вибір бібліотек для реалізації алгоритмів комп'ютерного зору. Враховуючи вимоги завдання було обрано наступні бібліотеки:

— TensorFlow - основна бібліотека для створення та навчання нейронної мережі, необхідна для створення моделі розпізнавання зображень, навчання її на датасеті, збереження натренованої моделі та використання її для розпізнавання нових зображень;

— Keras – бібліотека-доповнення TensorFlow, що дозволяє спростити роботу, наприклад простіше створювати шари нейромережі, швидше описувати архітектуру моделі чи запускати навчання;

— pandas - бібліотека для роботи з даними, у проєкті використовується для роботи з таблицями датасету, читання CSV файлів та підготовки даних перед навчанням;

— Pillow - бібліотека для обробки зображень, що необхідна для відкривання зображень, зміни їх розміру та конвертування формату щоб підготувати зображення перед подачею у нейромережу;

— NumPy - це базова бібліотека для роботи з числовими масивами та матрицями. У розробці вона буде використана для перетворення зображень у масиви чисел, математичних операцій над даними, підготовки даних перед подачею у нейромережу та нормалізації пікселів (наприклад, значення від 0 до 1);

— Tkinter – бібліотека необхідна для графічного інтерфейсу.

Використання зазначених бібліотек дозволяє реалізувати систему розпізнавання зображень. Поєднання TensorFlow та Keras забезпечує створення та навчання нейронної мережі, Pillow виконує попередню обробку зображень, pandas використовується для роботи з даними, а Tkinter дозволяє реалізувати зручний і зрозумілий графічний інтерфейс користувача для взаємодії з програмою. Бібліотека NumPy застосовується для виконання математичних операцій та роботи з багатовимірними масивами даних, що дозволяє обробляти піксельні значення зображень перед їх подачею до нейронної мережі. В результаті буде розроблено кросплатформний застосунок, що дозволяє використовувати різні моделі комп'ютерного зору для розпізнавання об'єктів на зображеннях.

2.3 Вибір нейромережових моделей для розпізнавання образів

Розроблене програмне забезпечення має задовольняти наступні вимоги:

- максимально можлива точність і різноманітність моделей;
- різноманітність тренувальних даних, включно з можливістю завантажити особистий даних;
- можливість збереження натренованих моделей у файл;
- незалежність від мережі Інтернет;
- зручний і зрозумілий графічний інтерфейс;
- інформування користувача про помилки через інтерфейс.

Розробка програмного забезпечення, що надає можливість розпізнавати об'єкти на зображеннях неможливе без використання ШІ, а саме глибокого навчання, отже неможливе і без використання розпізнавальних моделей. Для даної задачі найбільш доречними будуть такі моделі як CNN, MLP та YOLO.

CNN (Convolutional Neural Network) – найбільш використовувана згорткова нейронна мережа, що використовуються в галузі комп'ютерного зору. Їхня здатність автоматично вивчати просторові ієрархії ознак із зображень робить їх найкращим вибором для таких завдань.

CNN витягує ознаки із зображень, а потім передає ці ознаки в повністю зв'язану нейронну мережу для створення прогнозу. Шари вилучення ознак у мережі допомагають зменшити кількість ознак з великого масиву значень окремих пікселів до більш керованого набору ознак, який використовується для підтримки прогнозування міток [30]. Структуру даної моделі можна побачити на рисунку 2.1.

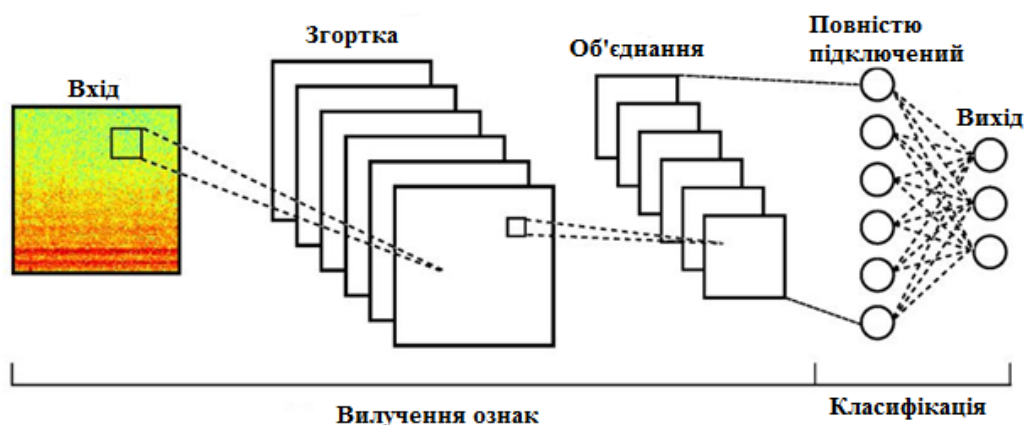


Рисунок 2.1 – Структура CNN[31]

Алгоритм роботи моделі CNN:

а) CNN отримує вхідне зображення, яке попередньо обробляється для забезпечення однорідності розміру та формату;

б) згорткові шари: до вхідного зображення застосовуються фільтри для вилучення таких ознак, як краї, текстури та форми. Розмір вихідного об'єму обчислюється за формулою (2.1):

$$W_{out} = \frac{W - F + 2P}{S} + 1, \quad (2.1)$$

де W — вага;

F — просторовий розмір;

P — величина доповнення;

S — крок [32].

в) об'єднання шарів: карти ознак, згенеровані згортковими шарами, піддаються зниженню розмірності;

г) повністю зв'язані шари: зменшені карти ознак пропускаються через повністю зв'язані шари для отримання кінцевого результату, такого як мітка класифікації;

д) CNN виводить прогноз, такий як клас зображення [33].

Перцептрон — один із найпростіших типів штучних нейронів, що використовуються в машинному навчанні. Це будівельний блок штучних нейронних мереж, який навчається на позначених даних для виконання завдань класифікації та розпізнавання образів, зазвичай на лінійно роздільних даних.

Одношаровий перцептрон складається з одного шару штучних нейронів, які називаються перцептронами. Але коли при поєднанні великої кількості перцептронів разом у шари, виходить багатошаровий перцептрон (MLP). Це дозволяє мережі вивчати складніші закономірності, комбінуючи прості рішення від кожного перцептрона. І це робить MLP потужними інструментами для таких завдань, як розпізнавання зображень та обробка природної мови [34].

Отже, MLP (Multi-Layer Perceptron) - це тип нейронної мережі прямого зв'язку, що складається з повністю пов'язаних нейронів з нелінійним типом функції активації [35]. Структуру даної моделі можна побачити на рисунку 2.2.

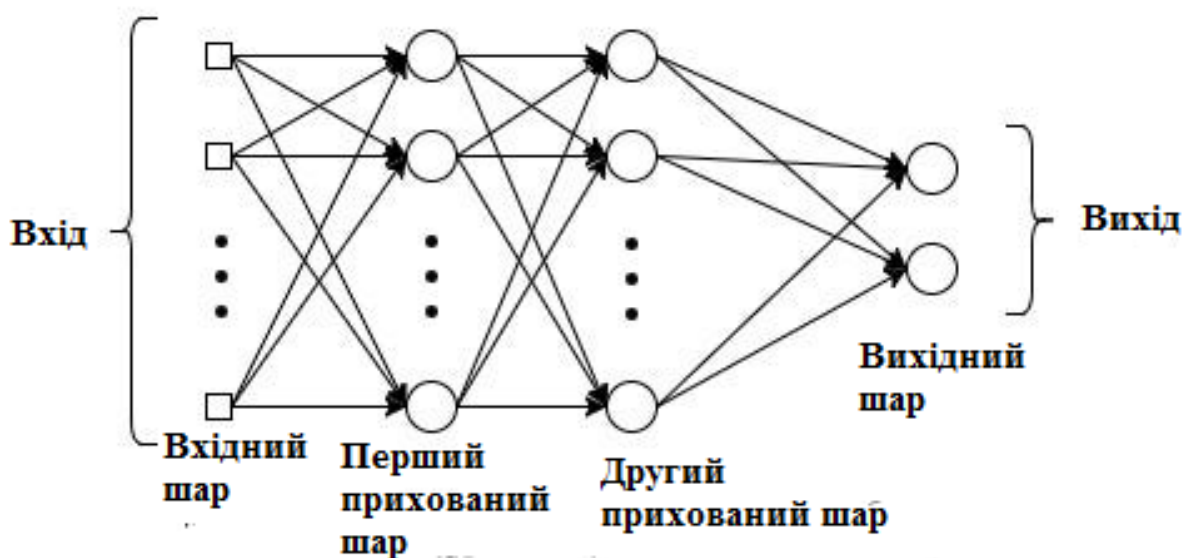


Рисунок 2.2 – Структура MLP[36]

Алгоритм роботи моделі MLP:

а) вхідний шар;

1) вхідний шар MLP отримує вхідні дані, які можуть бути ознаками, витягнутими з вхідних зразків у наборі даних. Кожен нейрон у вхідному

шарі представляє одну ознаку.

2) нейрони вхідного шару не виконують жодних обчислень; вони просто передають вхідні значення нейронам першого прихованого шару;

б) приховані шари;

1) приховані шари MLP складаються з взаємопов'язаних нейронів, які виконують обчислення над вхідними даними;

2) кожен нейрон у прихованому шарі отримує вхідні дані від усіх нейронів попереднього шару. Вхідні дані множаться на відповідні вагові коефіцієнти, що позначаються як w . Вагові коефіцієнти визначають, наскільки вхідні дані від одного нейрона впливають на вихідні дані іншого;

3) окрім вагових коефіцієнтів, кожен нейрон у прихованому шарі має пов'язане зміщення, яке позначається як b . Зміщення забезпечує додатковий вхідний сигнал для нейрона, дозволяючи йому регулювати свій вихідний поріг. Як і ваги, зміщення засвоюються під час навчання;

4) для кожного нейрона в прихованому шарі або вихідному шарі обчислюється зважена сума його вхідних даних. Це включає множення кожного вхідного сигналу на відповідну вагу, підсумовування цих добутків та додавання зміщення. Зважена сума вхідних даних обчислюється за формулою (2.2):

$$W = \sum_{i=1}^n (w_i * x_i) + b, \quad (2.2)$$

де W — Зважена сума вхідних даних;

b — зміщення;

n — загальна кількість вхідних з'єднань;

w_i — вага для i -го входу;

x_i — i -те вхідне значення.

5) зважена сума потім пропускається через функцію активації, що позначається як f . Функція активації вводить нелінійність у мережи, дозволяючи їй навчатися та представляти складні взаємозв'язки в даних.

Функція активації визначає вихідний діапазон нейрона та його поведінку у відповідь на різні вхідні значення. Вибір функції активації залежить від характеру завдання та бажаних властивостей мережі;

в) вихідний шар;

1) вихідний шар MLP створює кінцеві прогнози або виходи мережі. Кількість нейронів у вихідному шарі залежить від виконуваного завдання;

2) кожен нейрон у вихідному шарі отримує вхідний сигнал від нейронів останнього прихованого шару та застосовує функцію активації. Ця функція активації зазвичай відрізняється від тих, що використовуються в прихованих шарах, і створює кінцеве вихідне значення або прогноз;

3) після того, як мережа згенерує вихідний сигнал, наступним кроком є обчислення втрат за допомогою функції втрат. Для задачі класифікації зазвичай використовується бінарна функція втрат крос-ентропії, що обчислюється за формулою (2.3):

$$L = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\tilde{y}_i) + (1 - y_i) \log(1 - \tilde{y}_i)] \quad (2.3)$$

де W — Зважена сума вхідних даних;

y_i — фактична мітка;

$\tilde{y}_i$ — передбачувана мітка;

N — кількість зразків [37].

YOLO (You Only Look Once) - популярна модель виявлення об'єктів, відома своєю швидкістю та точністю. Модель пропонує використовувати наскрізну нейронну мережу, яка одночасно прогнозує обмежувальні рамки та ймовірності класів. Хоча алгоритми, такі як Faster RCNN, працюють, виявляючи можливі області інтересу за допомогою мережі пропозицій регіонів (Region Proposal Network), а потім виконуючи розпізнавання цих областей окремо, YOLO виконує всі свої прогнози за допомогою одного повністю зв'язаного шару. Методи, що використовують мережі пропозицій регіонів, виконують кілька ітерацій для одного

й того ж зображення, тоді як YOLO обходиться однією ітерацією [38]. Структуру даної моделі можна побачити на рисунку 2.3.

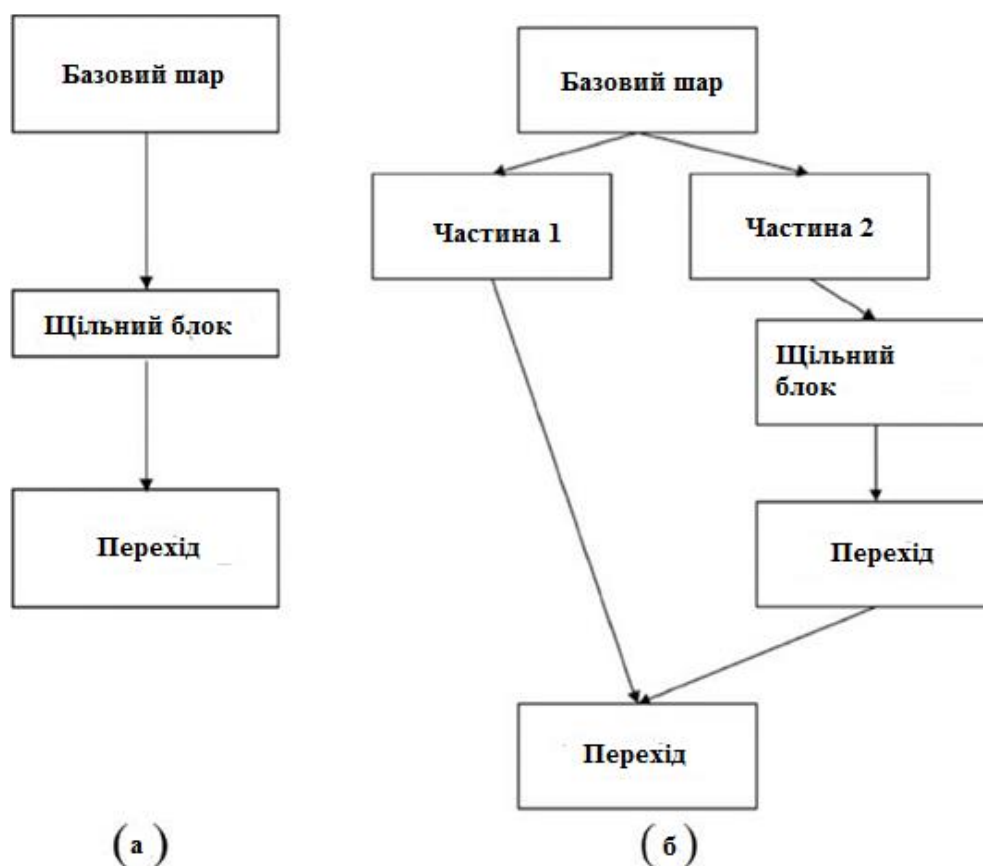


Рисунок 2.3 – Структура YOLO[39]

Алгоритм YOLO приймає зображення як вхідні дані, а потім використовує просту глибоку згорткову нейронну мережу для виявлення об'єктів на зображенні.

Алгоритм роботи моделі YOLO:

а) перші 20 шарів згортки моделі попередньо навчаються за допомогою ImageNet шляхом додавання тимчасового об'єднання середніх значень та повністю зв'язного шару;

б) потім ця попередньо навчена модель перетворюється для виконання виявлення, оскільки попередні дослідження показали, що додавання шарів згортки та зв'язних шарів до попередньо навченої мережі покращує продуктивність.

Останній повністю зв'язний шар YOLO прогнозує як ймовірності класів, так і координати обмежувальної рамки;

в) YOLO розділяє вхідне зображення на сітку $S \times S$. Якщо центр об'єкта потрапляє в комірку сітки, ця комірка відповідає за виявлення цього об'єкта. Кожна комірка сітки передбачає B обмежувальних рамок та показники достовірності для цих рамок. Ці показники достовірності відображають, наскільки впевнена модель, що рамка містить об'єкт, і наскільки точною, на її думку, є передбачена рамка;

г) YOLO прогнозує кілька обмежувальних рамок на комірку сітки. Під час навчання нам потрібен лише один предиктор обмежувальних рамок, який відповідав би за кожен об'єкт. YOLO призначає один предиктор, який буде «відповідальним» за прогнозування об'єкта на основі того, який прогноз має найвищий поточний IOU з базовим істинним значенням. Це призводить до спеціалізації між предикторами обмежувальних рамок. Кожен предиктор стає кращим у прогнозуванні певних розмірів, співвідношень сторін або класів об'єктів, покращуючи загальний показник повноти;

д) одним з ключових методів, що використовуються в моделях YOLO, є немаксимальне придушення (NMS). NMS – це крок постобробки, який використовується для підвищення точності та ефективності виявлення об'єктів. Під час виявлення об'єктів поширеною є генерація кількох обмежувальних рамок для одного об'єкта на зображенні. Ці обмежувальні рамки можуть перекриватися або розташовуватися в різних положеннях, але всі вони представляють один і той самий об'єкт. NMS використовується для ідентифікації та видалення надлишкових або неправильних обмежувальних рамок і для виведення однієї обмежувальної рамки для кожного об'єкта на зображенні;

Обрані моделі мають як свої недоліки так і переваги, детальне порівняння можна побачити в таблиці 2.4

Таблиця 2.4 – Порівняння обраних моделей

	CNN	MLP	YOLO
Тип моделі	Глибока згорткова нейронна мережа	Багатошаровий перцептрон	Мережа для детекції об'єктів
Основна задача	Класифікація зображень	Класифікація	Детекція та локалізація об'єктів
Набір даних, що використовує	CIFAR-10	CIFAR-10	COCO / власні зображення
Архітектура	Згорткові + pooling + fully connected шари	Fully connected шари	CNN backbone + detection head
Кількість шарів	5–20+	3–10	~24 convolutional + 2 fully connected (YOLOv1)
Типи шарів	Convolution, Pooling, Dense	Dense (Fully connected)	Convolution, BatchNorm, Detection layers
Функції активації	ReLU, Softmax	ReLU, Sigmoid, Tanh	Leaky ReLU, Softmax
Механізм виділення ознак	Автоматичне виділення ознак згортками	Ознаки не виділяються автоматично	Виділення ознак через CNN backbone
Кількість параметрів	Середня	Дуже велика	Велика
Швидкість навчання	Середня	Швидка	Повільніша
Швидкість прогнозування	Швидка	Дуже швидка	Дуже швидка
Робота з просторовою інформацією	Так	Ні	Так
Незалежність	Самостійна мережа	Самостійна мережа	Використовує CNN
Основна перевага	Ефективна робота із зображеннями	Простота архітектури	Детекція об'єктів у реальному часі
Основний недолік	Висока обчислювальна складність	Погано працює з зображеннями	Складна архітектура[40]-[44]

Також є ще такі моделі як, наприклад RCNN, але їх не буде використано через ряд недоліків таких як дуже повільна робота, висока обчислювальна складність та велике використання пам'яті [45].

2.4 Висновок до розділу

В результаті проведеної роботи було проаналізовано та обрано найбільш доцільну для використання мову програмування - Python. Розробка продукту спрямованого на вирішення задач комп'ютерного зору неможлива без використання додаткових бібліотек. Для розробки даного програмного забезпечення було обрано такі бібліотеки як TensorFlow, Keras, pandas, Pillow, NumPy, Tkinter. Поєднання у використанні даних бібліотек надає змогу створити програмне забезпечення з використанням моделей комп'ютерного зору для розпізнавання об'єктів на зображеннях.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення

Структура програмного забезпечення, що складається з 6 основних python файлів, полягає в розподіленні функціоналу на окремі модулі, що забезпечує покращення структури коду, спрощення супроводу та повторне використання компонентів. Структуру та її схему можна побачити на рисунку 3.1 її складові, а самі модулі, можна побачити в таблиці 3.1

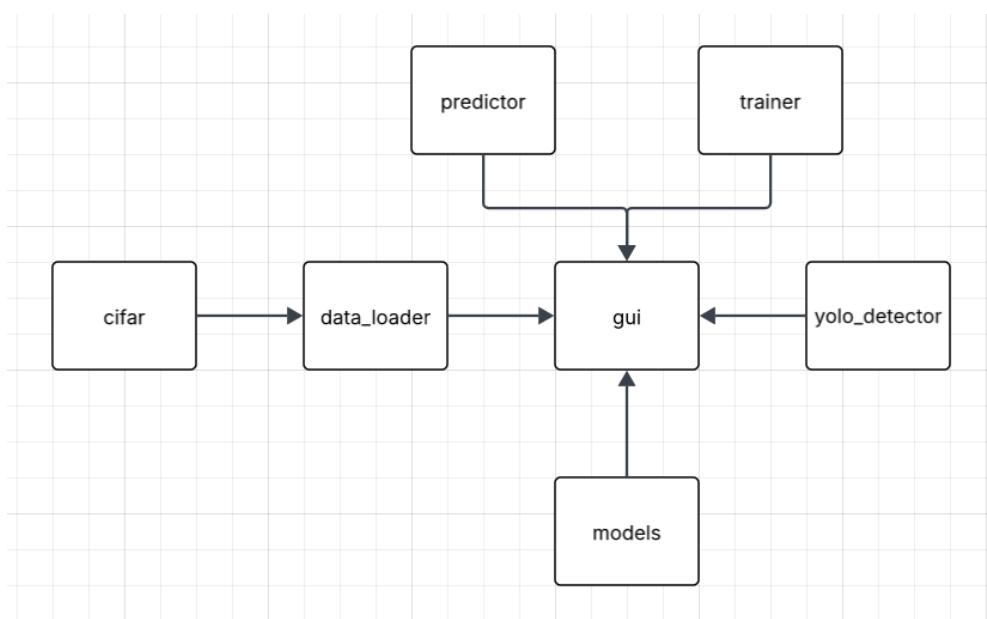


Рисунок 3.1 - Схема структури

Відповідно до рисунку 3.1 можна зробити висновок, що структура застосунку полягає в тому, що модуль графічного інтерфейсу `gui` є серцевим елементом. Він викликає і використовує всі створені раніше модулі для коректної роботи застосунку.

Таблиця 3.1 - Модулі програмного забезпечення

Назва	Призначення
cifar	Завантаження датасету CIFAR-10
data_loader	Завантаження датасетів
gui	Графічний інтерфейс та ініціалізація системи
models	Опис структури моделей CNN та MLP
predictor	Завантаження моделі і прогнозування
trainer	Тренування моделі
yolo_detector	Виклик і виявлення об'єктів за допомогою моделі YOLO

Робота модулів, що б виконували розпізнавання об'єктів за допомогою моделей нейронних мереж неможливе без використання бібліотек. Всі модулі використовували бібліотеку tensorflow. Також в модулях було використано такі бібліотеки як ultralytics для завантаження моделі YOLO, PIL для роботи з зображеннями, numpy для роботи з масивами даних, tkinter для графічного інтерфейсу, os для роботи з операційною системою і файлами, pickle для збереження і подальшого використання збереженої натренованої моделі та threading для багатопоточності.

В результаті користувач отримує повноцінний застосунок, що надає змогу тренувати, зберігати і використовувати моделі навіть без інтернету для подальшого їх використання з метою розпізнавання об'єктів на зображення.

3.2 Функціонування програмного забезпечення розпізнавання об'єктів на зображеннях

Щоб краще зрозуміти як відбувається взаємодія між користувачем і програмним забезпеченням було створено схему, що зображена на рисунку 3.2

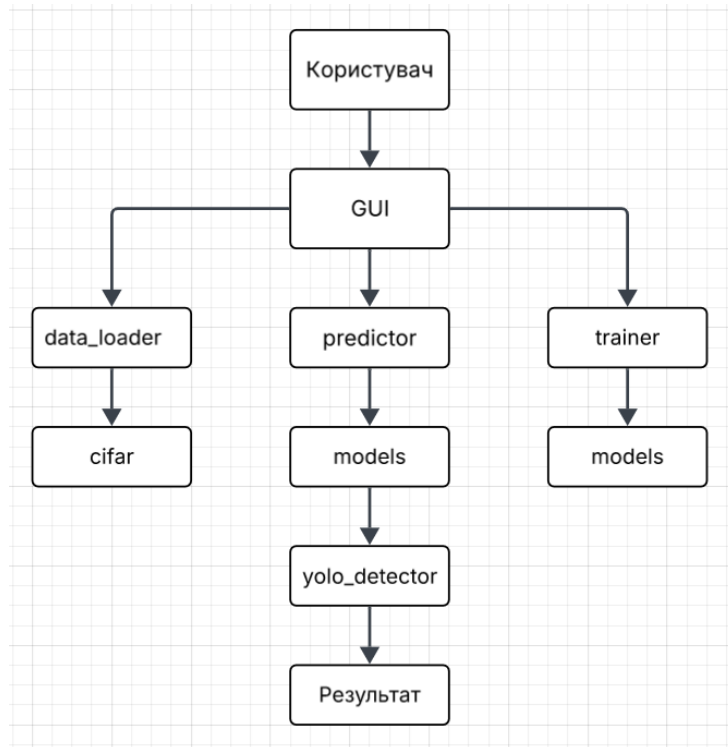


Рисунок 3.2 - Схема функціонування

Як можна побачити зі схеми на рисунку 3.2 функціонування застосунку відбувається наступним чином:

- а) користувач взаємодіє з графічним інтерфейсом;
- б) завантаження даних;
 - 1) модуль data_loader отримує дані з модулю cifar;
 - 2) виконується підготовка даних;
- в) режим навчання;
 - 1) модуль trainer навчає модель;
 - 2) результат зберігається у models;
- г) режим розпізнавання;
 - 1) модуль predictor використовує модель з models;
 - 2) або викликається yolo_detector, за умови його обрання користувачем, для детекції об'єктів);
- д) обробка результату;
 - 1) результат повертається в GUI модуль;
 - 2) результат відображається користувачу.

3.3 Розробка інтерфейсу застосунку

Як вже було згадано раніше, в розділі 3.1, інтерфейс забезпечується за допомогою модуля `gui.py` та бібліотеки `tkinter`. Розроблене програмне забезпечення не вимагає від розробника створення масштабного і складного інтерфейсу. Він має відповідати наступним вимогам:

- бути простим і зрозумілим;
- надавати користувачу можливість обирати набір даних і модель;
- надавати користувачу можливість завантажити свій набір даних;
- надавати користувачу можливість тренувати і завантажувати модель;
- надавати користувачу можливість завантажувати зображення;
- надавати користувачу можливість запустити розпізнавання;
- виводити результат розпізнавання об'єктів.

Ці вимоги забезпечують користувачу комфортну взаємодію із застосунком. Інтерфейс не містить зайвих елементів, а отже не відволікає від роботи та надає змогу виконати призначення програмного забезпечення – розпізнати об'єкт на зображенні за допомогою однієї з запропонованих моделей. Відповідно до вище описаних вимог інтерфейс містить:

- головне вікно – екран розміром 420 пікселів в ширину і 440 пікселів у висоту;
- помітки – підписи деяких структурних елементів;
- кнопки для тренування, збереження та передбачення моделі – прямокутні стандартні кнопки розміром від 12 до 18 символів, що запускають виконання тренування, збереження або передбачення;
- радіо кнопки для вибору моделі та датасету – радіо кнопки, що забезпечують вибір тільки 1 елемента з переліку, моделей чи датасетів у даному випадку.

Побачити, розроблений інтерфейс можна на рисунку 3.3.

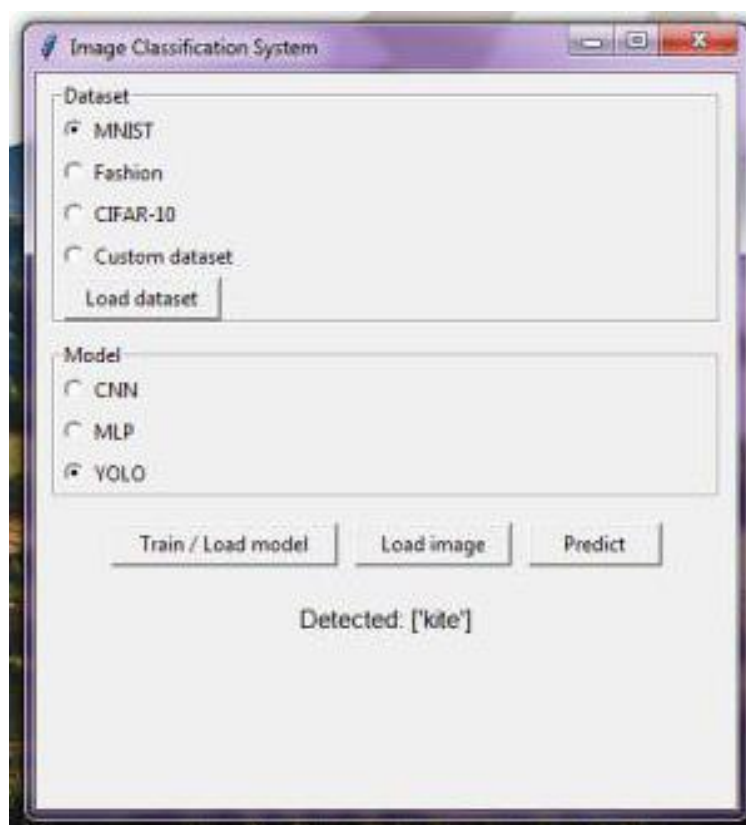


Рисунок 3.3 - Інтерфейс застосунку

3.4 Висновок до розділу

В результаті проведеної в даному розділі роботи було створено програмне забезпечення починаючи від визначення його структури і закінчуючи його функціонуванням та інтерфейсом. На етапі розробки структури було вирішено обрати модульну структуру через ряд переваг вказаних в розділі 3.1. Наступним кроком було розписано функціонування користувача з застосунком, а саме з кожним модулем. Відповідно до схем структури та функціонування було розроблено та поєднано між собою програмні python модулі. Наступним і фінальним кроком було розроблення інтерфейсу відповідно до вимог і модулів.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програмного забезпечення

Застосунок Image Classification System призначений для використання певних нейронних мереж щоб використати їх для розпізнавання об'єктів на зображеннях після попереднього їх навчання на обраних користувачем наборах даних. Крім прямого призначення застосунок також має і навчальне – можливість користувача побачити як відбувається тренування нейромоделі. Розроблене програмне забезпечення дозволяє виконувати наступні функціональні можливості:

- обрання набору даних для тренування моделі;
- завантаження персонального набору даних;
- обрання моделі нейромереж;
- тренування моделі на обраному наборі даних;
- збереження моделі(повністю натренованої і частково);
- завантаження зображення;
- розпізнавання об'єкту на фото.

Для коректної роботи застосунку програмне забезпечення має задовольняти умови:

- операційна система Windows7 і вище;
- Python версія 3.8.6 і вище;
- оперативна пам'ять від 4Гб;
- об'єм вільного місця на жорсткому диску від 1Гб;
- процесор від 3.30 ГГц.

4.2 Характеристики програмного забезпечення

Image Classification System – це застосунок, що містить одну вкладку, розроблену за допомогою високорівневої мови програмування Python і бібліотеки для графічного інтерфейсу tkinter. На цій вкладці відбувається вся послідовність

процесів необхідна для роботи застосунку – від обрання набору даних до запуску розпізнавання об’єкту. Бібліотеки TensorFlow, Keras та pandas забезпечують безперебійну роботу нейронних мереж, а бібліотека Pillow надає можливість обробки зображень.

4.3 Інструкція з використання програми

4.3.1 Звернення до програми

Для запуску роботи програми необхідно відкрити в будь-якому Python IDE файл `gui.py` та запустити його. Результатом буде відкрита вкладка застосунку.

4.3.2 Вхідні дані

Вхідними даними програмного забезпечення є зображення у форматах `jpg`, `png` тощо, що представлене у вигляді багатовимірного масиву пікселів.

Вхідними даними програмного забезпечення є вектор ймовірностей належності вхідного зображення до кожного з класів, а також визначений клас з максимальною ймовірністю.

4.3.3 Повідомлення

В процесі взаємодії з застосунком користувач може отримати повідомлення про ненатреновану модель, не завантажене фото для розпізнавання об’єкту на ньому та про необхідність обрання папки з персональним набором даних.

4.4 Виконання програмного забезпечення

Після запуску файлу `gui.py` відкривається головна і єдина вкладка застосунку, що зображена на рисунку 4.1

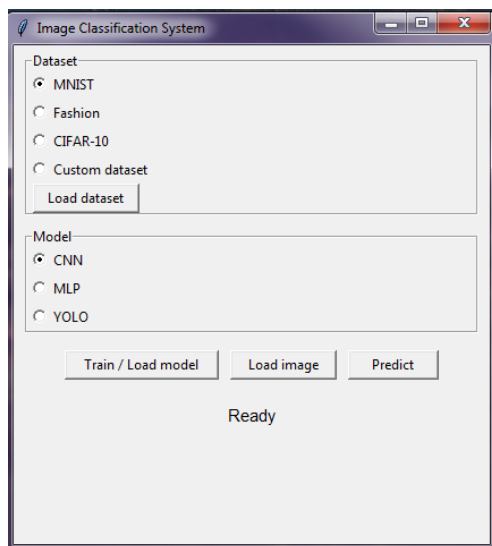


Рисунок 4.1 – Головна вкладка

Першим, що бачить користувач – набір датасетів, з яких необхідно обрати один, шляхом кліку на радіо кнопку. Наступним йде блок з назвами нейромоделей, де також необхідно обрати один варіант, як і у випадку датасетів. Після вибору датасету і моделі вікно виглядатиме як показано на рисунку 4.2.

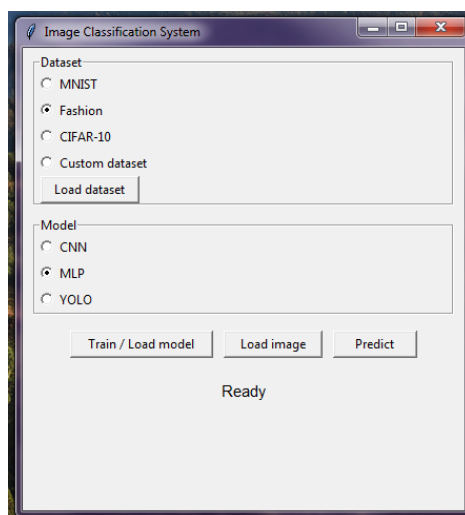


Рисунок 4.2 – Обрані датасет і модель

У випадку завантаження персонального датасету необхідно натиснути на кнопку «Load dataset» результатом чого буде вікно обрання папки з набором даних, як можна побачити на рисунку 4.3

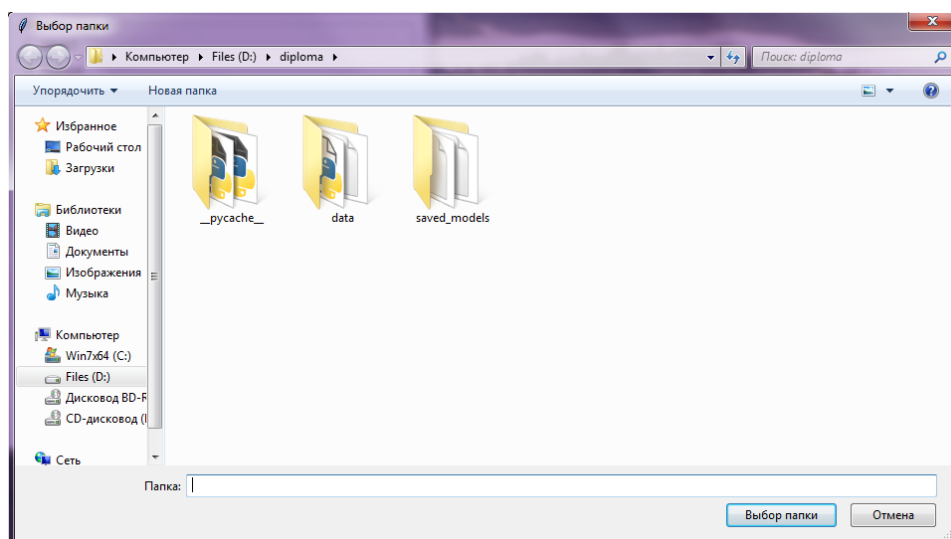


Рисунок 4.3 – Обрання персонального датасету

Наступний крок взаємодії з програмою – тренування або завантаження моделі, після завершення процесу користувача буде повідомлено, як можна побачити на рисунку 4.4

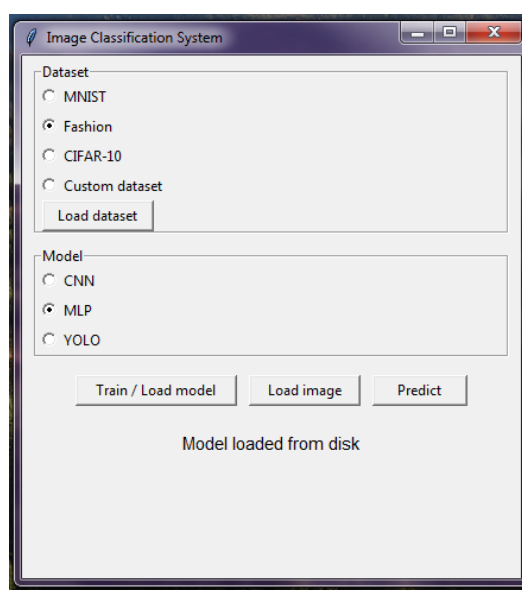


Рисунок 4.4 – Результат тренування/завантаження моделі

Після тренування або завантаження моделі необхідно завантажити фото, що робиться за допомогою кліку на кнопку «Load image», що відкриє вікно для вибору зображення, як можна побачити на рисунку 4.5. Після вибору зображення користувач побачить відповідне повідомлення, як показано на рисунку 4.6

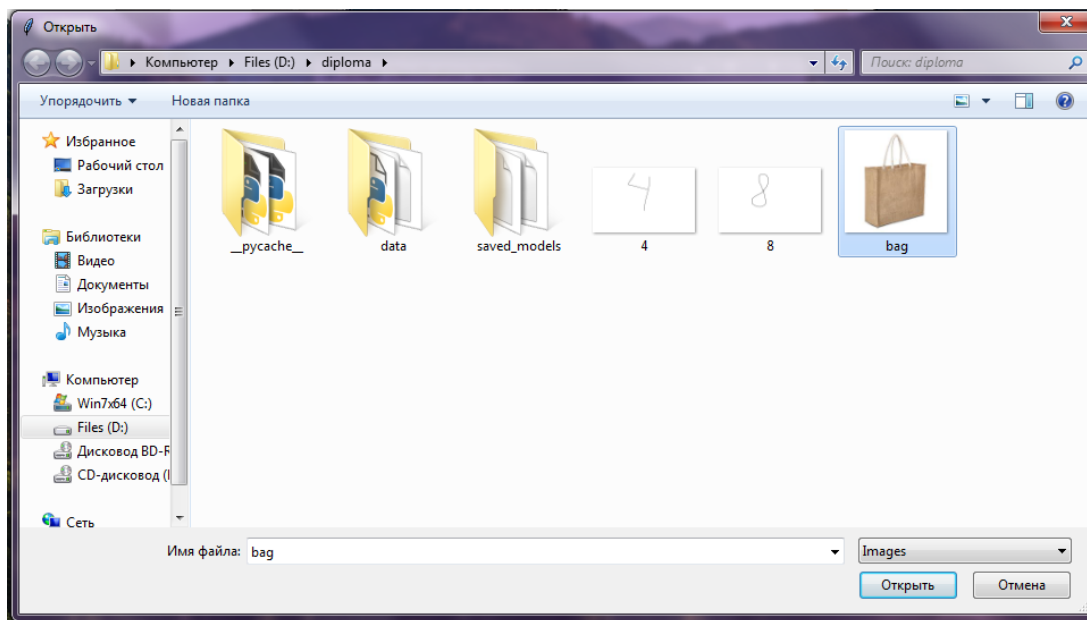


Рисунок 4.5 – Вибір зображення для розпізнавання

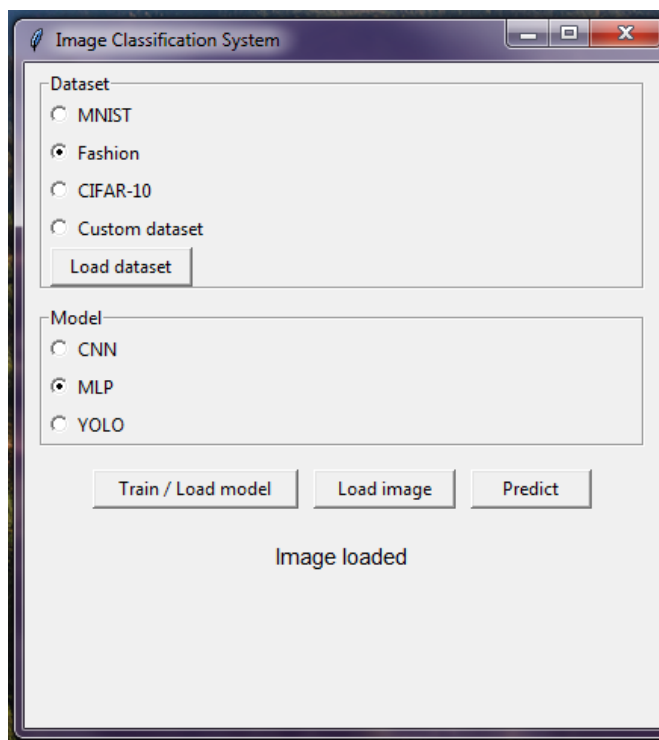


Рисунок 4.6 – Успішне завантаження зображення

Останнім кроком є натискання на кнопку «Predict» щоб запустити і отримати результати передбачення. Результат буде виглядати як на рисунку 4.7. Результатом є 8, відповідно до розшифрування в таблиці 4.1, це сумка, що є правильною відповіддю.

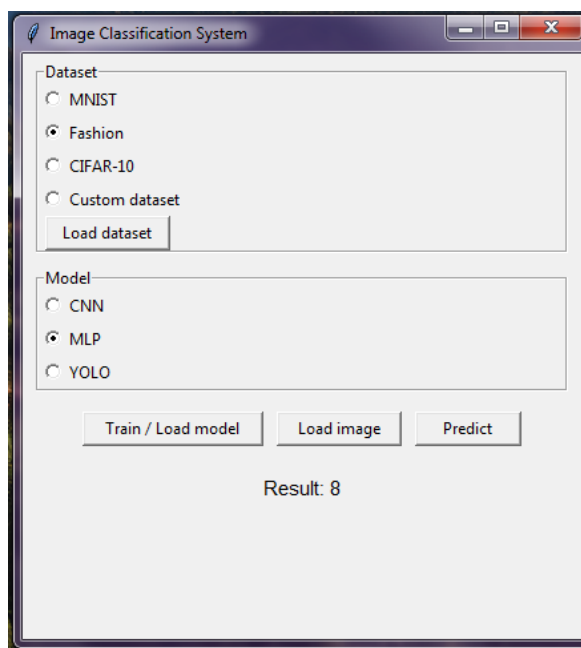


Рисунок 4.7 – Результат розпізнавання

Таблиця 4.1 – Розшифрування міток для набору даних MNIST Fashion

Номер	Клас
0	T-shirt/top (футболка)
1	Trouser (штани)
2	Pullover (свєтр)
3	Dress (сукня)
4	Coat (пальто)
5	Sandal (сандалі)
6	Shirt (сорочка)
7	Sneaker (кросівки)
8	Bag (сумка)
9	Ankle boot (черевики)

4.5 Тестування програмного забезпечення

Останнім етапом було проведено тестування програмного забезпечення і проаналізовано результати.

Виявлено, що програма розпізнавання об'єктів на зображенні є періодично неточною. Причиною неточності може бути те, що тренування відбувається лише 3 епохи, що є доволі невеликим значенням але через слабе програмне забезпечення невелика кількість епох було необхідною мірою. Збільшення епох навчання може призвести до покращення точності програмного забезпечення.

4.6 Висновок до розділу

Результатом даного розділу є опис роботи, вимоги до програмного забезпечення та інструкція взаємодії з застосунком. Також було проведено тестування в процесі якого було знайдено недоліки програми, що пов'язані з технічною складовою програмного забезпечення, на якому запущено застосунок.

ВИСНОВКИ

В результаті дипломного проєкту бакалавра було розроблено програмне забезпечення Image Classification System, що призначене для розпізнавання об'єктів на зображеннях за допомогою нейромережових моделей.

Першим кроком до розробки став аналіз предметної області, що включав в себе перш за все ознайомлення з комп'ютерним зором та детальне порівняння машинного та глибокого навчання. Після їх порівняння було взято до уваги, що для розпізнавання об'єктів на зображеннях краще підійде глибоке навчання. Далі аналіз містив розбір проблематики предметної області. Ще одним кроком було детальне ознайомлення з вже наявними аналогами, які пізніше було детально порівняно один з одним. В результаті аналізу проблемної області було визначено головне завдання роботи та головну перевагу над аналогами – незалежність від Інтернету.

Наступним етапом у виконанні дипломної роботи було визначення матеріалів і методів розробки. Першим необхідним кроком був вибір мови програмування, для цього було обрано найбільш доцільно використовувані Python, Rust чи C++. В результаті детального порівняння було обрано найкращий варіант - високорівнева мова програмування Python, що найкраще підходить для завдання розпізнавання об'єктів. Відразу після мов було обрано бібліотеки, адже без бібліотек програмна реалізація відповідно до завдання неможлива. Після обрання мови й бібліотек необхідно обрати середовище розробки. Оскільки завдання не вимагає надскладних реалізацій як середовище розробки було обрано надпростий IDLE. На завершення було проведено доволі обширний аналіз нейромережових моделей CNN, MLP та YOLO, їх характеристик, особливостей та здібностей.

Третій етап виконання завдання дипломної роботи - безпосередня розробка і опис розробленого. Першим кроком було визначено структуру програмного забезпечення, яка використовує модульний підхід і містить 6 модулів. Наступний крок - опис функціонування застосунку, що включає користувача і його взаємодію

з програмою за допомогою графічного інтерфейсу. Завершує третій етап опис вимог до графічного інтерфейсу та опис вже розробленого інтерфейсу.

Четвертим і останнім етапом є етап, що містить призначення, застосування і технічні вимоги до програмного забезпечення на якому буде запущено застосунок. Вимоги є доволі незначними і охоплюють навіть доволі слабкі комп'ютери та ноутбуки. Також етап містить опис програмного забезпечення, та інструкцію і можливі сценарії його використання програми. Завершує етап тестування програмного забезпечення, в результаті якого було визначено, що кількість епох для навчання недостатня для високого рівня точності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Топ технологій розпізнавання зображень у 2025. URL:
<https://wezom.com.ua/ua/blog/top-tehnologiy-rozpiznavannya-zobrazhen-u-2025> (дата звернення: 16.04.2026).
2. Виявлення об'єктів. URL:
https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D1%8F%D0%B2%D0%BB%D1%8F%D0%BD%D0%BD%D1%8F_%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D1%96%D0%B2 (дата звернення: 16.04.2026).
3. Технології комп'ютерного зору. URL:
<https://metinvest.digital/ua/page/1028> (дата звернення: 16.04.2026).
4. Штучний інтелект для розпізнавання зображень: що це таке, як він працює та приклади. URL: <https://uk.shaip.com/blog/what-is-ai-image-recognition-and-how-does-it-work/> (дата звернення: 16.04.2026).
5. An In-Depth Comparison Machine Learning and Deep Learning. URL:
<https://viso.ai/deep-learning/deep-learning-vs-machine-learning/> (дата звернення: 16.04.2026).
6. Convolutional Neural Networks, Explained. URL:
<https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939/> (дата звернення: 16.04.2026).
7. Object Recognition. 3 things you need to know. URL:
<https://www.mathworks.com/solutions/image-processing-computer-vision/object-recognition.html> (дата звернення: 16.04.2026).
8. Teachable Machine. URL: <https://teachablemachine.withgoogle.com/> (дата звернення: 16.04.2026).
9. Teachable Machine URL: <https://teachablemachine.withgoogle.com/> (дата звернення: 16.04.2026).
10. MLOps. URL: <https://uk.wikipedia.org/wiki/MLOps> (дата звернення: 16.04.2026).
11. Meet Edge Impulse Studio: Easily Build and Deploy Edge AI Models. URL:

<https://www.elektormagazine.com/articles/meet-edge-impulse-studio-easily-build-deploy-edge-ai-models> (дата звернення: 16.04.2026).

12. Edgeimpulse | Seeed Studio Wiki URL:

<https://wiki.seeedstudio.com/edgeimpulse/> (дата звернення: 16.04.2026).

13. Roboflow. URL: <https://roboflow.com/> (дата звернення: 16.04.2026).

14. Getting Started with Roboflow URL: <https://blog.roboflow.com/getting-started-with-roboflow/> (дата звернення: 16.04.2026).

15. LabelImg for Image Annotation. URL: <https://viso.ai/computer-vision/labelimg-for-image-annotation/> (дата звернення: 16.04.2026).

16. GitHub - HumanSignal/labelImg: LabelImg is now part of the Label Studio community. The popular image annotation tool created by Tzutalin URL: <https://github.com/HumanSignal/labelImg> (дата звернення: 16.04.2026).

17. TensorBoard: TensorFlow's visualization toolkit. URL: <https://www.tensorflow.org/tensorboard> (дата звернення: 16.04.2026).

18. TensorBoard - Weights & Biases Documentation URL: <https://docs.wandb.ai/models/integrations/tensorboard> (дата звернення: 16.04.2026).

19. Python. URL: <https://uk.wikipedia.org/wiki/Python> (дата звернення: 16.04.2026).

20. Rust (мова програмування). URL: [https://uk.wikipedia.org/wiki/Rust_\(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F\)](https://uk.wikipedia.org/wiki/Rust_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) (дата звернення: 16.04.2026).

21. C++. URL: <https://uk.wikipedia.org/wiki/C%2B%2B> (дата звернення: 16.04.2026).

22. Python 3.14.3 documentation. URL: <https://docs.python.org/3/> (дата звернення: 16.04.2026).

23. The Rust Programming Language. URL: <https://doc.rust-lang.org/book/> (дата звернення: 16.04.2026).

24. News, Status & Discussion about Standard C++. URL: <https://isocpp.org/> (дата звернення: 16.04.2026).

25. Learn Rust. URL: <https://rust-lang.org/learn/> (дата звернення: 16.04.2026).
26. GIL. URL: <https://wiki.python.org/moin/GlobalInterpreterLock> (дата звернення: 16.04.2026).
27. IDLE — Python editor and shell. URL: <https://docs.python.org/3/library/idle.html> (дата звернення: 16.04.2026).
28. PyCharm. URL: <https://www.jetbrains.com/pycharm/> (дата звернення: 16.04.2026).
29. The open source AI code editor. URL: <https://code.visualstudio.com/> (дата звернення: 16.04.2026).
30. Convolutional Neural Network (CNN) in Tensorflow. URL: <https://www.geeksforgeeks.org/deep-learning/convolutional-neural-network-cnn-in-tensorflow/> (дата звернення: 16.04.2026).
31. Schematic diagram of a basic CNN architecture. URL: https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-CNN-architecture_fig2_384565500 (дата звернення: 16.04.2026).
32. Convolutional Neural Network using TensorFlow. URL: <https://medium.com/@BriceNicodem/convolutional-neural-network-using-tensorflow-3d5b5d3a38a5> (дата звернення: 16.04.2026).
33. Convolutional Neural Network (CNN) in Deep Learning. URL: <https://www.geeksforgeeks.org/deep-learning/convolutional-neural-network-cnn-in-machine-learning/> (дата звернення: 16.04.2026).
34. What is a Perceptron? URL: <https://www.freecodecamp.org/news/build-a-multilayer-perceptron-with-examples-and-python-code/#heading-what-is-a-perceptron> (дата звернення: 16.04.2026).
35. Multilayer Perceptrons in Machine Learning: A Comprehensive Guide. URL: <https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning> (дата звернення: 16.04.2026).
36. A Simple overview of Multilayer Perceptron(MLP) URL: <https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/> (дата звернення: 16.04.2026).

37. Multi-Layer Perceptron Learning in Tensorflow. URL:
<https://www.geeksforgeeks.org/deep-learning/multi-layer-perceptron-learning-in-tensorflow/> (дата звернення: 16.04.2026).

38. YOLO: Algorithm for Object Detection Explained. URL:
<https://www.v7labs.com/blog/yolo-object-detection> (дата звернення: 16.04.2026).

39. YOLO URL: https://www.researchgate.net/figure/CSPNet-vs-DenseNet-architecture-utilized-in-YOLO-v4-73-a-CSPNet-b-DenseNet_fig22_362557835 (дата звернення: 16.04.2026).

40. Convolutional neural network. URL:
https://en.wikipedia.org/wiki/Convolutional_neural_network (дата звернення: 16.04.2026).

41. Multilayer perceptron. URL:
https://en.wikipedia.org/wiki/Multilayer_perceptron (дата звернення: 16.04.2026).

42. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. URL: <https://www.mdpi.com/2079-3197/11/3/52> (дата звернення: 16.04.2026).

43. Redmon J., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. URL: <https://arxiv.org/abs/1506.02640> (дата звернення: 16.04.2026).

44. Acharya U., et al. Deep Convolutional Neural Networks for Image Classification. URL: https://www.researchgate.net/publication/317496930_Deep_Convolutional_Neural_Networks_for_Image_Classification_A_Comprehensive_Review (дата звернення: 16.04.2026).

45. Fast R-CNN. URL: <https://naokishibuya.medium.com/fast-r-cnn-28213957e552> (дата звернення: 16.04.2026).

ДОДАТОК А
Текст програми

A.1 Текст файла `cifar.py`

```
from tensorflow.keras.datasets import cifar10
cifar10.load_data ()
```

A.2 Текст файла `data_loader.py`

```
from tensorflow import keras
from tensorflow.keras import datasets
import numpy as np
import os
import pickle
from keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing import image_dataset_from_directory

def load_mnist():
    input_shape = (28, 28, 1)
    num_classes = 10
    (x_train,y_train),(x_test,y_test) = keras.datasets.mnist.load_data()
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
    x_train, x_test = x_train / 255.0, x_test / 255.0
    return x_train, y_train, x_test, y_test, input_shape, num_classes

BASE_DIR = os.path.dirname(os.path.abspath(__file__))

def load_fashion_mnist():
    path = os.path.join(BASE_DIR, "data", "fashion-mnist.npz")

    if not os.path.exists(path):
        raise FileNotFoundError(f"Fashion-MNIST not found at:\n{path}")

    data = np.load(path)

    x_train = data["x_train"]
    y_train = data["y_train"]
    x_test = data["x_test"]
    y_test = data["y_test"]

    input_shape = (28, 28, 1)
```

```

num_classes = 10

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

x_train, x_test = x_train / 255.0, x_test / 255.0

return x_train, y_train, x_test, y_test, input_shape, num_classes

def load_cifar10():
    path = os.path.join(BASE_DIR, "data", "cifar-10", "cifar-10-batches-py")

    if not os.path.exists(path):
        raise FileNotFoundError(f"CIFAR-10 not found at:\n{path}")

    def load_batch(file):
        with open(file, 'rb') as f:
            data = pickle.load(f, encoding='bytes')
            return data[b'data'], data[b'labels']

    x_train, y_train = [], []

    for i in range(1, 6):
        batch_file = os.path.join(path, f"data_batch_{i}")
        if not os.path.exists(batch_file):
            raise FileNotFoundError(f"Missing file: {batch_file}")

        data, labels = load_batch(batch_file)
        x_train.append(data)
        y_train.extend(labels)

    x_train = np.concatenate(x_train).reshape(-1, 3, 32, 32)
    x_train = x_train.transpose(0, 2, 3, 1)

    test_data, test_labels = load_batch(os.path.join(path, "test_batch"))
    x_test = test_data.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)

    x_train = x_train / 255.0
    x_test = x_test / 255.0

    input_shape = (32, 32, 3)
    num_classes = 10
    name_classes = [
        "Airplane", "Automobile", "Bird", "Cat", "Deer",

```

```

        "Dog", "Frog", "Horse", "Ship", "Truck"
    ]

    return x_train, np.array(y_train), x_test, np.array(test_labels), input_shape,
    num_classes, name_classes

def load_custom_dataset(path, img_size=(64,64), batch_size=32):

    train_ds = image_dataset_from_directory(
        path,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=img_size,
        batch_size=batch_size
    )

    test_ds = image_dataset_from_directory (
        path,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=img_size,
        batch_size=batch_size
    )

    class_names = train_ds.class_names
    num_classes = len(class_names)
    input_shape = (img_size[0], img_size[1], 3)

    train_images, train_labels = [], []
    for x, y in train_ds:
        train_images.append(x.numpy())
        train_labels.append(y.numpy())

    test_images, test_labels = [], []
    for x, y in test_ds:
        test_images.append(x.numpy())
        test_labels.append(y.numpy())

    x_train = np.concatenate(train_images).astype("float32") / 255.0
    y_train = np.concatenate(train_labels)

    x_test = np.concatenate(test_images).astype("float32") / 255.0

```

```
y_test = np.concatenate(test_labels)
```

```
return x_train, y_train, x_test, y_test, input_shape, num_classes, class_names
```

A.3 Текст файла gui.py

```
import os
import threading
import tensorflow as tf
import tkinter as tk
from tkinter import filedialog, messagebox
from data_loader import load_mnist, load_fashion_mnist, load_cifar10
from models import cnn, mlp
from trainer import compiling, training, save_model
from predictor import preprocess_image, predict_image
from data_loader import load_custom_dataset
from yolo_detector import detect_objects

current_model = None
image_path = None
input_shape = None
name_classes = None
is_training = False

def get_model_path():
    dataset_name = dataset.get().lower()
    model_name = model_var.get().lower()
    os.makedirs("saved_models", exist_ok=True)
    return f"saved_models/{dataset_name}_{model_name}.keras"

def select_custom_dataset():
    global dataset_path
    dataset_path = filedialog.askdirectory()
    if dataset_path:
        set_status("Dataset selected")

def set_status(text):
    result_var.set(text)
    root.update_idletasks()

def detect_yolo():
    if image_path is None:
        result_var.set("Load image first")
```

```

    return

results = detect_objects(image_path)
result_var.set(f"Detected: {results}")

def train():
    global is_training
    if is_training:
        messagebox.showinfo("Training", "Training is already running")
    return

threading.Thread(target=train_worker, daemon=True).start()

def train_worker():
    global current_model, input_shape, name_classes, is_training
    is_training = True
    set_status("Preparing data...")

    try:
        model_path = get_model_path()

        if os.path.exists(model_path):
            current_model = tf.keras.models.load_model(model_path)

            if dataset.get() in ["MNIST", "Fashion"]:
                input_shape = (28, 28, 1)
                name_classes = None
            else:
                input_shape = (32, 32, 3)
                name_classes = [
                    "Airplane", "Automobile", "Bird", "Cat", "Deer",
                    "Dog", "Frog", "Horse", "Ship", "Truck"
                ]

            set_status("Model loaded from disk")
            is_training = False
            return

        if dataset.get() == "MNIST":
            x_train, y_train, x_test, y_test, input_shape, num_classes = load_mnist()
            name_classes = None

        elif dataset.get() == "Fashion":
            x_train, y_train, x_test, y_test, input_shape, num_classes = load_fashion_mnist()

```

```

name_classes = None

elif dataset.get() == "CIFAR-10":
    x_train, y_train, x_test, y_test, input_shape, num_classes, name_classes =
load_cifar10()

elif dataset.get() == "custom":
    if not dataset_path:
        raise ValueError("Select custom dataset folder first")
    x_train, y_train, x_test, y_test, input_shape, num_classes, name_classes =
load_custom_dataset(dataset_path)

else:
    raise ValueError("Unknown dataset")

set_status("Building model...")

if model_var.get() == "YOLO":
    set_status("YOLO does not require training. Use Detect button.")
    is_training = False
    return

if model_var.get() == "CNN":
    model, model_type = cnn(input_shape, num_classes)
elif model_var.get() == "MLP":
    model, model_type = mlp(input_shape, num_classes)
else:
    raise ValueError("Unknown model")

model = compiling(model, input_shape)

set_status("Training model...")

training(x_train, y_train, x_test, y_test, model, model_type)

save_model(model, model_path)
current_model = model

set_status("Model trained and saved successfully")

except Exception as e:
    messagebox.showerror("Error", str(e))
    set_status("Error occurred")

```

```

finally:
    is_training = False

def load_image():
    global image_path
    image_path = filedialog.askopenfilename(
        filetypes=[("Images", "*.png *.jpg *.jpeg")]
    )
    if image_path:
        set_status("Image loaded")

def predict():
    if model_var.get() == "YOLO":
        detect_yolo()
        return

    if current_model is None:
        set_status("Train or load model first")
        return

    if image_path is None:
        set_status("Load image first")
        return

    img = preprocess_image(image_path, input_shape)
    result = predict_image(current_model, img, name_classes)
    set_status(f"Result: {result}")

root = tk.Tk()
root.geometry("420x440")
root.title("Image Classification System")

dataset_frame = tk.LabelFrame(root, text="Dataset")
dataset_frame.pack(fill="x", padx=10, pady=5)

dataset = tk.StringVar(value="MNIST")
tk.Radiobutton(dataset_frame, text="MNIST", variable=dataset,
value="MNIST").pack(anchor="w")
tk.Radiobutton(dataset_frame, text="Fashion", variable=dataset,
value="Fashion").pack(anchor="w")
tk.Radiobutton(dataset_frame, text="CIFAR-10", variable=dataset, value="CIFAR-
10").pack(anchor="w")
tk.Radiobutton(dataset_frame, text="Custom dataset", variable=dataset,
value="custom").pack(anchor="w")

```

```

tk.Button(dataset_frame, text="Load dataset", width=12,
command=select_custom_dataset).pack(side="left", padx=5)

model_frame = tk.LabelFrame(root, text="Model")
model_frame.pack(fill="x", padx=10, pady=5)

model_var = tk.StringVar(value="CNN")
tk.Radiobutton(model_frame, text="CNN", variable=model_var,
value="CNN").pack(anchor="w")
tk.Radiobutton(model_frame, text="MLP", variable=model_var,
value="MLP").pack(anchor="w")
tk.Radiobutton(model_frame, text="YOLO", variable=model_var,
value="YOLO").pack(anchor="w")

button_frame = tk.Frame(root)
button_frame.pack(pady=10)

tk.Button(button_frame, text="Train / Load model", width=18,
command=train).pack(side="left", padx=5)
tk.Button(button_frame, text="Load image", width=12,
command=load_image).pack(side="left", padx=5)
tk.Button(button_frame, text="Predict", width=10, command=predict).pack(side="left",
padx=5)

result_var = tk.StringVar(value="Ready")
label_output = tk.Label(root, textvariable=result_var, font=("Helvetica", 11),
wraplength=380)
label_output.pack(pady=10)

root.mainloop()

```

A.4 Текст файла models.py

```

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

def cnn(input_shape, num_classes):
    model_type = "cnn"
    model = Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D(pool_size=(2, 2)),

```

```

        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model, model_type

def mlp(input_shape, num_classes):
    model_type = "mlp"
    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(num_classes, activation='softmax'),
    ])
    return model, model_type

```

A.5 Текст файла predictor.py

```

import numpy as np
from tensorflow.keras.models import load_model
from PIL import Image

def load_trained_model(model_path):
    model = load_model(model_path)
    return model

def preprocess_image(image_path, input_shape):
    img = Image.open(image_path)

    if input_shape == (32, 32, 3):
        img = img.convert("RGB")
        img = img.resize((32, 32))
        img_array = np.array(img)
    else:
        img = img.convert("L")
        img = img.resize((28, 28))
        img_array = np.array(img)
        img_array = img_array.reshape(28, 28, 1)

    img_array = img_array.astype("float32") / 255.0

```

```
img_array = np.expand_dims(img_array, axis=0)

return img_array
```

```
def predict_image(model, image_array, class_names=None):
    predictions = model.predict(image_array)
    predicted_class = np.argmax(predictions)

    if class_names:
        return class_names[predicted_class]
    else:
        return predicted_class
```

A.6 Текст файла `trainer.py`

```
import tensorflow as tf

def compiling(model, input_shape):
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

def training(x_train, y_train, x_test, y_test, model, model_type):
    if model_type == "cnn":
        history = model.fit(
            x_train, y_train,
            epochs=3,
            batch_size=64,
            validation_data=(x_test, y_test)
        )

    elif model_type == "mlp":
        history = model.fit(
            x_train, y_train,
            epochs=3,
            batch_size=64,
            validation_split=0.2
        )
```

```
return history
```

```
def save_model(model, path):  
    model.save(path)
```

A.7 Текст файла yolo_detector.py

```
from ultralytics import YOLO  
  
model = YOLO("yolov8n.pt")  
  
def detect_objects(image_path):  
    results = model(image_path)  
    names = model.names  
  
    detected = []  
    for r in results:  
        for c in r.names:  
            detected.append(names[int(c)])  
  
    return detected
```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
Кафедра програмних засобів
Дипломна робота на тему:

РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ДЛЯ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

Виконала студентка групи
КНТ-212
Керівник
д.т.н., професор

Ірина ПАНЧЕНКО

Андрій ОЛІЙНИК

1

Рисунок Б.1 – Слайд 1

Мета, об'єкт та предмет дослідження

Мета роботи – зменшення кількості часу необхідного для розпізнавання об'єктів на наданому користувачем зображенні, шляхом використання нейромережових моделей за допомогою мови програмування Python та бібліотеки TensorFlow.

Об'єкт дослідження – процес розпізнавання об'єктів на цифрових зображеннях.

Предмет дослідження – методи машинного та глибокого навчання для класифікації та детекції об'єктів на зображеннях.

2

Рисунок Б.2 – Слайд 2

Порівняння машинного та глибокого навчання

	ML	DL
Категорія	Штучний інтелект	Штучний інтелект
Підмножина	Ні	Так, є підмножиною машинного навчання
Залежність	Від вхідних даних для навчання	Від шарів
Навчання	Дозволяє порівняно швидше навчання на основі даних; більше даних дорівнює кращим результатам.	Потрібні інтенсивні обчислення для навчання нейронних мереж з кількома несприятливими та невидимими шарами.
Продуктивність	Деякі форми досягли «плато у продуктивності».	Використання нейронних мереж та наявність надшвидких комп'ютерів прискорили зростання.
Ручне втручання	Щоразу, коли йдеться про нове навчання, розробники-люди повинні втручатися та адаптувати алгоритм для того, щоб навчання відбувалося.	Нейронні мережі сприяють багаторівневному навчанню, де розумні алгоритми можуть навчити машину використовувати знання, отримані з одного шару на наступний, для подальшого навчання без втручання людини.
Навчання	Розробник-людина вказує машині, який тип функції шукати. Вимагає формулювання проблеми, щоб розбити проблему на різні частини, які потрібно буде вирішити згодом, а потім об'єднати результати на завершальному етапі.	Процес вилучення ознак повністю автоматизований. Як результат, вилучення ознак у глибокому навчанні є точнішим та орієнтованим на результат. Вирішуйте проблему від початку до кінця, роблячи процес навчання швидшим та надійнішим.
Дані	Залежить від керованого дослідження вибірок даних, які все ще великі, але порівняно менші.	Оскільки нейронні мережі спираються на багаторівневі знання без втручання людини, для навчання потрібна велика кількість даних.
Точність	Помилки розробника можуть призвести до неправильних рішень та низької точності.	Можливості самонавчання дозволяють досягати швидших і точніших результатів[4].

3

Рисунок Б.3 – Слайд 3

Наявні аналоги

Критерій	Google Teachable Machine	Edge Impulse Studio	RoboFlow	Labelling	TensorBoard
Тип інструменту	Онлайн-платформа для навчання моделей	Платформа для розробки моделей ML для edge-пристроїв	Платформа для підготовки датасетів і навчання моделей	Інструмент для розмітки зображень	Інструмент для візуалізації процесу навчання
Основне призначення	Створення моделей класифікації без програмування	Розробка та оптимізація ML моделей для IoT	Підготовка даних, аугментація та навчання моделей	Розмітка об'єктів на зображеннях	Моніторинг та аналіз навчання нейронних мереж
Тип задач	Класифікація зображень, звуку, поз	Класифікація та детекція	Детекція об'єктів, класифікація	Детекція об'єктів	Аналіз навчання моделей
Необхідність програмування	Ні	Частково	Частково	Ні	Так
Робота з датасетами	Обмежена	Повна підтримка	Розширені можливості	Лише розмітка	Не працює з датасетами
Підтримка нейронних мереж	TensorFlow	TensorFlow, PyTorch	YOLO, TensorFlow, PyTorch	Не підтримує	Підтримує TensorFlow
Можливість тренування моделей	Так	Так	Так	Ні	Ні
Візуалізація результатів	Базова	Детальна	Детальна	Відсутня	Розширена
Робота без Інтернету	Ні	Частково	Ні	Так	Так
Основна перевага	Простота використання	Оптимізація для edge-пристроїв	Потужні інструменти роботи з даними	Простота розмітки	Детальний аналіз навчання
Основний недолік	Обмежена гнучкість	Потребує налаштування	Частинна функцій платна	Немає тренування моделей	Немає навчання моделей

4

Рисунок Б.4 – Слайд 4

Вибір мови програмування

	Python	Rust	C++
Тип мови	Інтерпретована високорівнева мова	Компільована системна мова	Компільована системна мова
Типізація	Динамічна	Статична	Статична
Парадигми програмування	ООП, процедурна, функціональна	Системна, функціональна, процедурна	ООП, процедурна, шаблонна
Швидкість виконання	Низька–середня	Висока	Дуже висока
Управління пам'яттю	Garbage Collector	Ownership + Borrowing	Ручне або smart pointers
Безпека пам'яті	Висока, але з більшими витратами продуктивності	Дуже висока завдяки borrow checker	Низька, можливі помилки типу use-after-free
Багатопоточність	Обмежена (GIL)	Потужна та безпечна	Потужна
Складність навчання	Легка	Складна	Складна
Основні сфери використання	Data Science, AI, Web, automation	Системне ПЗ, backend, embedded	Game dev, системне ПЗ, high-performance
Продуктивність	Низька за компільовані мови	Близька до C++	Дуже висока
Кросплатформність	Висока	Висока	Висока

5

Рисунок Б.5 – Слайд 5

Вибір середовища розробки

Критерій	IDLE	PyCharm	Visual Studio Code
Тип середовища	Просте IDE	Професійне IDE	Редактор коду з розширеннями
Встановлення	Встановлюється разом з Python	Встановлюється окремо	Встановлюється окремо
Споживання ресурсів	Низьке	Високе	Середнє
Складність освоєння	Низька	Середня	Середня
Налагодження	Базове	Розширене	Розширене
Підходить для ML	Так (через бібліотеки)	Так	Так
Використання у великих проєктах	Так але обмежене	Так	Так

6

Рисунок Б.6 – Слайд 6

Вибір нейромережевих моделей для розпізнавання образів

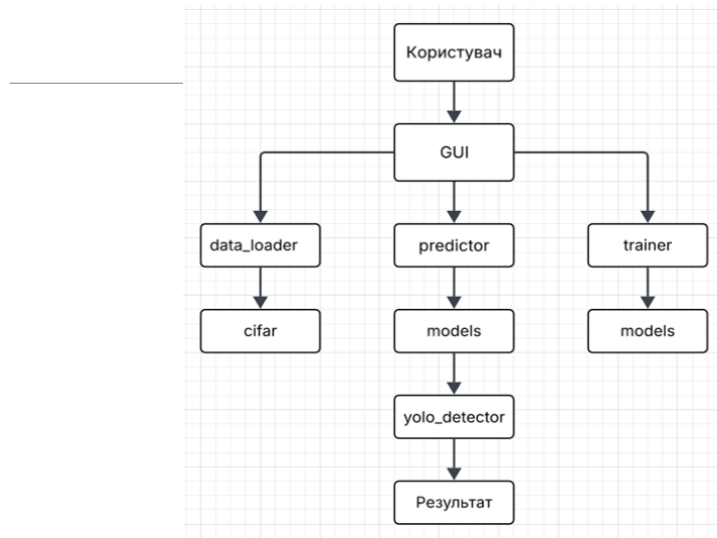
	CNN	MLP	YOLO
Тип моделі	Глибока згорткова нейронна мережа	Багатошаровий перцептрон	Мережа для детекції об'єктів
Основна задача	Класифікація зображень	Класифікація	Детекція та локалізація об'єктів
Набір даних, що використовує	CIFAR-10	CIFAR-10	COCO / власні зображення
Архітектура	Згорткові + pooling + fully connected шари	Fully connected шари	CNN backbone + detection head
Кількість шарів	5–20+	3–10	~24 convolutional + 2 fully connected
Типи шарів	Convolution, Pooling, Dense	Dense (Fully connected)	Convolution, BatchNorm, Detection layers
Функції активації	ReLU, Softmax	ReLU, Sigmoid, Tanh	Leaky ReLU, Softmax
Механізм виділення ознак	Автоматичне виділення ознак згортками	Ознаки не виділяються автоматично	Виділення ознак через CNN backbone
Кількість параметрів	Середня	Дуже велика	Велика
Швидкість навчання	Середня	Швидко	Повільніша
Швидкість прогнозування	Швидко	Дуже швидко	Дуже швидко
Робота з просторовою інформацією	Так	Ні	Так
Незалежність	Самостійна мережа	Самостійна мережа	Використовує CNN
Основна перевага	Ефективна робота із зображеннями	Простота архітектури	Детекція об'єктів у реальному часі
Основний недолік	Висока обчислювальна складність	Погано працює з зображеннями	Складна архітектура

Рисунок Б.7 – Слайд 7



Рисунок Б.8 – Слайд 8

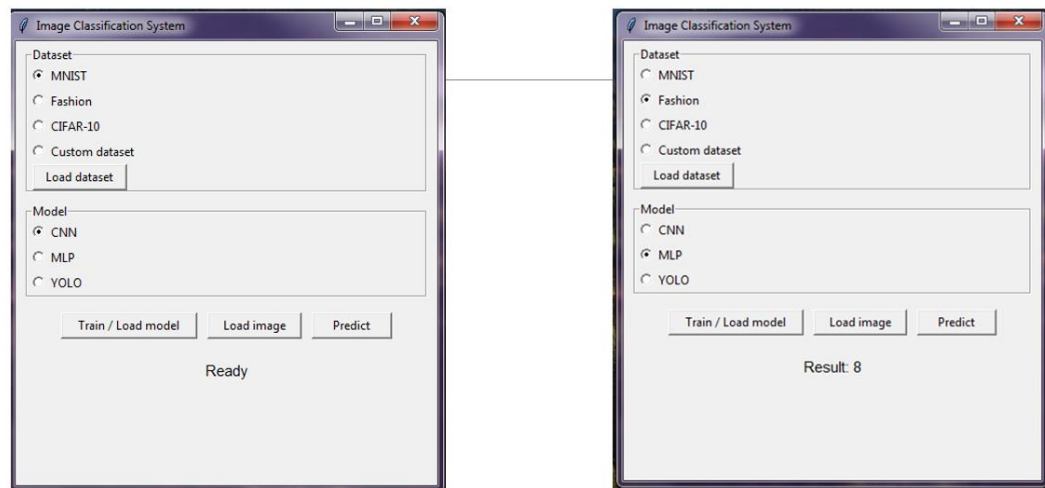
Функціонування програми



9

Рисунок Б.9 – Слайд 9

Результат



10

Рисунок Б.10 – Слайд 10

Висновки

- В результаті дипломного проекту бакалавра було розроблено програмне забезпечення Image Classification System, що призначене для розпізнавання об'єктів на зображеннях за допомогою нейромережових моделей.
- Розроблена програма та її інтерфейс задовільняють всі функціональні вимоги.
- На етапі тестування програмного забезпечення було виявлено, що кількість епох для навчання недостатня для високого рівня точності. Наслідком невеликої кількості епох є технічні причини. Отже збільшення кількості епох з 3 до, наприклад 10, значно підвищить точність прогностування.