Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Програмні засоби розпізнавання об'єктів
на зображеннях з використанням машинного навчання

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІз-21-1
Олексій ЧЕРНИХ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Анастасія ЛУЦЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Черних Олексію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмні засоби розпізнавання об'єктів на зображеннях з використанням машинного навчання

затверджена наказом по університету від " 05 " травня 2025 р. № 72 Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) мова програмування: python;

2) типи зображень для розпізнавання: кольорові зображення розміром 32x32 cifar10;

3) технологія машинного навчання: deep learning, згорткова нейронна мережа, tensorflow;

4) точність розпізнавання: тестові дані – 71,22%

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області та постановка завдання

2) вибір програмних засобів

3) реалізація та тестування, аналіз результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 14 слайдів.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	06.05.2025-09.05.2025	
2	Вибір технології розробки та	10.05.2025-13.05.2025	
	інструментальних засобів		
3	Розробка алгоритмічного забезпечення	14.05.2025-17.05.2025	
4	Розробка програмних модулів	19.05.2025-28.05.2025	
5	Відлагодження програмних модулів	29.05.2025-05.06.2025	
6	Оформлення матеріалів кваліфікаційної роботи	06.06.2025-09.06.2025	
7	Подання кваліфікаційної роботи керівникові та	10.06.2025-11.06.2025	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	12.06.2025-13.06.2025	
	вання		

Дата видачі завдання " 5 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Анастасія ЛУЦЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 75 с., 35 рис., 1 табл., 2 дод., 12 джерел.

DEEP LEARNING, PYTHON, CONVOLUTIONAL NEURAL NETWORKS, DATASET, EPOCH, ACCURACY, LOSS.

Мета кваліфікаційної роботи полягає у розробці програмного засобу для розпізнавання об'єктів на зображеннях із застосуванням методів машинного навчання. Комп'ютерний зір широко використовується в різних галузях, таких як медицина, безпека, промисловість, автономні транспортні засоби, а також технології штучного інтелекту стрімко розвиваються, тому тема актуальна по сьогоднішній день.

У ході виконання кваліфікаційної роботи було проаналізовано підходи до обробки зображень та методи розпізнавання об'єктів нейромережами. У практичній частині було спроектовано та навчено модель нейромережі, реалізовано програмну систему, що взаємодіє з моделлю для розпізнавання обраних зображенн. У склад програми входять наступні етапи: передобробка вхідних даних, завантаження попередньо натренованої моделі нейромережі та тестування її точності. Відібрано набір, що містить 20 зображень та проведено на ньому оцінку точності моделей двох версій з різною кількістю епох, щоб продемонтструвати на реальному прикладі як поводить себе нейромережа з різним рівнем навчання. У результаті виконаної роботи створено програму для розпізнавання об'єктів на зображеннях.

ABSTRACT

Bachelor’s thesis: 75 pages, 35 figures, 1 tables, 2 appendices, 12 sources.

DEEP LEARNING, PYTHON, CONVOLUTIONAL NEURAL NETWORKS, DATASET, EPOCH, ACCURACY, LOSS.

The major goal of this thesis is to develop software tools for recognizing objects in images using machine learning methods. The relevance of the topic is attached to the widespread use of computer vision in various industries, such as the security industry, medicine, autonomous vehicles, industry, as well as due to the development of artificial intelligence technologies.

In order to complete the thesis, an analysis of approaches to image processing and methods for recognizing objects using neural networks was conducted. In the practical part, a software system was implemented that performs automatic recognition of objects in static images. The system includes the stages of pre-processing of input data, training a convolutional neural network on a prepared set of images, as well as testing the quality of the model. The accuracy of the models of two versions with different numbers of epochs was assessed in order to demonstrate on a real example how a neural network behaves with different levels of training. As a result of the work, an object recognition tool based on modern machine learning methods was created.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ**ПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕН**
ВСТУП .....**ПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕНО.**
1 ТЕОРЕТИЧНА ЧАСТИНА ..**ПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕНО.**
1.1 Основні задачі комп'ютерного зору 10
1.2 Методи машинного навчання у розпізнаванні об’єктів..... 104
1.3 Архітектура та принцип дії CNN..... 16
1.4 Огляд інструментів та бібліотек Python для розпізнавання зображень..... 22
1.5 Аналіз існуючих рішень і постановка задачі 25
2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 29
2.1 Обґрунтування нейромережі та програмного забезпечення 29
2.2 Архітектура моделі 30
2.3 Архітектура системи розпізнавання..... 32
2.4 Вибір та підготовка навчальних даних 34
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 36
3.1 Середовище розробки та інструменти 36
3.2 Побудова та налаштування моделі..... 36
3.3 Ключові модулі програми 44
3.4 Візуалізація результатів розпізнавання 46
4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 51
4.1 Результати тестування..... 51
4.2 Аналіз отриманих результатів 52
ВИСНОВКИ..... 53
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 54
ДОДАТОК А Графічний матеріал кваліфікаційної роботи..... 55
ДОДАТОК Б Код програми 63

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – прикладний програмний інтерфейс (англ., Application Programming Interface)

AI – штучний інтелект (англ., Artificial Intelligence)

CNN – згорткова нейронна мережа (англ., Convolutional Neural Network)

DL – глибоке навчання (англ., Deep Learning)

GPU – графічний процесор (англ., Graphics Processing Unit)

HOG – гістограма напрямлених градієнтів (англ., Histogram of Oriented Gradients)

ML – машинне навчання (англ., Machine Learning)

SIFT – масштабоінваріантне ознакове перетворення (англ., Scale-Invariant Feature Transform)

SURF – прискорені стійкі ознаки (англ., Speeded-Up Robust Features)

SVM – метод опорних векторів (англ., Support Vector Machine)

SSD – англ., Single Shot MultiBox Detector

ВСТУП

Сучасний розвиток інформаційних технологій супроводжується стрімким зростанням обсягів цифрової інформації, значну частину якої складають зображення та відео. У зв'язку з цим фактором, усе більшої актуальності набувають системи автоматичного аналізу та інтерпретації візуальних даних. Одним із ключових напрямів у цій сфері є розпізнавання об'єктів на зображеннях, тобто технологія, що дає змогу виявляти, класифікувати та локалізовувати різноманітні об'єкти в цифровому зображенні або відеопотоці.

Традиційні алгоритми комп'ютерного зору, які базуються на ручному виокремленні ознак і жорстко заданих правилах, часто не здатні забезпечити необхідну точність і гнучкість при обробці складних або неоднорідних даних. У цьому контексті нові можливості відкриває застосування машинного навчання (ML), зокрема глибинного навчання (DL) та згорткових нейромереж (CNN), які автоматично виявляють релевантні ознаки на основі великої кількості навчальних прикладів. Такі моделі показали високу ефективність у завданнях класифікації зображень, детекції об'єктів, сегментації сцен та інших задачах комп'ютерного зору.

Актуальність теми полягає в тому, що впровадження програмних засобів для розпізнавання об'єктів дозволяє автоматизувати процеси, які раніше потребували участі людини, підвищити точність і швидкість обробки візуальної інформації, а також створює передумови для подальшого розвитку систем штучного інтелекту (AI).

Метою даної кваліфікаційної роботи є розробка програмних засобів розпізнавання об'єктів на зображеннях із використанням методів ML, зокрема CNN, за допомогою мови програмування Python.

Для досягнення поставленої мети у роботі вирішуються такі основні завдання: аналіз сучасних підходів до розпізнавання об'єктів на зображеннях;

обґрунтування вибору інструментальних засобів і технологій реалізації; проєктування та реалізація програмної системи розпізнавання об'єктів; тестування ефективності розробленої моделі.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Основні задачі комп’ютерного зору

Комп'ютерний зір це напрямок штучного інтелекту, основною задачею якого є отримання, обробка, аналіз та визначення зорової інформації з реального світу. Іншими словами, це про здатність комп’ютерних систем сприймати зображення або відео схоже до того, як це робить людина. Даний напрям AI набув великого значення в різноманітних галузях та має широке застосування у медицині, промисловості, транспорті, робототехніці, безпеці тощо.

Основні задачі комп'ютерного зору, що розглядаються це класифікація зображень, локалізація, детекція об'єктів та сегментація зображення. Класифікація це задача для визначення класу (категорії), до якої належить усе зображення. До прикладу модель може відповісти, чи правда зображення містить в собі, або є кішкою, собакою, автомобілем, тобто якоюсь конкретною категорією об’єкта. Локалізація представляє собою знаходження положення об'єкта на зображенні в конкретному місці за допомогою будування обмежувальної рамки у вигляді чотирикутника, або іншого індикатора навколо цього об’єкта. Ця задача відрізняється від класифікації тим, що таким чином не лише визначається наявність об'єкту певного класу, а ще й вказується де саме він знаходиться на зображенні, що дозволяє отримувати більше інформації та приміняти даний підхід у багатьох більш продвинутих та складних системах. Детекція є більш комплексною задачею у порівнянні з попередніми двома підходами. Вона поєднує ці дві задачі та розширює функціонал таким чином, що на одному зображенні може бути знайдено та зафіксовано присутність декількох об'єктів різних класів, причому вони можуть бути від’ємними та непов’язаними одне з одним. Принцип роботи цих трьох задач продемонстровано на рисунку 1.1 на

прикладі зображень котів і собак.

Рисунок 1.1 – Принцип роботи класифікації, локалізації та детекції об’єктів

Сегментація це такий підхід, в якому відбувається поділ зображення на підгрупи (або сегменти). Ці сегменти відповідають певним об'єктам або класам, що зменшує загальну складність зображення тим, що з'являється можливість аналізу та обробки кожного сегмента окремо. Є три основні типи сегментації: семантична, сегментація на місцях (або більш відома як інстанс-сегментація) та паноптична сегментація. Семантична сегментація працює наступним чином: пікселі на зображенні розташовуються на базі деяких класів, що представляють об’єкти (наприклад небо, дорога, автомобіль). Інстанс-сегментація напроти класифікує окремо кожен об'єкт на зображенні, навіть якщо вони належать до одной й тої ж самой категорії. Паноптична сегментація є новішою технікою, ніж попередні дві: вона представляє собою поєднання семантичної сегментації та інстанс-сегментації. В ній передбачається ідентичність кожного об'єкта, відокремлюючи кожен екземпляр кожного об'єкта на зображенні [1]. На рисунку 1.2 зображений принцип роботи сегментації зображень.

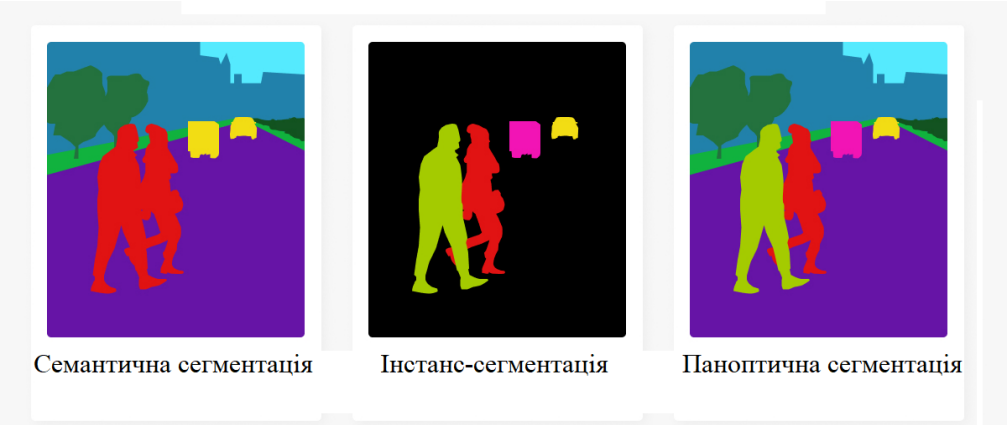

Рисунок 1.2 – Принцип роботи семантичної, інстанс-сегментації та паноптичної сегментації

Також існує більш продвинутий підхід відстеження (або трекінг) об'єктів. Це така задача аналізу відео, де ідентифікуються об’єкти, виявляється їхнє місцезнаходження, а потім відстежуються їх переміщення між кадрами відео, навіть коли об'єкти частково або повністю перекриваються іншими об'єктами в кадрі. Дана задача є важливою для систем відеоспостереження, автономного транспорту та розширеної реальності [2]. На рисунку 1.3 демонструється приклад роботи трекінгу об'єктів для спортивних змагань, де під час відеоетеру відстежуються спортсмени та ідентифікуються кожен окремо за своїм унікальним номером.

Рисунок 1.3 – Принцип роботи відстеження об’єктів

Ще одною задачею, де система повинна виявляти відхилення від норми

у зображеннях або відео, це виявлення аномалій. До прикладу можна привести контроль якості на виробництві, медичну діагностику або аналіз безпеки. Ефективність цієї задачі дозволяє прибирати з виробничого ланцюга елементи або частини в поганому стані. В результаті виробничі витрати знижуються через уникнення виробнитства та продажу дефектної продукції. Виявлення аномалій на заводах є корисним інструментом для систем контролю якості завдяки своїм особливостям і є великим викликом для інженерів з ML [3]. Також ця задача показала себе дуже ефективною у знаходженні несумісності пропорцій або аномальної анатомії на зображеннях, що генеруються за допомогою AI, коли ті ще розвивалися, наприклад виявлення шести пальців на руках, зайвих ніг та інших неправильних форм об'єктів, таких як люди, тварини, рослини тощо. На рисунку 1.4 продемонстровано приклад пласту зображень без аномалій та з аномаліями.

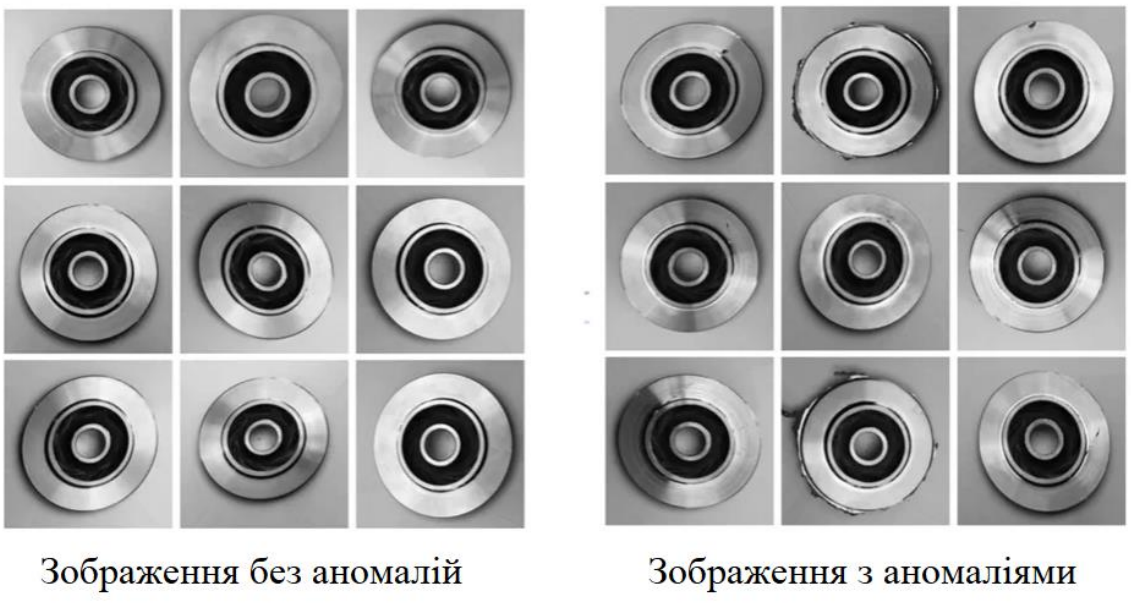

Рисунок 1.4 – Приклад зображень підшипників без аномалій та з аномаліями

Також доволі важливою є задача для розпізнавання облич. Суть цієї задачі у виявленні обличчя на зображенні та ідентифікації особи, завдяки

аналізу закономірностей в рисах обличчя. Приблизний принцип роботи заключається у виявленні облич на зображеннях або відео та вилучення їх характерних ознак (наприклад, відстань між очима або форма підборіддя) і порівняння цих ознак з базою даних. Тобто основний процес зазвичай складається з трьох етапів: знаходження обличчя на зображенні, кодування унікальних даних про обличчя в математичне представлення і зіставлення, тобто порівняння цих даних зі збереженими шаблонами. Сучасні системи часто використовують моделі DL, такі як CNN, для автоматизації та підвищення точності на цих етапах [4]. На рисунку 1.5 зображений приклад роботи задачі для продвинутого розпізнавання обличь з відображенням геометрії, "скелету" обличчя особи.

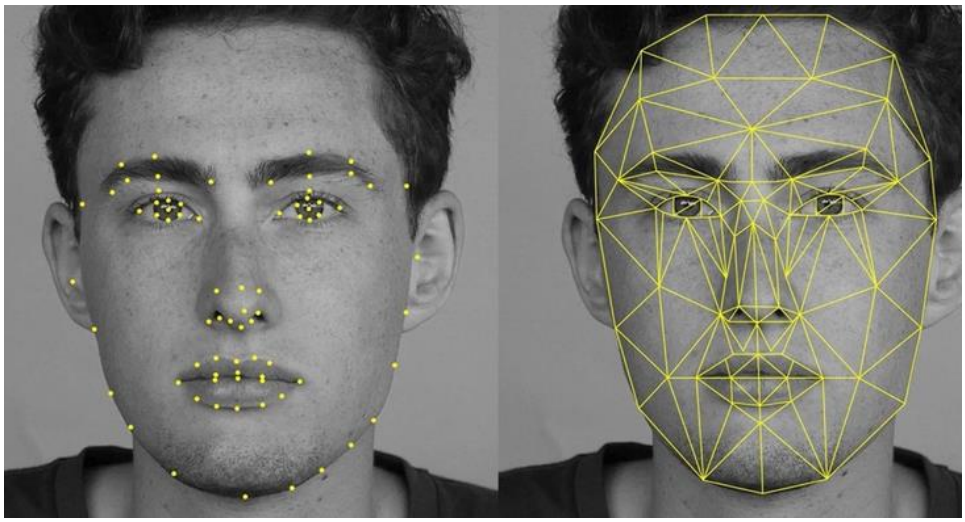

Рисунок 1.5 – Приклад виконання задачі продвинутого розпізнавання обличь

1.2 Методи машинного навчання у розпізнаванні об’єктів

ML являється ведучим підходом у створенні інтелектуальних інформаційних систем, що самонавчаються та можуть приймати рішення на базі масиву даних. Завдяки ML тренувати моделі виявляти питомі візуальні риси, що відносяться до конкретних класів об'єктів, без необхідності самотужки створювати жорсткі правила, стало більш ніж продуктивно та ефективно, якщо говорити у контексті комп'ютерного зору та задач

розпізнавання об'єктів. Методи ML, що підходять для таких родів завданнь, умовно поділяються на традиційні та глибинні підходи.

Розпізнавання об'єктів базувалося на комбінації попередньої обробки зображення, коли глибокі нейронні мережі ще не з’явилися у звичному нам на сьогоднішній момент вигляді. Тобто спочатку зображення опрацьовували певними методами, наприклад використовувати фільтрацію шуму, покращення контрасту, перетворення в чорно-біле тощо, а вже потім виконувався сам процес розпізнавання. Основні етапи включали в себе виділення ознак та класифікацію. Виділення ознак відбувалося за допомогою побудови векторів ознак на основі таких алгоритмів як SIFT, HOG, SURF тощо. Класифікація виконувалася за допомогою таких алгоритмів машинного навчання як SVM, Random Forest, k-Nearest Neighbors, Logistic Regression. Хоч вони і мають деякі обмеження у масштабуванні, точності та гнучкості при роботі зі складними або об’ємними за розміром наборами зображеннь, ці методи все ще ефективні для свого часу та для простих задач [5].

Одна з технологій, яка отримала значний розвиток з появою електронних обчислювальних машин, що працюють з великими обсягами даних, стали методи DL, а саме CNN. CNN автоматично навчаються ефективно витягати корисні ознаки зображень під час навчання, що відрізняється від класичних підходів. CNN складається із суміжних оглядових шарів, шарів активації, шарів пулінгу та шарів повного зв'язку [6]. Їх, як і саму згорткову нейронну мережу, буде розглянуто пізніше.

У розпізнаванні об'єктів зазвичай використовується «навчання з учителем», або контрольоване навчання, коли модель навчається на мітках у зображеннях. Проте останнім часом зростає інтерес до напівконтрольованого навчання, «навчання без учителя», тобто неконтрольованого навчання, і самонавчання, де використовуються, наприклад, методи попереднього навчання моделей на неанотованих даних. Проте навчання без учителя та з учителем часто обговорюють разом. На відміну від алгоритмів неконтрольованого навчання, алгоритми керованого навчання

використовують анотовані дані. На основі цих даних вони або прогнозують майбутні результати, або відносять дані до певних категорій на основі проблеми регресії або класифікації, яку вони намагаються вирішити.

Хоча алгоритми керованого навчання, як правило, більш точні, ніж некеровані моделі навчання, вони вимагають попереднього втручання людини для належного маркування даних. Однак ці марковані набори даних дозволяють алгоритмам керованого навчання уникати обчислювальної складності, оскільки їм не потрібен великий навчальний набір для отримання запланованих результатів. Напівконтрольоване навчання відбувається, коли тільки частина вхідних даних була позначена. Неконтрольоване і напівконтрольоване навчання зазвичай являються більш раціональними альтернативами, оскільки покладатися на експертизу назначених кадрів в предметній області для належного маркування даних не дуже вигідно з точки зору часу і коштів [7].

1.3 Згорткові нейронні мережі: архітектура та принцип дії

CNN як одна з типів нейромереж, яка працює по принципу витягування ознак через фільтри, спеціально розроблена для передбачення зображень або даних, що мають матричну або тензорну структуру. Так само за принципом роботи зорової кори людини, CNN виявляють вхідні візуальні патерни проходячи глибокорівневу обробку [8]. Саме через цю функцію автоматичного виділення ознак CNN і стали у задачах класифікації, детекції та сегментації основним інструментом.

Як вже було зазначено раніше, CNN складається з таких основних шарів як згортковий, шар активації, пулінгу, повнозв'язний шар та нормалізація. Далі буде розглянуто кожен з них детальніше.

Ключовою структурною частиною даного типу нейромереж є згортковий шар, оскільки більшість обчислень саме в ньому і проходить (рис. 1.6). Згорткові шари обробляють локальні шаблони, та якщо говорити у

контексті зображень – шаблони невеликих двовимірних вікон вхідних даних. Ця характеристика дає змогу зрозуміти дві важливі речі: шаблони, що проходять крізь згорткові шари є інваріантими до зсувів. Тобто після обробки деякої частини зображення CNN зможе визначити цю ж частину вже в іншому місці та іншому ракурсі; вони здатні до вивчення просторової ієрархії шаблонів. Перший згортковий шар буде вивчати невеликі локальні шаблони, такі як краї, коли другий вже буде затрагувати більш значущі, що складаються з шаблонів, що повертає попередній шар.

Детектор ознак – це двовимірний масив вагових коефіцієнтів, який представляє частину зображення. Хоча вони можуть мати різний розмір, розмір фільтра зазвичай становить матрицю 3x3. Це також визначає розмір сприйнятливого поля. Потім фільтр накладається на частину зображення, і обчислюється скалярний добуток між вхідними пікселями і фільтром. Цей добуток потім подається у вихідний масив. Після цього фільтр зсувається на крок, повторюючи процес до тих пір, поки ядро не пройде усе зображення. Кінцевий результат із серії добутків вхідних даних і фільтра називається картою ознак, картою активації або згорнутою ознакою (рисунок 1.7).

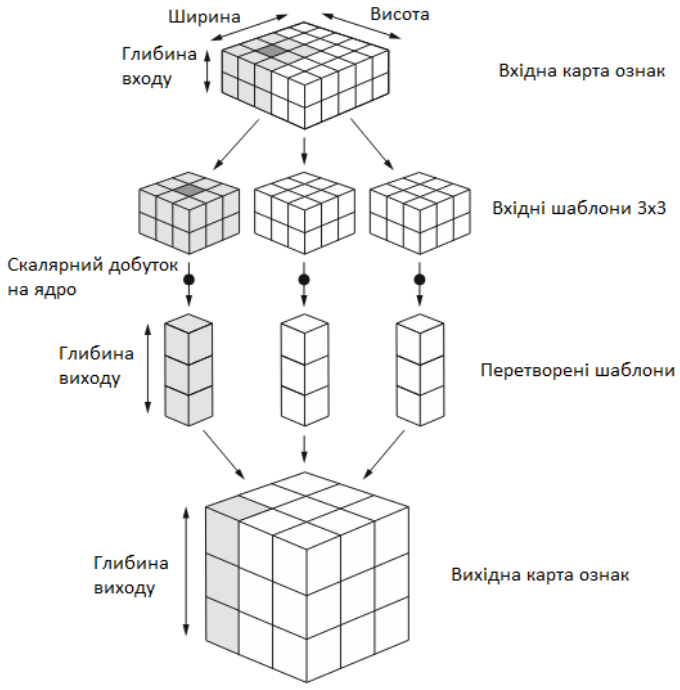

Рисунок 1.6 – Структура згорткового шару

Ваги в детекторі ознак не змінюються коли він рухається по зображенню, іншими словами це називається розподіл параметрів. Значення ваг налаштовуються під час тренування зворотнім розповсюдженням та градієнтним спуском. Є 3 гіперпараметри, що впливають на об'єм вихідних даних, які треба задати до початку навчання нейромережі. Це кількість фільтрів, крок та нульове заповнення.

Кількість фільтрів впливає на глибину вихідних даних. Наприклад, три різні фільтри дадуть три різні карти ознак, створюючи глибину, що дорівнює трьом. Крок – це відстань або кількість пікселів, на яку ядро переміщується по вхідній матриці. Більший крок дає менший результат, тому значення кроку в два і більше пікселів дуже рідко де коли зустрічаються. Нульове заповнення зазвичай використовується, коли фільтри не підходять до вхідного зображення. При цьому всі елементи, що виходять за межі вхідної матриці, прирівнюються до нуля, що дає на виході більший або однаковий за розміром результат. Існує три типи заповнення: дійсне, однакове та повне заповнення. Дійсне заповнення також відоме як відсутність проміжків. У цьому випадку остання згортка відкидається, якщо розміри не вирівнюються. Однакове заповнення гарантує, що вихідний шар має той самий розмір, що й вхідний. Повне заповнення збільшує розмір вихідного шару шляхом додавання нулів до межі вхідного шару.

Після кожної операції згортки CNN застосовує перетворення випрямленої лінійної одиниці до карти ознак, вносячи нелінійність у модель. Ще її називають функцією активації ReLU. Вона використовується для перетворення від’ємних значень в нуль. Без функцій активацій нейромережа зможе навчатися лише на лінійних перетвореннях вхідних даних.

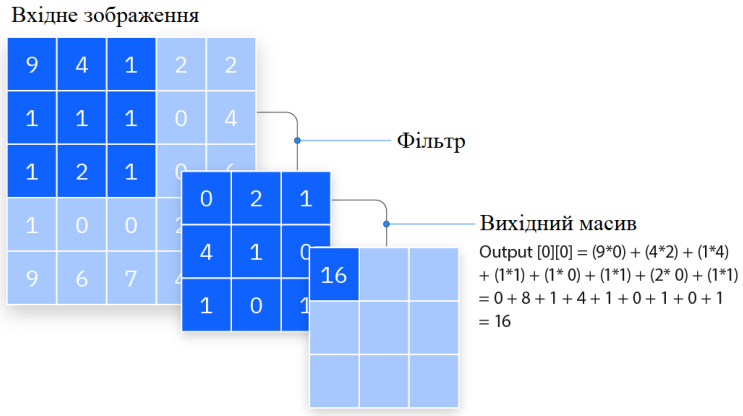

Рисунок 1.7 – Приклад обчислення вагового коефіцієнту для побудови карти ознак

Шар пулінгу відповідає за агресивне зменшення карти ознак, подібно до згорткового шару. Операція об'єднання сусідніх значень діє наступним чином: із вхідної карти ознак вилучається вікно заданого розміру (зазвичай це матриця 2x2), та з нього вибирається максимальне, чи середнє значення для кожного каналу. У випадку з максимальним об’єднанням обирається максимальне з 4 значень матриці, а у випадку з усередненим знаходиться середнє арифметичне. Зазвичай частіше всього використовується максимальне об’єднання. Концептуально це може нагадувати згротку, проте перетворення тут відбувається за жорстко заданою тензорною операцією максимального (усередненого) об’єднання. Ключова відмінність від згортки в тому, що пулінг проводиться з кроком 2 коли розмірність вікна пулінгу дорівнює 2x2, що рівно ділить роздільну здатність карти ознак в два рази. [13] (рисунок 1.8).

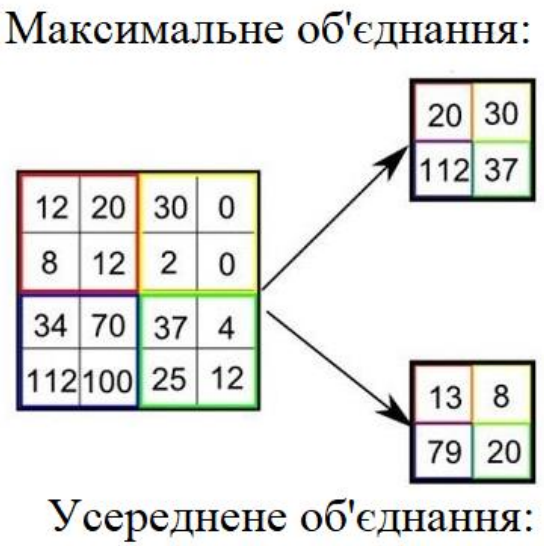

Рисунок 1.8 – Приклад виконання максимального і усередненого об’єднань

Головний мінус операції об'єднання це те, що в її результаті на виході втрачається значна кількість інформації. Проте навіть попри це, у цього шара є свої плюси. Дана операція може зменшити складність зображення, що позитивно вплине на швидкість роботи нейромережі, в принципі це і є основною задачею даного шару.

Повнозв'язний шар відрізняється від частково з'єднаних тим, що значення пікселів вхідного зображення у других з вихідним шаром не пов'язані напряму. Однак у повністю зв'язному шарі кожна вершина вихідного шару з'єднується з вершиною попереднього шару безпосередньо. Класифікація є основною задачею даного шару. Відбувається це на основі ознак, що він отримує через попередні шари та їх різні фільтри. Функції ReLu, як правило, використовують згорткові та об'єднувальні шари, коли повнозв'язні зазвичай використовують функцію активації softmax, що розприділяє дані та створює з них ймовірність від 0 до 1.

Архітектура типової CNN на простому прикладі продемонстрована на рисунку 1.9.

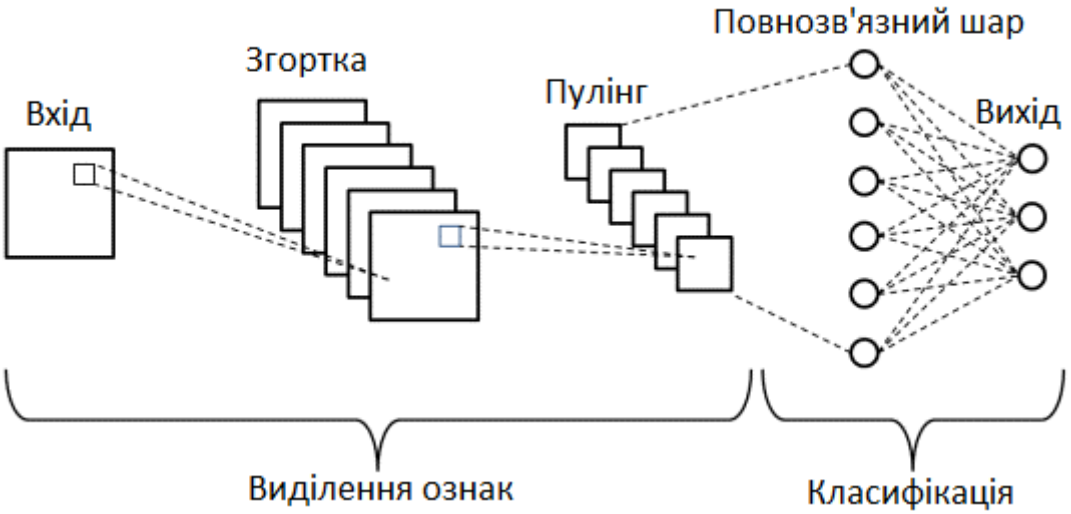

Рисунок 1.9 – Типова архітектура CNN

Які переваги є у CNN і чому вони стали настільки популярними у сфері комп'ютерного зору? Ключовою є змога виділяти ознаки з вхідних даних автоматично, без втручання розробника та ручного проєктування ознак, як у традиційних методів. Завдяки побудові згорток і операцій пулінгу CNN інваріантні до змін масштабу та зсувів. Це призводить до того, що можна створювати більш обширну вибірку, маючи лише невелику кількість зображень. Дані нейромережі підходять як для простих задач, так і для роботи з великими датасетами, зокрема з ImageNet. Крім того, CNN застосовується зараз багато де, та не лише для аналізу зображень: у відеоаналітиці, медичних дослідженнях, автономному водінні, розпізнаванні голосу та багатьох інших сферах.

Які недоліки є у CNN? Основні два недоліки, це розмір і якість вибірки та обчислювальний ресурс. Щоб модель навчилася більш ефективно, для цього їм треба великі обсяги даних, що може стати проблемою у випадку нема можливості створити або зібрати якісну навчальну вибірку. Зодо обчислювальних ресурсів, дані нейромережі потребують потужних GPU з наявністю тензорних ядер, а подібні відеокарти коштують дорого. Хоч і з CNN можна працювати навіть на слабких системах, проте результат буде відрізнятися від версій з більшим доступом до обчислювальних ресурсів.

1.4 Огляд інструментів та бібліотек Python для розпізнавання зображень

Мова програмування Python є однією з найпопулярніших у сфері ML, комп'ютерного зору та аналізу зображень (рис. 1.10). Вона має широкий спектр бібліотек, що забезпечують як базову обробку зображень, так і побудову та навчання нейронних мереж. Розглянемо основні інструменти Python, які використовуються для розпізнавання об'єктів на зображеннях із використанням ML.

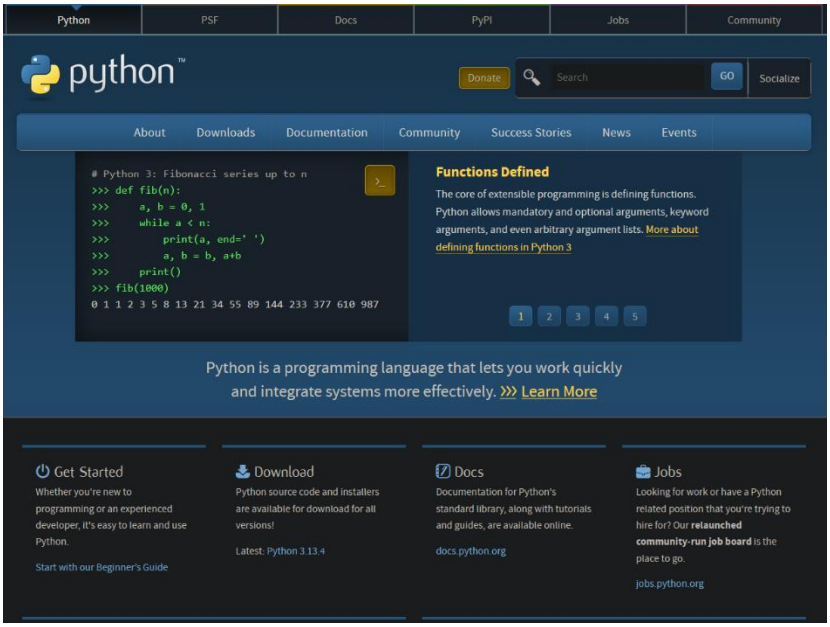

Рисунок 1.10 – Офіційний веб-сайт Python

NumPy та Pandas хоч і не являються бібліотеками для роботи з нейронними мережами, проте все ще є незамінними в якості допоміжних бібліотек. NumPy є бібліотекою для числових обчислень, що забезпечує швидку роботу з багатовимірними масивами, що є основою для представлення зображень у вигляді тензорів (рис. 1.11). Pandas це зручний інструмент для обробки табличних даних. Часто використовується для зберігання метаданих зображень, результатів класифікації, аналітики тощо (рис. 1.12).

Рисунок 1.11 – Приклад роботи NumPy

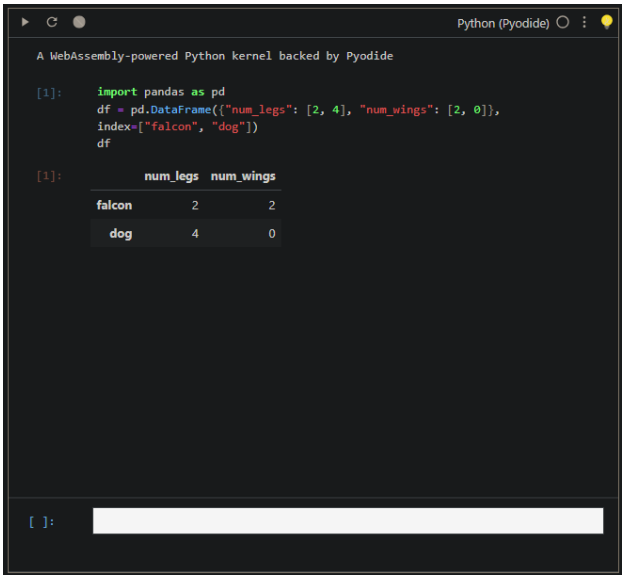

Рисунок 1.12 – Приклад роботи Pandas

Для роботи з графіками і візуалізації зображень використовують Matplotlib і його розширення Seaborn, що підходить для побудови статистичних графіків, часто використовується для візуального аналізу результатів класифікації, точності, втрат тощо (рис. 1.13).

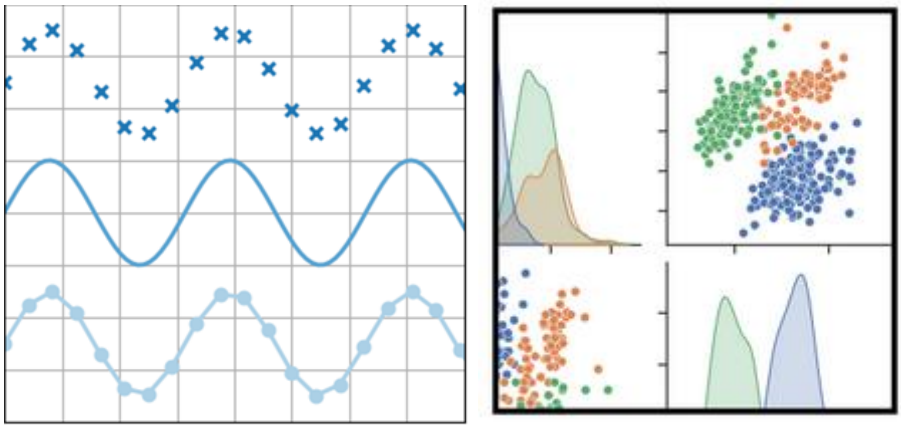

Рисунок 1.13 – Зліва направо: 1) Приклад графіку Matplotlib; 2) Приклад графіку з використанням розширення Seaborn

OpenCV є одною з найпотужніших бібліотек для обробки зображень та відео (рис. 1.14). Вона часто виступає у якості першого етапу попередньої обробки даних перед подачею в нейронну мережу. OpenCV підтримує широкий спектр мов програмування, зокрема C++, Python, Java тощо, і доступний на різних платформах, включаючи Windows, Linux, OS X, Android та iOS. Активно ведеться розробка інтерфейсів для високошвидкісних обчислень на графічних процесорах з використанням технологій CUDA та OpenCL. Для мови Python існує спеціальний прикладний програмний інтерфейс (API) OpenCV-Python, який поєднує в собі переваги C++ API OpenCV та зручність мови Python. Він базується на бібліотеці Numpy, про який вже було сказано раніше. Усі структури масивів OpenCV автоматично перетворюються в масиви Numpy і навпаки, що спрощує інтеграцію з іншими бібліотеками, такими як Matplotlib, які також використовують Numpy [9].

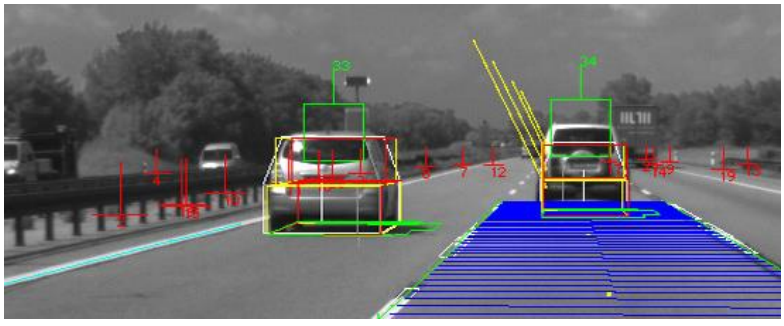

Рисунок 1.14 – Приклад роботи OpenCV

TensorFlow це потужний і гнучкий фреймворк від компанії Google, призначений для створення та навчання моделей ML (рис. 1.15). Він працює з багатовимірними тензорами, підтримує обчислення на графічних процесорах і пропонує інструменти для оптимізації процесів. TensorFlow надає можливість працювати як на низькому рівні через API TensorFlow Core, так і на високому рівні за допомогою Keras. Keras, у свою чергу, є високорівневим API до TensorFlow, який значно спрощує побудову нейронних мереж завдяки зрозумілому й зручному синтаксису. Особливо ефективним він є для задач розпізнавання зображень: дозволяє створювати CNN, використовувати передтреновані моделі, такі як VGG16, ResNet чи MobileNet, а також застосовувати пакет ImageDataGenerator для аугментації даних з метою підвищення якості навчання моделей.

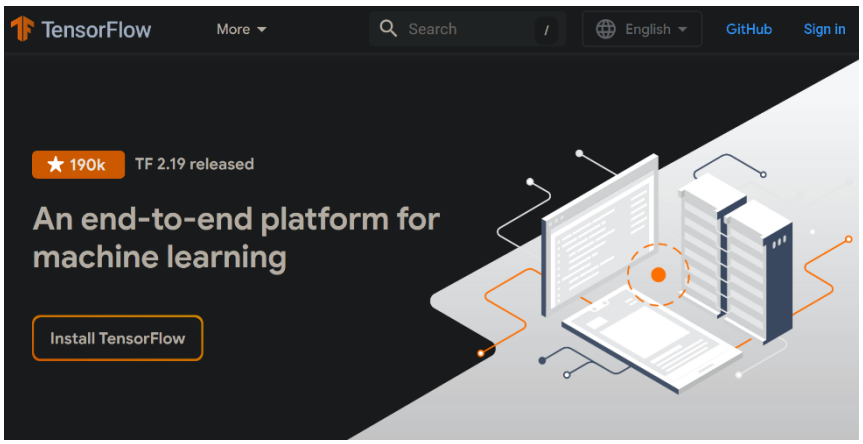

Рисунок 1.15 – Офіційний сайт Tensorflow

1.5 Аналіз існуючих рішень і постановка задачі

Серед найпопулярніших архітектур CNN для розпізнавання об'єктів варто виділити кілька рішень, що зарекомендували себе в різних сферах застосування. Розглянемо їх та визначимо переваги та недоліки кожної з них.

VGGNet це архітектура на базі CNN, що має до 19 згорткових шарів з

фільтрами 3 на 3 (рис. 1.16). Найвідоміші з моделей VGG16 та VGG19, що побудовані на основі VGGNet, підтримують 16 та 19 згорткових шарів відповідно. В ImageNet дані моделі досягають точності тестування близько 92-95%, що входить до п'ятірки найкращих серед подібних моделей. У порівнянні з AlexNet, більш застарілою архітектурою, дана модель використовує замість великих фільтрів 11x11 та 5x5, лише фільтри розміром 3x3. Для навчання такої моделі протягом більше ніж двох тижнів використовували GPU Nvidia Titan Black. Модель VGG16 може класифікувати фотографії за тисячею різних категорій [12]. Основним недоліком даного рішення є його великий розмір та швидкість навчання.

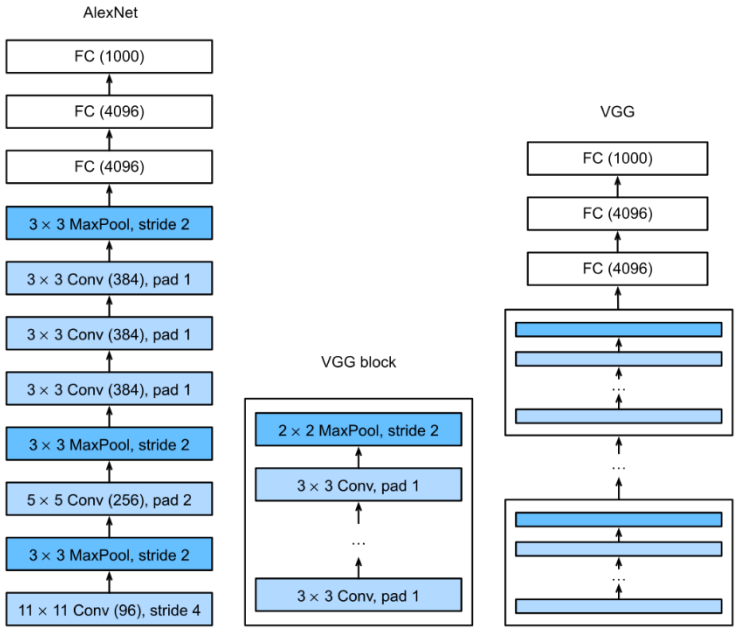

Рисунок 1.16 – Архітектура VGGNet (справа) у порівнянні з AlexNet (зліва)

Архітектура ResNet (рис. 1.17) трохи новіша за VGG, та обганяє її по тестам у точності ImageNet, досягаючи значення в 96%. Основною унікальною особливістю даної архітектури є використання залишових з'єднань, що дозволяють проводити більш ефективне тренування навіть при дуже великій глибині з кількістю шарів до 152 (модель ResNet152). Залишкові з'єднання забезпечують ще один шлях для даних, щоб вони змогли досягти пізніших частин нейромережі, пропускаючи деякі шари.

Проте попри таку високу точність у даної архітектури є суттєві недоліки, пов’язані з дуже високими обчислювальними витратами. Також реалізація залишкових з’єднань набагато ускладнює архітектуру у порівнянні з тими ж VGG чи AlexNet.

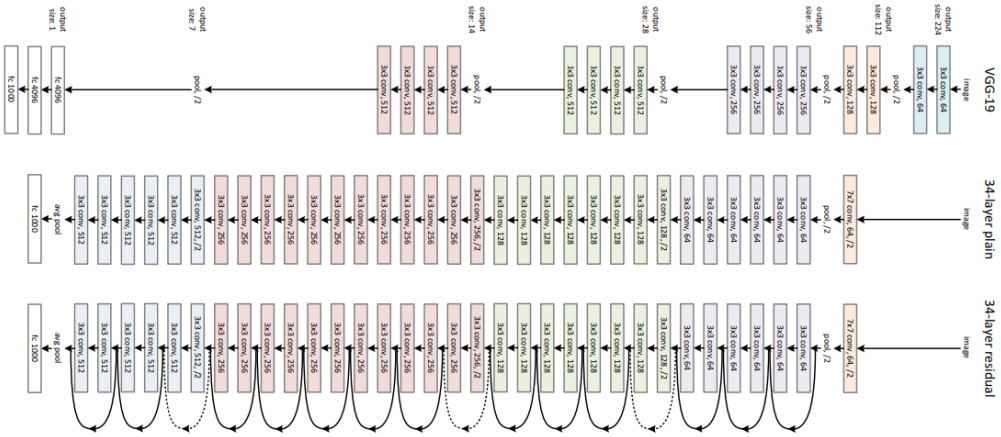

Рисунок 1.17 – Порівняння масштабів архітектури ResNet з VGG19

Крім локальних рішень, доступна низка хмарних сервісів, таких як Google Vision API, Amazon Rekognition і Microsoft Azure Computer Vision. Вони дозволяють розпізнавати об'єкти, обличчя, текст та інші елементи на зображеннях, однак часто вимагають платної підписки, мають обмежений доступ до внутрішніх моделей або не дозволяють гнучко налаштовувати процес розпізнавання.

Попри широкі можливості сучасних рішень для розпізнавання об'єктів, вони всі загалом мають низку суттєвих недоліків. Насамперед, багато моделей є складними в налаштуванні та вимагають глибоких знань у сфері ML. Крім того, значна частина з них має високі обчислювальні вимоги. Для ефективної роботи часто необхідне використання потужного GPU. Ще однією проблемою є обмежена адаптованість до нових типів об'єктів або вузькоспеціалізованих задач: без додаткового донавчання моделі не завжди здатні забезпечити якісний результат. Окрему увагу слід приділити залежності від хмарних сервісів. Не всі проєкти можуть або хочуть використовувати сторонні платформи через питання конфіденційності або

високу вартість таких рішень.

У межах даної кваліфікаційної роботи постановлена наступна мета: реалізувати програмний засіб для розпізнавання об'єктів на зображеннях за допомогою CNN, створеної на мові програмування Python. Важливою характеристикою розробленої системи має бути її здатність функціонувати автономно на локальному комп'ютері, без потреби у підключенні до зовнішніх хмарних сервісів. Для досягнення поставленої мети було поставлено низку завдань. Спочатку буде побудована та зпроектована архітектура CNN, що найкраще підійде для задач класифікації об'єктів. Буде визначено оптимальну архітектуру, здатну забезпечити баланс між точністю, швидкістю обробки та ресурсними вимогами. Також буде зпроектована система розпізнавання, тобто сама програма, що буде працювати з вже навченою моделлю та видавати результат. Наступним з етапів стане відбір готового набору зображень, з подальшою його обробкою для покращення якості навчання. Безпосередню реалізацію моделі буде здійснено із використанням мови програмування Python та сучасних бібліотек для ML: TensorFlow Keras. Після реалізації моделі буде проведено її навчання. Буде навчено дві моделі з різним рівнем епох для порівняння. Для перевірки роботи моделі буде реалізована система розпізнавання у вигляді інтерфейсу користувача, що матиме функціонал для тестування моделей на нових зображеннях. Завершальним етапом стане узагальнення отриманих результатів та формування висновку по роботі програми.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Обґрунтування вибору нейромережі та програмного забезпечення

Для розв’язання поставленої задачі кваліфікаційної роботи та створення програмного забезпечення для розпізнавання об’єктів на зображеннях було обрано нейронну мережу CNN. Такий вибір зумовлений тим, що CNN є одним із найефективніших та найпопулярніших типів нейронних мереж для задач, пов’язаних з обробкою та аналізом зображень.

Архітектура CNN не потребує ручного виділення ознак, як це робилося на більш класичних та вже застарілих системах. Вона сама виділяє ознаки в процесі свого навчання. У контексті поставленої задачі CNN гарно себе проявляє у випадках, коли об’єкт розміщений на зображеннях в різних своїх положеннях, ракурсах і тд. Тобто модель буде більш придатна до зсувів у матрицях та не буде потребувати більш фіксованого датасету, де потрібно узлагоджувати усі зображення між собою, щоб не було великої різниці в тому, де об’єкт знаходиться в координатній площині. Як вже було зазначено вище, CNN має автоматичне виділення ознак у зображенні. Модель навчатиметься самостійно знаходити та виділяти питомі риси об’єкта на зображеннях без попереднього втручання розробника. Під питомими рисами мається на увазі як низькорівневі (краї, контури) так і високорівневі (форми, структури) ознаки, що знову ж таки, нівелює потрібність втручання у процес людиною. І одне з ключових, це те, що CNN реалізований у багатьох фреймворках, має дуже велику ресурсну базу, багато статтей та загалом інформації та якісної підтримки.

Для реалізації моделі на основі CNN була вибрана мова програмування Python версії 3.11, що сама по собі дуже ефективна при роботі з великими обсягами даних та в цілому одна з найкращих мов програмування у Data Science. Причиною вибору саме версії 3.11 послужило більша кількість LTS

бібліотек та ресурсів саме для цієї версії мови програмування, пов’язаних з темою роботи, тобто з нейронними мережами. Середовищем розробки виступає IDE PyCharm Community версії 2024.3. Вибір зумовлений формально та суб’єктивно, так як причиною послужила більш комфортна розробка та гнучкість в налаштуванні.

Для CNN є реалізації у вигляді бібліотеки TensorFlow та її API Keras, що будуть використані в подальшій розробці програмного забезпечення та для створення, налагодження та тренування моделі нейронної мережі. Ці фреймворки мають дуже широку спільноту, підтримуються до сих пір постійними оновленнями та нововведеннями, не потребують занадто багато ресурсів та можуть бути використаними в цілях роботи подібного рівня. Також були обрані такі допоміжні популярні бібліотеки як Matplotlib для роботи з графіками та NumPy для роботи з числовими даними і матрицями. Їх вибір обумовлений тим, що без даних бібліотек зараз важко уявити розробку програмного забезпечення у галузі нейронних мереж. Для реалізації інтерфейсу програмного забезпечення було обрано бібліотеку PyQt5, оскільки ця бібліотека має широку користувацьку базу.

2.2 Архітектура моделі

Архітектура моделі нейромережі повинна дотримуватися базовим правилам послідовності та багаторівневості. Зпроектована архітектура зображена на рисунку 2.1.

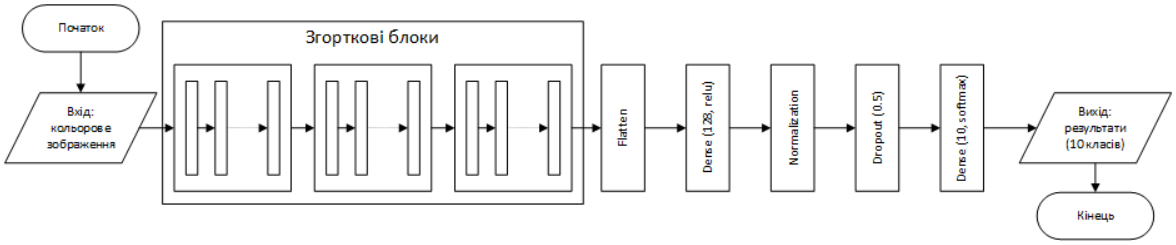

Рисунок 2.1 – Архітектура моделі нейромережі, що розробляється

Архітектура розроблена для моделі CNN має покрокове виділення, обробку та класифікацію ознак на вхідному зображенні розміром 32x32 пікселі з трьома колірними каналами RGB. Модель складається з трьох основних зготкових блоків, кожен з яких вміщає в собі два згорткові шари з 32 фільтрами 3x3, шаром активації та нормалізації для стабільнішого процесу навчання. В кінці кожного цього блоку йде шар пулінгу максимального об’єднання з розміром масштабування 2x2 для зменшення розмірності карти ознак, а також шари дропаутів, що скидають частину випадкових нейронів для запобігання перенавчанню. Схема згорткового блока зображена на рисунку 2.2. Перший такий згортковий блок містить 32 фільтри, другий 64, а третій 128. Така відмінність та поступове збільшення кількості фільтрів дозволить моделі вивчати все більш складніші деталі та ознаки на зображенні. Далі після згорткових блоків йде повнозв’язна частина моделі, де спочатку використовується шар сплющення, для перетворення масиву даних в одновимірний вектор, а потім, як і раніше, до нього також йдуть шари нормалізації та дропауту, для зменшення переобучення. І останнім шаром є лінійний шар з 10 нейронами, де буде також функція softmax, що зробить оцінки ймовірностей приналежності зображення до кожного з 10 класів.

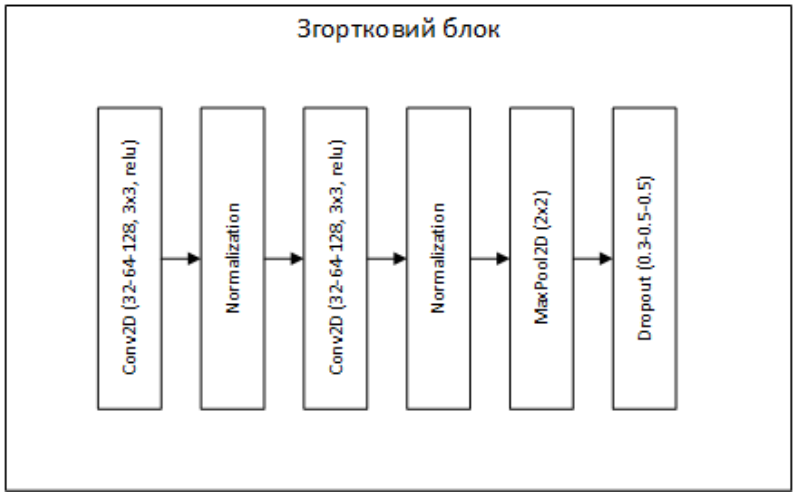

Рисунок 2.2 – Схема згорткового блоку моделі нейромережі

2.3 Архітектура системи розпізнавання

Схема програмної системи розпізнавання зображена на рисунку 2.2. Загалом це буде додаток для комп'ютерних систем з інтерфейсом користувача. Мета додатку полягає у зручному тестуванні моделі та відображенні вихідних результатів розпізнавання. Ця програма буде використовувати вже попередньо натреновану модель. Архітектуру програми можна поділити умовно на кілька ключових логічних компонентів, кожен з яких відповідає за функціональний блок.

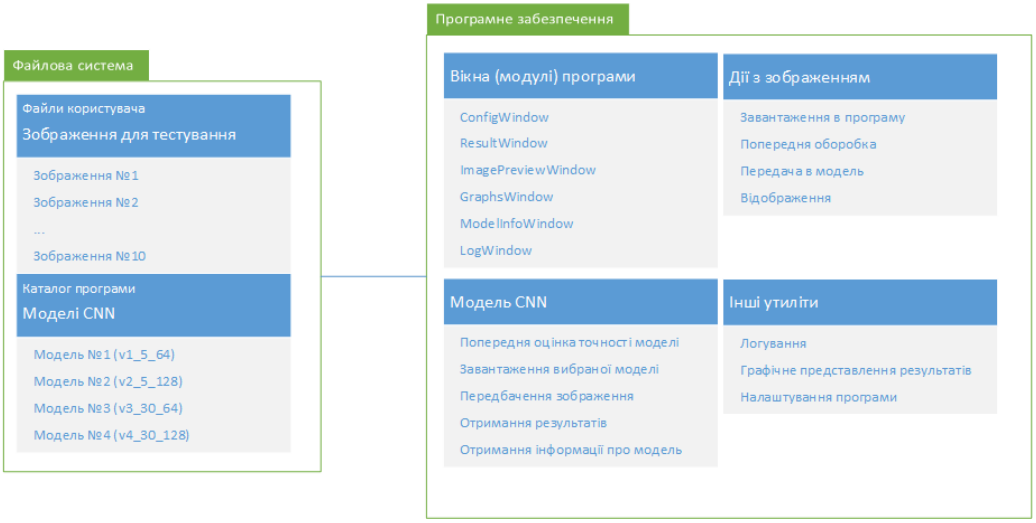

Рисунок 2.2 – Архітектура поточної програми, що розробляється

Блок вікон (модулів) програми вміщає в собі різні вікна та їх базові елементи інтерфейсу, що будуть потрібні для тестування моделі. ConfigWindow є першим і осовним вікном програми, де будуть усі потрібні налаштування перед тестуванням моделей нейромереж: кнопка вибору зображення, кнопка початку передбачення, випадаючий список, з якого можна вибрати, яка саме модель буде розпізнавати зображення, а також поля для галочок, що визначатимуть які додаткові вікна інформації відкриватимуться разом з завершенням передбачення. Всього таких вікон буде п’ять: ResultWindow, де будуть представлені результати передбачення у відсотках; ImagePreviewWindow, де відображатиметься попередній перегляд вибраного зображення; GraphsWindow, де будуть накреслені графіки тренування моделей; ModelInfoWindow, де можна переглянути основну інформацію про вибрану модель, та LogWindow, що буде слугувати дурналом подій, і текст якого можна буде зберегти у окремий текстовий файл. Також зберегти іншу інформацію в окремий текстовий файл можна буде і в ModelInfoWindow та в ResultWindow.

Блок дій з зображенням містить ряд функціоналу, що виконується у програмі та напряму взаємодіє з зображенням. Завантаження зображення в програму виконуватиметься на стадії налаштування ConfigWindow, де користувач зможе вибрати зображення із файлової системи. Також буде попередня обробка зображення перед його подальшим відправленням у нейромережу. Модель нейромережі отримає це оброблене зображення та вже у своєму блоці буде виконувати потрібні дії. Зображення відображатиметься у допоміжному вікні ImagePreviewWindow.

Блок моделі CNN має функціонал, напряму пов’язаний з моделлю нейромережі. Він містить завантаження попередньо навченої моделі з файлової мережі (каталогу програми), що завантажується після вибору моделі та старту передбачення. Також повинна бути попередня оцінка точності вибраної моделі, що буде відображена в модулі інформації про

модель, куди також буде передано структуру моделі model_summary. Оновним функціоналом даного блоку програми є передбачення зображення. Обчислюється ймовірності належності до тих чи інших із 10 класів, результати сортуються та передаються для подальшого відображення у ResultWindow.

Блок інших утиліт містить допоміжний функціонал, що виконується на протязі роботи програми, а саме логування основних процесів програми, графічне представлення результатів та налаштування програми.

2.4 Вибір та підготовка навчальних даних

Для реалізації та тестування моделі було обрано публічно доступний набір зображень CIFAR-10, який містить 60 000 кольорових зображень розміром 32 квадратних пікселі, розділених на 10 класів (рисунок 2.3): літак, автомобіль, птах, кіт, олень, собака, жаба, кінь, корабель і вантажівка. Загалом цей датасет включає п’ятдесят тисяч зображень, що виділені для навчання, а також десять тисяч для тестування. Кожен клас на себе загалом має по шість тисяч зображень. Серед його переваг варто відзначити зручний формат і невеликий розмір вже підготовлених та заздалегідь оброблених зображень. Це дуже актуально в поточній роботі, так як дозволяє тренувати моделі навіть на системах з обмеженими ресурсами. Також цей набір має широку популярність у науковій спільноті, тож буде просто порівняти результати роботи моделі для перевірки працездатності та точності на прикладі інших моделей, що базуються на цьому датасеті.

Класи повністю виключають один одного, тобто іншими словами, між класами немає збіжностей. Наприклад: автомобілі не містять зображення вантажівок і навпаки, хоч обидва класи являються наземним транспортом. Клас автомобіля включає в себе седани, позашляховики тощо. Клас вантажівок включає тільки великі вантажівки. Ні те, ні інше не включає

пікапи [10].

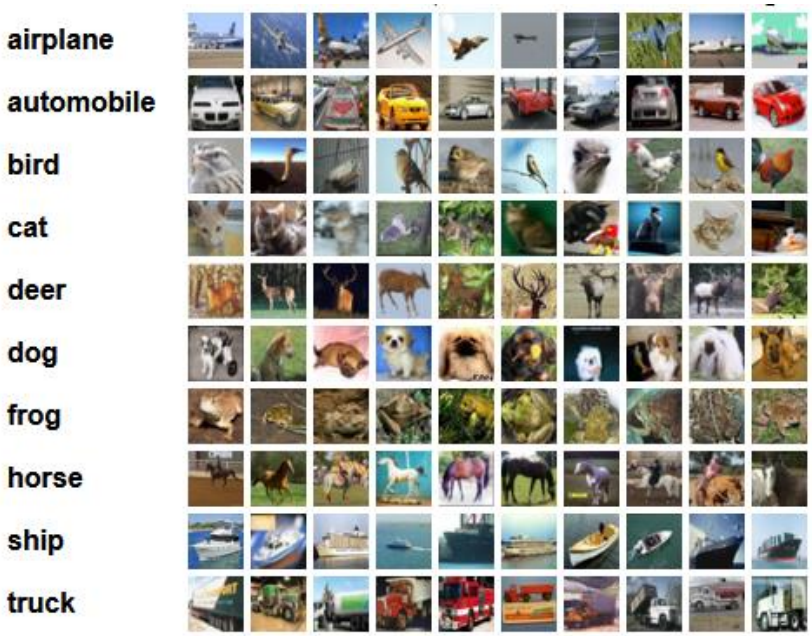

Рисунок 2.3 – Демонстрація класів в датасеті CIFAR-10, а саме 10 випадкових зображень з кожного класу

Перед подачею зображень на вхід нейронної мережі необхідно буде виконати їх попередню підготовку, а саме нормалізувати піксельні значення, тобто привести всі значення пікселів до діапазону від 0 до 1, що покращить збіжність навчального процесу. Потрібно також буде перетворити мітки класів у формат one-hot encoding, де кожна мітка повинна бути представлена як вектор, у якому лише одна позиція дорівнює одиниці, а решта це нулі. Такий формат є стандартним для багатокласової класифікації в нейронних мережах. Також необхідно розділити дані на навчальну та валідаційну вибірки: приблизно 90% зображень виділяється для навчання, а 10% для валідації, щоб можна було відстежити якість навчання та у випадку запобігти переобученню.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Середовище розробки та інструменти

Для створення та тренування моделі на базі CNN та середовища її тестування вуло використано наступне програмне та апаратне забезпечення. Комп’ютерна система (КС) на основі процесора Ryzen 5 3400G з 4 ядрами з 16 GB RAM, що використовує архітектуру Zen+, а також оснащений інтегрованою графікою Radeon RX Vega 11, що і використовується в поточній системі в якості основного графічного ядра, тому усі операції проводитимуться суто на CPU. КС працює в оболонці операційної системи Windows 11. Основною мовою програмування для реалізації всіх етапів роботи використовується Python 3.11. Середовище розробки - IDE PyCharm Community версії 2024.3. Далі йде перелік основних бібліотек, що використовувались в роботі:

- Tensorflow Keras для побудови та навчання нейронної мережі;
- NumPy для числових обчислень;
- Matplotlib для візуалізації результатів.

Набір даних CIFAR-10 був отриманий з їх основного репозиторію через tf.keras.datasets.cifar10.

3.2 Побудова та налаштування моделі

Для створення моделі на початку завантажуються ключові бібліотеки для роботи з нейронними мережами, робиться це через pip install, як наприклад бібліотека tensorflow (рис. 3.1).

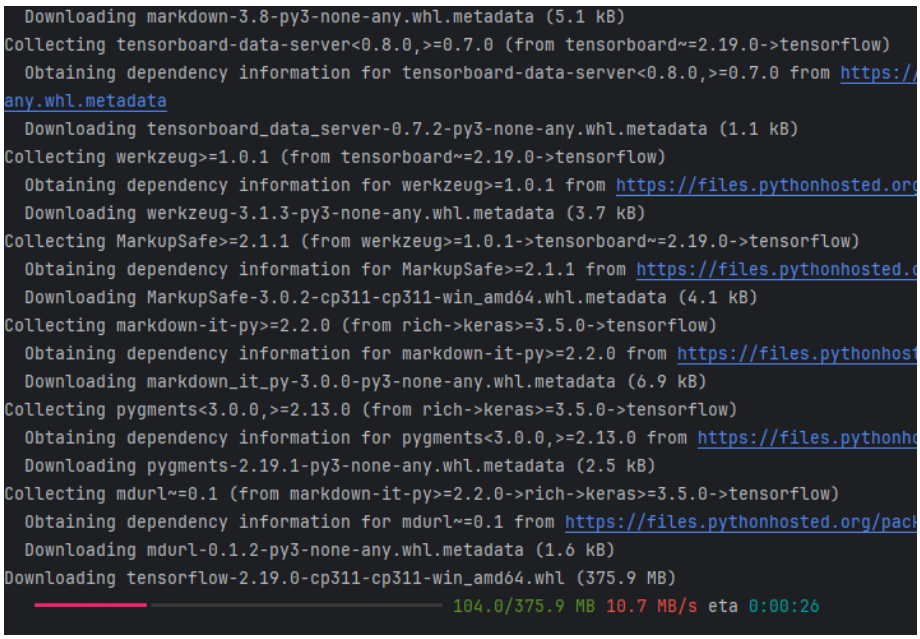

Рисунок 3.1 – Процес завантаження бібліотеки tensorflow

Після завантаження основних бібліотек створюється головний .py сценарій, де буде код побудови та тренування моделі. Імпортуються ті самі бібліотеки та їх основні компоненти для роботи сценарію (рис 3.2), де tensorflow це основна бібліотека для машинного навчання, keras це високорівневий API для побудови і навчання моделей що входить до складу tensorflow, layers і models – два модулі, що відповідають за різні шари нейронних мереж та методи створення моделей, і utils – модуль з допоміжними функціями, які в основному знадобляться на етапі підгтовки даних.

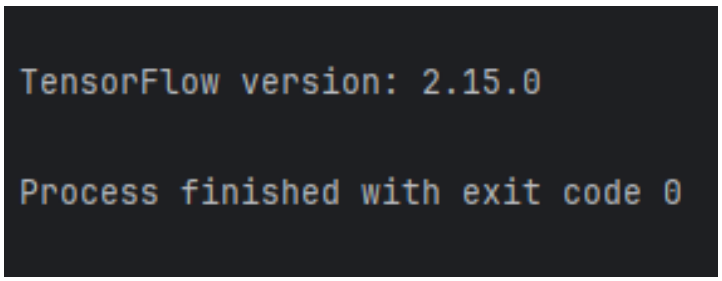

Рисунок 3.2 – Повідомлення про успішність імпортування бібліотеки tensorflow

Далі завантажується набір даних. Оскільки використовується інтернет-ресурс, то і завантажувати його треба з репозиторію. У API keras для цього є зручний інтерфейс, що дозволяє в один рядок завантажити увесь набір даних. Причому підтримується ще багато різноманітних датасетів, зокрема нам потрібний CIFAR-10 (рис. 3.3). Функція load_data() повертає кортеж із навчального та тестового наборів даних, тому змінні використовуємо відповідні, де x_train – масив зображень для навчання, y_train – мітки до навчальних зображень, а x_test та y_test – масив і мітки для тестових зображень відповідно.

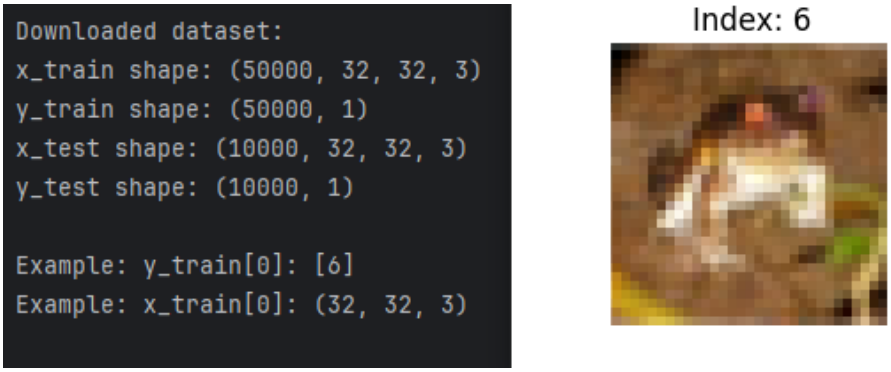

Рисунок 3.3 – Завантаження датасету та його перевірка

Після етапу завантаження даних йде їх попередня обробка. Для початку їх потрібно перетворити з цілих чісел у числа з плаваючою комою. Дані початково складаються з пікселів у діапазоні від 0 до 255, що відноситься до типу uint8, тобто 8-бітове ціле число. Найоптимізованішим варіантом для нейромереж буде використання нормалізованих вхідних даних, зазвичай це діапазон від 0 до 1 (або від -1 до 1). Тому перед діленням треба переконатися, що дані мають тип float, інакше ділення цілих чисел може призвести до втрати точності. Після зведення до чисел с плаваючою комою як раз наступним йде їх нормалізація, про яку було зазначено раніше (рис. 3.4).

```
x_train shape: (50000, 32, 32, 3)
y_train shape (one-hot): (50000, 10)
x_test shape: (10000, 32, 32, 3)
y_test shape (one-hot): (10000, 10)

Example: x_train[0][0][0]: [0.23137255 0.24313725 0.24705882]
One-hot example: y_train[0]: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

Process finished with exit code 0
```

Рисунок 3.4 – Демонстрація форматування набору даних

Окрім наборів даних також потрібно потурбуватися про мітки цих даних, тобто y_train та y_test. На рисунку 3.4 зображене перетворення міток класів, що спочатку виглядають приблизно так – [[3], [1], [2], [0]…], у one-hot вектори, що перетворить їх у приблизно наступний вигляд, де початкова мітка 0 дорівнює вектору зі значеннями [1 0 0 0 0 0 0 0 0 0], де 1 – це бінарне позначення мітки класу, які варіюються від 0 до 9. Необхідність такого формату міток даних буде пояснено пізніше на етапі формування моделі.

Після проведення основних операцій для підготовки даних можна приступати до реалізації моделі. Лістинг 3.1 містить повний код визначення моделі. Її архітектуру вже було розібрано у попередніх розділах, тому зосереджуватимось на програмній частині та опустимо структуру. На початку в першому вхідному шарі задається input_shape, де розмірність даних 32 на 32 пікселі з трьома каналами RGB. Для простоти поділимо код на 3 основних згорткових блоки та розглянемо їх. Перший блок має 2 згорткових шарів Conv2D з 32 фільтрами та наступні допоміжні шари: BatchNormalization() стабілізує навчання, нормалізуючи вхідні дані. Застосовується перетворення, яке підтримує середнє значення виходу близьким до 0, а стандартне відхилення виходу приблизно біля 1. MaxPooling2D(), що зменшує роздільну здатність вхідних даних вздовж його просторових вимірів (висоти і ширини), беручи максимальне значення у вхідному вікні, розмір якого визначається параметром pool_size, для кожного

каналу вхідних даних. Результуючий вихідний результат, при використанні параметра «valid» (за замовчуванням, тому в коді він явно не прописаний), має наступну просторову формулу:

$$O = f(\frac{(i-p)}{s}) + 1,\ i >= p \tag{1.1}$$

Де O – вихідна форма, f – формула math.floor(), що завжди округлює до меншого значення та повертає найбільше ціле число, менше або рівне заданому числу; i – вхідна форма, p – розмір просторового вікна (pool_size); s – кроки, що визначають наскільки зміщується вікно. Якщо не задано явно, за замовчуванням використовується значення pool_size.

Dropout() випадковим чином встановлює вхідні одиниці на нулі з частотою швидкості на кожному кроці протягом часу навчання, щоб запобігти перенавчанню. Вхідні дані, що не встановлені нулями, масштабуються на 1/(1 - швидкість) таким чином, що сума за всіма вхідними даними залишається незмінною.

Другий блок має аналогічну структуру та різниться лише кількістю фільтрів 64 та більшим значенням Dropout. Третій слідує аналогічному принципу, переходячи на 128 фільтрів. Вихідний блок містить наступні шари: Flatten() перетворює багатовимірний набір даних в одновимірний вектор, Dense() являється повнозв’язним шаром, набір даних проходить ту ж саму процедуру нормалізації та скидання за допомогою BatchNormalization() та Dropout() відповідно, та завершальним шаром буде шар класифікації, що використовує той самй Dense() з відповідною кількістю классам набору даних.

Лістинг 3.1 – Побудова моделі (файл train.py)

```
model = models.Sequential()
```

```
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(32,32,3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.5))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))    # num_classes = 10
```

Наступний код програми компілює побудовану модель та проводить її навчання. Параметр optimizer у метода model.compile() використовує оптимізатор adam. Взагалі оптимізація Адама це стохастичний метод градієнтного спуску, що базується на адаптивній оцінці моментів першого та другого порядку. Метод є обчислювально ефективним, має невеликі вимоги до пам'яті, є інваріантним до діагонального масштабування градієнтів та добре підходить для задач, які є великими за обсягом даних чи параметрів [11]. Categorical_crossentropy це функція втрат для багатокласової класифікації. Використовується разом із one-hot мітками, тобто те, що було виконано ще на початку перед формуванням структури моделі, а саме де було виконано функцію to_categorical(). Параметр metrics приймає аргумент «accuracy», що буде виводити точність на тренуванні і валідації після кожної епохи для спостереження.

Метод model.fit() приймає x_train та y_train – тренувальні попередньо

оброблені дані та мітки тренувальних даних відповідно. Параметр batch_size відповідає за те, скільки зразків проходять через модель за один крок, аргументом якого було вибрано число 32. Значення 32 було обрано відповідно до потужності системи. За них відповідає параметр epochs, що в даному зразкові коду приймає значення 30. Епохи визначають те, скільки разів модель бачить та проходить через увесь датасет. Чим більша кількість епох, тим довший процес тренування, але й результат буде більш точним на виході. Зазвичай саме цей параметр найвпливовіший на кількість затраченого часу на тренування, так як сам процес перегляду моделлю датасету хоч і займає деякий абстрактний проміжок часу, але при цьому кожна нова епоха буде повторювати той же процес, що помножує цей час на кількість епох. Останній явно присвоєний параметр validation_data відповідає за дані для валідації після кожної епохи. Це дає змогу бачити, чи не перенавчається модель.

Змінна history приймає усі дані про тренування з методу model.fit(), зокрема втрати на тренуваннях по епохах, точність на тренуваннях, втрати на валідації та точність на валідації. Ці дані можна буде як переглядати в реальному часі під час тренування моделі, так і використати потім після тренування для відображення статистичної інформації за допомогою графіків, діаграм тощо.

Рисунок 3.5 – Процес тренування моделі нейромережі

На рисунку 3.5 зображений скріншот консолі під час процесу тренування моделі нейромережі. Відображається поточна, та загальна кількість епох, що повинна пройти модель, кількість батчів, що обробляється,

час, що затрачений на одну епоху та скільки мілісекунд займає один крок. Відображається поточні втрати та точність на тренувальній та валідаційній вибірках.

Одним із завершаючих етапів сценарію йде остання перевірка точності моделі на тестових даних x_test y_test. Перед цим, як і перед тренуванням моделі потрібно також попередньо обробити ці тестові дані, щоб переконатися, що тестові дані мають тип плаваючої коми і масштабовані до діапазону від 0 до 1. Метод model.evaluate() запускає повну перевірку моделі на невідомих для неї тестових даних, так як x_test та y_test модель ще не бачила. Метод повертає значення втрат test_loss і метрики test_acc, тобто точність, що були вказані під час компіляції моделі model.compile(). Параметр verbose зі значенням 2 є перемикачем для виводу стислої інформації у консоль. Та окрім цього ще присутній print() що і виводить простий текст у консоль з точністю test_acc. Фінальним штрихом буде збереження моделі у окремий .keras файл, а також збереження історії тренування моделі у файл формату .pkl, бо їх використання вже буде задіяне в іншій програмі. Окремо повідомимо про успішність збереження моделі для більшої флюїдності коду під час дебагінгу у випадку помилок. Те, як виглядатиме кінцевий результат тренування моделі зображено на рисунку 3.6. Значення точності та втрат у цьому скріншоті не відображають дійсності.

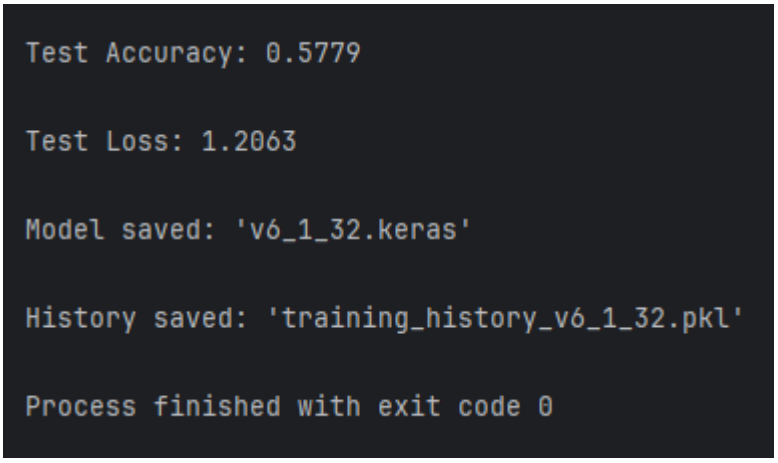

Рисунок 3.6 – Приклад результату тренування моделі

3.3 Ключові модулі програми

Основна програма складається з декількох сценаріїв – main.py, qt_imports.py та сценарії вікон програми – config_window.py, image_preview_window.py, results_window.py, graphs_window.py, model_info_window.py. Сценарій main.py виконує функцію ініціалізації програми, модуль qt_imports.py містить більшість імпортів сторонніх бібліотек, непов’язаних з нейромережами, таких як qt5 для роботи з віконними додатками, config_window.py є основним, так як там проходить більшість усього функціоналу програми, включаючи роботу з інтерфейсом, підключення нейромережі, розпізнавання вибраного зображення тощо. Модуль image_preview_window.py містить реалізацію вікна для перегляду вибраних зображень, модуль results_window.py – вікно результатів розпізнавання, модуль graphs_window.py – вікно графічного представлення процесу навчання вибраної моделі нейромережі, а model_info_window.py – вікно, де можна переглянути структуру моделі та іншу інформацію про неї.

Виконуються основні підключення бібліотек, де знов ж таки вже відомий tensorflow, його API keras, допоміжні бібліотеки numpy для роботи з числовими даними та matplotlib для графічного представлення результатів.

На частині користувацького інтерфейсу дуже сильно загострювати увагу не має сенсу, тому буде продемонстровано основні рядки коду, що прямо відносяться до нейромережі та її прямого функціоналу пов’язаного з нею. Будується частина віконного додатку класу ConfigWindow що наслідується від QWidget, де присутні такі інтерфейсні елементи, як от кнопка вибору зображення QPushButton, до кліку на яку прив’язаний внутрішній метод класу pick_image(), що відкриває модальне вікно, через яке користувач програми зможе обрати одне, або декілька зображень для подальшого їх розпізнавання (рис. 3.7).

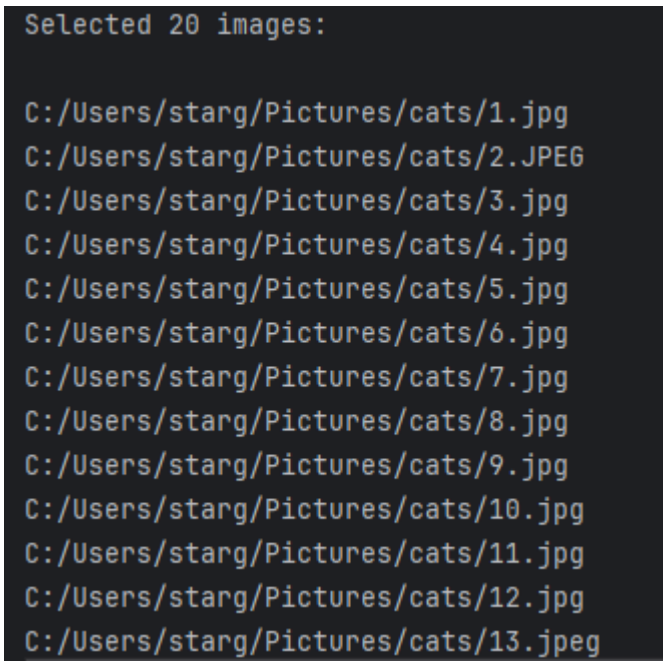

Рисунок 3.7 – Вибір зображеннь

Потрібно зауважити, що оскільки це інша програма, що ніяк не пов’язана з попереднім сценарієм train.py окрім його кінцевого результату у вигляді файлу моделі нейромережі, потрібно провести ще раз ту саму процедуру з початку, тобто завантажити дані з датасету, зокрема саме тестові дані, оскільки тренувальні в кінцевій програмі вже не потрібні, що здійснюється за допомогою пустих touple з «заглушками» там, де повертаються тренувальні дані. Ну і проведення попередньої обробки, такої ж, що проводилася до цього, інакше модель відмовлятиметься працювати з іншим форматом даних.

Метод start_prediction() це ключовий метод, що здійснює класифікацію обраного користувачем зображення. Спочатку обробляються чекбокси налаштувань для того, щоб визначити, які вікна відкривати після закінчення розпізнавання зображення. Далі йде виклик завантаження моделі та запуску передбачення за допомогою load_model(model_name), де ключовим аргументом є model_name, що був вказаний у випадаючому списку. Після завантаження моделі відбувається одноразова її оцінка точності. Завантажується з датасету тільки тестова частина (x_test, y_test) =

cifar10.load_data(). Y_test преобразовується з форми (1000, 1) до одновимірного вектора за допомогою flatten(). X_test нормалізується до діапазону [0,1] з [0, 255]. Y_test також перетворюється у формат міток one-hot encoding за допомогою tf.one_hot(), бо модель очікуватиме такий формат для багатокласової класифікації. Далі оцінюється точність та втрати моделі за допомогою model.evaluate(). Отримується текстовий опис структури моделі для ModelInfoWindow через get_model_summary_text(). Також завантажується історія тренування моделі. Метод classify_image() що приймає параметром масив шляхів до зображення, класифікує їх та повертає результати класифікації model_results, що виводиться у консолі та передається у ResultsWindow. Залежно від активних чекбоксів викликаються відповідні методи для відображення додаткових вікон.

```
-- Start Arguments --
Model: v1_10_64
Image: ['C:/Users/starg/Pictures/cats/1.jpg',
Logging: False
Graphs: False
Model Info: False
Preview Image: False
Show Results: False
```

Рисунок 3.8 – Виведення інформації після запуску у консоль

3.4 Візуалізація результатів розпізнавання

На рисунку 3.9 зображений скріншот основного початкового вікна програми ConfigWindow. Натиснувши на кнопку «Select Image» відкриється вікно вибору одного або декількох файлів з розширенням .png, .jpg, .jpeg, .bmp та інших розширень файлів, що відповідають зображенню. Після вибору зображення, у верхній частині програми з’явиться повідомлення про кількість обраних зображень. Щоб почати передбачення зображення потрібно натиснути кнопку «GO». Але перед цим варто виставити потрібні

чекбокси, щоб відобразились ті вікна, які потрібні для тестування. Можна обрати як усі вікна разом, проставивши галочки під кожним пунктом, та можна обрати лише деякі, якщо немає потреби у інших. Оберемо усі пункти для наглядності, щоб продемонструвати усі вікна програми.

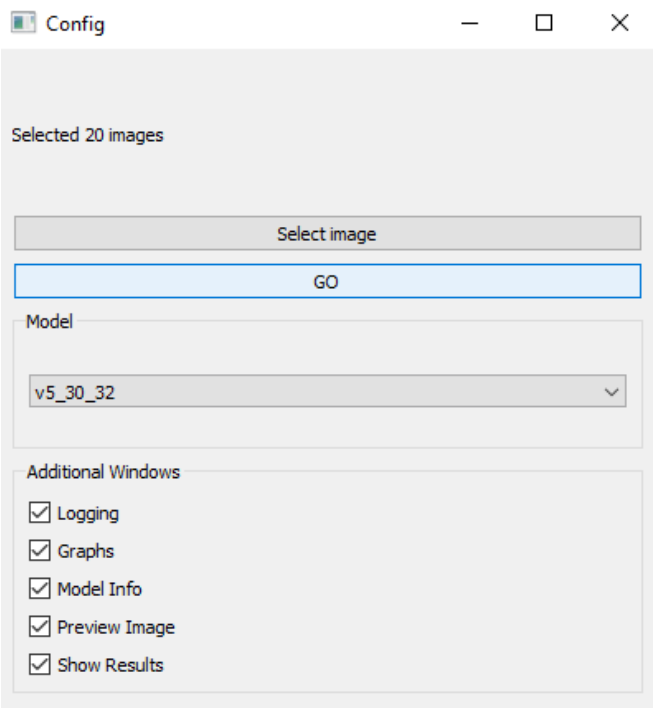

Рисунок 3.9 – Головне вікно програми ConfigWindow

На рисунку 3.10 зображено основне вікно результатів розпізнавання. У цьому вікні відображені усі обрані зображення, вказаний шлях до цих зображень та їх порядковий номер. Навпроти кожного зображення відображено клас, який був розпізнаний у зображенні та його відсоток вірогідності. Також є функція округлення результатів до сотих і збереження результатів у текстовий файл під назвою results.txt.

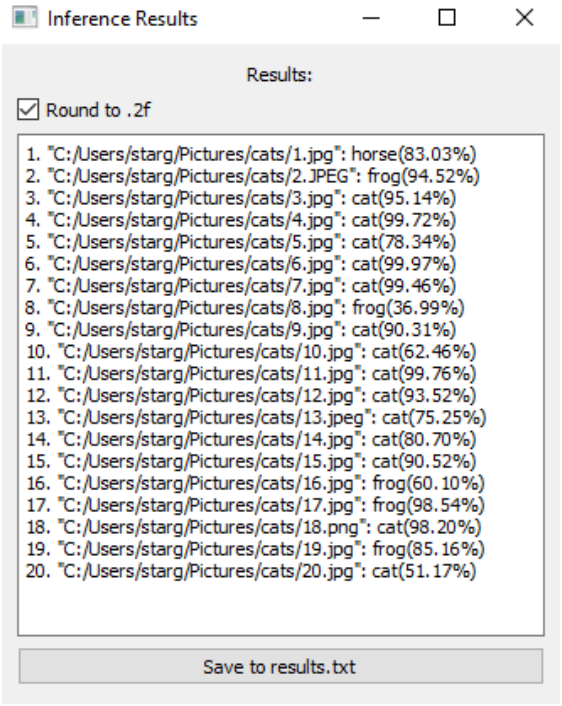

Рисунок 3.10 – Вікно програми ResultsWindow

На рисунку 3.11 зображене вікно ImagePreviewWindow, де можна переглянути усі обрані зображення. Перемикатися між ними можна за допомогою стрілочок, а назва та шлях до файлу кожного зображення відображується зверху.

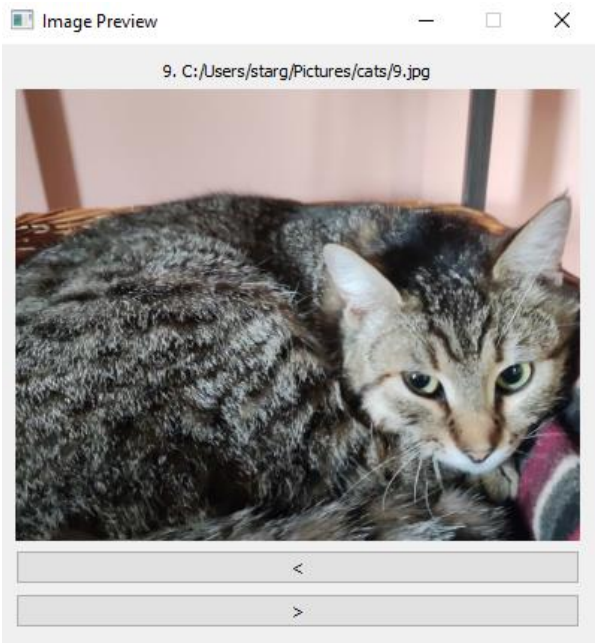

Рисунок 3.11 – Вікно програми ImagePreviewWindow

На рисунку 3.12 зображено вікно програми GraphsWindow. Відображені графіки точності та втрат на тренувальній та валідаційній вибірках під час навчання моделі нейромережі, де синій це тренувальна вибірка, а помаранчевий це валідаційна вибірка.

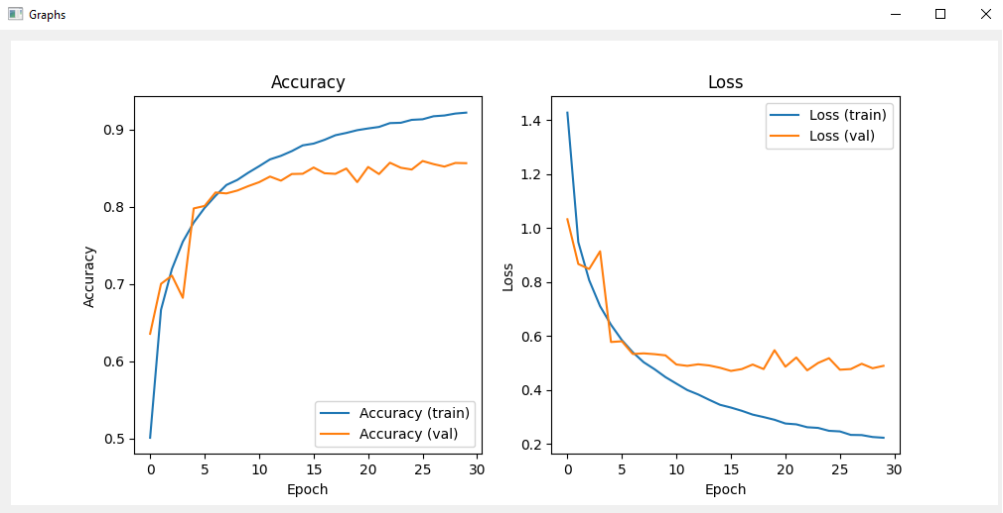

Рисунок 3.12 – Вікно програми GraphsWindow

На рисунку 3.13 зображено вікно програми LogsWindow. Тут відображється журнал подій, де нотуються усі важливі події, що відбулися у процесі роботи програми на різних стадіях, від початку її роботи, до передбачення зображень моделями.

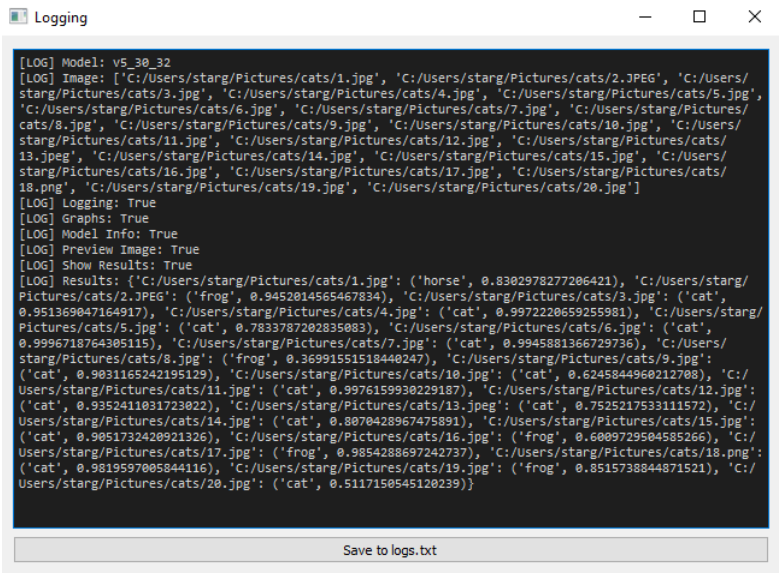

Рисунок 3.13 – Вікно програми LogsWindow

На рисунку 3.14 зображено вікно програми ModelInfoWindow. У цьому вікні демонструється структура та тип моделі нейромережі, оцінка точності та втрат на валідаційній вибірці, кількість параметрів та інше.

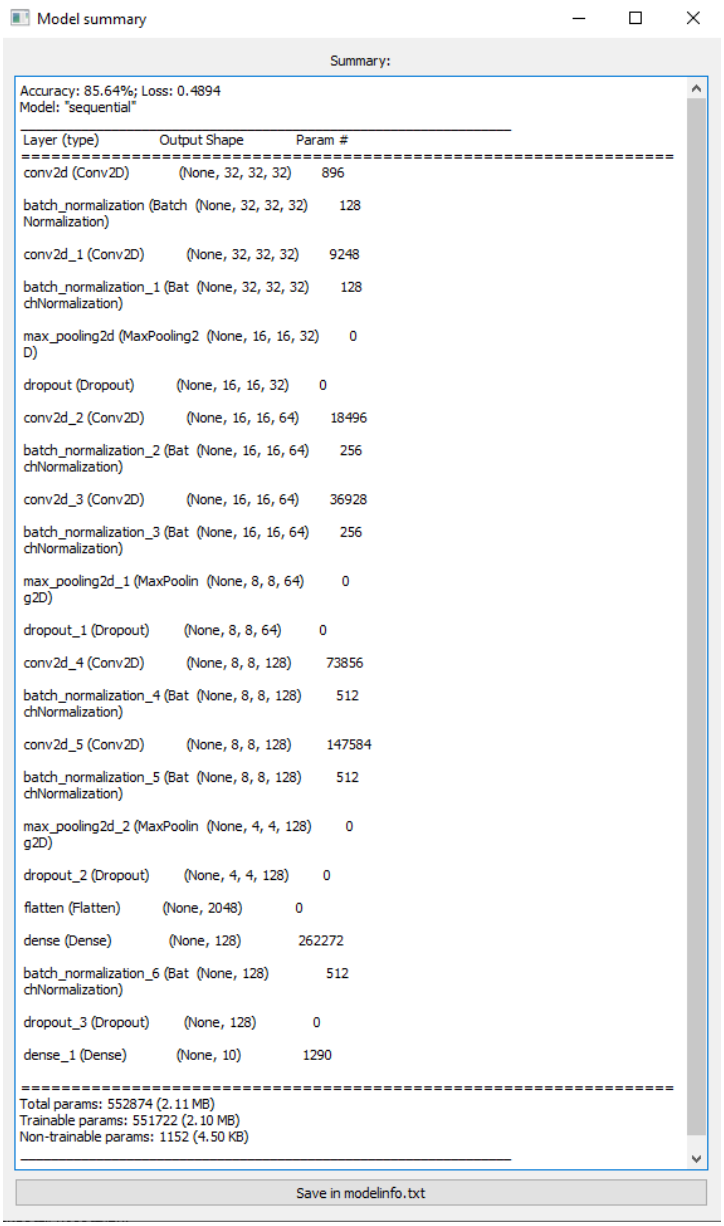

Рисунок 3.14 – Вікно програми ModelInfoWindow

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Результати тестування

Після успішного тренування декількох версій моделі та створення для них програмної оболонки для тестування, було проведено порівняння результатів розпізнавання котів на 20 зображеннях з різних джерел та з різними вхідними параметрами. Вихідні значення результатів передбачення, що ці зображення саме класифікуються як зображення котів, наведені в таблиці 4.1.

Таблиця 4.1 – Вірогідності розпізнавання зображень котів різних моделей

Зображення, №	Вірогідності розпізнавання, %	
	V1_10_32	V2_30_32
1	55.45	81.38
2	10.27	24.23
3	95.83	88.40
4	97.35	99.87
5	16.57	99.72
6	99.10	99.97
7	98.61	99.46
8	39.72	56.32
…	…	…
20	18.93	51.17
Середнє знач.	47.84	71.22

4.2 Аналіз результатів тестування

Для початку варто приділити увагу вхідним даним цього тестування. Зображення, що були використані для збору даної вибірки вірогідностей розпізнавання (табл. 4.1), спеціально були відібрані вручну так, щоб тестування вийшло максимально правдоподібним, та мало під собою реалістичний характер. Це значить, що кожне з цих десяти зображень є унікальним, має різні початкові розміри, різне освітлення, попереньо оброблене/необроблене, може бути звичайною фотографією, снятою на смартфон тощо. Але у всіх зображень є одна спільна однакова характеристика, щоб підходити для моделей – вони всі мають приблизне співвідношення 1:1, тобто ширина і висота зображень приблизно однакова, проте загальна кількість пікселів необов’язково співпадає. Зображення №1 є фотографією кота в незвичній позі та з його відзеркаленням у дзеркалі. №2 є нейтральним поміж усих зображень, хоч і фон трохи зливається з кольором шерсті кота. №3 та №4 мають котів зі схожою позою, але відрізняються тим, що №4 має повністю білий фон, а №3 звичайний фон кімнати, тощо. Тобто усі зображення відрізняються, щоб досягти більш різносторонніх результатів розпізнавання.

Перше, що кидається в очі під час перегляду результатів, це явна вища точність другої моделі над першою. Оскільки друга модель пройшла через 30 епох під час тренування, а перша лише через 10, можемо спостерігати явну перевагу в передбаченні зображень з більш складними і комплексними вхідними. Там де зображення більш стандартизоване, наприклад зображення №4, різниця між обома моделями не сягає значного розриву, а лише досягає ~2%.

Хоч і друга версія моделі краще за першу, проте у обох спостерігаються неправильне передбачення, що свідчить про недостатній обчислювальний ресурс та недоліки в архітектурі, що підлягають подальшому покращенню.

ВИСНОВКИ

За підсумком кваліфікаційної роботи були розроблені та натреновані моделі CNN, що показали свою достатнью ефективність навіть при низькому рівні архітектури та ресурсних обмеженнях, що дає підстави вважати сферу стрімко розвиваючоюся, хоч і моделі мають суттєві недоліки, що можна виправити при наявності належного технічного обладнання. Було створено програмну оболонку для використання та тестування моделей нейромереж, що має основні компоненти для взаємодії та огляду їх архітектури та точності у розпізнаванні вибраних зображень з файлової системи.

Проведено аналіз та продемонстровано, що моделі з різним рівнем підготовки показують різний рівень ефективності, знаходження більш складних ознак, розпізнавання об’єктів на більш комплексних поверхнях, з модифікованою структурою зображення та з різним тоном кольорів та рівнем освітлення: модель з вищим рівнем підготовки в середньому домінує приблизно на 25%. Зображення, що були відібрані для тестування є такими, що відображають реалістичний сценарій роботи програми, коли на вхід подається необроблене поверхнево звичайне зображення, що може статися у програмах, що використовують дані типи моделей та виконують подібні задачі класифікації, коли користувач може використати зображення поганої якості або фотографію з замиленням, затемненням тощо.

Таким чином, результати роботи підтверджують, що для подібного роду задач дані нейромережі можуть використовуватися в роботі у різних галузях, де неодмінно знадобляться технології розпізнавання, класифікації зображень, проте і мають суттєві недоліки, що підлягають подальшому вивченню та редагуванню.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is image segmentation: the basics and key techniques [стаття] / Mindy Support, 2022.

2. An overview of Object Tracking: Use Cases, Challenges, and Applications [стаття] / Sertis Vision Lab : Sertis AI Research, 2024.

3. Mia Morton : Anomaly Detection with Computer Vision [стаття] / Medium, 2020.

4. What is facial recognition in computer vision? [документація] / Milvus, 2025.

5. Maksim Markov : Traditional machine learning algorithms for machine vision [стаття] / Kapernikov, 2022.

6. Zoumana Keita : An introduction to Convolutional Neural Networks (CNNs) [стаття] / Datacamp, 2023.

7. What is unsupervised learning? [стаття] / IBM, 2021.

8. Thomas R., Zikopoulos P., Soule K. : AI Value Creators: Beyond the Generative AI User Mindset [книга] – Sebastopol: O'Reilly Media, 2025. 388 с.

9. Mordvintsev A., Rahman K. A. : Introduction to OpenCV-Python Tutorials [документація]: Open Source Computer Vision, 2025.

10. Krizhevsky A. : Learning Multiple Layers of Features from Tiny Images [книга], 2009.

11. Kingma D. P., Ba J. : Adam: A Method of Stochastic Optimization [книга]: 3rd International Conference for Learning Representations, San Diego, 2015.

12. Bangar S. : VGG-Net Architecture Explained [стаття]: Medium, 2022.

13. Chollet F. Deep Learning with Python. Second Edition. [книга] – Shelter Island, NY : Manning Publications Co., 2021. – 466 с.