

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 проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДІВ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННІ
ЗА ДОПОМОГОЮ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ
RESEARCH AND SOFTWARE IMPLEMENTATION
OF METHODS FOR OBJECT RECOGNITION IN IMAGES
USING CONVOLUTIONAL NEURAL NETWORKS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-142м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОЗЛОВ В.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СУББОТІН С.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОЗЛОВА Валерія Валентиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів розпізнавання об'єктів на зображенні за допомогою згорткових нейронних мереж. Research and Software Implementation of Methods for Object Recognition in Images Using Convolutional Neural Networks

керівник проєкту (роботи) д.т.н., професор, СУББОТІН Сергій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Аналіз та проєктування архітектури нейромережевих моделей. 3. Проєктування архітектури програмного забезпечення. 4. Реалізація компонентів програмного забезпечення. 5. Тестування та експериментальне дослідження програмного забезпечення. 6. Керівництво програміста. 7. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-7 Основна частина	СУББОТІН С.О., професор		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “08” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Опис керівництва програміста та користувача	9 тиждень	Розділ 6, 7
8	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
9	Нормоконтроль та рецензування.	12 тиждень	
10	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Валерій КОЗЛОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій СУББОТІН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
134 с., 18 табл., 68 рис., 2 дод., 53 джерела.

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ,
КЛАСИФІКАЦІЯ, РОЗПІЗНАВАННЯ ОБ'ЄКТІВ, VGG, YOLOV1,
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, PYTHON, PYTORCH, PYQT.

Об'єкт дослідження – процес класифікації та розпізнавання об'єктів на зображенні.

Предмет дослідження – методи та алгоритми класифікації та розпізнавання об'єктів на зображенні за допомогою нейронних мереж.

Мета роботи – дослідження та програмна реалізація методів розпізнавання об'єктів на зображенні за допомогою згорткових нейронних мереж.

Матеріали, методи та технічні засоби: інтерпретована високорівнева мова програмування Python, програмні пакети PyTorch та PyQt, інтегроване середовище розробки PyCharm, операційна система Windows 10.

Результати. Експериментально доведено доцільність та ефективність використання запропонованих підходів до вдосконалення існуючих архітектр нейромережевих моделей VGG та YOLOv1.

Висновки. Розроблене програмне забезпечення надає можливості навчання та використання нейромережевих моделей з базовою та вдосконаленою архітектурою VGG та YOLOv1 для вирішення прикладних задач класифікації та розпізнавання об'єктів на зображенні на довільному наборі даних.

Галузь використання – класифікація та розпізнавання об'єктів на зображенні в програмних системах аналізу та обробки графічних даних.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 134 pages, 18 tables, 68 figures, 2 appendixes, 53 sources.

ARTIFICIAL NEURAL NETWORKS, CONVOLUTIONAL NEURAL NETWORKS, CLASSIFICATION, OBJECT DETECTION, VGG, YOLOV1, SOFTWARE, PYTHON, PYTORCH, PYQT.

The object of research is the process of classification and object detection in images.

The subject of research is methods and algorithms for classifying and object detection in image using artificial neural networks.

The goal of the work is research and software implementation of methods for recognizing objects in an image using convolutional neural networks.

Materials, methods and technical means: interpreted high-level programming language Python, software packages PyTorch and PyQt, integrated development environment PyCharm, Windows 10 operating system.

Results. The expediency and effectiveness of the proposed approaches to improving the existing architectures of the neural network models VGG and YOLOv1 have been experimentally proved.

Conclusions. The developed software provides the ability to train and use artificial neural network models with the basic and improved architectures of VGG and YOLOv1 to solve applied problems of classification and object detection in image on an arbitrary dataset.

The field of application is classification and object detection in images in software systems for analyzing and processing graphic data.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Огляд задачі розпізнавання об'єктів на зображенні.....	11
1.1.1 Актуальність та необхідність розпізнавання об'єктів	11
1.1.2 Види задач розпізнавання об'єктів на зображенні	12
1.1.3 Методи та підходи до розпізнавання об'єктів на зображенні	16
1.2 Огляд штучних нейронних мереж та їх видів	18
1.2.1 Характеристика штучного нейрона	18
1.2.2 Загальна структура нейронної мережі та її навчання.....	20
1.2.3 Види нейронних мереж	22
1.3 Аналіз архітектури згорткових нейронних мереж	24
1.3.1 Загальна архітектура згорткових нейронних мереж	24
1.3.2 Згорткові шари	24
1.3.3 Агрегувальні шари	26
1.3.4 Згладжування, повнозв'язні шари та функції активації нейронів ..	27
1.3.5 Нормована експоненційна функція та функція втрат	31
1.4 Аналіз видів згорткових нейронних мереж.....	32
1.4.1 Види згорткових нейронних мереж для класифікації зображень...	32
1.4.2 Види згорткових нейронних мереж для розпізнавання об'єктів	34
1.5 Аналіз існуючих програмних реалізацій	37
1.6 Висновки за розділом 1	38
2 Аналіз та проєктування архітектури нейромережових моделей.....	40
2.1 Дослідження та проєктування згорткових нейронних мереж для класифікації зображень	40
2.1.1 Дослідження архітектури нейромережових моделей VGG	40

2.1.2 Вдосконалення та оптимізація архітектури нейромережових моделей VGG	41
2.2 Дослідження та проєктування згорткових нейронних мереж для розпізнавання об'єктів	43
2.2.1 Дослідження архітектури нейромережової моделі YOLOv1	43
2.2.2 Вдосконалення та оптимізація архітектури нейромережової моделі YOLOv1	47
2.3 Висновки за розділом 2	48
3 Проєктування архітектури програмного забезпечення	49
3.1 Аналіз вимог до програмного забезпечення	49
3.2 Вибір засобів розробки	51
3.2.1 Вибір мови програмування	52
3.2.2 Вибір бібліотеки для роботи з нейромережевими моделями	54
3.2.3 Вибір бібліотеки для реалізації графічного інтерфейсу	56
3.3 Прототипування графічного інтерфейсу користувача	57
3.4 Висновки за розділом 3	58
4 Реалізація компонентів програмного забезпечення	59
4.1 Структура програмного забезпечення	59
4.2 Реалізовані програмні класи	61
4.3 Реалізація нейромережових моделей	66
4.4 Структура навчальних даних	67
4.4.1 Структура навчальних даних для класифікації	67
4.4.2 Структура навчальних даних для розпізнавання об'єктів	68
4.5 Реалізація графічного інтерфейсу користувача	69
4.6 Висновки за розділом 4	72
5 Тестування та експериментальне дослідження програмного забезпечення	73
5.1 Тестування програмного забезпечення	73
5.2 Експериментальне дослідження програмного забезпечення	76
5.2.1 Задача класифікації зображень	76
5.2.2 Задача розпізнавання об'єктів на зображенні	79

5.3 Висновки за розділом 5	81
6 Керівництво програміста.....	83
6.1 Призначення та умови застосування програмного забезпечення	83
6.2 Вимоги до програмного забезпечення	84
6.3 Характеристики програмного забезпечення	84
6.4 Звернення до програмного забезпечення	86
6.5 Вхідні та вихідні дані програмного забезпечення	87
6.6 Повідомлення програмісту.....	87
7 Керівництво користувача	88
7.1 Призначення програмного забезпечення.....	88
7.2 Умови виконання програмного забезпечення.....	88
7.3 Виконання програмного забезпечення	89
7.4 Повідомлення користувачу	94
Висновки	96
Перелік джерел посилань	98
Додаток А Текст програми.....	103
Додаток Б Слайди презентації	125

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CNN	– Convolutional Neural Network;
HOG	– Histogram of Oriented Gradients;
IDE	– Integrated Development Environment;
ILSVRC	– ImageNet Large Scale Visual Recognition Challenge;
RAM	– Random Access Memory;
ReLU	– Rectified Linear Unit;
RGB	– Read, Green, Blue;
SIFT	– Scale-Invariant Feature Transform;
SVM	– Support Vector Machines;
TSRS	– Traffic Signs Recognition System;
VGG	– Visual Geometry Group;
YOLO	– You Only Look Once;
БД	– база даних;
ОС	– операційна система;
ПЗ	– програмне забезпечення .

ВСТУП

Розвиток комп'ютерних та інформаційних технологій, автоматизація та інтелектуалізація різних процесів підвищують ефективність та якість життя людини. Однією з найважливіших та найскладніших складових в сучасних програмних системах є обробка графічної інформації.

Методи класифікації та розпізнавання об'єктів на зображенні застосовуються в різних галузях, починаючи від медичної діагностики і закінчуючи системами відеоспостереження. Однією з перспективних галузей впровадження технологій обробки й аналізу графічних даних є автомобільна індустрія, в якій впроваджуються системи допомоги водію у розпізнаванні дорожніх знаків.

Для вирішення задач класифікації та розпізнавання об'єктів використовують різні архітектури згорткових нейронних мереж, наприклад, VGG та YOLOv1. Кожна з мереж має певні переваги та недоліки. Такі нейромережеві моделі можуть бути вдосконалені за допомогою додавання нових чи оновлення існуючих архітектурних особливостей, що має значну практичну цінність для розробки високоефективних програмних систем.

Дослідження та вдосконалення архітектур нейромережевих моделей для класифікації та розпізнавання об'єктів на зображенні сприяє підвищенню якості та ефективності використання таких нейромережевих моделей при вирішенні різних прикладних задач, а програмне забезпечення, що надає можливість працювати з такими моделями штучних нейронних мереж має значну практичну цінність.

Мета роботи – розробка програмного забезпечення для розпізнавання об'єктів на зображенні за допомогою згорткових нейронних мереж.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд задачі розпізнавання об'єктів на зображенні

1.1.1 Актуальність та необхідність розпізнавання об'єктів

Розвиток сучасного суспільства тісно пов'язаний з розвитком та модернізацією інформаційних систем та комп'ютерних технологій. Автоматизація та інтелектуалізація різних процесів стає важливим фактором для підвищення ефективності, покращення якості життя людини та забезпечення безпеки суспільства загалом.

Автоматизація та інтелектуалізація у різних галузях діяльності людини базується на обробці та аналізі різних видів інформації, при цьому однією з найважливіших та найскладніших складових є обробка графічної інформації.

Використання графічних даних, таких як зображення, графіки та відео, стало невід'ємною частиною повсякденного досвіду людини, що охоплює різноманітні сфери діяльності, тому класифікація та розпізнавання об'єктів на основі графічних даних стає невід'ємною складовою цього процесу.

Методи класифікації та розпізнавання об'єктів на зображенні застосовуються в різних галузях, починаючи від медичної діагностики, де вони використовуються для обробки медичних зображень, і закінчуючи системами відеоспостереження, де вони застосовуються для аналізу зображень з відеокамер.

Однією з перспективних галузей впровадження технологій обробки й аналізу графічних даних є автомобільна індустрія. Так автовиробники Volkswagen, Toyota, та Mercedes-Benz пропонують своїм клієнтам додаткові опції, що засновані на обробці та аналізі зображень отриманих з розташованих на автомобілі камер: система розпізнавання дорожніх знаків, система розпізнавання та утримання смуги руху, система контролю та моніторингу сліпих зон [1]-[3].

Збільшення попиту на сучасні автомобілі відображає актуальність та затребуваність вбудованих систем, що спрямовані на підвищення рівня безпеки та забезпечення підтримки водія у різноманітних ситуаціях. За статистикою асоціації українських автомобілевиробників «Укравтопром» продажі нових автомобілей у жовтні 2023 року в Україні склали 5 735 штук, що на 82,7% більше ніж за аналогічний період у 2022 році та на 0,7% більше ніж у попередньому місяці (табл. 1.1) [4].

Таблиця 1.1 – Статистика продажів автомобілей в Україні в 2023 році

Бренд	Жовтень 2022, шт.	Вересень 2023, шт.	Жовтень 2023, шт.	Зміни до жовтня 2022, %	Зміни до вересня 2023, %
Toyota	665	960	929	+39,7	-3,2
Volkswagen	188	653	574	+205,3	-12,1
Skoda	326	464	477	+46,3	+2,8
Mercedes-Benz	230	183	224	-2,6%	+22,4
Volvo	59	56	89	+50,8	+58,9
Інші	1671	3377	3442	+105,9	+1,9
Загалом	3 139	5 693	5735	+82,7	+0,7

Системи розпізнавання дорожніх знаків та розпізнавання смуги руху стають стандартом для сучасних автомобілів. У зв'язку з цим важливим завданням стає розробка ефективних методів розпізнавання дорожніх знаків на зображенні отриманому з вбудованої в автомобіль камери.

1.1.2 Види задач розпізнавання об'єктів на зображенні

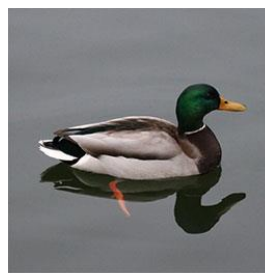
Розпізнавання об'єктів на зображенні є основною складовою задачі комп'ютерного зору. Комп'ютерний зір – це область досліджень у галузі штучного інтелекту, яка направлена на розробку та вдосконалення методів та технологій, що дозволяють комп'ютеру аналізувати та інтерпретувати графічні дані [5].

Основними задачами комп'ютерного зору є класифікація, локалізація, детекція та сегментація об'єктів на зображенні (рис. 1.1).



Рисунок 1.1 – Задачі комп'ютерного зору

Задача класифікації полягає в тому, щоб віднести вхідні дані до однієї з попередньо визначених категорій або класів. Під час класифікації передбачається, що вхідне зображення містить лише один об'єкт, тому зазвичай вихідними даними класифікації є мітка класу чи категорії, до якої було віднесено зображення (рис. 1.2) [6].



селезень

Рисунок 1.2 – Приклад задачі класифікації

Задача локалізації передбачає класифікацію зображення та подальше визначення розташування класифікованого об'єкта на зображенні. Зазвичай розташування об'єкта визначається за допомогою обмежувальної

коробки (bounding box), яка описує координати, розміри та межі об'єкта. В якості стандартної форми обмежувальної коробки використовують прямокутник, задаючи координати центру та розміри сторін або координати верхньої лівої та нижньої правої вершин (рис. 1.3) [7].

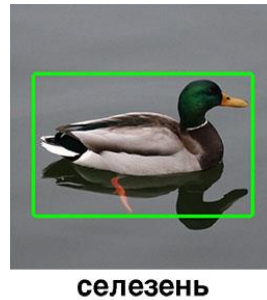


Рисунок 1.3 – Приклад задачі локалізації

Задача детекції полягає в розпізнаванні декількох об'єктів на зображенні з подальшою класифікацією та визначенням розташування кожного з них за допомогою обмежувальної коробки. На відміну від задач класифікації та локалізації, де передбачається, що вхідне зображення містить лише один об'єкт, в задачі детекції передбачається наявність декількох об'єктів, тому визначається наявність та розташування різних об'єктів різних класів чи категорій (рис. 1.4) [8].

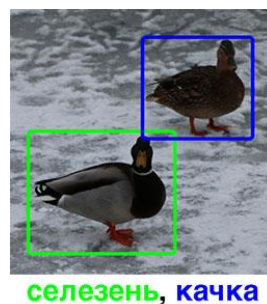


Рисунок 1.4 – Приклад задачі детекції

Задача сегментації полягає у присвоєнні кожному пікселю зображення певного класу або категорії для подальшого його співставлення до відповідного об'єкта або області зображення. Зазвичай вихідними даними

задачі сегментації є деталізована маска, яка описує та виділяє межі декількох знайдених об'єктів на зображенні (рис. 1.5) [9].

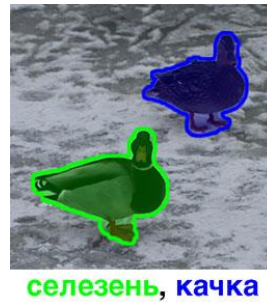


Рисунок 1.5 – Приклад задачі сегментації

Розглянуті задачі комп'ютерного зору характеризуються різною ступеню деталізації результатів, кількістю досліджуваних об'єктів на зображенні та основною метою (табл. 1.2).

Таблиця 1.2 – Порівняння задач комп'ютерного зору

Особливість	Класифікація	Локалізація	Детекція	Сегментація
Мета	Визначити клас об'єкта	Визначити клас та розташування об'єкта	Визначити клас та розташування об'єктів	Визначити клас для кожного пікселя
Кількість об'єктів на зображенні	Один	Один	Декілька	Декілька
Критерії розрахунку точності	Відсоток правильно класифікованих	Відповідність розташувань та розмірів	Відповідність розташувань та розмірів	Подібність маски до еталону
Швидкість розпізнавання	Швидко	Середньо	Середньо	Довго
Складність реалізації	Легко	Середньо	Складно	Більш складно
Обсяг даних для навчання	Мало	Середньо	Середньо	Багато

Так, як для розпізнавання дорожніх знаків важлива швидкість роботи та можливість класифікації з подальшим визначенням розташування декількох об'єктів на зображенні, то необхідно вирішити задачі класифікації та детекції.

1.1.3 Методи та підходи до розпізнавання об'єктів на зображенні

Основні методи та підходи до вирішення задач комп'ютерного зору поділяють на два види: класичні методи комп'ютерного зору та методи на основі глибокого навчання (рис. 1.6).

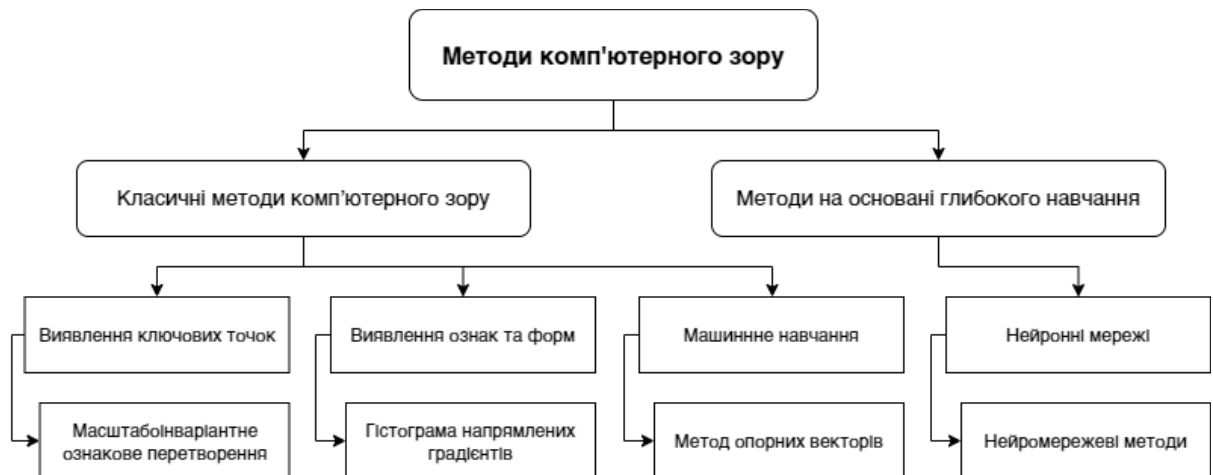


Рисунок 1.6 – Методи комп'ютерного зору

Метод масштабоінваріантного ознакового перетворення (SIFT) – це метод виявлення ознак, який зменшує вміст зображення до кінцевого набору ключових точок, які використовуються для виявлення схожих ознак на інших зображеннях. Спочатку в SIFT знаходять набори ключових ознак для контрольних зображень і записують їх у базі даних. Об'єкти на нових зображеннях розпізнають шляхом порівняння всіх еталонних ознак з бази даних до ознак знайдених на досліджуваному зображенні. В методі масштабоінваріантного ознакового перетворення використовується просторова піраміда зображень різних розмірів, яка дозволяє виявляти ключові ознаки, не прив'язуючись до масштабу об'єктів на зображенні [10].

Метод гістограми напрямлених градієнтів (HOG) – це метод описання структури частин зображення за допомогою аналізу напрямів градієнтів яскравості у різних локальних частинах зображення. На відміну від SIFT,

HOG знаходить не тільки набір ключових точок в якості ключових ознак, а й може виділяти особливості текстури та форми [11].

Метод опорних векторів (SVM) базується на аналізі даних за допомогою опорно-векторних машин, що використовуються для побудови гіперплощин у просторі ознак та подальшого віднесення точок досліджуваних даних до найближчої побудованої гіперплощини. Метод SVM має достатню узагальнюючу здатність та може використовуватися для аналізу як цифрових так і графічних даних [12].

Нейромережеві методи базуються на обробці й інтрепретації вхідних даних на основі використання штучних нейронних мереж. Вони використовуються для вирішення завдань обробки інформації, навчання та виявлення закономірностей у великих наборах різних типів даних [13].

Кожен з методів комп'ютерного зору характеризується різними особливостями реалізація та має свої переваги та недоліки (табл. 1.3).

Таблиця 1.3 – Порівняння методів комп'ютерного зору

Особливість	SIFT	HOG	SVM	Нейромережеві методи
Необхідність навчання	Ні	Ні	Так	Так
Складність навчання	-	-	Середньо	Складно
Складність донавчання	-	-	Середньо	Легко
Час розпізнавання	Довго	Довго	Середньо	Швидко
Складність реалізації	Середньо	Середньо	Складно	Складно
Точність	Середня	Середня	Середня	Висока

Розвиток інформаційних та комп'ютерних технологій, збільшення обчислювальних можливостей сучасних пристроїв та реалізація нових алгоритмів, сприяють тому, що найефективнішими та найперспективнішими методами для розпізнавання об'єктів на зображенні стають методи в основі яких є використання штучних нейронних мереж, адже вони надають найкращу точність та найбільшу швидкість розпізнавання.

1.2 Огляд штучних нейронних мереж та їх видів

1.2.1 Характеристика штучного нейрона

Штучна нейронна мережа – це обчислювальна математична модель, та її апаратна чи програмна реалізація, яка базується на принципах побудови та взаємодії біологічної нейронної мережі живого організму [14]. Основною структурною частиною штучної нейронної мережі є штучний нейрон.

Штучний нейрон є обчислювальною одиницею, яка заснована на принципах побудови моделі біологічного нейрона (рис. 1.7). Біологічний нейрон за допомогою гіллястих структур, дендритів, приймає сигнали з інших нейронів чи сенсорних клітин. Тіло нейрона опрацьовує вхідні сигнали та у випадку перевищення відповідного порогового значення, генерує та передає електричний імпульс на єдиний вихідний відросток, аксон. Коли вихідний електричний імпульс досягає кінця аксона, то у синапсу, просторі між аксоном та дендритами інших нейронів, відбувається вивільнення хімічних речовин, нейромедіаторів, що призводить до передачі сигналу наступному нейрону.

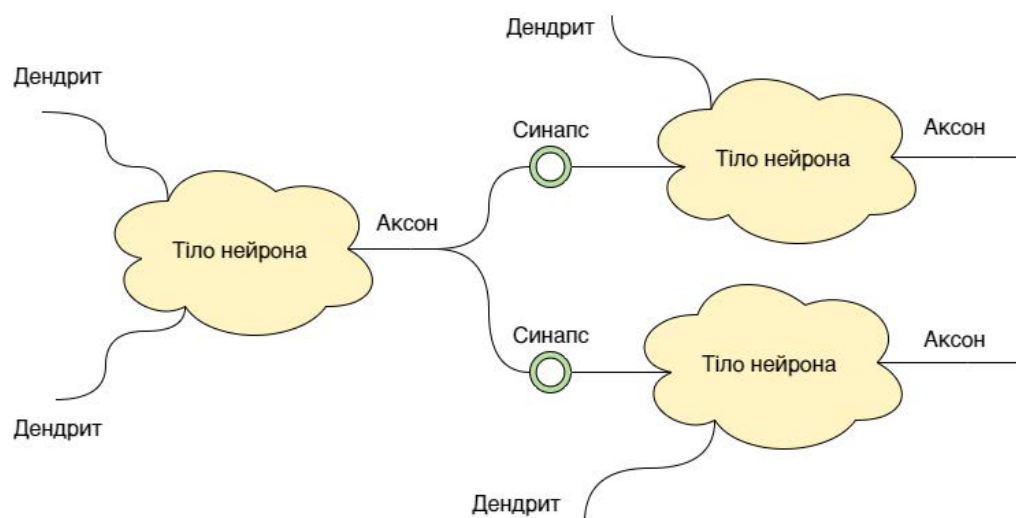


Рисунок 1.7 – Модель біологічного нейрона

Принцип будови та роботи моделі штучного нейрону відповідає принципу будови та роботи біологічного нейрону. Він приймає певний набір

вхідних сигналів з багатьох входів, виконує їх подальший аналіз та обробку й у якості вихідного сигналу повертає отриманий результат (рис. 1.8).

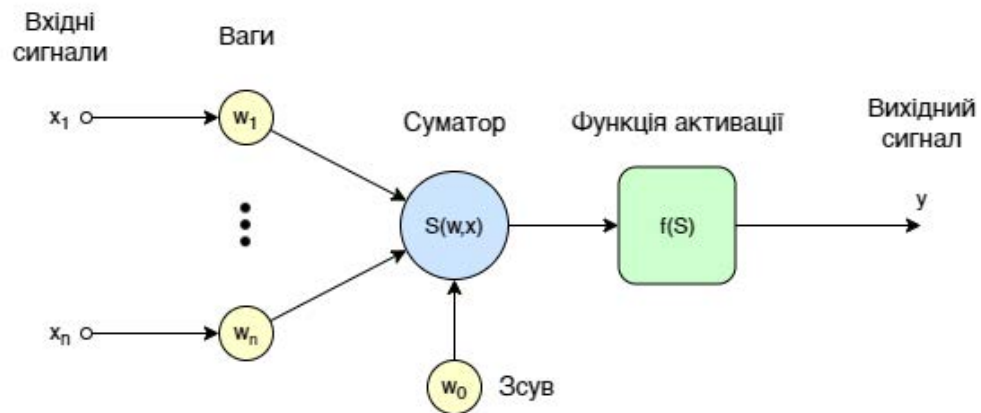


Рисунок 1.8 – Модель штучного нейрона

В якості вхідного набору сигналів, штучний нейрон приймає вектор значень $x = \{x_i\}, i = 1, 2, \dots, n$, де n – розмірність вхідного набору сигналів. Кожен елемент вектора x_i зіставляється з відповідним значенням ваги w_i вектора вагових коефіцієнтів зв'язків нейрону $w = \{w_i\}$. За допомогою функції суматора $S(w, x)$ відбувається поєднання зважених вхідних сигналів та коефіцієнта зсуву w_0 . Після проходження через функцію активації нейрону $f(S)$, результат обробки вхідних сигналів повертається в якості єдиного вихідного сигналу [15].

При вирішенні практичних задач в якості функції суматора прийнято використовувати функцію зваженої суми (1.1):

$$S(w, x) = \sum_{i=1}^n w_i x_i + w_0. \quad (1.1)$$

Штучні нейрони є примітивними але ефективними обчислювальними одиницями, за допомогою поєднання яких створюють різні архітектури складних нейронних мереж, які здатні не тільки аналізувати значення вхідних сигналів, а й навчатись та вирішувати складні інтелектуальні задачі.

1.2.2 Загальна структура нейронної мережі та її навчання

Штучні нейронні мережі складаються з поєднаних в деяку кількість шарів штучних нейронів. У типовій штучній нейронній мережі виділяють три основні типи шарів: вхідний, прихований та вихідний (рис. 1.9).

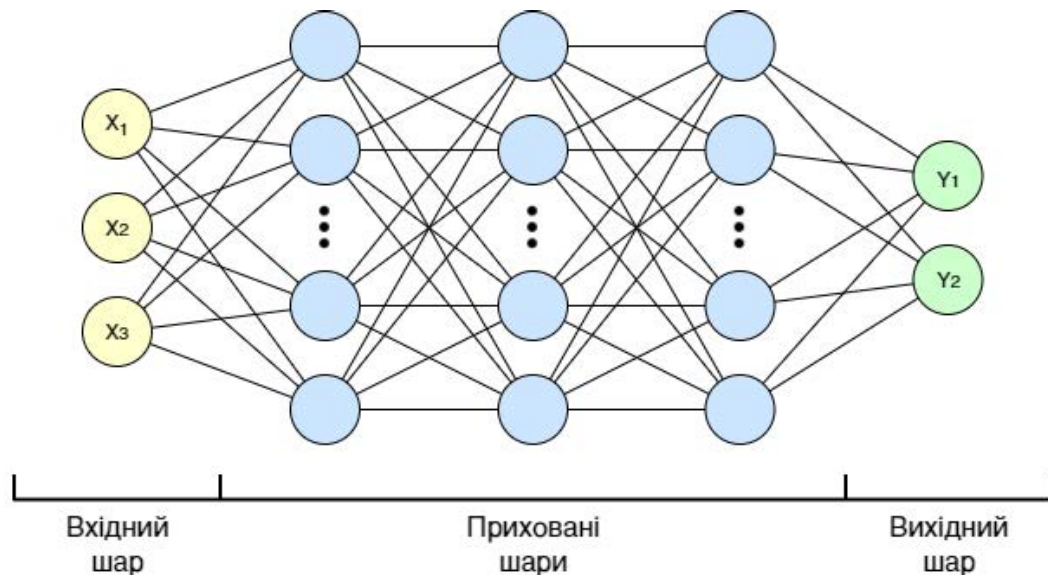


Рисунок 1.9 – Загальна структура нейронної мережі

Вхідний шар відповідає за прийняття та розподілення вхідних сигналів. Цей шар не виконує жодних перетворень та обчислень, а лише отримує вхідні дані та передає їх наступному шару. Він є невід'ємною складовою архітектури будь-якої штучної нейронної мережі і завжди є її першим шаром.

Приховані шари є проміжною ланкою між вхідним та вихідним шарами. Вони виконують всі обчислення і відповідають за виділення закономірностей та особливостей у вхідних даних. Кількість прихованих шарів може варіюватися в залежності від вирішуваної задачі. Таким чином вважається, що перші приховані шари відповідають за аналіз більш загальних особливостей, а наступні – за аналіз більш специфічних складових [16].

Вихідний шар приймає дані від попередніх шарів і генерує остаточне вихідне значення, яке може бути інтерпретовано як відповідь штучної

нейронної мережі на поставлене завдання. Зазвичай кількість нейронів у вихідному шару встановлюється в залежності від задачі навчання.

Якщо архітектура мережі складається тільки з вхідного та вихідного шару, то таку мережу називають одношаровою нейронною мережею. Якщо в архітектурі мережі є приховані шари, то таку мережу називають багатошаровою нейронною мережею. При вирішенні практичних задач завжди використовують багатошарові нейронні мережі, які містять значну кількість прихованих шарів.

Головне завдання мережі – це навчитися правильно визначати зв'язок між вхідними та вихідними даними. Процес навчання нейронної мережі є тривалим та ресурсоємним.

На початку процесу тренування випадковим чином ініціалізуються вагові коефіцієнти для всіх зв'язків. Далі процес навчання поділяється на два етапи: пряме поширення помилки та зворотне поширення помилки [17].

Під час прямого поширення помилки на вхід до мережі подаються дані для тренування. Мережа робить своє передбачення. Отримані вихідні значення порівнюються з очікуваними вихідними значеннями та за допомогою однієї з функцій втрат (перехресна ентропія, середня квадратична похибка, тощо) визначається значення втрат.

Під час зворотного поширення помилки обчислюється, величина зміни вагових коефіцієнтів, яка має сприяти зменшенню значення втрат. Для обчислення використовують градієнтні методи, зокрема алгоритм стохастичного градієнтного спуску. Далі вагові коефіцієнти корегують відповідно до необхідної величини зміни.

Цей алгоритм дій повторюється для всіх даних в навчальному наборі, а один такий повний цикл називається епохою.

В залежності від виду мережі кількість необхідних епох може варіюватись. Оптимальна кількість епох для навчання визначається балансом між задіяними ресурсами, часом а також досягнутим рівнем точності, який потрібний для вирішення конкретного завдання.

1.2.3 Види нейронних мереж

Для вирішення різних нових та типових задач, поєднуючи різні шари, використовуючи різні правила та комбінації взаємозв'язків між нейронами у різних шарах, створюють різні види штучних нейронних мереж (рис. 1.10).



Рисунок 1.10 – Види штучних нейронних мереж

Нейронні мережі прямого поширення є видом штучних нейронних мереж, в яких дані, що обчислюються, лінійно переміщуються тільки в одному напрямку, від вхідного до вихідного шару [18]. Такі нейронні мережі теоретично можуть використовуватися для вирішення задач класифікації та регресії, але на практиці нейронні мережі прямого поширення найчастіше використовуються в якості складових частин інших архітектур.

Нейронні мережі з латеральними зв'язками на відміну від мереж прямого поширення, дозволяють нейронам взаємодіяти не тільки з нейронами попереднього та наступного шарів, але й з іншими нейронами того ж самого шару, поліпшуючи розуміння та сприйняття досліджуваного контексту.

Нейронні мережі зі зворотними зв'язками дозволяють нейронам взаємодіяти з нейронами попередніх шарів, дозволяючи передавати сигнали не тільки від вхідного до вихідного шару, а й у зворотній послідовності, що додає можливість для корекції та навчання мережі на основі оброблених на інших шарах даних.

Нейронні мережі з латеральними та зворотними зв'язками широко використовуються для аналізу та обробки послідовних даних, особливо для розпізнавання тексту та мовлення.

Клас глибоких нейронних мереж складається з великої кількості архітектур, які розроблені для вирішення різноманітних інтелектуальних задач. Одним із підвидів мереж цього класу є згорткові нейронні мережі, які за рахунок використання механізмів схожих на особливості зорової кори людини, можуть аналізувати та розпізнавати образи на зображенні [19].

Кожен з видів моделей нейронних мереж має різну архітектуру, особливості реалізації та налаштування (табл. 1.4).

Таблиця 1.4 – Порівняння видів нейронних мереж

Особливість	Прямого поширення	З латеральними зв'язками	Зі зворотними зв'язками	Згорткові
Програмна документація	Багато	Середньо	Середньо	Багато
Складність реалізації	Легко	Складно	Складно	Середньо
Тривалість навчання	Швидко	Середньо	Середньо	Довго
Швидкість обчислень	Швидко	Середньо	Середньо	Середньо
Обробка послідовних даних	Ні	Так	Так	Так
Розпізнавання зображень	Теоретично	Теоретично	Ні	Так
Класифікація та розпізнавання об'єктів на зображенні	Ні	Ні	Ні	Так

Так як при вирішенні задачі розпізнавання дорожніх знаків за допомогою нейромережових методів спираються на обробку зображень отриманих з вбудованої в автомобіль камери, то доцільно використовувати згорткові нейронні мережі, адже вони найкраще підходять для аналізу зображень та вирішення поставленої задачі класифікації та розпізнавання об'єктів.

1.3 Аналіз архітектури згорткових нейронних мереж

1.3.1 Загальна архітектура згорткових нейронних мереж

Згорткові нейронні мережі (CNN) – це підвид архітектури нейронних мереж глибокого навчання, який широко використовується для аналізу й обробки зображень з метою вирішення основних задач комп'ютерного зору. Невід'ємною складовою архітектури нейронних мереж є згорткові шари які відповідають за виявлення високорівневих та низькорівневих ознак та закономірностей в зображеннях [20].

За архітектурою згорткові нейронні мережі є багат шаровими мережами прямого поширення та зазвичай складаються з вхідного, згорткового, агрегувального, повнозв'язного та вихідного шарів (рис. 1.11).

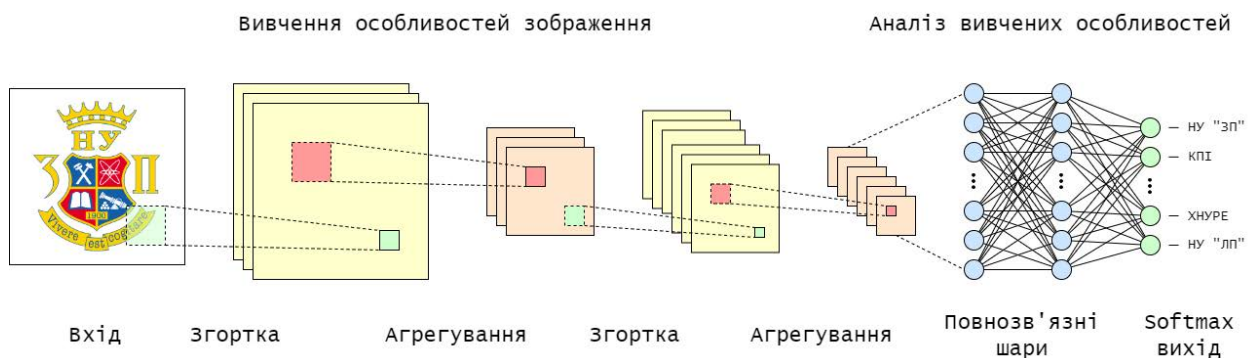


Рисунок 1.11 – Архітектура згорткових нейронних мереж

1.3.2 Згорткові шари

Згорткові шари (Convolutional Layers) є ключовим архітектурним компонентом згорткових нейронних мереж, а основною операцією яка в них обчислюється є операція згортки.

В згортковому шарі використовуються ядра згортки – невеликі квадратні матриці, з набором параметрів що навчаються. Вони ковзають по області зображення застосовуючи операцію згортки. Взаємодіючи з різними регіонами вхідного зображення вони виділяють їх локальні особливості.

Операція згортки полягає в обчисленні сум добутків ваг ядер згортки з відповідною областю значень елементів вхідних даних. В результаті виконання згортки обчислену суму для кожної області вхідних даних записують у вихідну матрицю, яку називають картою активації (рис. 1.12).

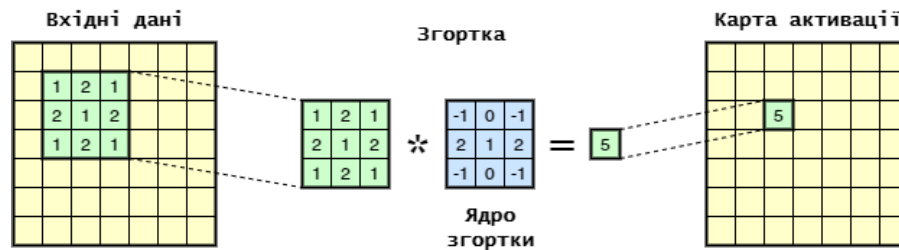


Рисунок 1.12 – Операція згортки

Під час проходження по набору вхідних даних ядро згортки переміщується з заданим значенням кроку. Так як крайні елементи матриці вхідних даних ніколи не потрапляють до центру ядра згортки, то до меж вхідної матриці додають нульові значення. Така операція називається нульовим доповненням.

Якщо $x_{i,j}$ є елементом досліджуваної частини вхідної матриці, а $w_{i,j}$ вагою відповідного елемента ядра згортки розмірності n , то операцію згортки можна представити таким математичним виразом (1.2):

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,n} \end{bmatrix} \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,n} \end{bmatrix} = \sum_{i=1}^n \sum_{j=1}^n x_{i,j} w_{i,j} . \quad (1.2)$$

Значення елементів карти активації показують чи наявні у визначених областях вхідних даних особливості описані використаним ядром згортки. Використання різних ядер згортки в межах згорткових шарів дозволяє виявляти різні особливості вхідних даних (рис. 1.13).

Використання декількох послідовних згорткових шарів дозволяє створювати адаптивні та гнучкі мережі. Завдяки першим згортковим шарам,

штучна нейронна мережа вивчає загальні та типові риси вхідних даних, такі як лінії, кути, а за допомогою наступних шарів вона вивчає більш складні риси, які є комбінацією попередньо знайдених особливостей [21].

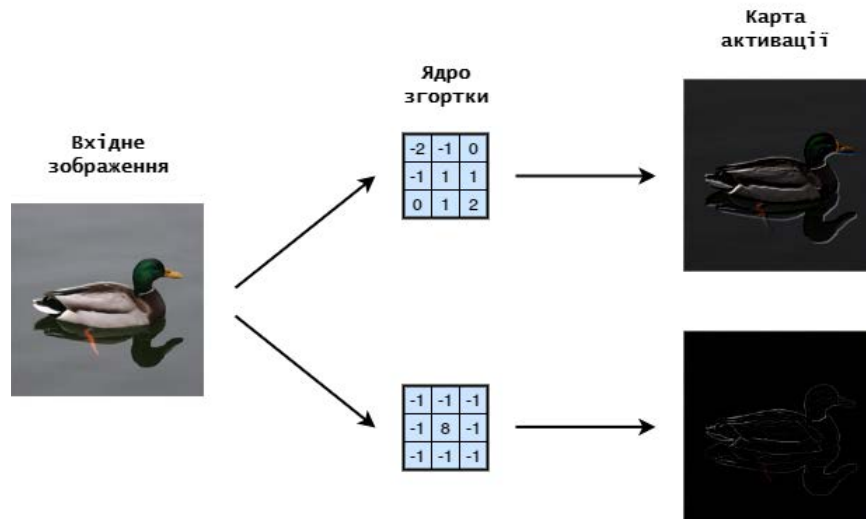


Рисунок 1.13 – Приклади використання різних ядер згортки

1.3.3 Агрегувальні шари

Агрегувальні шари використовуються для зменшення розмірностей карт активації зі збереженням виявлених особливостей зображень. Використання агрегації дозволяє зменшити число параметрів мережі та кількість подальших обчислень.

Під час агрегації агрегувальний фільтр з певним кроком переміщується по отриманій після згорткового шару карті активацій та виконує об'єднання регіону, який підпадає під дію фільтра, використовуючи обраний вид операції об'єднання. В результаті агрегації набір значень вибраного регіону замінюється на значення результату обчислень.

Зазвичай розмір фільтра агрегації значно менше за розмір самої карти активацій. На практиці найчастіше використовують фільтр розміром 2×2 з кроком 2, що дозволяє зберегти всю інформацію про виявлені особливості та зменшити розмірність карти активацій в два рази.

Існує два види операції об'єднання: максимізаційне та усереднювальне агрегування. За допомогою усереднювального агрегування обчислюється середнє значення регіону карти активацій, який підпадає під фільтр, а за допомогою максимізаційного – максимальне значення [22].

В архітектурі згорткових нейронних мереж найчастіше використовується максимізаційне агрегування, тому, що воно допомагає зберегти більше деталей та зафіксувати більш значимі особливості зображення (рис. 1.14).



Рисунок 1.14 – Максимізаційне агрегування

Операцію максимізаційного агрегування можна задати за допомогою математичного виразу (1.3):

$$x_{i,j} = \max_{k=0,l=0}^n \{x_{(i+k)(j+l)}\}, \quad (1.3)$$

де $x_{i,j}$ – значення одного з елементів карти активації, що підпадає під фільтр;

n – розмір агрегаційного фільтра.

1.3.4 Згладжування, повнозв'язні шари та функції активації нейронів

Після проходження вхідних даних через згорткові та агрегаційні шари оброблені дані у вигляді набору карт активацій передаються на вхід повнозв'язних шарів для остаточної класифікації та регресії.

Так як в результаті своєї роботи згорткові та агрегувальні шари повертають набір даних у вигляді набору двовимірних матриць, а повнозв'язні шари приймають на вхід тільки одновимірний вектор, то для перетворення отриманих даних використовують шар згладжування, який перетворює тривимірний об'єм даних в одновимірний вектор (рис. 1.15) [23].

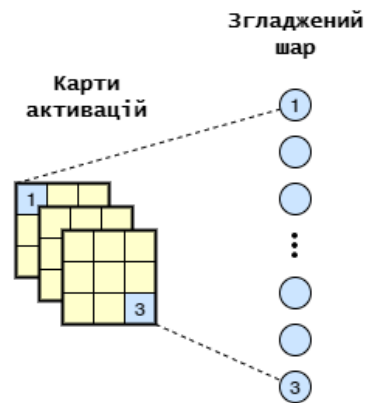


Рисунок 1.15 – Згладжування

Згорткові шари ефективно виявляють локальні особливості вхідних даних, а повнозв'язні шари допомагають об'єднати всі обчислені дані та здійснити остаточний аналіз комбінацій виявлених високорівневих ознак. Повнозв'язні шари допомагають моделі отримувати загальну інформацію, об'єднуючи характеристики різних локальних особливостей, вивчати високорівневі ознаки та складні шаблони в даних, тим самим вирішуючи задачі класифікації та регресії (рис. 1.16).

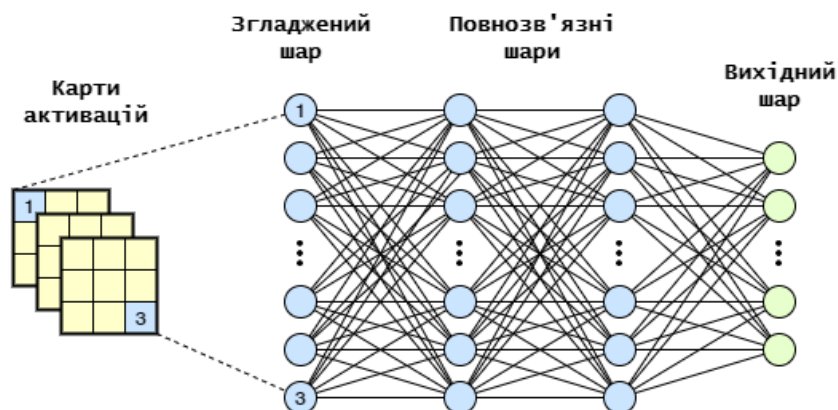


Рисунок 1.16 – Структура згладжувального та повнозв'язних шарів

В якості архітектури повнозв'язних шарів використовується багатошаровий персептрон, на вихідному шарі якого за допомогою використання нормовано експоненційної функції обчислюється розподіл ймовірностей приналежності вхідного зображення до певних класів.

Варто зауважити, що після обчислень значення вхідних сигналів суматором штучного нейрону будь-якого шару, вихідний сигнал може приймати будь-яке значення в діапазоні $(-\infty; +\infty)$. Штучний нейрон не знає порогового значення, при досягненні якого він має бути активований, тому отримане значення нормалізують за допомогою функції активації.

Функція активації визначає вихідне значення нейрона залежно від результату порівняння зваженої суми входів і порогового значення. Серед найчастіше використовуваних функцій активації нейронів виділяють: порогову, лінійну, сигмоїдну, ReLU та Leaky ReLU [24].

Порогова функція (1.4) повертає тільки два значення, через що може використовуватися лише для задач бінарної класифікації. Вона перетворює значення сигналу на 1, якщо сигнал більше чи дорівнює пороговому значенню a , та на 0 для всі інших значень сигналу:

$$f(x) = \begin{cases} 1, & x \geq a \\ 0, & x < a \end{cases} \quad (1.4)$$

Лінійна функція (1.5) виражається простою лінійною залежністю де значення активації нейрона пропорційно дорівнює значенню вхідного сигналу:

$$f(x) = x. \quad (1.5)$$

Сигмоїдна функція (1.6) дозволяє перетворювати вхідний сигнал на будь-яке значення в діапазоні $(0; 1)$. Вона є нелінійною функцією активації та має гладкий вигляд:

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (1.6)$$

Функція ReLU (1.7) є нелінійною функцією та вимагає меншої кількості обчислень ніж сигмоїдна функція. ReLU перетворює від'ємні значення сигналу на 0 та залишає всі інші значення без змін:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}. \quad (1.7)$$

Функція Leaky ReLU (1.8) є підвидом нелінійної функції активації ReLU. Вона передає лише частину від'ємних значень сигналу та залишає всі додатні значення без змін. Такий підхід дозволяє зменшити кількість потенційно неактивних нейронів під час навчання мережі:

$$f(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}, \quad (1.8)$$

де a – деякий невеликий коефіцієнт.

В згорткових нейронних мережах функції активації розміщуються після кожного згорткового або повнозв'язаного шару. Зважаючи на особливості розглянутих функцій активації (табл. 1.5), можна зробити висновок, що найкраще використовувати ReLU та Leaky ReLU.

Таблиця 1.5 – Порівняння основних функцій активації нейронів

Особливість	Порогова	Лінійна	Сигмоїдна	ReLU	Leaky ReLU
Діапазон значень	{0, 1}	$(-\infty; +\infty)$	(0; 1)	$[0; +\infty)$	$(-\infty; +\infty)$
Лінійність	Ні	Так	Ні	Ні	Ні
Ріномірність розподілу	Ні	Так	Ні	Ні	Так
Незадіяні нейрони	Так	Ні	Ні	Так	Ні
Ресурсоємність обчислень	Мало	Мало	Багато	Мало	Мало

1.3.5 Нормована експоненційна функція та функція втрат

Вихідний шар згорткової нейронної мережі складається з необроблених вихідних даних, які називають логітами. В задачах класифікації вважається, що кожен логіт співвідноситься з відповідним класом. Щоб перетворити значення логітів в розподіл ймовірностей приналежності зображення до певного класу використовується нормована експоненційна функція.

Нормована експоненційна функція (Softmax) – це вид логістичної функції активації, що перетворює вектор необроблених числових значень на ймовірнісний розподіл (рис. 1.17).

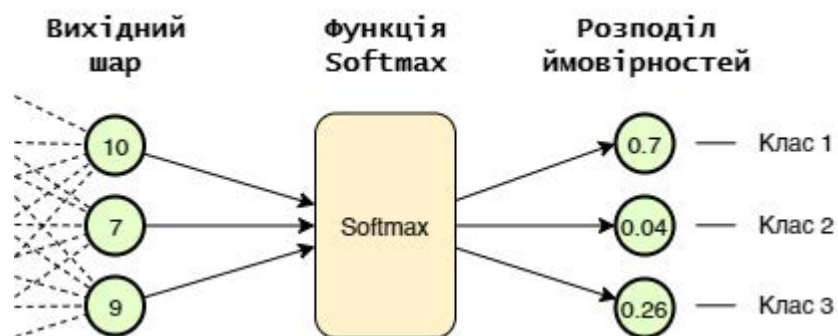


Рисунок 1.17 – Використання функції Softmax

Функція Softmax широко використовується для багатокласової класифікації. Кожне значення елемента вихідного вектора нормованої експоненційної функції (1.9) є ймовірністю приналежності зображення до відповідного класу, а сума всіх значень завжди дорівнює одиниці [25]:

$$f(x_i) = \frac{x_i}{\sum_{k=1}^K x_k}, \quad (1.9)$$

де x_i – значення відповідного нейрона вихідного шару;

K – число класів.

Під час навчання згорткової нейронної мережі для вирішення задачі класифікації після застосування функції Softmax, за допомогою функції перехресної ентропії обчислюються втрати моделі [26].

Функція перехресної ентропії (Cross-Entropy Loss) – це функція втрат, яку використовують для кількісної оцінки різниці між двома розподілами ймовірностей. Функція (1.10) вимірює розбіжність між імовірнісним розподілом, передбаченим моделлю, і фактичним розподілом класів [27]:

$$L(\hat{y}, y) = - \sum_k^K y_k \log \hat{y}_k, \quad (1.10)$$

де y_k – фактичний індикатор наявності класу k у вхідних даних;

$\hat{y}_k$ – передбачена моделлю ймовірність наявності класу k у вхідних даних;

K – число класів.

Обчислення та аналіз втрат є невід’ємною частиною навчання будь-якої нейронної мережі при використанні навчання методом зворотного поширення помилки. В залежності від вирішуваної задачі, для обчислень значення втрат можуть використовуватися й інші функції.

1.4 Аналіз видів згорткових нейронних мереж

1.4.1 Види згорткових нейронних мереж для класифікації зображень

Використовуючи різну розмірність вхідних даних, поєднуючи та налаштовуючи згорткові, агрегувальні, повнозв’язні шари, змінюючи їх типи зв’язків та застосовуючи функцію Softmax до вихідного шару, створюють та вдосконалюють різні види архітектур згорткових нейронних мереж для розпізнавання зображень. Серед основних видів згорткових нейронних мереж для класифікації зображень можна виділити такі: LeNet, AlexNet, ZFNet, та VGGNet.

LeNet – це сімейство згорткових нейронних мереж, яке було запропоновано в 1989 році французьким інженером Яном ЛеКуном [28]. Мережа приймає одноканальне зображення відтінків сірого розміром 32x32 пікселів та за рахунок двох-трьох згорткових шарів може класифікувати його на невелику кількість класів. LeNet розроблювалася для розпізнавання рукописних цифр, та може бути використана для виявлення тривіальних особливостей зображення.

Згорткова нейронна мережа AlexNet була розроблена Алексом Крижевським, Іллею Суцкевером та Джефрі Гінтоном у 2012 році. Мережа приймає трьохканальне кольорове зображення розміром 227x227 пікселів та за рахунок п'яти згорткових шарів може виявляти високорівневі комбіновані особливості зображення [29]. На змаганнях із розпізнавання великої кількості зображень ILSVRC-2012 мережа AlexNet перевершила результати класичних алгоритмів комп'ютерного зору, досягнув точності в 83,7%. Винайдення AlexNet стало ключовою точкою в розвитку та початку використання згорткових нейронних мереж в задачах розпізнавання зображень.

Мережа ZFNet була розроблена Метью Зейлером та Робом Фергусом у 2013 році [30]. Мережа базується на принципах роботи мережі AlexNet. Вона приймає трьохканальне кольорове зображення розміром 224x224 пікселів та за рахунок використання тією ж кількості згорткових шарів але з меншим розміром ядер згортки може розпізнавати більш складні об'єкти. Мережа ZFNet стала переможцем на змаганнях ILSVRC-2013 з результатом 88,3% точності, що на 4,6% перевершило результати мережі AlexNet.

VGGNet – це сімейство згорткових нейронних мереж, яке було представлено у 2014 році вченими Кареном Сімоняном та Ендрю Зіссерманом [31]. Особливістю мереж VGGNet є те, що вони приймають трьохканальне кольорове зображення розміром 224x224 пікселів та використовують глибинну архітектуру з восьми-шістнадцяти згорткових шарів з маленьким ядром згортки. Мережі VGGNet перемогли на змаганнях ILSVRC-2014 у 2014 році досягнувши точності в 92,7%.

Розглянуті моделі нейронних мереж для класифікації зображень мають свої переваги та недоліки (табл. 1.6).

Таблиця 1.6 – Порівняння згорткових нейронних мереж

Особливість	LeNet	AlexNet	ZFNet	VGGNet
Рік винайдення	1989	2012	2013-2014	2014
Кількість вхідних каналів	1	3	3	3
Розмір вхідного зображення	32x32	227x227	224x224	224x224
Кількість шарів з параметрами	5	8	8	11-19
Кількість згорткових шарів	3	5	5	8-16
Ядро згортки	5x5	11x11	7x7	3x3
Функція активації нейронів	Гіперболічний тангенс	ReLU	ReLU	ReLU
Тип агрегування	Усереднювальне	Максимізаційне	Максимізаційне	Максимізаційне
Складність архітектури	Просто	Середньо	Середньо	Середньо
Час розпізнавання	Швидко	Середньо	Середньо	Довше
Час навчання	Швидко	Середньо	Середньо	Довше
Точність на змаганні ILSVRC	-	83,6%	88,3%	92,7%

Моделі сімейства VGGNet можуть оброблювати кольорові зображення, мають високу точність розпізнавання, доволі просту архітектуру та ні в чому не поступається іншим моделям. Тому для вирішення задачі класифікації дорожніх знаків за допомогою згорткових нейронних мереж доцільно використовувати моделі сімейства VGGNet.

1.4.2 Види згорткових нейронних мереж для розпізнавання об'єктів

Загальна архітектура згорткових нейронних мереж розрахована на вирішення задачі класифікації. Таким чином вважається, що кожне вхідне зображення містить лише один об'єкт. В реальному світі найчастіше зображення містять деяку сукупність об'єктів, кожен з яких складається з певного набору особливостей, які його характеризують.

Для того, щоб дати можливість згортковій нейронній мережі знаходити декілька об'єктів, треба трохи модернізувати її архітектуру. Серед архітектур, що базуються на згорткових нейронних мережах, та використовуються для розпізнавання об'єктів на зображенні можна виділити два основні види: двоступеневі детектори об'єктів та одноступеневі детектори об'єктів [32].

Двоступеневі детектори розділяють завдання розпізнавання об'єктів на два етапи. На першому етапі за допомогою деяких алгоритмів або допоміжних нейронних мереж вони виділяють області інтересів (RoI) на зображенні. На другому етапі вони за допомогою згорткової нейронної мережі виявляють особливості для кожного регіону та виконують для нього класифікацію та регресію. Основними прикладами архітектур двоступеневих детекторів є R-CNN, Fast R-CNN та Faster R-CNN.

Регіональні згорткові нейронні мережі (R-CNN) були запропоновані Россом Гіршиком у 2014 році [33]. За допомогою алгоритму вибіркового пошуку (Selective Search Algorithm) з зображення вилучається 2000 RoI [34]. Для кожної із областей інтересів за допомогою попередньо натренованої мережі AlexNet виявляють її особливості. В результаті роботи отримують набір обмежувальних коробок з ймовірностями приналежності певного класу до них. Мережа показує непогану точність, але через те, що кожна з областей окремо проходить через згорткову мережу, то обчислення результатів для одного зображення займає дуже багато часу.

У 2015 році Росс Гіршик запропонував оновлений підхід – швидко регіональну згорткову нейронну мережу Fast R-CNN [35]. Її особливість полягає в тому, що після визначення областей інтересів за допомогою алгоритму вибіркового пошуку, ціле вхідне зображення передається до згорткової нейронної мережі, щоб згенерувати загальну карту ознак зображення. На основі отриманих даних використовується шар об'єднання RoI, на основі якого відбувається прогнозування класу запропонованої області, а також значень зсуву для обмежувальної рамки.

Трохи пізніше у 2015 році Шаоцин Рен запропонував швидшу регіональну згорткову нейронну мережу Faster R-CNN [36]. Замість алгоритму вибіркового пошуку було запропоновано використовувати додаткову нейронну мережу для генерації областей інтересів. Не зважаючи на те, що в Fast R-CNN та Faster R-CNN кожен регіон інтересів окремо не досліджується, ці мережі є ресурсоемними та не можуть використовуватись для швидкої обробки великої кількості вхідних даних.

Одноступеневий детектор не використовує процес вилучення областей інтересів для локалізації об'єктів на зображенні. Він складається з єдиної згорткової мережі, яка прогнозує обмежувальні рамки та імовірності наявності в них об'єкта певного класу за допомогою аналізу частин зображення, які мають високу імовірність наявності об'єктів. Однією з архітектур одноступеневих детекторів об'єктів є мережа YOLOv1.

YOLOv1 (You Only Look Once) – це згорткова мережа для розпізнавання об'єктів на зображенні опублікована Джозефом Редмоном у 2015 році [37]. YOLOv1 розглядає розпізнавання об'єктів як задачу регресії розділених у просторі обмежувальних коробок з пов'язаними імовірностями приналежності до класів. Прогнозування обмежувальних коробок відбувається в один прохід цілого зображення по мережі, за рахунок чого може бути легко оптимізоване. За рахунок невеликої кількості обчислень YOLOv1 працює набагато швидше за інші алгоритми розпізнавання об'єктів.

Розглянуті моделі нейронних мереж для розпізнавання об'єктів на зображенні мають різну архітектуру, складність реалізації та навчання, ресурсоемність обчислень (табл. 1.7) [38].

Таблиця 1.7 – Порівняння згорткових нейронних мереж

Особливість	R-CNN	Fast R-CNN	Faster R-CNN	YOLOv1
1	2	3	4	5
Рік винайдення	2014	2015	2015	2015
Максимальна кількість обмежувальних коробок	2000	2000	-	98

Продовження таблиці 1.7

1	2	3	4	5
Одноступеневий детектор	Ні	Ні	Ні	Так
Обробка зображень в режимі реального часу	Ні	Ні	Теоретично	Так
Обробка відео в режимі реального часу	Ні	Ні	Ні	Так
Час навчання	Довго	Середньо	Швидше	Швидше
Складність реалізації	Середньо	Складно	Складно	Середньо
Ресурсоемність обчислень	Багато	Багато	Середньо	Мало
Кількість зображень в секунду	6	0,5	18	45
Усереднена точність на PASCAL VOC	53,5%	70%	62,1%	63,4%

При розпізнаванні дорожніх знаків на зображенні важливим критерієм є точність та швидкість розпізнавання. Зважаючи на те, що YOLOv1 може швидко оброблювати зображення в режимі реального часу, має просту архітектуру та показує відносно велику точність, то для вирішення задачі розпізнавання дорожніх знаків на зображенні за допомогою згорткових нейронних мереж доцільно використовувати саме її.

1.5 Аналіз існуючих програмних реалізацій

Універсального програмного забезпечення, яке б допомагало вирішувати сукупність різних видів задач не існує. Існуюче програмне забезпечення, яке могло б вирішувати поставлене завдання є пропрієтарним програмним забезпеченням з закритим вихідним кодом. Воно розроблюється виключно під замовлення автомобілебудівних компаній та використовується як невід’ємна частина їх програм. Програмного забезпечення для розпізнавання дорожніх знаків на зображенні отриманому з вбудованої в автомобіль камери не знайдено у відкритому доступі. Це пов’язано з тим, що впровадження та розробка таких систем є ресурсоемним процесом, що вимагає значних матеріальних витрат, а саме програмне забезпечення є інтелектуальною цінністю компанії та надає особливі переваги перед іншими брендами, привертаючи нових клієнтів.

В мережі інтернет можна знайти різні бібліотеки та фреймворки машинного навчання на основі яких можна розроблювати складні системи та реалізовувати будь-які архітектури нейронних мереж. Використання таких бібліотек може спростити процес розробки необхідного програмного забезпечення, адже він допомагає реалізувати необхідний функціонал не втрачаючи час на створення примітивних функцій та методів.

Але для використання таких ресурсів необхідно володіти базовими навичками роботи з вибраним фреймворком та вміти розроблювати програмне забезпечення. Такий варіант не підходить для звичайного користувача, адже вимагає апаратних ресурсів та певних знань для реалізації.

Зважаючи на отримані результати аналізу існуючого програмного забезпечення, можна зробити висновок, що розробка вільного програмного забезпечення для розпізнавання об'єктів, а саме дорожніх знаків, на зображенні є перспективною та затребуваною задачею.

1.6 Висновки за розділом 1

Збільшення попиту на сучасні автомобілі відображає актуальність та затребуваність вбудованих систем, що спрямовані на підвищення рівня безпеки та забезпечення підтримки водія у різноманітних ситуаціях. Системи розпізнавання дорожніх знаків та розпізнавання смуги руху стають стандартом для сучасних автомобілів. У зв'язку з цим важливим завданням стає розробка ефективних методів розпізнавання дорожніх знаків на зображенні отриманому з вбудованої в автомобіль камери.

Найефективнішими та найперспективнішими методами для розпізнавання об'єктів на зображенні є методи в основі яких є використання штучних нейронних мереж. Так як при вирішенні задачі розпізнавання дорожніх знаків за допомогою нейромережевих методів спираються на обробку зображень отриманих з вбудованої в автомобіль камери, то доцільно використовувати згорткові нейронні мережі, адже вони найкраще підходять

для обробки зображень, вирішення задач класифікації та розпізнавання об'єктів.

Проаналізовано загальну архітектуру та види згорткових нейронних мереж, в результаті чого зроблено висновок, що для вирішення задач класифікації зображень доцільно використовувати моделі сімейства VGGNet, а для вирішення задач розпізнавання об'єктів на зображенні доцільно використовувати модель YOLOv1.

Підсумовуючи аналіз існуючого програмного забезпечення, можна зробити висновок, що розробка вільного програмного забезпечення для розпізнавання об'єктів, а саме дорожніх знаків, на зображенні є перспективною та затребуваною задачею.

Для досягнення поставленої мети роботи необхідно вирішити такі завдання:

- виконати аналіз вимог до програмного;
- вдосконалити архітектуру обраних мереж;
- обрати програмні засоби;
- виконати реалізацію програмного забезпечення;
- провести тестування та експериментальне дослідження програми;
- розробити програмну документацію.

2 АНАЛІЗ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ НЕЙРОМЕРЕЖЕВИХ МОДЕЛЕЙ

2.1 Дослідження та проєктування згорткових нейронних мереж для класифікації зображень

2.1.1 Дослідження архітектури нейромережевих моделей VGG

Нейромережіві моделі сімейства VGGNet є класичним представником згорткових нейронних мереж. VGGNet складається з шести архітектур, основна відмінність між якими – кількість використовуваних згорткових шарів. Серед архітектур сімейства VGGNet найчастіше використовують VGG11, VGG13, VGG16 та VGG19 (рис. 2.1).

VGG11	VGG13	VGG16	VGG19
11 вагових шарів	13 вагових шарів	16 вагових шарів	19 вагових шарів
Вхідний шар 224×224×3 (RGB зображення)			
Згортка 3×3×64, ReLU	Згортка 3×3×64, ReLU Згортка 3×3×64, ReLU	Згортка 3×3×64, ReLU Згортка 3×3×64, ReLU	Згортка 3×3×64, ReLU Згортка 3×3×64, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×128, ReLU	Згортка 3×3×128, ReLU Згортка 3×3×128, ReLU	Згортка 3×3×128, ReLU Згортка 3×3×128, ReLU	Згортка 3×3×128, ReLU Згортка 3×3×128, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU	Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU	Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU	Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU Згортка 3×3×256, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU	Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU Згортка 3×3×512, ReLU
Максимізаційне агрегування 2×2 s=2			
Повноз'єднаний шар ×4096, ReLU			
Повноз'єднаний шар ×4096, ReLU			
Повноз'єднаний шар ×1000			
Нормована експоненціальна функція (Softmax)			

Рисунок 2.1 – Архітектура VGG мереж

На вхід мережі VGG подається вхідне трьохканальне кольорове RGB зображення фіксованого розміру 224x224 пікселів. Зображення проходить через набір згорткових шарів, які використовують ядро згортки розміром 3x3, зі значенням кроку та вирівнювання 1. Після декількох послідовних

шарів згортки виконується максимізаційне агрегування, яке використовує агрегувальний фільтр розміром 2×2 з кроком 2. Після набору згорткових та агрегувальних шарів вивлений набір ознак передається на три повнозв'язні шари, перші два з яких повертають вектор розмірністю 4096, а останній вектор розмірністю 1000. В якості функції активації після кожного згорткового та перших двох повноз'єднаних шарів використовують функцію ReLU. На останньому кроці застосовують нормовано експоненційну функцію.

Так як архітектура мережі VGG була створена для вирішення задачі класифікації на основі набору даних ImageNet, то останній повноз'єднаний шар повертає вектор розмірністю 1000. При вирішенні задачі класифікації для інших наборів даних, розмірність вектора, що повертає останній повноз'єднаний шар має бути змінена відповідно до кількості класів у наборі.

Під час навчання після перших двох повнозв'язних шарів використовують функцію регуляризації dropout зі значенням коефіцієнта відсіву 0,5, що дозволяє уникнути перенавчення мережі за допомогою відсіювання певного відсотку випадково вибраних нейронів. Мережу навчають за допомогою метода зворотного поширення помилки, а в якості функції втрат використовується функція перехресної ентропії.

2.1.2 Вдосконалення та оптимізація архітектури нейромережевих моделей VGG

Архітектура мереж VGG проста й ефективна. Вона дозволяє моделям досягати високих показників точності на великих наборах даних. Проте за рахунок глибини мережі та великої кількості послідовних згорткових шарів, моделі необхідно виконувати значну кількість обчислень для отримання вихідних даних. Кількість параметрів та обчислень впливають на час навчання та розпізнавання, через що навчання мережі VGG займає дуже багато часу.

Вдосконалити архітектуру мереж VGG та зменшити час їх навчання можна за допомогою техніки батч-нормалізації. Батч-нормалізація – це метод підвищення ефективності та стабілізації роботи нейронної мережі, який полягає в опрацюванні та нормалізації даних які передаються між прихованими шарами [39].

Шар батч-нормалізації виконує нормалізацію значень активації центруючи і масштабуючи їх за допомогою середнього та стандартного відхилення набору опрацьовуваних даних. Батч-нормалізація допомагає уникнути занулення чи великих значень градієнтів під час навчання, дозволяє зменшити залежність мережі від значень ініціалізованих ваг [40].

В результаті використання батч-нормалізації можна нормалізувати дані, які передаються між прихованими шарами мережі тим самим вдосконалив узагальнюючу здатність моделі та прискорив час її навчання.

Для вдосконалення архітектури мереж VGG прийнято рішення розмістити шари батч-нормалізації перед функціями активації нейронів всіх згорткових шарів мереж VGG (рис. 2.2).

VGG11	VGG13	VGG16	VGG19
Вхідний шар 224×224×3 (RGB зображення)			
Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU
Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU
Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU
Максимізаційне агрегування 2×2 s=2			
Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU
Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU
Максимізаційне агрегування 2×2 s=2			
Повноз'єднаний шар ×4096, ReLU			
Повноз'єднаний шар ×4096, ReLU			
Повноз'єднаний шар ×1000			
Нормована експоненціальна функція (Softmax)			

Рисунок 2.2 – Вдосконалена архітектура VGG мереж

2.2 Дослідження та проєктування згорткових нейронних мереж для розпізнавання об'єктів

2.2.1 Дослідження архітектури нейромережевої моделі YOLOv1

Нейромережева модель YOLOv1 є згортковою нейронною мережею, розробленою для вирішення задач розпізнавання об'єктів. Її архітектура схожа на архітектуру класичних згорткових нейронних мереж, за винятком представлення даних і їх інтерпретації в останніх шарах.

YOLOv1 обчислює особливості всього зображення для передбачення кожної обмежувальної коробки. YOLOv1 розділяє вхідне зображення на сітку розміром $S \times S$ і якщо центр обмежувальної рамки об'єкта лежить в комірці сітки, то ця комірка відповідає за виявлення цього об'єкта. Кожна комірка містить B обмежувальних коробок та значення впевненості для кожної з них. Значення впевненості відображає впевненість моделі в тому, що обмежувальна коробка містить об'єкт.

Кожна обмежувальна коробка складається з п'яти параметрів x, y, w, h, p . Координати (x, y) є значенням координат центру обмежувальної рамки відносно комірки сітки, значення (w, h) є значеннями ширини та висоти обмежувальної рамки відносно всього вхідного зображення, а значення p є значенням впевненості.

Вважається, що значення впевненості в ідеалі має дорівнювати значенню IoU між передбаченою та правдивою обмежувальною коробками. Кожна комірка сітки також містить розподіл імовірностей приналежності об'єкта комірці до одного з K класів. Варто зауважити, що не залежно від кількості B обмежувальних коробок одна комірка сітки містить тільки один розподіл імовірностей приналежності об'єкта до класу, тобто в рамках однієї комірки може бути виявлено об'єкти тільки одно з класів.

Мережа YOLOv1 моделює задачу виявлення об'єктів, як задачу регресії. Вона ділить вхідне зображення на сітку $S \times S$ і для кожної комірки сітки прогнозує B обмежувальних коробок, значення впевненості та розподіл

імовірностей C класів для цих коробок. Передбачення моделі формуються у вигляді $S \times S \times (B * 5 + C)$ тензора (рис. 2.3).

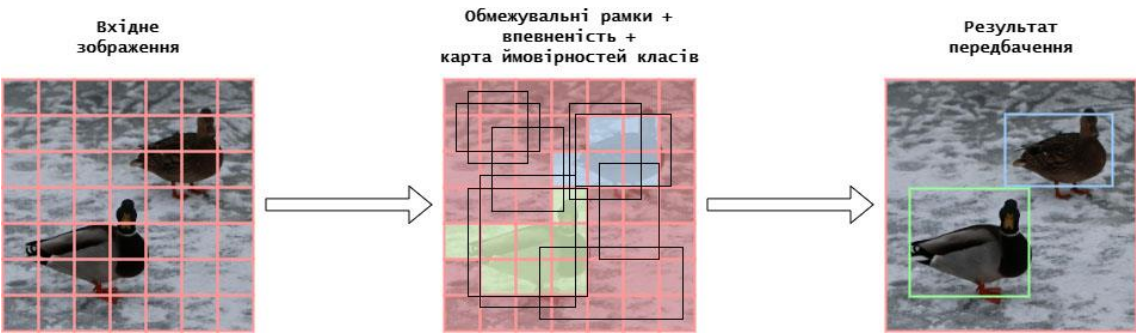


Рисунок 2.3 – Обчислення даних в мережі YOLOv1

Архітектура мережі YOLOv1 представлена на рис.2.4.

YOLOv1
27 вагових шарів
Вхідний шар 224×224×3 (RGB зображення)
Згортка 7×7×64 s=2 p=3, LReLU
Максимізаційне агрегування 2×2 s=2
Згортка 3×3×192, LReLU
Максимізаційне агрегування 2×2 s=2
Згортка 1×1×128 p=0, LReLU
Згортка 3×3×256, LReLU
Згортка 1×1×256 p=0, LReLU
Згортка 3×3×512, LReLU
Максимізаційне агрегування 2×2 s=2
Згортка 1×1×256 p=0, LReLU
Згортка 3×3×512, LReLU
Згортка 1×1×256 p=0, LReLU
Згортка 3×3×512, LReLU
Згортка 1×1×256 p=0, LReLU
Згортка 3×3×512, LReLU
Згортка 1×1×512 p=0, LReLU
Згортка 3×3×1024, LReLU
Максимізаційне агрегування 2×2 s=2
Згортка 1×1×512 p=0, LReLU
Згортка 3×3×1024, LReLU
Згортка 1×1×512 p=0, LReLU
Згортка 3×3×1024, LReLU
Згортка 3×3×1024, LReLU
Згортка 3×3×1024 s=2, LReLU
Згортка 3×3×1024, LReLU
Згортка 3×3×1024, LReLU
Повноз'єднаний шар ×4096, Linear
Повноз'єднаний шар ×S*S*(B*5+C)

Рисунок 2.4 – Архітектура мережі YOLOv1

На вхід мережі YOLOv1 подається вхідне трьохканальне кольорове RGB зображення фіксованого розміру 448x448 пікселів. Зображення проходить через набір згорткових шарів, які використовують ядра згортки різного розміру та значень кроку і вирівнювання. Після деяких шарів згортки виконується максимізаційне агрегування, яке використовує агрегувальний фільтр розміром 2x2 з кроком 2.

Після двадцяти згорткових шарів які використовуються для виявлення особливостей зображення, розміщено ще чотири додаткові згорткові шари які комбінують знайдені особливості та передають їх для аналізу на два повнозв'язні шари, перший з яких повертає вектор розмірністю 4096, а останній вектор розмірністю $S \times S \times (B * 5 + C)$.

В якості функції активації після кожного згорткового шару використовують функцію Leaky ReLU, а для останнього повнозв'язного шару – лінійну.

В базовій архітектурі мережі YOLOv1 використовуються значення параметрів $S = 7$, $B = 2$. Так як мережа розроблювалась для розпізнавання об'єктів на наборі даних PASCAL VOC, то кількість класів $C = 20$. При вирішенні задач розпізнавання об'єктів для інших наборів даних, розмірність вектора, що повертає останній повноз'єднаний шар має бути змінена на значення, що дорівнює $S \times S \times (B * 5 + C)$ [41].

Під час навчання спочатку тренують двадцять перших шарів виявляти особливості у вхідних даних на задачі класифікації для зображень розміром 224x224 пікселів. Потім збільшують розмір вхідних зображень до 448x448 та донавчають мережу вирішувати задачу розпізнавання об'єктів. Після першого повнозв'язного шару використовують функцію регуляризації dropout зі значенням коефіцієнта відсіву 0,5.

Мережу навчають за допомогою метода зворотного поширення помилки. В якості функції втрат використовується особлива функція, заснована на принципі підрахунку суми середньоквадратичних похибок між складовими прогнозованих та істинних даних [42].

Функцію втрат яка використовується в мережі YOLOv1 можна описати за допомогою формули (2.1):

$$\begin{aligned}
 Loss = & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
 & + \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned} \tag{2.1}$$

де 1_i^{obj} – значення чи дійсно є об'єкт в комірці сітки;

1_{ij}^{obj} – значення чи відповідає певна обмежувальна коробка за виявлення;

λ_{coord} , λ_{noobj} – коефіцієнти штрафів.

Базові значення коефіцієнтів штрафів, які запропоновані автором мережі – $\lambda_{coord} = 5$ та $\lambda_{noobj} = 0,5$.

Під час розпізнавання об'єктів на зображенні мережа може згенерувати надлишкові обмежувальні коробки. Для фільтрації згенерованих коробок використовують алгоритм Non-maximum Suppression.

Non-maximum Suppression – це алгоритм який фільтрує вхідний набір обмежувальних коробок відкидаючи коробки з малим значенням впевненості та залишаючи найкращі з набору коробок, що перекриваються. Спочатку відкидають обмежувальні коробки зі значенням впевненості менше за threshold. Далі для обмежувальних коробок однакових класів розраховують значення IoU. Отримані значення сортують за спаданням. Якщо значення IoU для однієї пари отриманих обмежувальних рамок більше за IoU threshold, то з цієї пари залишають обмежувальну коробку з більшим значенням впевненості [43].

2.2.2 Вдосконалення та оптимізація архітектури нейромережевої моделі YOLOv1

Архітектура мережі YOLOv1 дозволяє швидко розпізнавати об'єкти на зображеннях, в деяких випадках нехтуючи точністю розпізнавання. Точність розпізнавання мережі можна підвищити оновивши архітектуру виявляча особливостей а зменшити час навчання та нормалізувати дані між прихованими шарами за допомогою шарів батч-нормалізації.

Для вдосконалення архітектури мережі YOLOv1 прийнято рішення розмістити шари батч-нормалізації перед функціями активації нейронів всіх згорткових шарів. Також вирішено в якості виявляча особливостей використати попередньо навчену частину мереж VGG, інтегрувавши архітектуру виявляча особливостей мережі VGG в YOLOv1 (рис. 2.5).

YOLOv1	YOLOv1_VGG11	YOLOv1_VGG13	YOLOv1_VGG16	YOLOv1_VGG19
Вхідний шар 224×224×3 (RGB зображення)				
Згортка 7×7×64 s=2 p=3, BN, LReLU	Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU Згортка 3×3×64, BN, ReLU	Згортка 3×3×64, BN, ReLU Згортка 3×3×64, BN, ReLU
Максимізаційне агрегування 2×2 s=2				
Згортка 3×3×192, BN, LReLU	Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU Згортка 3×3×128, BN, ReLU	Згортка 3×3×128, BN, ReLU Згортка 3×3×128, BN, ReLU
Максимізаційне агрегування 2×2 s=2				
Згортка 1×1×128 p=0, BN, LReLU Згортка 3×3×256, BN, LReLU Згортка 1×1×256 p=0, BN, LReLU Згортка 3×3×512, BN, LReLU	Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU	Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU Згортка 3×3×256, BN, ReLU
Максимізаційне агрегування 2×2 s=2				
Згортка 1×1×256 p=0, BN, LReLU Згортка 3×3×512, BN, LReLU Згортка 1×1×256 p=0, BN, LReLU Згортка 3×3×512, BN, LReLU Згортка 1×1×256 p=0, BN, LReLU Згортка 3×3×512, BN, LReLU Згортка 1×1×256 p=0, BN, LReLU Згортка 3×3×512, BN, LReLU Згортка 1×1×512 p=0, BN, LReLU Згортка 3×3×1024, BN, LReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU
Максимізаційне агрегування 2×2 s=2				
Згортка 1×1×512 p=0, BN, LReLU Згортка 3×3×1024, BN, LReLU Згортка 1×1×512 p=0, BN, LReLU Згортка 3×3×1024, BN, LReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU	Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU Згортка 3×3×512, BN, ReLU
Максимізаційне агрегування 2×2 s=2				
Згортка 3×3×1024, BN, LReLU Згортка 3×3×1024 s=2, BN, LReLU Згортка 3×3×1024, BN, LReLU Згортка 3×3×1024, BN, LReLU				
Повноз'єднаний шар ×4096, Linear				
Повноз'єднаний шар ×S*S*(B*5+C)				

Рисунок 2.5 – Вдосконалені архітектури мережі YOLOv1

2.3 Висновки за розділом 2

Проведено докладний аналіз архітектури згорткових нейронних мереж сімейства VGGNet та YOLOv1. Визначено основні принципи побудови, навчання, та використання цих мереж для вирішення прикладних задач. Проаналізовано потік даних по мережі та розглянуто використовувані функції втрат.

В результаті дослідження знайдено та проаналізовано недоліки архітектур мереж VGG та YOLOv1.

Для вдосконалення архітектури мереж сімейства VGGNet прийнято рішення розмістити шари батч-нормалізації перед функціями активації нейронів всіх згорткових шарів мереж VGG.

Для вдосконалення архітектури мережі YOLOv1 прийнято рішення розмістити шари батч-нормалізації перед функціями активації нейронів всіх згорткових шарів. Також вирішено в якості виявляча особливостей використати згорткову частину попередньо навчених мереж VGG, тим самим інтегрувавши архітектуру вдосконаленого виявляча особливостей мережі VGG в архітектуру мережі YOLOv1.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз вимог до програмного забезпечення

Під час проєктування програмного забезпечення виконують докладний аналіз вимог, та висувують основні функціональні та нефункціональні вимоги до нього. Такий процес дозволяє виявити основні задачі та особливості які мають бути вирішені в процесі розробки. В результаті аналізу предметної області та вимог до сучасних програмних систем, з метою розробки програмного забезпечення для виявлення дорожніх знаків на зображенні з вбудованою в автомобіль камери, було висунуто основні функціональні та нефункціональні вимоги (рис. 3.1).



Рисунок 3.1 – Висунуті вимоги до розроблюваного ПЗ

Для розпізнавання дорожніх знаків програмне забезпечення має надавати можливість навчання нейромережевої моделі, завантаження зображення до системи з подальшою класифікацією чи розпізнаванням об'єктів на ньому за допомогою вибраної зі списку натренованої нейромережевої моделі з подальшим відображенням результатів.

Оскільки дизайн дорожніх знаків або їх форма можуть бути змінені чи доповнені відповідно до змін в законодавчих актах чи правилах, то програмне забезпечення повинно надавати можливість перенавчати або донавчати моделі для розпізнавання нових чи оновлених еталонних даних.

Під час навчання нейромережових моделей програмне забезпечення має відображати детальну статистичну інформацію про поточний стан та характеристики моделі.

Графічний інтерфейс користувача має бути інтуїтивно зрозумілим, бути стійким до помилок та неочікуваних ситуацій, мати швидкий відклик на будь-які дії користувача в системі.

Програмна документація повинна складатися з керівництва програміста та користувача, містити опис структури, особливостей реалізації ПЗ та список мінімальних вимог до програмного чи апаратного забезпечення, на якому може використовуватися розроблена система.

На рисунку 3.2 представлено діаграму прецедентів (Use Case diagram) розроблюваного програмного забезпечення.

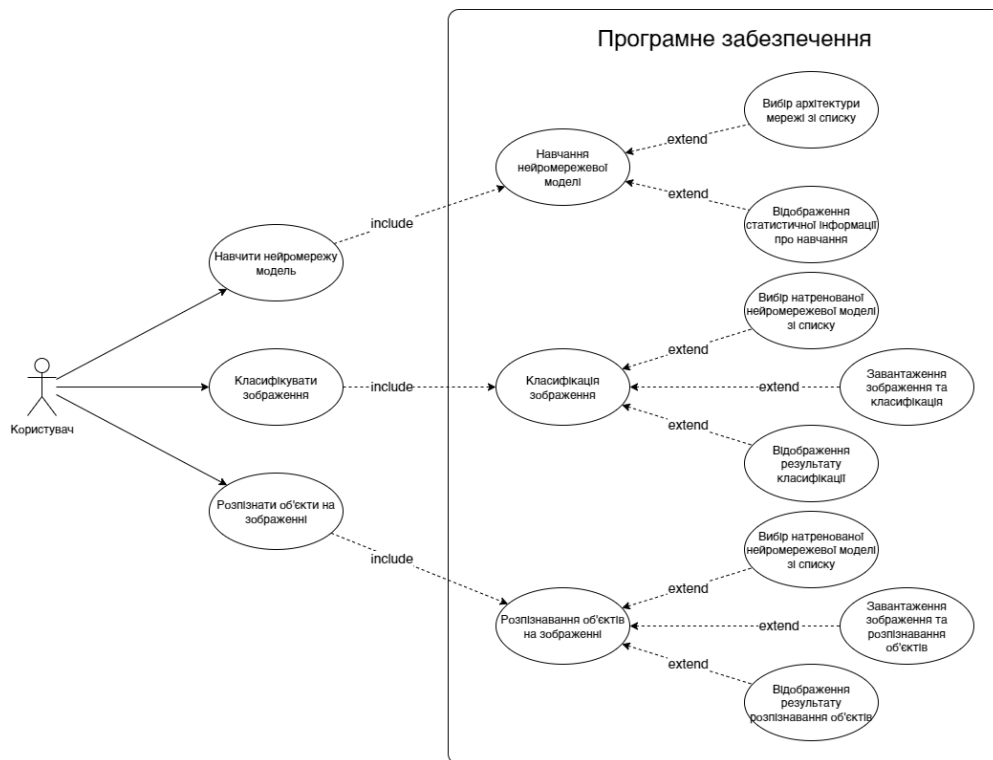


Рисунок 3.2 – Діаграма прецедентів розроблюваного ПЗ

Діаграма прецедентів дозволяє концептуально відобразити програмну систему, демонструючи її основні складові процеси, потоки даних та варіанти її використання [44]. Вона відображає залежність між акторами та прецедентами. В рамках цієї діаграми актором виступає користувач який щось виконує, а прецедентом – функціональна частина самого ПЗ.

Працюючи з програмою користувач виконує такі основні функції: навчання нейромережевої моделі, перегляд детальної статистичної інформації про модель, класифікація зображення та розпізнавання об'єктів на зображенні за допомогою попередньо натренованої нейромережевої моделі. Користувач, натискаючи на кнопки та виконуючи будь-які дії за допомогою графічного інтерфейсу, надсилає сигнали ПЗ про необхідність виконання певного набору дій.

Програмне забезпечення отримує сигнал від користувача та запускає відповідний набір дій: виконує програмний код, оновлює компоненти графічного інтерфейсу, тренує нейромережеві моделі, завантажує зображення в пам'ять, класифікує його чи розпізнає об'єкти на ньому.

3.2 Вибір засобів розробки

Розробка будь-якого ПЗ розпочинається з вибору мови програмування, якою його буде написано. Від мови програмування залежать особливості програмної реалізації та можливості застосування розроблюваного ПЗ на різних пристроях з різним апаратним та програмним забезпеченням.

Мова програмування зумовлює набір певних вимог до структури програмного коду, використовуваних фреймворків та бібліотек. Різні фреймворки та бібліотеки допомагають вирішити певний набір задач та можуть бути використані з метою спрощення процесу розробки, адже в них може бути реалізована частина необхідного функціоналу, наприклад логіка малювання та відображення графічного інтерфейсу чи графічних даних.

3.2.1 Вибір мови програмування

Мова програмування – це формальна мова за допомогою якої описується комп’ютерна програма. Мова програмування складається з певного набору лексичних та синтаксичних правил, які визначають структуру та послідовність виконання команд. Під час виконання коду спеціальна програма перетворює послідовність написаних команд в машинний код, після чого надсилає їх на виконання процесору комп’ютера.

Існує безліч мов програмування (рис. 3.3), серед яких для роботи з нейромережевими моделями найчастіше використовуються C++, C#, JavaScript та Python.

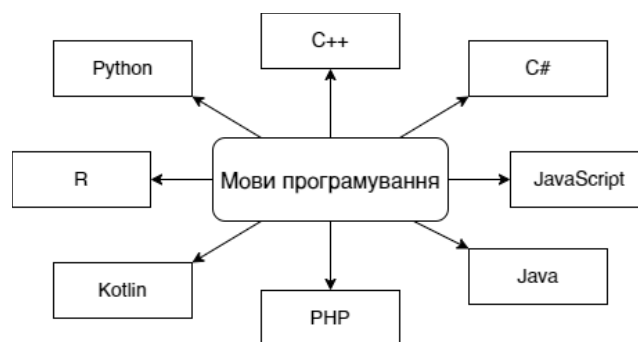


Рисунок 3.3 – Мови програмування

Мова програмування C++ є однією з найпопулярніших та найвідоміших мов програмування у світі. C++ є строго типізованою компільованою високорівневою мовою програмування [45]. Вона підтримує такі парадигми програмування: об’єктно-орієнтовану, узагальнену та процедурну. Вона найчастіше використовується для розробки десктопних застосунків. Останнім часом розробники популярних фреймворків машинного навчання додають нові API для роботи з бібліотеками машинного навчання за допомогою мови програмування C++.

Мова програмування C#, є представників мов програмування сімейства C. Як і C++, C# є строго типізованою компільованою високорівневою мовою програмування з підтримкою багатьох парадигм

програмування [46]. Мова програмування розроблюється і підтримується компанією Microsoft. Вона використовується для створення десктопних та web-застосунків. В 2018 році компанією Microsoft для C# було презентовано додаткову відкриту бібліотеку машинного навчання ML.NET, яка використовується для роботи з нейромережевими моделями.

JavaScript є динамічно типізованою інтерпретованою високорівневою мовою програмування [47]. Вона підтримує об'єктно-орієнтовану, імперативну та функціональну парадигми програмування та використовується для створення клієнт-серверних застосунків. На JavaScript для роботи з машинним навчанням існує декілька бібліотек, однією з яких є TensorFlow.js.

Мова програмування Python є динамічно типізованою інтерпретованою мультипарадигмальною високорівневою мовою програмування [48]. Вона використовується для створення різних видів програмного забезпечення. Популярність мови програмування Python також зумовлена наявністю великої кількості фреймворків машинного навчання.

Кожна мова програмування має свої переваги та недоліки. Тому під час вирішення конкретних задач, треба вибирати саме ту мову програмування, що надає якнайбільше переваг (табл. 3.1).

Таблиця 3.1 – Порівняння мов програмування

Особливість	C++	C#	JavaScript	Python
1	2	3	4	5
Рік винайдення	1983	2001	1995	1991
Рік останнього оновлення	2020	2023	2022	2023
Тип типізації	Строга	Строга	Динамічна	Динамічна
Інтерпретованість	Ні	Так/Ні	Так	Так
Мультипарадигмальність	Так	Так	Так	Так
Швидкість виконання коду	Швидко	Середньо	Середньо	Середньо
Складність синтаксису	Складно	Середньо	Середньо	Легко
Керування пам'яттю	Вручну	Авто	Авто	Авто
Розмір програми	Велика	Велика	Середня	Мала

Продовження таблиці 3.1

1	2	3	4	5
Кількість фреймворків машинного навчання	Мало	Мало	Середньо	Багато
Спільнота	Середньо	Середньо	Середньо	Багато
Докладна програмна документація	Середньо	Багато	Багато	Багато

Відповідно до результатів порівняльного аналізу, можна сказати, що для вирішення задач навчання нейромережових моделей, класифікації та розпізнавання об'єктів на зображенні найкраще підходить мова програмування Python. Вона має простий та інтуїтивно зрозумілий синтаксис, має динамічну типізацію, автоматичний алгоритм керування виділеною пам'яттю та велику кількість гарно документованих ефективних бібліотек які можуть бути використані для машинного навчання.

Для розробки програмного забезпечення використовують інтегровані середовища розробки (IDE). Серед існуючих інтегрованих середовищ розробки було вибрано IDE PyCharm від компанії JetBrains. PyCharm є вільнорозповсюджуваним крос-платформним IDE, яке може працювати під керуванням різних операційних систем. PyCharm має широкий функціонал для аналіз та редагування програмного коду, запуску та відлагоджування програми.

3.2.2 Вибір бібліотеки для роботи з нейромережевими моделями

Важливою частиною реалізації програмного забезпечення для роботи з нейронними мережами є використання такої бібліотеки машинного навчання, яка допомагає ефективніше вирішити поставлені задачі. Серед багатьох бібліотек машинного навчання (рис. 3.4), на практиці найчастіше використовують TensorFlow та PyTorch.

TensorFlow – це відкрита вільнорозповсюджувана програмна бібліотека для машинного навчання, що дозволяє вирішувати завдання

побудови та тренування нейронних мереж [49]. Вона розроблена компанією Google.

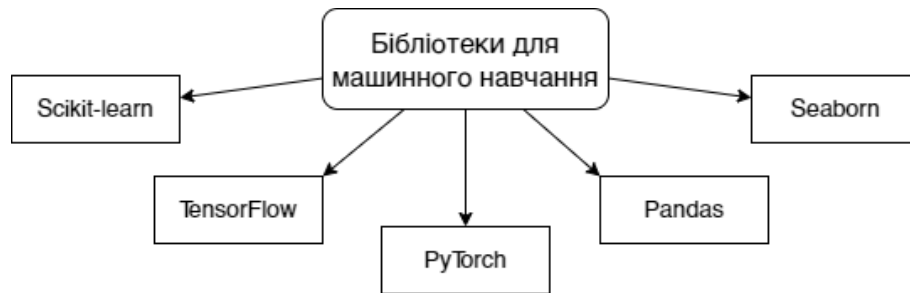


Рисунок 3.4 – Бібліотеки для машинного навчання

PyTorch – це фреймворк машинного навчання з відкритим вихідним кодом для мови програмування Python. Він складається з цілої екосистеми бібліотек, розроблених різними компаніями [50]. Функціонал бібліотек TensorFlow та PyTorch схожий, вони можуть використовувати графічний процесор для виконання обчислень, мають програмну документацію та достатній набір методів в API.

В результаті аналізу виявлених переваг та недоліків бібліотек TensorFlow та PyTorch (табл. 3.2), можна зробити висновок, що для вирішення поставленої задачі найкраще підходить бібліотека PyTorch. Вона частіше використовується для вирішення практичних задач, має більш легкий та зрозумілий API, дозволяє детально налаштовувати як архітектуру мережі так і процес навчання, має більшу спільноту та краще задокументована.

Таблиця 3.2 – Порівняння бібліотек для машинного навчання

Особливість	TensorFlow	PyTorch
Рік розробки	2015	2016
Підтримка глибинного навчання	Так	Так
Повний контроль над процесом навчання	Ні	Так
Обробка помилок	Під час компіляції	Під час виконання
Інтуїтивність API методів	Середньо	Добре
Кількість програмної документації та посібників	Середньо	Багато

3.2.3 Вибір бібліотеки для реалізації графічного інтерфейсу

Для зручної взаємодії користувача з програмним забезпеченням необхідно реалізувати графічний інтерфейс користувача, адже не кожному буде зручно спілкуватися з програмою лише за допомогою команд в командному рядку. Серед бібліотек які дозволяють реалізувати графічний інтерфейс можна виділити PySimpleGUI, Tkinter та PyQt (рис. 3.5).

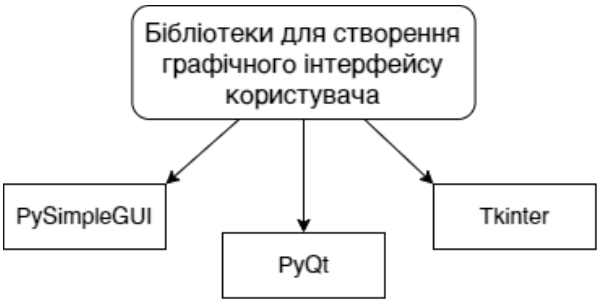


Рисунок 3.5 – Бібліотеки для створення графічного інтерфейсу

PySimpleGUI та Tkinter дозволяють швидко та зручно створювати простий графічний інтерфейс користувача. PyQt дозволяє створювати складні графічні інтерфейси з великою кількістю елементів та для редагування зовнішнього вигляду елементів використовувати таблиці стилів (табл. 3.3).

Таблиця 3.3 – Порівняння бібліотек для графічного інтерфейсу

Особливість	PySimpleGUI	Tkinter	PyQt
Стандартна бібліотека	Ні	Так	Ні
Кросплатформність та уніфікація інтерфейсу	Ні	Ні	Так
Редагування зовнішнього вигляду за допомогою таблиці стилів	Ні	Ні	Так
Кількість вбудованих елементів	Мало	Середньо	Багато
Можливість перевизначення вбудованих елементів	Ні	Ні	Так
Програмна документація	Середньо	Середньо	Багато

В результаті порівняння бібліотек графічного інтерфейсу, можна зробити висновок, що для створення графічного інтерфейсу користувача

найкраще підходить бібліотека PyQt. Для відображення статистичних даних у вигляді графіків прийнято рішення використовувати бібліотеку Matplotlib.

3.3 Прототипування графічного інтерфейсу користувача

Під час розробки ПЗ виконують прототипування графічного інтерфейсу. Тільки після узгодження прототипів із замовником приступають до побудови інтерфейсу та його інтеграції в систему. В якості прототипів графічного інтерфейсу користувача використовують вайрфрейми.

Відповідно до проведеного аналізу вимог, функціональної частини програми розроблено прототипи графічного інтерфейсу (рис. 3.6).

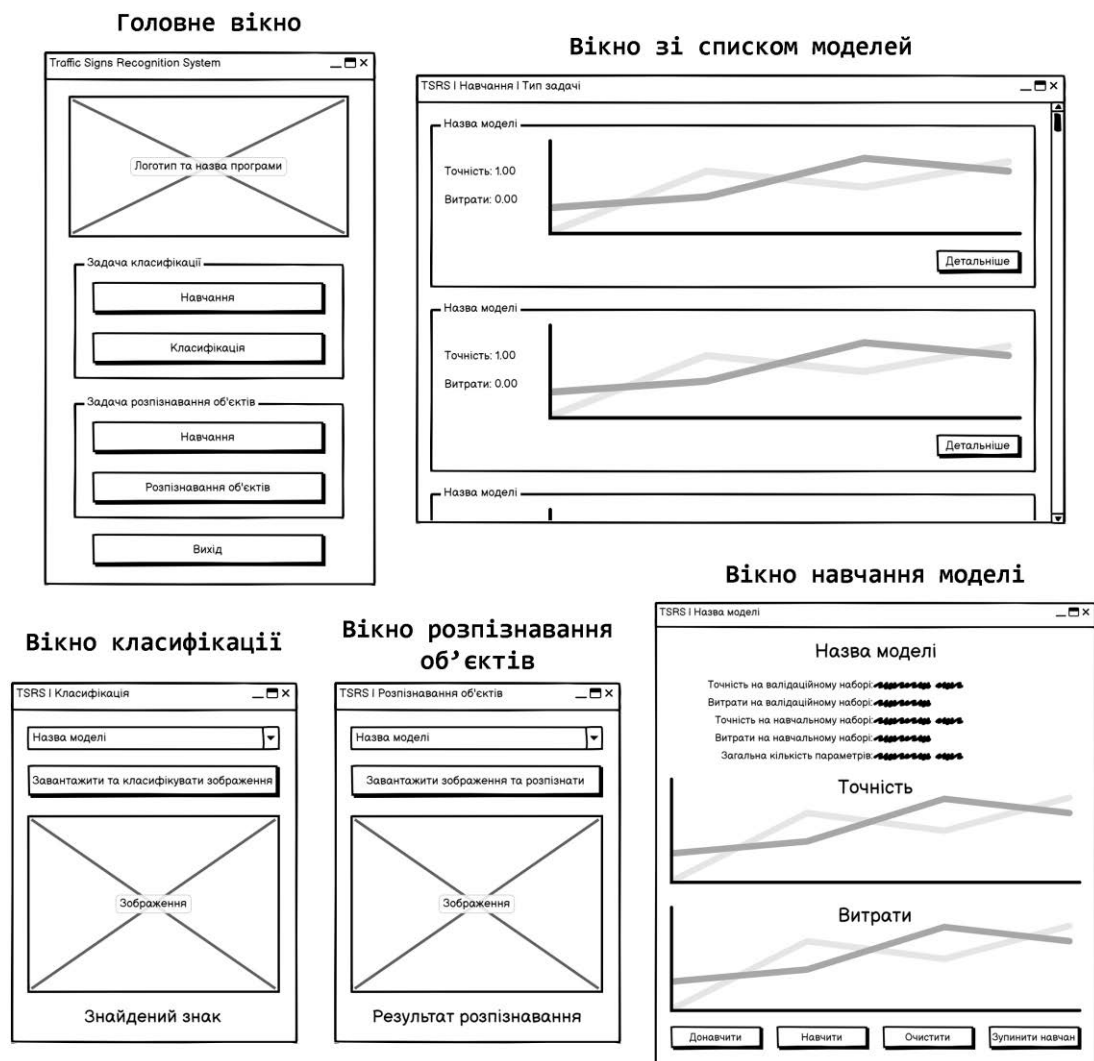


Рисунок 3.6 – Вайрфрейми графічного інтерфейсу програмного забезпечення

3.4 Висновки за розділом 3

Виконано докладний аналіз вимог до програмного забезпечення, що проєктується, відповідно до поставленої мети роботи. Визначено основні та допоміжні засоби розробки, виконано прототипування графічного інтерфейсу.

Відповідно до визначених функціональних та нефункціональних вимог, розроблюване програмне забезпечення має надавати можливість навчання нейромережевої моделі, завантаження зображення до системи з подальшою класифікацією чи розпізнаванням об'єктів на ньому за допомогою вибраної зі списку натренованої нейромережевої моделі з подальшим відображенням результатів. Графічний інтерфейс розроблюваного ПЗ має бути інтуїтивно зрозумілим, мати швидкий відклик на будь-які дії користувача в системі та бути стійким до неочікуваних виняткових ситуацій.

В якості інструментальних засобів вибрано інтерпретовану високорівневу мову програмування Python, бібліотеки PyTorch, PyQt та Matplotlib, інтегроване середовище розробки PyCharm.

4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Структура програмного забезпечення

Розроблена програмна реалізація складається з 1 проєкта, 8 основних папок, 34 виконуваних файлів, що містять 33 програмні класи. Основну структуру проєкта наведено на рисунку 4.1.

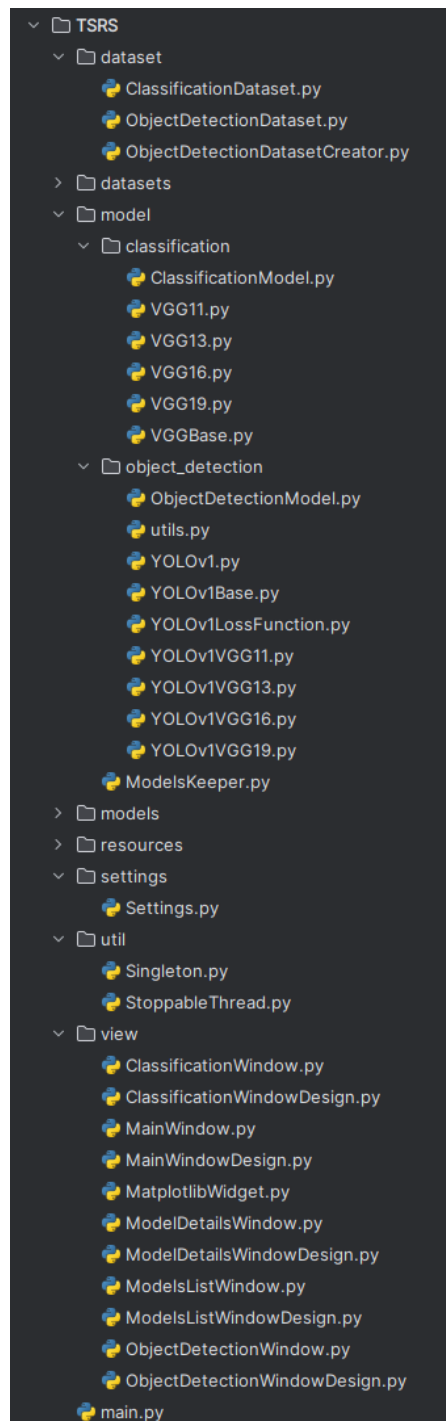


Рисунок 4.1 – Основна структура розробленого ПЗ

Діректорія зі скороченою назвою програми «TSRS» є кореневим каталогом проєкта програмного забезпечення. Вона містить в собі набір каталогів в яких зберігаються виконувані файли програми, набори даних для навчання, ваги та деталі навчених звичайних та вдосконалених нейромережових моделей VGG та YOLOv1, додаткові ресурси.

В папці `dataset` зберігаються виконувані файли, що використовуються для обробки, організації та підготовки наборів навчальних даних для задач класифікації та розпізнавання об'єктів.

Каталог `datasets` складається з папок `classification` та `object_detection`. В папці `datasets\classification` зберігається попередньо підготовлений навчальний набір даних для задачі класифікації, а в папці `datasets\object_detection` – набір даних для задачі розпізнавання об'єктів.

Каталог `model` складається з папок `classification` та `object_detection`. В папці `model\classification` зберігаються виконувані файли з програмними класами, що реалізують архітектури звичайних та вдосконалених нейромережових моделей VGG, організовують процес навчання та збереження моделей, класифікації зображень. В папці `model\object_detection` зберігаються виконувані файли з програмними класами, що реалізують архітектури звичайних та вдосконалених нейромережових моделей YOLOv1, організовують процес навчання та збереження моделей, розпізнавання об'єктів на зображенні.

Каталог `models` складається з папок `classification` та `object_detection`. В папках `models\classification` та `models\object_detection` зберігаються ваги та деталі навчених звичайних та вдосконалених нейромережових моделей для вирішення задач класифікації та розпізнавання об'єктів відповідно. При організації процесу донавчання, класифікації чи розпізнавання об'єктів ваги та деталі моделей завантажуються саме з цих папок. Кожна навчена модель зберігається в окремій папці з назвою моделі.

В папці `resources` зберігається логотип програми, зображення плейсхолдер, що використовується для вікон класифікації та розпізнавання

об'єктів, файл TrueType шрифту з кирилицею для додавання міток розпізнаних об'єктів на зображення.

В папці settings зберігається виконуваний файл, що містить основні налаштування ПЗ: шляхи до навчальних наборів даних, папок за замовчанням для збережених нейромережевих моделей.

В папці util зберігаються допоміжні виконувані файли, які реалізують породжувальний патерн проєктування Singleton та засоби роботи з багатонитевими процесами.

В папці view зберігаються виконувані файли, що реалізують графічний інтерфейс користувача, та відповідають за організацію взаємодії користувача з моделями даних.

В кореневій директорії зберігається виконуваний файл main.py, що містить логіку запуску графічного інтерфейсу користувача. З нього має починатися виконання програми.

4.2 Реалізовані програмні класи

Кожен програмний клас зберігається у відповідному файлі з назвою, яка відповідає назві програмного класу та розширенням «.py» за винятком класу-переліку TargetModelTask що зберігається в одному файлі з класом ModelsKeeper.

Клас ClassificationDataset призначений для підготовки навчального набору даних для задачі класифікації. Він відповідає за групування зображень та їх розподіл на дані для тренування, валідації та тестування. Також він відповідає за нормалізацію зображення, його конвертування його розмірів до таких, які приймаються на вхід спроектованою нейромережевою моделлю. Серед основних методів цього класу можна виділити такі:

- get_train_loader – отримання завантажувача даних для тренування;
- get_validation_loader – отримання завантажувача даних для валідації;
- get_classes – отримання переліку класів для класифікації;

– `get_default_image_inference_transforms` – статичний метод для отримання перетворень, що використовуються для нормалізації вхідного зображення.

Клас `ObjectDetectionDatasetCreator` призначений для створення завантажувача даних задачі розпізнавання об'єктів з відповідної папки з даними. Серед основних методів цього класу можна виділити такі:

- `__len__` – отримання кількості елементів в наборі даних;
- `__getitem__` – отримання елемента з набору даних за індексом.

Клас `ObjectDetectionDataset` призначений для підготовки навчального набору даних для задачі розпізнавання об'єктів. Він відповідає за розподіл даних на дані для тренування, валідації та тестування. Також він відповідає за нормалізацію зображення, його конвертування його розмірів до таких, які приймаються на вхід спроектованою нейромережевою моделлю. Серед основних методів цього класу можна виділити такі:

- `get_train_loader` – отримання завантажувача даних для тренування;
- `get_validation_loader` – отримання завантажувача даних для валідації;
- `get_classes` – отримання переліку класів для класифікації;
- `get_default_image_inference_transforms` – статичний метод для отримання перетворень, що використовуються для нормалізації вхідного зображення.

Клас `VGGBase` уособлює архітектуру нейромережевої моделі VGG загалом. Він описує основні компоненти її архітектури, приймаючи на вхід стек шарів відповідальних за виявлення особливостей. Серед основних методів класу `VGGBase` можна виділити такі:

- `__init__` – конструктор класу `VGGBase`;
- `forward` – отримання тензору з результатом обчислення вхідних даних за допомогою мережі;
- `freeze_features_weights` – заморожує ваги шарів для виявлення особливостей зображення. Використовується для навчання мережі методом `transfer learning`;

- `make_convolutional_layer` – статичний метод для отримання згорткового шару з вказаною кількістю вхідних та вихідних карт активації;

- `make_classifier` – статичний метод для отримання повнозв'язних шарів, що є класифікатором мережі, з розмірністю вихідного тензора, що дорівнює кількості класів.

Класи VGG11, VGG13, VGG16, VGG19 відповідають за створення архітектури відповідної до назви мережі. Вони наслідуються від класу VGGBase та перевизначають стек згорткових шарів відповідно до архітектури мережі. Основними методами класів VGG11, VGG13, VGG16, VGG19 є:

- `get_pretrained_features` – отримання попередньо натренованого класифікатора відповідної мережі на наборі даних ImageNet;

- `make_features` – отримання класифікатора відповідної мережі з випадково ініціалізованими значеннями ваг.

Клас `ClassificationModel` уособлює нейромережеву модель для класифікації загалом. Він відповідає за ініціалізацію чи завантаження однієї з архітектур нейромережевих моделей VGG, процес навчання та збереження мережі, передбачення на основі вхідних даних. Серед основних методів класу `ClassificationModel` можна виділити такі:

- `load_model_details` – завантаження статистичних деталей моделі;
- `load_model_weights` – завантаження значень ваг мережі;
- `init_new_model_and_optimizer` – ініціалізація мережі без значень ваг;
- `train` – навчання нейромережевої моделі за допомогою вибраного набору даних;

- `predict` – передбачення класу до якого належить вхідне зображення;
- `save` – збереження статистичних деталей, ваг та ваг оптимізатора нейромережевої моделі;

- `get_model_by_name` – статичний метод, що в якості моделі повертає архітектуру нейронної мережі VGG за назвою.

Клас `YOLOv1Base` уособлює архітектуру нейромережевої моделі `YOLOv1` загалом. Він описує основні компоненти її архітектури, приймаючи на вхід стек шарів відповідальних за виявлення особливостей. Серед основних методів класу `YOLOv1Base` можна виділити такі:

- `__init__` – конструктор класу `YOLOv1Base`;
- `forward` – отримання тензору з результатом обчислення вхідних даних за допомогою мережі;
- `freeze_features_weights` – заморожує ваги шарів для виявлення особливостей зображення, для навчання мережі методом `transfer learning`;
- `make_convolutional_layer` – статичний метод для отримання згорткового шару з вказаною кількістю вхідних та вихідних карт активації, розміру ядра згортки, кроку та вирівнювання;
- `make_features_neck` – статичний метод для отримання об'єднувальних шарів між шарами детектора та виявлення особливостей;
- `make_detector` – статичний метод для отримання повнозв'язних шарів, детектора мережі з відповідною до архітектури `YOLOv1` розмірністю.

Класи `YOLOv1`, `YOLOv1VGG11`, `YOLOv1VGG13`, `YOLOv1VGG16`, `YOLOv1VGG19` відповідають за створення архітектури відповідної до назви мережі. Вони наслідуються від класу `YOLOv1Base` та перевизначають стек згорткових шарів відповідно до архітектури використовуваної мережі.

Клас `YOLOv1LossFunction` є реалізацією функції втрат нейромережевих моделей `YOLOv1`. Він має один метод `forward`, який відповідає зарозрахунок значення втрат між вихідними даними мережі та еталонними даними з навчального набору.

Клас `ObjectDetectionModel` уособлює нейромережеву модель для розпізнавання зображень загалом. Він відповідає за ініціалізацію чи завантаження однієї з архітектур нейромережевих моделей `YOLOv1`, процес навчання та збереження мережі, передбачення на основі вхідних даних. Клас `ObjectDetectionModel` за набором методів схожий на клас `ClassificationModel`. Серед основних методів класу `ObjectDetectionModel` можна виділити такі:

- `train` – навчання нейромережевої моделі за допомогою вибраного набору даних задачі розпізнавання зображень;
- `predict` – передбачення набору обмежувальних коробок і їх імовірностями приналежності до класів для розпізнаних об'єктів на вхідному зображенні;
- `get_model_by_name` – статичний метод, що в якості моделі повертає архітектуру нейронної мережі YOLOv1 за назвою.

Клас `ModelsKeeper` є агрегувальною сутністю. Він містить всі нейромережеві моделі VGG та YOLOv1 та відповідає за надання всім вікнам програми доступу до моделей. Серед основних методів класу `ModelsKeeper` можна виділити такі:

- `get_models` – отримання списку моделей за певним типом задачі;
- `get_model_by_name` – отримання моделі за назвою та типом задачі.

Клас `Settings` зберігає набір параметрів за замовчанням, які використовуються програмою.

Клас `Singleton` реалізує породжувальний патерн проєктування `Singleton` та використовується в якості батьківського класу `ModelsKeeper`.

Клас `StoppableThread` реалізує засоби роботи з багатонитевими процесами, а саме можливість екстреної зупинки потоку.

Клас `MatplotlibWidget` реалізує інструменти відображення статистичних даних у вигляді графіків для віджетів `PyQt`.

Клас `MainWindow` реалізовує графічний інтерфейс та обробку дій користувача в головному вікні програми.

Клас `ModelsListWindow` реалізовує графічний інтерфейс та обробку дій користувача в вікні зі списком моделей.

Клас `ModelDetailsWindow` реалізовує графічний інтерфейс та обробку дій користувача в вікні навчання мережі.

Клас `ClassificationWindow` реалізовує графічний інтерфейс та обробку дій користувача в вікні класифікації зображень.

Клас `ObjectDetectionWindow` реалізовує графічний інтерфейс та обробку дій користувача в вікні розпізнавання об'єктів на зображенні.

Класи з постфіксом `Design` реалізують складові та основні стилі відповідних вікон програми які описані у відповідних класах.

На рисунку 4.2 представлено діаграму розроблених програмних класів програмного забезпечення.

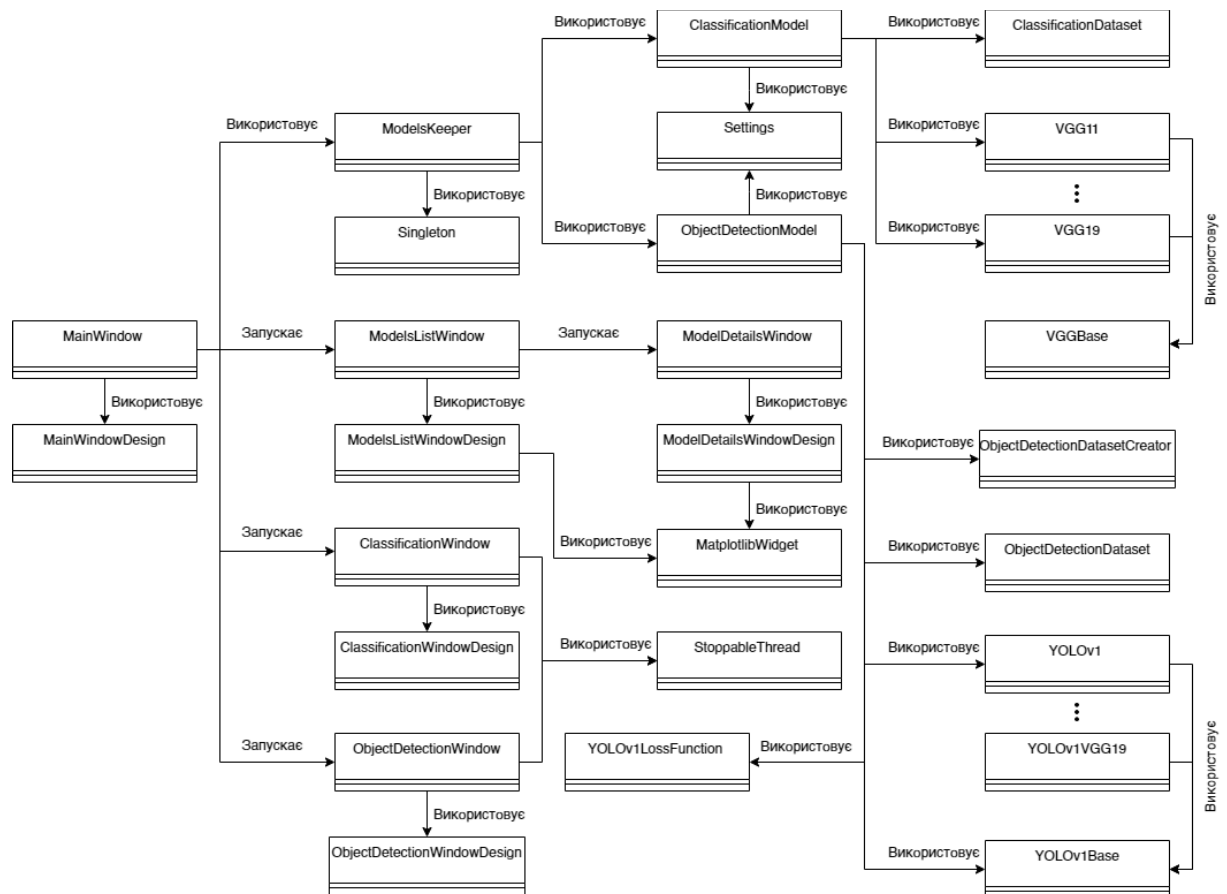


Рисунок 4.2 – Діаграма класів програми

4.3 Реалізація нейромережевих моделей

Основою програми є набір нейромережевих моделей які використовують для вирішення задач класифікації та розпізнавання об'єктів на зображенні. Для реалізації архітектури нейронних мереж VGG та YOLOv1 за допомогою бібліотеки машинного навчання PyTorch використано набір

шарів з пакету `torch.nn`. Короткий опис та основні характеристики використаних шарів наведено в таблиці 4.1.

Таблиця 4.1 – Використані шари для побудови нейрних мереж

Шар	Опис	Параметри
<code>torch.nn.Conv2d</code>	Згортковий шар	Кількість фільтрів на вході та виході Розмір ядра згортки Значення кроку та вирівнювання
<code>torch.BatchNorm2d</code>	Шар батч-нормалізації	Кількість каналів на виході
<code>torch.nn.ReLU</code>	Функція активації ReLU	-
<code>torch.nn.LeakyReLU</code>	Функція активації LeakyReLU	Значення коефіцієнту для від'ємного значення активації
<code>torch.nn.MaxPool2d</code>	Агрегувальний шар, в якому використовується максимізаційне агрегування	Розмір фільтра Крок
<code>torch.nn.AdaptiveAvgPool2d</code>	Адаптація розмірності даних до певних значень	Вихідний розмір
<code>torch.Flatten</code>	Операція згладжування тензору	Розмірність
<code>torch.nn.Linear</code>	Повнозв'язний шар	Кількість вхідних та вихідних каналів
<code>torch.nn.Dropout</code>	Шар випадкового вилучення	Значення коефіцієнту вилучення
<code>torch.nn.Softmax</code>	Нормована експоненційна функція	Розмірність

4.4 Структура навчальних даних

4.4.1 Структура навчальних даних для класифікації

Для навчання нейромережових моделей VGG для задачі класифікації використано набір даних з БД German Traffic Sign Recognition Benchmark (GTSRB) [51]. Для експериментального дослідження нейромережових моделей VGG розмір набору даних було зменшено до 2500 зображень, що містять 10 різних видів найважливіших дорожніх знаків: знаки пріоритетів, обмежувальні та попереджальні.

Для роботи з навчальним набором даних для класифікації використовується клас `ClassificationDataset`. Він використовує клас `torchvision.datasets.ImageFolder` для зчитування даних, та повертає завантажувач даних використовуючи клас `torch.utils.data.DataLoader`, який використовується під час навчання мереж з метою ділення набору даних на

невеликі батчі. При використанні такого підходу зображення кожного з видів класу зберігаються в папці з назвою цього класу. Всі папки з класами розміщуються в одному каталозі. Один каталог відповідає за один набір даних, наприклад дані для навчання чи валідації (рис. 4.3).

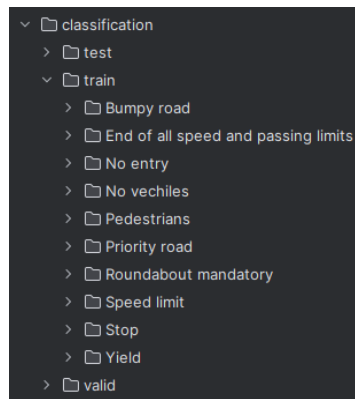


Рисунок 4.3 – Структура навчального набору даних для класифікації

4.4.2 Структура навчальних даних для розпізнавання об'єктів

Для навчання нейромережових моделей YOLOv1 використано набір даних з платформи для змагань з аналітики та моделювання нейронних мереж Kaggle [52]. Набір містить 877 зображень з чотирма класами дорожніх знаків та об'єктів: світлофори, знаки стоп, обмеження швидкості та пішохідний перехід.

Для роботи з навчальним набором даних для розпізнавання об'єктів використовується клас `ObjectDetectionDataset`. Він використовує клас `ObjectDetectionDatasetCreator` для зчитування даних, та повертає завантажувач даних використовуючи клас `torch.utils.data.DataLoader`. Для зберігання даних було використано формат YOLO [53]. При використанні такого формату всі зображення зберігаються в папці `images`. Для кожного зображення існує файл з відповідною назвою та розширенням `«.txt»` в папці `labels`. Цей файл містить опис об'єктів на зображенні в форматі YOLO. Кожен рядок це новий об'єкт, перше значення в рядку це індекс класу, до якого приналежить об'єкт, а наступні чотири значення відповідно є

координатами центру та розмірами обмежувальної рамки цього об'єкту (розміри та координати описуються відносно розміру всього зображення). В каталозі з папками `images` та `labels` зберігаються файли які містять список зображень для навчання, валідації та тестування, та файл з описом набору даних (рис. 4.4).

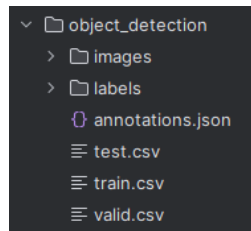


Рисунок 4.4 – Структура набору даних для розпізнавання об'єктів

4.5 Реалізація графічного інтерфейсу користувача

На рисунку 4.5 наведено скріншот головного вікна розробленого програмного забезпечення.



Рисунок 4.5 – Графічний інтерфейс головного вікна

На рисунку 4.6 наведено скріншот вікна зі списком нейромережевих моделей.

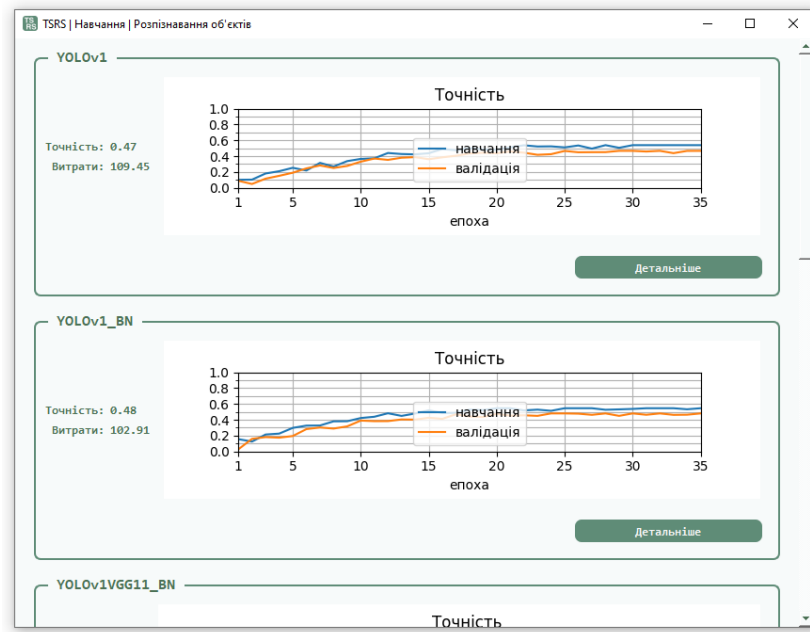


Рисунок 4.6 – Графічний інтерфейс вікна зі списком моделей

На рисунку 4.7 наведено скріншот вікна навчання та перегляду деталей однієї з нейромережевих моделей.

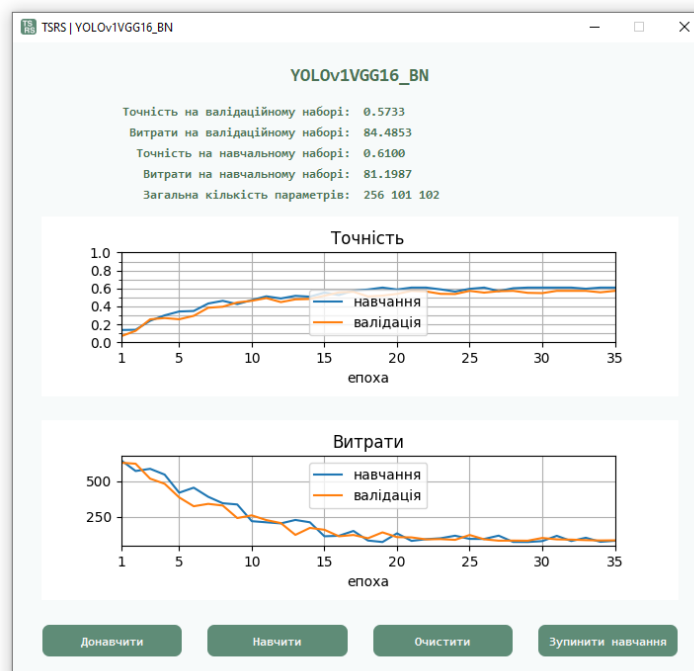


Рисунок 4.7 – Графічний інтерфейс вікна навчання та перегляду деталей

На рисунку 4.8 наведено скріншот вікна класифікації вхідного зображення.

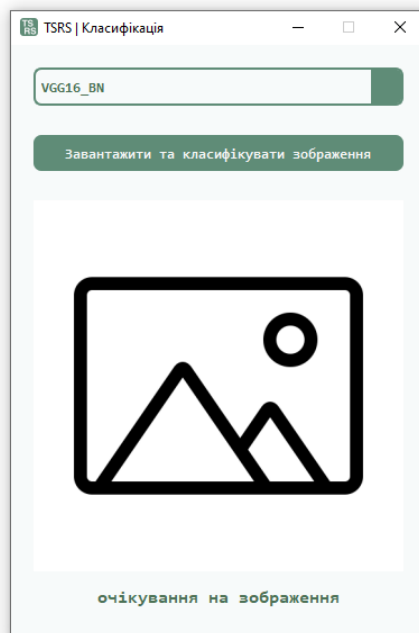


Рисунок 4.8 – Графічний інтерфейс вікна класифікації вхідного зображення

На рисунку 4.9 наведено скріншот вікна розпізнавання об'єктів на вхідному зображенні.



Рисунок 4.9 – Графічний інтерфейс вікна розпізнавання об'єктів

4.6 Висновки за розділом 4

За результатами проведеної роботи було реалізовано основну структуру попередньо спроектованого програмного забезпечення, розроблено всі необхідні для роботи програмні класи та виконувані файли.

В результаті реалізації програмного забезпечення за допомогою інструментів бібліотеки PyTorch реалізовано звичайні та вдосконалені архітектури нейромережових моделей VGG для класифікації зображень та YOLOv1 для розпізнавання об'єктів на зображенні. Обрано навчальні набори даних для вирішення задачі розпізнавання дорожніх знаків на зображенні та розглянуто їх структуру.

Розроблено графічний інтерфейс користувача відповідно до висунутих функціональних та нефункціональних вимог.

5 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування програмного забезпечення

Для перевірки розробленого програмного забезпечення на відповідність висунутим функціональним та нефункціональним вимогам необхідно виконати його тестування. Процес тестування та перевірки якості ПЗ є невід'ємною частиною його життєвого циклу та складається з планування, проєктування та виконання написаних тестів.

Для перевірки дотримання основних функціональних вимог до програмного забезпечення використовують тест-кейси, які описують набір кроків та умов, які необхідні для перевірки працездатності тестованого функціоналу або його частини. Відповідно до функціональних вимог, які були висунуті до ПЗ, створено необхідну сукупність тест-кейсів, за допомогою яких проведено тестування усіх основних функціональних можливостей розробленої програми (табл. 5.1 – табл. 5.4).

Таблиця 5.1 – Тест-кейс перегляду моделей та їх деталей

Короткий опис	Перевірка можливості перегляду нейромережових моделей та їх деталей			
Передумови	Відкрито головне вікно з меню програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	В підменю задачі класифікації натиснути на кнопку навчання	Відкрито вікно з моделями для задачі класифікації зображень. Кожна модель містить графік, значення її точності та витрат	Пройдено
	3	Натиснути на кнопку детальніше будь-якої нейромережової моделі зі списку для класифікації	Відкрито вікно навчання моделі. Відображено назву вибраної мережі її статистичні деталі, графіки точності та витрат	Пройдено
	4	Закрити вікно навчання та вікно зі списком моделей	Відкрито головне вікно з меню програми	Пройдено
	5	В підменю задачі розпізнавання об'єктів натиснути на кнопку навчання	Відкрито вікно з моделями для задачі розпізнавання об'єктів на зображенні. Кожна модель містить графік, значення її точності та витрат	Пройдено
	6	Натиснути на кнопку детальніше будь-якої неромережової моделі зі списку для розпізнавання об'єктів на зображенні	Відкрито вікно навчання моделі. Відображено назву вибраної мережі її статистичні деталі, графіки точності та витрат	Пройдено

Таблиця 5.2 – Тест-кейс навчання нейромережевої моделі

Короткий опис	Перевірка можливості навчання нейромережевої моделі			
Передумови	Відкрито головне вікно з меню програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	В підменю задачі розпізнавання об'єктів натиснути на кнопку навчання	Відкрито вікно з моделями для задачі розпізнавання об'єктів на зображенні	Пройдено
	3	Натиснути на кнопку детальніше будь-якої нейромережевої моделі	Відкрито вікно навчання моделі.	Пройдено
	4	Натиснути на кнопку навчити	Якщо модель не була попередньо навчена, то розпочато навчання. Якщо модель була попередньо навчена, то відображено діалогове вікно після погодження з яким прогрес навчання скинуто та розпочато знов. Поруч з назвою моделі з'явилося відповідне повідомлення.	Пройдено
	5	Натиснути на кнопку зупинити навчання	Навчання моделі зупинено	Пройдено
	6	Натиснути на кнопку донавчити	Розпочато процес донавчання попередньо натренованої моделі. Якщо модель не була попередньо натренованою, то розпочато процес навчання. Поруч з назвою моделі з'явилося відповідне повідомлення.	Пройдено
	7	Зупинити навчання та натиснути на кнопку очистити	Якщо модель натренована, то відображено діалогове вікно після погодження з яким процес навчання буде скинуто	Пройдено
	8	Закрити вікно навчання моделі	Відкрито вікно з моделями для задачі розпізнавання об'єктів	Пройдено

Таблиця 5.3 – Тест-кейс класифікації зображення

Короткий опис	Перевірка можливості класифікації зображення			
Передумови	Відкрито головне вікно з меню програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	В підменю задачі класифікації натиснути на кнопку класифікація	Відкрито вікно для завантаження та класифікації зображення	Пройдено
	3	Натиснути на комбінований список моделей	Відображено список навчених моделей для задачі класифікації.	Пройдено
	4	Вибрати зі списку модель	Вибрано модель для класифікації. В якості вибраного значення в комбінованому списку написано назву вибраної моделі	Пройдено
	5	Натиснути на кнопку завантажити та класифікувати зображення	Відкрито системний провідник файлів для вибору файлу зображення з пам'яті комп'ютера	Пройдено
	6	Вибрати зображення та натиснути кнопку відкрити	Замість плейсхолдера у вікні відображено вибране зображення. Розпочато процес класифікації. В результаті класифікації результат відображено під зображенням	Пройдено

Тест-кейс розпізнавання об'єктів наведено в таблиці 5.4. Інші перевірки оформлено у вигляді контрольного списку (табл. 5.5).

Таблиця 5.4 – Тест-кейс розпізнавання об'єктів на зображенні

Короткий опис	Перевірка можливості розпізнавання об'єктів на зображенні			
Передумови	Відкрито головне вікно з меню програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	В підменю задачі розпізнавання об'єктів натиснути на кнопку розпізнавання об'єктів	Відкрито вікно для завантаження зображення та розпізнавання об'єктів на ньому	Пройдено
	3	Натиснути на комбінований список моделей	Відображено список навчених моделей для задачі розпізнавання об'єктів.	Пройдено
	4	Вибрати зі списку модель	В комбінованому списку написано назву вибраної моделі	Пройдено
	5	Натиснути на кнопку завантажити зображення та розпізнати об'єкти	Відкрито системний провідник файлів для вибору файлу зображення з пам'яті комп'ютера	Пройдено
	6	Вибрати зображення та натиснути кнопку відкрити	Замість плейсхолдера у вікні відображено вибране зображення. Розпочато процес розпізнавання об'єктів. В результаті розпізнавання об'єктів знайдені об'єкти виділено обмежувальними рамками	Пройдено

Таблиця 5.5 – Контрольний список

Вимога	Статус
Головне вікно з меню програми	
Відображення назви і логотипу програми	Пройдено
Фіксований розмір та макет вікна	Пройдено
Вікна зі списками нейромережових моделей	
Наявність нейромережових моделей у списку відповідно до задачі	Пройдено
Відображення назви моделей, графіків, значень точності та втрат	Пройдено
Прокуртка вікна за допомогою повзунка чи колеса миші	Пройдено
Змінюваний розмір вікна та адаптивний макет	Пройдено
Вікно навчання моделі та перегляду її деталей	
Наявність детальної інформації про модель	Пройдено
Можливість організації навчання та донавчання моделі	Пройдено
Вікно класифікації зображення	
Наявність навчених моделей в комбінованому списку	Пройдено
Можливість вибрати та завантажити зображення з пам'яті комп'ютера	Пройдено
Відображення зображення та результату класифікації	Пройдено
Вікно розпізнавання об'єктів на зображенні	
Наявність навчених моделей в комбінованому списку	Пройдено
Можливість вибрати та завантажити зображення з пам'яті комп'ютера	Пройдено
Відображення зображення та результатів розпізнавання об'єктів	Пройдено
Загально	
Взаємодія графічного інтерфейсу з діями користувача	Пройдено
Читабельний шрифт та розмір тексту	Пройдено
Повідомлення користувачу при виконанні потенційно не бажаних дій	Пройдено

В результаті успішно виконаного тестування програмного забезпечення, можна сказати, що воно повністю відповідає всім висунутим функціональним та нефункціональним вимогам.

5.2 Експериментальне дослідження програмного забезпечення

5.2.1 Задача класифікації зображень

Навчання кожної з нейромережових моделей, що реалізують базові та вдосконалені архітектури сімейства VGGNet для класифікації зображення виконувалось впродовж 10 епох на навчальному наборі даних, що складається з зображень різних видів дорожніх знаків.

Під час навчання нейромережових моделей задачі класифікації, було збережено ваги, інформацію про кількість параметрів, історію змін точності та витрат, для всіх реалізованих моделей. Графіки зміни точності нейромережових моделей VGG під час навчання наведено на рисунку 5.1.

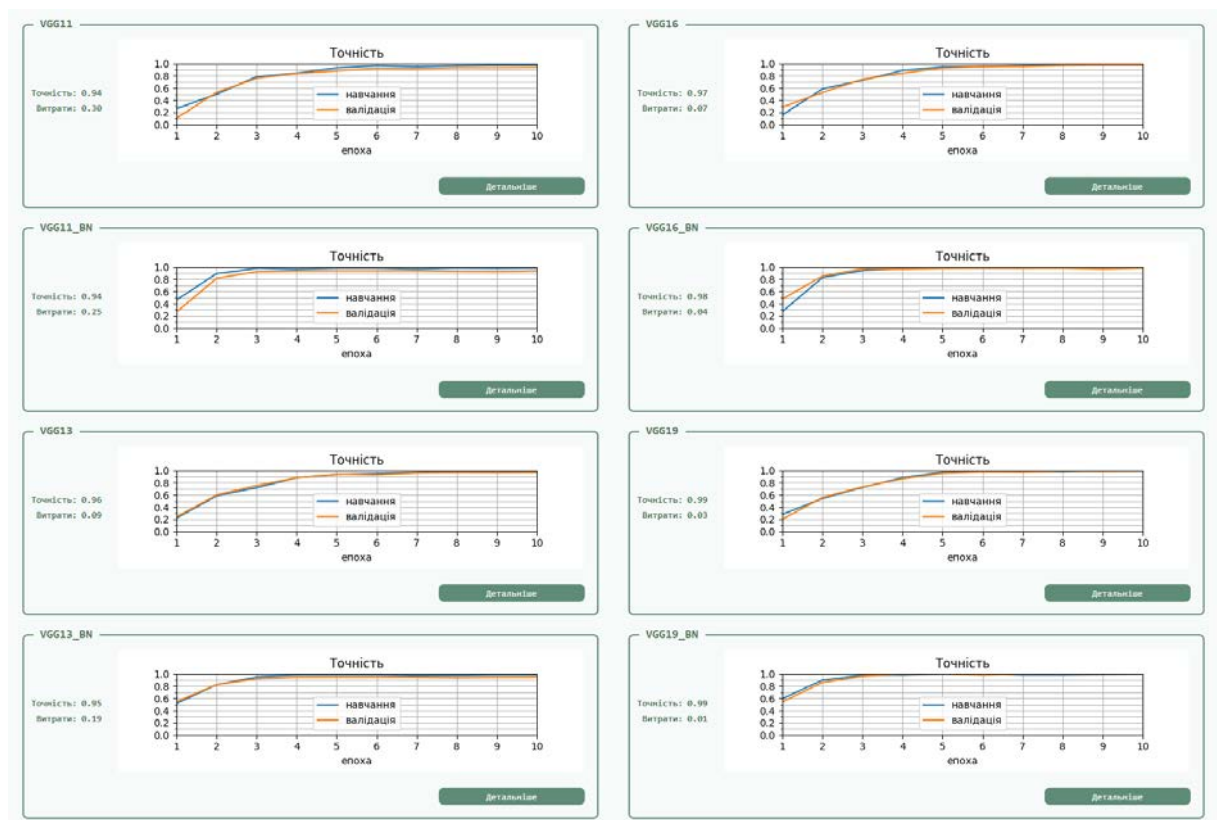


Рисунок 5.1 – Графіки зміни точності моделей під час навчання

За допомогою розробленого програмного забезпечення нейромережеві моделі можуть бути використані для вирішення задачі класифікації або донавчання на оновленому чи доповненому наборі даних.

Всі реалізовані нейромережеві моделі архітектури VGG досягли точності класифікації зображень більше ніж 93% на валідаційному наборі даних. Детальне порівняння отриманих результатів наведено в таблиці 5.6.

Таблиця 5.6 – Порівняння отриманих результатів моделей VGG

Характеристика	VGG11	VGG11 BN	VGG13	VGG13 BN	VGG16	VGG16 BN	VGG19	VGG19 BN
Точність моделі на наборі даних для навчання, %	97,15	97,95	98,35	98,00	98,65	98,75	99,25	99,30
Точність моделі на наборі даних для валідації, %	93,60	94,00	96,40	94,80	97,20	98,00	98,80	99,20
Втрати моделі на наборі даних для навчання	0,0509	0,0524	0,0435	0,0515	0,0353	0,0318	0,0212	0,0110
Втрати моделі на наборі даних для валідації	0,3031	0,2508	0,0904	0,1905	0,0734	0,0420	0,0309	0,0118
Кількість шарів згортки	8	8	10	10	13	13	16	16
Кількість параметрів з вагами, тис. шт.	128807	128812	128991	128997	134301	134309	139611	139622
Вага файлу з вагами, МБ	503,162	503,218	503,884	503,946	524,628	524,714	545,371	545,482
Епохи	10	10	10	10	10	10	10	10
Час навчання однієї епохи	40хв	25 хв	50хв	35хв	1 год 10 хв	45 хв	1 год 20 хв	50 хв
Загальний час навчання	6 год 40 хв	4 год 10 хв	8 год 20 хв	5 год 50 хв	11 год 40 хв	7 год 30 хв	13 год 20 хв	8 год 20 хв
Епоха на якій перевищено точність у 80% на навчальному наборі даних	4	2	4	2	4	2	4	2
Епоха на якій перевищено точність у 80% на валідаційному наборі даних	4	2	4	2	4	2	4	2
Епоха на якій перевищено точність у 90% на навчальному наборі даних	5	3	5	3	5	3	5	3
Епоха на якій перевищено точність у 90% на валідаційному наборі даних	6	3	5	3	5	3	5	3

Відповідно до отриманих результатів можна сказати, що на точність класифікації зображень нейромереживих моделей VGG значно впливає кількість використовуваних згорткових шарів в архітектурі моделі. Так модель VGG19 з найбільшою кількістю згорткових шарів досягає точності на валідаційному наборі даних в 98,8%, модель VGG16 в 97,2% а модель VGG11 з найменшою кількістю згорткових шарів – лише в 93,6%.

Збільшення кількості згорткових шарів впливає на збільшення кількості вагових параметрів, що призводить до збільшення ваги файлу в якому зберігаються ваги. Так ваги моделі VGG11 з найменшою кількістю вагових параметрів можуть бути записані у файлі розміром 503 МБ, а ваги моделі VGG19 з найбільшою кількістю вагових параметрів у файлі розміром 545 МБ. Найбільша різниця у розмірі файлів становить 42 МБ, та є достатньо маленьким значенням в масштабі сучасних програмних систем.

Збільшення кількості вагових параметрів також впливає на ресурсоемність обчислень, що призводить до збільшення часу необхідного для проведення всіх розрахунків та оптимізації нейромережевої моделі. Так при навчанні моделей базової архітектури VGG на навчальному наборі даних, загальний час навчання нейромережевої моделі VGG11 склав 6 год. 40 хв., моделі VGG13 – 8 год. 20 хв., моделі VGG16 – 11 год. 40 хв., а моделі VGG19 – 13 год. 20хв.

За допомогою використання шарів батч-нормалізації вдалось зменшити загальний час навчання моделей з вдосконаленою архітектурою VGG майже в 1,5 рази. Так загальний час навчання вдосконаленої нейромережевої моделі VGG11_BN склав 4 год. 10 хв, VGG13_BN склав 5 год. 50 хв, VGG16_BN склав 7 год. 30 хв, VGG19_BN склав 8 год. 20 хв.

Також використання шарів батч-нормалізації добре вплинуло на узагальнюючі можливості мереж, що посприяло більш швидкому досягненню точності класифікації. Так моделі з вдосконаленою архітектурою вже після третьої епохи змогли досягнути точності класифікації у 90%, на відміну від моделей з базовою архітектурою, які досягли такої ж точності лише після п'ятої епохи. При цьому деякі з вдосконалених моделей перевершили базові моделі в точності класифікації як на навчальному, так і на валідаційному наборі даних.

Швидкість навчання та точність результатів класифікації є важливими характеристиками, на основі яких роблять вибір, яку архітектуру використовувати при вирішенні тих чи інших прикладних задач. Моделі з

вдосконаленою архітектурою VGG показали менший час навчання та кращу узагальнюючу спроможність.

Відповідно до отриманих результатів, можна сказати, що нейромережеві моделі з вдосконаленою архітектурою VGG показали кращі результати, ніж моделі які реалізують базову архітектуру. Тим самим доведено ефективність запропонованих змін до архітектури мереж.

5.2.2 Задача розпізнавання об'єктів на зображенні

Навчання кожної з нейромережевих моделей, що реалізують базову та вдосконалені архітектури мережі YOLOv1 для розпізнавання об'єктів на зображенні виконувалось впродовж 35 епох на навчальному наборі даних, що складається з зображень різних видів дорожніх знаків.

Під час навчання нейромережевих моделей задачі розпізнавання об'єктів на зображенні, було збережено ваги, інформацію про кількість параметрів, історію змін точності та витрат, для всіх реалізованих моделей. Графіки зміни точності нейромережевих моделей YOLOv1 під час навчання наведено на рисунку 5.2.

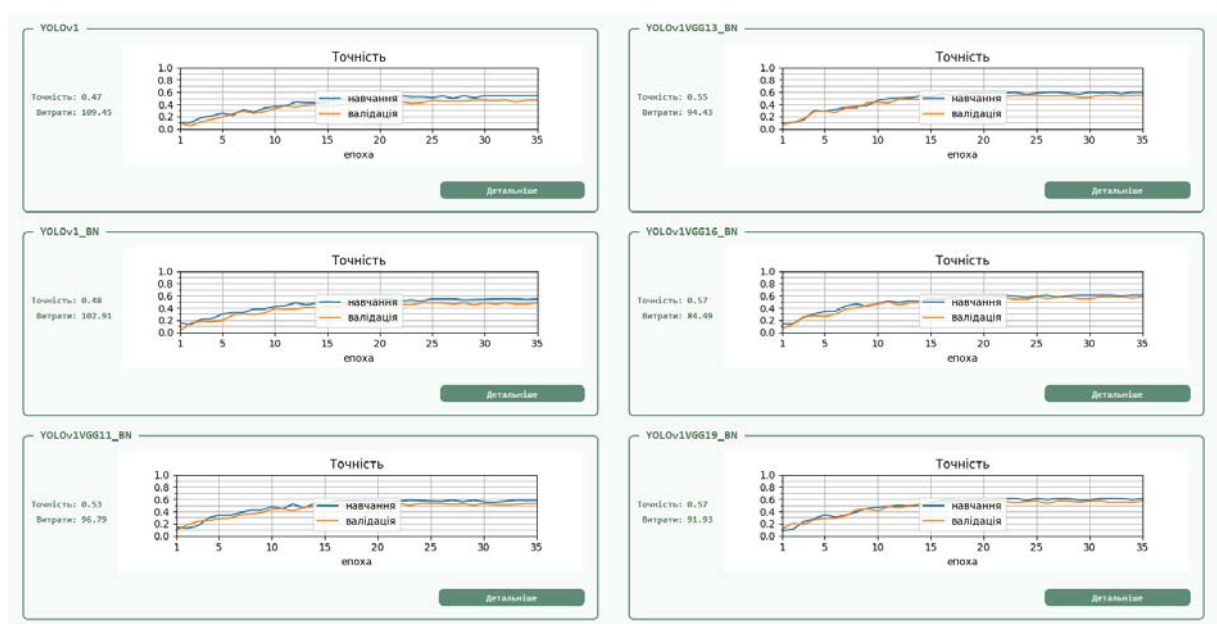


Рисунок 5.2 – Графіки зміни точності моделей під час навчання

Реалізовані нейромережеві моделі архітектури YOLOv1 досягли середньої точності розпізнавання об'єктів на зображенні більше ніж 45% на валідаційному наборі даних. Детальне порівняння отриманих результатів наведено в таблиці 5.7.

Таблиця 5.7 – Порівняння отриманих результатів моделей YOLOv1

Характеристика	YOLOv1	YOLOv1 BN	YOLOv1 VGG11 BN	YOLOv1 VGG13 BN	YOLOv1 VGG16 BN	YOLOv1 VGG19 BN
Точність моделі на наборі даних для навчання, %	53,86	54,57	58,57	59,71	61,00	61,14
Точність моделі на наборі даних для валідації, %	46,67	48,00	53,33	54,67	57,33	57,33
Втрати моделі на наборі даних для навчання	86	90	83	78	81	77
Втрати моделі на наборі даних для валідації	109	102	96	94	84	91
Кількість шарів згортки	20+4	20+4	8+4	10+4	13+4	16+4
Функція активації	LReLU Linear	LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear
Кількість параметрів з вагами, тис. шт.	268491	268517	250603	250788	256101	261413
Вага файлу з вагами, МБ	1048	1049	979	979	1000	1021
Епохи	35	35	35	35	35	35
Час навчання однієї епохи	25 хв	20хв	40хв	45 хв	1 год	1 год 10 хв
Загальний час навчання	14 год 35 хв	11 год 40хв	23 год 20 хв	26 год 15 хв	35 год	40 год 50 хв

Відповідно до отриманих результатів можна сказати, що на точність розпізнавання об'єктів на зображенні звичайних та вдосконалених нейромережевих моделей YOLOv1 впливає не тільки кількість згорткових шарів, а й особливості використовуваних розмірів ядер згортки та їх комбінацій.

Всі порівнювані моделі мають приблизну кількість вагових параметрів, а файли в яких зберігаються ці ваги займають приблизно однаковий розмір в 1 ГБ. Найбільша різниця у розмірі файлів становить 69 МБ, що є достатньо маленьким значенням для сучасних програмних систем.

Основна різниця між реалізованими моделями – це точність розпізнавання об’єктів на зображенні та час їх навчання. Мережі YOLOv1, які використовують в якості виявляча особливостей вдосконалену архітектуру мереж VGG досягають більшої точності ніж моделі на основі базової архітектури проте вимагають майже в 2-3 рази більше часу на навчання.

Так на наборі даних для валідації моделі YOLOv1_VGG19_BN та YOLOv1_VGG16_BN досягають точності в 57,33%, модель YOLOv1_VGG13_BN – 54,67%, модель YOLOv1_VGG11_BN – 53,333%, YOLOv1_BN – 48% та базова модель YOLOv1 – 46,67%.

За допомогою використання шарів батч-нормалізації вдалось зменшити загальний час навчання моделі з вдосконаленою архітектурою YOLOv1_BN у порівнянні з базовою архітектурою YOLOv1 на 20%.

Відповідно до отриманих результатів можна сказати, що ефективно вдосконалити базову архітектуру нейромережевої моделі YOLOv1 можна декількома способами. Якщо основною метою є зменшення часу навчання то архітектуру мережі можна вдосконалити додав шари батч-нормалізації. Якщо основною метою є поліпшення точності розпізнавання, то для вдосконалення архітектури мережі варто інтегрувати вдосконалену архітектуру нейромережевих моделей VGG в якості виявляча особливостей. Такий підхід є доцільним для використання в складних програмних системах які можуть довго проходити стадію розробки, проте для яких найважливіше значення має значення точності розпізнавання.

5.3 Висновки за розділом 5

В результаті роботи успішно протестовано розроблене програмне забезпечення, проведено експериментальне дослідження ефективності вдосконалення архітектур нейромережевих моделей для задач класифікації та розпізнавання об’єктів на зображенні.

Виконано тестування на визначеному контрольному списку та наборі тест-кейсів. Відповідно до результатів проведеного тестування можна сказати, що розроблене програмне забезпечення повністю відповідає всім висунутим функціональним та нефункціональним вимогам.

Проведено експериментальне дослідження розробленого програмного забезпечення. Відповідно до отриманих результатів експериментального дослідження, можна сказати, що нейромережеві моделі з вдосконаленою архітектурою показали кращі результати, ніж моделі які реалізують базову архітектуру, що доводить ефективність запропонованих змін до архітектури мереж.

6 КЕРІВНИЦТВО ПРОГРАМІСТА

6.1 Призначення та умови застосування програмного забезпечення

Програмне забезпечення призначено для вирішення задачі класифікації та розпізнавання дорожніх знаків на зображенні за допомогою використання нейромережових моделей VGG та YOLOv1.

Воно надає можливість навчання нейромережових моделей, завантаження зображення до системи з подальшою класифікацією чи розпізнаванням об'єктів на ньому за допомогою вибраної зі списку натренованої нейромережової моделі та подальшим відображенням результатів.

Програма розроблена мовою програмування Python за допомогою інтегрованого середовища розробки PyCharm. Програмне забезпечення базується на використанні бібліотек PyTorch, Pandas, Pillow, PyQt та Matplotlib.

В якості апаратних засобів для використання програмного забезпечення повинен використовуватися комп'ютер з характеристиками не нижче:

- процесор Intel Core i5-7xxx;
- RAM 8ГБ;
- жорсткий диск з обсягом вільного місця 16ГБ;
- маніпулятор типу миша і/або клавіатура.

В якості програмних засобів для використання програмного забезпечення повинні використовуватись:

- ОС Windows 10 чи новіше;
- Python 3.11;
- система керування пакетами pip 23.3.1;
- бібліотека PyTorch 2.1.0;
- бібліотека Pandas 2.1.2;

- Pillow 10.1.0;
- PyQt 6.4.2;
- Matplotlib 3.8.0.

6.2 Вимоги до програмного забезпечення

Функціональні вимоги до програми:

- можливість вибору необхідної моделі зі списку нейронних мереж;
- можливість навчання та донавчання моделі на довільному навчальному наборі даних;
- можливість перегляду детальної інформації про модель, результатів та графіків її навчання;
- можливість вибору моделі для класифікації або розпізнавання об'єктів на зображенні;
- можливість завантаження зображення, його класифікація та відображення результату;
- можливість завантаження зображення, розпізнавання об'єктів на ньому та відображення результату.

Нефункціональні вимоги до програми:

- інтуїтивно зрозумілий та простий графічний інтерфейс користувача;
- швидкий відгук на дії користувача;
- стійкість до помилок та неочікуваних ситуацій;
- докладна програмна документація.

6.3 Характеристики програмного забезпечення

Програмне забезпечення займає 10ГБ вільного місця. В залежності від кількості натренованих нейромережових моделей та навчальних наборів даних обсяг пам'яті може бути змінений. Програмне забезпечення може працювати в основному так і в фоновому режимі.

Структура папок програми наведена на рисунку 6.1.

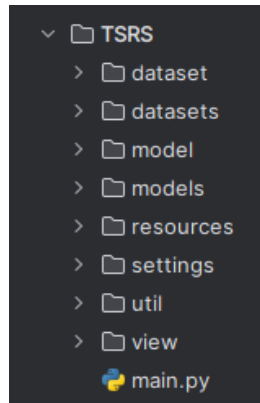


Рисунок 6.1 – Структура програми

Програмне забезпечення складається з таких файлів:

- ClassificationDataset.py – містить клас ClassificationDataset;
- ObjectDetectionDataset.py – містить клас ObjectDetectionDataset;
- ObjectDetectionDatasetCreator.py – містить програмний клас ObjectDetectionDatasetCreator;
- ClassificationModel.py – містить клас ClassificationModel;
- VGG11.py – містить програмний клас VGG11;
- VGG13.py – містить програмний клас VGG13;
- VGG16.py – містить програмний клас VGG16;
- VGG19.py – містить програмний клас VGG19;
- VGGBase.py – містить програмний клас VGGBase;
- ObjectDetectionModel.py – містить клас ObjectDetectionModel;
- utils.py – містить допоміжні методи для класа ObjectDetectionModel;
- YOLOv1.py – містить програмний клас YOLOv1;
- YOLOv1Base.py – містить програмний клас YOLOv1Base;
- YOLOv1LossFunction.py – містить клас YOLOv1LossFunction;
- YOLOv1VGG11.py – містить програмний клас YOLOv1VGG11;
- YOLOv1VGG13.py – містить програмний клас YOLOv1VGG13;
- YOLOv1VGG16.py – містить програмний клас YOLOv1VGG16;

- YOLOv1VGG19.py – містить програмний клас YOLOv1VGG19;
- Settings.py – містить програмний клас Settings;
- Singleton.py – містить програмний клас Singleton;
- StoppableThread.py – містить програмний клас StoppableThread;
- ClassificationWindow.py – містить клас ClassificationWindow;
- ClassificationWindowDesign.py – дизайн вікна класифікації;
- MainWindow.py – містить клас MainWindow;
- MainWindowDesign.py – дизайн головного вікна програми;
- MatplotlibWidget.py – містить програмний клас MatplotlibWidget;
- ModelDetailsWindow.py – містить клас ModelDetailsWindow;
- ModelDetailsWindowDesign.py – дизайн вікна навчання;
- ModelsListWindow.py – містить клас ModelsListWindow;
- ModelsListWindowDesign.py – дизайн вікна зі списком моделей;
- ObjectDetectionWindow.py – містить клас ObjectDetectionWindow;
- ObjectDetectionWindowDesign.py – дизайн вікна розпізнавання;
- main.py – виконуваний файл з функцією main;
- logo.png – файл з логотипом програми;
- placeholder.png – файл з зображенням плейсхолдером;
- font.ttf – файл з шрифтом для відображення тексту на зображенні;
- навчальні набори даних;
- файли зі збереженими вагами моделей.

6.4 Звернення до програмного забезпечення

Для запуску програмного забезпечення необхідно запустити виконуваний файл main.py. Зробити це можна двома способами:

- запустивши файл main.py з системного провідника файлів;
- за допомогою командного рядка.

Після запуску програми подальша взаємодія з програмою відбувається за допомогою графічного інтерфейсу користувача.

6.5 Вхідні та вихідні дані програмного забезпечення

Вхідними даними програмного забезпечення є навчальні набори даних з каталогами що містять папки з зображеннями та анотаціями до них, файли з детальною статистичною інформацією про моделі, файли зі збереженими ваговими параметрами навчених нейромережових моделей, файли ресурсів програми, завантажувані для класифікації чи розпізнавання об'єктів зображення. Також вхідними даними до ПЗ вважаються результати взаємодії користувача з системою, а саме натискання кнопок, гортання списків, вибір об'єктів в комбінованому списку.

Вихідними даними програмного забезпечення є графічне представлення інформації, результати класифікації та розпізнавання об'єктів, оновлені чи доповнені файли з ваговими параметрами нейромережових моделей та інформацією про їх навчання.

6.6 Повідомлення програмісту

Під час роботи програмного забезпечення в консолі виводиться та дублюється інформація про дії, які користувач виконує в системі. Одним із прикладів може слугувати виведення в консоль додаткової інформації під час навчання нейромережових моделей.

Додаткові повідомлення можуть виводитися в консоль за допомогою внутрішньої логіки методів використовуваних бібліотек: PyTorch, Pandas, Pillow, PyQt та Matplotlib.

7 КЕРІВНИЦТВО КОРИСТУВАЧА

7.1 Призначення програмного забезпечення

Програмне забезпечення призначено для класифікації та розпізнавання дорожніх знаків на зображенні за допомогою використання нейромережових моделей VGG та YOLOv1.

Програмне забезпечення надає користувачу можливість:

- вибирати необхідну модель зі списку нейронних мереж;
- навчати нейромережові моделі на навчальному наборі даних;
- переглядати детальну інформацію про нейромережову модель, результати та графіки її навчання;
- вибирати необхідну модель для класифікації або розпізнавання об'єктів на зображенні;
- завантажувати зображення для класифікації та переглядати результат розпізнавання;
- завантажувати зображення для розпізнавання об'єктів та переглядати результат розпізнавання.

7.2 Умови виконання програмного забезпечення

В якості апаратних засобів для використання програмного забезпечення повинен використовуватися комп'ютер з характеристиками не нижче:

- процесор Intel Core i5-7xxx;
- RAM 8ГБ;
- жорсткий диск з обсягом вільного місця 16ГБ;
- маніпулятор типу миша і/або клавіатура.

В якості програмних засобів для використання програмного забезпечення повинні використовуватись:

- ОС Windows 10 чи новіше;

- Python 3.11;
- система керування пакетами pip 23.3.1;
- бібліотека PyTorch 2.1.0;
- бібліотека Pandas 2.1.2;
- Pillow 10.1.0;
- PyQt 6.4.2;
- Matplotlib 3.8.0.

7.3 Виконання програмного забезпечення

Для запуску програмного забезпечення необхідно запустити виконуваний файл main.py за допомогою інтерфейсу системного провідника файлів або командного рядка.

В результаті запуску програми відкриється головне вікно програмного забезпечення з меню програми (рис. 7.1).

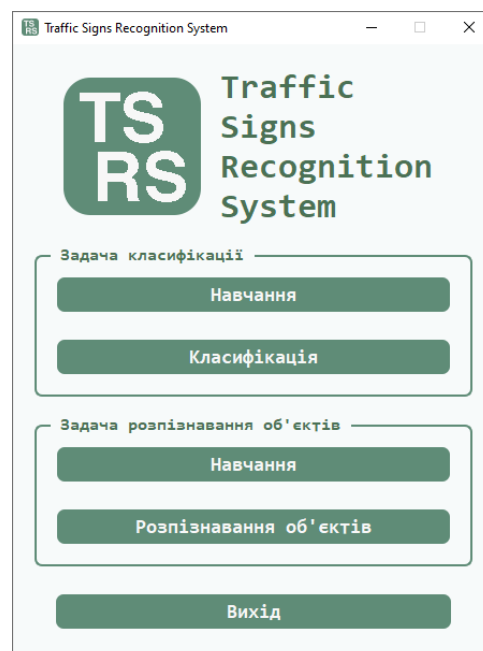


Рисунок 7.1 – Головне вікно з меню програми

На головному вікні програми відображено логотип, назву програми та п'ять основних функціональних кнопок які розділено на декілька груп:

– група задача класифікації містить дві кнопки. Кнопка навчання відповідає за відкриття вікна з переліком моделей для задачі класифікації, а кнопка класифікація – за відкриття вікна класифікації дорожніх знаків;

– група розпізнавання об’єктів містить дві кнопки. Кнопка навчання відповідає за відкриття вікна з переліком моделей для задачі розпізнавання об’єктів, а кнопка розпізнавання об’єктів – за відкриття вікна розпізнавання дорожніх знаків на зображенні;

– група загальних кнопок містить кнопку вихід, яка відповідає за закриття головного меню програми та зупинку роботи програми.

Після натискання на кнопку навчання групи задача класифікації відкриється вікно з переліком нейромережових моделей архітектури VGG (рис. 7.2).

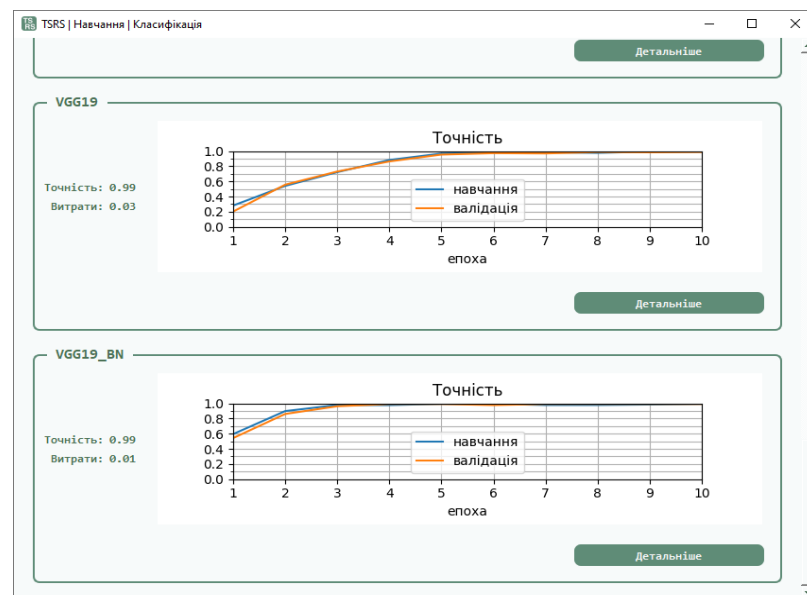


Рисунок 7.2 – Вікно з переліком моделей VGG для задачі класифікації

У вікні відображено перелік всіх моделей VGG для задачі класифікації. Для кожної моделі відображено графік зміни точності під час навчання та валідації даних, точність та витрат поточної версії моделі. Користувач може гортати список за допомогою полоси прокрутки.

Після натискання на кнопку детальніше відкриється вікно навчання відповідної нейромережової моделі для задачі класифікації (рис. 7.3).

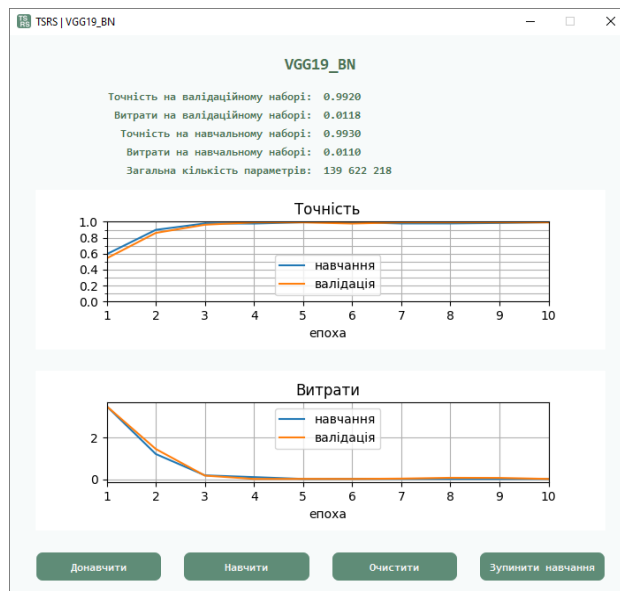


Рисунок 7.3 – Вікно навчання VGG моделі для задачі класифікації

Вікно навчання містить детальну інформацію про значення точності, втрат, кількості параметрів моделі, графіки зміни точності та витрат на навчальному та валідаційному наборі даних під час навчання.

Щоб навчити попередньо не навчену модель необхідно натиснути на кнопку донавчити чи навчити. Під час навчання моделі моруч з її назвою відображається поточний статус процесу. Відповідно до оновлення результатів навчання оновлюються й дані у графічному інтерфейсі.

Для донавчання попередньо навченої моделі необхідно натиснути на кнопку донавчання. Для перенавчання необхідно натиснути кнопку навчити та в діалоговому вікні підтвердити намір перенавчання.

Щоб зупинити процес навчання необхідно натиснути на кнопку зупинити навчання. Поруч з назвою моделі оновиться її статус. Видалити ваги попередньо навченої моделі можна натиснувши на кнопку очистити та підтвердити намір за допомогою діалогового вікна.

Для відкриття вікна класифікації для класифікації дорожніх знаків на зображенні необхідно у головному вікні з меню програми натиснути на кнопку класифікація групи задача класифікації.

Після натискання на кнопку класифікація відкриється вікно класифікації за допомогою навчених нейромережових моделей (рис. 7.4).



Рисунок 7.4 – Вікно класифікації зображень

Для класифікації дорожніх знаків на зображенні необхідно в комбінованому списку вибрати необхідну натреновану неймережеву модель VGG. Далі необхідно натиснути на кнопку завантажити та класифікувати зображення. У вікні системного провідника файлів необхідно вибрати зображення з пам'яті комп'ютера та натиснути кнопку відкрити. Результати класифікації, а саме назву дорожнього знаку буде відображено під зображенням.

Принцип роботи з вікнами перегляду списку та навчання неймережевих моделей для задачі розпізнавання дорожніх знаків за допомогою неймережевих моделей архітектури YOLOv1 відповідає принципу роботи з вікнами перегляду списку та навчання неймережевих моделей для задачі класифікації.

Після натискання на кнопку навчання групи задача розпізнавання об'єктів відкриється вікно з переліком неймережевих моделей архітектури YOLOv1 (рис. 7.5).

Після натискання на кнопку детальніше будь-якої неймережевої моделі з переліку, відкриється вікно навчання відповідної неймережевої моделі для задачі розпізнавання об'єктів (рис. 7.6).

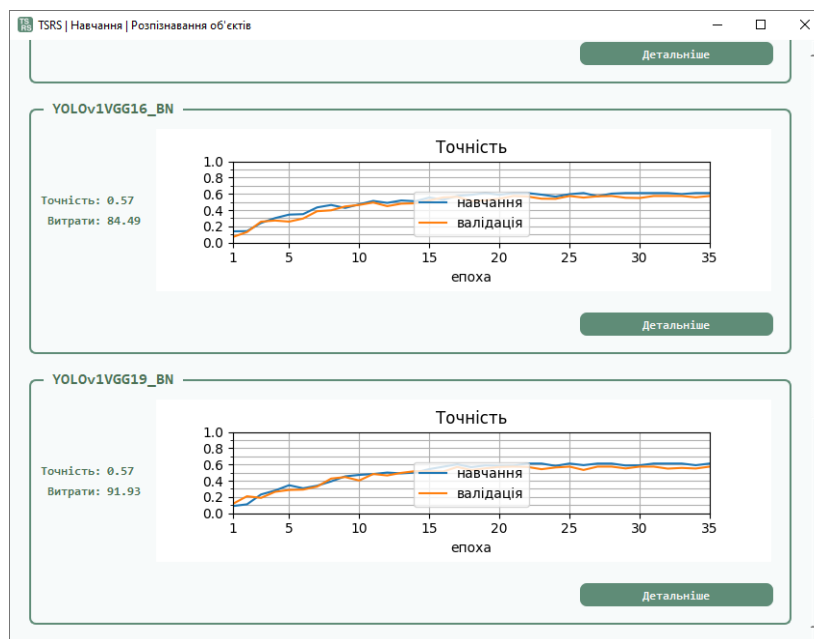


Рисунок 7.5 – Вікно з переліком моделей YOLOv1 для задачі розпізнавання

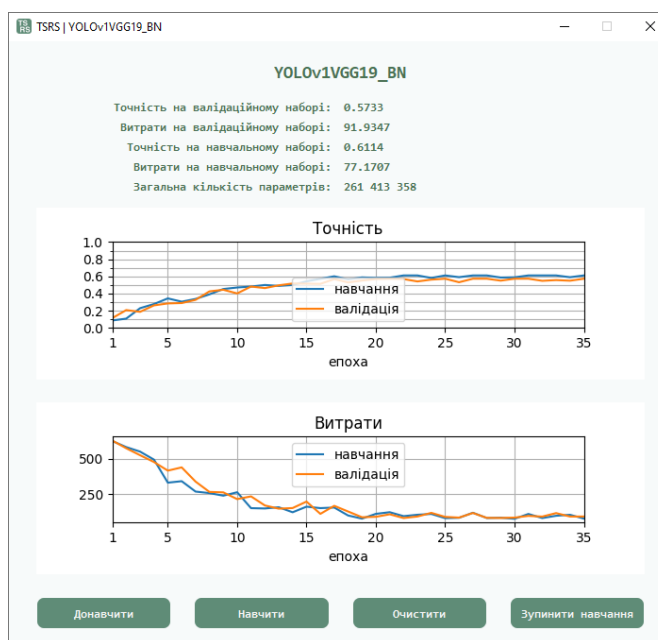


Рисунок 7.6 – Вікно навчання YOLOv1 моделі для задачі розпізнавання

Для відкриття вікна розпізнавання дорожніх знаків на зображенні необхідно у головному вікні з меню програми натиснути на кнопку розпізнавання об'єктів групи задача розпізнавання об'єктів.

Після натискання на кнопку розпізнавання об'єктів відкриється вікно розпізнавання об'єктів за допомогою попередньо навчених нейромережових моделей (рис. 7.7).

Для розпізнавання дорожніх знаків на зображенні необхідно в комбінованому списку вибрати необхідну натреновану неймережеву модель YOLOv1. Далі необхідно натиснути на кнопку завантажити зображення та розпізнати об'єкти. У вікні системного провідника файлів необхідно вибрати зображення з пам'яті комп'ютера та натиснути кнопку відкрити. Результати розпізнавання об'єктів, а саме набір обмежувальних рамок та мітка класу будуть відображені поверх завантаженого зображення.



Рисунок 7.7 – Вікно розпізнавання об'єктів на зображенні

Для закриття програми необхідно натиснути на кнопку вихід в головному вікні з меню програми або натиснути на значок хрестика на вікні програми.

7.4 Повідомлення користувачу

Для допомоги користувачу та запобіганню не бажаних дій, під час виконання деяких операцій передбачено діалогові вікна з попередженнями.

При натисканні на кнопку навчити попередньо натренованої моделі відображається діалогове вікно підтвердження наміру очищення ваг та перенавчання моделі (рис. 7.8).

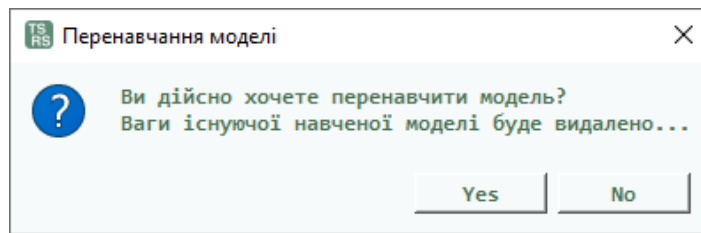


Рисунок 7.8 – Вікно підтвердження наміру перенавчання моделі

При натисканні на кнопку очистити відображається діалогове вікно підтвердження наміру очищення ваг натренованої моделі (рис. 7.9).

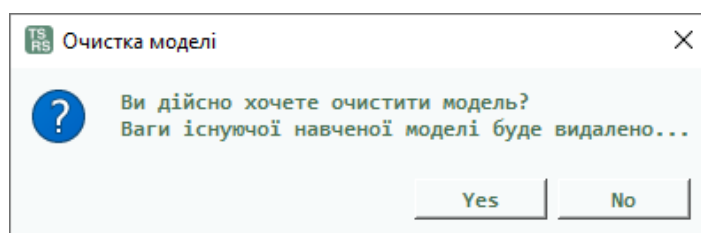


Рисунок 7.9 – Вікно підтвердження наміру очищення ваг моделі

При натисканні на кнопку вийти відображається діалогове вікно підтвердження наміру закриття програми (рис. 7.10).

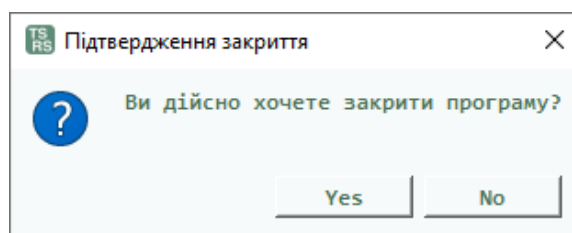


Рисунок 7.10 – Вікно підтвердження наміру закриття програми

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи магістра вирішено задачу класифікації та розпізнавання об'єктів на зображенні за допомогою згорткових нейронних мереж.

Проведено детальний аналіз предметної області та досліджено архітектури та основні характеристики згорткових нейромережевих моделей VGG для вирішення задачі класифікації та YOLOv1 для вирішення задачі розпізнавання об'єктів. Відповідно до проведеного аналізу запропоновано підходи до вдосконалення існуючих архітектр розглянутих нейромережевих моделей.

Відповідно до поставленої мети роботи виконано докладний аналіз функціональних та нефункціональних вимог до програмного забезпечення. В якості інструментальних засобів розробки програми вибрано інтерпретовану високорівневу мову програмування Python, програмні пакети PyTorch та PyQt, інтегроване середовище розробки PyCharm.

Відповідно до висунутих вимог реалізовано основну структуру спроектованого програмного забезпечення, розроблено всі необхідні для роботи програмні класи та виконувані файли. Реалізовано базову та вдосконалену архітектури нейромережевих моделей VGG для класифікації зображень та YOLOv1 для розпізнавання об'єктів на зображенні. Обрано та проаналізовано навчальні набори даних для вирішення прикладної задачі класифікації та розпізнавання дорожніх знаків на зображенні. Виконано тестування на визначеному контрольному списку та наборі тест-кейсів.

Проведено експериментальне дослідження розробленого програмного забезпечення. Всі реалізовані нейромережеві моделі архітектури VGG досягли точності класифікації зображень більше ніж 93% на валідаційному наборі даних. Моделі з вдосконаленою архітектурою VGG показали менший час навчання та кращу узагальнюючу спроможність. Всі нейромережеві моделі архітектури YOLOv1 досягли відносно хорошій середньої точності

розпізнавання об'єктів на зображенні на валідаційному наборі даних, при цьому моделі з вдосконаленою архітектурою YOLOv1 перевершили моделі з базовою архітектурою за значенням точності на 10%, при цьому збільшивши час навчання в два рази.

Відповідно до отриманих результатів експериментального дослідження програмного забезпечення можна сказати, що доцільність використання запропонованих підходів до вдосконалення існуючих архітектр нейромережевих моделей VGG для вирішення задачі класифікації та YOLOv1 для вирішення задачі розпізнавання об'єктів доведено на практиці та підтверджено отриманими результатами.

Практична цінність спроектованого та розробленого програмного забезпечення полягає в тому, що воно надає можливості навчання та використання нейромережевих моделей з базовою та вдосконаленою архітектурою VGG та YOLOv1 для вирішення прикладних задач класифікації та розпізнавання об'єктів на зображенні на довільному наборі даних. Розроблене програмне забезпечення може використовуватися як частина програмної системи автомобіля для виявлення та розпізнавання дорожніх знаків на зображенні отриманому з вбудованої в автомобіль камери.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Volkswagen CC | Volkswagen в Україні [Електрон. ресурс]. – Режим доступу: <https://www.volkswagen.ua/servis-ta-aksesuari/poperedni-modeli/volkswagen-cc>.
2. EQC | Mercedes-Benz | Офіційний сайт в Україні [Електрон. ресурс]. – Режим доступу: <https://www.mercedes-benz.ua/models/eqc>.
3. Toyota RAV4 (Тойота Рав 4) [Електрон. ресурс]. – Режим доступу: <https://toyota.com.ua/rav4>.
4. Статистика продажів автомобілів у жовтні 2023 [Електрон. ресурс]. – Режим доступу: <https://ukrautoprom.com.ua/statystyka-prodazhiv-avtomobiliv-u-zhovtni-2023>.
5. Computer vision [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Computer_vision.
6. Going deep into object detection [Electronic resource]. – Access mode: <https://towardsdatascience.com/going-deep-into-object-detection-bed442d92b34>.
7. Localization and Object Detection with Deep Learning [Electronic resource]. – Access mode: <https://towardsdatascience.com/localization-and-object-detection-with-deep-learning-67b5aca67f22>.
8. Object detection [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Object_detection.
9. Object Detection vs Object Recognition vs Image Segmentation [Electronic resource]. – Access mode: <https://www.i2tutorials.com/object-detection-vs-object-recognition-vs-image-segmentation>.
10. Scale-invariant feature transform [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform.
11. Histogram of oriented gradients [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.
12. Support vector machine [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Support_vector_machine.

13. What is a Neural Network? [Electronic resource]. – Access mode: <https://aws.amazon.com/what-is/neural-network>.
14. Субботін, С.О. Нейронні мережі: теорія та практика: навч. посіб. / С.О. Субботін. – Житомир : Вид. О.О. Євєнюк, 2020. – 184 с.
15. Artificial neuron [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Artificial_neuron.
16. The Essential Guide to Neural Network Architectures [Electronic resource]. – Access mode: <https://www.v7labs.com/blog/neural-network-architectures-guide>.
17. Forward and Backward Propagation — Understanding it to master the model training process [Electronic resource]. – Access mode: <https://medium.com/geekculture/forward-and-backward-propagation-understanding-it-to-master-the-model-training-process-3819727dc5c1>.
18. Feed Forward Neural Network [Electronic resource]. – Access mode: <https://deeptai.org/machine-learning-glossary-and-terms/feed-forward-neural-network>.
19. Convolutional neural network [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Convolutional_neural_network.
20. What are convolutional neural networks? [Electronic resource]. – Access mode: <https://www.ibm.com/topics/convolutional-neural-networks>.
21. Convolutional Neural Networks, Explained [Electronic resource]. – Access mode: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>.
22. CNN | Introduction to Pooling Layer [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer>.
23. The Flattening and Full Connection Steps of Convolutional Neural Networks [Electronic resource]. – Access mode: <https://www.nickmccullum.com/python-deep-learning/flattening-full-connection>.
24. Activation Functions in Neural Networks [12 Types & Use Cases] [Electronic resource]. – Access mode: <https://www.v7labs.com/blog/neural->

networks-activation-functions.

25. Softmax Activation Function — How It Actually Works [Electronic resource]. – Access mode: <https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78>.

26. Cross-Entropy Loss Function [Electronic resource]. – Access mode: <https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e>.

27. Multi-class cross entropy loss [Electronic resource]. – Access mode: <https://www.oreilly.com/library/view/hands-on-convolutional-neural/9781789130331/7f34b72e-f571-49d2-a37a-4ed6f8011c93.xhtml>.

28. Gradient-based learning applied to document recognition [Electronic resource]. – Access mode: <https://ieeexplore.ieee.org/document/726791>.

29. ImageNet classification with deep convolutional neural networks [Electronic resource]. – Access mode: <https://dl.acm.org/doi/10.1145/3065386>.

30. Visualizing and Understanding Convolutional Networks [Electronic resource]. – Access mode: <https://arxiv.org/abs/1311.2901>.

31. Very Deep Convolutional Networks for Large-Scale Image Recognition [Electronic resource]. – Access mode: <https://arxiv.org/abs/1409.1556>.

32. The Ultimate Guide to Object Detection [Electronic resource]. – Access mode: <https://www.v7labs.com/blog/object-detection-guide>.

33. Rich feature hierarchies for accurate object detection and semantic segmentation [Electronic resource]. – Access mode: <https://arxiv.org/abs/1311.2524>.

34. Selective Search for Object Detection | R-CNN [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cnn>.

35. Fast R-CNN [Electronic resource]. – Access mode: <https://arxiv.org/abs/1504.08083>.

36. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Electronic resource]. – Access mode:

<https://arxiv.org/abs/1506.01497>.

37. You Only Look Once: Unified, Real-Time Object Detection [Electronic resource]. – Access mode: <https://arxiv.org/abs/1506.02640>.

38. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [Electronic resource]. – Access mode: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.

39. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Electronic resource]. – Access mode: <https://arxiv.org/abs/1502.03167>.

40. Batch Norm Explained Visually — How it works, and why neural networks need it [Electronic resource]. – Access mode: <https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739>.

41. Internals Of YOLOv1 For Object Detection [Electronic resource]. – Access mode: <https://niteshkumardew11.medium.com/internals-of-yolov1-for-object-detection-52c3e6ea523e>.

42. C_2.03 YOLO Loss for training [Electronic resource]. – Access mode: <https://wikidocs.net/167690>.

43. Non Max Suppression (NMS) [Electronic resource]. – Access mode: <https://medium.com/analytics-vidhya/non-max-suppression-nms-6623e6572536>.

44. Use case diagram [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Use_case_diagram.

45. Current Status : Standard C++ [Electronic resource]. – Access mode: <https://isocpp.org/std/status>.

46. C# docs – get started [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/dotnet/csharp>.

47. ECMA-262 – ECMA International [Electronic resource]. – Access mode: <https://ecma-international.org/publications-and-standards/standards/ecma-262>.

48. Python 3.12.1 documentation [Electronic resource]. – Access mode: <https://docs.python.org/3>.

49. Why TensorFlow [Electronic resource]. – Access mode: <https://www.tensorflow.org/about>.

50. Features | PyTorch [Electronic resource]. – Access mode: <https://pytorch.org/features>.

51. German Traffic Sign Benchmarks [Electronic resource]. – Access mode: https://benchmark.ini.rub.de/gtsrb_news.html.

52. Road Sign Detection [Electronic resource]. – Access mode: <https://www.kaggle.com/datasets/andrewmvd/road-sign-detection/data>.

53. Object Detection Datasets Overview [Electronic resource]. – Access mode: <https://docs.ultralytics.com/datasets/detect>.

ДОДАТОК А
Текст програми

A.1 Файл main.py

```
import sys

from PyQt6.QtWidgets import QApplication

from view.MainWindow import MainWindow

def main():
    application = QApplication(sys.argv)
    main_window = MainWindow()
    main_window.show()
    sys.exit(application.exec())

if __name__ == '__main__':
    main()
```

A.2 Файл ClassificationDataset.py

```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

class ClassificationDataset:
    ROAD_SIGNS_TRANSLATION = {"Bumpy road": "Вибоїна",
                              "End of all speed and passing limits": "Кінець заборон",
                              "No entry": "В'їзд заборонено",
                              "No vechiles": "Рух заборонено",
                              "Pedestrians": "Пішохідний перехід",
                              "Priority road": "Головна дорога",
                              "Roundabout mandatory": "Круговий рух",
                              "Speed limit": "Обмеження швидкості",
                              "Stop": "Стоп",
                              "Yield": "Дати дорогу"}

    def __init__(self, path):
        image_transforms = ClassificationDataset.get_default_image_inference_transforms()

        self.__train_dataset = datasets.ImageFolder(root=path+"train",
            transform=image_transforms)
        self.__validation_dataset = datasets.ImageFolder(root=path+"valid",
            transform=image_transforms)
        self.__test_dataset = datasets.ImageFolder(root=path+"test",
            transform=image_transforms)

    def get_train_loader(self, batch_size=32, num_workers=4, pin_memory=True,
        shuffle=True, drop_last=True):
        train_loader = DataLoader(
            dataset=self.__train_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
```

```

        shuffle=shuffle,
        drop_last=drop_last
    )

    return train_loader

    def get_validation_loader(self, batch_size=32, num_workers=4, pin_memory=True,
shuffle=True, drop_last=True):
        validation_loader = DataLoader(
            dataset=self.__validation_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
            shuffle=shuffle,
            drop_last=drop_last
        )

        return validation_loader

    def get_test_loader(self, batch_size=32, num_workers=4, pin_memory=True,
shuffle=True, drop_last=True):
        test_loader = DataLoader(
            dataset=self.__test_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
            shuffle=shuffle,
            drop_last=drop_last
        )

        return test_loader

    def get_num_of_classes(self):
        return len(self.__train_dataset.classes)

    def get_classes(self):
        return {value: key for key, value in
self.__train_dataset.class_to_idx.items()}

    @staticmethod
    def get_default_image_inference_transforms():
        resize_size = (256, 256)
        crop_size = (224, 224)
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]

        image_transforms = []

        image_transforms += [transforms.Resize(resize_size,
interpolation=transforms.InterpolationMode.BILINEAR)]
        image_transforms += [transforms.CenterCrop(crop_size)]
        image_transforms += [transforms.ToTensor()]
        image_transforms += [transforms.Normalize(mean, std)]

        return transforms.Compose(image_transforms)

```

A.3 Файл ObjectDetectionDatasetCreator .py

```

import os
import pandas as pd
import torch
from PIL import Image

class ObjectDetectionDatasetCreator(torch.utils.data.Dataset):
    def __init__(self, path, data_link_file, num_of_classes, transform=None):
        self.data_link = pd.read_csv(path + data_link_file)
        self.images_dir = path + "images"
        self.labels_dir = path + "labels"
        self.transform = transform

        self.c = num_of_classes
        self.s = 7
        self.b = 2

    def __len__(self):
        return len(self.data_link)

    def __getitem__(self, index):
        image_path = os.path.join(self.images_dir, self.data_link.iloc[index, 0])
        image = Image.open(image_path).convert("RGB")
        if self.transform is not None:
            image = self.transform(image)

        label_path = os.path.join(self.labels_dir, self.data_link.iloc[index, 1])

        target_boxes = []
        with (open(label_path) as f):
            for labels in f.readlines():

                labels_array = labels.replace("\n", "").split()
                cls = int(labels_array[0])
                cx, cy, w, h = [float(value) for value in labels_array[1:]]
                target_boxes.append([cls, cx, cy, w, h])

        targets = torch.zeros((self.s, self.s, self.b * 5 + self.c))
        for target_box in target_boxes:
            cls, cx, cy, w, h = target_box
            col, row = int(self.s * cx), int(self.s * cy)
            cx, cy = self.s * cx - col, self.s * cy - row
            w, h = self.s * w, self.s * h

            if targets[row, col, 4] == 0:
                targets[row, col, 4] = 1
                bounding_box = torch.tensor([cx, cy, w, h])
                targets[row, col, 0:4] = bounding_box
                targets[row, col, self.b * 5 + cls] = 1

        return image, targets

```

A.4 Файл ObjectDetectionDataset.py

```

import json

from torch.utils.data import DataLoader
from torchvision import transforms

from dataset.ObjectDetectionDatasetCreator import ObjectDetectionDatasetCreator
class ObjectDetectionDataset:
    ROAD_SIGNS_TRANSLATION = {"Traffic Light": "Світлофор",
                              "Speed limit": "Обмеження швидкості",
                              "Crosswalk": "Пішохідний перехід",
                              "Stop": "Стоп"}

    def __init__(self, path):
        image_transforms =
ObjectDetectionDataset.get_default_image_inference_transforms()

        with open(path + "annotations.json") as json_file:
            annotations = json.load(json_file)
            self.classes = {int(key): value for key, value in
annotations["classes"].items()}

            self.__train_dataset = ObjectDetectionDatasetCreator(path, "train.csv",
self.get_num_of_classes(),
                                                                image_transforms)
            self.__validation_dataset = ObjectDetectionDatasetCreator(path, "valid.csv",
self.get_num_of_classes(),
                                                                image_transforms)
            self.__test_dataset = ObjectDetectionDatasetCreator(path, "test.csv",
self.get_num_of_classes(),
                                                                image_transforms)

    def get_train_loader(self, batch_size=32, num_workers=4, pin_memory=True,
shuffle=True, drop_last=True):
        train_loader = DataLoader(
            dataset=self.__train_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
            shuffle=shuffle,
            drop_last=drop_last
        )

        return train_loader

    def get_validation_loader(self, batch_size=32, num_workers=4, pin_memory=True,
shuffle=True, drop_last=True):
        validation_loader = DataLoader(
            dataset=self.__validation_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
            shuffle=shuffle,
            drop_last=drop_last
        )

        return validation_loader

```

```

    def get_test_loader(self, batch_size=32, num_workers=4, pin_memory=True,
shuffle=True, drop_last=True):
        test_loader = DataLoader(
            dataset=self.__test_dataset,
            batch_size=batch_size,
            num_workers=num_workers,
            pin_memory=pin_memory,
            shuffle=shuffle,
            drop_last=drop_last
        )

        return test_loader

    def get_num_of_classes(self):
        return len(self.classes)

    def get_classes(self):
        return self.classes

    @staticmethod
    def get_default_image_inference_transforms():
        resize_size = (448, 448)
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]

        image_transforms = []

        image_transforms += [transforms.Resize(resize_size,
interpolation=transforms.InterpolationMode.BILINEAR)]
        image_transforms += [transforms.ToTensor()]
        image_transforms += [transforms.Normalize(mean, std)]

        return transforms.Compose(image_transforms)

```

A.5 Файл VGGBase.py

```

import torch
import torch.nn as nn

class VGGBase(nn.Module):
    dropout_probability = 0.5

    def __init__(self, features, num_of_classes, dropout):
        super().__init__()

        self.features = features
        self.adaptive_avg_pool = nn.AdaptiveAvgPool2d(output_size=(7, 7))
        self.classifier = VGGBase.make_classifier(num_of_classes, dropout)

    def forward(self, input_tensor):
        output_tensor = self.features(input_tensor)
        output_tensor = self.adaptive_avg_pool(output_tensor)
        output_tensor = torch.flatten(output_tensor, 1)
        output_tensor = self.classifier(output_tensor)

        return output_tensor

    def freeze_features_weights(self):

```

```

        for param in self.features:
            param.requires_grad = False

    @staticmethod
    def make_convolutional_layer(in_channels, out_channels,
                                batch_normalization=False):
        layers = []

        layers += [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)]
        if batch_normalization:
            layers += [nn.BatchNorm2d(out_channels)]
        layers += [nn.ReLU(inplace=True)]

        return layers

    @staticmethod
    def make_classifier(num_of_classes, dropout=False):
        layers = []

        layers += [nn.Linear(512 * 7 * 7, 4096)]
        layers += [nn.ReLU(inplace=True)]
        if dropout:
            layers += [nn.Dropout(p=VGGBase.dropout_probability)]
        layers += [nn.Linear(4096, 4096)]
        layers += [nn.ReLU(inplace=True)]
        if dropout:
            layers += [nn.Dropout(p=VGGBase.dropout_probability)]
        layers += [nn.Linear(4096, num_of_classes)]

        return nn.Sequential(*layers)

```

A.6 Файл VGG16.py

```

import torch.nn as nn
import torchvision.models as torch_models

from model.classification.VGGBase import VGGBase

class VGG16(VGGBase):
    def __init__(self, num_of_classes, pretrained_features=False,
                 batch_normalization=False, dropout=False):
        if pretrained_features:
            features = VGG16.get_pretrained_features(batch_normalization)
        else:
            features = VGG16.make_features(batch_normalization)

        super().__init__(features, num_of_classes, dropout)

    @staticmethod
    def get_pretrained_features(batch_normalization=False):
        if batch_normalization:
            features =
torch_models.vgg16_bn(weights=torch_models.VGG16_BN_Weights.IMAGENET1K_V1).features
        else:
            features =
torch_models.vgg16(weights=torch_models.VGG16_Weights.IMAGENET1K_V1).features

        return features

```

```

@staticmethod
def make_features(batch_normalization=False):
    layers = []

    layers += VGGBase.make_convolutional_layer(3, 64, batch_normalization)
    layers += VGGBase.make_convolutional_layer(64, 64, batch_normalization)
    layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

    layers += VGGBase.make_convolutional_layer(64, 128, batch_normalization)
    layers += VGGBase.make_convolutional_layer(128, 128, batch_normalization)
    layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

    layers += VGGBase.make_convolutional_layer(128, 256, batch_normalization)
    layers += VGGBase.make_convolutional_layer(256, 256, batch_normalization)
    layers += VGGBase.make_convolutional_layer(256, 256, batch_normalization)
    layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

    layers += VGGBase.make_convolutional_layer(256, 512, batch_normalization)
    layers += VGGBase.make_convolutional_layer(512, 512, batch_normalization)
    layers += VGGBase.make_convolutional_layer(512, 512, batch_normalization)
    layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

    layers += VGGBase.make_convolutional_layer(512, 512, batch_normalization)
    layers += VGGBase.make_convolutional_layer(512, 512, batch_normalization)
    layers += VGGBase.make_convolutional_layer(512, 512, batch_normalization)
    layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

    return nn.Sequential(*layers)

```

A.7 Файл ClassificationModel.py

```

import json
import os
import shutil
from datetime import datetime

import torch
import torch.nn as nn
import torch.optim as optim
from PIL import Image

from dataset.ClassificationDataset import ClassificationDataset
from model.classification.VGG11 import VGG11
from model.classification.VGG13 import VGG13
from model.classification.VGG16 import VGG16
from model.classification.VGG19 import VGG19

class ClassificationModel:
    weights_file_name = "weights.pth"
    details_file_name = "details.json"
    optimizer_weights_file_name = "optimizer_weights.pth"

    learning_rate = 1e-4
    epochs_count = 20

    def __init__(self, name, path):
    def is_saved_model(self):
    def clear(self):

```

```

def clear_model(self):
def clear_model_details(self):
def delete(self):
def load_model_details(self):
def load_model_weights(self):
def load_optimizer_weights(self):
def load_model_and_optimizer_weights(self):
def init_new_model_and_optimizer(self, num_of_classes):
def prepare_model_for_training(self, dataset, freeze_features=True):
def process_model_after_training(self):
    @staticmethod
    def get_device():

def train(self, dataset):
    device = ClassificationModel.get_device()

    train_loader = dataset.get_train_loader()
    validation_loader = dataset.get_validation_loader()

    train_batch_count = len(train_loader)
    validation_batch_count = len(validation_loader)

    train_items_count = len(train_loader.dataset)
    if train_loader.drop_last:
        train_items_count = train_loader.batch_size * train_batch_count

    validation_items_count = len(validation_loader.dataset)
    if validation_loader.drop_last:
        validation_items_count = validation_loader.batch_size *
validation_batch_count

    criterion = ClassificationModel.get_default_criterion()

    print(f"Start training {self.name} model")
    for epoch in range(ClassificationModel.epochs_count):
        print(f"Start epoch {epoch + 1}/{ClassificationModel.epochs_count} at
{datetime.now()}")

        self.model.train()

        train_loss = 0
        train_accuracy = 0

        for batch, (inputs, labels) in enumerate(train_loader):
            print(f"Start batch {batch + 1}/{train_batch_count}")

            inputs = inputs.to(device)
            labels = labels.to(device)

            self.optimizer.zero_grad()

            outputs = self.model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            self.optimizer.step()

            train_loss += loss.item()
            train_accuracy += torch.sum(outputs.argmax(1) == labels).item()

        train_loss /= train_batch_count
        train_accuracy /= train_items_count

```

```

        print(f"End epoch {epoch + 1}/{ClassificationModel.epochs_count} at
{datetime.now()})."
        f" Training loss: {train_loss} Training accuracy:
{train_accuracy}")

        print(f"Start validation at {datetime.now()}")

        self.model.eval()

        validation_loss = 0
        validation_accuracy = 0

        with torch.no_grad():
            for batch, (inputs, labels) in enumerate(validation_loader):
                print(f"Start batch {batch + 1}/{validation_batch_count}")

                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = self.model(inputs)
                validation_loss += criterion(outputs, labels).item()
                validation_accuracy += torch.sum(outputs.argmax(1) ==
labels).item()

            validation_loss /= validation_batch_count
            validation_accuracy /= validation_items_count

        print(f"End validation at {datetime.now()})."
        f" Validation loss: {validation_loss} Validation accuracy:
{validation_accuracy}")

        print("Update model details")
        self.train_history["train_loss"].append(train_loss)
        self.train_history["train_accuracy"].append(train_accuracy)
        self.train_history["validation_loss"].append(validation_loss)
        self.train_history["validation_accuracy"].append(validation_accuracy)
        self.test_results = {"test_loss": validation_loss, "test_accuracy":
validation_accuracy}

        print("Save model to disk")
        self.save()

        self.last_change_time = datetime.now()

        print(f"End training {self.name} model")

    def prepare_model_for_predicting(self):
    def process_model_after_predicting(self):

    def predict(self, img_path):
        image = Image.open(img_path).convert("RGB")
        transform = ClassificationDataset.get_default_image_inference_transforms()
        transformed_image = transform(image)

        self.model.eval()
        with torch.no_grad():
            inputs = transformed_image.to(ClassificationModel.get_device())
            inputs = inputs.unsqueeze(0)
            outputs = self.model(inputs)

            predicted_class = torch.argmax(torch.softmax(outputs, dim=1),
dim=1).item()

```

```

        prediction = self.classes[predicted_class]

        if prediction in ClassificationDataset.ROAD_SIGNS_TRANSLATION:
            return ClassificationDataset.ROAD_SIGNS_TRANSLATION[prediction]
        else:
            return prediction

    def save(self):

    @staticmethod
    def get_default_optimizer(model):
        return optim.Adam(model.parameters(), lr=ClassificationModel.learning_rate)

    @staticmethod
    def get_default_criterion():
        return nn.CrossEntropyLoss()

    @staticmethod
    def get_model_by_name(name, num_of_classes, pretrained_features):
        if name == "VGG11":
            return VGG11(num_of_classes, pretrained_features, dropout=True)
        elif name == "VGG11_BN":
            return VGG11(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
        if name == "VGG13":
            return VGG13(num_of_classes, pretrained_features, dropout=True)
        elif name == "VGG13_BN":
            return VGG13(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
        if name == "VGG16":
            return VGG16(num_of_classes, pretrained_features, dropout=True)
        elif name == "VGG16_BN":
            return VGG16(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
        if name == "VGG19":
            return VGG19(num_of_classes, pretrained_features, dropout=True)
        elif name == "VGG19_BN":
            return VGG19(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)

```

A.8 Файл YOLOv1LossFunction.py

```

import torch
import torch.nn as nn

import model.object_detection.utils as utils

class YOLOv1LossFunction(nn.Module):
    coordinates_penalty = 5
    noobject_penalty = 0.5
    eps = 1e-7

    def __init__(self, num_of_classes):
        super().__init__()

        self.c = num_of_classes
        self.s = 7
        self.b = 2

```

```

self.mse_loss_function = nn.MSELoss(reduction='sum')

def forward(self, outputs, targets):
    outputs = outputs.reshape(-1, self.s, self.s, self.b * 5 + self.c)

    ious = [
        utils.intersection_over_union(outputs[..., 0:4], targets[..., 0:4],
        eps=YOLOv1LossFunction.eps),
        utils.intersection_over_union(outputs[..., 5:9], targets[..., 0:4],
        eps=YOLOv1LossFunction.eps)
    ]
    concatenated_ious = torch.cat([tensor.unsqueeze(0) for tensor in ious],
    dim=0)
    max_values, max_indices = torch.max(concatenated_ious, dim=0)
    box_exists = targets[..., 4].unsqueeze(3)

    outputs_responsible_box = max_indices * outputs[..., 5:10] + (1 -
    max_indices) * outputs[..., 0:5]
    outputs_box = box_exists * outputs_responsible_box
    targets_box = box_exists * targets[..., 0:5]

    outputs_box[..., 2:4] = torch.sign(outputs_box[..., 2:4]) *
    torch.sqrt(torch.abs(outputs_box[..., 2:4] + YOLOv1LossFunction.eps))
    targets_box[..., 2:4] = torch.sqrt(torch.abs(targets_box[..., 2:4]))

    coordinates_loss = self.mse_loss_function(outputs_box[..., 0:4],
    targets_box[..., 0:4])

    object_loss = self.mse_loss_function(outputs_box[..., 4:5], targets_box[...
    4:5])

    noobject_outputs_box_1 = (1 - box_exists) * outputs[..., 4:5]
    noobject_outputs_box_2 = (1 - box_exists) * outputs[..., 9:10]
    noobject_targets = targets[..., 4:5]

    noobject_loss = self.mse_loss_function(noobject_outputs_box_1,
    noobject_targets)
    noobject_loss += self.mse_loss_function(noobject_outputs_box_2,
    noobject_targets)

    outputs_class = box_exists * outputs[..., 10:]
    targets_class = box_exists * targets[..., 10:]
    class_loss = self.mse_loss_function(outputs_class, targets_class)

    loss = YOLOv1LossFunction.coordinates_penalty * coordinates_loss
    loss += object_loss
    loss += YOLOv1LossFunction.noobject_penalty * noobject_loss
    loss += class_loss

    return loss

```

A.9 Файл YOLOv1Base.py

```

import torch
import torch.nn as nn

class YOLOv1Base(nn.Module):
    dropout_probability = 0.5

```

```

def __init__(self, features, features_out_channels, num_of_classes,
batch_normalization, dropout):
    super().__init__()

    self.c = num_of_classes
    self.s = 7
    self.b = 2
    self.features = features
    self.features_neck = YOLOv1Base.make_features_neck(features_out_channels,
batch_normalization)
    self.adaptive_avg_pool = nn.AdaptiveAvgPool2d(output_size=(7, 7))
    self.detector = YOLOv1Base.make_detector(self.c, self.s, self.b, dropout)

def forward(self, input_tensor):
    output_tensor = self.features(input_tensor)
    output_tensor = self.features_neck(output_tensor)
    output_tensor = self.adaptive_avg_pool(output_tensor)
    output_tensor = torch.flatten(output_tensor, 1)
    output_tensor = self.detector(output_tensor)

    return output_tensor

def freeze_features_weights(self):
    for param in self.features:
        param.requires_grad = False

    @staticmethod
    def make_convolutional_layer(in_channels, out_channels,
batch_normalization=False, kernel_size=3, stride=1, padding=1):
        layers = []

        layers += [nn.Conv2d(in_channels, out_channels, kernel_size=kernel_size,
stride=stride, padding=padding)]

        if batch_normalization:
            layers += [nn.BatchNorm2d(out_channels)]

        layers += [nn.LeakyReLU(negative_slope=1e-1, inplace=True)]

        return layers

    @staticmethod
    def make_features_neck(in_channels=1024, batch_normalization=False):
        layers = []

        layers += YOLOv1Base.make_convolutional_layer(in_channels, 1024,
batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(1024, 1024,
batch_normalization, stride=2)
        layers += YOLOv1Base.make_convolutional_layer(1024, 1024,
batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(1024, 1024,
batch_normalization)

        return nn.Sequential(*layers)

    @staticmethod
    def make_detector(c, s=7, b=2, dropout=False):
        layers = []

        layers += [nn.Linear(1024 * 7 * 7, 4096)]
        layers += [nn.LeakyReLU(negative_slope=1, inplace=True)]

```

```

if dropout:
    layers += [nn.Dropout(p=YOLOv1Base.dropout_probability)]
layers += [nn.Linear(4096, s * s * (b * 5 + c))]

return nn.Sequential(*layers)

```

A.10 Файл YOLOv1.py

```

import torch.nn as nn

from model.object_detection.YOLOv1Base import YOLOv1Base

class YOLOv1(YOLOv1Base):
    def __init__(self, num_of_classes, pretrained_features=False,
batch_normalization=False, dropout=False):
        features, features_out_channels = YOLOv1.make_features(batch_normalization)

        super().__init__(features, features_out_channels, num_of_classes,
batch_normalization, dropout)

    @staticmethod
    def make_features(batch_normalization=False):
        layers = []

        layers += YOLOv1Base.make_convolutional_layer(3, 64,
batch_normalization, kernel_size=7, stride=2, padding=3)
        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

        layers += YOLOv1Base.make_convolutional_layer(64, 192, batch_normalization)
        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

        layers += YOLOv1Base.make_convolutional_layer(192, 128, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(128, 256, batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(256, 256, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(256, 512, batch_normalization)
        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

        layers += YOLOv1Base.make_convolutional_layer(512, 256, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(256, 512, batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(512, 256, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(256, 512, batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(512, 256, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(256, 512, batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(512, 256, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(256, 512, batch_normalization)

        layers += YOLOv1Base.make_convolutional_layer(512, 512, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(512, 1024, batch_normalization)
        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]

```

```

        layers += YOLOv1Base.make_convolutional_layer(1024, 512, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(512, 1024, batch_normalization)
        layers += YOLOv1Base.make_convolutional_layer(1024, 512, batch_normalization,
kernel_size=1, padding=0)
        layers += YOLOv1Base.make_convolutional_layer(512, 1024, batch_normalization)

    return nn.Sequential(*layers), 1024

```

A.11 Файл YOLOv1VGG11.py

```

from model.classification.VGG11 import VGG11
from model.object_detection.YOLOv1Base import YOLOv1Base

class YOLOv1VGG11(YOLOv1Base):
    def __init__(self, num_of_classes, pretrained_features=False,
batch_normalization=False, dropout=False):
        if pretrained_features:
            features, features_out_channels =
VGG11.get_pretrained_features(batch_normalization), 512
        else:
            features, features_out_channels =
VGG11.make_features(batch_normalization), 512

        super().__init__(features, features_out_channels, num_of_classes,
batch_normalization, dropout)

```

A.12 Файл YOLOv1VGG13.py

```

from model.classification.VGG13 import VGG13
from model.object_detection.YOLOv1Base import YOLOv1Base

class YOLOv1VGG13(YOLOv1Base):
    def __init__(self, num_of_classes, pretrained_features=False,
batch_normalization=False, dropout=False):
        if pretrained_features:
            features, features_out_channels =
VGG13.get_pretrained_features(batch_normalization), 512
        else:
            features, features_out_channels =
VGG13.make_features(batch_normalization), 512

        super().__init__(features, features_out_channels, num_of_classes,
batch_normalization, dropout)

```

A.13 Файл YOLOv1VGG16.py

```

from model.classification.VGG16 import VGG16
from model.object_detection.YOLOv1Base import YOLOv1Base

```

```

class YOLOv1VGG16(YOLOv1Base):
    def __init__(self, num_of_classes, pretrained_features=False,
batch_normalization=False, dropout=False):
        if pretrained_features:
            features, features_out_channels =
VGG16.get_pretrained_features(batch_normalization), 512
        else:
            features, features_out_channels =
VGG16.make_features(batch_normalization), 512
        super().__init__(features, features_out_channels, num_of_classes,
batch_normalization, dropout)

```

A.14 Файл YOLOv1VGG19.py

```

from model.classification.VGG19 import VGG19
from model.object_detection.YOLOv1Base import YOLOv1Base

class YOLOv1VGG19(YOLOv1Base):
    def __init__(self, num_of_classes, pretrained_features=False,
batch_normalization=False, dropout=False):
        if pretrained_features:
            features, features_out_channels =
VGG19.get_pretrained_features(batch_normalization), 512
        else:
            features, features_out_channels =
VGG19.make_features(batch_normalization), 512

        super().__init__(features, features_out_channels, num_of_classes,
batch_normalization, dropout)

```

A.15 Файл ObjectDetectionModel.py

```

import json
import os
import shutil
from datetime import datetime

import torch
import torch.optim as optim
from PIL import Image
from PIL import ImageDraw, ImageFont

import model.object_detection.utils as utils
from dataset.ObjectDetectionDataset import ObjectDetectionDataset
from model.object_detection.YOLOv1 import YOLOv1
from model.object_detection.YOLOv1LossFunction import YOLOv1LossFunction
from model.object_detection.YOLOv1VGG11 import YOLOv1VGG11
from model.object_detection.YOLOv1VGG13 import YOLOv1VGG13
from model.object_detection.YOLOv1VGG16 import YOLOv1VGG16
from model.object_detection.YOLOv1VGG19 import YOLOv1VGG19
from settings.Settings import Settings

class ObjectDetectionModel:

```

```

weights_file_name = "weights.pth"
details_file_name = "details.json"
optimizer_weights_file_name = "optimizer_weights.pth"

learning_rate = 1e-5
epochs_count = 100

def __init__(self, name, path):
def is_saved_model(self):
def clear(self):
def clear_model(self):
def clear_model_details(self):
def delete(self):
def load_model_details(self):
def load_model_weights(self):
def load_optimizer_weights(self):
def load_model_and_optimizer_weights(self):
def init_new_model_and_optimizer(self, num_of_classes):
def prepare_model_for_training(self, dataset, freeze_features=True):
def process_model_after_training(self):
@staticmethod
def get_device():

def train(self, dataset):
    device = ObjectDetectionModel.get_device()

    train_loader = dataset.get_train_loader()
    validation_loader = dataset.get_validation_loader()

    train_batch_count = len(train_loader)
    validation_batch_count = len(validation_loader)

    num_of_classes = len(self.classes)
    criterion = ObjectDetectionModel.get_default_criterion(num_of_classes)

    print(f"Start training {self.name} model")
    for epoch in range(ObjectDetectionModel.epochs_count):
        print(f"Start epoch {epoch + 1}/{ObjectDetectionModel.epochs_count} at
{datetime.now()}")

        self.model.eval()

        with torch.no_grad():
            outputs_box, targets_box = utils.get_bounding_boxes(train_loader,
self.model, num_of_classes,
device=device)
            train_map = utils.mean_average_precision(outputs_box, targets_box,
num_of_classes).item()

        self.model.train()
        print(f"Train mAP {train_map}")

        train_loss = 0

        for batch, (inputs, labels) in enumerate(train_loader):
            print(f"Start batch {batch + 1}/{train_batch_count}")

            inputs = inputs.to(device)
            labels = labels.to(device)

            self.optimizer.zero_grad()

```

```

        outputs = self.model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        self.optimizer.step()

        train_loss += loss.item()

    train_loss /= train_batch_count

    print(f"End epoch {epoch + 1}/{ObjectDetectionModel.epochs_count} at
{datetime.now()}."
          f" Training loss: {train_loss} Training mAP: {train_map}")

    print(f"Start validation at {datetime.now()}")

    self.model.eval()

    validation_loss = 0

    with torch.no_grad():
        outputs_box, targets_box =
utils.get_bounding_boxes(validation_loader, self.model, num_of_classes,
                           device=device)
        validation_map = utils.mean_average_precision(outputs_box,
targets_box, num_of_classes).item()

        for batch, (inputs, labels) in enumerate(validation_loader):
            print(f"Start batch {batch + 1}/{validation_batch_count}")

            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = self.model(inputs)
            validation_loss += criterion(outputs, labels).item()

        validation_loss /= validation_batch_count

    print(f"End validation at {datetime.now()}."
          f" Validation loss: {validation_loss} Validation mAP:
{validation_map}")

    print("Update model details")
    self.train_history["train_loss"].append(train_loss)
    self.train_history["train_accuracy"].append(train_map)
    self.train_history["validation_loss"].append(validation_loss)
    self.train_history["validation_accuracy"].append(validation_map)
    self.test_results = {"test_loss": validation_loss, "test_accuracy":
validation_map}

    print("Save model to disk")
    self.save()

    self.last_change_time = datetime.now()

    print(f"End training {self.name} model")

def prepare_model_for_predicting(self):
def process_model_after_predicting(self):

def predict(self, img_path):
    image = Image.open(img_path).convert("RGB")
    transform = ObjectDetectionDataset.get_default_image_inference_transforms()

```

```

transformed_image = transform(image)

device = ObjectDetectionModel.get_device()

self.model.eval()
with torch.no_grad():
    inputs = transformed_image.to(ObjectDetectionModel.get_device())
    inputs = inputs.unsqueeze(0)
    outputs = self.model(inputs)
    num_of_classes = len(self.classes)
    bounding_boxes = utils.convert_batch_output_to_bounding_boxes(outputs,
num_of_classes, device=device)
    bounding_boxes = utils.non_max_suppression(bounding_boxes[0])

    image_with_bounding_boxes = Image.open(img_path).convert("RGB")
    self.draw_predictions_on_image(image_with_bounding_boxes, bounding_boxes)

    return image_with_bounding_boxes

def draw_predictions_on_image(self, image, bounding_boxes):
    draw_image = ImageDraw.Draw(image)

    for bounding_box in bounding_boxes:
        cls, probability, cx, cy, w, h = bounding_box

        half_w, half_y = w / 2, h / 2
        x1, y1 = cx - half_w, cy - half_y
        x2, y2 = cx + half_w, cy + half_y

        image_width, image_height = image.size
        image_x1, image_y1 = x1 * image_width, y1 * image_height
        image_x2, image_y2 = x2 * image_width, y2 * image_height

        predicted_class_name = self.classes[cls]

        if predicted_class_name in ObjectDetectionDataset.ROAD_SIGNS_TRANSLATION:
            predicted_class_name =
ObjectDetectionDataset.ROAD_SIGNS_TRANSLATION[predicted_class_name]

            draw_image.rounded_rectangle((image_x1, image_y1, image_x2, image_y2),
radius=5, outline="#39FF14", width=5)
            font = ImageFont.truetype(Settings.RESOURCE_PATH + "font.ttf", 24,
encoding='UTF-8')
            draw_image.text((image_x1, image_y1 - 24), predicted_class_name,
fill="#39FF14", font=font)

    def save(self):

    @staticmethod
    def get_default_optimizer(model):
        return optim.Adam(model.parameters(), lr=ObjectDetectionModel.learning_rate)

    @staticmethod
    def get_default_criterion(num_of_classes):
        return YOLOv1LossFunction(num_of_classes)

    @staticmethod
    def get_model_by_name(name, num_of_classes, pretrained_features):
        if name == "YOLOv1":
            return YOLOv1(num_of_classes, pretrained_features, dropout=True)
        if name == "YOLOv1_BN":

```

```

        return YOLOv1(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
    if name == "YOLOv1VGG11":
        return YOLOv1VGG11(num_of_classes, pretrained_features, dropout=True)
    if name == "YOLOv1VGG11_BN":
        return YOLOv1VGG11(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
    if name == "YOLOv1VGG13":
        return YOLOv1VGG13(num_of_classes, pretrained_features, dropout=True)
    if name == "YOLOv1VGG13_BN":
        return YOLOv1VGG13(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
    if name == "YOLOv1VGG16":
        return YOLOv1VGG16(num_of_classes, pretrained_features, dropout=True)
    if name == "YOLOv1VGG16_BN":
        return YOLOv1VGG16(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)
    if name == "YOLOv1VGG19":
        return YOLOv1VGG19(num_of_classes, pretrained_features, dropout=True)
    if name == "YOLOv1VGG19_BN":
        return YOLOv1VGG19(num_of_classes, pretrained_features,
batch_normalization=True, dropout=True)

```

A.16 Файл ModelsKeeper.py

```

from enum import Enum

from settings.Settings import Settings
from dataset.ClassificationDataset import ClassificationDataset
from dataset.ObjectDetectionDataset import ObjectDetectionDataset
from model.classification.ClassificationModel import ClassificationModel
from model.object_detection.ObjectDetectionModel import ObjectDetectionModel
from util.Singleton import Singleton

class TargetModelTask(Enum):
    CLASSIFICATION = 1
    OBJECT_DETECTION = 2

class ModelsKeeper(metaclass=Singleton):
    def __init__(self):
        self.classification_models = {
            "VGG11": ClassificationModel("VGG11", Settings.VGG11_PATH),
            "VGG11_BN": ClassificationModel("VGG11_BN", Settings.VGG11_BN_PATH),
            "VGG13": ClassificationModel("VGG13", Settings.VGG13_PATH),
            "VGG13_BN": ClassificationModel("VGG13_BN", Settings.VGG13_BN_PATH),
            "VGG16": ClassificationModel("VGG16", Settings.VGG16_PATH),
            "VGG16_BN": ClassificationModel("VGG16_BN", Settings.VGG16_BN_PATH),
            "VGG19": ClassificationModel("VGG19", Settings.VGG19_PATH),
            "VGG19_BN": ClassificationModel("VGG19_BN", Settings.VGG19_BN_PATH),
        }

        self.object_detection_models = {
            "YOLOv1": ObjectDetectionModel("YOLOv1", Settings.YOLOV1_PATH),
            "YOLOv1_BN": ObjectDetectionModel("YOLOv1_BN", Settings.YOLOV1_BN_PATH),
            # "YOLOv1VGG11": ObjectDetectionModel("YOLOv1VGG11",
Settings.YOLOV1VGG11_PATH),

```

```

        "YOLOv1VGG11_BN": ObjectDetectionModel("YOLOv1VGG11_BN",
Settings.YOLOv1VGG11_BN_PATH),
        # "YOLOv1VGG13": ObjectDetectionModel("YOLOv1VGG13",
Settings.YOLOv1VGG13_PATH),
        "YOLOv1VGG13_BN": ObjectDetectionModel("YOLOv1VGG13_BN",
Settings.YOLOv1VGG13_BN_PATH),
        # "YOLOv1VGG16": ObjectDetectionModel("YOLOv1VGG16",
Settings.YOLOv1VGG16_PATH),
        "YOLOv1VGG16_BN": ObjectDetectionModel("YOLOv1VGG16_BN",
Settings.YOLOv1VGG16_BN_PATH),
        # "YOLOv1VGG19": ObjectDetectionModel("YOLOv1VGG19",
Settings.YOLOv1VGG19_PATH),
        "YOLOv1VGG19_BN": ObjectDetectionModel("YOLOv1VGG19_BN",
Settings.YOLOv1VGG19_BN_PATH),
    }

    self.models = {
        TargetModelTask.CLASSIFICATION: self.classification_models,
        TargetModelTask.OBJECT_DETECTION: self.object_detection_models
    }

    def get_models(self, target_model_task):
        return self.models[target_model_task]

    def get_model_by_name(self, name, target_model_task):
        if name in self.models[target_model_task].keys():
            return self.models[target_model_task][name]
        else:
            return None

    @staticmethod
    def get_dataset_by_target_task(target_model_task):
        if target_model_task == TargetModelTask.CLASSIFICATION:
            return ClassificationDataset(Settings.CLASSIFICATION_DATASET_PATH)
        elif target_model_task == TargetModelTask.OBJECT_DETECTION:
            return ObjectDetectionDataset(Settings.OBJECT_DETECTION_DATASET_PATH)
        else:
            return None

```

A.17 Файл Settings.py

```

import os
from pathlib import Path

class Settings:
    PROJECT_PATH = str(Path(__file__).parent.parent)

    DATASETS_PATH = PROJECT_PATH + os.sep + "datasets" + os.sep
    CLASSIFICATION_DATASET_PATH = DATASETS_PATH + "classification" + os.sep
    OBJECT_DETECTION_DATASET_PATH = DATASETS_PATH + "object_detection" + os.sep

    MODELS_PATH = PROJECT_PATH + os.sep + "models" + os.sep
    CLASSIFICATION_MODELS_PATH = MODELS_PATH + "classification" + os.sep
    OBJECT_DETECTION_MODELS_PATH = MODELS_PATH + "object_detection" + os.sep

    VGG11_PATH = CLASSIFICATION_MODELS_PATH + "VGG11" + os.sep
    VGG11_BN_PATH = CLASSIFICATION_MODELS_PATH + "VGG11_BN" + os.sep
    VGG13_PATH = CLASSIFICATION_MODELS_PATH + "VGG13" + os.sep

```

```
VGG13_BN_PATH = CLASSIFICATION_MODELS_PATH + "VGG13_BN" + os.sep
VGG16_PATH = CLASSIFICATION_MODELS_PATH + "VGG16" + os.sep
VGG16_BN_PATH = CLASSIFICATION_MODELS_PATH + "VGG16_BN" + os.sep
VGG19_PATH = CLASSIFICATION_MODELS_PATH + "VGG19" + os.sep
VGG19_BN_PATH = CLASSIFICATION_MODELS_PATH + "VGG19_BN" + os.sep

YOLOV1_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1" + os.sep
YOLOV1_BN_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1_BN" + os.sep
YOLOV1VGG11_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1YOLOv1VGG11" + os.sep
YOLOV1VGG11_BN_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG11_BN" + os.sep
YOLOV1VGG13_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG13" + os.sep
YOLOV1VGG13_BN_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG13_BN" + os.sep
YOLOV1VGG16_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG16" + os.sep
YOLOV1VGG16_BN_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG16_BN" + os.sep
YOLOV1VGG19_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG19" + os.sep
YOLOV1VGG19_BN_PATH = OBJECT_DETECTION_MODELS_PATH + "YOLOv1VGG19_BN" + os.sep

RESOURCES_PATH = PROJECT_PATH + os.sep + "resources" + os.sep
```

ДОДАТОК Б
Слайди презентації



Рисунок Б.1 – Слайд презентації № 1

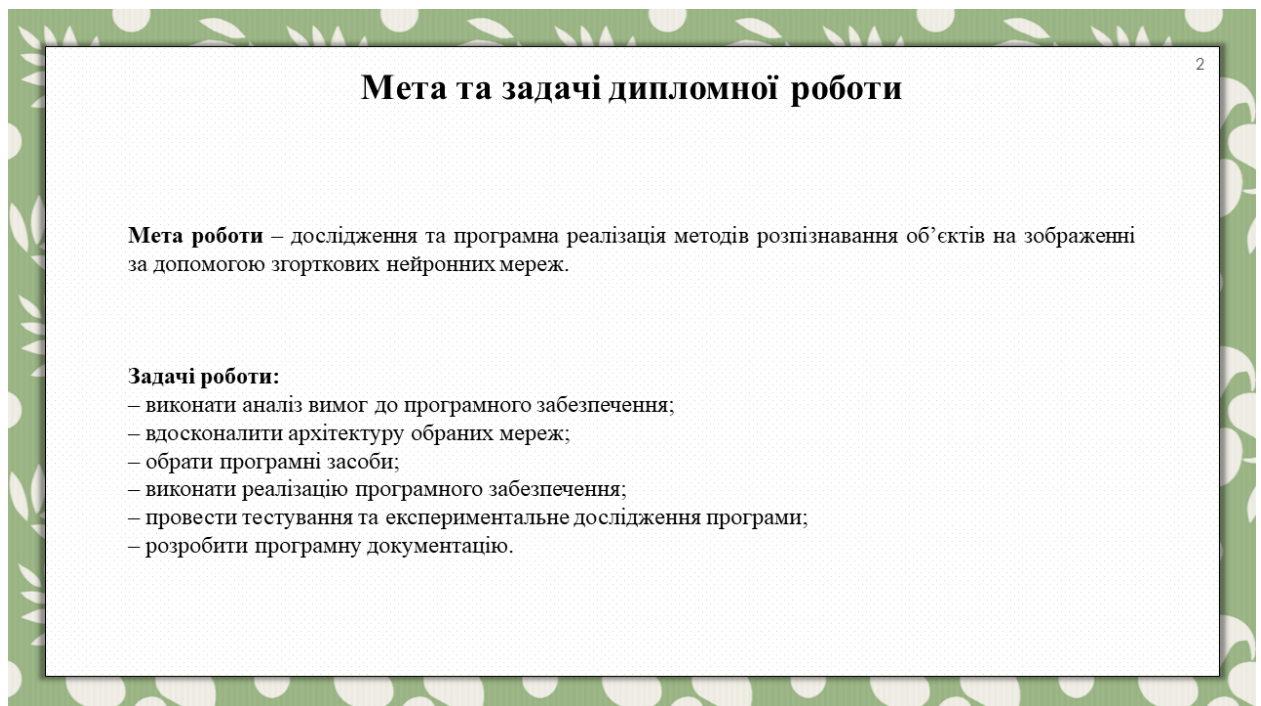
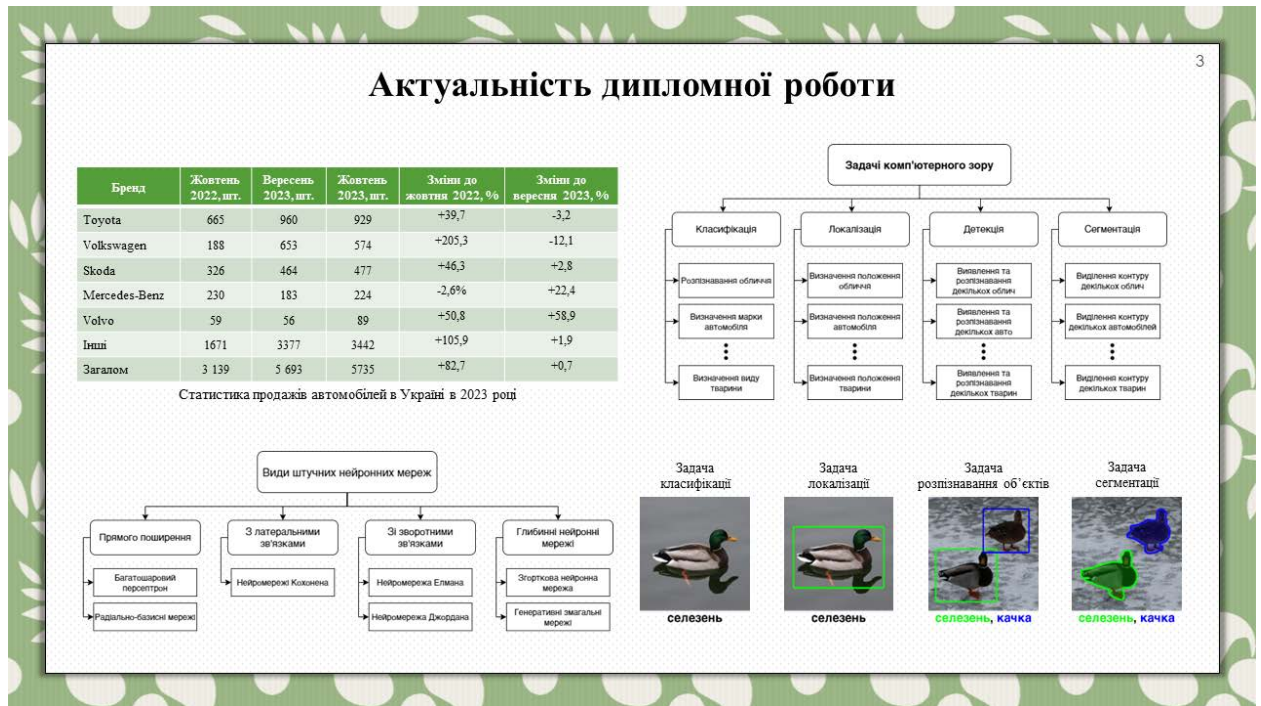


Рисунок Б.2 – Слайд презентації № 2



Згорткові нейронні мережі для класифікації

5

Особливість	LeNet	AlexNet	ZFNet	VGGNet
Рік винайдення	1989	2012	2013-2014	2014
Кількість входних каналів	1	3	3	3
Розмір входного зображення	32x32	227x227	224x224	224x224
Кількість шарів з параметрами	5	8	8	11-19
Кількість згорткових шарів	3	5	5	8-16
Ядро згортки	5x5	11x11	7x7	3x3
Функція активації нейронів	Гіперболічний тангенс	ReLU	ReLU	ReLU
Тип агрегування	Усереднювальне	Максимізаційне	Максимізаційне	Максимізаційне
Складність архітектури	Просто	Середньо	Середньо	Середньо
Час розпізнавання	Швидко	Середньо	Середньо	Довше
Час навчання	Швидко	Середньо	Середньо	Довше
Точність на змаганні ILSVRC	-	83,6%	88,3%	92,7%

Порівняння архітектур згорткових нейронних мереж для класифікації

VGG11	VGG13	VGG16	VGG19
Вхідний шар 224x224x3 (RGB зображення)			
Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Максимізаційне агрегування 2x2 s=2			
Повний пов'язаний шар <4096, ReLU			
Повний пов'язаний шар <4096, ReLU			
Повний пов'язаний шар <1000			
Нормована експоненціальна функція (Softmax)			

Вдосконалена архітектура VGG мереж

Рисунок Б.5 – Слайд презентації № 5

Згорткові нейронні мережі для розпізнавання об'єктів

6

Особливість	R-CNN	Fast R-CNN	Faster R-CNN	YOLOv1
Рік винайдення	2014	2015	2015	2015
Максимальна кількість обмежувальних коробок	2000	2000	-	98
Одноступеневий детектор	Ні	Ні	Ні	Так
Обробка зображень в режимі реального часу	Ні	Ні	Теоретично	Так
Обробка відео в режимі реального часу	Ні	Ні	Ні	Так
Час навчання	Довго	Середньо	Швидше	Швидше
Складність реалізації	Середньо	Складно	Складно	Середньо
Ресурсоеміст обчислень	Багато	Багато	Середньо	Мало
Кількість зображень в секунду	6	0,5	18	45
Усереднена точність на PASCAL VOC	53,5%	70%	62,1%	63,4%

Порівняння архітектур згорткових нейронних мереж для розпізнавання об'єктів

YOLOv1	YOLOv1 VGG11	YOLOv1 VGG13	YOLOv1 VGG16	YOLOv1 VGG19
Вхідний шар 224x224x3 (RGB зображення)				
Згортка 3x3x64 s=2 p=0, BN, LReLU	Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU	Згортка 3x3x64, BN, ReLU
Максимізаційне агрегування 2x2 s=2				
Згортка 3x3x192, BN, LReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU
Максимізаційне агрегування 2x2 s=2				
Згортка 1x1x128 p=0, BN, LReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU
Згортка 3x3x256, BN, LReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU	Згортка 3x3x256, BN, ReLU
Згортка 1x1x256 p=0, BN, LReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU	Згортка 3x3x128, BN, ReLU
Максимізаційне агрегування 2x2 s=2				
Згортка 1x1x256 p=0, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Згортка 3x3x512, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Згортка 1x1x256 p=0, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Згортка 3x3x512, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Згортка 1x1x512 p=0, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Згортка 3x3x1024, BN, LReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU	Згортка 3x3x512, BN, ReLU
Максимізаційне агрегування 2x2 s=2				
Згортка 3x3x1024, BN, LReLU	Згортка 3x3x1024 s=2 p=0, BN, LReLU	Згортка 3x3x1024, BN, LReLU	Згортка 3x3x1024, BN, LReLU	Згортка 3x3x1024, BN, LReLU
Максимізаційне агрегування 2x2 s=2				
Повний пов'язаний шар <4096, Linear				
Повний пов'язаний шар <5*5*(B+1+C)				

Вдосконалена архітектура мережі YOLOv1

Рисунок Б.6 – Слайд презентації № 6



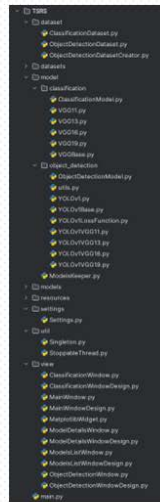
Рисунок Б.7 – Слайд презентації № 7



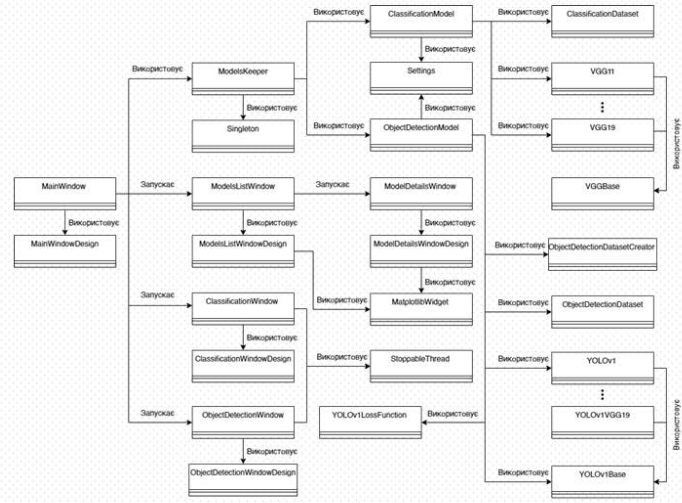
Рисунок Б.8 – Слайд презентації № 8

Структура програмного забезпечення

9



Загальна структура розробленої програми



Діаграма класів програми

Рисунок Б.9 – Слайд презентації № 9

Тестування

10

Короткий опис	Перевірка можливості навчання нейромережевої моделі		
Передумови	Відкрито головне вікно з меню програми		
Кроки	№	Покрокові дії	Очікуваний результат
	1	В підменю задачі розпізнавання об'єктів натиснути на кнопку навчання	Відкрито вікно з моделями для задачі розпізнавання об'єктів на зображенні
	3	Натиснути на кнопку детальніше будь-якої нейромережевої моделі	Відкрито вікно навчання моделі.
	4	Натиснути на кнопку навчити	Якщо модель не була попередньо навчена, то розпочато навчання. Якщо модель була попередньо навчена, то відображено діалогове вікно після погодження з яким прогрес навчання скинуто та розпочато знов. Поруч з назвою моделі з'явилось відповідне повідомлення.
	5	Натиснути на кнопку зупинити навчання	Навчання моделі зупинено
	6	Натиснути на кнопку донавчити	Якщо модель не була попередньо натренованою, то розпочато процес навчання. Поруч з назвою моделі з'явилось відповідне повідомлення.
	7	Зупинити навчання та натиснути на кнопку очистити	Якщо модель натренована, то відображено діалогове вікно після погодження з яким процес навчання буде скинуто
	8	Закрити вікно навчання моделі	Відкрито вікно з моделями для задачі розпізнавання об'єктів

Тест-кейс перевірки процесу навчання моделі

Вимога	Статус
Головне вікно з меню програми	Пройдено
Відображення назви і логотипу програми	Пройдено
Фіксований розмір та макет вікна	Пройдено
Вікна зі списками нейромережевих моделей	Пройдено
Наявність нейромережевих моделей у списку відповідно до задачі	Пройдено
Відображення назви моделей, графіків, значень точності та втрат	Пройдено
Прокрутка вікна за допомогою повзунок чи колеса миші	Пройдено
Змінюваний розмір вікна та адаптивний макет	Пройдено
Вікно навчання моделі та перегляду її деталей	Пройдено
Наявність детальної інформації про модель	Пройдено
Можливість організації навчання та донавчання моделі	Пройдено
Вікно класифікації зображення	Пройдено
Наявність навчених моделей в комбінованому списку	Пройдено
Можливість вибрати та завантажити зображення з пам'яті комп'ютера	Пройдено
Відображення зображення та результату класифікації	Пройдено
Вікно розпізнавання об'єктів на зображенні	Пройдено
Наявність навчених моделей в комбінованому списку	Пройдено
Можливість вибрати та завантажити зображення з пам'яті комп'ютера	Пройдено
Відображення зображення та результату розпізнавання об'єктів	Пройдено
Загально	Пройдено
Взаємодія графічного інтерфейсу з діями користувача	Пройдено
Читабельний шрифт та розмір тексту	Пройдено
Повідомлення користувачу при виконанні потенційно небезпечних дій	Пройдено

Контрольний список тестування

Рисунок Б.10 – Слайд презентації № 10

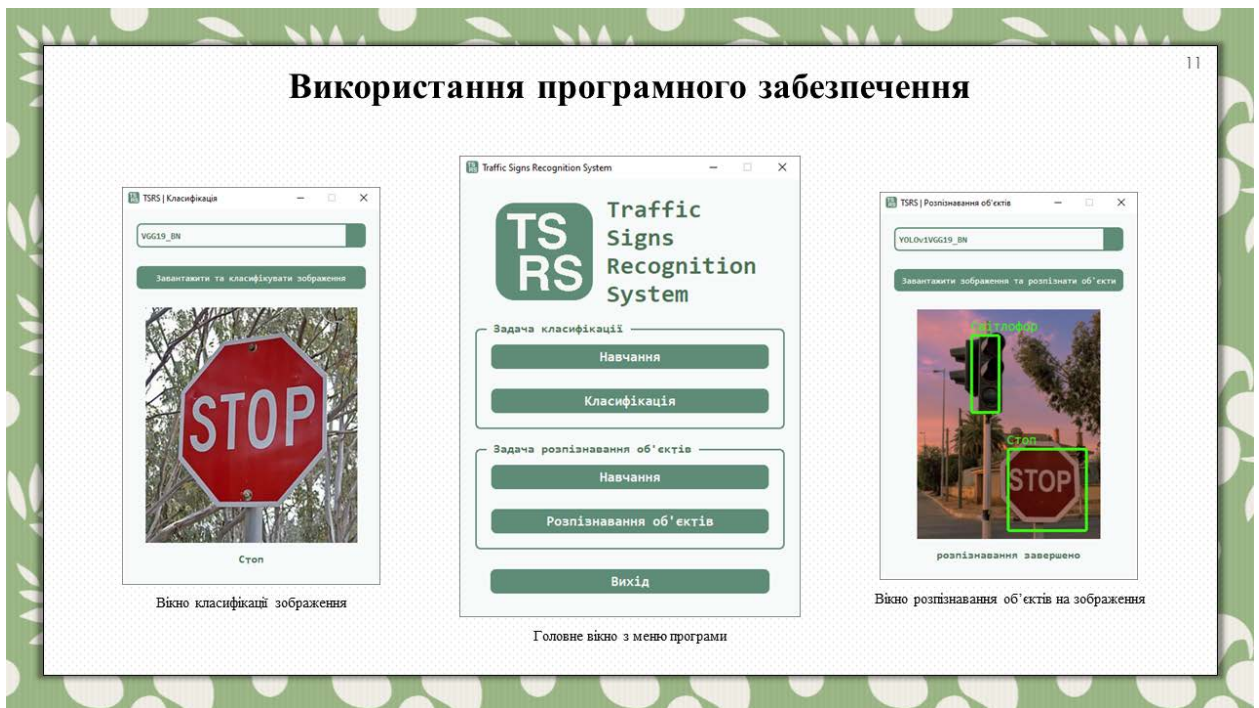


Рисунок Б.11 – Слайд презентації № 11



Рисунок Б.12 – Слайд презентації № 12



Рисунок Б.13 – Слайд презентації № 13

14

Експериментальне дослідження програми

Характеристика	VGG11	VGG11 BN	VGG13	VGG13 BN	VGG16	VGG16 BN	VGG19	VGG19 BN
Точність моделі на наборі даних для навчання, %	97,15	97,95	98,35	98,00	98,65	98,75	99,25	99,30
Точність моделі на наборі даних для валідації, %	93,60	94,00	96,40	94,80	97,20	98,00	98,80	99,20
Втрати моделі на наборі даних для навчання	0,0509	0,0524	0,0435	0,0515	0,0353	0,0318	0,0212	0,0110
Втрати моделі на наборі даних для валідації	0,3031	0,2508	0,0904	0,1905	0,0734	0,0420	0,0309	0,0118
Кількість шарів згортки	8	8	10	10	13	13	16	16
Кількість параметрів з вагами, тис. шт.	128807	128812	128991	128997	134301	134309	139611	139622
Вага файлу з вагами, МБ	503,162	503,218	503,884	503,946	524,628	524,714	545,371	545,482
Епохи	10	10	10	10	10	10	10	10
Час навчання однієї епохи	40хв	25 хв	50хв	35хв	1 год 10 хв	45 хв	1 год 20 хв	50 хв
Загальний час навчання	6 год 40 хв	4 год 10 хв	8 год 20 хв	5 год 50 хв	11 год 40 хв	7 год 30 хв	13 год 20 хв	8 год 20 хв
Епоха на якій перевищено точність у 80% на навчальному наборі даних	4	2	4	2	4	2	4	2
Епоха на якій перевищено точність у 80% на валідаційному наборі даних	4	2	4	2	4	2	4	2
Епоха на якій перевищено точність у 90% на навчальному наборі даних	5	3	5	3	5	3	5	3
Епоха на якій перевищено точність у 90% на валідаційному наборі даних	6	3	5	3	5	3	5	3

Результати навчання та використання мереж VGG для задачі класифікації зображення

Рисунок Б.14 – Слайд презентації № 14



Рисунок Б.15 – Слайд презентації № 15

16

Експериментальне дослідження програми

Характеристика	YOLOv1	YOLOv1 BN	YOLOv1 VGG11 BN	YOLOv1 VGG13 BN	YOLOv1 VGG16 BN	YOLOv1 VGG19 BN
Точність моделі на наборі даних для навчання, %	53,86	54,57	58,57	59,71	61,00	61,14
Точність моделі на наборі даних для валідації, %	46,67	48,00	53,33	54,67	57,33	57,33
Втрати моделі на наборі даних для навчання	86	90	83	78	81	77
Втрати моделі на наборі даних для валідації	109	102	96	94	84	91
Кількість шарів згортки	20+4	20+4	8+4	10+4	13+4	16+4
Функція активації	LReLU Linear	LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear	ReLU LReLU Linear
Кількість параметрів з вагами, тис. шт.	268491	268517	250603	250788	256101	261413
Вага файлу з вагами, МБ	1048	1049	979	979	1000	1021
Епохи	35	35	35	35	35	35
Час навчання однієї епохи	25 хв	20хв	40хв	45 хв	1 год	1 год 10 хв
Загальний час навчання	14 год 35 хв	11 год 40хв	23 год 20 хв	26 год 15 хв	35 год	40 год 50 хв

Результати навчання та використання мереж YOLOv1 для задачі розпізнавання об'єктів на зображенні

Рисунок Б.16 – Слайд презентації № 16

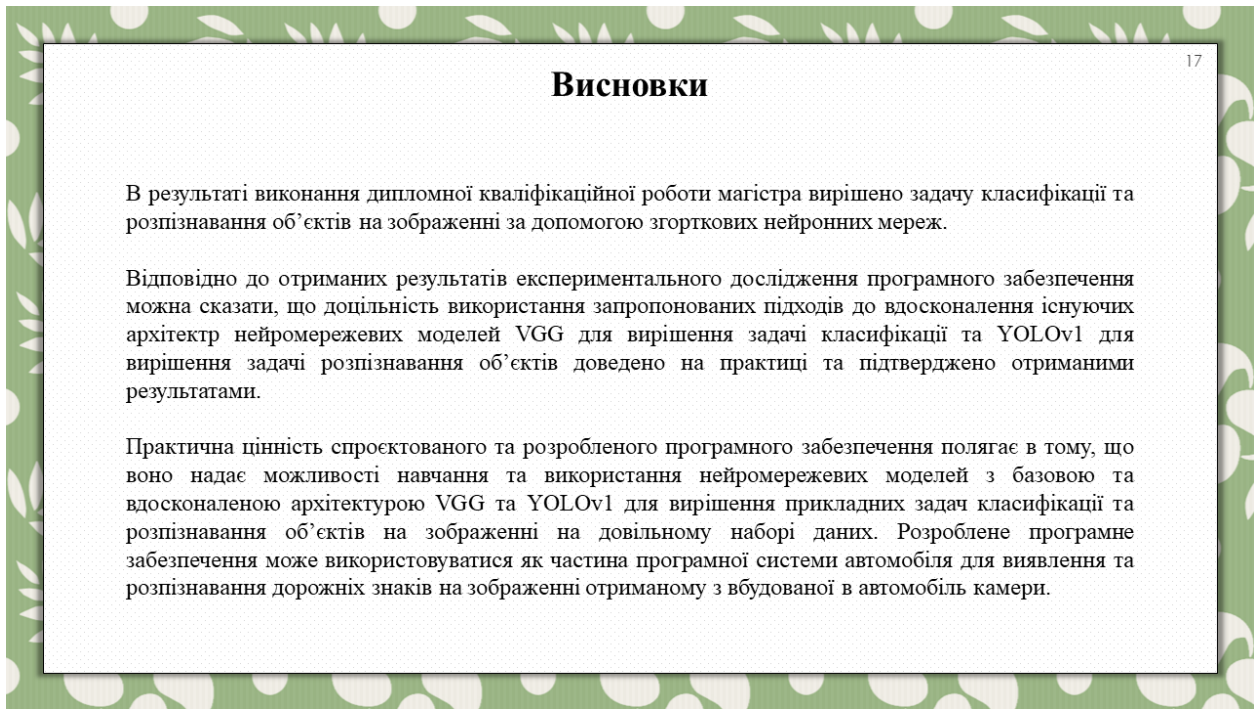


Рисунок Б.17 – Слайд презентації № 17

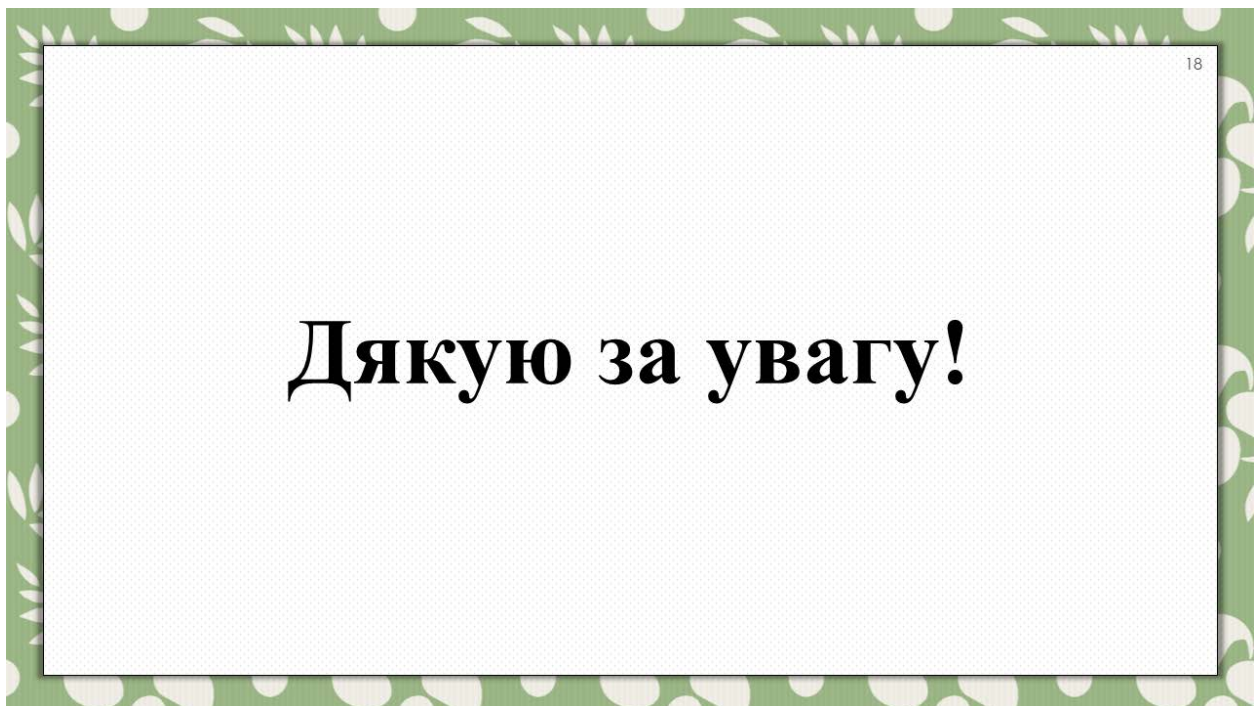


Рисунок Б.18 – Слайд презентації № 18