НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: «Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування»

Виконав:
студент 4 курсу, групи ІО-16
Лисак Андрій Анатолійович

Керівник:
асистент кафедри ОТ
Коренко Дмитро Володимирович _____
(підпис)

Консультант:
Пономаренко Артем Миколайович _____
(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента
Лисака Андрія Анатолійовича

1. Тема проєкту *«Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування»*.
Керівник проєкту**:** Коренко Дмитро Володимирович, асистент кафедри ОТ
затверджені наказом по університету від 4 квітня 2025 року №1326-с

2. Термін здачі студентом закінченого проєкту _____.

3. Вихідні дані до проєкту: технічна документація, теоретичні та статистичні дані, відкриті датасети з зображеннями твердих побутових відходів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): аналіз предметної області, вивчення підходів до застосування машинного навчання, вибір архітектури нейронної мережі, розробка API для взаємодії з модельним ядром, розробка, тренування та тестування моделі ШІ, інтеграція моделі у API, розроблення мобільного застосунку як клієнта для інтеграції, тестування розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): схема архітектури системи класифікації сміття, структурна схема розробленої CNN.

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Пономаренко А. М.		

7. Дата видачі завдання 28 листопада 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми дипломного проєкту*	*29.11.2024 – 01.12.2024*	
2.	*Вивчення літератури, аналіз завдання, вибір інструментів*	*01.12.2024 – 25.04.2025*	
3.	*Підготовка датасету, дослідження його структури*	*28.04.2025 – 02.05.2025*	
4.	*Розробка REST API та програмної частини*	*05.05.2025 – 09.05.2025*	
5.	*Побудова та тренування моделі штучного інтелекту*	*12.05.2025 –13.05.2025*	
6.	*Тестування, коригування моделі, обчислення метрик*	*14.04.2025 – 16.05.2025*	
7.	*Розробка клієнтського застосунку для інтеграції API на Android.*	*16.05.2025 – 30.05.2025*	
8.	*Передзахист*	*30.05.2025 – 15.06.2025*	
9.	*Захист дипломного проєкту*	*16.06.2025*	

Студент-дипломник _____ Андрій ЛИСАК
(підпис)

Керівник проєкту _____ Дмитро КОРЕНКО
(підпис)

АНОТАЦІЯ

У дипломному проєкті представлено інтелектуальну систему для класифікації твердих побутових відходів з метою підтримки екологічного сортування. В основі рішення лежить згорткова нейронна мережа, навчена на базі датасету Garbage Dataset із застосуванням методів аугментації даних (обертання, масштабування, зміна яскравості тощо) для підвищення генеративної здатності моделі. Архітектура реалізована з використанням фреймворку DeepLearning4j, а серверна частина – вигляді REST API за допомогою Spring Boot, що забезпечує можливість інтеграції з клієнтськими застосунками. Якість класифікації оцінено за метриками точності, повноти, прецизійності та F1-міри. Розроблену систему передбачено для інтеграції в інтелектуальні платформи автоматизованого сортування сміття. Для демонстрації роботи системи у мобільному середовищі реалізований Android-застосунок з підтримкою обробки зображень у реальному часі.

ANNOTATION

This thesis presents an intelligent system for the classification of municipal solid waste aimed at supporting ecological sorting. The core of the solution is a convolutional neural network trained on the Garbage Dataset with the use of data augmentation techniques (rotation, scaling, brightness adjustment, etc.) to improve the model's generalization capability. The architecture is implemented using the DeepLearning4j framework, while the server-side component is developed as a REST API with Spring Boot, enabling integration with client application. The classification performance was evaluated using standard metrics: accuracy, recall, precision, and F1-score. The proposed system is intended for integration into intelligent platforms for automated waste sorting. To demonstrate the system's functionality in a mobile environment, an Android application was developed with support for real-time image processing.

справки	Формат	*Значення*	*Найменування*	Кіл. Листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	*ІАЛЦ.466538.002 ТЗ*	Система розпізнавання об’єктів або	4		
			сцен для класифікації сміття та			
			екологічного сортування			
			Технічне завдання			
	А4	*ІАЛЦ.466538.003 ПЗ*	Система розпізнавання об’єктів або	66		
			сцен для класифікації сміття та			
			екологічного сортування			
			Пояснювальна записка			
	А4	*ІАЛЦ.466538.004 Д1*	Система розпізнавання об’єктів або	1		
			сцен для класифікації сміття та			
			екологічного сортування			
			Діаграма послідовності взаємодії			
			компонентів			
	А4	*ІАЛЦ.466538.005 Д2*	Система розпізнавання об’єктів або	1		
			сцен для класифікації сміття та			
			екологічного сортування			
			Функціональна схема			
			(Діаграма класів)			
	А4	*ІАЛЦ.466538.006 Д3*	Система розпізнавання об’єктів або	1		
			сцен для класифікації сміття та			
			екологічного сортування			
			Алгоритм роботи системи			
			класифікації сміття			
	А4	*ІАЛЦ.466538.007 Д4*	Система розпізнавання об’єктів або	13		
			сцен для класифікації сміття та			
			екологічного сортування			
			Текст програмного коду			

					ІАЛЦ.466530.001 ОА
Змін.	Арк.	**№ докум.**	**Підпис**	**Дата**	
Розроб.		Лисак А.А.			*Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування* Опис альбому
Перев.		Коренко Д. В.			
Реценз.					
Н. Контр.					
Затверд.					

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ» ФІОТ ІО-16

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування»

Київ - 2025

ЗМІСТ

					ІАЛЦ.466530.002 ТЗ		
Змін.	Арк.	**№ докум.**	**Підпис**	**Дата**			
Розроб.		Лисак А.А.			*Система розпізнавання об'єктів або сцен для класифікації сміття та екологічного сортування* Технічне завдання	Літ.	Аркуш 1 / Аркушів 4
Перев.		Коренко Д. В.					
Реценз.						***НТУУ «КПІ» ФІОТ ІО-16***	
Н. Контр.		TODO					
Затверд.		TODO					

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Дане технічне завдання стосується створення інтелектуальної системи розпізнавання об’єктів або сцен для класифікації сміття з подальшим екологічним сортуванням.

Розроблена система призначена для використання в інфраструктурі розумного міста, зокрема в сортувальних центрах, контейнерах для відходів нового покоління або у мобільних застосунках для допомоги користувачам у побуті.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки проєкту стало завдання на бакалаврську дипломну роботу за освітньо-професійною програмою «Комп’ютерні системи та мережі» спеціальності 123 «Комп’ютерна інженерія», затверджене кафедрою Обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою проєкту є реалізація автоматизованої системи, що дозволяє на основі зображень визначати тип сміття та пропонувати правильний варіант його сортування. Система орієнтована на екологічну ефективність та зручність використання в реальному середовищі – як через інтерфейс, так і вбудовування в «розумне» обладання.

4. ДЖЕРЕЛА РОЗРОБКИ

Під час розробки системи використовувались наступні джерела:

- Наукова література з тем штучного інтелекту та комп’ютерного зору;
- Технічна документація з фреймворків DeepLearning4j, Spring Boot;
- Опис відкритого датасету Garbage Dataset;
- Статті та інші матеріали з Інтернету щодо класифікації зображень та екологічного сортування відходів.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Функціональні вимоги

- Обробка зображення з метою розпізнати тип сміття.
- Класифікація по категоріях (скло, папір, картон, пластик, метал, інше).
- Надання доступу до функціоналу через API.

5.2 Вимоги до програмного забезпечення

- Java 17 або вище.
- Spring Boot (версія 3.x).
- DeepLearning4j.
- Maven як система збирання.

5.3 Вимоги до апаратної платформи

- Процесор не нижче Intel Core i5 або аналог.
- Встановлена оперативна пам’ять від 8 ГБ.
- Вільне місце на диску від 4 ГБ.
- Підключення до мережі Інтернет.
- Інтегрований графічний процесор або GPU.

6. ЕТАПИ РОЗРОБКИ

№	Назва етапу	Дата
6.1	Ознайомлення з тематикою, аналіз джерел	20.02.2025
6.2	Складання та погодження технічного завдання	05.03.2025
6.3	Обробка та підготовка датасету	15.03.2025
6.4	Побудова та навчання моделі CNN	01.04.2025
6.5	Реалізація REST API через Spring Boot	15.04.2025
6.6	Тестування та аналіз результатів	25.04.2025

6.7	Оптимізація, виправлення, фіналізація	01.05.2025
6.8	Оформлення пояснювальної записки	10.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CNN	Convolutional Neural Network – Згорткова нейронна мережа)
API	Application Programming Interface – Прикладний програмний інтерфейс
DL4J	DeepLearning4j – фреймворк глибокого навчання для мови Java
HTTP	HyperText Transfer Protocol – протокол передачі гіпертекстових документів
JSON	JavaScript Object Notation – текстовий формат обміну даних між комп’ютерами
REST API	Representational State Transfer API – архітектурний стиль для створення веб-сервісів
GPU	Graphics Processing Unit – графічний процесор
ReLU	Rectified Linear Unit – функція активації

ВСТУП

Сучасний світ стикається з никзою глобальних екологічних проблем, серед яких одним із найбільш критичних є зростання обсягів побутовий і промислових відходів. За оцінками міжнародних організацій, щороку у світі утворюється понад дві мільярди тонн твердих відходів, значна частина яких підлягає переробці. Ефективне сортування сміття на етапі його утворення є важливою складовою побудови сталого суспільства та розвитку циркулярної економіки.

Однак, на практиці процес сортування часто ускладнюється через людський фактор: недостатню обізнаність, невідповідальне ставлення або складність правил сортування. Як наслідок, значна частина відходів потрапляє на полігони без належної обробки, що призводить до забруднення довкілля та втрати цінних ресурсів.

Одним із перспективних напрямів вирішення цієї проблеми є автоматизація процесів сортування із залученням сучасних інформаційних технологій, зокрема технологій машинного навчання та комп'ютерного зору. Завдяки можливостям глибокого навчання, системи розпізнавання зображень набули значного розвитку і демонструють високу точність у задачах класифікації об'єктів.

У даному проєкті розробляється система розпізнавання об'єктів або сцен для класифікації сміття та екологічного сортування на основі зображень. В основу системи покладено використання згорткових нейронних мереж (CNN), що дозволяють ефективно працювати з візуальними даними. Для реалізації серверної частини використано фреймворк Spring Boot, який забезпечує створення зручного REST API для інтеграції з клієнтськими застосунками або зовнішніми системами.

У межах проєкту було проведено дослідження предметної області, проаналізовано існуючі рішення в сфері автоматизації сортування сміття, розроблено архітектуру системи, здійснено навчання нейронної мережі на основі

датасету Garbage Dataset із застосування методів агументації даних, а також проведено тестування та оцінювання розробленої моделі за основними якісними метриками (точність, повнота, F1-міра).

Актуальність теми обумовлена зростаючими екологічними викликами та необхідністю впровадження технологічних рішень для зменшення негативного впливу відоходів на довкілля. Запропоноване рішення має практичне значення для модернізації інфраструктури поводження з відходами та сприяння екологічній освіті населення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема накопичення та неправильного сортування сміття

З кожним роком кількість твердих побутових відходів у світі стрімко зростає. За даними Світового банку, щорічно у світі утворюється понад 2 мільярди тонн відходів, і ця цифра продовжує збільшуватися у зв'язку із зростанням населення та урбанізацією. Значна частина цих відходів могла б бути перероблена за умови правильного сортування, однак на практиці часто відбувається змішування різних типів матеріалів, що ускладнює їх подальшу обробку та переробку.

Неправильне сортування сміття призводить до таких наслідків:

- Збільшення обсвігів відходів на полігонах
- Забруденення ґрунтів, підземних вод та атмосфери.
- Витрати додаткових ресурсів на вторинну переробку.
- Погіршення екологічної ситуації в регіонах.

В Україні ситуація також залишається складною: за даними Міністерства захисту довкілля та природних ресурсів, понад 90% побутових відходів захоронюється на полігонах без попереднього сортування.

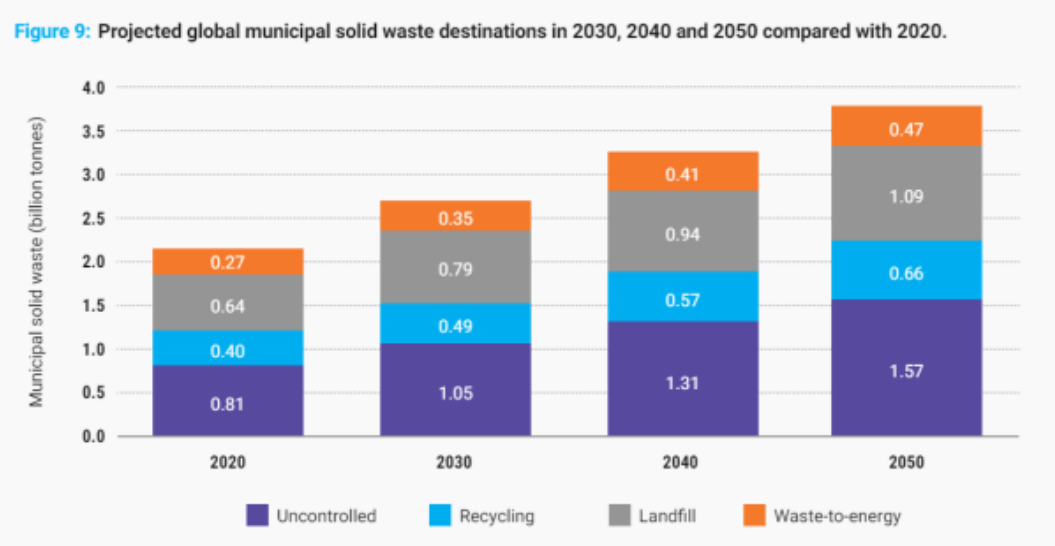


Рисунок 1.1 – Прогнозовані глобальні напрямки поводження з твердими побутовими відходами

Таким чином, проблема ефективного сортування сміття є однією з ключових для досягнення сталого розвитку та збереження навколишнього середовища.

1.2 Методи поводження з відходами

Питання ефективного поводження з побутовими та промисловими відходами передбачає застосування різних підходів залежно від фізичних, хімічних та біологічних властивостей відходів. Основна мета – мінімізувати негативний вплив на довкілля та забезпечити повторне використання ресурсів.

- Механічна обробка – передбачає подрібнення (дроблення, помел) і укрупнення (гранулювання, таблетування, брикетування) з метою підготовки до подальших етапів.
- Класифікація і сортування – реалізується за допомогою механічних або фізичних процесів (гуркотіння, сепарація) для відділення фракцій за матеріалами та розмірами.
- Збагачування – включає процеси гравітаційної, флотаційної, магнітної та електричної сепарації, які дозволяють вилучити цінні компоненти.
- Термічна переробка – охоплює піроліз, газифікацію, а також спалювання з утилізацією теплової енергії.
- Вилуговування і зневоднювання – переважно застосовуються у спеціалізованих промислових галузях.

На рисунку 1.1 представлено узагальнену схему основних методів підготовки та переробки відходів.

Рисунок 1.2 – Методи підготовкит і переробки відходів

Серед усіх зазначених підходів найбільше значення для автоматизованого екологічного сортування мають процеси класифікації та механічної сепарації, які можуть бути частково або повністю автоматизовані за допомогою технологій машинного зору та штучного інтелекту.

1.3 Роль автоматизації у сортуванні сміття

Традиційне ручне сортування сміття є трудомістким, повільним тап не завжди ефективним процесом. Воно залежить від людського фактора – уважності, досвіду, мотивації працівників, а також вимагає спеціального навчання та безперервного контролю. Крім того, ручне сортування відходів нерідко супроводжується контактами з небезпечними речовинами, що створює ризики для здоров’я.

Автоматизація процесів сортування дозволяє вирішити одразу кількап проблем:

- Підвіщити точність і швидкість сортування;
- Зменшити витрати на оплату праці;
- Забезпечити безпеку персоналу;

- Підвищити якість вторинної сировини;
- Забезпечити стабільність і прогнозованість результатів;

Застосування комп’ютерного зору, сенсорних систем та алгоритмів машинного навчання дає змогу автоматичним системам розпізнавати об’єкти на стрічці сортувального конвеєра, класифікувати їх за типом матеріалу та керувати виконавчими механізмами (наприклад, маніпуляторами, які відокремлюють об’єкти у відповідні контейнери).

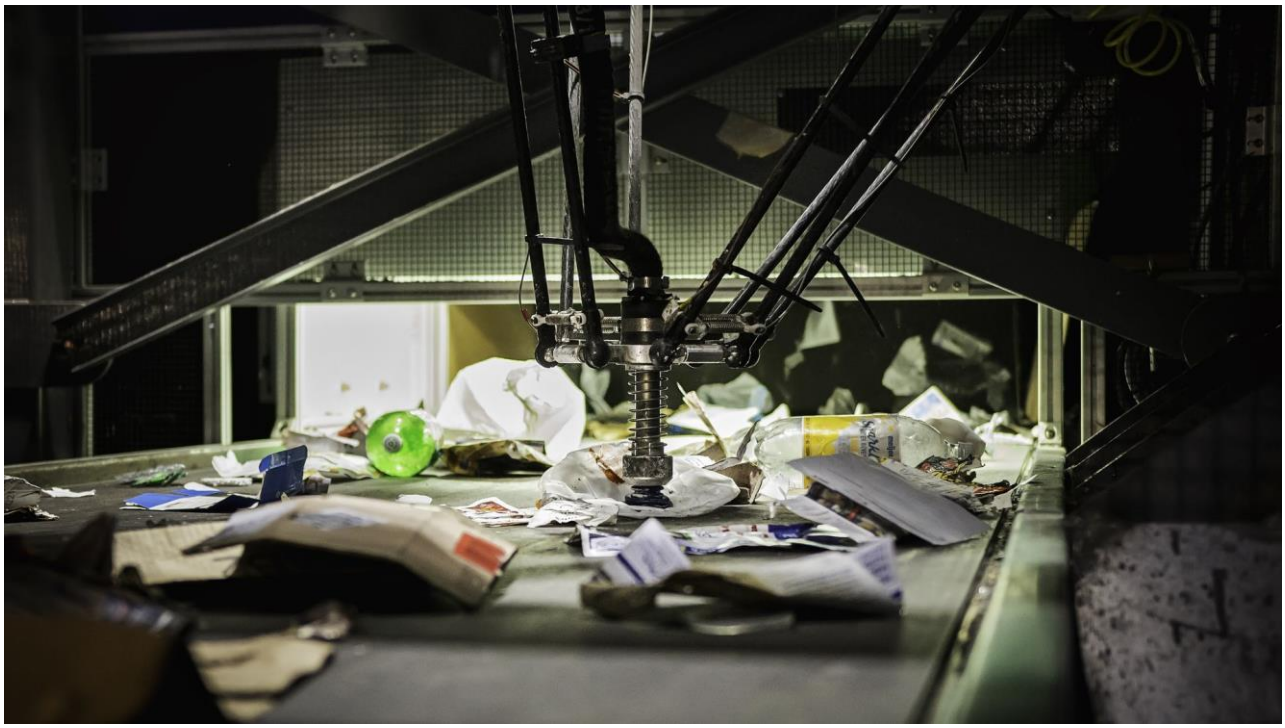


Рисунок 1.3 – Високошвидкісна робототехнічна система від AMP Robotics на переробному підприємстві Waste Connections

Такі системи вже впроваджуються у багатьох країнах світу. Наприклад, у Німеччині, Японії та Шцевії використовуються роботизовані установки з камерою та ШІ, що можуть розпізнавати понад 20 типів відходів. В Україні аналогічні проєкти також починають з’являтися, зокрема у сфері муніципального управління відохдами.

Таким чином, автоматизація процесів сортування є не лише технічною інновацією, а й невід’ємною частиною системи екологічного управління та ресурсоефективності.

1.4 Огляд сучасних рішень для автоматизації сортування

Ідея автоматизованого сортування сміття активно реалізується в різних країнах світу як на рівні муніципального управління, так і в приватному секторі. На ринку існує чимало технологічних рішень, які поєднують апаратну частину (сенсори, камери, маніпулятори) з інтелектуальним програмним забезпеченням на базі машинного навчання.

Серед найбільш відомих прикладів:

- AMP Robotics (США) – компанія, яка розробляє роботів для сміттєсортувальних ліній на базі глибокого навчання. Їхні системи можуть розпізнавати мавтеріали (пластик, метал, папір) з точністю понад 90% у реальному часі.
- ZenRobotics (Фінляндія) – компанія, яка створила першу в світі комерційну систему роботизованого сортування з використанням комп’ютерного зору та ШІ.
- Tomra Sorting (Норвегія) – виробник сенсорних систем сортування, які використовуються як для вміття, так і в аграрному секторі. Їхні системи використовують спектральний аналіз для точного розпізнавання матеріалів.
- CleanRobotics (США) – розробила сміттєву урну «TrashBot», яка самостійно сортує викинути предмети завдяки вбудованій камері, сенсорам та AI-моделі.

Рисунок 1.4 – Сучасні рішення для автоматизації сортування від AMP Robotics, Tomra Sorting, ZenRobotics

В Україні розвиток таких систем лише починається, але вже існують ініціативи з автоматизованого збору й сортування сміття. Наприклад:

- **Сервіси сортуваня в додатках** типу «Ecomap» або «Сортуй», які допомагають користувачам визначити тип відходів за фото.
- **Експериментальні сортувальні станції** у великих містах, що використовуються механізовані лінії з елементами візуального контролю.

Таким чином, світова практика демонструє високий потенціал автоматизації у сфері екологічного сортування. Впровадження подібних систем у країнах з перехідною економікою, зокрема в Україні, є перспективним напрямом як з технологічної, так і з екологічної точки зору.

1.5 Постановка задачі для дипломного проєкту

Автоматизація сортування сміття є технічно складною задачею, що вимагає інтеграції декількох компонентів: засобів збору вхідних даних (зображень), програмних методів аналізу, а також механізмів взаємодії з користувачем або іншим обладнанням. З урахуванням вивчених рішень та наявних технологій у межах цього дипломного проєкту ставиться завдання створити програмну систему, що виконує розпізнавання об’єктів або сцен на зображенні та визначає тип відходів з метою подальшого екологічного сортування.

Система має відповідати таким основним вимогам:

- Отримувати на вхід зображення у форматі PNG/JPG.
- Виконувати попередню обробку зображення та передбачену аугментацію (обертання, масштабування тощо).
- Класифікувати об’єкт за однією з фіксованих категорій (пластик, папір, скло, метал, органіка, інше)
- Повертати рекомендацію щодо контейнера для утилізації.
- Надати API для інтеграції з мобільними застосунками або зовнішніми пристроями (наприклад, розумною урною або сортувальною лінією).

Архітектурно система повинна складатися з двох основних частин:

- **Модуль глибокого навчання**, реалізований на базі DeepLearning4j.
- **Серверна частина**, побудована на фреймворку Spring Boot, що забезпечує API для обробки запитів та повернення результатів.

Крім того, з метою демонстрації роботи розробленої системи, передбачено створення мобільного застосунку на платформі **Android.** Він дозволяє користувачеві завантажити або зробити фото, надіслати його на сервер та отримати результат класифікації у вигляді рекомендації щодо правильного контейнера. Це забезпечує приклад інтеграції клієнтської частини з основним API, а також демонструє практичну цінність та зручність користування системою.

Для навчання моделі використовується відкритий датасет Garbage Dataset, що містить зображення поширених видів сміття. Для підвищення точності застосовуються методи розширення даних (аугментації). Навчальна вибірка розділяється на тренувальну, валідаційну та тестову частини з метою забезпечення коректної оцінки якості моделі.

Результатом дипломного проєкту має стати прототип програмної системи, здатний приймати зображення, класифікувати тип відходів, повертати результат у зручному форматі та бути готовуим до інтеграції з інешими програмними або апаратними компонентами.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проаналізовано предметну область автоматизованого сортування відходів, визначено основні екологічні та технологічні аспекти проблеми, а також окреслено актуальність використання сучасних інформаційних технологій для її вирішення. Було розглянуто основні методи поводження з твердими побутовими відходами, зокрема сортування, переробку, термічну та біологічну обробку.

Особливу увагу приділено огляду існуючих автоматизованих рішень на базі комп'ютерного зору та штучного інтелекту, зокрема роботизованим системам сортування, що вже впроваджуються в розвинених країнах. Аналіз показав високий потенціал використання технологій глибокого навчання для розпізнавання типів відходів за зображенням.

У результаті проведеного аналізу сформульовано задачу дипломного проєкту: створення інтелектуальної системи, яка дозволяє на основі зображень класфиікувати сміття за матеріалом з метою екологічного сортування. Передбачено розробку серверної частини з API та демонстраційного Android-додатку як клієнта для інтеграції та перевірки роботи системи.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ МАШИННОГО НАВЧАННЯ ТА КОМП'ЮТЕРНОГО ЗОРУ

2.1 Загальні відомості про машинне навчання

Машинне навчання (англ. *Machine Learning, ML*) – це галузь досліджень штучного інтелекту, що зосереджується на створенні алгоритмів, здатних виявляти закономірності в даних та приймати рішення без явного програмування. Замість написання жорстких правил для кожного випадку, система «вчиться» на основі прикладів і набуває здатності узагальнювати інформацію.

Основні типи машинного навчання:

- Навчання з учителем (supervised learning) – система тренується на основі даних, які мають правильні мітки (наприклад, зображення + назва об'єкта). Це підхід, що використовується в даному дипломному проєкті.
- Навчання без учителя (unsupervised learning) – використовується для пошуку структури в немаркованих даних (кластеризація, зниження розмірності).
- Навчання з підкріпленням (reinforcement learning) – система взаємодіє із середовищем, отримує нагороду/штраф за дії та вчиться через досвід.

Машинне навчання є основою для розв'язання багатьох сучасних задач: розпізнавання образів, обробка природної мови, пердбачення поведінки користувачів, управління роботами, медична діагностика тощо.

У контексті даного проєкту, задача класифікації сміття за зображенням належить до **завдань навачння з учителем**, де нейронна мережа навчається на основі прикладів типових зображень відходів.

2.2 Глибоке навчання та нейронні мережі

2.2.1 Структура штучної нейронної мережі (ШНМ)

Штучна нейронна мережа – це модель, натхненна роботою біологічного мозку, що складається з елементарних обчилювальних одиниць – **нейронів**, об'єднаних у **шари**. Типова структура складається з:

- **Вхідного шару**, що приймає вхідні дані (наприклад, пікселі зображення);
- **Прихованих шарів**, де відбувається обчислення та трансформація інформації;
- **Вихідного шару**, що формує фінальний результат (наприклад, ймовірності належності до певного класу);

Кожен нейрон у мережі отримує на вхід значення від попередніх нейронів, перемножує їх на вагові коефіцієнти, додає зміщення та обчислює вихід через **функцію активації**. Найпоширеніші функції активації:

- Sigmoid
- ReLU (Rectified Linear Unit)
- Tanh

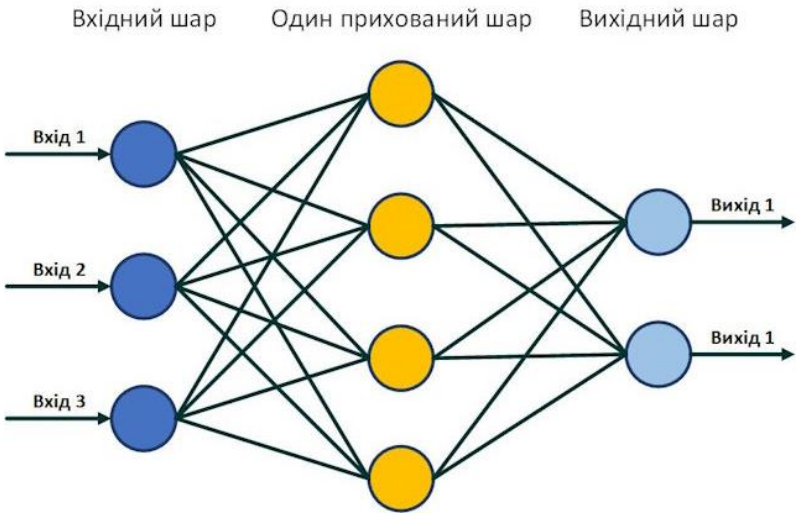


Рисунок 2.1 – Базова структура неглибокої штучної нейронної мережі

Мережа навчається шляхом коригування ваг нейронів на основі функції втрат, використовуючи алгоритми оптимізації (про це у розділі 2.3).

2.2.2 Глибоке навчання та багатошарові мережі

Глибоке навчання (англ. *Deep Learning*) є підрозділом машинного навчання, що використовує багатошарові нейронні мережі для моделювання складних залежностей у даних. Його головна особливість – наявність великої кількості **прихованих шарів**, що дозволяють автоматично витягувати ознаки з вхідної інформації без необхідності ручного конструювання ознак.

Чим більша кількість шарів, тим глибшою є модель, і тим більше рівнів абстракції може розпізнавати. Наприклад, у задачах комп'ютерного зору нижчі шари можуть реагувати на прості елементи зображення (лінії, контури), середні – на фрагменти об'єктів, а вищі – на цілісні об'єкти.

Глибокі моделі зазвичай потребуються великої кількості даних та обчислювальних ресурсів для ефективного навчання, однак завдяки зростанню обчислювальної потужності та доступності фреймворків (TensorFlow, PyTorch, DL4J тощо) ці підходи стали практично застосовними.

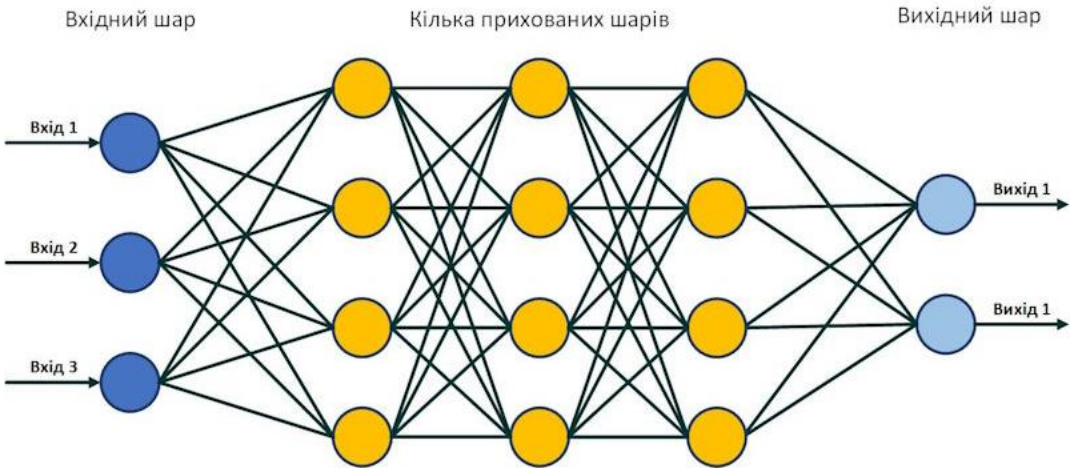


Рисунок 2.1 – Узагальнена структура глибокої нейронної мережі

2.2.3 Згортокові нейронні мережі (CNN)

Згорткові нейронні мережі (*Convolutional Neural Networks)* – це тип глибоких нейронних мереж, спеціально призначених для обробки зображень. Вони враховують просторову структуру вхідних даних, що робить їх особливо ефективними в задачах комп'ютерного зору: класифікації об'єктів, розпізнаванні образів, детекції, сегментації тощо.

Основними компонентами CNN є:

- **Згортковий шар (convolutional layer)** – виконує операцію згортки з фільтрами (ядрами), що виявляють локальні ознаки на зображенні.
- **Шар активації (наприклад, ReLU)** – додає нелійність до моделі.
- **Pooling-шар (subsampling)** – зменшує розмірність зображення, зберігаючи важливу інформацію (наприклад, MaxPooling).
- **Повнозв'язні шари (fully connected layers)** – приймають зменшену вхідну інформацію та формують остаточний вивід.

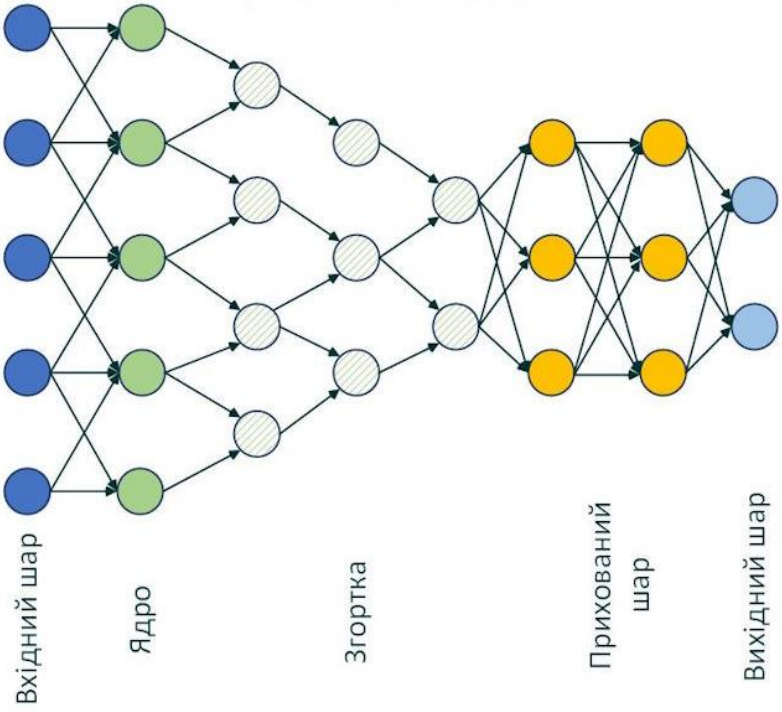


Рисунок 2.3 – Архітектура згорткової нейронної мережі

Типова CNN починається з кількох згорткових і pooling-шарів, які виділяють характеристики зображення, після чого йдуть повнозв'язні шари, які на основі цих характеристик виконують класифікацію.

Завдяки своїй архітектурі CNN автоматично навчаються знаходити найрелевантніші ознаки для розпізнавання, що усуває необхідність ручного виділення ознак.

У рамках даного проєкту CNN використовується як основа для класифікації зображень відходів з датасету Garbage Dataset, де кожне зображення має бути віднесене до певної категорії сміття.

2.3 Процес навчання моделі машинного навчання

Ефективність будь-якої системи, що базується на глибокому машинному навчанні, залежить від якості процесу її тренування. Цей процес включає формування датасету, обрання функції втрат, підбір алгоритму оптимізації та організацію навчального циклу.

2.3.1 Функція втрат

Функція втрат (*loss function*) – це числова міра того, наскільки модель помиляється при пердбаченні результату. Під час навчання модель намагається мінімізувати значення цієї функції, коригуючи внгутрішні параметри (ваги нейронів).

У задачах класифікації найбільш поширеною функцією втрат є **категоріальна крос-ентропія** *(categorical cross-entropy)*, яка визначається як:

$$L = -\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i)$$

де y_i – істинна мітка (1 або 0), $\hat{y}_i$ – передбачена ймовірність для класу i.

Ця функція штрафує модель сильніше, якщо вона була дуже впевнена у неправильному рішенні.

2.3.2 Оптимізатори

Оптимізатори – це алгоритми, які коригують ваги мережі на основі значення функції втрат і обраної швидкості навчання *(learning rate)*. Найпопулярніші:

- **SGD (Stochastic Gradient Descent)** – базовий варіант градієнтного спуску.
- **Adam (Adaptive Moment Estimation)** – сучасний адаптивний метод, який поєднує переваги RMSProp та Momentum. Саме **Adam** найчастіше використовується для CNN.

Оптимізатор поступово зменшує помилку моделі, оновлюючи параметри на кожній ітерації (епосі) навчання.

2.3.3 Розбиття вибірки

Для забезпечення коректного оцінювання моделі датасет зазвичай поділяється на три частини:

- **Тренувальна вибірка (training set)** – для безпосереднього навчання моделі.
- **Валідаційна вибірка (validation set)** – для контролю навчання та уникнення перенавчання.
- **Тестова вибірка (test set)** – для остаточного незалежного оцінювання після завершення навчання.

У межах проєкту для навчання та валідації використано датасет Garbage Dataset, для тестування використано незалежний датасет TrashNet.

2.3.4 Методи підвищення якості (аугментація)

Агументація даних – це метод штучного збільшення обсягу навчального датасету за рахунок модифікації існуючих зображенью. Це дозволяє:

- Зменшити ризик перенавчання;
- підвищити загальну узагальнювальну здатність моделі;

До стандартних методів аугментації належать:

- повороти (rotation);
- масштабування (scaling);
- выддзеркалення (flipping);
- зміна яскравості або констрасту (brightness/contrast shift);
- зсув (transition);

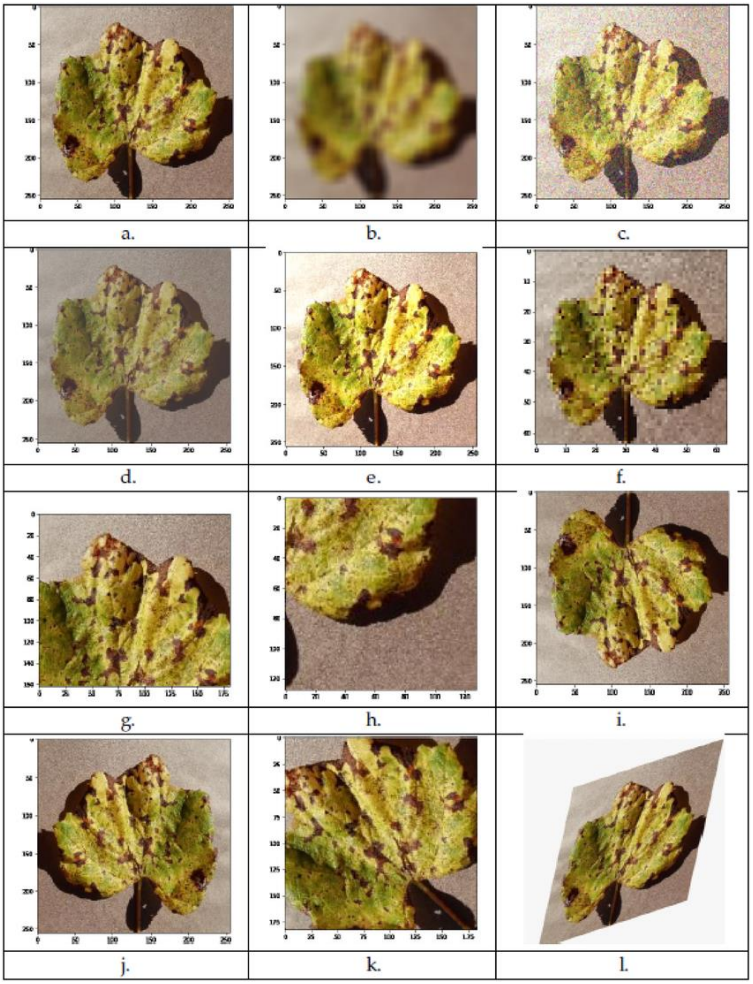


Рисунок 2.4 – Приклади аугментації зображень для підвищення якості навчання

У проєкті використовуються обертання, горизонтальні віддзеркалення та варіації яскравості, що дозволяють ефективно збільшити набір навчальних прикладів без потреби в додаткових зображеннях.

2.4 Метрики оцінки класифікаційних моделей

Оцінка якості роботи класифікаційної моделі є невід'ємною частиною будь-якого проєкту з машинного навчання. Залежно від специфіки задачі, застосовуються різні метрики. У випадку багатокласової класифікації зображень сміття доцільно використовувати такі ключові показники:

Accuracy (точність класифікації)

Це базова метрика, яка показує частку правильно класифікованих прикладів серед усіх:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

де:

- TP – правильно передбачені позитивні приклади,
- TN – правильно передбачені негативні,
- FP – хибнопозитивні,
- FN – хибнонегативні.

Accuracy добре працює, коли класи збалансовані, але може вводити в оману при перекосі.

Precision (прецизійність / точність по класу)

Показує, яка частка передбачень класу є справді правильними:

$$Precision = \frac{TP}{TP + FP}$$

Це важливо у випадках, коли **небажано робити хибнопозитивні передбачення**.

Recall (повнота)

Показує, яку частку справжніх позитивних випадків модель змогла знайти:

$$Recall = \frac{TP}{TP + FN}$$

Це критично, коли важливо **не пропустити позитивні випадки.**

F1-score (F1-міра)

Середнє гармнонійне між precision і recall. Добре підходить для оцінки збалансованості між ними:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

F1-міра часто використовується як основна метрика в багатокласових задачах, особливо якщо класифікація незбалансована.

У проєкті використовуються всі перелічені метрики для тестової вибірки, а **F1-score** – як основна метрика, що відображає загальну ефективність моделі.

2.5 Комп'ютерний зір та обробка зображень у задачах класифікації сміття

Комп'ютерний зір (англ. *Computer Vision*) – це галузь штучного інтелекту, яка вивчає методи автоматичного аналізу та розпізнавання візуальної інформації зображень і відео. У контексті класифікації сміття комп'ютерний зір дозволяє:

1. **Отримувати дані** з різних джерел: цифрові камери, мобільні пристрої, стаціонарні сенсори.
2. **Попередньо обробляти зображення**: змінювати розмір, нормалізувати, усувати шум.
3. **Виділяти ознаки** об'єктів за допомогою CNN, що вчаться автоматично розпізнавати патерни.
4. **Класифікувати об'єкти** за категоріями (пластик, скло, папір, метал, органіка тощо).

Обробка зображень складається з кількох кроків:

- **Завантаження і масштабування** (resizing) до розміру, необхідного для мережі (наприклад, 100×100 пікселів).
- **Нормалізація** пікселів (перетворення значень у діапазон [0,1] або [-1, 1]).
- **Детекція контурів** (за потреби, наприклад, для виділення об'єкта з фону)

- **Аугментація** (як описано в 2.3.4) для підвищення різноманітності даних
- **Подача зображення в CNN** для отримання вектору ознак.
- **Класифікація** отриманого вектору за допомогою фінальних шарів мережі та softmax-функції.

Рисунок 2.5 – Схема обробки зображення в класифікаційній системі

В цьому проєкті використовуються датасети Garbage Dataset та TrashNet, які містять шість категорій сміття: папір, скло, пластик, метал, картон та інше. Кожне зображення проходить через описані етапи обробки, після чого CNN формує ймовірнісний розподіл по класах.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі розглянуто теоретичні основи машинного навчання та комп’ютерного зору, що лежать в основі розпізнавання зображень. Детально описано:

- загальні підходи до ML,
- структуру глибоких нейронних мереж та основні компоненти CNN,
- процес навчання моделі (функція втрат, оптимізатори, розбиття даних, аугментація),
- метрики оцінки якості класифікації,
- етапи обробки зображень у задачах сортування сміття.

Ці знання формують теоретичний фундамент для подальшої розробки та реалізації системи автоматичної класифікації відходів.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура рішення

3.1.1 Загальний опис архітектури

Для розробки системи автоматизованого сортування сміття було обрано архітектуру, яка складається з трьох основних компонентів:

Модуль машинного навчання (Модель CNN):

Модуль, що відповідає за класифікацію зображень сміття. Для цього використовується згорткова нейронна мережа (CNN), навчена на датасеті Garbage Dataset. Модель, яка проходить процес навчання на сервері, зберігається в файлі (наприклад, у форматі .zip або .bin), який потім завантажується для інференсу під час роботи серверної частини.

Серверна частина (Spring Boot API):

Це основний компонент, який забезпечує взаємодію між клієнтом (мобільний додаток або веб-клієнт) і системою класифікації. Сервер обробляє HTTP-запити, отримує зображення від користувача, передає їх до нейронної мережі для класифікації та повертає результат у вигляді категорії сміття.

- Сервер використовує фреймворк Spring Boot, що дозволяє швидко створювати Rest API для обміну даними.
- **DeepLearning4j** використовується для інференсу (обробки зображення за допомогою натренованної моделі).
- Зображення надсилаються на сервер через HTTP-запити (наприклад, POST-запит з файлами).

Клієнтська частина (мобільний додаток):

Мобільний додаток – це інтерфейс для користувача, який дозволяє завантажувати зображення сміття, відправляти їх на сервер для класифікації та отримувати рекомендації для сортування.

- **Android-додаток** побудований за допомогою Android Studio.

- Додаток здійснює запити до REST API через HTTP та отримує результат класифікації, після чого відображає його користувачеві.

3.1.2 Схема архітектури рішення

1. Мобільний додаток (Android)
    - Користувач вибирає/завантажує зображення.
    - Мобільний додаток відправляє зображення через HTTP-запит на сервер.
2. Серверна частина (Spring Boot API)
    - Отримує запит, обробляє зображення.
    - Завантажує модель CNN, виконує інференс.
    - Повертає результат класифікації у вигляді категорії сміття.
3. Модель CNN
    - Обробляє зображення та передбачає клас відходу (пластик, скло, метал, папір, картон, інше)

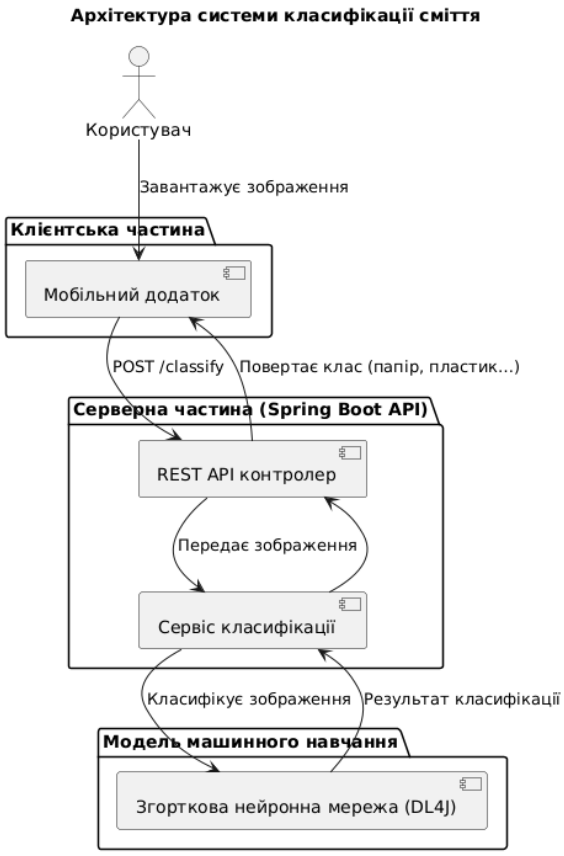


Рисунок 3.1 – Блок-схема архітектури рішення

3.1.3 Технології та переваги архітектури

У розробці системи використовуються такі основні технології:

- **DeepLearning4j** – для побудови згорткової нейронної мережі та класифікації зображень.
- **Spring Boot** – для реалізації серверної частини як REST API.
- **Android Studio** – для створення клієнтського МЗ.
- **HTTP / JSON** – для обміну даними між клієнтом та сервером.

Вибрана архітектура має низку переваг:

- **Модульність** – усі частини системи відокремлені, що спрощує обслуговування та розвиток.
- **Масштабованість** – можна легко додати нові категорії сміття або нові канали взаємодії (веб-клієнт, сенсорна система тощо).
- **Універсальність** – REST API дозволяє легко інтегрувати систему в інші проєкти або пристрої.
- **Готовність до розширення** – система легко адаптується до інших мов, середовищ або типів відходів.

3.2 Реалізація серверної частини

Серверна частина система реалізована за допомогою фреймворку **Spring Boot** із використанням мови програмування Java. Основна її мета – забезпечити взаємодію між клієнтом (мобільним або десктопним застосунком) та класифікаційною нейронною мережею, яка виконує розпізнавання зображень сміття.

3.2.1 Структура проєкту

Проєкт має типову структуру Spring Boot-застосунку та складається з таких основних пакетів:

- controller – приймає HTTP-запити, керує потоком даних.
- service – обробка логіки: завантаження зображення, передача до моделі.
- model – містить класи для представлення результатів.

- dto – містить класи для передачі даних через API.
- config – конфігурація нейронної мережі та завантаження моделі.
- util – утилітарні класи для конвертації зображень у формат, придатний для DL4J.

Рисунок 3.2 – Структура папок проєкту

3.2.2 REST API: опис і ендпоінти

Сервер надає один основний ендпоінт:

Метод	Шлях	Призначення
POST	api/v1/classify	Завантажити зображення і отримати клас сміття

Запит:

- Тип: multipart/form-data
- Поле: image (JPEG, PNG)

Відповідь:

```
{
  "class": "PLASTIC",
  "confidence": 0.91
}
```

3.2.3 Клас контролера

Клас `TrashClassificationController` розташовується в пакеті controller і є точкою входу для зовнішніх HTTP-запитів до системи класифікацій. Він виконує роль посередника між клієнтським застосунком та внутрішньою логікою сервісу.

Призначення:

- Приймає POST-запити з файлом зображення.
- Перевіряє неавяність і коректність файлу.
- Передає зображення на обробку в сервісний клас.
- Повертає клієнту результат класифікації у вигляді DTO.

Опис анотацій:

Анотація	Призначення
`@RestController`	Позначає клас як REST-контролер (Spring автоматично конвертує повенуті об’єкти у JSON).
`@RequestMapping ("api/v1/classify")`	Встановлює базовий маршрут для всіх методів цього контролера.
`@PostMapping`	Приймає POST-запит.
`@RequestParam("image")`	Вказує на параметр у запиті, яким є файл зображення.

Реалізація класу:

```
@RestController
@RequestMapping("api/v1/classify")
@AllArgsConstructor
public class TrashClassificationController {
    private final TrashClassificationService classificationService;

    @PostMapping
    public ResponseEntity<TrashClassificationResponse>
classifyImage(@RequestParam("image") MultipartFile imageFile) {
        if (imageFile.isEmpty()) {
            return ResponseEntity.badRequest().build();
        }

        TrashClassificationResponse result =
classificationService.classifyTrash(imageFile);
        return ResponseEntity.ok(result);
    }
}
```

Пояснення логіки:

1. Перевірка файлу: якщо файл порожній – повертається 400 Bad Request.
2. Передача до сервісу: через метод `classifyImage(...)` зображення надсилається в `TrashClassificationService`, де обробляється та передається до моделі.
3. Повернення результату: Сервіс у відповідь надсилає об'єкт `TrashClassificationResponse`, який Spring Boot автоматично серіалізує в JSON.

3.2.4 Клас сервісу `TrashClassificationService`

Клас `TrashClassificationService` є центральним компонентом бізнес-логіки системи. Він відповідає за обробку вхідного зображення, підготовку його до класифікації, виклик нейронної мережі та формування результату, що передається на рівень контролеру.

Основні задачі сервісу:

1. Зчитати зображення з `MultipartFile`.
2. Мастабуватит його до розміру, який очікує модель (наприклад, 100×100).
3. Застосувати нормалізацію значень пікселів (до діапазону [0, 1]).
4. Перетворити зображення у формат `INDArray` (формат DL4J).
5. Передати у модель та отримати масив ймовірностей.
6. Визначити клас з найбільшою ймовірністю.
7. Повернути результат як екземпляр DTO `TrashClassificationResponse`.

Опис анотацій:

Анотація	Призначення
`@Service`	Позначає клас як сервісний компонент, який обробляє логіку. Spring автоматично реєструє його як bean і дає змогу використовувати через ін'єкцію залежностей

`@RequiredArgsConstructor (Lombok)`	Генерує конструктор з всіма `final` полями, що спрощує ін'єкцію залежностей. Забезпечує чисту та безпечну ініціалізацію сервісу без явного конструктора.

Реалізація класу:

```
@Service
@RequiredArgsConstructor
public class TrashClassificationService {
    private final MultiLayerNetwork model;
    private final List<String> labels = Arrays.stream(TrashType.values())
            .map(Enum::name)
            .collect(Collectors.toList());
    public TrashClassificationResponse classifyTrash(MultipartFile
imageFile) {
        try {
            // Loading the image
            NativeImageLoader loader = new NativeImageLoader(100, 100, 3);
            INDArray image = loader.asMatrix(imageFile.getInputStream());

            // Pixels normalizing
            ImagePreProcessingScaler scaler = new
ImagePreProcessingScaler(0, 1);
            scaler.transform(image);

            // Inference
            INDArray output = model.output(image);
            int classIndex = Nd4j.argMax(output, 1).getInt(0);
            double confidence = output.getDouble(0, classIndex);
            String predictedClass = labels.get(classIndex);

            return new TrashClassificationResponse(predictedClass,
confidence);
        } catch (IOException e) {
            return new TrashClassificationResponse("ERROR", 0.0);
        }
    }
}
```

Пояснення до коду:

- `NativeImageLoader` – перетворює зображення у формат INDArray, що сумісний з DL4J.
- `ImagePreProcessingScaler` – масштабує значення пікселів (наприклад 0-255 → 0-1).

- `model.output(image)` – передає зображення у модель і повертає вектор ймовірностей.
- `Nd4j.argMax(...)` – визначає індекс класу з найвищою ймовірністю.
- `Arrays.of(...)` – список класів сміття, які відповідають вихідному шару моделі.

3.2.5 Клас DTO `TrashClassificationResponse`

Клас `TrashClassificationResponse` є об'єктом передачі даних (DTO), який використовується для **повернення результатів класифікації клієнту**. Він інкапсулює:

- Передбачений клас сміття.
- Ступінь впевненості моделі в передбаченні (від 0 до 1).

Призначення:

- Використовується контролером для серіалізації відповіді у формат JSON.
- Дає змогу клієнту легко обробляти результат.
- Стандартизує структуру всіх відповідей сервера на запити `/classify`.

Реалізація класу:

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TrashClassificationResponse {
    private String predictedClass;
    private double confidence;
}
```

Опис анотацій:

Анотація	Призначення
`@Data`	Генерує геттери, сеттери, `equals()`, `hasCode()` і `toString()` – зручно для DTO.
`@NoArgsConstructor`	Створює конструктор без аргументів.
`@AllArgsConstructor`	Створює конструктор з усіма полями.

Приклад JSON-відповіді на запит:

```
{
  "class": "GLASS",
  "confidence": 0.9147
}
```

Цей DTO-клас є важливим компонентом API, який дозволяє відокремити **внутрішню логіку (сервіс + модель)** від **зовнішнього формату відповіді**, що надсилається клієнтом.

3.2.6 Клас моделі `TrashType`

Клас `TrashType` є переліком (`enum`), який описує **підтримувані типи сміття**, що класифікуються системою. Він виконує роль **єдиного джерела істини** для класифікації в усій серверній частині: від моделі до відповіді клієнту.

Призначення:

- Забезпечує **чіткий перелік категорій**, які підтримуються системою.
- Дозволяє уникнути «магічних рядків» у коді.
- Служить як **модель** для об’єднання логіки інференсу та відповіді.

Реалізація класу:

```
public enum TrashType {
    PLASTIC,
    METAL,
    GLASS,
    PAPER,
    CARDBOARD,
    OTHER
}
```

Пояснення до коду:

- Кожне значення `enum` відповідає **одному класу з датасету TrashNet**.
- Значення OTHER використовується як загальний клас для об’єктів, які не підпадають під основні категорії.
- Усі значення використовуються для:
    - Формування відповіді.
    - Визначення класу моделі.
    - Прив’язки до індексу в масиві ймовірностей (`output.getDouble(...)`).

Переваги такого підходу:

- Покращується **читабельність** та **підтримуваність** коду.
- Можна легко **додавати нові категорії** в майбутньому.
- Відповідає принципам **типобезпеки** та **уніфікації предметної області**.

Усі логічні шари системи – сервіс, контролер, DTO – використовують значення `TrashType`, що гарантує узгодженість результатів класифікації.

3.2.7 Конфігурація моделі (`ModelConfig`)

Клас `ModelConfig` відповідає за завантаженням попередньо натренованої моделі нейронної мережі при старті застосунку. Він надає готовий об'єкт `MultiLayerNetwork`, який буде ін'єктовано в `TrashClassificationService` для виконання інференсу (класифікації).

Призначення:

- Завантажити модель з локального файлу.
- Сконфігурувати її перед використанням.
- Зареєструвати об'єкт MultiLayerNetwork як Spring Bean для подальшої ін'єкції залежностей.

Реалізація класу:

```
@Configuration
public class ModelConfig {
    @Bean
    public MultiLayerNetwork trashClassificationModel() throws IOException {
        File modelFile = new File("model/model.zip");
        MultiLayerNetwork model =
ModelSerializer.restoreMultiLayerNetwork(modelFile);
        model.init();

        return model;
    }
}
```

Опис анотацій:

Анотація	Призначення
`@Configuration`	Призначає клас як конфігураційний, який містить методи для створення Spring Bean'ів.

`@Bean`	Позначає метод, що створює **bean**, який буде керований Spring-контейнером.

Пояснення логіки роботи:

- DL4J-серіалізована модель зберігається у файлі після навчання.
- При старті застосунку Spring викликає метод `trashClassificationModel()` та завнтажує її у пам’ять.
- Сервіс `TrashClassificationService` отримує цю модель через автоматичну ін’єкцію.

Така конфігурація дозволяє централізовано контролювати модель, не дублювати логіку в кожному сервісу та легко змінювати її при оновленні.

3.3 Розробка згорткової нейронної мережі для класифікації сміття

У цьому підрозділі описується процес розробки згорткової нейронної мережі (CNN) для класифікації зображень сміття, зокрема для розпізнавання типів відходів на основі зображень. Мережа була навчена на основі Garbage Dataset – відкритого датасету, що містить зображення різних категорій сміття.

Модуль був реалізований із використанням DeepLearning4j (DL4J) – фреймворку для машинного навчання на платформі Java, який дозволяє будуватит, тренувати та інтерпретувати нейронні мережі.

3.3.1 Вибір і підготовка навчального датасету

Для навчання моделі було використано датасет Garbage Dataset, який складається з шести категорій сміття:

Таблиця 3.1 – Кількість зображень кожного класу в датасеті

№	Клас	Кількість зображень
1	GLASS	3061
2	PAPER	1680
3	CARDBOARD	1825

4	PLASTIC	1984
5	METAL	1020
6	TRASH (Інше)	947

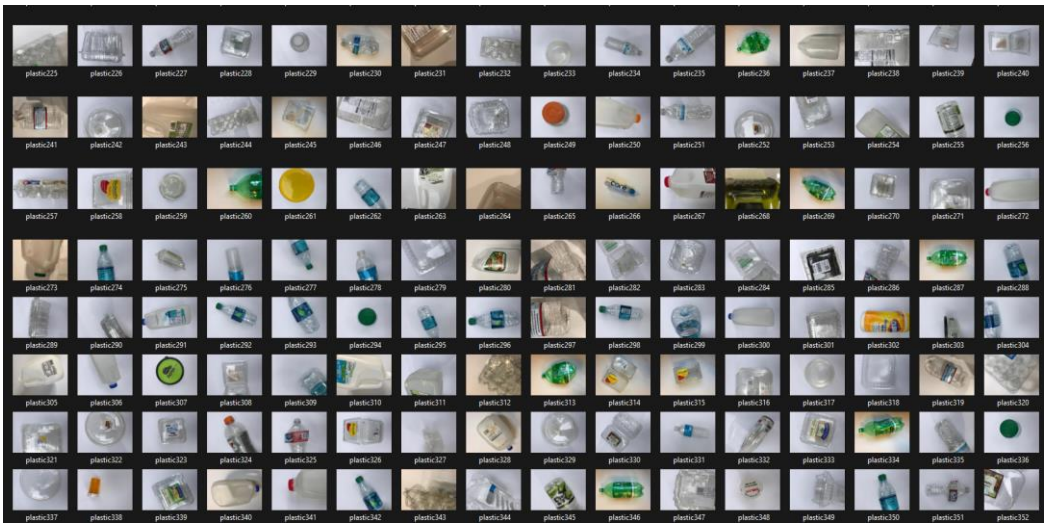


Рисунок 3.3 – Приклад структури даних у папці `plastic`

3.3.2 Аугментація зображень

Зважаючи на нерівномірну кількість зображень у кожному класі та загальний обсяг датасету, було прийнято рішення застосувати аугментацію зображень. Це дозволяє розширити навчальну вибірку та зменшити ймовірність перенавчання моделі.

Мета аугментації:

- Збалансувати кількість зображень у класах.
- Надати моделі більшу варіативність даних.
- Покращити здатність до узагальнення на нових прикладах.

Застосовані методи аугментації:

- Обертання зображення;
- Горизонтальне й вертикальне віддзеркалення;
- Масштабування за зсув;
- Зміна яскравості / контрасту;
- Деформація;

Реалізація в DeepLearning4j:

У DL4J аугментація роеалізується за допомогою класів `ImageTransform` з бібліотеки `DataVec`. Вони застосовуються до потоку даних у момент зчитування:

```
List<ImageTransform> transforms = Arrays.asList(
        new FlipImageTransform(1),            // горизонтальне
віддзеркалення
        new RotateImageTransform(15),         // обертання на 15°
        new WarpImageTransform(10),           // деформація
        new ScaleImageTransform(0.9f),        // масштабування
        new ColorConversionTransform(COLOR_BGR2HSV) // зміна простору
кольору
);
```

Такий підхід прибирає потребу зберігати згенеровані зображення на диск, а також підвищує гнучкість при розширенні стратегії навчання.

3.3.3 Вибір архітектури моделі

Для розв'язання задачі класифікації сміття було вирішено використовувати **згорткову нейронну мережу (Convolutional Neural Network, CNN**, оскільки вона є найефективнішою архітектурою для задач комп'ютерного зору, зокрема для розпізнавання об'єктів на зображеннях.

Таблиця 3.2 – Архітектура побудованої моделі

Шар	Тип	Параметри
Вхідний	Input	100×100×3 (RGB зображення)
Згортковий	Convolution2D	32 фільтри, 3×3, активація ReLU
Пулінговий	MaxPooling2D	2×2
Згортковий	Convolution2D	64 фільтри, 3×3, активація ReLU
Пулінговий	MaxPooling2D	2×2
Повнозв'язний	Dense	128 нейронів, активація ReLU
Вихідний	Dense	6 нейронів (по одному на клас), Softmax

Особливості реалізації:

- Активаційна функція ReLU (Rectified Linear Utin) використовується у всіх прихованих шарах для прискорення збіжності.
- Вихідний шар використовує Softmax, що дозволяє моделі повертати ймовірності належності до кожного 6 класів.

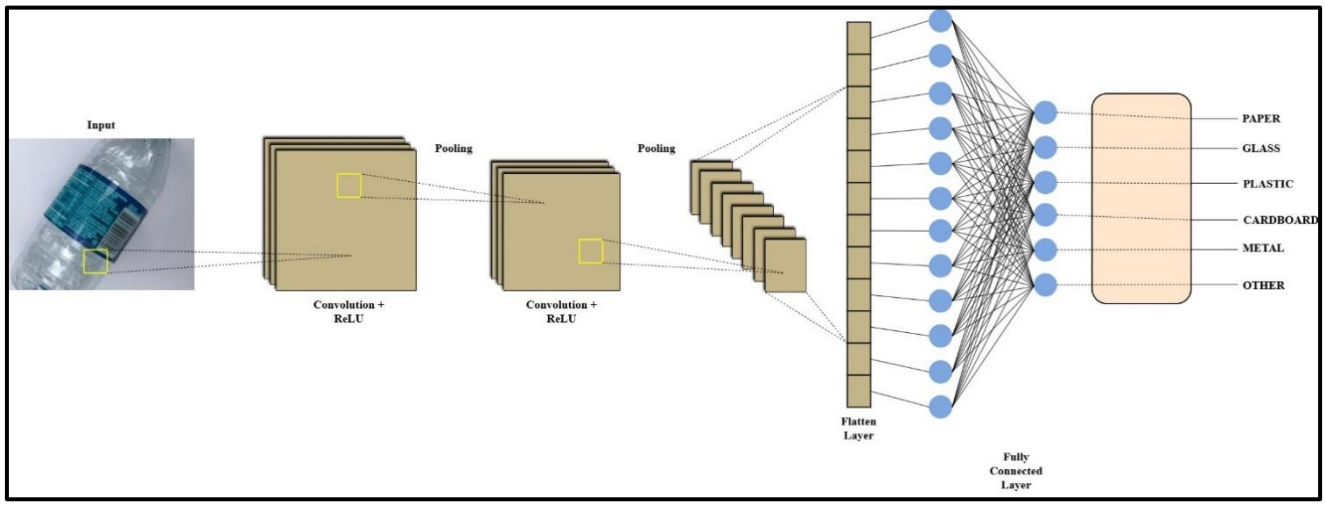


Рисунок 3.4 – Спрощена структура розробленої CNN

Опис функціональних ролей кожного шару моделі:

Кожен з шарів згорткової нейронної мережі виконує специфічну функцію в процесі обробки зображення.

Зокрема, **згорткові шари (Convolutional Layers)** відповідають за виявлення локальних ознак на зображеннях, таких як краї, текстури або прості геометричні фрагменти. Ці шари застосовують фільтри, які навчаються в процесі тренування й автоматично виділяють ревелантні ознаки об’єктів.

Пулінгові шари (MaxPooling Layers) зменшують просторову розмірність тензорів, що дозволяє зменшити кіулькість параметрів та підвищити узагальнюючу здатність мережі. Вони також сприяють інваріативності до незначних зсувів об’єктів на зображеннях.

Нарешті, **вихідний шар з функцією активації Softmax** перетворює результати обчислень на набір ймовірностей належності до кожного з класів: plastic, metal, glass, paper, cardboard, trash.

Таким чином, обрана архітектура забезпечує поетапну трансформацію вхідного зображення у вектор високорівневих ознак, які дозволяють моделі ефективно виконувати класифікацію.

3.3.4 Навчання моделі

Відповідно до описаної архітектурі моделі за допомогою фреймворку DeepLearning4j було реалізовано повний цикл навчання. Після повнозв'язного шару було додано Dropout-шар із коефіцієнтом 0.5 для уникнення перенавчання.

Реалізація класу `TrainModel`:

```
public class TrainModel {

    public static void main(String[] args) throws Exception {
        int height = 100;
        int width = 100;
        int channels = 3;
        int batchSize = 32;
        int numClasses = 6;
        int numEpochs = 20;
        long seed = 123;

        String dataPath = "src/main/resources/training_dataset";
        File datasetDir = new File(dataPath);
        ImageRecordReader recordReader = new ImageRecordReader(height,
width, channels, new ParentPathLabelGenerator());

        recordReader.initialize(new FileSplit(datasetDir, new
Random(seed)));
        ImageTransform transform = new MultiImageTransform(
                new FlipImageTransform(1),
                new RotateImageTransform(15)
        );
        DataSetIterator dataIter = new RecordReaderDataSetIterator(
                recordReader, batchSize, 1, numClasses);
        dataIter.setPreProcessor(new ImagePreProcessingScaler(0, 1));

        MultiLayerConfiguration config = new
NeuralNetConfiguration.Builder()
                .seed(seed)
                .updater(new Adam(0.0001))
                .weightInit(WeightInit.XAVIER)
                .list()
                .layer(new ConvolutionLayer.Builder()
                        .kernelSize(3, 3).stride(1,
1).nIn(channels).nOut(32)
                        .activation(Activation.RELU).build())
                .layer(new BatchNormalization())
                .layer(new SubsamplingLayer.Builder()
                        .kernelSize(2,
2).poolingType(PoolingType.MAX).build())
```

```
                .layer(new ConvolutionLayer.Builder()
                        .kernelSize(3, 3).stride(1, 1).nOut(64)
                        .activation(Activation.RELU).build())
                .layer(new BatchNormalization())
                .layer(new SubsamplingLayer.Builder()
                        .kernelSize(2,
2).poolingType(PoolingType.MAX).build())
                .layer(new
DenseLayer.Builder().nOut(128).activation(Activation.RELU).build())
                .layer(new DropoutLayer(0.5))
                .layer(new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                        .nOut(numClasses)
                       .activation(Activation.SOFTMAX)
                        .build())
                .setInputType(InputType.convolutional(height, width,
channels))
                .build();

        MultiLayerNetwork model = new MultiLayerNetwork(config);
        model.init();
        model.setListeners(new ScoreIterationListener(10));

        System.out.println("Початок навчання...");
        for (int i = 0; i < numEpochs; i++) {
            model.fit(dataIter);
            dataIter.reset();
            System.out.println("Епоха " + (i + 1) + " завершена");
        }

        Evaluation eval = model.evaluate(dataIter);
        System.out.println(eval.stats());

        File modelFile = new File("model.zip");
        ModelSerializer.writeModel(model, modelFile, true);
        System.out.println("Модель збережена у файл: " +
modelFile.getAbsolutePath());
    }
}
```

Перед навчанням було здійснено нормалізацію вхідних даних до діапазону [0,1]. Навчання моделі тривало 20 епох, після кожної з яких проводилось обнулення ітератора. У якості відображення прогресу було реалізовано вивдення `score` кожні 10 ітерацій.

Оцінка моделі:

Модель показала дуже високу точність на навчальному наборі даних. Підсумкова метрика оцінювання:

- Accuracy (точність): 0.9947
- Precision (прецизійність): 0.9950

- Recall (середня повнота): 0.9957
- F1-score (середня F1-міра): 0.9953

Таблиця 3.3 – Матриця неточностей під час навчання моделі

Фактичний/Передбачений	cardboard	glass	metal	paper	plastic	trash
cardboard	1820	2	0	0	3	0
glass	1	3027	0	11	21	1
metal	2	1	1017	0	1	0
paper	0	0	0	1677	3	0
plastic	3	3	0	0	1976	2
trash	0	0	0	1	2	944

Ці результати свідчать про ефективне навчання моделі та правильну класифікацію об’єктів усіх шести категорій.

Після завершення тренування модель було збережно у файл `model.zip` та правильну класифікацію об’єктів усіх шести категорій.

3.3.5 Тестування моделі

Після завершення навчання нейронної мережі необхідно оцінити її здатність узагальнювати знання на нових, раніше невідомих даних. Це досягається шляхом тестування моделі на окремому наборі зображень, які не використовувалися під час навчання.

Для об’єктивної оцінки моделі було використано окремий датасет TrashNet, який теж містить фотографії сміття цих шести категорій.

Реалізація класу `TestModel`:

```
public class TestModel {
    public static void main(String[] args) throws Exception {
        int height = 100;
        int width = 100;
        int channels = 3;
        int batchSize = 32;
        int numClasses = 6;
        long seed = 123;
```

```
        File modelFile = new File("model.zip");
        MultiLayerNetwork model =
ModelSerializer.restoreMultiLayerNetwork(modelFile);

        File testData = new
File("src/main/resources/testing_dataset");
        FileSplit testSplit = new FileSplit(testData,
NativeImageLoader.ALLOWED_FORMATS, new Random(seed));
        ParentPathLabelGenerator labelMaker = new
ParentPathLabelGenerator();
        ImageRecordReader testRecordReader = new
ImageRecordReader(height, width, channels, labelMaker);
        testRecordReader.initialize(testSplit);
        DataSetIterator testIter = new
RecordReaderDataSetIterator(testRecordReader, batchSize, 1,
numClasses);
        testIter.setPreProcessor(new ImagePreProcessingScaler(0, 1));

        Evaluation eval = model.evaluate(testIter);
        System.out.println(eval.stats());

        System.out.println("Матриця плутанини:");
        System.out.println(eval.getConfusionMatrix().toCSV());
    }
}
```

Метрики оцінювання

Результати тесування моделі представлені за допомогою наступних метрик:

- Accuracy (точність): 0.9877
- Precision (прецизійність): 0.9896
- Recall (середня повнота): 0.9878
- F1-score (середня F1-міра): 0.9886

Ці метрики обчисленні за макро-усередненням, тобто середнє значення метрик для кожного класу без урахування їх частоти у вибірці. Такий підхід дозволяє оцінити ефективність моделі на кожному класі окремо, що особливо важливо при наявності дисбалансу в даних.

Матриця неточностей:

Матриця неточностей демонструє кількість правильних та неправильних передбачень моделі для кожного класу:

Таблиця 3.4 – Матриця неточностей під час тестування моделі

Фактичний/Передбачений	cardboard	glass	metal	paper	plastic	trash
cardboard	402	1	0	0	0	0
glass	0	481	1	1	18	0
metal	1	0	408	0	1	0
paper	0	0	0	591	3	0
plastic	0	3	0	0	479	0
trash	0	1	0	1	0	135

3.3.6 Аналіз результатів тестування моделі

З матриці видно, що модель демонструє високу точність класифікації для всіх класів. Найбільша кількість помилок спостерігається при класифікації скла та пластику, де деякі зразки скла були помилково класифіковані як пластик. Це пов’язано з візуальною схожістю між цими матеріалами. Також, важливо підмітити, що модель не плутає метал та скло, адже цим двом матеріалам властивий блиск, що часто виникає проблеми в моделі при класифікації.

Отримані результати свідчать про високу ефективність моделі в задачі багатокласофої класифікації відходів. Високі значення точності, прецизійності, повноти та F1-міри вказують на здатність моделі точно розпізнавати різні категорії об’єктів. Матриця неточностей дозволяє ідентифікувати конкретні класи, де модель має потенціал для покращення, що може бути корисним для подальшого вдосконалення моделі.

3.4 Розробка клієнтського мобільного застосунку

3.4.1 Загальна ідея і функціонал

Для демонстрації можливостей побудованої системи класифікації сміття було розроблено клієнтський мобільний застосунок для операційної системи Android. Метою цього застосунку є перевірка працездатності REST API сервера та демонстрація принципу взаємодії між серверною частиною та клієнтськими додатками.

Основні фукнціональні можливості застосунку:

- Захоплення зображень за допомогою камери пристрою.
- Вибір зображень із локальної галереї.
- Надсилання зображень на сервер через API-інтерфейс.
- Отримання результатів класифікації та їх виведення на екран.

Таким чином, мобільний застосунок є прототипом і може бути використаний як основа для подальшої розробки повноцінного застосунку. Він дозволяє перевірити правильність роботи серверного API, наявність взаємодії між клієнтом та сервером, а також отримати первинний досвід роботи з інтеграцією моделі машинного навчання у зовнішні системи.

3.4.2 Розробка графічного інтерфейсу користувача

Графічний інтерфейс користувача (GUI) є важливим компонентом клієнтської частини системи, що забезпечує інтуїтивно зрозумілу та зручну взаємодію користувача із системою класифікації сміття.

Основна мета інтерфейсу – забезпечити користувача можливістю завантажувати фотографії відходів для їх класифікації та переглядати отримані результати у зручній формі.

Основні елементи інтерфейсу:

- **PreviewView** – вікно попереднього перегляду з камери.
- **Кнопка «Зробити фото» -** для захоплення зображень через камеру.

- **Кнопка «Вибрати фото з галереї»** - для вибору із локального середовища.
- **Текстове поле результату** – для відображення результатів класифікації.

Інтерфейс було реалізовано з використанням бібліотеки **CameraX** для роботи з камерою та **Retrofit** для відправки HTTP-запитів. На рис. 2.9 показано макет графічного інтерфейсу користувача.

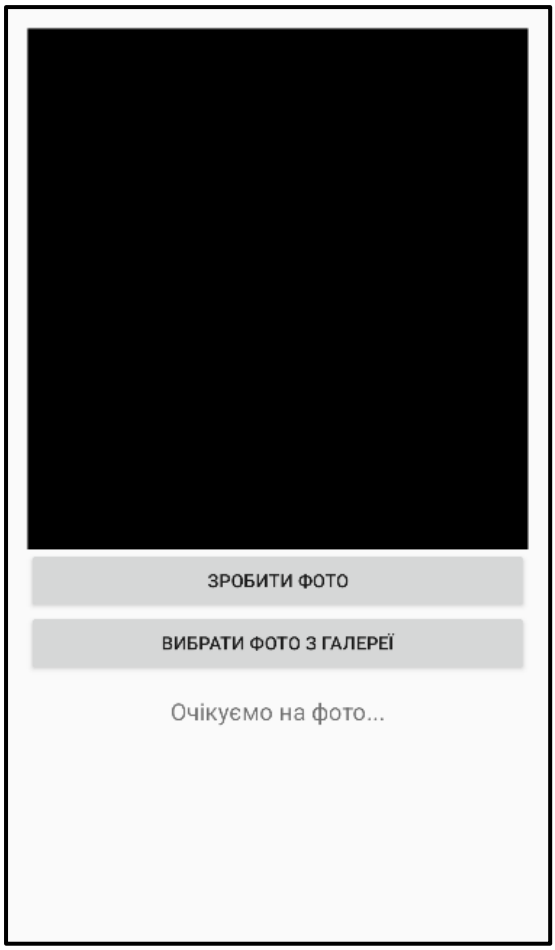


Рисунок 3.5 – Макет мобільного застосунку

3.4.3 Логіка роботи клієнтського застосунку та його аріхтектура

Клієнтський мобільний застосунок є важливим компонентом системи класифікації сміття. Його основна мета – надати користувачеві простий, зручний і ефективний інтерфейс для взаємодії із серверною частиною системи. Завдяки застосунку користувачможе завантажувати фотографії об’єктів для класифікації, отримувати результати обробки та переглядати їх на мобільному пристрої.

Основні завдання, які виконує застосунок:

- **Отримання зображень для класифікації** – через камеру пристрою або завантаження з галереї.
- **Формування запиту дл серверного API** у форматі multipart/form-data, що дозволяє передати зображення для обробки.
- **Отримання та обробка відповіді сервера**, яка містить інформацію про тип відходу та рівень впевненості моделі.
- **Виведення результату класифікації на екран** у зрозумілій формі.
- **Обробка помилок –** наприклад, якщо сервер недоступний або виникла помилка мережі, користувач отримує відповідне повідомлення.

Розробка клієнтської частини здійснювалась за допомогою **Kotlin** – офіційної мови програмування для Android, а також низки бібліотек і фреймворків:

- **Retrofit** – для взаємодії з REST API сервера.
- **CameraX** – для роботи з камерою пристрою.
- **OkHttp** – для обробки HTTP-запитів.
- **Android Jetpack Components** – для спрощення роботи з життєвим циклом застосунку та доступом до ресурсів пристрою.

Архітектура клієнтської частини:

Клієнтський застосунок має модульну архітектуру, де кожен компонент відповідає за певний функціонал:

- MainActivity.kt – головний екран застосунку, що обробляє взаємодію з користувачем.
- ApiService.kt – інтерфейс для опису API-запитів до серверної частини.
- ClassificationResponse.kt – модель для отримання результату класифікації.
- CameraHelper.kt – допоміжний клас для роботи з камерою.
- activity_main.xml – XML-макет графічного інтерфейсу користувача.

Логіка роботи застосунку:

Після запуску застосунок запитує дозвіл на використання камери та доступ до зовнішнього сховища для отримання фото. Якщо дозвіл надано, камера активується, і користувач бачить попередній перегляд зображення в елементі PreviewView. Для зйомки фото достатньо настиснути кнопку «Зробити фото». Фото автоматично зберігається в тимчасовому сховищі та передається на сервер для класифікації.

Альтернативно користувач може завантажити фото з галереї. Для цього у застосунку реалізована кнопка «Вибрати фото з галереї», яка відкриває стандартний файловий діалог для вибору зображення.

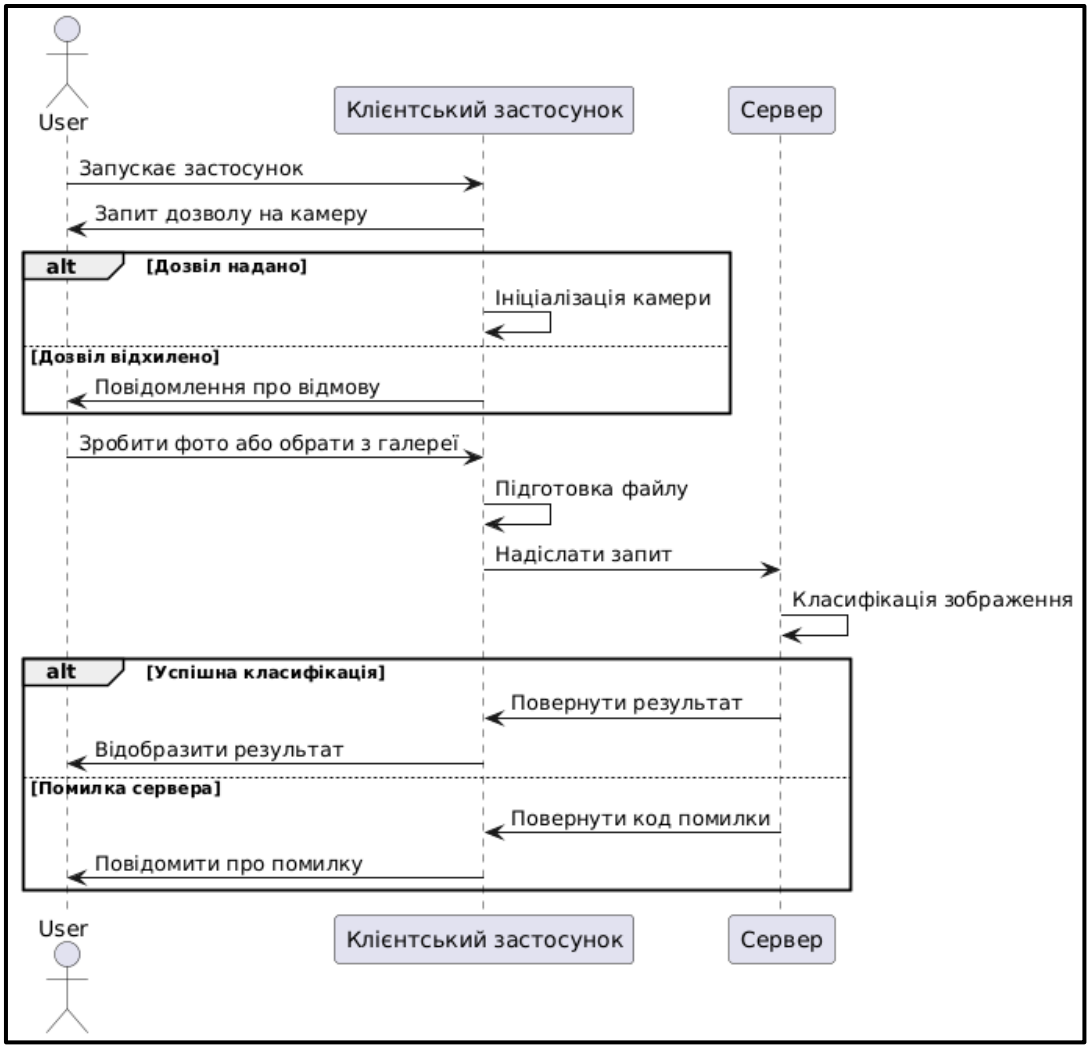


Рисунок 3.6 – Діаграма послідовності роботи клієнтського застосунку з сервером

Надсилання запиту на сервер:

Після вибору фото з галереї або захоплення з камери застосунок формує HTTP-запит у форматі **multipart/form-data**, який містить файл зображення. Запит надсилається за допомогою бібліотеки Retrofit, а серверна частина повертає результат класифікації у форматі JSON:

```
{
"predictedClass": "PLASTIC",
"confidence": 0.87
}
```

Цей результат обробляється застосуком: англійська назва класу мапується на українську назву (наприклад, «PLASTIC» → «Пластик»), і виводиться на екран.

3.4.4 Реалізація модулю роботи з камерою:

Однією з ключових функцій клієнтського застосунку є можливість робити фотографії відходів для їх подальшої класифікації. Для цього використовується модуль CameraX, що дозволяє інтегрувати камеру пристрою без необхідності низькорівневої роботи з API.

При ініціалізації камери викликається метод `startCamera()`, який створює екземпляр об'єкта `ImageCapture` для захоплення зображень. Далі за допомогою класу `CameraHelper` налаштовується попередній перегляд камери (`PreviewView`) та прив'язка життєвого циклу камери до активності користувача.

Реалізація методу `startCamera()`:

```
private fun startCamera() {
imageCapture = ImageCapture.Builder().build()
CameraHelper.startCamera(this, previewView, imageCapture,
cameraExecutor)
}
```

Коли користувач натискає кнопку захоплення фотографії, викликається метод `takePhoto()`. Він створює унікальний файл для збереження фотографії, а згодом налаштовується об'єкт `OutputFileOptions`, що вказує на шлях збереження фото, після чого запускається процес захоплення зображення.

У разі успішного збереження файлу зображення автоматично викликається метод `uploadImage()`, що надсилає HTTP-запит із зображенням для класифікації.

Якщо під час захоплення зображення виникає помилка, вона обробляється в методі `onError()` і відповідне повідомлення відображається користувачу.

Реалізація методу `takePhoto()`:

```
private fun takePhoto() {
    val photoFile = File(
        externalCacheDir,
        SimpleDateFormat("yyyy-MM-dd-HH-mm-ss-SSS", Locale.US)
            .format(System.currentTimeMillis()) + ".jpg"
    )

    val outputOptions =
ImageCapture.OutputFileOptions.Builder(photoFile).build()

    imageCapture.takePicture(
        outputOptions,
        ContextCompat.getMainExecutor(this),
        object : ImageCapture.OnImageSavedCallback {
            override fun onError(exc: ImageCaptureException) {
                resultText.text = "Помилка зйомки: ${exc.message}"
            }

            override fun onImageSaved(output:
ImageCapture.OutputFileResults) {
                resultText.text = "Фото зроблено. Відправляємо..."
                uploadImage(photoFile)
            }
        }
    )
}
```

3.4.5 Реалізація модулю комунікації з REST API

Пісял захоплення або вибору фото користувачем, клієнтський застосунок автоматично здійснює надсилання цього зображення на сервер для класифікації. Для цього використовується HTTP POST-запит із типом `multipart/form-data`, який дозволяє передавати файли у тілі запиту. Реалізація цієї функції здійснюється за допомогою бібліотеки Retrofit, що є популярним інструментом для роботи з HTTP-запитами у середовищі Android.

Основні кроки надсилання фото на сервер:

- Файл зображення конвертується у об’єкт `RequestBody`, в якому вказується MIME-тип файлу.
- Далі створюється об’єкт `MultipartBody.Part`, що містить ім’я файлу, його вміст та метадані.

- Конфігурується екземпляр Retrofit, в якому задається базова URL-адреса серверу, клієнт (`OkHttpClient`) та конвертер для роботи з JSON.
- Створюється інтерфейс `ApiService`, який визначає метод `uploadImage()` для виконання POST-запиту на сервер.

Після виклику методу `uploadImage()`, зображення надсилається на сервер, де відбувається його обробка та класифікація за допомогою попередньо натренованої моделі машинного навчання. У випадку успішної відповіді севрер повертає об'єкт `ClassificationResponse`, який містить назву передбаченого класу відходів та рівень впевненості моделі. Після цього результат відображаєтсья користувачу на екрані застосунку.

У випадку помилки (наприклад, якщо сервер недоступний або запит некоректний), користувачу виводиться повідомлення про помилку.

Реалізація методу `uploadImage()`:

```
private fun uploadImage(imageFile: File) {
    val requestFile = RequestBody.create("image/*".toMediaTypeOrNull(),
imageFile)
    val body = MultipartBody.Part.createFormData("image",
imageFile.name, requestFile)

    val retrofit = Retrofit.Builder()
        .baseUrl("https://huge-lizard-illegally.ngrok-free.app/")
        .client(OkHttpClient())
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val service = retrofit.create(ApiService::class.java)

    service.uploadImage(body).enqueue(object :
retrofit2.Callback<ClassificationResponse> {
        override fun onResponse(
            call: Call<ClassificationResponse>,
            response: retrofit2.Response<ClassificationResponse>
        ) {
            if (response.isSuccessful) {
                val predictedClass = response.body()?.predictedClass ?:
"Невідомо"
                val translatedClass = trashTypeMap[predictedClass] ?:
"Невідомо"
                resultText.text = "Результат: $translatedClass"
            } else {
                resultText.text = "Помилка: ${response.code()}"
            }
        }
```

```
override fun onFailure(call: Call<ClassificationResponse>, t:
Throwable) {
            resultText.text = "Помилка запиту: ${t.message}"
        }
    })
}
```

3.4.6 Тестування клієнтського застосунку

Для перевірки працездатності клієнтського застосунку було проведено ручне тестування основних сценаріїв його використання. Метою тестування було переконатися, що затосунок коректно взаємодіє з серверною частиною через REST API, забезпечує обробку зображень з камери або галереї та виводить результати класифікації на екран.

Під час тестування було перевірено наступні функції:

- Отримання доступу до камери пристрою на здійснення фото.
- Вибір зображення з галереї.
- Надсилання отриманого зображення на сервер для класифікації.
- Обробка відповіді від сервера та відображення результату класифікації.
- Обробка помилок мережі або некоректної відповіді від сервера.

Таблиця 3.5 – Сценарії для тестування клієнтського застосунку

№	Назва класу	Вхідні дані	Очікуваних результат	Фактичний результат
1	Захоплення фото	Фото зроблено камерою	Зображення збережено, відправлено на сервер, отримано результат класифікації	Тест пройдено
2	Вибір фото з галереї	Обране фото	Зображення надіслано на сервер, отримано результат класифікації	Тест пройдено
3	Відмова в доступі до камери	Відхилено дозвіл	Повідомлення про помилку	Тест пройдено
4	Відсутність інтернет-з’єднання	Немає мережі	Повідомлення про помилку	Тест пройдено
5	Вибір некоректного файлу	Некоректний файл (не фото)	Повідомлення про помилку	Тест пройдено

Результати тестування свідчать про стабільну роботу клієнтського застосунку в типових сценаріях використання. Додатково, при необхідності, функционал можна розширити для обробки додаткових типів помилок або покращення інтерфейсу користувача.

На основі реалізованого прототипу проведемо дослідження та аналіз його роботи, а також оцінку ефективності системи для різних категорій відходів. Це дозволить визначити сильні та слабкі сторони розробленої системи, виявити потенційні шляхи для її подальшого вдосконалення та підвищення якості класифікації сміття.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було проведено комплексну розробку програмного рішення для задачі класифікації сміття та екологічного сортування. Описано архітектуру системи, що включає серверну частину з REST API, модель машинного навчання на основні згорткової нейронної мережі (CNN), а також клієнтський мобільний застосунок для операційної системи Android.

Було реалізовано модель класифікації зображень, проведено навчання на підготовленому датасеті та отримано високу точність класифікації на тестових даних. Розроблений REST API дозволяє інтегрувати модель у сторонні рішення та забезпечує гнучку взаємодію між сервером та клієнтом.

Клієнтський застолсунок реалізує основні функції: захоплення зображень за допомогою камери або вибір з галереї, відправка зображень на сервер для класифікації, отримання та відображення результатів. Особливу увагу було приділено релазіації взаємодії клієнт-сервер через REST API за допомогою бібліотеки Retrofit.

У межах роботи проведено тестування окремих компонентів та системи в цілому: перевірено коректність роботи моделі, REST API та клієнтської частини. Результати тестування підтвердили працездатність системи та її здатність викоувати завдання класифікації сміття з високою точністю.

Розроблено система є прототипом, який може бути використаний для подальшого вдосконалення: додавання нових функцій, інтеграція з іншими платформами, оптимізація продуктивності.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Опис тестових сценаріїв та умов тестування

Для перевірки працездатності розробленої системи було проведено комплексне тестування всіх її компонентів: серверної частини (REST API), моделі класифікації зображень на основі CNN, а також клієнтського мобільного застосунку для Android.

Тестування проводилося в умовах, наближених до реального використання, з використанням власного окремого датасету, який не застосовувався для навчання моделі з шістьма категоріями відходів:

- Картон (CARDBOARD)
- Скло (GLASS)
- Метал (METAL)
- Папір (PAPER)
- Пластик (PLASTIC)
- Інше сміття (TRASH)

Середовище тестування:

- Операційна система: Windows 11
- IDE: IntelliJ IDEA 2025
- Мова програмування: Java + Kotlin (Android)
- Тестовий пристрій: Android-емулятор API 31 (Pixel 5), фізичний пристрій Samsung Galaxy Note 10 Lite N770F
- Серверна частина: локальний сервер з використанням ngrok для створення тунелю до зовнішньої статичної url адреси.

Основні тестові сценарії:

1. Первірка запуску серверного API та доступності ендпоїнтів.
2. Перевірка завантаження навченої моделі та її використання для класифікації.

3. Надсилання зображення з клієнтського застосунку на сервер для класифікації.
4. Отримання результату класифікації та правильність відображення категорії.
5. Обробка помилкових ситуацій
    - Відсутність інтернет з’єднання
    - Відправка некоректного файлу (не зображення)
    - Спроба запиту без дозволів на камеру/галерею
6. Перевірка швидкодії застосунку при обробці серії зображень
7. Вимірювання часу відповіді сервера на запит класифікації
8. Аналіз матриці неточностей та показників точності (Accuracy, Presicion, Recall, F1-score)

Тестові дані:

Для тестування було використано два набори даних:

1. Датасет з ~2500 тестових зображень всіх класів, який не використовувався при тренуванні моделі
2. Реальні фотографії, зроблені за допомогою мобільного пристрою (для перевірки інтеграції)

4.2 Результати тестування

За результатами проведеного тестування було отримано такі дані, що підтверджують ефективність розробленої системи для класифікації сміття.

4.2.1 Основні метрики моделі

Використовуючи підготовлений тестовий датасет, було отримано такі показники якості моделі:

- Точність (Accuracy): 98.77%
- Прецизійність (Precision): 98.96%
- Повнота (Recall): 98.78%
- F1-міра: 98.86%

Ці значення підтверджують високий рівень здатності класифікувати тверді відходи моделі на нових, раніше невідомих даних.

4.2.2 Матриця плутанини

Результати класифікації візуалізовані за допомогою матриці плутанини (confusion matrix).

Таблиця 4.1 – Матриця неточностей під час тестування моделі

Фактичний/Передбачений	cardboard	glass	metal	paper	plastic	trash
cardboard	402	1	0	0	0	0
glass	0	481	1	1	18	0
metal	1	0	408	0	1	0
paper	0	0	0	591	3	0
plastic	0	3	0	0	479	0
trash	0	1	0	1	0	135

4.2.3 Аналіз результатів

З наведених даних видно, що найбільша кількість помилок спостерігається при класифікації категорій «скло» та «пластик», що пов’язано з візуальною схожістю деяких зразків.

Однак, загальні результати вказують на високу ефективність моделі, що дозволяє використовувати її для автоматичної класифікації відходів.

4.2.4 Перевірка клієнтської частини

Під час тестування клієнтського мобільного застосунку було перевірено такі сценарії:

- Захоплення фото з камери та його відправка на сервер – працює коректно
- Вибір фото з галереї та його відправка – працює коректно

- Обробка помилок: при відсутності інтернет-з’єднання або відповіді від сервера користувач отримує повідомлення про помилку
- Час обробки одного запиту (відправка зображення, отримання відповіді) – в середньому **2-4 секунди**.

4.2.5 Висновки за результатами тестування

Розроблена система продемонструвала стабільну та надійну роботу в умовах, наближених до реального використання.

Система дозволяє ефективно класифікувати відходи за категоріями, а також інтегрувати модель у сторонні сервіси за допомогою REST API.

Мобільний застосунок, як клієнтська частина, успішно забезпечує взаємодію користувача із системою, дозволяючи надсилати зображення та отримувати результати класифікацій.

Результати тестування підтверджують, що розроблена система готова для використання як прототип або основа для подальшого розширення та вдосконалення.

4.3 Аналіз роботи системи та можливості її вдосконалення

4.3.1 Аналіз роботи системи

У процесі розробки та тестування системи класифікації сміття було отримано ряд результатів, що дозволяють оцінити її функціональність та потенціал розвитку.

Система складається з трьох основних компонентів:

- **Серверна частина**, яка реалізує REST API та обробляє запити від клієнтів, використовуючи модель машинного навчання для класифікації зображень.
- **Модель згорткової нейронної мережі (CNN)**, навчена на підготовленому датасеті, що демонструє високі показники точності.
- **Клієнтський мобільний застосунок для Android**, який дозволяє користувачам надсилати фотографії для класифікації та отримувати результати у зручному вигляді.

Система показала:

- **Високу точність** класифікації на тестових даних (~ 98.7%).
- **Надійність роботи REST API**, що стабільно обробляє запити.
- **Функціональність** клієнтського застосунку, який забезпечує базову взаємодію між користувачем та системою.

4.3.2 Ідентифіковані недоліки та можливості вдосконалення

1. **Обмежений функціонал клієнтської частини**

   Поточна версія мобільного застосунку реалізує базові можливості (зйомка фото, вибір з галереї, надсилання запиту та отримання результату). Для розширення функціональності доцільно додати:

   - Історія класифікацій для користувача.
   - Можливість перегляду статистики по категоріях сміття.
   - Реалізацію профілів користувача та реєстрацію.

2. **Оптимізація моделі**

   - Виконати додаткове навчання моделі на розширеному датасеті для підвищення якості класифікації складних випадків, зокрема для категорій, що часто плутаються (наприклад, скло та пластик).
   - Використати методи аугментації даних та регуляризації для запобігання перенавчанню.

3. **Покращення продуктивності**

   - Зменшення часу обробки запитів за рахунок оптимізації REST API.
   - Додати механізми кешування результатів на клієнтській частині для пришвидшення повторних запитів.

4. **Розширення інтерфейсу**

   - Додати мультимовну підтримку.

- Розробити веб-інтерфейс для доступу до системи з десктоп платформ.
- Інтегрувати систему в інші сервіси, наприклад, системи управління сміттєзбірними пунктами.

5. Інтеграція додаткових технологій

- Використати TensorFlow Lite або ONNX для розгортання моделі безпосередньо на мобільному пристрої, що дозволить працювати офлайн.

4.3.3 Висновок

Система є прототипом, який демонструє можливості використання машинного навчання для автоматичної класифікації сміття. Вона може бути основою для розробки повноцінного продукту, орієнтованого на екологічне сортування відходів, підвищення обізнаності користувачів та полегшення процесу переробки.

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було проведено тестування та аналіз роботи розробленої системи для класифікації сміття та екологічного сортування. Було досліджено точність моделі машинного навчання, оцінено її ефективність за допомогою метрик (accuracy, precision, recall, F1-score) та проведено тестування взаємодії клієнтського мобільного застосунку з среверною частиною через REST API.

Матриця неточностей дозволила визначити класи, де модель демонструє найкращі результати, а також виявити категорії, де виникають помилки класифікації. Це дозволяє окреслити напрямки для майбутнього вдосконалення системи, наприклад, шляхом розширення навчальної вибірки, покращення архітектури моделі або використання додаткових даних.

Отримані результати підтверджують працездатність роозробленої системи, її здатність інтегруватися у клієнтські застосунки, а також високу точність класифікації об’єктів. Проведене тестування довело, що система може ефективно вирішувати завдання багатокласової класифікації зображень сміття, що є важливим кроком на шляху до її практичного застосування у сфері екологічного сортування відходів.

Подальший розвиток системи передбачає оптимізацію моделі, розширення функціоналу клієнтського затосунку та проведення експериментів на більш складних датасетах для підвищення її універсальності та точності.

ВИСНОВКИ

У дипломній роботі було розроблено програмну систему для класифікації сміття та екологічного сортування на основі методів машинного навчання. Система включає среверну частину з REST API, модель машинного навчання на основі згорткової нейронної мережі (CNN), а також клієнтський мобільний застосунок для операційної системи Android, який забезпечує взаємодію користувача із сервером.

Під час роботи над проєктом було виконано:

- Огляд предметної області та аналіз існуючих рішень для автоматичної класифікації сміття;
- Розробку архітектури системи, що дозволяє забезпечити модульність, машстабованість та інтеграцію із зовнішніми додатками через REST API;
- Розробку моделі класифікації твердих відходів на основі CNN, проведення навчання моделі на спеціально підготовленому датасеті та аналіз результатів класифікації;
- Реалізацію серверної частини системи, яка забезпечує отримання зображень, обробку запитів та повернення результатів класифікації;
- Розробку прототипу клієнтського мобільного затосунку для Android, який дозволяє завантажувати зображення з камери або галереї, надсилаючи їх на сервер та отримувати результат класифікації;
- Проведення тестування системи, включаючи оцінку точності моделі, аналіз матриці неточностей та перевірку роботи клієнт-серверної взаємодії.

Результати тестування показали, що розроблена система здатна ефективно виконувати багатокласову класифікацію зображень сміття з високими показниками точності (accuracy – 98,77%, precision – 98,96%, recall – 98,78%, F1-score – 98,86%).

Розроблена система є прототипом, що демонструє можливості використання методів штучного інтелекту для автоматизації процесу сортування відходів. Вона має потенціал для подальшого розвитку, який може включати:

- Розширення фукнціоналу клієнтського застосунку;
- Розробку повноцінного користувацького інтерфейсу з додатковими можливостями;
- Інтеграцію із зовнішніми системи для збору та аналізу даних;
- Проведення додаткових експериментрів з використанням більших та більш різноманітних датасетів для підвищення точності та універсальності моделі.

Отримані результати підтверджують доцільність використання сучасних технологій машинного навчання та нейронних мереж для вирішення задач автоматичного сортування сміття. Розроблена система може стати основою для подальших досліджень та розробок у сфері екологічних технологій та сталого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванов І.І., Петров П.П. Основи машинного навчання. – Київ: Техніка, 2020. – 312 с.
2. LeCun Y., Bengio Y., Hinton G. Deep Learning // Nature. – 2015. – №512. – С. 436-444.
3. Chollet F. Deep Learning with Python. – Manning Publications, 2017. – 361 p.
4. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016. – 775 p.
5. Android Developers. [Електронний ресурс] – Доступно за посиланням: https://developer.android.com/docs
6. Retrofit – Type-safe HTTP client for Android and Java [Електронний ресурс] – Доступно за посиланням: https://square.github.io/retrofit/
7. OkHttp. [Електронний ресурс] – Доступно за посиланням: https://square.github.io/okhttp/
8. Nd4j & DeepLearning4j Documentation. [Електронний ресурс] – Доступно за посиланням: https://deeplearning4j.konduit.ai/
9. Garbage Image Dataset [Електронний ресурс] – Доступно за посиланням: https://www.kaggle.com/datasets/farzadnekouei/trash-type-image-dataset
10. TrashNet Dataset [Електронний ресурс] – Доступно за посиланням: https://github.com/garythung/trashnet
11. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016.
12. Recycl3r. What are the recycling rates in the world? [Електронний ресурс]. – Доступно за посиланням: https://recycl3r.com/what-are-the-recycling-rates-in-the-world/

13. Research Paper. Core.ac.uk – [Електронний ресурс]. – Доступно за посиланням: https://core.ac.uk/reader/78066340
14. Recycling robots and AI in landfill. Axios, 2023 [Електронний ресурс]. – Доступно за посиланням: https://www.axios.com/2023/04/04/recycling-robots-ai-landfill
15. Data augmentation technique examples [Електронний ресурс]. – Доступно за посиланням: https://www.researchgate.net/figure/Data-augmentation-technique-examples-a-Original-Image-b-Blur-c-Random-Gaussian_fig4_347674295

ДОДАТОК 1

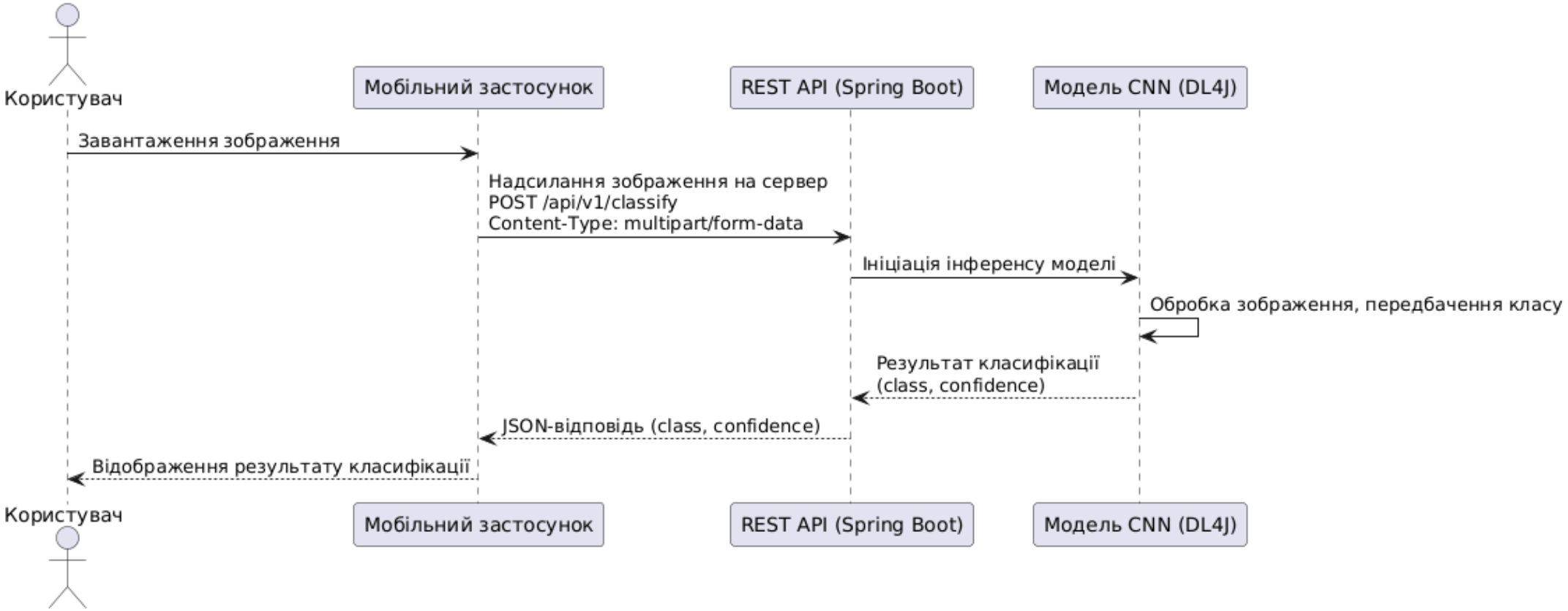
Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування

Діаграма послідовності взаємодії компонентів

ІАЛЦ.466530.004 Д1

Київ 2025 р.

Діаграма послідовності взаємодії компонентів системи класифікації зображень сміття на основі CNN



					ІАЛЦ.466530.004 Д1		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Лисак А.А.			*Система розпізнавання об'єктів або сцен для класифікації сміття та екологічного сортування*	Літ.	Аркуш 1 / Аркушів 1
Перев.		Коренко Д. В.					
Реценз.					Діаграма послідовності взаємодії компонентів	***НТУУ «КПІ» ФІОТ ІО-16***	
Н. Контр.							
Затверд.							

ДОДАТОК 2

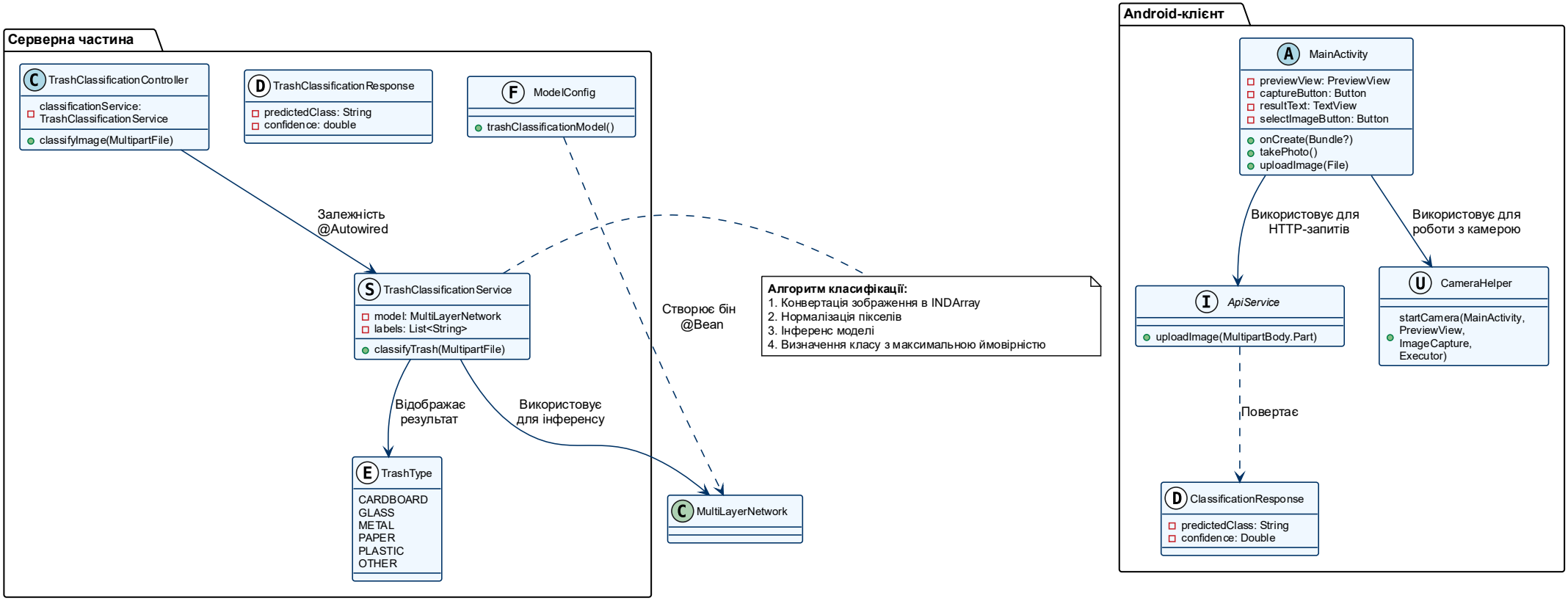
Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування

Функціональна схема (Діаграма класів)

ІАЛЦ.466530.005 Д2

Київ 2025 р.

Діаграма класів системи класифікації зображень сміття на основі CNN



					ІАЛЦ.466530.005 Д2	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Лисак А.А.			*Система розпізнавання об'єктів або сцен для класифікації сміття та екологічного сортування*	Літ. / Аркуш 1 / Аркушів 1
Перев.		Коренко Д. В.				
Реценз.					Функціональна схема (Діаграма класів)	*НТУУ «КПІ» ФІОТ ІО-16*
Н. Контр.						
Затверд.						

ДОДАТОК 3

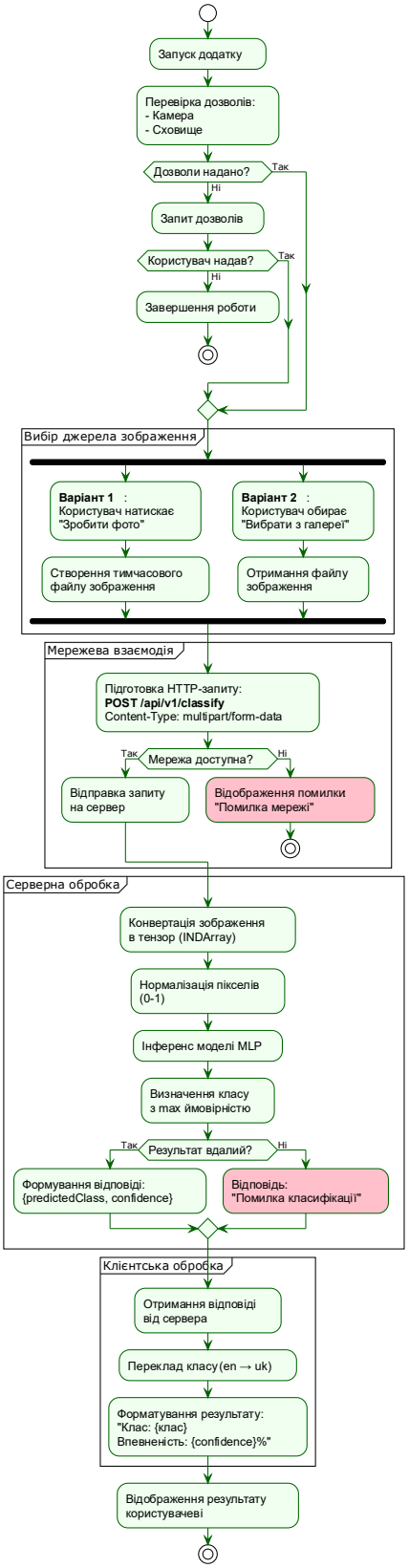
Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування

Алгоритм роботи системи класифікації сміття

ІАЛЦ.466530.006 ДЗ

Київ 2025 р.

Алгоритм роботи системи класифікації сміття



					ІАЛЦ.466530.006 ДЗ		
Змін.	Арк.	**№ докум.**	**Підпис**	**Дата**			
Розроб.		Лисак А.А.			*Система розпізнавання об'єктів або сцен для класифікації сміття та екологічного сортування* Алгоритм роботи системи класифікації сміття	Літ.	Аркуш 1 / Аркушів 1
Перев.		Коренко Д. В.					
Реценз.						***НТУУ «КПІ» ФІОТ ІО-16***	
Н. Контр.							
Затверд.							

ДОДАТОК 4

Система розпізнавання об’єктів або сцен для класифікації сміття та екологічного сортування

Текст програмного коду

ІАЛЦ.466530.007 Д4

Київ 2025 р.

Серверна частина:

TrashClassificationController.java

```
package ua.lysaka.thesis.trashclassifier.controller;

import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController
import ua.lysaka.thesis.trashclassifier.dto.TrashClassificationResponse;
import ua.lysaka.thesis.trashclassifier.service.TrashClassificationService;

@RestController
@RequestMapping("api/v1/classify")
@AllArgsConstructor
public class TrashClassificationController {
    private final TrashClassificationService classificationService;

    @PostMapping
    public ResponseEntity<TrashClassificationResponse>
classifyImage(@RequestParam("image") MultipartFile imageFile) {
        if (imageFile.isEmpty()) {
            return ResponseEntity.badRequest().build();
        }

        TrashClassificationResponse result =
classificationService.classifyTrash(imageFile);
        return ResponseEntity.ok(result);
    }
}
```

TrashClassificationService.java

```
package ua.lysaka.thesis.trashclassifier.service;

import lombok.RequiredArgsConstructor;
import org.datavec.image.loader.NativeImageLoader;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.api.preprocessor.ImagePreProcessingScaler;
import org.nd4j.linalg.factory.Nd4j;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import ua.lysaka.thesis.trashclassifier.dto.TrashClassificationResponse
import ua.lysaka.thesis.trashclassifier.model.TrashType;
```

```
import java.io.IOException;
import java.util.Arrays;
import java.util.List;

@Service
@RequiredArgsConstructor
public class TrashClassificationService {
    private final MultiLayerNetwork model;
    private final List<String> labels = Arrays.stream(TrashType.values())
            .map(Enum::name)
            .toList();

    public TrashClassificationResponse classifyTrash(MultipartFile imageFile) {
        try {
            // Loading the image
            NativeImageLoader loader = new NativeImageLoader(100, 100, 3);
            INDArray image = loader.asMatrix(imageFile.getInputStream());

            // Pixels normalizing
            ImagePreProcessingScaler scaler = new ImagePreProcessingScaler(0,
1);
            scaler.transform(image);

            // Inference
            INDArray output = model.output(image);

            double[] probabilities = output.toDoubleVector();

            // Print every class with probability
            for (int i = 0; i < labels.size(); i++) {
                System.out.printf("%s: %.4f%n", labels.get(i),
probabilities[i]);
            }

            int classIndex = Nd4j.argMax(output, 1).getInt(0);
            double confidence = output.getDouble(0, classIndex);
            String predictedClass = labels.get(classIndex);

            return new TrashClassificationResponse(predictedClass, confidence);
        } catch (IOException e) {
            return new TrashClassificationResponse("ERROR", 0.0);
        }
    }
}
```

ModelConfig.java

```java
package ua.lysaka.thesis.trashclassifier.config;

import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.util.ModelSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.io.File;
import java.io.IOException;

@Configuration
public class ModelConfig {
    @Bean
    public MultiLayerNetwork trashClassificationModel() throws IOException {
        File modelFile = new File("src/main/resources/model/model.zip");
        MultiLayerNetwork model =
ModelSerializer.restoreMultiLayerNetwork(modelFile);
        model.init();

        return model;
    }
}
```

TrashClassificationResponse.java

```java
package ua.lysaka.thesis.trashclassifier.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class TrashClassificationResponse {
    private String predictedClass;
    private double confidence;
}
```

TrashType.java

```java
package ua.lysaka.thesis.trashclassifier.model;

public enum TrashType {
    CARDBOARD,
    GLASS,
    METAL,
    PAPER,
    PLASTIC,
    OTHER
}
```

TrashClassifierApplication.java

```
package ua.lysaka.thesis.trashclassifier;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TrashclassifierApplication {

    public static void main(String[] args) {
        SpringApplication.run(TrashclassifierApplication.class, args);
    }

}
```

Клієнтська частина:

MainActivity.kt

```
package com.example.trashclassifierclient

import android.Manifest
import android.content.pm.PackageManager
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.ImageCapture
import androidx.camera.core.ImageCaptureException
import androidx.camera.view.PreviewView
import androidx.core.content.ContextCompat
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.MultipartBody
import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.Call
import java.io.File
import java.text.SimpleDateFormat
import java.util.*
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
import com.example.trashclassifierclient.api.ApiService
import com.example.trashclassifierclient.api.ClassificationResponse
import com.example.trashclassifierclient.camera.CameraHelper
import okhttp3.RequestBody
import android.content.Intent
import android.net.Uri
import android.provider.MediaStore

class MainActivity : AppCompatActivity() {

    private lateinit var previewView: PreviewView
```

```
    private lateinit var captureButton: Button
    private lateinit var resultText: TextView
    private lateinit var selectImageButton: Button

    private lateinit var imageCapture: ImageCapture
    private lateinit var cameraExecutor: ExecutorService

    private val trashTypeMap = mapOf(
        "CARDBOARD" to "Картон",
        "GLASS" to "Скло",
        "METAL" to "Метал",
        "PAPER" to "Папір",
        "PLASTIC" to "Пластик",
        "OTHER" to "Інше"
    )

    private val requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted ->
        if (isGranted) {
            startCamera()
        } else {
            resultText.text = "Дозвіл на камеру не надано"
        }
    }

    private val requestGalleryPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted ->
        if (isGranted) {
            openGallery()
        } else {
            resultText.text = "Дозвіл на доступ до галереї не надано"
        }
    }


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        previewView = findViewById(R.id.previewView)
        captureButton = findViewById(R.id.captureButton)
        resultText = findViewById(R.id.resultText)
        selectImageButton = findViewById(R.id.selectImageButton)

        cameraExecutor = Executors.newSingleThreadExecutor()

        if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.CAMERA)
            == PackageManager.PERMISSION_GRANTED
        ) {
            startCamera()
        } else {
            requestPermissionLauncher.launch(Manifest.permission.CAMERA)
        }
```

```
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_EXTERNAL_STORAGE)
        == PackageManager.PERMISSION_GRANTED
    ) {
        selectImageButton.setOnClickListener {
            if (ContextCompat.checkSelfPermission(
                    this,
                    Manifest.permission.READ_EXTERNAL_STORAGE
                )
                == PackageManager.PERMISSION_GRANTED
            ) {
                openGallery()
            } else {

requestGalleryPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL
_STORAGE)
            }
        }
    } else {

requestGalleryPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL
_STORAGE)
    }

    captureButton.setOnClickListener {
        takePhoto()
    }
}

private fun startCamera() {
    imageCapture = ImageCapture.Builder().build()
    CameraHelper.startCamera(this, previewView, imageCapture,
cameraExecutor)
}

private fun takePhoto() {
    val photoFile = File(
        externalCacheDir,
        SimpleDateFormat("yyyy-MM-dd-HH-mm-ss-SSS", Locale.US)
            .format(System.currentTimeMillis()) + ".jpg"
    )

    val outputOptions =
ImageCapture.OutputFileOptions.Builder(photoFile).build()

    imageCapture.takePicture(
        outputOptions,
        ContextCompat.getMainExecutor(this),
        object : ImageCapture.OnImageSavedCallback {
            override fun onError(exc: ImageCaptureException) {
                resultText.text = "Помилка зйомки: ${exc.message}"
            }
```

```
        override fun onImageSaved(output: ImageCapture.OutputFileResults) {
                        resultText.text = "Фото зроблено. Відправляємо..."
                        uploadImage(photoFile)
                    }
                }
            )
        }

        private fun openGallery() {
            val intent = Intent(Intent.ACTION_PICK,
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
            intent.type = "image/*"
            startActivityForResult(intent, REQUEST_CODE_GALLERY)
        }

        override fun onActivityResult(requestCode: Int, resultCode: Int, data:
        Intent?) {
            super.onActivityResult(requestCode, resultCode, data)

            if (requestCode == REQUEST_CODE_GALLERY && resultCode == RESULT_OK) {
                val selectedImageUri = data?.data

                selectedImageUri?.let {
                    val imagePath = getRealPathFromURI(it)
                    val imageFile = File(imagePath)

                    uploadImage(imageFile)
                }
            }
        }

        fun getRealPathFromURI(uri: Uri): String? {
            val cursor = contentResolver.query(uri, null, null, null, null)
            cursor?.apply {
                moveToFirst()
                val columnIndex = getColumnIndex(MediaStore.Images.Media.DATA)
                val filePath = getString(columnIndex)
                close()
                return filePath
            }
            return null
        }

        private fun uploadImage(imageFile: File) {
            val requestFile = RequestBody.create("image/*".toMediaTypeOrNull(),
        imageFile)
            val body = MultipartBody.Part.createFormData("image", imageFile.name,
        requestFile)

            val retrofit = Retrofit.Builder()
                .baseUrl("https://huge-lizard-illegally.ngrok-free.app/")
                .client(OkHttpClient())
                .addConverterFactory(GsonConverterFactory.create())
                .build()

            val service = retrofit.create(ApiService::class.java)
```

```
        service.uploadImage(body).enqueue(object :
retrofit2.Callback<ClassificationResponse> {
            override fun onResponse(
                call: Call<ClassificationResponse>,
                response: retrofit2.Response<ClassificationResponse>
            ) {
                if (response.isSuccessful) {
                    val predictedClass = response.body()?.predictedClass
?: "Невідомо"
                    val translatedClass = trashTypeMap[predictedClass]
?: "Невідомо"
                    resultText.text = "Результат: $translatedClass"
                } else {
                    resultText.text = "Помилка: ${response.code()}"
                }
            }

            override fun onFailure(call: Call<ClassificationResponse>,
t: Throwable) {
                resultText.text = "Помилка запиту: ${t.message}"
            }
        })
    }

    companion object {
        const val REQUEST_CODE_GALLERY = 1
    }
}
```

ApiService.kt

```
package com.example.trashclassifierclient.api

import okhttp3.MultipartBody
import retrofit2.Call
import retrofit2.http.Multipart
import retrofit2.http.POST
import retrofit2.http.Part

interface ApiService {
    @Multipart
    @POST("api/v1/classify")
    fun uploadImage(
        @Part image: MultipartBody.Part
    ): Call<ClassificationResponse>
}
```

ClassificationResponse.kt

```
package com.example.trashclassifierclient.api

data class ClassificationResponse(
    val predictedClass: String,
    val confidence: Double
)
```

CameraHelper.kt

```
package com.example.trashclassifierclient.camera

import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageCapture
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.camera.view.PreviewView
import com.example.trashclassifierclient.MainActivity
import androidx.core.content.ContextCompat
import java.util.concurrent.Executor

class CameraHelper {

    companion object {
        fun startCamera(
            activity: MainActivity,
            previewView: PreviewView,
            imageCapture: ImageCapture,
            cameraExecutor: Executor
        ) {
            val cameraProviderFuture =
ProcessCameraProvider.getInstance(activity)

            cameraProviderFuture.addListener({
                val cameraProvider = cameraProviderFuture.get()

                val preview = Preview.Builder().build().also {
                    it.setSurfaceProvider(previewView.surfaceProvider)
                }

                val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

                try {
                    cameraProvider.unbindAll()
                    cameraProvider.bindToLifecycle(
                        activity, cameraSelector, preview, imageCapture
                    )
                } catch (e: Exception) {
                }
            }, ContextCompat.getMainExecutor(activity))
        }
    }
}
```

Проєкт для навчання моделі CNN:

TrainModel.java

```
package ua.lysaka.trashclassifiertrain;

import org.datavec.api.io.labels.ParentPathLabelGenerator;
import org.datavec.api.split.FileSplit;
import org.datavec.image.recordreader.ImageRecordReader;
import org.datavec.image.transform.*;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.nn.conf.*;
import org.deeplearning4j.nn.conf.inputs.InputType;
import org.deeplearning4j.nn.conf.layers.*;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.evaluation.classification.Evaluation;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.dataset.api.preprocessor.ImagePreProcessingScaler;
import org.nd4j.linalg.learning.config.Adam;
import org.nd4j.linalg.lossfunctions.LossFunctions;

import java.io.File;
import java.util.Random;


public class TrainModel {

    public static void main(String[] args) throws Exception {
        int height = 100;
        int width = 100;
        int channels = 3;
        int batchSize = 32;
        int numClasses = 6;
        int numEpochs = 20;
        long seed = 123;

        String dataPath = "src/main/resources/training_dataset";
        File datasetDir = new File(dataPath);
        ImageRecordReader recordReader = new ImageRecordReader(height,
width, channels, new ParentPathLabelGenerator());

        recordReader.initialize(new FileSplit(datasetDir, new
Random(seed)));
        ImageTransform transform = new MultiImageTransform(
                new FlipImageTransform(1),
                new RotateImageTransform(15)
        );
        DataSetIterator dataIter = new RecordReaderDataSetIterator(
                recordReader, batchSize, 1, numClasses);
        dataIter.setPreProcessor(new ImagePreProcessingScaler(0, 1));
```

```
        MultiLayerConfiguration config = new
NeuralNetConfiguration.Builder()
                .seed(seed)
                .updater(new Adam(0.0001))
                .weightInit(WeightInit.XAVIER)
                .list()
                .layer(new ConvolutionLayer.Builder()
                        .kernelSize(3, 3).stride(1,
1).nIn(channels).nOut(32)
                        .activation(Activation.RELU).build())
                .layer(new BatchNormalization())
                .layer(new SubsamplingLayer.Builder()
                        .kernelSize(2,
2).poolingType(PoolingType.MAX).build())
                .layer(new ConvolutionLayer.Builder()
                        .kernelSize(3, 3).stride(1, 1).nOut(64)
                        .activation(Activation.RELU).build())
                .layer(new BatchNormalization())
                .layer(new SubsamplingLayer.Builder()
                        .kernelSize(2,
2).poolingType(PoolingType.MAX).build())
                .layer(new
DenseLayer.Builder().nOut(128).activation(Activation.RELU).build())
                .layer(new DropoutLayer(0.5))
                .layer(new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                        .nOut(numClasses)
                        .activation(Activation.SOFTMAX)
                        .build())
                .setInputType(InputType.convolutional(height, width,
channels))
                .build();

        MultiLayerNetwork model = new MultiLayerNetwork(config);
        model.init();
        model.setListeners(new ScoreIterationListener(10));

        System.out.println("Початок навчання...");
        for (int i = 0; i < numEpochs; i++) {
            model.fit(dataIter);
            dataIter.reset();
            System.out.println("Епоха " + (i + 1) + " завершена");
        }

        Evaluation eval = model.evaluate(dataIter);
        System.out.println(eval.stats());

        File modelFile = new File("model.zip");
        ModelSerializer.writeModel(model, modelFile, true);
        System.out.println("Модель збережена у файл: " +
modelFile.getAbsolutePath());
    }
}
```

TestModel.java

```
package ua.lysaka.trashclassifiertrain;

import org.datavec.api.io.labels.ParentPathLabelGenerator;
import org.datavec.api.split.FileSplit;
import org.datavec.image.loader.NativeImageLoader;
import org.datavec.image.recordreader.ImageRecordReader;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.dataset.api.preprocessor.ImagePreProcessingScaler;

import java.io.File;
import java.util.Random;

public class TestModel {

    public static void main(String[] args) throws Exception {
        int height = 100;
        int width = 100;
        int channels = 3;
        int batchSize = 32;
        int numClasses = 6;
        long seed = 123;

        File modelFile = new File("model.zip");
        MultiLayerNetwork model =
ModelSerializer.restoreMultiLayerNetwork(modelFile);

        File testData = new File("src/main/resources/testing_dataset");
        FileSplit testSplit = new FileSplit(testData,
NativeImageLoader.ALLOWED_FORMATS, new Random(seed));
        ParentPathLabelGenerator labelMaker = new
ParentPathLabelGenerator();
        ImageRecordReader testRecordReader = new
ImageRecordReader(height, width, channels, labelMaker);
        testRecordReader.initialize(testSplit);
        DataSetIterator testIter = new
RecordReaderDataSetIterator(testRecordReader, batchSize, 1, numClasses);
        testIter.setPreProcessor(new ImagePreProcessingScaler(0, 1));

        Evaluation eval = model.evaluate(testIter);
        System.out.println(eval.stats());

        System.out.println("Матриця плутанини:");
        System.out.println(eval.getConfusionMatrix().toCSV());
    }
}
```

TrashClassifierTrainApplication.java

```
package ua.lysaka.trashclassifiertrain;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TrashClassifierTrainApplication {

    public static void main(String[] args) {
        SpringApplication.run(TrashClassifierTrainApplication.class,
args);
    }

}
```