

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
РОЗПІЗНАВАННЯ ОБ'ЄКТІВ ДЛЯ АВТОМАТИЗАЦІЇ
СОРТУВАННЯ СМІТТЯ
RESEARCH AND SOFTWARE IMPLEMENTATION OF OBJECT
RECOGNITION METHODS FOR AUTOMATED GARBAGE SORTING

Виконав: студент 2 курсу, групи КНТ-120м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Костюк П.М.

(прізвище та ініціали)

Керівник Зайко Т.А.

(прізвище та ініціали)

Рецензент Коцур М.І.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Костюка Павла Миколайовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів розпізнавання об'єктів для автоматизації сортування сміття. Research and Software Implementation of Object Recognition Methods for Automated Garbage Sorting

керівник проєкту (роботи) Зайко Тетяна Анатоліївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “05” листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) 5 грудня 2021 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблемної області та постановка завдань дослідження. 2. Матеріали та методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Зайко Т.А., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “4” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-13 тиждень	Додатки
8	Нормоконтроль та рецензування.	14-15 тижні	
9	Захист роботи.	16 тиждень	

Студент(ка)

(підпис) Костюк П.М.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Зайко Т.А.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
113 с., 23 рис., 5 табл., 4 дод., 22 джерела.

РОЗПІЗНАВАННЯ, КЛАСИФІКАЦІЯ, СЕГМЕНТАЦІЯ, ГРАФІЧНЕ
ЗОБРАЖЕННЯ, МАШИННЕ НАВЧАННЯ, АВТОМАТИЗАЦІЯ.

Об'єкт дослідження – процес розпізнавання та класифікації об'єктів за графічними зображеннями.

Предмет дослідження – методи розпізнавання та класифікації об'єктів по зображенням.

Мета роботи – створення методу і програмного комплексу для класифікації певного типу об'єктів і розпізнавання даних об'єктів по зображеннях з декількох точок.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, технології лінійного прогнозування, мова програмування C++ та бібліотека комп'ютерного зору OpenCV, персональний комп'ютер з процесором i3-10100F 3.6GHz під управлінням операційної системи Microsoft Windows 10.

Результати. Створено програмний застосунок, що реалізує метод розпізнавання та класифікації об'єктів за зображеннями.

Висновки. Розроблено програмний застосунок для розпізнавання та класифікації об'єктів за зображеннями.

Галузь використання – системи сортування, розпізнавання та діагностування.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 113 pages, 23 figures, 4 tables, 4 appendixes, 22 sources.

RECOGNITION, CLASSIFICATION, SEGMENTATION, GRAPHICAL REPRESENTATION, MACHINE LEARNING, AUTOMATION.

The object of research is the process of recognizing and classifying objects based on graphic images.

The subject of the research is Methods for recognizing and classifying objects by Image.

The aim of this work is to create a method and software package for classifying a certain type of object and recognizing these objects based on images from several points.

Materials, methods and technical means: structural and Object-Oriented Programming, linear forecasting technologies, the C++ programming language and the OpenCV computer vision library, a personal computer with an i3-10100f 3.6 GHz processor running the Microsoft Windows 10 operating system.

Results. A software application has been created that implements a method for recognizing and classifying objects based on images.

Conclusions. A software application has been developed for recognizing and classifying objects based on images.

The field of use is sorting, recognition, and diagnostic systems.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблемної області та постановка завдань дослідження	11
1.1 Технологія автоматичного РО	11
1.2 Моделі класифікації для РО.....	13
1.3 Опис задачі РО	16
1.4 Використання методів навчання класифікатора для РО серед аналогів.....	21
1.5 Постановка завдання	22
1.6 Висновки за розділом 1	24
2 Матеріали та методи.....	25
2.1 Використання методів сегментації зображень при розробці методів РО.....	25
2.2 Проектування алгоритмів роботи методів РО	42
2.3 Висновки за розділом 2	47
3 Розробка архітектури програми	49
3.1 Вибір технології розробки	49
3.2 Проектування розробки.....	51
3.3 Вимоги до технічних засобів	55
3.4 Висновки за розділом 3	55
4 Основні рішення щодо реалізації компонентів системи.....	56
4.1 Клас QassifierTrainer	56
4.2 Клас Predictor.....	57
4.3 Клас PredictionsCycle	58
4.4 Клас Segment	59
4.5 Клас SegmentFilter.....	59
4.6 Структура Params	60
4.7 Деталі реалізації	61

4.8 Висновки за розділом 4	68
5 Експлуатація, тестування та експериментальне дослідження програми	69
5.1 Опис стратегії тестування	69
5.2 Результати тестування	71
5.3 Висновки за результатами тестування.....	74
Висновки	76
Перелік джерел посилання	77
Додаток А Технічне завдання	81
Додаток Б Опис програми	87
Додаток В Текст програми	91
Додаток Д Слайди презентації.....	107

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BRIEF	– binary robust independent elementary features;
GFTT	– good features to track;
GLOH	– gradient location-orientation histogram;
ORB	– oriented FAST and rotated BRIEF;
SIFT	– scale invariant feature transform;
SURF	– speeded up robust features;
SVM	– метод опорних векторів;
МН	– машинне навчання;
ОС	– операційною системою;
ПЗ	– програмний затосунок;
ПК	– персональний комп'ютер;
РО	– розпізнавання образів.

ВСТУП

Зі зростанням автоматизації виробництва виникла необхідність забезпечити роботу багатьох технологічних процесів без участі людини. Головним чином це процеси, пов'язані з рутинною, одноманітною роботою, або небезпечні для людини.

Зараз комп'ютерний зір і, зокрема, його підрозділ, розпізнавання образів, є інтенсивно розвивається галуззю комп'ютерних знань. На даний момент його можливості обмежені, для багатьох завдань розпізнавання, наприклад, неможливо зробити рішення, що працює при будь-яких умовах. Незважаючи на це, йому знаходиться багато практичних застосувань.

Розпізнавання образів знаходить застосування при таких завданнях, як:

- створення автомобілів, які можуть пересуватися самостійно;
- у біоінформатиці при обробці результатом з геноміки, протеоміки, метаболоміки;
- у медицині як допоміжний метод при аналізі медичних зображень;
- ідентифікація особи за біометричною інформацією;
- у промисловості: для розпізнавання деталей інтелектуальними роботами, для оцінки якості виробів;
- обробка відсканованих сторінок тексту.

Однією з робіт, для яких актуальна потреба в автоматизації, є прибирання сміття.

Тому було вирішено розробити програмний застосунок (ПЗ), здатну розпізнати сміття у високій надійності. Дана розробка призначається для використання в керуючій системі для робіт, що збирають сміття.

Метою роботи є створення методу і програмного комплексу для класифікації певного типу об'єктів і розпізнавання даних об'єктів по зображеннях з декількох точок.

Для досягнення поставленої мети слід вирішити наступні основні завдання:

- розглянути предметну область і існуючі аналоги;
- спроектувати програмний комплекс, його основні сутності та алгоритми;
- реалізувати програмний комплекс;
- протестувати програмний комплекс.

Наукову новизну роботи складають використана комбінація методів для такої складної для розпізнавання у зв'язку з величезною внутрішньокласовою мінливістю категорії, як сміття.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Перший розділ буде присвячено детальному аналізу проблемної області: технологія автоматичного розпізнавання образів. Особливу увагу буде приділено використанню методів розпізнавання графічних об'єктів. Під час роботи буде проведено дослідження основних термінів таких, як розпізнавання образів (РО), моделі РО, класифікатор для РО.

1.1 Технологія автоматичного РО

Через появу великих даних і технологій машинного навчання (МН) стало доступно багато даних, які раніше були або виведені, або припущені. Ці дані, засновані на більш надійних джерелах, надали можливість використовувати більш складні методи аналізу даних для отримання додаткових переваг для бізнесу [1]-[6].

Іншими словами, тепер, коли дізналися більше, слід перейти від мети отримання самої інформації до аналізу і розуміння даних, які вже надходили.

З усіх інструментів, що використовуються у великих даних, РО знаходиться в центрі уваги. Воно включає в себе ядро аналітики великих даних – технологія витягує з даних інформацію і розкриває приховані за нею значення [1]-[6].

РО дає стратегічну перевагу для компанії, яка робить її здатною до постійного вдосконалення і розвитку на постійно мінливому ринку.

РО – це процес виділення і сегментування даних відповідно до встановлених критеріїв або загальними елементами, який виконується спеціальними алгоритмами [1]-[6].

РО має справу з автоматичним виявленням закономірностей в даних за допомогою комп'ютерних алгоритмів і з використанням цих

закономірностей для прийняття дій, таких як класифікація даних за різними категоріями.

Іншими словами, розпізнавання образів ідентифікує речі за їх характеристиками.

Самі дані можуть бути чим завгодно [1]-[6]:

- текст;
- зображення;
- звук.

Будь-яка інформація послідовного характеру може оброблятися за допомогою алгоритмів розпізнавання образів, що робить послідовності зрозумілими і робить можливим їх практичне використання [1]-[6].

Можливість розпізнавання базується на схожості подібних об'єктів. Незважаючи на те, що всі явища і предмети не схожі один на одного, між деякими з них завжди можна знайти подібності за тією чи іншою ознакою [1]-[6].

Всі методи РО діляться на два види: методи, засновані на теорії рішень і структурні методи [1]-[6]. Перші засновані на обчисленні за допомогою кількісних величин, таких як довжина, текстурна і т.д. другі орієнтовані на образи, для опису яких більше підходять якісні величини, наприклад реляційні [1]-[6]. Також в розпізнаванні об'єктів важливу роль відіграє навчання на основі відомої вибірки [1]-[6].

Під образом мається на увазі деяка впорядкована сукупність ознак [1]-[6]. Класом образів називається сукупність об'єктів з однаковими властивостями [1]-[6]. Класифікатором або вирішальним правилом називається правило віднесення образу до одного з класів на підставі його вектора ознак [1]-[6]. На практиці широке застосування мають три форми представлення ознак: вектор ознак (для кількісних величин), символічний рядок і дерева ознак (для структурних величин) [1]-[6].

Методи, засновані на зіставленні, являють собою набори векторів ознак кожного класу об'єктів. Новий образ буде віднесений до того класу,

який виявиться найбільш близьким, в межах заздалегідь заданої метрики [1]-[6]. Очевидно, що найпростіший підхід полягає в пошуку мінімальної відстані, яка обчислюється за допомогою евклідових норм між векторами ознак невідомого об'єкта і векторами прототипу [1]-[6]. Висновок про приналежність об'єкта до певного класу відбувається за найменшою з цих відстаней [1]-[6]. Мінімальний класифікатор відстані добре працює в тих випадках, де відстань між точками математичного очікування класів велике в порівнянні з діапазоном розкиду об'єктів кожного класу [1]-[6].

Не менш важливими є методи розпізнавання образів, засновані на імовірнісних класифікаторах, через випадковостей, які впливають на породження класів образів [1]-[6]. Отже, необхідно виробити такий оптимальний підхід, при використанні якого виявиться найменша ймовірність появи помилок [1]-[6].

1.2 Моделі класифікації для РО

Розглянемо загальні елементи моделі класифікації [7]-[9].

Клас – множина об'єктів мають загальні властивості. Для об'єктів одного класу передбачається наявність схожості [7]-[9]. Для завдання розпізнавання може бути визначено довільну кількість класів, більше 1 [7]-[9]. Кількість класів позначається числом S [7]-[9]. Кожен клас має свою ідентифікуючу мітку класу.

Класифікація – процес призначення міток класу об'єктам, згідно деякого опису властивостей цих об'єктів. Класифікатор-пристрій, який в якості вхідних даних отримує набір ознак об'єкта, а в якості результату видає мітку класу [7]-[9].

Верифікація – процес зіставлення примірника об'єкта з однією моделлю об'єкта або описом класу [7]-[9].

Під образом будемо розуміти найменування області в просторі ознак, в якій відображається безліч об'єктів або явищ матеріального світу [7]-[9].

Ознака-кількісний опис тієї чи іншої властивості досліджуваного предмета або явища [7]-[9].

Простір ознак Це N -мірний простір, визначений для даного завдання розпізнавання, де N -фіксоване число вимірюваних ознак для будь-яких об'єктів [7]-[9]. Вектор з простору ознак x , відповідний об'єкту задачі розпізнавання це N -мірний вектор з компонентами $(x_1, x_2, \dots, x_N)$, які є значеннями ознак для даного об'єкта [7]-[9].

Іншими словами, розпізнавання образів можна визначити, як віднесення вихідних даних до певного класу за допомогою виділення істотних ознак або властивостей, що характеризують ці дані, із загальної маси несуттєвих деталей [7]-[9].

Прикладами задач класифікації є [1]-[9]:

- розпізнавання символів;
- розпізнавання мови;
- встановлення медичного діагнозу;
- прогноз погоди;
- розпізнавання облич
- класифікація документів та ін.

Найчастіше вихідним матеріалом служить отримане з камери зображення [1]-[9]. Завдання можна сформулювати як отримання векторів ознак для кожного класу на розглянутому зображенні [1]-[9]. Процес можна розглядати як процес кодування, що полягає в присвоєнні значення кожній ознаці з простору ознак для кожного класу [1]-[9].

Другим завданням розпізнавання є виділення характерних ознак або властивостей з вихідних зображень [1]-[9]. Це завдання можна віднести до попередньої обробки [1]-[9]. Якщо розглянути завдання розпізнавання мови, можна виділити такі ознаки як голосні і приголосні звуки [1]-[9]. Ознака повинна представляти з себе характерну властивість конкретного класу, при цьому загальні для цього класу [1]-[9]. Ознаки, що характеризують відмінності між-міжкласові ознаки. Ознаки загальні для всіх класів не несуть

корисної інформації і не розглядаються як ознаки в задачі розпізнавання [1]-[9]. Вибір ознак є одним з важливих завдань, пов'язаних з побудовою системи розпізнавання [1]-[9].

Після того, як визначені ознаки необхідно визначити оптимальну вирішальну процедуру для класифікації [7]-[9]. Розглянемо систему розпізнавання образів, призначену для розпізнавання різних M класів, позначених як $m_1, m_2, \dots, m_n$ [7]-[9]. Тоді можна вважати, що простір образів складається з M областей, кожна містить точки, відповідні чином з одного класу. Тоді завдання розпізнавання може розглядатися як побудова меж, що розділяють M класів, виходячи з прийнятих векторів вимірювань [7]-[9].

Рішення задачі попередньої обробки зображення, виділення ознак і завдання отримання оптимального рішення і класифікації зазвичай пов'язане з необхідністю провести оцінку ряду параметрів [7]-[9]. Це призводить до завдання оцінки параметрів [7]-[9]. Крім того, очевидно, що виділення ознак може використовувати додаткову інформацію виходячи з природи класів [7]-[9].

Порівняння об'єктів можна проводити на основі їх уявлення у вигляді векторів вимірювань. Дані вимірювань зручно представляти у вигляді речових чисел. Тоді подібність векторів ознак двох об'єктів може бути описано за допомогою евклідової відстані [7]-[9].

$$\|x_1 - x_2\| = \sqrt{\sum_{i=1, d} (x_1[i] - x_2[i])^2}, \quad (1.1)$$

де d – розмірність вектора ознаки x .

Розділяють 3 групи методів РО [7]-[9].

1. Порівняння зі зразком. У цю групу входить класифікація по найближчому середньому, класифікація по відстані до найближчого сусіда [7]-[9]. Також до групи порівняння зі зразком можна віднести структурні методи розпізнавання [7]-[9].

2. Статистичний метод. Як видно з назви, статистичні методи використовують деяку статистичну інформацію при вирішенні завдання розпізнавання [7]-[9]. Метод визначає приналежність об'єкта до конкретного класу на основі ймовірності у ряді випадків це зводиться до визначення апостеріорної ймовірності належності об'єкта до певного класу, за умови, що ознаки цього об'єкта прийняли відповідні значення [7]-[9]. Прикладом служить метод на основі байєсівського вирішального правила [7]-[9].

3. Нейронні мережі. Окремий клас методів розпізнавання. Відмінною особливістю від інших є здатність навчатися [7]-[9].

1.3 Опис задачі РО

Розпізнавання образів-це завдання класифікації або ідентифікації об'єктів на зображенні або відеоряді [2]-[5], [7]-[9].

Формальна постановка задачі класифікації визначається наступним чином: потрібно вивести вирішальне правило, що оцінює індикаторну функцію, що розбиває простір образів на p непересічних класів, на підставі функції, що ставить у відповідність кожному об'єкту з простору образів крапку в просторі ознак [2]-[5], [7]-[9].

Також слід ввести визначення прецеденту. Прецедент-це образ, для якого відомий клас.

На підставі наявності або відсутності прецедентної інформації завдання розпізнавання діляться на завдання з навчанням і без навчання.

Завдання розпізнавання на основі наявного безлічі прецедентів називається класифікацією з навчанням (або з учителем) [2]-[5], [7]-[9].

У тому випадку, якщо є безліч векторів ознак, отриманих для деякого набору образів, але правильна класифікація цих образів невідома, виникає завдання поділу цих образів на класи за подібністю відповідних векторів ознак. Це завдання називається кластеризацією або розпізнаванням без навчання [2]-[5], [7]-[9].

При побудові системи розпізнавання образів виділяються наступні завдання [2]-[5], [7]-[9]:

- сегментація і виділення об'єкта;
- вибір ознак об'єкта;
- навчання класифікатора і класифікація.

РО (класифікація) є однією з областей комп'ютерного зору.

Комп'ютерне зір являє собою область знань, що вивчає методи для отримання, обробки, аналізу і розуміння зображень і в цілому багаторазової інформації з реального світу для того, щоб зробити чисельну або символну інформацію, в тому числі у формі рішень.

Класичним завданням комп'ютерного зору є визначення, чи містить зображення якийсь певний об'єкт, властивість або дію [2]-[5], [7]-[9].

1.3.1 Сегментація і виділення об'єкта

Сегментація – це процес поділу зображення на безліч сегментів.

Метою сегментації є спрощення та/або зміна представлення зображення в таке, яке простіше аналізувати. Таким чином, сегментація призначає кожному пікселю зображення мітку так, що пікселі з однаковими мітками поділяють певні характеристики [10]-[13].

Сегментація ділиться на автоматичну та інтерактивну. Автоматична сегментація не вимагає участі користувача, інтерактивна-вимагає. Для даного проекту в зв'язку з умовами завдання підходять тільки методи автоматичної сегментації [10]-[13].

Виділяють наступні алгоритми сегментації [10]-[13].

1. Порогові методи. Перетворюють зображення в відтінки сірого до бінарного, чорно-білого на підставі порога. Дані методи добре підходять для зображень, області яких значно відрізняються за яскравістю.

2. Методи, засновані на кластеризації. Вирішують задачу кластеризації, то є розбивають обраний простір ознак на класи на основі

щільність розподілу. Дані методи зручні, коли відомо кількість класів об'єктів, що містяться в зображенні.

3. Методи з використанням гістограми. Обчислюється гістограма по всіх пікселях зображення і її мінімуми і максимуми використовуються для знаходження кластерів на зображенні. Можна порівнювати колір або яскравість. Ефективна за рахунок того, що вимагає всього один прохід по пікселях. Недоліком цього методу є те, що йому буває складно знайти помітні максимуми і мінімуми на зображенні.

4. Виділення країв. В даному випадку завдання сегментації приводиться до задачі пошуку меж регіонів. Напівтонове зображення розглядається як функція (x, y) і шукаються максимуми градієнта даної функції. Для цього використовуються методи диференціальної геометрії. Недоліком методу є те, що часто знайдені краї є розірваними, а для сегментації необхідні замкнуті контури. Дані методи найчастіше використовуються як основа для інших методів сегментації.

5. Методи розростання областей. Основа-угруповання сусідніх елементів з однаковими або близькими рівнями яскравості, що призводить до об'єднання в однорідні області. Підходять, якщо має місце стійка зв'язність всередині окремих сегментів.

6. Методи розрізу графу. Зображення розглядається як граф з вершинами – пікселями і дугами переходів між ними. Ціна переходу визначається ваговою функцією. Проводиться мінімальний розріз графа.

Так як для сегментації потрібно задавати багато параметрів, які для кожної конкретної задачі підбираються евристично, зараз одним з підходів для вирішення даного завдання є МН. Перспективним напрямком є методи сегментації, що навчаються [10]-[13].

1.3.2 Вибір ознак об'єкту

Для подальшого МН об'єкт повинен бути представлений в ознаковому поданні і слід сформуванати вектор ознак об'єкта (дескриптор) [7]-[9].

Дескриптори являють собою структуру для опису властивостей об'єкта [7]-[9].

Дескриптори поділяються наступним чином [7]-[9].

1. Глобальний. Їх отримують на основі аналізу зображення в цілому. Використовуються при вирішенні завдання пошуку за загальною подобою і характеризуються невисоким рівнем якості. Корисні в випадку, якщо доступний сегментований об'єкт.

Глобальні дескриптори можуть описувати [7]-[9]:

- колір;
- текстуру (визначається по розподіленню рівня яскравості на зображенні): матриці суміжності (матриці спільного розподілу яскравості на зображенні – матриці збігів сірого рівня), ознаки Тамури, вейвлет-аналіз;
- форму об'єктів. Найпростіші ознаки – центр ваги фігур, площа, напрямок головної осі. Методи діляться на методи засновані на зовнішньому поданні (контур об'єкта, дескриптори кордонів) і внутрішньому поданні (фігура в цілому, дескриптори областей).

2. Локальний. Локальні дескриптори описують певну область зображення. Для цього спочатку знаходяться ключові точки. Локальні дескриптори більш стійкі до трансформацій зображення [7]-[9].

Відомі алгоритми опису локальних особливостей: Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Good Features To Track (GFTT), Gradient location-orientation histogram (GLOH), Дескриптори семейства HOG (SHOG, PHOG, OHOG, GHOG), Binary Robust Independent Elementary Features (BRIEF), Oriented FAST and Rotated BRIEF (ORB).

1.3.3 Навчання класифікатора та класифікація

Як вже було описано вище, класифікація являє собою один з розділів МН [7]-[9].

Виділяють наступні основні методи розв'язання задачі класифікації [7]-[9].

1. Байєсівський Класифікатор (статистична класифікація). Заснований на оцінюванні щільності розподілу класів за навчальною вибіркою [7]-[9].

2. Нейронні мережі. Засновані на принципі коннективізму – в них з'єднується велика кількість відносно простих елементів, а навчання зводиться до побудови оптимальної структури зв'язків і налаштування параметрів зв'язків [7]-[9].

В даний момент дуже добре працюють згорткові нейронні мережі. Але їх недоліком є схильність до перенавчання і впливає з цього необхідність великої навчальної вибірки [7]-[9].

3. Лінійний роздільник. Заснований на явному побудові розділяє поверхні в просторі об'єктів. Одним з популярних алгоритмів навчання лінійного роздільника є метод опорних векторів (SVM).

4. Індукція правил (пошук закономірностей). Логічні алгоритми класифікації являють собою композиції простих, легко інтерпретованих правил. Одним з популярних алгоритмів навчання лінійного роздільника є метод Random Trees.

5. Алгоритмічна композиція. Являє собою об'єднання декількох алгоритмів в композицію, з розрахунком на те, що їх похибки взаємно компенсуються.

6. Скорочення розмірності.

7. Вибір моделі.

1.4 Використання методів навчання класифікатора для РО серед аналогів

Серед прямих аналогів, що мають подібні з системою, що проектується виділяються наступні [7]-[9].

1. Intelligent Garbage Classifier. Intelligent Garbage Classifier є системою для візуальної класифікації та поділу (сортування) твердих відходів. Дана система призначена для використання на підприємствах по сортуванню сміття.

2. Lazy Cleaner 9000. Lazy Cleaner 9000 є системою для автоматичного розпізнавання сміття. Розпізнавання проводиться на основі відмінності сегментів зображення від середнього кольору зображення.

3. Trash-Finder. Trash Finder є системою автоматичного розпізнавання певних типів сміття (банку, упаковка від мила, м'яч). Відомостей про алгоритми роботи даної системи не представлено.

Порівняльна в табличному вигляді аналогів і проектованої системи представлена у табл. 1.1.

Таблиця 1.1 – Порівняння методів навчання класифікатора для РО серед аналогів

Критерії порівняння	Intelligent Garbage Classifier	Lazy Cleaner 9000	Власна розробка
1	2	3	4
Метод сегментації	Пороговий метод, сегментація Watershed	Розрахунок середнього значення кольору RGB і виділення сильно відрізняються від нього частин зображення	Сегментація GrabCut та метод сегментації на графах

Продовження таблиці 1.1

1	2	3	4
Дескриптори	Дескриптор Фур'є, Hu moments дескриптор	Відсутні	Дескриптор локальних особливостей, а також дескриптори центру мас, гістограми кольору, гістограми кутів, тощо
Класифікатор	K-local hyperplane nearest neighbours	Відсутній	SVM та Random forest
Інтерфейс користувача	Windows Presentation Foundation	Не надається	Windows Presentation Foundation

1.5 Постановка завдання

Метою дипломної кваліфікаційної роботи магістра є створення методу та його реалізація через програмний комплекс для класифікації певного типу об'єктів і розпізнавання даних об'єктів по зображеннях з декількох точок.

Проектована система призначена для застосування роботами при зборі сміття.

Зображення з декількох точок передбачається використовувати для підвищення надійності класифікації.

Система повинна відповідати наступним вимогам:

а) застосунок повинен забезпечувати можливість навчання класифікатора:

1) система повинна бути надана можливість ввести шлях до каталогу, що містить навчальну вибірку:

- запит на введення шляху;
- шлях для доступу до даних для тренування вводиться як аргумент командного рядка при запуску програми;
- якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;

б) застосунок повинен забезпечувати можливість бінарної класифікації об'єкта на зображенні:

1) додавання шляхів до зображень для класифікації повинно відбуватися через консоль при запиті програми:

- якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;

2) повинна бути забезпечена можливість введення до чотирьох зображень;

3) результат класифікації виводиться на консоль;

4) повинна бути забезпечена можливість перервати цикл класифікації:

- після переривання циклу класифікації повинен бути виданий результат;

5) якщо відсутній необхідний класифікатор, на консоль виводиться повідомлення про помилку;

в) застосунок повинен забезпечувати можливість бінарної класифікації об'єкта на зображенні в режимі тестування;

1) повинна бути надана можливість ввести шлях до каталогу, що містить каталоги з тестовими даними:

- потреба введення шляху шлях;
- шлях вводиться як аргумент командного рядка при запуску програми;

- якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;
- якщо відсутній необхідний класифікатор, на консоль виводиться повідомлення про помилку;
- г) повинна бути забезпечена можливість вибору параметрів роботи програми;
 - 1) вибір параметрів роботи програми проводиться за допомогою аргументів командного рядка при запуску програми;
 - 2) повинна бути забезпечена можливість вибору методу сегментації;
 - 3) повинна бути забезпечена можливість вибору дескрипторів;
 - 4) повинна бути забезпечена можливість вибору класифікатора;
- д) надійність розпізнавання повинна становити не менше 80%;
- ж) система повинна займати не більше 2 МБ в пам'яті.

1.6 Висновки за розділом 1

У цьому розділі були вирішені наступні завдання:

- а) була визначена мета роботи;
- б) були визначені основні вимоги до проектованої системи;
- в) був проведений аналіз предметної області. Були визначені основні етапи алгоритму класифікації зображення: сегментація зображення, створення вектора ознак, навчання класифікатора/розпізнавання. Були розглянуті основні підходи кожної з даних завдань;
- г) був проведений аналіз аналогів. Були визначені характеристики таких систем, як Intelligent Garbage Classifier і Lazy Cleaner 9000 і проведено їх порівняння з проектованою системою.

2 МАТЕРІАЛИ ТА МЕТОДИ

В першому розділі було проаналізовано сучасне становище проблемної області РО. Додаткову увагу було приділено аналізу задач РО, методам сегментації, вибору ознак об'єктів та навчання класифікаторів.

В даний час існує безліч завдань, в яких потрібно прийняти деяке рішення в залежності від присутності на зображенні об'єкта або класифікувати його [7]-[9]. Здатність розпізнавати вважається основною властивістю біологічних істот, в той час як комп'ютерні системи цією властивістю в повній мірі не володіють [7]-[9].

Саме тому актуальною залишається задача розробки методів для автоматизації процесів РО та прийняття рішень при РО.

Метою розділу є аналіз існуючих методів сегментації зображень та навчання класифікатора для розробки нових підходів до РО.

2.1 Використання методів сегментації зображень при розробці методів РО

Сегментація зображення-це розбиття зображення на безліч покривають його областей. Сегментація застосовується в багатьох областях, наприклад, у виробництві для індикації дефектів при складанні деталей, в медицині для первинної обробки знімків, також для складання карт місцевості по знімках із супутників. Розглянемо кілька методів, що гарно імплементуються до бібліотеки комп'ютерного зору OpenCV [10]-[13].

2.1.1 Метод сегментації GrabCut

Це інтерактивний метод виділення об'єкта, розроблявся як більш зручна альтернатива магнітному ласо (щоб виділити об'єкт, користувачеві було потрібно обвести його контур за допомогою миші). Для роботи методу

досить укласти об'єкт разом з частиною фону в прямокутник (grab). Сегментування об'єкта відбудеться автоматично (cut) [10]-[13].

Можуть виникнути складнощі при сегментації, якщо всередині обмежує прямокутника присутні кольори, які зустрічаються у великій кількості не тільки в об'єкті, але і на тлі. В цьому випадку можна поставити додаткові мітки об'єкта (червона лінія) і фону (синя лінія) [10]-[13].

Розглянемо ідею методу. За основу взято алгоритм інтерактивної сегментації GraphCut, де користувачеві треба поставити маркери на фон і на об'єкт. Зображення розглядається як масив $z(z_1, z_2, \dots, z_n, \dots, z_N)$. z – значення інтенсивності пікселів, N – загальне число пікселів. Для відділення об'єкта від фону метод визначає значення елементів масиву прозорості $a(a_1, a_2, \dots, a_n, \dots, a_N)$, причому a_n може приймати два значення, якщо $a_n = 0$, значить піксель належить фону, якщо $a_n = 1$, то об'єкту. Внутрішній параметр θ містить гістограму розподілу інтенсивності переднього плану і гістограму фону [10]-[13]:

$$\theta = \{h(z; a), a = 0, 1\}. \quad (2.1)$$

Завдання сегментації-знайти невідомі a_n . Розглядається функція енергії:

$$E(a, \theta, z) = U(a, \theta, z) + V(a, z). \quad (2.2)$$

Причому мінімум енергії відповідає найкращій сегментації [10–13].

$$U(a, \theta, z) = -\sum_n \log h(z_n, a_n), \quad (2.4)$$

$$V(a, z) = \sum_{(m,n) \in C} \frac{1}{dis(m,n)} [a_n \neq a_m] \exp(-\beta(z_m - z_n)^2), \quad (2.5)$$

де $V(a, z)$ – доданок відповідає за зв'язок між пікселями. Сума йде по всіх парах пікселів, які є сусідами;

$dis(m, n)$ – евклідова відстань;

$[a_n \neq a_m]$ – відповідає за участь пар в сумі, якщо $a_n = a_m$, то ця пара не буде враховуватися;

$U(a, \theta, z)$ – відповідає за якість сегментації, тобто поділ об'єкта від фону.

Знайшовши глобальний мінімум функції енергії E , отримаємо масив прозорості $\hat{\alpha} = \arg \min_a E(a, \theta)$. Для мінімізації функції енергії, зображення описується як граф і шукається мінімальний розріз графа. На відміну від GraphCut в алгоритмі GrabCut пікселі розглядаються в red-green-blue (RGB) просторі, тому для опису колірної статистики використовують суміш гауссіан [10]-[13].

Як видно, із завданням відділення конкретного об'єкта від фону добре впорається GrabCut [10]-[13].

2.1.2 Сегментація за допомогою розрізів на графах

Другий спосіб виділення об'єкта на тлі також заснований на теорії графів. З усіх розглянутих способів він має найбільш природний інтерфейс: користувач просто відзначає деяку множина A пікселів, що належать об'єкту, і деяку множина B пікселів, що належать фону. Оскільки ці пікселі не зобов'язані бути поруч з кордоном, така розмітка не вимагає від користувача особливих зусиль. Результатом методу служить сегментація, в якій все множина A віднесено до об'єкта, а множина B – до фону [10]-[13].

Якщо результат виділення з першого разу не задовольняє користувача, він додає в вихідні множинаі пікселі, допозначає їх на зображенні. Наприклад, якщо метод помилково відніс шматок об'єкта до фону, користувач відзначає частину пікселів цього шматка як пікселі об'єкта

(множина A). Результатом перезапуску методу служить уточнена сегментація [10]-[13].

Розглянемо, як працює сам метод. На цей раз побудуємо граф на растрі наступним чином. Вершини графа розташуємо в центрі кожного пікселя. Далі під кольором вершини ми будемо розуміти колір відповідного пікселя. Будемо також вважати, що вершина належить множині A (або B), якщо відповідній множині належить відповідний піксель [10]-[13].

Кожну вершину з'єднаємо з сусідніми вершинами (тобто вершинами, розташованими в центрах 8-зв'язно сусідніх пікселів). Таким чином, отримані ребра з'єднують центри сусідніх пікселів. Припишемо кожному ребру вагу:

$$\lambda \frac{1}{L} \exp(-\sigma \rho(C_1, C_2)), \quad (2.6)$$

де L – геометрична довжина ребра;

C_1 та C_2 – кольори вершин, що з'єднуються ребром;

λ та σ – деякі (позитивні) параметри. Відзначимо, що дана вага тим менше, чим більше різниця між кольорами вершин.

Додамо в граф дві вершини, звані виток і сток (граф з цими вершинами називається мережею). Ці вершини будемо називати термінальними. Інші вершини будемо називати піксельними. З'єднаємо стік і витік ребрами з кожною вершиною графа. Ребрам, що з'єднує витік з вершинами множини A , і ребрам, що з'єднує стік з вершинами множини B , припишемо нескінченну вагу [10]-[13].

Розглянемо розподіл кольорів вершин множини A (як гістограму або як набір кластерів, отриманих методом k -середніх). Для всіх піксельних вершин не з множини A , припишемо ребрам, що з'єднує їх з витком, вага, пропорційний узгодженості їх кольору з цим розподілом кольорів. Наприклад, для гістограмного розподілу узгодженість відповідає потужності відповідної корзини. Таким чином, вага ребра буде тим більше, чим більше

схожий колір вершини на кольори вершин множини A . аналогічну процедуру виконаємо для множини B і ребер, що з'єднують Піксельні вершини зі стоком. В результаті всім ребрам нашого розширеного графа будуть приписані ваги [10]-[13].

Назвемо розрізом на нашому графі розбиття всіх його вершин на дві непересічні множини – витокова і стокова, причому витокова множина повинна містити витік, а Стокова множина, відповідно, стік. Будемо називати ребра, що з'єднують вершини з різних множин, розрізаними. Розріз можна також представляти як розрізає поверхню, що перетинає всі розрізані ребра і відокремлює витік від стоку. Будь-який розріз задає сегментацію вихідного зображення наступним чином: будемо вважати, що вершини, що потрапили в витокове множина (крім самого витоку), відповідають пікселям об'єкта, а інші, що потрапили в стокове множина (крім самого стоку), відповідають пікселям фону [10]-[13].

Вагою розрізу назвемо суму ваг всіх розрізаних ребер. Мінімальним розрізом назвемо розріз з мінімальною вагою. Зауважимо, що всього розрізів 2^P , де P – число пікселів (так як кожна піксельна вершина може виявитися або в витоковому, або в стоковому множини). Але при цьому існує метод пошуку мінімального розрізу на графі (відомий також як метод пошуку максимального потоку в мережі), що дозволяє знайти мінімальний розріз за поліноміальний від P час, причому на практиці для графів, схожих на описаний в даному розділі, він працює за лінійне від P час. Цей широко використовується в сучасній обчислювальній дискретної математики метод можна знайти в багатьох підручниках [10]-[13].

Мінімальний розріз буде проходити так, що [10]-[13]:

Пікселі множини A будуть віднесені до об'єкта, пікселі множини B – до фону. Це гарантується нескінченністю ваг ребер, що з'єднують відповідні Піксельні вершини з відповідними термінальними.

Межа між об'єктом і фоном буде проведена по можливості між пікселями з сильно відрізняються квітами. Це забезпечується вибором ваг для ребер, що з'єднують Піксельні вершини (2.6).

Пікселі, схожі за кольором на пікселі множини A , будуть по можливості віднесені до об'єкта, а пікселі, схожі за кольором на пікселі множини B , – до фону. Це гарантується вибором ваг ребер, що з'єднують відповідні Піксельні вершини з відповідними термінальними.

Останні два пункти можуть входити в протиріччя один з одним. Наприклад, ділянка зображення може бути схожий за кольором на фон (пікселі множини B), але оточений навколо по периметру пікселями множини A і при цьому не відділений від них різкою межею. У таких випадках вибір параметра λ у формулі (2.6) встановлює баланс між останніми двома пунктами. Збільшуючи значення λ , ми збільшуємо важливість того, щоб межа між фоном і об'єктом проходила між пікселями з різними кольорами, а зменшуючи, ми збільшуємо важливість того, щоб пікселі, схожі за кольором на пікселі множини $A(B)$, були віднесені до об'єкта (фону) [10]-[13].

2.1.3 Метод МН класифікатора SVM

Як відомо, завдання МН розділені на дві основні категорії – класифікація і регресія. Залежно від того, яке з цих завдань стоїть, і який є датасет для цього завдання, слід обирати який саме метод використовувати [14]-[16].

Метод SVM – це лінійний алгоритм використовуваний в задачах класифікації та регресії. Даний алгоритм має широке застосування на практиці і може вирішувати як лінійні так і нелінійні завдання. Суть роботи SVM проста: метод створює лінію або гіперплощину, яка розділяє дані на класи [14]-[16].

Основним завданням методу є знайти найбільш правильну лінію, або гіперплощину, яка розділяє дані на два класи. SVM це метод, який отримує на вході дані, і повертає розділяючу лінію [14]-[16].

Розглянемо наступний приклад. Припустимо у нас є набір даних, і ми хочемо класифікувати і розділити червоні квадрати від синіх кіл (припустимо позитивне і негативне). Основною метою в даній задачі буде знайти ідеальну лінію яка розділить ці два класи [14]-[16].

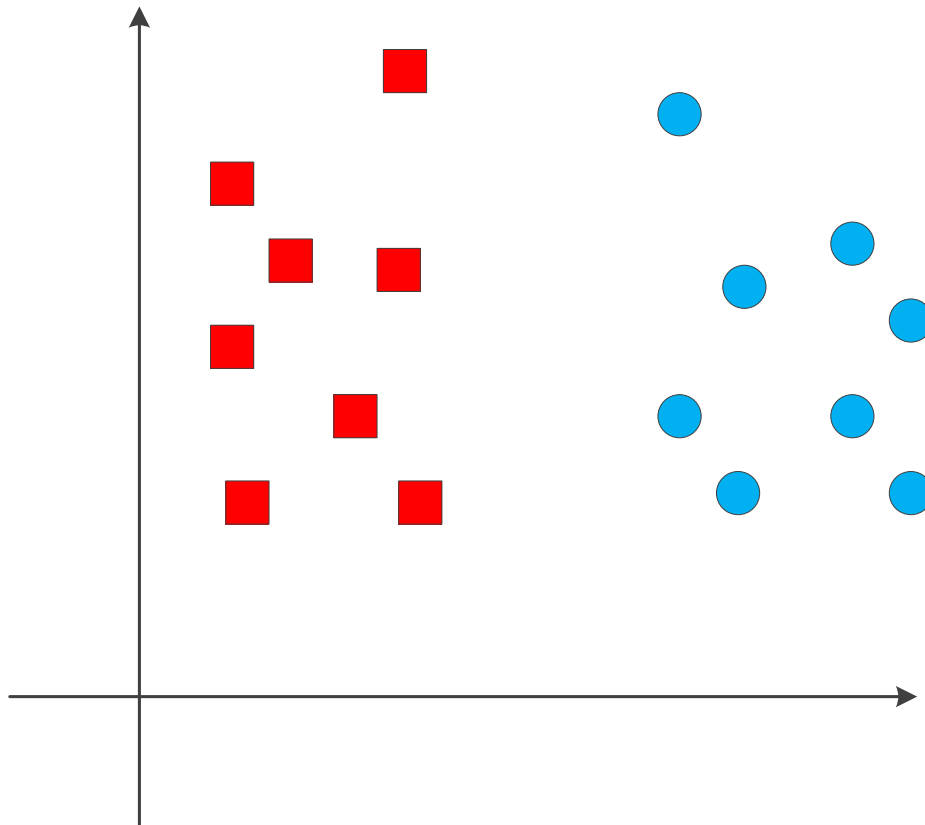


Рисунок 2.1 – Приклад завдання для SVM

Слід знайти ідеальну лінію, або гіперплощину, яка розділить набір даних на синій і червоний класи [14]-[16].

Як можна помітити, немає однієї, унікальної, лінії, яка б вирішувала таке завдання. Можливо підібрати безліч таких ліній, які можуть розділити ці два класи. Як же саме SVM знаходить ідеальну лінію, і що слід розглядати, як ідеальний розподіл [14]-[16].

На наступному прикладі можна побачити приклад, коли одразу декілька ліній підходять для розділення класів екземплярів – рис. 2.2.

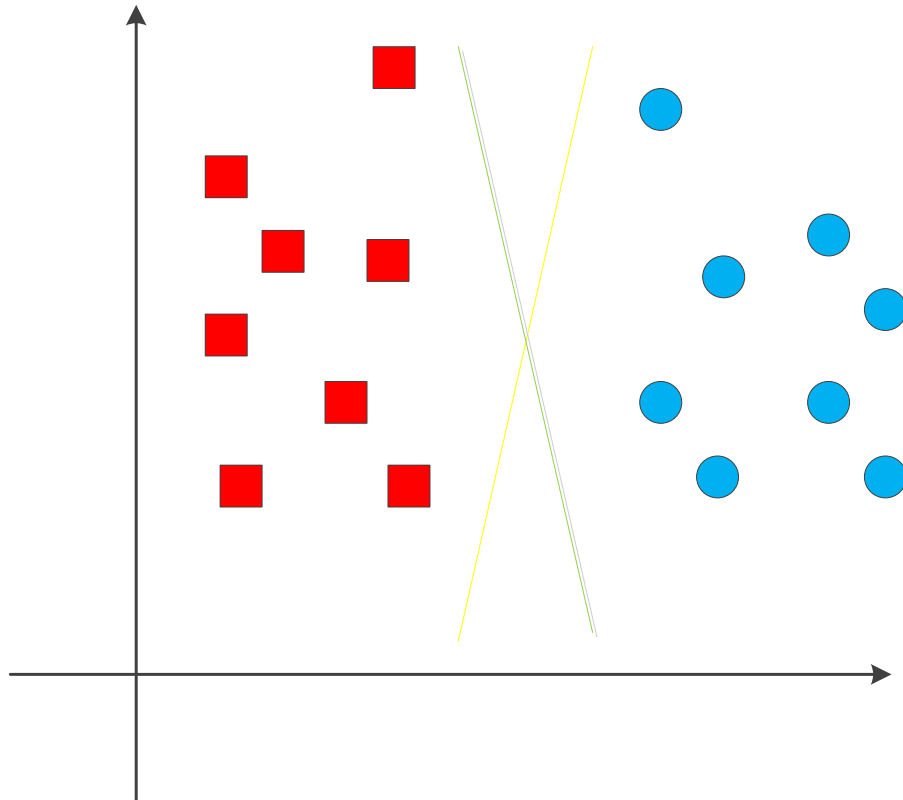


Рисунок 2.2 – Варіанти розділу екземплярів за класами

Якщо обрати жовту лінію, це правильно – це та сама лінія, яку б вибрав метод SVM. В даному прикладі можна інтуїтивно зрозуміти що жовта лінія розділяє і відповідно класифікує два класи краще зеленої [14]-[16].

У випадку з зеленою лінією – вона розташована занадто близько до червоного класу. Незважаючи на те, що вона вірно класифікувала всі об'єкти поточного набору даних, така лінія не буде генералізованою – не буде так само добре поводитися з незнайомим набором даних. Завдання знаходження генералізованої розділяє двох класів є одним з основних завдань в МН.

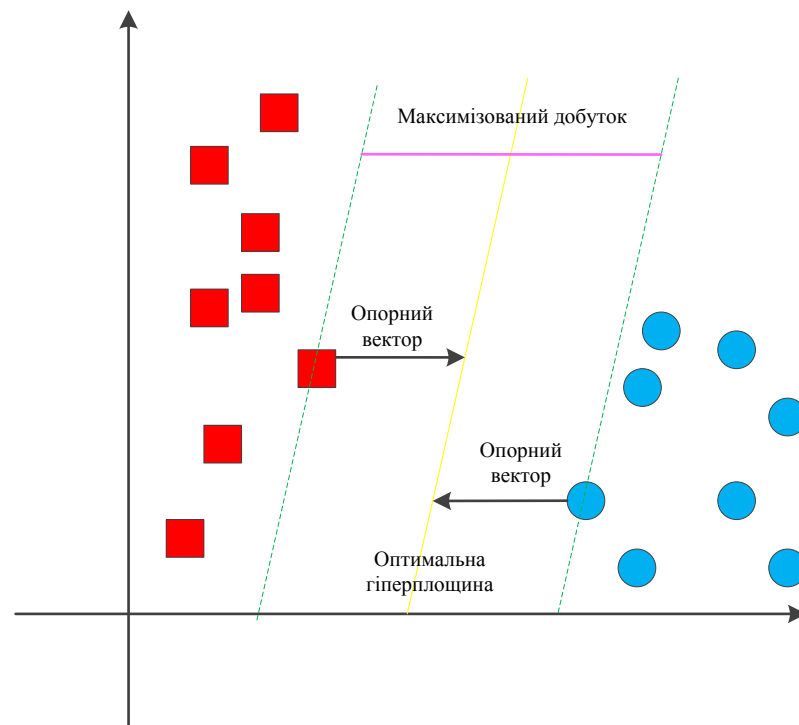


Рисунок 2.3 – Обґрунтування вибору оптимальної гіперплощини

Розглянемо наступний приклад, з більш складним датасетом, який не можна розділити лінійно – рис. 2.4.

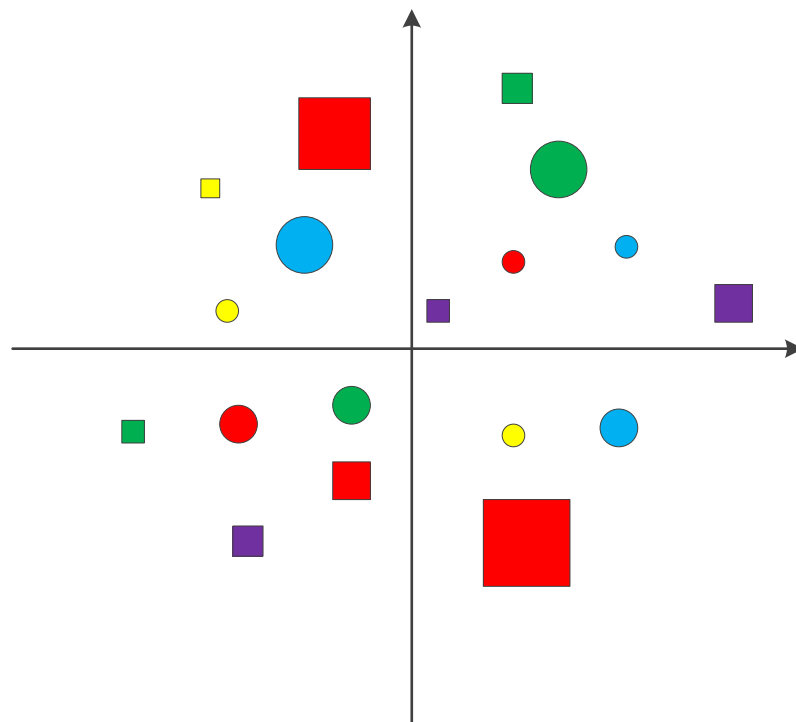


Рисунок 2.4 – Приклад більш складного датасету

Очевидно, що цей набір даних не можна розділити лінійно. Ми не можемо накреслити пряму лінію, яка б класифікувала ці дані [14]-[16]. Але, цей датасет можна розділити лінійно, додавши додатковий вимір, яке ми назвемо віссю z . уявімо, що координати на осі z регулюються наступним обмеженням [14]-[16]:

$$z = x + y. \quad (2.7)$$

Таким чином, ордината z представлена з квадрата відстані точки до початку осі.

Нижче наведена візуалізація того ж набору даних, на осі z – рис. 2.5.

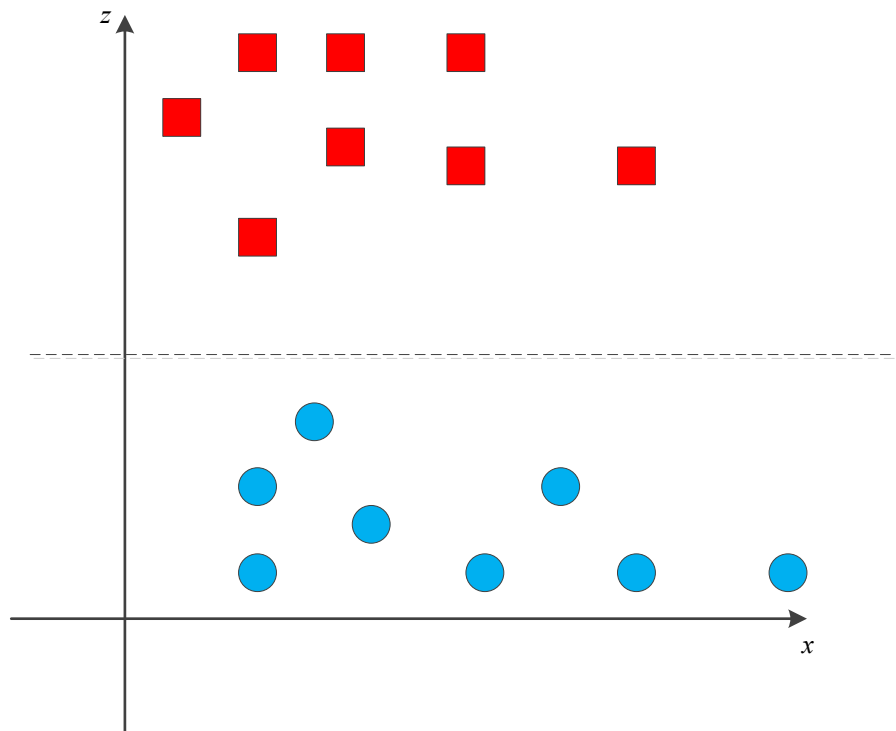


Рисунок 2.5 – Візуалізація того набору даних, на осі z

Тепер дані можна розділити лінійно. Припустимо, що лінія розділяє дані $z = k$, де k константа. Якщо $z = x + y$, то отже і:

$$k = x + y, \quad (2.7)$$

– формула кола. Таким чином, ми можемо спроектувати наш лінійний роздільник, назад до початкової кількості вимірювань вибірки, використовуючи цю трансформацію [14]-[16].

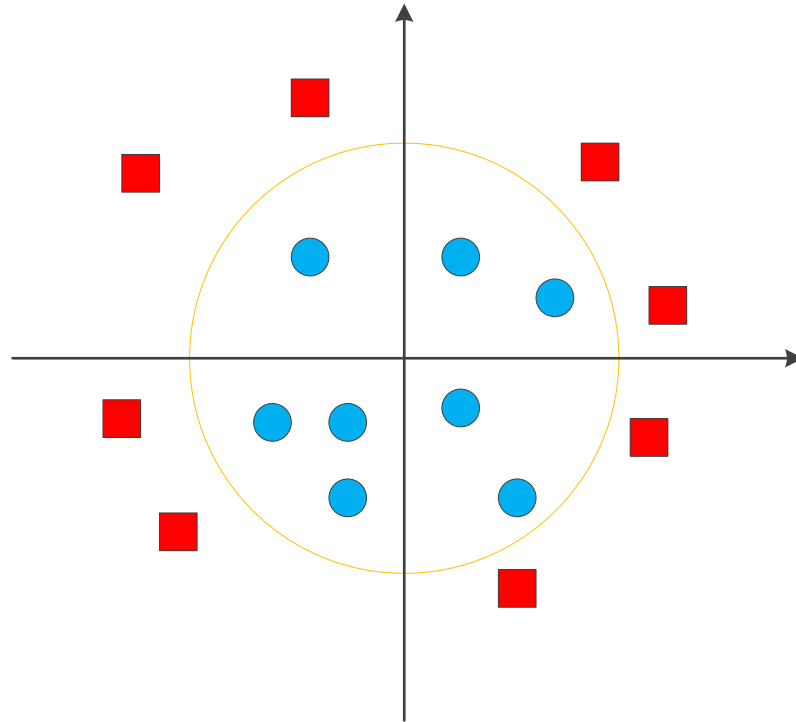


Рисунок 2.6 – Визначення гіперплощини

В результаті, ми можемо класифікувати нелінійний набір даних додавши до нього додатковий вимір, а потім, привести назад до вихідного вигляду використовуючи математичну трансформацію [14]-[16]. Однак, не з усіма наборами даних можна з такою ж легкістю повернути таку трансформацію.

Гіперплощина. Тепер, після ознайомлення з логікою методу, перейдемо до формального визначення гіперплощинності [14]-[16].

Гіперплоскість – це $n-1$ -мірна підплощина в n -мірному евклідовому просторі, яка розділяє простір на дві окремі частини.

Наприклад, уявімо що лінія представлена у вигляді одновимірного евклідова простору (тобто датасет лежить на прямій). Виберіть точку на цій лінії. Ця точка розділить набір даних, в нашому випадку лінію, на дві

частини. У лінії є одна міра, а у точки 0 заходів. Отже, точка – це гіперплоскість лінії [14]-[16].

Для двовимірного датасета, приклад якого був наведений раніше, розділяюча пряма була тією самою гіперплоскістю. Простіше кажучи, для n -мірного простору існує $n-1$ -мірна гіперплоскість, що розділяє цей простір на дві частини [14]-[16].

Параметри – це аргументи які ви передаєте при створенні класифікатора. Нижче я навів кілька найважливіших параметрів SVM: C .

Даний параметр допомагає відрегулювати ту тонку грань між гладкістю і точністю класифікації об'єктів навчальної вибірки. Чим більше значення C тим більше об'єктів навчальної вибірки будуть правильно класифіковані [14]-[16].

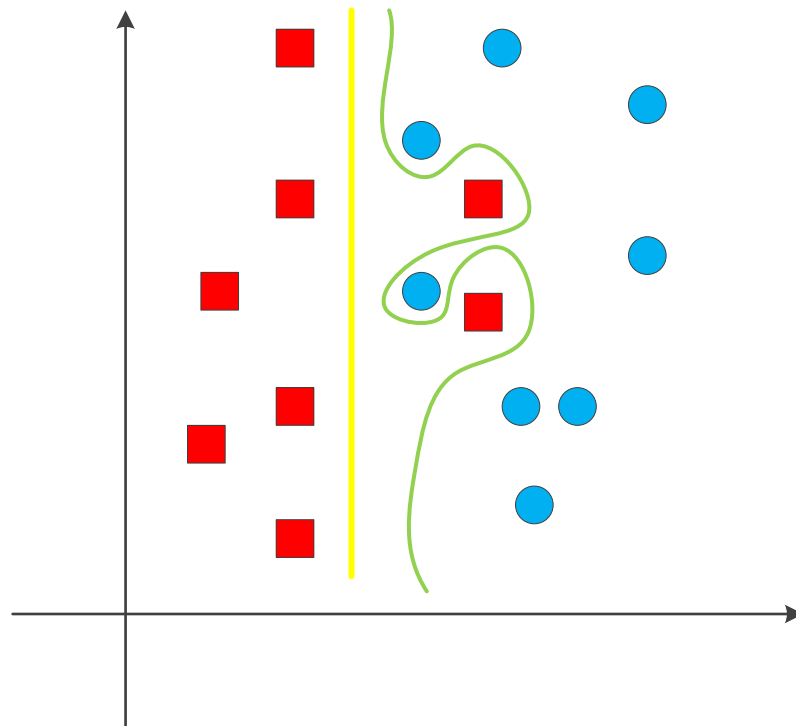


Рисунок 2.7 – Випадок неправильної класифікації

У цьому прикладі є кілька порогів прийняття рішень, які можна визначити для цієї конкретної вибірки. Слід звернути увагу на пряму (представлена на графіку – рис. 2.7 – у вигляді жовтої лінії) порога рішень. Вона досить таки проста, і з цієї причини, кілька об'єктів були класифіковані

невірно. Ці точки, які були класифіковані неправильно називаються викидами в даних [14]-[16].

Також можливо налаштувати параметри таким чином, що в кінцевому підсумку отримаємо більш вигнуту лінію (зелена – поріг рішень), яка буде класифікувати асбсолютно всі дані навчальної вибірки правильно. Звичайно, в такому випадку, шанси того, що отримана модель зможе генералізувати і показати настільки ж хороші результати на нових даних, катастрофічно мала. Отже, якщо намагаються досягти точності при навчанні моделі, варто націлитися на щось більш рівне, пряме. Чим вище число C тим більше заплутана гіперплощинність буде у вашій моделі, але і вище число вірно-класифікованих об'єктів навчальної вибірки. Тому, важливо підкручувати параметри моделі під конкретний набір даних, щоб уникнути перенавчання але, в той же час досягти високої точності [14]-[16].

Гама: γ . В офіційній документації говориться, що γ визначає наскільки далеко кожен з елементів в наборі даних має вплив при визначенні ідеальної лінії. Чим нижче γ , тим більше елементів, навіть тих, які досить далекі від розділяє лінії, беруть участь в процесі вибору цієї самої лінії. Якщо ж, γ висока, тоді алгоритм буде спиратися тільки на тих елементах, які найбільш близькі до самої лінії [14]-[16].

Якщо задати рівень гами занадто високим, тоді в процесі прийняття рішення про розташування лінії будуть брати участь тільки найближчі до лінії елементи. Це допоможе ігнорувати викиди в даних. Алгоритм SVM влаштований таким чином, що точки розташовані найбільш близько відносно один одного мають більшу вагу при прийнятті рішення. Однак при правильному налаштуванні C і γ можна домогтися оптимального результату, який побудує більш лінійну гіперплощинність і, отже, більш генералізовану [14]-[16].

2.1.4 Метод МН Random forest

На відміну від більшості використовуваних в МН методів, метод Random forest (випадкового лісу) не вимагає складної теорії [17], [18].

Метод Random forest – універсальний метод МН, суть якого полягає у використанні ансамблю вирішальних дерев [17], [18]. Саме по собі вирішальне дерево надає вкрай невисоку якість класифікації, але через велику їх кількість результат значно поліпшується [17], [18]. Також це один з небагатьох методів, який можна використовувати в абсолютній більшості завдань [17], [18].

Завдяки своїй гнучкості Random forest застосовується для вирішення практично будь-яких проблем в області МН. Сюди відносяться класифікації (RandomForestClassifier) і регресії (RandomForestRegressor), а також більш складні завдання, на зразок відбору ознак, пошуку викидів/аномалій і кластеризації [17], [18].

Основним полем для застосування алгоритму випадкового дерева є перші два пункти, рішення інших завдань будується вже на їх основі.

Порядок дій в алгоритмі [17], [18]:

- завантаження даних;
- у заданому наборі даних визначається випадкова вибірка;
- далі метод побудує за вибіркою дерево рішень;
- дерево будується, поки в кожному листі не більше N об'єктів, або поки не буде досягнута певна висота;
- потім буде отримано результат прогнозування з кожного дерева рішень;
- на цьому етапі голосування буде проводитися для кожного прогнозованого результату: ми вибираємо кращу ознаку, робимо розбиття в дереві по ньому і повторюємо цей пункт до вичерпання вибірки;
- в кінці вибирається результат прогнозу з найбільшою кількістю голосів. Це і є остаточний результат прогнозування.

Теоретична складова алгоритму випадкового дерева. У порівнянні з іншими методами МН, теоретична частина алгоритму Random forest просто. У нас немає великого обсягу теорії, необхідна тільки формула підсумкового класифікатора $a(x)$ [17], [18]:

$$a(x) = \frac{1}{N} \sum_{i=1}^N b_i(x), \quad (2.8)$$

де N – кількість дерев;

i - лічильник для дерев;

b – вирішальне дерево;

x – згенерована на основі даних вибірка.

Варто також відзначити, що для завдання класифікації ми вибираємо рішення голосуванням по більшості, а в задачі регресії – середнім.

Слід розглянути необхідні параметри алгоритму [17], [18].

1. Число дерев: `n_estimators` [17], [18]. Чим більше дерев, тим краще якість. Варто відзначити, що час настройки і роботи Random forest будуть пропорційно збільшуватися, що може позначитися на продуктивності [17], [18].

Часто при великому збільшенні `n_estimators` якість на навчальній вибірці може навіть доходити до 100%, в той час як якість на тесті виходить на асимптоту, що сигналізує про перенавчання нашої моделі. Кращий спосіб уникнути цього – прикинути, скільки дерев вам достатньо, зафіксувавши момент, коли якість тесту ще не стає стабільно-незмінним.

2. Критерій розщеплення: `criterion`. Також один з найважливіших параметрів для побудови, але без значної можливості вибору. У бібліотеці `sklearn` для задач класифікації реалізовані критерії `gini` і `entropy`. Вони відповідають класичним критеріям розщеплення: `gini` та `entropy`.

У свою чергу, для задач регресії реалізовані два критерії (mse і mae), які є функціями помилок Mean Square Error і Mean Absolute Error відповідно. Практично у всіх завданнях використовується критерій mse [17], [18].

Простий метод перебору допоможе вибрати, що використовувати для вирішення конкретної проблеми [17], [18].

3. Число ознак для вибору розщеплення: `max_features`. При збільшенні `max_features` збільшується час побудови лісу, а дерева стають схожими один на одного. У задачах класифікації він за замовчуванням дорівнює $\sqrt{n}$, в задачах регресії – $n/3$.

Є одним з найважливіших параметрів в методі. Він налаштовується в першу чергу, після того, як ми визначили тип нашого завдання.

4. Мінімальне число об'єктів для розщеплення: `min_samples_split`. Другорядний за своїм значенням параметр, його можна залишити в стані за замовчуванням.

5. Обмеження кількості об'єктів у листі: `min_samples_leaf`. Аналогічно з `min_samples_split`, але при збільшенні даного параметра якість моделі на навчанні падає, в той час як час побудови моделі скорочується.

6. Максимальна глибина дерев: `max_depth`. Чим менше максимальна глибина, тим швидше будується і працює метод випадкового дерева.

При збільшенні глибини різко зростає якість як на навчанні моделі, так і на її тестуванні. Якщо у вас є можливість і час для побудови глибоких дерев, то рекомендується використовувати максимальне значення даного параметра.

Неглибокі дерева рекомендується використовувати в завданнях зі значною кількістю шумових об'єктів (викидів) [17], [18].

Серед переваг методу зазвичай виділяють [17], [18]:

- має високу точність передбачення, яка порівнянна з результатами градієнтного бустингу;
- не вимагає ретельного налаштування параметрів, добре працює з коробки;

- практично не чутливий до викидів в даних через випадкове семплювання (random sample);
- не чутливий до масштабування і до інших монотонних перетворень значень ознак;
- рідко перенавчається. На практиці додавання дерев тільки покращує композицію;
- у разі наявності проблеми перенавчання, вона долається шляхом усереднення або об'єднання результатів різних дерев рішень;
- здатний ефективно обробляти дані з великим числом ознак і класів;
- добре працює з пропущеними даними – зберігає хорошу точність навіть при їх наявності;
- однаково добре обробляє як безперервні, так і дискретні ознаки;
- висока паралелізуємость і масштабованість.
- Серед недоліків методу виділяють [17], [18]:
- для реалізації методу випадкового дерева потрібен значний обсяг обчислювальних ресурсів;
- великий розмір моделей;
- побудова випадкового лісу забирає більше часу, ніж дерева рішень або лінійні методи;
- метод схильний до перенавчання на зашумлених даних;
- немає формальних висновків, таких як P-values, які використовуються для оцінки важливості змінних;
- на відміну від більш простих методів, результати випадкового лісу складніше інтерпретувати;
- коли у вибірці дуже багато розріджених ознак, таких як Тексти або набори слів (bag of words), метод працює гірше ніж лінійні методи;

- на відміну від лінійної регресії, Random forest не володіє можливістю екстраполяції. Це можна вважати і плюсом, так як в разі викидів не буде екстремальних значень;
- якщо дані містять групи ознак з кореляцією, які мають схожу значимість для міток, то перевага віддається невеликим групам перед великими, що веде до недонавчання;
- процес прогнозування з використанням випадкових лісів дуже трудомісткий в порівнянні з іншими методами.

Метод Random forest – це універсальний метод МН учителем. Його можна використовувати в безлічі завдань, але в основному він застосовується в проблемах класифікації та регресії [17], [18].

Дієвий і простий в розумінні, метод має значний недолік – помітне уповільнення роботи при певному налаштуванні параметрів всередині нього.

Метод Random forest можна використовувати, якщо потрібні надзвичайно точні результати або у є величезний обсяг даних для обробки, і потрібен досить потужний метод, який дозволить вам ефективно обробити всі дані [17], [18].

2.2 Проектування алгоритмів роботи методів РО

2.2.1 Робота модуля навчання класифікатора

Алгоритм роботи модуля навчання класифікатора складається з декількох етапів.

На початку на вхід подається шлях до каталогу з даними для тренування.

Після чого, якщо каталогу не існує або його вміст не відповідає вимогам програми – видається повідомлення про помилку і проводиться вихід з програми.

Далі відбувається завантаження зображення і створення їх масиву, а також масиву відповідних їм міток класу.

Тепер може розпочинатися сегментація кожного з зображень об'єктом класу похідного від `Segmenter`, зазначеного в структурі `Params`.

Після чого відбувається створення об'єкта класу `Segment` для кожного із зображень.

Після цього можливо виконати розрахунок дескрипторів, зазначених у структурі `Params`, для кожного сегмента.

Блок-схема алгоритму наведена на рис. 2.8.



Рисунок 2.8 – Робота модуля з навчання класифікатору

Тепер можна створити для кожного сегмента масиву, що включає всі розраховані ознаки в певному порядку.

Наступним кроком стане створення класифікатора класу похідного від Classifier, зазначеного в структурі Params.

Тепер починається навчання класифікатора на основі результуючого масиву ознак для кожного зображення і відповідної йому мітки класу.

Фіналом є збереження класифікатора під зазначеним у класі classifier ім'ям.

2.2.2 Робота модуля класифікації

Алгоритм роботи модуля класифікації складається з наступних етапів:

а) модуль вступає в цикл класифікації зображень (до чотирьох ітерацій):

1) на кожній ітерації циклу модуль отримує рядок, що містить адресу зображення або код виходу;

2) якщо отримано код виходу, то проводиться розрахунок середнього значення попередніх результатів і видається остаточна відповідь;

3) якщо за даною адресою немає зображення, видається повідомлення про помилку і проводиться завершення роботи модуля;

4) завантаження зображення;

5) сегментація зображення об'єктом класу похідного від Segmenter, зазначеного в структурі Params і отримання крайніх точок найбільш підходящого сегмента. Створення об'єкта класу Segment для сегмента, виданого як результат попереднього етапу;

6) розрахунок дескрипторів, зазначених у структурі Params, для сегмента;

7) створення для сегмента масиву, що включає всі розраховані ознаки в певному порядку;

8) завантаження класифікатора класу похідного від Classifier, зазначеного в структурі Params;

9) якщо Класифікатор не завантажений, модуль видає повідомлення про помилку і завершує роботу;

10) у Класифікатор подається результуючий масив ознак;

11) Класифікатор видає рішення у вигляді числа float, що характеризує дальність об'єкта на зображенні від кордону, що розділяє класи.

12) збереження результату в масив результатів;

13) розрахунок середнього значення попередніх результатів і, якщо воно далі від межі (точніше) порогового, видача остаточного рішення;

б) після закінчення роботи циклу модуль перевіряє, чи далі середнє значення попередніх результатів від межі (точніше), ніж мінімальне порогове, і, якщо далі, видає остаточне рішення;

в) якщо середнє значення попередніх результатів ближче від кордону, ніж мінімальне порогове, видається негативний результат.

Блок-схема алгоритму наведена на рис. 2.9.

2.2.3 Робота модуля фільтрації сегментів

Алгоритм роботи модуля фільтрації сегментів складається з наступних етап:

1. На вхід подається вихідне зображення і матриця з мітками сегментів.

2. Модуль отримує зображення після обробки. Даний метод реалізує початкову стадію сегментації, повертає фільтроване, постеризоване (тобто відображене з маленькою кількістю тонів) зі згладженими колірними градієнтами і дрібно-зерною текстурою.

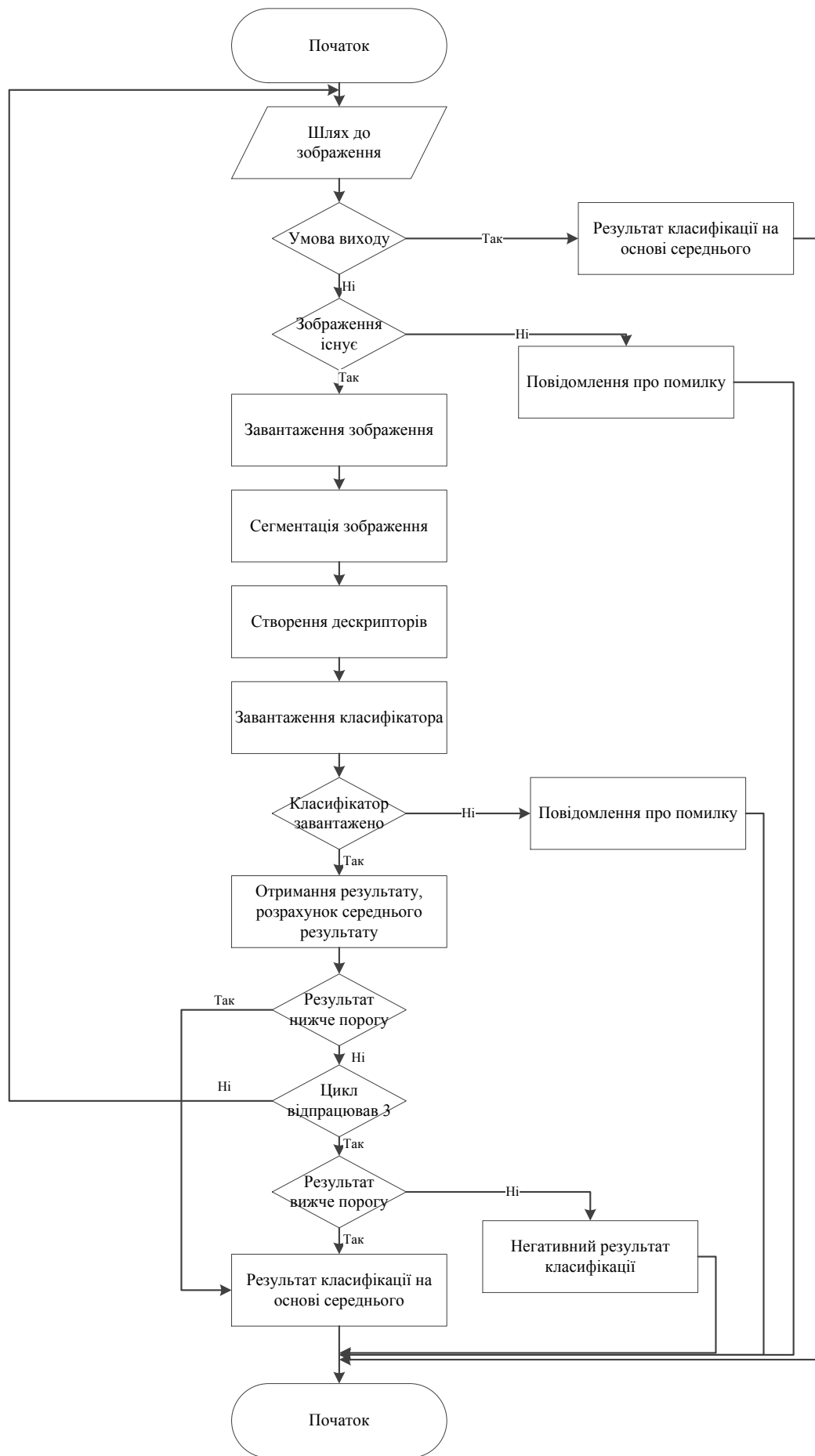


Рисунок 2.9 –Робота модуля класифікації

3. Обчислюється зображення з контурами за алгоритмом Кенні. Для більшої кількості знайдених контурів два зображення з контурами (одне - з вихідним зображенням як вхідним, інше – із зображенням, отриманим в результаті).

4. На підставі відсутності значної (визначається порогом) кількості меж Кенні між сегментами з вхідного сегмента зображення, сегменти склеваються. При цьому сегменти, які стосуються кордонів не розглядаються.

5. Вираховуються ключові точки на зображенні.

6. Розраховуються дескриптори ключових точок.

7. За допомогою функції OpenCV partition і внутрішнього класу equivalence ключові точки розбиваються на групи.

8. За допомогою функції OpenCV convexHull для кожної групи ключових точок (якщо точок більше 5) створюється зображення з маскою опуклого багатокутника, що включає всі точки групи.

9. Для кожного із сегментів, отриманих після п. 4, створюється об'єкт класу Segment.

10. Кожен із сегментів оцінюється за такими параметрами: віддаленість від центру зображення, розмір відносно розміру зображення, максимальна частина площі будь-якого з опуклого багатокутника, що включає згруповані ключові точки (в%), що лежить всередині даного сегмента.

11. Вибирається і повертається найбільш підходящий сегмент.

12. Якщо відповідного сегмента не виявлено, видається виняток (яке буде оброблено модулем класифікації).

2.3 Висновки за розділом 2

У другому розділі проаналізовано методи МН, що будуть використовуватися для РО. Так для сегментації зображень було обрано

методи: GrabCut та сегментація за графом. Для навчання класифікаторів РО буде використовуватися методи: SVM та Random forest.

Додатково були розроблені алгоритми роботи модуля навчання класифікатора, модуля класифікації та модуля фільтрації сегментів.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Другий розділ було присвячено дослідженню запропонованих методів для автоматизації процесу РО за графічним зображенням.

Третій розділ буде присвячено висвітленню питань з проектування архітектури та реалізації ПЗ для дослідження запропонованих методів та можливих модифікацій. Особливу увагу буде присвячено вибору технічних засобів: мови програмування, середовища розробки, тощо.

3.1 Вибір технології розробки

Для реалізації була обрана мова C++. Так як при розробці даного проекту необхідно використовувати бібліотеку комп'ютерного зору OpenCV, то, так як вона надає інтерфейс тільки для мов програмування C, C++, Java, Python, вибір проводився тільки серед них. При порівнянні C++, Python і Java був обраний C++ на підставі наведених нижче переваг [19], [20].

Переваги C++ [19], [20]:

- а) OpenCV написаний на C++ і його первинний інтерфейс на C++;
Всі нові розробки і алгоритми даної бібліотеки зараз створюються на C++;
- б) більшість opensource коду і бібліотек для комп'ютерного зору створюються на C++;
- в) промислова розробка в сфері комп'ютерного зору також найчастіше ведеться на C++;
- г) мова підходить для рішень, що вимагають високої продуктивності;
- д) лідер з науково-технічних додатків [14].

Для економії часу і зусиль вирішено вибирати з уже знайомих середовищ розробки для C++ [19], [20].

Таким чином вибір стояв між Visual Studio і Qt Creator. Була обрана Visual Studio.

Основними недоліки Visual Studio в порівнянні з Qt Creator є відсутність хорошого GUI редактора і кроссплатформенності, але, так як розробляється додаток є консольним і розробка ведеться в ОС Windows, дані недоліки не грають ролі.

У свою чергу, Visual Studio володіє кращим по сраненню з Qt Creator відладчиком, що і пояснює її вибір в якості середовища для розробки даного проекту [19], [20].

Також в даному проекті використовується бібліотека OpenCV (Open Source Computer Vision) [21], [22]. OpenCV – бібліотека алгоритмів комп'ютерного зору і МН з відкритим вихідним кодом. Дана бібліотека кроссплатформенна і вільна для використання під ліцензією open-source BSD. На даний момент бібліотека включає в себе більше 2500 оптимізованих алгоритмів, серед яких є як класичні, так і передові. Бібліотека використовується приватними компаніями, дослідницькими групами та державними установами [21], [22].

OpenCV включає в себе в числі інших [21], [22]:

- методи обробки зображень;
- методи сегментації;
- методи МН;
- чисельні методи загального призначення;
- методи для калібрування камери і 3D реконструкції;
- методи для роботи з відео;
- методи для завантаження і збереження даних.

У таблиці 3.1 представлено результати порівняння комбінації C++ зі OpenCV та інших комбінацій мов програмування із бібліотеками МН та прикладних пакетів [21], [22].

Як видно з порівняння комбінація зі C++ та OpenCV добре себе зарекомендувала. Такий підхід може використовуватися для більшості задач РО пов'язаних із графічною інформацією [21], [22].

Таблиця 3.1 – Порівняння мов програмування

Критерії порівняння	Мови програмування		
	C++ та OpenCV	Python та TensorFlow	MATLAB
Спільнота	++	++	++
Продуктивність роботи застосунків	++	+	–
Підтримка сегментації зображень для РО	++	±	±
Паралельна обробка запитів та реалізація	++	++	±
Простота розроблення графічного інтерфейсу	+	±	±

3.2 Проектування розробки

Для вирішення завдання класифікації об'єктів потрібно створити набір модулів, який включає:

- модуль, відповідальний за навчання класифікатора. Модуль включає в себе клас Classifier Trainer;
- модуль, відповідальний за розпізнавання об'єкта з використанням навченого класифікатора. Модуль включає в себе класи Predictor і PredictionsCycle;
- допоміжні класи. Клас Segment призначений для роботи з даними про сегмент зображення. Класи, що відповідають за реалізації етапів: сегментації-Segment, Segment Filter, створення дескриптора – Detector, Descriptor, класифікації – Classifier.

Було вирішено реалізувати можливість вибору і комбінації різних класифікаторів, дескрипторів, детекторів і методів сегментації і можливість подальшого розширення. Цією метою служить структура Params і ієрархії класів Segmenter, Segment Filter, Descriptor, KeypointsDetector і Classifier.

UML-діаграма класів наведена на рис. 3.1.

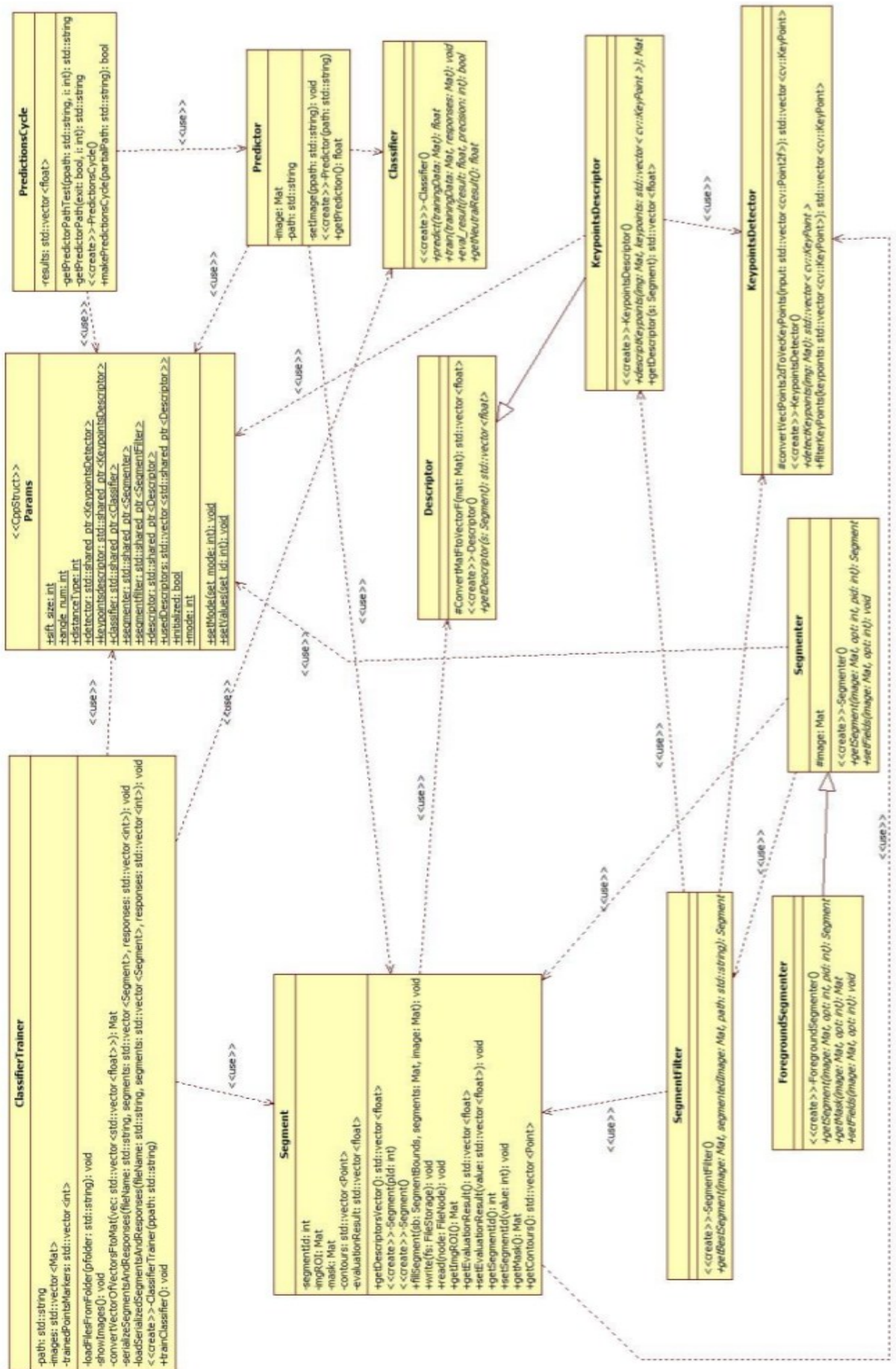


Рисунок 3.1 – UML-діаграма класів

3.2.1 Ієрархія класів та похідних від Segmenter

Клас Segmenter відповідає за сегментацію зображення. Він є абстрактним класом, необхідним для того, щоб надати інтерфейс для взаємодії з конкретними класами для можливості розширення в майбутньому. Таким чином використовується шаблон проектування «Стратегія».

Клас Segmenter має два похідних від нього класу:

- ForegroundSegmenter. Також є абстрактним класом для можливого розширення. Його нащадком є SegmenterGrabCut. Даний клас виробляє сегментацію методом GrabCut, при цьому розділяючи зображення на два сегменти - фон і об'єкт. Даний метод сегментації заснований на розрізі графа;
- SegmenterGraph. Даний клас також виробляє сегментацію, засновану на розрізі графа, але дана сегментація ділить зображення на безліч сегментів. Алгоритм даної сегментації був запропонований в роботі [20].

3.2.2 Ієрархія класів та похідних від KeypointsDetector

Клас KeypointsDetector є абстрактним класом, необхідним для того, щоб надати інтерфейс для взаємодії з конкретними класами для можливості розширення в майбутньому. Таким чином використовується шаблон проектування «Стратегія».

По суті клас KeypointsDetector є обгорткою над методами бібліотеки OpenCV, які не надають уніфікований інтерфейс.

Реалізовані наступні класи детекторів:

- детектор Харріса;
- детектор GoodFeaturesToTrack;
- детектор FAST;
- детектор ORB;

- детектор SIMT;
- детектор SURF.

3.2.3 Ієрархія класів та похідних від Descriptor

Клас Descriptor є абстрактним класом, необхідним для того, щоб надати інтерфейс для взаємодії з конкретними класами для можливості розширення в майбутньому. Таким чином використовується шаблон проектування «Стратегія».

KeypointsDescriptor є похідним від класу Descriptor абстрактним класом, що має те ж призначення, але як інтерфейс дескрипторів ключових точок.

Дескриптори, використовувані в даній роботі, за винятком дескрипторів ключових точок, були розроблені в рамках даної роботи.

Реалізовані наступні класи дескрипторів:

- дескриптор центру мас;
- дескриптор гістограми кольору;
- дескриптор гістограми кутів;
- дескриптор гістограми довжин;
- дескриптор заповнення простору.

Реалізовані наступні класи дескрипторів ключових точок: дескриптор SIFT; дескриптор SURF.

3.2.4 Ієрархія класів та похідних від Classifier

Клас Classifier є абстрактним класом, необхідним для того, щоб надати інтерфейс для взаємодії з конкретними класами для можливості розширення в майбутньому. Таким чином використовується шаблон проектування «Стратегія».

Клас Classifier, крім зазначеної вище функції, також має метод для оцінки результату класифікації.

Реалізовано наступні класи класифікаторів:

- класифікатор SVM (Support vector Machines);
- класифікатор Str Trees (Random Trees).

Класифікатор SVM є лінійним класифікатором, тобто здійснює пошук оптимальної розділяє гіперплощини.

Класифікатор SvRTrees є класифікатором, заснованим на індукції правил.

Важливою для даної роботи перевагою обох класифікаторів є відсутність схильності до перенавчання [19].

3.3 Вимоги до технічних засобів

Вимоги до складу і параметрів технічних засобів

Технічна характеристика:

- персональний комп'ютер (ПК) з операційною системою (ОС) Microsoft Windows 8.1 та старше;
- процесор – Intel Core i3-10100F CPU 4,3 GHz, ядер: 1, логічних процесорів: 2;
- мікрофон Realtek High Definition Audio.

Вимоги до інформаційної та програмної сумісності:

- ОС ПК – Windows: Windows XP і вище, Linux, OS X.

3.4 Висновки за розділом 3

Під час роботи над розділом було досліджено технічні засоби, які будуть використовуватися під час розробки ПЗ, що реалізує запропонований метод. Так у якості мови програмування було обрано C++ та бібліотеку OpenCV, через потужну базу вбудованих методів та бібліотек з комп'ютерного зору. Також достатню увагу було приділено проектуванню ПЗ та розробленню ієрархії класів.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У третьому розділі було досліджено та вирішено ряд питань з проектування архітектури та реалізації ПЗ для дослідження запропонованого методу та можливих модифікацій. Особливу увагу було присвячено вибору технічних засобів: мови програмування та формату кодування звукових файлів.

Для вирішення завдання класифікації об'єктів потрібно створити набір модулів, який включає:

- модуль, відповідальний за навчання класифікатора. Модуль включає в себе клас Classifier Trainer;
- модуль, відповідальний за розпізнавання об'єкта з використанням навченого класифікатора. Модуль включає в себе класи Predictor і PredictionsCycle;
- допоміжні класи. Клас Segment призначений для роботи з даними про сегмент зображення. Класи, що відповідають за реалізації етапів: сегментації-Segment, Segment Filter, створення дескриптора-Detector, Descriptor, класифікації – Classifier.

У розділі описано результати проектування та реалізації з допомогою C++ та OpenCV.

4.1 Клас ClassifierTrainer

Як вже було зазначено, клас ClassifierTrainer відповідає за навчання класифікатора.

ClassifierTrainer отримує на вхід набір зображень і набір міток, відповідних даним зображенням. Результатом роботи даного модуля є навчений Класифікатор у формі файлу формату XML.

Робота модуля ClassifierTrainer складається з наступних етапів:

- отримання набору зображень і набору міток;
- сегментація кожного зображення;
- створення дескриптора для кожного із зображень;
- навчання класифікатора на основі набору дескрипторів і набору відповідних міток.

Основними полями класу є поля, представлені нижче. `std:: vector < Mat> images` є полем, що зберігає набір зображень для навчання класифікатора

`std:: vector < int> trainedPointsMarkers` є полем, що зберігає набір міток, що визначають клас об'єкта на відповідному зображенні

Основними методами класу є методи, представлені нижче. `ClassifierTrainer (String ppath)` є конструктором, який приймає на вхід шлях до папки з даними для навчання.

`void loadFilesFromFolder (String pfolder)` є методом, що завантажує дані з зазначеної папки в поле `images` і, відповідно, мітки в поле `trainedPointsMarkers`.

`void QassifierTramer:: tramQassifierO` є методом, який, використовуючи інші методи класу і об'єкти класів `Segment` і класів, похідних від `Segmenter` і `Classifier`, навчають класифікатор.

4.2 Клас Predictor

Як вже було зазначено, клас `Predictor` відповідає за розпізнавання об'єкта на зображенні, використовуючи навчений Класифікатор.

`Predictor` отримує на вхід набір від одного до трьох зображень. Результатом роботи даного модуля є рішення, чи є об'єкт на зображеннях об'єктом шуканого класу.

Робота модуля `Predictor` складається з наступних етапів:

- отримання зображення;
- сегментація зображення;

- створення дескриптора для зображення;
- отримання рішення класифікатора.

Основними полем класу є `mat image`, що зберігає зображення для обробки.

Основними методами класу є методи, представлені нижче.

`Predictor (std:: string path)` є конструктором класу, який приймає на вхід шлях до зображення, яке слід завантажити і обробити.

`void setImage (std::string ppath)` є методом, що виробляє первинну обробку зображення (приводить його до фіксованого розміру).

`float getPredictions ()` - є методом, який, використовуючи об'єкти класів `Segment` і класів, похідних від `Segmenter` і `Classifier`, знаходить на зображенні об'єкт `i`, після розпізнавання його за допомогою класифікатора, видає результат.

4.3 Клас `PredictionsCycle`

Клас `PredictionsCycle` відповідає за цикл, який запитує додаткові зображення об'єкта в разі недостатньої точності результату.

Основними методами класу є методи, представлені нижче.

`bool makePredictionsCycle (std:: string partialPath="")` є методом, що реалізує цикл, в якому послідовно обробляються кілька зображень для більш точного результату класифікації.

`std::string getPredictorPathTest(std:: string path, int i)` є методом, призначеним для формування рядка, що містить шлях до каталогу в режимі тестування класифікації.

`std:: string getPredictorPath (bool& exit, int i)` є методом, призначеним для формування рядка, що містить шлях до каталогу в режимі класифікації.

4.4 Клас Segment

Клас Segment відповідає за роботу з даними про певний передбачуваний об'єкт на зображенні.

Його основною функцією є зберігання даних про певний передбачуваний об'єкт на зображенні і робота з ними.

Основними полями класу є поля, представлені нижче.

`int segmentId` є полем класу, що містить ідентифікатор об'єкта.

`Mat imgROI` є частиною зображення, що містить в собі передполаганий об'єкт і додатковий відступ з кожного боку(необхідний для пошуку контурів і деяких видів сегментації)

`Mat mask` є маскою, де в місцях присутності об'єкта знаходяться 1, а в інших – 0.

`std::vector<Point> contours` є масивом точок, що складають контур об'єкта. Контур апроксимований. По даному масиву будується маска.

`struct SegmentBounds` є внутрішньою структурою, що використовується для заповнення сегмента. Дана структура містить крайні точки передполаганого об'єкта і ідентифікатор.

`std::vector<float> evaluationResult` є масивом, що містить оцінки за різними критеріями ймовірності того, що даний об'єкт є основним об'єктом зображення.

Основними методами класу є методи, представлені нижче.

`void fillSegment (SegmentBounds sb, Mat & segments, Mat& image)` є методом, що створює і заповнює сегмент.

4.5 Клас SegmentFilter

Клас SegmentFilter відповідає за фільтрацію сегментів для тих класів сегментації, які сегментують зображення більш, ніж на дві частини. Даний

клас є абстрактним і служить інтерфейсом для можливості розширення в майбутньому.

`SegmentFilterImpl` є єдиним похідним від класу `segmentfilter` класом.

Основними методами класу є методи, представлені нижче.

`Segment getBestSegment (Mat& image, Mat& segmentedImage, std::string path)` є методом, що приймає вихідне зображення і матрицю з мітками і видає оптимальний з відповідних сегментів.

`Mat getMeanShiftFiltering ()` є методом, який, використовуючи поле `image`, проводить фільтрацію методом `MeanShift` і видає пастеризоване зображення.

`Mat getCannyContours (Mat& imageSegment)` є методом, приймаючим на вхід матрицю з мітками і видає на виході матрицю з контурами, розрахованими за алгоритмом Кенні

`Mat uniteSegmentsBasedOnCannyBorders (Mat &contoursCanny, Mat & labelsGraph)` є методом, який об'єднує мітки на основі відсутності між ними достатньої кількості контурів.

`std::vector<Segment> calculateSegments (Mat& segments)` є методом, який отримує на вхід матрицю з метрами і створює об'єкт класу `Segment` для кожної мітки

`std::vector<float> evaluatePossibleObject(Mat& imgSegm, int label, std::vector<Mat> kpAreasMasks)` є методом, що оцінює сегмент як потенційний об'єкт

`bool findBestSegment (std::vector<Segment>& segments, Segment& resSegment)` є методом, що визначає оптимальний з відповідних сегментів.

4.6 Структура Params

Структура `Params` призначена для зберігання параметрів роботи модулів `ClassifierTrainer` і `Predictor`. Дана структура зберігає режим роботи програми (навчання, Класифікація, тестування класифікації) і посилання на

об'єкти класів, похідних від KeypointsDetector, KeypointsDescriptor, Descriptor і Classifier.

4.7 Деталі реалізації

В цьому підрозділі будуть описані методи, реалізація яких потребувала роздумів і була пов'язана з тими чи іншими складнощами (проблемами).

4.7.1 Серіалізація класу Segment

Необхідність реалізації серіалізації стала очевидна під час модульного тестування модуля навчання класифікатора в процесі розробки у зв'язку з тривалістю сегментації. Було вирішено серіалізувати отримані в результаті сегменти і таким чином мати можливість істотно більш швидко перенавчати Класифікатор. Для серіалізації класу були використані класи бібліотеки OpenCV FileStorage і FileNode. Серіалізація здійснюється за заданою адресою у форматі XML.

4.7.2 Метод fillSegment класу Segment

Метод fillSegment створює і заповнює сегмент, використовуючи крайні точки передполагаемого об'єкта, зображення і сегментоване зображення. Метод знаходить цікавить сегмент на вихідному зображенні, первісну маску, виробляє на даній масці морфологічну операцію ерозії для видалення шуму, знаходить максимальний контур (так як після ерозії могли з'явитися дрібні окремі контури), апроксимує контур, створює результуючу маску по контуру.

4.7.3 Метод `getSegment` класу `SegmenterGraph`

Сегментація на графах добре розділяє зображення на безліч областей (на відміну від сегментації GrabCut, що ділить на дві області), що дуже корисно, якщо об'єкт сфотографований з відносно далекої відстані. Але після цього потрібно будь-яким чином об'єднати ті чи інші сегменти і відфільтрувати їх для отримання того сегмента, який представляє об'єкт. Сегмент отриманий в результаті даних дій, в більшості випадків досить точно відображає об'єкт на зображенні, але маска, отримана після даних маніпуляцій, має відносно рвані краї. Тому після знаходження необхідного сегмента маска коригується за допомогою сегментації Огабсі (так як сегмент, що містить об'єкт, вже виділений Огабсі дає хороші результати).

4.7.4 Метод `getCannyContours` класу `SegmentFilterImpl`

В даному методі довелося розраховувати контури Кенні по двох зображеннях: по початковому зображенню і по зображенню після пастеризації Меалбий, – і потім об'єднувати їх. Це було пов'язано з необхідністю мати більш зв'язні контури.

4.7.5 Метод `uniteSegmentsBasedOnCannyBorders` класу `SegmentFilterImpl`

Даний метод об'єднує сегменти, які не мають між собою достатньо кордонів.

Спочатку створюється матриця контурів сегментів на підставі матриці сегментів.

Після цього для кожної точки контуру сегментів в певній околиці шукається контур Кенні. Якщо такого контуру не знаходиться, точка

сегмента наноситься на матрицю контурів, потенційно підлягають видаленню.

Для кордону між кожними двома сегментами, що межують, створюється структура `std::vector<CountForBorders> bordersCount`

Після цього, якщо більше певного відсотка точок контуру між сегментами потенційно підлягають видаленню, і довжина межі більш певного значення, мітки сегментів заносяться в структуру `AdjacentSegments` і додаються в масив `AdjacentSegments`.

Даний масив подається в функцію `Mat mergeSegments(std::vector<AdjacentSegments> adSegm, Mat& initialImage_original)`, яка повертає матрицю сегментів, де сегменти, зазначені в одній структурі `AdjacentSegments` об'єднані. Функція повертає дану матрицю.

4.7.6 Метод `mergeSegments` класу `SegmentFilterImpl`

Ця функція отримує на вхід масив структур, що містять мітки сегментів, які необхідно об'єднати, і матрицю сегментів.

Об'єднання сегментів відбувається за допомогою теорії графів.

Кожним об'єктом структури `AdjacentSegments` визначається дуга графа і дві суміжні вершини.

Необхідно знайти всі зв'язкові графи.

Для кожної мітки матриці сегментів створюється об'єкт структури `Vertex`.

Структури `Vertex` зберігаються в `std::map<int, Vertex> vertices`. Структура даних словник була обрана через необхідність шукати значення по ключу.

Проводиться перебір всіх об'єктом структури `AdjacentSegments` при якому мітки сегмента додаються в поле `adjacent` суміжного сегмента.

Після цього відбувається обхід в глибину вершин. При цьому заповнюється `static std::map<int, std::vector<int>> joined`. В даному словнику ключ-значення першої вершини, а `std::vector<int>` - масив вершин, в які є шлях з першої вершини.

Таким чином кожен елемент словника являє собою зв'язний граф.

Після цього створюється матриця, на якій мітки, відповідні вершинам зв'язкового графа, змінюються на мітку, відповідну першій вершині даного графа.

З функції повертається дана матриця.

4.7.7 Метод `computeNumberOfDotsOnBorder` класу `SegmentFilterImpl`

Даний метод заповнює масив структур `CountForBorders`. Дана структура описує кордон між двома сегментами. Структура містить мітки межують об'єктів, кількість точок кордону, кількість точок кордону, які можливо підлягають видаленню.

Для цього спочатку заповнюється масив структур `DotOnBorder`. `DotOnBorder`-структура, що представляє точку на кордоні. Структура містить координати точки `i, j` та мітки межують сегментів.

Масив структур `DotOnBorder` заповнюється на підставі матриці кордонів `contours` і матриці сегментів `labelsGraph` (необхідна, так як з неї береться мітка другого сегмента).

Після цього при переборі масиву структур `DotOnBorder` вираховується поле `CountForBorders count`, що містить кількість точок на кордоні.

Аналогічно, на підставі матриць `removed` і `labelsGraph`, заповнюється поле `countRemoved` – містить кількість точок кордону, які можливо підлягають видаленню.

4.7.8 Метод `evaluatePossibleObject` класу `SegmentFilterImpl`

Метод оцінює характеристики сегмента:

- 1) віддаленість від центру зображення;
- 2) розмір відносно розміру зображення;
- 3) максимальна частина площі будь-якого з опуклого багатокутника, що включає згруповані ключові точки (у %), що лежить всередині даного сегмента.

Метод приймає на вхід матрицю сегментів, мітку і масив матриць опуклих багатокутників, побудованих за згрупованим методом `detpartition` ключовим точкам.

Для отримання маски певного сегмента на загальній матриці сегментів використовується функція `OpenCV inRange`.

Для визначення відстані від сегмента до точки існують функція `OpenCV pointPolygonTest`.

На виході метод оцінки видає масив з трьох значень, відповідних характеристикам.

4.7.9 Метод `buildKeyPointsAreasMasks` класу `SegmentFilterImpl`

Даний метод будує матриці з масками опуклих багатокутників, що включають згруповані ключові точки.

Багатокутники будуються тільки по групах, що включають більше 5 точок.

Багатокутники будуються за допомогою функції `OpenCV convexHull`.

На вихід метод видає масив матриць з масками сформованих багатокутників.

4.7.10 Метод `removeSegmentsOnBorders` класу `SegmentFilterImpl`

Метод присвоює всім сегментам, що стикаються з кордоном зображення, мітку -1.

Спочатку знаходяться всі мітки, що визначають сегменти, що стикаються з кордоном зображення. Потім відбувається обхід копії матриці сегментів, де відповідним даним міткам сегментам присвоюється мітка -1.

Метод повертає відкориговану копію матриці сегментів.

4.7.11 Метод `findBestSegment` класу `SegmentFilterImpl`

Метод на основі поля `evaluationResult` вибирає сегмент, з найбільшою ймовірністю характеризує шуканий об'єкт.

Для кожної характеристики сегмента задаються вага і два пороги: високий і мінімальний.

Спочатку проводиться спроба знайти сегменти, за всіма характеристиками проходять по високому порогу.

Якщо такі не знайдені, проводиться спроба знайти сегменти, за всіма характеристиками проходять по мінімальному порогу.

Якщо такі не знайдені, проводиться спроба знайти сегменти, за розміром і віддаленості від центру зображення проходять по мінімальному порогу.

Якщо такі не знайдені, генерується виняток з повідомленням про помилку «segment not found».

Далі зі знайдених сегментів вибирається найкращий відповідно їх характеристикам і вагам характеристик.

Знайдений сегмент повертається методом.

4.7.12 Метод `getKeypointsPartition` класу `SegmentFilterImpl`

Метод ділить ключові точки на групи відповідно до функції еквівалентності, визначеної в класі `Equivalence`

Для цього створюється масив пар `KeyPoint`, `std::vector<float>`, куди заносяться пари, що складаються з ключової точки і відповідного їй дескриптора

Після цього на основі даного масиву і методу класу `Equivalence` функція `OpenCV partition` групує ключові точки, привласнюючи їм відповідні точки.

Після цього в масив масивів `Point` заносяться групи точок.

Метод повертає даний масив.

4.7.13 Внутрішній клас `Equivalence` класу `SegmentFilterImpl`

Даний клас – це внутрішній допоміжний клас, необхідний для створення предикату функції `OpenCV partition`.

У класі є поле `Mat& imageSegment`, що представляють собою зображення після пастеризації, і метод `bool operator()(const std::pair<KeyPoint, std::vector>& a, const std::pair<KeyPoint, vector>& b)`, що представляє собою функцію еквівалентності.

Спочатку метод визначає відстань між елементами `a` і `b`.

Якщо відстань більше певного порогу, метод повертає `false`.

Після цього вважається метрика відмінності між дескрипторами. Так як для різних дескрипторів підходять різні метрики, використовувана метрика визначається по полю структури `Params distType`.

Далі вираховується різниця кольорів в ключових точках.

Після цього розраховується результуюча різниця як сума відстаней, відстань між дескрипторами і різниці в кольорі, помножених на вагові коефіцієнти.

Після цього результуюча різниця порівнюється з порогом. Якщо різниця менше порога, метод повертає true, інакше – false

4.8 Висновки за розділом 4

В четвертому розділі була визначена концептуальна модель системи, реалізовані попередньо спроектовані класи системи та основні алгоритми. Було розроблено 23 класи.

Також у розділі були визначені межі застосовності методу, вимоги до структури використовуваних каталогів. Були проаналізовані тонкощі та додаткові механізми, що використовувалися при розробці.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Метою роботи є створення методу і програмного комплексу для класифікації певного типу об'єктів і розпізнавання даних об'єктів по зображеннях з декількох точок.

Проектована система призначена для застосування роботами при зборі сміття.

Зображення з декількох точок передбачається використовувати для підвищення надійності класифікації.

Розділ буде присвячено опису стратегії тестування, збору результатів еспериментального дослідження та аналізу цих результатів.

По завершенню розділу має бути встановлена якість роботи запропонованих методів та розробленого ПЗ.

5.1 Опис стратегії тестування

На основі поставленого завдання був сформований набір тестів стосовно проектованої системи можна виділити два завдання тестування.

По-перше, слід здійснити перевірку функціоналу ПЗ. Потрібно провести перевірку функціоналу Програми з точки зору користувача (кінцеве тестування з боку користувача).

Дана перевірка проводиться на основі вимог до проектованої системи. Заплановані наступні тест-кейси.

1. Позитивний:

- 1.1) перевірка режиму класифікації;
- 1.2) перевірка режиму тестової класифікації;
- 1.3) перевірка режиму навчання класифікатора.

2. Негативний:

2.1) перевірка режиму класифікації. Неправильно введений шлях;

2.2) перевірка режиму тестової класифікації. Неправильно передано параметр командного рядка-шлях до каталогу;

2.3) перевірка режиму класифікації. Відсутній файл необхідного класифікатора.

По-друге, слід оцінити якість методу.

Основою перевірки є тестова вибірка, в якій визначено відповідність між зображеннями об'єктів та їх класами.

Для оцінки якості алгоритму використовуються такі метрики, як точність (accuracy), достовірність (precision), повнота (recall).

Можливі комбінації результатів класифікації об'єктів та їх істинних значень наведені в таблиці 5.1. Негативними зображеннями є зображення, на яких відсутній шуканий об'єкт, позитивними – на яких присутній.

Таблиця 5.1 – Таблиця сполученості бінарного класифікатора

		Справжнє значення	
		Позитивне	Негативне
Результат класифікатора	Позитивний	True Positive	False Positive
	Негативний	False Negative	True Negative

Точність (accuracy) – метрика, яка визначає частку об'єктів, за якими було прийнято правильне рішення:

$$Accuracy = \frac{P}{N}, \quad (5.1)$$

де P – кількість документів, за якими класифікатор прийняв правильне рішення;

N – розмір тестової вибірки.

Достовірність (precision) – метрика, яка визначає яку кількість об'єктів, передбачених як позитивні, насправді ними є:

$$Precision = \frac{TP}{TP + FP}, \quad (5.2)$$

де TP – кількість позитивних зображень, за якими видано вірне рішення класифікації;

FP – кількість негативних зображень, за якими видано невірне рішення класифікації.

Повнота (recall) – метрика, яка визначає частку реальних позитивних об'єктів, за якими було прийнято правильне рішення:

$$Recall = \frac{TP}{TP + FN}, \quad (5.3)$$

де TP – кількість позитивних зображень, за якими видано вірне рішення класифікації;

FN – кількість позитивних зображень, за якими видано невірне рішення класифікації.

У табл. 5.2 наведено приклади зображень, що використовувалися для навчання та тестування класифікатора. Загалом вибірка була розподілена на навчальну та тестову підвибірки. В самих підвибірках було наведено розподіл за позитивним та негативним класом.

5.2 Результати тестування

Модульне та інтеграційне тестування проводилися в процесі написання коду.

Таблиця 5.2 – Приклади зображень, використаних у навчальній та тестовій вибірках

	Навчальна вибірка		Тестова вибірка	
Позитивні				
				
Негативні				
				

Всі тест-кейси, визначені для функціонального тестування, були проведені. Всі тест-кейси були пройдені успішно.

Вимога низького споживання пам'яті (менше 2 Mb) також виконується.

Таблиця з результатами тест-кейсів наведена в табл. 5.3.

Оцінка якості методу проводилася на тестовій вибірці, що містить 17 комплектів позитивних зображень і 17 комплектів негативних зображень.

Навчальна вибірка при навчанні класифікатора складала 59 негативних і 329 позитивних зображень.

Була проведена оцінка якості алгоритму з наступними параметрами:

1) дескриптори: дескриптор гістограми кутів, дескриптор гістограми довжин, дескриптор кольору, дескриптор заповнення простору;

2) метод сегментації-GrabCut;

3) класифікатор-Random Trees.

Правильно було класифіковано 11 позитивних зображень та 12 негативних зображень.

В результаті оцінки були отримані наступні метрики:

- 1) точність склала 67,6%;
- 2) достовірність склала 68,8%;
- 3) повнота склала 64,7%.

Таблиця 5.3 – Таблиця з результатами тест-кейсів

Опис	Передумова	Кроки	Очікувані результати	Результат
1	2	3	4	5
Позитивні тести				
Перевірка режиму класифікації	1. У каталозі програми знаходиться файл з навченим класифікатором необхідного типу	1. Введені відповідні параметри командного рядка 2. При кожному запиті застосунку введений коректний шлях до зображення	На консоль виводиться результат класифікації (число 0 або 1)	Пройдено
Перевірка режиму тестової класифікації	1. У каталозі програми знаходиться файл з навченим класифікатором необхідного типу 2. У каталозі, адреса якого передається як аргумент, знаходиться вміст, що відповідає вимогам програми	1. Введено відповідні параметри командного рядка	У текстовому файлі з певним ім'ям, розташованому в каталозі програми, занесені результати класифікації (числа 0 або 1)	Пройдено

Продовження таблиці 5.3

1	2	3	4	5
Перевірка режиму навчання класифікатора	1. У каталозі, адреса якого передається як аргумент, знаходиться вміст, що відповідає вимогам програми	1. Введені відповідні параметри командного рядка	Створено або перезаписано класифікатор	Пройдено
Негативні тести				
Перевірка режиму тестової класифікації . Неправильно передано параметр командного рядка-шлях до каталогу	1. У каталозі програми знаходиться файл з навченим класифікатором необхідного типу 2. У каталозі, адреса якого передається як аргумент, знаходиться вміст, що відповідає вимогам застосунку	1. Введені параметри командного рядка. Каталог по вказаному шляху відсутній	На консоль виводиться повідомлення про помилку. Застосунок завершує роботу	Пройдено
Перевірка режиму класифікації . Відсутній файл необхідного класифікатора	1. У каталозі програми відсутній файл з навченим класифікатором необхідного типу	1. Введені відповідні параметри командного рядка 2. При кожному запиті програми введений коректний шлях до зображення	На консоль виводиться повідомлення про помилку. Додаток завершує роботу	Пройдено

5.3 Висновки за результатами тестування

У п'ятому розділі було проведено функціональне тестування проектованої системи та оцінка якості алгоритму. Функціональне тестування було пройдено успішно. Оцінка якості алгоритму показала, що надійність

алгоритму становить 67,6%. Отже, вимога по надійності не виконується. Тим часом, враховуючи невеликий розмір навчальної вибірки, можна розраховувати на поліпшення надійності розпізнавання при її збільшенні.

Поставлені завдання в цілому виконані, за винятком високої надійності. Можливими шляхами досягнення необхідної надійності є використання МН сегментації зображень або згорткових нейронних мереж.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи магістра, було розглянуто основні методи автоматичної класифікації об'єктів за зображенням:

- SVM;
- Random forest;
- k-ближчих сусідів.

Було розглянуто загальну структуру систем автоматичної класифікації за зображеннями. Структура складається з трьох етапів:

- модуль сегментація;
- модуль розпізнавання.

Під час роботи були визначені межі застосовності методів, вимоги до структури використовуваних каталогів. Були реалізовані спроектовані в класи.

У п'ятому розділі було проведено функціональне тестування проектованої системи та оцінка якості методів. Функціональне тестування було пройдено успішно. Оцінка якості алгоритму показала, що надійність алгоритму становить 67,6%. Отже, вимога по надійності не виконується. Тим часом, враховуючи невеликий розмір навчальної вибірки, можна розраховувати на поліпшення надійності розпізнавання при її збільшенні.

Під час розробки програми та аналізу розробленого програмного комплексу з'явилися ідеї з удосконалення, такі як: використання штучних нейронних мереж з метою видалення шуму та підвищення точності.

Нові можливості зможуть розширити коло вирішення практичних задач, адже зменшення шуму при скануванні в реальному часі значно полегшить роботу та підвищить кількість точно класифікованих об'єктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Заяць В.М. Методи розпізнавання образів [Текст] : навч. посіб. для студ. спец. 7.0804.03 "Програмне забезпечення автоматизованих систем" базового напрямку "Комп'ютерні науки" / В. М. Заяць, Р. М. Камінський ; Національний ун-т "Львівська політехніка". - Л. : Видавництво Національного університету "Львівська політехніка", 2004. - 173 с.
2. Гавриленко В. В. Теорія розпізнавання образів [Текст] : навч. посіб. для студентів Нац. трансп. ун-ту, що навчаються за напрямом "6.050101 Комп'ютерні науки" / Гавриленко В. В., Іванченко Г. Ф., Шевченко Г. С. ; Нац. трансп. ун-т, Каф. інформ. систем і технологій. - Київ : НТУ, 2015. - 75 с.
3. Математичні та програмні засоби для прийняття рішень, розпізнавання образів й інтелектуального діагностування [Текст] : монографія / Субботін С. О. [та ін.] ; під заг. ред. д-ра техн. наук, проф. С. О. Субботіна ; Нац. ун-т "Запоріж. політехніка". - Запоріжжя : Нац. ун-т "Запоріж. політехніка", 2020. - 270 с.
4. Побігайло В. А. Побудова системи розпізнавання образів у пристроях обмеження струмів короткого замикання [Електронний ресурс] / В. А. Побігайло, В. П. Калінчик // Енергетика: економіка, технології, екологія. - 2020. - № 2. - С. 103-119.
5. Ткаченко Т. М. Розпізнавання акустичних образів риб при моніторингу прісноводних екосистем [Електронний ресурс] / Т. М. Ткаченко, Ю. Г. Пількевич, Г. М. Розорінов // Екологічна безпека та природокористування. - 2021. - Вип. 1. - С. 20-34.
6. Шамрін Р. В. Методологічний підхід до прогнозування фінансових часових рядів на основі розпізнавання образів у структурі цінових кривих [Електронний ресурс] / Р. В. Шамрін // Науковий вісник Ужгородського національного університету. Серія : Міжнародні економічні відносини та світове господарство. - 2015. - Вип. 5. - С. 200-204.

7. Захожай О. І. Прийняття рішень на основі пошуку груп ідентичних класифікацій в багатопараметричних комбінованих системах розпізнавання образів [Електронний ресурс] / О. І. Захожай, В. О. Лифар, О. І. Батурін // Вісник Східноукраїнського національного університету імені Володимира Даля. - 2019. - № 2. - С. 36-42.

8. Басюк Т. Аналіз та класифікація основних методів розпізнавання образів на площині проекції [Електронний ресурс] / Т. Басюк, Я. Пушко // Вісник Національного університету "Львівська політехніка". Комп'ютерні науки та інформаційні технології. - 2015. - № 826. - С. 291-298.

9. Щуліпенко Л. І. Патогенетичні механізми формування дизрегуляторних станів у операторів транспорту: моделювання, класифікація, формалізоване розпізнавання [Текст] : автореф. дис. ... канд. мед. наук : 14.03.04 / Щуліпенко Леся Ігорівна ; Вищ. держ. навч. закл. України "Буковин. держ. мед. ун-т". - Чернівці, 2016. - 20, [1] с.

10. Мартинюк Т. Б. Кореляція, фільтрація та сегментація зображень [Текст] : лабораторний практикум для студ. спец. "Лазерна та оптоелектронна техніка" / Т. Б. Мартинюк [и др.] ; Вінницький національний технічний ун-т. - Вінниця : ВНТУ, 2006. - 80 с.

11. Полякова М. В. Сегментація зображень стохастичних текстур амплітудно-детекторним методом у просторі вейвлет-перетворення [Електронний ресурс] / М. В. Полякова, Н. П. Волкова, О. В. Іванова // Автоматика. Автоматизація. Електротехнічні комплекси та системи. - 2008. - № 2. - С. 81-88.

12. Журба А. О. Фрактальна сегментація зображень для дослідження природних об'єктів та поверхонь [Електронний ресурс] / А. О. Журба, Д. І. Журба // Системні технології. - 2018. - Вип. 3. - С. 131-137.

13. Голуб С. В. Сегментація зображень при індуктивному моделюванні проблемних ділянок мамограм [Електронний ресурс] / С. В. Голуб, С. Г. Палаш // Вісник Інженерної академії України. - 2014. - Вип. 2. - С. 53-57.

14. Boltenkov V. O. Information Technology for Automated Assessment of the Artillery Barrels Wear Based on Svm Classifier [Електронний ресурс] / V. O. Boltenkov, M. V. Maksymov, Y. V. Dobrynin // Applied aspects of information technology. - 2020. - Vol. 3, no 3. - С. 117–132.
15. Kuleshova V. V. Mathematical concepts overview of cluster analysis, SVM and neural networks in mass service networks [Електронний ресурс] / V. V. Kuleshova, A. G. Lozhkovskiy // Наукові праці ОНАЗ ім. О. С. Попова. - 2017. - № 2. - С. 81-88.
16. Шеремет О. І. Метод опорних векторів (SVM) [Електронний ресурс] / О. І. Шеремет, О. В. Садовой // Математичне моделювання. - 2013. - № 1. - С. 13-17.
17. Gromey D. Construction of a parametric model of competitive access in relational databases by using a random forest method [Електронний ресурс] / D. Gromey, E. Lebedenko, D. Nikolaev, T. Rozhkova // Восточно-Европейский журнал передовых технологий. - 2019. - № 3(2). - С. 15-24.
18. Самигулина Г. А. Построение оптимальной иммунносетевой модели для компьютерного молекулярного дизайна сульфаниламидов на основе алгоритма random forest [Електронний ресурс] / Г. А. Самигулина, З. И. Самигулина // Вісник Національного технічного університету "ХПІ". Серія : Інформатика та моделювання. - 2016. - № 21. - С. 102-108.
19. Stroustrup B. Programming: Principles and Practice Using C++ (2nd Edition) [Text] / B. Stroustrup. – С.: Addison-Wesley Professional, 2015. – 1312 p.
20. Lippman S.B., Lajoie J., Moo B.E. C++ Primer (5th Edition) [Text] / S.B. Lippman, J. Lajoie, B.E. Moo. – Cambridge : Addison-Wesley Professional, 2012. – 976 p.
21. Ших Н. В. Аналіз роботи фронтальних класифікаторів бібліотеки OpenCV / Н. В. Ших, І. О. Шаклеїна // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2019. – № 35. – С. 107-111.

22. Системи адаптації роботів і технологія OpenCV [Текст] : навч. посіб. / О. М. Цимбал, А. І. Бронніков ; Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2019. – 144 с.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.2 Призначення та область застосування

Програма розпізнавання та класифікації об'єктів на графічних зображеннях. Розроблена програма реалізована з використанням C++ та бібліотеки OpenCV.

A.2 Підстави для розробки

Підставою для розробки є завдання на магістерську роботу, яке було визначене за темою «Дослідження та програмна реалізація методів розпізнавання об'єктів для автоматизації сортування сміття» і затверджене наказом № 435 від 05 листопада 2021 р. за Національним університетом «Запорізька політехніка».

A.3 Призначення розробки

Програма призначена для класифікації об'єктів за графічними зображенням для сортування сміття.

A.4 Вимоги до програми

Система повинна відповідати наступним вимогам:

а) застосунок повинен забезпечувати можливість навчання класифікатора:

2) система повинна бути надана можливість ввести шлях до каталогу, що містить навчальну вибірку:

- запит на введення шляху;

- шлях для доступу до даних для тренування вводиться

як аргумент командного рядка при запуску програми;

- якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;
- б) застосунок повинен забезпечувати можливість бінарної класифікації об'єкта на зображенні:
 - б) додавання шляхів до зображень для класифікації повинно відбуватися через консоль при запиті програми:
 - якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;
 - 7) повинна бути забезпечена можливість введення до чотирьох зображень;
 - 8) результат класифікації виводиться на консоль;
 - 9) повинна бути забезпечена можливість перервати цикл класифікації:
 - після переривання циклу класифікації повинен бути виданий результат;
 - 10) якщо відсутній необхідний класифікатор, на консоль виводиться повідомлення про помилку;
- в) застосунок повинен забезпечувати можливість бінарної класифікації об'єкта на зображенні в режимі тестування;
 - 2) повинна бути надана можливість ввести шлях до каталогу, що містить каталоги з тестовими даними:
 - потреба введення шляху шлях;
 - шлях вводиться як аргумент командного рядка при запуску програми;
 - якщо введено некоректний шлях, на консоль виводиться повідомлення про помилку;
 - якщо відсутній необхідний класифікатор, на консоль виводиться повідомлення про помилку;
- г) повинна бути забезпечена можливість вибору параметрів роботи програми;

- 5) вибір параметрів роботи програми проводиться за допомогою аргументів командного рядка при запуску програми;
- 6) повинна бути забезпечена можливість вибору методу сегментації;
- 7) повинна бути забезпечена можливість вибору дескрипторів;
- 8) повинна бути забезпечена можливість вибору класифікатора;
- д) надійність розпізнавання повинна становити не менше 80%;
- ж) система повинна займати не більше 2 МБ в пам'яті.

A.4.1 Вимоги до надійності

Для сталого функціонування програмного забезпечення необхідно:

- наявність безперебійних блоків живлення ПК.

Програма повинна видавати відмову доступу до комп'ютера при спробі несанкціонованого доступу.

A.4.2 Вимоги до технічних та програмних засобів

Вимоги до складу і параметрів технічних засобів

Технічна характеристика:

- ПК з ОС Microsoft Windows 8.1 та старше;
- процесор – Intel Core i3-10100F CPU 4,3 GHz, ядер: 1, логічних процесорів: 2;
- мікрофон Realtek High Definition Audio.

Вимоги до інформаційної та програмної сумісності:

- ОС ПК – Windows: Windows XP і вище, Linux, OS X.

A.5 Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

- технічне завдання;
- опис проектування;
- опис тестування;
- опис програми.

A.6 Стадії і етапи розробки

A.6.1 Стадії розробки

Розробка повинна бути проведена в три стадії:

- розробка технічного завдання;
- робоче проектування;
- впровадження.

A.6.2 Етапи розробки

На стадії розробки технічного завдання повинен бути виконаний етап розробки, погодження та затвердження справжнього технічного завдання [18], [20].

На стадії робочого проектування повинні бути виконані перераховані нижче етапи робіт [18], [20]:

- розробка програми;
- розробка програмної документації;
- випробування програми.

На стадії впровадження повинен бути виконаний етап розробки підготовка і передача програми [18], [20].

A.6.3 Зміст робіт по етапах

На етапі розробки технічного завдання повинні бути виконані перераховані нижче роботи [18], [20]:

- постановка задачі;
- визначення і уточнення вимог до технічних засобів;
- визначення вимог до програми;
- визначення стадій, етапів і термінів розробки програми та документації на неї;
- узгодження і затвердження технічного завдання.

На етапі розробки програми повинна бути виконана робота з програмування (кодування) і налагодженні програми [18], [20].

На етапі розробки програмної документації повинна бути виконана розробка програмних документів відповідно до вимог до складу документації [18], [20].

На етапі випробувань програми повинні бути виконані перераховані нижче види робіт [18], [20]:

- розробка, узгодження та затвердження та методики випробувань;
- проведення приймально-здавальних випробувань;
- коректування програми і програмної документації за результатами випробувань.

ДОДАТОК Б
Опис програми

Б.1 Загальні свідчення

Програма розпізнавання та класифікації об'єктів на графічних зображеннях. Розроблена програма реалізована з використанням C++ та бібліотеки OpenCV.

Б.2 Функціональне призначення

Програма призначена для класифікації об'єктів за графічними зображенням для сортування сміття.

Б.3 Опис логічної структури

Для вирішення завдання класифікації об'єктів потрібно створити набір модулів, який включає:

- модуль, відповідальний за навчання класифікатора. Модуль включає в себе клас Classifier Trainer;
- модуль, відповідальний за розпізнавання об'єкта з використанням навченого класифікатора. Модуль включає в себе класи Predictor і PredictionsCycle;
- допоміжні класи. Клас Segment призначений для роботи з даними про сегмент зображення. Класи, що відповідають за реалізації етапів: сегментації-Segment, Segment Filter, створення дескриптора – Detector, Descriptor, класифікації – Classifier.

Було вирішено реалізувати можливість вибору і комбінації різних класифікаторів, дескрипторів, детекторів і методів сегментації і можливість подальшого розширення. Цією метою служить структура Params і ієрархії класів Segmenter, Segment Filter, Descriptor, KeypointsDetector і Classifier.

UML-діаграма класів наведена на рис. Б.1.

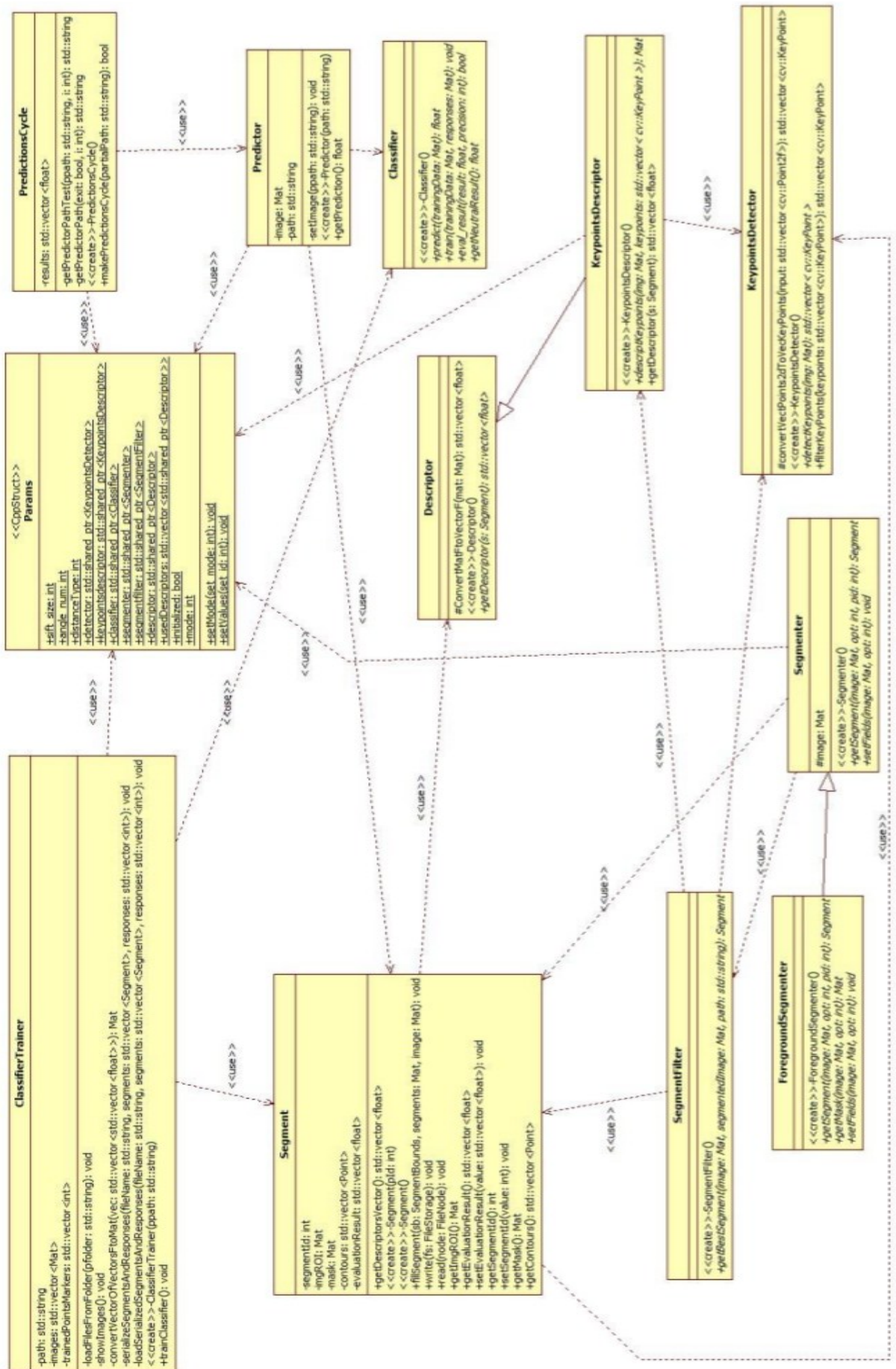


Рисунок Б.1 – UML-діаграма класів

Б.4 Використані технічні засоби

Вимоги до складу і параметрів технічних засобів

Технічна характеристика:

- ПК з ОС Microsoft Windows 8.1 та старше;
- процесор – Intel Core i3-10100F CPU 4,3 GHz, ядер: 1, логічних процесорів: 2;
- мікрофон Realtek High Definition Audio.

Вимоги до інформаційної та програмної сумісності:

- ОС ПК – Windows: Windows XP і вище, Linux, OS X.

ДОДАТОК В
Текст програми

```

Classifier.h
using namespace cv;
class Classifier {
public:
Classifier();
virtual float predict(const Mat& trainingData)=0;
virtual void train(const Mat& trainingData, const Mat& responses)=0;
virtual bool eval_result(float result, int &precision)=0;
virtual float getNeutralResult()=0;
};
class ClassifierSVM: public Classifier
{
public:
ClassifierSVM();
virtual float predict(const Mat& trainingData);
virtual void train(const Mat& trainingData, const Mat& responses);
virtual bool eval_result(float result, int &precision);
virtual float getNeutralResult();
class ClassifierCvRTrees: public Classifier
{
public:
ClassifierCvRTrees();
virtual float predict(const Mat& trainingData);
virtual void train(const Mat& trainingData, const Mat& responses);
virtual bool eval_result(float result, int &precision);
virtual float getNeutralResult();
};
#endif // CLASSIFIER H
Classifier.cpp
Classifier::Classifier()
{
}
float ClassifierSVM::predict(const Mat& trainingData){
CvSVM SVM;
SVM.load("SVMClassifier.xml"); if(SVM.get_support_vector_count()==0){ throw
std::exception("no classifier");
}
float result=SVM.predict(trainingData, true); return result;
}
void ClassifierSVM::train(const Mat& trainingData, const Mat& responses){ // Set
up SVM's parameters CvSVMParams params; params.svm_type =
CvSVM::C_SVC; params.kernel_type = CvSVM::LINEAR;
params.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);

```

```

SVM Mira/«(trainingData, responses, Mat(), Mat(), params);
SVM.save("SVMClassifier.xml");
#ifdef DEBUG
std::cout<<"classifier trained"<<std::endl;
#endif
}
bool ClassifierSVM::eva/_resM/t(float result, int& precision^
float resTheshold=1.5; float resMinTheshold=0.3; if(abs(result)>resTheshold){
precision=2; if(result>0){
return false;
}else{ return true;
}
}else if(abs(result)>resTheshold>resMinTheshold){ precision=1; if(result>0){
return false;
}else{ return true;
}
}else{
precision=0; return false;
}
}
float ClassifierSVM::getNeutralResult(){ return 0;
}
float QassifierCvRTrees::pred/'cf(const Mat& trainingData){ CvRTrees CvRTrees;
CvRTrees./oadC'CvRTreesQassifier.xmlT); int x=CvRTrees.get_tree_count();
if(CvRTrees.get_tree_count()==0){ throw std::exception("no classifier");
}
float result=CvRTreespred/cf_prob(trainingData); return result;
}
void QassifierCvRTrees::/ra/'«(const Mat& trainingData, const Mat& responses){
// Train the CvRTrees CvRTrees CvRTrees;
CvRTrees./ra/'«(trainingData, CV_ROW_SAMPLE, responses);
CvRTrees.saveC'CvRTreesQassifier.xmlT);
#ifdef DEBUG
std::cout<<"classifier trained"<<std::endl;
#endif
}
bool ClassifierCvRTrees::eva/_resM/(float result, int& precision){
float resTheshold=0.8; float resMinTheshold=0.55; if(result>resTheshold| 11 -
result>resTheshold) { precision=2; if(result>0.5){
return true;
}else{
return false;
}
}else if(result>resMinTheshold| 11 -result>resTheshold){ precision=1;
if(result>0.5){

```

```

return true;//second class-trash }else{
return false;
}
}else{
precision=0; return false;
}
}
float ClassifierCvRTrees::getNeutralResult(){ return 0.5;
}
ClassifierTrainer
Classifier.h
#include "segment.h"
using namespace cv;
class ClassifierTrainer {
std::string path;
std::vector<Mat> images;
std::vector<int> trainedPointsMarkers;
void loadFilesFromFolder(std::string pfolder="For_training");
void showImages();
Mat convertVectorOfVectorsFtoMat(std::vector<std::vector<float>> vec);
void serializeSegmentsAndResponses(std::string fileName,
std::vector<Segment>& segments, std::vector<int>& responses);
void loadSerializedSegmentsAndResponses(std::string fileName,
std::vector<Segment>& segments, std::vector<int>& responses); public:
ClassifierTrainer(std::string ppath); void trainClassifier();
#endif // CLASSIFIERTRAINER_H
Classifier.cpp
ClassifierTrainer::ClassifierTrainer(std::string ppath)
{
path=ppath;
}
void ClassifierTrainer::loadFilesFromFolder(std::string pfolder)
{
std::vector<String> filenames, filenamesTrue, filenamesFalse; std::string
folderTrue = pfolder+std::string("\\True"); std::string folderFalse =
pfolder+std::string("\\False"); try{
glob(folderTrue, filenamesTrue); glob(folderFalse, filenamesFalse);
}catch(...){
throw std::exception("folder doesn't exist or folder structure is incorrect");
}
filenames=filenamesTrue;
filenames.reserve(filenamesFalse.size()+filenames.size());
filenames.insert(filenames.end(), filenamesFalse.begin(),
filenamesFalse.end());for(int i=0 ;i<filenamesTrue.size() ;i++) {
trainedPointsMarkers.push_back(1);

```

```

}
for(int i=0 ;i<filenamesFalse. size();i++) {
trainedPointsMarkers.push_back(0);
}
for(size_t i = 0; i < filenames.size(); ++i)
{
Mat img = imread(filenames[i], CV_LOAD_IMAGE_COLOR); double
kNewWidth=700;
double kNewHeight=(kNewWidth/img.cols)*img.rows;
resize(img,img,cvSize(kNewWidth, kNewHeight)); images.push_back(img);
}
}
void ClassifierTrainer::showImages()
{
for(int i=0;i<images.size();i++){
String str="image "+std::to_string(i)+" label
"+std::to_string(trainedPointsMarkers[i]); imshow(str, images[i]);
}
}
void ClassifierTrainer::serializeSegmentsAndResponses(std::string fileName,
std::vector<Segment>& segments, std::vector<int>& responses){
std::string segmentsFileName=fileName+"segments.xml";
FileStorage fs(segmentsFileName, FileStorage::WRITE); const char *
segmStr="segments"; fs << segmStr; fs << "{";
for (int i = 0; i < segments.size(); i++)
{
char intStr [33]; itoa(i, intStr, 10);
std::string str=segmStr + string(intStr); fs << str;
Segment tmp = segments[i]; fs << tmp;
}
fs << "}"; fs.release();
std::string responsesFileName=fileName+"responses.xml";
fs.o/>e<<(responsesFileName, FileStorage::WRITE); const char *
respStr="responses"; write( fs , respStr, responses );
fs.release();
}
void ClassifierTrainer::loadSerializedSegmentsAndResponses(std::string
fileName, std::vector<Segment>& segments, std::vector<int>& responses){
std::string segmentsFileName=fileName+"segments.xml";
FileStorage fs(segmentsFileName, FileStorage::READ); const char *
segmStr="segments";
segments.clear();
FileNode fn = fs[segmStr]; if (fn.empty()){ return;
}

```

```

FileNodeIterator current = fn.begin(), it_end = fn.end(); // Go through the node for
(; current != it_end; ++current)
{
    Segment tmp;
    FileNode item = *current; item >> tmp;
    segments.push_back(tmp);
}
fs.release();
std::string responsesFileName=fileName+"responses.xml";
fs.open(responsesFileName, FileStorage::READ); const char *
respStr="responses";
FileNode responsesFileNode = fs[respStr]; read( responsesFileNode, responses );
fs.release();
}
void ClassifierTrainer::trainClassifier() {
#ifdef DEBUG int segmcounter=0;
#endif
loadFilesFromFolder(path); std::vector<Segment> segments; for(size_t i = 0; i <
images.size(); i++) {
    SegmenterGrabCut * sgk=new SegmenterGrabCut();
    Segment segm=sgk->getSegment(images[i], 5, 1); segments.push_back(segm);
#ifdef DEBUG
    segmcounter++;
    std::cout<<"segment " <<segmcounter<<" done"<<std::endl;
#endif
}
std::vector<Segment> serSegments; std::vector<int> serResponses;
loadSerializedSegmentsAndResponses("Serialized", serSegments, serResponses);
for(int i=0; i<serSegments.size();i++){
    serSegments[i].setSegmentId(segments.size()+i);
}
segments.insert( segments.end(), serSegments.begin(), serSegments.end() );
trainedPointsMarkers.insert( trainedPointsMarkers.end(), serResponses.begin(),
serResponses.end() ); serializeSegmentsAndResponses("Serialized", segments,
trainedPointsMarkers); std::vector<std::vector<float>> descriptors; for(size_t i = 0;
i < segments.size(); i++){
    std::vector<float> res=segments[i].getDescriptorsVector();
    descriptors.push_back(res);
}
Mat resMat=convertVectorOfVectorsFtoMat(descriptors);
Mat m(trainedPointsMarkers);
Params::classifier->train(resMat, m);
Mat
ClassifierTrainer::convertVectorOfVectorsFtoMat(std::vector<std::vector<float>>

```

```

vec){ cv::Mat vecMat(vec.size(), vec.at(0).size(), CV_32FC1); for(int i=0;
i<vecMat.rows; ++i){ for(int j=0; j<vecMat.cols; ++j){
vecMat.at<float>(i, j) = vec.at(i).at(j);
}
}
return vecMat;
}
Descriptor
Descriptor.h
using namespace cv;
class Descriptor {
protected:
std::vector<float> ConvertMatFtoVectorF(Mat& mat); public:
Descriptor();
virtual std::vector<float> getDescriptor(Segment& s)=0;
};
class DescriptorMeanColor:public Descriptor
{
public:
DescriptorMeanColor();
virtual std::vector<float> getDescriptor(Segment& s);
};
class DescriptorColorHistogram:public Descriptor
{
public:
DescriptorColorHistogram();
virtual std::vector<float> getDescriptor(Segment& s);
};
class DescriptorAngles:public Descriptor
{
public:
DescriptorAngles();
virtual std::vector<float> getDescriptor(Segment& s);
};
class DescriptorCenterOfMass:public Descriptor
{
public:
DescriptorCenterOfMass();
virtual std::vector<float> getDescriptor(Segment& s);
};
{
public:
DescriptorFilledArea();
virtual std::vector<float> getDescriptor(Segment& s);
};

```

```

class DescriptorLengthHistogram:public Descriptor
{
public:
DescriptorLengthHistogram();
virtual std::vector<float> getDescriptor(Segment& s);
};
#endif // DESCRIPTOR H
Descriptor.cpp
Descriptor::Descriptor()
{
}
std::vector<float> Descriptor::ConvertMatFtoVectorF(Mat& mat){
std::vector<float> array; if (mat.isContinuous()) {
array.assign((float*)mat.datastart, (float*)mat.dataend);
} else {
for (int i = 0; i < mat.rows; ++i) {
array.insert(array.end(), (float*)mat.ptr<uchar>(i),
(float*)mat.ptr<uchar>(i)+mat.cols);
}
}
return array;
}
DescriptorMeanColor::DescriptorMeanColor(){
}
std::vector<float> DescriptorMeanColor::geiDescripfor(Segment& s){
Scalar meanColor= mean(s.getImgROI(), s.getMask()); std::vector<float> result;
for(int i=0; i<4;i++){ result.push_back(meanColor[i]);
}
return result;
}
DescriptorColorHistogram::DescriptorColorHistogram(){
}
std::vector<float> DescriptorColorHistogram::getDescriptor(Segment& s){
std::vector<Mat> bgr_planes; split( s.getImgROI(), bgr_planes );
int histSize = 256;
( for B,G,R ) float range[] = { 0, 256 } ; const float* histRange = { range };
bool uniform = true; bool accumulate = false;
Mat b_hist, g_hist, r_hist; std::vector<Mat> colorHists;
// Compute the histograms:
calcHist( &bgr_planes[0], 1, 0, s.getMask(), b_hist, 1, &histSize, &histRange,
uniform, accumulate ); calcHist( &bgr_planes[1], 1, 0, s.getMask(), g_hist, 1,
&histSize, &histRange, uniform, accumulate ); calcHist( &bgr_planes[2], 1, 0,
s.getMask(), r_hist, 1, &histSize, &histRange, uniform, accumulate ); #if DEBUG
int hist_w = 512; int hist_h = 400; int bin_w = cvRound( (double) hist_w/histSize
);

```

```

Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
[ 0, histImage.rows ]
normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
for( int i = 1; i < histSize; i++ )
{
line( histImage, Point( bin_w*(i-1), hist_h - cvRound(b_hist.at<float>(i-1)) ) ,
Point( bin_w*(i), hist_h - cvRound(b_hist.at<float>(i)) ) ,
Scalar( 255, 0, 0), 2, 8, 0 );
line( histImage, Point( bin_w*(i-1), hist_h - cvRound(g_hist.at<float>(i-1)) ) ,
Point( bin_w*(i), hist_h - cvRound(g_hist.at<float>(i)) ) ,
Scalar( 0, 255, 0), 2, 8, 0 );
line( histImage, Point( bin_w*(i-1), hist_h - cvRound(r_hist.at<float>(i-1)) ) ,
Point( bin_w*(i), hist_h - cvRound(r_hist.at<float>(i)) ) ,
Scalar( 0, 0, 255), 2, 8, 0 );
}
namedWindow("calcHist Demo", CV_WINDOW_AUTOSIZE );
imshow("calcHist Demo", histImage );
#endif
colorHists.push_back(b_hist); colorHists.push_back(g_hist);
colorHists.push_back(r_hist); std::vector<float> finalColorDescriptor; for(int i=0
;i<colorHists.size();i++) {
std::vector<float> colVec=ConvertMatFtoVectorF(colorHists[i]);
finalColorDescriptor.insert( finalColorDescriptor.end(),colVec.begin(),
colVec.end() );
}
return finalColorDescriptor;
}
DescriptorAngles::DescriptorAngles(){
std::vector<float> DescnptorAngles::getDescriptor(Segment&. s){
std::vector<float> angles; std::vector<Point> contours=s.getContours(); size_t
contoursSize=contours.size(); for(int i=0;i<contoursSize;i++){
Point p 1=contours [i%contoursSize];
Point p2=contours [(i+1 )%contours Size];
Point p3=contours[(i+2)%contoursSize];
Vec2i v1(p2.x-p1.x,p2.y-p1.y);
Vec2i v2(p3.x-p2.x,p3.y-p2.y);
float
cos=(v1[0]*v2[0]+v1[1]*v2[1])/(sqrt(pow(v1[0],2)+pow(v1[1],2))*sqrt(pow(v2[0]
,2)+pow(v2[1],2))); float arccos = acos(cos); float angle=arccos*180/3.14;
angles.push_back(angle);
}
cv::Mat anglesM ( angles, true );
float range[] = { 0, 180 } ;

```

```

const float* histRange = { range };
bool uniform = true; bool accumulate = false;
Mat hist;
calcHist( &anglesM, 1, 0, Mat(), hist, 1, &histSize, &histRange, uniform,
accumulate ); //normalize
double normKoeff=angles.size(); divide(normKoeff, hist, hist);
#ifdef DEBUG // Draw the histogram int hist_w = 512; int hist_h = 200; int bin_w =
cvRound( (double) hist_w/histSize );
Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
channel for( int i = 1; i < histSize; i++ )
{
line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist.at<float>(i-1)) ), Point(
bin_w*(i), hist_h - cvRound(hist.at<float>(i)) ),
Scalar( 255, 0, 0), 2, 8, 0 );
}
namedWindow("calcHist Demo2", CV_WINDOW_AUTOSIZE );
imshow("calcHist Demo2", histImage );
#endif
std::vector<float> angVec=ConvertMatFtoVectorF(hist); return angVec;
}
std::vector<float> DescriptorCenterOfMass::gelDescr/>for(Segment& s){ Point2f
cen(0,0);
for ( size_t j=0; j<s.getContours0.size0; j++ )
{
cen.x += s.getContours()[j].x; cen.y += s.getContours()[j].y;
}
cen.x /= s.getContours().size0;//because we have only one contour cen.y /=
s.getContours().size0;
Point2f rectCen(s.getImgROI().cols/2,s.getImgROI0.rows/2); cv::Vec2f diff(cen.x-
rectCen.x, cen.y-rectCen.y); std::vector<float> result; for(int i=0; i<2;i++){
result.push_back(diff[i]);
}
return result;
}
DescriptorFiUedArea::DescriptorFiUedAreaO{
}
std::vector<float> DescriptorFilledArea::gelDescr/>for(Segment& s){
Rect bRect=Rect(1,1,s.getMask0.cols-2,s.getMask0.rows-2);
Mat narrowedMask=s.getMask()(bRect);
float TotalNumberOfPixels = narrowedMask.cols * narrowedMask.rows; float
NonZeroPixels = countNonZero(narrowedMask); std::vector<float> result;
result.push_back(NonZeroPixels/TotalNumberOfPixels); return result;
}
DescriptorLengthHistogram::DescriptorLengthHistogramO{
}

```

```

std::vector<float> DescriptorLengthHistogram::getDescr/>>for(Segment& s){
vector<float> lengths;
std::vector<Point> contours=s.getContours(); size_t contoursSize=contours.size();
for(int i=0;i<contoursSize;i++){
Point p1=contours [i%contoursSize];
Point p2=contours [(i+1)%contoursSize]; float length=sqrt(pow((p2.x-
p1.x),2)+pow((p2.y-p1.y),2)); lengths.push_back(length);
}
cv::Mat lengthsM ( lengths, true );
std::vector<float>::iterator maxLength;
maxLength = std::max_element(lengths.begin(), lengths.end());
float range[] = { 0, *maxLength } ;
const float* histRange = { range };
bool uniform = true; bool accumulate = false;
Mat hist;
calcHist( &lengthsM, 1, 0, Mat(), hist, 1, &histSize, &histRange, uniform,
accumulate ); //normalize
double normKoef=lengths.size(); divide(normKoef, hist, hist);
#ifdef DEBUG // Draw the histogram int hist_w = 512; int hist_h = 200; int bin_w =
cvRound( (double) hist_w/histSize );
Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
{
line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist.at<float>(i-1)) ),
Point( bin_w*(i), hist_h - cvRound(hist.at<float>(i)) ),
Scalar( 255, 0, 0), 2, 8, 0 );
}
namedWindow("calcHist Demo3", CV_WINDOW_AUTOSIZE );
imshow("calcHist Demo3", histImage );
#endif
std::vector<float> lenVec=ConvertMatFtoVectorF(hist); return lenVec;
}
KeypointsDescriptor
KeypointsDescriptor.h
using namespace cv;
class KeypointsDescriptor:public Descriptor
{
public:
KeypointsDescriptor();
virtual Mat descriptKeypoints(Mat& img, std::vector< cv::KeyPoint >&
keypoints)=0; virtual std::vector<float> getDescriptor(Segment& s);
};
class KeypointsDescriptorSIFT: public KeypointsDescriptor {
public:
KeypointsDescriptorSIFT ();

```

```

virtual Mat descriptKeypoints(Mat& img, std::vector< cv::KeyPoint >&
keypoints);
};
class KeypointsDescriptorSURF: public KeypointsDescriptor
{
public:
KeypointsDescriptorSURF();
virtual Mat descriptKeypoints(Mat& img, std::vector< cv::KeyPoint >&
keypoints);
};
KeypointsDescriptor::KeypointsDescriptor()
{
}
std::vector<float> KeypointsDescriptor::getDescr/ptor(Segment& s){ std::vector<
cv::KeyPoint > keypoints; keypoints=Params ::detector-
>detectKeypoints(s.getImgROI()); keypoints=Params ::detector-
>filterKeyPoints(keypoints);
Mat descriptors=descr/ptKeypoints(s.getImgROI(), keypoints); std::vector<float>
result=ConvertMatFtoVectorF(descriptors);
return result;
}
KeypointsDescriptorSIFT ::KeypointsDescriptorSIFT()
{
}
Mat KeypointsDescriptorSIFT::descriptKeypoints(Mat& img, std::vector<
cv::KeyPoint >& keypoints){ Mat descriptors;
cv::DescriptorExtractor *extractor; extractor = new cv::SIFT(0, 3, 0.04, 10.0, 1.6);
extractor->compute(img, keypoints, descriptors); return descriptors;
}
KeypointsDescriptorSURF::KeypointsDescriptorSURF()
{
}
Mat KeypointsDescriptorSURF::descriptKeypoints(Mat& img, std::vector<
cv::KeyPoint >& keypoints){ Mat descriptors;
cv::DescriptorExtractor *extractor;
extractor = new cv::SURF(400); extractor->compute(img, keypoints, descriptors);
return descriptors;
}
KeypointsDetector
KeypointsDetector.h
#ifndef KEYPOINTSDETECTOR_H #define KEYPOINTSDETECTOR_H
#include <opencv\cv.h>
using namespace cv;
class KeypointsDetector {
protected:

```

```

std::vector<cv::KeyPoint> convertVectPoints2dToVecKeyPoints(std::vector<cv:
:Point2f> input); public:
KeypointsDetector();
virtual std::vector< cv::KeyPoint > detectKeypoints(Mat& img)=0;
std::vector<cv::KeyPoint> filterKeyPoints(std::vector<cv: :KeyPoint> keypoints);
};
class KeypointsDetectorHarris: public KeypointsDetector
{
public:
KeypointsDetectorHarris();
std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
class KeypointsDetectorGFTT: public KeypointsDetector
{
public:
KeypointsDetectorGFTT();
std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
class KeypointsDetectorFAST: public KeypointsDetector
{
public:
KeypointsDetectorF AST();
std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
class KeypointsDetectorORB: public KeypointsDetector
{
public:
KeypointsDetectorORB ();
std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
class KeypointsDetectorSIFT: public KeypointsDetector
{
public:
KeypointsDetectorSIFT ();
virtual std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
class KeypointsDetectorSURF: public KeypointsDetector {
public:
KeypointsDetectorSURF();
std::vector< cv::KeyPoint > detectKeypoints(Mat& img);
};
KeypointsDetector::KeypointsDetector()
{
}

```

```

std::vector<cv::KeyPoint>
KeypointsDetector::filterKeyPoints(std::vector<cv::KeyPoint> keypoints)//should
be used before descriptors!!!!!!
{
float threshold=0.02; keypoints.erase(
std::remove_if(keypoints.begin(), keypoints.end(), [&threshold](const KeyPoint &
a){return a.response<threshold;}), keypoints.end()
);
return keypoints;
}
std::vector<cv::KeyPoint>
KeypointsDetector::convertVectPoints2dToVecKeyPoints(std:
:vector<cv::Point2f> input){ std::vector<cv::KeyPoint> keyvec; for( size_t i = 0; i
< input.size(); i++ ) { keyvec.push_back(cv::KeyPoint(input[i], 1.f));
}
return keyvec;
}
KeypointsDetectorHarris::KeypointsDetectorHarris()
{
}
std::vector< cv::KeyPoint > KeypointsDetectorHarris::defecfKeypoi«fc(Mat&
img)
{
Mat src_gray;
cvtColor( img, src_gray, CV_BGR2GRAY );
Mat dst, dst_norm, dst_norm_scaled; dst = Mat::zeros( img.size(), CV_32FC1 );
cornerHarris( src_gray, dst, blockSize, apertureSize, k, BORDER_DEFAULT );
/// Normalizing
normalize( dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat() );
convertScaleAbs( dst_norm, dst_norm_scaled );
/// Drawing a circle around corners int thresh = 200;
for( int j = 0; j < dst_norm.rows ; j++ )
{ for( int i = 0; i < dst_norm.cols; i++ )
{
if( (int) dst_norm.at<float>(j,i) > thresh )
{
circle( dst_norm_scaled, Point( i, j ), 5, Scalar(0), 2, 8, 0 );
}
}
}
}
#ifdef DEBUG /// Showing the result
char* corners_window = "Corners detected"; namedWindow( corners_window,
CV_WINDOW_AUTOSIZE ); imshow( corners_window, dst_norm_scaled );
#endif
std::vector< cv::Point2f > corners;

```

```

Mat mat=dst_norm_scaled; if (mat.isContinuous()) { corners.assign(mat.datastart,
mat.dataend);
} else {
for (int i = 0; i < mat.rows; ++i) {
corners.insert(corners.end(), mat.ptr<uchar>(i), mat.ptr<uchar>(i)+mat.cols);
}
}
return convertVectPoints2dToVecKeyPoints(corners);
}
KeypointsDetectorGFTT ::KeypointsDetectorGFTT ()
{
}
std::vector< cv::KeyPoint > KeypointsDetectorGFTT::detectKeypoints(Mat& img)
{
Mat src_gray, dst; std::vector< cv::Point2f > corners; cvtColor( img, src_gray,
CV_BGR2GRAY ); dst = Mat::zeros( img.size(), CV_32FC1 );
goodFeaturesToTrack(src_gray, corners, 500, 0.03,10, noArray(), 2, true,0.04);
for( size_t i = 0; i < corners.size(); i++ )
{
cv::circle( img, corners[i], 10, cv::Scalar( 255. ), -1 );
}
#ifdef DEBUG
cv::imshow("Keypoints", img );
#endif
return convertVectPoints2dToVecKeyPoints(corners);
}
KeypointsDetectorFAST ::KeypointsDetectorFAST()
{
}
std::vector< cv::KeyPoint > KeypointsDetectorFAST::detectKeypoints(Mat& img)
73
Mat src_gray, dst;
std::vector< cv::KeyPoint > corners;
int threshold=30
; cvtColor( img, src_gray, COLOR_BGR2GRAY );
FAST( src_gray, corners, threshold, nonmaxSuppression);
KeyPointsFilter::retainBest(corners, 5000); for( size_t i = 0; i < corners.size(); i++
)
{
cv::circle( img, corners[i].pt, 10, cv::Scalar( 255. ), -1 );
}
#ifdef DEBUG
cv::imshow("Keypoints", img );
#endif
return corners;

```

```

}
KeypointsDetectorORB ::KeypointsDetectorORB()
{
}
std::vector< cv::KeyPoint > KeypointsDetectorORB::detectKeypoints(Mat& img)
{
    Mat src_gray, dst;
    cvtColor( img, src_gray, CV_BGR2GRAY ); int nfeatures=3 ;//
    float scaleFactor=1.2f; int nlevels=8; int edgeThreshold=31; int firstLevel=0; int
    WTA_K=2;
    int scoreType=ORB::HARRIS_SCORE; int patchSize=31; int fastThreshold=20;
    ORB
    det(nfeatures,scaleFactor,nlevels,edgeThreshold,firstLevel,WTA_K,scoreType,pat
    chSize); std::vector< cv::KeyPoint > corners;
    Mat orbdescriptors;
    det.detect(src_gray, corners);
    for( size_t i = 0; i < corners.size(); i++ )
    {
        cv::circle( img, corners[i].pt, 10, cv::Scalar( 255. ), -1 );
    }
    #if DEBUG
    cv::imshow("Keypoints", img );
    #endif
    return corners;
}
KeypointsDetectorSIFT ::KeypointsDetectorSIFT ()
{
}
std::vector< cv::KeyPoint > KeypointsDetectorSIFT::detectKeypoints(Mat& img)
{
    Mat src_gray, dst;

```

ДОДАТОК Д
Слайди презентації

Дослідження та програмна реалізація методів розпізнавання об'єктів для автоматизації сортування сміття

Керівник: Зайко Тетяна Анатоліївна
к.т.н., доцент

Студент: Костюк Павло Миколайович
КНТ-120м

Рисунок Д.1 – Слайд №1

Об'єкт, предмет та мета роботи

- **Об'єкт дослідження** – процес розпізнавання та класифікації об'єктів за графічними зображеннями.
- **Предмет дослідження** – методи розпізнавання та класифікації об'єктів по зображенням.
- **Метою роботи** є створення методу і програмного комплексу для класифікації певного типу об'єктів і розпізнавання даних об'єктів по зображеннях з декількох точок.

Рисунок Д.2 – Слайд №2

Актуальність роботи та наукова новизна

- В даний час існує безліч завдань, в яких потрібно прийняти деяке рішення в залежності від присутності на зображенні об'єкта або класифікувати його. Здатність розпізнавати вважається основною властивістю біологічних істот, в той час як комп'ютерні системи цією властивістю в повній мірі не володіють.
- Саме тому актуальною залишається задача розробки методів для автоматизації процесів розпізнавання образів.
- Наукову новизну роботи складають використана комбінація методів для такої складної для розпізнавання у зв'язку з величезною внутрішньокласовою мінливістю категорії, як сміття.

Рисунок Д.3 – Слайд №3

Використання методів для розпізнавання образів серед аналогів

Критерії порівняння	Intelligent Garbage Classifier	Lazy Cleaner 9000	Власна розробка
Метод сегментації	Пороговий метод сегментації Watershed	Розрахунок середнього значення кольору RGB і виділення сильно відрізняються від нього частин зображення	Сегментація GrabCut та метод сегментації на графах
Дескриптори	Дескриптор Фур'є, Hu moments дескриптор	Відсутні	Дескриптор локальних особливостей, а також дескриптори центру мас, гістограми кольору, гістограми кутів, гістограми довжин, заповнення простір
Класифікатор	K-local hyperplane nearest neighbours (HKNN)	Відсутній	SVM та Random forest
Інтерфейс користувача	Windows Presentation Foundation	Не надається	Windows Presentation Foundation

Рисунок Д.4 – Слайд №4

Методи, що використовувалися під час роботи

Сегментація:

- сегментації GrabCut;
- сегментація за допомогою розрізів на графах.

Методи машинного навчання:

- метод опорних векторів (SVM);
- метод Random forest.

Рисунок Д.5 – Слайд №5

Робота модуля з навчання класифікатору



Рисунок Д.6 – Слайд №6



Рисунок Д.7 – Слайд №7

Порівняння мов програмування

Критерії порівняння	Мови програмування		
	C++ та OpenCV	Python та TensorFlow	MATLAB
Спільнота	++	++	++
Продуктивність роботи застосунків	++	+	-
Підтримка сегментації зображень для РО	++	±	±
Паралельна обробка запитів та реалізація	++	++	±
Простота розроблення графічного інтерфейсу	+	±	±

Корфедра ПЗ, НУ "Запорізька політехніка"

● 8

Рисунок Д.8 – Слайд №8

Тест-кейси				
Опис	Передумова	Кроки	Очікувані результати	Результат
Позитивні тести				
Перевірка режиму класифікації	1. У каталозі програми знаходиться файл з визначеним класифікатором необхідного типу	1. Вибір параметри рядка 2. При якому типі застосунок виконаний	На консолі виводиться результат класифікації (число 0 або 1)	Процінюю
Перевірка режиму текстової класифікації	1. У каталозі програми знаходиться файл з визначеним класифікатором необхідного типу 2. У каталозі адреси якого передається як аргумент, знаходиться ім'я, що відповідає вимогам програми	1. Вибір параметри рядка	У текстовому файлі з певним розширенням у каталозі програми, виводить результати класифікації (числа 0 або 1)	Процінюю
Перевірка режиму визначення класифікатора	1. У каталозі адреси якого передається як аргумент, знаходиться ім'я, що відповідає вимогам програми	1. Вибір параметри рядка	Сторону або перетиснені класифікатор	Процінюю
Негативні тести				
Перевірка режиму текстової класифікації. Неправильно передано параметр рядка-ім'я до каталогу	1. У каталозі програми знаходиться файл з визначеним класифікатором необхідного типу 2. У каталозі адреси якого передається як аргумент, знаходиться ім'я, що відповідає вимогам застосунку	1. Вибір параметри рядка 2. При якому типі застосунок виконаний	На консолі виводиться повідомлення про помилку. Каталог по якому застосунок завершує роботу	Процінюю
Перевірка режиму класифікації. Відсутній файл необхідного класифікатора	1. У каталозі програми знаходиться файл з визначеним класифікатором необхідного типу	1. Вибір параметри рядка 2. При якому типі застосунок виконаний	На консолі виводиться повідомлення про помилку. Додаток завершує роботу	Процінюю

Рисунок Д.11 – Слайд №11

Висновки

Під час виконання дипломної кваліфікаційної роботи магістра, було розглянуто основні методи автоматичної класифікації об'єктів за зображенням:

- SVM;
- Random forest;
- k-ближчих сусідів.

В результаті оцінки були отримані наступні метрики:

- 1)точність склала 67,6%;
- 2)достовірність склала 68,8%;
- 3)повнота склала 64,7%.

Під час розробки програми та аналізу розробленого програмного комплексу з'явилися ідеї з удосконалення, такі як: використання штучних нейронних мереж з метою видалення шуму та підвищення точності.

Нові можливості зможуть розширити коло вирішення практичних задач, адже зменшення шуму при скануванні в реальному часі значно полегшить роботу та підвищить кількість точно класифікованих об'єктів.

Кафедра ПЗ, НУ "Запорізька політехніка"
11

Рисунок Д.12 – Слайд №12