

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 ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ.
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS
FOR OBJECT RECOGNITION IN IMAGES

Виконав: студент(ка) 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Предко В.В.

(прізвище та ініціали)

Керівник Гофман Є.О.

(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)

Предко Вікторії Юріївни
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів ідентифікації об'єктів на зображеннях. Research And Software Implementation Of Methods For Object Recognition In Images

керівник проєкту (роботи) Гофман Євгеній Олександрович, к.т.н.
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “5” листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) грудень 2021 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. 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
1-4 Основна частина	Гофман Є.О., 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 та дослідження предметної області	2-5 тижні	Розділи 1, 2
3	Розробка структури програми	6-8 тиждень	Розділ 2
4	Розробка програми	9-12 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	13 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	14 тиждень	Додатки
7	Захист роботи	16 тиждень	

Студент(ка)

(підпис) Предко В.Ю.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Гофман Є.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
100 с., 23 рис., 15 табл., 4 дод., 17 джерел.

ІДЕНТИФІКАЦІЯ, ОБ'ЄКТ, ЗОБРАЖЕННЯ, МОВА
ПРОГРАМУВАННЯ C++, OPENCV.

Об'єкт дослідження – процес ідентифікації об'єктів на зображеннях.

Предмет дослідження – методи та програмні засоби виявлення об'єктів на зображеннях.

Метою роботи є дослідження та програмна реалізація методів ідентифікації об'єктів на зображеннях.

Матеріали, методи та технічні засоби: мова програмування C++, середовище розробки Clion, бібліотека OpenCV.

Результати. Розроблено метод та програмне забезпечення, призначені для розв'язання задачі ідентифікації об'єктів на зображеннях на основі машинного навчання.

Практична цінність роботи полягає в розробці програмного забезпечення для розв'язання практичної задачі ідентифікації об'єктів на зображеннях.

Висновки. Розроблено програмне забезпечення, що використовує методи машинного навчання та може застосовуватися для розв'язання завдань ідентифікації об'єктів на зображеннях.

Галузь використання – комп'ютерний зір, віртуальні асистенти, розважальні системи.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 100 pages, 23 figures, 15 tables, 4 appendixes, 17 sources.

IDENTIFICATION, OBJECT, IMAGE, C ++ PROGRAMMING LANGUAGE, OPENCV.

The object of research is the process of identifying objects in images.

The subject of the research is software for detecting objects in images.

The purpose of this work is research and software implementation of methods for identifying objects in images.

Materials, methods and technical tools: C++ programming language, Clion development environment, OpenCV library.

Results. The software to solve the problem of identifying objects in images based on machine learning has been developed.

The practical value of the work lies in the development of software to solve the practical problem of identifying objects in images.

Conclusions. Software that implements that uses machine learning techniques and can be used to solve problems of identifying objects in images has been developed.

Scope of use – computer vision, virtual assistants, entertainment systems.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	9
Вступ	10
1 Аналіз предметної області	12
1.1 Постановка завдання ідентифікації об'єктів на зображеннях	12
1.2 Аналіз методу скануючого вікна для ідентифікації об'єктів.....	13
1.3 Дослідження методів відстеження об'єктів на кадрах	14
1.3.1 Метод оптичного потоку та метод Канаді-Лукаса-Томаші.....	14
1.3.2 Метод активної моделі форми	15
1.3.3 Використання ансамблю випадкових дерев регресії для вирішення задачі виявлення об'єктів	16
1.4 Аналіз програмних засобів для виявлення об'єктів на зображеннях.....	17
1.5 Дослідження методів ідентифікації об'єктів на зображеннях	19
1.5.1 Метод порівняння шаблонів	19
1.5.2 Метод Віюлі-Джонса	20
1.5.3 Штучні нейронні мережі	21
1.5.4 Порівняння методів ідентифікації об'єктів на зображеннях	22
1.6 Висновки за розділом 1	23
2 Матеріали і методи	24
2.1 Вибір мови програмування та інших технологій розробки	24
2.2 Розробка узагальненої схеми алгоритму виявлення об'єктів на зображеннях.....	26
2.2.1 Виявлення об'єктів. Метод Віюлі-Джонса.....	29
2.2.2 Відстеження об'єктів. Активна форма моделі та ансамбль регресорів	32
2.3 Розроблення модифікованого методу ідентифікації об'єктів на зображеннях	34
2.4 Висновки за розділом 2	37
3 Опис програми	38

3.1	Опис основних розроблених класів, структур та функцій	38
3.1.1	Клас Detector	39
3.1.2	Структура Point.....	39
3.1.3	Клас PointToPointFitter.....	40
3.1.4	Структура Rule	41
3.1.5	Структура FeaturePoint	42
3.1.6	Структура RegressionTree.....	42
3.1.7	Клас Regressor.....	43
3.1.8	Клас Cascade	44
3.1.9	Клас RegressorTrainer.....	45
3.1.10	Клас Shape	45
3.1.11	Структура TrainingConfig.....	47
3.1.12	Структура TrainingSample	47
3.1.13	Файли Utilities.h та Utilities.cpp	47
3.1.14	Файл TrainingMain.cpp.....	48
3.1.15	Інші файли.....	49
3.2	Опис особливостей реалізованих алгоритмів.....	49
3.2.1	Підготовка даних до навчання регресору	49
3.2.2	Процес генерації дерев та правил.....	50
3.3	Конфігурація навчання системи.....	52
3.4	Висновки за розділом 3	53
4	Експлуатація, тестування та експериментальне дослідження програми.....	54
4.1	Призначення й умови застосування програми	54
4.2	Характеристики програми	55
4.3	Інструкція по експлуатації програми	55
4.3.1	Звернення до програми.....	55
4.3.2	Вхідні й вихідні дані	56
4.3.3	Повідомлення.....	56
4.4	Тестування та результати роботи з програмою ідентифікації об'єктів на зображеннях	57

4.5 Висновки за розділом 4.....	61
Висновки.....	62
Перелік джерел посилання.....	64
Додаток А Технічне завдання.....	66
Додаток Б Опис програми.....	70
Додаток В Текст програми.....	74
Додаток Д Слайди презентації	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AAM – Active Appearance Model (активна модель форми);

AdaBoost – Adaptive Boosting;

API – application programming interface (прикладний програмний інтерфейс);

OpenCV – Open Source Computer Vision Library (бібліотека комп'ютерного зору з відкритим вихідним кодом);

ООП – об'єктно орієнтоване програмування;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

ТЗ – технічне завдання.

ВСТУП

Останнім часом активного розвитку набуває галузь комп'ютерного зору, пов'язана з ідентифікацією об'єктів на зображеннях. У цій галузі виникає багато актуальних задач, все більше з яких отримують також і практичне застосування. Комп'ютерний зір – це галузь комп'ютерних наук, що займається проблемами отримання комп'ютерами високорівневої інформації з цифрових зображень та відео. Основна мета даної галузі – отримати можливість вирішувати за допомогою комп'ютерних програм усі задачі, на які здатне людське око.

Однією з основних задач в галузі комп'ютерного зору є виявлення об'єктів на зображеннях або відеозаписах, та відстеження виявлених об'єктів на кадрах.

Тому актуальною є задача розробки технології, що дозволить виконувати виявлення та відстеження об'єктів заданого класу на зображеннях.

Метою роботи є дослідження та програмна реалізація методів ідентифікації об'єктів на зображеннях.

Для досягнення поставленої мети в роботі потрібно вирішити такі задачі:

- виконати дослідження методів та систем виявлення та відстеження об'єктів на зображеннях;
- дослідити існуючу технологію, що дозволяє вирішувати задачу виявлення об'єктів заданого класу та відстеження виявлених об'єктів;
- розробити модифікований метод, що дозволяє поліпшити точність виявлення та прискорити швидкодію системи;
- розробити програму ідентифікації об'єктів заданого класу на зображеннях;
- виконати експериментальне дослідження програми, створеної на основі розробленого методу.

Об'єкт дослідження – процес ідентифікації об'єктів на зображеннях.

Предмет дослідження – методи та програмні засоби виявлення об'єктів на зображеннях.

Методи дослідження. Для розробки технології застосовані методи теорії комп'ютерного зору та інтелектуального аналізу даних.

Наукова новизна одержаних результатів. Розроблено удосконалений ідентифікації об'єктів на зображеннях, у якому, на відміну від існуючих, на етапі виявлення облич використовується модифікований алгоритм для ідентифікації об'єктів, обернених під довільним кутом, крім того, використовується вдосконалений процесу навчання системи шляхом використання кросвалідації, що дозволяє підвищити точність ідентифікації об'єктів.

Практичне значення одержаних результатів полягає у тому, що розроблено програмне забезпечення для ідентифікації об'єктів на зображеннях, яке може бути використано у різних прикладних застосунках від зору роботів до розпізнавання образів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання ідентифікації об'єктів на зображеннях

В галузі машинного зору існують дві окремі задачі: виявлення об'єктів заданого класу на зображенні та відстеженні певних об'єктів на послідовності зображень.

Завдання виявлення об'єктів розглядають як задачу бінарної класифікації: навчений на тестових даних класифікатор приймає рішення, чи належить дане зображення до шуканого класу.

Часто використовується техніка вікна ковзання, або скануючого вікна. При використанні такого підходу, класифікатор аналізує не усе зображення, а різні його частини, як правило прямокутної чи квадратної форми. Послідовно обираються різні частини зображення різного розміру та з різним положенням на вихідному зображенні. Таким чином можливо визначати об'єкти, що мають різний розмір та можуть розміщуватися у довільній частині кадру.

Об'єкт треба вміти відрізнити від фону та знаходити певні характеристичні точки, за якими потім проводиться відстеження.

Задача виявлення об'єктів на зображеннях може бути розділена на дві підзадачі:

- виявлення довільних об'єктів, що задаються “он-лайн”, тобто у процесі роботи алгоритму;
- виявлення об'єктів, що належать до заздалегідь відомого класу.

У першому випадку, алгоритми виявлення мають більшу гнучкість у роботі, бо не обмежуються певним набором класів, а можуть визначати довільні об'єкти без перенавчання. Недоліком такого підходу є відносно невисока точність виявлення, бо навчання трекеру відбувається безпосередньо у процесі обробки зображення, тому відсутні якісні матеріали для навчання алгоритму визначення об'єктів. Алгоритми даного класу можуть часто “втрачати” об'єкти, що відстежуються, та давати велику кількість хибнопозитивних результатів.

Для виявлення об'єктів певного заздалегідь відомого класу застосовується попередньо навчений класифікатор. Такі алгоритми мають більш високу точність та надійність визначення шуканих об'єктів та кращу загальну якість роботи.

1.2 Аналіз методу скануючого вікна для ідентифікації об'єктів

Випадки, коли шуканий об'єкт займає усю площу зображення, на якому проводиться пошук, є досить нечастими. Навпаки, у більшості випадків шуканий об'єкт може бути будь-якого довільного розміру та бути розташованим у різних частинах зображень [1]-[4].

Для того, щоб мати змогу виявляти об'єкти у таких умовах, найчастіше застосовують метод скануючого вікна, яке також називають вікном ковзання.

Під вікном у даному випадку мається на увазі деяка ділянка вихідного зображення, яка передається на вхід класифікатора. Після цього кроку можна отримати відповідь, чи знаходиться шуканий об'єкт в даній частині зображення [1]-[4].

Для найбільш ефективного пошуку скануюче вікно варіюють у розмірах у процесі пошуку. Також обирається оптимальний шаг для зміщення вікна на кожному кроці, який дозволяє послідовно проаналізувати усю площу зображення. При цьому довжину кроку обирають таку, щоб кожне наступне вікно перекривало частину попереднього (як правило, не менше половини площі) [1]-[4].

Даний метод є досить простим, але добре підходить для локалізації та виявлення об'єктів незалежно від їх масштабу та положення на вихідному зображенні.

Проте, можуть виникати проблеми з пошуком об'єктів, пропорції обмежуючих прямокутників яких сильно відрізняються від пропорцій прямокутнику скануючого вікна. Подолати цей недолік можна налаштуванням

пропорцій вікна для пошуку, або змінюючи його пропорції безпосередньо під час процесу пошуку [1]-[4].

Також, при використанні скануючого вікна можлива ситуація, коли один і той самий об'єкт буде захоплений у декілька різних близько розташованих вікон і, таким чином, порахований декілька разів. Даний недолік може бути усунутий шляхом так званого “пригнічення немаксимальних вікон”: серед усіх перекриваючих одне одного вікон обираються вікна з локальним максимумом класифікатора [1].

1.3 Дослідження методів відстеження об'єктів на кадрах

Задача відстеження об'єктів на зображеннях може бути сформульована, як визначення позиції шуканого об'єкту на кадрах.

Сучасні методи та технології мають достатню швидкодію, щоб виконувати виявлення шуканого об'єкту на кожному кадрі. Але це не є оптимальним підходом як з точки зору швидкодії, так і отримуваної інформації [3].

1.3.1 Метод оптичного потоку та метод Канаді-Лукаса-Томаші

Оптичний потік – це представлення видимого сліду руху об'єктів, поверхонь, і граней візуальної сцени, що спостерігається під час відносного руху між спостерігачем (наприклад, око людини або камера) і сцени [4].

Використання оптичного потоку дозволяє відстежувати видимий рух усіх точок зображення. Метод оптичного потоку використовується в робототехніці та в галузі комп'ютерного зору для виявлення та відстеження об'єктів, виявлення руху тощо.

Проте, використання щільного оптичного потоку на відео, особливо з великою роздільною здатністю, потребує багато обчислень, та є надлишково складним для задачі, поставленої в роботі.

Канаді, Лукас та Томаші запропонували метод, заснований на методі оптичного потоку, але, замість відстеження щільного потоку, тобто, усієї площини зображення, було запропоновано обирати окремі характеристичні точки на області зображення [5].

Даний алгоритм є ефективним для виявлення об'єктів з точки зору швидкодії та надійності результату, але має суттєві обмеження:

- вимагається постійність яскравості характеристичних точок: для ефективної роботи алгоритму, пікселі, з яких складається зображення об'єкту, що відстежується, не повинні змінювати свій зовнішній вигляд (у чорно-білому поданні це означає, що значення інтенсивності пікселя повинно бути незмінним);

- “просторова узгодженість”: для ефективної роботи алгоритму робиться припущення, що сусідні точки, які належать до однієї поверхні, рухаються подібно одна до одної та в результаті руху проектуються на близькі одне до одної позиції на новому кадрі [2].

1.3.2 Метод активної моделі форми

Активна модель форми (Active Shape Model, ASM) [6] – це статистичні моделі форми об'єктів, які можуть багаторазово деформуватися, для підгонки до об'єкта, що присутній на новому зображенні. Ці моделі були розроблені Тімом Кутесом та Крісом Тейлором в 1995 році [1]-[4].

У контексті даного методу під поняттям форми об'єкта мається на увазі множина точок, що контролюється формою моделі. При цьому, форми для окремих екземплярів обмежені моделлю розподілу точок статистичної форми моделі, щоб модель могла змінюватися тільки в межах розмічених прикладів з навчальної вибірки.

Мета алгоритму знаходження та опису форми об'єктів на зображенні за допомогою деформованих моделей – зіставити модель з новим зображенням.

Алгоритм складається з двох дій, що чергуються:

- пошук на зображенні навколо кожної точки кращої позиції для даної точки;
- оновлення параметрів моделі шляхом максимальної відповідності умовам з новими знайденими позиціями.

Щоб знайти кращу позицію для кожної точки можна шукати чіткі краї, а можна поєднати статистичну модель з тим, що очікується для даної точки. Оригінальний метод передбачає використання відстані Махаланобіса для обчислення найкращої позиції для кожного орієнтира точки.

Цей метод широко застосовується для аналізу зображень облич, механічних вузлів та медичних зображень (в 2D та 3D).

Алгоритм використовує різницю між поточною оцінкою зовнішнього вигляду і цільового зображення, використовуючи процес оптимізації. Користуючись методом найменших квадратів, він може підстроюватись до нових зображень дуже швидко.

Алгоритм тісно пов'язаний з іншим поняттям комп'ютерного зору – активною моделлю зовнішнього вигляду.

Активна модель зовнішнього вигляду (Active Appearance Model, AAM) [7] – алгоритм комп'ютерного зору для узгодження (підбирання) статистичної моделі форми об'єкта та зовнішнього вигляду до нового зображення. Вони будуються під час тренувального етапу використовуючи набір зображень разом з координатами орієнтирів на них. Такий підхід широко використовується для узгодження і відстеження обличчя і для інтерпретації медичних зображень.

1.3.3 Використання ансамблю випадкових дерев регресії для вирішення задачі виявлення об'єктів

Ансамбль дерев регресії або ліс випадкових вирішуючих дерев [8] – один із алгоритмів, що використовується для задачі автоматичної розмітки об'єктів заданого класу за заздалегідь відомим шаблоном, що складається з

ключових точок деякого усередненого контуру для об'єктів даного класу. Особливістю даного підходу є те, що в вершинах дерева використовуються слабкі класифікатори, і в задачах обробки зображень це можуть бути порівняння різниць пікселів із поточного дескриптору з фіксованим значенням. В той же час, цей алгоритм передбачає створення великої кількості випадкових невисоких дерев, сумарний результат роботи яких дає хороший загальний результат. Таку сукупність дерев називають ансамблем, вона і буде виступати у якості одного регресора.

1.4 Аналіз програмних засобів для виявлення об'єктів на зображеннях

OpenCV [9] (Open Source Computer Vision Library, бібліотека комп'ютерного зору з відкритим вихідним кодом) – бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека розроблена Intel і на тепер підтримується Willow Garage та Itseez. Реалізована на C / C++, також розробляється для Python, Java, Ruby, Matlab, Lua та інших мов. Може вільно використовуватися в академічних та комерційних цілях – поширюється на умовах ліцензії BSD. OpenCV – найбільш популярна та різноманітна за функціоналом бібліотека у цій сфері. Завдяки відкритості та невеликій кількості ліцензійних обмежень OpenCV має велике співтовариство розробників та швидко розвивається. Бібліотека містить понад 2500 оптимізованих алгоритмів, серед яких повний набір як класичних так і практичних алгоритмів машинного навчання і комп'ютерного зору [9].

Проте великий розмір цієї бібліотеки є також і наслідком декількох її недоліків: не дотримано єдиного стилю написання коду у різних модулях, деякі алгоритми є недостатньо оптимізованими та протестованими [9].

Google Vision API – платформа Google, що надає технології комп'ютерного зору, що можуть бути використані для виявлення об'єктів на

зображеннях. Але дана платформа основана на мережевому інтерфейсі і надає лише публічний API, доступний через Інтернет, що повністю виключає можливість використання її функціоналу в реальному часі [9].

MATLAB – пакет прикладних програм для числового аналізу, а також мова програмування, що використовується в даному пакеті. Система створена компанією The MathWorks і є зручним засобом для роботи з математичними матрицями, малювання функцій, роботи з алгоритмами, створення робочих оболонок (user interfaces) з програмами в інших мовах програмування. В той же час MATLAB має певний набір функцій для роботи з комп'ютерним зором, що об'єднані у модулі під назвою Computer Vision System Toolbox. Computer Vision System Toolbox надає алгоритми, функції та програми для проектування і моделювання алгоритмів комп'ютерного зору й обробки відео. Ви можете виконувати виявлення об'єктів і стеження, функцію виявлення та вилучення особливостей, стерео бачення, калібрування камери, і виявлення руху. Система інструментів також надає інструменти для обробки і відображення відео, анотації об'єктів, малювання графіки тощо. Алгоритми доступні як функції Matlab, системні об'єкти і Simulink блоки.

Dlib – сучасна крос-платформна бібліотека, яка містить алгоритми машинного навчання і інструменти для створення складного програмного забезпечення в C++ для вирішення реальних задач. Вона використовується в промисловості і наукових колах в широкому діапазоні областей, включаючи робототехніку, вбудовані пристрої, мобільні телефони і великі високопродуктивні обчислювальні середовища. Розвиток бібліотеки почався в 2002 році. Dlib стала включати широкую множину інструментів. Станом на 2016 рік, вона містить компоненти програмного забезпечення для роботи з мережами, потоками, графічним користувацьким інтерфейсом, структур даних, лінійної алгебри, машинного навчання, обробки зображень, інтелектуального аналізу даних, XML розбору і розбору тексту, чисельної оптимізації, Байєсовських мереж та ін. В останні роки велика частина

розробки бібліотеки була зосереджена на створенні широкого набору статистичних інструментів машинного навчання.

CCV або C-based/Cached/Core Computer Vision Library – сучасна бібліотека алгоритмів комп’ютерного бачення з відкритим вихідним кодом. Написана на мові програмування C, вона є кросплатформеною, легковісною, не потребує статичного компонування. CCV надає можливості обробки зображень, пошуку об’єктів, трекінгу, класифікації, використання нейронних мереж, в тому числі й згорткових нейронних мереж. Реалізація бібліотеки є високо продуктивною, містить власні методи керування пам’яттю, має документацію. Та в цілому CCV є менш потужною та менш популярною у порівнянні із OpenCV.

1.5 Дослідження методів ідентифікації об’єктів на зображеннях

1.5.1 Метод порівняння шаблонів

Порівняння шаблонів – це один з найпростіших методів пошуку об’єктів на зображенні.

Сутність метода полягає у тому, що, маючи зображення шуканого об’єкту у якості зразка, можна попиксельно зіставити даний зразок із різними частинами зображення, на якому проводиться пошук.

Пошук об’єкту виконується на основі “помилки порівняння”, тобто різниці між зразком та деякою ділянкою зображення, на якому відбувається пошук. Ця різниця може бути знайдена різними методами:

- методи зіставлення квадратичної різниці;
- методи зіставлення кореляції;
- методи зіставлення коефіцієнту кореляції;
- нормалізовані методи.

Метод порівняння шаблонів може бути використаний, коли шуканий об’єкт може переміщуватися по зображенню, але не змінює при цьому свого зовнішнього вигляду (тобто, форми, кольору тощо). Якщо ж об’єкт якимось

чином змінює вигляд між кадрами, ефективність пошуку стане неприйнятно низькою.

Даний метод є дуже простим, бо не використовує ніяких засобів класифікації, а виконує лише просте зіставлення, тому є дуже ненадійним та має низьку точність виявлення [2].

1.5.2 Метод Віоли-Джонса

Метод Віоли-Джонса – це один з традиційних методів машинного навчання для пошуку об'єктів на зображеннях. Даний метод може бути застосований для виявлення будь-яких класів об'єктів, але при створенні методу автори орієнтувалися в першу чергу на задачу виявлення облич [1]-[4].

Алгоритм, запропонований авторами даного методу, використовує поєднання простих характеристичних рис дерева рішень за допомогою методу AdaBoost та побудову каскаду із отриманих таким чином класифікаторів для досягнення високої швидкості та точності виявлення [1]-[4].

Важливо зауважити, що метод Віоли-Джонса може бути використаний лише для виявлення об'єктів (тобто, бінарної класифікації зображень), але не для їх розпізнавання [1]-[4].

Алгоритм роботи методу складається з наступних кроків [1]-[4]:

- 1) вибір характеристичних рис Хаара;
- 2) побудова інтегральної форми зображення;
- 3) тренування класифікаторів за допомогою AdaBoost;
- 4) побудова каскаду класифікаторів.

Основними перевагами методу Віоли-Джонса є [1]-[4]:

- низький рівень хибнопозитивних результатів;
- висока швидкодія (до 15 кадрів на секунду);
- стійкість до похибок.

До недоліків методу можна віднести [1]-[4]:

- невисока точність у деяких випадках (візуальні перешкоди, значна зміна форми або положення об'єктів відносно представлених у навчальній вибірці);
- достатньо довгий процес навчання класифікатору;
- застосування безпосередньо методу Віоли-Джонса дозволяє лише виявляти об'єкти, але не відстежувати їх.

1.5.3 Штучні нейронні мережі

На сьогоднішній день, великої популярності набувають алгоритми обробки зображень, які базуються на використанні різних типів штучних нейронних мереж для вирішення задач комп'ютерного зору [1]-[4].

Штучні нейронні мережі застосовуються також і для виявлення та класифікації об'єктів на зображеннях [1]-[4].

Найбільш широко застосованими для задачі виявлення об'єктів є згорткові нейронні мережі. Це один з видів багат шарових перцептронів, який добре підходить для виявлення об'єктів на зображеннях, бо потребує відносно мало попередньої обробки даних, а результат роботи не залежить від зсуву або повороту шуканого об'єкту на зображенні [1]-[4].

Методи, засновані на використанні штучних нейронних мереж, мають такі переваги [1]-[4]:

- можливе виявлення об'єктів декількох класів одним класифікатором;
- в цілому дають більшу точність та надійність результатів.

До недоліків даної групи методів можна віднести [1]-[4]:

- недостатньо висока швидкодія: класифікатори, побудовані на основі штучних нейронних мереж дають досить високу якість результатів, але, як правило, не дають достатньої швидкості виявлення для використання їх в реальному часі (не менше 30 кадрів/сек.);

- високу обчислювальну вартість роботи: для використання класифікаторів, побудованих на основі штучних нейронних мереж, як правило,

потрібна потужніша техніка та більша кількість обчислювальних ресурсів, що обмежує спектр використання таких класифікаторів;

– важко формалізується етап вибору архітектури мережі (кількість нейронів, верств, характер зв'язків).

1.5.4 Порівняння методів ідентифікації об'єктів на зображеннях

У таблиці 1.1 представлено порівняння методів виявлення об'єктів на зображеннях.

Таблиця 1.1 — Порівняння методів виявлення об'єктів на зображеннях

Критерії порівняння	Методи		
	метод порівняння шаблонів	метод Віюлі-Джонса	штучні нейронні мережі
Простота	+	±	–
Надійність	–	+	+
Точність	–	+	+
Швидкість навчання	+	–	+
Швидкість виявлення	–	+	–
Стійкість до змін об'єкта	–	–	+
Розпізнавання об'єктів	–	–	+

Порівняння методів показало, що для вирішення задачі роботи метод порівня шаблонів не підходить, тому що хоч і має досить просту реалізацію, не має достатньої точності та надійності.

Метод Віоли-Джонса та штучні нейронні мережі підходять для вирішення задачі роботи. Зважаючи на простоту реалізації та швидкість виявлення об'єктів було обрано метод Віоли-Джонса для реалізації алгоритму виявлення об'єктів на зображеннях.

Методи відстеження об'єктів були розглянуті у даному розділі, їх порівняння не наводиться, так як задача відстеження не є основною задачею роботи, а лише розширює функціонал системи.

1.6 Висновки за розділом 1

Дослідження існуючих методів виявлення об'єктів заданого класу на зображеннях дозволило сформулювати завдання дослідження: необхідно розробити метод та програмне забезпечення, що дозволять вирішити задачу виявлення об'єктів на зображеннях.

В даній роботі буде розглянуто задачу виявлення облич людей як окремий випадок проблеми виявлення та відстеження об'єктів заданого класу на зображеннях. На даному прикладі буде проведена розробка та виконане експериментально дослідження методів та технологій виявлення об'єктів на зображеннях.

З огляду на аналіз описаних в даному розділі методів та технологій, було вирішено реалізувати технологію виявлення об'єктів на основі методу Віоли-Джонса з використанням характеристичних рис Хаара та з побудовою каскаду класифікаторів, навчених з допомогою методу AdaBoost.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування та інших технологій розробки

Реалізація технології виявлення об'єктів на зображеннях явно не потребує створення класів, організації ієрархії даних або якихось складних архітектурних рішень. Але для полегшення написання коду та покращення його логічної інтерпретації було вирішено об'єднати певні дані та алгоритми у структури та класи.

Для задоволення вимог швидкодії розроблюваної технології потребується можливість безпосередньої швидкої обробки числових даних.

Таким чином, враховуючи вище названі вимоги було вирішено використовувати мову програмування C++ для реалізації системи.

C++ – це мова програмування високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Вона вже протягом багатьох років є потужним інструментом програмування, що дозволяє швидко та зручно оброблювати дані, і водночас, будувати об'єктно-орієнтовані системи через класи, має доповнення до стандартної бібліотеки порівняно з C і підтримує обробку винятків, простори імен, вбудовані функції та перевантаження імен і операторів [14].

Також розробка технології виявлення та відстеження об'єктів потребує можливостей тривіальної роботи з зображеннями, такої як – завантаження, обрізання, масштабування, зміна кольору тощо, тому було вирішено підключити до проекту відкриту бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення OpenCV.

OpenCV – бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення

зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях [15].

Для даної системи немає потреби розроблювати користувацький візуальний інтерфейс, так як програма є лише реалізацією технології, що вирішує окремі задачі комп'ютерного зору і може бути використана в інших програмних системах. Тому для демонстрації можливостей виявлення обличчя та трекінгу було вирішено використовувати вбудовані функції OpenCV, які надають можливість отримувати зображення з веб-камери чи файлу та виводити його на екран.

Для збереження та відтворення результатів навчання реалізована примітивна серіалізація регресорів у бінарний файл.

У таблиці 2.1 надано порівняльну характеристику мов програмування, що можуть застосовуватися для програмної реалізації методів ідентифікації об'єктів на зображеннях.

Таблиця 2.1 – Порівняння мов програмування для програмної реалізації методів ідентифікації об'єктів на зображеннях

Критерій порівняння мов програмування	C++	Java	C#
Зручність використання для програмної реалізації методів ідентифікації об'єктів на зображеннях	+	±	—
Наявність бібліотек для ідентифікації об'єктів на зображеннях	+	±	±
Можливість обробки баз даних	+	+	+
Зручність обробки виключних ситуацій	+	±	±

Продовження таблиці 2.1

Критерій порівняння мов програмування	C++	Java	C#
Засоби створення інтерфейсу користувача	+	±	±

Таким чином для програмної реалізації методів ідентифікації об'єктів на зображеннях обрано мову програмування C++.

2.2 Розробка узагальненої схеми алгоритму виявлення об'єктів на зображеннях

Як вже було згадано у попередньому розділі, для успішного вирішення поставленої задачі, розроблювана технологія повинна вирішувати дві підзадачі:

- виявлення об'єкту заданого класу у кадрі;
- відстеження виявленого об'єкту при його переміщенні.

Для вирішення підзадачі виявлення об'єктів планується використовувати підхід на основі методу Віюлі-Джонса з використанням характеристичних рис Хаара та навчанням класифікаторів за допомогою методу AdaBoost.

Запропонована у даній роботі технологія виявлення об'єктів на зображеннях працює за наступним алгоритмом, що працює ітеративно, кадр за кадром:

Етап 1. Отримання вхідних даних.

На цьому кроці на вхід алгоритм отримує поточний кадр.

Етап 2. Пошук об'єктів.

Якщо на попередніх ітераціях не було виявлено шуканого об'єкту або при накопиченні певного значення похибки відстеження, виконується пошук об'єктів.

На етапі пошуку виконуються наступні кроки:

- 1) з використанням скануючого вікна класифікатор аналізує кожну ділянку зображення;
- 2) якщо для даного вікна класифікатор дає позитивний результат, вважаємо, що об'єкт знайдено, і дана ділянка зображення передається на наступний крок;
- 3) якщо у процесі пошуку не було знайдено жодного об'єкту заданого класу, встановлюється флаг необхідності виявлення на наступній ітерації алгоритм одразу переходить до обробки наступного кадру.

Етап 3. Відстеження.

Для кожного знайденого на поточній або у попередніх ітераціях об'єкту виконується етап відстеження.

На даному етапі виконуються наступні кроки:

- 1) якщо даний об'єкт був виявлений на поточній ітерації (тобто, з'явився у даному кадрі), то за допомогою методу активної форми регресори узгоджують усереднену форму із поточним об'єктом;
- 2) якщо даний об'єкт був присутній і на поточному і на попередньому кадрі, регресори намагаються провести узгодження активної форми, асоційованої з даним об'єктом на попередній ітерації.

Етап 4. Обробка результатів ітерації.

Для усіх об'єктів заданого класу, що були знайдені на поточному кадрі, зберігається інформація про їх положення та активні моделі форми; при цьому флаг необхідності виявлення не встановлюється і на наступному кадрі алгоритм опрацьовує вже відомі об'єкти.

Етап 5. Завершення ітерації.

Алгоритм переходить до обробки наступного кадру.

Узагальнена блок-схема роботи алгоритму запропонованої технології наведена на рис. 2.1.

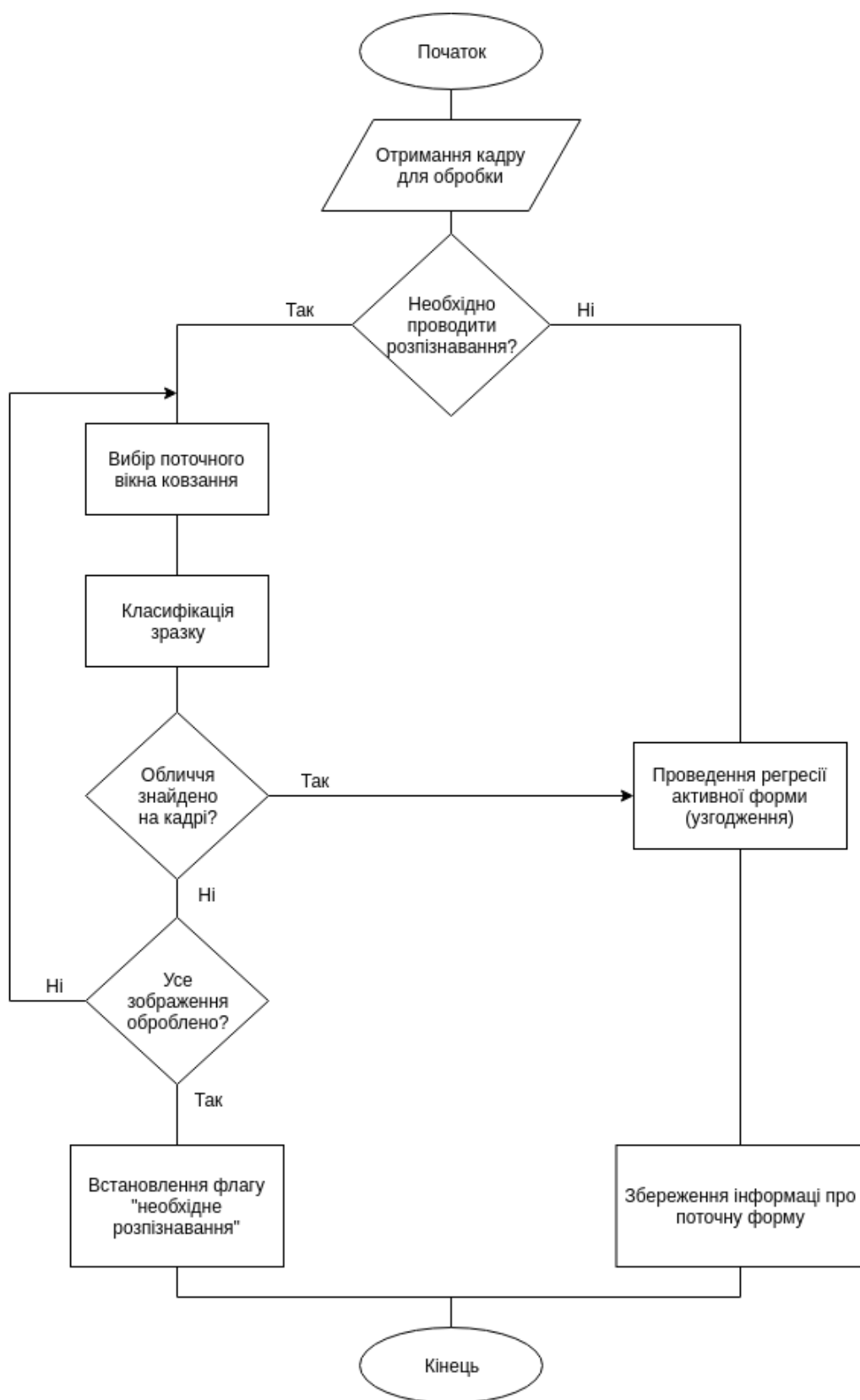


Рисунок 2.1 – Узагальнена блок-схема роботи алгоритму

2.2.1 Виявлення об'єктів. Метод Віоли-Джонса

Для вирішення підзадачі виявлення об'єктів у розроблюваній технології використовується підхід на основі методу Віоли-Джонса.

Метод був запропонований в 2001 році Паулом Віолою та Майклом Джонсом. Алгоритм може розпізнавати різні класи зображень, проте основним завданням при його створенні було виявлення облич.

На сьогодні метод Віоли-Джонса є найбільш популярним методом для пошуку області обличчя на зображеннях через його високу швидкість і ефективність. Цей метод зробив можливим реалізацію виділення облич на зображеннях у реальному часі в практичних додатках.

Алгоритм роботи методу складається з наступних кроків:

- 1) вибір характеристичних рис Хаара;
- 2) побудова інтегральної форми зображення;
- 3) тренування класифікаторів за допомогою AdaBoost;
- 4) побудова каскаду класифікаторів.

Приклад характеристичних рис Хаара наведено на рис 2.2, [10].

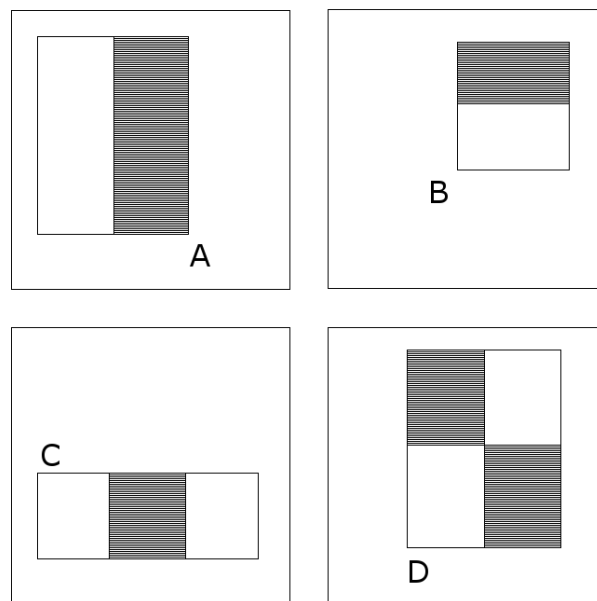


Рисунок 2.2 – Характеристичні риси Хаара

Даний підхід передбачує використання простих характеристичних рис прямокутної форми, які, в свою чергу, складаються з 2-4 чорних та білих прямокутників.

Зображення, що аналізується, подається у чорно-білому режимі. Значенням кожного пікселю є його інтенсивність. Значення кожної ознаки розраховується як різниця між сумою значень пікселів у білих прямокутниках та сумою пікселів у чорних прямокутниках. Розрахунок значення кожної риси описаний формулою (2.1) [10].

$$F = \sum(p_w) - \sum(p_b), \quad (2.1)$$

де F – значення ознаки,

p_w – значення пікселів у білих прямокутниках,

p_b – значення пікселів у чорних прямокутниках.

За рахунок простоти ознак, а також при використанні інтегральної форми зображення обчислення значення кожної ознаки виконується за константний час. Проте, висока швидкість обрахунку ознаки не компенсує значну кількість різних можливих ознак. Наприклад, при стандартному розмірі аналізованого зображення в 24x24 пікселя, можливі 162 тисячі різних ознак і розраховувати їх усі занадто довго для використання у алгоритмі, що працює у реальному часі. Тому в алгоритмі Віоли-Джонса використовується варіація алгоритму навчання AdaBoost, як для вибору ознак, так і для настройки класифікаторів. При використанні AdaBoost обирається обмежена кількість найбільш інформативних ознак, що дозволяє зменшити кількість необхідних розрахунків без втрати точності виявлення.

Алгоритм навчання будує “сильний” класифікатор з лінійної комбінації “слабких” (якими являються ознака Хаара), при цьому кожному зі слабких класифікаторів у процесі навчання присвоюється деяка вага. Принцип побудови “сильного” представлено формулою (2.2).

$$h(x) = \operatorname{sgn} \left(\sum_{j=1}^M a_j \cdot h_j(x) \right), \quad (2.2)$$

де $h(x)$ – “сильний” класифікатор,

M – кількість ознак,

a_j – вага j -го класифікатора,

$h_j(x)$ – j -й слабкий класифікатор, кожен з яких визначається за формулою (2.3).

$$h_j(x) = \begin{cases} -s_j, & \text{якщо } f_j < \Theta_j; \\ s_j, & \text{в іншому випадку,} \end{cases} \quad (2.3)$$

де f_j – значення j -ї ознаки,

s_j – полярність j -го класифікатора, визначається у процесі навчання,

Θ_j – порогове значення j -го класифікатора, визначається у процесі навчання.

Використання каскаду класифікаторів дозволяє отримати високу точність виявлення за рахунок використання декількох “сильних” класифікаторів, які базуються на різних ознаках, не втрачаючи при цьому швидкодії за рахунок раннього відкидання негативних зразків при класифікації.

Каскад будується так, що перші класифікатори є складаються з простих ознак і не потребують складного обчислення, але при цьому пропускають усі лише позитивні зразки. Наступні класифікатори є більш складними та орієнтуються на відсіювання хибнопозитивних зразків. Таким чином, зразок, що проходить по усіх класифікаторах з каскаду, буде вважатися позитивно класифікованим, усі інші зразки будуть відкинуті, не доходячи до кінця [10].

2.2.2 Відстеження об'єктів. Активна форма моделі та ансамбль регресорів

Для вирішення підзадачі відстеження об'єктів на прикладі обличч використовується метод активної форми моделі, а саме – активної форми зовнішнього вигляду. Така модель являє собою множина характеристичних точок, певним чином розташованих одна відносно одної, що складають приблизну форму моделі об'єкту заданого класу. Така модель будується у вигляді усереднених значень у процесі навчання алгоритму відстеження.

Використання активної моделі форми для відстеження дозволяє отримувати в результаті роботи алгоритму не просто позицію об'єкту, представлену обмежувальним прямокутником, а ще й інформацію про те, як саме розташований об'єкт та взаємне розташування його частин. У випадку відстеження обличчя можна отримувати таку інформацію: нахил, кут повороту, відкриті чи закриті рот та очі тощо.

Безпосередньо задача відстеження положення об'єкту вирішується за рахунок узгодження позицій ключових точок моделі із дійсною формою об'єкту, що відстежується, послідовно для кожного кадру. Результатом узгодження буде перетворення подібності – тобто, інформація про переміщення кожної з точок, а з нею і про перетворення форми в цілому.

Для обчислення такого перетворення та вирішення задачі узгодження форми моделі використовується ансамбль дерев регресії.

Ансамбль дерев регресії – один із алгоритмів, що використовується для задачі автоматичної розмітки об'єктів. Його перевагою є те, що слабкі класифікатори, що містяться в вершинах дерева, можуть бути просто порівняннями різниці пікселів із поточного зразка з фіксованим значенням. Завдяки цьому можна досягти високої швидкодії алгоритму, адже порівняння двох значень – швидка та елементарна операція. В той же час, завдяки швидкодії можна використати високу кількість таких випадкових невисоких

дерев, сумарний результат роботи яких дає хорошу загальну якість. Таку сукупність дерев називають ансамблем, вона і виступає у якості одного регресора. Сама регресія відбувається за рахунок складання трансформацій, що знаходяться в листах кожного з дерев, в які потрапляє поточний зразок.

Враховуючи те, що шуканий об'єкт на зображенні може знаходитись в будь-якій його частині, під будь-яким кутом та з будь-яким масштабом, інформацію про ключові точки варто уніфікувати, тобто оброблювати в якихось локальних координатах. Таку локальну систему координат можна знайти, розв'язуючи задачу пошуку перетворення подібності, що переводить точки типової середньої моделі для даного набору ключових точок у координати, такі, що сума відстаней до відповідних поточних ключових точок буде мінімальною. Це в свою чергу зводиться до звичайного методу найменших квадратів [11].

Таким чином, відомі методи виявлення та відстеження об'єктів на зображеннях, що вирішує задачу пошуку та відстеження облич, заснована на класичному методі Віоли-Джонса та використанні ансамблів випадкових вирішуючих дерев [11].

Дана технологія показує хороші результати і розв'язує поставлені задачі, проте вона має ряд недоліків у якості роботи, які можуть бути усунені за рахунок вдосконалення алгоритму та процесу навчання безпосередньо.

До основних напрямів покращення описаної технології можна віднести такі:

- вдосконалення безпосередньо процесу роботи алгоритму;
- вдосконалення процесу навчання.

Також існує ряд екстремальних умов, у яких алгоритм дає поганий результат, що також може бути вирішено при вдосконаленні процесу навчання.

2.3 Розроблення модифікованого методу ідентифікації об'єктів на зображеннях

Для вдосконалення результатів роботи існуючої технології виявлення об'єктів на зображеннях, в даній роботі пропонується імплементація наступних покращень:

- вдосконалення етапу виявлення обличч шляхом модифікації алгоритму для виявлення обличч, обернених під довільним кутом;
- вдосконалення процесу навчання системи шляхом використання кросвалідації.

Наявна технологія виявлення об'єктів здатна знаходити обличчя, кут нахилу яких не перевищує 20-30 градусів у площині зображення.

І, хоча, як правило, на зображеннях обличчя представлені вертикально, у багатьох випадках розташування камери або рух людей може призводити до того, що обличчя буде мати великий кут нахилу, або бути перевернутим.

Можливі два основних підходи до вирішення даної проблеми:

- 1) навчання детектора на базі зразків зображень обличч, які повернуті під різними кутами нахилу;
- 2) модифікація алгоритму виявлення таким чином, щоб перед основним етапом виявлення знаходити ділянки зображення, на яких можуть бути розташовані об'єкти-кандидати під довільним кутом нахилу.

Перший підхід не потребує модифікації алгоритму, але має суттєві недоліки:

- навчальна база повинна бути збільшена у декілька разів за рахунок додавання великої кількості різноманітних зразків зображень; це, в свою чергу, значно збільшить час, необхідний на навчання алгоритму;
- використання більшої кількості різних зразків обличч, обернених на різні кути, збільшить кількість використовуваних ознак; це негативно вплине на час роботи класифікатору, та, таким чином, на час виявлення та швидкодію системи в цілому.

Другий підхід також потребує розширення навчальної бази, але дозволяє мінімізувати втрати часу на виявлення. Для цього на першому етапі виявлення пропонується використовувати більш простий алгоритм пошуку, який дозволить обирати об'єкти-кандидати за простими ознаками, обчислювати їх кут нахилу, перевертати зразок так, щоб він був розташований вертикально, та передавати на вхід класифікатору, що навчений точному виявленню вертикальних облич.

У ролі простого класифікатору пропонується використовувати пошук лінії очей на зображенні. Такий пошук виконується з використанням тих самих ознак Хаара, які було попередньо описано, але при цьому значно спростити класифікатор, бо для пошуку тільки очей потрібно використовувати значно менше ознак і меншу кількість детекторів у каскаді.

Після того, як будуть знайдені координати правого та лівого ока, кут нахилу можливого обличчя може бути обчислений за формулою (2.4).

$$d = \arctan\left(\frac{y_L - y_R}{x_L - x_R}\right), \quad (2.4)$$

де d – кут нахилу,

y_L – координата лівого ока за віссю y ,

y_R – координата правого ока за віссю y ,

x_L – координата лівого ока за віссю x ,

x_R – координата правого ока за віссю x .

Далі, зразок потрібно повернути на знайдений кут, але у протилежному напрямку, отримавши таким чином вертикально зорієнтоване зображення кандидата обличчя. Після цього знайдений зразок може бути переданий на вхід детектору облич.

Кросвалідація – метод оцінювання достовірності математичної моделі з метою перевірки, наскільки результати статистичного аналізу узагальнюються

на незалежному наборі даних. Переважно використовується в задачах прогнозування для оцінювання точності моделі.

Одноразова перехресна перевірка передбачає розбиття вибірки на взаємодоповнювані підвибірки з метою проведення аналізу на одній частині (що називається навчальним набором) і перевірки аналізу на іншій частині (що називається контрольним, або тестовим набором). Для зниження дисперсії здійснюється багаторазова перехресна перевірка із застосуванням різних розбиттів, і результати цих перевірок усереднюються. Метою кросвалідації є тестування спроможності моделі прогнозувати нові дані, які не використовувались для її навчання [12].

Використання кросвалідації дає можливість зменшити вплив таких факторів, як перенавчання або упередженість вибору [13], а також дати уявлення про те, як модель буде узагальнюватися на незалежний набір даних (тобто невідомий набір даних, наприклад, з реальної проблеми).

Для вдосконалення технології виявлення облич на зображеннях пропонується використовувати багаторазову кросвалідацію.

Для виконання кросвалідації з N кроків, вибірка X_L розбивається N у різних способів згідно з формулою (2.5).

$$X^L = X_n^m \cup X_n^k, \quad (2.5)$$

де X_L – початкова вибірка з усіма даними,

X_n^m – навчальна вибірка з m екземплярів,

X_n^k – тестова вибірка з $k = L - m$ екземплярів,

N – кількість кроків кросвалідації,

n – номер розбиття, $n = 1 \dots N$.

Таким чином формула (2.6) описує загальний процес кросвалідації.

$$CV(\mu, X^L) = \frac{1}{N} \cdot \sum_{n=1}^N Q(\mu(X_n^m), X_n^k), \quad (2.6)$$

де μ – алгоритм навчання,

CV – результат кросвалідації,

Q – функція оцінки якості алгоритму.

2.4 Висновки за розділом 2

Для створення програмного забезпечення для ідентифікації об'єктів на зображеннях було обрано мову програмування C++, яка є досить зручною для розробки програм подібного роду. Також роботи з зображеннями, зокрема для завантаження, обрізання, масштабування, зміни кольору тощо, було вирішено підключити до проекту відкриту бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення OpenCV, яка має відкритий код.

Розроблено удосконалений метод ідентифікації об'єктів на зображеннях, у якому, на відміну від існуючих, на етапі виявлення облич використовується модифікований алгоритм для ідентифікації об'єктів, обернених під довільним кутом, крім того, використовується вдосконалений процесу навчання системи шляхом використання кросвалідації, що дозволяє підвищити точність ідентифікації об'єктів.

Встановлено задачі та вимоги до розроблюваної технології. Правильне розуміння цих задач та вимог дозволило обрати алгоритми оптимізації та машинного навчання, що дозволять вирішити ці задачі. Ключовими алгоритмами є каскад із ансамблів дерев регресії та класифікації, та AdaBoost, завдяки якому навчатиметься безпосередньо сама класифікація.

3 ОПИС ПРОГРАМИ

3.1 Опис основних розроблених класів, структур та функцій

Для реалізації спроектованої системи були створені класи та структури. У процесі розробки вони були розподілені по файлам по відповідним каталогам та файлам. Структура системи наведена на рисунку 3.1.

В даному розділі розглянуто структуру програми більш детально.

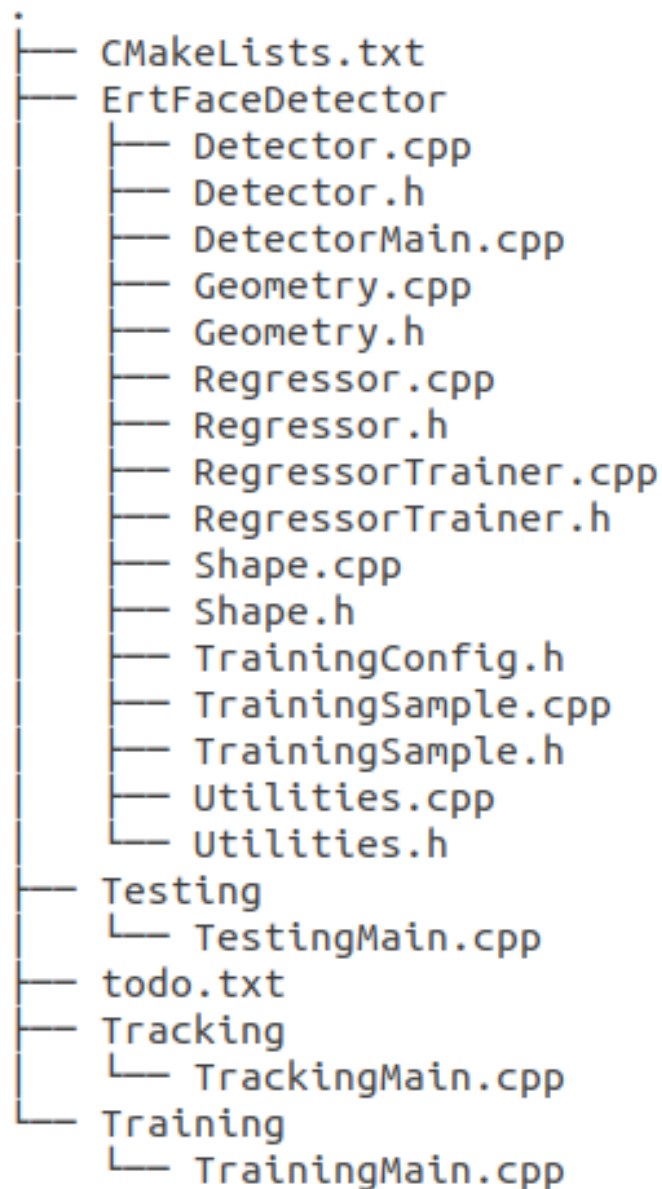


Рисунок 3.1 – Структура розробленої системи

3.1.1 Клас Detector

Опис класу розташовується у файлі Detector.h, реалізація – у файлі Detector.cpp. В цьому класі реалізована логіка пошуку облич. Дані та методи класу описані в таблиці 3.1.

Таблиця 3.1 – Опис класу Detector

Назва	Опис
Дані:	
cascade	каскад для класифікації
currentRegressorIndex	поточний індекс регресора
currentTreeIndex	поточний індекс дерева
Методи:	
detect()	збір даних про кадр
setCascade()	встановлення змінної cascade
setRegressorIndex()	встановлення змінної currentRegressorIndex
getRegressorIndex()	отримання значення змінної currentRegressorIndex
detectSingleScale()	визначення параметрів для відображення кадру на половині екрану
detectMultiScale()	визначення параметрів для відображення кадру на весь екран
validate()	перевірка, чи відповідає завантажений кадр необхідним параметрам

3.1.2 Структура Point

Опис класу розташовується у файлі Geometry.h, реалізація – у файлі Geometry.cpp. Ця структура описує собою точку на площині, містить координати X та Y та реалізує базові операції, які можна виконувати над комплексним числом або двовимірним вектором, а саме: додавання,

віднімання, множення, ділення, пошук оберненого, довжина та норма вектора. Дані та методи структури описані в таблиці 3.2.

Таблиця 3.2 – Опис структури Point

Назва	Опис
Дані:	
x	координата по осі x
y	координата по осі y
Методи:	
operator=()	перевантаження оператора дорівнює для роботи над комплексними числами та двовірним вектором
operator+()	перевантаження оператора додавання для роботи над комплексними числами та двовірним вектором
operator-()	перевантаження оператора віднімання для роботи над комплексними числами та двовірним вектором
operator/()	перевантаження оператора ділення для роботи над комплексними числами та двовірним вектором
operator*()	перевантаження оператора множення для роботи над комплексними числами та двовірним вектором

3.1.3 Клас PointToPointFitter

Опис класу розташовується у файлі Geometry.h, реалізація – у файлі Geometry.cpp. Цей клас реалізує собою алгоритм пошуку перетворення подібності, що мінімізує відстань від однієї множини точок до іншої. Реалізація алгоритму є швидкою й миттєвою (online), тобто дозволяє

отримувати результати після кожної пари доданих точок без перерахунку усіх минулих заданих пар точок. Дані та методи класу описані в таблиці 3.3.

Таблиця 3.3 – Опис класу PointToPointFitter

Назва	Опис
Дані:	
scaleRotate	масштабування та поворот простору
translate	здви́г простору
Методи:	
getTransform()	підгін форми під кадр
add()	додавання та запис координат форми
fit()	обернення, масштабування та переміщення форми

3.1.4 Структура Rule

Опис класу розташовується у файлі Regressor.h, реалізація – у файлі Regressor.cpp. Дані та методи структури описані в таблиці 3.4.

Таблиця 3.4 – Опис структури Rule

Назва	Опис
Дані:	
idx1	перший індекс правила для дерева регресії
idx2	другий індекс правила для дерева регресії
threshold	порогова величина
Методи:	
testRight()	перевірка правила при проходженні дерева регресії

Структура Rule описує собою правило за яким необхідно оброблювати дескриптори форми у вершині дерева регресії, що не є листом. Фактично, вона описує цю вершину.

При проходженні по дереву правило перевіряється наступним чином: чи більша різниця індексів за порогове значення, якщо так, виконується перехід в правого сина, інакше – в лівого.

3.1.5 Структура FeaturePoint

Опис класу розташовується у файлі Regressor.h, реалізація – у файлі Regressor.cpp. Ця невеличка структура містить всього два поля – індекс ключової точки, до якого вона прив'язана, та здви́г в локальних координатах відносно цієї ключової точки, за яким необхідно вираховування значення пікселя для дескриптора форми. Таким чином вона є описом однієї точки-ознаки для вирахування дескриптора. Дані структури описані в таблиці 3.5.

Таблиця 3.5 – Опис структури FeaturePoint

Назва	Опис
Дані:	
idx	індекс ключової точки
offset	здви́г в локальних координатах відносно ключової точки

3.1.6 Структура RegressionTree

Опис класу розташовується у файлі Regressor.h, реалізація – у файлі Regressor.cpp. Ця структура кодує собою одне дерево з ансамблю.

В ній також реалізовані методи, що приймають дескриптор, та повертають індекс листа (що необхідно під час навчання), або значення

трансформації та класифікації із листа, який відповідає даному дескриптору. Дані та методи структури описані в таблиці 3.6.

Таблиця 3.6 – Опис структури RegressionTree

Назва	Опис
Дані:	
rules	правила внутрішніх вершин
leafValues	трансформації для вершин-листів
scores	значення класифікації в листах
threshold	порогове значення для класифікації
scoresMean	математичне очікування
scoresStd	стандартне відхилення значень класифікації для нормалізації
Методи:	
evaluate()	отримання значення трансформації для даного дескриптора
leafIndex()	отримання індексу листа для навчання
outputSize()	отримання кількості трансформацій для вершин-листів
evaluateWithScore()	отримання значення класифікації для даного дескриптора

3.1.7 Клас Regressor

Опис класу розташовується у файлі Regressor.h, реалізація – у файлі Regressor.cpp. Клас Regressor реалізує в собі логіку одного регресора, тобто ансамблю з дерев регресії.

В класі Regressor реалізований метод, який приймає зображення та початкову форму, виконує регресію та класифікацію, та повертає бінарне

значення, чи описує форма на зображенні обличчя, та значення класифікації як коефіцієнт достовірності. Дані та методи класу описані в таблиці 3.7.

Таблиця 3.7 – Опис класу Regressor

Назва	Опис
Дані:	
ensemble	ансамбль дерев
featurePoints	множина FeaturePoint-ів
meanShapePtr	посилання на середню форму
unitsTransform	матриця трансформації із стиснених координат в координати форми
Методи:	
process()	приймає зображення та початкову форму, виконує регресію та класифікацію, та повертає бінарне значення, чи описує форма на зображенні обличчя, та значення класифікації як коефіцієнт достовірності
getMeanShapePtr()	отримання посилання на середню форму

3.1.8 Клас Cascade

Опис класу розташовується у файлі Regressor.h, реалізація – у файлі Regressor.cpp. Цей клас унасліджується від `std::vector<Regressor>`, тобто він зберігає у собі каскад регресорів. Метод класу описані в таблиці 3.8.

Таблиця 3.8 – Опис класу Cascade

Назва	Опис
Методи:	
process()	Виклик методу process() класу Regressor для кожного із регресорів

3.1.9 Клас RegressorTrainer

Опис класу розташовується у файлі RegressorTrainer.h, реалізація – у файлі RegressorTrainer.cpp. В цьому класі реалізований алгоритм навчання регресора. Власних даних клас не має. Методи класу описані в таблиці 3.9.

Таблиця 3.9 – Опис класу RegressorTrainerector

Назва	Опис
Методи:	
train()	приймає тренувальну вибірку, негативну вибірку та конфігурацію для навчання, та повертає навчений регресор
generateRuleForRegression	пошук найкращого правила для регресії
generateRuleForClassification	пошук найкращого правила для класифікації

3.1.10 Клас Shape

Опис класу розташовується у файлі Shape.h, реалізація – у файлі Shape.cpp. Клас Shape описує форму, тобто множину розмічених точок одного обличчя. Дані та методи класу описані в таблиці 3.10.

Таблиця 3.10 – Опис класу Shape

Назва	Опис
Дані:	
localCoordinates	локальні координати ключових точок
translate	здви́г форми
scaleRotate	перетворення подібності

Продовження таблиці 3.10

Назва	Опис
Методи:	
getLandamrkCount()	отримання кількості ключових точок
getAngle()	отримання значення вугла перетворення
getLocalCoordinates()	отримання локальних координат
setLocalCoordinates	встановлення значення локальних координат
getActionUnits()	отримання значення перетворення матриці
setActionUnits()	встановлення значення перетворення матриці
getProjectionTranslate()	отримання проекції здвигу
setProjectionTranslate()	встановлення значення проекції здвигу
getProjectionScale()	отримання проекції масштабування
setProjectionScale()	встановлення значення проекції масштабування
setAngle()	встановлення значення вугла перетворення
getProjectionScaleRotate()	отримання проекції перетворення
setProjectionScaleRotate()	встановлення значення проекції перетворення
getLandmarkProjections()	отримання проекції координат на зображення

Дані, що зберігаються у цьому класі - локальні координати ключових точок та перетворення подібності, що переводить локальні координати у координати на зображенні, до якого відноситься ця форма.

3.1.11 Структура TrainingConfig

Опис класу розташовується у файлі TrainingConfig.h. В даній структурі зберігаються значення, що конфігурують процес навчання та містять загальні допоміжні дані. Більш детально ці значення описані нижче.

3.1.12 Структура TrainingSample

Опис класу розташовується у файлі TrainingSample.h, реалізація – у файлі TrainingSample.cpp.

Дані структури описані в таблиці 3.11.

Таблиця 3.11 – Опис структури TrainingSample

Назва	Опис
image	посилання на зображення
truthShapePtr	істинна форма
guessShapePtr	форма-здогадка
transform	трансформація в локальних координатах
residualTransform	залишкова трансформація
descriptor	представлення ділянки зображення
weight	вага зразку для класифікації
score	значення класифікації

3.1.13 Файли Utilities.h та Utilities.cpp

В цих файлах реалізовані допоміжні функції, описані в таблиці 3.12.

Таблиця 3.12 – Опис функцій у файлах Utilities

Назва	Опис
Функції:	
showShapeOnImage()	малює форму на зображення
expandData()	розширення тренувальної вибірки
add()	векторна операція додавання
sub()	векторна операція віднімання
mul()	векторна операція скалярного добутку

3.1.14 Файл TrainingMain.cpp

Цей файл містить головну виконувану функцію, в якій реалізовано процес підготовки навчання та його виконання. В цій функції відбувається завантаження тренувальних даних, попередня обробка зображень, генерація та розширення даних для навчання і безпосередньо визивається тренування кожного регресора і додавання його до каскаду. Функції, реалізовані у файлі, описані в таблиці 3.13.

Таблиця 3.13 – Опис функцій у файлі TrainingMain.cpp

Назва	Опис
Функції:	
loadData()	завантаження тренувальних даних
calculateMeanShape()	визначення форми для порівняння
main()	основна функція реалізації тренування

3.1.15 Інші файли

У файлі TrackingMain.cpp реалізований функціонал для демонстрації можливостей навченого каскаду – відстежування обличчя.

У файлі DetectionMain.cpp реалізований функціонал для демонстрації можливостей навченого каскаду – виявлення обличчя.

У файлі TestingMain.cpp реалізований функціонал для тестування навченого каскаду.

3.2 Опис особливостей реалізованих алгоритмів

3.2.1 Підготовка даних до навчання регресору

Розглянемо алгоритм підготовки даних до навчання регресора покроково:

Крок 1. Генерація навчальних даних.

Генеруються дані у форматі «істинна форма, форма-здогадка» та формують множину пар.

Крок 2. Генерація точок-ознак.

Для даного регресора генерується множину випадкових точок-ознак, значення пікселів яких формуватимуть дескриптор.

Крок 3. Розрахування дескрипторів.

Далі для кожної пари з навчальних даних вираховуються дескриптори із форми-здогадки та відповідного цій парі зображення.

Крок 4. Перевід у іншу систему координат.

Після цього істинна форма кожної пари переводиться у систему координат форми-здогадки.

Крок 5. Розрахування векторів-здвигів.

У новій системі координат вираховуються вектори-здвиги, що мають бути додані до локальних точок форми-здогадки, щоб отримати істинну форму.

Крок 6. Стиснення системи координат трансформації.

Так як модель, що використовувалася при реалізації системи містить 68 точок, то одна трансформація кодується 136 дійсними числами. Очевидно, що кількість ступенів свободи нормальної форми набагато менша за це число, адже відносне розташування багатьох точок моделі є постійним. Тому для прискорення роботи системи виконується стиснення системи координат трансформації до 40 чисел. Це робиться за допомогою алгоритму сингулярного розкладу матриці, або SVD, який реалізований у бібліотеці OpenCV. Після цього всі трансформації кодуються вже 40 числами.

Крок 7. Навчання регресора.

Далі настає фаза безпосереднього навчання регресора, мета якого за даними дескрипторами вирахувати шукану трансформацію.

3.2.2 Процес генерації дерев та правил

Процес генерації дерев та правил є досить складним та має багато умов. Розглянемо його детальніше:

Крок 1. Визначення предмету оптимізації.

Перед генерацією випадково визначається, що саме буде оптимізувати згенероване дерево – помилку класифікації чи регресії.

Крок 2. Генерація дерева ансамблю.

На другому кроці послідовно генерується кожне дерево ансамблю. Після чого кожен його лист має вектор-трансформацію, що віднімається від залишкової трансформації всіх зразків, що потрапили до цього листа, та має значення класифікації, яке додається до всіх зразків, що потрапили до цього листа.

Крок 3. Вибір правила для мінімізації.

Під час генерації дерева рекурсивно генеруються його вершини. На кожному такому кроці перебирається велика кількість випадкових потенційних правил і серед них обирається таке, що мінімізує значення функції-помилки.

Крок 4. Оптимізація.

Варіант 1. Оптимізація регресії.

Якщо оптимізується регресія, то обирається таке правило, що після нього сума квадратів довжин векторів залишкових трансформацій є мінімальною. А значення трансформації в листі вираховується як середнє арифметичне залишкових трансформацій усіх зразків, що потрапили до цієї вершини.

Варіант 2. Оптимізація класифікації.

Якщо ж оптимізується класифікація, то обирається таке правило, що як найкраще розділяє множини позитивних та негативних зразків. Значення класифікації в листі вираховується по формулі (3.1):

$$R = \frac{\log(\sum(S_p)) - \log(\sum(S_n))}{2}, \quad (3.1)$$

де R – значення класифікації в листі,

S_p – ваги позитивних зразків, що потрапили до цього листа,

S_n – ваги негативних зразків.

Крок 5. Розрахунок ваги зразка.

Вага кожного зразка перераховується після генерації кожного дерева за формулою (3.2):

$$w_i = \begin{cases} e^{f_i}, & \text{якщо } s_i \in S_p; \\ e^{-f_i}, & \text{якщо } s_i \in S_n, \end{cases} \quad (3.2)$$

де w_i – вага i -го зразка,

f_i – поточне значення класифікація для i -го зразка (0 від початку),

s_i – i -й зразок,

S_p – множина позитивних зразків, що потрапили до цього листа,

S_n – множина негативних зразків.

Крок 6. Нормалізація ваг.

Після перерахунку всі ваги нормалізуються так, щоб сума їх значень дорівнювала 1.

Крок 7. Визначення порогової величини.

Також для кожного дерева визначається порогова величина, при досягненні якої можна відкидати зразок як точно негативний. Значення цієї величини підбирається таким, щоб через неї відкидався дуже малий відсоток позитивних зразків, як помилково негативних (порядку 1-20 зразків на дерево при розмірі тренувальної вибірки у 300 тисяч зразків).

3.3 Конфігурація навчання системи

Структура з налаштованими макропараметрами навчання системи зображена на рисунку 3.2:

```
struct TrainingConfig {
    static const int cascadeSize = 8;
    static const int treeDepth = 4;
    int ruleCount[cascadeSize] = {256, 240, 224, 208, 192, 176, 160, 256};
    int featurePointCount[cascadeSize] = {150, 250, 350, 500, 550, 550, 550, 550};
    int ensembleSize[cascadeSize] = {150, 200, 250, 350, 450, 500, 500, 500};
    double nu[cascadeSize] = {0.15, 0.15, 0.15, 0.15, 0.15, 0.1, 0.1, 0.1};
    double classificationProbability[cascadeSize] = {0.9, 0.8, 0.7, 0.6, 0.4, 0.2, 0.1, 0.1};
    const int minNegativeDataSize[cascadeSize] = {400000, 150000, 100000, 90000, 50000, 25000, 15000};
    double bestPositiveNegativeToIgnore = 0.001;

    int regressorIndex = 0;

    double kthFactorInitial = 0.00022;
    double kthFactorFinal = 0.00003;
    int unitCount = 40;
    int normalizationStep = 300;

    int mirrorIndices[68] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
                             26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 27, 28, 29, 30, 35, 34, 33,
                             32, 31, 45, 44, 43, 42, 47, 46, 39, 38, 37, 36, 41, 40, 54, 53, 52,
                             51, 50, 49, 48, 59, 58, 57, 56, 55, 64, 63, 62, 61, 60, 67, 66, 65};

    ShapePtr meanShapePtr;
    std::default_random_engine rng;
};
```

Рисунок 3.2 – Конфігурація системи

Більшість макропараметрів підбирається інтуїтивно або оптимізується вручну за рахунок не можливості фактичної оптимізації.

Розглянемо найважливіші параметри:

- cascadeSize – розмір каскаду;
- treeDepth – глибина одного дерева (кількість правил, тобто дерево глибини 4 має 16 листів);
- ruleCount – кількість правил, з яких обирається найкраще при генерації одній внутрішньої вершини;
- featurePointCount – кількість точок-ознак, які генеруються для заданого регресора;
- ensembleSize – розмір ансамблю заданого регресора;
- pu – коефіцієнт з яким приймається трансформація із листа дерева.

Інша популярна назва – learning rate;

- classificationProbability – ймовірність того, що дерево буде оптимізувати класифікацію, а не регресію;
- unitCount – розмірність простору до якого стискаються початкові трансформації.

3.4 Висновки за розділом 3

В цьому розділі було описано і обґрунтовано вибір інструментів для реалізації технології. Були вказані та описані класи, структури та найважливіші функції, що описують дані, формалізують логіку коду та реалізують алгоритми машинного навчання. Також в цьому розділі були по крокам розписані алгоритми розв'язку поставлених задач. Наведена конфігурація системи з макропараметрами, які не піддаються автоматичній оптимізації.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програмний продукт призначений для розв’язання практичних завдань ідентифікації об’єктів на зображеннях.

Програмний продукт ідентифікації об’єктів на зображеннях забезпечує виконання функцій:

- обробка навчальних вибірок даних, що складаються з зображень;
- навчання моделі для виявлення об’єктів на основі заданої вибірки зображень;
- обробка тестових вибірок даних та ідентифікація об’єктів на зображеннях;
- перевірка коректності вхідних даних;
- відображення результатів роботи програми ідентифікації об’єктів на зображеннях.

Для експлуатації програмного продукту ідентифікації об’єктів на зображеннях необхідні такі програмно-технічні засоби:

- процесор 1.6 гігагерц (ГГц) і вище;
- 64-розрядний процесор;
- RAM гігабайти (ГБ) ОЗУ;
- жорсткий диск 900 мегабайт (МБ) вільного місця;
- для апаратного прискорення графіки необхідний екран з роздільною здатністю 1366 x 768 пікселів.

Функціональні характеристики розробленої програмної системи ідентифікації об’єктів на зображеннях – відповідають технічному завданню, поданому у додатку А. Опис програми ідентифікації об’єктів на зображеннях та текст програми ідентифікації об’єктів на зображеннях наведені в додатках Б та В.

4.2 Характеристики програми

Програма ідентифікації об'єктів на зображеннях розроблена за допомогою мови програмування C++. Для забезпечення можливостей роботи з зображеннями, зокрема їх завантаження, обрізання, масштабування, зміни кольору тощо, використано відкриту бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення OpenCV.

Час роботи програми ідентифікації об'єктів на зображеннях залежить від обсягу навчальної вибірки та конфігурації комп'ютера.

Програму ідентифікації об'єктів на зображеннях реалізовано за допомогою об'єктно-орієнтованого підходу, створено відповідні класи та структури. У процесі розробки вони були розподілені по файлам по відповідним каталогам та файлам.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для завантаження програми ідентифікації об'єктів на зображеннях необхідно скопіювати файли програми з носія на комп'ютер у відповідну директорію.

Для початку роботи з програмою ідентифікації об'єктів на зображеннях необхідно скорегувати файл з налаштованими макропараметрами навчання системи (рис. 3.2).

Основні налаштовувані параметри такі:

- cascadeSize – розмір каскаду;
- treeDepth – глибина одного дерева (кількість правил, тобто дерево глибини 4 має 16 листів);
- ruleCount – кількість правил, з яких обирається найкраще при генерації одній внутрішньої вершини;

- `featurePointCount` – кількість точок-ознак, які генеруються для заданого регресора;

- `ensembleSize` – розмір ансамблю заданого регресора;

- `pu` – коефіцієнт з яким приймається трансформація із листа дерева.

Інша популярна назва – `learning rate`;

- `classificationProbability` – ймовірність того, що дерево буде оптимізувати класифікацію, а не регресію;

- `unitCount` – розмірність простору до якого стискаються початкові трансформації.

Після введення налаштовуваних параметрів програма запускається з консолі.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програми ідентифікації об'єктів на зображеннях є набір зображень для навчання розпізнавальних моделей, а також множина зображень для виявлення об'єктів (наприклад, облич) на відповідних зображеннях.

Вихідними даними програми ідентифікації об'єктів на зображеннях є розпізнані об'єкти (наприклад, облич) на відповідних зображеннях.

4.3.3 Повідомлення

Розроблена програма ідентифікації об'єктів на зображеннях забезпечує стійке функціонування. У випадку виникнення виняткових ситуацій, наприклад, коли користувач задає вхідні дані, що не відповідають графічним зображенням, програма видає відповідні повідомлення.

4.4 Тестування та результати роботи з програмою ідентифікації об'єктів на зображеннях

Розроблена система вирішує задачі виявлення об'єктів заданого класу на зображеннях.

Розроблена система має наступні функціональні можливості:

- пошук та виявлення одного чи декількох облич заданого класу на кадрах;
- відстеження обличчя на послідовних кадрах.

Розроблена система здатна ефективно вирішувати описані задачі.

На рисунку 4.1 наведено діаграму послідовності тестування системи.

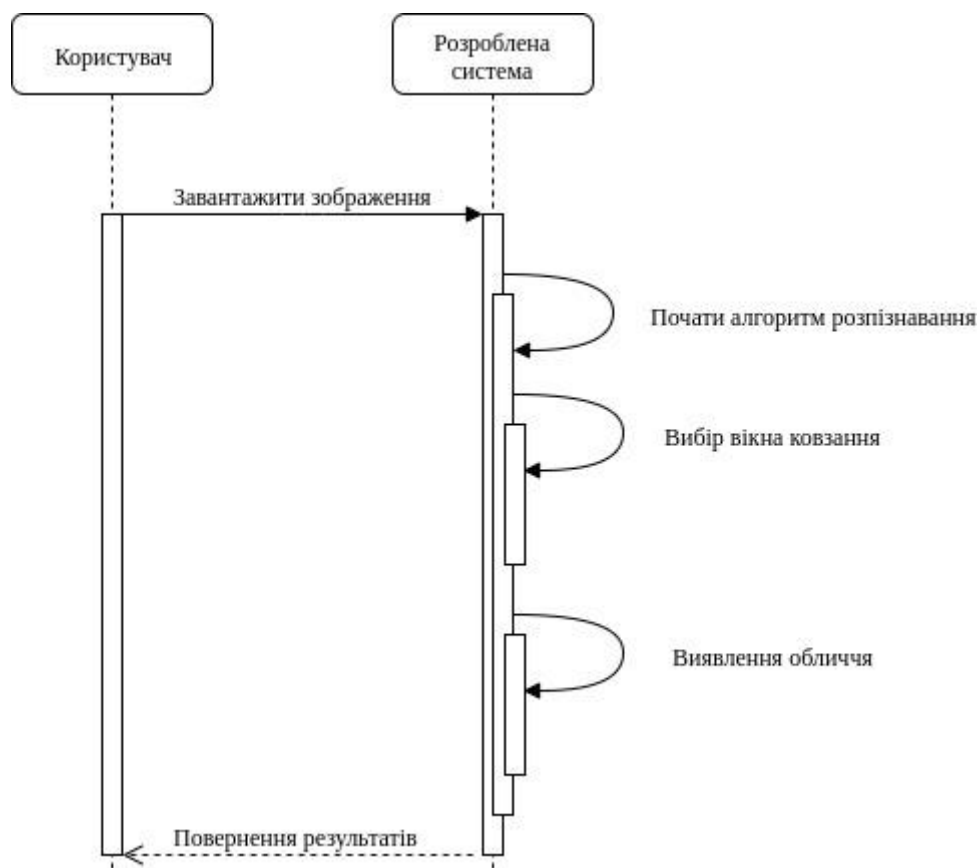


Рисунок 4.1 – Діаграма послідовності

Розглянемо приклад роботи системи до модифікування (рис. 4.2 та 4.3).

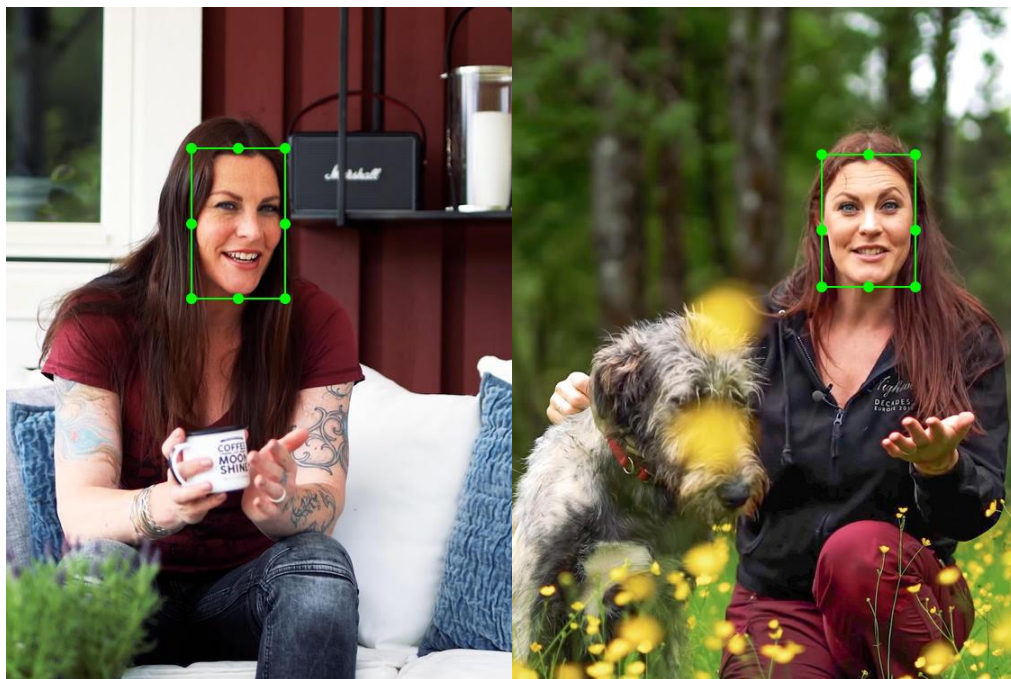


Рисунок 4.2 – Результат пошуку обличчя без нахилу на зображенні немодифікованим алгоритмом (знайдено)



Рисунок 4.3 – Результат пошуку обличчя з нахилом на зображенні немодифікованим алгоритмом (не знайдено)

Приклади результатів роботи розробленої модифікованої системи наведено на рис. 4.4 та 4.5.

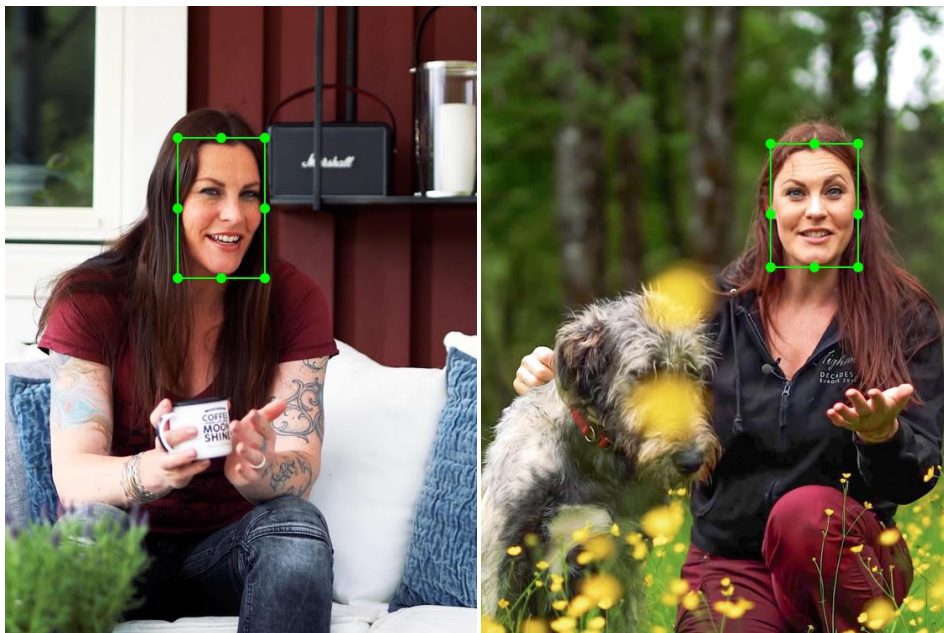


Рисунок 4.4 – Результат пошуку обличчя без нахилу на зображенні розробленим алгоритмом (знайдено)



Рисунок 4.5 – Результат пошуку обличчя з нахилом на зображенні розробленим алгоритмом (знайдено)

На основі проведеного експерименту можна побачити, що модифікований алгоритм знаходить обличчя з нахилом на кадрах, а початковий алгоритм – ні.

Виконаємо порівняльний аналіз функціональних можливостей розробленої програми ідентифікації об'єктів на зображеннях з відомими аналогами.

З відомих аналогів, що реалізують технологію, подібну розробленої у даній роботі, можна відзначити такі системи:

- OpenCV;
- Google Vision API;
- Dlib Library.

Порівняння функціональних можливостей існуючих аналогів систем для виявлення та відстеження облич людей і розробленої системи наведено в таблиці 4.1.

Таблиця 4.1 – Порівняльний аналіз програмних систем для виявлення та відстеження облич людей

Критерії порівняння	Програмні системи			
	OpenCV	Google Vision API	Dlib Library	Розроблена програма
Пошук облич	+	+	+	+
Відстеження обличчя	±	–	+	+
Виявлення облич з нахилом	–	±	–	+
Портативність	+	± (веб API)	+	+

На основі проведеного аналізу можна зробити висновок, що розроблена програма ідентифікації об'єктів на зображеннях, на відміну від існуючих аналогів, дозволяє виконувати пошук облич з нахилом та є портативною.

4.5 Висновки за розділом 4

Описано та досліджено функціональні можливості розробленої системи. Функціональні можливості розробленої програми були порівняні з іншими відомими системами, що працюють у сферах аналізу зображень облич за допомогою алгоритмів комп'ютерного зору. Для оцінки якості роботи системи було проведено автоматичне тестування, яке показало високу достовірність розпізнавання у розробленого алгоритму.

Водночас аналіз та тестування дозволили виявити ряд недоліків та проблемних умов роботи реалізованої технології. Були запропоновані ідеї та напрямки розвитку, що дозволять в майбутньому покращити якість роботи технології.

Розроблена в результаті виконання роботи система вирішує поставлені задачі та показує хороші результати.

Проте, аналіз результатів і оцінка точності роботи розробленої системи показують, що можливе покращення технології за рахунок усунення недоліків та оптимізації алгоритму роботи.

За результатами експериментального дослідження можна виділити ряд умов, при яких дана технологія показує погані результати:

- погане освітлення зображення (затемнені або засвітлені ділянки);
- великі кути повороту обличчя (не в площині зображення);
- часткове перекриття обличчя іншими об'єктами.

Серед ідей у покращенні роботи системи можна виділити наступні:

- вдосконалення алгоритму відстежування з метою зменшення негативного впливу тимчасового часткового або повного перекриття обличчя, що відстежується, іншими об'єктами;
- розширення тренувальної бази; це дозволить покращити якість навчання, поліпшити якість роботи в екстремальних умовах та зменшити ефект перенавчання.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес та методи ідентифікації об'єктів на зображеннях.

За результатами дослідження існуючих методів виявлення об'єктів заданого класу було сформульоване завдання дослідження: необхідно розробити технологію, що дозволяє вирішити задачу виявлення об'єктів на зображеннях. Було запропоновано підходи до її вирішення.

В роботі було розглянуто задачу виявлення облич людей як окремий випадок проблеми виявлення об'єктів заданого класу на зображеннях. Було розглянуто та описано модифікації, які можливо внести для вдосконалення існуючої технології пошуку та відстеження об'єктів. Реалізовано технологію виявлення об'єктів на основі методу Віюлі-Джонса з використанням характеристичних рис Хаара та з побудовою каскаду класифікаторів, навчених з допомогою методу AdaBoost на прикладі задачі виявлення облич людей як окремого випадку проблеми виявлення.

Для створення програмного забезпечення для ідентифікації об'єктів на зображеннях було обрано мову програмування C++, яка є досить зручною для розробки програм подібного роду. Також роботи з зображеннями, зокрема для завантаження, обрізання, масштабування, зміни кольору тощо, було вирішено підключити до проекту відкриту бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення OpenCV, яка має відкритий код.

Наукова новизна роботи полягає у тому, що розроблено удосконалений ідентифікації об'єктів на зображеннях, у якому, на відміну від існуючих, на етапі виявлення облич використовується модифікований алгоритм для ідентифікації об'єктів, обернених під довільним кутом, крім того, використовується вдосконалений процесу навчання системи шляхом

використання кросвалідації, що дозволяє підвищити точність ідентифікації об'єктів.

Практичне значення одержаних результатів полягає у тому, що розроблено програмне забезпечення для ідентифікації об'єктів на зображеннях, яке може бути використано у різних прикладних застосунках від зору роботів до розпізнавання образів.

Встановлено задачі та вимоги до розроблюваної технології. Правильне розуміння цих задач та вимог дозволило обрати алгоритми оптимізації та машинного навчання, що дозволять вирішити ці задачі. Ключовими алгоритмами є каскад із ансамблів дерев регресії та класифікації, та AdaBoost, завдяки якому навчатиметься безпосередньо сама класифікація.

Були описані та досліджені функціональні можливості розробленої системи. Ці функціональні можливості були порівняні з іншими відомими системами, що працюють у сферах аналізу зображень облич за допомогою алгоритмів комп'ютерного зору. Для оцінки якості роботи системи було проведено автоматичне тестування, яке показало високу достовірність розпізнавання у розробленого алгоритму. Аналіз та тестування дозволили виявити ряд недоліків та проблемних умов роботи реалізованої технології. Були запропоновані ідеї та напрямки розвитку, що дозволять в майбутньому покращити якість роботи технології.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. David A. Forsyth. Computer Vision: A Modern Approach [Text] / David A. Forsyth, Jean Ponce. – New Jersey: Prentice Hall, 2002. – 693 p.
2. Dan Casas. Real-Time Face Tracking Methods [Text] / Dan Casas. – Bellaterra: Universitat Autònoma de Barcelona, 2009. – 74 p.
3. Object Tracking using OpenCV [Electronic resource]. – Access mode: <https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>.
4. Andrew Burton. Thinking in Perspective: Critical Essays in the Study of Thought Processes [Text] / Andrew Burton, John Radford. – London: Methuen, 1978. – 232 p.
5. Jianbo Shi. Good features to track [Text] / Jianbo Shi, Carlo Tomasi // IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94), 1994. – P. 593-600.
6. Ginneken B. Active Shape Model Segmentation with Optimal Features [Text] / Ginneken Bram // IEEE Transactions on Medical Imaging, 2002. – Vol.21, No.8. – P. 924-933.
7. Cootes T. Active appearance models [Text] / T. Cootes, G. Edwards, C. Taylor // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001. – Vol. 23, No.6. – P. 681–685.
8. Breiman L. Random forests [Text] / L. Breiman // Machine learning, 2001. – Volume 45, Issue 1. – P. 5–32.
9. Документація фреймворку OpenCV [Electronic resource]. – Access mode: <http://docs.opencv.org/>.
10. Viola P. Rapid object detection using a boosted cascade of simple features [Text] / P. Viola, M. Jones // Proc. of CVPR, 2001. – Vol.1. – P. 511-518.
11. Yevhen Zadorozhnii. Development and implementation of human face alignment and tracking in video streams [Text] / Yevhen Zadorozhnii, Yevhenii Tverdokhlib, Tetiana Fedoronchak and Natalia Myronova // IEEE Second

International Conference on Data Stream Mining & Processing (DSMP), 2018. – P. 574-579.

12. Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection [Text] / R. Kohavi // Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995. – Vol.2. – P. 1137–1143.

13. Gavin C. Cawley. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation [Text] / Cawley, Gavin C., Talbot, Nicola L. C. // Journal of Machine Learning Research, 2010. – Vol.11. – P. 2079–2107.

14. Bjarne Stroustrup. The C++ Programming Language (Third ed.) [Text] / Stroustrup, Bjarne. – Addison-Wesley, 1997. – 912 p.

15. Kari Pulli. Realtime Computer Vision with OpenCV [Text] / Kari Pulli, Baksheev Anatoly, Kirill Korniyakov, Victor Eruhimov // Queue, 2012. – P. 40–56.

16. FDDB: Face Detection Data Set and Benchmark [Electronic resource]. – Access mode: <http://vis-www.cs.umass.edu/fddb>.

17. Christopher D. Brown. Receiver operating characteristic curves and related decision measures: a tutorial [Text] / Christopher D. Brown, Herbert T. Davis // Chemometrics and Intelligent Laboratory Systems, 2006. – P. 24–38.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення ідентифікації об'єктів на зображеннях може застосовуватися для вирішення задач розпізнавання образів.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація методів ідентифікації об'єктів на зображеннях», затверджене наказом Національного університету «Запорізька політехніка» № 435 від 5 листопада 2021 р.

A.2 Призначення розробки

Програмний продукт ідентифікації об'єктів на зображеннях призначений для навчання на великих об'ємах інформації (зображень) з подальшою здатністю виявлення облич на зображеннях. Розроблена програма здатна ідентифікувати образи з певним нахилом на зображенні.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати виконання наступних функцій:

- обробка навчальних вибірок даних, що складаються з зображень;
- навчання моделі для виявлення об'єктів на основі заданої вибірки зображень;
- обробка тестових вибірок даних та ідентифікація об'єктів на зображеннях;
- перевірка коректності вхідних даних;

- відображення результатів роботи програми ідентифікації об'єктів на зображеннях.

A.3.2 Вимоги до інтерфейсу програми

Комп'ютерна програма ідентифікації об'єктів на зображеннях повинна забезпечувати можливість розпізнавання образів шляхом обробки графічних файлів. Взаємодія користувача з програмою ідентифікації об'єктів на зображеннях можлива за допомогою інтерфейсу командного рядка.

A.3.3 Вимоги до надійності

Програма ідентифікації об'єктів на зображеннях повинна обробляти виняткові ситуації.

A.3.4 Умови експлуатації

Експлуатація програми ідентифікації об'єктів на зображеннях повинна здійснюватися відповідно до документації на розроблене програмне забезпечення. Програма ідентифікації об'єктів на зображеннях може бути записана як на диску.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного продукту ідентифікації об'єктів на зображеннях необхідні такі програмно-технічні засоби:

- процесор 1.6 гігагерц (ГГц) і вище;
- 64-розрядний процесор;
- RAM гігабайти (ГБ) ОЗУ;
- жорсткий диск 900 мегабайт (МБ) вільного місця;

– для апаратного прискорення графіки необхідний екран з роздільною здатністю 1366 x 768 пікселів.

A.3.6 Вимоги до маркування і пакування

Програма ідентифікації об'єктів на зображеннях може поставлятися на різних носіях. На упаковці повинна бути назва програми «Програма ідентифікації об'єктів на зображеннях».

A.4 Вхідні дані до роботи

Вхідними даними до програми ідентифікації об'єктів на зображеннях є набір зображень для навчання розпізнавальних моделей, а також множина зображень для виявлення об'єктів (наприклад, облич) на відповідних зображеннях.

Вихідними даними програми ідентифікації об'єктів на зображеннях є розпізнані об'єкти (наприклад, облич) на відповідних зображеннях.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Програма ідентифікації об'єктів на зображеннях розроблена за допомогою середовища розробки Clion та мови програмування C++.

Б.2 Функціональне призначення

Програмний продукт призначений для розв'язання практичних завдань ідентифікації об'єктів на зображеннях.

Програма ідентифікації об'єктів на зображеннях забезпечує виконання наступних функцій:

- обробка навчальних вибірок даних, що складаються з зображень;
- навчання моделі для виявлення об'єктів на основі заданої вибірки зображень;
- обробка тестових вибірок даних та ідентифікація об'єктів на зображеннях;
- перевірка коректності вхідних даних;
- відображення результатів роботи програми ідентифікації об'єктів на зображеннях.

Б.3 Опис логічної структури

Програма ідентифікації об'єктів на зображеннях представляє застосунок, що складається з різних програмних модулів.

Для реалізації програми ідентифікації об'єктів на зображеннях було створено класи та структури. У процесі розробки вони були розподілені по файлам по відповідним каталогам та файлам.

Б.4 Використані технічні засоби

Для експлуатації програмного продукту ідентифікації об'єктів на зображеннях необхідні такі програмно-технічні засоби:

- процесор 1.6 гігагерц (ГГц) і вище;
- 64-розрядний процесор;
- RAM гігабайти (ГБ) ОЗУ;
- жорсткий диск 900 мегабайт (МБ) вільного місця;
- для апаратного прискорення графіки необхідний екран з роздільною здатністю 1366 x 768 пікселів.

Б.5 Виклик і завантаження

Для завантаження програми ідентифікації об'єктів на зображеннях необхідно скопіювати файли програми з носія на комп'ютер у відповідну директорію.

Для початку роботи з програмою необхідно ідентифікації об'єктів на зображеннях скорегувати файл з налаштованими макропараметрами навчання системи.

Основні налаштовувані параметри такі:

- cascadeSize – розмір каскаду;
- treeDepth – глибина одного дерева (кількість правил, тобто дерево глибини 4 має 16 листів);
- ruleCount – кількість правил, з яких обирається найкраще при генерації одній внутрішньої вершини;
- featurePointCount – кількість точок-ознак, які генеруються для заданого регресора;
- ensembleSize – розмір ансамблю заданого регресора;
- nu – коефіцієнт з яким приймається трансформація із листа дерева.

Інша популярна назва – learning rate;

– `classificationProbability` – ймовірність того, що дерево буде оптимізувати класифікацію, а не регресію;

– `unitCount` – розмірність простору до якого стискаються початкові трансформації.

Після введення налаштовуваних параметрів програма запускається з консолі.

Б.6 Вхідні та вихідні дані

Вхідними даними до програми ідентифікації об'єктів на зображеннях є набір зображень для навчання розпізнавальних моделей, а також множина зображень для виявлення об'єктів (наприклад, облич) на відповідних зображеннях.

Вихідними даними програми ідентифікації об'єктів на зображеннях є розпізнані об'єкти (наприклад, облич) на відповідних зображеннях.

ДОДАТОК В
Текст програми

```

/// Detector.cpp

#include "Detector.h"
#include "Shape.h"

bool Detector::validate(const cv::Mat& image, double& score,
Shape& shape, int& n) {
    shape.setLocalCoordinates(cascade[0].getMeanShapePtr() -
>getLocalCoordinates());
    score = 0;
    Shape::Point scaleRotate;
    Shape::Point transalte;
    for (int t = 0; t < currentRegressorIndex; ++t) {
        auto& regressor = cascade[t];
        bool result = regressor.process(image, shape, score,
true);
        if (!result) {
            return false;
        }
    }
    for (int k = 0; k <= currentTreeIndex; k++) {
        Regressor& regressor = cascade[currentRegressorIndex];
        bool result = regressor.process(image, shape, score,
true);
        if (!result) {
            return false;
        }
    }
    return true;
}

void Detector::detectSingleScale(cv::Mat image, int step, int
windowSize, std::vector<cv::Rect>& rects,
                                std::vector<double>& scores,
std::vector<Shape>& shapes, DetectionStatistic& statistic) {
    const int xMax = image.cols - windowHeight / 2;
    const int yMax = image.rows - windowHeight / 2;
    const int xStep = step;
    const int yStep = step;

    rects.clear();
    scores.clear();
    shapes.clear();

    Shape shape;
    for (int y = windowHeight / 2; y <= yMax; y += yStep) {
        for (int x = windowHeight / 2; x <= xMax; x += xStep) {
            cv::Rect roi(x - windowHeight / 2, y - windowHeight / 2,
windowSize, windowHeight);
            double score;
            shape.setProjectionTranslate(Shape::Point(x, y));

shape.setProjectionScaleRotate(Shape::Point(windowSize, 0));
            int reject_length;
            bool is_face = validate(image, score, shape,
reject_length);
            if (is_face) {

```

```

        rects.push_back(roi);
        scores.push_back(score);
        shapes.push_back(shape);
        statistic.face_patch_n++;
    } else {
        statistic.nonface_patch_n++;
        statistic.cart_gothrough_n += reject_length;
    }
}

}

}

void Detector::detectMultiScale(cv::Mat image,
std::vector<cv::Rect>& rects, std::vector<double>& scores,
                                std::vector<Shape>& shapes,
DetectionStatistic& statistic) {
    int width = image.cols;
    int height = image.rows;
    double windowSize = std::max(image.rows, image.cols) * 0.07;
    const double factor = 1.2;
    double step = windowSize * 0.2;

    rects.clear();
    scores.clear();
    shapes.clear();

    while ((width >= windowSize) && (height >= windowSize)) {
        std::vector<cv::Rect> rects_;
        std::vector<double> scores_;
        std::vector<Shape> shapes_;
        detectSingleScale(image, step, windowSize, rects_,
scores_, shapes_, statistic);
        rects.insert(rects.end(), rects_.begin(), rects_.end());
        scores.insert(scores.end(), scores_.begin(),
scores_.end());
        shapes.insert(shapes.end(), shapes_.begin(),
shapes_.end());

        windowSize *= factor;
        step *= factor;;
    }
    statistic.patch_n = statistic.face_patch_n +
statistic.nonface_patch_n;
    statistic.average_cart_n = statistic.cart_gothrough_n /
statistic.nonface_patch_n;
}

void Detector::detectMultiScale1(cv::Mat image,
std::vector<cv::Rect>& rects, std::vector<double>& scores,
                                std::vector<Shape>& shapes,
DetectionStatistic& statistic) {}

static vector<int> nms(const vector<cv::Rect>& rects, const
vector<double>& scores, double overlap) {
    const int n = static_cast<int>(rects.size());
    vector<double> areas(n);

    typedef std::multimap<double, int> ScoreMapper;

```

```

ScoreMapper map;
for (int i = 0; i < n; i++) {
    map.insert(ScoreMapper::value_type(scores[i], i));
    areas[i] = rects[i].width * rects[i].height;
}

int picked_n = 0;
vector<int> picked(n);
while (map.size() != 0) {
    int last = map.rbegin()->second;
    picked[picked_n] = last;
    picked_n++;
    for (ScoreMapper::iterator it = map.begin(); it !=
map.end();) {
        int idx = it->second;
        double x1 = std::max(rects[idx].x, rects[last].x);
        double y1 = std::max(rects[idx].y, rects[last].y);
        double x2 = std::min(rects[idx].x + rects[idx].width,
rects[last].x + rects[last].width);
        double y2 = std::min(rects[idx].y +
rects[idx].height, rects[last].y + rects[last].height);
        double w = std::max(0., x2 - x1);
        double h = std::max(0., y2 - y1);
        double ov = w * h / (std::min(areas[idx],
areas[last]) - 0 * w * h);
        if (ov > overlap) {
            ScoreMapper::iterator tmp = it;
            tmp++;
            map.erase(it);
            it = tmp;
        } else {
            it++;
        }
    }
}

picked.resize(picked_n);
return picked;
}

void Detector::detect(cv::Mat image, std::vector<Shape>& shapes,
std::vector<cv::Rect>& rects,
                        std::vector<double>& scores) {
    std::vector<cv::Rect> rects_;
    std::vector<double> scores_;
    std::vector<Shape> shapes_;
    DetectionStatistic statistic;
    int detectorIndex = 0;
    currentRegressorIndex = static_cast<int>(cascade.size());
    if (detectorIndex == 0) {
        detectMultiScale(image, rects_, scores_, shapes_,
statistic);
    } else {
        detectMultiScale1(image, rects_, scores_, shapes_,
statistic);
    }
    const double overlap = 0.5;
    std::vector<int> picked;

```

```

    bool doNms = true;
    if (doNms) {
        picked = nms(rects_, scores_, overlap);
    } else {
        const int n = static_cast<int>(rects_.size());
        picked.resize(n);
        for (int i = 0; i < n; i++) {
            picked[i] = i;
        }
    }
    if (picked.empty()) {
        return;
    }
    const int iterationCount = 3;
    for (int i = static_cast<int>(picked.size()) - 1; i >= 0; i--)
    {
        const int index = picked[i];
        bool bad = false;
        for (int iteration = 0; iteration < iterationCount;
++iteration) {
            int tmp = 0;
            if (!validate(image, scores_[index], shapes_[index],
tmp)) {
                bad = true;
                break;
            }
        }
        if (bad) {
            picked.erase(picked.begin() + i);
        }
    }
    std::sort(picked.begin(), picked.end(), [&](int x, int y) {
return scores_[x] > scores_[y]; });
    for (int i = 0; i < picked.size(); ++i) {
        const int index = picked[i];
        rects.push_back(rects_[index]);
        shapes.push_back(shapes_[index]);
        scores.push_back(scores_[index]);
    }
}

```

```

/// Detector.h

```

```

#ifndef Detector_h
#define Detector_h
#include "Regressor.h"
#include "Utilities.h"
#include "Shape.h"

struct DetectionStatistic {
public:
    DetectionStatistic()
        : patch_n(0), face_patch_n(0), nonface_patch_n(0),
cart_gothrough_n(0.) {
    }
}

```

```

        int patch_n;
        int face_patch_n;
        int nonface_patch_n;
        double cart_gothrough_n;
        double average_cart_n; // nonface patch reject length
    };

    class Detector {
    public:
        void detect(cv::Mat image, std::vector<Shape>& shapes,
std::vector<cv::Rect>& rects, std::vector<double>& scores);

        void setCascade(const Cascade& value) { cascade = value; }

        const Cascade& getCascade() const { return cascade; }

        void setRegressorIndex(int value) { currentRegressorIndex =
value; }

        int getRegressorIndex() const { return currentRegressorIndex;
}

    protected:
        void detectMultiScale(cv::Mat image, std::vector<cv::Rect>&
rects, std::vector<double>& scores,
                                std::vector<Shape>& shapes,
DetectionStatistic& statistic);
        void detectMultiScale1(cv::Mat image, std::vector<cv::Rect>&
rects, std::vector<double>& scores,
                                std::vector<Shape>& shapes,
DetectionStatistic& statistic);
        void detectSingleScale(cv::Mat image, int step, int win_size,
std::vector<cv::Rect>& rects,
                                std::vector<double>& scores,
std::vector<Shape>& shapes, DetectionStatistic& statistic);
        bool validate(const cv::Mat& image, double& score, Shape&
shape, int& n);

        Cascade cascade;
        int currentRegressorIndex = 0;
        int currentTreeIndex = -1;
    };

#endif /* Detector_h */

/// TrainingSample.h
#ifdef TrainingSample_h
#define TrainingSample_h
#include <opencv2/core/core.hpp>
#include "Shape.h"
#include <vector>

struct TrainingSample {
    cv::Mat image;
    ShapePtr truthShapePtr;
    ShapePtr guessShapePtr;
    std::vector<float> transform;

```

```

        std::vector<float> residualTransform;
        std::vector<int> descriptor;
        double weight;
        double score = 0;
    };

#endif /* TrainingSample_h */

/// TrainingConfig.h
#ifdef TrainingConfig_h
#define TrainingConfig_h
#include "Shape.h"
#include <random>

struct TrainingConfig {
    static const int cascadeSize = 7;
    static const int treeDepth = 4;
    int ruleCount[cascadeSize] = {256, 240, 224, 208, 192, 176,
160};
    int featurePointCount[cascadeSize] = {250, 350, 450, 500, 550,
550, 550};
    int ensembleSize[cascadeSize] = {150, 200, 250, 350, 450, 500,
500};
    double nu[cascadeSize] = {0.15, 0.15, 0.15, 0.15, 0.15, 0.1,
0.1};
    double classificationProbability[cascadeSize] = {0.9, 0.8,
0.7, 0.6, 0.4, 0.2, 0.1};
    const int minNegativeDataSize[cascadeSize] = {400000, 150000,
100000, 90000, 50000, 30000, 20000};
    double bestPositiveNegativeToIgnore = 0.001;

    int regressorIndex = 0;

    double kthFactorInitial = 0.00022;
    double kthFactorFinal = 0.00003;
    int unitCount = 40;
    int normalizationStep = 50;

    int mirrorIndices[68] = {16, 15, 14, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1, 0,
26, 25, 24, 23, 22, 21, 20, 19,
18, 17, 27, 28, 29, 30, 35, 34, 33,
32, 31, 45, 44, 43, 42, 47, 46,
39, 38, 37, 36, 41, 40, 54, 53, 52,
51, 50, 49, 48, 59, 58, 57, 56,
55, 64, 63, 62, 61, 60, 67, 66, 65};
    ShapePtr meanShapePtr;
    std::default_random_engine rng;
};

#endif /* TrainingConfig_h */

/// Shape.h
#ifdef Shape_h
#define Shape_h
#include <vector>

```

```

#include <memory>
#include <iostream>
#include "Geometry.h"

class Shape;
template<class T>
using vector = std::vector<T>;
using ShapePtr = std::shared_ptr<Shape>;

class Shape {
public:
    using Point = Point<float>;

    Shape();
    Shape(int landmarkCount);

    float getAngle() const;

    int getLandamrkCount() const { return
static_cast<int>(localCoordinates.size()); }

    const vector<Point>& getLocalCoordinates() const { return
localCoordinates; }
    void setLocalCoordinates(const vector<Point>& value) {
localCoordinates = value; }

    vector<float> getActionUnits() const {
        vector<float> result;
        for (auto landmark : localCoordinates) {
            result.push_back(landmark.x);
            result.push_back(landmark.y);
        }
        return result;
    }

    void setActionUnits(const vector<float>& data) {
        int landmarkCount = static_cast<int>(data.size() / 2);
        localCoordinates.resize(landmarkCount);
        for (int i = 0; i < landmarkCount; ++i) {
            localCoordinates[i].x = data[i * 2 + 0];
            localCoordinates[i].y = data[i * 2 + 1];
        }
    }

    Point getProjectionTranslate() const { return translate; }
    void setProjectionTranslate(Point value) { translate = value; }

    float getProjectionScale() const { return abs(scaleRotate); }
    void setProjectionScale(float value) { scaleRotate =
scaleRotate * (value / abs(scaleRotate)); }

    void setAngle(float value) {
        scaleRotate = Point(std::cos(value), std::sin(value)) *
getProjectionScale();
    }
}

```

```

        Point getProjectionScaleRotate() const { return scaleRotate;
    }

    void setProjectionScaleRotate(Point value) { scaleRotate =
value; }

    vector<Point> getLandmarkProjections() const;

    friend std::ostream& operator<<(std::ostream& stream, const
Shape& shape);
    friend std::istream& operator>>(std::istream& stream, Shape&
shape);
protected:
    vector<Point> localCoordinates;
    Point translate;
    Point scaleRotate = Point(1, 0);};
#endif /* Shape_h */

```

```

/// Shape.cpp
#include "Shape.h"

Shape::Shape() : scaleRotate(1, 0), translate(0, 0) {}
Shape::Shape(int landmarkCount) : Shape() {
    localCoordinates.resize(landmarkCount);
}

vector<Shape::Point> Shape::getLandmarkProjections() const {
    int landamrkCount = getLandamrkCount();
    vector<Shape::Point> result(landamrkCount);
    for (int i = 0; i < landamrkCount; ++i) {
        result[i] = localCoordinates[i] * scaleRotate +
translate;
    }
    return result;
}

float Shape::getAngle() const {
    return std::atan2(scaleRotate.y, scaleRotate.x);
}

std::ostream& operator<<(std::ostream& stream, const Shape&
shape) {
    size_t landmarkCount = shape.localCoordinates.size();
    stream.write((const char*)&landmarkCount,
sizeof(landmarkCount));
    for (size_t i = 0; i < shape.localCoordinates.size(); ++i) {
        stream.write((const char*)&shape.localCoordinates[i].x,
sizeof(float));
        stream.write((const char*)&shape.localCoordinates[i].y,
sizeof(float));
    }
    stream.write((const char*)&shape.scaleRotate.x,
sizeof(float));
    stream.write((const char*)&shape.scaleRotate.y,
sizeof(float));
    stream.write((const char*)&shape.translate.x, sizeof(float));
    stream.write((const char*)&shape.translate.y, sizeof(float));
    return stream;
}

```

```

    }

    std::istream& operator>>(std::istream& stream, Shape& shape) {
        size_t landmarkCount;
        stream.read((char*)&landmarkCount, sizeof(landmarkCount));
        shape.localCoordinates.resize(landmarkCount);
        for (size_t i = 0; i < shape.localCoordinates.size(); ++i) {
            stream.read((char*)&shape.localCoordinates[i].x,
sizeof(float));
            stream.read((char*)&shape.localCoordinates[i].y,
sizeof(float));
        }
        stream.read((char*)&shape.scaleRotate.x, sizeof(float));
        stream.read((char*)&shape.scaleRotate.y, sizeof(float));
        stream.read((char*)&shape.translate.x, sizeof(float));
        stream.read((char*)&shape.translate.y, sizeof(float));
        return stream;
    }
}

```

```

/// TestingMain.cpp
#include <fstream>
#include <string>
#include <opencv2/opencv.hpp>
#include "Regressor.h"
#include "Shape.h"
#include "Utilities.h"
#include "Detector.h"

int main(int argc, const char * argv[]) {
    Cascade cascade;
    std::ifstream stream("classRegressorV2.bin",
std::ios::binary);
    stream >> cascade;
    Detector detector;
    detector.setCascade(cascade);

    int res[2][2] = {};
    int totalFaces = 0;
    int totalDetected = 0;
    int totalBadDetected = 0;

    cv::CascadeClassifier classifier;
    classifier.load("haarcascade_frontalface_alt2.xml");

    for (int idx = 1; idx <= 10; ++idx) {
        std::string idxString = std::to_string(idx);
        if (idxString.size() < 2) {
            idxString = "0" + idxString;
        }
        std::string path = "FDDB/FDDB-folds/FDDB-fold-" +
idxString + "-ellipseList.txt";
        std::ifstream fin(path);

        std::string imagePath;
        double passedTime = 0;
        while (fin >> imagePath) {

```

```

double mjRadius;
double mnRadius;
double angle;
double centerX, centerY;
imagePath = "FDDB/" + imagePath + ".jpg";
int faceCount;
fin >> faceCount;
cv::Mat image = cv::imread(imagePath);
cv::Mat gray;
cv::cvtColor(image, gray, CV_BGR2GRAY);
std::vector<Shape> shapes;
std::vector<cv::Rect> rects;
std::vector<double> scores;

double begin = clock();
detector.detect(gray, shapes, rects, scores);
double end = clock();
passedTime += (end - begin) / CLOCKS_PER_SEC;

rects.clear();
for (auto& shape : shapes) {
    auto projections =
shape.getLandmarkProjections();
    double minX = 1e10;
    double minY = 1e10;
    double maxX = -1e10;
    double maxY = -1e10;
    for (auto point : projections) {
        if (point.x < minX) {
            minX = point.x;
        }
        if (point.x > maxX) {
            maxX = point.x;
        }
        if (point.y < minY) {
            minY = point.y;
        }
        if (point.y > maxY) {
            maxY = point.y;
        }
    }
    rects.push_back(cv::Rect(minX, minY, maxX - minX,
maxY - minY));
}
vector<bool> used(shapes.size());
totalDetected += shapes.size();
totalFaces += faceCount;
while (faceCount--) {
    auto draw = image.clone();
    fin >> mjRadius >> mnRadius >> angle >> centerX
>> centerY;

    int t;
    fin >> t;
    Shape::Point center(centerX, centerY);
    double radius = (mjRadius + mnRadius) / 2;
    bool ok = false;
    int j = 0;
    for (auto shape : shapes) {

```

```

        auto translate =
shape.getProjectionTranslate();
        auto scale = shape.getProjectionScale() / 2;
        auto coef = scale / radius;
        auto dist = abs(translate - center);
        if (dist <= radius * 0.7 && coef >= 0.5 &&
coef <= 1.5) {
            ok = true;
            used[j] = true;
            break;
        }
        ++j;
    }
    res[0][ok]++;
}
for (int i = 0; i < shapes.size(); ++i) {
    if (!used[i]) {
        ++totalBadDetected;
    }
}
for (int i = 0; i < used.size(); ++i) {
    if (!used[i]) {
        ++totalBadDetected;
    }
}
double truePositive = res[0][1] * 100.0 / totalFaces;
double falseNegative = 100.0 - truePositive;
double falsePositive = totalBadDetected * 100.0 /
totalDetected;;
    DBN(totalFaces, truePositive, falseNegative,
falsePositive);
    DBN(passedTime / totalFaces);
}
}
return 0;
}

```

```

/// TrainingMain.cpp

```

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <fstream>
#include <memory>
#include <set>
#include <random>
#include <opencv2/opencv.hpp>
#include "TrainingConfig.h"
#include "RegressorTrainer.h"
#include "Shape.h"
#include "Utilities.h"

#include <glob.h>
std::string resourcesPath = "../large_face_landmark_dataset/";

std::vector<std::string> folders = {
    "trainset1",

```

```

        "trainset2",
        "trainset3",
        "trainset4",
    };

    void loadData(std::string path, std::vector<TrainingSample>&
data, TrainingConfig& config) {
        path += "*";
        glob_t glob_result;
        glob(path.c_str(), GLOB_TILDE, NULL, &glob_result);
        for (unsigned int i = 0; i < glob_result.gl_pathc; i++) {
            std::string file = glob_result.gl_pathv[i];
            std::string extension = "";
            int dotPosition = int(file.size()) - 1;
            for (; dotPosition >= 0; --dotPosition) {
                if (file[dotPosition] == '.') break;
                extension += file[dotPosition];
            }
            if (dotPosition < 0) continue;
            std::reverse(extension.begin(), extension.end());
            if (extension != "png" && extension != "jpeg" &&
extension != "jpg") {
                continue;
            }
            auto name = file.substr(0, dotPosition);
            bool mirror = false;
            if (file.find_last_of("mirror") == dotPosition - 1) {
                mirror = true;
                name = name.substr(0, name.size() -
std::strlen("_mirror"));
            }
            std::string pointsPath = name + ".pts";
            cv::Mat image = cv::imread(file,
CV_LOAD_IMAGE_GRAYSCALE);

            if (image.empty()) continue;
            std::ifstream stream(pointsPath);
            const int landmarkCount = 68;
            std::vector<Shape::Point> landmarks;
            std::string line;
            std::getline(stream, line);
            std::getline(stream, line);
            std::getline(stream, line);

            for (int i = 0; i < landmarkCount; ++i) {
                Shape::Point point;
                stream >> point.x >> point.y;
                if (mirror) {
                    point.x = image.cols - point.x;
                }
                landmarks.push_back(point);
            }
            if (mirror) {
                for (int i = 0; i < landmarkCount; ++i) {
                    int j = config.mirrorIndices[i];
                    if (j > i) {
                        std::swap(landmarks[i], landmarks[j]);
                    }
                }
            }
        }
    }

```

```

    }
}
// crop
int minX = std::numeric_limits<int>::max();
int maxX = std::numeric_limits<int>::min();
int minY = std::numeric_limits<int>::max();
int maxY = std::numeric_limits<int>::min();
for (int i = 0; i < landmarkCount; ++i) {
    int x = landmarks[i].x;
    int y = landmarks[i].y;
    if (x < minX) {
        minX = x;
    }
    if (x > maxX) {
        maxX = x;
    }
    if (y < minY) {
        minY = y;
    }
    if (y > maxY) {
        maxY = y;
    }
}
const double cropFactor = 2.8;
cv::Rect cropRegion;
cropRegion.width = (maxX - minX) * cropFactor;
cropRegion.height = (maxY - minY) * cropFactor;
cropRegion.x = (maxX + minX - cropRegion.width) / 2;
cropRegion.y = (maxY + minY - cropRegion.height) / 2;
if (cropRegion.x < 0) {
    cropRegion.x = 0;
}
if (cropRegion.y < 0) {
    cropRegion.y = 0;
}
if (cropRegion.x + cropRegion.width >= image.cols) {
    cropRegion.width = image.cols - cropRegion.x;
}
if (cropRegion.y + cropRegion.height >= image.rows) {
    cropRegion.height = image.rows - cropRegion.y;
}
if (cropRegion.size() != image.size()) {
    image = image(cropRegion).clone();
    for (int i = 0; i < landmarkCount; ++i) {
        landmarks[i].x -= cropRegion.x;
        landmarks[i].y -= cropRegion.y;
    }
}

auto area = image.size().area();
double scaleFactor = std::sqrt(area * 1.0 / (1 << 19));
if (scaleFactor > 1.2) {
    auto oldSize = image.size();
    cv::Size newSize(oldSize.width / scaleFactor,
oldSize.height / scaleFactor);
    cv::resize(image, image, newSize);
    for (int i = 0; i < landmarkCount; ++i) {

```

```

        landmarks[i].x *= newSize.width * 1.0 /
oldSize.width;
        landmarks[i].y *= newSize.height * 1.0 /
oldSize.height;
    }
}

Shape shape;
shape.setProjectionTranslate({0, 0});
shape.setProjectionScaleRotate({1, 0});
shape.setLocalCoordinates(landmarks);
data.push_back({image, std::make_shared<Shape>(shape)});
}
DBN(path, data.size());
globfree(&glob_result);

}

Shape calculateMeanShape(const std::vector<TrainingSample>& data)
{
    Shape meanShape;
    const int landamrkCount = data[0].truthShapePtr-
>getLandamrkCount();
    std::vector<Shape::Point> meanLandmarks(landamrkCount);
    for (auto& sample: data) {
        auto& truthShape = *sample.truthShapePtr;
        auto landmarks = truthShape.getLandmarkProjections();
        int minX = std::numeric_limits<int>::max();
        int maxX = std::numeric_limits<int>::min();
        int minY = std::numeric_limits<int>::max();
        int maxY = std::numeric_limits<int>::min();
        for (int i = 0; i < landamrkCount; ++i) {
            int x = landmarks[i].x;
            int y = landmarks[i].y;
            if (x < minX) {
                minX = x;
            }
            if (x > maxX) {
                maxX = x;
            }
            if (y < minY) {
                minY = y;
            }
            if (y > maxY) {
                maxY = y;
            }
        }
        for (int i = 0; i < landamrkCount; ++i) {
            landmarks[i].x = (landmarks[i].x - minX) / (maxX -
minX) - 0.5;
            landmarks[i].y = (landmarks[i].y - minY) / (maxY -
minY) - 0.5;
        }
        for (int i = 0; i < landamrkCount; ++i) {
            meanLandmarks[i] += landmarks[i];
        }
    }
    for (int i = 0; i < landamrkCount; ++i) {

```

```

        meanLandmarks[i] /= data.size();
    }
    std::vector<bool> used(landamrkCount);
    std::set<int> centralLandamrks = {27, 28, 29, 30, 33, 51, 62,
66, 57, 8};
    vector<int> indices;
    for (int i = 0; i < landamrkCount; ++i) {
        indices.push_back(i);
    }
    std::sort(indices.begin(), indices.end(), [&](int x, int y) {
        return std::abs(meanLandmarks[x].x) >
std::abs(meanLandmarks[y].x);
    });
    for (int i : indices) {
        if (used[i]) continue;
        int bestJ = -1;
        double bestDistance = 1e100;
        for (int j = 0; j < landamrkCount; ++j) {
            if (used[j]) continue;
            auto point = meanLandmarks[j];
            point.x = -point.x;
            double distance = abs(point - meanLandmarks[i]);
            if (distance < bestDistance) {
                bestDistance = distance;
                bestJ = j;
            }
        }
        assert(bestJ != -1);
        used[i] = true;
        used[bestJ] = true;
        auto mirrored = meanLandmarks[bestJ];
        mirrored.x = -mirrored.x;
        auto meanPoint = (meanLandmarks[i] + mirrored) / 2;
        meanLandmarks[i] = meanPoint;
        meanPoint.x = -meanPoint.x;
        meanLandmarks[bestJ] = meanPoint;
    }
    meanShape.setLocalCoordinates(meanLandmarks);
    return meanShape;
}

int main(int argc, const char * argv[]) {
    TrainingConfig config;
    std::vector<TrainingSample> trainingData;
    for (auto folder : folders) {
        loadData(resourcesPath + folder + "/", trainingData,
config);
    }
    std::random_shuffle(trainingData.begin(),
trainingData.end());
    DBN(trainingData.size());
    // calcualte mean shape
    auto meanShape = calculateMeanShape(trainingData);
    DBN("mean shape calculated");

    // refit landmarks
    for (auto& sample : trainingData) {
        refit(meanShape, *sample.truthShapePtr);
    }
}

```

```

    }
    DBN("refited");
    auto initialTrainingData = trainingData;
    std::vector<TrainingSample> testingData;
    for (int i = 0; i < trainingData.size() * 0.05; ++i) {
        testingData.push_back(trainingData.back());
        trainingData.pop_back();
    }
    auto generateNegativeData = [&](int expectedSize) {
        const int negativeDataPartCount = 6;
        int onePartSize = static_cast<int>(expectedSize /
negativeDataPartCount / initialTrainingData.size() *

initialTrainingData.size());
        std::vector<TrainingSample>
negativeDataParts[negativeDataPartCount];
        for (int i = 0; i < negativeDataPartCount; ++i) {
            negativeDataParts[i] = initialTrainingData;
        }
        expandData(negativeDataParts[0], onePartSize, 10, config,
{0.4, 1.00}, {-35, 35}, {-0.5, 0.5});
        expandData(negativeDataParts[1], onePartSize, 10, config,
{-1.00, -0.4}, {-35, 35}, {-0.5, 0.5});
        expandData(negativeDataParts[2], onePartSize, 10, config,
{-0.2, 0.2}, {-35, 35}, {-1.5, -0.5});
        expandData(negativeDataParts[3], onePartSize, 10, config,
{-0.2, 0.2}, {-35, 35}, {0.5, 1.5});
        expandData(negativeDataParts[4], onePartSize, 10, config,
{-0.2, 0.2}, {-180, -65}, {-0.2, 0.2});
        expandData(negativeDataParts[5], onePartSize, 10, config,
{-0.2, 0.2}, {65, 180}, {-0.2, 0.2});
        auto negativeData = std::move(negativeDataParts[0]);
        for (int i = 1; i < negativeDataPartCount; ++i) {
            negativeData.insert(negativeData.end(),
negativeDataParts[i].begin(), negativeDataParts[i].end());
            negativeDataParts[i] = std::vector<TrainingSample>();
        }
        return negativeData;
    };
    auto negativeData = generateNegativeData(440000);
    expandData(testingData, 10000, 100, config);
    DBN("expanded");
    DBN(trainingData.size());
    DBN(negativeData.size());
    config.meanShapePtr = std::make_shared<Shape>(meanShape);
    auto calcDistance = [&](const std::vector<TrainingSample>&
data){
        double res = 0;
        for (auto& sample : data) {
            auto truthLandmarks = sample.truthShapePtr-
>getLandmarkProjections();
            auto guessLandmarks = sample.guessShapePtr-
>getLandmarkProjections();
            double t = 0;
            for (int i = 0; i < guessLandmarks.size(); ++i) {
                t += norm(guessLandmarks[i] - truthLandmarks[i]);
            }

```

```

        res += std::sqrt(t) / sample.truthShapePtr->getProjectionScale();
    }
    return res / data.size();
};

Cascade cascade;
for (int i = 0; i < config.cascadeSize; ++i) {
    const int minNegativeSize =
config.minNegativeDataSize[config.regressorIndex];
    while (negativeData.size() < minNegativeSize) {
        auto newNegativeData =
generateNegativeData(minNegativeSize);
        for (auto& sample : newNegativeData) {
            bool result = cascade.process(sample.image,
*sample.guessShapePtr, true);
            if (result) {
                negativeData.push_back(sample);
            }
        }
        DBN(negativeData.size());
    }
    trainingData.clear();
    auto newTrainingData = initialTrainingData;
    expandData(newTrainingData, 300000, 2, config);
    trainingData.reserve(newTrainingData.size());
    for (auto& sample : newTrainingData) {
        bool result = cascade.process(sample.image,
*sample.guessShapePtr, true);
        if (result) {
            trainingData.push_back(sample);
        }
    }
    DBN(trainingData.size());
    DBN(negativeData.size());
    auto testingDistanceBefore = calcDistance(testingData);
    auto trainingDistanceBefore = calcDistance(trainingData);
    printf("Testing distance before = %f\n",
testingDistanceBefore);
    printf("Training distance before = %f\n",
trainingDistanceBefore);
    RegressorTrainer trainer;
    Regressor regressor = trainer.train(trainingData,
negativeData, config);
    int cntFailed = 0;
    for (auto& sample : testingData) {
        sample.score = 0;
        cntFailed += !regressor.process(sample.image,
*sample.guessShapePtr, sample.score, true);
    }
    printf("Failed testing = %.1f%%\n", cntFailed * 1.0 /
testingData.size());
    for (auto& sample : trainingData) {
        regressor.process(sample.image,
*sample.guessShapePtr);
    }
    for (auto& sample : negativeData) {

```

```

        regressor.process(sample.image,
*sample.guessShapePtr);
    }
    auto testingDistance = calcDistance(testingData);
    auto trainingDistance = calcDistance(trainingData);
    printf("Testing distance after = %f\n", testingDistance);
    printf("Training distance after = %f\n",
trainingDistance);
    puts("-----");
    printf("Testing improvement  = %.1f%%\n", (1 -
testingDistance / testingDistanceBefore) * 100);
    printf("Training improvement = %.1f%%\n", (1 -
trainingDistance / trainingDistanceBefore) * 100);
    puts("=====");
    cascade.push_back(regressor);
    puts("");
    puts("");
    std::ofstream of("trackRegressorV2.bin",
std::ios::binary);
    of << cascade;
    config.regressorIndex++;
    config.kthFactorInitial /= 2;
}
return 0;
}

```

ДОДАТОК Д
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДІВ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ НА
ЗОБРАЖЕННЯХ**

Виконала
ст. гр. КНТ-110м

В. Ю. Предко

Керівник
к.т.н., доцент

Є. О. Гофман

Рисунок Д.1 – Слайд 1

Об'єктом дослідження є процес ідентифікації об'єктів на зображеннях.

Предмет дослідження – методи та програмні засоби виявлення об'єктів на зображеннях.

Мета роботи – дослідження та програмна реалізація методів ідентифікації об'єктів на зображеннях.

Рисунок Д.2 – Слайд 2

Завдання роботи:

- виконати дослідження методів та систем виявлення та відстеження об'єктів на зображеннях;
- дослідити існуючу технологію, що дозволяє вирішувати задачу виявлення об'єктів заданого класу та відстеження виявлених об'єктів;
- розробити модифікований метод, що дозволяє поліпшити точність виявлення та прискорити швидкодію системи;
- розробити програму ідентифікації об'єктів заданого класу на зображеннях;
- виконати експериментальне дослідження програми, створеної на основі розробленого методу.

3

Рисунок Д.3 – Слайд 3

Методи виявлення об'єктів на зображеннях

Критерії порівняння	Методи		
	Метод порівняння шаблонів	Метод Віолі-Джонса	Штучні нейронні мережі
Простота	+	±	–
Надійність	–	+	+
Точність	–	+	+
Швидкість навчання	+	–	+
Швидкість виявлення	–	+	–
Стійкість до змін об'єкта	–	–	+
Розпізнавання об'єктів	–	–	+

4

Рисунок Д.4 – Слайд 4

Модифікований метод ідентифікації облич на зображення

- Використовується лінія очей для обчислення кута повороту зразка
- Зразок обертається у протилежний бік та передається детектору

$$d = \arctan \left(\frac{y_L - y_R}{x_L - x_R} \right)$$

5

Рисунок Д.5 – Слайд 5

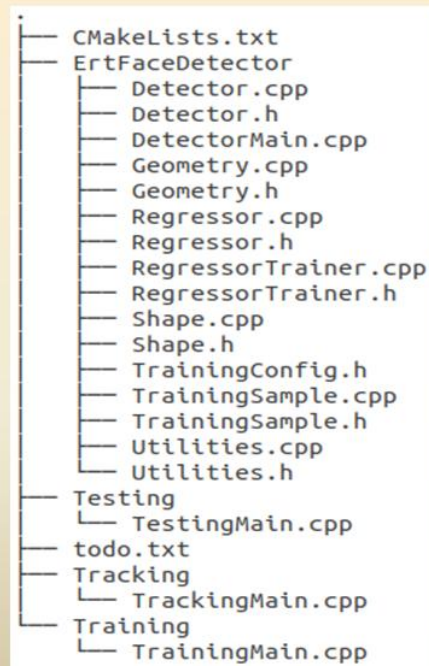
Вибір мови програмування

Критерій порівняння мов програмування	C++	Java	C#
Зручність використання для програмної реалізації методів ідентифікації об'єктів на зображеннях	+	+—	—
Наявність бібліотек для ідентифікації об'єктів на зображеннях	+	+—	+—
Можливість обробки баз даних	+	+	+
Зручність обробки виключних ситуацій	+	+—	+—
Засоби створення інтерфейсу користувача	+	+—	+—

6

Рисунок Д.6 – Слайд 6

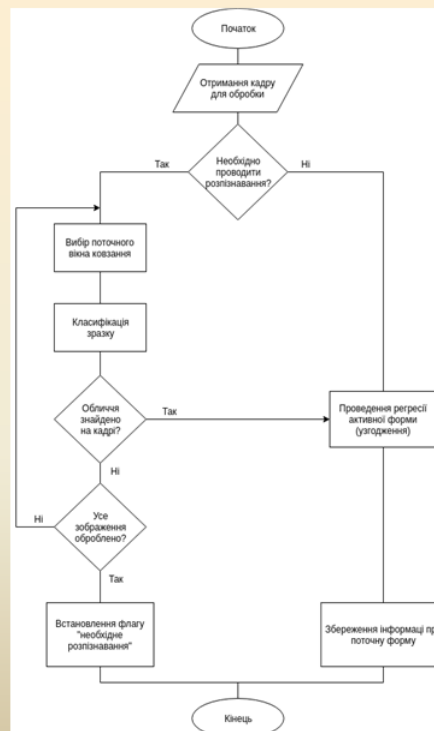
Файлова структура розробленої програми ідентифікації об'єктів на зображення



7

Рисунок Д.7 – Слайд 7

Узагальнена схема роботи програми



8

Рисунок Д.8 – Слайд 8

Робота з програмою. Конфігурація програмного забезпечення

```
struct TrainingConfig {
    static const int cascadeSize = 8;
    static const int treeDepth = 4;
    int ruleCount[cascadeSize] = {256, 240, 224, 208, 192, 176, 160, 256};
    int featurePointCount[cascadeSize] = {150, 250, 350, 500, 550, 550, 550, 550};
    int ensembleSize[cascadeSize] = {150, 200, 250, 350, 450, 500, 500, 500};
    double nu[cascadeSize] = {0.15, 0.15, 0.15, 0.15, 0.15, 0.1, 0.1, 0.1};
    double classificationProbability[cascadeSize] = {0.9, 0.8, 0.7, 0.6, 0.4, 0.2, 0.1, 0.1};
    const int minNegativeDataSize[cascadeSize] = {400000, 150000, 100000, 90000, 50000, 25000, 15000};
    double bestPositiveNegativeToIgnore = 0.001;

    int regressorIndex = 0;

    double kthFactorInitial = 0.00022;
    double kthFactorFinal = 0.00003;
    int unitCount = 40;
    int normalizationStep = 300;

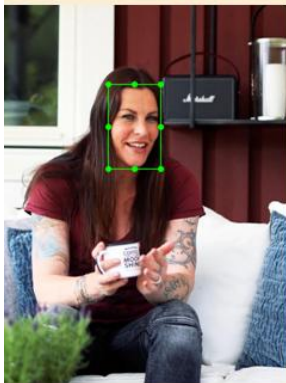
    int mirrorIndices[68] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
                             26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 27, 28, 29, 30, 35, 34, 33,
                             32, 31, 45, 44, 43, 42, 47, 46, 39, 38, 37, 36, 41, 40, 54, 53, 52,
                             51, 50, 49, 48, 59, 58, 57, 56, 55, 64, 63, 62, 61, 60, 67, 66, 65};

    ShapePtr meanShapePtr;
    std::default_random_engine rng;
};
```

9

Рисунок Д.9 – Слайд 9

Результати роботи програми



Результат пошуку обличчя без нахилу на
зображенні немодифікованим алгоритмом
(знайдено)



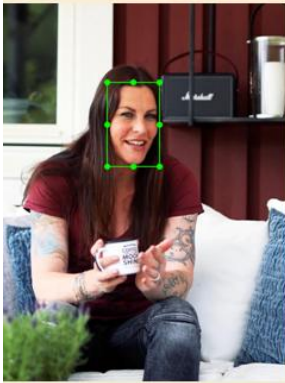
Результат пошуку обличчя знахилом на
зображенні немодифікованим алгоритмом
(не знайдено)



10

Рисунок Д.10 – Слайд 10

Результати роботи програми



Результат пошуку обличчя без нахилу на зображенні модифікованим алгоритмом (знайдено)

Результат пошуку обличчя знахилом на зображенні модифікованим алгоритмом (знайдено)

11

Рисунок Д.11 – Слайд 11

Порівняльний аналіз систем для виявлення та відстеження обличч людей

Критерії порівняння	Програмні системи			
	OpenCV	Google Vision API	Dlib Library	Розроблена програма
Пошук облич	+	+	+	+
Відстеження обличчя	±	–	+	+
Виявлення обличч з нахилом	–	±	–	+
Портативність	+	± (веб API)	+	+

12

Рисунок Д.12 – Слайд 12

Висновки

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес та методи ідентифікації об'єктів на зображеннях.

В роботі було розглянуто задачу ідентифікації облич людей як окремий випадок проблеми виявлення об'єктів заданого класу на зображеннях. Було розглянуто та описано модифікації, які можливо внести для вдосконалення існуючої технології пошуку та відстеження об'єктів. Для створення програмного забезпечення для ідентифікації об'єктів на зображеннях було обрано мову програмування C++, яка є досить зручною для розробки програм подібного роду. Також роботи з зображеннями, зокрема для завантаження, обрізання, масштабування, зміни кольору тощо, було вирішено підключити до проекту відкриту бібліотеку функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення OpenCV, яка має відкритий код.

13

Рисунок Д.13 – Слайд 13

Висновки

Наукова новизна роботи полягає у тому, що розроблено удосконалений ідентифікації об'єктів на зображеннях, у якому, на відміну від існуючих, на етапі виявлення облич використовується модифікований алгоритм для ідентифікації об'єктів, обернених під довільним кутом, крім того, використовується вдосконалений процесу навчання системи шляхом використання кросвалідації, що дозволяє підвищити точність ідентифікації об'єктів.

Практичне значення одержаних результатів полягає у тому, що розроблено програмне забезпечення для ідентифікації об'єктів на зображеннях, яке може бути використано у різних прикладних застосунках від зору роботів до розпізнавання образів.

Аналіз та тестування дозволили виявити ряд недоліків та проблемних умов роботи реалізованої технології. Були запропоновані ідеї та напрямки розвитку, що дозволять в майбутньому покращити якість роботи технології.

14

Рисунок Д.14 – Слайд 14