Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інженерії програмного забезпечення

Чумаченко Данило Кирилович,
студент групи АС-172

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Програмна система машинного зору для біометричної ідентифікації осіб за двовимірними зображеннями з використанням методів штучного інтелекту.
Підтема 2. Клієнтська частина

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Крісілов Віктор Анатолійович,
доктор технічних наук, професор

Одеса – 2022

ЗМІСТ

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ 4
АНОТАЦІЯ 6
ВСТУП 7
1 АНАЛІЗ ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 10
1.1 Короткий огляд розвитку біометричних технологій 10
1.2 Аналіз архітектурного підходу 13
1.3 Порівняльний аналіз аналогів 15
1.4 Висновки до розділу 19
2 РОЗРОБКА СИСТЕМИ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 20
2.1 Класифікація задач аналізу зображень 20
2.2 Аналіз та вибір методів обробки зображень 25
2.3 Вибір датасету для навчання нейронної мережі 31
2.4 Висновки до розділу 32
3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 33
3.1 Варіанти використання для клієнтської частини системи 33
3.2 Нефункціональні вимоги для клієнтської частини 40
3.3 Висновки до розділу 42
4 ПРОЕКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 43
4.1 Проектування архітектури системи 43
4.2 Структури даних системи клієнтської частини системи 46
4.3 Проектування інтерфейсу застосування 49
4.4 Алгоритми взаємодії з інтерфейсом 51

4.4 Проектування програмних класів 53
4.5 Висновки до розділу 61
5 ПРОГРАМНА РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 62
5.1 Особливості створення Android-застосунків та системні вимоги 62
5.2 Налаштування адаптивного інтерфейсу 63
5.3 Приклади використання інтерфейсу 64
5.3 Висновки до розділу 70
6 ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ 71
6.1 Тестування роботи алгоритмів обробки двовимірних зображень 71
6.2 Дослідження впливу кута повороту зображення на результат 74
6.3 Висновки до розділу 76
ВИСНОВКИ 77
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 78
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ 80

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інженерії програмного забезпечення

Рівень вищої освіти: другий (магістерський)
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Комлева Н. О.
«___» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чумаченко Данило Кириловича, група АС-172

1. Тема роботи: Програмна система машинного зору для біометричної ідентифікації осіб за двовимірними зображеннями з використанням методів штучного інтелекту. Підтема 2. Клієнтська частина.

Керівник роботи: Крісілов Віктор Анатолійович, доктор технічних наук, професор.

затверджені наказом ректора від «20» жовтня 2022 р. № 399-в.

2. Зміст роботи: критичний аналіз існуючих рішень, специфікація вимог, розробка системи, проектування клієнтської частини програмної системи, програмна реалізація клієнтської частини системи, тестування системи

3. Перелік ілюстративного матеріалу: згідно зі слайдами презентації.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Дата видачі завдання: «01» вересня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз особливостей предметної області	15.09.2022	виконав
2	Аналіз та вибір методів обробки зображень	01.10.2022	виконав
3	Специфікація вимог	15.10.2022	виконав
4	Проектування клієнтської частини системи	22.10.2022	виконав
5	Програмна реалізація клієнтської частини проекту та бази даних	10.11.2022	виконав
6	Дослідження та тестування програмного забезпечення	22.11.2022	виконав
7	Оформлення документації	05.12.2022	виконав

Здобувач вищої освіти _____ Д. К. Чумаченко

Керівник роботи _____ В. А. Крісілов

АНОТАЦІЯ

Метою роботи є підвищення достовірності біометричної ідентифікації осіб за двовимірними зображеннями з використанням методів штучного інтелекту. В даній кваліфікаційній роботі розроблено клієнтську частину програмної системи машинного зору, а також опрацьовано методи попередньої обробки двовимірних зображень.

Методи розробки засновані на використанні нейронних мереж, які виконують біометричну ідентифікацію за зображеннями обличчя людини та за її соматотипом. Розробка виконана на основі Android з використанням мови програмування Kotlin.

Ключові слова: машинний зір, архітектура клієнтської частини, біометрична ідентифікація, штучний інтелект, Android, Kotlin.

ABSTRACT

The aim of the work is to increase the accuracy of biometric identification of persons based on two-dimensional images using artificial intelligence methods. In this qualification work, the client part of the machine vision software system was developed, as well as the methods of preliminary processing of two-dimensional images were developed.

Processing methods are based on the use of neural networks that perform biometric identification based on images of a person's face and their somatotype. The development is based on Android using the Kotlin programming language.

Keywords: machine vision, client-side architecture, biometric identification, artificial intelligence, Android, Kotlin.

ВСТУП

«Біометрична ідентифікація» – це загальний термін для технологій, які дозволяють збігатися між «живим» цифровим зображенням частини тіла та попередньо записаним зображенням тієї самої частини, зазвичай індексованим відповідно до особистої чи фінансової інформації.. Біометричні ідентифікатори включають цифрові відбитки пальців, сканування сітківки ока, геометрію руки, характеристики обличчя та голосові моделі.

Системи біометричного сканування, як правило, записують не весь відбиток фізичної характеристики, а лише ту частину, або «шаблон», який повинен бути незмінним у часі в межах деякого статистичного обмеження. Оскільки тіло змінюється з плином часу статистичний алгоритм має бути достатньо еластичним, щоб зіставити збережене зображення з пізнішим живим скануванням тієї ж особи, зазвичай без зіставлення двох подібних осіб. Це створює обмеження на унікальність зображень, які долаються використанням кількох зображень однієї особи або біометричного зображення та іншої інформації. У деяких програмах ідентифікаційні дані можна перевірити в межах мільйонного населення.

Хтось, хто переглядає біометричну базу даних, може не побачити жодного зображення, і йому знадобляться для цього спеціальні алгоритми (наприклад, щоб побачити дані щодо зображення сітківки ока чи відбитка пальця).

Оскільки такі алгоритми та формули є запатентованими, користуючись алгоритмами з компанії А не можливо розшифрувати дані з компанії В. Однак існують економічні стимули для розробки стандартів шифрування; наприклад, двом банкам, які використовують різних постачальників біометричних даних, знадобляться такі стандарти, щоб забезпечити перехресне розпізнавання відбитків пальців для транзакцій банкоматів.

Порівняно з візуальним порівнянням підписів або посвідчень особи з фото, біометрична ідентифікація є менш помилковою та потенційно набагато швидшою.

Це спонукало до використання біометрії для некримінальних державних і комерційних додатків. Джеймс Л. Веймен, директор тестового центру біометрії в Університеті штату Сан-Хосе, згадує ряд таких програм, які були активними станом на травень 1998 року [1], включаючи іммиграційні системи, системи безпеки в аеропортах, реєстрацію робочого часу працівників, розподіл соціальних пільг і програми отримання водійських прав.

Міжнародна асоціація біометричної промисловості (МАБП) запропонувала відкритий список потенційних застосувань, включаючи реєстрацію виборців, доступ до медичних записів, банківські операції, «національні системи ідентифікації» та «батьківський контроль». Останні два пункти вони не уточнюють.

Деякі програми біометричної ідентифікації не є добровільними, як-от системи відстеження злочинців; майже всі інші програми є добровільними. Добровільні програми є або обов'язковими, коли біометричний ідентифікатор потрібний для отримання певної вигоди (доступу до рахунку та ін.), або необов'язковими, коли допускається інша форма ідентифікатора. Додаткові системи дозволяють користувачам підключатися, добровільно користуючись біометричним методом, або відмовлятися, якщо вони цього не хочуть. Усі торговельні та фінансові застосування, де біометричні ідентифікатори забезпечують швидшу оплату або доступ до облікових записів, повинні бути системами підключення, доки біометричні ідентифікатори не стануть такими ж поширеними, як адреси електронної пошти.

Об'єктом дослідження є біометрична ідентифікація осіб за двовимірними зображеннями.

Предметом є клієнтська частина програмної системи машинного зору.

Метою роботи є підвищення достовірності біометричної ідентифікації осіб за двовимірними зображеннями з використанням методів штучного інтелекту. В даній кваліфікаційній роботі розроблено клієнтську частину програмної системи машинного зору, а також опрацьовано методи попередньої обробки двовимірних

зображень.

Розглянемо задачі, що необхідно виконати для досягнення поставленої мети:

1) Короткий огляд розвитку біометричних технологій;
2) Аналіз архітектурного підходу;
3) Аналіз аналогів;
4) Класифікація задач аналізу зображень;
5) Аналіз та вибір методів обробки зображень;
6) Вибір датасету для навчання нейронної мережі;
7) Варіанти використання для клієнтської частини системи
8) Нефункціональні вимоги для клієнтської частини
9) розробка архітектури клієнт-серверної частини на основі Android;
10) розробка бази даних системи;
11) налаштування взаємодії клієнта з сервером;
12) реалізація функцій;
13) проведення дослідження та тестування функцій;

Представлена робота містить 6 розділів. У першому розділі було розглянуто сучасний стан проблеми ідентифікації та обробки зображень. У другому розділі виконано розробку для клієнтської частини системи. У третьому розділі визначені варіанти використання та нефункціональні вимоги для клієнтської частини системи. У четвертому розділі було описано проектування клієнтської частини системи, а саме основні використовувані об’єкти і програмне забезпечення. У п’ятому розділі було описано застосування клієнтської частини системи і була представлена інструкція з використання клієнтської системи. У шостому розділі проведено тестування та описано контрольний приклад.

1 АНАЛІЗ ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ОБЛАСТІ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

1.1 Короткий огляд розвитку біометричних технологій

Існує безліч біометричних методів, які широко використовуються або досліджуються. До них відносяться зображення обличчя (як оптичне, так і інфрачервоне), геометрія рук і пальців, методи на основі ока (райдужка і сітківка), підпис, голос, геометрія вен, натискання клавіш і зображення відбитків пальців і долонь. Розглянемо більш детально деякі з цих методів.

Обличчя. Зображення обличчя, ймовірно, є найпоширенішою біометричною характеристикою, яка використовується людьми для ідентифікації особи. Ідентифікація на основі обличчя є однією з найактивніших областей досліджень, із застосуваннями, починаючи від статичної контрольованої верифікації фотографій до динамічної неконтрольованої ідентифікації обличчя на захараченому фоні. Підходи до розпізнавання обличчя зазвичай базуються на розташуванні та формі атрибутів обличчя, таких як очі, брови, ніс, губи та форма підборіддя та їх просторових співвідношеннях; загальний (глобальний) аналіз зображення обличчя та його розбивка на ряд канонічних облич або їх комбінацію.

Хоча продуктивність комерційно доступних систем є прийнятною, залишається сумнівним, чи саме обличчя без будь-якої контекстної інформації є достатньою основою для розпізнавання людини за великою кількістю ідентичностей із надзвичайно високим рівнем достовірності. Важко розпізнати обличчя на зображеннях, зроблених із двох кардинально різних ракурсів. Крім того, сучасні системи розпізнавання обличчя накладають ряд обмежень на спосіб отримання зображень обличчя, іноді вимагаючи простого фону або спеціального освітлення. Для того, щоб системи розпізнавання обличчя були широко поширені, вони повинні автоматично визначати, чи присутній обличчя на отриманому зображенні; знайти обличчя, якщо воно є; і розпізнати обличчя із загальної точки зору.

Термограма обличчя. Основна судинна система в обличчі людини виробляє унікальний підпис обличчя, коли тепло проходить через тканини обличчя та виділяється зі шкіри. Такі підписи обличчя можна зафіксувати за допомогою інфрачервоної камери, що призводить до отримання зображення, яке називається «термограма обличчя». Стверджується, що термограма обличчя унікальна для кожної людини і не піддається маскуванню. Вважається, що навіть пластична хірургія, яка не перенаправляє кровотік по венах, не впливає на формування термограми обличчя. Термограма обличчя – це неінтрузивна біометрична техніка, яка дозволяє підтвердити особу без контакту. Заявлена перевага розпізнавання на основі термограми обличчя над візуальним розпізнаванням обличчя за допомогою CCD (Charged Coupled Device) [2] камер ґрунтується на наступних спостереженнях: інфрачервона камера може захопити термограму обличчя за дуже слабкого зовнішнього освітлення або взагалі за відсутності світла; судинна структура може бути більш насиченою інформацією та залишається інваріантною до навмисних чи ненавмисних варіацій візуального вигляду обличчя.

Хоча це може бути правдою, що термограми обличчя є унікальними для кожної людини, не було доведено, що термограми обличчя є достатньо розрізнювальними. Термограми обличчя можуть значною мірою залежати від ряду факторів, таких як емоційний стан суб’єктів або температура тіла, і, як і розпізнавання обличчя, розпізнавання термограми обличчя залежить від зору.

Відбитки пальців. Люди використовували відбитки пальців для ідентифікації особистості протягом століть, і дійсність ідентифікації за відбитками пальців була добре встановлена. Відбиток пальця — це малюнок виступів і борозен на поверхні кінчика пальця, формування яких визначається у внутрішньоутробному періоді.

З розвитком твердотільних датчиків гранична вартість включення біометричної системи на основі відбитків пальців незабаром може стати доступною для багатьох застосувань. Отже, очікується, що відбитки пальців лідируватимуть у біометричних програмах у найближчому майбутньому, причому численні відбитки пальців

нададуть достатню інформацію для широкомасштабного розпізнавання за участю мільйонів осіб. Однією з проблем технології відбитків пальців є її неприйнятність типовим користувачем, оскільки відбитки пальців традиційно асоціювалися з кримінальними розслідуваннями та роботою поліції. Інша проблема полягає в тому, що автоматична ідентифікація відбитків пальців зазвичай потребує великої кількості обчислювальних ресурсів. Нарешті, відбитки пальців невеликої частини населення можуть бути непридатними для автоматичної ідентифікації через генетичні, вікові, екологічні чи професійні причини.

Геометрія руки. В якості біометричних характеристик можна використовувати різні вимірювання руки людини, включаючи її форму, довжину і ширину пальців. Біометричні системи на основі геометрії руки були встановлені в сотнях місць по всьому світу. Техніка дуже проста, відносно проста у використанні та недорога.

Візерунок сітківки. Візерунок, утворений венами під поверхнею сітківки ока, є стабільним і унікальним і, отже, є точною та можливою характеристикою для розпізнавання. Цифрові зображення візерунків сітківки можна отримати шляхом проектування пучка візуального або інфрачервоного світла низької інтенсивності в око та захоплення зображення сітківки за допомогою оптики, схожої на ретинаскоп.

Райдужка — це кільцева область ока, обмежена зіницею та склерою (білком ока) з обох боків[3]. Візуальна текстура райдужної оболонки стабілізується протягом перших двох років життя, а її складна структура несе дуже характерну інформацію, корисну для ідентифікації особи. Початкові доступні результати щодо достовірності та швидкості ідентифікації на основі райдужної оболонки є багатообіцяючими та вказують на можливість широкомасштабного розпізнавання за допомогою інформації райдужної оболонки.

Підпис. Кожна людина має унікальний стиль почерку. Однак немає двох повністю ідентичних підписів людини; відхилення від типового підпису також залежать від фізичного та емоційного стану людини. Достовірність ідентифікації

систем, заснованих на цій дуже поведінковій біометрії, прийнятна, але, здається, недостатньо висока, щоб привести до широкомасштабного розпізнавання.

Виступ. Мовлення – це переважно поведінкова біометрія. Мовлення людини є характерним, але може не містити достатньо інваріантної інформації, щоб запропонувати широкомасштабне розпізнавання.

1.2 Аналіз архітектурного підходу

Для зручної і правильної розробки було вирішено розділитися на клієнтську і серверну. Перевагами клієнт-серверної архітектури є наступні:

1) мережа клієнт-сервер має централізоване управління. тобто централізовані облікові записи користувачів, безпеку і доступ для спрощення адміністрування мережі;
2) систему можна масштабувати для підтримки великої кількості клієнтів одночасно;
3) оскільки всі дані зберігаються на сервері, легко зробити резервну копію;
4) зменшується реплікація даних - дані зберігаються на серверах замість кожного клієнта.

Однак не буває архітектур, які мають лише переваги. Тому потрібно зважити та оцінити співвідношення між перевагами та недоліками.

Недоліки клієнт-серверної мережі:

1) відмова сервера призводить до відмови всієї системи;
2) установка і управління є дорогим, так як потрібне спеціальне обладнання (сервер) і спеціальне програмне забезпечення як для серверу, так і для клієнта;
3) потрібні професійні фахівці для обслуговування серверів та інших технічних деталей системи.

У даній роботі демонструється клієнтська сторона, її взаємодія з сервером і користувачем. Також опрацьовано методи попередньої обобки двовимірних зображень.

Застосовуватись для розробки буде мова програмування Kotlin оскільки [4]:

1) при роботі з Kotlin можна скоротити код;
2) такий код буде більш прозорим і зрозумілим;
3) в міру розширення проекту на мові Java кількість рядків коду збільшується, Kotlin вирішує цю проблему;
4) Kotlin повністю сумісний з Java і рекомендований компанією Google до використання.

В рамках даної кваліфікаційної роботи, необхідно виконати наступні завдання, для забезпечення повного функціоналу:

- Розробка клієнт-серверну частину на основі Android. Необхідно розробити клієнт-серверну частину, яка буде відповідати за взаємодію користувача з серверною частиною за допомогою мобільного пристрою. Клієнтська частина буде складатися з 3 головних екранів:

1) екран перегляду зображень, отриманих з камери;
2) екран перегляду зображень, отриманих з репозиторію;
3) екран повідомлень стосовно результатів проведення біометричної ідентифікації;
4) екран, який буде відповідати за управління і налаштування функцій біометричної ідентифікації.

Всі інші екрани (налаштування, підтримка користувача і т. п.) будуть додаватися в міру необхідності.

- Налаштування взаємодії клієнта з сервером. Для вирішення даного завдання, потрібно вирішити ряд проблем, які пов'язані з тим, що ми повинні розробити клієнтську частину на Android платформі, в якій необхідно реалізувати загальну авторизацію, яка дала б можливість зробити спілкування користувача з ПЗ

максимально доброзичливим і комфортним.

- Розробка БД для клієнтської частини. В Android для розробки локальної БД використовується SQLite, оскільки це дуже легка база даних, для використання якої в Android були розроблені спеціальні бібліотеки, які дозволяють зробити розробку максимально вигідною. Локальна БД особливо знадобитися для формування повідомлень від сервера, так як необхідно буде зберігати останні події.

- Розробка архітектури клієнтської частини програми. Архітектура, яка буде застосовуватися в розробці клієнтської частини, повинна буде максимально реалізувати потреби системи.

- Розробка алгоритмів обробки зображення для нормалізації рівня яскравості, зниження шуму зображення, розрахунок градієнта для опрацювання окремих погіршених фрагментів та детектор кутів Харріса для підвищення якості зображення.

1.3 Порівняльний аналіз аналогів

Крім алгоритмів-аналогів розроблюваної системи Fast R-CNN, RetinalNet, R-FCN та YOLO, які вже були описані в серверної частині роботи, у даній кваліфікаційній роботі слід зосередитись саме на реалізаціях для Андроід та відповідних інтерфейсах.

Найбільш розповсюдженою є реалізація біометричної ідентифікації на основі відбитків пальців. На рис. 1.1 показані основні екранні форми Андроід-програми iFING Scanner, яка працює як безконтактний сканер відбитків пальців [5].

Ця програма фотографує пальці Андроід-користувача за допомогою вбудованої у мобільний пристрою камери мобільного пристрою, створює зображення відбитків пальців, але не збирає іншу особисту інформацію. Зйомка та попередній перегляд (ескізів) зображень відбитків пальців є безкоштовними. Більш розширені можливості потребують платної підписки.

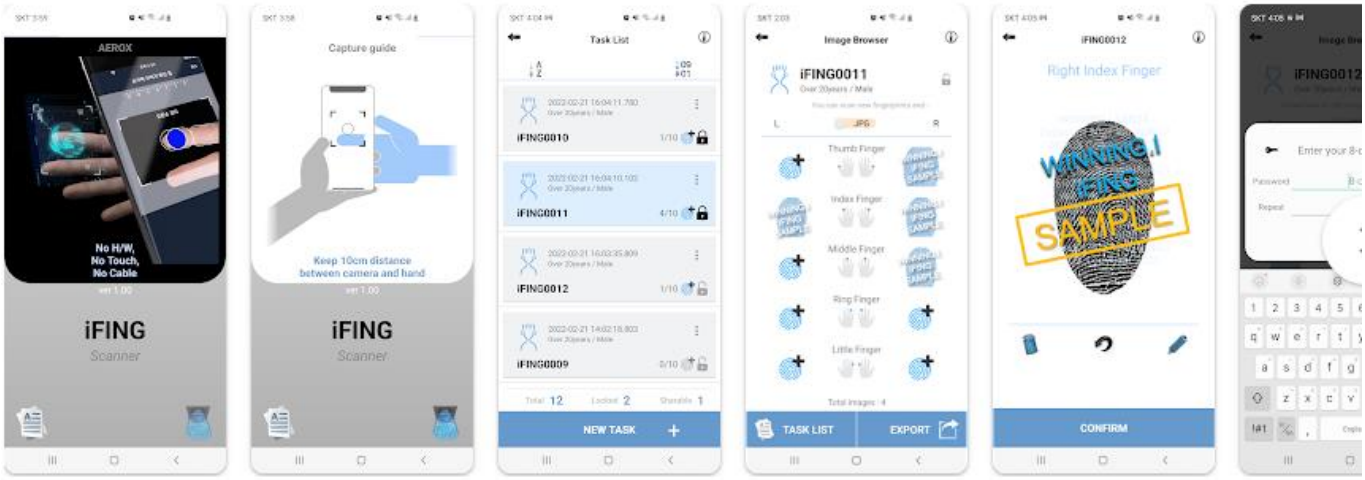

Рисунок 1.1 – Основні форми iFING Scanner

Застосунок для автентифікації 2FA орієнтований на підвищення безпеки даних та профілів соціальних мереж за допомогою двофакторної перевірки та безкоштовної MFA – багатофакторної автентифікації (рис. 1.2). За його допомогою можна «захистити облікові записи на різних платформах, таких як Facebook, Instagram, Snapchat, Telegram, Google, Microsoft і багатьох інших – понад 1000 сервісів.

Безпека визначається тим, як додаток збирає та кому передає ваші дані. Способи забезпечення конфіденційності й захисту даних можуть різнитися залежно від використання застосунку, регіону та віку користувача. Розробник, який надає цю інформацію, може оновлювати її.» [6]

Незважаючи на унікальні можливості, застосунок не може вважатись прямим аналогом розроблюваної системи.

На рис. 1.3 показана робота Андроід-застосунку Insyt Biometric, теж призначеного для ідентифікації користувача за відбитками пальців. Він потребує операційну систему Android 4.4 і новіших версій та займає лише 37 МБ. Проте, користувачі відмічають недостатки у зручності інтерфейсу, та ставлять оцінку 2,9 з максимальних 5.

«Цей застосунок має доступ до таких даних: фото, медіа-вміст, файли носія USB, змінення чи видалення вмісту носія USB» [7].

Рисунок 1.2 – Основні екранні форми застосунку для автентифікації 2FA

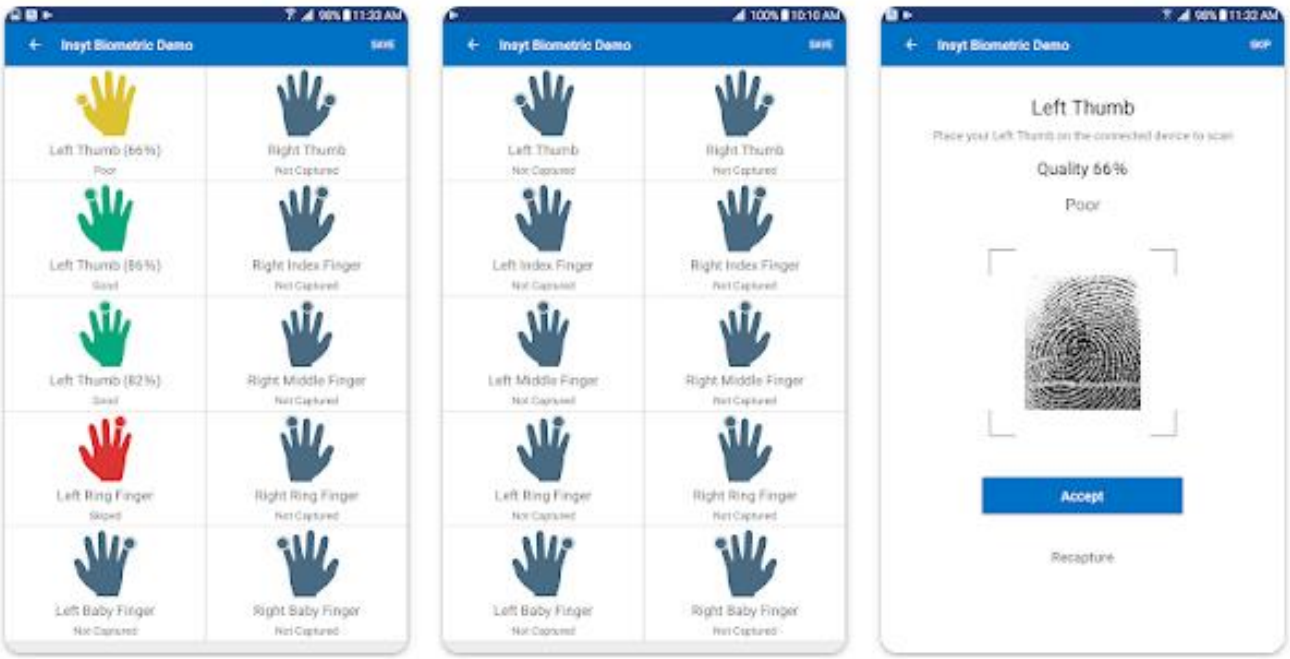

Рисунок 1.3 – Робота застосунку Insyt Biometric

Застосунок BioID Facial Recognition виконує біометричну ідентифікацію за зображеннями обличчя користувача (рис. 1.4). Дані не передаються третім сторонам. Вони є зашифрованими, що підвищує безпеку використання Андроід-застосунку.

Рисунок 1.4 – Основні форми BioID Facial Recognition

При необхідності дані можна видалити з системи, для цього потрібно попередньо надіслати запит [8]. Оцінка користувачів для BioID Facial Recognition є невисокою – 2,7 з 5.

Останній аналог, який варто проаналізувати, це DNI BioFacial, який дозволяє зробити фото та провести біометричну перевірку особи (рис. 1.5). Застосунок виконує безпечне шифрування даних та має більш високу оцінку – 3,9 [9].

Рисунок 1.5 – Робота Андроід-застосунку DNI BioFacial

Таким чином, вивчення аналогів та аналіз їх сильних та слабких сторін дозволило виявити загальні вимоги до клієнтської складової:

– застосунок повинен мати дружній та зрозумілий інтерфейс з невеликою кількістю екранних форм;

– при розробці застосунку не доцільно використовувати платні бібліотеки та скіни з тим, щоб це не впливало на кінцеву вартість продукту;

– з метою розширення кола користувачів застосунок не повинен орієнтуватись на надостанні версії ОС Андроід;

– застосунок повинен відповідати вимогам для можливості його подальшого розповсюдження через Google Play.

1.4 Висновки до розділу

Виходячи з усього розглянутого у розділі 1, можна зробити висновок, що розробка системи машинного зору для проведення біометричної ідентифікації з використанням Андроід-пристрою, є актуальним завданням. Огляд сучасних рішень та аналіз сильних та слабких сторін дозволив сформулювати загальні вимоги до клієнтської частини програмної системи.

2 РОЗРОБКА СИСТЕМИ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

2.1 Класифікація задач аналізу зображень

У даний час область досліджень, що стосуються аналізу зображень обличчя людини, розвивається дуже стрімко. Постійно з’являються нові задачі аналізу, які можна віднести до різних класів.

На рис. 2.1 показана класифікація найбільш відомих (розповсюджених) задач аналізу зображень обличчя людини. Зрозуміло, що для вирішення задач кожного з наведених класів необхідно застосування спеціального математичного апарату та інформаційно-обчислювальних інструментів.

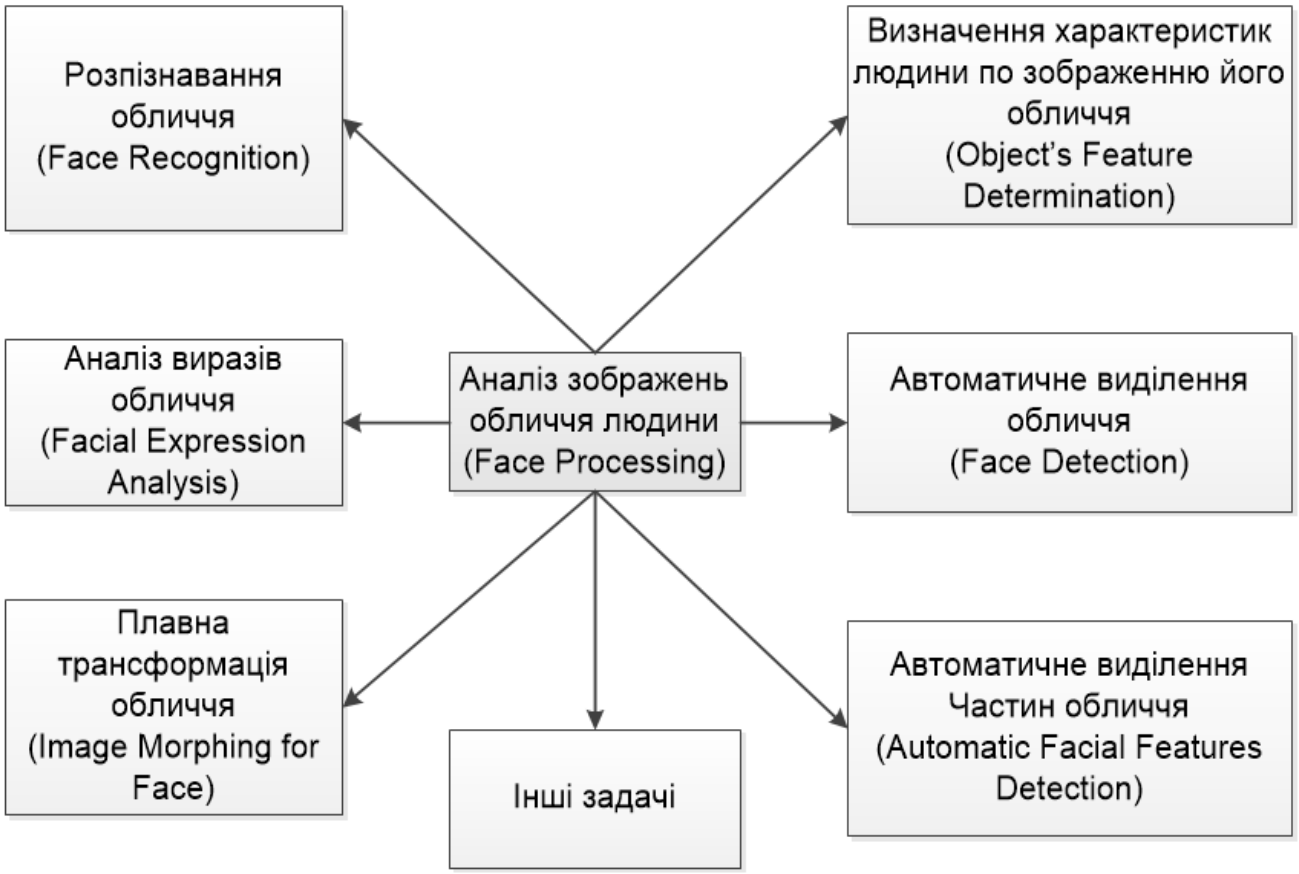

Рисунок 2.1 – Схема класифікації задач аналізу зображень обличчя людини

У даній роботи вирішується задача класу «Розпізнавання обличчя» (Face Recognition).

Цей клас задач може вирішуватись на основі:

– двовимірних зображень, які надходять на вхід системи;

– потокового відео, з якого виділяються зображення в режимі реального часу.

Відносно до характеристик процесу розпізнавання він може проводитись в режимах online чи offline.

Існує досить багато принципово різних підходів та методів для вирішення задачі ідентифікація обличчя (рис. 2.2).

Основними критеріями при оцінці та виборі потрібного методу є обчислювальна вартість алгоритмів, побудованих на їх основі, і достовірність розпізнавання.

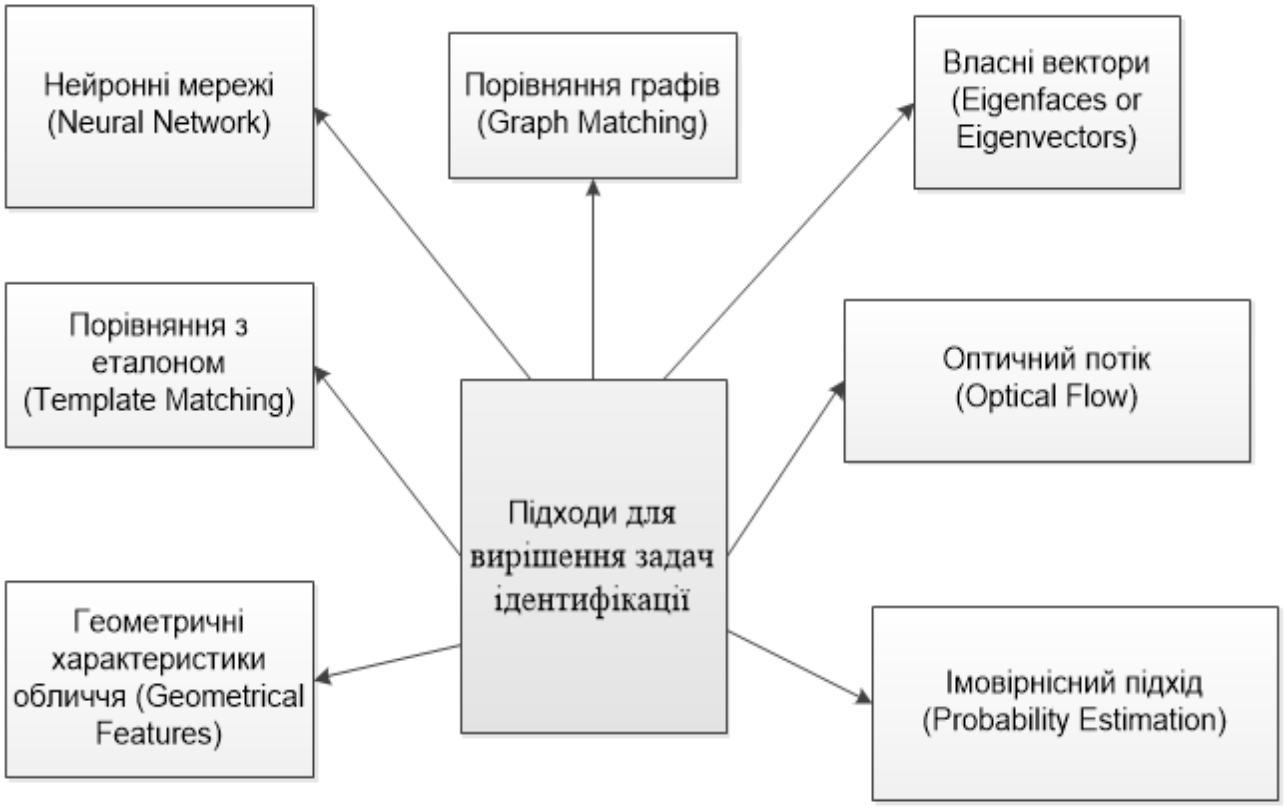

Рисунок 2.2 – Класифікація підходів та методів біометричної ідентифікації

Спосіб власних векторів (EigenVectors, EigenFaces).

Метод власних векторів ще називають методом головних компонентів осіб. Він є прикладом того, як математичні методи (метод аналізу основних компонентів), які успішно застосовувалися в інших областях, виявилися ефективно адаптованими до розпізнавання людей за їхніми портретами.

Будь-яке цифрове зображення може бути представлене у вигляді вектора у просторі ознак. Якщо зображення описується w×h пікселями, розмірність найпростішого векторного простору, до якого даний вектор належить, дорівнюватиме добутку w на h і, відповідно, базис подібного векторного простору буде складатися з w×h векторів. Однак у зв'язку з тим, що всі людські особи схожі між собою (овальна форма з носом, ротом, очима тощо), всі вектори, що описують зображення облич, будуть розміщуватись у вузько обмеженій області даного векторного простору. Тому при вирішенні завдання розпізнавання людей за портретом опис та зберігання всього векторного простору не є раціональним.

Таким чином, постає питання побудови простору меншої розмірності, в якому зображення людських осіб описуються компактніше. Однією з варіантів є простір, базисними векторами якого є основні компоненти всіх зображень осіб, що містяться в ньому.

Розмірність такого простору заздалегідь визначити неможливо, але вона набагато менша за розмірність векторного простору всіх зображень. З вищесказаного випливає, що головною метою методу аналізу принципових компонентів є значне зменшення розмірності простору ознак таким чином, щоб воно якнайкраще описувало "типові" образи, що належать безлічі портретів. У разі застосування даного методу для ідентифікації осіб такими образами слугуватимуть навчальні зображення.

Іншими словами, за допомогою аналізу основних компонентів вдається виявити всілякі мінливості в навчальному наборі зображень осіб та описати цю мінливість за допомогою декількох змінних. Ці змінні є w×h – розмірні вектори, які називаються

власними. Якщо перетворити подібні вектори на зображення, то одержувані картинки відображатимуть головні компоненти представленої навчальної множини (також звані власними особами). Таким чином, за рахунок зниження розмірності простору базисних векторів, в якому знаходяться зображення, домагаються хороших показників як швидкості, так і достовірності розпізнавання.

Імовірнісний підхід (Probability estimation).

Подібно до попереднього методу в імовірнісних моделях також використовується навчальний набір. У цьому формуються два класи із усіх варіантів уявлення об'єктів: внутриоб’єктної і зовнішньої мінливості, тобто, відбираються ознаки, за якими всі портрети поділяються на два класи:

– портрет даної людини;
– усі інші портрети.

Функції щільності ймовірності для кожного класу оцінюються за допомогою згаданої вище навчальної множини і згодом використовуються для обчислення міри схожості, яка ґрунтується на отриманих дослідним шляхом ймовірностях. Крім того, для отримання більш точних результатів іноді використовується модель ймовірності деякого фізичного процесу, за допомогою якої і формується остаточна міра схожості двох зображень.

Спосіб зіставлення з стандартом (Template Matching).

У цьому підході процес розпізнавання розбивається частини, відповідні окремим рис обличчя. Кожна фотографія, що надходить на вхід системи, що розпізнає, повинна являти собою фронтальне зображення особи людини з певною для конкретної бази даних кількістю масок, що представляють основні для ідентифікації регіони особи (наприклад, очі, ніс, рот і нижня частина особи). Крім того, розташування даних масок повинні бути однаково нормалізовані (наприклад, щодо очей) для всіх зображень у базі даних.

Під час процесу розпізнавання, коли частини вхідного зображення по черзі порівнюються з частинами зображення, що зберігається в базі, використовується

вектор, що відображає результат порівняння в балах (один бал за кожну рису особи) і обчислюється шляхом нормалізованої взаємної кореляції (втім, методи порівняння можуть бути різними). Після чого вхідне зображення класифікується відповідно до максимально набраних балів. Є також деякі різновиди цього підходу, наприклад, з еталонами, що змінюються в процесі порівняння.

Підходи, що ґрунтуються на нейронних мережах (Neural Networks Approaches).

Принципи функціонування систем, побудованих на нейронних мережах (іноді їх також називають автоасоціативною пам'яттю), полягають у тому, щоб у відповідь на деяку вхідну сукупність даних, звану "ключом", видати на вихід що зберігається в мережі і найбільш близьку до вхідної за значеннями сукупність такої ж розмірності. У разі розпізнавання облич ключем є зображення обличчя людини. Нейроном називається осередок мережі, яка є найпростішим елементом пам'яті.

Порядок роботи нейронних мереж наступний: 1-й крок – зображення оцифровується та кодується у вигляді вектора; 2-й крок – кожна координата вектора розташовується в окремому осередку, пов'язаному з усіма іншими осередками (навчання або налаштування системи відбувається шляхом зміни ваг зв'язків між осередками); 3-й крок - зображення осіб фільтруються через нейромережу, при цьому вхідне зображення трансформується в найближче запам'ятоване, яке подається на вихід. На даний момент цей підхід є одним із найпопулярніших.

Аналіз оптичних потоків (Optical Flow).

Застосування цього з метою ідентифікації осіб визнано досить ефективним, але дорогим з обчислювальної точки зору і в практичних додатках не використовується. Суть методу наступна: порівнювані зображення A(i) і B(i) перетворюються на багатошарові зрізані піраміди шляхом багаторазового згортання чотирьох сусідніх пікселів в один із середнім арифметичним значенням яскравості.

Після завершення даного процесу на відповідних шарах двох різних пірамід проводять пошук відповідних між собою найкраще груп пікселів. До кожного блоку зображення A(i) визначається вектор усунення.

Цей вектор уточнює зміщення між центрами блоку A(i) і найбільш близьким до нього блоком B(i). Аналогічно будуються вектори для зображення B(i). Аналізуючи вийшли системи векторів, можна дійти невтішного висновку про ступеня схожості порівнюваних зображень.

2.2 Аналіз та вибір методів обробки зображень

Комп'ютерні системи інтерпретують довільні зображення як послідовність пікселів, кожен з цих пікселів має власний набір значень кольору. Пікселі є необробленими базовими блоками зображення. Будь-яке зображення складається з набору пікселів, які задаються кольором або яскравістю світла. При розгляді зображення як двовимірної сітку кожен квадрат представляти один піксель.

Розглянемо зображення з градаціями сірого кольору. При цьому кожен піксель є скалярним значенням, що лежить в межі від 0 до 255. Нульове значення відповідає «чорному» кольору, а 255 – «білому». Значення між 0 і 255 мають різні відтінки сірого кольору; значення, які ближче до 0, є темнішими, а значення, які ближче до 255 - світлішими. Градієнтне зображення в градаціях сірого демонструє темніші пікселі з лівого боку, а світліші з правого. З рис. 2.3 можна зрозуміти те, як значення в градаціях сірого перетворюються на двовимірний масив цілих чисел.

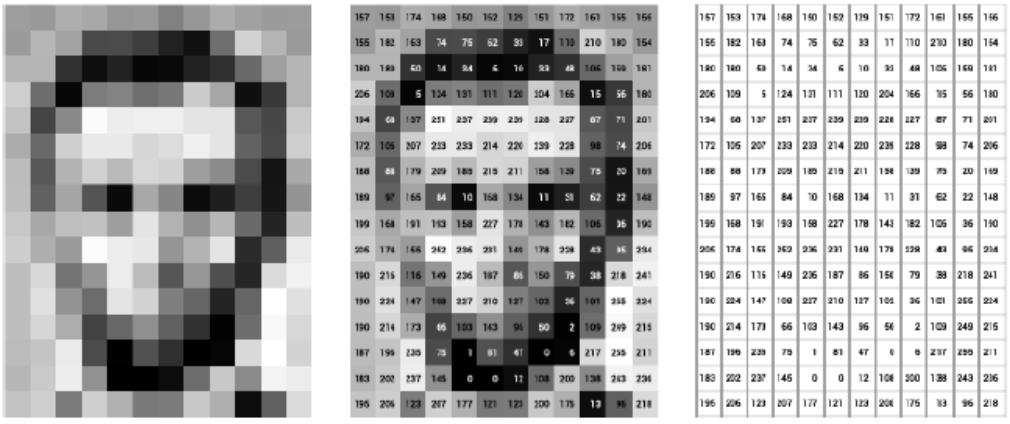

Рисунок 2.3 – Матричне перетворення зображень

При кольоровому зображенні у форматі RGB пікселі не є скалярними значеннями, як було у одноканальних зображеннях у градаціях сірого. Натомість кожен піксель представляється списком із трьох значень: одне значення для Red - компонента червоного, друге для Green - зеленого та третє для Blue - синього. Щоб визначити колір пікселя у зазначеній колірній моделі RGB потрібно підрахувати вимірювану «кількість» червоного, зеленого та синього кольорів для даного пікселя. Усі три канали Red, Green і Blue можуть мати значення в діапазоні [0, 255], тобто всього 256 відтінків. Нульове значення означає відсутність каналу (кольору Red, Green чи Blue), а 255 - це повне представлення відповідного кольору. Враховуючи, що значення будь-якого пікселя можуть бути лише в діапазоні [0, 255], то для представлення яскравості зазвичай використовуються 8-бітові беззнакові цілі числа.

Для задач розпізнавання, ідентифікації та класифікації графічних об'єктів необхідно попередньо обробляти зображення. Враховуючи три значення Red, Green та Blue, їх можна об'єднати у кортеж, який і буде представляти певний колір у просторі RGB.

Розглянемо типові проблеми, що виникають при обробці зображень.

1) Масштаб. Різні зображення мають різний масштаб. Предмети, які ми сприймаємо як однакові, насправді зазвичай займають різну площу різних зображеннях.

2) Положення. Об'єкт, що нас цікавить, може знаходитися в різних місцях зображення.

3) Фон та перешкоди. Предмет, який ми сприймаємо як щось окреме, на зображенні може бути не виділений, і може розташовуватись на тлі інших предметів. Крім того, зображення може бути не ідеальним і може бути схильне до всякого роду спотворень і перешкод.

4) Проекція, обертання та кут огляду. Зображення є лише двовимірною проекцією нашого тривимірного світу. Тому поворот об'єкта та зміна кута огляду кардинальним чином впливають на його двовимірну проекцію – зображення. Один і

той самий об'єкт може давати зовсім різну картинку, залежно від повороту чи відстані до нього.

5) Відсутність зображення. Об'єкт, який нас цікавить, може бути відсутнім на зображенні.

6) Пошкоджений формат. Файл зображення може мати невідповідний формат чи бути пошкодженим.

Перетворення рівня яскравості (Brightness level conversion)

Після зчитування зображення до масиву, до нього можна застосувати різні математичні операції.

Найпростіший приклад – перетворення рівня яскравості напівтонового зображення. Візьмемо довільну функцію f, що відображає інтервал 0...255 (або, за бажанням, 0…1) в себе, тобто область значень збігається з областю визначення.

Іншим прикладом перетворення яскравості є вирівнювання гістограми. Ця операція змінює гістограму яскравості, щоб результуюча гістограма містила всі можливі значення яскравості і при цьому приблизно в однаковій кількості. Вона часто застосовується для нормування яскравості перед подальшою обробкою, а також підвищення контрастності. В даному проекті випадку для перетворення використовується функція розподілу (cumulative distribution function, CDF) значень пікселів у зображенні.

Зниження шуму (noise reduction) – один із способів підвищення якості корисного сигналу на зображенні. Для того, щоб згладити шум застосовують розмиття по Гаусс. Для цього застосовується метод згортки зображень з ядром гауса (наприклад, 3x3, 5x5, 7x7 і т. д. пікселів). Розмір ядра залежить від очікуваного ефекту розмиття. В основному найменше ядро – це менш помітна пляма. При обробці зображення ядро (згортка матриці) є невеликою матрицею. Воно використовується для розмиття, підвищення різкості, виявлення країв та багато іншого.

Розрахунок градієнта (Gradient Calculation) – визначає інтенсивність та напрямок краю шляхом обчислення градієнта зображення з використанням операторів виявлення краю. Градієнт – це векторна величина, що показує напрямок якнайшвидшого зростання двовимірної функції яскравості зображення.

Краї відповідають зміні інтенсивності пікселів. Щоб виявити її, найпростіше застосувати фільтри, які виділяють цю зміну інтенсивності в обох напрямках: горизонтальному (x) та вертикальному (y).

На рис. 2.4 показані початкові зображення, на рис. 2.5 – результат їх обробки.

Рисунок 2.4 – Початкові фронтальні зображення

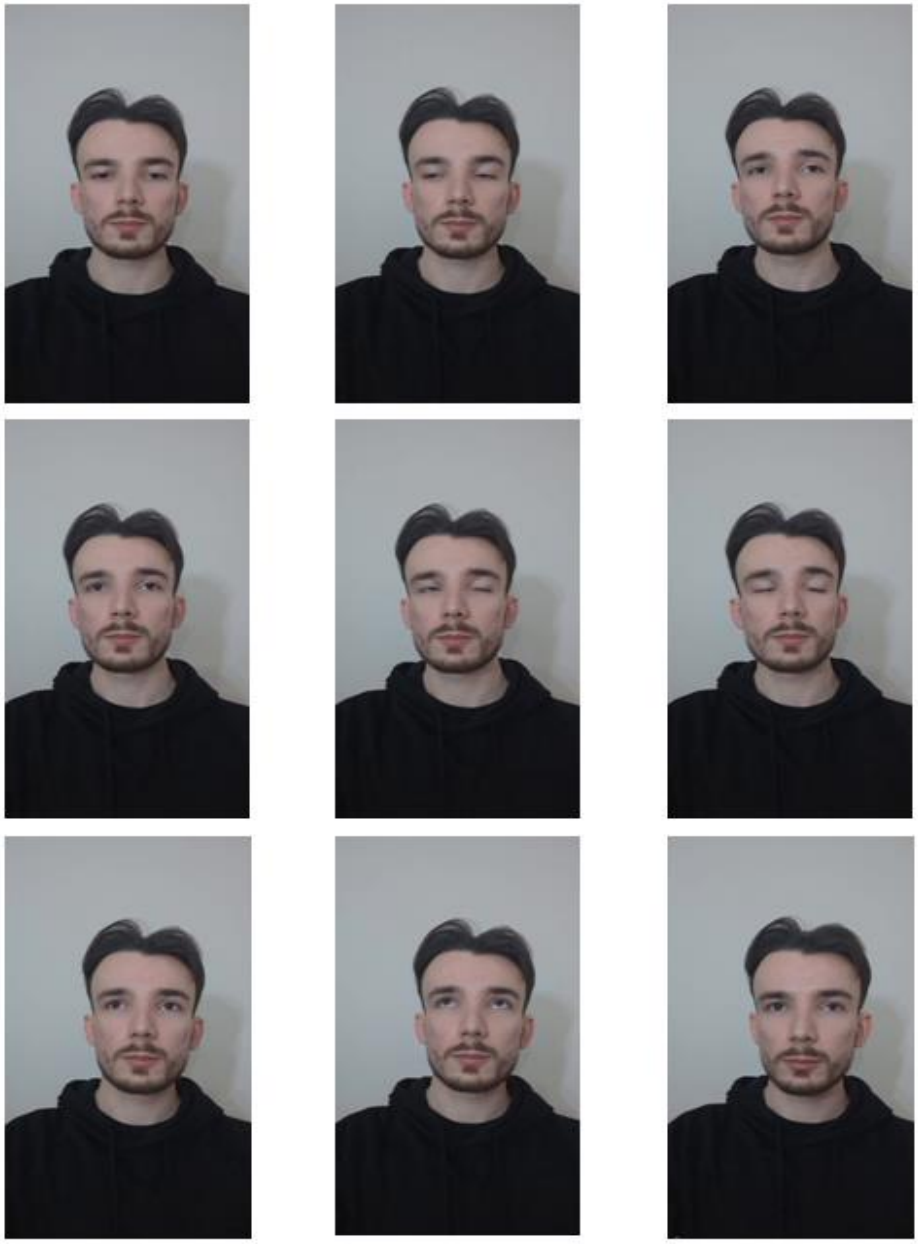

Рисунок 2.5 – Результати обробки фронтальних зображень

Детектор кутів Харріса (Harris angle detector)

Алгоритм виявлення кутів Харріса – один із найпростіших детекторів кутів об'єктів [10]. Ідея полягає в тому, щоб знайти особливі точки, в околиці яких є межі в кількох напрямках, це і є кутові точки. Обчислюється вектор градієнта зображення, який утворює матрицю:

$$M = V * V^T = \begin{bmatrix} V_x \\ V_y \end{bmatrix} * \begin{bmatrix} V_x & V_y \end{bmatrix} = \begin{bmatrix} V_x^2 & V_x V_y \\ V_x V_y & V_y^2 \end{bmatrix}, \quad (2.1)$$

де V – вектор градієнта зображення, V_x та V_y – його похідні.

Відповідно до значень елементів матриці, для кожної точки обчислюються параметри l_1 та l_2 для детекції кутів Харріса:

– якщо обидва параметри мають великі позитивні значення, то в точці є кут;

– якщо l_1 має велике позитивне значення, а наближається до 0, то ситуація з наявністю кута не визначена;

– якщо обидва параметри наближається до 0, то в точці кута немає.

На рис. 2.6 показаний результат роботи алгоритму виділення кутів Харріса. Застосування алгоритму допомагає знизити шумову складову зображення. Він має адаптивний поріг та дозволяє виконувати налаштування задля того, щоб не втратити особливі точки зображення [11].

Рисунок 2.6 – Результат роботи алгоритму виділення кутів Харріса

2.3 Вибір датасету для навчання нейронної мережі

Для навчання нейронної мережі обрано датасет Labeled Faces [12]. Він може бути застосований для перевірки та ідентифікації облич, але, відповідно до ліцензії, не повинен використовуватись для будь-яких комерційних цілей. Як зазначено в описі, датасет призначений щоб допомогти дослідницькому співтовариству досягти успіхів у перевірці обличчя, а не для того, щоб забезпечити ретельну перевірку комерційних алгоритмів перед розгортанням відповідних програмних продуктів.

Набір даних містить понад 13 000 зображень облич, зібраних з Інтернету. Датасет здебільшого представлений особами молодого та середнього віку, 13% з них – це жінки, 87% – чоловіки. Дуже мало дітей, немає немовлят, дуже мало людей старше 80 років. Крім того, багато етносів мають дуже незначне представництво або взагалі відсутні.

Кожне обличчя було позначено іменем зображеної людини. 1680 із зображених людей мають дві або більше різних фотографій у наборі даних. Єдиним обмеженням для цих облич є те, що вони були виявлені детектором облич Віоли-Джонса.

Додаткові умови, такі як погане освітлення, екстремальна поза, сильні оклюзії, низька роздільна здатність та інші важливі фактори, не притаманні даному датасету.

Зараз існує чотири різні набори зображень датасету, включаючи оригінальні та різні типи «вирівняних» зображень. Вирівняні зображення включають «зображення з воронкою» (ICCV 2007) і «зображення з глибокою воронкою» (NIPS 2012). Серед них глибокі воронкоподібні зображення дають кращі результати для більшості алгоритмів перевірки обличчя порівняно з оригінальними зображеннями та воронкоподібними зображеннями (ICCV 2007).

Звичайна перевірка обличчя стосується головним чином великих внутрішньокласових варіацій, таких як поза, освітленість і вираз обличчя. Після перевірки баз даних датасету можна визначити головний обмежуючий фактор для невимушеного завдання перевірки обличчя: верифікація за своєю природою є

проблемою, у якій багато прикладів є дуже простими з великою дисперсією між класами, оскільки збір бази даних датасету базується на припущенні випадкової атаки самозванця. Однак для практичного використання це ймовірно, що відчайдушний самозванець може спробувати підробити справжнього користувача, шукаючи схожих на вигляд людей. Щоб імітувати цю навмисну фальсифікаційну атаку, при створенні бази даних датасету навмисно додано 3000 пар схожих облич у папках оригінальних зображень за допомогою людського краудсорсингу.

Також датасет доповнений понад 1000 зображеннями людей у повний зріст задля можливості проведення ідентифікації за соматотипом. 16% з них – це жінки, 84% – чоловіки. Діти молодше 10 років відсутні.

2.4 Висновки до розділу

У другому розділі проведено класифікацію задач аналізу зображень, а також підходів та методів біометричної ідентифікації.

Проаналізовано найбільш розповсюджені методи обробки зображень. Реалізовано методи перетворення рівня яскравості, зниження шуму та розрахунку градієнта для покращення якості зображення. Застосовано ці методи для підвищення якості набору фронтальних зображень. Розглянуто та опрацьовано алгоритм виявлення кутів Харріса для додаткового зниження шумової складової та підвищення якості подальшої ідентифікації зображень.

Обрано датасет для навчання нейронної мережі, яка в подальшому буде виконувати біометричну ідентифікацію зображень.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ МАШИННОГО ЗОРУ ДЛЯ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

3.1 Варіанти використання для клієнтської частини системи

Слід зазначити, що у переважної більшості функціоналу задіяні як клієнтська, так і серверна складові. Для наочності специфікації функціональних вимог сформуємо UML діаграму використання розроблюваної системи (рис. 3.1) та покажем на описах прецедентів специфіку клієнтської частини.

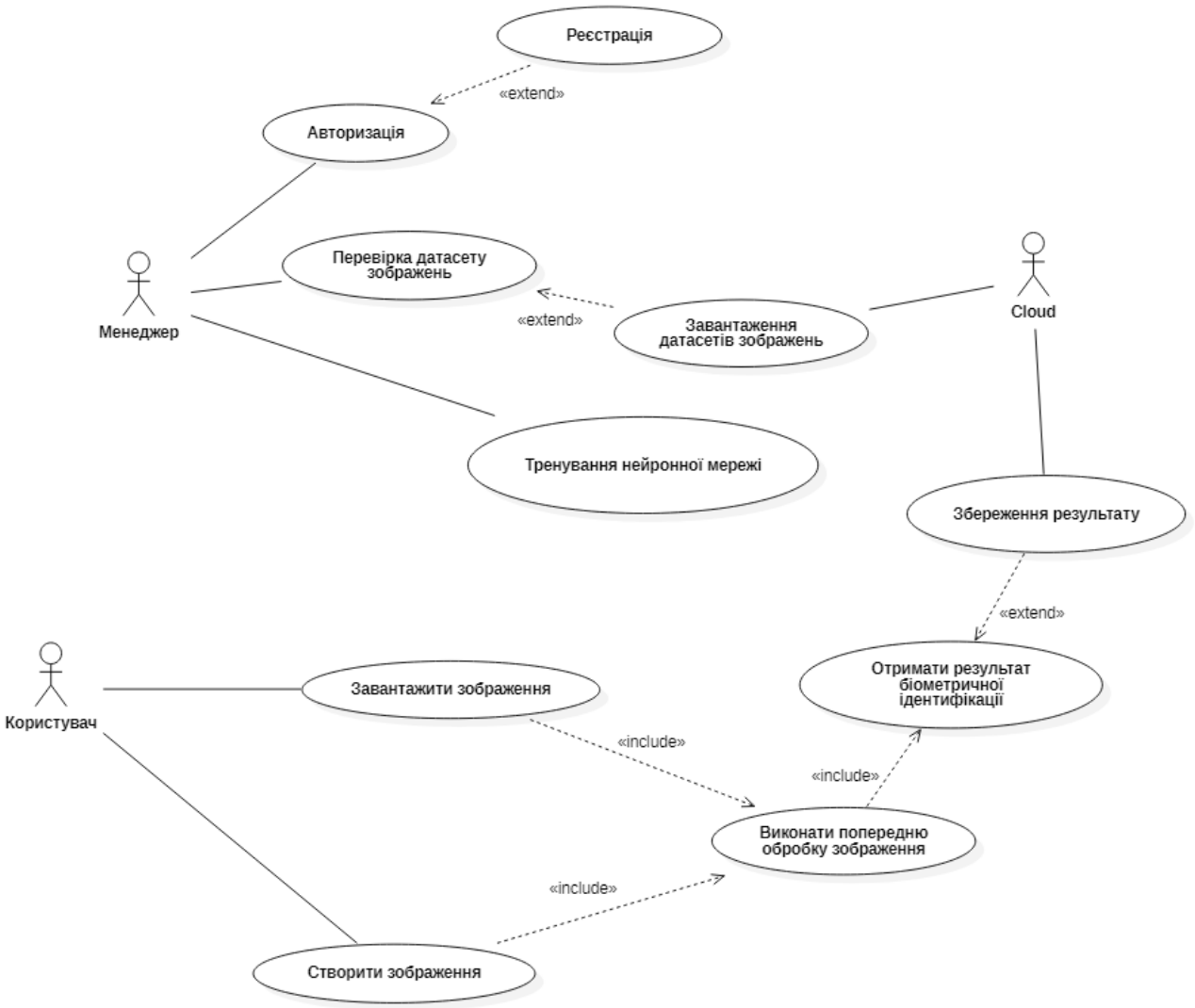

Рисунок 3.1– Діаграма UseCase розроблюваної системи для клієнтської частини

Кожен прецедент повинен мати гарантію успіху, тригер, передумову, основний сценарій та розширення.

Прецедент: Реєстрація.

Гарантія успіху: менеджер був зареєстрований в системі машинного зору.

Тригер: система зафіксувала вхід нового менеджера у систему машинного зору.

Передумова: менеджер потрапив до форми реєстрації.

Основний сценарій:

1) менеджер вводить у відповідні поля форми електронну адресу та пароль;
2) система машинного зору передає дані на серверну частину для перевірки;
3) клієнтська частина отримує результати перевірки;
4) система виводить повідомлення менеджеру, що він є зареєстрованим.

Розширення:

а *. Помилки у реєстрації нового менеджера.

а1. Менеджер не ввів логін та пароль у форму.

а2.1. Система машинного зору повідомляє менеджера про помилку при реєстрації, просить перевірити справність Інтернет підключення та повторно ввести дані.

а2.2. При повторному вводі даних система машинного зору знову передає введені дані на сервер з метою реєстрації нового менеджера.

Прецедент: Авторизація.

Гарантія успіху: менеджер був авторизований в системі машинного зору.

Тригер: система машинного зору зафіксувала вхід менеджера у систему та спробу авторизуватись.

Передумова: менеджер потрапив до форми реєстрації.

Основний сценарій:

1) менеджер вводить у відповідні поля електронну адресу та пароль;

2) система машинного зору передає дані на сервер;
3) сервер підтверджує коректність даних та надає відповідний акаунт;
4) менеджера повідомляють, що він авторизований, та надають йому доступ до акаунту.

Розширення:

а *. Збій авторизації менеджера.

а1. Система виводить повідомлення про наявність помилки.

а2.1 Сервер повертає тип помилки.

а2.2 Повідомлення про тип помилки виводиться на екран менеджеру.

а2.3 Система знову надає форму авторизації та пропонує ввести дані.

а2.4. Система знову передає дані на сервер з метою авторизації менеджера.

Примітка. Припустимі типи помилок:

- логін не відповідає формату;
- пароль не відповідає формату;
- введені логін та пароль вже існують;
- немає зв’язку з репозиторієм.

Прецедент: Завантаження датасету зображень.

Гарантія успіху: датасет доступний менеджеру.

Тригер: менеджер бажає завантажити датасет.

Основний сценарій:

1) клієнтська частина звертається до серверу за переліком датасетів;
2) сервер повертає список, клієнтська частина виводить його на форму;
3) менеджер переглядає список та обирає потрібний датасет;
4) клієнтська частина передає запит на сервер;
5) клієнтська частина надає користувачеві пул зображень з завантаженого датасету.

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку завантаження датасету.

а2.1Система пропонує перевірити справність Інтернет підключення та спробувати завантажити датасет ще раз.

а2.2 При повторній спробі клієнтська частина знову намагається звернутись до серверу.

Прецедент: Перевірка датасету зображень.

Гарантія успіху: інформація щодо датасету була відображена на екрані.

Передумова: менеджер потрапив до форми з інформацією по датасету.

Основний сценарій:

1) клієнтська частина системи надсилає запит до сервера для перевірки датасету;
2) клієнтська частина системи отримує дані;
3) отримані дані відображаються на екрані.

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку передачі запиту до сервера для перевірки датасету.

а2.1 Система пропонує перевірити справність Інтернет підключення та спробувати надіслати запит до сервера для перевірки датасету ще раз.

а2.2 При повторній спробі клієнтська частина знову намагається звернутись до серверу.

Прецедент: Тренування нейронної мережі.

Гарантія успіху: тренування нейромережі проведено.

Передумова: менеджер обрав датасет для тренування.

Основний сценарій:

1) клієнтська частина системи надсилає менеджеру форму з елементами управління для запуску тренування;
2) менеджер запускає процес;
3) клієнтська частина отримує від серверу дані щодо оцінки якості валідації зображень та надає їх менеджеру.

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку роботи серверу. Дані щодо оцінки якості валідації зображень відсутні.

а2.1 Система пропонує перевірити справність Інтернет підключення та спробувати отримати відповідь від серверу ще раз.

а2.2 При повторній спробі клієнтська частина знову намагається звернутись до серверу.

Прецедент: Завантажити зображення.

Гарантія успіху: зображення для проведення біометричної ідентифікації завантажено.

Передумова: користувач має натреновану нейронну мережу.

Основний сценарій:

1) клієнтська частина системи надає користувачеві форму для завантаження зображення;
2) користувач завантажує потрібне зображення;
3) клієнтська частина пропонує користувачеві виконати попередню обробку зображення:
4) користувач погоджується з обробкою;
5) клієнтська частина робить запит на сервер стосовно обробки, сервер опрацьовує прецедент «Виконати попередню обробку зображення».

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку роботи серверу. Дані щодо оцінки якості валідації зображень відсутні.

а2.1 Система пропонує перевірити справність Інтернет підключення та спробувати отримати відповідь від серверу ще раз.

а2.2 При повторній спробі клієнтська частина знову намагається звернутись до серверу.

Прецедент: Створити зображення.

Гарантія успіху: зображення для проведення біометричної ідентифікації завантажено.

Передумова: користувач має натреновану нейронну мережу.

Основний сценарій:

1) система отримує доступ до камери, користувач робить фото;
2) клієнтська частина пропонує користувачеві виконати попередню обробку зображення:
3) користувач погоджується з обробкою;
4) клієнтська частина робить запит на сервер стосовно обробки, сервер опрацьовує прецедент «Виконати попередню обробку зображення».

Розширення:

а *. Не працює камера.

а1. Система повідомляє про помилку роботи, ідентифікація неможлива.

б *. Збій звернення до серверу:

б1. Система повідомляє про помилку роботи серверу. Дані щодо оцінки якості валідації зображень відсутні.

б2.1 Система пропонує перевірити справність Інтернет підключення та спробувати отримати відповідь від серверу ще раз.

б2.2 При повторній спробі клієнтська частина знову намагається звернутись до серверу.

Прецедент: Виконати попередню обробку зображення.

Гарантія успіху: зображення оброблено.

Передумова: користувач обрав зображення.

Основний сценарій:

1) клієнтська частина системи передає серверу зображення;
2) сервер відпрацьовує прецедент «Отримати результат біометричної ідентифікації»;
3) сервер повертає зображення з покращеними для ідентифікації характеристиками та результатами ідентифікації.

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку роботи серверу. Дані щодо результатів обробки зображення відсутні.

Прецедент: Отримати результат біометричної ідентифікації.

Гарантія успіху: користувач отримав результат ідентифікації.

Передумова: користувач обрав зображення.

Основний сценарій:

1) сервер виконує біометричну ідентифікацію за зображенням обличчя чи геометрії тіла людини;
2) сервер повертає клієнтській частині системи результат ідентифікації зображення;
3) клієнтська частина виводить результат на форму.

Розширення:

а *. Збій звернення до серверу:

а1. Система повідомляє про помилку роботи серверу. Дані щодо результатів біометричної ідентифікації відсутні.

Прецедент: Збереження результату.

Гарантія успіху: результат ідентифікації збережений в Cloud.

Передумова: користувач отримав результат ідентифікації.

Основний сценарій:

1) користувач натискає на формі кнопку для збереження;
2) якщо користувач не є авторизованим у хмарному середовищі, система надає йому форму для авторизації, та користувач вводить логін та пароль;
3) клієнтська частина системи надсилає запит до сервера для збереження результату;
4) користувач отримує повідомлення про збереження.

Розширення:

а *. Користувач не має жодного аккаунту у хмарних середовищах.

а1. Система пропонує зберегти результат локально на мобільний пристрій.

б *. Збій звернення до серверу:

б1. Система повідомляє про помилку передачі запиту до сервера для збереження результату.

б2.1 Система пропонує перевірити справність Інтернет підключення та спробувати надіслати запит до сервера для збереження результату ще раз.

3.2 Нефункціональні вимоги для клієнтської частини

Якісні вимоги для клієнтської складової системи:

а) Надійність

Якщо менеджер видаляє зображення, то система реагує на це не більш ніж через 2 секунди.

Якщо менеджер запрошує список зображень датасету, то система реагує на це не більш ніж через 3 секунди.

Якщо менеджер запрошує список датасетів, то система реагує на це не більш ніж через 5 секунд.

Якщо менеджер чи звичайний користувач почав виконання біометричної ідентифікації, то система надає результат не більш ніж через 1 хвилину.

Якщо менеджер запрошує перегляд набору сеанів тренування нейромережі, то система реагує на це не більш ніж за 2 секунди.

б) Функціональність

Якщо система машинного зору виконує запит на видалення зображення, то клієнт отримує результат від сервера не менш ніж у 95% випадків.

Якщо система машинного зору виконує запит на отримання списку зображень, то ймовірність повернення списку зображень на форму інтерфейса складає 0,9.

Якщо система машинного зору виконує запит на отримання списку зображень, то клієнт отримує результат від сервера у цілісності не менш ніж у 90% випадків.

Якщо система машинного зору виконує запит на тренування нейромережі, то клієнт отримає невизначений результат не більш ніж в 1 випадку зі 100.

в) Супроводжуваність

Якщо у систему машинного зору треба запровадити новий функціонал, то він почне працювати не більш ніж через 1,5 місяці.

Якщо з системи машинного зору треба видалити застарілу частину функціоналу, то зміни вступять в силу не більш ніж через 1 тиждень.

Якщо у системі машинного зору треба змінити частину функціоналу, то то зміни вступять в силу не більш ніж через 3-4 тижня.

г) Портативність

Якщо систему машинного зору треба буде перенести на іншу операційну систему (Linux, IOS, MacOS), то процес переносу для базової версії триватиме не більш ніж 1 місяць, а для повної версії – не більше ніж 3 місяці.

Якщо систему треба встановити на Андроід-пристрій, то процес встановлення триватиме не більш ніж 3 хвилини.

3.3 Висновки до розділу

У даному розділі створена та описана специфікація вимог до клієнтської частини системи машинного зору для біометричної ідентифікації. Створена діаграма варіантів використання системи, яка є практично загальною для серверної та клієнтської частин. При описі прецедентів формалізовані гарантії успіху, тригери, передумови, основні сценарії та розширення саме для клієнтської частини.

Для комплексного представлення та врахування всіх вимог до системи були розглянуті нефункціональні вимоги для атрибутів надійності, функціональності, супроводжуваності та портативності.

4 ПРОЕКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

4.1 Проектування архітектури системи

На рис. 4.1 наведена загальна клієнт-серверна архітектура з підтримкою локальної бази даних та хмарного середовища Cloud. У даніх роботі опрацьовані клієнтська частина та локальна база даних, що виділені сірим кольором. Додатково реалізовані механізми обробки зображень, що увійшли у серверну складову.

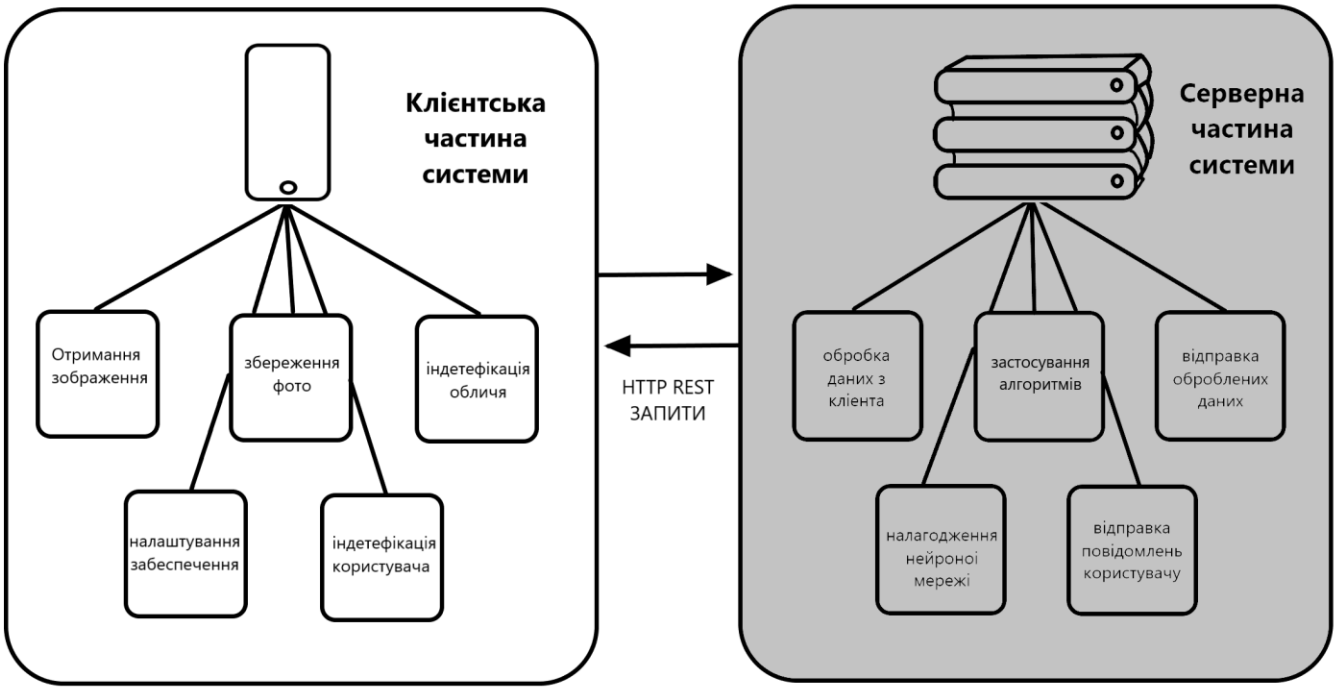

Рисунок 4.1– Архітектура програмного забезпечення

«Функції, які реалізуються на сервері:

– зберігання, доступ, захист і резервне копіювання даних;
– обробка клієнтського запиту;
– відправлення результату (відповіді) клієнту.

Функції, які реалізуються на стороні клієнта:

– надання користувальницького інтерфейсу;

– формулювання запиту до сервера і його відправка;

– отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові.» [13]

Збереження даних може відбуватись як у хмарному середовищі, так і к локальній пам’яті мобільного пристрою.

Основні переваги такої архітектури наступні:

– Масштабованість (система дозволяє будувати складні мережеві конфігурації з багатьма сотнями користувачів);

– Високий рівень безпеки (безпека даних значною мірою залежить від завдання програми);

– Висока надійність (система має вбудований механізм роботи з транзакціями, включаючи їх відкат);

– Настроюваність (система здатна переналаштовуватися при виникненні збоїв);

– Заміна або модифікація програмного забезпечення на будь-якому рівні можлива без впливу на інші рівні.

Для проектування архітектури програмної системи доцільно використовувати шаблони проектування. Вони вважаються добре сформованою мовою для представлення дизайну програмного забезпечення. Широко відомою є класифікація шаблонів проектування на сімейства шаблонів, для кожного з яких створено набір графічних нотацій, їх різновиди та еволюції.

Шаблон проектування — це чітко визначене рішення повторюваної проблеми. З роками кількість шаблонів і доменів шаблонів проектування розширилася, оскільки шаблони є досвідом експертів у цій області, зафіксованим у абстракції вищого рівня. Це спонукало інших працювати над мовами для шаблонів проектування, щоб систематично документувати абстракції, деталізовані в шаблоні

проектування, а не фіксувати алгоритми та дані. Ці мови специфікації шаблонів проектування мають різні варіанти, націлені на різні аспекти шаблонів проектування.

Для звертання до великої кількості зображень використаний поведінковий шаблон Ітератор (Iterator). Він дозволяє надавати послідовний доступ до елементів множини (зображень) незалежно від внутрішньої організації датасету (рис. 4.2).

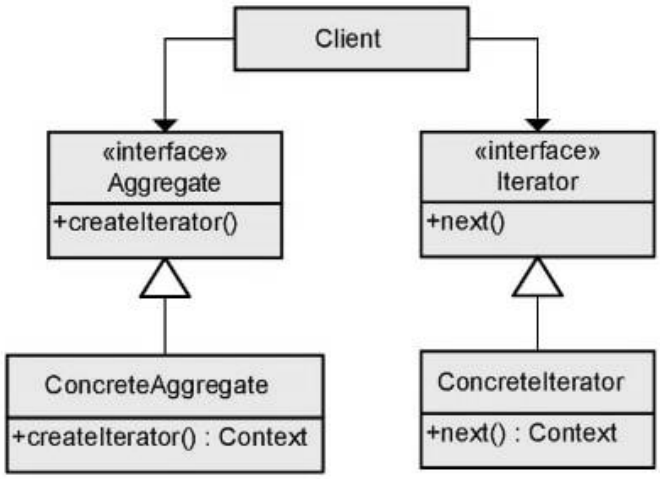

Рисунок 4.2 – Шаблон проектування Ітератор

Шаблон Iterator є, мабуть, найвідомішим із так званих шаблонів проектування Gang of Four, який «забезпечує спосіб послідовного доступу до елементів сукупного об’єкта без розкриття його основного представлення» [14]. Це досягається шляхом визначення інтерфейсу Iterator (наприклад, представлення операцій для ініціалізації ітерації, доступу до поточного елемента, переходу до наступного елемента та перевірки завершення), який очікується реалізація об’єктів колекції — можливо, опосередковано, через підоб’єкт.

Саме такий інтерфейс був включений у бібліотеки найбільш відомих мов програмування з моменту їх створення. Це допомагає уникнути необхідності писати шаблонний код для керування ітерацією елементів колекції; це робить код чистішим і простішим.

4.2 Структури даних системи клієнтської частини системи

Розробимо реляційну модель для збереження даних системи машинного зору для того. На рис. 4.3 продемонстровано концептуальне зображення бази даних. За допомогою даної моделі виділимо ключові сутності і позначимо міжтабличні зв'язки.

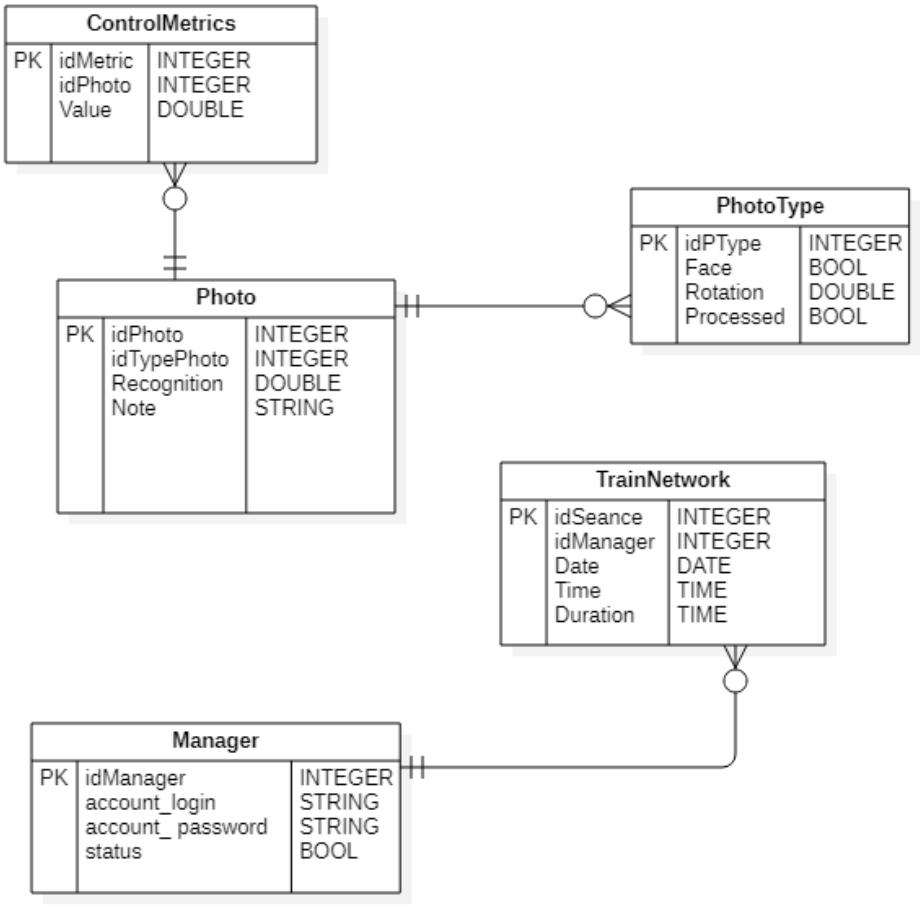

Рисунок 4.3– Реляційна модель бази даних

Для системи були обрані наступні сутності:

Photo – зображення, за якими системою виконується біометрична ідентифікація;

PhotoType – тип та деякі характеристики цього зображення;

ControlMetrics – метрики, відповідно до яких у найпростішому випадку виконується ідентифікація. У більш складному випадку додається множина особливих точок для розпізнавання, значення яких не зберігаються у базі, тобто зберігається тільки результат їх опрацювання;

Manager – людина, що навчає нейромережі на датасетах та повинна мати права доступу до цього функціоналу;

TrainNetwork – реляційна сутність, яка містить допоміжні дані стосовно нейронної мережі.

Визначимо більш детально усі сутності цієї схеми та визначимо їх призначення та типи даних.

1. Photo – зображення, за якими системою виконується біометрична ідентифікація.

Має наступні поля:

1) унікальний цифровий ідентифікатор зображення – INTEGER;
2) ідентифікатор типу фотографії – INTEGER;
3) точність розпізнавання відповідно до певної людини – DOUBLE;
4) загальні примітки у разі потреби – STRING.

Кількість записів в таблиці відповідає об’єму датасету.

2. PhotoType – тип та деякі характеристики зображення. Має наступні поля:

1) унікальний цифровий ідентифікатор типу – INTEGER;
2) флаг, який визначає чи є фото зображенням обличчя, або зображенням всієї фігури цілком (для біометричної ідентифікації за соматотипом людини) – BOOL (false – зображення фігури цілком, true – зображення обличчя);
3) кут повороту людини на фото – DOUBLE. Вимірюється відносно «нульової» позиції, коли людина дивиться прямо в камеру;

4) флаг, який показує чи було зображення вже оброблено з метою покращення якості для збільшення достовірності подальшої ідентифікації – BOOL (false – зображення не оброблялось, true – зображення є результатом обробки).

Сутність використовується для того, щоб обирати зображення з певними характеристиками для навчання нейронної мережі.

Кількість записів в таблиці віддзеркалює різноманітність використовуваних зображень.

3. ControlMetrics – метрики для найпростішої ідентифікації.
   1) унікальний цифровий ідентифікатор метрики – INTEGER;
   2) ідентифікатор зображення, до якого виміряна метрика – INTEGER;
   3) значення метрики для визначеного зображення – DOUBLE.

Так як метрики фактично є відстанями між особливими точками, тип значення – дійсний числовий.

Кількість записів в таблиці відповідає кількості метрик, при розширенні цієї кількості буде збільшуватись кількість рядків в таблиці.

4. Manager – сутність, яка є представленням в системі людини, що навчає нейромережі. У загальному випадку один менеджер може навчати одну чи декілька нейромережі на одному чи на декількох датасетах.

Має наступні поля:

1) унікальний цифровий ідентифікатор користувача – INTEGER;
2) логін для доступу до аккаунту – STRING;
3) пароль для доступу до аккаунту – STRING;
4) статус авторизації в системі – BOOL.

Логін та пароль повинні містити від 6 до 20 символів. Пароль повинен включати хоча б 1 прописну букву, хоча б 1 строкову букву та хоча б 1 цифру.

Логін та пароль не повинні співпадати. Статус авторизації є true при успішній авторизації.

5. TrainNetwork – реляційна сутність с даними по тренуванню нейронної мережі.

Має наступні поля:

1) унікальний цифровий ідентифікатор сеансу тренування – INTEGER;
2) унікальний цифровий ідентифікатор користувача – INTEGER;
3) дата початку сеансу тренування нейромережі – DATE;
4) час початку сеансу тренування нейромережі – TIME;
5) тривалість тренування нейромережі – TIME.

4.3 Проектування інтерфейсу застосування

Дизайн, орієнтований на користувача, чітко рекомендує в життєвому циклі розробки інтерфейсу користувача (UI) певний етап, на якому може бути створений прототип інтерфейсу користувача на основі вхідних даних зацікавлених сторін майбутньої системи: дизайнерів, розробників, спеціалістів з зручності використання, графічних експертів і кінцевих користувачів. Коли приходить час висловити та зібрати вимоги користувачів, ці зацікавлені сторони можуть висловлювати свої ідеї дуже різними способами. Отже, це означає, що стадія прототипування повинна включати всі ці типи вхідних даних та інтегрувати їх в єдиний дизайн.

Найбільш зручні інструменти для підтримки дизайну інтерфейсу користувача дозволяють створювати ескізи форм інтерфейсу, оскільки ескіз, ймовірно, представляє найбільшу природний спосіб передачі ідей для людини.

Для розробки зрозумілого та якісного інтерфейсу користувача Андроїд-застосунку спочатку розроблено прототипи інтерфейсу [15, 16]. Для створення прототипу основних графічних форм мобільного застосунку системи машинного зору використано інструмент Mobile App Wireframe.

Каркасний шаблон мобільного застосування доповнює розрив між початковою ідеєю та готовим продуктом перед впровадженням.

На рис. 4.4 показано прототип форми входу до системи, створення акаунту менеджера та його подальший доступний функціонал. Рисунок 4.5 демонструє роботу з датасетом.

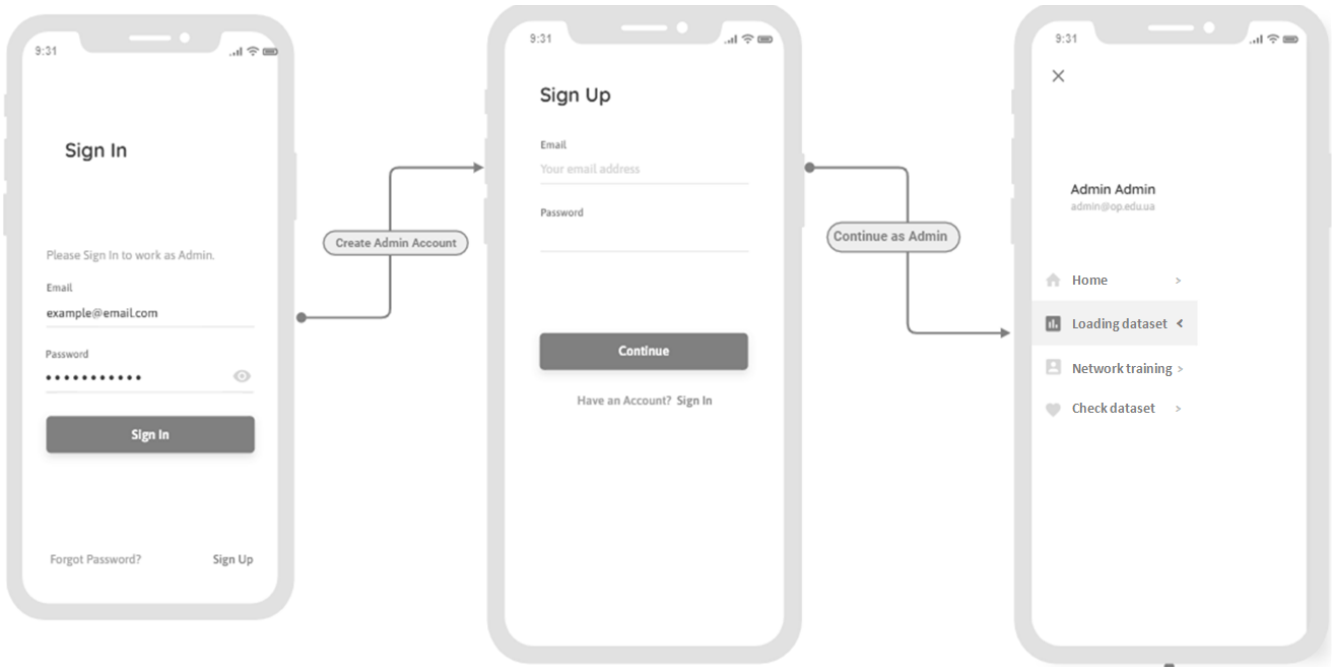

Рисунок 4.4 – Прототипування створення акаунту менеджера

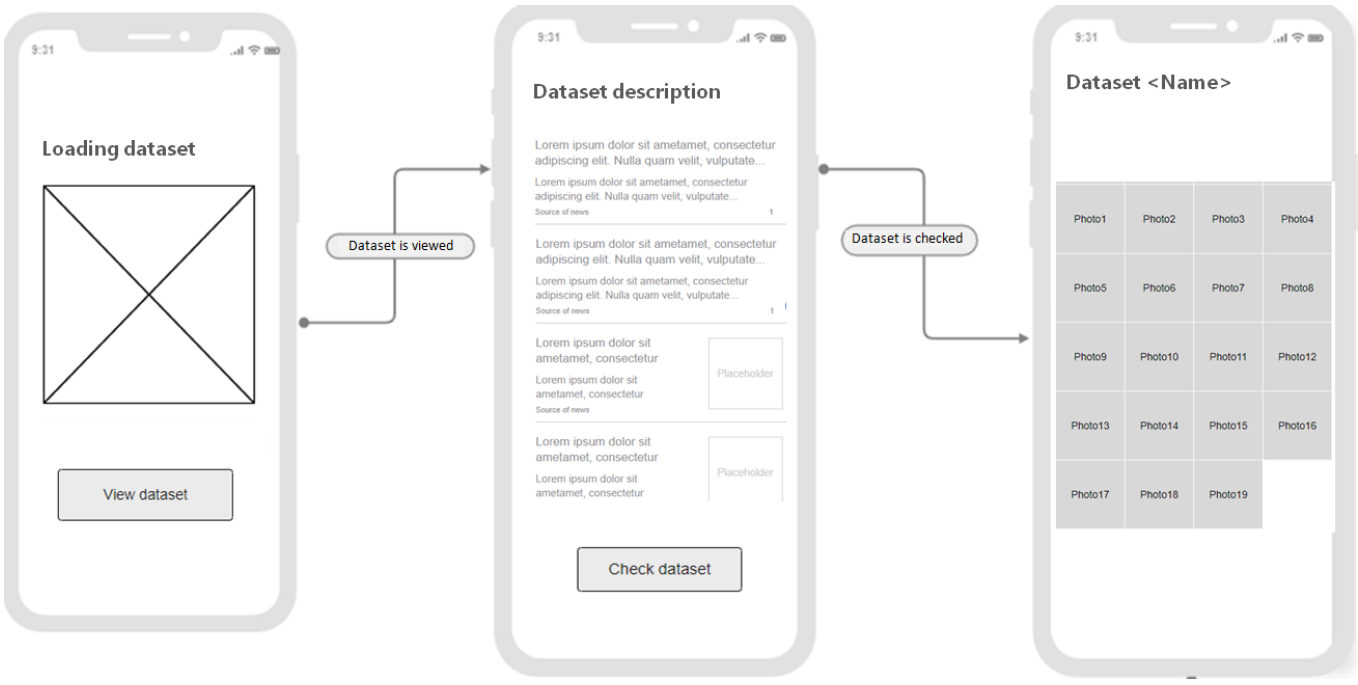

Рисунок 4.5 – Прототипування роботи з датасетом

На рис. 4.6 показано навчання (тренування) нейромережі після вибору певного датасету.

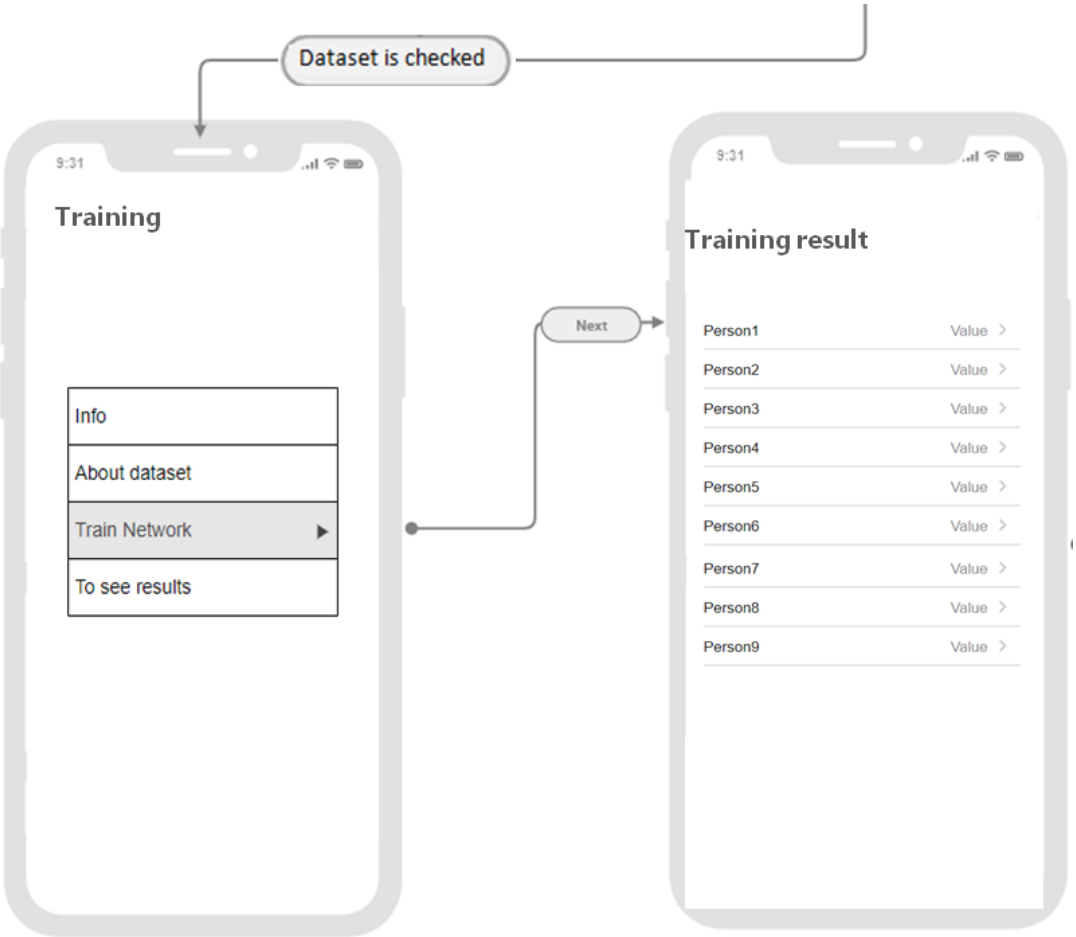

Рисунок 4.6 – Прототипування тренування нейромережі

4.4 Алгоритми взаємодії з інтерфейсом

На ринку з’являється все більше і більше комп’ютерних пристроїв, кожен з яких має дуже різні характеристики щодо потужності ЦП, розміру дисплея, пам’яті тощо. Ця ситуація призвела до відновлення світових досліджень щодо адаптації інтерфейсу користувача (UI), що визначається як здатність інтерфейсу користувача надавати доступ до повного функціоналу, зберігаючи попередньо визначені властивості зручності використання.

На рис. 4.7 наведений алгоритм взаємодії менеджера з інтерфейсом при навчанні нейронної мережі.

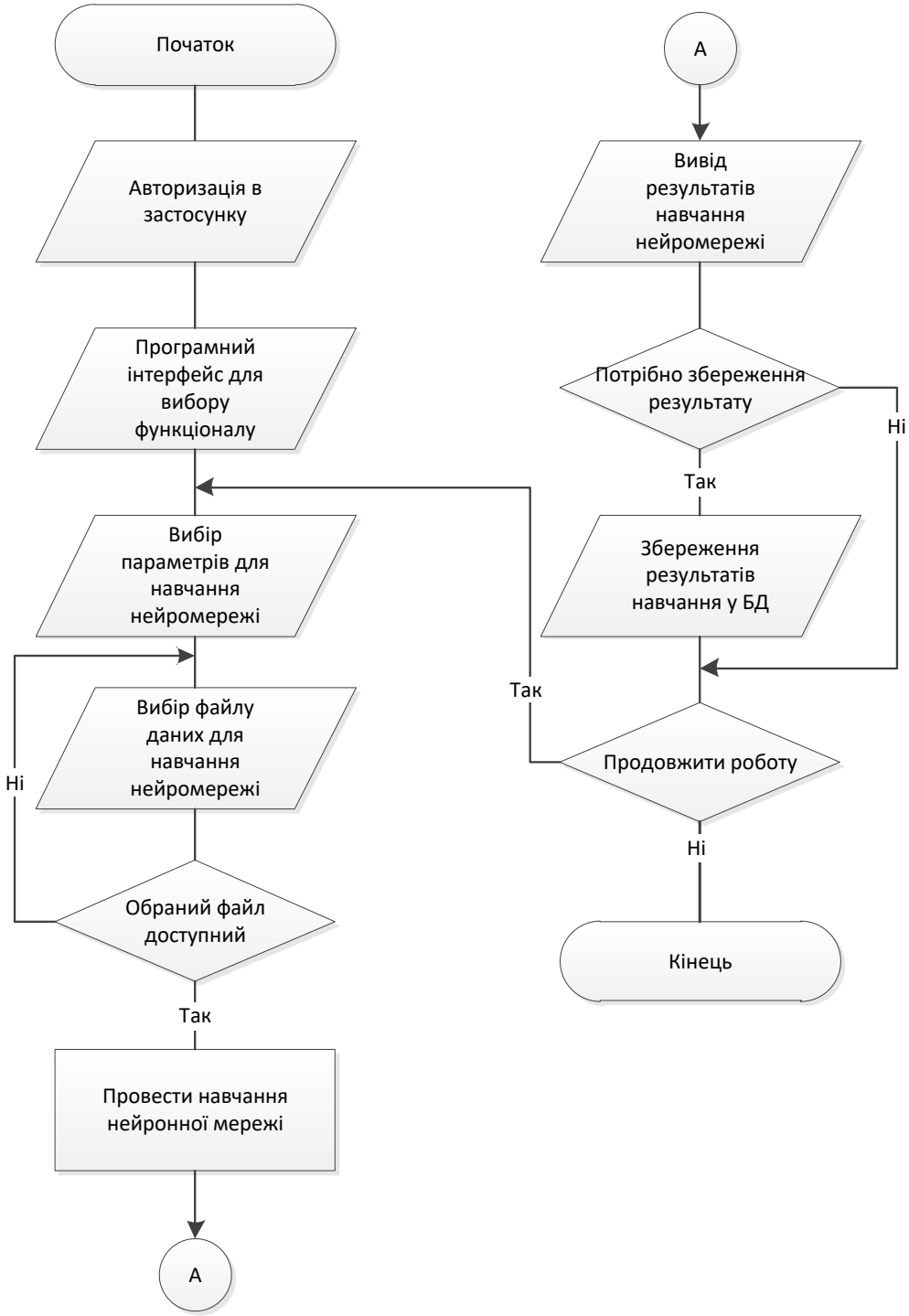

Рисунок 4.6 – Схема алгоритму взаємодії з інтерфейсом при навчанні нейронної мережі

Відповідно до алгоритму, спочатку виконується автоматизація в програмній системі, після чого система надає менеджеру відповідний функціонал.

Менеджер задає параметри для навчання нейромережі та обирає файл з даними (датасет). Якщо дані в файлі доступні та непошкоджені, починається процес навчання.

Отримані результати навчання виводяться на екран, при необхідності менеджер може зберегти їх у БД.

Якщо потрібно продовжити роботу, менеджер може обрати нові параметри та датасет для подальшого навчання нейромережі.

Після того, як нейромережа є натренованою, її може використовувати звичайний користувач для проведення ідентифікації.

На рис. 4.8 наведений алгоритм взаємодії звичайного користувача з інтерфейсом при проведенні біометричної ідентифікації.

Авторизація користувача необхідна лише у тому випадку, коли він бажає зберігати дані у хмарному сховищі, прив’язаному до його акаунту, в іншому випадку такої необхідності немає. Після надання системою функціоналу, користувач обирає (чи робить) зображення для ідентифікації та обирає нейромережі.

Процес біометричної ідентифікації може пройти вдало чи ні. Після проведення вдалої біометричної ідентифікації користувач може зберегти отриманий результат на Cloud чи в локальну базу.

4.4 Проектування програмних класів

Уніфікована мова моделювання (UML) включає різні типи діаграм, які допомагають ефективно вивчати, аналізувати, документувати, проектувати або розробляти будь-яке програмне забезпечення. Тому UML-діаграми є великою перевагою для дослідників, розробників програмного забезпечення та академіків.

Діаграми класів є найбільш широко використовуваними діаграмами UML для цієї мети.

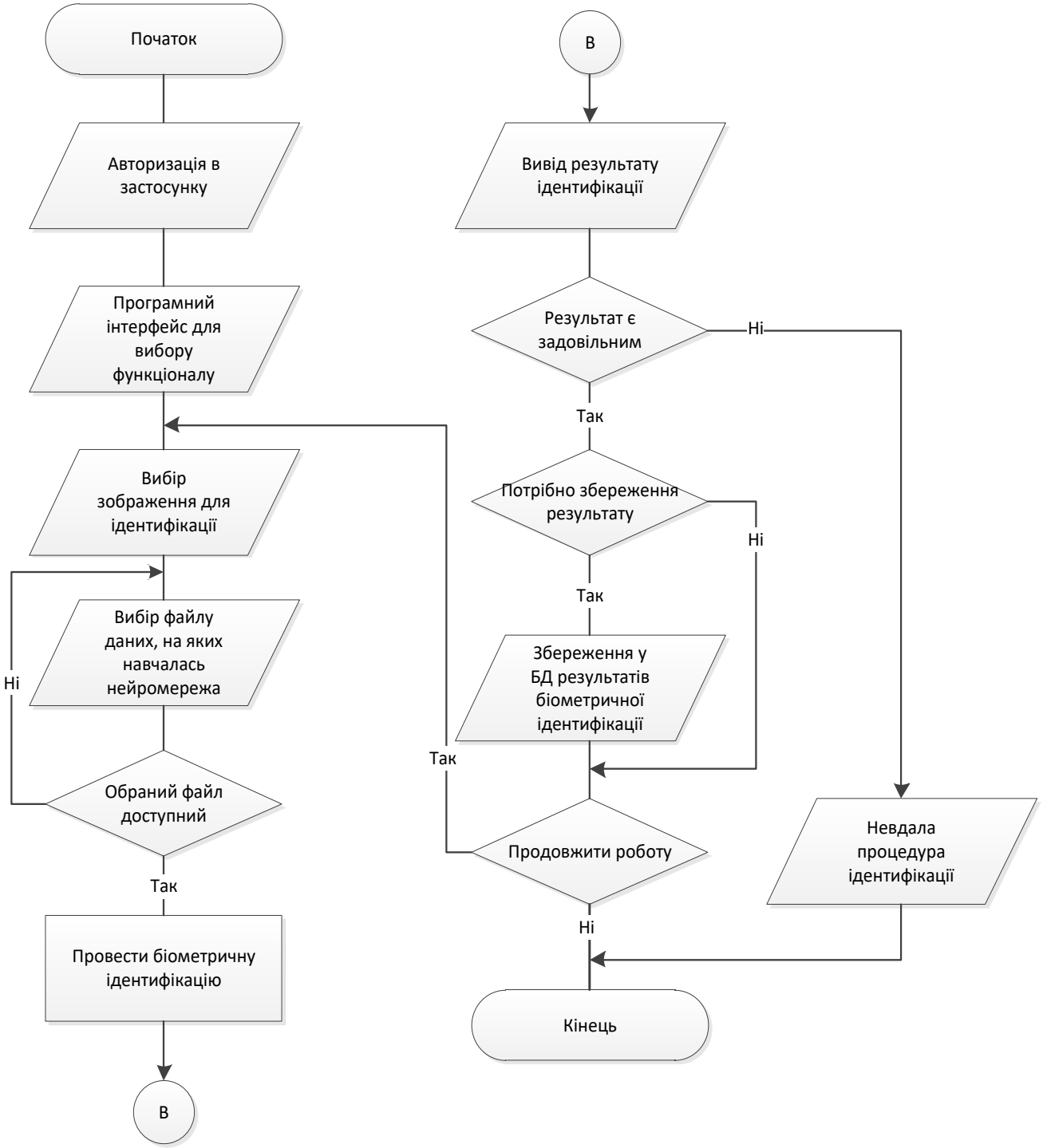

Рисунок 4.7 – Схема алгоритму взаємодії з інтерфейсом при проведенні біометричної ідентифікації

В даний час уніфікована мова моделювання UML є галузевим стандартом для моделювання програмних систем. Діаграма класів мови UML «відображає онтологію домену і за змістом еквівалентна інформаційній моделі за методом С. Шлеєр та С. Меллора: визначається склад класів об’єктів як базових абстракцій та їхні взаємовідносини.

Причому нотація для опису класів забезпечує відокремлення опису функцій від опису даних, застосування принципів інкапсуляції і наслідування даних.

Діаграма має вигляд символів класів – так званих іконок та зв’язків між ними. Терміном іконка позначають стандартизоване, фіксованої форми, візуальне зображення. Іконка класу має форму прямокутника, який може поділятися на дві або три частини. Верхня його частина обов’язкова, вона містить ім’я класу. Друга й третя частини прямокутника можуть наводитися або пропускатися і містять: друга − список атрибутів класу, третя – список операцій класу.» [17]

На рис. 4.8 наведена діаграма програмних класів клієнтської частини системи машинного зору. Вона містить декілька класів, розглянемо їх та відповідні їм атрибути та методи більш детально.

Клас App є стартовим класом системи, з його допомогою починаються процеси запуска системи машинного зору.

Атрибути та методи класу App:

start(): status – запуск процесів системи, ініціалізація змінних та станів системи. Повертається статус запуска, який включає код статусу та його пояснення;

quit(): status – безпечний вихід з системи, повертається статус виходу;

linkToDB(url): bool – підключення до бази даних з датасетами, повертається true у разі успіху.

Клас DBBuilder відповідає за зв’язок системи з репозиторієм.

Атрибути та методи класу DBBuilder:

urlDB: string – адреса доступу до даних датасетів;

build(): DB – підключення бази до системи.

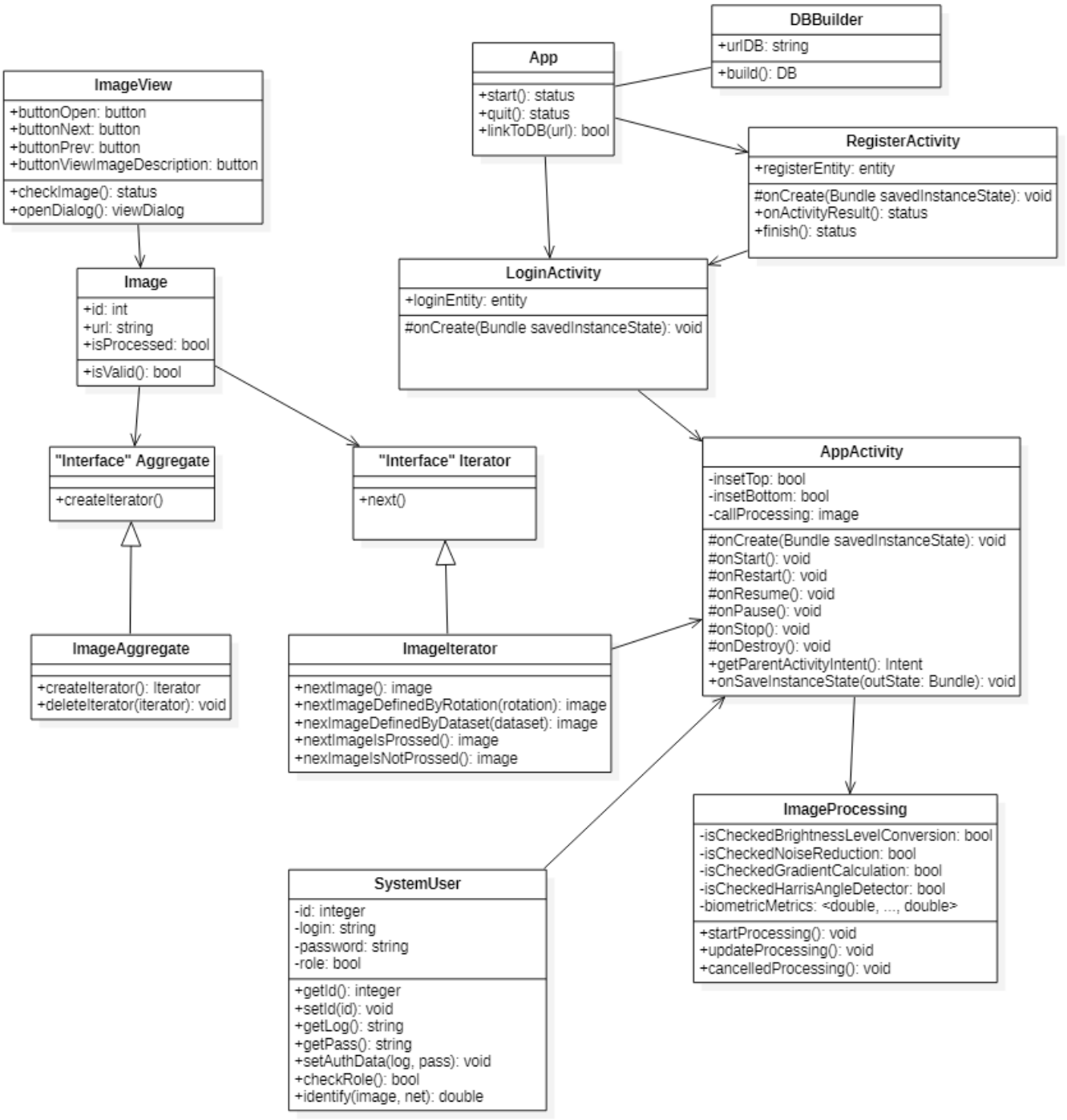

Рисунок 4.8 – Діаграма програмних класів клієнтської частини системи

Клас RegisterActivity – активність інтерфейсу для реєстрації користувача системи (менеджера).

Атрибути та методи класу RegisterActivity:

registerEntity: entity – сутність для реєстрації нового користувача;

onCreate(Bundle savedInstanceState): void – створення нового актора, збереження даних в екземплярі класу Bundle, що реалізує асоціативний масив, тобто зберігає пари ключ-значення;

onActivityResult(): status – статус отриманого результату;

finish(): status – завершення роботи активності, метод дозволяє економити ресурси, які використовує застосування.

Клас LoginActivity – активність інтерфейсу для авторизації користувача системи (менеджера).

Атрибути та методи класу LoginActivity:

loginEntity: entity – сутність для авторизації зареєстрованого користувача;

onCreate(Bundle savedInstanceState): void – доступ користувача (менеджера) до власного функціоналу та налаштувань.

Клас AppActivity – головне актівіті інтерфейсу для роботи системи машинного зору для біометричної ідентифікації.

Атрибути та методи класу AppActivity:

insetTop: bool – чи належить елемент інтерфейсу до верхньої частини елементів на insets;

insetBottom: bool – чи належить елемент інтерфейсу до нижньої частини елементів на insets;

callProcessing: image – зображення, яке потребує обробки;

onCreate(Bundle savedInstanceState): void – створення головного інстансу застосунку;

onStart(): void – запуск інстансу;

onRestart(): void – перезапуск інстансу з можливістю оновлення;

onResume(): void – отримання оновлень та продовження роботи;

onPause(): void – тимчасова зупинка виконання інстансу, припинення обміну повідомленнями;

onStop(): void – зупинка виконання інстансу;

onDestroy(): void – знищення інстансу;

getParentActivityIntent(): Intent – повернення інтенту для запуску батьківської активності;

onSaveInstanceState(outState:Bundle): void – фіксує стан активності перед її знищенням. Використовується для звільнення ресурсів для використання іншими застосунками.

Клас SystemUser використовується для забезпечення роботи користувача (менеджера або звичайного користувача) програмної системи.

Атрибути та методи класу SystemUser:

id: integer – ідентифікатор менеджера;

login: string – логін у шифрованому представленні;

password: string – пароль у шифрованому представленні;

role: bool – роль користувача, role=true для менеджера та false для звичайного користувача;

getId(): integer – метод для отримання ідентифікатора менеджера;

setId(id): void – встановлення ідентифікатора для менеджера, відбувається при його авторизації у системі;

getLog(): string – метод для отримання логіну менеджера, використовується для його аутентифікації;

getPass(): string – метод для отримання пароля менеджера, використовується для його аутентифікації;

setAuthData(log, pass): void – метод для встановлення аутентифікаційних даних менеджера;

checkRole(): bool – визначення ролі поточного користувача системи;

identify(image, net): double – ідентифікація зображення image з використанням натренованої нейромережі net.

Клас Image використовується для забезпечення роботи з зображеннями, на основі яких виконується біометрична ідентифікація.

Атрибути та методи класу Image:

id: int – ідентифікатор зображення;

url: string – адреса, за якою зберігається зображення. Використовується для звернення до нього;

isProcessed: bool – прапорець, що визначає чи було вже оброблено це зображення;

isValid() – метод, який визначає чи є зображення валідним.

Клас ImageView є видом для класу Image.

Атрибути та методи класу ImageView:

buttonOpen: button – кнопка, по натисканню на яку зображення відкривається на виводиться на встановлену область екрану;

buttonNext: button – кнопка, по натисканню на яку здійснюється перехід на наступне зображення;

buttonPrev: button – кнопка, по натисканню на яку здійснюється перехід на попереднє зображення;

buttonViewImageDescription: button – кнопка, по натисканню на яку виводиться опис зображення;

checkImage(): status – метод для перевірки цілісності зображення;

openDialog(): viewDialog – метод для вікна діалогу.

"Interface" Aggregate – інтерфейс для агрегатора, що створює ітератори.

createIterator() – абстрактний метод для створення ітератору.

"Interface" Iterator – інтерфейс для покрокового ітератора.

next() – абстрактний метод для отримання наступного зображення у пулі (датасеті) зображень.

Клас ImageAggregate – клас, що успадковується від "Interface" Aggregate та створює ітератори.

createIterator(): Iterator – створення нового ітератора для перебору зображень, метод повертає ітератор;

deleteIterator(in iterator): void – метод для видалення ітератора.

Клас ImageIterator – клас, що успадковується від "Interface" Iterator та створює певні реалізації ітераторів.

nextImage(): image – перехід до наступного зображення;

nextImageDefinedByRotation(in rotation): image – перехід до наступного зображення, яке визначено певним кутом повороту;

nexImageDefinedByDataset(in dataset): image – перехід до наступного зображення, яке належить до певного датасету;

nextImageIsProssed(): image – метод для отримання наступного обробленого зображення;

nexImageIsNotProssed(): image – метод для отримання наступного необробленого (оригінального) зображення.

Клас ImageProcessing – клас, який реалізує попередню обробку зображень (з датасету та також зроблених з камери пристрою) з метою покращення їх якості та збільшення достовірності біометричної ідентифікації.

Атрибути та методи класу ImageProcessing:

isCheckedBrightnessLevelConversion: bool – атрибут, що визначає, чи був обран алгоритм перетворення рівня яскравості;

isCheckedNoiseReduction: bool – атрибут, що визначає, чи був обран алгоритм зниження шуму;

isCheckedGradientCalculation: bool – атрибут, що визначає, чи був обран алгоритм розрахунку градієнта;

isCheckedHarrisAngleDetector: bool – атрибут, що визначає, чи був обран алгоритм виявлення кутів Харріса;

biometricMetrics: <double, ..., double> – біометричні показники (Базові метрики біометричної ідентифікації обличчя);

startProcessing(): void – початок обробки зображення відповідно до обраних алгоритмів;

updateProcessing(): void – оновлення зображення відповідно до додатково обраних алгоритмів;

cancelledProcessing(): void – відміна обробки, зображення залишається у початковому вигляді.

4.5 Висновки до розділу

У розділі було виконано проектування системи машинного зору. Для цього була попередньо розроблена клієнт-серверна архітектура системи з підтримкою локальної бази даних та хмарного середовища Cloud, а також з використанням шаблону проектування Ітератор.

Розроблено реляційну модель для збереження даних системи машинного зору. Виділено ключові сутності та міжтабличні зв'язки.

Виконано проектування інтерфейсу користувача. Створено прототипи основних програмних форм.

Створено алгоритми взаємодії з інтерфейсом при навчанні нейронної мережі та при проведенні біометричної ідентифікації.

Виконано проектування програмних класів та створення відповідної UML діаграми, а також опису атрибутів та методів.

5 ПРОГРАМНА РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

5.1 Особливості створення Android-застосунків та системні вимоги

Розглянемо особливості розробки програмних продуктів під ОС Android.

«Операційна система Android є багатокористувацькою системою Linux, в якій кожен застосунок є іншим користувачем.

За замовчуванням система привласнює кожному застосунку ID (унікальний ідентифікатор) користувача Linux (ID використовується тільки системою та невидимий для застосунку).

Система встановлює права доступу для всіх файлів, так що тільки даний ідентифікатор, наданий застосунку може отримати до них доступ.

Кожен процес має свою віртуальну машину та є ізольованим від інших програм. За замовчуванням кожний застосунок запускається у своєму процесі Linux.

Android запускає процес, коли будь-який з компонентів застосунку, необхідний для виконання, має бути запущений. Потім Android знищує процес, якщо він більше не потрібен, чи то коли система повинна відновити пам'ять для роботи інших застосунків.» [18]

Системні вимоги до клієнтської частини:

1) операційна система: не нижче Android 7.0;
2) процесор: 2 ГГЦ або більш швидкий процесор;
3) оперативна пам'ять: мінімум 512 МБ, 2 МБ рекомендується;
4) жорсткий диск: 170 МБ доступного місця на жорсткому диску для встановлення;
5) швидкість інтернет: мінімальна 500 КБ/с.

Система потребує встановлення користувачем та надання усіх необхідних дозволів. Застосування може бути встановлене завдяки файлу формату apk (формат

архівних файлів для «Android»), так і за допомогою PlayMarket (крамниця застосунків від Google).

5.2 Налаштування адаптивного інтерфейсу

Адаптивне розташування елементів інтерфейсу передбачає, що вони будуть «однаково добре відображатись на всіх можливих пристроях: комп'ютерах, планшетах, телефонах» [19].

```
boolean isOpened = false;

public void setListenerToRootView() {
    final View activityRootView = getWindow().getDecorView().findViewById(android.R.id.content);
    activityRootView.getViewTreeObserver().addOnGlobalLayoutListener(new OnGlobalLayoutListener() {
        @Override
        public void onGlobalLayout() {

            int heightDiff = activityRootView.getRootView().getHeight() - activityRootView.getHeight();
            if (heightDiff > 100) { // 99% of the time the height diff will be due to a keyboard.
                Toast.makeText(getApplicationContext(), "softKeyboardup", 0).show();

                if (isOpened == false) {
                   //To Do
                }
                isOpened = true;
            } else if (isOpened == true) {
                Toast.makeText(getApplicationContext(), "softkeyborad", 0).show();
                isOpened = false;
            }
        }
    });
}
```

Рисунок 5.1 – Приклад налаштування адаптивного інтерфейсу

```
@Override
public WindowInsets dispatchApplyWindowInsets(WindowInsets insets) {
    insets = super.dispatchApplyWindowInsets(insets);
    if (!insets.isConsumed()) {
        final int count = getChildCount();
        for (int i = 0; i < count; i++) {
            insets = getChildAt(i).dispatchApplyWindowInsets(insets);
            if (insets.isConsumed()) {
                break;
            }
        }
    }
    return insets;
}
```

Рисунок 5.2 – Оптимізація верстки з використанням insets

Перевизначення методу dispatchApplyWindowInsets дозволяє всім дочірнім елементам завжди віддавати всі оброблені та необроблені insets, які прийшли.

```
class WindowInsetFrameLayout @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleRes: Int = 0
) : FrameLayout(context, attrs, defStyleRes) {

    override fun dispatchApplyWindowInsets(insets: WindowInsets): WindowInsets {
        for (child in (0 until childCount)) {
            getChildAt(child).dispatchApplyWindowInsets(insets)
        }
        return insets.consumeSystemWindowInsets()
    }
}
```

Рисунок 5.3 – Створення власної ViewGroup

5.3 Приклади використання інтерфейсу

Після встановлення застосунку на екрані простою з’явиться піктограма, натиснувши на яку можна запустити систему.

При реєстрації до цього сервісу відправляється запит на зберігання нового користувача за його електронною адресою та паролем. Після успішної реєстрації користувачу надається унікальний токен, який дає змогу серверу ідентифікувати користувача, щоб користувач мав доступ тільки до тих дані, які належать йому.

На даному екрані бачимо:

1) два поля «Email» та «Password» для внесення даних користувача;
2) напис «Sign In» який можна натискати тим самим змінюючи поля з реєстрації на авторизацію;
3) кнопку «Register» для підтвердження реєстрації.

Детальніше показано на рис. 5.4.

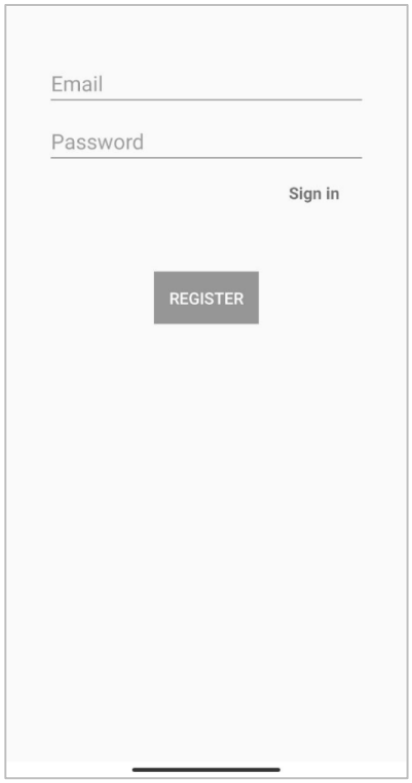

Рисунок 5.4 – Екран реєстрації у застосунку

На рисунку бачимо, що коли користувач потрапляє до екрану авторизації, кнопка «Register» є неактивною, тому що поля Email та Password не заповнені.

Заповнимо поля вірно, та натиснемо кнопку «Register» (рис. 5.5).

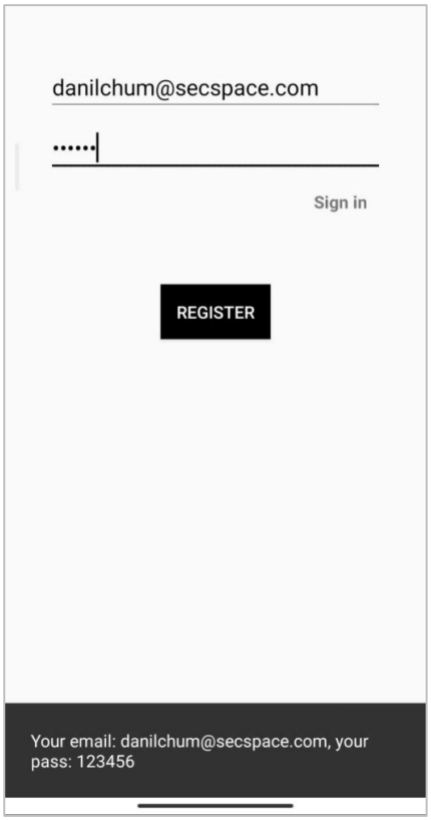

Рисунок 5.5 – Повідомлення про реєстрування

Після того, як користувачу буде відображене повідомлення, про те, що він авторизований, потрапляємо на головний екран.

Якщо користувач вже зареєстрований, то необхідно використати екран авторизації, введемо необхідні дані та натиснемо кнопку «SignIn», та побачимо повідомлення об успішній авторизації, після чого потрапимо до головного екрану (рис. 5.6 – 5.7).

Рисунок 5.6 – Вікно авторизації

Рисунок 5.7 – Повідомлення про успішну авторизацію

На рис. 5.8 показаний екран завантаження датасету для тренування нейромережі, в якому можемо побачити наступні елементи:

1) назву датасету;
2) короткий опис датасету;
3) кількість наборів зображень;
4) три крапки зверху у правому кутку для відкриття налаштувань застосування;
5) три рисочки у лівому кутку для відкриття навігаційної панелі;
6) кнопки «Previous», «Next» та цифри – для переключення між датасетами;
7) кнопки «Load» та «Cancel» для підтвердження чи відхилення завантаження датасету.

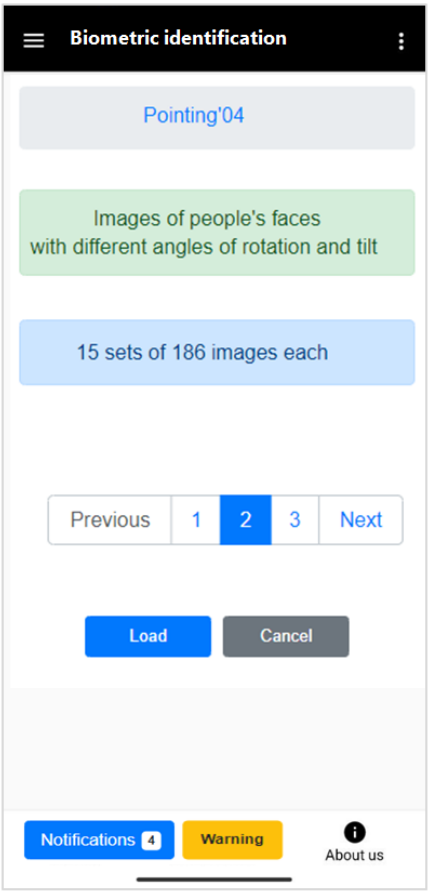

Рисунок 5.8 – Екран завантаження датасету для тренування нейромережі

На рис. 5.9 та 5.10 показані екрани для завантаження зображення для розпізнавання та для виводу результату розпізнавання.

Після розпізнавання користувач натискає кнопку «Actions» для вибору дії, яку він хоче зробити після розпізнавання (рис. 5.11).

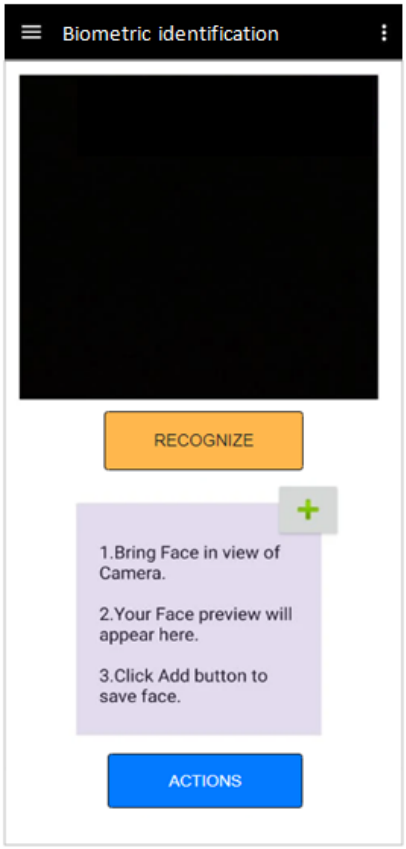

Рисунок 5.9 – Екран завантаження зображення

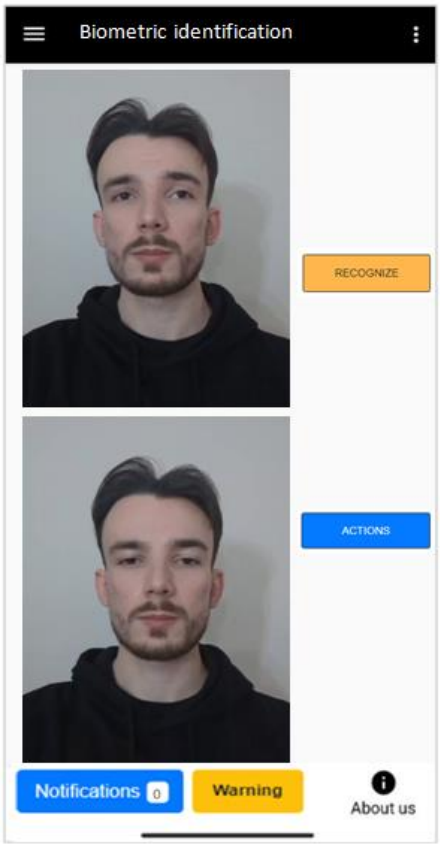

Рисунок 5.10 – Екран виводу результату

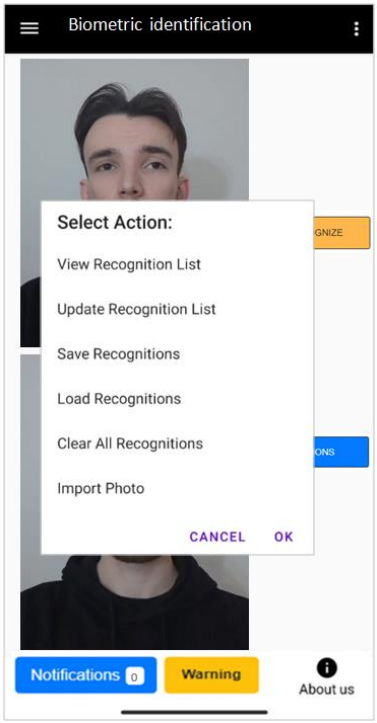

Рисунок 5.11 – Екран повідомлень

5.3 Висновки до розділу

У даному розділі розглянуто особливості Андроід-застосувань та способи налаштування адаптивного інтерфейсу. Розглянута оптимізація верстки з використанням insets, а також створення власної ViewGroup з застосуванням insets.

Розглянуто приклади інтерфейсу користувача:

– екран реєстрації та повідомлення про реєстрацію;

– екран авторизації та повідомлення про успішну авторизацію;

– екран завантаження датасету для тренування нейромережі;

– екрани завантаження зображення, отримання результату та вибору дій над результатом.

6 ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

6.1 Тестування роботи алгоритмів обробки двовимірних зображень

На початку дослідження проведемо тестування роботи алгоритмів обробки двовимірних зображень, не спираючись на складну структуру згорткової нейромережі, тому що зараз нас цікавлять саме характеристики якості зображень після їх обробки.

Оберемо найпростішу архітектуру нейронної мережі – одношаровий персептрон (рис. 6.1) з функцією активації RELU:

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} = \max(x, 0) \quad (6.1)$$

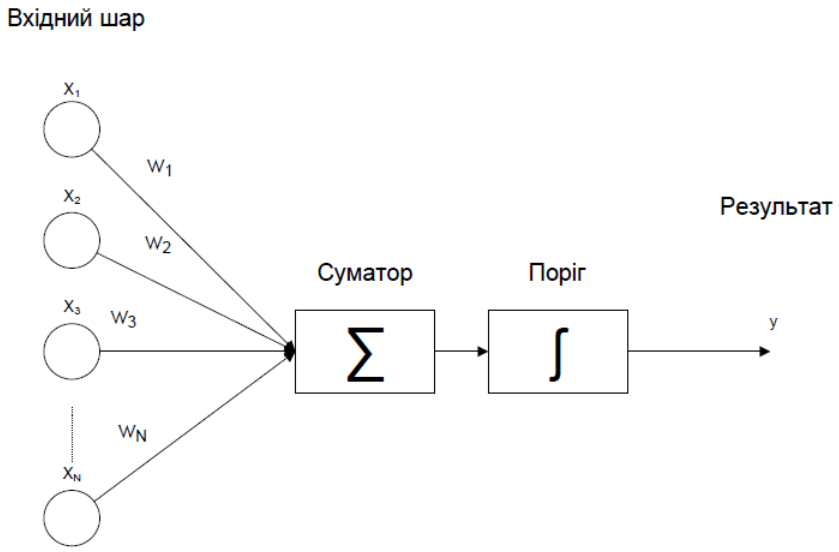

Рисунок 6.1 – Схема нейромережі – одношарового персептрона

На вхід нейромережі подається значення базових метрик біометричної ідентифікації обличчя. У якості виходу є результат ідентифікації певної людини.

На рис. 6.2 показаний результат ідентифікації на зображеннях без попередньої обробки, зроблених на сучасну камеру Андроід-пристрою. На рис. 6.3 містяться результати після обробки зображень перетворенням рівня яскравості

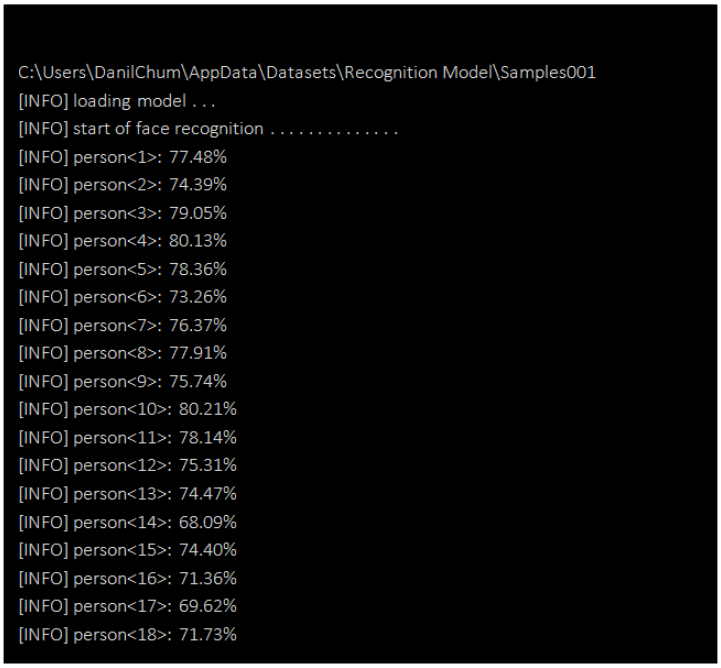

Рисунок 6.2 – Результати ідентифікації по початковим зображенням

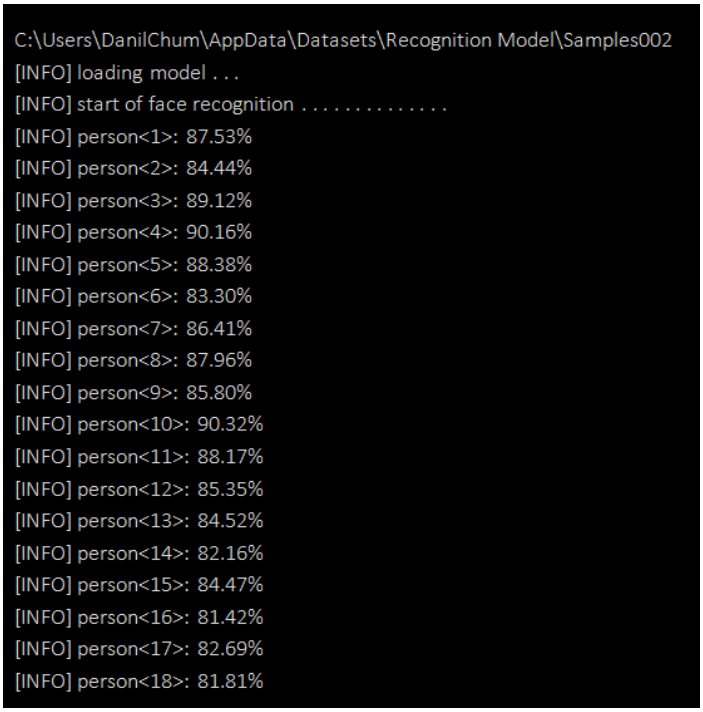

Рисунок 6.3 – Результати ідентифікації після алгоритму Brightness level conversion

До попередніх результатов додаємо обробку алгоритмом зниження шуму (рис. 6.4). На рис. 6.5 наведены результати після застосування алгоритму Розрахунок градієнта. Бачимо, що достовірність ще підвищилась.

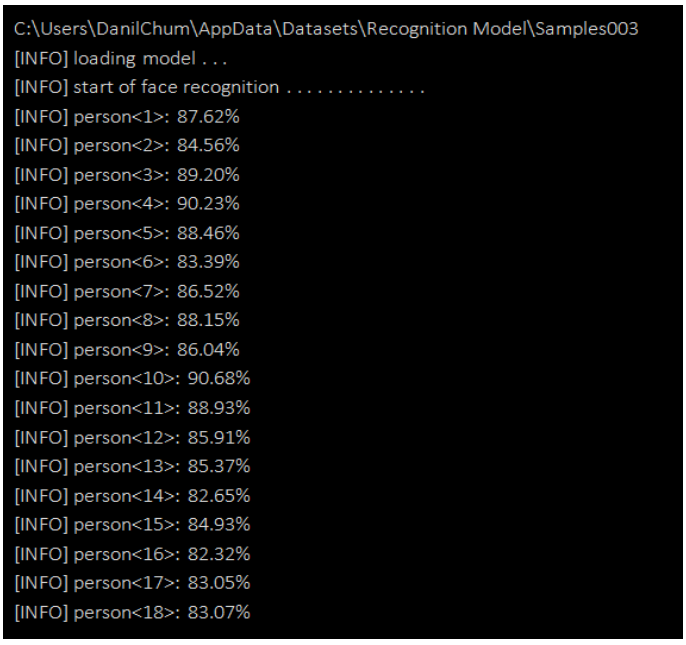

Рисунок 6.4 – Результати ідентифікації після алгоритму Noise reduction

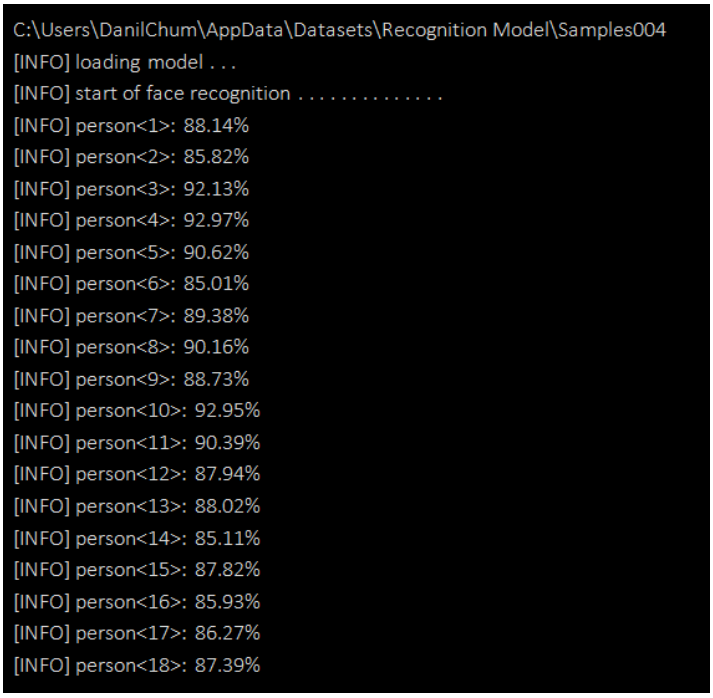

Рисунок 6.5 – Результати ідентифікації після алгоритму Gradient Calculation

До отриманих зображень додаємо останню обробку – детектор кутів Харріса (рис. 6.6). Як бачимо, достовірність ідентифікації збільшується.

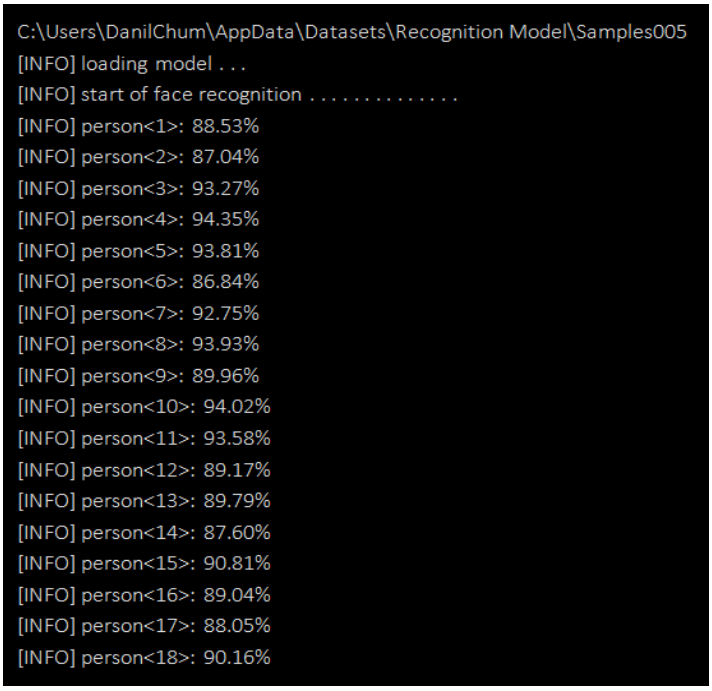

Рисунок 6.6 – Результати ідентифікації після алгоритму Harris angle detector

Тепер візьмемо згорткову нейронну мережу, топологія якої була обрана в серверної частині розробки, та проведемо ідентифікацію оброблених зображень (рис. 6.7).

Як можна побачити, попередня обробка зображень суттєво вплинула на результати достовірності біометричної ідентифікації зображень.

6.2 Дослідження впливу кута повороту зображення на результат

Усі попередні наведені дослідження стосувались фронтальних зображень, як обличчя, так і геометрії тіла. Але, крім того, потрібно вивчити вплив кута повороту зображення на результат ідентифікації, тому що переважна більшість зображень відхиляються від нормалі.

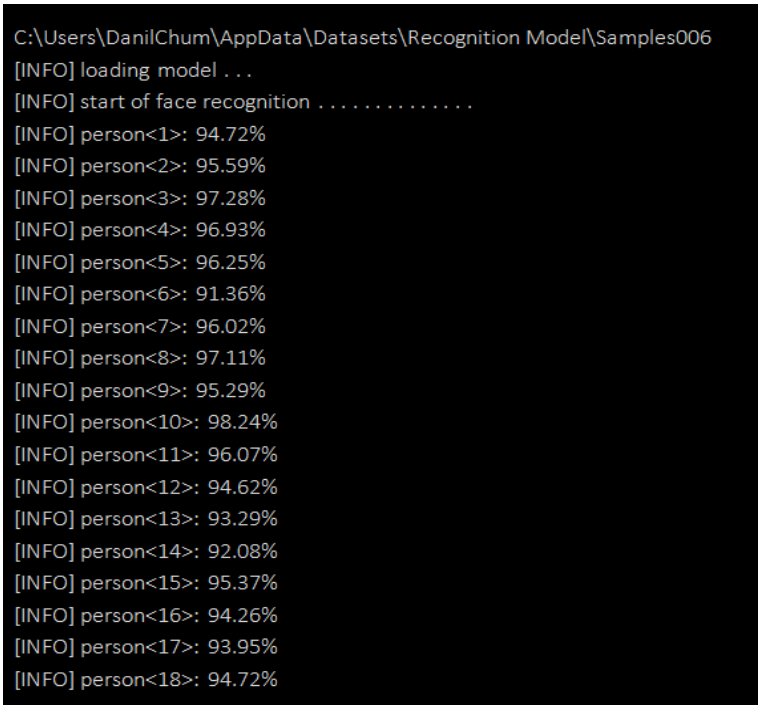

Рисунок 6.7 – Результати ідентифікації оброблених зображень згортковою нейромережею

У табл. 6.1 наведена кількість успішних ідентифікацій (з точністю >=80%) для відповідних діапазонів кутів поворота. Для кожного діапазону розглядалось по 100 зображень.

Кут відхилення від нормалі	Зображення обличчя	Зображення у повний зріст
0°-20°	92%	89%
20°-50°	74%	65%
50°-90°	51%	43%
90°-120°	38%	32%
120°-180°	27%	16%

Таблиця 6.1 – Результати розпізнования в залежностші від кута зображення.

Як і слід було очікувати, так як нейронна мережа тренувалась переважно на фронтальних зображеннях, розпізнавання при наявності кутів є дуже слабким. Для покращення ситуації потрібно тренувати нейромережі на різноманітних зображеннях.

6.3 Висновки до розділу

У шостому розділі було проведено дослідження алгоритмів обробки двовимірних зображень. Результати застосування алгоритмів обробки показав суттєве зростання достовірності ідентифікації. Використання згорткової нейромережі дозволило підвищити достовірність у середньому до 95,26%.

При цьому ж дослідження успішності ідентифікацій при відхиленні зображення на певний кут показало невисокий результат, що свідчить про необхідність додаткового тренування нейромережі на зображеннях з кутами повороту.

ВИСНОВКИ

У даній кваліфікаційній роботі розроблено клієнтську частину програмної системи машинного зору для біометричної ідентифікації осіб за двовимірними зображеннями з використанням методів штучного інтелекту. Також в роботі реалізовані алгоритми попередньої обробки зображень. Використання системи дозволило підвищити достовірність ідентифікації за обличчям людини у середньому до 95,26% на фронтальних зображеннях.

У першому розділі проаналізовані сучасні програмні рішення за темою дослідження, зроблено висновок щодо актуальності завдання.

У другому розділі проведено класифікацію задач аналізу зображень. Проаналізовано найбільш розповсюджені методи обробки зображень: перетворення рівня яскравості, зниження шуму та розрахунку градієнта для покращення якості зображення.

У третьому розділі створена та описана специфікація вимог до клієнтської частини системи машинного зору для біометричної ідентифікації. Розроблена діаграма варіантів використання системи, розглянуті нефункціональні вимоги.

Четвертий розділ містить проектування системи машинного зору на базі клієнт-серверної архітектури. Розроблено реляційну модель для збереження даних та UML-діаграми.

У п’ятому розділі розглянуто особливості Андроід-застосувань, способи налаштування адаптивного інтерфейсу та приклади клієнтського інтерфейсу.

Шостий розділ містить результати дослідження та тестування алгоритмів обробки двовимірних зображень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A piece of yourself": Ethical issues in biometric identification [Електроний ресурс] https://www.academia.edu/32713161/A_piece_of_yourself_Ethical_issues_in_biometric_identification

2. Charge – coupled device [Електроний ресурс] https://www.techtarget.com/searchstorage/definition/charge-coupled-device.

3. Відмінності між розпізнаванням райдужки і скануванням сітківки [Електроний ресурс] https://worldvision.com.ua/articles/razlichiya-mezhdu-raspoznavaniem-raduzhki-i-skanirovaniem-setchatki

4. Bruce Eckel, Svetlana Isakova. Atomic Kotlin. Mindview LLC. 2021. 636 p.

5. iFING Scanner [Електроний ресурс] https://play.google.com/store/apps/details?id=com.winningi.www.aerox_scanner&hl=uk&gl=US

6. Додаток для автентифікації 2FA [Електроний ресурс] https://play.google.com/store/apps/details?id=com.pixsterstudio.authenticator&hl=uk&gl=US

7. Insyt Biometric [Електроний ресурс] https://play.google.com/store/apps/details?id=com.esoko.insyt.secugenfingerprint&hl=uk&gl=US

8. BioID Facial Recognition [Електроний ресурс] https://play.google.com/store/apps/details?id=com.bioid.authenticator&hl=uk&gl=US

9. DNI BioFacial [Електроний ресурс] https://play.google.com/store/apps/details?id=pe.gob.reniec.dnibioface&hl=uk&gl=US

10. Harris Corner Detector [Електроний ресурс] https://www.it.lut.fi/kurssit/07-08/CT20A6100/seminars/2009-2010/Harris.pdf

11. What is a corner? [Електроний ресурс] https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/COMPSCI%20773%20feature%20point%20detection.pdf

12. Labeled Faces in the Wild [Електроний ресурс] / http://vis-www.cs.umass.edu. – Режим доступу: http://vis-www.cs.umass.edu/lfw/

13. Клієнт-серверна архітектура [Електроний ресурс] https://training.qatestlab.com – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995. 344 p.

15. Дизайн звичних речей /Дональд А. Норман – Клуб Сімейного Дозвілля, 2019 – 320 с.

16. Дивовижні технології. Дизайн та інтернет речей /Девід Роуз – Клуб Сімейного Дозвілля, 2018. – 336 с.

17. Проектування програмних засобів систем управління. Частина 1. Основи об`єктно-орієнтованого проектування : навчальний посібник / О. М. Бевз, В. М. Папінов, Ю. А. Скидан. – Вінниця : ВНТУ, 2010. – 125 с.

18. Design Patterns in Android — Adapter [Електроний ресурс] /Paulina Szklarska. - Режим доступу: https://pszklarska.medium.com/design-patterns-in-android-adapter-875538c343c3

19. Адаптивна верстка [Електроний ресурс] https://webtune.com.ua - Режим доступу: https://webtune.com.ua/statti/web-rozrobka/adaptyvna-verstka-sajtu/

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
package com.chumachenko.identification.app.di

import androidx.room.Room
import com.chumachenko.core.common.ResourceManager
import com.chumachenko.core.common.recognize.AppRecognize
import com.chumachenko.core.data.storage.IdentificationsPreferencesManager
import com.chumachenko.core.data.storage.database.IdentificationDatabase
import org.koin.dsl.module

val appModule = module {

single { ResourceManager(get()) }

single { IdentificationPreferencesManager(get()) }

single {
Room.databaseBuilder(
get(),
IdentificationDatabase::class.java, "Database"
).fallbackToDestructiveMigration()
            .build()
}

single { AppRecognize() }
}

====
package com.chumachenko.identification.app.di

import com.chumachenko.core.data.networking.CoreApi
import com.chumachenko.core.data.networking.InfoApi
import com.chumachenko.identification.BuildConfig.BASE_URL
import com.google.gson.Gson
import com.google.gson.GsonBuilder
import okhttp3.OkHttpClient
import org.koin.dsl.module
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import java.util.concurrent.TimeUnit

val networkingModule = module {

single { com.chumachenko.core.common.NetworkUtils(get()) }

single<Gson>{
GsonBuilder().create()
}

single {
OkHttpClient.Builder().apply {
writeTimeout(60, TimeUnit.SECONDS)
readTimeout(60, TimeUnit.SECONDS)
}.build()
}

single<Retrofit>{
```

```
Retrofit.Builder()
            .baseUrl(BASE_URL)
            .client(get())
            .addConverterFactory(GsonConverterFactory.create(get()))
            .build()
}

single { provideApi<CoreApi>(get()) }
single { provideApi<InfoApi>(get()) }

}

inline fun <reified T>provideApi(retrofit: Retrofit): T {
return retrofit.create(T::class.java)
}

====
package com.chumachenko.identification.app.di

import com.chumachenko.core.data.repository.CoreRepository
import com.chumachenko.info.repository.InfoRepository
import com.chumachenko.core.data.storage.cache.SearchCache
import com.chumachenko.core.domain.interactor.CoreInteractor
import com.chumachenko.info.domain.interactor.InfoInteractor
import com.chumachenko.core.ui.CoreViewModel
import com.chumachenko.info.ui.InfoViewModel

import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.dsl.module

val screenModule = module {

//Core
factory { CoreRepository(get(), get(), get()) }
factory { CoreInteractor(get()) }
viewModel { CoreViewModel(get(), get(), get(), get(), get()) }
single { SearchCache() }

//Info
factory { InfoRepository(get(), get()) }
factory { InfoInteractor(get()) }
viewModel { InfoViewModel(get(), get(), get()) }

}

====
package com.chumachenko.identification.app.ui

import android.content.Context
import android.content.Intent
import android.graphics.Color
import android.os.Build
import android.os.Bundle
import android.view.WindowManager
import android.widget.FrameLayout
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.WindowCompat
import androidx.core.view.WindowInsetsCompat
import androidx.core.view.WindowInsetsControllerCompat
```

```
import com.chumachenko.core.common.InsetHolder
import com.chumachenko.core.extensions.dpToPixel
import com.chumachenko.core.ui.CoreProcessing
import com.chumachenko.identification.R
import com.chumachenko.identification.databinding.ActivityMainBinding

class AppActivity : AppCompatActivity(), InsetHolder {

override val topInset: Int
get() = _insetTop
override val bottomInset: Int
get() = _insetBottom

private var _insetTop = 0
private var _insetBottom = 0
private var isWindowInsetSet = false

    private lateinit var binding: ActivityMainBinding

override fun onSaveInstanceState(outState: Bundle) {
outState.putInt(INSET_TOP, _insetTop)
outState.putInt(INSET_BOTTOM, _insetBottom)
outState.putBoolean(IS_WINDOW_INSET_SET, isWindowInsetSet)
super.onSaveInstanceState(outState)
    }

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
binding = ActivityMainBinding.inflate(layoutInflater)
setContentView(binding.root)

savedInstanceState?.apply {
_insetTop = getInt(INSET_TOP)
_insetBottom = getInt(INSET_BOTTOM)
isWindowInsetSet = getBoolean(IS_WINDOW_INSET_SET)
}

window.apply {
addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)

WindowCompat.setDecorFitsSystemWindows(this, false)

val insetsController = WindowInsetsControllerCompat(this, binding.root)
insetsController.isAppearanceLightStatusBars = true
statusBarColor = Color.TRANSPARENT
}

if (isWindowInsetSet) {
if (savedInstanceState == null) startCoreProcessing()
        } else {
binding.root.setOnApplyWindowInsetsListener { _, insets ->
val insetsCompat = WindowInsetsCompat.toWindowInsetsCompat(insets)
                    .getInsets(WindowInsetsCompat.Type.systemGestures())

if (!isWindowInsetSet) {
_insetTop = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
val displayCutout = insets.displayCutout
displayCutout?.safeInsetTop ?: insetsCompat.top
} else {
```

```
dpToPixel(24, this).toInt()
                    }

_insetBottom = WindowInsetsCompat.toWindowInsetsCompat(insets)
                        .getInsets(WindowInsetsCompat.Type.systemBars()).bottom

if (_insetBottom == 0 &&Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q
&&insetsCompat.bottom != 0) {
_insetBottom = insetsCompat.bottom
}

isWindowInsetSet = true

                    val params = binding.root.layoutParams as
FrameLayout.LayoutParams
params.bottomMargin = _insetBottom
                    binding.root.layoutParams = params

if (savedInstanceState == null) startCoreProcessing()
                }
insets
}
}
startCoreProcessing()
    }

private fun startCoreProcessing() {
val transaction = supportProcessingManager.beginTransaction()
            .replace(R.id.container, CoreProcessing.newInstance())
if (!supportProcessingManager.isStateSaved) {
transaction.commit()
    }
    }

override fun onNewIntent(intent: Intent?) {
super.onNewIntent(intent)
this.intent = intent
}

companion object {
private const val INSET_TOP = "insetTop"
private const val INSET_BOTTOM = "insetBottom"
private const val IS_WINDOW_INSET_SET = "isWindowInsetSet"

fun newIntent(context: Context): Intent {
return Intent(context, AppActivity::class.java)
        }
    }
}

====
package com.chumachenko.identification.app.ui

import android.os.Bundle
import android.os.Handler
import android.os.Looper
import androidx.appcompat.app.AppCompatActivity
import com.chumachenko.identification.R
```

```
class SplashActivity : AppCompatActivity(R.layout.activity_splash) {

private val uiHandler = Handler(Looper.getMainLooper())

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)

uiHandler.imageDelayed({
startActivity(AppActivity.newIntent(this))
finish()
}, 500)
    }

override fun onDestroy() {
uiHandler.removeCallbacksAndMessages(null)
super.onDestroy()
    }
}

=====
package com.chumachenko.identification.app

import android.app.Application
import androidx.Processing.app.Processing
import androidx.Processing.app.ProcessingManager
import androidx.Processing.app.ProcessingTransaction
import com.chumachenko.core.common.Router
import com.chumachenko.core.data.model.Identification
import com.chumachenko.identification.R
import com.chumachenko.info.ui.InfoProcessing

class Application : Application(), Router {

private fun addProcessing(
containerViewId: Int,
ProcessingManager: ProcessingManager,
Processing: Processing,
transition: Int
) {
val transaction = ProcessingManager
.beginTransaction()
            .add(
containerViewId,
Processing,
Processing.javaClass.simpleName
)
            .addToBackStack(Processing.javaClass.simpleName)
            .setTransition(transition)

if (!ProcessingManager.isStateSaved) {
transaction.commit()
        }
    }

private fun addBottomSheet(
ProcessingManager: ProcessingManager,
Processing: Processing,
transition: Int
) {
```

```
val transaction = ProcessingManager
.beginTransaction()
            .replace(
R.id.bottomSheetContainer,
Processing
)
            .addToBackStack(Processing.javaClass.simpleName)
            .setTransition(transition)

if (!ProcessingManager.isStateSaved) {
transaction.commit()
        }
    }

override fun openInfoScreen(ProcessingManager: ProcessingManager, info: Info) {
addBottomSheet(
ProcessingManager,
InfoProcessing.newInstance(info),
ProcessingTransaction.TRANSIT_PROCESSING_FADE
)
    }
}

=====
package com.chumachenko.identification.app

import android.content.Context
import androidx.startup.Initializer
import com.chumachenko.identification.app.di.appModule
import com.chumachenko.identification.app.di.networkingModule
import com.chumachenko.identification.app.di.screenModule
import org.koin.android.ext.koin.androidContext
import org.koin.core.KoinApplication
import org.koin.core.context.GlobalContext.startKoin

class KoinInitializer : Initializer<KoinApplication> {
override fun create(context: Context): KoinApplication {
return startKoin {
androidContext(context)
modules(
listOf(
appModule,
networkingModule,
screenModule
)
            )
}
}

override fun dependencies(): MutableList<Class<out Initializer<*>>> {
return mutableListOf()
    }
}


====
package com.chumachenko.core.ui

import android.Manifest
```

```
import android.content.pm.PackageManager
import android.hd.image.ImageAccessException
import android.hd.image.ImageCharacteristics
import android.hd.image.ImageManager
import android.media.Image.Plane
import android.media.ImageReader
import android.media.ImageReader.OnImageAvailableListener
import android.os.*
import android.util.Size
import android.view.KeyEvent
import android.view.Surface
import android.view.View
import android.view.WindowManager
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.chumachenko.core.R

abstract class ImageActivity : AppCompatActivity(), OnImageAvailableListener {

private var handler: Handler? = null
    private var handlerThread: HandlerThread? = null
  private var isProcessingFrame = false
    private val yuvBytes = arrayOfNulls<ByteArray>(3)
private var rgbBytes: IntArray? = null
    private var luminanceStride = 0
private var previewWidth = 0
private var previewHeight = 0
private var postInferenceCallback: Runnable? = null
    private var imageConverter: Runnable? = null

    var isDebug = false
        private set

    override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(null)
window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
setContentView(R.layout.activity_image)
if (hasPermission()) {
setProcessing()
        } else {
requestPermission()
        }
    }

protected fun getRgbBytes(): IntArray? {
imageConverter!!.run()
return rgbBytes
}

protected val luminance: ByteArray?
protected get() = yuvBytes[0]

override fun onImageAvailable(reader: ImageReader) {
if (previewWidth == 0 || previewHeight == 0) {
return
}
if (rgbBytes == null) {
rgbBytes = IntArray(previewWidth * previewHeight)
        }
```

```
try {
val image = reader.acquireLatestImage() ?: return
            if (isProcessingFrame) {
image.close()
return
}
isProcessingFrame = true
Trace.beginSection("imageAvailable")
val planes = image.planes
fillBytes(planes, yuvBytes)
luminanceStride = planes[0].rowStride
val uvRowStride = planes[1].rowStride
val uvPixelStride = planes[1].pixelStride
imageConverter = Runnable {
ImageUtils.convertYUV420ToARGB8888(
yuvBytes[0],
yuvBytes[1],
yuvBytes[2],
previewWidth,
previewHeight,
luminanceStride,
uvRowStride,
uvPixelStride,
rgbBytes
)
}
postInferenceCallback = Runnable {
image.close()
isProcessingFrame = false
}
processImage()
        } catch (e: Exception) {
Trace.endSection()
return
}
Trace.endSection()
    }

public override fun onResume() {
super.onResume()
handlerThread = HandlerThread("inference")
handlerThread!!.start()
handler = Handler(handlerThread!!.looper)
    }

@Synchronized
public override fun onPause() {
handlerThread!!.quitSafely()
try {
handlerThread!!.join()
handlerThread = null
handler = null
} catch (e: InterruptedException) {
e.printStackTrace()
        }
super.onPause()
    }

@Synchronized
```

```
public override fun onStop() {
super.onStop()
    }

@Synchronized
public override fun onDestroy() {
super.onDestroy()
    }

override fun onWindowFocusChanged(hasFocus: Boolean) {
super.onWindowFocusChanged(hasFocus)
if (hasFocus) {
window.decorView.systemUiVisibility = (View.SYSTEM_UI_FLAG_LAYOUT_STABLE
or View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
or View.SYSTEM_UI_FLAG_FULLSCREEN
or View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY)
        }
    }

@Synchronized
protected fun runInBackground(r: Runnable?) {
if (handler != null) {
handler!!.post(r!!)
        }
    }

override fun onRequestPermissionsResult(
requestCode: Int, permissions: Array<String>, grantResults: IntArray
) {
if (requestCode == PERMISSIONS_REQUEST) {
if (grantResults.size >0 &&grantResults[0] == PackageManager.PERMISSION_GRANTED
&&grantResults[1] == PackageManager.PERMISSION_GRANTED) {
setProcessing()
            } else {
requestPermission()
            }
        }
    }

private fun hasPermission(): Boolean {
return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
checkSelfPermission(PERMISSION_IMAGE) == PackageManager.PERMISSION_GRANTED &&
checkSelfPermission(PERMISSION_STORAGE) == PackageManager.PERMISSION_GRANTED
} else {
true
}
    }

private fun requestPermission() {
if (shouldShowRequestPermissionRationale(PERMISSION_IMAGE) ||
shouldShowRequestPermissionRationale(PERMISSION_STORAGE)
        ) {
Toast.makeText(
this@ImageActivity,
"Image and storage permission", Toast.LENGTH_LONG
).show()
        }
requestPermissions(arrayOf(PERMISSION_IMAGE, PERMISSION_STORAGE),
PERMISSIONS_REQUEST)
```

```
    }

private fun chooseImage(): String? {
val manager = getSystemService(IMAGE_SERVICE) as ImageManager
try {
for(imageId in manager.imageIdList) {
val characteristics = manager.getImageCharacteristics(imageId)


val facing = characteristics.get(ImageCharacteristics.LENS_FACING)
if (facing != null &&facing == ImageCharacteristics.LENS_FACING_FRONT) {
continue
}
val map = characteristics.get(ImageCharacteristics.SCALER_STREAM_CONFIGURATION_MAP)
                    ?: continue
 return imageId
}
        } catch (e: ImageAccessException) {
e.printStackTrace()
        }
return null
}

private fun setProcessing() {
val imageId = chooseImage()
val imageProcessing: ImageConnectionProcessing =
ImageConnectionProcessing.newInstance(
{ size, rotation ->
previewHeight = size.getHeight()
previewWidth = size.getWidth()
onPreviewSizeChosen(size, rotation)
},
this,
layoutId,
desiredPreviewFrameSize
)
imageProcessing.setImage(imageId)
ProcessingManager
.beginTransaction()
            .replace(R.id.container, imageProcessing)
           .commit()
    }

protected fun fillBytes(planes: Array<Plane>, yuvBytes: Array<ByteArray?>) {
for (i in planes.indices) {
val buffer = planes[i].buffer
if (yuvBytes[i] == null) {
yuvBytes[i] = ByteArray(buffer.capacity())
            }
buffer[yuvBytes[i]]
        }
    }

fun requestRender() {
val overlay: OverlayView = findViewById(R.id.debug_overlay)
if (overlay != null) {
overlay.postInvalidate()
        }
    }
```

```
fun addCallback(callback: OverlayView.DrawCallback?) {
val overlay: OverlayView = findViewById(R.id.debug_overlay)
if (overlay != null) {
overlay.addCallback(callback)
        }
    }

fun onSetDebug(debug: Boolean) {}
override fun onKeyDown(keyCode: Int, event: KeyEvent): Boolean {
if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN || keyCode == KeyEvent.KEYCODE_VOLUME_UP)
{
isDebug = !isDebug
requestRender()
onSetDebug(isDebug)
return true
}
return super.onKeyDown(keyCode, event)
    }

protected fun readyForNextImage() {
if (postInferenceCallback != null) {
postInferenceCallback!!.run()
        }
    }

protected val screenOrientation: Int
protected get() = when (windowManager.defaultDisplay.rotation) {
Surface.ROTATION_270 ->270
Surface.ROTATION_180 ->180
Surface.ROTATION_90 ->90
else ->0
}

protected abstract fun processImage()
protected abstract fun onPreviewSizeChosen(size: Size?, rotation: Int)
protected abstract val layoutId: Int
protected abstract val desiredPreviewFrameSize: Size?

companion object {
private const val PERMISSIONS_REQUEST = 1
private const val PERMISSION_IMAGE = Manifest.permission.IMAGE
private const val PERMISSION_STORAGE = Manifest.permission.WRITE_EXTERNAL_STORAGE
}
}


import android.R
import android.app.*
import android.content.Context
import android.content.DialogInterface
import android.content.res.Configuration
import android.graphics.ImageFormat
import android.graphics.Matrix
import android.graphics.RectF
import android.graphics.SurfaceTexture
import android.hd.image.*
import android.hd.image.ImageTrainSession.TrainCallback
import android.media.ImageReader
```

```
import android.media.ImageReader.OnImageAvailableListener
import android.os.Bundle
import android.os.Handler
import android.os.HandlerThread
import android.text.TextUtils
import android.util.Size
import android.util.SparseIntArray
import android.view.LayoutInflater
import android.view.Surface
import android.view.TextureView.SurfaceTextureListener
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import java.util.*
import java.util.concurrent.Semaphore
import java.util.concurrent.TimeUnit

open class ImageConnectionProcessingprivate constructor(
connectionCallback: ImageConnectionProcessing.ConnectionCallback,
imageListener: OnImageAvailableListener,
layout: Int,
inputSize: Size
) : Processing() {
companion object {
private const valMINIMUM_PREVIEW_SIZE = 320
private valORIENTATIONS = SparseIntArray()
private const valPROCESSING_DIALOG = "dialog"

protected fun chooseOptimalSize(choices: Array<Size>, width: Int, height: Int): Size?
{
valminSize= width.coerceAtMost(height).coerceAtLeast(MINIMUM_PREVIEW_SIZE)
valdesiredSize= Size(width, height)

var exactSizeFound= false
valbigEnough: MutableList<Size?> = ArrayList()
valtooSmall: MutableList<Size?> = ArrayList()
for (option in choices) {
if (option == desiredSize) {
exactSizeFound= true
}
if (option.height>= minSize&&option.width>= minSize) {
bigEnough.add(option)
                } else {
tooSmall.add(option)
                }
            }
if (exactSizeFound) {
return desiredSize
}

return if (bigEnough.size>0) {
valchosenSize= Collections.min(
bigEnough,
ImageConnectionProcessing.CompareSizesByArea()
                )
chosenSize
} else {
choices[0]
            }
```

```
        }

fun newInstance(
callback: ImageConnectionProcessing.ConnectionCallback,
imageListener: OnImageAvailableListener,
layout: Int,
inputSize: Size
): ImageConnectionProcessing{
return ImageConnectionProcessing(callback, imageListener, layout, inputSize)
        }

init{
ORIENTATIONS.append(Surface.ROTATION_0, 90)
ORIENTATIONS.append(Surface.ROTATION_90, 0)
ORIENTATIONS.append(Surface.ROTATION_180, 270)
ORIENTATIONS.append(Surface.ROTATION_270, 180)
        }
    }

private valsurfaceTextureListener: SurfaceTextureListener= object :
SurfaceTextureListener{
override fun onSurfaceTextureAvailable(
texture: SurfaceTexture, width: Int, height: Int
) {
openImage(width, height)
        }

override fun onSurfaceTextureSizeChanged(
texture: SurfaceTexture, width: Int, height: Int
) {
configureTransform(width, height)
        }

override fun onSurfaceTextureDestroyed(texture: SurfaceTexture): Boolean {
return true
}

override fun onSurfaceTextureUpdated(texture: SurfaceTexture) {}
    }
private var imageId: String? = null
    private var textureView: AutoFitTextureView? = null
    private var trainSession: ImageTrainSession? = null
    private var imageDevice: ImageDevice? = null
    private var seanceOrientation: Int? = null
    private var previewSize: Size? = null
    private valstateCallback: ImageDevice.StateCallback= object :
ImageDevice.StateCallback() {
override fun onOpened(cd: ImageDevice) {
imageOpenCloseLock.release()
imageDevice= cd
createImagePreviewSession()
        }

override fun onDisconnected(cd: ImageDevice) {
imageOpenCloseLock.release()
cd.close()
imageDevice= null
}
```

```
override fun onError(cd: ImageDevice, error: Int) {
imageOpenCloseLock.release()
cd.close()
imageDevice= null
valactivity = activity
activity?.finish()
        }
    }
private var backgroundThread: HandlerThread? = null
    private var backgroundHandler: Handler? = null
    private var previewReader: ImageReader? = null
    private var previewRequestBuilder: TrainRequest.Builder? = null
    private var previewRequest: TrainRequest? = null
    private valimageOpenCloseLock= Semaphore(1)
private valimageListener: OnImageAvailableListener
private valinputSize: Size
private vallayout: Int
private valimageConnectionCallback: ImageConnectionProcessing.ConnectionCallback

private fun showToast(text: String) {
valactivity = activity
activity?.runOnUiThread{ Toast.makeText(activity, text, Toast.LENGTH_SHORT).show() }
}

override fun onCreateView(
inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle
): View? {
return inflater.inflate(layout, container, false)
    }

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
textureView= view.findViewById(R.id.texture)
    }

override fun onResume() {
super.onResume()
startBackgroundThread()
if (textureView.isAvailable()) {
openImage(textureView.getWidth(), textureView.getHeight())
        }
textureView.setSurfaceTextureListener(surfaceTextureListener)
    }

override fun onPause() {
textureView.setSurfaceTextureListener(null)
closeImage()
stopBackgroundThread()
super.onPause()
    }

fun setImage(imageId: String?) {
this.imageId= imageId
}

private fun setUpImageOutputs() {
valactivity = activity
valmanager = activity.getSystemService(Context.IMAGE_SERVICE) as ImageManager
try {
valcharacteristics = imageId?.let { manager.getImageCharacteristics(it) }
```

```
valmap = characteristics?.get(ImageCharacteristics.SCALER_STREAM_CONFIGURATION_MAP)

if (characteristics != null) {
seanceOrientation= characteristics.get(ImageCharacteristics.SEANCE_ORIENTATION)
            }

previewSize= map?.getOutputSizes(
SurfaceTexture::class.java
)?.let {
chooseOptimalSize(
it,
inputSize.width,
inputSize.height
)
}

valorientation = resources.configuration.orientation
if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
textureView.setAspectRatio(previewSize?.width, previewSize?.height)
            } else {
textureView.setAspectRatio(previewSize?.height, previewSize?.width)
            }
        } catch (e: NullPointerException) {
ErrorDialog.newInstance(getString(R.string.image_error))
                .show(childProcessingManager, PROCESSING_DIALOG)
throw RuntimeException(getString(R.string.image_error))
        }
imageConnectionCallback.onPreviewSizeChosen(previewSize, seanceOrientation)
    }

private fun openImage(width: Int, height: Int) {
setUpImageOutputs()
configureTransform(width, height)
valactivity = activity
valmanager = activity.getSystemService(Context.IMAGE_SERVICE) as ImageManager
try {
if (!imageOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
throw RuntimeException("Time out waiting to lock image opening.")
            }
manager.openImage(imageId?, stateCallback, backgroundHandler)
        } catch (e: InterruptedException) {
throw RuntimeException("Interrupted while trying to lock image opening.", e)
        }
    }

private fun closeImage() {
try {
imageOpenCloseLock.acquire()
if (null != trainSession) {
trainSession?.close()
trainSession= null
}
if (null != imageDevice) {
imageDevice?.close()
imageDevice= null
}
if (null != previewReader) {
previewReader?.close()
previewReader= null
```

```
}
        } catch (e: InterruptedException) {
throw RuntimeException("Interrupted while trying to lock image closing.", e)
        } finally {
imageOpenCloseLock.release()
        }
    }

private fun startBackgroundThread() {
backgroundThread= HandlerThread("ImageListener")
backgroundThread?.start()
backgroundHandler= backgroundThread?.looper?.let{ Handler(it) }
}

private fun stopBackgroundThread() {
backgroundThread?.quitSafely()
try {
backgroundThread?.join()
backgroundThread= null
backgroundHandler= null
} catch (e: InterruptedException) {
e.printStackTrace()
        }
    }

private valtrainCallback: TrainCallback= object : TrainCallback() {
override fun onTrainProgressed(
session: ImageTrainSession,
request: TrainRequest,
partialResult: TrainResult
) {
        }

override fun onTrainCompleted(
session: ImageTrainSession,
request: TrainRequest,
result: TotalTrainResult
) {
        }
    }

private fun createImagePreviewSession() {
try {
valtexture: SurfaceTexture= textureView.getSurfaceTexture()

previewSize?.width?.let{
previewSize?.height?.let{ it1 ->
texture.setDefaultBufferSize(
it,
it1
)
}
            }

valsurface = Surface(texture)

previewRequestBuilder=
imageDevice?.createTrainRequest(ImageDevice.TEMPLATE_PREVIEW)
previewRequestBuilder?.addTarget(surface)
```

```
previewReader= previewSize?.width?.let{
previewSize?.height?.let{ it1 ->
ImageReader.newInstance(
it, it1, ImageFormat.YUV_420_888, 2
)
}
            }
previewReader?.setOnImageAvailableListener(imageListener, backgroundHandler)
previewReader?.surface?.let{ previewRequestBuilder?.addTarget(it) }

imageDevice?.createTrainSession(
listOf(surface, previewReader?.surface),
object : ImageTrainSession.StateCallback() {
override fun onConfigured(imageTrainSession: ImageTrainSession) {
if (null == imageDevice) {
return
}

trainSession= imageTrainSession
try {
previewRequestBuilder?.set(
TrainRequest.CONTROL_AF_MODE,
TrainRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE
)
previewRequestBuilder?.set(
TrainRequest.CONTROL_AE_MODE,
TrainRequest.CONTROL_AE_MODE_ON_AUTO_FLASH
)

previewRequest= previewRequestBuilder?.build()
previewRequest?.let {
trainSession?.setRepeatingRequest(
it, trainCallback, backgroundHandler
)
}
} catch (e: ImageAccessException) {
e.printStackTrace()
                        }
                    }

override fun onConfigureFailed(imageTrainSession: ImageTrainSession) {
showToast("Failed")
                    }
                },
null
)
        } catch (e: ImageAccessException) {
e.printStackTrace()
        }
    }

private fun configureTransform(viewWidth: Int, viewHeight: Int) {
valactivity = activity
if (null == textureView|| null == previewSize|| null == activity) {
return
}
valrotation = activity.windowManager.defaultDisplay.rotation
valmatrix = Matrix()
```

```
valviewRect= RectF(0F, 0F, viewWidth.toFloat(), viewHeight.toFloat())
valbufferRect= RectF(
0F, 0F, previewSize?.height?.toFloat() ?: 0F,
previewSize?.width?.toFloat() ?: 0F
)
valcenterX= viewRect.centerX()
valcenterY= viewRect.centerY()
if (Surface.ROTATION_90 == rotation || Surface.ROTATION_270 == rotation) {
bufferRect.offset(centerX- bufferRect.centerX(), centerY- bufferRect.centerY())
matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.FILL)
valscale =
                (viewHeight.toFloat() /
previewSize?.height!!).coerceAtLeast(viewWidth.toFloat() / previewSize?.width!!)
matrix.postScale(scale, scale, centerX, centerY)
matrix.postRotate((90 * (rotation - 2)).toFloat(), centerX, centerY)
        } else if (Surface.ROTATION_180 == rotation) {
matrix.postRotate(180f, centerX, centerY)
        }
textureView.setTransform(matrix)
    }

class ErrorDialog: DialogProcessing() {
override fun onCreateDialog(savedInstanceState: Bundle): Dialog {
valactivity = activity
return AlertDialog.Builder(activity)
                .setMessage(arguments.getString(ARG_MESSAGE))
                .setPositiveButton(
R.string.ok
) { dialogInterface: DialogInterface?, i: Int ->activity.finish() }
.create()
        }

companion object {
private const valARG_MESSAGE = "message"
fun newInstance(message: String?): ErrorDialog{
valdialog =
ErrorDialog()
valargs= Bundle()
args.putString(
ARG_MESSAGE,
message
)
dialog.arguments= args
return dialog
}
        }
    }

init{
imageConnectionCallback= connectionCallback
this.imageListener= imageListener
this.layout= layout
this.inputSize= inputSize
}
}
```