Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка системи ідентифікації людини за зображенням.

Виконав(ла): студент(ка) 4 курсу, групи СНс-41
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Стельмах Б.В. (прізвище та ініціали)

Керівник (підпис) Матійчук Л.П. (прізвище та ініціали)
Нормоконтроль (підпис) Липак Г.І. (прізвище та ініціали)
Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)
Рецензент (підпис) Жаровський Р.О. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
"8" червня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Стельмах Богдан Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розробка системи ідентифікації людини за зображенням.**

Керівник роботи **к.т.н., доц. Боднарчук І.О.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від "*14*" *травня* 2026 року № *4/9-237*

2. Термін подання студентом завершеної роботи 26 червня 2026 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ; Аналіз предметної області та огляд технологій; Загальна характеристика задачі розпізнавання облич; Серверна та локальна обробка біометричних даних; Детектування облич на зображенні; Формування векторних ознак обличчя; Векторні бази даних та пошук найближчого сусіда; Виявлення підробок (anti-spoofing); Висновки до першого розділу; Проєктування системи розпізнавання облич; Функціональні та нефункціональні вимоги; Загальна архітектура застосунку; Структура програмного проєкту; Реалізація модуля детектування облич; Реалізація модуля формування вкладень; Реалізація роботи з векторною базою даних; Реалізація сценарію розпізнавання; Обробка кадрів камери та відображення результатів; Обробка помилок та граничних випадків; Оцінювання продуктивності; Охорона праці та безпека в надзвичайних ситуаціях; Висновки; Список використаних джерел; Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Титульний слайд; Актуальність теми; Об'єкт, предмет і методи дослідження; Конвеєр розпізнавання облич; Архітектура застосунку; Проєктування векторної бази даних; Алгоритм реєстрації особи; Розпізнавання у реальному часі та метрика; Сценарії використання та інтерфейс; Тестування та продуктивність; Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026 – 27.01.2026	*Виконано*
2.	Підбір джерел по темі роботи	28.01.2026 – 01.04.2026	*Виконано*
3.	Оформлення першого розділу	15.04.2026	*Виконано*
4.	Оформлення другого розділу	20.04.2026	
5.	Оформлення третього розділу	30.04.2026	*Виконано*
6.	Виконання завдання до підрозділу "Безпека		
7.	життєдіяльності, основи охорони праці"	15.05.2026	*Виконано*
8.	Оформлення кваліфікаційної роботи	07.06.2026	*Виконано*
9.	Перевірка на плагіат	07.06.2026	*Виконано*
10.	Нормоконтроль	09.06.2026	*Виконано*
11.	Попередній захист кваліфікаційної роботи	11.06.2026	*Виконано*
12.	Захист кваліфікаційної роботи	28.06.2026	
13.			
14.			
15.			

Студент _____ Стельмах Б.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Розробка системи ідентифікації людини за зображенням" // Кваліфікаційна робота освітнього рівня "Бакалавр" // Стельмах Богдан Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-41 // Тернопіль, 2026 // с. – 67, рис. – 9, таблиць – 5, джерел – 33, додатків – 4, сторінок додатків – 12.

Ключові слова: розпізнавання облич, FaceNet, TensorFlow Lite, обробка на пристрої, векторна база даних, ObjectBox, MediaPipe, Android, згорткова нейронна мережа, біометрична ідентифікація

Кваліфікаційна робота присвячена проєктуванню та програмній реалізації мобільного застосунку для операційної системи Android, що виконує розпізнавання облич повністю на пристрої користувача без використання серверної інфраструктури. Актуальність теми зумовлена зростанням попиту на біометричні системи ідентифікації в галузях обліку відвідуваності, електронної ідентифікації клієнтів та контролю доступу, а також посиленням вимог до конфіденційності персональних даних, що робить локальну обробку біометричних ознак переважним підходом.

Метою роботи є створення працездатного застосунку, який детектує обличчя у відеопотоці камери, формує векторні ознаки облич за допомогою згорткової нейронної мережі FaceNet, виконаної у середовищі TensorFlow Lite, зберігає ці ознаки у вбудованій векторній базі даних ObjectBox та визначає особу шляхом пошуку найближчого сусіда у векторному просторі.

У першому розділі проаналізовано предметну область, підходи до детектування облич та формування ознак, а також виконано огляд

інструментальних засобів. У другому розділі описано архітектуру системи, конвеєр розпізнавання, проєктування бази даних та алгоритмічну основу. У третьому розділі наведено особливості програмної реалізації основних модулів, результати тестування та оцінювання продуктивності.

Робота містить вступ, три розділи, висновки, список використаних джерел та додатки з повним програмним кодом. Загальний обсяг основної частини становить понад 36 сторінок, робота ілюстрована рисунками, блок-схемами та таблицями.

ABSTRACT

"Development of a Human Identification System Based on Images" // Qualification work of the educational level "Bachelor" // Stelmakh Bohdan // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group CHc-41 // Ternopil, 2026 // p. – 67, fig. – 9, tables – 5, references – 33, annexes – 4, pages for annexes – 12.

Keywords: face recognition, FaceNet, TensorFlow Lite, on-device processing, vector database, ObjectBox, MediaPipe, Android, convolutional neural network

The qualification work is devoted to the design and software implementation of a mobile application for the Android operating system that performs face recognition entirely on the user's device without relying on any server infrastructure. The relevance of the topic stems from the growing demand for biometric identification systems in attendance monitoring, electronic know-your-customer procedures and access control, as well as from increasing privacy requirements that make on-device processing of biometric features the preferred approach.

The aim of the work is to build a functional application that detects faces in the camera video stream, produces facial feature vectors using the FaceNet convolutional neural network executed within the TensorFlow Lite runtime, stores these features in the embedded ObjectBox vector database and determines a person's identity through nearest-neighbor search in the vector space.

The first chapter analyses the subject domain, the approaches to face detection and feature extraction, and reviews the development tools. The second chapter describes the system architecture, the recognition pipeline, the database design and

the algorithmic foundation. The third chapter presents the implementation of the main modules, the testing results and the performance evaluation.

ЗМІСТ

ВСТУП

Актуальність теми. Розпізнавання облич є однією з найбільш затребуваних задач комп'ютерного зору, у якій за зображенням обличчя людини потрібно встановити її особу шляхом пошуку відповідності серед попередньо анотованого набору еталонних зображень. Ця технологія має широке застосування у промисловості та повсякденному житті: облік відвідуваності у навчальних закладах та на підприємствах, електронна ідентифікація клієнтів (eKYC) у банківському секторі, відеоспостереження, а також розблокування пристроїв за обличчям. Тривалий час якісне розпізнавання облич вимагало значних обчислювальних ресурсів і реалізовувалося на серверній стороні, коли зображення передавалися до віддаленого програмного інтерфейсу для порівняння. Проте такий підхід має суттєві недоліки: він потребує додаткової серверної інфраструктури, вносить затримку через мережеві виклики та підвищує ризик витоку конфіденційних біометричних даних.

Сучасний стан розвитку мобільних пристроїв та технологій машинного навчання дозволяє переносити обчислення безпосередньо на смартфон користувача. Поява легких бібліотек детектування облич, таких як Google ML Kit та MediaPipe, оптимізованих середовищ виконання нейронних мереж на кшталт TensorFlow Lite, а також високопродуктивних вбудованих баз даних із підтримкою векторного пошуку, як-от ObjectBox, дає змогу побудувати повноцінний застосунок, що виконує розпізнавання облич повністю на пристрої. Такий підхід забезпечує низьку затримку, працездатність без підключення до мережі та значно зменшує ризики, пов'язані з передаванням персональних даних. Саме цим зумовлена актуальність обраної теми.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розроблення мобільного застосунку для платформи Android, який виконує розпізнавання облич на пристрої з використанням згорткової нейронної

мережі FaceNet та векторної бази даних. Для досягнення поставленої мети необхідно вирішити такі завдання:

1) проаналізувати предметну область розпізнавання облич, сфери застосування та наявні підходи до реалізації;

2) дослідити методи детектування облич на зображенні та у відеопотоці, а також методи формування векторних ознак облич;

3) обґрунтувати вибір інструментальних засобів та технологій для реалізації застосунку з обробкою даних на пристрої;

4) спроєктувати архітектуру системи, конвеєр розпізнавання та схему бази даних;

5) реалізувати програмні модулі детектування, формування ознак, збереження та пошуку у векторній базі;

6) провести тестування застосунку та оцінити його продуктивність.

Об'єкт дослідження – процес автоматичної ідентифікації особи за зображенням обличчя на мобільному пристрої.

Предмет дослідження – методи, алгоритми та програмні засоби розпізнавання облич із обробкою даних безпосередньо на пристрої під управлінням ОС Android.

Методи дослідження. У роботі використано методи теорії згорткових нейронних мереж, методи цифрової обробки зображень, метричні методи порівняння векторів (косинусна подібність), методи пошуку найближчого сусіда у векторному просторі, а також принципи об'єктно-орієнтованого та компонентного проєктування програмного забезпечення.

Практичне значення. Розроблений застосунок може бути використаний як основа для систем обліку відвідуваності, контролю доступу та інших біометричних рішень, що потребують роботи без постійного підключення до мережі та підвищеного рівня захисту персональних даних. Програмний код побудований за сучасними архітектурними принципами та є модульним, що спрощує його подальше розширення.

Наукова новизна та практична цінність. Наукова складова роботи полягає в обґрунтуванні та практичній перевірці конвеєра розпізнавання облич, що повністю виконується на пристрої та поєднує сучасні легкі моделі детектування, згорткову мережу формування ознак та векторну базу даних із наближеним пошуком найближчих сусідів. На відміну від рішень із серверним інференсом, запропонований підхід усуває залежність від мережі та забезпечує локальне зберігання біометричних ознак. Практична цінність полягає у створенні працездатного застосунку з модульною архітектурою, придатного до використання як основа для систем обліку відвідуваності, контролю доступу та електронної ідентифікації.

Апробація результатів. Працездатність розробленого застосунку перевірено шляхом функціонального тестування за основними сценаріями використання та оцінювання продуктивності окремих етапів конвеєра. Результати тестування підтвердили коректність роботи системи та її придатність для розпізнавання облич у реальному часі на мобільному пристрої.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі здійснено аналіз предметної області та огляд технологій. Другий розділ присвячено проєктуванню системи. У третьому розділі описано програмну реалізацію та результати тестування. Повний програмний код наведено у додатках.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ТЕХНОЛОГІЙ

1.1 Загальна характеристика задачі розпізнавання облич

Розпізнавання облич – це задача, у якій особу людини потрібно визначити за зображенням її обличчя шляхом пошуку відповідності серед анотованого набору еталонних зображень, що зберігаються у базі даних. На відміну від простого детектування обличчя, яке відповідає лише на питання «чи присутнє обличчя на зображенні та де саме воно розташоване», розпізнавання передбачає встановлення відповідності між виявленим обличчям та конкретним записом про особу.

Технологію розпізнавання облич можна розглядати як засіб автентифікації особи через зіставлення зображення з еталоном, що зберігається у базі даних. Серед типових сценаріїв застосування виокремлюють кілька основних напрямів, які детальніше розглянуто нижче.

Облік відвідуваності в навчальних закладах та офісах. Надійний застосунок розпізнавання облич здатний замінити сканування відбитків пальців або ідентифікаційних карток для фіксації щоденної присутності працівників на підприємстві, співробітників в офісі або студентів у навчальному закладі. У поєднанні з виявленням «живості» обличчя суттєво зменшується ймовірність фіктивної відмітки присутності, а потреба у спеціалізованому апаратному забезпеченні (сенсорах чи сканерах) зникає.

Електронна ідентифікація клієнтів (eKYC). Це процес, у якому уповноважені організації перевіряють і підтверджують персональні дані клієнта для надання певної послуги. Зіставлення обличчя клієнта під час процедури eKYC із зображенням, наведеним в офіційних документах, можна легко виконати за допомогою технології розпізнавання облич для великої кількості документів.

Аналітика поведінки клієнтів. Шляхом ідентифікації відвідувачів торгового центру або магазину можна персоналізувати набір послуг на основі історії покупок. Оскільки системи розпізнавання облич можуть слугувати засобом автентифікації, стають можливими рішення без традиційної каси, коли спосіб оплати визначається безпосередньо за зображенням обличчя.

Відеоспостереження з розпізнаванням облич. Цей напрям передбачає використання камер безпеки та програмного забезпечення для ідентифікації людей шляхом аналізу відеопотоку та рис обличчя. Він може допомагати у виявленні осіб під час розслідувань або в моніторингу поведінки відвідувачів.

Спільною рисою наведених сценаріїв є потреба у надійному, швидкому та захищеному механізмі зіставлення обличчя з еталоном. Виконання розпізнавання на портативному пристрої, такому як смартфон, є вигідним для застосувань на кшталт обліку відвідуваності чи розблокування за обличчям, оскільки не потребує спеціалізованого обладнання та може супроводжуватися й оновлюватися так само, як будь-який інший застосунок на пристрої.

1.2 Серверна та локальна обробка біометричних даних

Історично системи розпізнавання облич реалізовувалися із застосуванням серверного інференсу, коли зображення облич надсилаються до віддаленого програмного інтерфейсу для перевірки їх схожості. Такий підхід є цілком прийнятним для низки сценаріїв, проте має суттєві обмеження. По-перше, він потребує додаткового серверного сервісу, який необхідно розгортати, підтримувати та масштабувати. По-друге, мережеві виклики вносять затримку у роботу застосунку, що є критичним для систем, які працюють із живим відеопотоком камери. Необхідність обробки кадрів у реальному часі фактично унеможливлює серверний інференс для таких сценаріїв.

Виконання розпізнавання безпосередньо на пристрої дозволяє досягти низької затримки та зменшити ризик витоку даних, оскільки біометричні ознаки не залишають пристрій користувача. Побудова застосунку, що виконує розпізнавання облич на пристрої, є складним завданням, оскільки вимагає опрацювання кількох ключових питань: як детектувати обличчя із зображення або живого відеопотоку, де ефективно зберігати ознаки облич, щоб забезпечити швидкий і пам'яттєощадний пошук, і яким чином за двома зображеннями облич стверджувати їхню спільну ідентичність.

Сучасний стан розвитку платформи Android та машинного навчання дає змогу відповісти на всі ці питання. Бібліотеки Google ML Kit та MediaPipe забезпечують надзвичайно швидке детектування облич для Android, iOS та вебзастосунків. База даних ObjectBox є перспективною вбудованою NoSQL-системою з підтримкою векторного індексування. Нейромережа FaceNet формує вкладення (embeddings), які можна порівнювати для визначення особи за зображенням обличчя, а середовище TensorFlow Lite дозволяє виконувати FaceNet на Android за допомогою відповідної бібліотеки.

1.3 Детектування облич на зображенні

Детектування облич є першим етапом конвеєра розпізнавання. Його завдання – знайти на вхідному зображенні всі області, що містять обличчя, та повернути координати їхніх обмежувальних прямокутників (bounding boxes). Якість і швидкодія цього етапу безпосередньо впливають на роботу всієї системи, адже подальші модулі обробляють лише обрізані області облич.

Для мобільних застосунків доцільно використовувати спеціалізовані легкі детектори. У роботі розглянуто два таких засоби. MediaPipe Face Detector ґрунтується на моделі BlazeFace, оптимізованій для роботи на процесорах мобільних пристроїв у реальному часі. Модель постачається у форматі TensorFlow Lite та зберігається у каталозі ресурсів застосунку. Альтернативою

є Google ML Kit, що надає високорівневий програмний інтерфейс детектування облич із додатковими можливостями, такими як визначення орієнтиру обличчя та оцінювання ймовірності усмішки чи відкритості очей.

Важливою вимогою на етапі реєстрації особи є наявність рівно одного обличчя на вхідному зображенні. Оскільки кожне зображення прив'язується до імені однієї особи, для спрощення накладається обмеження: зображення має містити лише одне обличчя. У разі виявлення кількох облич або їх відсутності застосунок повертає відповідну помилку та не реєструє запис.

1.4 Формування векторних ознак обличчя

Ключовим елементом сучасних систем розпізнавання облич є перетворення зображення обличчя у компактне числове представлення – вектор ознак, або вкладення (face embedding). Таке вкладення фіксує суттєві характеристики обличчя, як-от відстань між очима, форму щелепи тощо, і має властивість, що вектори облич однієї людини розташовані у просторі ознак ближче один до одного, ніж вектори облич різних людей.

У роботі для формування вкладень використано модель FaceNet. Це згорткова нейронна мережа, яка приймає на вхід зображення обличчя фіксованого розміру та повертає вектор-вкладення. Модель FaceNet навчається таким чином, щоб мінімізувати відстань між векторами облич однієї особи та максимізувати відстань між векторами облич різних осіб, що робить отримані вектори придатними для прямого порівняння за метрикою. Загальну схему перетворення зображення у вектор-вкладення наведено на рисунку 1.1.

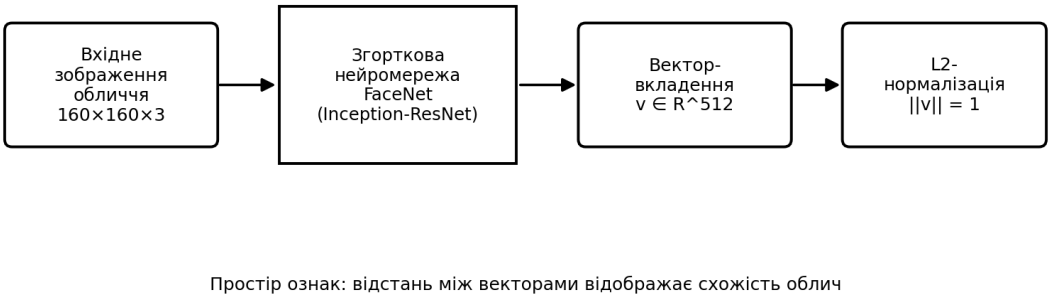

Рисунок 1.1 – Формування вектора-вкладення обличчя моделлю FaceNet

У застосунку використано варіант моделі FaceNet, що формує вкладення розмірності 512. Перед подаванням зображення до моделі виконується попереднє оброблення: масштабування до розміру 160×160 пікселів та нормалізація значень пікселів. Вхідним розміром зображення для моделі FaceNet є 160 пікселів, а розмірність вихідного вкладення становить 512 елементів. Така розмірність забезпечує достатню розрізнювальну здатність для надійного розпізнавання при прийнятних вимогах до пам'яті та обчислень.

1.5 Векторні бази даних та пошук найближчого сусіда

Після формування вкладень їх потрібно зберігати так, щоб забезпечити швидкий пошук найбільш схожого вектора серед усіх збережених. Класичні реляційні бази даних, такі як SQLite, не призначені для ефективного пошуку за схожістю у багатовимірному просторі. Тому у роботі використано спеціалізовану вбудовану базу даних ObjectBox, що підтримує векторне індексування.

ObjectBox є перспективною вбудованою NoSQL-системою, що за швидкодією перевершує SQLite та добре пристосована для зберігання даних на мобільних пристроях. Вона надає векторний індекс на основі алгоритму HNSW (Hierarchical Navigable Small World), що дозволяє виконувати наближений пошук найближчих сусідів (ANN) із високою швидкістю навіть

за великої кількості записів. За потреби база також підтримує точний лінійний пошук, коли важлива максимальна точність, а не швидкодія.

Сутність пошуку полягає в тому, що для нового зображення обличчя (запиту) формується вектор-запит, після чого виконується пошук найближчого сусіда серед збережених вкладень. Якщо знайдений найближчий сусід є достатньо близьким згідно з обраним порогом, система класифікує особу у кадрі як особу, пов'язану з цим сусідом; інакше особа вважається невпізнаною.

1.6 Виявлення підробок (anti-spoofing)

Системи розпізнавання облич уразливі до атак із використанням підробок – фотографій, відеозаписів або масок, які зловмисник демонструє камері замість справжнього обличчя. Для підвищення надійності у застосунку передбачено модуль виявлення підробок, що визначає, чи належить обличчя у кадрі живій людині. Цей модуль використовує спеціалізовану нейромережу, яка аналізує текстуру та інші особливості зображення обличчя і повертає оцінку «живості».

У реалізації застосовано модель родини MiniFASNet, ваги якої конвертовано у формат TensorFlow Lite. Модель отримує дві версії обрізаного зображення обличчя, масштабованого з різними коефіцієнтами, і на основі їхнього аналізу формує підсумкову оцінку. Якщо оцінка свідчить про підробку, результат розпізнавання відповідно позначається, що дозволяє відхиляти спроби фіктивної ідентифікації.

1.7 Параметри орієнтації обличчя

Окрім обмежувального прямокутника та оцінки живості, детектори облич часто повертають параметри просторової орієнтації обличчя – кути рисання (yaw), нахилу (pitch) та крену (roll). Ці параметри корисні для

оцінювання якості зображення обличчя та для відбору кадрів, придатних для розпізнавання. Геометричне представлення цих кутів наведено на рис. 1.2.

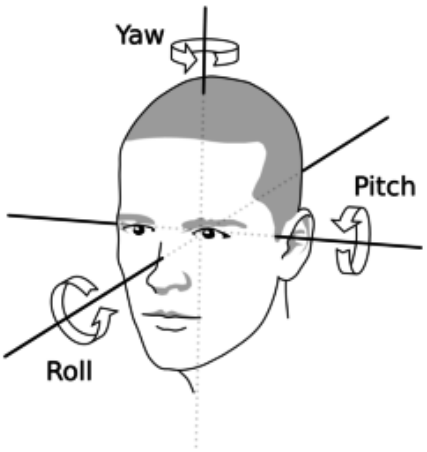

Рисунок 1.2 – Кути рисання (yaw), нахилу (pitch) та крену (roll) обличчя

Кут рисання (yaw) вказує ступінь повороту навколо вертикальної осі, що проходить крізь голову; поворот голови вбік змінює yaw. Кут нахилу (pitch) вимірює ступінь повороту навколо осі, що проходить через обидва вуха: погляд угору робить pitch від'ємним, а погляд униз – додатним. Кут крену (roll) вимірює ступінь повороту навколо осі, що проходить через ніс та потилицю, тобто бічний нахил обличчя. Для прямого погляду в камеру всі три кути близькі до нуля.

1.8 Огляд інструментальних засобів реалізації

Для реалізації застосунку обрано сучасний стек технологій розробки під Android. Основною мовою програмування є Kotlin, що є рекомендованою мовою для розробки Android-застосунків і забезпечує лаконічність та безпеку коду. Користувацький інтерфейс побудовано з використанням декларативного інструментарію Jetpack Compose.

Для роботи з камерою застосовано бібліотеку CameraX, що надає уніфікований інтерфейс доступу до камери та аналізу кадрів через компонент ImageAnalysis. Узагальнений перелік технологій наведено у таблиці 1.1.

Таблиця 1.1 – Інструментальні засоби реалізації застосунку

Складова системи	Технологія / бібліотека	Призначення
Мова програмування	Kotlin	Основна логіка застосунку
Інтерфейс користувача	Jetpack Compose	Декларативна побудова UI
Робота з камерою	CameraX	Прев'ю та аналіз кадрів
Детектування облич	MediaPipe / ML Kit	Пошук облич у кадрі
Формування ознак	FaceNet + TensorFlow Lite	Обчислення вкладень
Зберігання даних	ObjectBox	Векторна база, пошук NN
Виявлення підробок	MiniFASNet (TFLite)	Перевірка живості обличчя
Впровадження залежностей	Koin	Керування об'єктами

Детектування облич реалізовано засобами MediaPipe та ML Kit, формування вкладень – за допомогою TensorFlow Lite, зберігання та пошук – через ObjectBox.

1.9 Огляд існуючих рішень та порівняльний аналіз

Перш ніж проєктувати власну систему, доцільно розглянути наявні підходи та готові рішення у сфері розпізнавання облич на мобільних пристроях. Умовно їх можна поділити на три групи: хмарні сервіси, комерційні SDK з обробкою на пристрої та рішення з відкритим кодом на основі вільних бібліотек машинного навчання.

Хмарні сервіси, такі як програмні інтерфейси розпізнавання облич великих постачальників хмарних обчислень, забезпечують високу точність та не потребують виконання важких моделей на пристрої. Проте вони мають притаманні серверному підходу недоліки: залежність від мережевого

з'єднання, затримку, плату за використання та, найголовніше, передавання біометричних даних на сторонні сервери, що ускладнює дотримання вимог конфіденційності. Для застосувань, де важливі автономність та захист персональних даних, цей підхід є малопридатним.

Комерційні SDK з обробкою на пристрої, як-от рішення, що поєднують детектування облич та виявлення живості в єдиному пакеті, усувають потребу у мережевих викликах та забезпечують вбудовану перевірку живості. Їх перевагою є простота інтеграції та оптимізованість, однак вони зазвичай потребують ліцензійного ключа, є закритими щодо внутрішньої реалізації та обмежують можливості налаштування. Розпізнавання у таких SDK часто супроводжується поверненням додаткових параметрів обличчя – оцінки живості та кутів орієнтації.

Рішення з відкритим кодом, побудовані на вільних бібліотеках (MediaPipe, TensorFlow Lite, ObjectBox), надають максимальну гнучкість та прозорість. Розробник самостійно складає конвеєр із детектора облич, моделі формування ознак та векторної бази, повністю контролюючи кожен етап.

Таблиця 1.2 – Порівняння підходів до розпізнавання облич

Критерій	**Хмарний сервіс**	**Комерційний SDK**	**Відкритий код**
Обробка на пристрої	Ні	Так	Так
Робота без мережі	Ні	Так	Так
Захист даних	Низький	Високий	Високий
Гнучкість налаштування	Середня	Низька	Висока
Ліцензійні обмеження	Так	Так	Ні

Саме цей підхід обрано у роботі, оскільки він поєднує обробку на пристрої, відсутність ліцензійних обмежень та можливість адаптації під конкретні потреби. Узагальнене порівняння підходів наведено у таблиці 1.2.

1.10 Етапи конвеєра розпізнавання облич

Узагальнюючи розглянуті технології, можна виокремити чотири послідовні етапи, з яких складається будь-яка система розпізнавання облич: детектування обличчя, вирівнювання та попереднє оброблення, формування ознак та зіставлення. Розуміння цих етапів є основою для подальшого проєктування.

На етапі детектування система знаходить обличчя на зображенні та визначає їхні обмежувальні прямокутники. Етап попереднього оброблення передбачає обрізання обличчя, відновлення коректної орієнтації за метаданими EXIF, масштабування до фіксованого розміру та нормалізацію значень пікселів. Етап формування ознак перетворює підготовлене зображення у вектор-вкладення за допомогою згорткової нейромережі. Нарешті, на етапі зіставлення обчислюється міра схожості між вектором-запитом та збереженими векторами, і за пороговим правилом приймається рішення про впізнавання.

Якість роботи кожного етапу впливає на загальну точність системи. Помилки детектування призводять до того, що обличчя взагалі не потрапляє у подальше оброблення. Неякісне попереднє оброблення – наприклад, неправильна орієнтація чи нестандартна нормалізація – спотворює вхід моделі та погіршує якість вкладень. Тому під час проєктування важливо приділити увагу узгодженості всіх етапів конвеєра, що детальніше розглянуто у другому розділі.

1.11 Аспекти конфіденційності та безпеки біометричних даних

Обличчя людини є біометричною характеристикою, а отже, належить до особливо чутливих персональних даних. На відміну від пароля, обличчя неможливо змінити у разі його компрометації, тому захист біометричних даних має першочергове значення. Обробка облич на пристрої суттєво

зменшує ризики, оскільки вихідні зображення та сформовані вкладення не залишають пристрій користувача і не передаються до зовнішніх сервісів.

Додатково варто зауважити, що у запропонованій системі база зберігає не самі зображення облич, а їхні векторні вкладення. Хоча вкладення є похідними від зображення, відновити за ними початкове зображення обличчя у явному вигляді складно, що додає певний рівень захисту. Водночас слід враховувати, що вкладення все одно є біометричними даними, тому до них застосовуються ті самі принципи обмеження доступу та локального зберігання. Виявлення підробок, описане у підрозділі 1.6, доповнює систему захистом від атак презентації, коли зловмисник намагається видати себе за іншу особу за допомогою фотографії чи відеозапису.

1.12 Математичні основи формування та порівняння вкладень

Для глибшого розуміння роботи системи доцільно розглянути математичні основи, що лежать в основі формування та порівняння векторних вкладень. Згорткова нейронна мережа FaceNet реалізує відображення з простору зображень облич у багатовимірний евклідів простір ознак. Формально це відображення можна записати як функцію, що ставить у відповідність зображенню обличчя вектор фіксованої розмірності, причому вектори навчаються так, щоб геометрична близькість у просторі ознак відповідала схожості облич.

Навчання FaceNet ґрунтується на так званій триплетній функції втрат. Під час навчання розглядаються трійки зображень: опорне (anchor), позитивне (належить тій самій особі, що й опорне) та негативне (належить іншій особі). Функція втрат прагне зробити відстань між опорним і позитивним вкладеннями меншою за відстань між опорним і негативним вкладеннями на певну величину відступу (margin). У результаті такого навчання вектори облич однієї людини групуються у компактні кластери, а вектори різних людей

віддаляються один від одного, що й робить можливим розпізнавання через пошук найближчого сусіда.

На етапі зіставлення для порівняння двох вкладень використовується косинусна подібність – міра, що дорівнює косинусу кута між векторами. Вона обчислюється як відношення скалярного добутку векторів до добутку їхніх евклідових норм. Перевагою косинусної подібності є інваріантність до масштабу векторів: важливим є лише їхній напрямок у просторі ознак, а не довжина. Це особливо доречно для нормалізованих вкладень FaceNet, де інформація про особу закодована саме у напрямку вектора. Значення косинусної подібності, близькі до одиниці, свідчать про високу схожість облич, тоді як значення, близькі до нуля, вказують на їх відмінність.

Альтернативною мірою є евклідова відстань між вкладеннями, що також широко застосовується у системах розпізнавання облич. Для нормалізованих векторів евклідова відстань та косинусна подібність пов'язані монотонною залежністю, тому обидві міри дають узгоджені результати. У запропонованій системі обрано косинусну подібність, оскільки векторний індекс ObjectBox безпосередньо підтримує косинусний тип відстані, що спрощує реалізацію та забезпечує узгодженість між збереженням і пошуком.

1.13 Особливості наближеного пошуку найближчих сусідів

Зі зростанням кількості зареєстрованих осіб лінійний перебір усіх збережених вкладень стає обчислювально витратним. Для подолання цієї проблеми застосовуються алгоритми наближеного пошуку найближчих сусідів (Approximate Nearest Neighbor, ANN), які жертвують гарантією знаходження абсолютно точного найближчого сусіда заради суттєвого пришвидшення пошуку. База ObjectBox реалізує один із найефективніших таких алгоритмів – ієрархічні навігаційні графи малого світу (HNSW).

Принцип HNSW полягає у побудові багатошарової графової структури, де вузли відповідають збереженим векторам, а ребра з'єднують близькі вектори. Верхні шари містять невелику кількість вузлів із далекосяжними зв'язками для швидкого наближення до області пошуку, а нижні шари – щільніші зв'язки для точного уточнення. Пошук починається з верхнього шару та поступово спускається донизу, на кожному кроці переходячи до сусіда, найближчого до запиту. Така структура забезпечує логарифмічну складність пошуку відносно кількості записів, що дозволяє ефективно працювати навіть із великими базами.

Якість наближеного пошуку можна регулювати параметрами, зокрема кількістю кандидатів, що розглядаються під час обходу графа. Збільшення цього параметра підвищує точність пошуку ціною певного сповільнення. У застосунку передбачено можливість перемикання між наближеним пошуком HNSW та точним лінійним пошуком: перший режим є основним і забезпечує високу швидкодію, тоді як другий може застосовуватися у сценаріях, де критично важлива максимальна точність за невеликої кількості записів.

1.14 Висновки до першого розділу

У першому розділі проаналізовано предметну область розпізнавання облич, визначено основні сфери застосування цієї технології – облік відвідуваності, електронну ідентифікацію клієнтів, аналітику поведінки та відеоспостереження. Обґрунтовано доцільність виконання розпізнавання безпосередньо на пристрої порівняно з серверним інференсом, що забезпечує низьку затримку, працездатність без мережі та захист персональних даних.

Розглянуто ключові складові конвеєра розпізнавання: детектування облич за допомогою MediaPipe та ML Kit, формування векторних ознак моделлю FaceNet у середовищі TensorFlow Lite, зберігання та пошук вкладень у векторній базі ObjectBox, а також виявлення підробок. Обґрунтовано вибір

інструментальних засобів реалізації. Отримані результати аналізу є підставою для проєктування системи, яке виконано у наступному розділі.

2 ПРОЄКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ

2.1 Функціональні та нефункціональні вимоги

Перед проєктуванням системи сформовано перелік вимог до застосунку. Функціональні вимоги визначають, які саме дії має виконувати система, а нефункціональні – якісні характеристики її роботи. Діаграму варіантів використання застосунку наведено на рисунку 2.1.

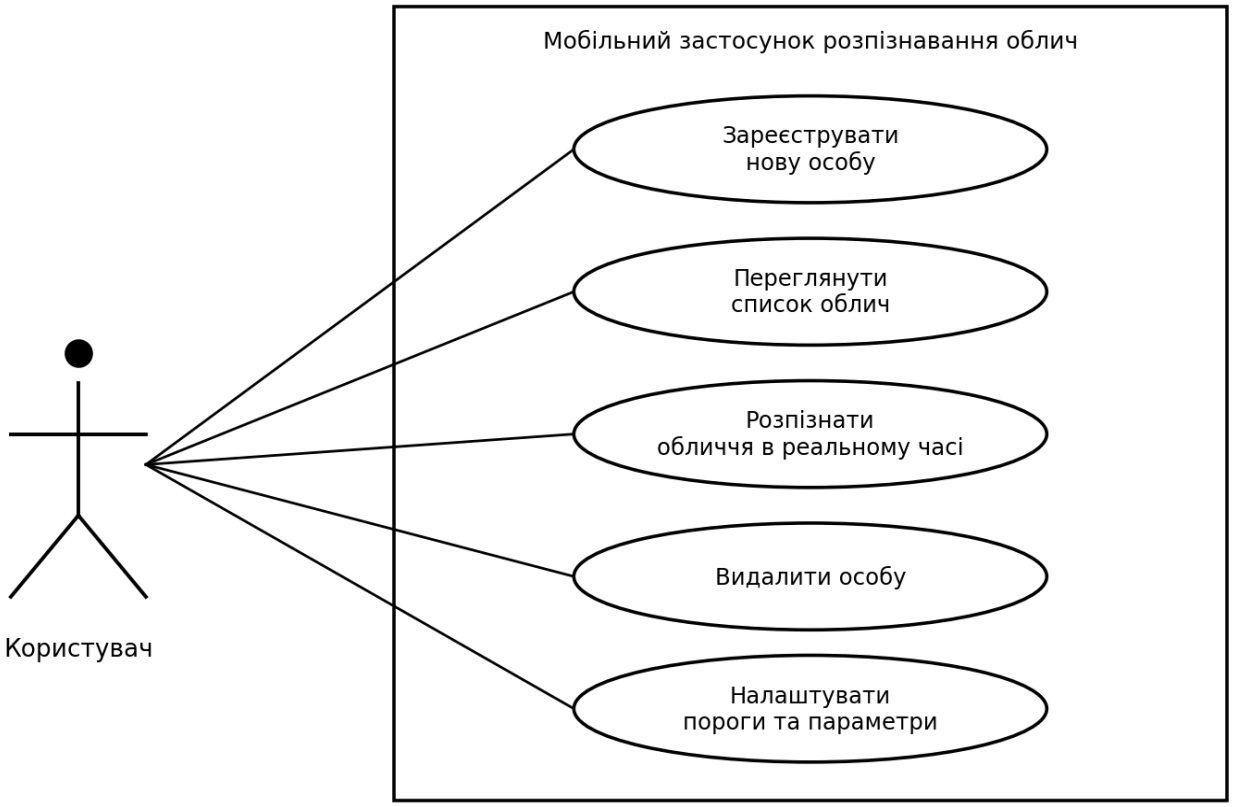

Рисунок 2.1 – Діаграма варіантів використання застосунку

До основних функціональних вимог належать наведені нижче.

1) Застосунок повинен надавати можливість зареєструвати нову особу, ввівши її ім'я та обравши одне чи кілька зображень обличчя.

2) Застосунок має детектувати обличчя на обраних зображеннях, обрізати їх та формувати векторні ознаки для збереження.

3) Застосунок повинен відображати список зареєстрованих осіб та надавати можливість видалення особи.

4) Застосунок має виконувати розпізнавання облич у реальному часі за відеопотоком камери, відображаючи рамку та ім'я впізнаної особи.

5) Застосунок повинен виконувати перевірку живості обличчя для протидії підробкам.

Нефункціональні вимоги охоплюють такі аспекти: уся обробка біометричних даних має виконуватися на пристрої без передавання даних у мережу; час реакції системи на кадр камери повинен бути достатньо малим для роботи у реальному часі; інтерфейс має бути простим та інтуїтивно зрозумілим; архітектура коду має бути модульною для спрощення супроводу та розширення.

2.2 Загальна архітектура застосунку

Архітектуру застосунку побудовано за принципом поділу на шари відповідно до сучасних рекомендацій щодо розробки Android-застосунків. Виокремлено три основні шари: рівень представлення (UI), рівень предметної області (Domain) та рівень даних (Data). Такий поділ забезпечує слабку зв'язаність компонентів, спрощує тестування та дозволяє замінювати реалізації окремих модулів без впливу на решту системи. Узагальнену структуру системи наведено на рисунку 2.2.

Рівень представлення реалізовано засобами Jetpack Compose і він містить екрани реєстрації особи, перегляду списку облич та екран розпізнавання у реальному часі. Рівень предметної області містить класи-сценарії (UseCase), що координують виконання основних операцій – додавання зображення особи до бази та розпізнавання обличчя у кадрі.

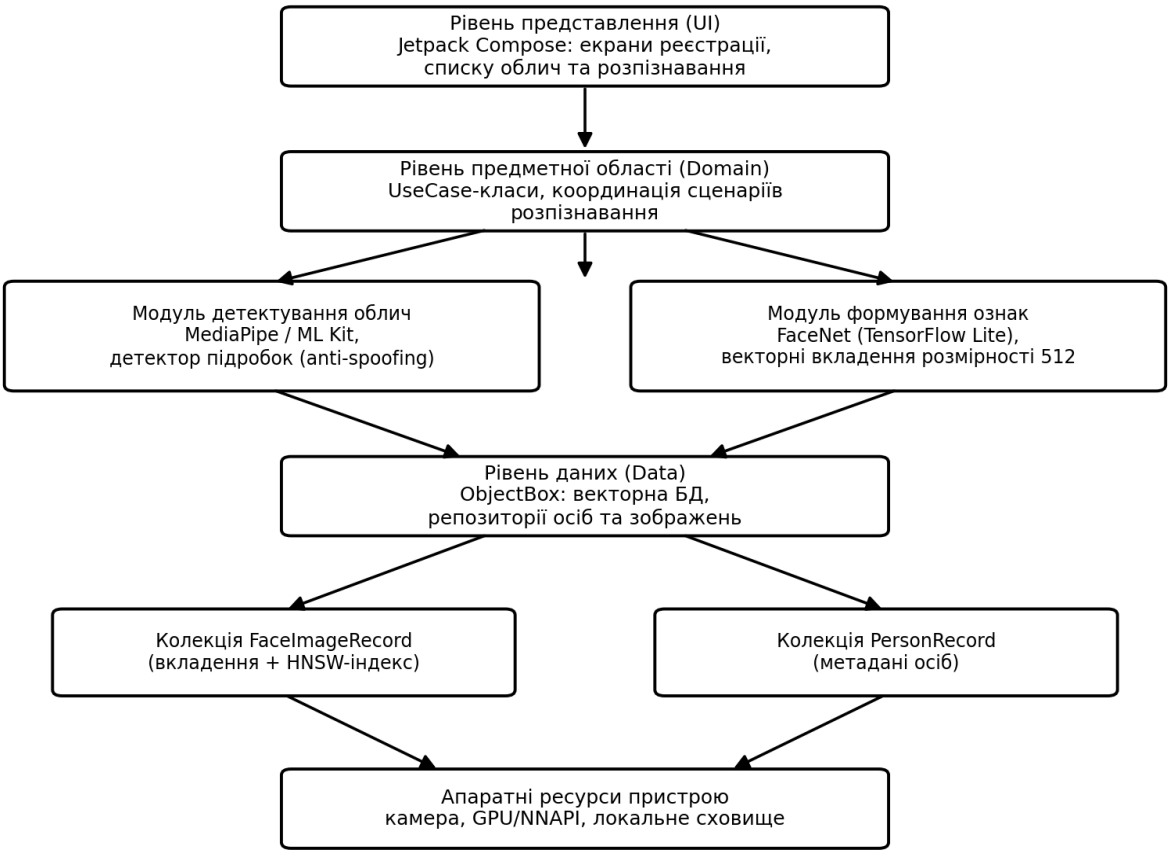

Рисунок 2.2 – Узагальнена архітектура застосунку

Рівень даних інкапсулює роботу з векторною базою ObjectBox та містить репозиторії для колекцій осіб і зображень. Для зв'язування компонентів між собою застосовано механізм впровадження залежностей на основі бібліотеки Koin.

2.3 Конвеєр розпізнавання облич

Центральним елементом системи є конвеєр розпізнавання облич, що визначає послідовність перетворень від вхідних даних до результату ідентифікації. Конвеєр складається з двох взаємопов'язаних потоків: потоку реєстрації, у якому формується база еталонних вкладень, та потоку розпізнавання, у якому виконується ідентифікація особи за кадром камери. Загальну схему конвеєра наведено на рис. 2.3.

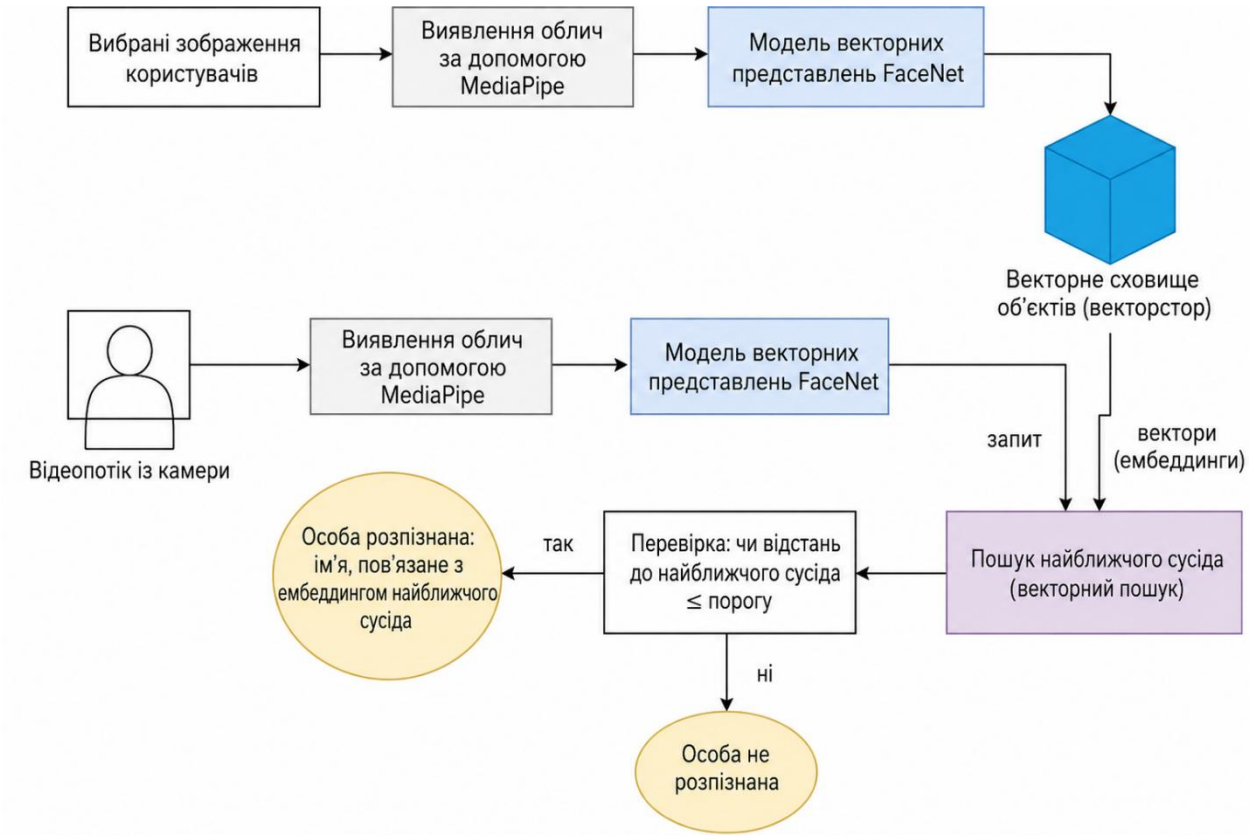

Рисунок 2.3 – Загальна схема конвеєра розпізнавання облич

Робота конвеєра відбувається у такій послідовності. Спочатку користувачеві пропонується обрати набір зображень, що належать одній особі, та ввести ім'я цієї особи як анотацію. За допомогою MediaPipe виконується детектування облич на обраних зображеннях і їх обрізання. Модель FaceNet перетворює обрізані обличчя у вектори-вкладення, які фіксують суттєві характеристики обличчя. Сформовані вкладення зберігаються у базі ObjectBox разом із даними про особу.

При появі нового обличчя перед камерою застосунок, аналогічно до етапу реєстрації, використовує MediaPipe для детектування облич у кадрі, отриманому з прев'ю камери, та обрізає обличчя. Модель FaceNet знову перетворює виявлені обличчя у вектори-вкладення. Вкладення з відеопотоку (запит) порівнюється зі збереженими у базі вкладеннями: виконується пошук найближчого сусіда для знаходження найбільш схожого збереженого вектора. Якщо найближчий сусід є достатньо близьким згідно з порогом, система

класифікує особу у кадрі як особу, пов'язану з цим сусідом; інакше особа не впізнається.

2.4 Проєктування бази даних

Для зберігання даних спроєктовано дві колекції (сутності) у базі ObjectBox. Перша колекція, FaceImageRecord, призначена для зберігання векторних вкладень облич разом із службовими полями. Друга колекція, PersonRecord, зберігає інформацію про зареєстрованих осіб. Такий поділ дозволяє пов'язувати кілька зображень (і відповідно кілька вкладень) з однією особою.

Сутність FaceImageRecord містить первинний ключ recordID, ідентифікатор особи personID, що походить із PersonRecord, ім'я особи personName та власне вектор-вкладення faceEmbedding. Поле вкладення анотовано спеціальною анотацією векторного індексу із зазначенням розмірності 512 та типу відстані – косинусної. Фрагмент опису сутності наведено нижче.

Лістинг 2.1 – Опис сутності FaceImageRecord

```
@Entity
data class FaceImageRecord(
    @Id var recordID: Long = 0,
    @Index var personID: Long = 0,
    var personName: String = "",
    @HnswIndex(
        dimensions = 512,
        distanceType = VectorDistanceType.COSINE,
    ) var faceEmbedding: FloatArray = floatArrayOf(),
)
```

Сутність PersonRecord містить первинний ключ personID, ім'я особи personName, кількість доданих зображень numImages та час додавання запису addTime.

Таблиця 2.1 – Структура колекцій бази даних

Колекція	Поле	Тип	Призначення
FaceImageRecord	recordID	Long	Первинний ключ
FaceImageRecord	personID	Long	Зовнішній ключ особи
FaceImageRecord	personName	String	Ім'я особи
FaceImageRecord	faceEmbedding	FloatArray	Вектор-вкладення (512-D)
PersonRecord	personID	Long	Первинний ключ
PersonRecord	personName	String	Ім'я особи
PersonRecord	numImages	Long	Кількість зображень
PersonRecord	addTime	Long	Час додавання

Логічну схему бази даних та зв'язки між колекціями наведено у таблиці 2.1.

2.5 Алгоритм реєстрації особи

Процес реєстрації особи реалізує наповнення бази еталонними вкладеннями. Користувач вводить ім'я особи та обирає одне чи кілька зображень через системний засіб вибору зображень (photo-picker). Кожне обране зображення зчитується у вигляді растрового зображення (Bitmap), при цьому враховуються метадані EXIF для коректного відновлення орієнтації знімка. Далі виконується детектування обличчя; за умови наявності рівно одного обличчя воно обрізається та подається до моделі FaceNet для обчислення вкладення, яке зберігається у базі. Блок-схему алгоритму реєстрації наведено на рисунку 2.4.

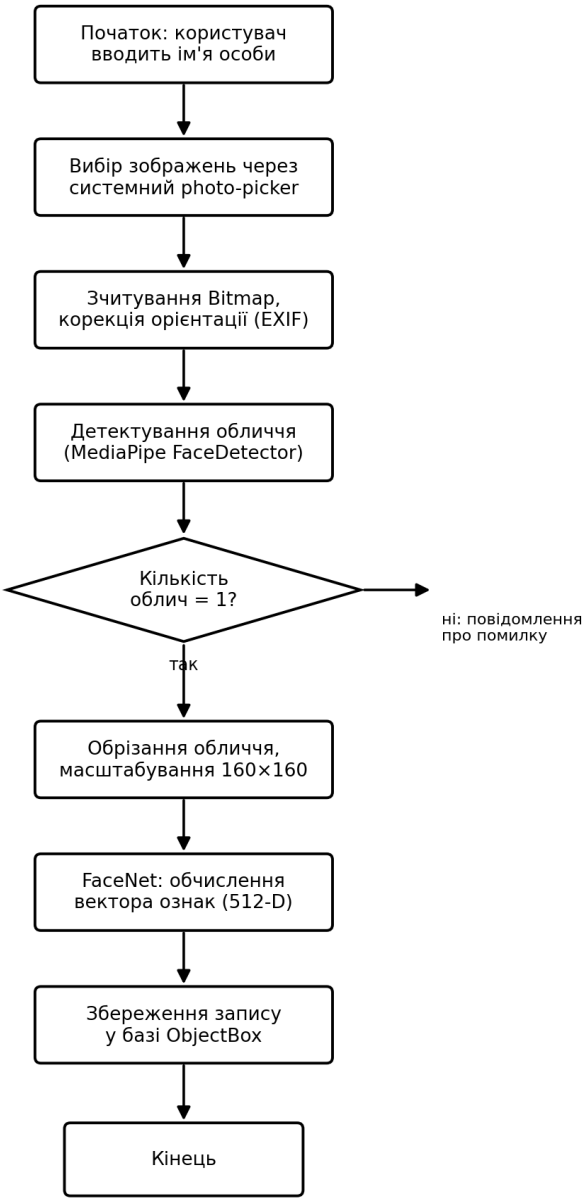

Рисунок 2.4 – Блок-схема алгоритму реєстрації особи

Якщо на зображенні виявлено більше одного обличчя або жодного, система не реєструє запис і повертає користувачеві відповідне повідомлення про помилку. Така перевірка забезпечує однозначну відповідність між зображенням та особою і запобігає потраплянню до бази некоректних вкладень.

2.6 Алгоритм розпізнавання у реальному часі

Алгоритм розпізнавання обробляє кожен кадр відеопотоку камери. Кадр, отриманий через компонент ImageAnalysis бібліотеки CameraX, перетворюється на растрове зображення, після чого виконується детектування всіх облич у кадрі. Для кожного виявленого обличчя обчислюється вектор-запит за допомогою FaceNet, після чого у векторній базі виконується пошук найближчого сусіда. Блок-схему алгоритму наведено на рис. 2.5.

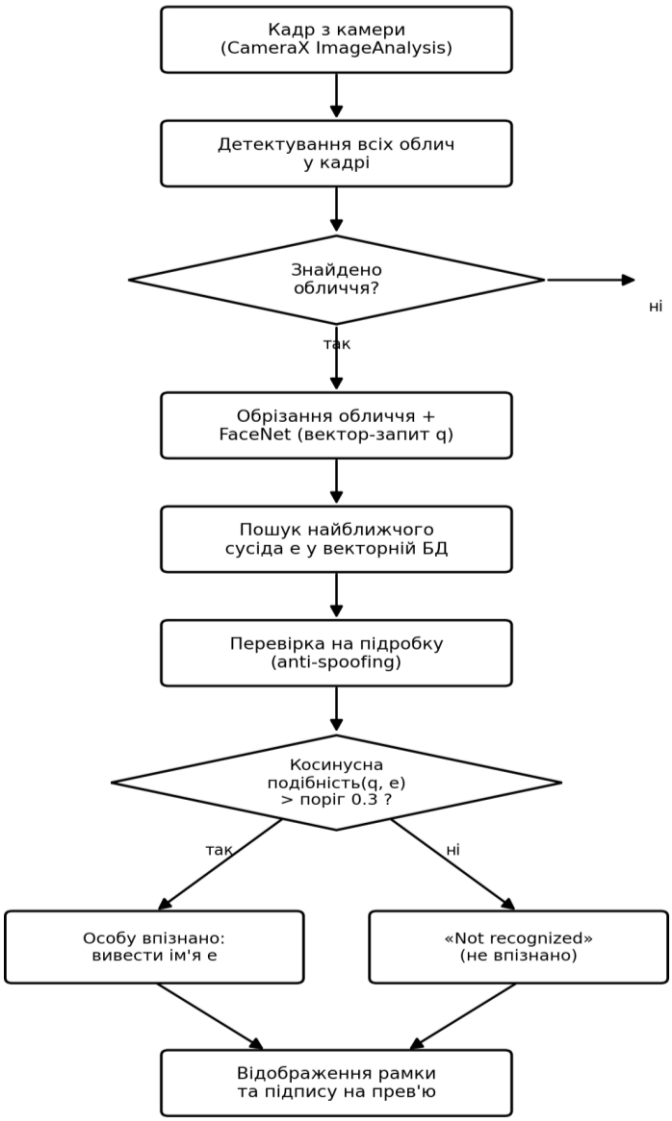

Рисунок 2.5 – Блок-схема алгоритму розпізнавання у реальному часі

Знайдений запис перевіряється на живість, а потім обчислюється косинусна подібність між запитом та найближчим сусідом.

Рішення про впізнавання приймається на основі порівняння косинусної подібності з фіксованим порогом. Якщо подібність між вектором-запитом та найближчим вкладенням перевищує поріг, особа вважається впізнаною і її ім'я відображається; інакше виводиться позначка «Not recognized». Результат – рамка обличчя з підписом – відображається поверх прев'ю камери. Описаний процес повторюється для кожного обличчя у кадрі.

2.7 Метрика порівняння вкладень

Для оцінювання схожості двох векторів-вкладень використано косинусну подібність – міру, що дорівнює косинусу кута між векторами у багатовимірному просторі. Косинусна подібність набуває значень від −1 до 1, причому значення, близьке до одиниці, свідчить про високу схожість облич, а близьке до нуля чи від'ємне – про їх відмінність. Вибір цієї метрики зумовлений тим, що вектори FaceNet після нормалізації порівнюються саме за напрямком, а не за абсолютною величиною.

Косинусна подібність двох векторів обчислюється як відношення їхнього скалярного добутку до добутку їхніх довжин (норм). У застосунку реалізовано безпосереднє обчислення цієї величини: накопичуються квадрати компонент кожного вектора для обчислення норм та попарні добутки компонент для скалярного добутку, після чого скалярний добуток ділиться на добуток норм. Поріг прийняття рішення підбирається експериментально; у базовій конфігурації застосунку він становить близько 0.3–0.4.

2.8 Обґрунтування вибору моделі FaceNet

Серед різних архітектур формування ознак облич для застосунку обрано модель FaceNet. Цей вибір зумовлений кількома чинниками. По-перше, FaceNet формує компактне вкладення фіксованої розмірності, яке безпосередньо придатне для порівняння за метрикою, без потреби у додатковому класифікаторі. По-друге, модель добре зарекомендувала себе на практиці та має доступні реалізації у форматі TensorFlow Lite, що дозволяє виконувати її на мобільних пристроях. По-третє, принцип навчання FaceNet, спрямований на зближення векторів облич однієї особи та віддалення векторів різних осіб, добре узгоджується із задачею пошуку найближчого сусіда.

Альтернативними архітектурами є моделі сімейства ArcFace та інші підходи з кутовими функціями втрат, що демонструють високу точність на еталонних наборах даних. Однак для мобільного застосунку, де важливі компактність моделі та швидкодія, FaceNet залишається збалансованим вибором. У застосунку передбачено можливість використання як 512-вимірного, так і 128-вимірного варіанта вкладень; за замовчуванням використано 512-вимірний варіант, що забезпечує вищу розрізнювальну здатність.

2.9 Структура збереження моделей та ресурсів

Нейромережеві моделі, що використовуються застосунком, зберігаються у каталозі ресурсів (assets) у форматі TensorFlow Lite. До них належать модель детектора облич BlazeFace, модель FaceNet для формування вкладень та дві моделі для виявлення підробок, що працюють із різними коефіцієнтами масштабування обличчя. Зберігання моделей безпосередньо у застосунку забезпечує його повну автономність – для роботи не потрібне завантаження моделей із мережі.

Під час першого запуску відповідних модулів моделі завантажуються у пам'ять та ініціалізують інтерпретатори TensorFlow Lite. Ініціалізація виконується одноразово, після чого інтерпретатори повторно використовуються для оброблення всіх кадрів, що дозволяє уникнути повторних витрат на завантаження. За наявності апаратного прискорення (графічного процесора або нейропроцесора) застосунок задіює відповідні делегати, що суттєво пришвидшує інференс.

2.10 Проєктування взаємодії компонентів

Для забезпечення слабкої зв'язаності компонентів у застосунку застосовано механізм впровадження залежностей на основі бібліотеки Koin. Замість того щоб кожен клас самостійно створював потрібні йому об'єкти, ці об'єкти конфігуруються централізовано та надаються класам у готовому вигляді. Так, у клас-сценарій ImageVectorUseCase впроваджуються модуль детектування, модуль виявлення підробок, векторна база та модуль формування вкладень.

Такий підхід має кілька переваг. Він спрощує заміну реалізацій – наприклад, детектор MediaPipe можна замінити на детектор ML Kit без зміни коду сценарію, оскільки обидва реалізують спільний базовий інтерфейс. Він також полегшує тестування, дозволяючи підставляти тестові реалізації залежностей. Загалом застосування впровадження залежностей разом із поділом на шари формує гнучку та супроводжувану архітектуру, що відповідає сучасним рекомендаціям щодо розробки Android-застосунків.

2.11 Потоки даних у системі

Для повного розуміння роботи застосунку важливо простежити, як дані переміщуються між компонентами системи у двох основних сценаріях – реєстрації та розпізнавання. У сценарії реєстрації вхідними даними є ім'я

особи та набір URI обраних зображень. Рівень представлення передає ці дані до моделі подання (ViewModel), яка зберігає список URI та делегує оброблення сценарію предметної області. Сценарій по черзі звертається до модуля детектування, отримуючи обрізані обличчя, потім до модуля формування вкладень, отримуючи вектори, і нарешті записує отримані вектори разом із метаданими особи до векторної бази.

У сценарії розпізнавання вхідними даними є кадри відеопотоку камери. Кожен кадр у вигляді растрового зображення надходить до сценарію розпізнавання, який спочатку звертається до модуля детектування для отримання всіх облич кадру разом з їхніми обмежувальними прямокутниками. Для кожного обличчя формується вектор-запит, який передається до векторної бази для пошуку найближчого сусіда. Знайдений запис разом із результатом перевірки живості повертається до сценарію, де приймається рішення про впізнавання. Підсумкові результати – імена осіб та координати рамок – передаються назад на рівень представлення для відображення поверх прев'ю камери.

Така організація потоків даних забезпечує однонаправлений рух інформації та чітке розмежування відповідальності між шарами. Рівень представлення не взаємодіє безпосередньо з базою даних чи нейромережами, а лише з моделлю подання та сценаріями; це відповідає принципам сучасної архітектури та полегшує супровід коду. Службові метадані, такі як час оброблення кожного етапу, збираються сценарієм розпізнавання та можуть передаватися на рівень представлення для відображення показників продуктивності.

2.12 Проєктування інтерфейсу користувача

Інтерфейс користувача спроєктовано з огляду на простоту та зрозумілість основних сценаріїв. Передбачено три ключові екрани. Головний

екран надає доступ до функцій реєстрації та розпізнавання і містить список зареєстрованих осіб. Екран реєстрації містить поле введення імені особи, кнопку вибору зображень із галереї та кнопку додавання до бази; кнопка додавання активується лише після введення імені. Екран розпізнавання відображає прев'ю камери з накладеними рамками та підписами виявлених облич.

Під час проєктування інтерфейсу враховано принципи зворотного зв'язку: користувач отримує наочні повідомлення про результат кожної дії – успішне додавання особи, помилку детектування чи результат розпізнавання. Для тривалих операцій, таких як оброблення кількох зображень під час реєстрації, передбачено індикацію стану виконання. Декларативний підхід Jetpack Compose дозволяє описувати інтерфейс як функцію від стану, що спрощує підтримання узгодженості між даними та їх відображенням і зменшує ймовірність помилок, пов'язаних із ручним оновленням елементів інтерфейсу.

2.13 Висновки до другого розділу

У другому розділі сформовано функціональні та нефункціональні вимоги до застосунку та спроєктовано його архітектуру за трирівневою схемою з поділом на рівні представлення, предметної області та даних. Розроблено конвеєр розпізнавання облич, що поєднує потоки реєстрації та ідентифікації, а також спроєктовано схему бази даних із двох колекцій – для вкладень облич та для метаданих осіб.

Розроблено та подано у вигляді блок-схем алгоритми реєстрації особи та розпізнавання у реальному часі, обґрунтовано вибір косинусної подібності як метрики порівняння вкладень та принцип прийняття рішення про впізнавання за пороговим значенням. Спроєктовані рішення є основою для програмної реалізації, опис якої наведено у третьому розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Структура програмного проєкту

Програмний проєкт організовано відповідно до спроєктованої трирівневої архітектури. Вихідний код розподілено за пакетами, що відповідають рівням системи: пакет data містить класи доступу до бази даних та опис сутностей; пакет domain містить класи-сценарії та модулі детектування облич, формування вкладень і виявлення підробок; пакет presentation містить екрани та компоненти інтерфейсу. Окремий пакет di відповідає за конфігурацію впровадження залежностей.

Такий поділ забезпечує чітку відповідальність кожного модуля та полегшує супровід. Нейромережеві моделі у форматі TensorFlow Lite (модель FaceNet, модель детектора BlazeFace та моделі виявлення підробок) зберігаються у каталозі ресурсів застосунку та завантажуються під час ініціалізації відповідних класів. Повний вихідний код основних модулів наведено у додатках А–Г.

3.2 Реалізація модуля детектування облич

Модуль детектування облич інкапсульовано у класі MediapipeFaceDetector. Під час ініціалізації клас завантажує модель детектора з каталогу ресурсів та налаштовує параметри детектування. Назва моделі та базові параметри задаються наведеним нижче чином.

Лістинг 3.1 – Ініціалізація детектора облич MediaPipe

```
    private val modelName = "blaze_face_short_range.tflite"
    private val baseOptions =
        BaseOptions.builder().setModelAssetPath(modelName).build()
    private val faceDetectorOptions =
        FaceDetector.FaceDetectorOptions.builder()
```

```
            .setBaseOptions(baseOptions)
            .setRunningMode(RunningMode.IMAGE)
            .build()
    private val faceDetector =
        FaceDetector.createFromOptions(context, faceDetectorOptions)
```

Клас надає метод обрізання обличчя із зображення за його URI. Метод зчитує растрове зображення, виконує детектування та перевіряє, що знайдено рівно одне обличчя. Логіку перевірки кількості облич та повернення відповідних помилок наведено нижче.

Лістинг 3.2 – Перевірка кількості виявлених облич

```
    val faces = faceDetector
        .detect(BitmapImageBuilder(imageBitmap).build())
        .detections()
    if (faces.size > 1) {
        return Result.failure(AppException(ErrorCode.MULTIPLE_FACES))
    } else if (faces.size == 0) {
        return Result.failure(AppException(ErrorCode.NO_FACE))
    } else {
        val rect = faces[0].boundingBox().toRect()
        // ... перевірка меж та обрізання обличчя
    }
```

Для розпізнавання у реальному часі клас також надає метод, що повертає всі обрізані обличчя кадру разом з їхніми обмежувальними прямокутниками. Перед обрізанням кожен прямокутник перевіряється на відповідність межам зображення допоміжною функцією, що запобігає виходу за межі растрового зображення. Повний код модуля наведено у додатку А.

3.3 Реалізація модуля формування вкладень

Формування векторних ознак реалізовано у класі FaceNet. Під час ініціалізації клас створює інтерпретатор TensorFlow Lite та завантажує модель. За наявності підтримки графічного прискорювача додається відповідний делегат GPU, що пришвидшує обчислення; інакше використовується багатопотокове виконання на процесорі. Основні параметри моделі – вхідний розмір зображення та розмірність вкладення – задано константами.

Лістинг 3.3 – Параметри та попереднє оброблення для FaceNet

```
// Вхідний розмір зображення для моделі FaceNet
private val imgSize = 160
// Розмірність вихідного вкладення
private val embeddingDim = 512


private val imageTensorProcessor =
    ImageProcessor.Builder()
        .add(ResizeOp(imgSize, imgSize, ResizeOp.ResizeMethod.BILINEAR))
        .add(NormalizeOp())
        .build()
```

Перед подаванням до моделі зображення масштабується до розміру 160×160 пікселів та нормалізується. Метод обчислення вкладення приймає растрове зображення обличчя, перетворює його на буфер та виконує модель, повертаючи вектор-вкладення у вигляді масиву чисел із рухомою комою. Виконання моделі винесено у фоновий потік за допомогою механізму корутин, щоб не блокувати інтерфейс користувача. Повний код наведено у додатку Б.

3.4 Реалізація роботи з векторною базою даних

Робота з векторною базою інкапсульована у двох класах: ImagesVectorDB для колекції вкладень та PersonDB для колекції осіб. Клас

ImagesVectorDB надає методи додавання запису, пошуку найближчого сусіда та видалення записів за ідентифікатором особи. Метод пошуку підтримує два режими: наближений пошук засобами векторного індексу ObjectBox та точний лінійний пошук. Фрагмент методу додавання запису та виклику наближеного пошуку наведено нижче.

Лістинг 3.4 – Додавання запису та наближений пошук найближчих сусідів

```
fun addFaceImageRecord(record: FaceImageRecord) {
    imagesBox.put(record)
}


// Наближений пошук засобами векторного індексу
return imagesBox
    .query(FaceImageRecord_.faceEmbedding
        .nearestNeighbors(embedding, 10))
    .build()
    .findWithScores()
    .map { it.get() }
    .firstOrNull()
```

У режимі точного пошуку всі записи бази перебираються лінійно, а для прискорення обчислень перебір розподіляється між кількома потоками за допомогою корутин. Для кожного запису обчислюється косинусна відстань до вектора-запиту, після чого обирається запис з найбільшою подібністю. Клас PersonDB, своєю чергою, надає методи додавання та видалення особи, підрахунку кількості записів та отримання реактивного потоку списку осіб для відображення в інтерфейсі. Повний код обох класів наведено у додатку В.

3.5 Реалізація сценарію розпізнавання

Координацію основних операцій виконує клас-сценарій ImageVectorUseCase, у який впроваджуються модулі детектування, виявлення підробок, векторна база та модуль формування вкладень. Метод додавання зображення особи послідовно виконує детектування й обрізання обличчя, обчислення вкладення та збереження запису у базі. Скорочений варіант цього методу наведено нижче.

Лістинг 3.5 – Метод додавання зображення особи до бази

```
suspend fun addImage(
    personID: Long, personName: String, imageUri: Uri
): Result<Boolean> {
    val faceDetectionResult = faceDetector.getCroppedFace(imageUri)
    if (faceDetectionResult.isSuccess) {
        val embedding =
            faceNet.getFaceEmbedding(faceDetectionResult.getOrNull()!!)
        imagesVectorDB.addFaceImageRecord(
            FaceImageRecord(
                personID = personID,
                personName = personName,
                faceEmbedding = embedding,
            ),
        )
        return Result.success(true)
    } else {
        return Result.failure(faceDetectionResult.exceptionOrNull()!!)
    }
}
```

Метод розпізнавання отримує растрове зображення кадру, детектує всі обличчя, а для кожного з них обчислює вектор-запит, виконує пошук найближчого сусіда та перевірку живості. Рішення про впізнавання

приймається порівнянням косинусної подібності з порогом. Логіку прийняття рішення наведено нижче.

Лістинг 3.6 – Прийняття рішення про впізнавання за порогом подібності

```
val distance = cosineDistance(embedding, recognitionResult.faceEmbedding)
if (distance > 0.3) {
    faceRecognitionResults.add(
        FaceRecognitionResult(recognitionResult.personName,
            boundingBox, spoofResult))
} else {
    faceRecognitionResults.add(
        FaceRecognitionResult("Not recognized",
            boundingBox, spoofResult))
}
```

Окрім результату розпізнавання, метод збирає метрики продуктивності – час детектування, формування вкладення, векторного пошуку та перевірки живості, що дозволяє – контролювати швидкодію системи. Повний код сценарію наведено у додатку Г.

3.6. Реалізація користувацького інтерфейсу

Користувацький інтерфейс застосунку побудовано засобами Jetpack Compose і він складається з кількох екранів. Головний екран надає доступ до основних функцій – реєстрації особи та запуску розпізнавання. Екран реєстрації містить поле введення імені, кнопку вибору зображень та кнопку додавання до бази. Екран списку облич відображає зареєстрованих осіб із можливістю їх видалення. Зовнішній вигляд головного екрана застосунку наведено на рис. 3.1.

Рисунок 3.1 – Головний екран застосунку

Екран розпізнавання відображає прев'ю камери, поверх якого спеціальний компонент рисує обмежувальні рамки виявлених облич та підписи з іменами впізнаних осіб. Для коректного відображення результатів координати обмежувальних прямокутників, отримані у системі координат кадру, перераховуються у систему координат екрана. Внутрішню взаємодію SDK під час ідентифікації обличчя ілюструє схема на рис. 3.2, яка узагальнює послідовність перевірок живості та подібності.

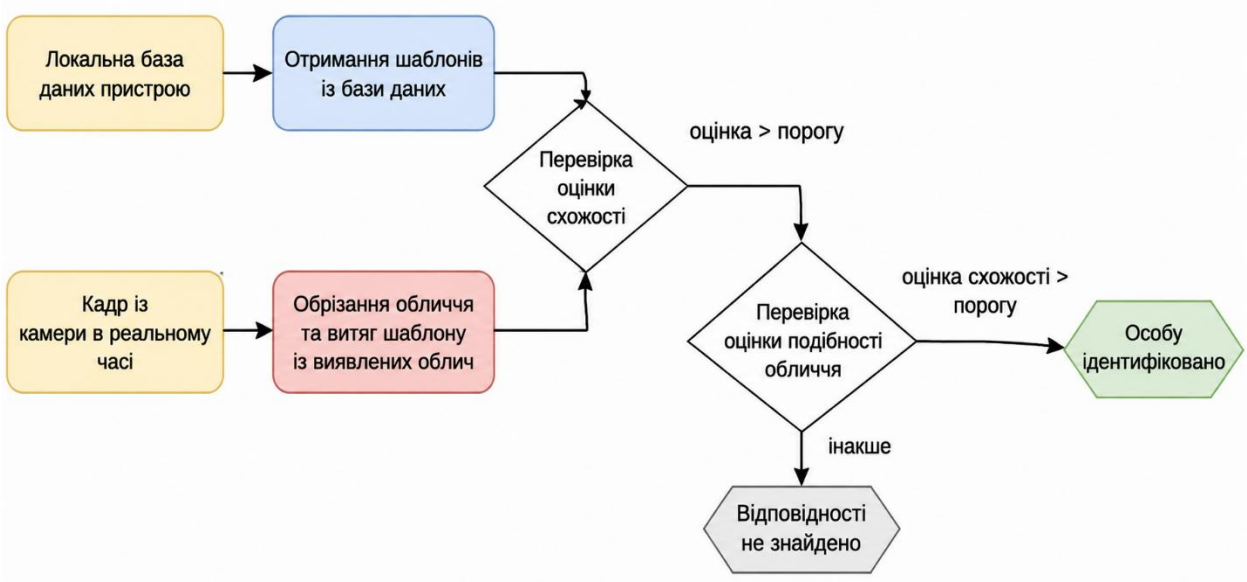

Рисунок 3.2 – Схема послідовності перевірок під час ідентифікації обличчя

3.7 Обробка кадрів камери та відображення результатів

Для отримання відеопотоку застосунок використовує бібліотеку CameraX, що надає компонент ImageAnalysis для покадрового аналізу. До цього компонента приєднується аналізатор, який для кожного кадру викликає сценарій розпізнавання. Оскільки оброблення кадру включає виконання нейромереж, воно є обчислювально витратним, тому виконується у фоновому потоці, а кадри, що надходять під час оброблення попереднього, пропускаються – це запобігає накопиченню черги та забезпечує плавність роботи.

Результати розпізнавання відображаються поверх прев'ю камери за допомогою спеціального компонента-накладки, що перевизначає метод рисування. Для кожного виявленого обличчя рисується обмежувальна рамка та підпис з іменем особи або позначкою про невпізнавання. Важливим технічним аспектом є перерахунок координат: обмежувальні прямокутники, отримані у системі координат вхідного зображення, масштабуються відповідно до розмірів екрана прев'ю з урахуванням можливого

віддзеркалення для фронтальної камери. Це забезпечує точне суміщення рамки з реальним положенням обличчя на екрані.

3.8 Обробка помилок та граничних випадків

Надійність застосунку значною мірою визначається коректним опрацюванням помилок та граничних випадків. У реалізації застосовано уніфікований механізм оброблення помилок на основі типу результату, що інкапсулює або успішне значення, або виняток. Для типових помилкових ситуацій передбачено окремі коди помилок: відсутність обличчя на зображенні, наявність кількох облич, а також загальна помилка детектора.

Під час реєстрації особи кожна помилка супроводжується зрозумілим повідомленням користувачеві, що дозволяє йому виправити вхідні дані – наприклад, обрати зображення з єдиним чітко видимим обличчям. Під час розпізнавання у реальному часі відсутність облич у кадрі не вважається помилкою: система просто не відображає рамок до появи обличчя. Такий підхід забезпечує стабільну роботу застосунку за різноманітних умов зйомки та запобігає аварійним завершенням.

3.9 Можливості подальшого вдосконалення

Розроблений застосунок є завершеним функціональним рішенням, проте має потенціал для подальшого розвитку. Серед можливих напрямів удосконалення варто відзначити кілька основних.

– Підтримка кількох зображень на особу. Зберігання кількох вкладень для однієї особи та усереднення результатів пошуку може підвищити стійкість розпізнавання до змін освітлення, ракурсу та виразу обличчя.

– Адаптивний поріг. Замість фіксованого порога косинусної подібності можна застосувати адаптивне визначення порога залежно від статистики наявних вкладень, що зменшить кількість хибних спрацювань.

– Перенесення на інші платформи. Описаний конвеєр може бути реалізований на платформах iOS, Flutter або Kotlin Multiplatform, що розширить сферу застосування рішення.

– Інтеграція з обліковими системами. Застосунок може бути доповнений модулями експорту подій розпізнавання до зовнішніх систем обліку відвідуваності чи контролю доступу із дотриманням вимог захисту даних.

Реалізація цих удосконалень не потребує зміни базової архітектури завдяки її модульності, що підтверджує вдалість обраних проєктних рішень.

3.10 Попереднє оброблення зображень та нормалізація

Важливою складовою якісного формування вкладень є коректне попереднє оброблення вхідних зображень. У модулі FaceNet для цього застосовано конвеєр оброблення зображень, що послідовно виконує дві операції: масштабування обрізаного обличчя до розміру 160×160 пікселів методом білінійної інтерполяції та нормалізацію значень пікселів. Нормалізація приводить значення пікселів до діапазону, очікуваного моделлю, що є необхідною умовою коректної роботи нейромережі, навченої на відповідно підготовлених даних.

Узгодженість попереднього оброблення між етапами реєстрації та розпізнавання є критичною: якщо зображення на цих етапах оброблятимуться по-різному, вкладення виявляться несумісними, і навіть обличчя однієї особи матимуть низьку подібність. Тому в реалізації один і той самий конвеєр оброблення застосовується в обох сценаріях. Окрему увагу приділено відновленню орієнтації зображень за метаданими EXIF на етапі реєстрації, оскільки знімки з камери часто зберігаються з поворотом, який потрібно компенсувати перед детектуванням обличчя.

Допоміжні функції модуля детектування забезпечують додаткову надійність оброблення. Функція перевірки меж обмежувального прямокутника гарантує, що область обличчя не виходить за межі зображення, перш ніж виконати обрізання; це запобігає винятковим ситуаціям під час створення растрового зображення обрізаного обличчя. Функція повороту зображення застосовується для приведення знімка до правильної орієнтації. Сукупно ці заходи забезпечують стабільну та передбачувану роботу етапу формування ознак за різноманітних вхідних даних.

3.11 Тестування застосунку

Тестування застосунку виконувалося з метою перевірки коректності роботи основних функцій та оцінювання продуктивності. Застосовано функціональне тестування за сценаріями використання та вимірювання часу виконання окремих етапів конвеєра. Перелік основних тестових сценаріїв та очікувані результати наведено у таблиці 3.1.

Таблиця 3.1 – Основні тестові сценарії

№	Сценарій	Очікуваний результат
1	Реєстрація особи з одним обличчям	Запис успішно додано до бази
2	Реєстрація зображення з кількома обличчями	Повідомлення про помилку MULTIPLE_FACES
3	Реєстрація зображення без облич	Повідомлення про помилку NO_FACE
4	Розпізнавання зареєстрованої особи	Відображено ім'я особи
5	Розпізнавання незареєстрованої особи	Позначка «Not recognized»
6	Демонстрація фотографії камері	Виявлено підробку
7	Видалення особи зі списку	Записи особи видалено з бази

Результати функціонального тестування підтвердили коректну роботу всіх основних сценаріїв. Застосунок правильно обробляє граничні випадки –

відсутність обличчя, наявність кількох облич та спроби підробки. Видалення особи коректно вилучає всі пов'язані з нею вкладення з векторної бази.

3.12 Оцінювання точності розпізнавання

Окрім швидкодії, важливою характеристикою системи розпізнавання облич є її точність. Точність визначається здатністю системи правильно впізнавати зареєстрованих осіб (істинно позитивні спрацювання) та не помилково впізнавати незареєстрованих чи інших осіб (уникнення хибно позитивних спрацювань). Ці дві властивості перебувають у певному протиріччі та регулюються пороговим значенням косинусної подібності: зниження порога підвищує чутливість, але збільшує ризик хибних спрацювань, тоді як підвищення порога робить систему суворішою.

Якість розпізнавання суттєво залежить від якості еталонних зображень, доданих під час реєстрації. Зображення з добрим освітленням, прямим поглядом у камеру та чітко видимим обличчям дають вкладення, що добре відокремлюють особу від інших. Натомість зображення з нетиповим ракурсом, поганим освітленням або частковим перекриттям обличчя погіршують розрізнювальну здатність. Тому під час реєстрації рекомендується обирати якісні зображення, а перевірка наявності рівно одного обличчя на знімку слугує первинним фільтром якості вхідних даних.

Для практичного оцінювання точності доцільно сформувати невеликий набір тестових осіб, зареєструвати їх та провести серію спроб розпізнавання за різних умов – зміни освітлення, ракурсу та виразу обличчя. Результати таких спроб дозволяють підібрати оптимальне порогове значення для конкретного сценарію використання. Проведене у роботі функціональне тестування підтвердило, що за прийнятної якості еталонних зображень та порога косинусної подібності у діапазоні 0.3–0.4 система забезпечує стабільне впізнавання зареєстрованих осіб та коректно відхиляє незареєстрованих.

3.13 Оцінювання продуктивності

Для оцінювання продуктивності виміряно час виконання основних етапів конвеєра розпізнавання: детектування обличчя, формування вкладення та векторного пошуку. Виконання моделей із застосуванням графічного прискорювача та багатопотокового оброблення дозволяє досягти прийнятної швидкодії для роботи у реальному часі. Узагальнені орієнтовні показники часу виконання етапів наведено у таблиці 3.2.

Таблиця 3.2 – Орієнтовний час виконання етапів конвеєра

Етап конвеєра	**Орієнтовний час, мс**
Детектування обличчя	10–30
Формування вкладення (FaceNet)	20–60
Векторний пошук (ANN)	1–5
Перевірка живості	15–40

Наведені значення є орієнтовними та залежать від характеристик конкретного пристрою, наявності апаратного прискорення та кількості записів у базі. Векторний пошук засобами індексу HNSW виконується значно швидше за лінійний перебір, що особливо помітно при великій кількості зареєстрованих осіб. Загалом сумарний час оброблення одного обличчя залишається у межах, прийнятних для інтерактивної роботи із живим відеопотоком.

3.14 Висновки до третього розділу

У третьому розділі описано програмну реалізацію застосунку відповідно до спроєктованої архітектури. Реалізовано модулі детектування облич на основі MediaPipe, формування векторних ознак моделлю FaceNet у середовищі TensorFlow Lite, роботи з векторною базою ObjectBox у режимах наближеного

та точного пошуку, а також сценарій розпізнавання, що координує ці модулі. Користувацький інтерфейс побудовано засобами Jetpack Compose.

Проведено функціональне тестування за сімома сценаріями, яке підтвердило коректність роботи застосунку, зокрема правильне оброблення граничних випадків та виявлення підробок. Виконано оцінювання продуктивності окремих етапів конвеєра, що засвідчило придатність застосунку для роботи у реальному часі. Повний програмний код основних модулів винесено у додатки.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Обов'язки роботодавця щодо створення безпечних і нешкідливих умов праці

Згідно з чинним законодавством роботодавець зобов’язаний створити на робочому місці в кожному структурному підрозділі належні умови праці відповідно до нормативно-правових актів, а також забезпечити дотримання вимог законодавства щодо прав працівників з охорони праці [50].

З цією метою роботодавець забезпечує функціонування системи управління охороною праці, а саме:

– створює відповідні служби і призначає посадових осіб, які забезпечують вирішення конкретних питань охорони праці, затверджує інструкції про їх обов'язки, права та відповідальність за виконання покладених на них функцій, а також контролює їх додержання;

– розробляє за участю сторін колективного договору і реалізує комплексні заходи для досягнення встановлених нормативів та підвищення існуючого рівня охорони праці;

– забезпечує виконання необхідних профілактичних заходів відповідно до обставин, що змінюються;

– впроваджує прогресивні технології, досягнення науки і техніки, засоби механізації та автоматизації виробництва, вимоги ергономіки, позитивний досвід з охорони праці тощо;

– забезпечує належне утримання будівель і споруд, виробничого обладнання та устаткування, моніторинг за їх технічним станом;

– забезпечує усунення причин, що призводять до нещасних випадків, професійних захворювань, та здійснення профілактичних заходів, визначених комісіями за підсумками розслідування цих причин;

– організовує проведення аудиту охорони праці, лабораторних досліджень умов праці, оцінку технічного стану виробничого обладнання та устаткування, атестацій робочих місць на відповідність нормативно-правовим 59 актам з охорони праці в порядку і строки, що визначаються законодавством, та за їх підсумками вживає заходів до усунення небезпечних і шкідливих для здоров'я виробничих факторів;

– розробляє і затверджує положення, інструкції, інші акти з охорони праці, що діють у межах підприємства (далі - акти підприємства), та встановлюють правила виконання робіт і поведінки працівників на території підприємства, у виробничих приміщеннях, на будівельних майданчиках, робочих місцях відповідно до нормативно-правових актів з охорони праці, забезпечує безоплатно працівників нормативно-правовими актами та актами підприємства з охорони праці;

– здійснює контроль за додержанням працівником технологічних процесів, правил поводження з машинами, механізмами, устаткуванням та іншими засобами виробництва, використанням засобів колективного та індивідуального захисту, виконанням робіт відповідно до вимог з охорони праці, організовує пропаганду безпечних методів праці та співробітництво з працівниками у галузі охорони праці;

– вживає термінових заходів для допомоги потерпілим, залучає за необхідності професійні аварійно-рятувальні формування у разі виникнення на підприємстві аварій та нещасних випадків.

Роботодавець зобов’язаний забезпечити за свій рахунок придбання, комплектування, видачу та утримання засобів індивідуального захисту відповідно до нормативно-правових актів з охорони праці та колективного договору, а уразі передчасного зношення цих засобів не з вини працівника, замінити їх за свій рахунок [51].

Особлива увага має приділятися виявленню та усуненню причин, що можуть призвести до нещасних випадків, професійних захворювань,

здійсненню профілактичних заходів з метою недопущення аварії на виробництві. Для цього проводяться лабораторні дослідження умов праці, аналізується технічний стан виробничого обладнання та устаткування, здійснюється атестація робочих місць на відповідність їх нормативно-правовим актам з охорони праці, за підсумками 60 якої роботодавець розробляє та впроваджує заходи усунення небезпечних і шкідливих для здоров'я виробничих факторів [50].

Роботодавець через створену ним службу з охорони праці, комісію з питань охорони праці здійснює контроль за додержанням працівниками вимог виробничої санітарії, гігієни праці, техніки безпеки, використання засобів колективного та індивідуального захисту, виконання робіт згідно з розробленими і затвердженими на підприємстві положеннями, інструкціями та іншими актами з охорони праці.

У свою чергу, працівники, виконуючи свої трудові обов'язки, повинні дотримуватись трудової і технічної дисципліни, підвищувати продуктивність та якість праці.

Згідно з Законодавством України про охорону праці, зокрема статтею 18 Закону України «Про охорону праці», роботодавець зобов'язаний організувати навчання, інструктаж і перевірку знань з питань охорони праці для всіх працівників. Також це передбачено типовим положенням про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженим наказом Держгірпромнагляду № 15 від 26.01.2005 року а потім, Державній службі України з питань праці (Держпраці) згідно з розпорядженням Кабінету Міністрів України № 1021-р від 30 вересня 2015 року.

Основним документом, який регламентує взаємовідносини між трудовим колективом і роботодавцем, є колективний договір. Цей договір розробляється роботодавцем та профспілковою організацією підприємства і затверджується на зборах (конференції) трудового колективу.

Для запланованих заходів з охорони праці роботодавець зобов'язаний виділити цільові кошти та необхідні матеріальні ресурси, витрачати які на інші цілі заборонено. Порядок використання цих коштів визначається у колективному договорі і контролюється трудовим колективом.

До обов'язків роботодавця входить своєчасне проведення загальнообов'язкового державного соціального страхування працівників від нещасного випадку на виробництві та професійного захворювання, вживання термінових заходів для допомоги потерпілим, у т.ч. залучення за необхідності 61 професійних аварійно-рятувальних формувань, вести облік і розслідування нещасних випадків, професійних захворювань і аварій на виробництві.

Роботодавець також може за рахунок власних коштів здійснювати додаткові виплати потерпілим працівникам і членам їх сімей відповідно до колективного або трудового договору.

4.2 Питання щодо безпеки в надзвичайних ситуаціях

Належний рівень організації пожежної безпеки один із ключових елементів промислової безпеки підприємства. Щоб його досягнути, необхідно вжити низку заходів, зокрема забезпечити території підприємств, будинків, споруд, приміщень, технологічних установок первинними засобами пожежогасіння, які використовують на початку боротьби з пожежами, для їхньої локалізації та ліквідації.

Вимога щодо забезпечення первинними засобами пожежогасіння поширюється також на будівлі, споруди та приміщення, обладнані будь-якими типами систем пожежогасіння, пожежної сигналізації або внутрішніми пожежними кран-комплектами.

Засоби пожежогасіння переважно це речовини або їх комплекс, придатні для використання людиною для локалізації і (або) ліквідування пожежі на її початковій стадії (ДСТУ 2272:2006 «Пожежна безпека. Терміни та визначення

основних понять»). До первинних засобів пожежогасіння належать: вогнегасники; ящики з піском; бочки з водою; покривала з негорючого теплоізоляційного матеріалу; пожежні відра, совкові лопати, пожежний інструмент кирки, сокири, багри, ломи тощо.

Ці засоби використовують на початку боротьби з пожежами, щоб їх локалізувати та ліквідувати. Найефективнішим первинним засобом 62 пожежогасіння є вогнегасник. Порядок розміщенням вогнегасників на об'єкті врегульований Правилами з експлуатації та типовими нормами належності вогнегасників, затвердженими наказом Міністерства внутрішніх справ України від 15.01.2018 № 25. Документ поширюється на будинки і приміщення різного призначення, що експлуатуються, підприємства, установи та організації (незалежно від виду їх діяльності і форм власності) та механічні транспортні засоби.

Під час вибору засобів пожежогасіння потрібно врахувати фізико-хімічні та пожежонебезпечні властивості горючих речовин і матеріалів, їх взаємодію з вогнегасними речовинами, а також площу виробничих приміщень, відкритих майданчиків та установок. Відповідальними особами за своєчасне й повне оснащення об'єктів засобами пожежогасіння, їх технічне обслуговування, навчання працівників правилам користування вогнегасниками є власники цих об'єктів або орендарі згідно з договором оренди. До початку експлуатації об'єкти будинки, споруди, приміщення, технологічні установки забезпечити первинними засобами пожежогасіння згідно з Правила № 25. Необхідну кількість таких засобів окремо для кожного поверху та приміщення визначає відповідальний за пожежну безпеку на об'єкті. Переносні вогнегасники:

– навісити на вертикальні конструкції на висоті не більше ніж 1,5 м від рівня підлоги до нижнього торця вогнегасника та на відстані від дверей, достатній, щоб їх повністю відчинити;

– установити у шафи пожежних кран-комплектів, спеціальні тумби, на підставки, що надійно закріплені, на підлозі (якщо дозволяє конструкційне виконання), у пожежні щити (стенди).

Якщо в одному приміщенні розміщені декілька різних за пожежною небезпекою виробництв, не відділених одне від одного протипожежними стінами, то всі ці приміщення потрібно забезпечити вогнегасниками, пожежним інвентарем та іншими видами засобів пожежогасіння за нормами найбільш 63 небезпечного виробництва. У будинках адміністративного та побутового призначення і громадських будинках на кожному поверсі треба встановити не менше двох переносних (порошкових, водопінних або водяних) вогнегасників з масою заряду вогнегасної речовини 5 кг і більше. Як спростити облік вогнегасників Окрім того, потрібно передбачити по одному газовому вогнегаснику з величиною заряду вогнегасної речовини 3 кг і більше:

– на 20 м^2 площі підлоги в офісних приміщеннях з оргтехнікою, коморах, електрощитових, вентиляційних камерах та інших технічних приміщеннях;

– 50 м^2 площі підлоги в приміщеннях архівів, машзалів, бібліотек, музеїв.

Приміщення з оргтехнікою оснастити переносними газовими вогнегасниками з розрахунку один вогнегасник ВВК-1,4 чи ВВК-2, але не менше ніж один вогнегасник зазначених типів на приміщення. Для захисту квартир багатоквартирних житлових будинків і будинків індивідуальної забудови використовуються переносні вогнегасники з розрахунку один водяний (ВВ-5, ВВ-6), або водопінний (ВВП-6), або один порошковий (ВП-2, ВП-3) вогнегасник на одну квартиру або на один будинок індивідуальної забудови. Додатково будинки та приміщення, зазначені в пунктах 5, 6, 7 розділу VI Правил № 1417, можна оснастити ВВПА з масою заряду вогнегасної речовини 400 г і більше. Для захисту приміщень, призначених для виготовлення кулінарної продукції або приготування їжі, або все разом то

використовуються переносні вогнегасники з можливістю гасити пожежу класу F з розрахунку один вогнегасник на одне окреме робоче місце для виготовлення кулінарної продукції або приготування їжі. Щоб зазначити місце розташування первинних засобів пожежогасіння, потрібно установити вказівні знаки згідно з ДСТУ EN ISO 7010:2019 «Графічні символи. Кольори та знаки безпеки. Зареєстровані знаки безпеки». Знаки розміщуються на видимих місцях на висоті 2 – 2,5 м від рівня підлоги як усередині, так і поза приміщеннями (за потреби).

Первинні засоби пожежогасіння можна зберігати на пожежних щитах (стендах) червоного кольору, які встановлюють у виробничих, складських, 64 допоміжних приміщеннях, будинках, спорудах, а також на території підприємств. Пожежні щити встановлюються на території об’єкта площею понад 200 м^2 з розрахунку один щит на 5000 м^2 захищуваної площі. На пожежних щитах розміщуються ті первинні засоби гасіння пожежі, які можна застосовувати в певному приміщенні, споруді, установці. Склади пиломатеріалів, тари та волокнистих матеріалів забезпечуються необхідною понаднормовою кількістю пожежних щитів з набором первинних засобів пожежогасіння.

На пожежному щиті вказується порядковий номер після літерного індексу «ПЩ» та номер телефону для виклику пожежно-рятувальних підрозділів. Пожежні щити мають забезпечувати:

– захист вогнегасників від потрапляння прямих сонячних променів;

– захист знімних комплектувальних виробів від використання не за призначенням;

– зручність та оперативність зняття (витягання) закріплених комплектувальних виробів.

Пожежні щити встановлюються так, щоб вони не створювали перешкоди під час евакуації людей у разі пожежі.

Пожежний ручний інструмент на пожежному щиті має періодично обслуговуватись:

– очищати від пилу, бруду та слідів корозії;

– відновлювати фарбування відповідно до стандартів;

– випрямляти ломи та суцільнометалеві гаки після використання;

– відновлювати потрібні кути загострювання інструмента.

Пожежні покривала використовуються для гасіння невеликих осередків пожеж у разі займання речовин, горіння яких не може відбуватися без доступу повітря. Розмір пожежного покривала має бути не менше ніж 1×1 м, у місцях застосування та зберігання легкозаймистих речовин 2×1,5 м, горючих речовин 2×2 м. Ящики для піску повинні мати місткість 0,5, 1,0 або 3,0 м^3 і бути укомплектованими совковою лопатою. Місткість ящиків для піску, які є 65 елементом конструкції пожежного стенда, має бути не менше ніж 0,1 м^3. Конструкція ящика має забезпечувати зручність діставання піску та унеможливлювати потрапляння сміття й атмосферних опадів. Бочки з водою встановлюють у виробничих, складських та інших приміщеннях, спорудах у разі відсутності внутрішнього протипожежного водогону та за наявності горючих матеріалів. Їх кількість визначається з розрахунку одна бочка на 250 – 300 м^2 захищуваної площі. Місткість бочки для води має бути не менше ніж 0,2 м^3. Укомплектовується її пожежним відром місткістю не менше ніж 0,008 м^3.

4.3 Висновок до четвертого розділу

У четвертому розділі було розглянуто питання про охорону праці, що мають ключове значення в забезпеченні належних умов для збереження життя, здоров’я та працездатності працівників і населення. Зокрема, було досліджено обов’язки роботодавця щодо створення безпечних і нешкідливих умов праці, які передбачають організацію системного управління охороною праці,

забезпечення працівників інструкціями та засобами індивідуального захисту, проведення інструктажів і навчань, контроль за дотриманням правил техніки безпеки, а 68 також вжиття заходів з профілактики виробничого травматизму та професійних захворювань також вжиття заходів з профілактики виробничого травматизму та професійних захворювань.

Окрему увагу в розділі приділено організації пожежної безпеки як одному з ключових елементів промислової безпеки підприємства. Було проаналізовано вимоги до забезпечення об’єктів первинними засобами пожежогасіння (вогнегасниками, пожежними щитами, інвентарем), які є необхідними для локалізації та ліквідації загорянь на початковій стадії. Визначено порядок вибору, розміщення та технічного обслуговування цих засобів відповідно до чинних нормативних актів, а також наголошено на відповідальності власників об’єктів за їхню наявність та справність, що є гарантією мінімізації наслідків можливих пожеж.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну задачу розроблення мобільного застосунку для платформи Android, що виконує розпізнавання облич повністю на пристрої користувача. У ході виконання роботи отримано такі основні результати.

1) Проаналізовано предметну область розпізнавання облич та визначено основні сфери застосування цієї технології – облік відвідуваності, електронну ідентифікацію клієнтів, аналітику поведінки відвідувачів та відеоспостереження. Обґрунтовано перевагу обробки біометричних даних на пристрої над серверним інференсом за критеріями затримки, автономності та захисту персональних даних.

2) Досліджено методи детектування облич та формування векторних ознак. Для детектування обрано легкі засоби MediaPipe та ML Kit, а для формування ознак – згорткову нейронну мережу FaceNet, що формує вкладення розмірності 512 та виконується у середовищі TensorFlow Lite.

3) Обґрунтовано вибір інструментальних засобів: мови Kotlin, інтерфейсного інструментарію Jetpack Compose, бібліотеки CameraX для роботи з камерою та вбудованої векторної бази ObjectBox із підтримкою індексу HNSW для швидкого пошуку найближчих сусідів.

4) Спроєктовано трирівневу архітектуру застосунку з поділом на рівні представлення, предметної області та даних, розроблено конвеєр розпізнавання облич, що поєднує потоки реєстрації та ідентифікації, а також спроєктовано схему бази даних із двох колекцій.

5) Розроблено та подано у вигляді блок-схем алгоритми реєстрації особи та розпізнавання у реальному часі, обґрунтовано застосування косинусної подібності як метрики порівняння вкладень і принцип прийняття рішення про впізнавання за пороговим значенням.

6) Реалізовано програмні модулі детектування облич, формування вкладень, роботи з векторною базою та сценарій розпізнавання, а також користувацький інтерфейс. Передбачено модуль виявлення підробок на основі моделі MiniFASNet для протидії атакам із демонструванням фотографій.

7) Проведено функціональне тестування за основними сценаріями використання та оцінено продуктивність етапів конвеєра. Результати підтвердили коректність роботи застосунку та його придатність для розпізнавання у реальному часі.

Розроблений застосунок може бути використаний як основа для практичних біометричних рішень – систем обліку відвідуваності, контролю доступу та електронної ідентифікації, що потребують автономної роботи та підвищеного рівня захисту персональних даних. Модульна архітектура коду спрощує подальше розширення системи, зокрема додавання нових моделей, перенесення реалізації на інші платформи або інтеграцію з наявними інформаційними системами. Таким чином, мету кваліфікаційної роботи досягнуто, а поставлені завдання виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шумова, Л. О., Рязанцев, О. І., & Покришка, С. А. (2023). Моделі машинного навчання для формування рекомендацій. Вісник Східноукраїнського національного університету імені Володимира Даля, (2 (278)), 96-105.

2. Пасічник, В. В., Юнчик, В. Л., Кунанець, Н. Е., & Федонюк, А. А. (2022). Використання нечіткої логіки у процесі експертного оцінювання електронних навчальних ресурсів. Scientific Bulletin of UNFU, 32(4), 66-76.

3. Pankiv, Y., Kunanets, N., Artemenko, O., Veretennikova, N., & Nebesnyi, R. (2021, September). Project of an intelligent recommender system for parking vehicles in smart cities. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 2, pp. 419-422). IEEE.

4. Matsyuk, O., Nazaruk, M., Turbal, Y., Veretennikova, N., & Nebesnyi, R. (2018, September). Information analysis of procedures for choosing a future specialty. In Conference on Computer Science and Information Technologies (pp. 364-375). Cham: Springer International Publishing.

5. Небесний, Р. М. (2023). Рекомендаційна система формування команд виконавців з відповідними фаховими компетентностями (Doctoral dissertation, Тернопільський національний технічний університет імені Івана Пулюя).

6. Ржеуський, А., Кунанець, Н., & Стахів, М. (2018). Рекомендаційна система інформаційного обслуговування користувачів бібліотек. У Матеріали V науково-технічної конференції "Інформаційні моделі, системи та технології" (с. 37). Тернопіль: ТНТУ.

7. Schroff F., Kalenichenko D., Philbin J. FaceNet: A Unified Embedding for Face Recognition and Clustering. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015. P. 815–823.

8. Parkhi O. M., Vedaldi A., Zisserman A. Deep Face Recognition. Proceedings of the British Machine Vision Conference (BMVC). 2015. 12 p.

9. Taigman Y., Yang M., Ranzato M., Wolf L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. IEEE CVPR. 2014. P. 1701–1708.

10. Bazarevsky V. та ін. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. arXiv preprint arXiv:1907.05047. 2019. URL: https://arxiv.org/abs/1907.05047.

11. TensorFlow Lite. Офіційна документація. URL: https://www.tensorflow.org/lite (дата звернення: 10.05.2026).

12. MediaPipe Face Detector for Android. Google AI Edge. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector/android (дата звернення: 10.05.2026).

13. ML Kit Face Detection. Google Developers. URL: https://developers.google.com/ml-kit/vision/face-detection/android (дата звернення: 10.05.2026).

14. ObjectBox Documentation. URL: https://docs.objectbox.io/ (дата звернення: 12.05.2026).

15. ObjectBox Vector Search (On-Device Vector Database). URL: https://docs.objectbox.io/on-device-vector-search (дата звернення: 12.05.2026).

16. Malkov Yu. A., Yashunin D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Vol. 42, No. 4. P. 824–836.

17. Android Developers. Jetpack Compose. URL: https://developer.android.com/jetpack/compose (дата звернення: 14.05.2026).

18. Android Developers. CameraX Overview. URL: https://developer.android.com/media/camera/camerax (дата звернення: 14.05.2026).

19. Android Developers. Photo Picker. URL: https://developer.android.com/training/data-storage/shared/photopicker (дата звернення: 14.05.2026).

20. Kotlin Programming Language. Official Documentation. URL: https://kotlinlang.org/docs/home.html (дата звернення: 15.05.2026).

21. Koin – The pragmatic Kotlin & Kotlin Multiplatform Dependency Injection framework. URL: https://insert-koin.io/ (дата звернення: 15.05.2026).

22. Silent-Face-Anti-Spoofing (MiniFASNet). Minivision AI. GitHub. URL: https://github.com/minivision-ai/Silent-Face-Anti-Spoofing (дата звернення: 16.05.2026).

23. Liu Y., Jourabloo A., Liu X. Learning Deep Models for Face Anti-Spoofing: Binary or Auxiliary Supervision. IEEE CVPR. 2018. P. 389–398.

24. Deng J. та ін. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. IEEE CVPR. 2019. P. 4690–4699.

25. Shubham Panchal. On-Device Face Recognition In Android. GitHub repository. URL: https://github.com/shubham0204/OnDevice-Face-Recognition-Android (дата звернення: 18.05.2026).

26. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 p.

27. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Springer, 2022. 1230 p.

28. Howard A. та ін. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. 2017.

29. Голінько В. І. Охорона праці в галузі інформаційних технологій: навч. посіб. / В. І. Голінько, М. Ю. Іконніков, Я. Я. Лебедєв; М-во освіти і науки України, Держ. вищий навч. закл. "Нац. гірн. ун-т". - Дніпропетровськ: НГУ, 2015. - 246 с.

30. Гандзюк М.П. Основи охорони праці: Підручник. 4-е вид./Гандзюк М.П., Желібо Є.П., Халімовський М.О. - Київ: Каревела, 2008. – 384с.

31. Техноекологія та цивільна безпека. Частина «Цивільна безпека»: Навчальний посібник; укл.: Стручок В. С. Тернопіль: ФОП Паляниця В.А., 2022. 150 с.

32. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.

33. Умови праці працівників, які використовують у роботі персональні комп'ютери. Zolochiv.Net. URL: https://zolochiv.net/umovy-pratsi-pratsivnykiv-iaki-vykorystovuiut-u-roboti-personal-ni-komp-iutery/ (дата звернення: 25.10.2025).

ДОДАТКИ

ДОДАТОК А

Програмний код модуля детектування облич

(MediapipeFaceDetector.kt)

```
package com.ml.shubham0204.facenet_android.domain.face_detection

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Rect
import android.net.Uri
import androidx.core.graphics.toRect
import com.google.mediapipe.framework.image.BitmapImageBuilder
import com.google.mediapipe.tasks.core.BaseOptions
import com.google.mediapipe.tasks.vision.core.RunningMode
import com.google.mediapipe.tasks.vision.facedetector.FaceDetector
import com.ml.shubham0204.facenet_android.domain.AppException
import com.ml.shubham0204.facenet_android.domain.ErrorCode
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext

// Utility class for interacting with Mediapipe's Face Detector
// See https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector/android
class MediapipeFaceDetector(
    private val context: Context,
) : BaseFaceDetector() {
    // The model is stored in the assets folder
    private val modelName = "blaze_face_short_range.tflite"
    private val baseOptions = BaseOptions.builder().setModelAssetPath(modelName).build()
    private val faceDetectorOptions =
        FaceDetector.FaceDetectorOptions
            .builder()
            .setBaseOptions(baseOptions)
            .setRunningMode(RunningMode.IMAGE)
            .build()
    private val faceDetector = FaceDetector.createFromOptions(context, faceDetectorOptions)

    override suspend fun getCroppedFace(imageUri: Uri): Result<Bitmap> =
        withContext(Dispatchers.IO) {
            val imageBitmap =
                getBitmapFromUri(context,       imageUri)       ?:       return@withContext
Result.failure<Bitmap>(
                    AppException(ErrorCode.FACE_DETECTOR_FAILURE),
                )

            // We need exactly one face in the image, in other cases, return the
            // necessary errors
            val                                   faces                                   =
faceDetector.detect(BitmapImageBuilder(imageBitmap).build()).detections()
            if (faces.size > 1) {
                return@withContext
Result.failure<Bitmap>(AppException(ErrorCode.MULTIPLE_FACES))
            } else if (faces.size == 0) {
                return@withContext Result.failure<Bitmap>(AppException(ErrorCode.NO_FACE))
            } else {
                // Validate the bounding box and
                // return the cropped face
                val rect = faces[0].boundingBox().toRect()
                if (validateRect(imageBitmap, rect)) {
                    val croppedBitmap =
                        Bitmap.createBitmap(
                            imageBitmap,
                            rect.left,
                            rect.top,
                            rect.width(),
                            rect.height(),
                        )
                    return@withContext Result.success(croppedBitmap)
                } else {
                    return@withContext Result.failure<Bitmap>(
                        AppException(ErrorCode.FACE_DETECTOR_FAILURE),
                    )
                }
            }
        }
```

```
    // Detects multiple faces from the `frameBitmap`
    // and returns pairs of (croppedFace , boundingBoxRect)
    // Used by ImageVectorUseCase.kt
    override suspend fun getAllCroppedFaces(frameBitmap: Bitmap): List<Pair<Bitmap, Rect>> =
        withContext(Dispatchers.IO) {
            return@withContext faceDetector
                .detect(BitmapImageBuilder(frameBitmap).build())
                .detections()
                .filter { validateRect(frameBitmap, it.boundingBox().toRect()) }
                .map { detection -> detection.boundingBox().toRect() }
                .map { rect ->
                    val croppedBitmap =
                        Bitmap.createBitmap(
                            frameBitmap,
                            rect.left,
                            rect.top,
                            rect.width(),
                            rect.height(),
                        )
                    Pair(croppedBitmap, rect)
                }
        }
}
```

ДОДАТОК Б

Програмний код модуля формування вкладень (FaceNet.kt)

```
package com.ml.shubham0204.facenet_android.domain.embeddings

import android.content.Context
import android.graphics.Bitmap
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.koin.core.annotation.Single
import org.tensorflow.lite.DataType
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.gpu.CompatibilityList
import org.tensorflow.lite.gpu.GpuDelegate
import org.tensorflow.lite.support.common.FileUtil
import org.tensorflow.lite.support.common.TensorOperator
import org.tensorflow.lite.support.image.ImageProcessor
import org.tensorflow.lite.support.image.TensorImage
import org.tensorflow.lite.support.image.ops.ResizeOp
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer
import org.tensorflow.lite.support.tensorbuffer.TensorBufferFloat
import java.nio.ByteBuffer

// Derived from the original project:
//
https://github.com/shubham0204/FaceRecognition_With_FaceNet_Android/blob/master/app/src/main/java/c
om/ml/quaterion/facenetdetection/model/FaceNetModel.kt
// Utility class for FaceNet model
@Single
class FaceNet(
    context: Context,
    useGpu: Boolean = true,
    useXNNPack: Boolean = true,
) {
    // Input image size for FaceNet model.
    private val imgSize = 160

    // Output embedding size
    private val embeddingDim = 512

    private var interpreter: Interpreter
    private val imageTensorProcessor =
        ImageProcessor
            .Builder()
            .add(ResizeOp(imgSize, imgSize, ResizeOp.ResizeMethod.BILINEAR))
            .add(NormalizeOp())
            .build()

    init {
        // Initialize TFLiteInterpreter
        val interpreterOptions =
            Interpreter.Options().apply {
                // Add the GPU Delegate if supported.
                // See -> https://www.tensorflow.org/lite/performance/gpu#android
                if (useGpu) {
                    if (CompatibilityList().isDelegateSupportedOnThisDevice) {

addDelegate(GpuDelegate(CompatibilityList().bestOptionsForThisDevice))
                    }
                } else {
                    // Number of threads for computation
                    numThreads = 4
                }
                useXNNPACK = useXNNPack
                useNNAPI = true
            }
        interpreter =
            Interpreter(FileUtil.loadMappedFile(context,             "facenet_512.tflite"),
interpreterOptions)
    }

    // Gets an face embedding using FaceNet
    suspend fun getFaceEmbedding(image: Bitmap) =
        withContext(Dispatchers.Default) {
            return@withContext runFaceNet(convertBitmapToBuffer(image))[0]
        }
```

```
            // Run the FaceNet model
            private fun runFaceNet(inputs: Any): Array<FloatArray> {
                val faceNetModelOutputs = Array(1) { FloatArray(embeddingDim) }
                interpreter.run(inputs, faceNetModelOutputs)
                return faceNetModelOutputs
            }

            // Resize the given bitmap and convert it to a ByteBuffer
            private fun convertBitmapToBuffer(image: Bitmap): ByteBuffer =
imageTensorProcessor.process(TensorImage.fromBitmap(image)).buffer

            class NormalizeOp : TensorOperator {
                override fun apply(p0: TensorBuffer?): TensorBuffer {
                    val pixels = p0!!.floatArray.map { it / 255f }.toFloatArray()
                    val output = TensorBufferFloat.createFixedSize(p0.shape, DataType.FLOAT32)
                    output.loadArray(pixels)
                    return output
                }
            }
        }
```

ДОДАТОК В

Програмний код роботи з векторною базою даних

(ImagesVectorDB.kt, PersonDB.kt, DataModels.kt)

```
package com.ml.shubham0204.facenet_android.data

import io.objectbox.annotation.Entity
import io.objectbox.annotation.HnswIndex
import io.objectbox.annotation.Id
import io.objectbox.annotation.Index
import io.objectbox.annotation.VectorDistanceType

@Entity
data class FaceImageRecord(
    // primary-key of `FaceImageRecord`
    @Id var recordID: Long = 0,
    // personId is derived from `PersonRecord`
    @Index var personID: Long = 0,
    var personName: String = "",
    // the FaceNet-512 model provides a 512-dimensional embedding
    // the FaceNet model provides a 128-dimensional embedding
    @HnswIndex(
        dimensions = 512,
        distanceType = VectorDistanceType.COSINE,
    ) var faceEmbedding: FloatArray = floatArrayOf(),
)

@Entity
data class PersonRecord(
    // primary-key
    @Id var personID: Long = 0,
    var personName: String = "",
    // number of images selected by the user
    // under the name of the person
    var numImages: Long = 0,
    // time when the record was added
    var addTime: Long = 0,
)

data class RecognitionMetrics(
    val timeFaceDetection: Long,
    val timeVectorSearch: Long,
    val timeFaceEmbedding: Long,
    val timeFaceSpoofDetection: Long,
)



package com.ml.shubham0204.facenet_android.data

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.async
import kotlinx.coroutines.awaitAll
import kotlinx.coroutines.runBlocking
import org.koin.core.annotation.Single

@Single
class ImagesVectorDB {
    private val imagesBox = ObjectBoxStore.store.boxFor(FaceImageRecord::class.java)

    fun addFaceImageRecord(record: FaceImageRecord) {
        imagesBox.put(record)
    }

    fun getNearestEmbeddingPersonName(
        embedding: FloatArray,
        flatSearch: Boolean,
    ): FaceImageRecord? {
        // Enabling `flatSearch` disables ObjectBox's vector search (ANN based on HNSW)
        // and performs a linear-search to precisely compute the nearest neighbors
        if (flatSearch) {
            val allRecords = imagesBox.all
            val numThreads = 4
```

```
                    val batchSize = allRecords.size / numThreads
                    val batches = allRecords.chunked(batchSize)
                    val results =
                        runBlocking {
                            batches
                                .map { batch ->
                                    async(Dispatchers.Default) {
                                        var bestMatch: FaceImageRecord? = null
                                        var bestDistance = Float.NEGATIVE_INFINITY
                                        for (record in batch) {
                                            val      distance      =      cosineDistance(embedding,
record.faceEmbedding)
                                            if (distance > bestDistance) {
                                                bestDistance = distance
                                                bestMatch = record
                                            }
                                        }
                                        Pair(bestMatch, bestDistance)
                                    }
                                }.awaitAll()
                        }
                    return results.maxByOrNull { it.second }?.first
                }
                /*
                Use  maxResultCount  to  set  the  maximum  number  of  objects  to  return  by  the  ANN
condition.
                Hint: it can also be used as the "ef" HNSW parameter to increase the search quality
in combination
                with a query limit. For example, use maxResultCount of 100 with a Query limit of 10
to have 10 results
                that are of potentially better quality than just passing in 10 for maxResultCount
                (quality/performance tradeoff).
                 */
                return imagesBox
                    .query(FaceImageRecord_.faceEmbedding.nearestNeighbors(embedding, 10))
                    .build()
                    .findWithScores()
                    .map { it.get() }
                    .firstOrNull()
            }

            private fun cosineDistance(
                x1: FloatArray,
                x2: FloatArray,
            ): Float {
                var mag1 = 0.0f
                var mag2 = 0.0f
                var product = 0.0f
                for (i in x1.indices) {
                    mag1 += x1[i] * x1[i]
                    mag2 += x2[i] * x2[i]
                    product += x1[i] * x2[i]
                }
                mag1 = kotlin.math.sqrt(mag1)
                mag2 = kotlin.math.sqrt(mag2)
                return product / (mag1 * mag2)
            }

            fun removeFaceRecordsWithPersonID(personID: Long) {
                imagesBox.removeByIds(
                    imagesBox
                        .query(FaceImageRecord_.personID.equal(personID))
                        .build()
                        .findIds()
                        .toList(),
                )
            }
        }



        package com.ml.shubham0204.facenet_android.data

        import io.objectbox.kotlin.flow
        import kotlinx.coroutines.Dispatchers
        import kotlinx.coroutines.ExperimentalCoroutinesApi
        import kotlinx.coroutines.flow.Flow
        import kotlinx.coroutines.flow.flowOn
        import org.koin.core.annotation.Single

        @Single
```

```
class PersonDB {
    private val personBox = ObjectBoxStore.store.boxFor(PersonRecord::class.java)

    fun addPerson(person: PersonRecord): Long = personBox.put(person)

    fun removePerson(personID: Long) {
        personBox.removeByIds(listOf(personID))
    }

    // Returns the number of records present in the collection
    fun getCount(): Long = personBox.count()

    @OptIn(ExperimentalCoroutinesApi::class)
    fun getAll(): Flow<MutableList<PersonRecord>> =
        personBox
            .query(PersonRecord_.personID.notNull())
            .build()
            .flow()
            .flowOn(Dispatchers.IO)
}
```

ДОДАТОК Г

Програмний код сценарію розпізнавання (ImageVectorUseCase.kt) та виявлення підробок (FaceSpoofDetector.kt)

```
package com.ml.shubham0204.facenet_android.domain

import android.graphics.Bitmap
import android.graphics.Rect
import android.net.Uri
import com.ml.shubham0204.facenet_android.data.FaceImageRecord
import com.ml.shubham0204.facenet_android.data.ImagesVectorDB
import com.ml.shubham0204.facenet_android.data.RecognitionMetrics
import com.ml.shubham0204.facenet_android.domain.embeddings.FaceNet
import com.ml.shubham0204.facenet_android.domain.face_detection.BaseFaceDetector
import com.ml.shubham0204.facenet_android.domain.face_detection.FaceSpoofDetector
import com.ml.shubham0204.facenet_android.domain.face_detection.MediapipeFaceDetector
import org.koin.core.annotation.Single
import kotlin.math.pow
import kotlin.math.sqrt
import kotlin.time.DurationUnit
import kotlin.time.measureTimedValue

@Single
class ImageVectorUseCase(
    private val faceDetector: BaseFaceDetector,
    private val faceSpoofDetector: FaceSpoofDetector,
    private val imagesVectorDB: ImagesVectorDB,
    private val faceNet: FaceNet,
) {
    data class FaceRecognitionResult(
        val personName: String,
        val boundingBox: Rect,
        val spoofResult: FaceSpoofDetector.FaceSpoofResult? = null,
    )

    // Add the person's image to the database
    suspend fun addImage(
        personID: Long,
        personName: String,
        imageUri: Uri,
    ): Result<Boolean> {
        // Perform face-detection and get the cropped face as a Bitmap
        val faceDetectionResult = faceDetector.getCroppedFace(imageUri)
        if (faceDetectionResult.isSuccess) {
            // Get the embedding for the cropped face, and store it
            // in the database, along with `personId` and `personName`
            val embedding = faceNet.getFaceEmbedding(faceDetectionResult.getOrNull()!!)
            imagesVectorDB.addFaceImageRecord(
                FaceImageRecord(
                    personID = personID,
                    personName = personName,
                    faceEmbedding = embedding,
                ),
            )
            return Result.success(true)
        } else {
            return Result.failure(faceDetectionResult.exceptionOrNull()!!)
        }
    }

    // From the given frame, return the name of the person by performing
    // face recognition
    suspend fun getNearestPersonName(
        frameBitmap: Bitmap,
        flatSearch: Boolean,
    ): Pair<RecognitionMetrics?, List<FaceRecognitionResult>> {
        // Perform face-detection and get the cropped face as a Bitmap
        val (faceDetectionResult, t1) =
            measureTimedValue { faceDetector.getAllCroppedFaces(frameBitmap) }
        val faceRecognitionResults = ArrayList<FaceRecognitionResult>()
        var avgT2 = 0L
        var avgT3 = 0L
        var avgT4 = 0L

        for (result in faceDetectionResult) {
```

```
                    // Get the embedding for the cropped face (query embedding)
                    val (croppedBitmap, boundingBox) = result
                    val        (embedding,        t2)        =        measureTimedValue        {
faceNet.getFaceEmbedding(croppedBitmap) }
                    avgT2 += t2.toLong(DurationUnit.MILLISECONDS)
                    // Perform nearest-neighbor search
                    val (recognitionResult, t3) =
                        measureTimedValue { imagesVectorDB.getNearestEmbeddingPersonName(embedding,
flatSearch) }
                    avgT3 += t3.toLong(DurationUnit.MILLISECONDS)
                    if (recognitionResult == null) {
                        faceRecognitionResults.add(FaceRecognitionResult("Not          recognized",
boundingBox))
                        continue
                    }

                    val spoofResult = faceSpoofDetector.detectSpoof(frameBitmap, boundingBox)
                    avgT4 += spoofResult.timeMillis

                    // Calculate cosine similarity between the nearest-neighbor
                    // and the query embedding
                    val distance = cosineDistance(embedding, recognitionResult.faceEmbedding)
                    // If the distance > 0.4, we recognize the person
                    // else we conclude that the face does not match enough
                    if (distance > 0.3) {
                        faceRecognitionResults.add(
                            FaceRecognitionResult(recognitionResult.personName,        boundingBox,
spoofResult),
                        )
                    } else {
                        faceRecognitionResults.add(
                            FaceRecognitionResult("Not recognized", boundingBox, spoofResult),
                        )
                    }
                }
                val metrics =
                    if (faceDetectionResult.isNotEmpty()) {
                        RecognitionMetrics(
                            timeFaceDetection = t1.toLong(DurationUnit.MILLISECONDS),
                            timeFaceEmbedding = avgT2 / faceDetectionResult.size,
                            timeVectorSearch = avgT3 / faceDetectionResult.size,
                            timeFaceSpoofDetection = avgT4 / faceDetectionResult.size,
                        )
                    } else {
                        null
                    }

                return Pair(metrics, faceRecognitionResults)
            }

            private fun cosineDistance(
                x1: FloatArray,
                x2: FloatArray,
            ): Float {
                var mag1 = 0.0f
                var mag2 = 0.0f
                var product = 0.0f
                for (i in x1.indices) {
                    mag1 += x1[i].pow(2)
                    mag2 += x2[i].pow(2)
                    product += x1[i] * x2[i]
                }
                mag1 = sqrt(mag1)
                mag2 = sqrt(mag2)
                return product / (mag1 * mag2)
            }

            fun removeImages(personID: Long) {
                imagesVectorDB.removeFaceRecordsWithPersonID(personID)
            }
        }



        package com.ml.shubham0204.facenet_android.domain.face_detection

        import android.content.Context
        import android.graphics.Bitmap
        import android.graphics.Color
        import android.graphics.Rect
```

```
import androidx.core.graphics.get
import androidx.core.graphics.set
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.koin.core.annotation.Single
import org.tensorflow.lite.DataType
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.gpu.CompatibilityList
import org.tensorflow.lite.gpu.GpuDelegate
import org.tensorflow.lite.support.common.FileUtil
import org.tensorflow.lite.support.common.ops.CastOp
import org.tensorflow.lite.support.image.ImageProcessor
import org.tensorflow.lite.support.image.TensorImage
import kotlin.math.exp
import kotlin.time.DurationUnit
import kotlin.time.measureTime

/*

Utility class for interacting with FaceSpoofDetector

- It uses the MiniFASNet model from https://github.com/minivision-ai/Silent-Face-Anti-Spoofing
- The preprocessing methods are derived from https://github.com/serengil/deepface/blob/master/deepface/models/spoofing/FasNet.py
- The model weights are in the PyTorch format. To convert them to the TFLite format,
  check the notebook linked in the README of the project
- An instance of this class is injected in ImageVectorUseCase.kt

*/
@Single
class FaceSpoofDetector(
    context: Context,
    useGpu: Boolean = false,
    useXNNPack: Boolean = false,
    useNNAPI: Boolean = false,
) {
    data class FaceSpoofResult(
        val isSpoof: Boolean,
        val score: Float,
        val timeMillis: Long,
    )

    private val scale1 = 2.7f
    private val scale2 = 4.0f
    private val inputImageDim = 80
    private val outputDim = 3

    private var firstModelInterpreter: Interpreter
    private var secondModelInterpreter: Interpreter
    private val imageTensorProcessor =
        ImageProcessor
            .Builder()
            .add(CastOp(DataType.FLOAT32))
            .build()

    init {
        // Initialize TFLiteInterpreter
        val interpreterOptions =
            Interpreter.Options().apply {
                // Add the GPU Delegate if supported.
                // See -> https://www.tensorflow.org/lite/performance/gpu#android
                if (useGpu) {
                    if (CompatibilityList().isDelegateSupportedOnThisDevice) {
addDelegate(GpuDelegate(CompatibilityList().bestOptionsForThisDevice))
                    }
                } else {
                    // Number of threads for computation
                    numThreads = 4
                }
                useXNNPACK = useXNNPack
                this.useNNAPI = useNNAPI
            }
        firstModelInterpreter =
            Interpreter(
                FileUtil.loadMappedFile(context, "spoof_model_scale_2_7.tflite"),
                interpreterOptions,
            )
```

```
                secondModelInterpreter =
                    Interpreter(
                        FileUtil.loadMappedFile(context, "spoof_model_scale_4_0.tflite"),
                        interpreterOptions,
                    )
            }

            suspend fun detectSpoof(
                frameImage: Bitmap,
                faceRect: Rect,
            ): FaceSpoofResult =
                withContext(Dispatchers.Default) {
                    // Crop the images and scale the bounding boxes
                    // with the given two constants
                    // and perform RGB -> BGR conversion
                    val croppedImage1 =
                        crop(
                            origImage = frameImage,
                            bbox = faceRect,
                            bboxScale = scale1,
                            targetWidth = inputImageDim,
                            targetHeight = inputImageDim,
                        )
                    for (i in 0 until croppedImage1.width) {
                        for (j in 0 until croppedImage1.height) {
                            croppedImage1[i, j] =
                                Color.rgb(
                                    Color.blue(croppedImage1[i, j]),
                                    Color.green(croppedImage1[i, j]),
                                    Color.red(croppedImage1[i, j]),
                                )
                        }
                    }
                    val croppedImage2 =
                        crop(
                            origImage = frameImage,
                            bbox = faceRect,
                            bboxScale = scale2,
                            targetWidth = inputImageDim,
                            targetHeight = inputImageDim,
                        )
                    for (i in 0 until croppedImage2.width) {
                        for (j in 0 until croppedImage2.height) {
                            croppedImage2[i, j] =
                                Color.rgb(
                                    Color.blue(croppedImage2[i, j]),
                                    Color.green(croppedImage2[i, j]),
                                    Color.red(croppedImage2[i, j]),
                                )
                        }
                    }
                    val                                     input1                                     =
imageTensorProcessor.process(TensorImage.fromBitmap(croppedImage1)).buffer
                    val                                     input2                                     =
imageTensorProcessor.process(TensorImage.fromBitmap(croppedImage2)).buffer
                    val output1 = arrayOf(FloatArray(outputDim))
                    val output2 = arrayOf(FloatArray(outputDim))

                    val time =
                        measureTime {
                            firstModelInterpreter.run(input1, output1)
                            secondModelInterpreter.run(input2, output2)
                        }.toLong(DurationUnit.MILLISECONDS)

                    val output =
                        softMax(output1[0]).zip(softMax(output2[0])).map {
                            (it.first + it.second)
                        }
                    val label = output.indexOf(output.max())
                    val iSpoof = label != 1
                    val score = output[label] / 2f

                    return@withContext FaceSpoofResult(isSpoof = iSpoof, score = score, timeMillis =
time)
                }

            private fun softMax(x: FloatArray): FloatArray {
                val exp = x.map { exp(it) }
                val expSum = exp.sum()
                return exp.map { it / expSum }.toFloatArray()
            }
```

```
            private fun crop(
                origImage: Bitmap,
                bbox: Rect,
                bboxScale: Float,
                targetWidth: Int,
                targetHeight: Int,
            ): Bitmap {
                val srcWidth = origImage.width
                val srcHeight = origImage.height
                val scaledBox = getScaledBox(srcWidth, srcHeight, bbox, bboxScale)
                val croppedBitmap =
                    Bitmap.createBitmap(
                        origImage,
                        scaledBox.left,
                        scaledBox.top,
                        scaledBox.width(),
                        scaledBox.height(),
                    )
                return Bitmap.createScaledBitmap(croppedBitmap, targetWidth, targetHeight, true)
            }

            private fun getScaledBox(
                srcWidth: Int,
                srcHeight: Int,
                box: Rect,
                bboxScale: Float,
            ): Rect {
                val x = box.left
                val y = box.top
                val w = box.width()
                val h = box.height()
                val scale = floatArrayOf((srcHeight - 1f) / h, (srcWidth - 1f) / w, bboxScale).min()
                val newWidth = w * scale
                val newHeight = h * scale
                val centerX = w / 2 + x
                val centerY = h / 2 + y
                var topLeftX = centerX - newWidth / 2
                var topLeftY = centerY - newHeight / 2
                var bottomRightX = centerX + newWidth / 2
                var bottomRightY = centerY + newHeight / 2
                if (topLeftX < 0) {
                    bottomRightX -= topLeftX
                    topLeftX = 0f
                }
                if (topLeftY < 0) {
                    bottomRightY -= topLeftY
                    topLeftY = 0f
                }
                if (bottomRightX > srcWidth - 1) {
                    topLeftX -= (bottomRightX - (srcWidth - 1))
                    bottomRightX = (srcWidth - 1).toFloat()
                }
                if (bottomRightY > srcHeight - 1) {
                    topLeftY -= (bottomRightY - (srcHeight - 1))
                    bottomRightY = (srcHeight - 1).toFloat()
                }
                return      Rect(topLeftX.toInt(),      topLeftY.toInt(),      bottomRightX.toInt(),
bottomRightY.toInt())
            }
        }
```