НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень

Виконав студент IV курсу, групи ІП-12
(шифр групи)
Скорик Родіон Олегович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к. фіз.-мат. н., доц., Поперешняк С. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст.викл, д-р філософії, Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент д.т.н., проф., Жебка В. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Скорику Родіону Олеговичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень
керівник проєкту Поперешняк Світлана Володимирівна, к. фіз.-мат. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Попереднє обстеження предметної області: постановка завдання дипломного проєктування, аналіз предметної області, аналіз існуючих технічних рішень, аналіз та моделювання бізнес-процесів.
2) Розроблення вимог до програмного забезпечення: аналіз варіантів використання ПЗ, розробка функціональних та нефункціональних вимог, аналіз системних вимог та економічних показників ПЗ, постановка завдання.
3) Конструювання та розроблення ПЗ: побудова архітектури ПЗ, обґрунтування вибору засобів розробки, конструювання ПЗ та аналіз безпеки даних.
4) Аналіз якості та тестування ПЗ: статичний аналіз якості ПЗ, опис процесів тестування та наведення контрольного прикладу.
5) Розгортання та супровід ПЗ: опис процесів розгортання та супроводу застосунку.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	20.04.2025	
3	Постановка та формалізація задачі	23.04.2025	
4	Розробка інформаційного забезпечення	24.04.2025	
5	Алгоритмізація задачі	29.04.2025	
6	Обґрунтування вибору використаних технічних засобів	29.04.2025	
7	Розробка програмного забезпечення	12.05.2025	
8	Налагодження програми	14.05.2025	
9	Виконання графічних документів	28.05.2025	
10	Оформлення пояснювальної записки	01.06.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Родіон СКОРИК
(підпис) (ініціали, прізвище)

Керівник _____ Світлана ПОПЕРЕШНЯК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 23 таблиці, 29 рисунків та 34 джерела – загалом 71 сторінка.

Дипломний проєкт присвячений розробці програмного забезпечення для виявлення та ідентифікації облич у відеопотоці.

Метою проекту є покращення процедури ідентифікації та спрощення виявлення осіб на відео шляхом впровадження методів розпізнавання облич із розпізнавання живого.

У розділі «Передпроектне обстеження предметної області» описано формулювання задачі дипломної роботи, здійснено аналіз предметної області, проаналізовано існуюче аналогічне програмне й технічне забезпечення, а також побудовано відповідну модель бізнес-процесів.

Розділ 2 охоплює вивчення сценаріїв використання програмного продукту, функціональних і нефункціональних вимог, розгляд системних характеристик, економічних факторів та постановку задачі для розробки програмного рішення.

У розділі «Конструювання та розроблення програмного забезпечення» представлено архітектурну модель, обґрунтовано вибір інструментів розробки, подано опис ключових модулів програмного забезпечення й використаних алгоритмів та моделей, розглянуто питання інформаційної безпеки.

Розділ 4 зосереджений на методах оцінювання якості створеного програмного продукту, процесі тестування та аналізі контрольного прикладу.

У розділі «Розгортання та супровід програмного забезпечення» висвітлено процедуру встановлення програмного продукту на цільове середовище та організацію його подальшого технічного супроводу.

Результати роботи пройшли апробацію:

– Поперешняк С.В., Скорик Р.О., Купцов Д.В., Кравченко Р.В. Система розпізнавання обличчя людини у відеопотоці / Проблеми програмування. № 2-3. 2024. С. 296-304. http://doi.org/10.15407/pp2024.02-03.296

– Popereshnyak S., Skoryk R., Kuptsov D., Kravchenko R. (2024). Software Tool for Recognition of Human Face in Video Stream. In: Proceedings of the 14th International Scientific and Practical Programming Conference (UkrPROG 2024)

Kyiv, Ukraine, May 14-15, 2024. CEUR Workshop Proceedings, vol. 3806, : 325-336. Germany. ISSN 1613-0073. Available at: https://ceur-ws.org/Vol-3806/S_57_Popereshnyak_Skoryk_Kuptsov_Kravchenko.pdf (Scopus, dblp)

– Скорик Р.О. Поперешняк С.В. Програмне забезпечення розпізнавання обличчя у відеопотоці // Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і Світу». Збірник тез. – К.: ДУІКТ, 2023. – с. 175-177

– Скорик Р.О., Поперешняк С.В Програмне забезпечення розпізнавання обличчя у відеопотоці // Науково-практична конференція «Проблеми комп'ютерної інженерії». Збірник тез. – К.: ДУІКТ, 2023. - с. 52-54 - І місце, 2024 рік, ІІ тур Всеукраїнського конкурсу наукових досліджень студентів «Інформаційні технології в науці та виробництві», Херсонський національний технічний університет, м. Хмельницький.

– Результати роботи мають впровадження на виробничо-торгівельній приватній фірмі «АН», що підтверджено актом впровадження.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ ОБЛИЧ, ІДЕНТИФІКАЦІЯ ОБЛИЧ, ЗАХИСТ ВІД СПУФІНГУ, ВИЯВЛЕННЯ ЖИВОГО, КОМП'ЮТЕРНИЙ ЗІР, МАШИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, PYTORCH, DLIB, АНСАМБЛЬ МОДЕЛЕЙ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 23 tables, 29 figures and 34 sources – in total 70 pages.

The diploma project is dedicated to the development of software for face detection and identification in a video stream.

The goal of the project is to improve the identification procedure and simplify the detection of individuals in video footage by implementing face recognition methods with liveness detection.

The first section presents the formulation of the diploma task, an analysis of the subject domain, a review of existing analogous software and hardware solutions, and the development of a corresponding business process model.

Section 2 covers the study of software usage scenarios, the definition of functional and non-functional requirements, an overview of system characteristics and economic factors, as well as the formulation of the task for developing the software solution.

The section "Design and development of software" presents the architectural model, justifies the choice of development tools, describes the key software modules, algorithms, and models used, and addresses information security issues.

Section 4 focuses on methods for evaluating the quality of the developed software product, the testing process, and the analysis of a control example.

The section "Deployment and software maintenance" highlights the procedure for installing the software product in the target environment and organizing its further technical support.

The results of the work have undergone practical validation:

– Popereshnyak S.V., Skoryk R.O., Kuptsov D.V., Kravchenko R.V. Human Face Recognition System in a Video Stream // Problemy Programuvannia. No. 2–3. 2024. pp. 296–304. http://doi.org/10.15407/pp2024.02-03.296

– Popereshnyak S., Skoryk R., Kuptsov D., Kravchenko R. (2024). Software Tool for Recognition of Human Face in Video Stream. In: Proceedings of the 14th International Scientific and Practical Programming Conference (UkrPROG 2024), Kyiv, Ukraine, May 14–15, 2024. CEUR Workshop Proceedings, vol. 3806, pp. 325–

336. Germany. ISSN 1613-0073. Available at: https://ceur-ws.org/Vol-3806/S_57_Popereshnyak_Skoryk_Kuptsov_Kravchenko.pdf (Scopus, dblp)

– Skoryk R.O., Popereshnyak S.V. Face Recognition Software in a Video Stream // All-Ukrainian Scientific and Technical Conference “Technological Horizons: Research and Application of Information Technologies for the Technological Progress of Ukraine and the World”. Book of Abstracts. – Kyiv: DUIKT, 2023. – pp. 175–177

– Skoryk R.O., Popereshnyak S.V. Face Recognition Software in a Video Stream // Scientific and Practical Conference “Problems of Computer Engineering”. Book of Abstracts. – Kyiv: DUIKT, 2023. – pp. 52–54 – First Place, 2024, Second Round of the All-Ukrainian Student Research Competition “Information Technologies in Science and Industry”, Kherson National Technical University, Khmelnytskyi.

The results of the work have been implemented at the private production and trading company “AH”, as confirmed by the implementation act.

KEYWORDS: FACE RECOGNITION, FACE IDENTIFICATION, SPOOFING PROTECTION, LIVENESS DETECTION, COMPUTER VISION, MACHINE LEARNING, CONVOLUTIONAL NEURAL NETWORKS, PYTORCH, DLIB, MODEL ENSEMBLE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1229.045490.01.91	Технічне завдання	16	
3	А4	КПІ.ІП-1229.045490.02.81	Пояснювальна записка	71	
4	А4	КПІ.ІП-1229.045490.03.12	Текст програми	33	
5	А4	КПІ.ІП-1229.045490.04.51	Програма та методика тестування	5	
6	А4	КПІ.ІП-1229.045490.05.34	Керівництво користувача	8	
7	А3	КПІ.ІП-1229.045490.06.99	Графічний матеріал	4	

КПІ.ІП-1224.045490.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Скорик Р.О.		
Перевір.		Поперешняк С.В.		
Н.контр.		Головченко М.М.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-12

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧ У ВІДЕОПОТОЦІ З ДЕТЕКЦІЄЮ ШТУЧНО ЗМІНЕНИХ ЗОБРАЖЕНЬ

Технічне завдання

КПІ.ІП-1224.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Світлана ПОПЕРЕШНЯК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Родіон СКОРИК

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень

Галузь застосування:

Наведене технічне завдання поширюється на розробку застосунку «FaceFinder» [045490], котрий використовується для розпізнавання облич у відеопотоці із можливістю виявлення несправжніх облич та призначений для використання в системах безпеки, онлайн-ідентифікації, фінансових установах та інших сферах, де важлива достовірна верифікація особи.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «FaceFinder» є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень.

Метою розробки є покращення процедури ідентифікації та спрощення виявлення осіб на відео шляхом впровадження методів розпізнавання облич із розпізнавання живиго із можливістю роботи в реальному часі.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– інтерфейс головної сторінки має складатись із панелі навігації та трьох основних кнопок переходу: «Capture new face», «Start recognition» та «Manage labels», відповідно до рисунку 4.1.;

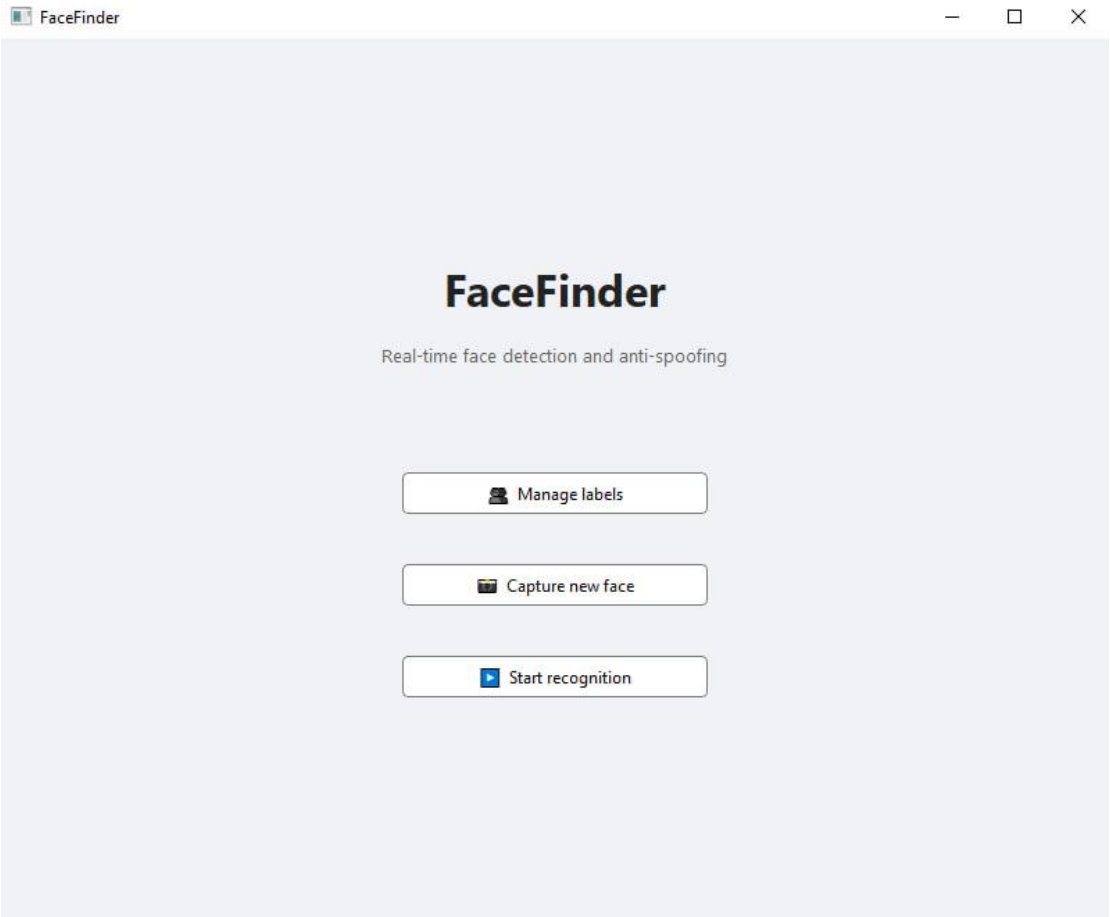


Рисунок 4.1 – Екранна форма головної сторінки застосунку

– інтерфейс менеджера міток має містити список з переліком збережених міток облич, а також кнопки редагування та видалення для кожної з них, як показано на рисунку 4.2;

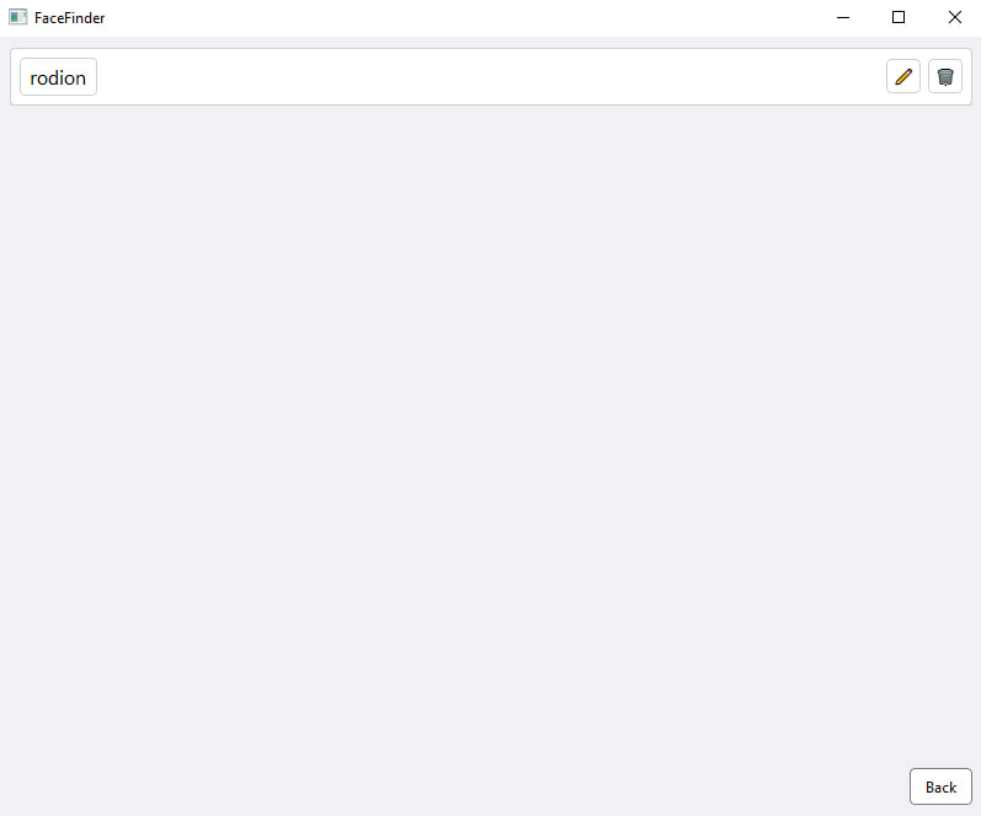


Рисунок 4.2 – Екранна форма менеджера міток

– інтерфейс сторінки захоплення має відкривати діалогове вікно для введення імені мітки та вибору джерела відео, як на рисунку 4.3. Після підтвердження повинен відкриватися екран із відеопотоком з вебкамери, кнопкою Capture (для покадрового захоплення) та Done (для завершення). Також має демонструватись прогрес захоплення, як зображено на рисунку 4.4;

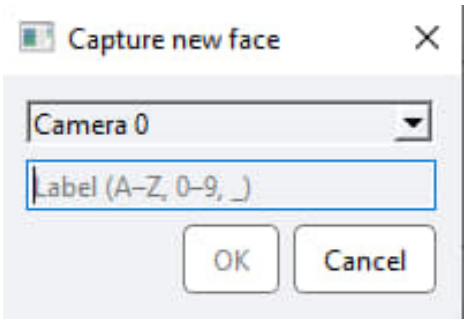


Рисунок 4.3 – Екранна форма діалогу захоплення

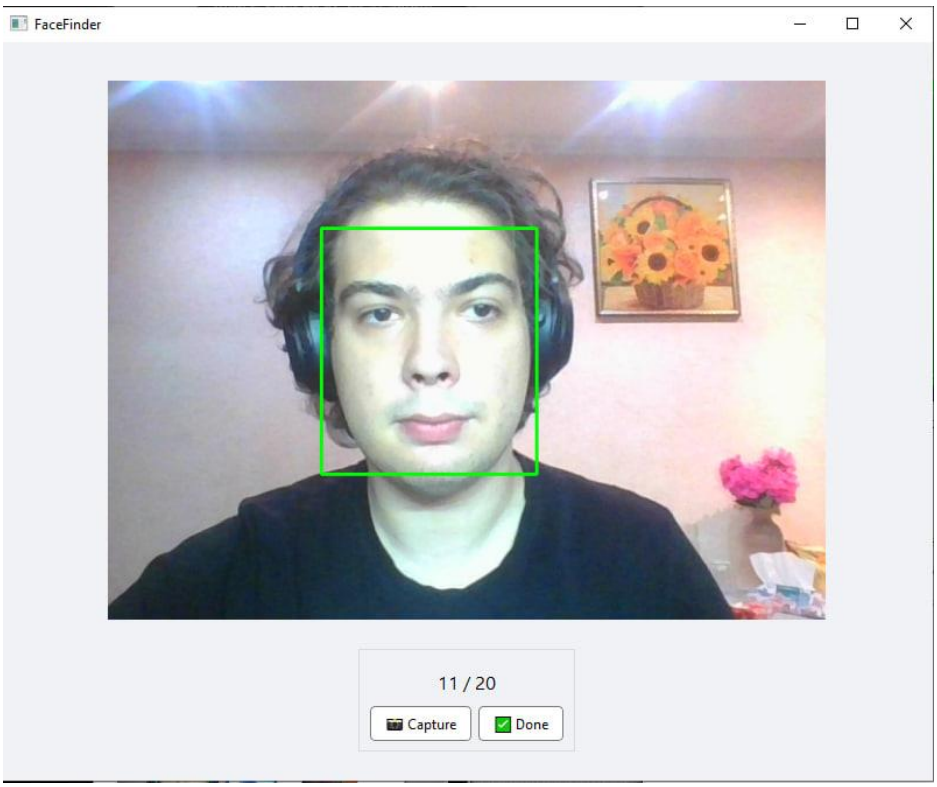


Рисунок 4.4 – Екранна форма сторінки захоплення

– інтерфейс сторінки розпізнавання має містити вибір джерела відео (камера або файл, рисуноки 4.5-7), вікно з відеопотоком у реальному часі та динамічні рамки навколо облич. Колір рамки залежить від FAS-передбачення (зелений – live, жовтий – сумнів, червоний – spoof). Над рамкою виводиться мітка особи або напис SPOOF, як показано на рисунку 4.8;

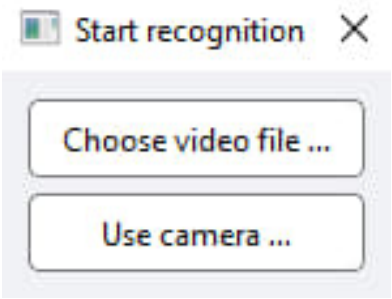


Рисунок 4.5 – Екранна форма вибору джерела відеопотоку

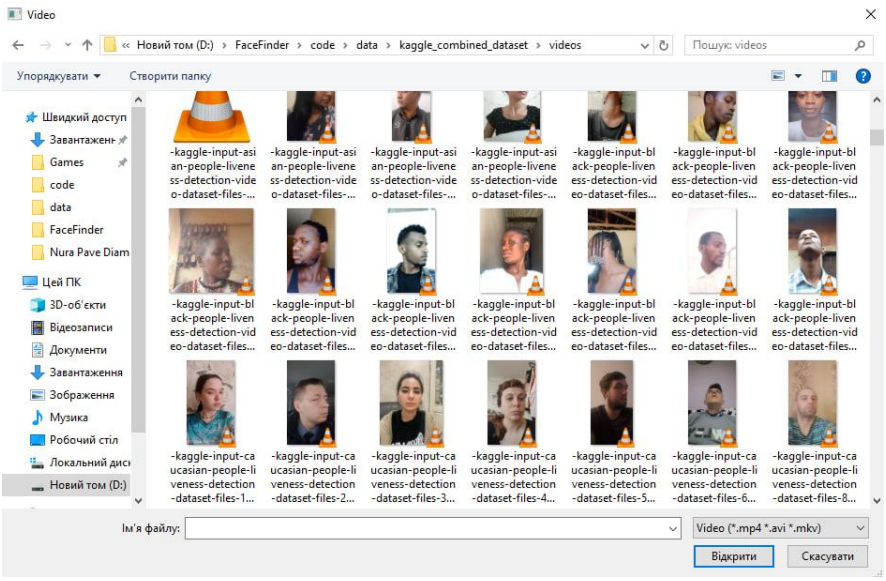


Рисунок 4.6 – Екранна форма вибору відеофайлу

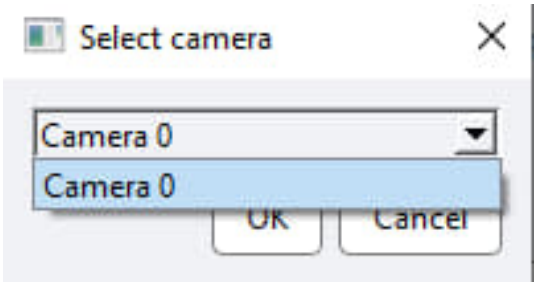


Рисунок 4.7 – Екранна форма вибору камери

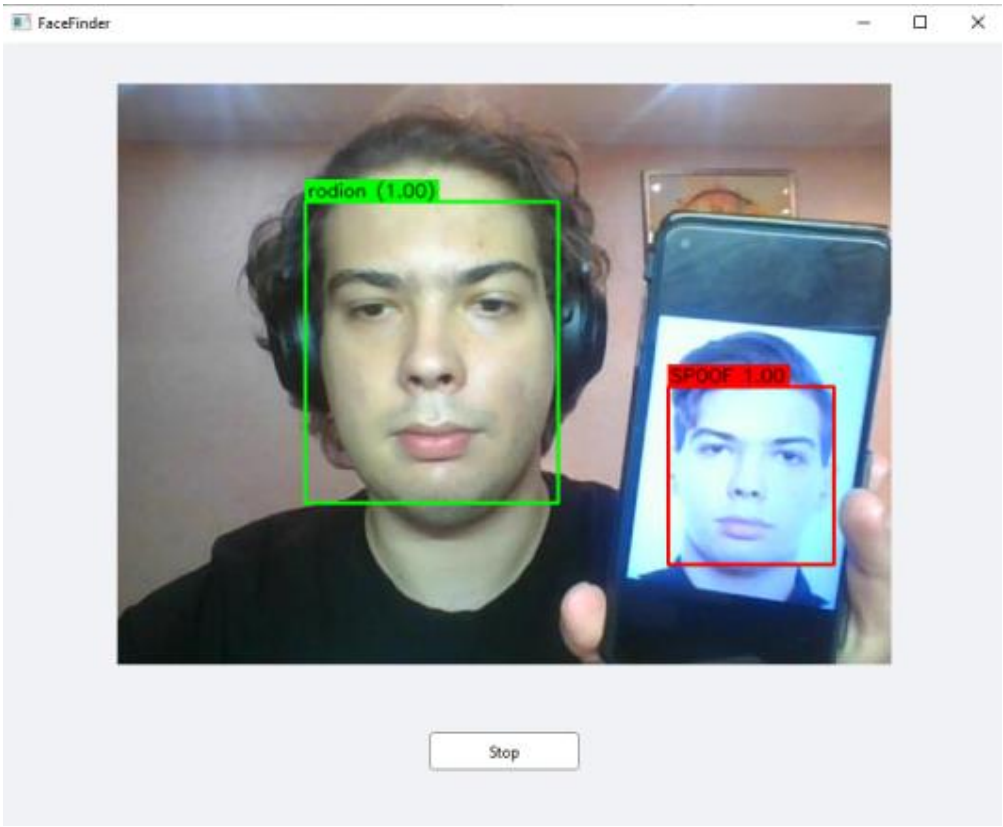


Рисунок 4.8 – Екранна форма сторінки розпізнавання

4.1.2 Отримання відеопотоку

– вибір джерела відео: вебкамера або відеофайл;

4.1.3 Робота з мітками

– перегляд списку усіх збережених міток облич;
– редагування мітки (перейменування) раніше збереженого обличчя;
– видалення кодування обличчя разом із міткою;
– перевірка формату та унікальності мітки під час додавання;

4.1.4 Виявлення облич

– виявлення облич у кожному кадрі відеопотоку;
– видалення передбачення з проєкту;
– перейменування передбачення.

4.1.5 Захоплення облич

– додавання одного нового обличчя з автоматичним виявленням;
– можливість введення мітки при захопленні обличч;
– підтримка багаторазового захоплення кадрів для покращення розпізнавання.
– вибір камери для захоплення

4.1.6 Розпізнавання облич

– розпізнавання облич на основі попередньо захоплених міток;

4.1.7 Виявлення штучно змінених облич

– автоматичне визначення, чи є обличчя справжнім (live), чи підробленим (spoof)

– виведення відповідної мітки (SPOOF) над рамкою обличчя у разі атаки;

– зміна кольору рамки залежно від рівня довіри (зелений, жовтий, червоний).

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– процесор: 4-ядерний 64-бітовий;

– об‘єм ОЗП: 8 Гб;

– об’єм дискового простору: 5 Гб;

– відеокарта: інтегрована або дискретна відеокарта з відеопам’яттю від 4 ГБ.

– операційна система: Windows 10 x64;

Рекомендована конфігурація технічних засобів для рбооти в реальному часі:

– процесор: Intel Core i7-9750H або аналогічний (6 ядер, 12 потоків);
– об‘єм ОЗП: 16 Гб;
– об’єм дискового простору: 10 Гб;
– операційна система: Windows 10 x64;
– Відеокарта: дискретна відеокарта з обсягом відеопам’яті не менше 6 ГБ (наприклад, NVIDIA GTX 1660 Ti або краща).
– Наявність зовнішньої чи вбудованої камери

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows 10 та Windows 11.

4.5.1 Вимоги до вхідних даних

Вхідні відео джерела мають бути відеопотоком з вебкамери або відеофайлом у форматі .mp4, .avi, .mov із частотою кадрів не менше 10 fps та роздільною здатністю від 640×480 пікселів;

4.5.2 Вимоги до вихідних даних

Результати мають відображатись на відео у вигляді прямокутної рамки навколо кожного виявленого обличчя з підписом мітки (якщо розпізнано), ймовірністю та статусом підробленості:

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на Github.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– діаграма бізнес-процесів;
– схема структурна варіантів використання;
– схема структурна компонентів програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	15.03	
2.	Аналіз існуючих методів розв’язання задачі	20.04	
3.	Постановка та формалізація задачі	23.04	Технічне завдання
4.	Розробка інформаційного забезпечення	24.04	Технічна документація
5.	Алгоритмізація задачі	29.04	Технічна документація
6.	Обґрунтування вибору використаних технічних засобів	29.04	Технічна документація
7.	Розробка програмного забезпечення	12.05	Тексти програмного забезпечення
8.	Налагодження програми	14.05	Тести, результати тестування
9.	Виконання графічних документів	28.05	Графічний матеріал проєкту
10.	Оформлення пояснювальної записки	01.06	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень**

КПІ.ІП-1224.045490.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
ER	–	Entity-Relation diagram.
CNN	–	Convolutional Neural Network – згорткова нейронна мережа
TL	–	Transfer Learning
FAS	–	Face Anti-Spoofing захист від або виявлення штучних облич
FR	–	Face Recognition – розпізнавання обличчя
FD	–	Face Detection – виявлення обличчя
PAD	–	Presentation Attack Detection – виявлення фізичних атак (physical attacks), таких як фото, відео або маски
DAD	–	Deepfake Attack Detection – виявлення цифрових атак (digital attacks), створених за допомогою deepfake
rPPG	–	Remote Photoplethysmography
TTA	–	Test-Time Augmentation – аугментація під час інференсу
OAP	–	Online Adaptation Pipeline – онлайн-налаштування (fine-tuning) моделі під час виконання
DA	–	Domain Adaptation – адаптація між доменами
DL	–	Deep Learning – глибоке навчання

ВСТУП

У сучасному цифровому світі розпізнавання облич стало однією з ключових технологій біометричної автентифікації, що широко використовується в мобільному банкінгу, системах контролю доступу, аеропортах, онлайн-сервісах, роботизованих системах і навіть у процесах найму працівників[1]. Його популярність зумовлена зручністю, безконтактністю та високим рівнем автоматизації. Однак популярність призводить до того, що сьогодні стикається з серйозними викликами, пов’язаними з її уразливістю до атак атак із штучною зміною обличчя, зокрема за допомогою фотографій, відео, 3D-масок тощо.

З розвитком генеративного штучного інтелекту, зокрема технологій створення дипфеків, зросла кількість випадків, коли зловмисники підроблюють обличчя для обходу систем розпізнавання. Один із показових прикладів – випадок, коли кандидат на посаду проходив співбесіди з підробленим аватаром, створеним за допомогою AI, що набуло широкого розголосу в [2]. Це свідчить не лише про ризики в кадровій сфері, але й про глибшу загрозу довіри до ідентифікаційних систем загалом.

За даними огляду наукових досліджень у сфері захисту від таких атак, ця проблема є надзвичайно актуальною, і з кожним роком зростає кількість публікацій, присвячених виявленню підробок та змін обличчя[3]. Увага наукової спільноти зосереджена на підвищенні надійності систем розпізнавання, здатних розрізнити справжнє обличчя від підробленого за допомогою аналізу текстур, рухів, температури шкіри та інших фізіологічних чи поведінкових ознак.

У рамках даного дипломного проєкту буде розроблено програмне забезпечення, що виконує розпізнавання обличчя з підтримкою захисту від фальсифікації. Метою є створення автономної, безпечної та надійної системи, здатної виявляти як реальні, так і підроблені обличчя у відеопотоці без потреби у зовнішньому з’єднанні.

Сфера застосування такої системи охоплює широкий спектр: від доступу до особистих пристроїв і платіжних систем до перевірки особистості під час онлайн-співбесід, контролю в публічних просторах, захисту персональних даних та протидії фейковим медіа. Реалізація захисту від можливих атак є критично важливою умовою для збереження довіри до біометричних систем та їх ефективного використання в умовах стрімкого розвитку загроз.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

У межах дипломного проєктування передбачається реалізація програмного забезпечення для розпізнавання облич з вбудованим модулем FAS. Завданням проєкту є створення десктопної системи, здатної працювати з відеопотоком у реальному часі, виявляти обличчя, розпізнавати користувача та визначати автентичність зображення без передачі даних на зовнішні сервери. Для досягнення цієї мети необхідно виконати наступні етапи:

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень (як програмних продуктів, так і алгоритмічних чи технічних рішень) обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– аналіз економічних показників програмного забезпечення ДП;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз безпеки даних програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Основний етап у процесі виялення облич у потоці це власне виявлення облич у кадрі, за чим опціонально слідує їх геометричне вирівнювання. Наступний етап це розпізнавання за допомогою порівняння із попередньо збереженими рисами особи. Паралельно або послідовно із цим відбувається перевірка на навність ознак штучних змін у виявленому обличчі[4].

Предметна галузь для цих задач охоплює багатодисциплінарне поле, яке поєднує методи комп'ютерного зору, обробки зображень, машинного навчання та розпізнавання образів. Вона охоплює ідентифікацію штучно модифікованих або підроблених облич, що особливо актуально в умовах стрімкого розвитку генеративних технологій. Основними завданнями є захоплення візуальної інформації з відео та зображень, виявлення ознак справжності обличчя, обробка зображень для покращення якості, а також побудова моделей, здатних відрізнити реальні риси від згенерованих або підроблених. При цьому системи повинні бути стійкими до широкого спектра атак, включаючи як фізичні підробки, так і новітні цифрові загрози [5].

Актуальні атаки можна розідлити на такі основні типи: PAD, FMA (Face Morphing Attack) і DAD. Фізичні атаки здійснюються за допомогою друкованих фотографій, попередньо записаних відео (replay attacks) або реалістичних 3D-масок, створених, зокрема, на 3D-принтерах. Також спостерігається використання атак із застосуванням складних текстур або матеріалів з рельєфом, що ускладнюють аналіз [7]. Морфінг-атаки поєднують риси кількох осіб у нове обличчя, яке проходить ідентифікацію як справжнє [8]. Цифрові атаки включають генерацію дипфейків за допомогою змагальних генеративних мереж (GAN), що дозволяє створювати реалістичні відео або зображення неіснуючих чи справжніх осіб [9]. Ці типи атак можуть доповнюватись одне одним, а також підробкою голосу для того, щоб успішно вводити в оману людей чи системи.

Для протидії таким загрозам застосовуються різноманітні підходи, які спираються на аналіз текстури, кольору та спектральних характеристик [6],

виявлення ознак живої людини, таких як кліпаняя, мікрорухи чи зміна міміки [6, 9], а також використання оптичного потоку для виявлення неприродної динаміки [10]. Окремо можна виділити 3D-реконструкцію обличчя та аналіз глибини, що дозволяє виявляти просторові невідповідності [7]. У деяких системах додатково використовуються сенсори, що охоплюють глибину чи інфрачервоний спектр, а останнім часом – також голосові та акустичні сигнали для перевірки особи.

Попри всі академічні досягнення в цій галузі, системи FAS стикаються з низкою практичних обмежень. Перш за все, гостро стоїть проблема доступності якісних і різноманітних датасетів, оскільки більшість наявних наборів даних обмежені малим переліком типів атак, різноманітністю, сценаріями або пристроями зйомки, що ще більше посилюється не бажанням широкого або комерційного розповсюдження таких даних, через наявність в них персональної інформаці. У свою чергу, це значно погіршує здатність доступних моделей до генералізації – вони добре працюють на тренувальних даних, але швидко втрачають точність на відео чи зображеннях з нових пристроїв, в інших умовах освітлення чи з небаченими раніше типами атак. Також це проявляється в тому, що варіція в використаних датасетах та метриках в наукових роботах ускладнює об’єктивну оцінку якості запропонованих рішень [6]. По-друге, дедалі більшу загрозу становлять DAD атаки, які переажно розглядались окремо від PAD, однак у контексті FAS вона поступово починає відігравати критичну роль. Врешті-решт, багато робіт нехтують оптимізацію та розрахунковою складністю, що призводить до неможливості застосування методів в реальному часі чи великих масшиабах.

Враховуючи ці виклики, останні праці в цій галузі роблять акцент на зменшені прогалини між виявленням PAD і DAD [11], а також DG, DA та TTA [12]. Наприклад, у [13] запропоновано поєднати загальну оцінку якості зображення з DL для кращої генералізації; у [14] пропонують використовувати rPPG для покращення стікості системи за різних умов; [15] зосередились на покращенні підходу до навчання CNN моделі,

використовуючи піксельні маски, щоб вивичити кращи набір ознак; у [11] розроблені підходи до синтетичного створення змінених облич шляхом аугментації спражніх, що дає змогу створити датасет, який забезпечує генералізацію на небачені моделлю загрози; [16] фокусується на покращені результатів за рахунок врахування персогналізованих дпних про обличчя, а [17] досліджує адаптацію моделі під час роботи системи (OAP).

У межах цього проєкту, разом із застосуванням стандартних підходів для виявлення то розпізнавання облич, для FAS було обрано підхід ансамблювання легких моделей, із подальшою оптимізацією архітектури для забезпечення можливості роботи в реальному часі на широкодоступних пристроях. Це включає підтримку як обробки окремих зображень, так і потокового відео, з урахуванням обмежень обчислювальних ресурсів та вимог до затримки. Метою є створення практичного рішення, яке може бути вбудоване у більші системи ідентифікації чи виявлення неправдивої інформації, або використовуватись окремо.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації FaceFinder. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Практичні підходи FAS можна подліити на активні та пасивні, залежно від потребм залученості користувача. Хоча пасивні системи захисту вимагають більше зусиль для розробки, вони дають можливість виконувати перевірку значно швидше, їх важче обманути попередньо записаними відео з потрібними рухами,

Для порівняння використаємо кілька програм-аналогів, що часто використовуються чи є лідерами ринку.

FaceTec розроблено компанією FaceTec Inc . Система забезпечує 3D-біометричну автентифікацію з FAS, підтримує локальне та хмарне розгортання. До їх переваг однозначно належить потужний маркетинг[18], однак на практиці якість продукту значно поступається обіцянкам. Вони заявляють про FRR (False Rejection Rate) менше 1%, однак це не відповідає дійсності, зокрема через потребу слідувати вказівкам і високу кількість помилок при розпізнаванні дій. Попри обіцянку затримки розпізнавання не більше 2 секунд, присутні числені скарги, що свідчать про можливі затримки аж до 15 секнд, що вже важко назвати роботою в реальному часі. Загалом, вони не наводять об'єктивних метрик, і попри заявлену участь у програмі нагороди за обхід системи[19], їх демо додаток блокує можливість запису екрану, що унеможливлює надання коректного відгуку.

Azure Face API розроблено Microsoft. Це хмарне API для виявлення, розпізнавання та аналізу облич, що може виявляти кілька осіб на кадрі. До переваг відносяться масштабованість, проста інтеграція та багатий функціонал. Недоліками є прив'язка до хмари, питання конфіденційності та можлива затримка в обробці, також відсутність заявленого захисту від DAD.

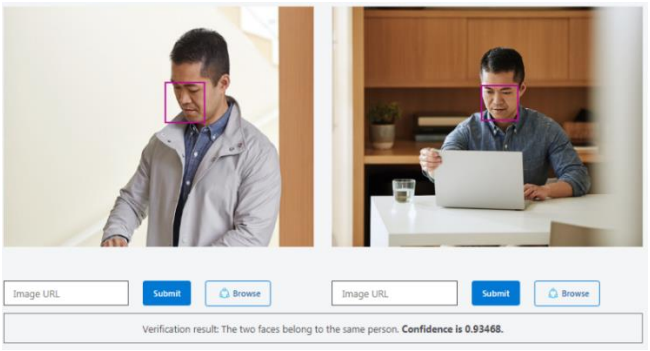


Рисунок 1.1 – Приклад розпізнавання обличчя за допомогою Azure Face API

BioID Facial Recognition розроблене BioID AG. Основні функціональні можливості це розпізнавання обличчя для біометричної аутентифікації. Використовується в різних сферах, включаючи фінансові установи та бізнес. Переваги BioID Facial Recognition: використання виключно біометричної інформації для аутентифікації, забезпечення безпеки для важливих операцій та транзакцій. Недоліками можна назвати вартість та складність

впровадження, поряд із недосконалим FAS: його доволі легко обійти, попри наявну сертифікацію високого рівня захисту

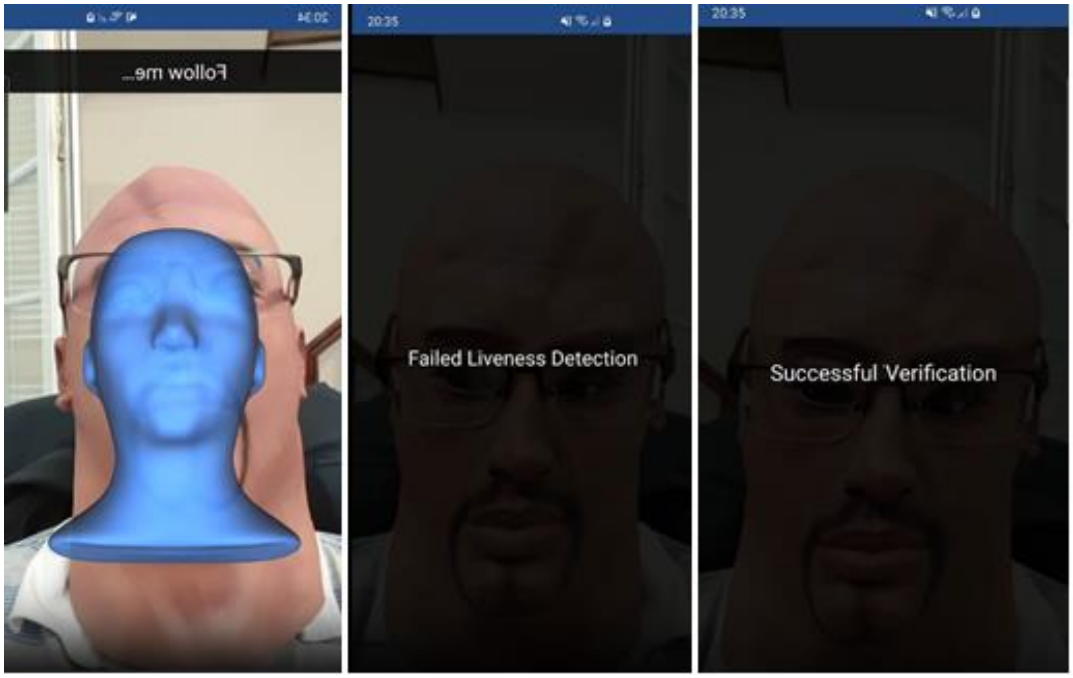


Рисунок 1.2 – Додаток BioID спочатку відкидає аутентифікацію за допомогою маски, однак при повторній спробі вона успішна

Intel RealSense ID поєднує спеціалізоване обладнання з алгоритмами розпізнавання для надійного захисту, що дає перевагу у точності, швидкості та конфіденційності. Головні недоліки – потреба у спеціальних сенсорах та обмежена гнучкість, відсутність демо додатку.

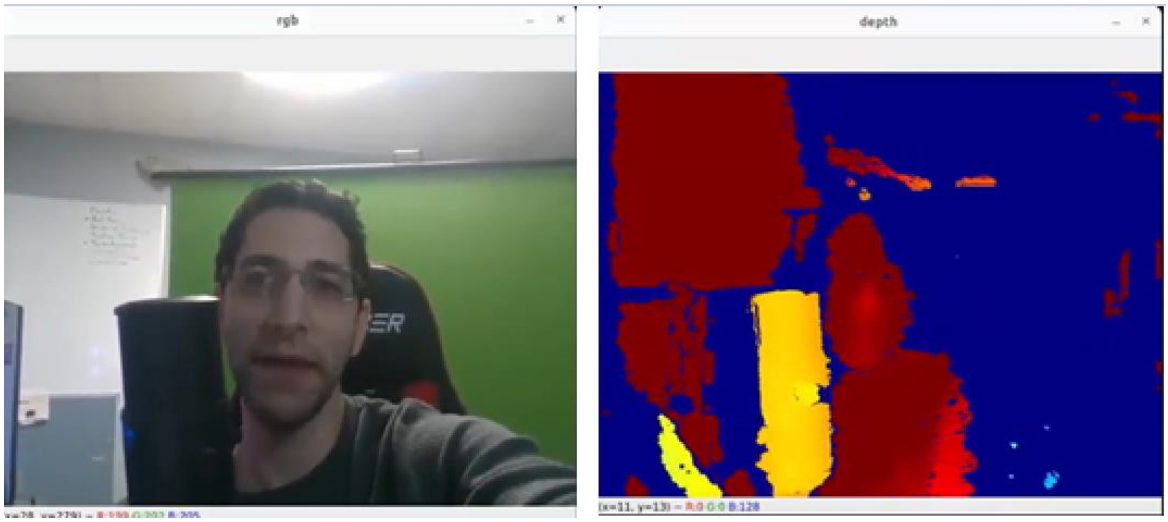


Рисунок 1.3 – Карта глибини отримана за допомогою Intel RealSense ID

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	FaceFinder	FaceTec	Azure Face API	BioID	Intel RealSense ID
Захист від фізичних атак	+	+	+	+	+
Захист від цифрових атак (діпфейків)	+	-	-	+	+
Отримання відеопотоку (файл/камера)	Файл і камера	Камера	Файл і камера	Камера	Файл і камера
Розпізнавання осіб (1 / багато)	Багато	1	Багато	1	Багато
Тип захисту (пасивний/активний)	Пасивний	Активний	Пасивний	Пасивний і Активний	Пасивний
Потреба у спеціальних сенсорах	-	-	-	-	+
Робота в реальному часі	+	+, але можливі сильні затримки	+	+	+
Платформа	ПК	Моб./Веб	Хмара	Моб./Веб	ПК + Сенсор

Продовження таблиці 1.1

Функціонал	FaceFinder	FaceTec	Azure Face API	BioID	Intel RealSense ID
Відкритість рішення та ціна використання	Відкрите	Пропрієтарне; ціна за запитом	Відкрите API; від $1.00 за 1,000 транзакцій	Пропрієтарне; ціна за запитом	Відкрите SDK

У результаті порівняння видно, що більшість комерційних рішень у сфері розпізнавання облич вимагає постійного підключення до мережі або спеціального обладнання, має закриту архітектуру та непрозоре ціноутворення. Поряд із цим якість FAS часто не відповідає заявленій, або не має об’єктивних метрик. Для створення життєздатного продукту потрібно забезпечити пасивний захист без залучення користувача, який добре працює зі звичайним обладнанням без спеціальних сенсорів, обробляє дані в реальному часі, може розгортатися локально, і дозволяє захоплювати обличчя для подальшої ідентифікації. Такий набір властивостей робить розробку гнучким і доступним рішенням, здатним бути конкурентноспрожним навіть поряд з великими комерційними продуктами.

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Систиматичні огляди літератури та попередні дослідження показують, що вже створенні нейронні мережі стабільно добре виконують задачі розпізнавання та виявлення облич [22]. Втім, найкращі в своему класі моделі можуть потребувати понад 10 гігабайтів відеопам’яті для запуску, при цьому не забезпечуючи достанью для роботи в реальному часі швидкість.

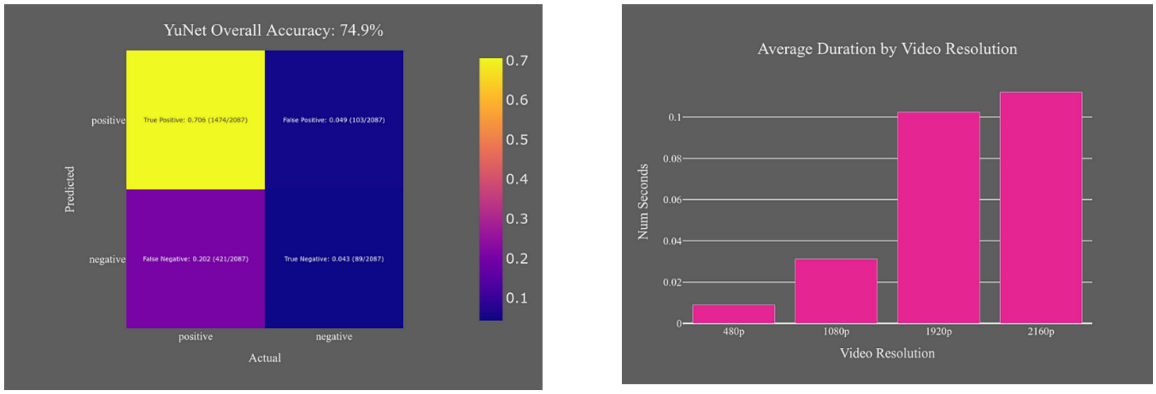


Рисунок 1.4 – Якість та швидкість роботи моделі YuNet

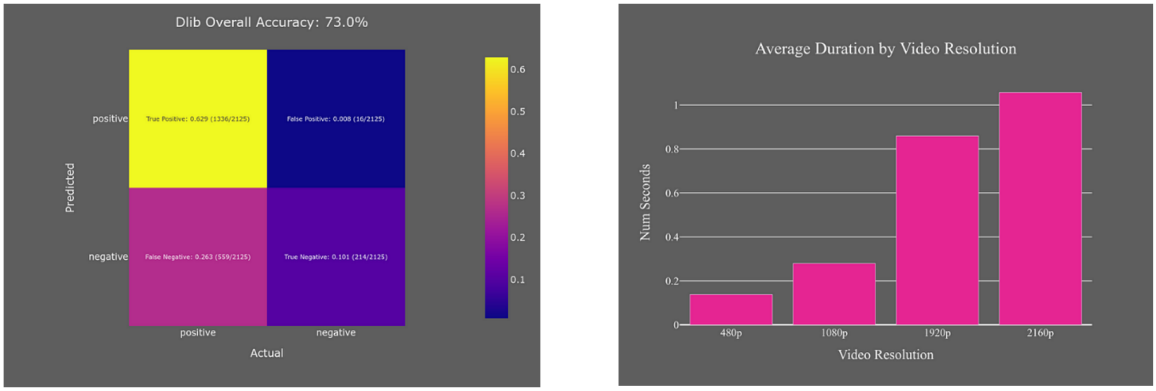


Рисунок 1.5 – Якість та швидкість роботи моделі Dlib

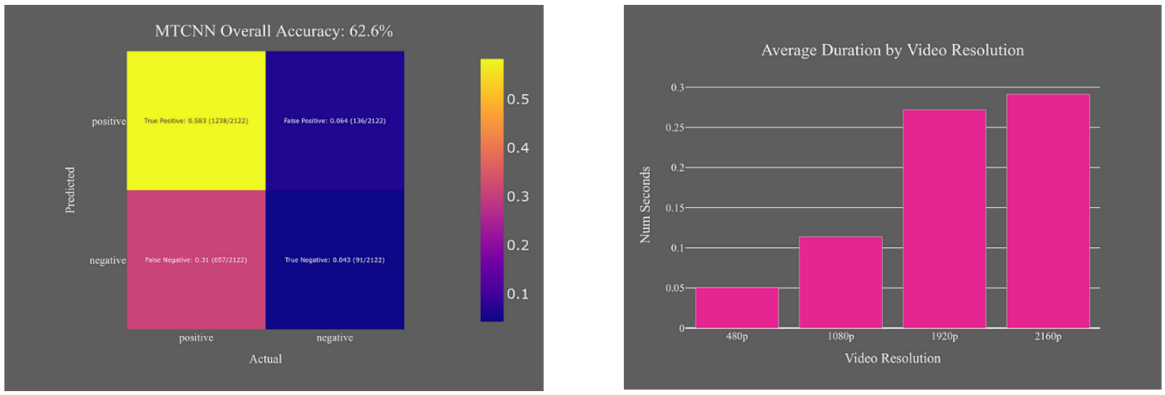


Рисунок 1.6 – Якість та швидкість роботи моделі MTCNN

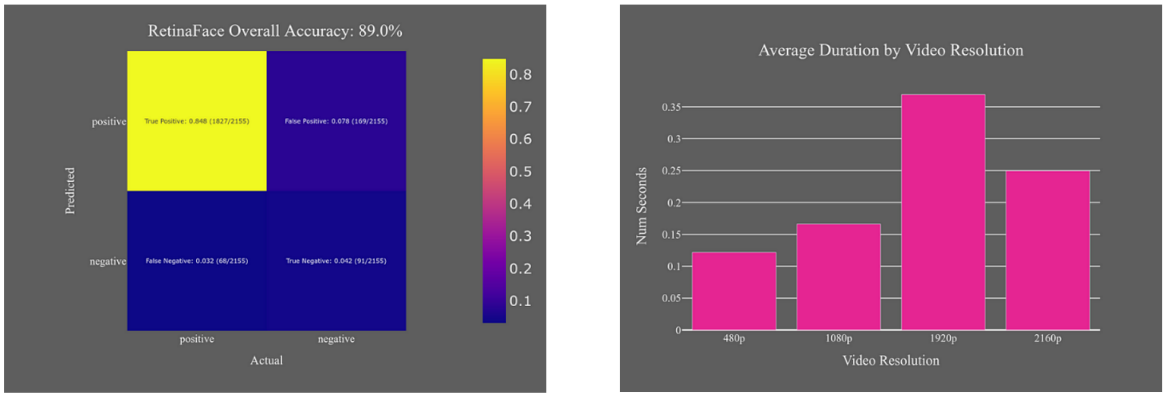


Рисунок 1.7 – Якість та швидкість роботи моделі RetinaFace

З наведених на 1.4-7 рисунках графіків, отриманих з [23], бачимо значну перевагу RetinaFace [24] в якості виявлення, однак YuNet [25] має значно краще співвідношення між якістю і швидкістю роботи.

Порівнюючи бібліотеки для розпізнавання облич, які підходять для роботи в реальному часі, є кілька варіантів кожен з яких має свої сильні й слабкі сторони. Наприклад, нейрона мережа ArcFace представлена в InsightFace демонструє високу, але потребує більше ресурсів і складнішої інтеграції. Dlib, у свою чергу, пропонує збалансоване рішення між продуктивністю та якістю, яке базується на моделі ResNet, яка генерує 128-вимірні вектори ознак для точного порівняння облич. Для датасету [26] Dlib показує точність в 96.8%, а ArcFace 96.7%. Окрім цього, Dlib дозволяє забезпечити вирівнювання та нормалізацію облич на основі ключових точок, гарантуючи однакове положення й масштаб. Завдяки цьому Dlib є оптимальним вибором для завдань розпізнавання облич у реальному часі, де важливі і точність, і швидкодія.

У більшості випадків, виявлення несправжніх облич зводиться до бінарної класифікації. Однак, ця продблема розвивається динамічіно, бо паралельно із захистом покращюються і методи обману. Більше того, на відміну від класичних представників такого роду задач, як визначення статі особи, наповненння окремого кадру чи зображення, сама форма обличчя, зачіска, стиль одягу чи інша високорівнева симантика майже не несе корисної інформації [20].

Обирабчи підхід до FAS, після аналізу аналогів можна зробити висновок, що пасивний підхід до виявлення живості (FAS) є кращим вибором, оскільки забезпечує високу точність, швидку обробку (лише одне селфі), менше навантаження на мережу та процесор, просту інтеграцію в існуючі системи та мінімальний рівень фрустрації користувачів, адже не потребує виконання додаткових дій. До того ж, він важчий для обходу, бо не розкриває методів виявлення і базується на мікротекстурному аналізі зображення, що ускладнює підробку навіть з використанням високоякісних дисплеїв і камер.

Історично методи FAS розвивалися від традиційних дескрипторів, таких як LBP, HOG або виявлення характерних рухів (наприклад, кліпання чи rPPG-сигнали), але їх слабка здатність до узагальнення призвела до поступового переходу на підходи з глибоким навчанням. Сучасні моделі базуються на CNN і трансформерах, використовують як бінарну, так і піксельну супервізію, зокрема, за допомогою псевдо-глибинних мап, карт відбиття або карт артефактів [12]. Актуальними є методи доменної адаптації (DA), генералізації (DG), навчання без джерел (SFDA) та адаптації під час інференсу (TTA). Все ширше застосовуються мультимодальні підходи з одночасним використанням кольорового та інфрачервоного зображення, глибини та аудіо даних, а також візуально-мовні моделі (наприклад, CLIP), які покращують інтерпретованість та універсальність рішень [21].

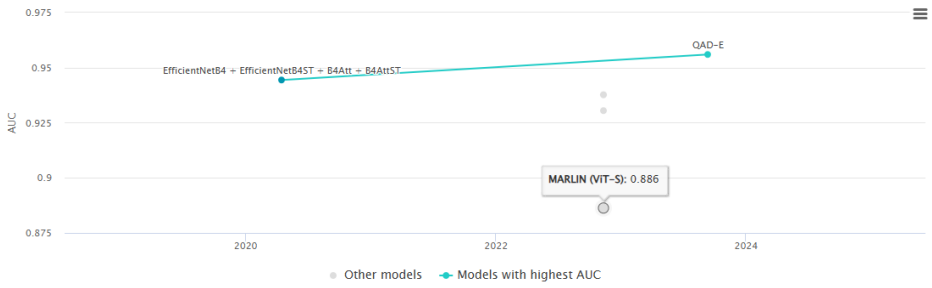


Рисунок 1.8 – Якість сучасних моделей в PAD[27]

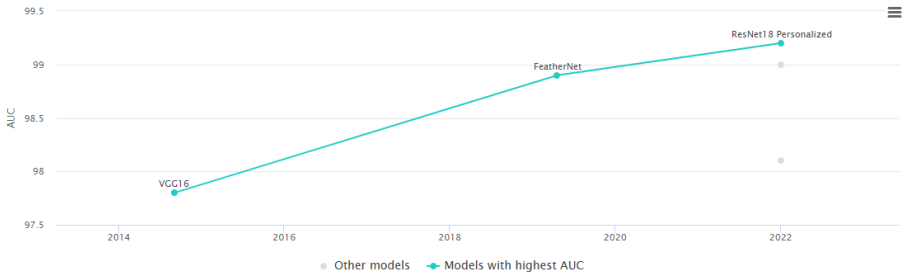


Рисунок 1.9 – Якість сучасних моделей в DAD[28]

З наведених графіків бачимо, що за умови продуманого підходу до навчання навіть невеликі меделі добре справляються з FAS на окремих датасетах.

CelebA-Spoof dataset був обраний як основний для тренування, оскільки є публічно доступним і на момент підготовки дослідження є найбільшим серед наборів даних для задач виявлення фізичних атак на систкми розпізнавання (PAD), згідно з [29]. Для задач виявлення цифрових атак (DAD) актуальним є інший публічний датасет – DFD (DeepFake Detection)[30], який містить приклади відео з дипфейками. Попри те, що для навчання моделей краще використовувати окремі зображення, адже з кожного відео можна взяти лише обмежену кількість кадрів, щоб уникнути перенавчання, для адекватної оцінки системи розпізнавання необхідне саме відео. Тому був створений окремий тестовий набір даних для PAD на основі різних публічно доступних відео з Kaggle. Хоча він поступається CelebA-Spoof за якістю анотації, цей набір містить повноцінні відео, що дає змогу краще перевірити працездатність системи в умовах, наближених до реальних. Додатковою перевагою є те, що всі використовувані датасети доступні на платформі Kaggle, що дозволяє скористатися її інфраструктурою для тренування моделей, зокрема використанням GPU.

Підсумовуючи, для розробки необхідно підібрати коректні моделі для виявлення і розпізнавання обличчя, що не будуть перешкоджати роботі в реальному часі. Для забезпечення FAS найнадійнішим рішенням буде ансамблювання невеликих моделей, які спеціалізуються як на PAD, так і на

DAD, донавчених за допомогою TL. Обрані датасети будуть використані для дотреновування моделей, так і оцінки якісті їх роботи.

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання бізнес-процесу використовується BPMN модель (рисунок 1.10) (рисунок слугує ілюстрацією BPMN моделі).

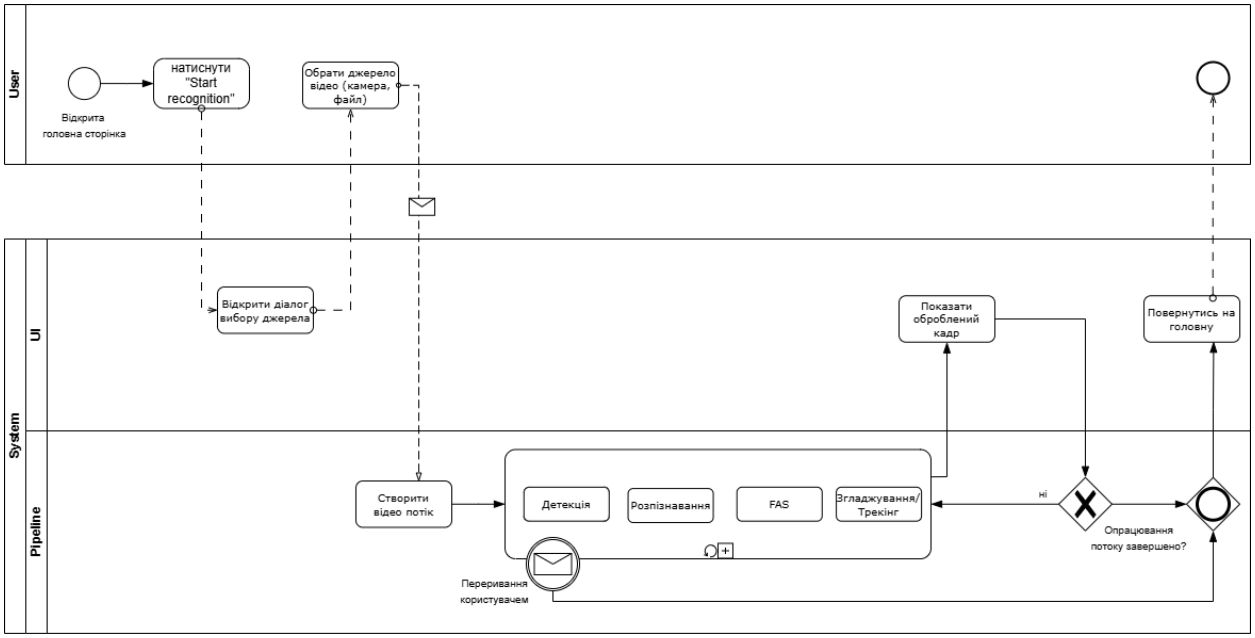


Рисунок 1.10 – Модель бізнес-процесу FAS пайплайну

Опис моделі бізнес-процесу захоплення нового обличчя:

– користувач переходить до вікна захоплення.;
– вводить назву мітки.;
– якщо мітка не відповідає вимогам – виводиться повідомлення про помилку;
– користувач послідовно захоплює 20 зображень;
– система виділяє характеристики з кожного зображення;
– система кодує та зберігає характеристики.

Опис моделі бізнес-процесу редагування міток:

– користувач переходить до вікна менеджера міток;
– обирає мітку і натискає кнопку "Перейменувати";

– вводить нову назву;

– якщо назва не відповідає вимогам – виводиться помилка;

– мітка оновлюється, система кодує і зберігає зміни.

Опис моделі бізнес-процесу видалення міток:

– користувач відкриває менеджер міток;

– обирає мітку і натискає "Видалити";

– мітка видаляється, система фіксує і зберігає зміни.

Опис моделі бізнес-процесу розпізнавання облич у реальному часі:

– користувач обирає джерело відеопотоку – вебкамеру чи готовий файл.

– система оброблює відеопотік ;

– із достатньою для роботи в реальному часі частотою виконується виявлення, розпізнаваня та пошук штучно змінених облич

– процес завершується у випадку переривання користувачем або вичерпанням відеопотоку.

Висновки до розділу

У цьому розділі було проведено комплексне передпроєктне дослідження предметної області, пов’язаної з розпізнаванням облич та виявленням підроблених зображень у відеопотоці. Було сформульовано завдання дипломного проєкту – створення десктопної системи FaceFinder, здатної виявляти, розпізнавати обличчя та виконувати Face Anti-Spoofing (FAS) у реальному часі без використання хмарних сервісів. Визначено перелік етапів, необхідних для реалізації проєкту, включаючи аналіз предметної області, розробку архітектури, реалізацію програмного забезпечення, тестування та супровід.

Проаналізовано предметну галузь, яка охоплює задачі виявлення маніпуляцій над обличчями в умовах зростаючої загрози цифрових атак, таких як дипфейки або фізичні підробки. Описано основні виклики, пов’язані з недосконалістю існуючих рішень, проблемами генералізації моделей, обмеженою доступністю якісних датасетів і потребою в оптимізації обчислень для досягнення роботи в реальному часі. Обрано підхід до побудови системи на основі ансамблю легковагових моделей з подальшою оптимізацією архітектури.

Проведено порівняльний аналіз існуючих програмних рішень, таких як FaceTec, Azure Face API, BioID та Intel RealSense ID. У таблиці наведено ключові відмінності між ними та проєктованим FaceFinder, зокрема щодо рівня захисту, типу взаємодії з користувачем, потреби в додатковому обладнанні та відкритості рішення. Встановлено, що запропонована розробка вигідно вирізняється своєю автономністю, прозорістю, гнучкістю та здатністю працювати на звичайному обладнанні.

Було розглянуто відомі алгоритмічні та технічні підходи до реалізації розпізнавання та виявлення фальсифікацій. Обґрунтовано вибір моделей YuNet для детекції, Dlib з ResNet для розпізнавання, а також ансамблю компактних CNN-моделей для FAS. Обрані методи забезпечують необхідний баланс між якістю, швидкодією та можливістю інтеграції в реальному часі. Визначено основні датасети (CelebA-Spoof, DFD та власний відеодатасет), які будуть використані як для навчання, так і для тестування системи.

Також було змодельовано ключові бізнес-процеси у вигляді BPMN-діаграм, що охоплюють захоплення, редагування, видалення міток облич та розпізнавання в реальному часі. Це дозволило формалізувати функціональність майбутньої системи, окреслити взаємодію користувача з інтерфейсом та визначити структуру ключових компонентів. Усе це створює надійну основу для переходу до етапу проєктування архітектури програмного забезпечення.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями з точки зору користувачів наведена у графічному матеріалі, креслення 1.

В таблицях 2.1-2.6 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Захоплення обличчя
Use case ID	UC-01
Goals	Додати нове обличчя для розпізнавання
Actors	Користувач
Trigger	Користувач бажає додати нове обличчя для подальшого розпізнавання
Pre-conditions	-
Flow of Events	Користувач переходить до вікна захоплення, вводить мітку для обличчя. На екрані з’являється вікно, на якому виділяється тільки 1 обличчя(0 якщо жодного немає в кадрі). Після цього користуач 20 разів натискає клавішу q для захоплення обличчя в поточному ракурсі. Після захоплення 20 зображень, користувач отримує повідомлення про успішне захоплення обличчя і повертається на початковий екран.
Extension	В випадку введення не коректної мітки, користувач отримує відповідне повідомлення і повертається на початковий екран. У випадку переривання процесу захоплення, створюється папка із уже захопленими зображеннями
Post-Condition	До бази даних додається захоплене обличчя із введеною міткою як назвою

Таблиця 2.2 – Варіант використання UC-02

Use case name	Отримання відеопотоку
Use case ID	UC-02
Goals	Надати доступ до потоку з вебкамери або відеофайлу
Actors	Користувач, Система
Trigger	Користувач запускає захоплення обличчя, розпізнавання або виявлення облич
Pre-conditions	Доступна вебкамера або файл з відео у відповідному форматі
Flow of Events	При запуску процесу захоплення або розпізнавання система запитує джерело відео. Якщо користувач обирає вебкамеру, відкривається доступ до потоку в реальному часі. Якщо вибрано файл, система завантажує його для покадрової обробки. Відеопотік передається у відповідний модуль для подальшого використання.
Extension	У разі недоступності джерела потоку користувач отримує відповідне повідомлення, і процес переривається
Post-Condition	Відеопотік ініціалізовано, він доступний для обробки

Таблиця 2.3 – Варіант використання UC-03

Use case name	Редагування міток
Use case ID	UC-03
Goals	Змінити назву або видалити захоплену мітку
Actors	Користувач
Trigger	Користувач бажає змінити або видалити мітку захопленого обличчя
Pre-conditions	Наявність хоча б однієї мітки

Продовження таблиці 2.3

Flow of Events	Користувач переходить до менеджера міток, де бачить список захоплених міток із прев'ю облич. Обравши потрібну мітку, він має можливість або змінити її назву, або видалити. У разі перейменування відкривається діалогове вікно для введення нової назви, після чого зміни застосовуються. У разі видалення мітки користувач підтверджує дію, і відповідна з мітка видаляється з системи.
Extension	Якщо введена некоректна назва – з'являється повідомлення. Якщо список міток порожній – показується відповідне повідомлення.
Post-Condition	Мітку перейменовано або видалено.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Виявлення облич
Use case ID	UC-04
Goals	Визначити координати облич на кадрах з відеопотоку
Actors	Система
Trigger	Початок роботи з відеопотоком для захоплення чи розпізнавання
Pre-conditions	Ініціалізований та активний відеопотік
Flow of Events	Система отримує кадри з відеопотоку та застосовує модель виявлення облич. На кожному кадрі визначаються координати облич, що використовуються для подальших дій
Extension	Якщо обличчя не виявлено, кадр пропускається для розпізнавання та FAS
Post-Condition	Отрмані координати облич на кадрі (чи нічого у випадку їх відустності)

Таблиця 2.5 – Варіант використання UC-05

Use case name	Розпізнавання облич
Use case ID	UC-05
Goals	Визначити особу за виявленим обличчям на основі попередньо збережених міток
Actors	Система
Trigger	Користувач обирає режим розпізнавання облич
Pre-conditions	Активний відеопотік та принаймні одне виявлене обличчя
Flow of Events	Система аналізує координати виявлених облич, отримує векторні ознаки та порівнює їх з попередньо збереженими мітками. Якщо схожість достатня, виводиться ім’я особи на екрані
Extension	Якщо збігів не знайдено – обличчя відмічається як “невідоме”
Post-Condition	Ідентифіковано або відмічено невідоме обличчя

Таблиця 2.6 – Варіант використання UC-06

Use case name	Виявлення штучно змінених облич
Use case ID	UC-06
Goals	Переконатися, що виявлене обличчя належить живій людині
Actors	Система
Trigger	Наявність розпізнаних облич
Pre-conditions	Активний відеопотік і виявлене обличчя
Flow of Events	Використовуючи виділений регіон з обличчям, система використовує внсамбль нейронних мереж для отрияння передбачення ймовірності підробки
Extension	Якщо присутні ознаки маніпуляцій чи підробки, обличчя позначається як несправжнє.
Post-Condition	Кожне обличчя класифіковане як справжнє або підозріле (несправжнє)

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.7 наведено загальну модель вимог, а в таблиці 2.8 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.7 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Захоплення обличчя	FR-1	Високий	Середній
2	Перегляд захоплених міток	FR-2	Середній	Низький
3	Редагування захоплених міток	FR-3	Середній	Низький
4	Видалення захоплених міток	FR-4	Середній	Низький
5	Отримання відеопотоку	FR-5	Високий	Середній
6	Виявлення облич у відеопотоці	FR-6	Високий	Високий
7	Розпізнавання облич у відеопотоці	FR-7	Високий	Високий
8	Збереження результатів розпізнавання	FR-8	Середній	Середній
9	Виявлення штучно змінених облич (FAS)	FR-9	Високий	Високий
10	Перевірка формату мітки	FR-10	Низький	Низький

Таблиця 2.8 – Перелік функціональних вимог

Назва	Опис
FR-1	Захоплення обличчя. Система повинна надавати можливість захоплювати одне нове обличчя за раз та призначати йому унікальну мітку.

Продовження таблиці 2.8

Назва	Опис
FR-2	Перегляд захоплених міток. Система повинна надавати можливість переглядати список усіх захоплених облич із відповідними мітками.
FR-3	Редагування захоплених міток. Система повинна дозволяти перейменовувати мітки, призначені раніше захопленим обличчям.
FR-4	Видалення захоплених міток. Система повинна надавати можливість видаляти захоплені обличчя разом з їх мітками.
FR-5	Отримання відеопотоку. Система повинна забезпечувати можливість отримувати відеопотік із вебкамери в реальному часі, або з попередньо записаного відеофайлу.
FR-6	Виявлення облич у відеопотоці. Система повинна автоматично виявляти обличчя у кожному кадрі відеопотоку.
FR-7	Розпізнавання облич у відеопотоці. Система повинна розпізнавати вже захоплені обличчя у поточному відеопотоці на основі збережених міток.
FR-8	Збереження результатів розпізнавання. Система повинна надавати можливість зберігати результати розпізнавання (з мітками та часом появи) у файл або базу даних.
FR-9	Виявлення штучно змінених облич (FAS) Система повинна виявляти чи обличчя є живим (реальним), чи відтвореним з фото чи відео, масками, підміненими за допомогою генеративних моделей тощо.

Продовження таблиці 2.8

Назва	Опис
FR-10	Перевірка формату мітки Введена мітка повинна містити лише латинські літери, цифри або символ підкреслення (_) та бути унікальною

№	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06
FR-1	+					
FR-2		+				
FR-3			+			
FR-4				+		
FR-5	+				+	+
FR-6					+	+
FR-7	+				+	+
FR-8					+	+
FR-9					+	+
FR-10					+	+

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Програмне забезпечення повинно відповідати ряду нефункціональних вимог, що забезпечують його стабільну, зручну та ефективну роботу.

Система повинна коректно обробляти виняткові ситуації, виводити повідомлення про помилки зрозуміло для користувача та забезпечувати безпечне завершення роботи у разі критичних збоїв.

Система повинна функціонувати в операційних системах сімейства Windows 64-bit, зокрема Windows 10.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим, з чітким відображенням елементів керування та логічною структурою навігації.

Система повинна забезпечувати розпізнавання облич у режимі реального часу без суттєвих затримок.

2.4 Аналіз системних вимог

Системні вимоги визначають апаратну конфігурацію, необхідну для стабільної та ефективної роботи програмного забезпечення в режимі реального часу.

Рекомендована конфігурація для роботі в реальному часі:

– Процесор: Intel Core i7-9750H або аналогічний (6 ядер, 12 потоків).

– Оперативна пам’ять: не менше 16 ГБ.

– Відеокарта: дискретна відеокарта з обсягом відеопам’яті не менше 6 ГБ (наприклад, NVIDIA GTX 1660 Ti або краща).

– Операційна система: Windows 10 x64.

За умови такої конфігурації система повинна забезпечувати стабільну роботу з частотою хоча б 10 кадрів на секунду при обробці відеопотоку в режимі реального часу.

2.5 Аналіз економічних показників програмного забезпечення

Здійснимо оцінку економічних показників програмного забезпечення здійснюється на основі методики Feature Points Analysis (FPA), яка дозволяє визначити трудомісткість, вартість і тривалість розробки, виходячи зі специфікації вимог до системи. У межах цього дипломного проекту було виділено основні компоненти, що забезпечують функціональність розпізнавання облич, маркування, редагування міток, перегляд результатів, валідацію реальності та виявлення атак. Відповідно до класифікації FPA, до системи віднесено: 1 внутрішній логічний файл (ILF) для збереження міток облич, 1 зовнішній інтерфейсний файл (EIF) – наприклад, для імпорту моделей розпізнавання, 4 входи користувача (EI), пов’язані із додаванням облич,

перейменуванням міток та запитами на перевірку, 3 виходи (EO) – повідомлення, відображення результатів і 2 запити (EQ) на фільтрацію або пошук. Усі ці елементи мають середню складність. На основі таблиці вагових коефіцієнтів отримано загальну кількість функціональних точок – 56 FP.

Для визначення економічної трудомісткості використано середній коефіцієнт 10 годин на 1 FP, що відповідає типовим показникам проектів середньої складності. Таким чином, орієнтовна загальна трудомісткість становить 560 годин. Ураховуючи, що проект виконує один розробник із погодинною ставкою $5 (при медіанній місячній зарплаті Junior ML-Engineer 800$ і стандартному навантаженні 160 годин на місяць), повна вартість реалізації проекту становить 2800$. За такого навантаження тривалість розробки оцінюється приблизно у 3,5 місяці.

Порівнюючи із типовим проектом на 100 FP (що вимагає 1000 годин розробки та коштує близько $5000), можна зробити висновок, що дане рішення належить до середнього класу за обсягом і вартістю. Наявність лише одного виконавця не створює критичного навантаження, а розрахована трудомісткість цілком укладається у часові межі, відведені на виконання дипломного проєкту. Отже, з точки зору економіки, проєкт є збалансованим за трудовими і фінансовими ресурсами та має високу рентабельність в умовах індивідуальної розробки.

2.6 Постановка завдання на розробку програмного забезпечення

У межах дипломного проєкту передбачається створення десктопного програмного забезпечення для розпізнавання облич із вбудованою перевіркою їх автентичності за допомогою методів FAS. Система працюватиме з відеопотоком, який надходитиме з вебкамери або відеофайлу.

Метою розробки є підвищення ефективності ідентифікації осіб у відеопотоці, що дозволить спростити процес перевірки та ідентифікації користувачів працівниками служб безпеки підприємства, чи користувачами що хочуть перевірити достовірність окремих відео. Це досягається шляхом

впровадження методів комп’ютерного зору для розпізнавання облич та визначення, чи відбувались із ними штучні маніпуляції, а також створенням доступного та зручного користувацького інтерфейсу.

Реалізація передбачає створення програмного модуля, який забезпечуватиме захоплення та обробку відео, виявлення і розпізнавання облич, перевірку їх на справжність, збереження результатів у локальній базі даних та візуалізацію даних у графічному інтерфейсі. У результаті розробки буде отримано програмний продукт, здатний у реальному часі ідентифікувати особу та перевіряти автентичність її обличчя без використання сторонніх серверів чи хмарних сервісів.

Висновки до розділу

У даному розділі було здійснено формування вимог до програмного забезпечення, що розробляється в межах дипломного проєкту. Проведено аналіз та опис основних функціональних можливостей системи, визначено сценарії взаємодії користувача з програмою, побудовано діаграму варіантів використання, а також надано деталізовані описи кожного з ключових use-case.

На основі визначених функцій і логіки роботи ПЗ сформовано перелік функціональних вимог із поділом за пріоритетами та рівнями ризику. У результаті цього етапу забезпечено чітке розуміння того, які задачі система повинна реалізовувати та які очікування мають бути задоволені на виході. Особливу увагу приділено підтримці FAS, що дозволяє системі не лише розпізнавати обличчя, а й виявляти їх справжність, що є критично важливим для сучасних систем безпеки.

Також було розроблено таблицю функціональних вимог з описом кожної з них, що дозволяє в подальшому використовувати її як основу для створення матриці трасування вимог та контролю їх виконання на етапі розробки. Включено і нефункціональні вимоги, що визначають стабільність,

швидкодію, сумісність із платформою Windows 10 x64, а також вимоги до зручності інтерфейсу користувача.

Визначено системні вимоги до обладнання, необхідного для роботи ПЗ в режимі реального часу з використанням сучасних методів комп’ютерного зору. Враховано технічні ресурси, необхідні для стабільної роботи при обробці відеопотоку та виконанні обчислювально складних операцій розпізнавання й виявлення підробок.

Окремо проаналізовано економічні показники проекту на основі методики FPA (Feature Points Analysis), що дало змогу обґрунтувати обсяг трудових витрат, визначити терміни виконання та вартість розробки. Показано, що розробка проєкту в межах одного виконавця є економічно доцільною та збалансованою як за ресурсами, так і за тривалістю реалізації.

Таким чином, у цьому розділі сформовано повну основу для наступного етапу проєктування системи – розробки архітектури програмного забезпечення, що відповідає всім технічним, функціональним, нефункціональним і економічним вимогам.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для реалізації даного програмного забезпечення було обрано модульну багаторівневу архітектуру, яка частково базується на патерні Model–View–Controller (MVC). Такий підхід забезпечує гнучке розділення відповідальностей, високу масштабованість та зручність тестування компонентів системи. Однак, через необхідність виконання складних обчислювальних задач в реальному часі (виявлення облич, розпізнавання особи, FAS), було прийнято рішення не строго дотримуватись патерну у випадках, коли це дозволяло уникати дублікації обчислень та коду.

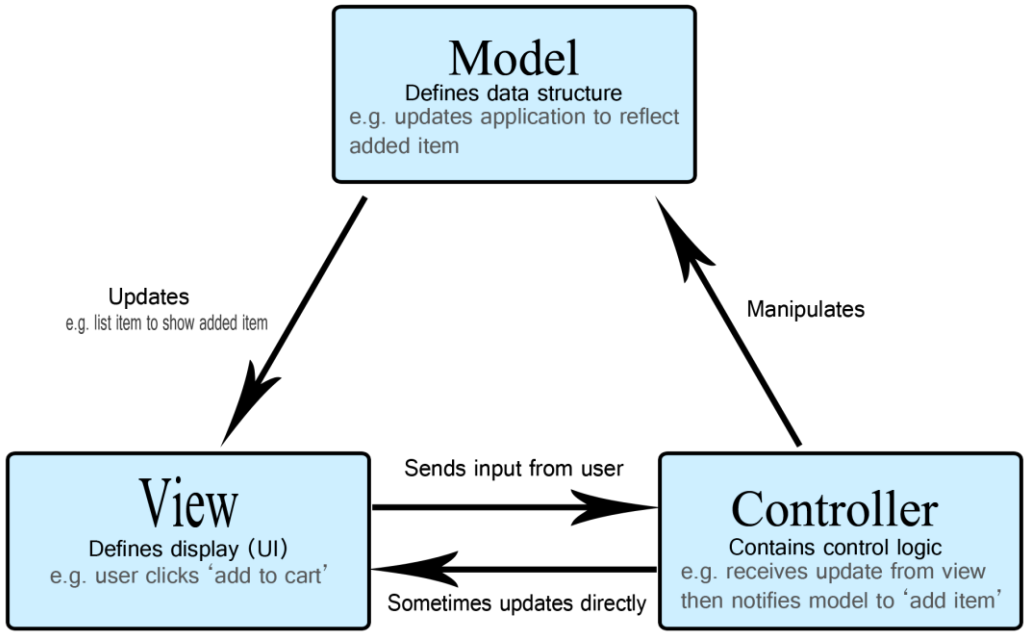


Рисунок 3.1 – компоненти патерну MVC

View компонент відповідає за введення від користувача та вивід результатів розпізнавання у графічний інтерфейс. Controller реалізований класами, що координують обробку відеопотоку через пайплайни: для захоплення обличчя чи забезпечення FAS, а ткож взаємодіють з сховищем

захоплених облич. Model представлений класами, які зберігають захоплені та поточні дані, а також реалізують їх обробку, включно з використанням нейронних мереж.

Архітектура системи побудована з урахуванням розділення відповідальностей, підтримки масштабованості та гнучкої інтеграції нових моделей та алгоритмів. Детальніше вона представлена у вигляді діграм C4 Model у графічному матеріалі, креслення 2, аркуші 1-2.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Під час розробки програмного забезпечення для розпізнавання облич із виявленням штучно змінених було прийнято низку архітектурних рішень для забезпечення модульності, зручності підтримки та локальної обробки даних. Архітектура додатку є подібною до MVC, де окремо реалізовано обробку даних, логіку взаємодії та графічний інтерфейс. Такий підхід дозволяє легко замінювати окремі модулі без впливу на решту системи.

Для гнучкості в коді застосовано патерни Adapter та Decorator – перший дозволяє узгоджено працювати з різними моделями розпізнавання, другий – обгортати та доповнювати логіку перевірки результатів. Користувачі не реєструються через зовнішні сервіси – доступ до програми здійснюється локально.

Мовою програмування було обрано Python, завдяки зручності і широкій підтримці інтеграції з бібліотеками штучного інтелекту. Нефункціональні вимоги забезпечуються через обробку помилок, підтримку Windows-платформи, інтуїтивний інтерфейс, реалізований через за допомогою PyQt, та швидку обробку даних завдяки оптимізованим моделям у PyTorch та ONNX. Для розпізнавання облич використовується бібліотека face_recognition, що є надбудовою над Dlib, а для роботи з відео – OpenCV. Такий стек технологій дозволив реалізувати проєкт швидко, ефективно і з можливістю майбутнього розширення.

Як середовище розробки було обрано Visual Studio Code (VS Code). Це легке, швидке та безкоштовне IDE з відкритим кодом, яке підтримує велику кількість мов програмування, зокрема Python. Основними перевагами є зручний інтерфейс, підтримка розширень і інтеграція з системами контролю версій. Завдяки численним плагінам, таким як Python Extension, Jupyter, Pylance та інші, можливо швидко налаштувати робоче середовище під потреби конкретного проєкту. Для навчання моделей у проєкті було використано платформу Kaggle, яка надає безкоштовний доступ до GPU та TPU-ресурсів. Це дозволяє ефективно тренувати навіть важкі моделі глибокого навчання без потреби в локальному обладнанні. Крім того, Kaggle підтримує прямий доступ до великої кількості публічних датасетів, включаючи ті, що використовуються для задач FAS.

3.3 Конструювання програмного забезпечення

Система "FaceFinder" реалізована як десктопний застосунок з графічним інтерфейсом, бізнес-логікою для обробки відеопотоку та нейромережевими моделями для виявлення, розпізнавання та FAS.

3.3.1 Розробка алгоритму виявлення штучно змінених облич у відеопотоці.

Алгоритм виявлення штучно змінених облич у відеопотоці можна розбити на кілька послідовних кроків: отримання відеопотоку, виявлення облич безпосередньо, кодування цих облич і порівняння з навною базою для розпізнавання, виявлення штучних змін, тобто наявності ознак PAD або DAD у виявлених обличчях. Додатковими етами можуть включати широкий спектр постобробки та бізнес лоігіки на основі отриманих результатів, в рамках цього проекту розглядаємо згладжування та трекінг облич. Робота алгоритму в реальному часі передбачає можливість опрацювання десятків ітерації на секунду.

Для створення відеопотоку було використано можливості бібліотеки OpenCV, яка дозволяє захоплювати відеопотік як з вебкамер, так і з відео. Оскільки для обробки в реальному часі необхідний швидкий доступ до кадрів потоку з мінімальною латентністю, було використано модуль Tread. Потік постійно оновлюється паралельно виконанню обробки попередньо взятого кардру, що значно зменшує затримки.

При попередньому досліджені, для виявлення облич значно виділились моделі YuNet та RetinaFace. YuNet – це надлегка та швидка модель для виявлення облич, розроблена для роботі в реальному часі. Її архітектура побудована повністю на глибоких згортках (depthwise separable convolutions), а кількість параметрів становить лише близбко 75 тис. Основу складають п’ять послідовних блоків (DWUnit або DWBlock), кожен з яких працює зі 64 каналами. Для об'єднання контексту на різних масштабах використовується Tiny-FPN – варіант класичної FPN, що зменшує кількість параметрів до 12% від оригіналу. Голова моделі реалізує безякірний (anchor-free) підхід: кожен піксель прогнозує одне обличчя (без сітки якорів), що пришвидшує обчислення та спрощує постобробку. Крім координат імовірнісної рамки, YuNet також прогнозує 5 ключових точок (landmarks). Завдяки своїй ефективності модель досягає до 100 кадрів на секунду на CPU, проте її точність падає в умовах складного освітлення, великих кутів повороту або часткових перекриттів. Як альтернатива була додана інша модель RetinaFace, яка навпаки, є повноцінною якірною (anchor-based) моделлю, орієнтованою на високу точність навіть у складних умовах. Її архітектура може використовувати різні бекбони – найчастіше ResNet-50, що забезпечує глибокий контекст, або спрощені варіанти (ResNet-18, MobileNet-0.25) для пришвидшення. Зв'язуюча частина моделі RetinaFace – це повна реалізація FPN із модулями контексту SSH (згортки з ядрами 3 на 3 та 5×5 з дилатацією), які збільшують ефективну зону сприйняття. Голова моделі також прогнозує не лише класи і рамки, але й 5 ключових точок. RetinaFace використовує щільну сітку якорів (до 100 тис. на зображення) з оптимізацією через Online Hard

Example Mining. Хоча така модель є важчою та повільнішою, вона суттєво переважає за точністю, особливо в умовах складного фону, нахилів обличчя або слабкого освітлення.

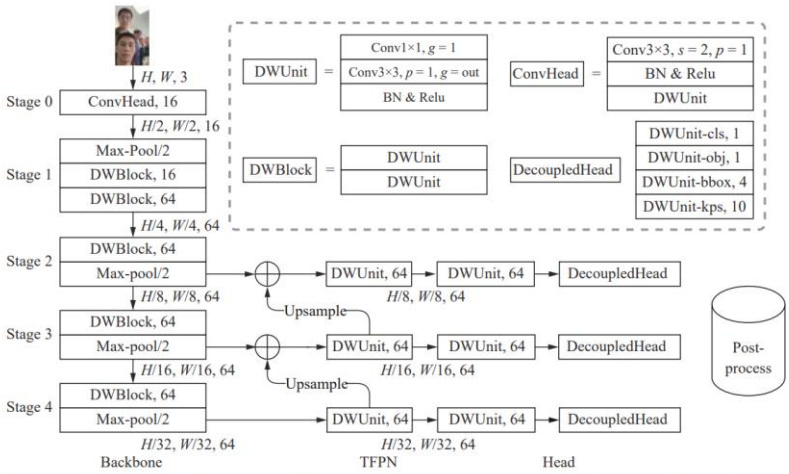


Fig. 4 Architecture of YuNet. It consists of a backbone, a TFPN neck and a head, which are all based on depthwise separable convolution. "ConvHead, 16" indicates a module named ConvHead with 16 output channels. All the head outputs will be concatenated together and produce detection results via non-maximum suppression (NMS).

Рисунок 3.2 – архітекткура YuNet[25]

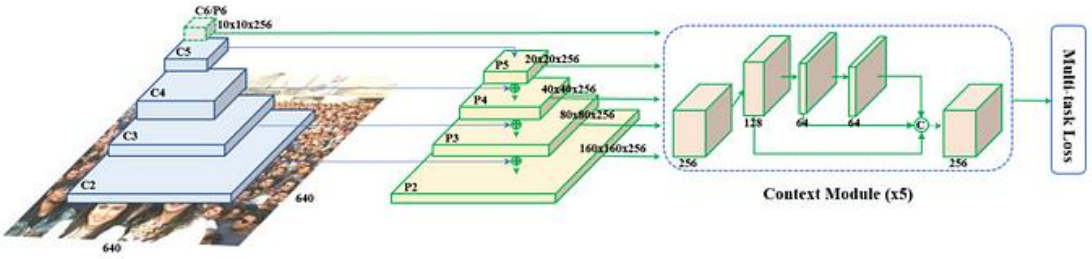


Figure 2. An overview of the proposed single-stage dense face localisation approach. RetinaFace is designed based on the feature pyramids with independent context modules. Following the context modules, we calculate a multi-task loss for each anchor.

Рисунок 3.3 – архітекткура RetinaFace[24]

Для розпінавання обличчя використовуються можливості бібліотеки Dlib. За допомогою MMOD (Max-Margin Object Detection)[31] виявляє 68 точок[32] на обличчі, після чого вони закодовуяться вектор властивостей довжиною 128 [32] за допомогою ResNet моделі, що дає сожливість порівнювати зображення із раніше збереженим – знайшовши різницю кодувань, отримаємо відстань між зображеннями, і чим вона менша, тим більша ймовірність, що це одна особа [33]. Пакет face_recognition дає зручний інтерфейс для такого рішення побудованого на основі Dlib.

Для реалізації модуля Face Anti-Spoofing (FAS) безпосередньо, у проєкті було вирішено використати ансамбль моделей, що поєднує рішення з двох сучасних робіт, рішення ISPL[34] на ICPR 2020 і фіналіста[11] CVPR 2024.

Перше рішення будується на ансамблі невеликий CNN моделей, що базуються на EfficientNet і спеціалізується на виявлення маніпуляцій над нобличчям у відео, тоді як друге рішення представляє набір моделей на основі різних бекбонів, натренованих із імітацією як фізичних, так і цифрових атак. Проаналзуємо початкові результати попередньо навчених моделей на валідаційній вибірці з датасетів (DFD та CelebA-Spoof). На рисунку 3.4 представлено графік метрики якості моделей (AUC) у відсотках проти швидкості їх роботи у мілісекундах на зображення, з використанням GPU.

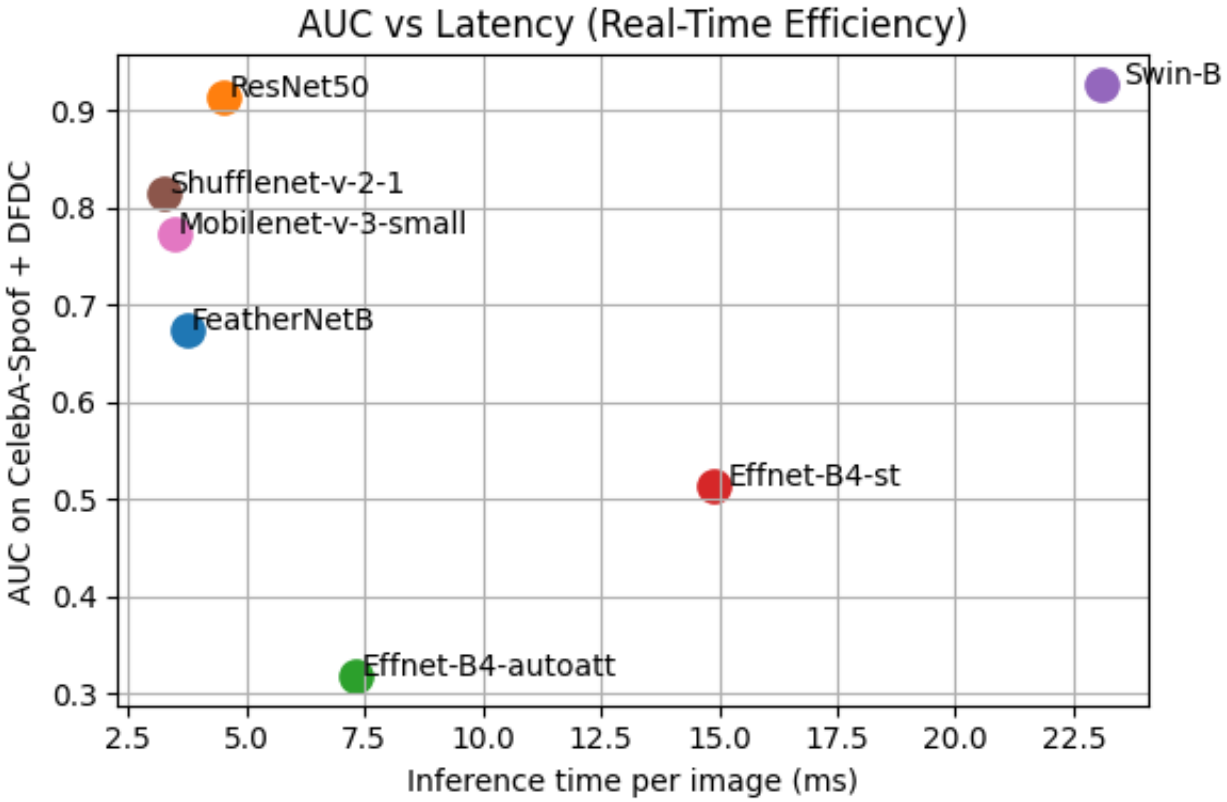


Рисунок 3.4 – Початкові показники моделей

З результатів бачимо, що найкраще себе показують моделі на основі ResNet-50, ShuffleNet V2, Swin-B і MobileNet V3 бекбонів, решта мають значно гірші початкові показники і їх донавчання буде менш вигідним. Щодо часу обробки одного зображення, то з обраних моделей лише Swin має високі показники (понад 22 мс), однак, за умови доступності GPU та паралелізації виконання моделей, значення досі прийнятні для рбооти в реальному часі.

Окремо варто відзначити архітектурні особливості цих моделей, які сприяють їх ефективності:

– Swin Transformer: ця модель використовує механізм "зсунутих вікон" (shifted windows), що дозволяє ефективно обчислювати увагу моделі в локальних областях зображення, зменшуючи залежность обчислювальної складності від розміру зображення до лінійної. Зсув вікон між шарами забезпечує взаємодію між різними областями зображення, що покращує здатність моделі захоплювати глобальний контекст.

– ResNet-50: Ця архітектура впроваджує залишкові з'єднання (skip connections), які дозволяють уникнути проблеми зникнення градієнтів при навчанні глибоких мереж. Основним будівельний блок складається з трьох послідовних згорткових шарів, оптимізованих для зменшення обчислювальних витрат без втрати точності.

– ShuffleNet V2: ця модель орієнтована на високу швидкість обробки, особливо на мобільних та вбудованих пристроях. Вона використовує операції розділення каналів (channel split) та перемішування каналів (channel shuffle), що забезпечує ефективне використання обчислювальних ресурсів при збереженні високої точності.

– MobileNet V3: ця архітектура була розроблена з урахуванням обмежень мобільних пристроїв. Вона поєднує в собі інверсні залишкові блоки (inverted residual blocks), механізми squeeze-and-excitation, а також ефективні активаційні функції, такі як Hard-Swish.

Результати моделей після донавчання можна побачити на рисунку 3.5. Очевидно, що час оброки одного зображення залишився незмінним, втім для усіх моделей показник AUC піднявся: Swin-B + 5%, ShuffleNet + 6%, ResNet-50 +4%, MobileNet +3%. Подальше покращення показників супроводжувалось перенавчанням, тож можна сказати, що моделі досягли плато для цього набору даних. Продовження навчання потребує оптимізації підходів (тонке налаштування гіперпараметрів, дистиляція знань) та покращення датасету, особливо в категоріях 3D масок, високоякісних дипфейків та відео, що показують спражніх людей, однак з великою кількістю постобробки.

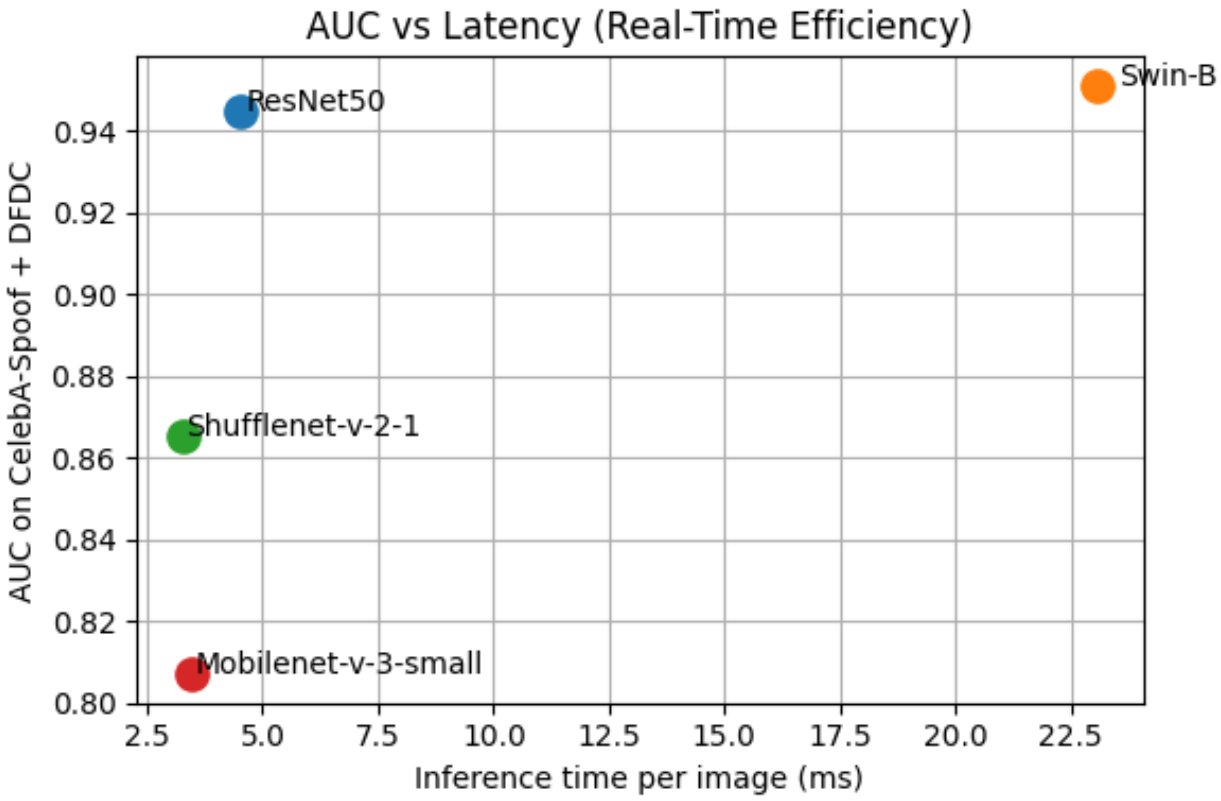


Рисунок 3.5 – Показники моделей після донавчання

Для побудови ансамблю з чотирьох відібраних моделей, окрім їх паралельного виконання, було реалізовано нормалізацію передбачень та подальше усереднення результатів. Щоб врахувати індивідуальну ефективність кожної моделі, було застосовано зважене усереднення, де ваги відповідали значенням AUC на валідаційній вибірці. Такий підхід дозволив підсилити внесок більш точних моделей та зменшити вплив менш надійних без потреби додаткового навчання мета-класифікатора. Покращення результатів иожна побачити розглянувши AUC метрику ансамбля на рисунку 3.6: отримано покращення на близько 0.024 в порівнянні з найкращим результатом окремої моделі (Swin-B). Хоча абсолютні показники невеликі, комбінація моделей з різними архітектурами та підходами до симуляції атак забезпечує гнучкість і підвищену стійкість системи до різних типів загроз. Поріг прийняття рішення про атаку в 0.69 юуло обрано за допомогою графіка на рисунку 3.7.

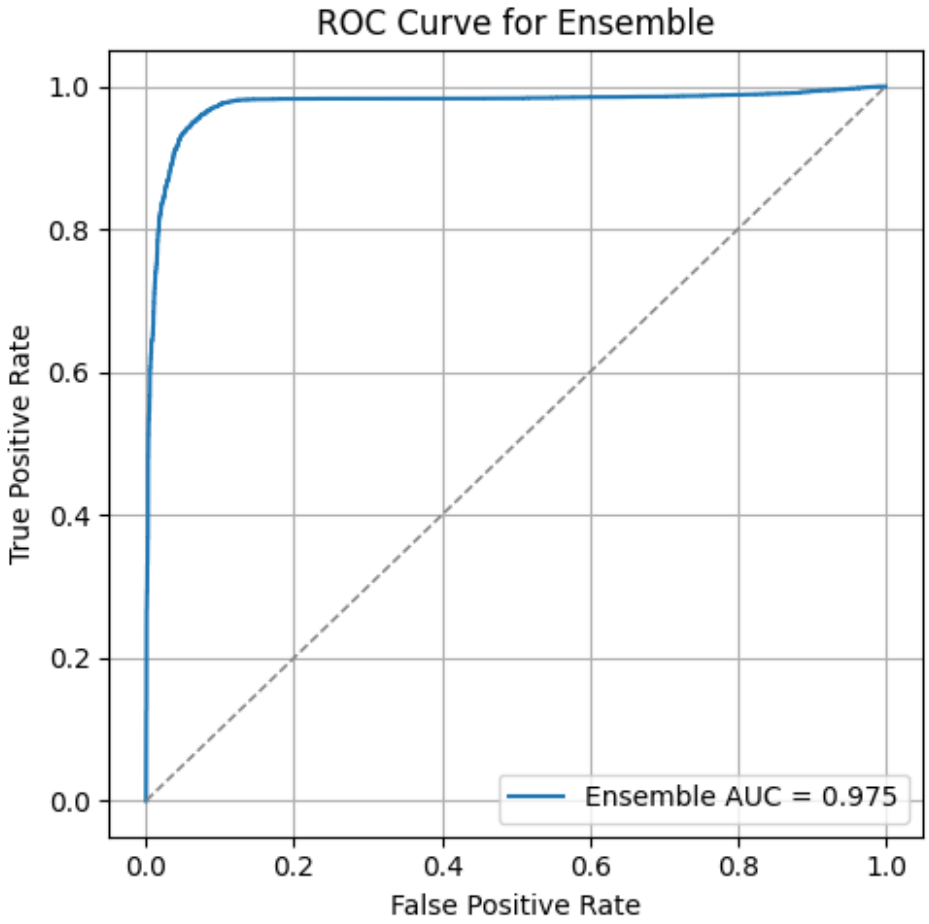


Рисунок 3.6 – ROC ансамблю моделей

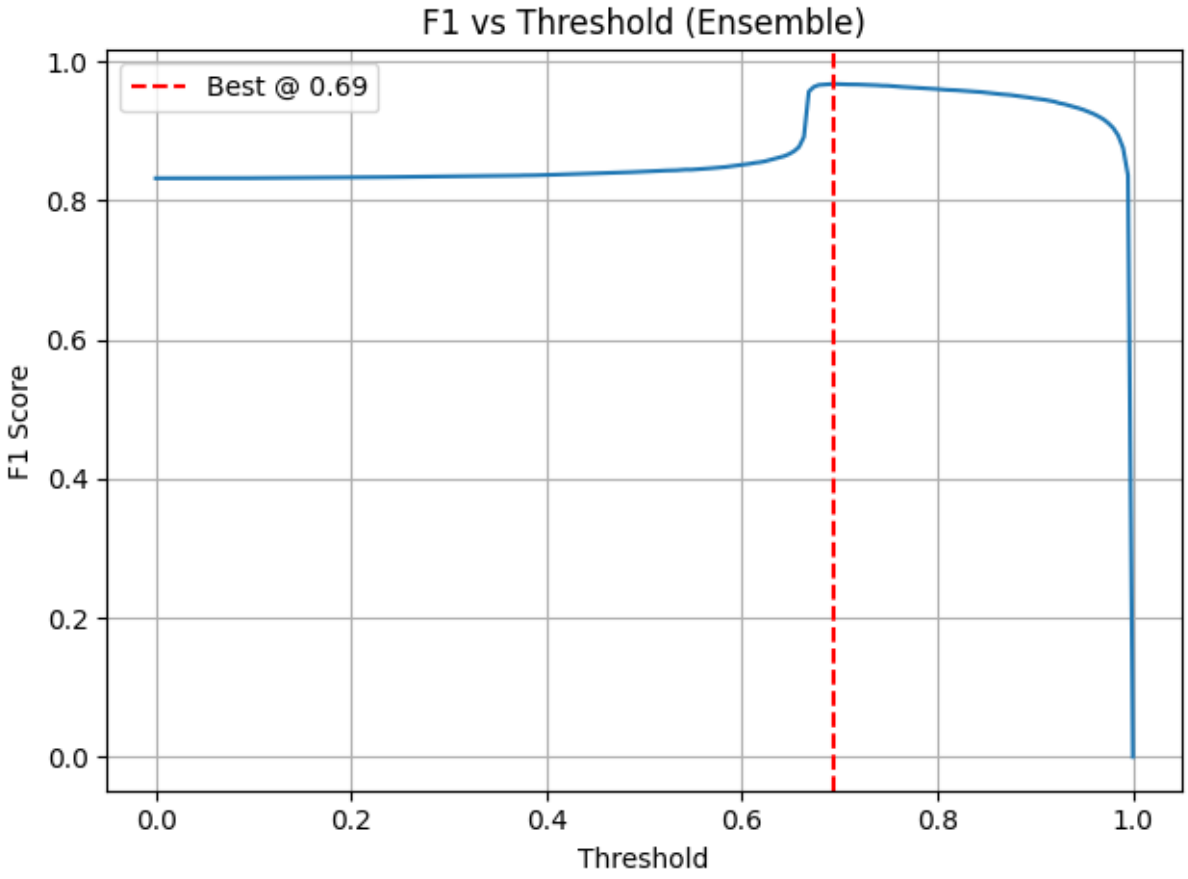


Рисунок 3.7 – Значення метрики F1 проти порога прийняття рішення про недостовірність обличчя

Для побудови повноцінної статистики по відео було реалізовано базовий механізм трекінгу облич, що дозволяє не лише отримувати передбачення для окремих кадрів, а й аналізувати динаміку появи та зникнення конкретних осіб у часі. Трекер порівнює нові виявлені обличчя з раніше баченими за допомогою векторів ознак (embedding) та просторового перекриття (IoU), присвоюючи кожному унікальний ідентифікатор і зберігаючи історію їх поява. Завдяки цьому стає можливим відстеження тривалості перебування осіб у кадрі, виявлення підозрілих змін (наприклад, раптової підміни обличчя), а також створення часової розмітки для подальшого аналізу або візуалізації.

У процесі навчання моделі працювали на рівні окремих зображень, що дозволяло уникнути перенавчання на однотипних прикладах. Використання багатьох схожих кадрів з одного відео призводить до надмірної адаптації моделі до фону, освітлення та положення обличчя, що знижує здатність до узагальнення. Водночас, при роботі з відео у реальному часі можливі різкі зміни сцени, освітлення або пози обличчя, що часто породжує викиди (outliers) як у передбаченнях класу, так і в положенні рамки. Крім того, значні коливання рамки обличчя виникають через рескейлінг та інтерполяцію, які, навіть при мінімальній затримці, впливають на якість візуалізації та порушують стабільність роботи системи. Для усунення цього ефекту було реалізовано згладжування у часі (temporal smoothing) за допомогою ковзного вікна (sliding window), що використовує середнє значення останніх N передбачень з історії трекінгу. Такий підхід було застосовано як до координат рамки обличчя, так і до FAS-оцінок, причому останні також проходять поріг для прийняття рішення. Згладжування здійснюється без зберігання окремих буферів, за рахунок повторного використання історії треку, що мінімізує обчислювальні витрати. Альтернативний підхід – згладжування на рівні верхніх шарів моделей, виявився недоцільним, оскільки у випадку багатьох різних облич та ансамблю моделей це потребувало б збереження окремих

буферів на кожну пару “обличчя-модель”, що використовувало б надмірну відео пам’ять та знижувало ефективність у реальному часі.

3.3.2 Опис структури бази даних захоплених облич

У якості системи управління базами даних у програмному забезпеченні FaceFinder використовується SQLite, оскільки вона є вбудованою СУБД, яка не потребує налаштування окремого сервера та добре підходить для десктопного програмного забезпечення з локальним зберіганням даних. База даних використовується для зберігання облич користувачів у вигляді їх векторних представлень (face embeddings), що дозволяє ефективно виконувати розпізнавання у режимі реального часу. Для кожної особи процес захоплення зберігає до 20 представлень, що забезпечує високу якість розпізнавання. Опис таблиць бази даних наведено у таблицях 3.1-3.2. Модель бази даних наведена на рисунку 3.8.

Таблиця 3.1 – Опис таблиці persons

Назва поля	Тип даних	Опис
uuid	TEXT (PRIMARY KEY)	Унікальний ідентифікатор користувача (обличчя)
name	TEXT	Ім’я або мітка користувача

Таблиця 3.2 – Опис таблиці encodings

Назва поля	Тип даних	Опис
id	INTEGER (PRIMARY KEY AUTOINCREMENT)	Унікальний ідентифікатор запису

Продовження таблиці 3.2

Назва поля	Тип даних	Опис
person_uuid	TEXT	Посилання на користувача у таблиці persons
encoding	BLOB	Закодований вектор обличчя

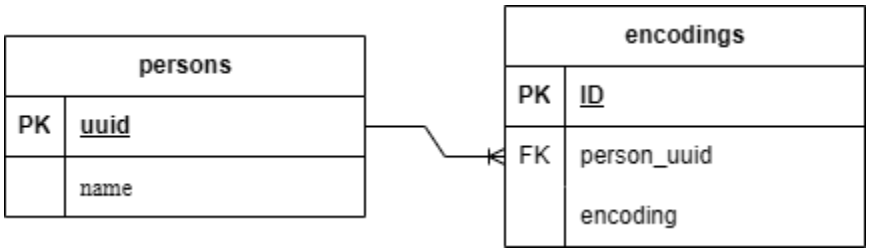


Рисунок 3.8 – ER діаграма сутностей

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.3.

Таблиця 3.3 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Jupyter-notebook	Програмне забезпечення використане для R&D процесів та донавчання нейромереж.
2	CMake	Крос-платформений інструмент для автоматизації процесу збірки програмного забезпечення. Застосувався для компіляціх бібліотек Python з підтримкую GPU.

Продовження таблиці 3.3

№ п/п	Назва утиліти	Опис застосування
3	OpenCV	(Open Source Computer Vision Library) - бібліотека з відкритим вихідним кодом для комп'ютерного зору та машинного навчання, яка використовувалась для отримання відеопотоків, обробки зображень та використання DL.
4	PyQt6	Версія бібліотеки Qt яка дозволяє роботу з Python і призначена для розробки декстопних інтерфейсів.
5	NumPy	Математична бібліотека використана для масивів та обрахунків.
6	PyTorch	Глибокий фреймворк машинного навчання, використовується для реалізації та виконання моделей FAS з підтримкоб GPU.
7	ONNX	Легка крос-платформна бібліотека для виконання моделей, експортованих у форматі ONNX. Застосовується для прискореної інференції моделей обличчя та спуфінгу.
8	SQLite	Вбудована система управління базами даних, що використовується для локального зберігання векторних представлень облич та імен осіб.
9	Visual Studio Code (VS Code)	Легке та швидке середовище для перегляду і редагування коду, з підтрикою багатьох плагінів та мов програмування

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Основні дані, захист яких потрібно забезпечити – біометрична інформація користувачів. Всі обличчя зберігаються локально у вигляді векторних представлень (face embeddings) у базі даних SQLite. Доступ до даних можливий лише з дозволу користувача, тому не застосовуються додаткові шифрування чи механізми авторизації.

Основною вразливістю подібних систем, як зазначено раніше, є можливість PAD і DAD атак. Щоб запобігти цьому, у систему інтегровано модуль FAS, побудований на базі нейронних мережі. Модель працює пасивно, без потреби в активній взаємодії користувача, і дозволяє виявляти більшість типових атак – включаючи зображення на екрані та друковані фото, дипфейки. Це критично для забезпечення надійної ідентифікації в реальному часі.

Система не використовує мережеві з'єднання чи віддалені сервери, тому ризики передачі даних мінімальні. Також не зберігаються відеопотоки чи зображення без вказівки на те користувача.

Таким чином, безпека даних у FaceFinder забезпечується локальним виконанням, обмеженим доступом до бази та вбудованим модулем захисту від штучно змінених облич.

Висновки до розділу

У цьому розділі було виконано повне конструювання та опис архітектури програмного забезпечення «FaceFinder», призначеного для розпізнавання облич з підтримкою FAS. Обрано модульну багаторівневу архітектуру, частково засновану на патерні MVC, з використанням C4-моделі для деталізації структури застосунку.

В рамках архітектурних рішень було обрано локальну обробку відео без зовнішніх з’єднань, що дозволяє зменшити ризики витоку персональних даних. Застосовано патерни Adapter і Decorator для підвищення гнучкості коду. Реалізовано обробку відеопотоку в реальному часі з мінімальною затримкою за допомогою OpenCV і паралелізації.

Для вирішення задач розпізнавання облич і антиспуфінгу використано сучасні бібліотеки та моделі: YuNet і RetinaFace – для детекції облич, Dlib з MMOD і ResNet – для отримання закодованих характеристик обличчя, а також ансамблі моделей на базі EfficientNet, Xception, ResNet50 та інших для реалізації FAS.

Було проаналізовано засоби розробки та обрано стек технологій, до якого увійшли Python, PyTorch, ONNX, OpenCV, PyQt6, face_recognition, NumPy, а також середовище VS Code і платформа Kaggle для навчання моделей.

Загалом, реалізована система є ефективним, локальним і безпечним рішенням, придатним для використання в задачах реального часу та з можливістю розширення.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метриками для оцінки якості ПЗ обрано наступні:

– цикломатична складність – характеризує складність коду, кількість незалежних шляхів, що потребують тестування. Обчислюється за допомогою утиліти radon і дозволяє виявити потенційно складні для супроводу ділянки коду.

– підтримуваність (Maintainaility) – показник зручності модифікації та супроводу коду, враховує довжину функцій, кількість операторів, вкладеність тощо. Обчислюється за допомогою утиліти radon

– оцінка якості коду за pylint – метрика від 0 до 10, що відображає дотримання рекомедованого стилю коду, відсутність типових помилок і дублювання коду, коректність структури, іменування змінних і документації. Значення понад 8 вказує на добру якість і відповідність PEP8.

– точність виявлення облич (Detection Accuracy) – визначає здатність системи правильно виявляти обличчя у відеопотоці. Обчислюється як відсоток правильно виявлених облич серед очікуваних.

– точність розпізнавання атак (FAS Accuracy) – показник ефективності модулів антиспуфінгу, що вимірює правильність класифікації справжніх та штучно змінених облич на валідаційних відеоданих.

– продуктивність (FPS) – кількість оброблених кадрів за секунду, що демонструє здатність системи працювати в реальному часі.

Статичний аналіз коду на за допомогою pylint дав початкову оцінку 8.86, показавши нтзьку проблем з докоментуванням методів чи знанадто складними функціями. Після виправлення більшості зауважень, оцінка піднялась до 9.71, втім решта проблем була пов’язана з тим, що аналізатор хибно визначав залежності деяких динамічних бібліотек (PyQt, OpenCV). У випадку

ігнорування хибних зауважень, матимемо ідеальне значення показника, що свідчить про високу якість коду і відповідність PEP8.

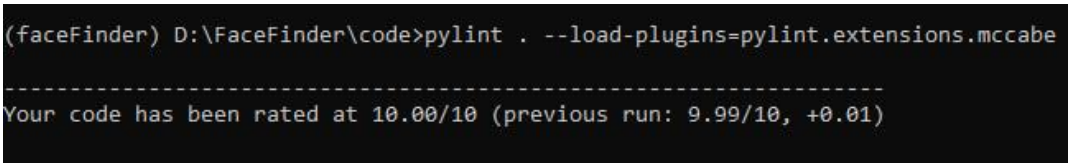


Рисунок 4.1 – Результат оцінки pylint

Оскільки тестування за допомогою radon відбувалось після основного масиву правок, воно не допомогло виявити нові недоліки, лише вказало на хорошу підтримуваність (оцінка A для всіх файлів проекту) та низьку цикломатичну складність у 2.58.

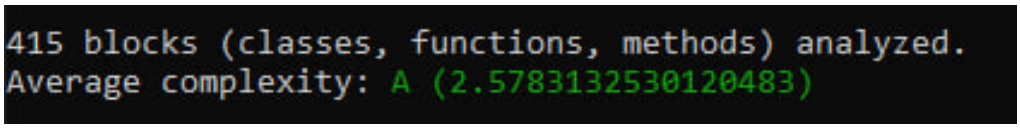


Рисунок 4.2 – Результат оцінки цикломатичної складності основних блоків програми за допомогою radon

Для тестування точності виявлення облич та розпізнавання атак було проведено оцінку програми на підготовленому відеодатасеті. Результати наведені у таблиці 4.1.

Таблиця 4.1 – Тестування точності

Детектор облич	FAS моделі	Точність виявлення облич	Точність розпізнавання атак	Ефективна точність
RetinaFace	ResNet-50, ShuffleNetV2, Swin-B, MobileNet V3	0.956	0.931	0.89
YuNet	ResNet-50, ShuffleNetV2, Swin-B, MobileNet V3	0.769	0.947	0.728

Продовження таблиці 4.1

Детектор облич	FAS моделі	Точність виявлення облич	Точність розпізнавання атак	Ефективна точність
RetinaFace	Swin-B	0.956	0.853	0.815
RetinaFace	ResNet-50	0.956	0.821	0.785
YuNet	ResNet-50	0.769	0.889	0.684

Оцінивши результати, бачимо, що YuNet справляється на порядок гірше і часто може не визначити обличчя. Це має побічний ефект: точність розпізнавання атак з YuNet як детектором зростає порівняно з RetinaFace, але це зумовлено тим, що обличчя у найскладніших умовах часто не були виявлені взагалі. Для комплексної оцінки системи, що враховує одночасно точність виявлення та FAS, було введено ефективну точність як додаткову метрику, яка демонструє частку правильно виявлених та розпізнаних облич. Загалом також спостерігається помітне падіння метрик FAS у порівнянні з інференсом на зображеннях. Причинами цього є зміна домену, нові для моделей ракурси облич та умови освітлення, а також той факт, що зображення використовувались як валідаційна вибірка, тоді як відео були повністю новими для моделі. Хоча найкращі ваги для зображень демонстрували також найкращі результати на відео, не можна виключати можливість перенавчання моделі на специфічні характеристики валідаційної вибірки. Ансамбль моделей продемонстрував менше падіння продуктивності порівняно з окремими моделями, що дозволяє зробити висновок про його вищу надійність та стійкість до змін середовища.

Тестування швидкодії проводилось незалежно від оцінки точності і полягало у збиранні статистики середньої кількості кадрів за секунду (FPS) на добірці відео з різною кількістю осіб у кадрі. Результати наведені у таблиці 4.2.

Таблиця 4.2 – Тестування швидкодії

Детектор облич	FAS моделі	Пристрій для інференсу FAS	Кількість кадрів на секнду
RetinaFace	ResNet-50, ShuffleNetV2, Swin-B, MobileNet V3	GPU	10
RetinaFace	Swin-B	GPU	14
RetinaFace	ResNet-50	GPU	22
YuNet	ResNet-50	CPU	5

Результати показують, що при використанні повного ансамблю моделей система балансує на межі допустимої продуктивності для роботи в реальному часі (приблизно 10 FPS). Найбільший вплив на зниження швидкодії має саме модель Swin-B, яка є найповільнішою серед використаних. Окремо варто відзначити відсутність практичної доцільності до інференсу без використання GPU: навіть при роботі лише з однією FAS-моделлю (ResNet-50) та детектором YuNet на CPU, система не досягає продуктивності, необхідної для реального часу.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.3 – 4.11.

Таблиця 4.3 – Тест 1.1 Захоплення обличчя з коректною міткою

Початковий стан системи	Користувач перебуває на головному екрані, система має хоча б одну камеру.
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає Capture new face, вводить коректну мітку, обличчя потрапляє в кадр, виконується 20 захоплень

Продовження таблиці 4.3

Очікуваний результат	Обличчя зберігається у датасеті, з’являється повідомлення про успішне захоплення
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.2 Введення некоректної мітки

Початковий стан системи	Користувач перебуває на екрані захоплення облич після натискання Capture new face
Вхідні дані	Мітка !@invalid%%
Опис проведення тесту	Користувач вводить некоректну мітку
Очікуваний результат	З’являється повідомлення про помилку, мітка не приймається
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.3 Перегляд міток

Початковий стан системи	Користувач перебуває на головному екрані, у системі збережено щонайменше одну мітку
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає Manage labels
Очікуваний результат	Відображається список усіх збережених міток
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.4 Перейменування мітки

Початковий стан системи	Користувач у вікні керування мітками, вибрана мітка user_01
Вхідні дані	Нова мітка new_user_2
Опис проведення тесту	Користувач натискає кнопку перейменування, вводить нову мітку
Очікуваний результат	Назва мітки оновлюється в інтерфейсі й у базі
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.4 Перейменування мітки

Початковий стан системи	Користувач у вікні керування мітками, вибрана мітка user_01
Вхідні дані	Нова мітка new_user_2
Опис проведення тесту	Користувач натискає кнопку перейменування, вводить нову мітку
Очікуваний результат	Назва мітки оновлюється в інтерфейсі й у базі
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.5 Видалення мітки

Початковий стан системи	Користувач у вікні Manage labels, є принаймні одна мітка
Вхідні дані	Відсутні

Продовження Таблиці 4.7

Опис проведення тесту	Користувач обирає мітку та натискає Delete
Очікуваний результат	Мітка видаляється з інтерфейсу та бази даних
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.6 Розпізнавання облич на відео

Початковий стан системи	Користувач перебуває на головному екрані, у системі є збережені мітки
Вхідні дані	Відеофайл із присутніми раніше захопленими обличчями
Опис проведення тесту	Користувач натискає Start Recognition, обирає у ділогому вікні файл як джерело, обирає потрібний файл
Очікуваний результат	Відображається оброблене відео з правильно розпізнаними особами, навколо їх обличя малюється зелена рамка, над якою присутній підпис мітки
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.7 Виявлення цифрових атак

Початковий стан системи	Користувач перебуває на головному екрані, у системі є збережені мітки
Вхідні дані	Відеофайл із дипфейком згенерованим на основі раніше захопленого обличчя
Опис проведення тесту	Користувач натискає Start Recognition, обирає у ділогому вікні файл як джерело, обирає потрібний файл

Продовження таблиці 4.9

Очікуваний результат	Відображається оброблене відео, замість мітки над особою є надпис про несправдність, рамка навколо обличчя має близький до червоного колір
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.8 Розпізнавання з вебкамери

Початковий стан системи	Користувач перебуває на головному екрані, у системі є збережені мітка його обличчя та доступна камера
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає Start Recognition, обирає у ділогому вікні камеру як джерело, обирає потрібну камеру
Очікуваний результат	Відображається оброблене відео з обраної камери, над особою є її мітка, рамка навколо обличчя має зелений колір
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.9 Виявлення фізичних атак

Початковий стан системи	Користувач перебуває на головному екрані, у системі є збережені мітка його обличчя та доступна камера
Вхідні дані	Сторонній пристрій з екраном (телефон) на якому є зображення користувача
Опис проведення тесту	Користувач натискає Start Recognition, обирає у ділогому вікні камеру як джерело, обирає потрібну камеру. Після цього він демонструє поряд із своїм обличчям своє зображення з телефону

Продовження таблиці 4.11

Початковий стан системи	Користувач перебуває на головному екрані, у системі є збережені мітка його обличчя та доступна камера
Вхідні дані	Сторонній пристрій з екраном (телефон) на якому є зображення користувача
Опис проведення тесту	Користувач натискає Start Recognition, обирає у ділогому вікні камеру як джерело, обирає потрібну камеру. Після цього він демонструє поряд із своїм обличчям своє зображення з телефону

4.3 Опис контрольного прикладу

Контрольний приклад демонструє повний цикл роботи системи від захоплення обличчя користувача до виявлення фізичної атаки презентації (демонстрації зображення обличчя з екрана телефону). Послідовність кроків, варіанти розгалужень та реакція інтерфейсу описані нижче, ілюстрації подано як рисунки.

Після запуску програми користувач потрапляє на головний екран, де представлено три основні дії: Manage labels, Capture new face та Start recognition.

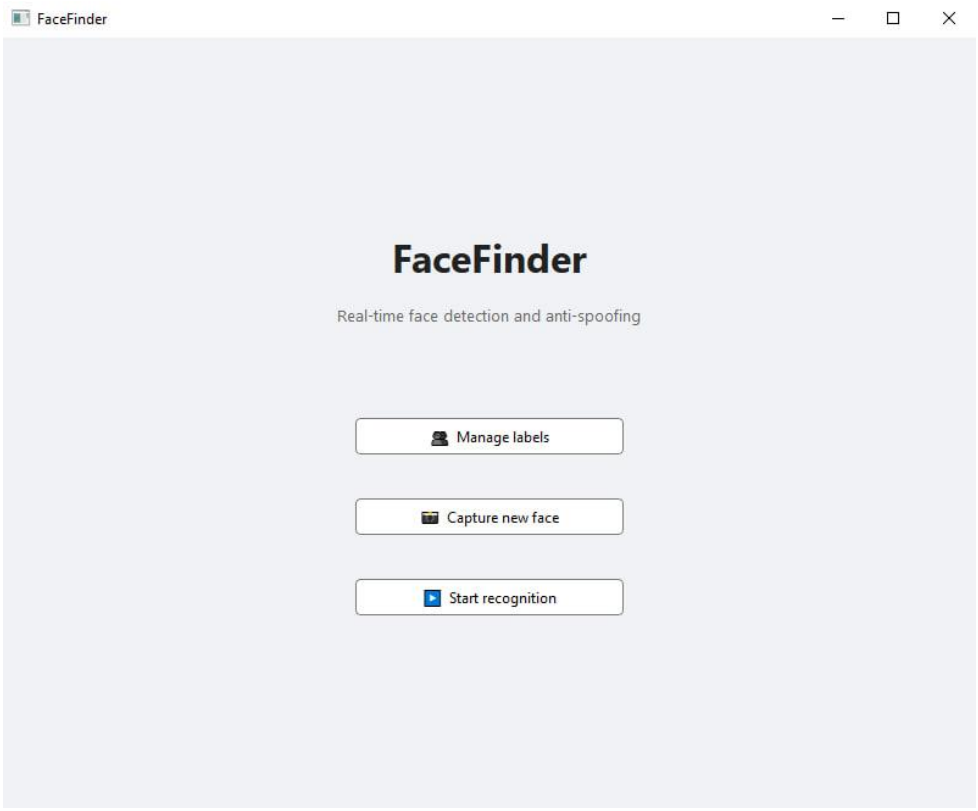


Рисунок 4.3 – Головний екран застосунку

Спочатку виконується захоплення нового обличчя – для цього натискається кнопка Capture new face, після чого з’являється діалогове вікно для вибору камери та введення мітки.

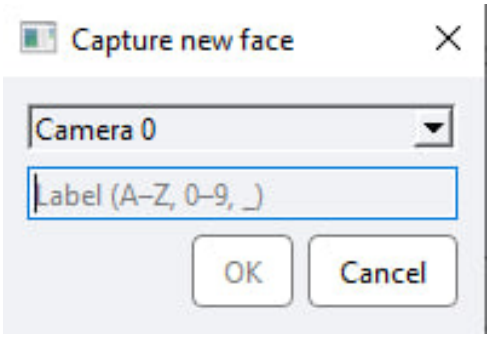


Рисунок 4.4 – Діалогове вікно перед захопленням обличчя

Після натискання OK відкривається вікно захоплення, де користувач бачить себе з зеленою рамкою навколо обличчя. Щоб виконати захоплення, потрібно натиснути Capture, захоплюючи зображення обличчя під різними ракурсами, до 20 кадрів. Процес можна завершити достроково, натиснувши кнопку Done.

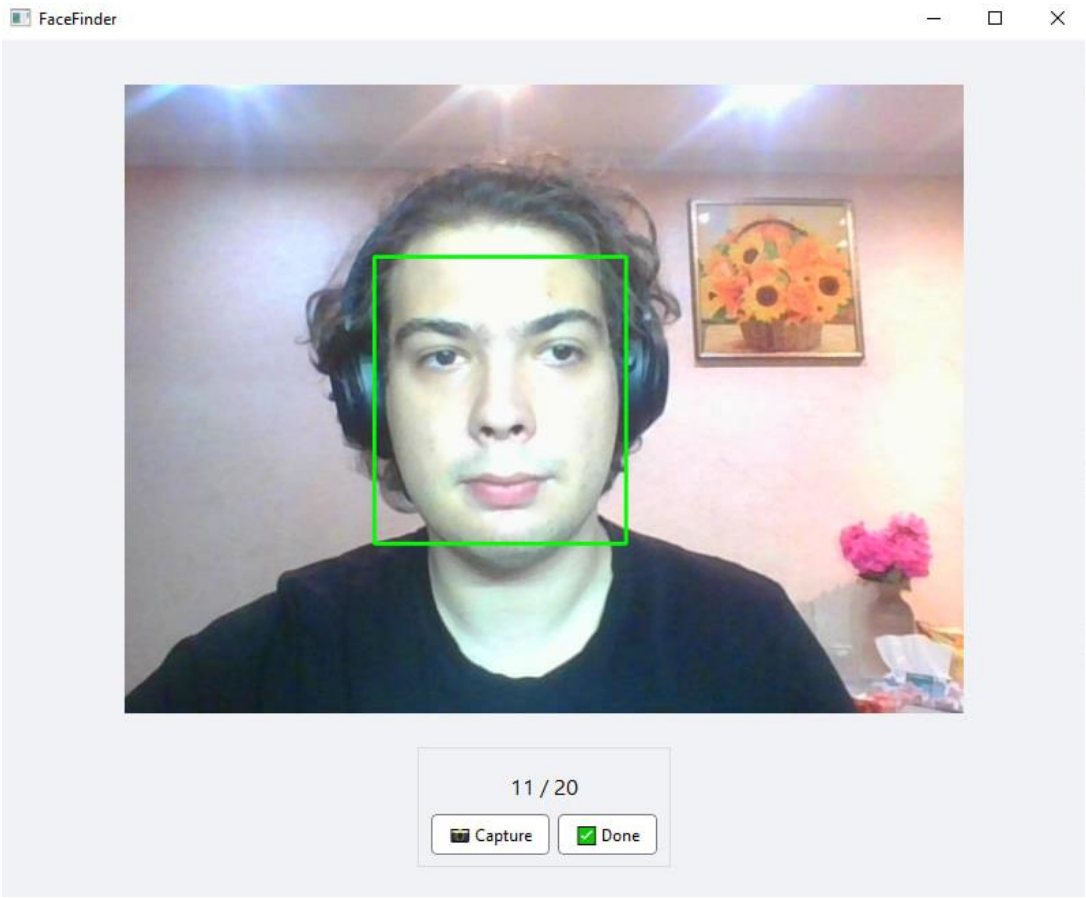


Рисунок 4.5 – Вікно захоплення обличчя

Після завершення захоплення користувач переходить до розділу Manage labels, натиснувши відповідну кнопку на головному екрані. Тут відображається список міток, зокрема щойно додана мітка. При потребі мітку можна перейменувати або видалити за відповідних кнопок. Таким чином можна переконатися, що обличчя збережено успішно.

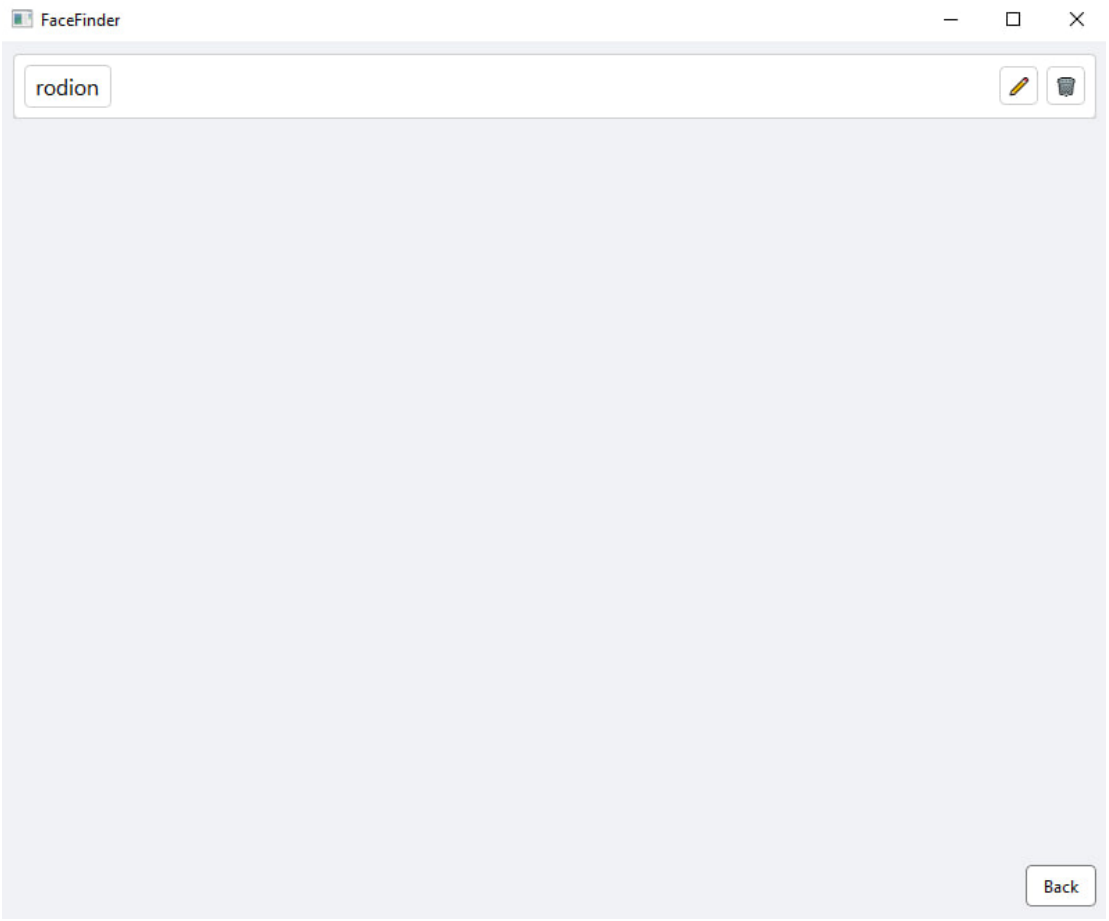


Рисунок 4.6 – Вікно менеджера міток

Далі користувач повертається на головний екран і натискає Start recognition. Відкривається діалогове вікно, в якому можна обрати джерело відео: файл або камеру. У цьому прикладі обирається камера. Після підтвердження відкривається вікно розпізнавання в реальному часі. Система виявляє обличчя користувача, виводить його мітку (rodion) над головою, обрамляє його зеленою рамкою та показує рівень впевненості.

Для симуляції атаки користувач підносить до камери телефон із зображенням свого обличчя. Система аналізує кадр і визначає, що це фізична атака презентації. Замість звичайної мітки над цим зображенням з’являється напис SPOOF, а рамка стає червоною. Таким чином, система коректно відрізняє реальне обличчя від зображення на екрані телефону.

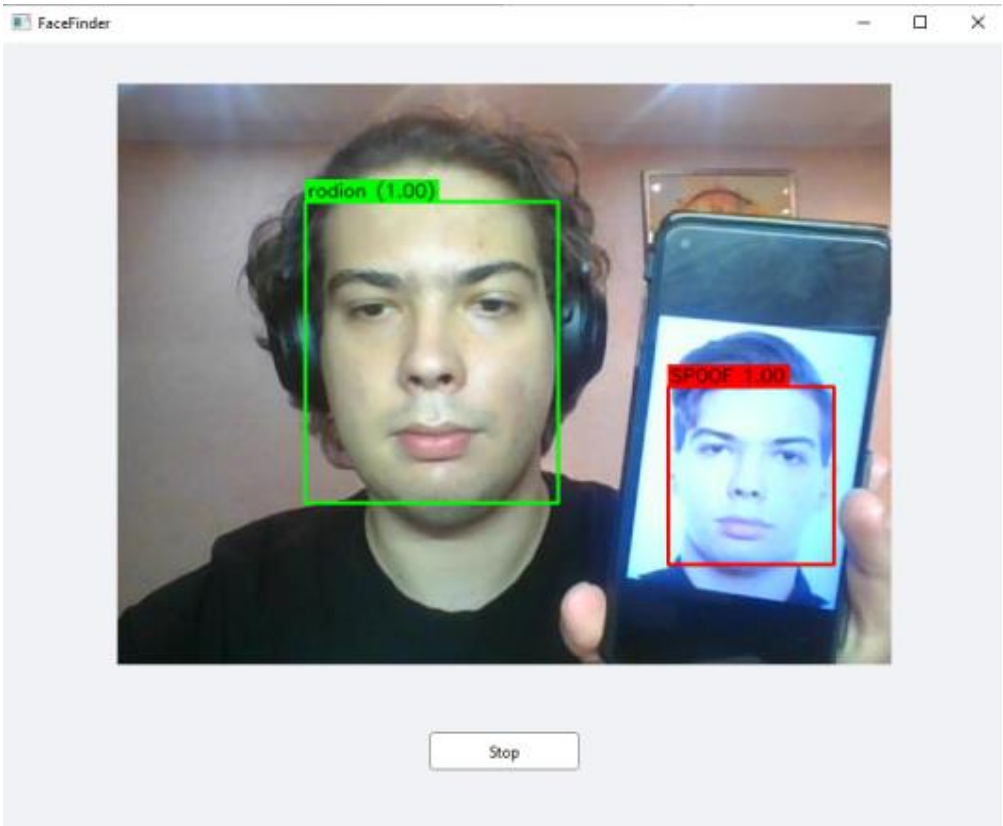


Рисунок 4.7 – Результат роботи системи на відеопотоці з камери

Цей контрольний приклад демонструє повний робочий цикл застосунку: від додавання нового користувача до надійного виявлення спроби обману системи. Він підтверджує функціональність основних компонентів: інтерфейсу, модулів захоплення, збереження, розпізнавання та FAS.

Висновки до розділу

У цьому розділі було виконано повний аналіз якості програмного забезпечення для системи розпізнавання облич з антиспуфінгом, а також проведено тестування функціональних і нефункціональних характеристик розробленого застосунку.

Було визначено основні метрики для оцінки якості ПЗ, серед яких: цикломатична складність, індекс підтримуваності, pylint-оцінка якості коду, точність виявлення облич, точність антиспуфінгу (FAS) та продуктивність у реальному часі (FPS). Статичний аналіз показав високий рівень якості коду: остаточна оцінка за pylint становила 9.71, а утиліта radon підтвердила низьку складність та високу підтримуваність усіх основних модулів (рівень A). Це

свідчить про хорошу структуру програмного коду та відповідність сучасним стандартам програмування.

У межах функціонального тестування було проведено перевірку точності системи на підготовленому відеодатасеті. Результати показали, що найкращі показники досягаються при використанні детектора RetinaFace у поєднанні з ансамблем FAS-моделей. Була також введена додаткова метрика ефективна точність, яка враховує як правильне виявлення облич, так і коректне розпізнавання атак. Виявлено, що ансамбль моделей демонструє вищу стійкість до змін у вхідних даних та менше падіння якості при переході з валідаційного на тестовий домен.

Тестування швидкодії показало, що повноцінна робота в реальному часі можлива лише за умови використання GPU. Використання CPU для інференсу FAS-моделей, навіть у спрощеній конфігурації, виявилося малоефективним для цільових сценаріїв. Найбільший вплив на продуктивність має модель Swin-B, яка хоча й демонструє високу точність, сповільнює обробку.

Було також проведено мануальне тестування, яке охоплює всі ключові сценарії: захоплення, перегляд, перейменування та видалення міток; запуск розпізнавання з камери або відео; виявлення фізичних та цифрових атак. Для опису основного сценарію було сформовано контрольний приклад, який демонструє повний цикл взаємодії з системою: від реєстрації нового користувача до успішного виявлення спуфінг-атаки на основі фото.

Усі тести завершились успішно, що свідчить про працездатність системи, відповідність функціональним вимогам та здатність роботи в реальному часі за наявності відповідних апаратних ресурсів. Зроблені висновки підтверджують доцільність застосування системи в практичних сценаріях виявлення особи та захисту від атак презентації.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Програма була скомпільована в .exe файл з усіми необхідними бібліотеками. Передбачено встановлення за допомогою msi файлу, для чого були використанні плагін auto-py-to-exe (який фактично є графічним інтерфейсом для бібліотеки Pyinstaller) для перетворення скрипта в .exe файл та Advanced Installer[23] для створення .msi файлу. Також у процесі довелось вирішити неочевидні конфлікти між бібліотеками, які не виникали при запуску коду як скрипта.

Встановлення за допомогою інсталятора має переваги у можливості оновлення та видалення стандартними засобами операційної системи. Для встановлення потрібно слідувати інструкціям інсталятора:

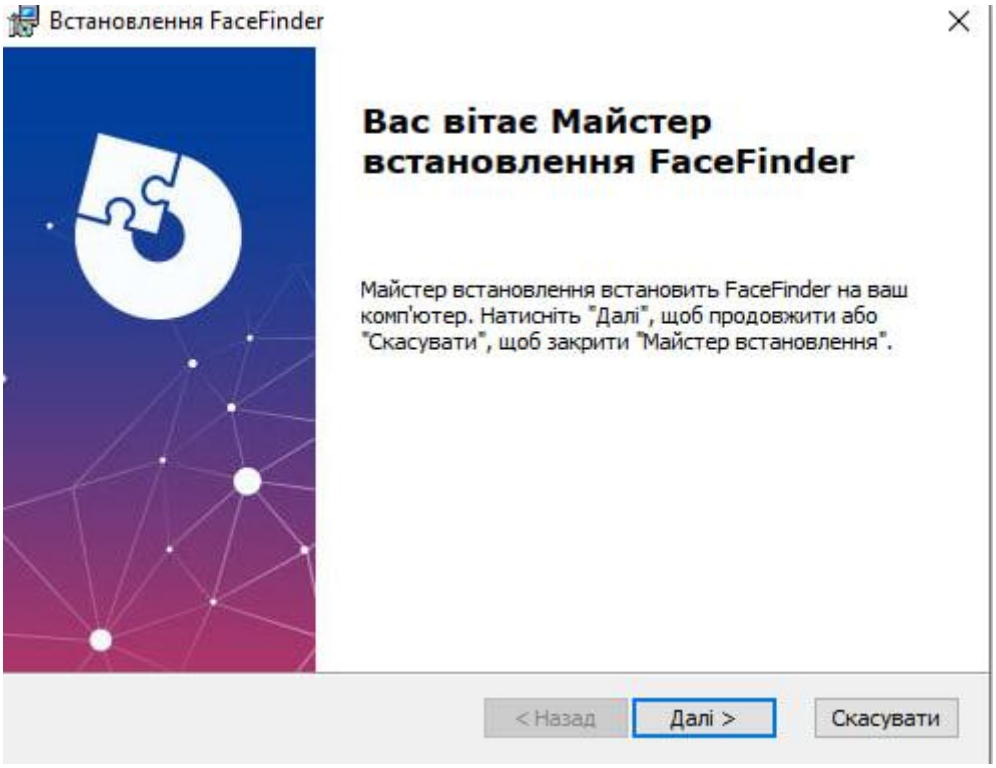


Рисунок 5.1 – перший етап інсталяції

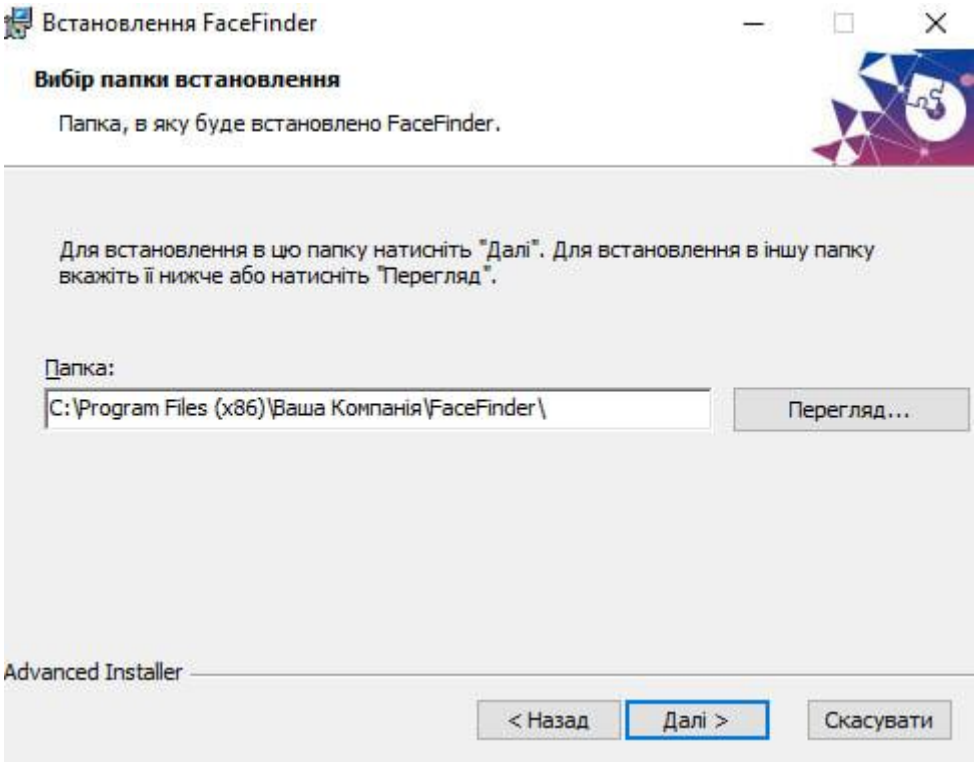


Рисунок 5.2 – другий етап інсталяції

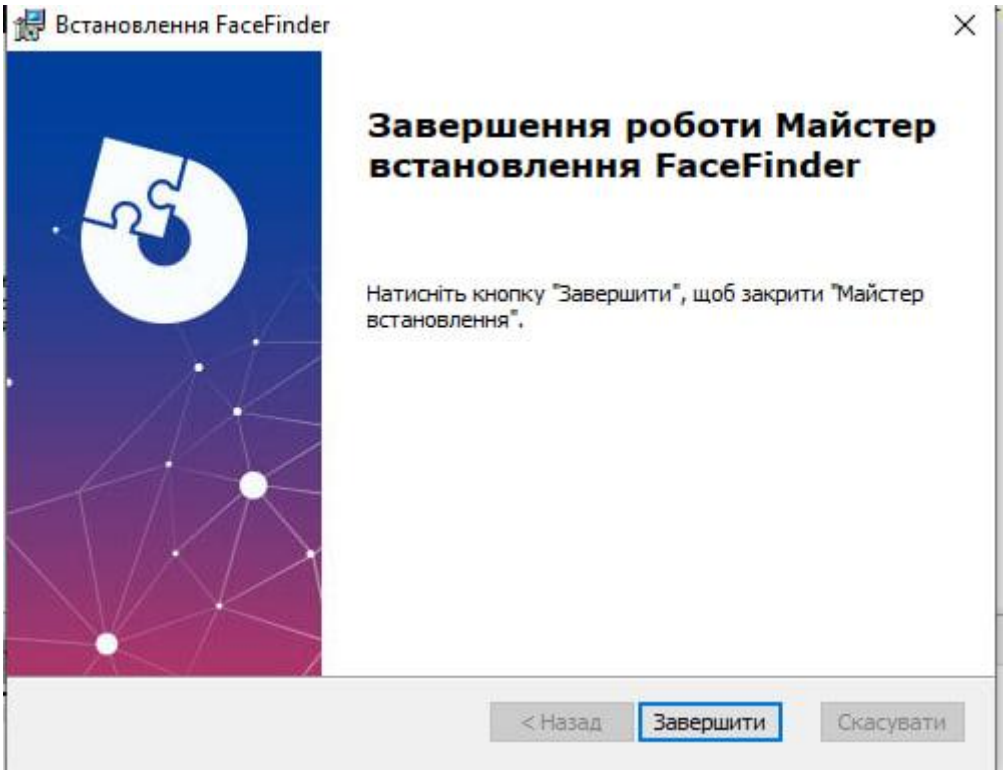


Рисунок 5.3 – фінальний етап інсталяції

Отож, для розгортання ПЗ потрібно виконати такі кроки:

– Встановити додаток за допомогою інсталятора

– Переконатись, що файли вагів моделй знаходяться у root папці разом з .exe файлом та розмістити їх туди за необхідності

– встановити CUDA Toolkit 12.1 + версії для прискорення роботи за допомогою відеокарти користуючись інструкціями Nvidia

– Виконати тестовий запуск ПЗ

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача».

Користувачі повинні мати можливість отримати нову версію застосунку з кожною версією що виходить. Для цього достатньо скачати і запустити інсталятор. Слідуючи його інструкціям, буде завантажено усі потрібні дані.

Висновки до розділу

У розділі описано процес впровадження програмного забезпечення, створення .exe-версії та інсталяційного пакета у форматі .msi. Для цього було використано інструменти auto-py-to-exe та Advanced Installer. У процесі компіляції були вирішені технічні конфлікти між бібліотеками.

Готове ПЗ може бути встановлено за допомогою стандартного інсталятора, що спрощує розгортання, оновлення та видалення програми. Для запуску системи користувачу потрібно: встановити застосунок, переконатися в наявності файлів ваг моделей у кореневій директорії, за потреби встановити CUDA Toolkit.

Оновлення реалізовано через повторне встановлення нової версії інсталятора. Інструкція користувача надана окремим документом. Загалом система підготовлена до використання кінцевими користувачами без необхідності роботи з вихідним кодом.

ВИСНОВКИ

У ході виконання дипломного проєкту було розроблено програмне забезпечення для розпізнавання облич у відеопотоці з підтримкою антиспуфінгу (FAS), що поєднує модулі детекції облич, розпізнавання, виявлення атак, трекінгу виявлених облич, згаладжування у часі та зручний графічний інтерфейс.

Основна мета проєкту це створення системи, здатної працювати в реальному часі та ефективно розпізнавати особу з перевіркою на справжність, і вона була досягнута. Реалізоване рішення забезпечує стабільну роботу з продуктивністю 10-20 кадрів на секунду залежно від конфігурації, демонструючи ефективну точність: близько 80% при високій частоті кадрів (близько 20) та до 90% при 10 FPS.

Це стало можливим завдяки реалізації таких функцій:

– використання ансамблю моделей для FAS, що підвищило стійкість до атак;

– підтримка зчитування облич з відео та вебкамери;

– впровадження згладжування у часі для стабілізації результатів;

– трекінг облич для підтримки послідовності виявлених осіб між кадрами;

– локальне збереження закодованих векторів облич у базі SQLite;

– користувацький інтерфейс на базі PyQt6.

Усі завдання, поставлені в рамках дипломного проєкту, включаючи проєктування, реалізацію, тестування, оптимізацію та підготовку інсталятора, були виконані на достатньому рівні, хоча залишається простір для розвитку.

Результати проєкту можуть бути застосовані в системах верифікації особи, верифікації справжньості відео, контролю доступу, відеоспостереження та безпеки. Доцільним є подальший розвиток системи у таких напрямах:

– покращення якості моделей виявлення облич та антиспуфінгу;

– оптимізація методів ансамблювання моделей з урахуванням точності та швидкодії;

– підвищення стійкості до складних умов, зокрема змінного освітлення та часткових перекриттів;

– оптимізація рботи системи для осіб які знаходяться на різній віддалі від камери

– розширення трекінгу і ведення статистики

Отримане програмне забезпечення є надійною базою для подальших досліджень, вдосконалення та практичного впровадження в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 2024 update on dhs’s use of face recognition & face capture technologies | homeland security. *U.S. Department of Homeland Security.* URL: https://www.dhs.gov/archive/news/2025/01/16/2024-update-dhss-use-face-recognition-face-capture-technologies (дата звернення: 13.05.2025).

2. Liporazzi B. How this recruiter realized she was interviewing an AI deepfake. *HR Brew*. URL: https://www.hr-brew.com/stories/2025/03/31/recruiter-interview-ai-deepfake (дата звернення: 13.05.2025).

3. Deep learning for face anti-spoofing: a survey / Z. Yu та ін. *IEEE transactions on pattern analysis and machine intelligence.* 2022. URL: https://doi.org/10.1109/tpami.2022.3215850 (дата звернення: 13.05.2025).

4. Face recognition pipeline clearly explained. *Medium.* URL: https://medium.com/backprop-labs/face-recognition-pipeline-clearly-explained-f57fc0082750 (дата звернення: 13.05.2025).

5. Face anti-spoofing techniques for liveness detection in security systems. *MobiDev.* URL: https://mobidev.biz/blog/face-anti-spoofing-prevent-fake-biometric-detection (дата звернення: 13.05.2025).

6. Kamat C. Face anti-spoofing methods: a comparative analysis through the lens of a comprehensive review. *International journal for research in applied science and engineering technology.* 2024. Т. 12, № 2. С. 514–526. URL: https://doi.org/10.22214/ijraset.2024.58383 (дата звернення: 13.05.2025).

7. Taming self-supervised learning for presentation attack detection: de-folding and de-mixing / Z. Kong та ін. *IEEE transactions on neural networks and learning systems.* 2023. С. 1–12. URL: https://doi.org/10.1109/tnnls.2023.3243229 (дата звернення: 13.06.2025).

8. Middle-shallow feature aggregation in multimodality for face anti-spoofing / C. Li та ін. *Scientific reports.* 2023. Т. 13, № 1. URL: https://doi.org/10.1038/s41598-023-36636-w (дата звернення: 13.05.2025).

9. Günay Yılmaz A., Turhal U., Nabiyev V. Face presentation attack detection performances of facial regions with multi-block LBP features. *Multimedia*

tools and applications. 2023. URL: https://doi.org/10.1007/s11042-023-14453-7 (дата звернення: 13.05.2025).

10. PatchNet / S.-M. Hu та ін. *ACM transactions on graphics.* 2013. Т. 32, № 6. С. 1–12. URL: https://doi.org/10.1145/2508363.2508381 (дата звернення: 13.05.2025).

11. Joint Physical-Digital Facial Attack Detection Via Simulating Spoofing Clues. / He X. та ін. *arXiv.org.* URL: https://arxiv.org/html/2404.08450v1 (дата звернення: 13.05.2025).

12. A survey on deep learning-based face anti-spoofing / P.-K. Huang та ін. *APSIPA transactions on signal and information processing.* 2024. Т. 13, № 1. URL: https://doi.org/10.1561/116.20240053 (дата звернення: 13.05.2025).

13. Solomon E., Cios K. J. FASS: face anti-spoofing system using image quality features and deep learning. *Electronics.* 2023. Т. 12, № 10. С. 2199. URL: https://doi.org/10.3390/electronics12102199 (дата звернення: 13.05.2025).

14. Birla L., Gupta P., Kumar S. SUNRISE: improving 3D mask face anti-spoofing for short videos using pre-emptive split and merge. *IEEE transactions on dependable and secure computing.* 2022. С. 1. URL: https://doi.org/10.1109/tdsc.2022.3168345 (дата звернення: 13.06.2025).

15. Kantarcı A., Dertli H., Ekenel H. K. Deep patch-wise supervision for presentation attack detection. *IET biometrics.* 2022. URL: https://doi.org/10.1049/bme2.12091 (дата звернення: 13.05.2025).

16. Learning deep models for face anti-spoofing: binary or auxiliary supervision. *IEEE Xplore.* URL: https://ieeexplore.ieee.org/document/8578146 (дата звернення: 13.06.2025).

17. FaceTec achieves exponential Q1 revenue growth and usage. *PRWeb.* URL: https://www.prweb.com/releases/facetec-achieves-exponential-q1-revenue-growth-and-usage-881269855.html (дата звернення: 13.06.2025).

18. Crowd-Sourced Biometric Security Testing Explained. SpoofBounty.com URL: https://www.spoofbounty.com/ (дата звернення: 13.05.2025).

19. Perdana R. N., Ardiyanto I., Nugroho H. A. A review on face anti-spoofing. *IJITEE (international journal of information technology and electrical engineering).* 2021. URL: https://doi.org/10.22146/ijitee.61827 (дата звернення: 13.05.2025).

20. Singh A. K., Joshi P., Nandi G. C. Face liveness detection through face structure analysis. *International journal of applied pattern recognition.* 2014. Т. 1, № 4. URL: https://doi.org/10.1504/ijapr.2014.068327 (дата звернення: 13.05.2025).

21. Visual Prompt Flexible-Modal Face Anti-Spoofing / Z. Yu та ін. *IEEE transactions on dependable and secure computing.* 2024. С. 1–10. URL: https://doi.org/10.1109/tdsc.2024.3520534 (дата звернення: 13.06.2025).

22. Jin Y., Lu J., Ruan Q. Coupled discriminative feature learning for heterogeneous face recognition. *IEEE transactions on information forensics and security.* 2015. Т. 10, № 3. С. 640–652. URL: https://doi.org/10.1109/tifs.2015.2390414 (дата звернення: 13.05.2025).

23. Stailey-Young A. What’s the best face detector?. *Medium.* URL: https://medium.com/pythons-gurus/what-is-the-best-face-detector-ab650d8c1225 (дата звернення: 13.05.2025).

24. RetinaFace: single-stage dense face localisation in the wild. arXiv.org. URL: https://arxiv.org/abs/1905.00641 (дата звернення: 13.05.2025).

25. Wu W., Peng H., Yu S. YuNet: a tiny millisecond-level face detector. *Machine intelligence research.* 2023. URL: https://doi.org/10.1007/s11633-023-1423-y (дата звернення: 13.05.2025).

26. Labeled faces in the wild for face recognition. *Sefik Ilkin Serengil.* URL: https://sefiks.com/2020/08/27/labeled-faces-in-the-wild-for-face-recognition/ (дата звернення: 13.05.2025).

27. FaceForensics++ benchmark (deepfake detection). *Papers With Code*. URL: https://paperswithcode.com/sota/deepfake-detection-on-faceforensics-1 (дата звернення: 13.06.2025).

28. SiW-Enroll5 benchmark (face anti-spoofing). *Papers With Code.* URL: https://paperswithcode.com/sota/face-anti-spoofing-on-siw-enroll5 (дата звернення: 13.06.2025).

29. Surantha N. Lightweight face recognition-based portable attendance system with liveness detection. *ScienceDirect.* URL: https://doi.org/10.1016/j.iot.2024.101089 (дата звернення: 13.05.2025).

30. Deep fake detection (DFD) entire original dataset. *Kaggle.* URL: https://www.kaggle.com/datasets/sanikatiwarekar/deep-fake-detection-dfd-entire-original-dataset. (дата звернення: 13.05.2025).

31. King D. Max-Margin object detection. *arXiv.org.* URL: https://arxiv.org/abs/1502.00046 (date of access: 13.06.2025).

32. Face Recognition documentation. *Face Recognition*. URL: https://face-recognition.readthedocs.io/en/latest/face_recognition.html#module-face_recognition.api. (дата звернення: 13.05.2025).

33. Dlib documentation. *dlib C++ Library.* URL: https://dlib.net/python/index.html#dlib_pybind11.face_recognition_model_v1 (дата звернення: 13.05.2025).

34. Video face manipulation detection through ensemble of cnns. *arXiv.org.* URL: https://arxiv.org/abs/2004.07676 (дата звернення: 13.06.2025).

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

StrikePlagiarism.com

Дата звіту 6/9/2025
Дата редагування 6/9/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Скорик_ПЗ
Автор Науковий керівник / Експерт
ІП-12_СкорикПоперешняк С.В.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

13.12%
13.12%
КП 1

10	10424	78533
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

ДОДАТОК Б ДИПЛОМ ПЕРЕМОЖЦЯ ВСЕУКРАЇНСЬКОГО КОНКУРСУ НАУКОВИХ ДОСЛІДЖЕНЬ СТУДЕНТІВ “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ВИРОБНИЦТВІ”

ДОДАТОК В АКТ ВПРОВАДЖЕННЯ

ЗАТВЕРДЖУЮ

Директор Виробничо-торгівельної

приватної фірми «АН»

Приступа І.Ф.

"10" січня 2024 р.

А К Т

про впровадження результатів наукових досліджень Скорика Родіона Олеговича, одержаних під час виконання наукової роботи «Система розпізнавання обличчя людини у відеопотоці»

Комісія у складі:

голови комісії – директора Виробничо-торгівельної приватної фірми «АН» Приступи І.Ф.,

та членів комісії: завідувач відділу безпеки Нижнього Д.Б., завідувача виробництвом Полякова В.С.

встановила, що наукові положення, розроблені особисто Скориком Р.О. реалізовані на підприємстві наступним чином:

Користувачами даної автоматизованої системи стали працівники відділу безпеки, а також до системи було внесено дані тестової вибірки співробітників, зокрема робився акцент на персоналі, який має доступ до приміщень із підвищеними вимогами до контролю та безпеки. Дана система інформувала користувачів про санкціонований і несанкціонований доступ до приміщень, ідентифікувала працівників на трансляціях відеокамер та їх записах, ідентифікувала спроби використати несправжні обличчя (макети, демонстрація фотографій та відео на пристроях). Завдяки впровадженню системи було покращено системи спостереження, що покращило ефективність та частково автоматизувало роботу відділу безпеки. Також система мала вплив на спостереження за дотриманням техніки безпеки та оцінку ефективності роботи та відвідуваності окремих працівників.

Впровадження даної системи допомогло оптимізувати низку процесів на підприємсві, зокрема підвищення рівня безпеки на території підприємства та спрощення контролю за керуванням персоналом. Також вона допомогла автоматизувати контроль відвідуваності та доступу, результатом чого стало зменшення витрат часу на управління персоналом.

Голова комісії:
директор Виробничо-торгівельної
приватної фірми «АН» Приступа І.Ф.

Члени комісії: завідувач відділу безпеки Нижній Д.Б.

завідувач виробництва Поляков В.С.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧ У ВІДЕОПОТОЦІ З ДЕТЕКЦІЄЮ ШТУЧНО ЗМІНЕНИХ ЗОБРАЖЕНЬ

Текст програми

КПІ.ІП-1224.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Світлана ПОПЕРЕШНЯК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Родіон СКОРИК

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/SkorykRodion/FaceFinder.git

Файл models.face_detectors.base.py
Реалізація функціональної задачі виявлення обличчя.

```
"""
Base module for face detector interfaces.

Defines the abstract interface for face detectors that provide bounding
boxes.
"""

from abc import ABC, abstractmethod
import numpy as np


class IDetector(ABC):
    """
    Abstract base class for face detectors.

    Provides an interface for detectors that return bounding boxes for faces
in an image.
    """

    @abstractmethod
    def get_face_locations(self, image: np.ndarray) -> np.ndarray:
        """
        Detects faces in the given image and returns their bounding boxes.

        Args:
            image (np.ndarray): Input image as a NumPy array.

        Returns:
            np.ndarray: Array of bounding boxes, each represented as (top,
right, bottom, left).
        """
```

Файл models.face_detectors.retina_detector.py
Реалізація функціональної задачі виявлення обличчя.

```
"""
RetinaFaceDetector module powered by InsightFace (SCRFD/RetinaFace).
Provides detection of faces and optional landmark extraction.
"""

from typing import Iterable
import numpy as np
from insightface.app import FaceAnalysis

from .base import IDetector


class RetinaFaceDetector(IDetector):
    """
    Face detector implementation using InsightFace (SCRFD/RetinaFace).

    Attributes:
        threshold (float): Detection confidence threshold.
        app (FaceAnalysis): InsightFace FaceAnalysis instance for detection.
    """
```

```
    def __init__(
        self,
        ctx_id: int = 0,
        det_size: tuple[int, int] = (640, 640),
        threshold: float = 0.5
    ) -> None:
        """
        Initialize RetinaFaceDetector.

        Args:
            ctx_id (int): Context ID for device selection (0 = first CUDA
GPU, -1 = CPU).
            det_size (Tuple[int, int]): Detection input size (width, height).
            threshold (float): Detection confidence threshold.
        """
        self.threshold = float(threshold)
        self.app = FaceAnalysis(allowed_modules=["detection"])
        self.app.prepare(ctx_id=ctx_id, det_size=det_size)

    def get_face_locations(self, image: np.ndarray) -> np.ndarray:
        """
        Detect faces and return bounding boxes.

        Args:
            image (np.ndarray): Input image in BGR format.

        Returns:
            np.ndarray: Array of bounding boxes in (top, right, bottom, left)
format.
        """
        faces = self.app.get(image)
        boxes = []
        for f in faces:
            if f.det_score < self.threshold:
                continue
            x1, y1, x2, y2 = f.bbox.astype(int)
            boxes.append((y1, x2, y2, x1))
        return np.asarray(boxes, dtype=int)

    def get_top_prediction(self, image: np.ndarray) -> np.ndarray:
        """
        Get the highest scoring face bounding box.

        Args:
            image (np.ndarray): Input image in BGR format.

        Returns:
            np.ndarray: Bounding box of the top prediction, or empty array if
none found.
        """
        faces = self.app.get(image)
        if not faces:
            return np.array([])
        best = max(faces, key=lambda f: f.det_score)
        if best.det_score < self.threshold:
            return np.array([])
        x1, y1, x2, y2 = best.bbox.astype(int)
        return np.array((y1, x2, y2, x1))

    def get_face_locations_with_landmarks(
        self, image: np.ndarray
    ) -> Iterable[tuple[tuple[int, int, int, int], np.ndarray]]:
        """
```

```
        Detect faces and return bounding boxes with landmarks.

        Args:
            image (np.ndarray): Input image in BGR format.

        Returns:
            Iterable[Tuple[Tuple[int, int, int, int], np.ndarray]]:
                Iterable of (bounding box, landmarks) tuples.
        """
        faces = self.app.get(image)
        out = []
        for f in faces:
            if f.det_score < self.threshold:
                continue
            x1, y1, x2, y2 = f.bbox.astype(int)
            box = (y1, x2, y2, x1)
            out.append((box, f.kps.astype(int)))
        return out
```

Файл models.face_detectors.yu_detector.py

Реалізація функціональної задачі виявлення обличчя.

```
"""
Face detection module using YuNet (ONNX) via OpenCV's FaceDetectorYN API.
Provides detection of faces and optional landmark extraction.
"""

from typing import Optional, Iterable
import cv2
import numpy as np

from .base import IDetector


class YuFaceDetector(IDetector):
    """
    Face detector using YuNet (ONNX) via OpenCV's FaceDetectorYN API.

    Supports face detection with optional landmark extraction.
    """

    def __init__(
        self,
        model_path: str =
"./model_weights/face_detection_yunet_2023mar.onnx",
        threshold: float = 0.001,
    ) -> None:
        """
        Initialize the YuFaceDetector.

        Args:
            model_path: Path to the YuNet ONNX model.
            threshold: Confidence threshold to filter detections.
        """
        self.detector: cv2.FaceDetectorYN = cv2.FaceDetectorYN.create(
            model_path, "", (320, 320)
        )
        self.input_size: tuple[int, int] = (320, 320)
        self.threshold: float = threshold
        self.image_size: Optional[tuple[int, int]] = None
        self.detections: Optional[np.ndarray] = None

    def set_input_size(self, width: int, height: int) -> None:
        """
        Update the model input size to match the given image size.
```

```
        Args:
            width: Image width.
            height: Image height.
        """
        self.image_size = (width, height)
        self.detector.setInputSize(self.image_size)

    def detect(self, image: np.ndarray) -> tuple[np.ndarray, float, float]:
        """
        Run face detection on a resized image and store scaled results.

        Args:
            image: Original input image.

        Returns:
            A tuple containing the resized image and scale factors (fx, fy).
        """
        original_height, original_width = image.shape[:2]
        target_width, target_height = self.input_size
        resized_image = cv2.resize(image, self.input_size)

        _, self.detections = self.detector.detect(resized_image)

        scale_x = original_width / target_width
        scale_y = original_height / target_height
        return resized_image, scale_x, scale_y

    def get_face_locations(self, image: np.ndarray) -> np.ndarray:
        """
        Get all face bounding boxes in the image, scaled to original
resolution.

        Args:
            image: Input image.

        Returns:
            Array of (top, right, bottom, left) tuples.
        """
        _, scale_x, scale_y = self.detect(image)
        face_locations: list[tuple[int, int, int, int]] = []

        if self.detections is not None:
            for face in self.detections:
                x, y, w, h, confidence = face[:5]
                if confidence > self.threshold:
                    top = int(y * scale_y)
                    right = int((x + w) * scale_x)
                    bottom = int((y + h) * scale_y)
                    left = int(x * scale_x)
                    face_locations.append((top, right, bottom, left))

        return np.array(face_locations)

    def get_top_prediction(self, image: np.ndarray) -> np.ndarray:
        """
        Get the most confident face detection in the image.

        Args:
            image: Input image.

        Returns:
            (top, right, bottom, left) tuple of the most confident detection,
            or empty array if none passes the threshold.
```

```
        """
        self.detect(image)

        if self.detections is None or len(self.detections) == 0:
            return np.array([])

        top_face = max(self.detections, key=lambda f: f[4])
        x, y, w, h, confidence = top_face[:5]

        if confidence < self.threshold:
            return np.array([])

        return np.array((int(y), int(x + w), int(y + h), int(x)))

    def get_face_locations_with_landmarks(
        self, image: np.ndarray
    ) -> Iterable[tuple[tuple[int, int, int, int], np.ndarray]]:
        """
        Get bounding boxes and landmarks for all detected faces.

        Args:
            image: Input image.

        Returns:
            Iterable of tuples, each containing:
                - a tuple (top, right, bottom, left) representing the
bounding box
                - a (5, 2) NumPy array of landmark coordinates: left eye,
right eye,
                  nose tip, left mouth corner, right mouth corner
        """
        self.detect(image)
        results: list[tuple[tuple[int, int, int, int], np.ndarray]] = []

        if self.detections is not None:
            for face in self.detections:
                x, y, w, h, confidence = face[:5]
                if confidence > self.threshold:
                    box = (int(y), int(x + w), int(y + h), int(x))
                    landmarks = face[5:15].reshape((5, 2)).astype(int)
                    results.append((box, landmarks))

        return results
```

Файл models.face_encoders.base.py

Реалізація функціональної задачі захоплення та розпізнавання обличчя кодування.

```
"""
Base module for face encoders.

Defines the IEncoder interface for encoding faces from images.
"""

from abc import ABC, abstractmethod
from typing import Optional, Tuple
import numpy as np


class IEncoder(ABC):
    """
    Interface for face encoders.

    Implementations should provide a method to encode a face from an image,
```

```
    given the face location.
    """

    @abstractmethod
    def encode(
        self, image: np.ndarray, face_location: Tuple[int, int, int, int]
    ) -> Optional[np.ndarray]:
        """
        Encode a face from the given image and face location.

        Args:
            image (np.ndarray): The input image as a NumPy array.
            face_location (Tuple[int, int, int, int]): The bounding box of
the face (top, right, bottom, left).

        Returns:
            Optional[np.ndarray]: The encoded face as a NumPy array, or None
if encoding fails.
        """
```

Файл models.face_encoders.dlib_encoder.py

Реалізація функціональної задачі захоплення та розпізнавання обличчя.

```
"""
This module provides a Dlib-based face encoder implementation using the
face_recognition library.
"""

from typing import Optional
import numpy as np
import face_recognition

from .base import IEncoder

class FaceEncoder(IEncoder):
    """
    Face encoder that uses the face_recognition library to extract face
embeddings from images.
    """

    def encode(
        self,
        image: np.ndarray,
        face_location: tuple[int, int, int, int]
    ) -> Optional[np.ndarray]:
        """
        Encode a face in the given image at the specified location.

        Args:
            image (np.ndarray): The input image as a NumPy array.
            face_location (tuple[int, int, int, int]): The bounding box of
the face (top, right, bottom, left).

        Returns:
            Optional[np.ndarray]: The face encoding as a NumPy array, or None
if no encoding is found.
        """
        encodings = face_recognition.face_encodings(image, [face_location])
        return encodings[0] if encodings else None
```

Файл inference_managers.capture_manager.py

Реалізація функціональної задачі захоплення обличчя.

```
from typing import Optional, Iterable
```

```
import numpy as np

from inputs.video.base import BaseVideoStream
from inputs.video.threaded import ThreadedVideoStream
from models.face_detectors.yu_detector import YuFaceDetector
from models.face_encoders.dlib_encoder import FaceEncoder
from storage.face_repository import FaceEncodingRepository

from . import base


class CaptureManager(base.IManager):
    """
    Manages video capture and face encoding collection.

    This class yields frames from a video stream along with face detection
results,
    collects face encodings upon request, and saves them to a repository.

    Attributes:
        stream (BaseVideoStream): The video stream source.
        detector: The face detector instance.
        encoder: The face encoder instance.
        max_samples (int): Maximum number of face encodings to collect.
        _buf (list): Buffer to store collected face encodings.
        _want_capture (bool): Flag to indicate if a capture is requested.
    """

    def __init__(
        self,
        cam_idx: int,
        max_samples: int,
        detector=None,
        encoder=None,
        stream: Optional[BaseVideoStream] = None,
    ):
        """
        Initialize the CaptureManager.

        Args:
            cam_idx (int): Camera index for video capture.
            max_samples (int): Maximum number of samples to collect.
            detector: Optional face detector instance.
            encoder: Optional face encoder instance.
            stream (Optional[BaseVideoStream]): Optional video stream
instance.
        """
        self.stream = stream or ThreadedVideoStream(source=cam_idx)
        self.detector = detector or YuFaceDetector()
        self.encoder = encoder or FaceEncoder()
        self.max_samples = max_samples
        self._buf = []
        self._want_capture = False

    def iter_run(self) -> Iterable[tuple[np.ndarray, dict]]:
        """
        Iterate over video frames, detect faces, and collect encodings.

        Yields:
            tuple: (frame_bgr, info_dict) where info_dict contains:
                - "faces": list of detected face bounding boxes as dicts
                - "count": number of collected samples
                - "max": maximum samples to collect
        """
```

```
            while True:
                ok, frame = self.stream.get_frame()
                if not ok:
                    break
                faces = []
                boxes = self.detector.get_face_locations(frame)
                if boxes.size:
                    bbox = boxes[0]
                    if self._want_capture:
                        enc = self.encoder.encode(frame, bbox)
                        if enc is not None:
                            self._buf.append(enc)
                        self._want_capture = False
                    faces.append(self._to_dict(bbox))

                yield frame, {
                    "faces": faces,
                    "count": len(self._buf),
                    "max": self.max_samples,
                }

                if len(self._buf) >= self.max_samples:
                    break

    @staticmethod
    def _to_dict(bbox: np.ndarray) -> dict:
        """
        Convert a bounding box array to a dictionary.

        Args:
            bbox (np.ndarray): Bounding box array.

        Returns:
            dict: Bounding box as a dictionary.
        """
        top, right, bottom, left = map(int, bbox.tolist())
        return {"bbox": (top, right, bottom, left)}

    def request_capture(self) -> None:
        """
        Request to capture the next detected face encoding.
        """
        self._want_capture = True

    def save(self, repo: FaceEncodingRepository, label: str) -> int:
        """
        Save collected face encodings to a repository.

        Args:
            repo: Repository instance with add_person(label, encodings)
method.
            label (str): Label for the person.

        Returns:
            int: Number of encodings saved.
        """
        if self._buf:
            repo.add_person(label, self._buf)
        return len(self._buf)

    def close(self) -> None:
        """
        Close the video stream.
        """
```

```
        self.stream.close()
```

Файл inference_managers\pipeline_manager.py

Реалізація функціональної задачі розпізнавання, вмявлення обличчя та штучних змін.

```
import time
from typing import Optional, Generator

import cv2
import torch

from pipeline.memory.data_handler import DataHandler
from pipeline.memory.face_instance import FaceInstance
from pipeline.runner import PipelineRunner
from pipeline.smoothing import ISmoother
from inputs.video.base import BaseVideoStream
import utils as global_utils


class PipelineManager(global_utils.configurable.IConfigurable):
    """
    Manages the face detection and recognition pipeline, including video
stream handling,
    face memory management, smoothing, and result serialization.

    Attributes:
        stream (Optional[BaseVideoStream]): The video stream source.
        pipeline (PipelineRunner): The main pipeline runner for detection and
recognition.
        memory (DataHandler): Handles face memory and state.
        fas_smoother (Optional[ISmoother]): Smoother for anti-spoofing
scores.
        bbox_smoother (Optional[ISmoother]): Smoother for bounding boxes.
    """

    def __init__(
        self,
        stream: Optional[BaseVideoStream],
        pipeline: PipelineRunner,
        memory: DataHandler,
        fas_smoother: Optional[ISmoother] = None,
        bbox_smoother: Optional[ISmoother] = None,
    ):
        """
        Initialize the PipelineManager.

        Args:
            stream (Optional[BaseVideoStream]): The video stream source.
            pipeline (PipelineRunner): The main pipeline runner.
            memory (DataHandler): Face memory handler.
            fas_smoother (Optional[ISmoother]): Smoother for anti-spoofing
scores.
            bbox_smoother (Optional[ISmoother]): Smoother for bounding boxes.
        """
        self.stream = stream
        self.pipeline = pipeline
        self.memory = memory
        self.fas_smoother = fas_smoother
        self.bbox_smoother = bbox_smoother

    def set_stream(self, stream: BaseVideoStream) -> None:
        """
        Set the video stream source.
```

```
        Args:
            stream (BaseVideoStream): The video stream to use.
        """
        self.stream = stream

    def iter_run(self) -> Generator[tuple, None, None]:
        """
        Generator that yields processed frames and face detection results.

        Yields:
            tuple: (frame (np.ndarray), outs (list[dict])) where each dict
contains
                    bbox, identity, spoof_score, and is_spoof.
        """
        if self.stream is None:
            raise RuntimeError(
                "Stream is not initialized. Use `set_stream()` before calling
run()."
            )

        while True:
            ok, frame = self.stream.get_frame()
            if not ok:
                break

            faces = self.memory.init_frame(frame)
            faces = self.pipeline.run(frame, faces)
            if self.fas_smoother:
                self.fas_smoother.smooth(faces, self.memory)
            if self.bbox_smoother:
                self.bbox_smoother.smooth(faces, self.memory)
            self.memory.update(faces)
            torch.cuda.reset_peak_memory_stats()
            torch.cuda.empty_cache()
            yield frame, [self._to_dict(f) for f in faces]

        self.close()

    def _to_dict(self, f: FaceInstance) -> dict:
        """
        Serialize a FaceInstance to a dictionary for GUI consumption.

        Args:
            f (FaceInstance): The face instance to serialize.

        Returns:
            dict: Serialized face data.
        """
        bbox = f.bbox_smoothed if f.bbox_smoothed is not None else f.bbox
        score = (
            f.spoof_score_smoothed
            if f.spoof_score_smoothed is not None
            else f.spoof_score
        )
        return {
            "bbox": bbox,
            "identity": getattr(f, "identity", None),
            "spoof_score": score if score is not None else 0.0,
            "is_spoof": getattr(f, "is_spoof", False),
        }

    def close(self) -> None:
        """
```

```
        Close the video stream.
        """
        self.stream.close()

    def run(self) -> None:
        """
        Run the pipeline in a loop, displaying results with FPS overlay.
        """
        if self.stream is None:
            raise RuntimeError(
                "Stream is not initialized. Use `set_stream()` before calling
run()."
            )

        prev_time = time.time()

        while True:
            ok, frame = self.stream.get_frame()
            if not ok:
                break

            faces = self.memory.init_frame(frame)
            faces = self.pipeline.run(frame, faces)
            self.memory.update(faces)
            if self.fas_smoother:
                self.fas_smoother.smooth(faces, self.memory)
            if self.bbox_smoother:
                self.bbox_smoother.smooth(faces, self.memory)

            output = global_utils.face_drawer.draw_raw_faces(frame.copy(),
faces)

            # FPS Calculation
            current_time = time.time()
            fps = 1.0 / (current_time - prev_time)
            prev_time = current_time

            cv2.putText(
                output,
                f"FPS: {fps:.2f}",
                (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX,
                1.0,
                (255, 255, 255),
                2,
                cv2.LINE_AA,
            )

            torch.cuda.reset_peak_memory_stats()
            torch.cuda.empty_cache()

            cv2.imshow("Face Inference", output)
            if cv2.waitKey(1) & 0xFF == ord("q"):
                break

        self.stream.close()
        cv2.destroyAllWindows()

    @classmethod
    def from_config(cls, cfg) -> "PipelineManager":
        """
        Instantiate PipelineManager from a configuration object.

        Args:
```

```
            cfg: Configuration object.

        Returns:
            PipelineManager: Configured pipeline manager instance.
        """
        memory = DataHandler()
        pipe = PipelineRunner.from_config(cfg.pipeline)
        bbox_sm = global_utils.config.instantiate(cfg.smoothers.get("bbox"))
        fas_sm = global_utils.config.instantiate(cfg.smoothers.get("fas"))
        return cls(
            None,
            pipe,
            memory,
            bbox_smoother=bbox_sm,
            fas_smoother=fas_sm,
        )
```

Файл executable_objects.cpp

Реалізація функціональної задачі отримання відеопотоку.

```
from threading import Thread
import time
from typing import Optional
import cv2

from .base import BaseVideoStream


class ThreadedVideoStream(BaseVideoStream):
    """
    A threaded video stream class that reads frames from a video source in a
background thread.

    Attributes:
        status (bool): Status of the last frame read.
        frame (Optional[any]): The most recent frame read from the video
source.
        sleep_time (float): Time to sleep between frame reads to match the
source FPS.
        thread (Thread): The background thread reading frames.
    """

    def __init__(
        self,
        max_width: Optional[int] = 640,
        max_height: Optional[int] = 480,
        source: Optional[str] = None,
    ) -> None:
        """
        Initialize the ThreadedVideoStream.

        Args:
            max_width (Optional[int]): Maximum width of the frame.
            max_height (Optional[int]): Maximum height of the frame.
            source (Optional[str]): Path or identifier for the video source.
        """
        super().__init__(max_width, max_height, source)
        self.status: bool = False
        self.frame: Optional[any] = None
        raw_fps: float = self.capture.get(cv2.CAP_PROP_FPS) or 30.0
        capped_fps: float = min(raw_fps, 30.0)
        self.sleep_time: float = 1.0 / capped_fps if capped_fps > 1.0 else
0.033
        self.thread: Thread = Thread(target=self._update, daemon=True)
        self.thread.start()
```

```
        time.sleep(1)  # Allow buffer fill

    def _update(self) -> None:
        """
        Continuously read frames from the video source in a background
thread.
        """
        while self.capture.isOpened():
            self.status, self.frame = self.capture.read()
            time.sleep(self.sleep_time)

    def get_frame(self) -> tuple[bool, Optional[any]]:
        """
        Get the latest frame from the video stream.

        Returns:
            Tuple[bool, Optional[any]]: A tuple containing the status and the
resized frame.
        """
        if self.frame is None:
            return False, None
        return self.status, self.resize_frame(self.frame)
```

Файл models.fas.wrappers.mobilenet.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
MobileNetFAS wrapper module.
"""

from pathlib import Path
from typing import Union
import torch

from .base import IFASModel
from ..architectures.utils import get_model, load_resume


class MobileNetFAS(IFASModel):
    """
    Wrapper for MobileNet-based Face Anti-Spoofing (FAS) models.

    Args:
        weights (Path): Path to the model weights.
        arch (str): Architecture variant
        input_size (int): Input image size.
        num_classes (int): Number of output classes.
        spoof_index (int): Index for spoof class.
        device (str | torch.device): Device to run the model on.
        trust_coef (float): Trust coefficient for predictions.
    """

    def __init__(
        self,
        weights: Path,
        arch: str = "mobilenetv3_large",
        input_size: int = 224,
        num_classes: int = 2,
        spoof_index: int = 1,
        trust_coef: float = 1.0,
        device: Union[str, torch.device] = "cuda",
    ) -> None:
        self.arch_name = arch
        super().__init__(
            weights=weights,
```

```
            device=device,
            input_size=input_size,
            num_classes=num_classes,
            spoof_index=spoof_index,
            trust_coef=trust_coef,
        )

    def _build_model(self) -> torch.nn.Module:
        """
        Build the MobileNet model.

        Returns:
            torch.nn.Module: The constructed model.
        """
        return get_model(self.arch_name, self.num_classes)

    def _load(self, weights: Path) -> None:
        """
        Load model weights from the specified path.

        Args:
            weights (Path): Path to the weights file.
        """
        load_resume(str(weights), self.model)
```

Файл models.fas.wrappers.shufflenetv2.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
ShuffleNetV2 model wrapper
"""

from pathlib import Path
from typing import Union
import torch

from .base import IFASModel
from ..architectures.utils import get_model, load_resume


class ShuffleNetFAS(IFASModel):
    """
    Face Anti-Spoofing model wrapper using ShuffleNetV2 architecture.

    Args:
        weights (Path): Path to the model weights.
        arch (str): Architecture variant
        input_size (int): Input image size.
        num_classes (int): Number of output classes.
        spoof_index (int): Index for spoof class.
        device (str | torch.device): Device to run the model on.
        trust_coef (float): Trust coefficient for predictions.
    """
    def __init__(
        self,
        weights: Path,
        arch: str = "shufflenetv2_1.0x",
        input_size: int = 224,
        num_classes: int = 2,
        spoof_index: int = 1,
        device: Union[str, torch.device] = "cuda",
        trust_coef: float = 1.0
    ):
        self.arch_name = arch
```

```
        super().__init__(
            weights=weights,
            device=device,
            input_size=input_size,
            num_classes=num_classes,
            spoof_index=spoof_index,
            trust_coef=trust_coef
        )

    def _build_model(self) -> torch.nn.Module:
        """
        Build the ShuffleNetV2 model.

        Returns:
            torch.nn.Module: Instantiated model.
        """
        return get_model(self.arch_name, self.num_classes)

    def _load(self, weights: Path) -> None:
        """
        Load model weights from the specified path.

        Args:
            weights (Path): Path to the weights file.
        """
        load_resume(str(weights), self.model)
```

Файл models.fas.wrappers. resnet50.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
ResNet50FAS  wrapper.
"""

from pathlib import Path
from typing import Union

import torch

from .base import IFASModel
from ..architectures.utils import get_model, load_resume


class ResNet50FAS(IFASModel):
    """
    Face anti-spoofing wrapper for ResNet-50 backbones.

    This class initializes and loads a ResNet-50 model for face anti-
spoofing,
    using utilities from the CVPR-2024 FAS-challenge repository.

    Args:
        weights (Path): Path to the model weights.
        arch (str): Architecture variant
        input_size (int): Input image size.
        num_classes (int): Number of output classes.
        spoof_index (int): Index for spoof class.
        device (str | torch.device): Device to run the model on.
        trust_coef (float): Trust coefficient for predictions.
    """

    def __init__(
        self,
        weights: Path,
        arch: str = "resnet50",
```

```
        input_size: int = 224,
        num_classes: int = 2,
        spoof_index: int = 1,
        device: Union[str, torch.device] = "cuda",
        trust_coef: float = 1.0,
    ) -> None:
        self.arch_name = arch
        super().__init__(
            weights=weights,
            device=device,
            input_size=input_size,
            num_classes=num_classes,
            spoof_index=spoof_index,
            trust_coef=trust_coef,
        )

    def _build_model(self) -> torch.nn.Module:
        """
        Build the ResNet-50 model using the specified architecture name and
number of classes.

        Returns:
            torch.nn.Module: The constructed model.
        """
        return get_model(self.arch_name, self.num_classes)

    def _load(self, weights: Path) -> None:
        """
        Load model weights from the specified path.

        Args:
            weights (Path): Path to the model weights.
        """
        load_resume(str(weights), self.model)
```

Файл models.fas.wrappers.swinv2.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
SwinV2FAS class wrapper
"""

from pathlib import Path
from typing import Union
import torch

from .base import IFASModel
from ..architectures.utils import get_model, load_resume


class SwinV2FAS(IFASModel):
    """
    SwinV2FAS is a wrapper class for SwinV2-based Face Anti-Spoofing models.

    Args:
        weights (Path): Path to the model weights.
        arch (str): Architecture variant
        input_size (int): Input image size.
        num_classes (int): Number of output classes.
        spoof_index (int): Index for spoof class.
        fp16 (bool): Whether to use half-precision.
        device (str | torch.device): Device to run the model on.
        trust_coef (float): Trust coefficient for predictions.
    """
```

```
    def __init__(
        self,
        weights: Path,
        arch: str = "swin_v2_t",
        input_size: int = 224,
        num_classes: int = 2,
        spoof_index: int = 1,
        fp16: bool = False,
        device: Union[str, torch.device] = "cuda",
        trust_coef: float = 1.0,
    ) -> None:
        self.arch_name: str = arch
        self.fp16: bool = fp16
        super().__init__(
            weights=weights,
            device=device,
            input_size=input_size,
            num_classes=num_classes,
            spoof_index=spoof_index,
            trust_coef=trust_coef,
        )

    def _build_model(self) -> torch.nn.Module:
        """
        Build the SwinV2 model with the specified configuration.

        Returns:
            torch.nn.Module: The constructed model.
        """
        return get_model(self.arch_name, self.num_classes, fp16=self.fp16)

    def _load(self, weights: Path) -> None:
        """
        Load model weights from the specified path.

        Args:
            weights (Path): Path to the weights file.
        """
        load_resume(str(weights), self.model)
```

Файл models.fas. ensemble.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
This module defines the FASEnsemble class for combining multiple face anti-
spoofing models
using various reduction strategies (mean, max, vote, weighted). Supports
optional parallel
inference on CUDA devices.
"""

from typing import Iterable
import torch
import numpy as np

from .wrappers.base import IFASModel


class FASEnsemble(torch.nn.Module):
    """
    Ensemble of face anti-spoofing models with configurable reduction
strategies.

    Args:
```

```
        models (Iterable[IFASModel]): An iterable of IFASModel instances to
ensemble.
        reduction (str, optional): Reduction method ('mean', 'max', 'vote',
'weighted'). Default is 'weighted'.
        parallel (bool, optional): Whether to run models in parallel on CUDA.
Default is False.

    Raises:
        ValueError: If an unknown reduction mode is specified or if
'weighted' reduction is used
                    and a model is missing the required 'trust_coef'
attribute.
    """

    def __init__(
        self,
        models: Iterable[IFASModel],
        reduction: str = "weighted",
        parallel: bool = True,
    ) -> None:
        super().__init__()
        self.models = list(models)
        self.reduction = reduction
        self.parallel = parallel
        if parallel and torch.cuda.is_available():
            self.streams = [torch.cuda.Stream() for _ in self.models]
        else:
            self.streams = [None] * len(self.models)

    @torch.inference_mode()
    def forward(self, crops: Iterable[np.ndarray]) -> torch.Tensor:
        """
        Forward pass for the ensemble.

        Args:
            crops (Iterable[np.ndarray]): Iterable of cropped face images as
numpy arrays.

        Returns:
            torch.Tensor: Ensemble prediction tensor.

        Raises:
            ValueError: If reduction mode is unknown or required model
attributes are missing.
        """
        outs = []

        if self.parallel and torch.cuda.is_available():
            for model, stream in zip(self.models, self.streams):
                with torch.cuda.stream(stream):
                    crop_copy = [c.copy() for c in crops]
                    out = model.predict_batch(crop_copy)
                    outs.append(out)
            for stream in self.streams:
                stream.synchronize()
        else:
            for model in self.models:
                crop_copy = [c.copy() for c in crops]
                out = model.predict_batch(crop_copy)
                outs.append(out)

        return self._reduce_outputs(torch.stack(outs, 0))  # shape:
(N_models, B)
```

```
    def _reduce_outputs(self, stack: torch.Tensor) -> torch.Tensor:
        if self.reduction == "mean":
            return stack.mean(0)
        if self.reduction == "max":
            return stack.max(0).values
        if self.reduction == "vote":
            return (stack >= 0.5).float().mean(0)
        if self.reduction == "weighted":
            try:
                weights = torch.tensor(
                    [m.trust_coef for m in self.models], dtype=torch.float32
                )
            except AttributeError as e:
                raise ValueError(
                    "All models must have a `.trust_coef` attribute for
'weighted' reduction."
                ) from e
            weights = weights.to(stack.device).view(-1, 1)
            weighted_sum = (stack * weights).sum(0)
            return weighted_sum / weights.sum()

        raise ValueError(f"Unknown reduction mode: {self.reduction}")
```

Файл pipeline.stages.fas_stage.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
FAS (Face Anti-Spoofing) pipeline stage module.

This module provides classes for integrating face anti-spoofing models
into a face processing pipeline. It includes a mock stage for testing
and a real stage that uses an ensemble of FAS models.
"""

from typing import Iterable, Optional
import numpy as np
import torch

from models.fas.ensemble import FASEnsemble
from .base_stage import PipelineStage
from ..memory.face_instance import FaceInstance


def _trbl_to_xyxy(bbox_trbl: Iterable[int]) -> tuple[int, int, int, int]:
    """
    Convert bounding box from (top, right, bottom, left) to (left, top,
right, bottom).

    Args:
        bbox_trbl (Iterable[int]): Bounding box as (top, right, bottom,
left).

    Returns:
        tuple[int, int, int, int]: Bounding box as (left, top, right,
bottom).
    """
    t, r, b, l = bbox_trbl
    return l, t, r, b


class MockFASStage(PipelineStage):
    """
    Mock FAS stage for testing. Assigns a fixed spoof score to each face.
```

```
    Args:
        min_score (float): Minimum possible spoof score.
        max_score (float): Maximum possible spoof score.
    """

    def __init__(self, min_score: float = 0.0, max_score: float = 1.0) ->
None:
        """
        Initialize the mock FAS stage.

        Args:
            min_score (float, optional): Minimum spoof score. Defaults to
0.0.
            max_score (float, optional): Maximum spoof score. Defaults to
1.0.
        """
        self.min_score = min_score
        self.max_score = max_score

    def process(
        self, frame: np.ndarray, faces: Iterable[FaceInstance]
    ) -> list[FaceInstance]:
        """
        Assign a spoof score of 0 to each face.

        Args:
            frame (np.ndarray): The input image frame.
            faces (Iterable[FaceInstance]): Iterable of FaceInstance objects.

        Returns:
            list[FaceInstance]: List of FaceInstance objects with spoof_score
set.
        """
        for face in faces:
            face.spoof_score = 0.0
        return list(faces)


class FASStage(PipelineStage):
    """
    Face Anti-Spoofing (FAS) pipeline stage using an ensemble model.

    Args:
        fas_model (FASEnsemble): The FASEnsemble model to use for spoof
detection.
        threshold (float): Spoof score threshold for classification.
        batch_size (int): Number of crops to process in a batch.
        padding (int): Padding (in pixels) to add around each face crop.
    """

    def __init__(
        self,
        fas_model: FASEnsemble,
        threshold: float = 0.05,
        batch_size: int = 32,
        padding: int = 0,
    ) -> None:
        """
        Initialize the FAS pipeline stage.

        Args:
            fas_model (FASEnsemble): The FASEnsemble model to use.
            threshold (float, optional): Spoof score threshold. Defaults to
0.05.
```

```
            batch_size (int, optional): Batch size for model inference.
Defaults to 32.
            padding (int, optional): Padding for face crops. Defaults to 0.
        """
        self.fas_model = fas_model
        self.threshold = threshold
        self.batch_size = batch_size
        self.padding = padding

    def _get_padded_bounds(self, bbox: Iterable[int]) -> tuple[int, int, int,
int]:
        """
        Calculate padded bounding box coordinates.

        Args:
            bbox (Iterable[int]): Bounding box in (top, right, bottom, left)
format.

        Returns:
            tuple[int, int, int, int]: Padded coordinates as (x1, y1, x2,
y2).
        """
        x1, y1, x2, y2 = _trbl_to_xyxy(bbox)
        return (
            x1 - self.padding,
            y1 - self.padding,
            x2 + self.padding,
            y2 + self.padding,
        )

    def _clamp_bounds(
        self, padded_bounds: tuple[int, int, int, int], frame_shape:
tuple[int, int]
    ) -> tuple[int, int, int, int]:
        """
        Clamp padded bounds to frame dimensions.

        Args:
            padded_bounds (tuple[int, int, int, int]): Padded bounds as (x1,
y1, x2, y2).
            frame_shape (tuple[int, int]): Frame shape as (height, width).

        Returns:
            tuple[int, int, int, int]: Clamped bounds as (x1, y1, x2, y2).
        """
        h, w = frame_shape
        x1_padded, y1_padded, x2_padded, y2_padded = padded_bounds

        return (
            max(0, x1_padded),
            max(0, y1_padded),
            min(w, x2_padded),
            min(h, y2_padded),
        )

    def _safe_crop(
        self, frame: np.ndarray, bbox: Iterable[int]
    ) -> Optional[np.ndarray]:
        """
        Extract a face crop from the frame with fixed padding.

        Args:
            frame (np.ndarray): The input image frame.
```

```
            bbox (Iterable[int]): Bounding box in (top, right, bottom, left)
format.

        Returns:
            Optional[np.ndarray]: Cropped face image or None if crop is
invalid.
        """
        padded_bounds = self._get_padded_bounds(bbox)
        final_bounds = self._clamp_bounds(padded_bounds, frame.shape[:2])

        x1_final, y1_final, x2_final, y2_final = final_bounds

        if x2_final <= x1_final or y2_final <= y1_final:
            return None

        crop = frame[y1_final:y2_final, x1_final:x2_final]
        return crop if crop.size > 0 else None

    def _prepare_crops(
        self, frame: np.ndarray, faces: Iterable[FaceInstance]
    ) -> tuple[list[FaceInstance], list[np.ndarray]]:
        """
        Prepare valid face crops and corresponding FaceInstance objects.

        Args:
            frame (np.ndarray): The input image frame.
            faces (Iterable[FaceInstance]): Iterable of FaceInstance objects.

        Returns:
            tuple[list[FaceInstance], list[np.ndarray]]: Valid faces and
their crops.
        """
        valid_faces = []
        crops = []
        for face in faces:
            crop = self._safe_crop(frame, face.bbox)
            if crop is not None:
                valid_faces.append(face)
                crops.append(crop)
            else:
                face.spoof_score = None
                face.is_spoof = False
        return valid_faces, crops

    def _predict_scores(self, crops: list[np.ndarray]) -> torch.Tensor:
        """
        Predict spoof scores for a list of face crops.

        Args:
            crops (list[np.ndarray]): List of face crops.

        Returns:
            torch.Tensor: Spoof scores for each crop.
        """
        all_scores = []
        for i in range(0, len(crops), self.batch_size):
            batch = crops[i : i + self.batch_size]
            all_scores.append(self.fas_model(batch))
        return torch.cat(all_scores, dim=0).cpu()

    def process(
        self, frame: np.ndarray, faces: Iterable[FaceInstance]
    ) -> list[FaceInstance]:
        """
```

```
        Process faces in the frame and assign spoof scores.

        Args:
            frame (np.ndarray): The input image frame.
            faces (Iterable[FaceInstance]): Iterable of FaceInstance objects.

        Returns:
            list[FaceInstance]: List of FaceInstance objects with spoof_score
and is_spoof set.
        """
        faces = list(faces)
        if not faces:
            return faces

        valid_faces, crops = self._prepare_crops(frame, faces)
        if not crops:
            return faces

        final_scores = self._predict_scores(crops)

        for face, score in zip(valid_faces, final_scores):
            face.spoof_score = float(score)
            face.is_spoof = face.spoof_score >= self.threshold

        return faces
```

Файл pipeline.stages.recognition_stage.py

Реалізація функціональної задачі розпізнавання обличчя.

```
"""
Recognition stage for the FaceFinder pipeline.

This module defines the RecognitionStage class, which is responsible for
encoding detected faces and matching them against known face encodings
to assign identities.
"""

from typing import Iterable
import numpy as np

from models.face_encoders.base import IEncoder
from storage.face_repository import FaceEncodingRepository
from .base_stage import PipelineStage
from ..memory.face_instance import FaceInstance


class RecognitionStage(PipelineStage):
    """
    Pipeline stage for recognizing faces by comparing encodings to known
identities.

    Attributes:
        encoder (IEncoder): Encoder used to generate face embeddings.
        repository (FaceEncodingRepository): Repository of known face
encodings.
        threshold (float): Distance threshold for recognition.
        known_encodings (np.ndarray): Array of known face encodings.
        known_names (list[str]): List of known identities.
    """

    def __init__(
        self,
        encoder: IEncoder,
        repository: FaceEncodingRepository,
        threshold: float = 0.6,
```

```
    ) -> None:
        """
        Initialize the RecognitionStage.

        Args:
            encoder (IEncoder): Encoder for face embeddings.
            repository (FaceEncodingRepository): Repository of known faces.
            threshold (float, optional): Distance threshold for recognition.
Defaults to 0.6.
        """
        self.encoder = encoder
        self.repository = repository
        self.threshold = threshold
        self.known_encodings, self.known_names = self.repository.get_all()

    def process(
        self, frame: np.ndarray, faces: Iterable[FaceInstance]
    ) -> list[FaceInstance]:
        """
        Encode faces in the frame and assign identities if matched.

        Args:
            frame (np.ndarray): The image frame containing faces.
            faces (Iterable[FaceInstance]): Detected face instances.

        Returns:
            list[FaceInstance]: Face instances with updated identities.
        """
        result = []
        for face in faces:
            encoding = self.encoder.encode(frame, face.bbox)
            face.encoding = encoding
            if encoding is not None and len(self.known_encodings) > 0:
                matches = [
                    np.linalg.norm(encoding - known) for known in
self.known_encodings
                ]
                best_match_idx = int(np.argmin(matches))
                if matches[best_match_idx] < self.threshold:
                    face.identity = self.known_names[best_match_idx]
            result.append(face)
        return result
```

Файл pipeline.stages.face_detection_stage.py

Реалізація функціональної задачі виявлення обличчя.

```
"""
Face detection pipeline stage module.

This module defines the FaceDetectionStage class, which applies a face
detector
to an input frame and returns detected face instances.
"""

from typing import Iterable
import numpy as np

from models.face_detectors.base import IDetector
from pipeline.stages.base_stage import PipelineStage
from pipeline.memory.face_instance import FaceInstance


class FaceDetectionStage(PipelineStage):
    """
    Pipeline stage for detecting faces in an image frame.
```

```
    Attributes:
        detector (IDetector): The face detector to use.
    """

    def __init__(self, detector: IDetector) -> None:
        """
        Initialize the FaceDetectionStage.

        Args:
            detector (IDetector): The face detector instance.
        """
        self.detector = detector

    def process(
        self, frame: np.ndarray, faces: Iterable[FaceInstance]
    ) -> list[FaceInstance]:
        """
        Detect faces in the given frame.

        Args:
            frame (np.ndarray): The image frame to process.
            faces (Iterable[FaceInstance]): Existing face instances (unused).

        Returns:
            list[FaceInstance]: List of detected face instances.
        """
        boxes = self.detector.get_face_locations(frame)
        return [FaceInstance(bbox=tuple(box)) for box in boxes]
```

Файл pipeline.smoothing.bbox_smoother.py

Реалізація функціональної задачі виявлення обличчя.

```
"""
bbox_smoother.py

Implements a moving-average smoother for bounding boxes using the history
maintained by DataHandler. No private buffers are stored in this class.
"""

from typing import Iterable
import numpy as np

from .base import ISmoother
from ..memory.face_instance import FaceInstance
from ..memory.data_handler import DataHandler


class BboxSmoother(ISmoother):
    """
    Moving-average smoother for bounding boxes.

    This smoother reuses the Track.history held by DataHandler to compute
    the moving average of bounding boxes for each face instance. No private
    buffers are stored in this class.
    """

    def __init__(self, window: int = 5) -> None:
        """
        Initialize the BboxSmoother.

        Args:
            window (int): The window size for the moving average.
```

```
        """
        self.window = int(window)

    def smooth(
        self,
        faces: Iterable[FaceInstance],
        memory: DataHandler,
    ) -> None:
        """
        Smooth the bounding boxes of the given face instances using a moving
average.

        Args:
            faces (Iterable[FaceInstance]): Iterable of face instances to
smooth.
            memory (DataHandler): DataHandler containing the tracks and their
histories.
        """

        for f in faces:
            if f.uuid is None or f.spoof_score is None:
                continue

            trk = memory.get_track(f.uuid)
            if trk is None:
                continue

            recent_boxes = [
                inst.bbox
                for inst in list(trk.history)[-self.window :]
                if inst.bbox is not None
            ]
            if recent_boxes:
                arr = np.asarray(recent_boxes, dtype=np.float32)  # shape
(k,4)
                mean_box = arr.mean(axis=0)  # per-coordinate mean
                f.bbox_smoothed = tuple(int(round(c)) for c in mean_box)
```

Файл pipeline.smoothing.fas_smoother.py

Реалізація функціональної задачі виявлення штучно змінених облич.

```
"""
Implements a moving-average smoother for face spoof scores,
reusing track history from DataHandler.
"""

from typing import Iterable
import numpy as np

from .base import ISmoother
from ..memory.face_instance import FaceInstance
from ..memory.data_handler import DataHandler


class FASSmoother(ISmoother):
    """
    Moving-average smoother for face spoof scores.

    This smoother reuses the Track.history held by DataHandler and does not
store private buffers.
    It updates each FaceInstance in-place with a smoothed spoof score and
spoof flag.
    """
```

```
    def __init__(self, window: int = 5, thresh: float = 0.50) -> None:
        """
        Initialize the FASSmoother.

        Args:
            window: Number of recent spoof scores to average.
            thresh: Threshold for classifying a face as spoof.
        """
        self.window = int(window)
        self.thresh = float(thresh)

    def smooth(
        self,
        faces: Iterable[FaceInstance],
        memory: DataHandler,
    ) -> None:
        """
        Smooth spoof scores for a collection of FaceInstance objects.

        For each face, computes the moving average of the last N spoof scores
        from its track history, and updates the face's smoothed spoof score
        and spoof flag in-place.

        Args:
            faces: Iterable of FaceInstance objects to smooth.
            memory: DataHandler containing track histories.
        """
        for face in faces:
            if face.uuid is None or face.spoof_score is None:
                continue

            track = memory.get_track(face.uuid)
            if track is None:
                continue

            # Collect the last N spoof-scores from the track's history
            recent_scores = [
                inst.spoof_score
                for inst in list(track.history)[-self.window :]
                if inst.spoof_score is not None
            ]

            if not recent_scores:
                continue

            avg = float(np.mean(recent_scores))

            # In-place update for this frame's FaceInstance
            face.spoof_score_smoothed = avg
            face.is_spoof = avg >= self.thresh
```

Файл pipeline.memory.data_handler.py

Реалізація функціональної задачі розпізнавання обличчя.

```
"""
Data handler module for managing face tracking across frames.
"""

from typing import Optional, Iterable
from collections import deque
from .face_instance import FaceInstance
from . import face_track
from . import utils
```

```
class DataHandler:
    """
    Handles association of detected faces with tracks and manages track
lifecycle.

    Attributes:
        embed_thr (float): Embedding similarity threshold for matching.
        iou_thr_id (float): IOU threshold for identity matching.
        iou_thr_blk (float): IOU threshold for blank matching.
        ttl_init (int): Initial time-to-live for tracks.
        _tracks (dict[str, face_track.Track]): Active tracks indexed by UUID.
        _next_uuid (int): Counter for generating new UUIDs.
    """

    def __init__(
        self,
        embed_thr: float = 0.55,
        iou_thr_id: float = 0.25,
        iou_thr_blank: float = 0.5,
        ttl_frames: int = 30,
    ) -> None:
        """
        Initialize the DataHandler.

        Args:
            embed_thr (float): Embedding similarity threshold.
            iou_thr_id (float): IOU threshold for identity matching.
            iou_thr_blank (float): IOU threshold for blank matching.
            ttl_frames (int): Number of frames a track remains alive without
updates.
        """
        self.embed_thr = float(embed_thr)
        self.iou_thr_id = float(iou_thr_id)
        self.iou_thr_blk = float(iou_thr_blank)
        self.ttl_init = int(ttl_frames)

        self._tracks: dict[str, face_track.Track] = {}
        self._next_uuid = 0

    def init_frame(self, _frame) -> Iterable[FaceInstance]:
        """
        Initialize a frame (currently returns an empty iterable).

        Args:
            _frame: Frame data (unused).

        Returns:
            Iterable[FaceInstance]: Empty iterable.
        """
        return []

    def update(self, faces: Iterable[FaceInstance]) -> None:
        """
        Associate faces with existing tracks or create new tracks.

        Args:
            faces (Iterable[FaceInstance]): Detected faces for the current
frame.
        """
        faces = list(faces)
        if not faces and not self._tracks:
            return

        unused_idx: set[int] = set(range(len(faces)))
```

```
        # Book-keeping: decrement TTL for all tracks
        for trk in self._tracks.values():
            trk.ttl -= 1

        # Greedy matching per track
        for uuid, trk in list(self._tracks.items()):
            best_candidate = None

            for idx in list(unused_idx):
                f = faces[idx]
                candidate = utils.evaluate_candidate(
                    f, trk, idx, self.embed_thr, self.iou_thr_blk
                )

                if candidate is None:
                    continue

                if best_candidate is None or utils.is_better_candidate(
                    candidate, best_candidate
                ):
                    best_candidate = candidate

            # Apply decision
            if best_candidate is not None:
                idx = best_candidate["idx"]
                f = faces[idx]
                f.uuid = uuid
                trk.history.append(f)
                trk.last = f
                trk.ttl = self.ttl_init
                unused_idx.discard(idx)

        # Start new tracks for remaining faces
        for idx in unused_idx:
            f = faces[idx]
            new_id = f"face_{self._next_uuid}"
            f.uuid = new_id
            self._next_uuid += 1
            self._tracks[new_id] = face_track.Track(
                uuid=new_id, last=f, history=deque([f], maxlen=50),
ttl=self.ttl_init
            )

        # Remove dead tracks
        for uuid in [u for u, t in self._tracks.items() if t.ttl <= 0]:
            del self._tracks[uuid]

    def get_track(self, uuid: str) -> Optional[face_track.Track]:
        """
        Retrieve a track by its UUID.

        Args:
            uuid (str): Track UUID.

        Returns:
            Optional[face_track.Track]: The track if found, else None.
        """
        return self._tracks.get(uuid)

    def live_uuids(self) -> Iterable[str]:
        """
        Get a list of currently live track UUIDs.
```

```
        Returns:
            Iterable[str]: List of live UUIDs.
        """
        return list(self._tracks.keys())
```

Файл storage.encoding_storage.py

Реалізація функціональної задач розпінавання та захоплення обличчя.

```
"""
Module for storing and managing face encodings using SQLite.

This module provides the SQLiteEncodingStorage class, which implements
the IEncodingStorage interface for persistent storage of face encodings
and associated person metadata in a SQLite database.
"""

import sqlite3
from typing import Iterable
import numpy as np

from .base import IEncodingStorage


class SQLiteEncodingStorage(IEncodingStorage):
    """
    SQLite-based implementation of the IEncodingStorage interface.

    Stores face encodings and associated person information in a SQLite
database.
    """

    def __init__(self, path: str = "face_encodings.db") -> None:
        """
        Initialize the storage with the given database path.

        Args:
            path: Path to the SQLite database file.
        """
        self.db_path = path
        self._initialize_db()

    def _connect(self) -> sqlite3.Connection:
        """
        Create and return a new SQLite database connection.

        Returns:
            sqlite3.Connection: A new database connection.
        """
        return sqlite3.connect(self.db_path)

    def _initialize_db(self) -> None:
        """
        Initialize the database schema if it does not exist.
        """
        with self._connect() as conn:
            cursor = conn.cursor()
            cursor.execute(
                """
                CREATE TABLE IF NOT EXISTS persons (
                    uuid TEXT PRIMARY KEY,
                    name TEXT NOT NULL
                )
            """
            )
            cursor.execute(
```

```
                """
                CREATE TABLE IF NOT EXISTS encodings (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    person_uuid TEXT NOT NULL,
                    encoding BLOB NOT NULL,
                    FOREIGN KEY (person_uuid) REFERENCES persons(uuid) ON
DELETE CASCADE
                )
            """
            )
            conn.commit()

    def load_encodings(self) -> tuple[Iterable[np.ndarray], Iterable[str]]:
        """
        Load all face encodings and their associated person names.

        Returns:
            Tuple[Iterable[np.ndarray], Iterable[str]]: A tuple containing an
iterable
                                        of encodings and an iterable of
names.
        """
        with self._connect() as conn:
            cursor = conn.cursor()
            cursor.execute(
                """
                SELECT p.name, e.encoding
                FROM encodings e
                JOIN persons p ON e.person_uuid = p.uuid
            """
            )

            names = []
            encodings = []
            for name, blob in cursor.fetchall():
                encoding = np.frombuffer(blob, dtype=np.float32)
                names.append(name)
                encodings.append(encoding)

        return encodings, names

    def add_person(self, name: str, person_uuid: str) -> None:
        """
        Add a new person to the database.

        Args:
            name: The name of the person.
            person_uuid: The unique identifier for the person.
        """
        with self._connect() as conn:
            conn.execute(
                "INSERT INTO persons (uuid, name) VALUES (?, ?)",
(person_uuid, name)
            )

    def add_encoding(self, person_uuid: str, encoding: np.ndarray) -> None:
        """
        Add a new face encoding for a person.

        Args:
            person_uuid: The unique identifier for the person.
            encoding: The face encoding as a NumPy array.
        """
        blob = encoding.astype(np.float32).tobytes()
```

```
        with self._connect() as conn:
            conn.execute(
                "INSERT INTO encodings (person_uuid, encoding) VALUES (?,
?)",
                (person_uuid, blob),
            )

    def delete_person(self, person_uuid: str) -> None:
        """
        Delete a person and all associated encodings from the database.

        Args:
            person_uuid: The unique identifier for the person.
        """
        with self._connect() as conn:
            conn.execute("DELETE FROM persons WHERE uuid = ?",
(person_uuid,))

    def rename_person(self, person_uuid: str, new_name: str) -> None:
        """
        Rename a person in the database.

        Args:
            person_uuid: The unique identifier for the person.
            new_name: The new name for the person.
        """
        with self._connect() as conn:
            conn.execute(
                "UPDATE persons SET name = ? WHERE uuid = ?", (new_name,
person_uuid)
            )

    def get_all_persons(self) -> dict[str, str]:
        """
        Retrieve all persons from the database.

        Returns:
            dict[str, str]: A dictionary mapping person UUIDs to names.
        """
        with self._connect() as conn:
            cursor = conn.execute("SELECT uuid, name FROM persons")
            return dict(cursor.fetchall())
```

Файл storage.face_repository.py

Реалізація функціональної задач розпінавання та захоплення обличчя.

```
"""
Repository class for managing face encodings and associated persons.
"""

from typing import Iterable
import uuid
import numpy as np

from .encoding_storage import IEncodingStorage


class FaceEncodingRepository:
    """
    Repository for managing face encodings and person identities.

    Attributes:
```

```
        storage (IEncodingStorage): Storage backend for encodings and person
data.
    """

    def __init__(self, storage: IEncodingStorage) -> None:
        """
        Initialize the repository with a storage backend.

        Args:
            storage (IEncodingStorage): The storage implementation to use.
        """
        self.storage = storage

    def add_person(self, name: str, encodings: Iterable[np.ndarray]) -> str:
        """
        Add a new person and their face encodings to the repository.

        Args:
            name (str): The name of the person.
            encodings (Iterable[np.ndarray]): Iterable of face encodings.

        Returns:
            str: The UUID of the newly added person.
        """
        person_uuid = uuid.uuid4().hex
        self.storage.add_person(name, person_uuid)
        for encoding in encodings:
            self.storage.add_encoding(person_uuid, encoding)
        return person_uuid

    def update_person(
        self, person_uuid: str, new_encodings: Iterable[np.ndarray]
    ) -> None:
        """
        Add new encodings to an existing person.

        Args:
            person_uuid (str): The UUID of the person.
            new_encodings (Iterable[np.ndarray]): Iterable of new face
encodings.
        """
        for encoding in new_encodings:
            self.storage.add_encoding(person_uuid, encoding)

    def delete_person(self, person_uuid: str) -> None:
        """
        Delete a person and their encodings from the repository.

        Args:
            person_uuid (str): The UUID of the person to delete.
        """
        self.storage.delete_person(person_uuid)

    def get_all(self):
        """
        Retrieve all persons and their encodings from the repository.

        Returns:
            Any: All stored persons and their encodings.
        """
        return self.storage.load_encodings()
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧ У ВІДЕОПОТОЦІ З ДЕТЕКЦІЄЮ ШТУЧНО ЗМІНЕНИХ ЗОБРАЖЕНЬ

Програма та методика тестування

КПІ.ІП-1224.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Світлана ПОПЕРЕШНЯК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Родіон СКОРИК

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень, розроблений з використанням мови програмування Python, з використанням PyQt, PyTorch, ONXX, OpenCV, Dlib та та системи управління базами даних SQLite.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- оцінка швидкодії під час роботи в реальному часі;
- перевірка точності виявлення облич;
- перевірка точності розпізнавання штучно змінених облич (спуфінгу);
- перевірка якості розпізнавання штучно змінених облич;
- перевірка збереження даних закодованих облич;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка роботи з різними джерелами відео (камера, відеофайл);
- оцінка зручності інтерфейсу користувача та повідомлень про помилки;

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- pylint – для статичного аналізу коду, виявлення помилок стилю та потенційних помилок;
- radon – для оцінки метрик якості коду (Cyclomatic Complexity, Maintainability Index);
- набір відео-даних (датасет) – для функціонального тестування детектора і FAS-моделей.

Порядок проведення тестування буде наступним:

- статичне тестування: перевірка коду за допомогою pylint на відповідність стандартам написання, аналіз складності коду та підтримуваності за допомогою radon.
- мануальне тестування: перевірка коректності роботи GUI (запуск, обробка відео, повідомлення про помилки) та логіки програми (коректне виявлення, розпізнавання та FAS)
- функціональне тестування: запуск на підготовленому відеодатасеті, збирання статистики щодо точності детекції та FAS
- Динамічне тестування продуктивності: оцінка швидкодії при обробці відеопотоку, вимірювання FPS.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧ У ВІДЕОПОТОЦІ З ДЕТЕКЦІЄЮ ШТУЧНО ЗМІНЕНИХ ЗОБРАЖЕНЬ

Керівництво користувача

КПІ.ІП-1224.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Світлана ПОПЕРЕШНЯК

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Родіон СКОРИК

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«FaceFinder» - це декстопний застосунок для для операційної системи Windows, призначений для розпізнавання облич у відеопотоці та виявлення штучно змінених облич. Програма дозволяє захоплювати нові обличчя з вебкамери або відео, призначати їм унікальні мітки, зберігати їх у локальну базу та виконувати розпізнавання в реальному часі. Крім того, система може визначати, чи є обличчя справжнім, чи це спроба обману – наприклад, фотографія на телефоні або дипфейк.

Програму реалізовано мовою Python із графічним інтерфейсом на базі PyQt6, а для роботи нейронних мереж використано фреймворки PyTorch, ONNX та бібліотеку OpenCV для обробки зображень. Готова збірка доступна у вигляді .exe-файлу з інсталяційним пакетом .msi, що дозволяє легко встановити застосунок на будь-який комп’ютер. Установку можна виконати за кілька кроків, а для пришвидшення роботи за допомогою відеокарти необхідне використання CUDA Toolkit.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– процесор: Intel Core i7-9750H або аналогічний (6 ядер, 12 потоків);

– об‘єм ОЗП: 16 Гб;

– об’єм дискового простору: 10 Гб;

– операційна система: Windows 10 x64;

– Відеокарта: дискретна відеокарта з обсягом відеопам’яті не менше 6 ГБ (наприклад, NVIDIA GTX 1660 Ti або краща).

– Наявність зовнішньої чи вбудованої камери

2.2 Завантаження застосунку

На поточному етапі застосунок доступний у вигляді інсталяційного пакета формату .msi, який можна завантажити з наданого джерела в офіційному репозиторії проєкту. Після завантаження .msi-файлу його можна запустити подвійним натисканням для початку встановлення.

Окрім інсталятора, у тому ж архіві або директорії користувач також знайде .exe-файл готової до запуску програми, а також допоміжні файли, включаючи конфігурації та моделі. Для коректної роботи системи необхідно переконатися, що файли ваг нейронних мереж (моделі розпізнавання та FAS) розміщено в одній папці з виконуваним файлом. Якщо ці файли не включені до пакета автоматично, їх слід завантажити окремо.

Для роботи в реальному часі потрібно додатково встановити CUDA Toolkit 12.1 або вище, відповідно до офіційної документації NVIDIA.

2.3 Перевірка коректної роботи

Після завершення встановлення на робочому столі або в меню «Пуск» має з’явитися іконка застосунку FaceFinder. Якщо іконка не з’явилася або

запуск не відбувається, встановлення вважається неуспішним і слід повторити його.

Для перевірки коректної роботи системи користувачу потрібно запустити програму та обрати одне з доступних джерел відео: вебкамеру або відеофайл. Це можна зробити через кнопку Start Recognition на головному екрані застосунку. У відкритому діалоговому вікні слід обрати відповідне джерело.

Після вибору джерела має з’явитись вікно з відеопотоком, де система повинна автоматично виявити обличчя в кадрі. Якщо обличчя розпізнається, воно обрамлюється кольоровою рамкою, а над ним з’являється відповідна мітка або позначка FAS (у випадку атаки). Це свідчить про коректну роботу всіх основних компонентів застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

Після запуску програми FaceFinder користувач потрапляє на головний екран, де доступні три основні дії:

– Manage labels – відкриття менеджера захоплених облич (міток);

– Capture new face – захоплення обличчя користувача з вебкамери;

– Start recognition – запуск процесу розпізнавання в реальному часі або з відеофайлу.

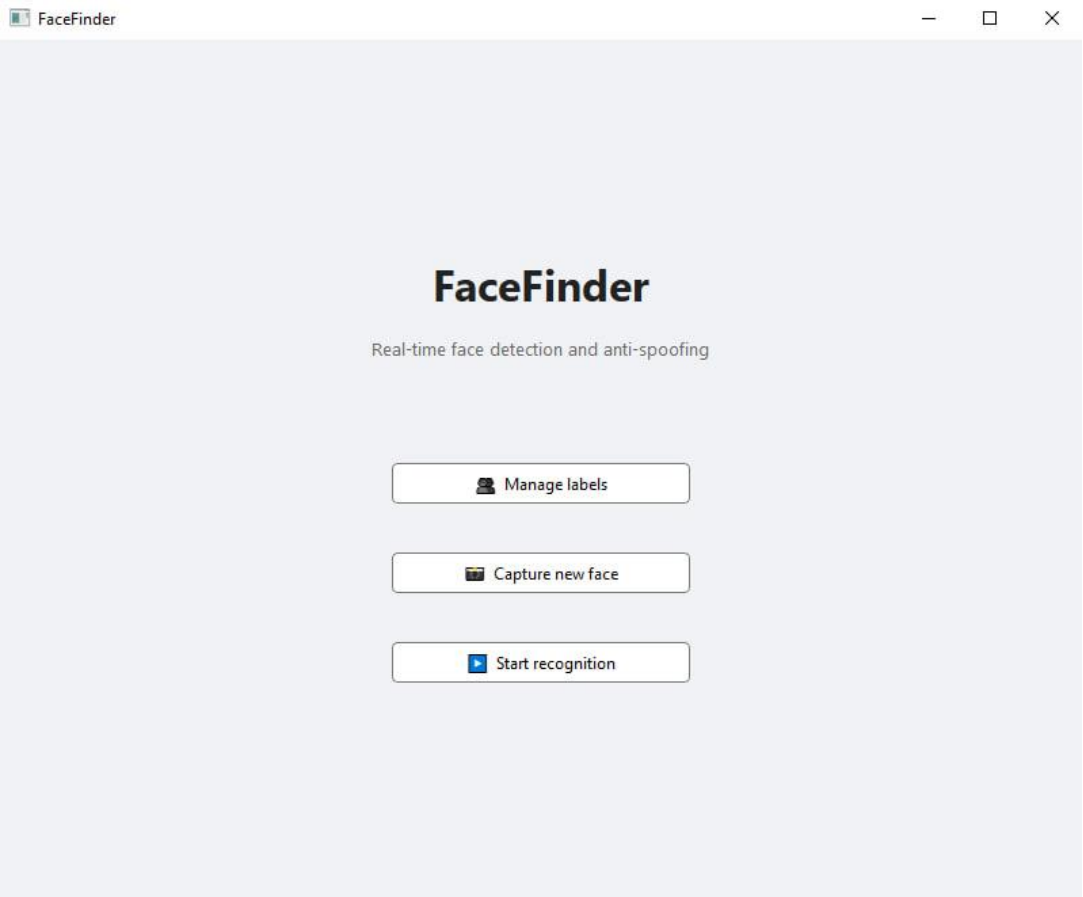


Рисунок 3.1 – Головний екран застосунку

Для додавання нового обличчя потрібно натиснути кнопку Capture new face. У діалоговому вікні користувач вводить мітку (ім’я) для нового обличчя та вибирає камеру.

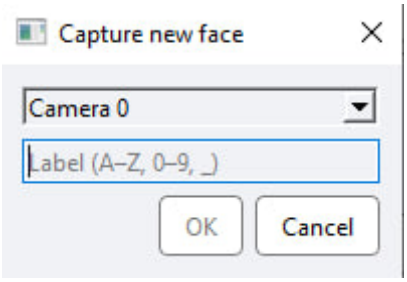


Рисунок 3.2 – Діалогове вікно перед захопленням обличчя

Після підтвердження відкривається вікно з відеопотоком, де система автоматично виявляє обличчя і позначає його зеленою рамкою. Користувач натискає кнопку Capture до 20 разів, змінюючи ракурси, щоб забезпечити якісне розпізнавання. Завершити захоплення достроково можна натисканням Done, або ж воно завршиться автоматично після захоплення 20 міток.

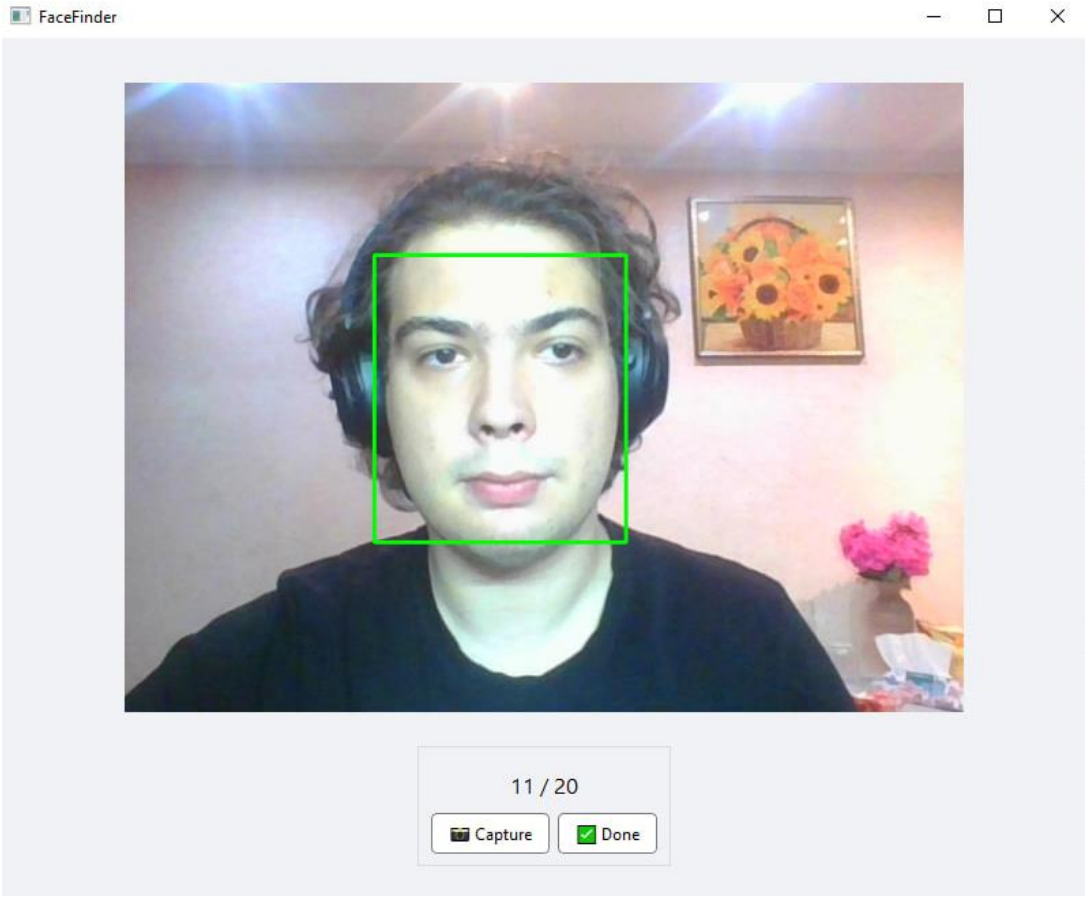


Рисунок 3.3 – Вікно захоплення обличчя

Після захоплення можна перейти до розділу Manage labels, щоб переглянути збережені мітки. Там відображається список усіх захоплених облич. Кожну мітку можна перейменувати або видалити. Це дає змогу перевірити, чи обличчя було збережено коректно.

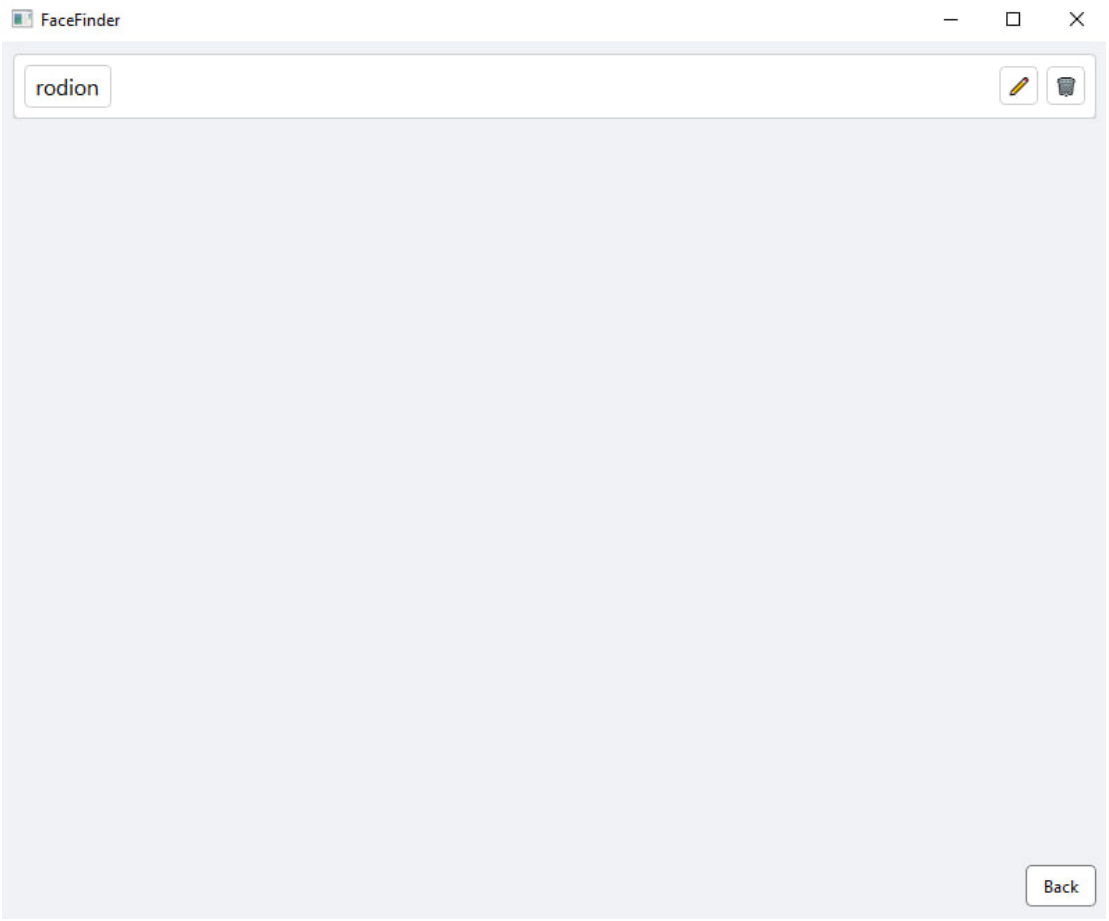


Рисунок 3.4 – Вікно менеджера міток

Для розпізнавання потрібно повернутися на головну сторінку й натиснути Start recognition. У діалоговому вікні обирається джерело відео: або камера, або відеофайл. Після вибору відкривається вікно, де в режимі реального часу буде виконано виявлення та розпізнавання облич.

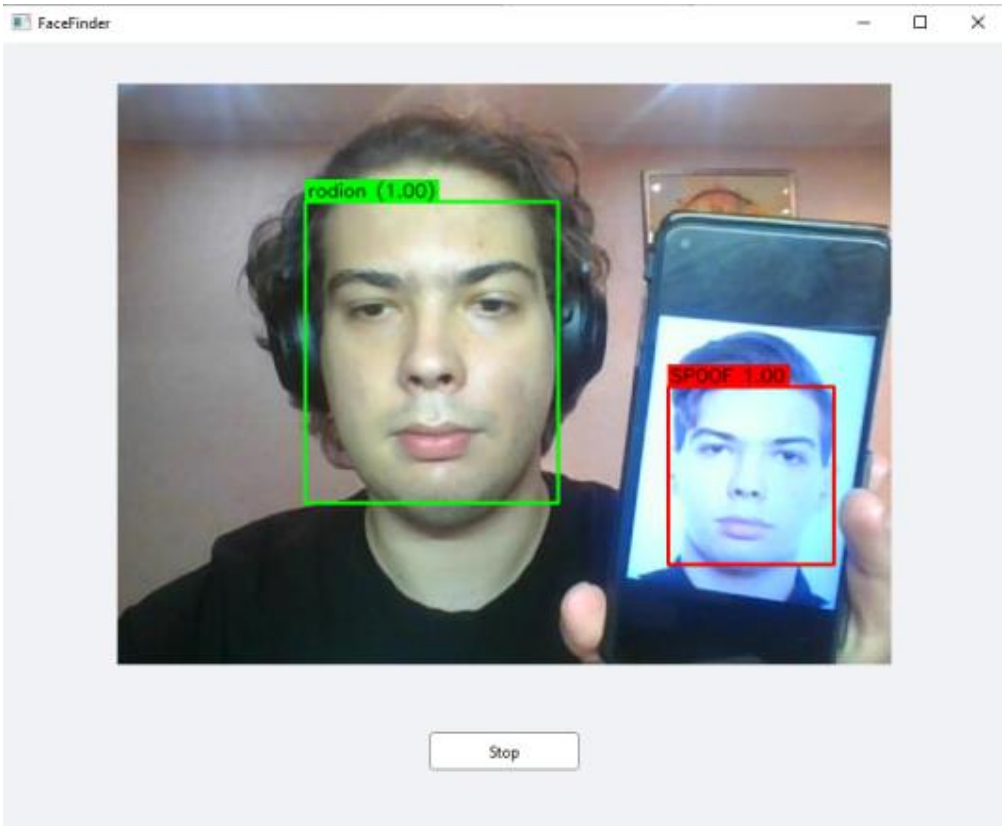


Рисунок 3.5 – Результат роботи системи на відеопотоці з камери

Рамка навколо обличчя змінює колір залежно від результату FAS-модуля:

– зелений – висока впевненість, що обличчя справжнє;
– жовтий або помаранчевий – нейтральна зона або сумнів;
– червоний – висока ймовірність спуфінгу (атаки).

Над рамкою виводиться мітка особи разом із числовим значенням ймовірності достовірності. У випадку, якщо система визначає атаку, мітка замінюється на напис SPOOF, а рамка змінюється на червону.

Для демонстрації атаки користувач може показати в камеру свій портрет на екрані телефону. У такому випадку система повинна відреагувати, підписавши зображення як SPOOF, а колір рамки навколо обличчя автоматично зміниться на червоний.

Така послідовність кроків дозволяє користувачу повністю освоїти роботу з застосунком: від додавання нового обличчя до успішного виявлення спроби обману системи. Якщо потрібно – можу підготувати цей текст із нумерацією кроків або вставками для рисунків.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧ У ВІДЕОПОТОЦІ З ДЕТЕКЦІЄЮ ШТУЧНО ЗМІНЕНИХ ЗОБРАЖЕНЬ

Графічний матеріал

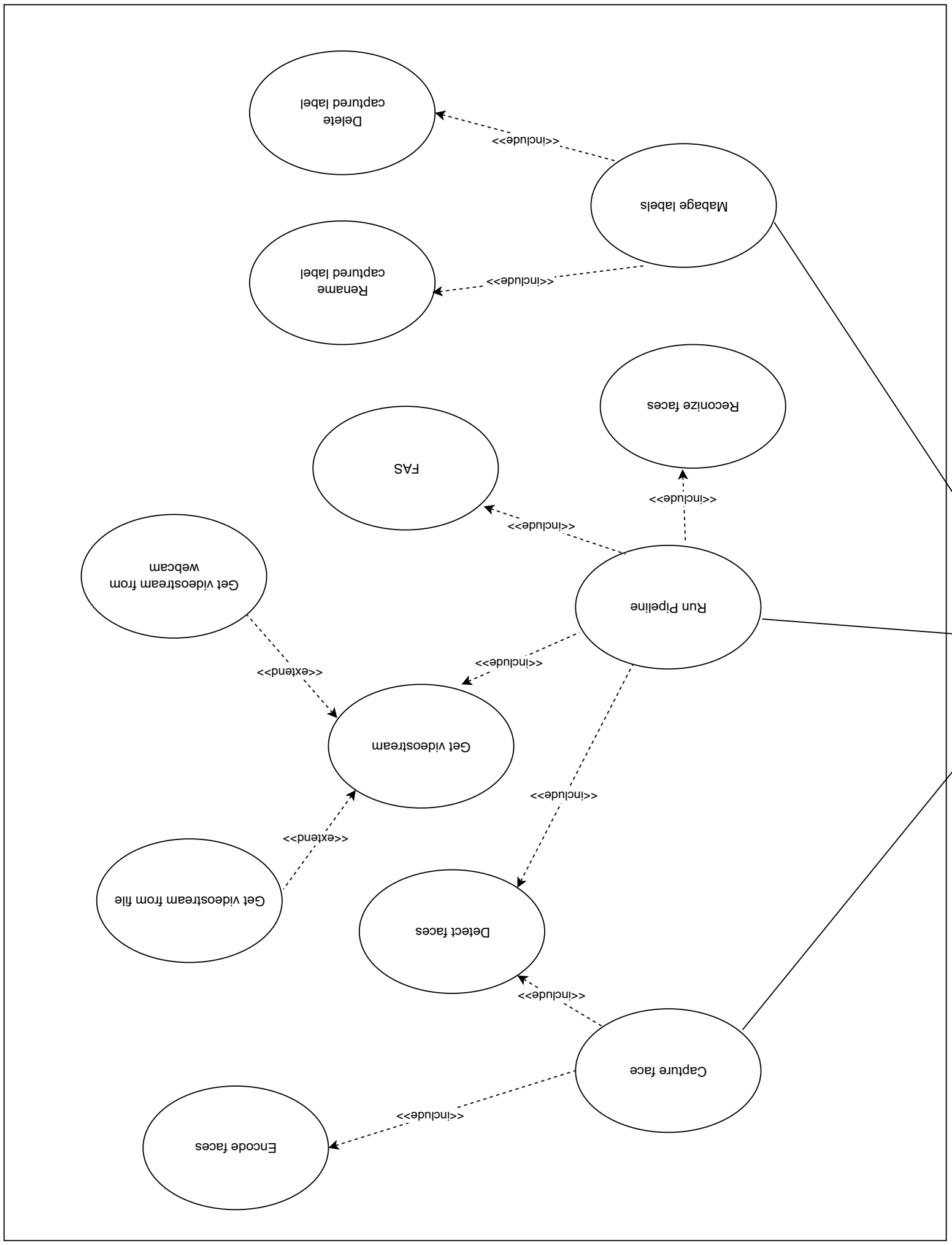
КПІ.ІП-1224.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Світлана ПОПЕРЕШНЯК

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

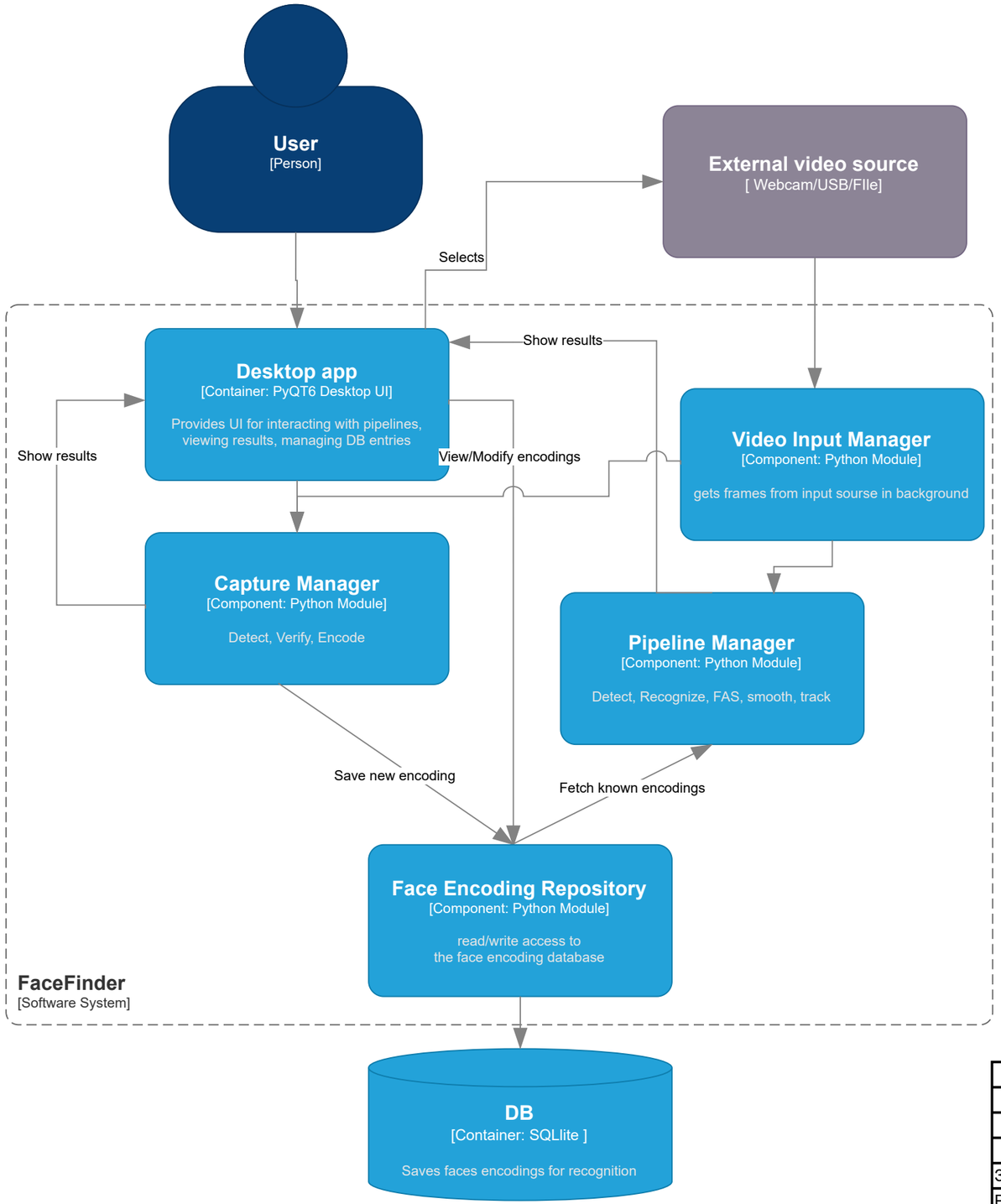
Виконавець:
_____ Родіон СКОРИК

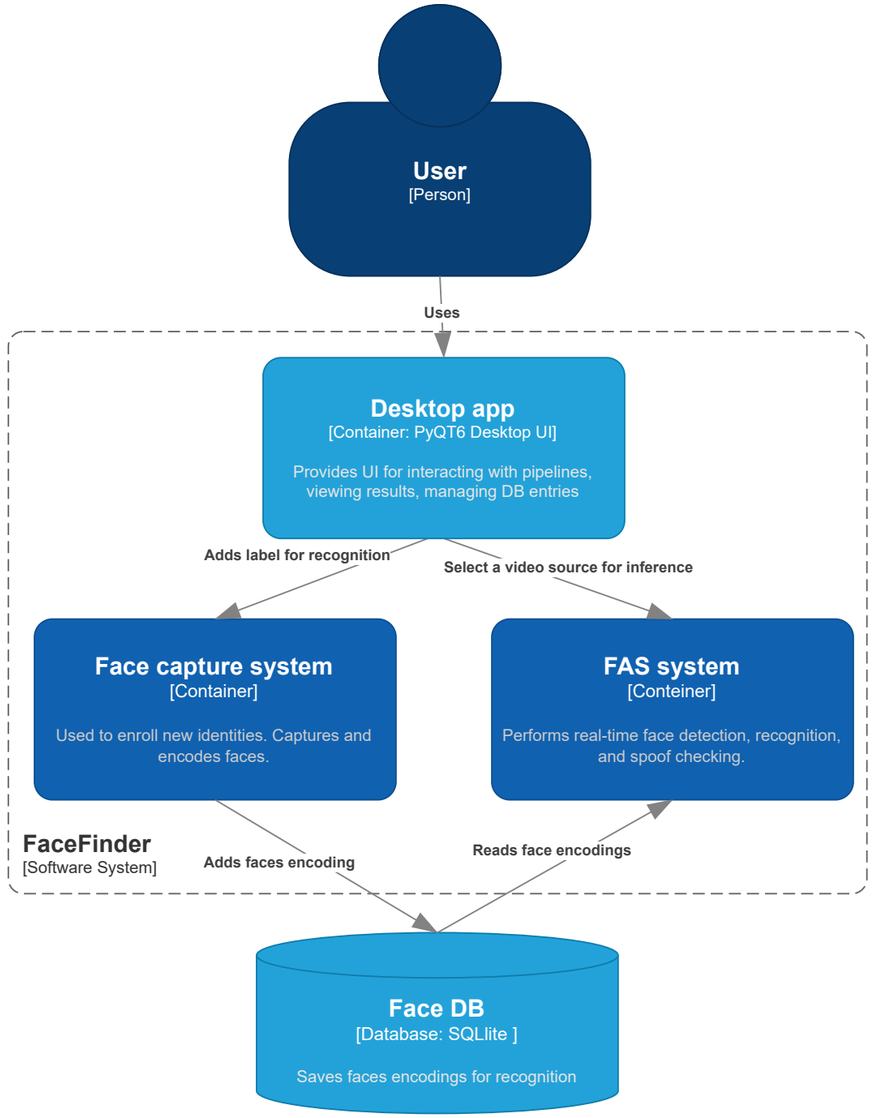
Київ – 2025



Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1224.045490.06.99.ССВ				
Розробив		Скорик Р.О.			Схема структура варіантів використань			Літ.	Маса	Масштаб
Перевірив		Полєщук С.В.								
Т. контр.								Аркуш 1		Аркушів 1
Н. контр.		Головченко М.М.			Застосунок для розпізнавання обличч у відеопотоці з детекцією штучно змінених зображень			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Затвердив		Жаріков Е.В.								

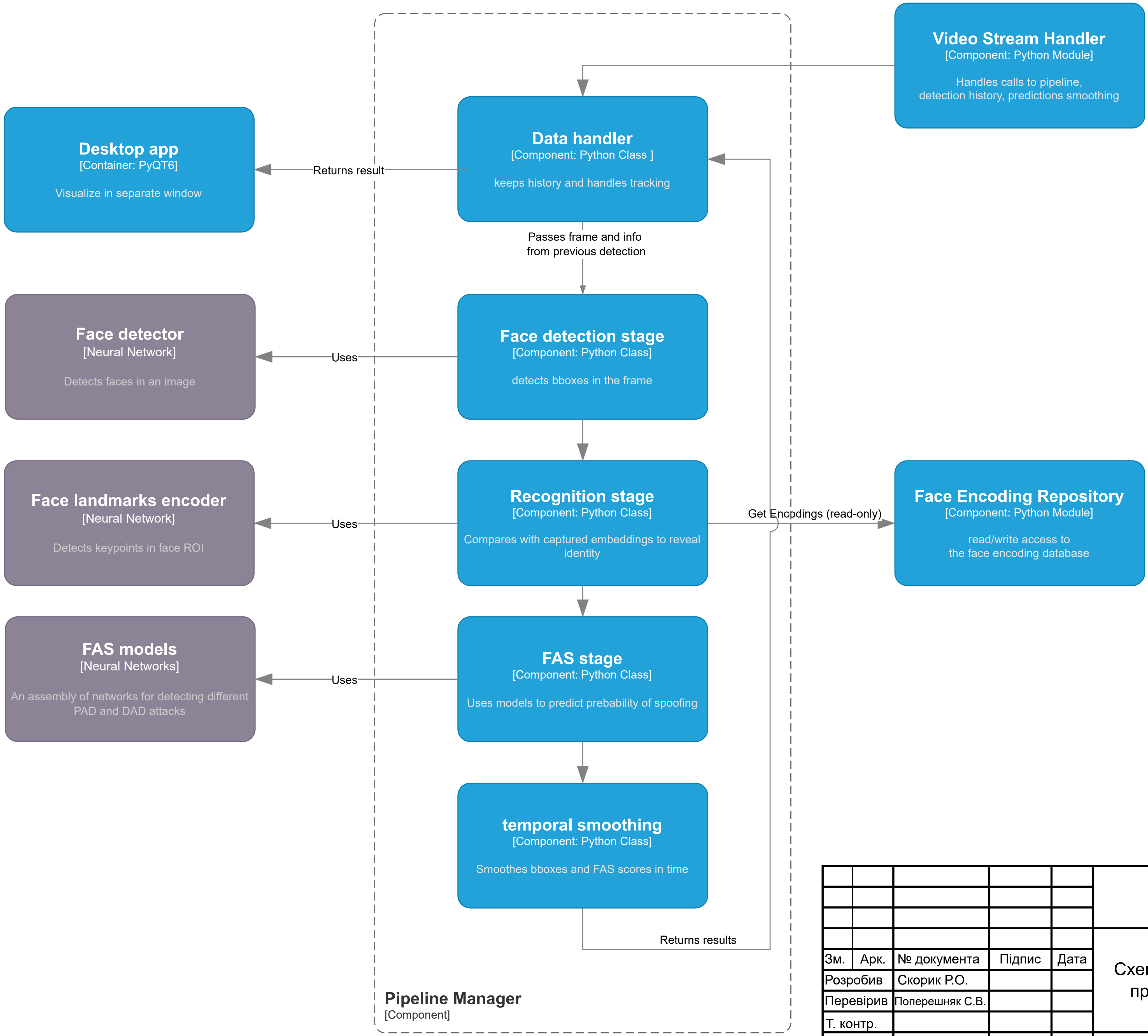
КПІ.ІП-1224.045490.06.99.ССМ





					КПІ.ІП-1224.045490.06.99.ССМ			
					Схема структурна компонентів програмного забезпечення	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Скорик Р.О.				Аркуш 1	Аркушів 2	
Перевірив		Поперешняк С.В.						
Т. контр.								
Н. контр.		Гловченко М.М			Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Затвердив		Жаріков Е.В.						

КПІ.ІП-1224.045490.06.99.ССМ



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Скорик Р.О.		
Перевірив		Попершняк С.В.		
Т. контр.				
Н. контр.		Гловченко М.М		
Затвердив		Жаріков Е.В.		

КПІ.ІП-1224.045490.06.99.ССМ

Схема структурна компонентів програмного забезпечення

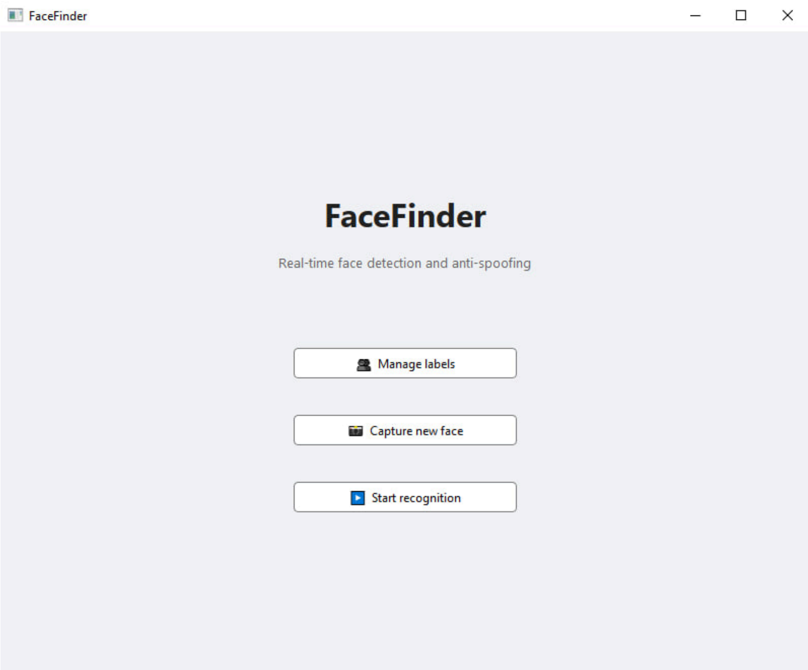
Літера	Маса	Масштаб

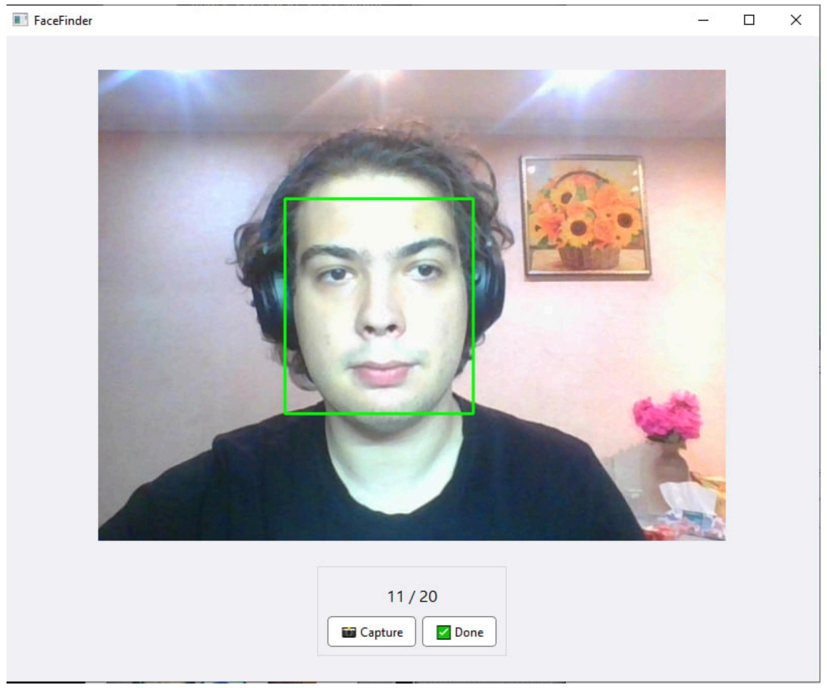
Аркуш 2 | Аркушів 2

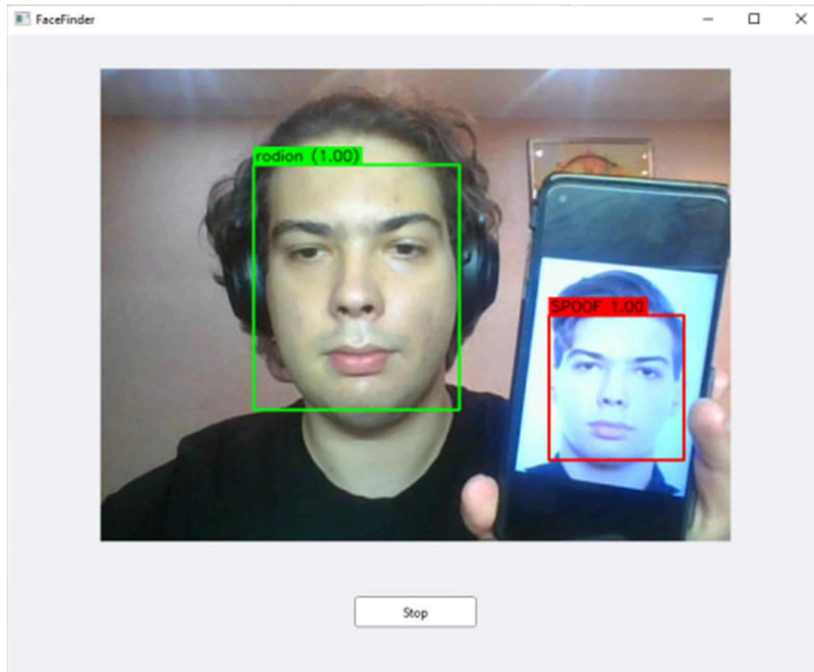
Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень

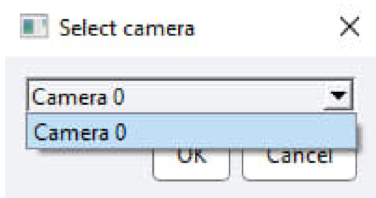
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12

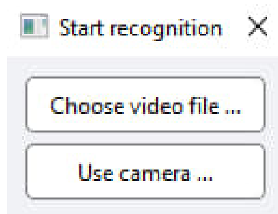
КПІ.ІП-1224.045490.06.99.КЕ

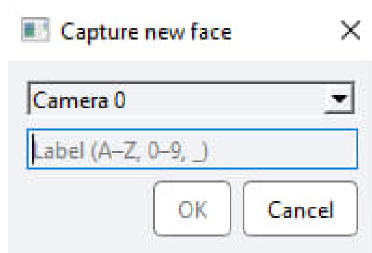


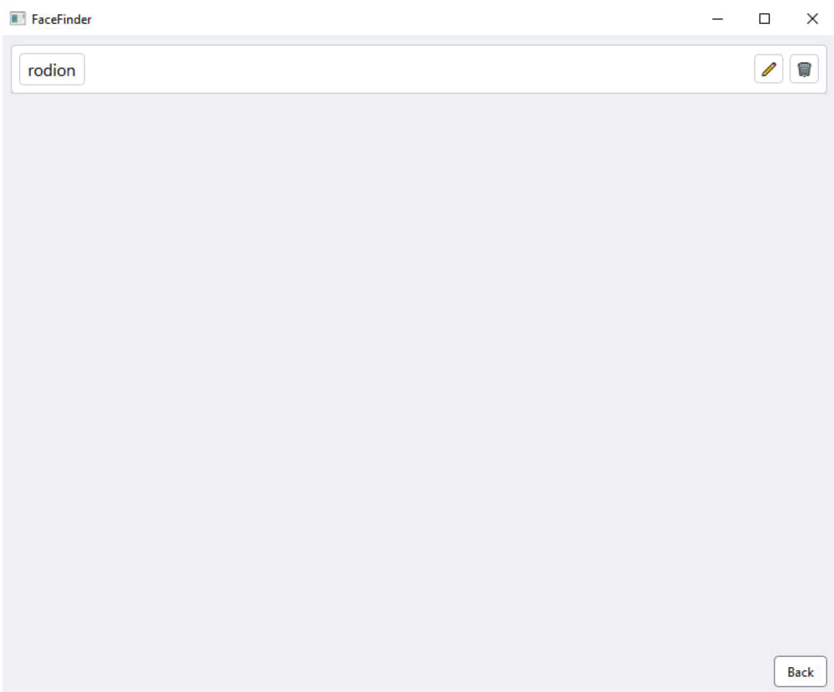












Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-1224.045490.06.99.КЕ		
Розробив	Скорик Р.О.				Креслення вигляду екранних форм	Літера	Маса	Масштаб
Перевірив	Поперешняк С.В.					Аркуш 1	Аркушів 1
Т. контр.					Застосунок для розпізнавання облич у відеопотоці з детекцією штучно змінених зображень	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	
Н. контр.	Гловченко М.М						
Затвердив	Жаріков Е.В.						