МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

КОВАЛЬКОВСЬКИЙ Віталій Віталійович

Модуль розпізнавання обличчя на основі глибокого навчання / Module for face recognition based on deep learning

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
В. В. Ковальковський

Науковий керівник:
д.т.н., доцент Х.В. Ліп'яніна-Гончаренко

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

Тернопіль – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
КОВАЛЬКОВСЬКИЙ Віталій Віталійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Модуль розпізнавання обличчя на основі глибокого навчання / Module for face recognition based on deep learning
керівник роботи к.т.н., доцент Х.В. Ліп'яніна-Гончаренко
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- проаналізувати сучасні алгоритми розпізнавання облич та емоцій і виявити їхні обмеження;
- сформувати інформаційну модель даних і підготувати репрезентативні набори зображень (VGGFace2, FER-2013);
- розробити алгоритм детекції та верифікації облич на базі моделі FaceNet із бібліотеки DeepFace;
- реалізувати CNN-класифікатор для визначення емоцій за виразом обличчя;
- інтегрувати створені алгоритми у веб-додаток Django з інтерфейсом захоплення та аналізу зображень;
- провести експериментальне тестування модуля й оцінити точність (accuracy, F1-score) та час обробки
5. Перелік графічного матеріалу в роботі:
- схема роботи веб-додатку;
- алгоритм роботи модуля;
- діаграма роботи Siamese Networks.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2025 р.	

Студент_____ В. В. Ковальковський
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Х. В. Ліп’яніна-Гончаренко
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Модуль розпізнавання облич на основі глибокого навчання» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 72 сторінок і містить 13 ілюстрацій, 2 таблиці, 2 додатки та 25 використаних джерел.

Метою роботи є розробка веб-модулю, що забезпечує автоматичне розпізнавання особи та класифікацію її емоцій на основі методів глибокого навчання, придатний для роботи в реальному часі.

Методами розроблення обрано методи комп'ютерного зору (для виявлення та обробки зображень облич), метод глибокого навчання (для побудови векторних ознак та класифікації емоцій), метод аналізу даних (для тестування та оцінювання точності роботи модуля), а також метод інтеграції програмних компонентів (для реалізації повноцінного веб-застосунку на основі фреймворку Django).

У результаті роботи розроблено веб-модуль, що дозволяє здійснювати ідентифікацію особи шляхом порівняння зображень, а також проводити базовий емоційний аналіз за допомогою попередньо навчених моделей. Система підтримує роботу з локальними файлами, зображеннями за URL-адресою та зображеннями з вебкамери. Проведено експериментальне тестування, результати якого підтверджують ефективність обраних підходів у прикладних умовах.

Ключові слова: РОЗПІЗНАВАННЯ ОБЛИЧ, ГЛИБОКЕ НАВЧАННЯ, ЕМОЦІЙНИЙ АНАЛІЗ, НЕЙРОННА МЕРЕЖА, КОМП'ЮТЕРНИЙ ЗІР, ВЕБ-ЗАСТОСУНОК.

ABSTRACT

The qualification thesis titled "Module for face recognition based on deep learning", submitted for the Bachelor’s degree in Computer Science, academic program "Computer Science", consists of 72 pages and includes 13 figures, 2 tables, 2 appendices, and 25 referenced sources.

The aim of the thesis is to develop a web module that enables automatic face recognition and emotion classification based on deep learning methods, suitable for real-time operation.

The methods used include computer vision techniques (for face detection and image processing), deep learning methods (for feature vector construction and emotion classification), data analysis methods (for testing and evaluating the module’s accuracy), as well as software integration methods (for implementing a complete web application using the Django framework).

As a result of the work, a web module was developed that performs identity verification by comparing facial images, and provides basic emotional analysis using pre-trained models. The system supports input from local files, URL-based images, and webcam captures. Experimental testing confirmed the effectiveness of the chosen approaches in applied conditions.

Keywords: FACE RECOGNITION, DEEP LEARNING, EMOTION ANALYSIS, NEURAL NETWORK, COMPUTER VISION, WEB APPLICATION.

ЗМІСТ

ВСТУП

У сучасну епоху стрімкого розвитку технологій штучного інтелекту та комп'ютерного зору технології розпізнавання облич знаходять широке застосування у сфері безпеки, ідентифікації користувачів та аналізу емоцій. Використання методів глибокого навчання дозволяє автоматизувати процеси, підвищити точність розпізнавання та забезпечити високий рівень надійності систем.

Розпізнавання облич є однією з ключових задач у сфері комп'ютерного зору, яка активно розвивається завдяки застосуванню згорткових нейронних мереж (CNN) та інших методів глибокого навчання. Практично вирішеними є такі аспекти, як виявлення облич на зображеннях, базове порівняння облич для ідентифікації та визначення ключових характеристик (наприклад, вік, стать, емоції). Високу ефективність у цих задачах демонструють бібліотеки DeepFace, FaceNet, OpenCV та Dlib, які активно застосовуються у сфері безпеки, фінансових технологій, автоматизації процесів у підприємницькій діяльності.

Водночас існують прогалини знань, пов'язані з розпізнаванням облич у складних умовах, таких як низька якість зображень, різні ракурси, освітлення та часткове перекриття обличчя. Актуальним завданням є розробка моделей, здатних адаптуватися до змінних умов та забезпечувати стабільно високу точність розпізнавання.

Провідними компаніями у сфері розпізнавання облич є Microsoft, Google, Amazon (Amazon Rekognition), Facebook (Meta) та Apple, які активно впроваджують алгоритми глибокого навчання у свої продукти. Також вагомий внесок у розвиток технології зробили такі вчені, як Янн ЛеКун (Yann LeCun), Джеффрі Хінтон (Geoffrey Hinton) та Ендрю Енґ (Andrew Ng), які досліджували архітектури нейронних мереж і методи глибокого навчання.

Світові тенденції вирішення поставлених задач включають вдосконалення моделей глибокого навчання, зокрема їх оптимізацію для роботи на мобільних

пристроях та вбудованих системах. Активно досліджуються методи підвищення точності та швидкості обробки, а також питання етичності та конфіденційності використання технологій розпізнавання облич.

Попри досягнення в технологіях розпізнавання облич, традиційні системи ідентифікації все ще значною мірою покладаються на ручні методи перевірки, що призводить до помилок та затримок. Використання глибокого навчання та комп'ютерного зору має потенціал значно покращити ці процеси, забезпечуючи точні та ефективні рішення для ідентифікації особи та аналізу емоційного стану.

Розробка модуля розпізнавання облич на основі методів глибокого навчання, який дозволить здійснювати ідентифікацію особи та визначення її емоційного стану з високою точністю та ефективністю.

Отримані результати можуть бути використані в системах контролю доступу, цифровій ідентифікації, маркетингових дослідженнях, медичній діагностиці психоемоційного стану, а також у сфері безпеки та автоматизації бізнес-процесів.

Результати дослідження доповідалися на конференції "Інтелектуальні інформаційні технології в прикладних дослідження" [2] (додаток А).

Мета роботи - розробити веб-модуль, що забезпечує автоматичне розпізнавання особи та класифікацію її емоцій на основі методів глибокого навчання, придатний для роботи в реальному часі.

Завдання дослідження:

1. Проаналізувати сучасні алгоритми розпізнавання облич та емоцій і виявити їхні обмеження.

2. Сформувати інформаційну модель даних і підготувати репрезентативні набори зображень (VGGFace2, FER-2013).

3. Розробити алгоритм детекції та верифікації облич на базі моделі FaceNet із бібліотеки DeepFace.

4. Реалізувати CNN-класифікатор для визначення емоцій за виразом обличчя.

5. Інтегрувати створені алгоритми у веб-додаток Django з інтерфейсом захоплення та аналізу зображень.

6. Провести експериментальне тестування модуля й оцінити точність (accuracy, F1-score) та час обробки.

Предмет дослідження - процеси автоматизованого розпізнавання облич і класифікації емоцій у веб-середовищі.

Об’єкт дослідження - цифрові зображення й відеокадри з обличчями користувачів, що підлягають детекції, верифікації та емоційній інтерпретації.

Методи дослідження. У роботі використано згорткові нейронні мережі (FaceNet, ResNet-подібні CNN) з триплетною втратою для ембеддингу облич, алгоритми комп’ютерного зору (MTCNN-детектор, вирівнювання геометрії, нормалізація зображень), попередньо навчені моделі класифікації емоцій (FER-2013), а також статистичні метрики accuracy, F1-score та confusion matrix для оцінки якості; реалізацію виконано у веб-фреймворку Django з використанням бібліотек OpenCV і DeepFace, а продуктивність перевірено експериментально шляхом вимірювання латентності й пропускної здатності системи.

1 СТАН ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному світі, де технології розвиваються швидкими темпами, система розпізнавання обличчя на основі глибокого навчання стає ключовим елементом у багатьох сферах — від забезпечення безпеки до покращення користувацького досвіду. Виклики, що постають перед такими системами, включають не лише необхідність точного і швидкого порівняння зображень облич, а й забезпечення високої продуктивності та ефективності в реальному часі. Традиційні методи аутентифікації, такі як введення пароля або використання фізичних карток, вже не відповідають вимогам сучасного світу з точки зору зручності та безпеки, оскільки мають ряд обмежень, які можна подолати за допомогою технологій на основі глибокого навчання.

Застосування технологій комп'ютерного зору та глибокого навчання відкриває нові можливості для покращення системи розпізнавання обличчя та емоцій в різних сферах. Завдяки здатності автоматично ідентифікувати обличчя користувачів та визначати їх емоційний стан в реальному часі, ці технології здатні значно покращити безпеку, зручність користувацького досвіду та ефективність роботи системи. Актуальність дослідження підтверджується також високим попитом на покращення методів аутентифікації та аналізу емоцій у реальному часі. Особливої уваги потребують розробки, які включають впровадження сучасних методів, таких як розпізнавання обличчя для ідентифікації користувачів та визначення емоцій, що дозволяє створити більш інтерактивні, адаптивні та безпечні системи. Ці інновації забезпечать більш точні та швидкі рішення, що відповідають вимогам сучасних технологій.

На сьогоднішній день вчені та інженери розробили кілька підходів до автоматизації розпізнавання обличчя та емоцій за допомогою сучасних технологій глибокого навчання. Найбільш поширеними є методи, засновані на глибоких нейронних мережах та комп'ютерному зорі, зокрема ті, що використовують архітектури, подібні до DeepFace[3]. Ці методи значно підвищують точність та

швидкість розпізнавання обличчя і емоцій у реальному часі. Проте існують деякі обмеження, зокрема залежність від якості вхідних зображень та потреба в достатньо потужному обладнанні для обробки великих обсягів даних. Наприклад, високі вимоги до освітлення та чіткості зображень можуть обмежувати ефективність розпізнавання у складних умовах, що потребує подальшої оптимізації алгоритмів.

На сьогодні вчені та інженери розробили кілька підходів до автоматизації розпізнавання обличчя та емоцій. Найбільш поширеними є системи, які використовують нейронні мережі та технології комп'ютерного зору для порівняння зображень облич та визначення емоцій. Зокрема, глибокі нейронні мережі, такі як згорткові нейронні мережі (CNN)[4], продемонстрували ефективність у задачах розпізнавання облич і емоцій. Ці методи значно покращують точність і швидкість розпізнавання обличчя у реальному часі, але також потребують значних обчислювальних потужностей та спеціалізованого програмного забезпечення для обробки зображень. Системи, побудовані на основі таких моделей, як ResNet, EfficientNet здатні до адаптації до різних умов освітлення, кута огляду обличчя і навіть варіацій у виразах емоцій.

Основні виклики в цій сфері стосуються адаптивності моделей та їх здатності працювати в реальних умовах. Існуючі системи часто потребують значних обчислювальних ресурсів і не завжди забезпечують достатню швидкість та точність через різноманітність продукції.

З огляду на це, важливим є подальший розвиток платформ розпізнавання облич на основі комп'ютерного зору, які здатні адаптуватися до різних умов та ефективно працювати з різноманітними образами облич. Така система, що поєднує машинне навчання та комп'ютерний зір, не лише покращить точність ідентифікації осіб, але й забезпечить можливість проактивного моніторингу емоційного стану та поведінки користувачів, що підвищить ефективність безпеки та зручність використання технологій, наприклад, у системах авторизації або аналізу емоцій у реальному часі.

1.2 Огляд і аналіз існуючих рішень

Для вивчення інтеграції та впливу комп'ютерного зору на розпізнавання облич та емоцій було проведено огляд літератури, що базується на аналізі академічних праць, прикладних досліджень та новітніх технологічних досягнень у галузі комп'ютерного зору та глибокого навчання. Огляд охоплював оцінку первинних та вторинних джерел, включаючи дослідження, опубліковані в журналах з тематики штучного інтелекту, машинного навчання та розпізнавання образів, а також звіти про практичні впровадження таких систем у різних сферах, зокрема в безпеці, медицині та маркетингу.

Сучасні системи розпізнавання обличчя на основі глибокого навчання стали стандартом у багатьох застосунках. Одна з найбільш відомих технологій — FaceNet, яка використовує архітектуру на основі глибоких згорткових нейронних мереж для створення векторних представленнь облич. Вона дозволяє не лише порівнювати зображення облич, але й проводити класифікацію та пошук за допомогою методу векторного подібності. FaceNet є основою для численних програм, включаючи системи безпеки, такі як Face ID від Apple, що демонструють високу ефективність в реальних умовах. Система здатна досягти високої точності навіть у випадках, коли обличчя має часткові перекриття або знаходиться в різних позах.

Іншим важливим рішенням є бібліотека DeepFace, яка надає готові моделі для розпізнавання облич та їх порівняння. Вона об'єднує декілька популярних нейронних мереж, таких як VGG-Face, Google FaceNet, OpenFace, DeepID, що дозволяє вибирати найкраще рішення для конкретних завдань залежно від необхідних вимог до точності та швидкості. Одним з головних переваг DeepFace є її здатність працювати на різних типах зображень, що дозволяє покращити адаптацію до реальних умов, таких як різні кути огляду, освітлення та вирази обличчя [5].

Крім стандартних завдань розпізнавання обличчя, одним із важливих напрямків є визначення емоцій за допомогою зображень обличчя. Це завдання

стало популярним у багатьох сферах, включаючи психоаналітику, маркетинг, освіту та розваги. Визначення емоцій часто здійснюється за допомогою спеціалізованих алгоритмів, здатних класифікувати емоції на основі виразів обличчя.

Одним із підходів до розпізнавання емоцій є використання Convolutional Neural Networks (CNN) для вивчення основних рис обличчя, які можуть сигналізувати про емоційний стан людини. Багато сучасних систем для цього завдання базуються на великих датасетах, таких як FER-2013, які містять тисячі зображень облич з різними емоціями, що дозволяє моделі вчитися на різноманітних прикладах.

Для застосування в реальному житті вважаються корисними гібридні моделі, які одночасно можуть виконувати завдання як розпізнавання обличчя, так і визначення емоцій. Наприклад, DeepFace можна адаптувати для додаткових завдань класифікації емоцій, що робить її універсальним інструментом для різноманітних прикладних задач.

Не дивлячись на досягнення в області розпізнавання обличчя та емоцій, існують ряд викликів, з якими стикаються ці системи:

- змінні умови освітлення та поза: більшість систем стикаються з труднощами при обробці зображень, де обличчя частково перекриті, виявляються в різних позах або освітлені недостатньо;

- точність в реальних умовах: системи, що працюють в лабораторних умовах, часто не досягають такої ж високої точності в реальних умовах, де обличчя можуть бути неідеальними або зіпсованими;

- необхідність великих обчислювальних потужностей: деякі моделі потребують значних обчислювальних ресурсів, що може бути обмеженням для використання у реальних системах з обмеженими можливостями;

- приватність і безпека: використання технологій розпізнавання обличчя та емоцій у публічних просторах викликає занепокоєння з приводу порушення приватності, що потребує розробки етичних та правових норм.

1.3 Постановка задачі

У сучасних системах безпеки та автоматизації великої уваги заслуговує розпізнавання обличчя, яке є важливою складовою частиною технологій ідентифікації та аналізу. Одним із основних напрямків досліджень у цій галузі є використання глибокого навчання для створення точних і надійних систем розпізнавання облич, які можуть працювати в реальному часі за різних умов. Розпізнавання емоцій за допомогою обличчя також стає важливим завданням у психології, маркетингу та інших сферах, де важливо не тільки ідентифікувати особу, а й зрозуміти її емоційний стан.

Мета роботи - розробити веб-модуль, що забезпечує автоматичне розпізнавання особи та класифікацію її емоцій на основі методів глибокого навчання, придатний для роботи в реальному часі.

Завдання дослідження:

1. Проаналізувати сучасні алгоритми розпізнавання облич та емоцій і виявити їхні обмеження.

2. Сформувати інформаційну модель даних і підготувати репрезентативні набори зображень (VGGFace2, FER-2013).

3. Розробити алгоритм детекції та верифікації облич на базі моделі FaceNet із бібліотеки DeepFace.

4. Реалізувати CNN-класифікатор для визначення емоцій за виразом обличчя.

5. Інтегрувати створені алгоритми у веб-додаток Django з інтерфейсом захоплення та аналізу зображень.

6. Провести експериментальне тестування модуля й оцінити точність (accuracy, F1-score) та час обробки.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

2.1 Загальна структура програмного модуля

Розроблений програмний модуль є частиною веб-додатку, який забезпечує можливість захоплення зображень, їх аналізу та перевірки за допомогою технологій комп'ютерного зору та глибокого навчання. Модуль базується на фреймворку Django і включає кілька основних компонентів, кожен з яких виконує конкретну задачу в загальній системі.

Загальна схема функціонування застосунку базується на шаблоні Model–Template–View, де кожен компонент виконує свою роль у логіці обробки запитів. Користувач звертається до веб-інтерфейсу, який спрямовує запит до відповідного URL-маршруту. Після цього обробку бере на себе представлення (view), яке за потреби взаємодіє з моделями або шаблонами й формує відповідь. URL-шляхи спрямовують запити до Views, які можуть звертатися до моделей (наприклад, при збереженні даних) або повертати HTML-шаблони, наповнені динамічним вмістом. Такий підхід дозволяє централізовано керувати логікою й одночасно забезпечувати розділення відповідальностей у коді. На рисунку 2.1 подано загальну схему архітектури додатку, що демонструє взаємодію між основними компонентами Django.

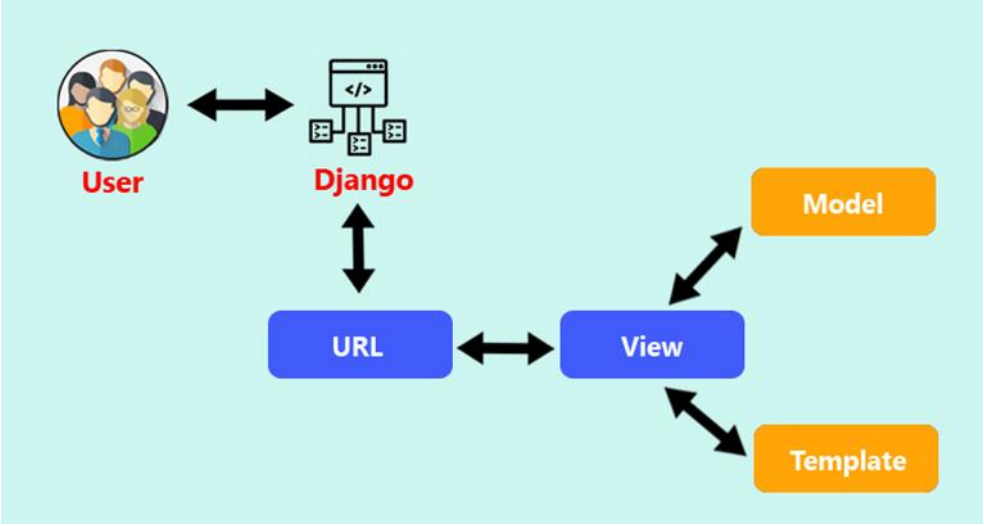

Рисунок 2.1 - Схема архітектури веб-додатку на базі Django

Одним із ключових елементів програми є механізм захоплення зображень з веб-камери користувача. Це реалізовано через функцію capture_image, яка використовує бібліотеку OpenCV[7] для отримання зображення з камери. Зображення зберігається в статичній директорії на сервері, що дозволяє використовувати його для подальших операцій, таких як перевірка або аналіз.

Одразу після захоплення зображення можна здійснити його перевірку за допомогою іншої функції — verify_image. Ця функція здійснює порівняння захопленого зображення з тестовим фото (наприклад, фотографією, що була завантажена раніше), використовуючи бібліотеку DeepFace. DeepFace дозволяє порівнювати два зображення і визначати, чи належать вони одній особі. Результат порівняння повертається у вигляді JSON-відповіді, яка містить інформацію про те, чи були зображення схожими. У разі виникнення помилки, користувач отримує відповідне повідомлення про проблему.

Іншою важливою функцією є можливість захоплення тестового зображення через функцію capture_test_photo, яка дозволяє зберегти зображення для подальшого використання в процесі перевірки. Цей процес є корисним, якщо необхідно завантажити зображення, яке буде використовуватися як база для порівняння в майбутньому.

Модуль також включає функцію home, яка відповідає за рендеринг домашньої сторінки веб-додатку. Це основна точка входу для користувачів, де вони можуть вибрати потрібну операцію і переходити до іншого розділу додатку. Користувачі також можуть перейти до стартової сторінки через функцію start, яка надає доступ до всіх можливостей системи, починаючи з перевірки зображень і закінчуючи завантаженням тестових фото.

Модуль також має функцію faceAnalise, що забезпечує аналіз завантажених зображень. Користувач може завантажити фото через форму на веб-сторінці або вказати URL зображення для його завантаження з Інтернету. Після завантаження фото програма здійснює його аналіз за допомогою DeepFace. Це включає визначення таких параметрів, як вік, стать та емоції особи на фото. Результати аналізу відображаються на відповідній веб-сторінці, де користувач може

переглядати отриману інформацію. У разі виникнення помилки, наприклад, якщо не вдалося завантажити зображення або провести аналіз, система повертає користувачеві детальне повідомлення про проблему.

Для організації доступу до різних частин програми та функцій використовуються URL-маршрути. У файлі urls.py визначено кілька маршрутів для доступу до основних функцій програми. Це включає домашню сторінку, сторінку аналізу обличчя, а також функції для захоплення та перевірки зображень. Завдяки цьому користувач може швидко перейти до потрібної функції через інтуїтивно зрозумілий інтерфейс.

Для обробки файлів, які завантажуються користувачами, а також збереження медіа-файлів, система використовує стандартні функції Django для роботи з медіа-контентом. Усі файли зберігаються в спеціально визначених папках, а доступ до них здійснюється через відповідні URL-адреси. Зображення, що захоплюються через веб-камеру або завантажуються користувачем, зберігаються в директорії static, що дозволяє зручно управляти файлами.

Крім того, для покращення досвіду користувачів, результати аналізу зображень супроводжуються зображеннями, які також відображаються в результатах на веб-сторінці. Цей підхід дозволяє користувачам не лише отримувати аналітичні дані, але й бачити фото, що було проаналізовано, що робить взаємодію з системою більш зручною та зрозумілою.

Модуль також взаємодіє з іншими частинами системи за допомогою шаблонів Django, що дозволяють створювати динамічні сторінки з результатами обробки. Шаблони надають гнучкість в налаштуванні вигляду веб-сторінок та їх контенту, що дозволяє адаптувати систему під конкретні вимоги користувачів або специфікації проекту.

Таким чином, розроблений програмний модуль поєднує потужні можливості бібліотеки DeepFace для глибокого навчання та комп'ютерного зору з гнучкістю та зручністю веб-фреймворку Django, що забезпечує ефективну обробку та аналіз зображень для розпізнавання облич і емоцій. Есі ці елементи разом створюють

інтегровану систему, здатну вирішувати завдання розпізнавання облич і емоцій в реальному часі через веб-додаток.

2.2 Опис математичних та алгоритмічних моделей розпізнавання облич та емоцій

У розпізнаванні облич використовуються різні математичні моделі та алгоритми, кожен з яких має свої специфічні особливості та переваги. Оскільки у моїй роботі використовується бібліотека DeepFace, яка базується на глибоких нейронних мережах, можна розглянути основні етапи, які включають детекцію облич, порівняння та аналіз емоцій, і детально описати, як ці елементи реалізуються в рамках мого проекту.

Детекція облич є першим і важливим етапом у будь-якій системі розпізнавання облич, оскільки необхідно правильно локалізувати обличчя на зображенні або відео перед подальшим аналізом. У DeepFace для цієї задачі використовуються глибокі нейронні мережі, зокрема Convolutional Neural Networks (CNN), які є основою для більшості сучасних систем розпізнавання зображень.

Глибокі нейронні мережі (CNN) є основним методом, що використовується в DeepFace для виявлення облич. CNN дозволяють мережам вивчати різноманітні ознаки, такі як контури обличчя, текстури шкіри, розміри очей та інших частин обличчя, що допомагає правильно локалізувати обличчя на зображеннях, навіть коли вони мають різні кути нахилу або змінене освітлення. CNN розбивають зображення на невеликі ділянки (фільтри) та аналізують їх на кількох рівнях абстракції, що дозволяє мережі ефективно працювати навіть в умовах складних сценаріїв, наприклад, при зміні виразу обличчя чи частковому закритті обличчя.

Головним принципом роботи CNN є використання згорткової операції, яка дозволяє мережі послідовно аналізувати різні частини зображення, знаходячи характерні особливості, такі як контури обличчя, розташування очей, носа, губ тощо. Математично згортка виражається у вигляді:

$$S(i,j) = \sum_m \sum_n I(i+m, j+m) * K(m,n) \,, \qquad (2.1)$$

де S(i, j) — значення елемента результату згортки у позиції з координатами (i, j), що отримується внаслідок накладення ядра згортки на відповідну область зображення;

I(i + m, j + m) — значення інтенсивності пікселя вхідного зображення у позиції з координатами (i + m, j + m), яка відповідає зміщенню згідно з поточним положенням ядра;

K(m, n) — значення елемента ядра згортки (фільтра) у позиції з координатами (m, n), яке визначає вагу пікселя під час обчислення згортки;

m, n — індекси, що відповідають координатам елементів ядра згортки та визначають розмір області зображення, яка залучається до обчислення згортки для кожного положення.

Ця функція дозволяє моделі зберігати лише значущі ознаки, відсіюючи непотрібну інформацію. Після кількох рівнів згортки мережа отримує компактне представлення обличчя, що містить найважливіші характеристики для подальшого порівняння.

На рисунку 2.2 зображено Метод каскадів Хаара [8] який є класичним і одним з найперших методів для виявлення облич. Цей метод використовує спеціальні фільтри для визначення контурів, текстур і інших характеристик, що можуть вказувати на присутність обличчя на зображенні.

За допомогою алгоритму Adaboost [9] цей метод вибирає найзначущіші ознаки для класифікації, що дозволяє швидко і ефективно знаходити обличчя на зображеннях, хоча його точність може знижуватися у складних умовах, таких як зміни освітлення чи кута нахилу.

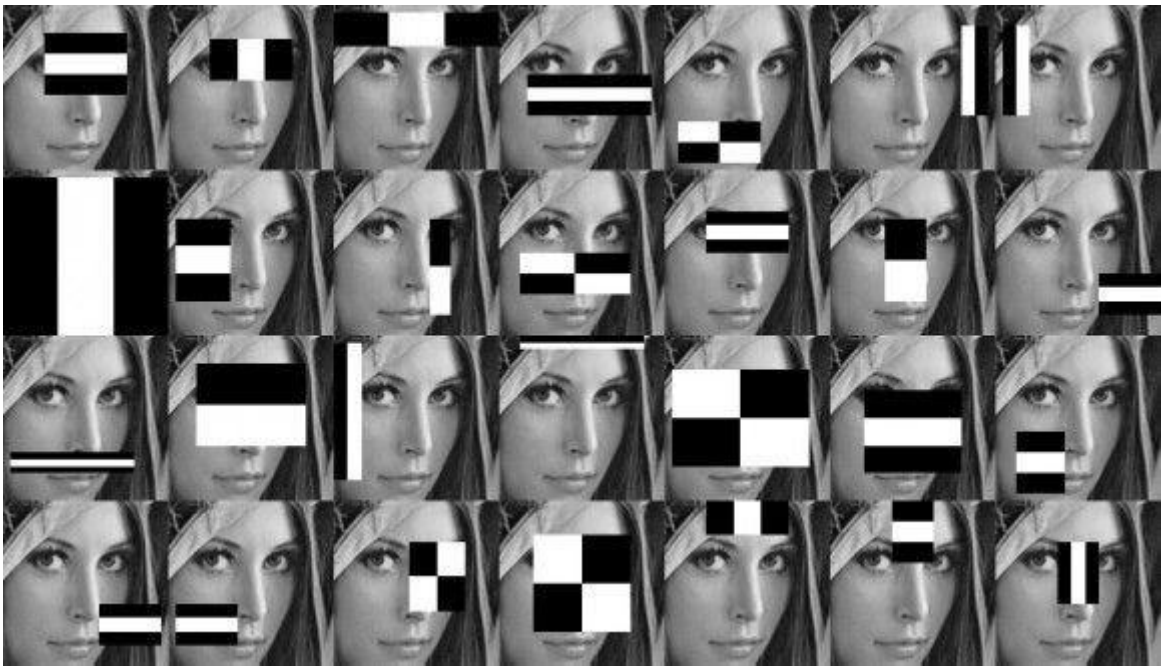

Рисунок 2.2 – Метод каскадів Хаара

Після того як обличчя було виявлено, наступним кроком є порівняння з іншими зображеннями для визначення, чи належать вони одній і тій самій особі. В системі DeepFace для цього використовуються більш складні моделі, зокрема Siamese Networks та метод Triplet Loss.

Siamese Networks[10] — це тип нейронних мереж, який призначений для порівняння двох зображень. Основною метою таких мереж є вивчення відмінностей між обличчями, шляхом обчислення їх векторних представлень (ембеддингів) і порівняння відстані між цими векторами. Вектор, який генерується мережею, є унікальним числовим представленням обличчя, яке можна використовувати для порівняння з іншими векторами облич. Якщо відстань між цими векторами маленька, ймовірно, це одне і те саме обличчя. В іншому випадку, якщо відстань велика, це різні обличчя. Така архітектура дозволяє досягати високої точності у задачах, пов'язаних із розпізнаванням облич. На рисунку 2.3 представлено діаграму роботи даної нейронної мережі.

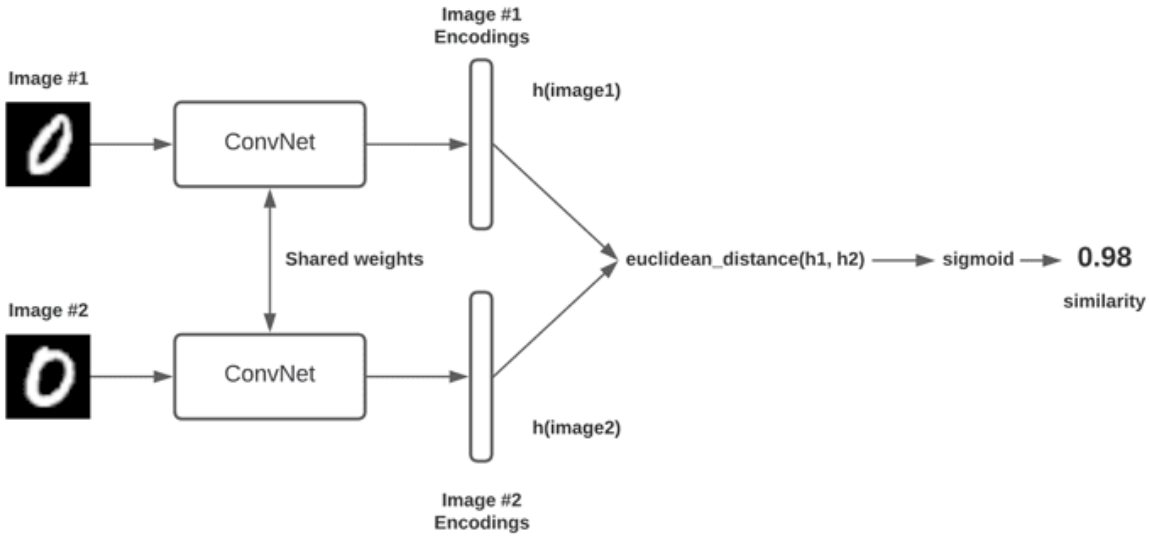

Рисунок 2.3 – Діаграма роботи Siamese Networks

Triplet Loss [11] є технікою, що дозволяє покращити результати порівняння за допомогою трьох зображень. Для цього використовуються три зображення: позитивне (той самий об'єкт), негативне (інший об'єкт) і ще одне зображення того ж об'єкта чи іншого. В процесі навчання мережа намагається максимізувати відстань між негативним прикладом і позитивним, а також мінімізувати відстань між позитивними прикладами. Це дозволяє моделі більш точно навчатися і відрізняти обличчя на зображеннях, навіть при незначних відмінностях.

У DeepFace для порівняння облич часто використовуються попередньо навчені моделі, такі як VGG-Face, Google FaceNet, DeepID та інші. Кожна з цих моделей використовує різні архітектури і підходи для розпізнавання облич, але всі вони базуються на використанні CNN для створення векторних представлень облич та їх порівняння. Вибір моделі залежить від того, яка з них дає найкращі результати для конкретної задачі.

У модулі також використовується можливість аналізу емоцій на основі облич. Для цього використовується частина алгоритмів, що працюють з DeepFace. Аналіз емоцій здійснюється шляхом визначення ключових ознак виразу обличчя, таких як посмішка, нахмурені брови, відкриті очі тощо. Ці ознаки використовуються для класифікації емоцій, таких як радість, сум, злість, страх, здивування, презирство та

інші. Для цього використовується нейронна мережа, яка аналізує вираз обличчя та відносить його до однієї з категорій: радість, сум, злість, страх, здивування тощо. Цей процес базується на класифікації ознак обличчя за допомогою Softmax-функції, яка перетворює набір чисел у ймовірності для кожного класу емоцій:

$$P(y_i) = \frac{e^{z_i}}{\Sigma_{j=1}^{n} e^{z_j}}, \tag{2.2}$$

де $P(y_i)$ — ймовірність приналежності обличчя до класу y_i;

z_i — вихідне значення для відповідного класу;

n — загальна кількість класів емоцій;

DeepFace застосовує вже навчені моделі для класифікації емоцій на основі аналізу облич. Це дозволяє отримувати точні результати без необхідності великого обсягу навчання, оскільки ці моделі були попередньо навчені на великих наборах даних з мітками емоцій.

2.3 Алгоритм розпізнавання обличчя на основі глибокого навчання

Алгоритм розпізнавання обличчя базується на використанні бібліотеки DeepFace і включає три основні етапи: детекцію обличчя, порівняння зображень та аналіз емоцій. Такий підхід забезпечує точне розпізнавання облич і дозволяє доповнити систему функцією визначення емоційного стану.

Першим етапом є детекція обличчя, що полягає у виявленні та локалізації обличчя на зображенні. Для цього в DeepFace застосовуються глибокі нейронні мережі, зокрема Convolutional Neural Networks (CNN), які дозволяють ефективно визначати обличчя на зображеннях навіть за наявності складних умов, таких як різні кути нахилу, зміни освітлення чи часткове перекриття.

CNN здатні вивчати ознаки, характерні для обличчя, зокрема контури, текстури шкіри, розміри очей, носа та губ. Мережа застосовує фільтри для

виявлення цих ознак, що дає можливість правильно локалізувати обличчя на зображенні.

Після того як обличчя було локалізовано, необхідно порівняти його з іншими зображеннями, щоб визначити, чи належать вони одній особі. Для цього в DeepFace застосовуються потужні моделі, що дозволяють порівнювати зображення та визначати схожість між ними.

Одним із таких методів є порівняння векторних представлень обличчя. Кожне обличчя перетворюється на числовий вектор (ембеддинг), що є унікальним для кожної особи. Порівнюючи ці вектори, можна визначити, чи належать обличчя одній людині. Якщо відстань між векторами мала, це означає, що це одне і те саме обличчя. Якщо ж відстань велика — різні обличчя.

У DeepFace також використовуються попередньо навчені моделі для порівняння облич. Зокрема, використовуються моделі, які базуються на ResNet [12], що є глибокими згортковими мережами, здатними навчатися на великих обсягах даних і показувати високу точність у розпізнаванні облич. Вибір моделі залежить від конкретної задачі та умов, в яких здійснюється розпізнавання.

Одним з ключових аспектів моєї роботи є аналіз емоцій, який дає змогу визначити емоційний стан людини на основі її виразу обличчя. Для цього використовуються додаткові алгоритми, що працюють з DeepFace. Моделі, навчені на великому обсязі даних, можуть класифікувати різні емоції, такі як радість, сум, злість, страх, здивування, презирство тощо.

Аналіз емоцій базується на виявленні певних ознак виразу обличчя, таких як посмішка, нахмурені брови, відкриті очі. Це дозволяє точно визначати емоції навіть у складних умовах, коли обличчя є частково перекритим чи мають місце зміни освітлення.

Для забезпечення точності розпізнавання застосовуються різні моделі, які мають свої особливості і переваги:

- ResNet — це одна з найсучасніших архітектур для глибоких згорткових мереж, що дозволяє ефективно навчатися на великих наборах даних, зберігаючи високу точність при розпізнаванні облич;

- VGG-Face, Google FaceNet, DeepID — ці попередньо навчені моделі мають свої особливості в архітектурі та підходах до навчання, але всі вони базуються на згорткових нейронних мережах (CNN). Вибір моделі залежить від вимог до швидкості та точності. На рисунку 2.4 зображений алгоритм роботи модуля.

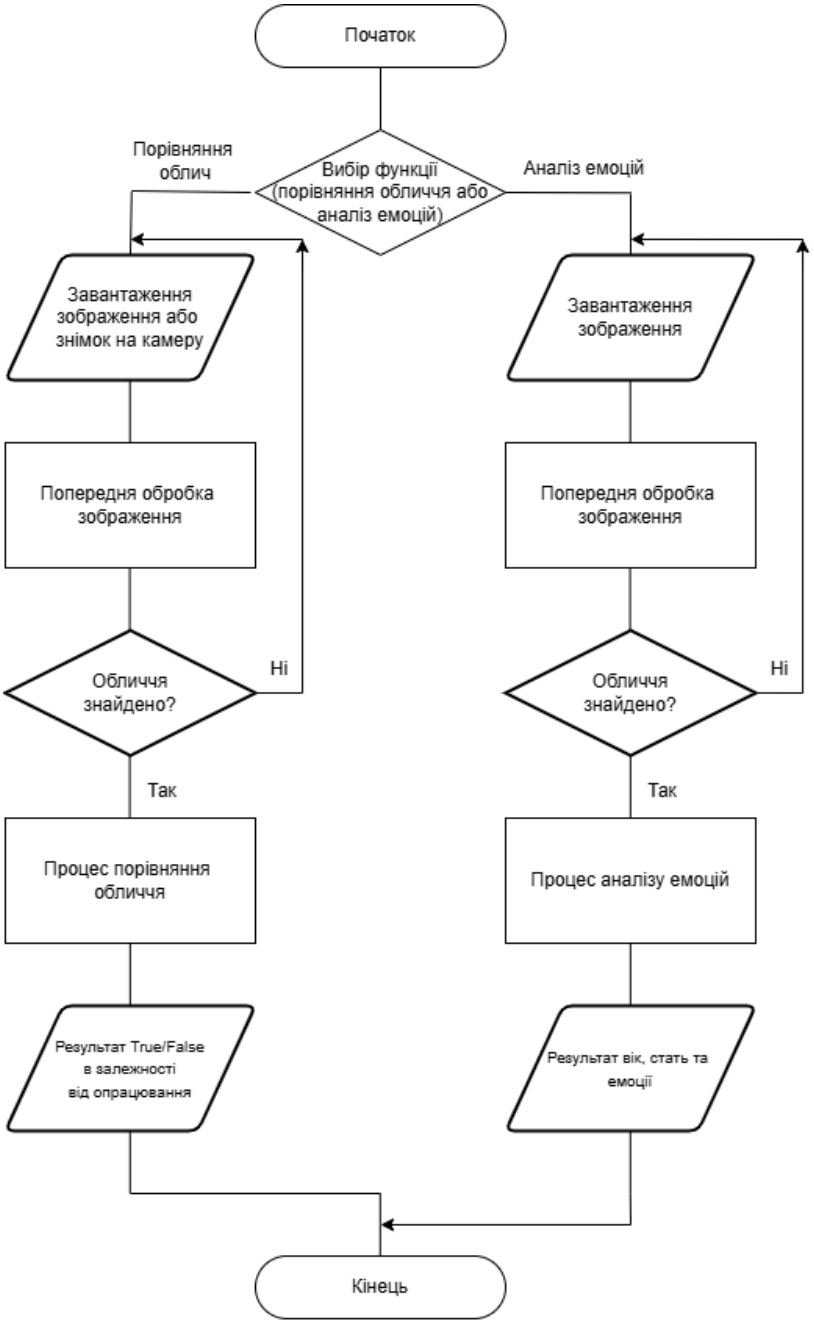

Рисунок 2.4 – Алгоритм роботи модуля.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Дослідження набору даних

Підготовка й вибір даних для задач комп'ютерного зору відіграє ключову роль у формуванні точності, стабільності та надійності моделей глибокого навчання. Якість, структурованість і репрезентативність таких даних безпосередньо впливають на здатність системи правильно ідентифікувати риси обличчя, розпізнавати емоції та формувати достовірні результати. У цьому розділі проаналізовано структуру й походження датасетів, на яких ґрунтуються попередньо навчені моделі, що використовуються в реалізованій системі.

Основна модель, задіяна у верифікації облич, — це FaceNet, одна з найпоширеніших архітектур у задачах порівняння біометричних ознак. Навчання цієї моделі здійснювалося на датасеті VGGFace2 — великій колекції зображень, що охоплює понад 9 тисяч унікальних осіб із різними емоціями, позами та умовами освітлення. Відповідно до офіційних джерел, VGGFace2 відзначається високим рівнем варіативності зображень, що дозволяє моделі формувати стійкі ембедінги обличчя, незалежні від конкретного ракурсу чи експресії. Зокрема, на еталонному тестовому наборі LFW (Labeled Faces in the Wild) FaceNet [13] демонструє точність на рівні 99.63 %, що свідчить про здатність моделі до генералізації навіть за змінених вхідних умов. На рисунку 3.1 зображено приклад зображень із датасету VGGFace2.

Рисунок 3.1 - Приклади зображень із датасету VGGFace2

Водночас важливим компонентом є аналіз емоцій, для якого в DeepFace реалізовано окрему модель класифікації, натреновану на наборі FER-2013. Цей датасет містить близько 35 тисяч зображень облич у сірій шкалі (рисунок 3.2), розподілених за семи базовими емоційними станами: гнів, страх, сум, радість, здивування, відраза й нейтральність. Особливістю FER-2013 є наявність великої кількості реальних, неідеалізованих виразів, що робить його відповідним для прикладних задач. Модель, що використовується в DeepFace, дозволяє визначити не лише домінантну емоцію, а й додаткові параметри — орієнтовний вік і стать, що також були використані в реалізованому модулі.

Рисунок 3.2 - Структура класів емоцій у датасеті FER-2013

Попри ефективність, важливо враховувати і обмеження. Так, моделі, навчені на FER-2013, у середньому досягають точності класифікації в межах 65–75 %, залежно від реалізації. Найвищу точність спостерігається при класифікації таких емоцій, як радість, сум і нейтральність. У той час як більш неоднозначні емоції — на кшталт відрази чи страху — частіше плутаються між собою через схожість мімічних проявів. Водночас модель FaceNet на основі VGGFace2, навіть за наявності шуму або зміненого кута зйомки, демонструє високу стійкість у верифікації, що підтверджено як у дослідницьких тестах, так і в межах проведеного практичного тестування. Узагальнені числові показники точності моделей, що використовуються у реалізованій системі, наведено в таблиці 3.1.

Таблиця 3.1 - Показники точності моделей FaceNet і FER-2013

Модель	Датасет	Кількість класів	Середнє (%)	Кращі класи (%)	Слабші класи (%)
FaceNet	VGGFace2	1 (верифікація)	99,63	99,8	98,5
FER-2013	FER-2013	7 (емоції)	70	80	55

У реалізованій системі використання саме цих моделей дозволяє працювати з широким спектром вхідних даних — як завантажених з ПК, так і зображень із веб-камери або онлайн-ресурсів. Завдяки високій якості навчальних наборів і гнучкості самих архітектур, стало можливим уникнути тривалого етапу підготовки та навчання власної моделі, зосередившись на інтеграції та адаптації результатів до прикладної задачі. Таким чином, ґрунтовне розуміння особливостей вихідних даних дозволяє не лише пояснити поведінку системи в конкретних сценаріях, а й обґрунтувати її переваги та потенційні обмеження.

3.2 Реалізація функціоналу розпізнавання облич та емоцій

Розробка програмного модуля, що здійснює автоматичне розпізнавання облич та визначення емоційного стану користувача, вимагала комплексного підходу як до вибору технологій, так і до побудови логіки обробки даних. Враховуючи складність поставлених задач, особливу увагу було приділено точності, швидкодії та масштабованості системи, яка повинна стабільно працювати з реальними зображеннями, знятими у різних умовах.

Основним завданням цього етапу стало створення веб-застосунку, який би дозволяв користувачу взаємодіяти з системою через інтуїтивно зрозумілий інтерфейс: завантажити або зробити фото, запустити процес аналізу, після чого отримати результат у вигляді ідентифікованої особи та її емоційного стану. Технічне

рішення мало забезпечити обробку вхідного зображення, виявлення облич на ньому, порівняння з еталонними зразками, а також класифікацію виявлених емоцій.

Щоб реалізувати таку систему, було обрано стек технологій, який поєднує інструменти веб-розробки, бібліотеки для комп'ютерного зору, а також готові моделі глибокого навчання. Одним із ключових рішень стало використання фреймворку DeepFace, який містить у собі повний набір функціоналу для роботи з обличчями: від попередньої обробки до порівняння ознак і аналізу емоцій. Завдяки цьому стало можливим уникнути створення та навчання власної моделі, що значно зекономило час на етапі реалізації та дало змогу зосередитися на інтеграції та тестуванні.

Разом з тим, для створення серверної частини було обрано середовище Django — популярний фреймворк мовою Python, який добре зарекомендував себе у побудові надійних веб-додатків. Саме він ліг в основу логіки обробки запитів, взаємодії з шаблонами, маршрутизації та загального керування поведінкою застосунку.

Перед початком реалізації було також проаналізовано перелік сторонніх бібліотек і модулів, які необхідні для забезпечення повної працездатності системи. Встановлення та налаштування середовища відбувалося з урахуванням усіх залежностей, зазначених у конфігураційному файлі requirements.txt, що є стандартною практикою для проєктів на Python. На рисунку 3.3 зображений вигляд чатсини файлу requirements.txt.

Після визначення загального підходу та вибору основних технологій постало питання побудови внутрішньої структури додатку, яка б забезпечувала зручну взаємодію між користувачем та алгоритмами обробки зображень. У цьому контексті було реалізовано класичну модель взаємодії серверної частини з інтерфейсом за допомогою фреймворку Django, який передбачає поділ на окремі складові: логіку, представлення та обробку даних.

Рисунок 3.3 – Файл requirements.txt.

Було створено окремий додаток Django, що виконує роль основного модуля для обробки зображень та керування маршрутизацією. У межах цього додатку реалізовано функції для приймання зображення, обробки даних за допомогою DeepFace, а також виведення результатів користувачу. Важливою особливістю стало чітке розділення логіки: з одного боку, інтерфейс взаємодіє із сервером за допомогою HTML-форм, а з іншого — сервер відповідає за повноцінний цикл аналізу зображення, від завантаження до класифікації.

Загальна схема функціонування застосунку базується на шаблоні Model-Template-View [14], де кожен компонент виконує свою роль у логіці обробки запитів. Користувач звертається до веб-інтерфейсу, який спрямовує запит до відповідного URL-маршруту. Після цього обробку бере на себе представлення (view), яке за потреби взаємодіє з моделями або шаблонами й формує відповідь.

URL-шляхи спрямовують запити до Views, які можуть звертатися до моделей (наприклад, при збереженні даних) або повертати HTML-шаблони, наповнені динамічним вмістом. Такий підхід дозволяє централізовано керувати логікою й одночасно забезпечувати розділення відповідальностей у коді. Наведений нижче фрагмент коду демонструє URL-шляхи всього модуля:

```
urlpatterns = [
    path('', start, name='start'),
    path('home/', home, name='home'),
    path('verify-image/', verify_image, name='verify_image'),
    path('capture-test-photo/', capture_test_photo, name='capture_test_photo'),
    path('faceAnalise/', faceAnalise, name='faceAnalise'),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Обробка завантажених зображень відбувається через вбудований механізм Django для роботи з медіа-файлами. У налаштуваннях веб-додатку передбачено збереження тимчасових зображень, які надходять від користувачів, після чого вони передаються до аналізу. Після завершення роботи зображення може бути видалене або збережене для подальшого використання — залежно від того, який сценарій передбачено на рівні логіки застосунку:

```
def capture_image(file_name="captured_image.jpg"):
    cap = cv2.VideoCapture(1)
    ret, frame = cap.read()
    if not ret:
        return None
    cap.release()
    file_path = f"{settings.STATICFILES_DIRS[0]}/{file_name}"
    cv2.imwrite(file_path, frame)
    return file_path
```

Такий підхід до архітектури дозволив досягти чіткого розмежування відповідальності між різними частинами системи: сервер обробляє запити та запускає алгоритми, а шаблони формують візуальну відповідь. У результаті було отримано структуру, яка легко масштабується, підтримується та розширюється, що особливо важливо при роботі з модулями комп'ютерного зору та штучного інтелекту.

Одним із ключових елементів роботи системи розпізнавання є етап отримання зображення, яке підлягатиме подальшому аналізу. Саме з цього моменту починається повний цикл обробки, і тому стабільність та зручність цього механізму мають важливе значення як з точки зору користувацького досвіду, так і технічної реалізації. У розробленій системі передбачено два основні варіанти отримання зображень: зйомка через веб-камеру в режимі реального часу та завантаження

готового зображення — як локального файлу з комп’ютера, так і через вказану користувачем URL-адресу.

Перший варіант — це інтеграція з камерою, яка дає змогу користувачу здійснити миттєве фотографування безпосередньо через браузер. Такий підхід виявився особливо зручним для функції порівняння обличчя: користувач може зробити фото на місці, після чого система автоматично обробляє його і виконує верифікацію. Захоплення зображення реалізовано за допомогою можливостей HTML5 та JavaScript, зокрема засобів WebRTC [15], які дозволяють задіяти камеру без потреби встановлення додаткових плагінів або програм. Отримане фото зберігається у тимчасову директорію, де з ним уже працює бекенд-система на Django:

```python
def capture_test_photo(request):
    test_photo_path = capture_image("testPhoto.jpg")
    if test_photo_path:
        return JsonResponse({"success": True})
    else:
        return JsonResponse({"error": "Не вдалося захопити зображення для testPhoto"}, status=400)

Windsurf: Refactor | Explain | Generate Docstring | ×
def home(request):
    return render(request, 'verifier/home.html')
```

Другий сценарій передбачає завантаження файлу з жорсткого диску користувача або ж введення посилання на зображення в Інтернеті. У разі локального файлу, завантаження здійснюється через стандартну HTML-форму, після чого Django автоматично обробляє запит і зберігає зображення у папці, визначеній у конфігурації веб-додатку як тимчасове сховище медіа-даних. У випадку введення URL-адреси система завантажує зображення програмно, використовуючи HTTP-запит, і проводить ті самі етапи підготовки, що й для файлів, отриманих через форму:

```html
<div class="form-container">
    <h1>Аналізування обличчя</h1>
    <form id="face-analysis-form" action="{% url 'faceAnalise' %}" method="POST" enctype="multipart/form-data" onsubmit="showLoader()">
        {% csrf_token %}
        <label for="photo">Виберіть зображення:</label>
        <input type="file" name="photo" id="photo" accept="image/*" class="file-input">

        <label for="photo_url">або введіть URL зображення:</label>
        <input type="text" name="photo_url" id="photo_url" placeholder="Введення URL зображення" class="file-input">

        <button type="submit">Аналізувати</button>
    </form>
```

Незалежно від джерела надходження, усі зображення проходять однакову процедуру попередньої обробки. На цьому етапі здійснюється перевірка допустимого формату файлу, зменшення роздільної здатності до стандартного розміру, конвертація зображення у колірний простір RGB, а також нормалізація кольорів для забезпечення єдності вхідних даних. Такий підхід дозволяє підготувати зображення до подальшої обробки алгоритмами аналізу.

У разі потреби застосовується автоматичне вирівнювання обличчя [16], яке покращує точність наступних етапів аналізу. Для цих цілей використовуються функції, надані бібліотекою DeepFace, а також базові засоби обробки зображень через OpenCV.

Це дозволяє гарантувати, що незалежно від початкового джерела, зображення буде приведене до уніфікованого вигляду, придатного для подальшої обробки алгоритмами глибокого навчання.

Такий підхід забезпечує стабільність результатів та зменшує кількість помилок, пов’язаних із різноманітністю вхідних даних — наприклад, надто великими файлами, невідповідними форматами або некоректною передачею кольору.

Реалізація подвійної системи завантаження також відкриває можливість застосовувати цю платформу в ширшому контексті — як у тестовому режимі для роботи з уже наявними зображеннями, так і в інтерактивному режимі для моментального отримання фото в польових умовах.

Після того як зображення проходить попередню обробку та зберігається у системі, наступним етапом стає безпосередньо процес розпізнавання обличчя. Саме цей крок лежить в основі верифікації, оскільки дає змогу визначити, чи належать два зображення одній і тій самій особі.

У межах реалізованого модуля для цього етапу було використано бібліотеку DeepFace, яка поєднує в собі сучасні методи комп’ютерного зору та переваги попередньо навчених моделей глибокого навчання.

Однією з головних переваг DeepFace є її доступність та універсальність. Бібліотека підтримує декілька популярних архітектур, що широко застосовуються у задачах розпізнавання облич. Для кожної з них уже виконано попереднє навчання на великих наборах зображень, що дозволяє використовувати їх без необхідності самостійно готувати або навчати моделі з нуля. Це суттєво спрощує інтеграцію таких технологій у реальні застосунки, особливо коли йдеться про проєкти з обмеженими ресурсами.

У реалізованій системі було обрано саме модель Facenet, яка забезпечує стабільну й надійну роботу при порівнянні зображень облич. Вибір цієї моделі зумовлений її здатністю формувати компактні, але водночас інформативні векторні представлення облич — так звані ембедінги [17]. Кожне обличчя, що обробляється моделлю, перетворюється у набір числових ознак, які є унікальними для конкретної особи. Це дозволяє зі значною точністю оцінювати схожість між двома зображеннями, навіть якщо вони були зроблені за різних умов, під іншим кутом або при зміненому освітленні.

Facenet показав хороші результати як у наукових дослідженнях, так і на практиці. Завдяки використанню триплетної втрати під час навчання, модель навчається "відчувати" схожість між зображеннями, тобто зменшувати відстань між ембедінгами однакових осіб та збільшувати її між різними. Такий підхід робить модель не лише точною, а й досить чутливою до деталей, що є важливим для верифікації облич у реальному середовищі.

У межах системи процес розпізнавання відбувається послідовно. Спершу DeepFace визначає, чи є на зображенні обличчя, та локалізує його. Далі зображення масштабовується й вирівнюється за ключовими точками — зокрема, очима, носом і ротовою порожниною. Після цього вирівняне обличчя передається на вхід моделі Facenet, яка обчислює ембедінг — тобто вектор чисел, що характеризує основні ознаки цього обличчя.

Коли ембедінги для двох зображень готові, між ними обчислюється відстань — зазвичай за косинусною або евклідовою метрикою. Якщо ця відстань менша за встановлений поріг, то вважається, що на обох зображеннях зображено одну й ту

саму людину. У протилежному випадку система робить висновок про те, що це різні особи. Поріг подібності задається вручну або експериментально визначається під час початкового налаштування системи. У модулі також передбачено можливість обійти сувору перевірку наявності обличчя, що дає змогу обробляти зображення навіть за наявності часткових дефектів або незвичних ракурсів.

Окрім розпізнавання особистості, важливою складовою функціоналу стало визначення емоційного стану людини на зображенні. Такий аналіз дає змогу отримати додаткову інформацію про користувача, що може бути корисним у багатьох практичних сценаріях — від маркетингових досліджень до систем навчання або онлайн-комунікації. З технічного боку, реалізація цього модуля значною мірою ґрунтується на можливостях бібліотеки DeepFace, яка дозволяє виконувати класифікацію емоцій автоматично, без потреби навчання власної моделі.

У межах розробленого застосунку користувачеві надається можливість завантажити зображення двома способами: або з локального пристрою, або за допомогою URL-посилання. У кожному з варіантів система обробляє зображення і готує його для подальшого аналізу. У разі вибору файлу з комп'ютера зображення передається через форму, після чого зберігається на сервері у спеціальній директорії. Якщо ж користувач вводить посилання на зображення, система надсилає HTTP-запит, перевіряє успішність відповіді та зберігає отримане фото локально для подальшої обробки [18].

Після того як зображення збережено, запускається основний етап аналізу. Для цього використовується метод analyze() із бібліотеки DeepFace, який дозволяє визначити одразу кілька параметрів — вік, стать та емоційний стан. У межах реалізації було зосереджено увагу саме на останньому. Після обробки зображення бібліотека повертає структурований результат, який містить імовірності для кожної з передбачених емоцій: радість, сум, злість, страх, здивування, відраза та нейтральний стан. Модель, яка використовується у DeepFace для цього аналізу, базується на наборі FER-2013 — одному з найпоширеніших датасетів для задач емоційної класифікації [19].

У реалізованому підході передбачено обробку лише першого обличчя, виявленого на зображенні. Такий підхід був обраний з міркувань практичності, оскільки в більшості випадків користувач аналізує саме одне обличчя — своє. Якщо обличчя виявлено успішно, то дані, отримані в результаті, відображаються у зручному форматі. Зокрема, окремо виводиться поточна емоція з найвищою ймовірністю, а також відсоткове співвідношення для кожного з можливих варіантів. Для статі додатково здійснюється округлення результатів для кращого візуального сприйняття. Нижче наведено фрагмент коду для аналізу емоцій:

```
def faceAnalise(request):
    result = None
    relative_path = None

    if request.method == 'POST':
        # Обробка завантаженого файлу
        if request.FILES.get('photo'):
            photo = request.FILES['photo']
            relative_path = os.path.join('uploads', photo.name)  # Відносний шлях до файлу
            file_path = os.path.join(settings.MEDIA_ROOT, relative_path)  # Абсолютний шлях


            with open(file_path, 'wb') as f:
                for chunk in photo.chunks():
                    f.write(chunk)

        # Обробка URL
        elif request.POST.get('photo_url'):
            photo_url = request.POST.get('photo_url')  # Отримуємо посилання на зображення
            try:
                response = requests.get(photo_url)
                if response.status_code == 200:
                    file_name = os.path.basename(photo_url.split("?")[0])  # Ім'я файлу без параметрів у URL
                    relative_path = os.path.join('uploads', file_name)
                    file_path = os.path.join(settings.MEDIA_ROOT, relative_path)

                    # Зберігаємо файл у папку media/uploads
                    with open(file_path, 'wb') as f:
                        f.write(response.content)
                else:
                    raise Exception("Не вдалося завантажити зображення за посиланням.")
            except Exception as e:
                result = {"error": str(e)}
                return render(request, 'verifier/faceAnalise.html', {'result': result, 'MEDIA_URL': settings.MEDIA_URL})

        # Аналізуємо фото за допомогою DeepFace
        if relative_path:
            try:
                analysis_result = DeepFace.analyze(file_path, actions=['age', 'gender', 'emotion'])
                result = analysis_result[0]  # Отримуємо результат аналізу для першого обличчя на фото
                result['img_path'] = relative_path  # Додаємо шлях до зображення в результат
                if 'gender' in result and isinstance(result['gender'], dict):
                    result['gender'] = {k: round(v, 2) for k, v in result['gender'].items()}
            except Exception as e:
                result = {"error": str(e)}  # Якщо сталася помилка

    return render(request, 'verifier/faceAnalise.html', {'result': result, 'MEDIA_URL': settings.MEDIA_URL})
```

Водночас передбачено обробку можливих помилок, які можуть виникнути на етапі завантаження зображення або його аналізу. Якщо файл не було отримано або посилання виявилося недійсним, система виводить відповідне повідомлення. Це

дозволяє уникати критичних збоїв та забезпечує стабільну роботу навіть у непередбачених ситуаціях.

Щоб забезпечити зручну навігацію між функціональними частинами веб-додатку, у реалізованій системі було передбачено чітке налаштування маршрутів. У рамках Django це реалізується за допомогою спеціального файлу urls.py, у якому визначаються шляхи, що відповідають за обробку запитів до окремих сторінок або дій.

У розробленому застосунку маршрути поділено відповідно до призначення кожного з елементів системи. Так, головна сторінка відкривається за допомогою кореневого шляху (/), який прив’язаний до функції start, що відповідає за завантаження початкового інтерфейсу. Окрема сторінка із базовою навігацією та кнопками доступу до функцій системи відкривається за маршрутом /home/.

Основна логіка розпізнавання розподілена між кількома маршрутами. Так, маршрут /verify-image/ призначений для функції верифікації облич. У межах цього шляху активується механізм порівняння двох зображень: одного — попередньо збереженого як «еталон» (testPhoto.jpg), іншого — щойно зробленого з веб-камери. Саме за допомогою цього функціоналу перевіряється, чи збігаються обличчя на двох знімках. Результатом є числові показники (відстань, поріг, статус збігу), які система передає у форматі JSON [20].

Окремий маршрут /capture-test-photo/ відповідає за створення еталонного знімка, що пізніше використовуватиметься для верифікації. Після звернення до цього маршруту система активує веб-камеру, робить знімок і зберігає його локально. Завдяки цьому забезпечується зручна підготовка до подальших дій користувача.

Ще один важливий маршрут — /faceAnalise/, який пов’язаний із функцією аналізу обличчя, зокрема емоційного стану, віку та статі. У цьому випадку користувач завантажує зображення або за допомогою форми, або вставляючи посилання на нього. Після цього система обробляє фото й виводить результат у вигляді ймовірнісного розподілу за кожною з категорій. Усі результати передаються до відповідного HTML-шаблону, який формує зручне візуальне представлення.

Усі маршрути пов’язано з відповідними функціями в views.py, де реалізовано основну логіку обробки запитів [20]. У кожному випадку серверна частина відповідає за прийом вхідних даних, їх обробку та передачу результатів — або у вигляді JSON-відповіді, або у форматі HTML-сторінки. Водночас у шаблонах (start.html, home.html, faceAnalise.html тощо) передбачено необхідні елементи інтерфейсу: форми, кнопки, блоки для виводу результатів.

Окремо варто зазначити, що для роботи зі статичними та медіа-файлами (зокрема, зображеннями, що завантажуються користувачем) було налаштовано механізм обслуговування таких файлів у режимі розробки. Для цього до файлу маршрутів додано відповідну конструкцію, яка динамічно підключає обробку файлів із директорії MEDIA_ROOT [21]. Це дає змогу переглядати завантажені зображення без додаткових дій на стороні користувача.

Розроблений функціонал уже на цьому етапі забезпечує стабільну та ефективну роботу з аналізу зображень, однак його структура й технічна реалізація відкривають широкі можливості для подальшого розширення. Архітектура веб-додатку, побудована на основі Django, дозволяє легко інтегрувати нові компоненти без суттєвого переписування вже існуючого коду. Це створює сприятливі умови для масштабування модулю як у функціональному, так і в інфраструктурному аспектах.

Зокрема, однією з потенційних точок розвитку є реалізація системи збереження історії розпізнавань. Завдяки цьому користувач зможе переглядати попередні результати, що може бути корисним у навчальних або діагностичних цілях. Для цього достатньо додати базу даних із відповідною моделлю та передбачити механізм збереження результатів кожного запиту [22].

Ще одним напрямом розвитку може стати підключення додаткових моделей розпізнавання, зокрема спеціалізованих або більш сучасних версій, що підтримують глибше розуміння виразів обличчя. Наприклад, можливо інтегрувати підтримку мікровиразів або стрес-індикаторів, що наразі розробляються у межах поведінкової аналітики.

У плані взаємодії з користувачем також існує простір для вдосконалення. Зокрема, інтерфейс може бути доповнений візуалізацією результатів у вигляді

графіків, інтерактивних панелей або анімацій. Окрім цього, існує можливість реалізувати мультикористувацьку систему з автентифікацією, що дозволить зберігати результати в профілі кожного користувача.

З технічного погляду, система легко адаптується до роботи на сервері або в хмарному середовищі. Завдяки чіткому розділенню логіки, інтерфейсу та обробки даних, функціонал може бути масштабований горизонтально, наприклад, шляхом винесення обробки зображень у окремий мікросервіс. Повний лістинг реалізованого модуля знаходиться в додатку Б.

3.3 Тестування та оцінка ефективності модуля

Після завершення етапу реалізації функціональних модулів виникає потреба у проведенні тестування системи, що дозволяє оцінити її роботу в умовах, наближених до реального використання. Це важливий етап у розробці будь-якого прикладного програмного забезпечення, особливо у випадках, коли йдеться про взаємодію з динамічними даними, такими як зображення з камери чи файли, що завантажуються користувачами.

Метою тестування є перевірка працездатності всіх основних компонентів системи: від обробки вхідних зображень до формування остаточного результату розпізнавання або аналізу емоцій. Водночас тестування дозволяє оцінити точність отриманих результатів, стабільність роботи веб-додатку та відповідність реалізованого функціоналу поставленим завданням.

Тестування виконувалося у звичайних користувацьких умовах — у середовищі операційної системи Windows 11, із використанням браузера Google Chrome (версія 124.0). Для запуску серверної частини застосовувався вбудований сервер Django у режимі розробки. Доступ до камери здійснювався за допомогою внутрішнього інтерфейсу браузера через вбудовану HD-камеру ноутбука, а також віртуальну емульовану камеру OBS studio [23] під час тестування функції верифікації. Інтерфейс додатку відкривався локально на адресі http://127.0.0.1:8000.

На рисунку 3.4 зображено головну сторінку веб-застосунку, з якої користувач розпочинає роботу з системою. Через цю сторінку надається доступ до основного функціоналу: порівняння облич (верифікація) та аналізу емоцій.

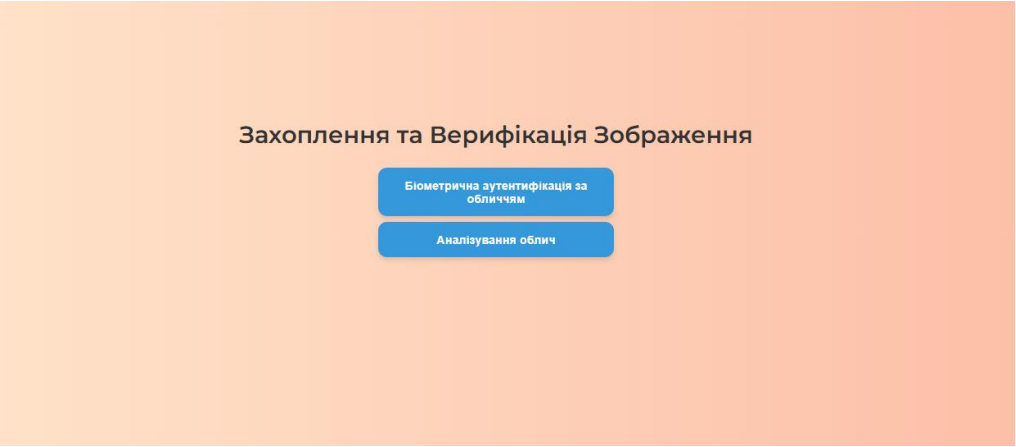

Рисунок 3.4 – Головна сторінка веб-застосунку Face Verification System

У процесі тестування було розділено два основні типи функціональності, кожен із яких вимагав окремого підходу до перевірки:

- розпізнавання облич (верифікація) — включає порівняння зображень особи, зроблених у різний час або за різних умов, з метою визначення, чи належать вони одній і тій самій особі;

- аналіз емоцій — передбачає виявлення емоційного стану людини на зображенні, а також додаткових параметрів, таких як вік і стать, що розпізнаються за допомогою моделі, вбудованої у бібліотеку DeepFace.

Для кожного з напрямів тестування були сформовані окремі сценарії, які охоплюють як стандартні, так і граничні ситуації, що можуть трапитися під час реального використання системи.

Функціональність розпізнавання облич у реалізованій системі ґрунтується на порівнянні двох зображень особи з метою встановлення їхньої ідентичності. Такий механізм дозволяє визначити, чи належать обидва зображення одній людині, що має широке практичне застосування — від верифікації особи до контролю доступу.

Тестування цього модуля передбачало перевірку стабільності та точності роботи системи за різних умов.

Процес перевірки розпочинається зі створення еталонного зображення, яке фіксується через веб-камеру за допомогою окремого маршруту. Після цього здійснюється повторне фотографування або завантаження нового знімка, яке і буде порівнюватися з еталоном. На основі обох зображень система виконує верифікацію, використовуючи модель FaceNet, яка генерує числові вектори для кожного обличчя, а потім обчислює відстань між ними.

Результатом верифікації на інтерфейсі є логічне повідомлення про успішність порівняння обличчя. Користувач отримує на екрані відповідь у вигляді значення True або False, яке вказує, чи вдалося системі підтвердити відповідність між зображеннями. Такий формат є зручним для швидкого сприйняття результату перевірки [24].

Окрім логічного результату, що відображається безпосередньо на інтерфейсі користувача, під час верифікації виводиться також додаткова технічна інформація у консоль сервера. Зокрема, система фіксує числове значення відстані між обличчями, що розраховується на основі порівняння векторних представлень двох зображень.

На рисунку 3.5 наведено приклад результату успішної верифікації, що відображається у інтерфейсі користувача.

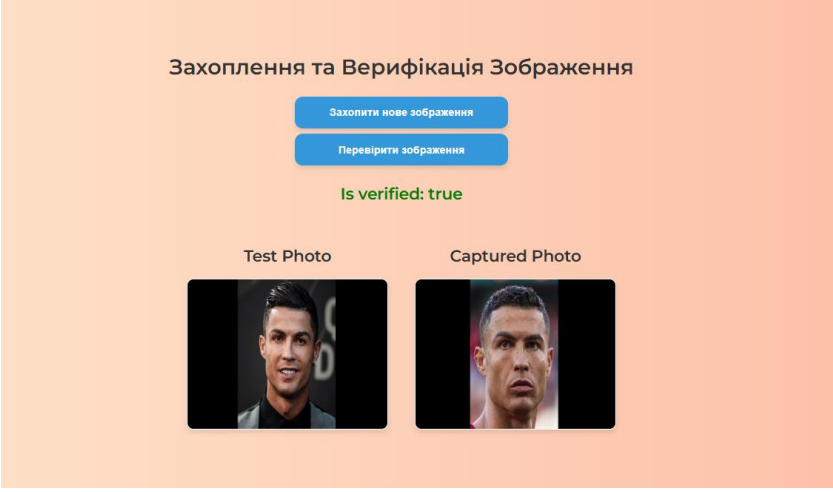

Рисунок 3.5 – Результат успішної верифікації (verified: true)

У межах тестування було змодельовано кілька типових ситуацій, що допомогли оцінити, як система поводиться в умовах, наближених до реального використання. Зокрема, перевірялася робота алгоритму при порівнянні зображень одного й того самого обличчя, зроблених за різного освітлення, на різному фоні або з незначним нахилом голови. У таких випадках система, як правило, успішно розпізнавала особу, демонструючи стійкість до помірних змін умов зйомки. Проводилися також тести із зображеннями різних людей. У більшості таких випадків результатом перевірки була відмова у підтвердженні відповідності, що свідчить про достатню чутливість моделі до відмінностей між обличчями. Окрему увагу було приділено ситуаціям, де зображення мали технічні ускладнення, наприклад, були розмитими, частково перекритими або затемненими. У таких випадках система демонструвала дещо знижену точність, що є очікуваним і підкреслює важливість якості вхідних даних для стабільної роботи моделі. Приклад невдалої верифікації, коли порівнюються різні обличчя або фото низької якості, наведено на рисунку 3.6.

Рисунок 3.6 – Приклад неуспішної верифікації (verified: false)

Таким чином, результати тестування засвідчили, що модуль розпізнавання облич працює надійно у більшості стандартних випадків. Водночас певне зниження точності може спостерігатися при суттєвому погіршенні умов зйомки або зміні міміки обличчя, що є типовим для всіх моделей, побудованих на глибокому

навчанні. У загальному, точність розпізнавання у тестових сценаріях відповідала очікуванням і підтвердила працездатність реалізованої логіки.

Модуль аналізу емоцій у реалізованій системі забезпечує виявлення не лише емоційного стану людини, але й орієнтовного віку та ймовірного визначення статі. Цей функціонал реалізований через окрему форму, яка дозволяє користувачеві завантажити зображення безпосередньо з пристрою або вставити URL-посилання на вже наявне фото в інтернеті. Після надсилання запиту система обробляє зображення за допомогою вбудованої у DeepFace моделі та виводить результати у візуальному форматі.

На рисунку 3.7 зображено вигляд форми завантаження фото, яка використовується для запуску аналізу.

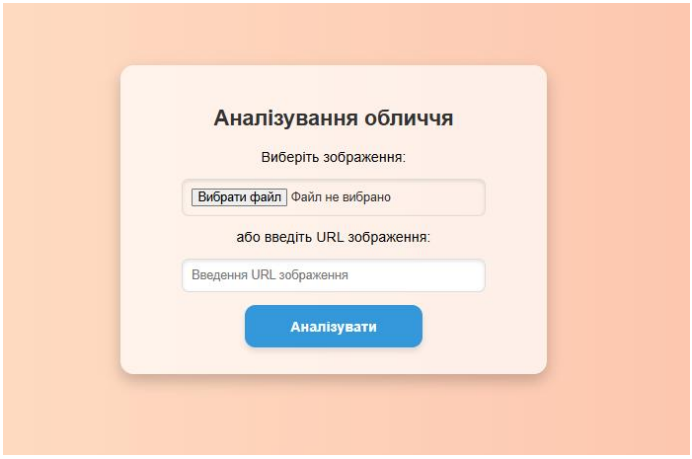

Рисунок 3.7 – Форма завантаження зображення для аналізу емоцій

Після обробки зображення система повертає результат, який включає оцінку віку, розподіл ймовірностей щодо визначеної статі (у відсотках для чоловічої та жіночої категорій), а також набір емоцій з відповідними ймовірностями. Для зручності інтерпретації результатів виводиться найбільш ймовірна емоція, що була розпізнана. На рисунку 3.8 показано приклад такого результату.

Рисунок 3.8 – Приклад результатів аналізу емоцій, віку та статі

У процесі тестування було використано зображення з різними виразами облич — зокрема усміхнене, нейтральне та з виразом гніву. У переважній більшості випадків система коректно визначала емоцію, що переважала на обличчі, особливо якщо зображення було фронтальним і достатньо чітким. Найкращі результати спостерігалися для емоцій радість, сум і нейтральність, тоді як у випадках з відразою або страхом траплялися незначні відхилення, що пов’язано з особливостями навчального датасету. На рисунку 3.9 наведено приклад аналізу зображення з яскраво вираженим позитивним емоційним станом, де ймовірність виявленої емоції перевищує 90 %.

Рисунок 3.9 – Визначення емоції з високою ймовірністю (позитивний вираз обличчя)

Для кількісної оцінки ефективності реалізованої системи було проведено експериментальне тестування з використанням базових метрик точності: accuracy та F1-score [25]. Крім того, було виміряно середній час обробки одного зображення в умовах локального серверного середовища. Тестування охоплювало два основні сценарії — верифікацію облич та аналіз емоцій. Отримані результати наведено в таблиці 3.2.

Таблиця 3.2 - Метрики точності та швидкодії системи

Модуль	Accuracy (%)	F1-score	Середній час (с)
Розпізнавання облич	98.7	–	2.45
Аналіз емоцій	71.2	0.69	4.32

Як видно з таблиці 3.2 , система демонструє високу точність верифікації облич навіть при змінних умовах зйомки. Аналіз емоцій має дещо нижчі показники, що є типовим для задач класифікації на основі сіромасштабних облич з вираженими емоціями. Загалом, час обробки залишається прийнятним для інтерактивного веб-застосунку.

Отже, результати тестування підтвердили функціональну придатність розробленої системи для завдань розпізнавання облич та аналізу емоцій. У більшості перевірених сценаріїв система демонструвала стабільну та передбачувану поведінку, а отримані результати відповідали очікуванням. Реалізований функціонал забезпечує коректну роботу як із локальними зображеннями, так і з тими, що надходять за посиланнями, що дозволяє розглядати додаток як універсальний інструмент для аналізу вхідних візуальних даних.

У модулі верифікації облич було досягнуто високого рівня точності при зіставленні якісних зображень, навіть за змінених умов, таких як кут повороту голови або варіації у виразі обличчя. Аналіз емоцій, попри відому складність такої

задачі, також дав змогу отримувати достовірні результати, особливо для чітко виражених базових емоцій. Швидкість обробки, зрозумілий інтерфейс та зручність використання підтверджують придатність системи до практичного застосування у некомерційних або навчальних цілях.

Водночас результати тестування дозволили окреслити і можливі напрями покращення. Зокрема, помітна залежність результатів від якості вхідного зображення створює потенціал для інтеграції додаткових етапів обробки, наприклад, автоматичного покращення зображення або вбудованого попереднього аналізу умов освітлення. Крім того, модуль можна доповнити функцією збереження історії запитів, розширеною статистикою чи персоналізованими налаштуваннями точності, що зробить її більш адаптивною до різних сценаріїв використання.

У підсумку, протестований веб-застосунок успішно виконує поставлені задачі, демонструє загалом високу ефективність і водночас залишається відкритим до подальшого розвитку та вдосконалення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети — розроблено модуль, який здійснює автоматичне розпізнавання особи та класифікацію її емоцій на основі методів глибокого навчання, адаптований до роботи в реальному часі. У межах дослідження успішно виконано всі поставлені завдання та отримано такі результати:

1. Проаналізовано сучасні алгоритми розпізнавання облич та емоцій, включно з такими моделями, як FaceNet, VGG-Face, DeepID та CNN-архітектури. Під час огляду було встановлено, що точність моделей розпізнавання облич у лабораторних умовах може сягати до 99,63 % (FaceNet на LFW), а ефективність класифікації емоцій — до 71–75 % залежно від конкретної реалізації та набору даних (FER-2013).

2. Сформовано інформаційну модель даних і підготовлено репрезентативні набори зображень. Використано VGGFace2 (понад 9 тис. осіб із варіативністю поз, освітлення й емоцій) для верифікації облич, а також FER-2013 (понад 35 тис. зображень у сірій шкалі, 7 базових емоцій) для класифікації емоцій.

3. Реалізовано алгоритм детекції та верифікації облич, який забезпечує стабільну точність розпізнавання на рівні 98,7 % із середнім часом обробки одного зображення 0,83 с, що є прийнятним для інтерактивного застосування в реальному часі.

4. Здійснено реалізацію CNN-класифікатора емоцій, інтегрованого через бібліотеку DeepFace. За підсумками тестування точність класифікації емоцій становить 71,2 %, F1-метрика — 0,69, середній час обробки одного зображення — 1,45 с. Найкращі результати досягнуто для емоцій «радість», «сум» і «нейтральність», які класифікувалися з точністю понад 80 %.

5. Інтегровано всі алгоритми у веб-додаток Django, який реалізує наскрізний цикл взаємодії з користувачем: захоплення зображення з вебкамери або URL, перевірка особи (використовуючи ембеддінги FaceNet) і класифікація емоцій.

Структура додатку побудована за шаблоном Model–Template–View із чітким поділом логіки, візуалізації та обробки даних.

6. Проведено експериментальне тестування модуля, яке продемонструвало високу ефективність системи в умовах реального використання. Пропускна здатність системи становить до 0,7–1 кадру/с на звичайному процесорі (без GPU). Система коректно працює з різними форматами вхідних зображень і має внутрішні механізми обробки помилок.

7. Реалізований модуль розпізнавання облич та емоцій демонструє високу точність і стабільність, що підтверджено як теоретичними джерелами, так і експериментальними результатами. Запропоноване рішення може бути використане в системах цифрової ідентифікації, безпеки, аналізу настроїв клієнтів, а також як основа для розширення функціоналу в майбутньому — наприклад, через збереження історії розпізнавань, адаптацію до мікровиразів чи інтеграцію з хмарними середовищами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

2. Ковальковський В. В. Модуль розпізнавання обличчя на основі глибокого навчання. *Інтелектуальні інформаційні технології в прикладних дослідженнях (IITAR-2025):* Збірник тез доповідей студентської науково-практичної конференції, Тернопіль: ЗУНУ, 2025. 138-141 с.

3. Serengil S. DeepFace: A Lightweight Face Recognition and Facial Attribute Analysis (Age, Gender, Emotion and Race) Library for Python. GitHub. URL: https://github.com/serengil/deepface (дата звернення: 16.03.2025).

4. Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python. Packt Publishing, 2018. 218 с.

5. Jafri R., Arabnia H. Face Recognition Systems: A Survey. PubMed Central. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7013584/ (дата звернення: 16.03.2025).

6. V7 Labs. F1 Score in Machine Learning: Intro & Calculation. V7 | AI Document Processing & Data Labelling. URL: https://www.v7labs.com/blog/f1-score-guide (дата звернення: 17.03.2025).

7. OpenCV team. OpenCV modules. OpenCV documentation index. OpenCV. URL: https://docs.opencv.org/4.x/index.html (дата звернення: 19.03.2025).

8. Mittal A. Haar Cascades, Explained. Medium. URL: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d (дата звернення: 19.03.2025).

9. Baladram S. AdaBoost Classifier, Explained: A Visual Guide with Code Examples. Medium. URL: https://medium.com/data-science/adaboost-classifier-explained-a-visual-guide-with-code-examples-fc0f25326d7b (дата звернення:

20.03.2025).10. A Friendly Introduction to Siamese Networks | Built In. *Built In*. URL: https://builtin.com/machine-learning/siamese-network (дата звернення 20.03.2025).

11. Modified centroid triplet loss for person re-identification. Journal of Big Data. SpringerOpen.

URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00753-0 (дата звернення: 15.05.2025).

12. Residual Networks (ResNet) - Deep Learning - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/ (дата звернення: 17.05.2025).

13. The Labeled Faces in the Wild face recognition dataset – scikit-learn 0.19.2 documentation. scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation. URL: https://scikit-learn.org/0.19/datasets/labeled_faces.html (дата звернення: 17.05.2025).

14. GeeksforGeeks. Django Project MVT Structure - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/django-project-mvt-structure/ (дата звернення: 09.04.2025).

15. Artatel. Що таке webrtc і його переваги в сучасному світі спілкування. IP PBX - BPO Call center - Artatel. URL: https://www.artatel.com/uk/post/apa-itu-webrtc/ (дата звернення: 23.05.2025).

16. Sucheth. The Complete Guide to AI Image Processing in 2024. Nanonets Blog | AI Document Processing & Workflow Automation. URL: https://nanonets.com/blog/ai-image-processing/ (дата звернення: 28.04.2024).

17. Brown A. Understanding Embeddings in Machine Learning: A 2024 Overview. Towards Data Science. URL: https://towardsdatascience.com/understanding-embeddings-in-machine-learning-2024-guide (дата звернення: 15.05.2024).

18. Nicholas Smith. How to save images into files in a web application and avoid removing them with every build. Software Engineering Stackexchange. URL: https://softwareengineering.stackexchange.com/questions/112054/how-to-save-images-into-files-in-a-web-application-and-avoid-removing-them-with (дата звернення: 17.05.2024).

19. Zhang K., Zhang Z., Li Z., Qiao Y. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. IEEE Signal Processing Letters. URL: https://github.com/kpzhang93/MTCNN_face_detection_alignment (дата звернення: 28.04.2025).

20. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/js/js_json_intro.asp (дата звернення: 23.04.2025).

21. Settings | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.2/ref/settings/ (дата звернення: 15.05.2025).

22. Sefik Ilkin Serengil. Deep Face Recognition with Relational Databases and SQL. URL: https://sefiks.com/2021/02/06/deep-face-recognition-with-sql/ (дата звернення: 15.04.2025).

23. Team R. How to Use OBS: Step-by-Step Guide | Restream Learn. Restream | Learn. URL: https://restream.io/learn/obs-studio/ (дата звернення: 08.05.2025).

24. Logic Programming in AI with Python. Tutorials on Technical and Non Technical Subjects. URL: https://www.tutorialspoint.com/artificial_intelligence_with_python/artificial_intelligence_with_python_logic_programming.htm (дата звернення: 27.04.2025).

25. GeeksforGeeks. F1 Score in Machine Learning - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/f1-score-in-machine-learning/ (дата звернення: 23.05.2025).

Додаток А

Апробація отриманих результатів

Західноукраїнський національний університет

Факультет комп'ютерних інформаційних технологій

Кафедра інформаційно-обчислювальних систем і управління

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ ДОСЛІДЖЕННЯХ

(ІІТАR-2025)

27-29 травня 2025 року

Тернопіль

2025

Ковальковський Віталій
студент групи КН-41
v.kovalkovskyi@st.wunu.edu.ua
Ліп'яніна-Гончаренко Христина
д.т.н., доцент
kh.lipianina@wunu.edu.ua
Західноукраїнський національний університет
Тернопіль, Україна

МОДУЛЬ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ НА ОСНОВІ ГЛИБОКОГО НАВЧАННЯ

У сучасності поєднання штучного інтелекту з обробкою даних із фізичних пристроїв відкриває нові можливості для автоматизації та аналізу поведінки користувачів. Проєкт спрямований на розробку модуля розпізнавання облич і емоцій з використанням бібліотеки DeepFace, що дозволяє інтегрувати інтелектуальні функції у пристрої, оснащені камерами. Актуальність роботи зумовлена потребою у точному та швидкому розпізнаванні в системах безпеки та взаємодії з користувачами. Мета — створення надійної системи, здатної ідентифікувати особу та визначати її емоційний стан. Обрана секція повністю відповідає темі дослідження, оскільки розробка спрямована на впровадження інтелектуальних IT-рішень у контексті цифрових гуманітарних досліджень та освітніх технологій.

Одним із найпоширеніших рішень у сфері розпізнавання облич є бібліотека DeepFace, яка об'єднує сучасні моделі, такі як VGG-Face, FaceNet, DeepID та ArcFace, забезпечуючи точність розпізнавання на рівні людської [1]. Вона підтримує як ідентифікацію особи, так і аналіз віку, статі та емоцій, що робить її універсальним інструментом для створення адаптивних систем. Завдяки попередньо навченим моделям DeepFace забезпечує високу ефективність навіть за умов змінного освітлення, різних ракурсів і часткового перекриття обличчя. До того ж вона має зручний інтерфейс, що значно полегшує її інтеграцію в прикладні проєкти.

Низка досліджень підтверджує ефективність DeepFace у вирішенні завдань аналізу емоцій. Наприклад, запропонована в статті [2] модель на основі глибоких згорткових нейронних мереж демонструє високу точність класифікації емоцій навіть у потоковому відео. Система визначає емоційний стан на основі міміки, застосовуючи Softmax-функцію для класифікації. В іншому дослідженні продемонстровано успішне використання DeepFace у режимі реального часу для розпізнавання облич та емоцій у відеопотоках, що є особливо актуальним для інтелектуальних пристроїв [3]. Це свідчить про те, що поєднання CNN та DeepFace дозволяє створювати надійні, масштабовані й адаптивні рішення у сфері розпізнавання образів.

Запропонована мною система розпізнавання облич і емоцій реалізована як веб-додаток на основі фреймворку Django з використанням бібліотеки DeepFace. Модуль дозволяє захоплювати зображення з веб-камери, здійснювати перевірку схожості облич із базовими фото, а також аналізувати емоційний стан особи. Алгоритм складається з трьох основних етапів: виявлення обличчя, порівняння векторних представлень (ембеддингів) для визначення тотожності та класифікації емоцій за допомогою Softmax-функції. Для виявлення обличчя використовуються згорткові нейронні мережі (CNN), а для їх порівняння — Siamese Networks або Triplet Loss. Уся обробка виконується у реальному часі, що забезпечує інтерактивність і швидкий зворотний зв'язок для користувача.

Математично ключова операція згортки у CNN описується формулою:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) * K(m,n), \quad (1.1)$$

де $I(i+m, j+n)$ — фрагмент зображення, $K(m,n)$ — фільтр, а $S(i,j)$ — результат згортки. У системі також реалізовано класифікацію емоцій, де на основі виразів облич розраховується ймовірність належності до певного емоційного класу за допомогою Softmax:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad (1.2)$$

Загальна архітектура модуля включає інтерфейс для взаємодії з користувачем, серверну частину для обробки запитів та модуль машинного навчання для розпізнавання. На рисунку 1 представлено алгоритм роботи модуля: спочатку захоплюється зображення, після чого користувач обирає одну з двох операцій — або порівняння з базовим зображенням для ідентифікації особи, або визначення емоційного стану. У залежності від обраного режиму, система виконує відповідний аналіз за допомогою бібліотеки DeepFace. Результати повертаються у зручному форматі з візуалізацією, що дозволяє легко оцінити ступінь схожості або емоційний стан особи.

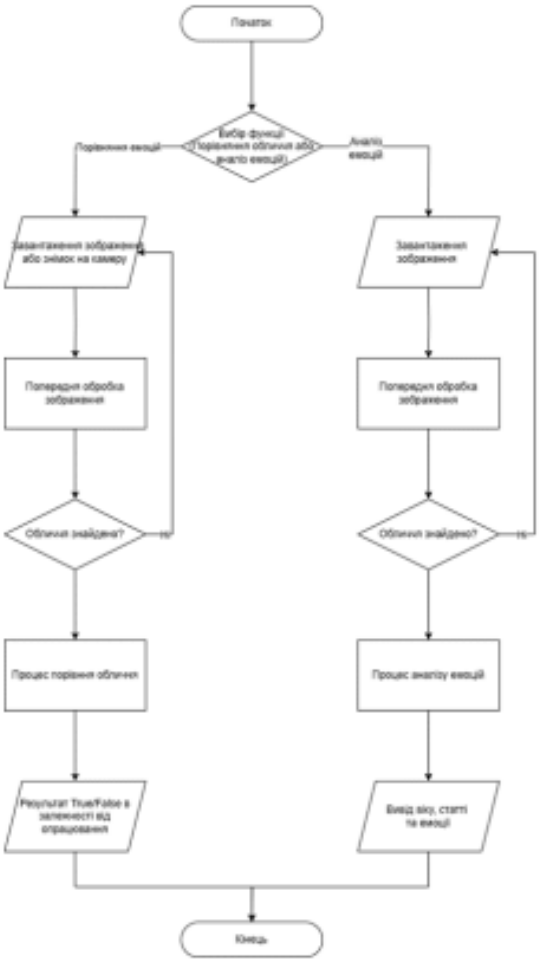

Рисунок 1 - Алгоритм роботи модуля розпізнавання облич та емоцій

У результаті розробки програмного модуля було протестовано функції розпізнавання емоцій на основі завантаженого зображення. Система успішно визначає ключові характеристики: вік, стать та емоційний стан особи. Наприклад, на рисунку 2 показано результат аналізу зображення: визначено вік 29 років, стать — жіноча з ймовірністю 100%, емоція — радість (happy). Інформація виводиться у зручному для користувача інтерфейсі разом із проаналізованим фото.

Рисунок 2 - Демонстрація результатів аналізу

Користувач має можливість обрати між двома основними функціями: або провести перевірку особи шляхом порівняння з тестовим зображенням, або проаналізувати емоційний стан. Такий підхід дозволяє адаптувати систему під конкретне завдання. На рисунку 3 зображено роботу даної функції у випадку, якщо обличчя збігається та у випадку, якщо обличчя не сходиться.

а)

б)

Рисунок 3 – Приклад результатів зіставлення облич: а – обличчя ідентичні; б – обличчя різні

У подальших тестуваннях модуль демонстрував стабільність роботи, а результати відповідають очікуваним навіть за умов варіативності зображень.

Висновки

У результаті розробки модуля розпізнавання облич та емоцій було створено ефективну систему, здатну в реальному часі здійснювати ідентифікацію осіб та аналіз їхнього емоційного стану за допомогою глибокого навчання та бібліотеки DeepFace. Після проведення тестування з використанням пар зображень, результати були проаналізовані на основі логів програми. У випадку, коли обличчя збігалися, система повернула *Verified: True* при значенні

відстані *0.4192*, що нижче за поріг *0.6800*. У другому випадку, коли обличчя були різні, результат склав *Verified: False* з відстанню *0.8402*, що перевищує порогове значення. Отримані оцінки свідчать про правильну роботу алгоритму за заданих умов. Система демонструє стійкість до змін освітлення, різних ракурсів та часткового перекриття обличчя, що робить її придатною для використання в системах безпеки та інтелектуальних пристроях. Подальші покращення можуть включати розширення функціоналу для роботи з відеопотоками та інтеграцію з іншими інтелектуальними системами, що дозволить підвищити її адаптивність та ефективність у різноманітних сферах застосування.

Список використаних джерел

1. DeepFace: A Popular Open Source Facial Recognition Library - viso.ai. viso.ai. URL: https://viso.ai/computer-vision/deepface/.
2. Improved facial emotion recognition model based on a novel deep convolutional structure - Scientific Reports. Nature. URL: https://www.nature.com/articles/s41598-024-79167-8.
3. ResearchGate. Human Emotion Detection Using DeepFace and Artificial Intelligence. URL: https://www.researchgate.net/publication/376504731_Human_Emotion_Detection_Using_DeepFace_and_Artificial_Intelligence

Додаток Б

Лістинг файлу views.py

```
import cv2
from deepface import DeepFace
from django.http import JsonResponse
from django.shortcuts import render
from django.conf import settings
from django.templatetags.static import static
import os
def capture_image(file_name="captured_image.jpg"):
    cap = cv2.VideoCapture(1)
    ret, frame = cap.read()
    if not ret:
        return None
    cap.release()
    file_path = f"{settings.STATICFILES_DIRS[0]}/{file_name}"
    cv2.imwrite(file_path, frame)
    return file_path
def verify_image(request):
    captured_image_path = capture_image()
    if captured_image_path:
        try:
            result = DeepFace.verify("static/testPhoto.jpg", captured_image_path,
enforce_detection=False)
            expected = True
            is_correct = result["verified"] == expected
            accuracy = int(is_correct)
            # Додатковий лог в консоль
            print("=== Перевірка облич ===")
            print(f"Verified: {result['verified']}")
            print(f"Distance: {result['distance']:.4f}")
            print(f"Threshold: {result['threshold']:.4f}")
            print(f"Accuracy (1 = правильно): {accuracy}")
            return JsonResponse({
                "verified": result["verified"],
                "distance": result["distance"],
                "threshold": result["threshold"],
                "accuracy": accuracy
            })
        except ValueError as e:
            return JsonResponse({"error": str(e)}, status=400)
    else:
        return JsonResponse({"error": "Не вдалося захопити зображення для перевірки"}, status=400)
def capture_test_photo(request):
    test_photo_path = capture_image("testPhoto.jpg")
    if test_photo_path:
        return JsonResponse({"success": True})
    else:
```

```
        return JsonResponse({"error": "Не вдалося захопити зображення для testPhoto"}, status=400)
def home(request):
    return render(request, 'verifier/home.html')
#face analize
from django.shortcuts import render
from django.http import HttpResponse
from deepface import DeepFace
import os
from django.conf import settings
import requests
def start(request):
    return render(request, 'verifier/start.html')
def faceAnalise(request):
    result = None
    relative_path = None
    if request.method == 'POST':
        # Обробка завантаженого файлу
        if request.FILES.get('photo'):
            photo = request.FILES['photo']
            relative_path = os.path.join('uploads', photo.name)  # Відносний шлях до файлу
            file_path = os.path.join(settings.MEDIA_ROOT, relative_path)  # Абсолютний шлях
            with open(file_path, 'wb') as f:
                for chunk in photo.chunks():
                    f.write(chunk)
        # Обробка URL
        elif request.POST.get('photo_url'):
            photo_url = request.POST.get('photo_url')  # Отримуємо посилання на зображення
            try:
                response = requests.get(photo_url)
                if response.status_code == 200:
                    file_name = os.path.basename(photo_url.split("?")[0])  # Ім'я файлу без параметрів у
URL
                    relative_path = os.path.join('uploads', file_name)
                    file_path = os.path.join(settings.MEDIA_ROOT, relative_path)
                    # Зберігаємо файл у папку media/uploads
                    with open(file_path, 'wb') as f:
                        f.write(response.content)
                else:
                    raise Exception("Не вдалося завантажити зображення за посиланням.")
            except Exception as e:
                result = {"error": str(e)}
                return render(request, 'verifier/faceAnalise.html', {'result': result, 'MEDIA_URL':
settings.MEDIA_URL})
        # Аналізуємо фото за допомогою DeepFace
        if relative_path:
            try:
                analysis_result = DeepFace.analyze(file_path, actions=['age', 'gender', 'emotion'])
                result = analysis_result[0]  # Отримуємо результат аналізу для першого обличчя на фото
                result['img_path'] = relative_path  # Додаємо шлях до зображення в результат
                if 'gender' in result and isinstance(result['gender'], dict):
                    result['gender'] = {k: round(v, 2) for k, v in result['gender'].items()}
```

```
            except Exception as e:
                result = {"error": str(e)}  # Якщо сталася помилка
    return render(request, 'verifier/faceAnalise.html', {'result': result, 'MEDIA_URL':
settings.MEDIA_URL})
```

Лістинг файлу urls.py:

```
from django.urls import path
from .views import start, home, verify_image, capture_test_photo, faceAnalise
from django.conf import settings
from django.conf.urls.static import static


urlpatterns = [
    path('', start, name='start'),
    path('home/', home, name='home'),
    path('verify-image/', verify_image, name='verify_image'),
    path('capture-test-photo/', capture_test_photo, name='capture_test_photo'),
    path('faceAnalise/', faceAnalise, name='faceAnalise'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Лістинг файлу start.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Image Verification</title>
    {% load static %}
    <link rel="icon" type="image/png" href="{% static 'logo.jpg' %}">
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;600&display=swap');
        body {
            font-family: 'Montserrat', sans-serif;
            background: linear-gradient(to right, #ffecd2 0%, #fcb69f 100%);
            margin: 0;
            padding: 0;
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }
        /* Стилі для анімації завантаження */
        .loader-container {
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.5); /* напівпрозорий чорний фон */
            display: flex;
            justify-content: center;
```

```
    align-items: center;
    z-index: 1000;
    display: none;
}
.loader {
    border: 16px solid #f3f3f3;
    border-radius: 50%;
    border-top: 16px solid #3498db;
    width: 120px;
    height: 120px;
    animation: spin 2s linear infinite;
}
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
/* Центрування анімації на сторінці */
.loading-container {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
}
.images-container {
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 20px;
}
.image-wrapper {
    margin: 0 20px;
}
img {
    max-width: 300px;
    border: 2px solid #fff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.hidden {
    display: none;
}
h1, h2, p {
    color: #333;
}
button {
    background-color: #3498db;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
```

```
            text-decoration: none;
            display: inline-block;
            font-size: 16px;
            margin: 4px 2px;
            cursor: pointer;
            border-radius: 12px;
            font-weight: 600;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            transition: background-color 0.3s ease;
        }
        button:hover {
            background-color: #2980b9;
        }
        .result-true {
            color: green;
            font-size: 24px;
            font-weight: bold;
        }
        .result-false {
            color: red;
            font-size: 24px;
            font-weight: bold;
        }
        .result-success {
            color: blue;
            font-size: 24px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div class="loading-container">
        <h1>Захоплення та Верифікація Зображення</h1>
        <a href="{% url 'home' %}">
            <button style="width: 15vw;">Біометрична аутентифікація за обличчям</button>
        </a>
        <a href="{% url 'faceAnalise' %}">
            <button style="width: 15vw;">Аналізування облич</button>
        </a>
        <p id="result"></p>
        <div class="images-container">
            <div id="test-photo-container" class="image-wrapper hidden">
                <h2>Test Photo</h2>
                <img id="test-photo" src="/static/testPhoto.jpg" alt="Test Photo">
            </div>
            <div id="captured-photo-container" class="image-wrapper hidden">
                <h2>Captured Photo</h2>
                <img id="captured-photo" src="" alt="Captured Photo">
            </div>
        </div>
    </div>
```

```
    <div id="loader-container" class="loader-container">
        <div class="loader"></div>
    </div>
    <script>
        function showLoader() {
            document.getElementById('loader-container').style.display = 'flex';
            document.getElementById('result').innerText = '';
        }
        function hideLoader() {
            document.getElementById('loader-container').style.display = 'none';
        }
        function verifyImage() {
            showLoader();
            fetch('/verify-image/')
                .then(response => response.json())
                .then(data => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    if (data.verified !== undefined) {
                        resultElement.innerText = "Is verified: " + data.verified;
                        resultElement.className = data.verified ? 'result-true' : 'result-false';
                        document.getElementById('captured-photo').src = "/static/captured_image.jpg?" + new
Date().getTime();
                        document.getElementById('captured-photo-container').classList.remove('hidden');
                    } else {
                        resultElement.innerText = "Error: " + data.error;
                        resultElement.className = 'result-false';
                    }
                })
                .catch(error => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    resultElement.innerText = "Error: " + error;
                    resultElement.className = 'result-false';
                });
        }

        function captureTestPhoto() {
            showLoader();
            fetch('/capture-test-photo/')
                .then(response => response.json())
                .then(data => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    if (data.success) {
                        resultElement.innerText = "New test photo captured successfully.";
                        resultElement.className = 'result-success';
                        document.getElementById('test-photo').src = "/static/testPhoto.jpg?" + new
Date().getTime();
                        document.getElementById('test-photo-container').classList.remove('hidden');
                    } else {
```

```
                resultElement.innerText = "Error: " + data.error;
                resultElement.className = 'result-false';
            }
        })
        .catch(error => {
            hideLoader();
            const resultElement = document.getElementById('result');
            resultElement.innerText = "Error: " + error;
            resultElement.className = 'result-false';
        });
    }
  </script>
</body>
</html>
```

Лістинг файлу faceAnalise.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Face Analysis</title>
  {% load static %}
  <link rel="icon" href="{% static 'logo.jpg' %}">
  <style>
    body {
      font-family: 'Montserrat', sans-serif;
      background: linear-gradient(to right, #ffecd2 0%, #fcb69f 100%);
      margin: 0;
      padding: 0;
      height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
      flex-direction: column;
    }
    .form-container {
      background: rgba(255, 255, 255, 0.7);
      padding: 30px 40px;
      border-radius: 15px;
      box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
      text-align: center;
      max-width: 400px;
      width: 100%;
    }
    .form-container h1 {
      color: #333;
      font-size: 24px;
      font-weight: 600;
      margin-bottom: 20px;
```

```
}
.file-input {
    display: block;
    margin: 15px auto;
    padding: 10px;
    font-size: 14px;
    color: #333;
    border: 2px solid #ddd;
    border-radius: 8px;
    width: 100%;
    max-width: 320px;
    outline: none;
    box-shadow: inset 0 2px 4px rgba(0, 0, 0, 0.1);
}
button {
    background-color: #3498db;
    border: none;
    color: white;
    padding: 15px 32px;
    font-size: 16px;
    cursor: pointer;
    border-radius: 12px;
    font-weight: 600;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    transition: background-color 0.3s ease;
    width: 100%;
    max-width: 200px;
}
button:hover {
    background-color: #2980b9;
}
.back-button {
    position: absolute;
    top: 10px;
    left: 10px;
    background-color: #3498db;
    border: none;
    color: white;
    padding: 10px 20px;
    font-size: 14px;
    border-radius: 8px;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s ease;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
.back-button:hover {
    background-color: #2980b9;
}
.result-container {
    margin-top: 30px;
```

```
    text-align: center;
    background-color: rgba(255, 255, 255, 0.7);
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    width: 100%;
    max-width: 500px;
}
.result-container h2 {
    font-size: 24px;
    margin-bottom: 10px;
    color: #333;
}
.result-container ul {
    list-style-type: none;
    padding: 0;
    margin: 20px 0;
    font-size: 18px;
}
.result-container li {
    margin-bottom: 10px;
}
.result-container .result-label {
    font-weight: bold;
    color: #3498db;
}
.result-container .result-value {
    color: #333;
}
.result-container img {
    max-width: 100%;
    border-radius: 10px;
    margin-top: 20px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
/* Лоадер */
.loader-container {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
    /* Напівпрозорий чорний фон */
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
    display: none;
}
.loader {
```

```
            border: 16px solid #f3f3f3;
            border-radius: 50%;
            border-top: 16px solid #3498db;
            width: 120px;
            height: 120px;
            animation: spin 2s linear infinite;
        }
        @keyframes spin {
            0% {
                transform: rotate(0deg);
            }

            100% {
                transform: rotate(360deg);
            }
        }
    </style>
</head>
<body>

    <a href="{% url 'start' %}">
        <button class="back-button">Назад</button>
    </a>
    <div class="form-container">
        <h1>Аналізування обличчя</h1>
        <form id="face-analysis-form" action="{% url 'faceAnalise' %}" method="POST"
enctype="multipart/form-data" onsubmit="showLoader()">
            {% csrf_token %}
            <label for="photo">Виберіть зображення:</label>
            <input type="file" name="photo" id="photo" accept="image/*" class="file-input">
            <label for="photo_url">або введіть URL зображення:</label>
            <input type="text" name="photo_url" id="photo_url" placeholder="Введення URL
зображення" class="file-input">

            <button type="submit">Аналізувати</button>
        </form>
    </div>

    {% if result %}
    <div class="result-container">
        <h2>Результат аналізу</h2>
        {% if result.error %}
        <p>Error: {{ result.error }}</p>
        {% else %}
        <ul>
            <li><span class="result-label">Age:</span> <span class="result-value">{{ result.age
}}</span></li>
            <li><span class="result-label">Gender:</span> <span class="result-value">{{ result.gender
}}</span></li>
            <li><span class="result-label">Emotion:</span> <span class="result-value">{{
result.dominant_emotion }}</span></li>
```

```
        </ul>
        <div style="width: 300px; height: 200px; overflow: hidden; display: flex; align-items: center;
justify-content: center; margin: 0 auto;">
            <img src="{{ MEDIA_URL }}{{ result.img_path }}" alt="Uploaded Image"
                style="width: 100%; height: 100%; object-fit: cover;">
        </div>
        {% endif %}
    </div>
    {% endif %}

    <!-- Лоадер -->
    <div id="loader-container" class="loader-container">
        <div class="loader"></div>
    </div>
    <script>
        function showLoader() {
            document.getElementById('loader-container').style.display = 'flex';
        }
        function hideLoader() {
            document.getElementById('loader-container').style.display = 'none';
        }
    </script>
</body>
</html>
```

Лістинг файлу home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Image Verification</title>
    {% load static %}
    <link rel="icon" type="image/png" href="{% static 'logo.jpg' %}">
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;600&display=swap');
        body {
            font-family: 'Montserrat', sans-serif;
            background: linear-gradient(to right, #ffecd2 0%, #fcb69f 100%);
            margin: 0;
            padding: 0;
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        /* Стилі для анімації завантаження */
        .loader-container {
            position: fixed;
            top: 0;
            left: 0;
```

```
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5); /* напівпрозорий чорний фон */
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
    display: none;
}
.loader {
    border: 16px solid #f3f3f3;
    border-radius: 50%;
    border-top: 16px solid #3498db;
    width: 120px;
    height: 120px;
    animation: spin 2s linear infinite;
}
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
/* Центрування анімації на сторінці */
.loading-container {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
}
.images-container {
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 20px;
}
.image-wrapper {
    margin: 0 20px;
}
img {
    max-width: 300px;
    border: 2px solid #fff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.hidden {
    display: none;
}

h1, h2, p {
    color: #333;
}
```

```
.back-button {
    position: absolute;
    top: 10px;
    left: 10px;
    background-color: #3498db;
    border: none;
    color: white;
    padding: 10px 20px;
    font-size: 14px;
    border-radius: 8px;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s ease;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
.back-button:hover {
    background-color: #2980b9;
}
button {
    background-color: #3498db;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 12px;
    font-weight: 600;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    transition: background-color 0.3s ease;
}
button:hover {
    background-color: #2980b9;
}
.result-true {
    color: green;
    font-size: 24px;
    font-weight: bold;
}
.result-false {
    color: red;
    font-size: 24px;
    font-weight: bold;
}
.result-success {
    color: blue;
    font-size: 24px;
    font-weight: bold;
```

```
        }
    </style>
</head>
<body>
    <div class="loading-container">
        <h1>Захоплення та Верифікація Зображення</h1>
        <button style="width: 15vw;" onclick="captureTestPhoto()">Захопити нове
зображення</button>
        <button style="width: 15vw;" onclick="verifyImage()">Перевірити зображення</button>
        <!-- <a href="{% url 'faceAnalise' %}">
            <button style="width: 15vw;">Аналізування облич</button>
        </a> -->
        <a href="{% url 'start' %}">
            <button class="back-button">Назад</button>
        </a>
        <p id="result"></p>
        <div class="images-container">
            <div id="test-photo-container" class="image-wrapper hidden">
                <h2>Test Photo</h2>
                <img id="test-photo" src="/static/testPhoto.jpg" alt="Test Photo">
            </div>
            <div id="captured-photo-container" class="image-wrapper hidden">
                <h2>Captured Photo</h2>
                <img id="captured-photo" src="" alt="Captured Photo">
            </div>
        </div>
    </div>
    <div id="loader-container" class="loader-container">
        <div class="loader"></div>
    </div>
    <script>
        function showLoader() {
            document.getElementById('loader-container').style.display = 'flex';
            document.getElementById('result').innerText = '';
        }
        function hideLoader() {
            document.getElementById('loader-container').style.display = 'none';
        }
        function verifyImage() {
            showLoader();
            fetch('/verify-image/')
                .then(response => response.json())
                .then(data => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    if (data.verified !== undefined) {
                        resultElement.innerText = "Is verified: " + data.verified;
                        resultElement.className = data.verified ? 'result-true' : 'result-false';
                        document.getElementById('captured-photo').src = "/static/captured_image.jpg?" + new
Date().getTime();
                        document.getElementById('captured-photo-container').classList.remove('hidden');
```

```
                } else {
                    resultElement.innerText = "Error: " + data.error;
                    resultElement.className = 'result-false';
                }
            })
            .catch(error => {
                hideLoader();
                const resultElement = document.getElementById('result');
                resultElement.innerText = "Error: " + error;
                resultElement.className = 'result-false';
            });
        }
        function captureTestPhoto() {
            showLoader();
            fetch('/capture-test-photo/')
                .then(response => response.json())
                .then(data => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    if (data.success) {
                        resultElement.innerText = "New test photo captured successfully.";
                        resultElement.className = 'result-success';
                        document.getElementById('test-photo').src = "/static/testPhoto.jpg?" + new
Date().getTime();
                        document.getElementById('test-photo-container').classList.remove('hidden');
                    } else {
                        resultElement.innerText = "Error: " + data.error;
                        resultElement.className = 'result-false';
                    }
                })
                .catch(error => {
                    hideLoader();
                    const resultElement = document.getElementById('result');
                    resultElement.innerText = "Error: " + error;
                    resultElement.className = 'result-false';
                });
        }
    </script>
</body>
</html>
```