КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

Система аутентифікації за допомогою розпізнавання обличчя

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:

студент 2 курсу магістратури
групи ТШІ-21
Гурленко Антон Євгенович

Науковий керівник:

Красовська Ганна Валеріївна
к. т. н., доцент

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № ____ від «____» травня 2020 р.
В.о. зав. кафедри _____ доц. Красовська Г.В.

Київ – 2020

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаної літератури із 55 джерел та 2 додатків. Загальний обсяг роботи 89 сторінок. Робота містить 12 таблиць та 36 рисунків.

Метою роботи є дослідження ефективності та швидкості роботи існуючих алгоритмів розпізнавання облич на зображеннях та розробка системи аутентифікації за допомогою алгоритмів розпізнавання облич.

Об'єктом дослідження є технологія ідентифікації особи за біометричними параметрами обличчя в системах аутентифікації та контролю доступу.

Предметом дослідження алгоритми і методи виявлення облич на зображенні та безпосереднього розпізнавання знайдених облич.

Результати роботи:

- Проведено дослідження точності та швидкості роботи існуючих алгоритмів розпізнавання облич, що базуються на популярних нейронних мережах.
- Розроблено систему аутентифікації за допомогою обличчя, з можливістю інтеграції в існуючі системи, що потребують аутентифікації.

Апробація роботи. Результати магістерської роботи (практичні розробки та наукові дослідження) доповідалися на конференціях:

1) міжнародний науковий симпозіум «Інтелектуальні рішення» (м. Ужгород, 15–20 квітня 2019 р.) [51];
2) міжнародна науково-практична конференція «Інформаційні технології та взаємодії» (м. Київ, 20 грудня 2019 р.) [52].

За матеріалами магістерської роботи є 2 публікації у збірниках матеріалів науково-практичних конференцій.

Ключові слова.

Нейронна мережа, розпізнавання облич, система аутентифікації, згорткова нейронна мережа.

ABSTRACT

Qualification work consists of an introduction, 3 chapters, conclusions, a list of references from 55 sources and 2 appendices. The total volume of the work is 89 pages. The work contains 12 tables and 36 figures.

The goal of the work is to study the efficiency and speed of existing algorithms for face recognition in images and to develop an authentication system using face recognition algorithms.

The object of research is the technology of identification of a person by biometric parameters of the face in the systems of authentication and access control.

The subject of research is algorithms and methods of face detection in the image and direct recognition of found faces.

Work results:

- A study of the accuracy and speed of existing face recognition algorithms based on popular neural networks.
- A face-to-face authentication system has been developed, with the possibility of integration into existing systems that require authentication.

Approbation of work. The results of the master's work (practical developments and research) were presented at conferences:

1. International Scientific Symposium "Intellectual Solutions" (Uzhgorod, April 15-20, 2019) [51];
2. International scientific-practical conference "Information technologies and interactions" (Kyiv, December 20, 2019) [52].

According to the materials of the master's work there are 2 publications in the collections of materials of scientific and practical conferences.

Keywords.

Neural network, face recognition, authentication system, convolutional neural network.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Аутентифікація (з англ. authentication) - процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.

Ідентифікація (з англ. identification) - процедура розпізнавання суб'єкта за його ідентифікатором.

Авторизація (з англ. authorization) - надання доступу до будь-якого ресурсу.

API (з англ. Application Programming Interface) - Інтерфейс прикладного програмування.

ASM - (з англ. Active Shape Models) - Моделі активної форми.

CNN - (з англ. Convolutional Neural Network) - Згорткова нейронна мережа.

CRUD - (з англ. create, read, update, and delete) - 4 базові функції управління даними «створення, зчитування, зміна і видалення».

DDFD (з англ. Deep Dense Face Detector) - Детектор облич глибокої щільності

FAR (з англ. False Acceptance Rate) - Коефіцієнт помилкового допуску, помилка другого роду.

FRR (з англ. False Rejection Rate) - Коефіцієнт помилкової відмови, помилка першого роду.

GUI (з англ. Graphical User Interface) - Графічний інтерфейс користувача.

HOG (з англ. Histogram of oriented gradient) - Гістограма орієнтованих градієнтів.

LBP (з англ. Local binary patterns) - Локальні бінарні шаблони.

LFW (з англ. Labeled Faces in the Wild) - “Маркеровані обличчя у дикій природі”. База даних для навчання та тестування НМ по розпізнаванню облич.

MTCNN (з англ. Multi-task Cascaded Convolutional Networks) - Багатозадачні каскадні згорткові мережі.

RBF NN (з англ. Radial Basis Function Neural Networks) - НМ з використанням радіальних базисних функцій.

RCNN (з англ. Retinal Connected Neural Network) - Сітківка з'єднана з нейронною мережею.

RINN (з англ. Rotation Invariant Neural Network) - Інваріантна нейронна мережа обертання.

ResNet (з англ. Residual Network) - Залишкова нейронна мережа.

VGG (з англ. Visual Geometry Group) - Кафедра інженерних наук, Оксфордський університет.

БД - База даних.

БНМ - Багатошарова нейронна мережа.

ЗНМ - Згорткова нейронна мережа.

НМ - Нейронна мережа.

ПЗ - Програмне забезпечення.

Препроцессинг - попередня обробка.

МГК / PCA - (з англ. principal component analysis, PCA) - Метод головних компонент.

ВСТУП

В даний час широко використовуються біометричні системи ідентифікації особистості. Звичайні системи ідентифікації вимагають знання пароля, наявності ключа, посвідчення особи або інших елементів ідентифікації, які можуть бути забуті або загублені. Натомість, біометричні системи спираються на унікальні біологічні характеристики людини, які важко імітувати, і які однозначно ідентифікують конкретну людину. Ці функції включають, наприклад, відбитки пальців, форму долоні, геометрію особи, зображення райдужної оболонки, зображення сітківки.

Завдання розпізнавання обличчя людини в природному або штучному середовищі і її ідентифікації завжди була однією з першочергових задач для дослідників, що працюють в області систем штучного зору і штучного інтелекту. Незважаючи на близькість завдань і методів, використовуваних при розробці альтернативних систем біометричної ідентифікації людини, таких як ідентифікація за відбитками пальців або зображення райдужної оболонки, системи ідентифікації особи явно поступаються цим системам. Тому проблема дослідження, вдосконалення та розробки сучасних методів і технологій розпізнавання людей по фото обличчя є актуальним завданням.

Існуючі системи аутентифікації за допомогою розпізнавання облич переважно орієнтовані на середній та великий бізнес, що зумовлює їх велику ціну та недоступність малому бізнесу і невеликим підприємствам. Зазвичай, вони пропонують свої послуги разом з обладнанням і зі щомісячною оплатою, що також накладає додаткові обмеження і збільшує витрати на обслуговування, яке може бути проведене лише власником системи. Залежність від системи, над якою немає повного контролю, призводить до додаткових ризиків, оскільки всі дані, зазвичай, зберігаються на зовнішніх серверах і при їх компрометації можливий витік конфіденційних даних. Щоб запобігти цьому, необхідно організувати процес аутентифікації таким чином, щоб всі компоненти лежали в межах інфраструктури компанії і не залежали від зовнішньої інфраструктури.

Метою роботи є вибір існуючого алгоритму розпізнавання облич на статичних зображеннях шляхом дослідження їх ефективності і швидкості роботи та розробка системи аутентифікації за допомогою обраного алгоритму.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

- провести аналіз існуючих системи для аутентифікації за обличчям;
- провести аналіз технології аутентифікації за обличчям та методів розпізнавання обличчя;
- дослідити ефективність та швидкість роботи існуючих алгоритмів розпізнавання облич на зображеннях;
- розробити систему аутентифікації за обличчям;
- дослідити ефективність та надійність розробленої системи.

Структура роботи.

В першому розділі був проведений аналіз різних методів у розпізнаванні облич. Також було проведено порівняння популярних реалізацій нейромереж, їх швидкості та точності і обрано метод розпізнавання облич для реалізації системи аутентифікації.

У другому розділі було описано алгоритм розпізнавання обличчя, використаний у системі та саму архітектуру системи аутентифікації за допомогою розпізнавання обличчя. Також описано процес реєстрації нового користувача та проходження аутентифікації.

У третьому розділі аргументовано вибір інструментів для розробки системи аутентифікації (мова програмування, СУБД, необхідні бібліотеки, фреймворки та інше). Описано схему бази даних, показані можливі способи інтеграції системи в інші системи та показані приклади роботи системи.

У висновках підводиться підсумок проведеної роботи та вказується напрямок майбутнього руху.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ТА СИСТЕМ РОЗПІЗНАВАННЯ ОБЛИЧ НА ФОТО

1.1 Узагальнена технологія розпізнавання облич на фото

Біометрична аутентифікація - це процес тестування і аутентифікації шляхом подання користувачем його відповідного біометричного зображення і перетворення цього зображення відповідно до попередньо визначеного протоколу аутентифікації.

Біометричні системи складаються з двох частин: апаратного забезпечення і спеціалізованого програмного забезпечення. До апаратного забезпечення входять сканери, термінали та інші пристрої, які записують той чи інший біометричний параметр і перетворюють отриману інформацію в цифрову модель, придатну для зчитування комп'ютером. Програмне забезпечення, в свою чергу, обробляє ці дані, порівнює їх з базою даних і приймає рішення, хто знаходиться перед терміналом.

Щоб біометрична система могла ідентифікувати користувача в майбутньому, необхідно спочатку записати інформацію про його ідентифікатори. Комерційні системи (на відміну від використовуваних правоохоронними органами і службами безпеки) зберігають їх цифрові моделі, а не зображення реальних ідентифікаторів. Коли користувач знову звертається до системи, модель ідентифікатора заново відтворюється і порівнюється з моделями, раніше збереженими в базі даних.

Одним з видів систем біометричної аутентифікації, який широко розповсюджуються в наш час, є розпізнавання облич на фото чи відео.

Розпізнавання облич - це автоматизована локалізація людського обличчя на фото або відеопотоці і, при необхідності, ідентифікація особистості людини на основі існуючих даних в базі даних.

Розпізнавання людини по фото обличчя виділяється серед біометричних систем тим, що, по-перше, не потрібно дорогого або спеціального устаткування. Для більшості застосунків достатньо звичайного персонального комп'ютера з

відеокамерою. По-друге, фізичний контакт з пристроями не потрібен. Вам не потрібно нічого торкатися або зупинятися і чекати, поки система запрацює. Достатньо зупинитися перед камерою на кілька секунд або просто пройти мимо.

Технології розпізнавання облич застосовуються в різних сферах, серед них:

1. фототехніка;
2. Пошук конкретного фото у великому наборі даних;
3. пошук потенційно небезпечних і підозрілих відвідувачів, фейс-контроль в сегменті розваг та громадського харчування;
4. криміналістика;
5. онлайн-платежі;
6. цифровий маркетинг, контекстна реклама;
7. телеконференції;
8. валідація банківських карт;
9. ідентифікація людей на фото в соцмережах.
10. контроль місць великого скупчення людей;
11. пошук зловмисників, системи попередження незаконного проникнення на територію об'єкта;
12. мобільні додатки;

Завдання розпізнавання людей по фото облич діляться на три види: контроль доступу, контроль фотографій в документах і пошук в великих базах даних. Ці завдання відрізняються як по системним вимогам розпізнавання, так і за методами вирішення і тому є окремими класами. Вимоги до помилок першого і другого типу для таких класів те відрізняються. Помилка першого типу (неправильне виявлення) - це ситуація, в якій об'єкт даного класу не розпізнається в системі. Помилка другого роду - це помилка, коли об’єкт одного виду ідентифікується як об’єкт іншого виду.

Потрібно також вказати відмінність у поняттях розпізнавання (ідентифікації)[1] і верифікації. При верифікації система підтверджує або спростовує той факт, що невідомий об'єкт належить до деякого, відомого системі,

класу. При розпізнаванні об'єкт, що належить невідомому класу, потрібно віднести до одного з відомих класів або зробити висновок, що цей об'єкт не належить ні до одного з відомих класів.

Велика змінність візуальних зображень є основною проблемою для систем штучного зору. Вона пов'язана зі змінами освітлення, кольору, масштабу і кутів огляду. Крім того, люди мають звичку ходити одягненими як по вулицях, так і в приміщенні, що призводить до значної зміни вигляду одного і того ж зображення людини. Однак найбільш важким завданням машинного зору є усунення неоднозначності, що виникає при проектуванні тривимірних об'єктів реального світу на плоских зображеннях. Яскравість і колір окремих пікселів на зображенні також залежить від великої кількості факторів, що важко піддаються прогнозуванні. До них належать:

- інтенсивність випромінювання і колір;
- розташування джерел світла і їх кількість;
- віддзеркалення від інших об'єктів та тіні.

Велика кількість даних, що містяться в зображенні значно ускладнює виявлення об'єктів на ньому. Зображення може містити тисячі точок, кожна з яких може бути важливою. Для використання всієї інформації, що міститься в зображенні, потрібен аналіз кожного пікселя на предмет приналежності його фону або об'єкту, при цьому враховуючи можливу змінність об'єктів. Такий аналіз потребує великої кількості пам'яті, необхідної для роботи алгоритму.

У роботі будь-якого алгоритму розпізнавання [2][3] або виявлення [4] особи слід виділити два логічних блоки: екстрактор характерних ознак і механізм класифікації [5][6]. Дія екстрактора заснована на виділенні з величезного потоку вхідних даних корисної для класифікатора інформації. При ідентифікації особистості цією інформацією можуть бути характеристики ознак, що виділяються однозначно (наприклад, застосовуване в криміналістиці відносне розташування очей, брів, губ і носа). Класифікатор, при прийнятті рішення про призначення мітки класу об'єкту, що розпізнається, повинен керуватися саме цими ознаками. Вибір ознак є найбільш важливим завданням. Очевидно, що при

їх виборі враховуються найбільш унікальні властивості, оскільки за ним можливо найбільш достовірно судити про належність об'єкта до того чи іншого класу.

Сам процес розпізнавання складається з трьох етапів: знаходження і локалізація області обличчя на фото, виділення ключових особливостей знайденого обличчя (ніс, очі, губи тощо) та кодування знайдених особливостей в одновимірний масив (так званий face encoding або вектор ознак) для подальшої ідентифікації особи.

1.2 Аналіз існуючих систем аутентифікації по обличчю

Більшості існуючих систем аутентифікації по обличчю є платними комерційними коробковими продуктами. Тому для аналізу було обрано системи, які є або повністю безкоштовними, або надають можливість безкоштовного використання протягом певного періоду. Даний список включає:

1. OpenBR (розробник Openbiometrics[7]);
2. Flandmark (Center for Machine Perception @ CTU in Prague[8]);
3. OpenFaceTracker (OpenFaceTracker[9]);
4. OpenEBTS (Open Biometrics Initiative[10]);
5. Bioenable Tech - iFace (bioenabletech[11]);
6. Bioenable Tech - vFace (bioenabletech[12]);
7. FacePlus (FasePlusPlus[13]);
8. DeepFace (DeepFace[14]).

Основні технологічні характеристики даних систем наведено у таблиці 1.1.

Таблиця 1.1 - Технологічні характеристики існуючих безкоштовних систем аутентифікації по обличчю

Назва	Розгортання	ОС	Особливості
OpenBR	Open-API, On-Premise	Windows, Mac, Linux	NIST-сумісне програмне забезпечення
Flandmark	Open-API	Windows,	Використовує

		Linux	Deformable Part Models (DPM)
OpenFaceTracker	Open-API	Windows	Розроблено у вигляді модульної бібліотеки
OpenEBTS	Open-API	Windows, Web-Based	Містить 2 відкриті API: OpenEBTS API та OpenEM1 API
Bioenable Tech - iFace	Open-API, On-Premise	Web-Based	Залежить від апаратного забезпечення
Bioenable Tech - vFace	Open-API, On-Premise	Web-Based	Залежить від апаратного забезпечення
FacePlus	Cloud-Hosted	Windows, Mac, Web-Based	Система захисту від шахрайства
DeepFace	Cloud-Hosted	Windows, Mac, Web-Based	Використовує БД облич для пошуку по фото чи імені

Перед порівнянням необхідно сформувати критерії, яким повинна задовольняти майбутня система:

1. системи повинна бути в публічному доступі;
2. можливість формування власної бази облич;
3. гнучкість інтеграції з іншими системами;
4. висока точність та швидкість обробки вхідних зображень;
5. можливість безкоштовного використання;
6. робота в оффлайн режимі.

В таблиці 1.2 наведено відповідності вищеописаних систем заданим критеріям:

Таблиця 1.2 - Відповідність оглянутих систем заданим критеріям

Назва	Публічний доступ	Формування бази	Гнучкість інтеграції	Висока точність та швидкість	Безкоштовне використання	Оффлайн режим
OpenBR	+	-	+	+	+	+
Flandmark	+	-	+	+/-	+	+
OpenFaceTracker	+	-	+	+/-	+	+
OpenEBTS	+	-	+	+	+	+
Bioenable Tech - iFac	+/-	+	+	+	-	-
Bioenable Tech - vFace	+/-	+	+	+	-	-
FacePlus	+/-	-	+	+	+/-	-
DeepFace	+/-	+	+	+	-	-

Більшість оглянутих систем вже давно не оновлюється, що ускладнює їх встановлення та використання. Також відсутність можливості створення бази облич унеможливлює їх використання в якості системи аутентифікації. Саме тому було вирішено розробити власну систему, яка задовольняє всім заданим вимогам.

1.3 Аналіз нейронних мереж для розпізнавання облич

Нейронні мережі відомі своєю точністю.[15] В наступних розділах описано найпопулярніші мережі, що використовуються для розпізнавання облич.

1.3.1 Метод головних компонент з використанням нейронних мереж

Метод головних компонент (МГК, англ. principal component analysis, PCA) - це метод зменшення розмірності заснований на перетворенні Карунена-Лоєва. У задачі розпізнавання осіб його переважно застосовують для представлення зображення особи за вектором малої розмірності (головні компоненти), який потім порівнюється зі збереженими у базі даних еталонними векторами [16].

Основна мета методу головних компонент полягає в тому, щоб значно зменшити розмірність простору ознак, щоб вони описували «типові» характеристики, властиві великій кількості людей, найкращим чином. Використовуючи цей метод, можна ідентифікувати різні ознаки в наборі навчальних зображень певної особи і описати ці ознаки на основі декількох ортогональних векторів, що називаються власними.

При такому підході для кожної людини навчання нейронної мережі відбувається на основі самого обличчя особи. В майбутньому така мережа буде визначати схожість особи, на якій була навчена нейронна мережа з довільною особою. Незважаючи на високу точність методу, він має суттєвий недолік: нейронна мережа потребує окремої моделі для кожної особи.

1.3.2 Детектор обличчя глибокої щільності

Детектор обличчя глибокої щільності (Deep Dense Face Detector, DDFD) базується на глибокій згортковій нейронній мережі і використовується для виявлення облич на фото. Ключовою особливістю є висока точність мережі в класифікації і виділенні характеристик для створення єдиного класифікатора для виявлення осіб при відносно невеликому використанні обчислювальних ресурсів, за рахунок простішої архітектури детектора. За основу автором було взято архітектуру AlexNet і перенавчано для розпізнавання облич на фото. Одна

людина в класі (одне й те ж саме фото обличчя) використовувалася для навчання в різних умовах освітлення/ракурсу і так далі. Для збільшення розміру вибірки використовувалися різні методи модифікації облич [17].

Не зважаючи на те, що відсоток точності розпізнавання значно знижувався при сильному повороті обличчя на фото, метод був стійкий до різних типів освітлення і загалом показав гарні результати в більшості випадків. Також існують різні підходи для роботи з вхідними даними, які допомагають у вирішенні цієї проблеми. Недоліком також є те, що мережу потрібно тренувати по-новому при додавання нових облич.

1.3.3 Нейронні мережі з використанням радіальних базисних функцій

Нейронні мережі з використанням радіальних базисних функцій мають одні з найвищих показників в точності і загалом гарно себе зарекомендували. В якості попередньої обробки, використовується комбінований підхід на базі методу головних компонент і моделей активної форми (від англ. Active Shape Models, ASM). Спочатку відбувається виділення контуру та форм обличчя для створення особистого профайлу за допомогою моделей активної форми, а потім використовується PCA алгоритм, щоб зменшити розмірність відображень осіб [18].

Частина ідентифікації використовує нейронні мережі з радіальної базисної функцією (RBF NN) для ідентифікації унікального шаблону, пов'язаного з кожною людиною.

1.3.4 Каскадні згорткові нейронні мережі

Z. Linz, G. Hua, X. Shenz, H. Liy, J. Brandtz, [19] запропонували нейронну мережу «Cascade» для розпізнавання осіб. Цей метод має потужні функціональні можливості виділення ознак при збереженні високої продуктивності. Даний каскад в згортковій нейронній мережі, вміє швидко виділяти фонові області на перших етапах, але детально враховує невелику кількість кандидатів на останньому етапі. Автор представив крок калібрування мережі, щоб підвищити

ефективність і зменшити кількість кандидатів на більш пізніх етапах. В результаті калібрувальні мережі забезпечують більш точне місце розташування обличчя на фото, використовуючи великі вікна сканування з меншим масштабуванням [20].

1.3.5 Білінійні згорткові нейронні мережі

A. Roy C. Tsung, Y. Lin, S. Maji, E. Miller [21] запропонували метод білінійної згорткової нейронної мережі для ідентифікації особи, який дав гарний ріст продуктивності на фото малої розмірності. Модель зазвмчай використовує алгоритм на основі двох згорткових нейронних мереж, що значно зменшує розрив між текстурами і деталями (рисунок 1.1).

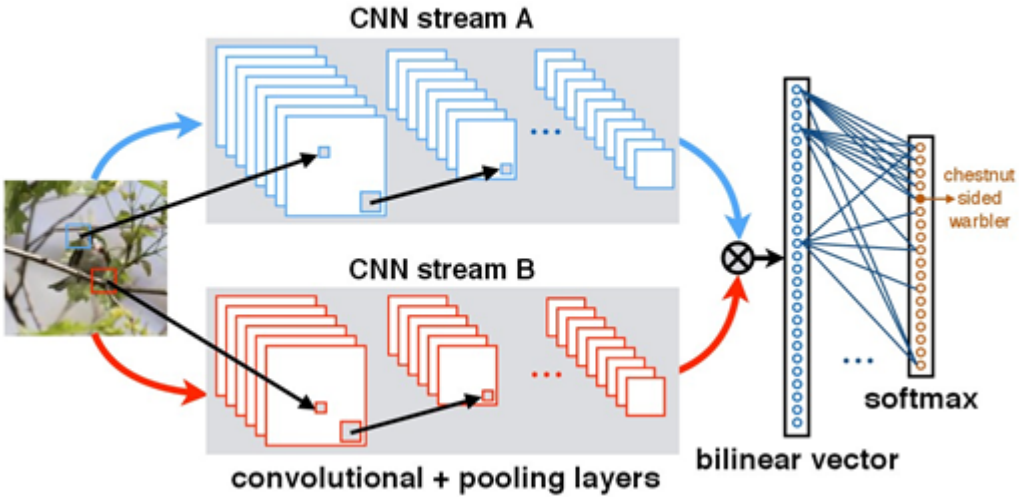


Рисунок 1.1. Структура Bilinear CNNs [22]

В основі архітектури лежить орієнтований ациклічний граф, з використанням зворотного градієнтного поширення обидві мережі можуть навчатися одночасно. Замість того, щоб вивчати згорткову нейронну мережу для розпізнавання осіб з нуля, що вимагає знаходження ідеальної архітектури нейронної мережі і масивних баз даних, білінійні згорткові нейронні мережі можуть використовувати попередньо підготовлені мережі і адаптувати їх до задачі розпізнавання обличчя.

1.3.6 Мережа зворотного поширення та нейронна мережа з використанням радіальних базисних функцій

M. Nandini, P. Bhargavi, G. Sekhar [23] запропонували комбінацію мережі зворотного поширення та нейронної мережі з використанням радіальних базисних функцій для розв’язку задачі розпізнавання осіб. Цей метод пропонує

новий підхід до розпізнавання людських облич. Розпізнавання виконується шляхом порівняння особливостей уже відомого обличчя з новим. Навчання нейронної мережі виконується з використанням мереж зворотного поширення і радіальної базисної функції (RBF). Нейронні мережі з радіальними базисними функціями є дуже привабливими для інженерних завдань. У них дуже компактна топологія, універсальні підходи і дуже висока швидкість навчання. Нейронна мережа зворотного поширення має тришарову архітектуру (вхідний шар, прихований шар і вихідний шар). Радіальна базисна функція використовується в якості ідентифікатора в системі розпізнавання осіб, і входи в цю мережу є результатами, отриманими в мережі зворотного поширення. Дана модель має досить високу точність розпізнавання, що дозволяє розширити і використовувати цей метод для зображень з різними видами фону, включаючи відео [24].

1.3.7 Нейронна мережа, з'єднана сіткою

Henry A. Rowley запропонував нейронну мережу, з'єднану сіткою (Retinal Connected Neural Network, RCNN) [25], розбиває зображення на вікна і перевіряє, чи містить кожне вікно фото обличчя. Першим компонентом системи є фільтр, який приймає в якості вхідних даних пікселі області зображення розміром 20×20 і генерує вихідний сигнал в діапазоні від 1 до -1, що вказує на наявність або відсутність особи. Щоб ідентифікувати особу в будь-якому місці на вході, фільтр застосовується в кожній точці зображення. Для ідентифікації осіб, розмір яких перевищує розмір вікна, вхідне зображення багаторазово зменшується. Фільтр не повинен змінювати положення і масштаб [26].

Основним недоліком існуючої системи є те, що вона розпізнає тільки вертикальні особи, дивіться на камеру [27]. Окремі версії систем можуть бути навчені для кожної орієнтації голови, і результати можуть бути об'єднані з використанням методів арбітражу [28].

1.4 Вибір нейронної мережі та її реалізації

В основі системи біометричної ідентифікації особи по її фотознімку лежить модуль розпізнавання облич. Для його реалізації потрібно обрати оптимальний алгоритм, що надає високу точність за максимально короткий час обробки зображення. Було виконано порівняння існуючих рішень для розпізнавання облич на фото. Найкраще себе зарекомендували методи засновані на нейронних мережах, тому далі зупинимося на їх порівнянні. Всі описані вище мережі мають існуючі реалізації на популярних мовах програмування. В даному розділі буде проведено порівняння популярних реалізацій нейронних мереж для детекції облич та виділення ключових особливостей:

- Face-recognition[29]
- MTCNN[30]
- Yoloface[31]
- Faced[32]
- Ultra light face detector[33]

Основний критерій формування даного списку, є можливість всіх реалізацій ефективно працювати, використовуючи лише ресурси процесора. Як відомо, для навчання нейронної мережі потрібні значні обчислювальні ресурси, тому зазвичай їх будують на фреймворках, які дозволяють використовувати ресурси графічного процесора. Вже навченій моделі, зазвичай, потрібно менше ресурсів і для роботи достатньо процесорної потужності. Сервер з графічним процесором не тільки збільшує фінансові витрати на його утримання, а й потребує більших затрат сил та часу при розробці та підтримці програмного продукту. Саме тому в даний список не потрапили методи, для роботи яких, наявність графічного процесора є необхідністю.

1.4.1 Face recognition

Дана реалізація поєднує в собі детектор обличчя і екстрактор ключових особливостей з подальшим їх перетворенням в вектор ознак.

В якості моделі використовується мережа ResNet з 29 шарами. Це, версія мережі ResNet-34[34] з кількома видаленими шарами, а кількість фільтрів на шар зменшено вдвічі.

Мережа навчалася з нуля на основі даних близько 3 мільйонів облич. Цей набір даних було отримано об'єднанням кількох наборів. Набір даних для скрабу (scraping) для обличчя, набір даних VGG, та велика кількість зображень, які були зібрані власноруч Девісом Кінгом (Davis King). Комбінований набір даних був очищений для усунення помилок маркування. Це було зроблено неодноразово тренуючи модель розпізнавання облич, а потім використовуючи методи кластеризації графів і ручної перевірки для очищення набору даних. Загальна кількість індивідуальних ідентифікаційних даних у наборі даних становить 7485.

Навчання мережі почалося з випадково ініціалізованих ваг і використовувало структуровану метричну втрату, яка намагається проеціювати всіх осіб в кола, що не перетинаються і мають радіус 0.6. Для навчання було використано бібліотеку dlib[35].

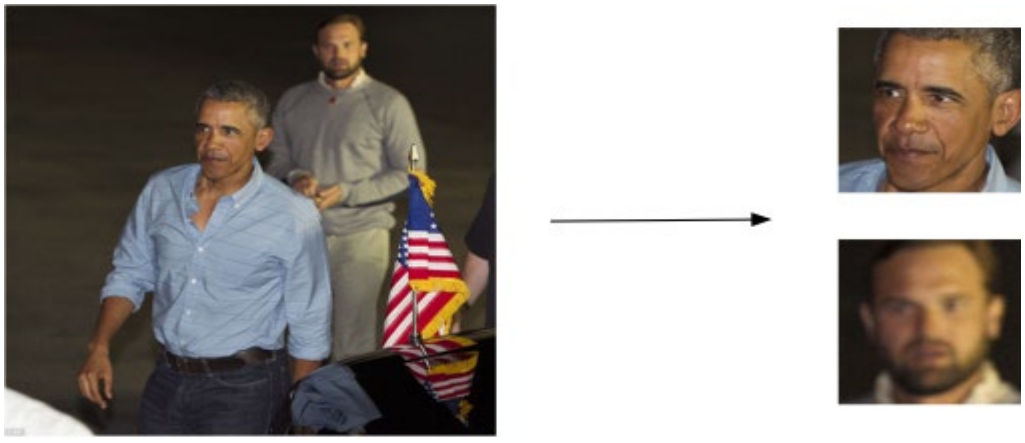


Рисунок 1.2. Приклад детекції обличчя[29]

1.4.2 MTCNN

Реалізація детектора обличчя MTCNN, написана з нуля, використовуючи як орієнтир реалізацію MTCNN від Девіда Сандберга з Facenet. Реалізація побудована на роботі Zhang, K et al. (2016) [ZHANG2016]. Мережа також здатна виділяти такі ключові особливості, як праве око, ліве око, ніс та контури рота.

Автором заявлена швидкість обробки в 8.798 секунд одного зображення розміром 4799x3599 пікселів, що містить одне обличчя.

Рисунок 1.3. Приклад детекції обличчя[30]

1.4.3 Faced

Faced складається з 2 нейронних мереж, обидві реалізовані за допомогою Tensorflow. Основа архітектури Faced, в значній мірі заснована на архітектурі YOLO. В основному, це повністю згоркова мережа (Fully Convolutional Network, FCN), яка запускає вхідне зображення розміром 288x288 через серію згорткових та об'єднуючих шарів (convolutional and pooling layers) (інші типи шарів не беруть участь).

Згорткові шари відповідають за виділення просторових особливостей. Об’єднуючі шари збільшують поле прийняття послідовних згорткових шарів.

Вихід архітектури - сітка розміром 9x9 (проти 13x13 в YOLO). Кожна комірка сітки відповідає за те, щоб передбачити, чи знаходиться обличчя всередині цієї комірки (проти YOLO, де кожна комірка може виявити до 5 різних об'єктів).

Кожна комірка сітки має 5 пов'язаних значень. Перше - це ймовірність *p*, що дана комірка містить центр обличчя. Інші 4 значення - це x та y центру, ширина та висота виявленого обличчя (відносно комірки).

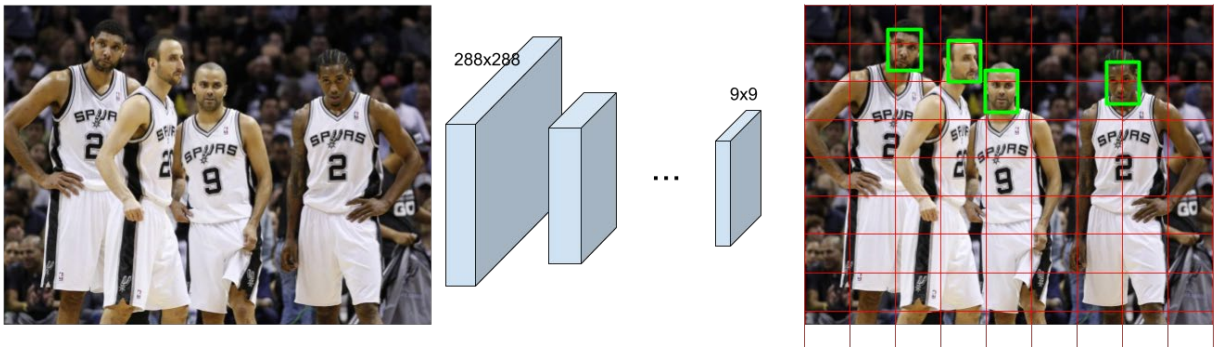


Рисунок 1.4. Основна архітектура Faced[32]

Виходи (x_center, y_center, ширина, висота) основної мережі були не настільки точними, як очікувалося. Тому була реалізована допоміжна мережа CNN, щоб взяти за вхід невелике зображення, що містить обличчя (обрізане виходами головної архітектури) та обмежити його границі ще раз до розміру самого обличчя. Єдине завдання допоміжної мережі - доповнити і вдосконалити вихідні координати основної мережі.

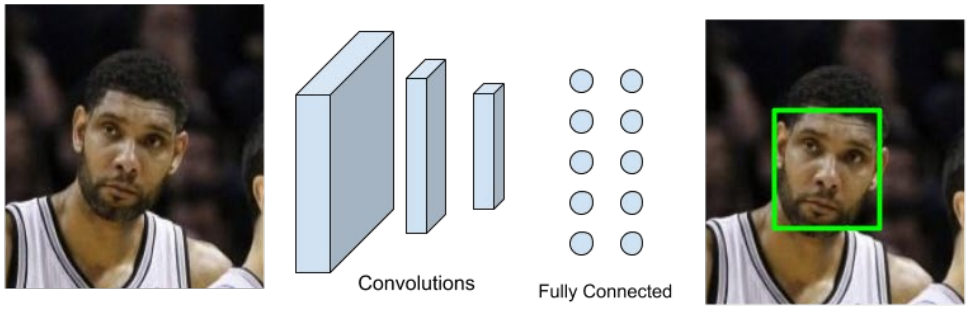


Рисунок 1.5. Обмеження рамки з обличчям допоміжною мережею[32]

1.4.4 Yoloface

Модель, що використовує глибинне навчання для виявлення обличчя за допомогою алгоритму YOLOv3. Сама по собі, модель YOLOv3 (You Only Look Once) - це найсучасніший алгоритм виявлення об'єктів у режимі реального часу. Опублікована модель розпізнає 80 різних об’єктів на зображеннях та відео. На відміну від оригінального YOLOv3, Yoloface було навчено розпізнавати виключно обличчя.

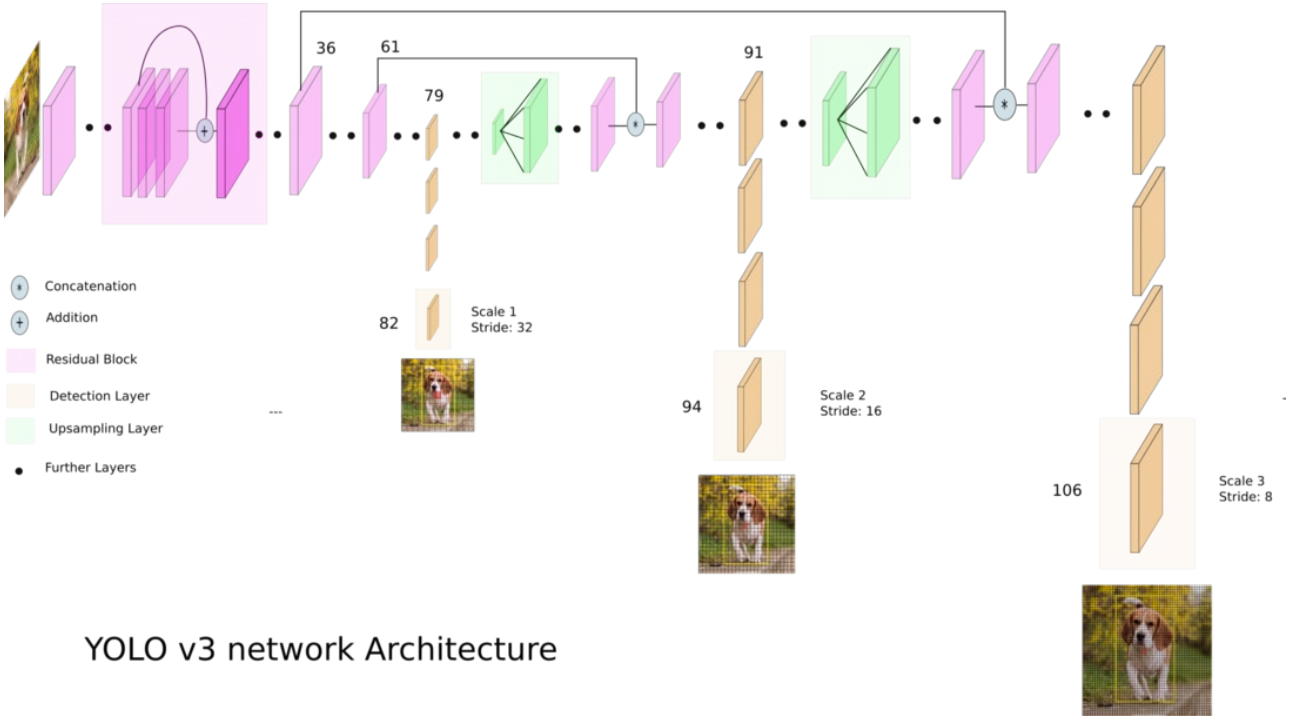


Рисунок 1.6. Архітектура мережі YOLOv3[31]

1.4.5 Ultra light face detector

Ultra light face detector - це ультралегка універсальна модель виявлення облич у режимі реального часу, призначена для обчислювальних пристроїв та пристроїв малої потужності (таких як ARM).

Основні особливості:

1. Що стосується розміру моделі, розмір файлу точності FP32 за замовчуванням (.pth) становить 1,04 ~ 1,1 МБ, а вихідний кадр int8 - близько 300 КБ.
2. При обчисленні, при вхідній роздільній здатності вхідного зображення 320x240, кількість операцій приблизно 90 ~ 109 MFlops.
3. Існує дві версії моделі: легка (трохи швидша) та версія-RFB (зі зміненим модулем RFB, більш висока точність).
4. В залежності від сценарія застосування, можна використовувати модель, що підтримує вхідні зображення розміром 320x240 та 640x480 пікселів.

1.5 Порівняння існуючих реалізацій розпізнавання облич

Для тестування було обрано датасет Labeled Faces in the Wild[36]. Даний датасет містить 13233 зображення, 5749 різних осіб (одна особа може мати декілька фото облич) та 1680 людей з двома і більше зображеннями.

Кожне зображення має розмір 250x250 jpg, виявлене та відцентроване за допомогою реалізації детектора обличчя Viola-Jones в бібліотеці OpenCV. Після знаходження детектором рамки обличчя, вона автоматично збільшується на коефіцієнт 2.2 у кожному вимірі для захоплення більшої частини голови, а потім масштабується до рівномірного розміру.

Рисунок 1.7. Приклад зображень облич з датасету[36]

Для тестування використовувався комп’ютер з наступною конфігурацією:

- ОС Windows 10
- CPU Intel Core i5 7200
- RAM 16GB
- Python 3.7.1

Тестування відбувалося наступним чином: на вхід кожній нейронній мережі подавалися всі зображення з датасету по одному. Замірявся час кожної детекції, а також зберігався результат детекції (було обличчя знайдено чи ні). В час не враховувалися зчитування та запис даних на/з диску. В кінці всі результати сумувалися і заносилися до порівняльної таблиці.

В таблиці 1.3 показані результати детекції облич на фото різними реалізаціями.

Таблиця 1.3 - Результат обробки всього датасету кожним методом

Метод	Час детекції, с	Знайдено облич	Відсоток від макс.
Face Recognition	592.82	14,088.00	0.8941355674
mtcnn	983.88	15,756.00	1
ultra_light_640	1,220.09	15,655.00	0.9935897436
faced	2,104.51	14,101.00	0.8949606499
yolov3-face	12,577.19	15,647.00	0.9930820005

Як видно з результатів, найшвидшим виявився метод Face Recognition, хоч він і знайшов 89.4% облич від максимального результату. Метод mtcnn знайшов максимальну кількість облич, але був на 60% повільнішим за Face Recognition. Для остаточного рішення проведемо ще один тест детекції облич всіма методами на фото, що містить великий натовп людей. Для уникнення неточних результатів через затримки нейронних мереж при старті, кожен тест був повторений 5 разів після повної ініціалізації НМ.

Рисунок 1.8. Результат детекції методами(зліва направо) faced, ultra_light_640, mtcnn, face recognition, yoloface

Таблиця 1.4 - Результат обробки фото з натовпом кожним методом

Метод	Час, с	Облич знайдено
Face Recognition	1.111	74
mtcnn	3.461	123
ultra_light_640	0.115	80
faced	0.762	17
yolov3-face	1.24	129

Як видно з результатів ultra_light_640 та faced виявилися найшвидшими при пошуку великої кількості облич на одному фото. mtcnn та yolov3-face знайшли найбільшу кількість облич при цьому були досить повільними. Вони також знайшли велику кількість облич, які були непридатні для ідентифікації особи:

Рисунок 1.9. Приклад частково видимого обличчя

Для систем аутентифікації більш характерний сценарій обробки великої кількості зображень та детекція лише повноцінних зображень облич у положенні, що придатне для розпізнавання. Саме тому для детекції облич при розробці системи аутентифікації в цій роботі було вирішено використати метод face recognition, оскільки він дає достатньо точні та стабільні результати детекції облич та є найшвидшим при обробці великої кількості вхідних зображень.

При побудові моделі розпізнавання обличчя існує декілька підходів. Два популярних підходи - навчання мережі на розпізнавання однієї особи і побудова

узагальненої моделі, яка здатна виділяти унікальні характеристики обличчя з фото різних людей.

При першому підході, на вхід моделі подаються фото облич особи, яку планується ідентифікувати та різні обличчя невідомих людей і модель навчається відрізняти основну особу від всіх інших.

При другому підході навчання такої мережі виконується на 3 зображеннях одночасно:

1. Перше зображення відомої особи
2. Друге зображення відомої особи
3. Зображення невідомої особи

Процес навчання відбувається таким чином, щоб виміри, отримані для облич 1 і 2 були ближчими, ніж виміри для облич 2 і 3. Після повторення цього кроку мільйони разів на мільйонах зображень тисяч людей, мережа вчиться надійно генерувати 128 вимірів для кожної особи так, що будь-які 10 зображень однієї особи даватимуть більш-менш однакові виміри. Після такого навчання мережа вмітиме генерувати вектор ознак для будь-яких облич.

При першому підході, єдиний спосіб додати підтримку розпізнавання багатьох осіб, це навчати мережу з початку для кожної нової особи. Це не лише додасть значного навантаження, оскільки для навчання потрібні обчислювальні потужності графічного процесора і це займає дуже багато часу (мова йде про години, а то й дні), а й в принципі неможливо. Для успішного навчання нейронної мережі потрібен якісний набір даних. Він має бути правильно збалансований і містити велику вибірку всіх категорій для запобігання перенавчання. Очевидно, що реєстрація нового користувача не повинна вимагати від нього сотень, а то й тисяч зображень його облич з різного ракурсу, освітлення і тд.

Більш практичним є використання узагальненої моделі, що здатна генерувати унікальні характеристики облич для різних людей. В публічному доступі є багато готових реалізацій таких мереж, проте багато з них були розроблені ентузіастами і не виділяються великою точністю та швидкістю. Інші

ж потребують значних ресурсів при роботі або повинні навчатися на власному наборі даних. Розробка архітектури мережі - це лише частина роботи. Велику роль в роботі мережі відіграють дані, на яких вона була навчена. Побудова якісного датасету потребує не лише багато часу, а й знання області застосування та архітектури мережі, яка буде навчатися на цьому датасеті. В даній роботі буде використано мережу з того ж пакету face recognition. Детальніше про архітектуру та принцип роботи методу розпізнавання буде розказано в наступному розділі, а поки проведемо тести швидкості та ефективності.

В якості вхідного набору даних буде використано ту саму базу даних облич Labeled Faces in the Wild. Дана база також містить спеціальні набори зображень, які називаються ‘Видами’ (Views), які були підібрані спеціально для конкретних цілей. В нашому тесті буде використано ‘Вид 2’. Він призначений для отримання оцінки ефективності розпізнавання і його слід використовувати лише для оцінки методу, а не для навчання.

Даний набір був створений наступним чином: базу даних випадково розділили на 10 наборів (рівномірно на рівні однієї особи). Потім, для кожного набору, випадковим чином було обрано 300 пар, що відповідають одній особі, і 300 пар, що належать різним особам попарно. Використовуючи такий розподіл, оцінити ефективність алгоритму можна за допомогою 10-кратної перехресної перевірки.

Рисунок 1.10. Приклад пари зображень, що відповідають одній особі (зліва) та двом різним особам (справа)[36]

Таблиця 1.5 - Результат роботи face recognition

Параметр	Значення
Час розпізнавання, с	5,743.40
Час розпізнавання одного фото, с	0.4786
Всього облич	12,000
Розпізнано облич	11,954
Відсоток розпізнаних облич	99.62%
Однакових пар розпізнано	2,896
Однакових пар всього	3,000
Відсоток розпізнавання однакових пар	96.53%
Різних пар розпізнано	2,950
Різних пар всього	3,000
Відсоток розпізнавання різних пар	99.33%

З даних результатів видно, що модель успішно розпізнала 99.62% облич з середнім часом 0.478 секунди на фото. Відсоток невдалих розпізнавань однакових пар (два фото однієї особи сприймалися як різні особи), що відповідає помилці першого роду, становить 3.47%. Відсоток невдалих розпізнавань різних пар (два фото різних осіб сприймалися, як одна особа), що відповідає помилці другого роду, становить 0.67%. Очевидно, що для системи аутентифікації помилка другого роду є більш критичною, оскільки існує можливість аутентифікації під ідентифікатором чужого користувача. У випадку помилки першого роду, користувачу потрібно просто спробувати пройти аутентифікацію ще раз. Необхідно зазначити, що в даному тесті використовувалося стандартне значення порогу - 0.6. В готовій системі цей поріг буде збільшено до 0.8, що на порядок зменшить ймовірність помилки другого роду. В плані швидкості метод показав достатні результати для реалізації високонавантаженого сервісу розпізнавання облич.

Висновки до розділу

Був проведений аналіз різних методів у розпізнаванні облич. Методи, побудовані на нейронних мережах, суттєво краще демонструють себе в тестах і мають кращу точність.

Через свою природу, застосування нейронних мереж є більш гнучким і в більшості випадків набагато точнішим. Це зумовлено великою кількістю архітектур мереж, їх параметрами та вхідними наборами даних. Нейронні мережі зарекомендували себе як потужний механізм для класифікації зображень, але протягом останнього часу вони активно використовуються і в інших областях комп'ютерного зору. Їм належать перші місця в змаганнях алгоритмів розпізнавання, таких як ImageNet, а якість розпізнавання близька, а іноді і перевищує людську.

Окрім того, конвуляційні мережі відносно легко вирішують проблеми високоточного розпізнавання, що викликають проблеми у людей, наприклад, визначення породи собаки чи марки автомобіля, та інші задачі, що потребують виділення специфічних ознак.

Серед недоліків можна виділити проблеми при обробці елементів малого розміру, і відсутність можливості справлятись зі спотвореннями, такими як розмиваючий фільтр або шум. Також недоліком є проблема вибору набору даних, який задовольнить проблему, що вирішується та буде досить якісним і великим для навчання нейронної мережі.

Також було проведено порівняння популярних реалізацій нейромереж, їх швидкості та точності. Результати тестів вкотре підтверджують, що рівня розвитку даних технологій достатньо для їх масового впровадження у різні сфери суспільного життя.

Через свої переваги, для розпізнаванні та ідентифікації осіб було вирішено обрати face recognition, який використовує комбінацію нейромереж CNN та ResNet. Дане рішення ляже в основу модуля аутентифікації майбутньої системи.

РОЗДІЛ 2. АРХІТЕКТУРА СИСТЕМИ АУТЕНТИФІКАЦІЇ ТА КОНТРОЛЯ ДОСТУПУ ЗА ДОПОМОГОЮ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ

2.1 Алгоритм розпізнавання обличчя

Перед поданням зображення на модуль розпізнавання обличчя воно проходить наступні етапи обробки.

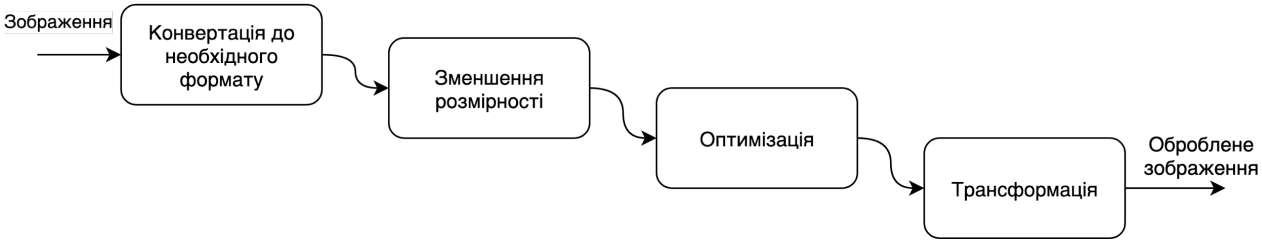


Рисунок 2.1 Основні етапи обробки зображення перед розпізнаванням

Отримане зображення з камери чи іншого джерела конвертується до підтримуваного формату (в нашому випадку png). Роздільна здатність зображення зменшується для прискорення обробки нейромережею. Це відбувається за наступним алгоритмом:

нова_ширина = 800

якщо *ширина_зображення* > *нова_ширина*:

відсоток_збільшення = *нова_ширина* / *ширина_зображення*

нова_висота = *стара_висота* * *відсоток_збільшення*

Таким чином зображення зменшується так, щоб його максимальна ширина була не більша, ніж 800 пікселів, але пропорції зображення зберігаються.

Потім зображення оптимізується та зменшується в якості до 95%. Це ніяк не впливає на якість роботи нейронної мережі, але значно її пришвидшує.

Останній етап це трансформація зображення. В наш час багато фото зберігають інформацію про орієнтацію в EXIF метаданих, замість того, щоб завжди зберігати фото в одній орієнтації. Коли ми відкриваємо якесь фото для перегляданння, воно автоматично повертається програмою для перегляданння в коректну орієнтації відповідно до метаданих. Нейронна мережа працює напряму з байтами зображення і не бере до уваги додаткові метадані крім основних

(алгоритм стиснення PNG, висота, ширина і тд), тому важливим етапом є поворот зображення, щоб обличчя на ньому було у вертикальному положенні.

Оброблене зображення подається на вхід модулю розпізнавання облич, який перетворює його на вектор ознак.

Модуль розпізнавання облич базується на архітектурі, в основі якої лежать наступні етапи:

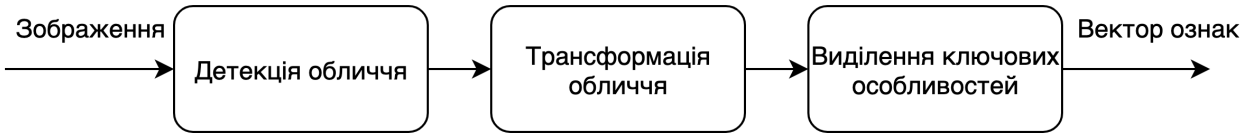


Рисунок 2.2. Основні етапи процесу розпізнавання обличчя

Першим кроком є детекція особи з вхідного зображення. Суть етапу полягає в підтвердженні чи має вхідне зображення обличчя і локалізації цієї області. Щоб знайти обличчя у зображенні, воно робиться чорно-білим, оскільки кольорові дані не потрібні для того, щоб знайти обличчя. Потім оглядається кожен піксель вхідного зображення по одному. Для кожного окремого пікселя потрібно оглянути пікселі, які безпосередньо його оточують. Мета - з'ясувати, наскільки темний поточний піксель відносно пікселів, які безпосередньо його оточують. Потім потрібно визначити напрям, у якому зображення стає темнішим і замінити цей піксель градієнтом, який вказує на цей напрям. Всі пікселі замінюються такими градієнтами. Ці градієнти показують перехід від світлого до темного у всьому зображенню. Якщо б аналізувалися пікселі безпосередньо, дуже темні зображення та дуже світлі зображення однієї людини мали б абсолютно різні значення пікселів. Але якщо враховувати напрямок зміни яскравості, то і дуже темні зображення, і дуже яскраві зображення врешті-решт матимуть однаково точне представлення.

Оскільки збереження градієнта для кожного пікселя дає занадто багато деталей, зображення розбивається на квадрати 16x16 пікселів, щоб можна було просто побачити основний потік світлого / темного на більш високому рівні. На кожному квадраті підраховується, скільки градієнтів вказують у кожному

головному напрямку (скільки вгору, вправо, вліво і т.д.). Потім ці квадрати замінюються на найсильніші напрямки градієнтів. В кінці зображення замінюється на просте представлення, яке охоплює основну структуру обличчя.

Рисунок 2.3. Оригінальне зображення[53]

Рисунок 2.4. Заміна пікселів градієнтами[53]

Щоб знайти обличчя на цьому зображенні необхідно знайти частину зображення, яка найбільш схожа на шаблон, отриманий з натренованих фото з відомим розташуванням облич.

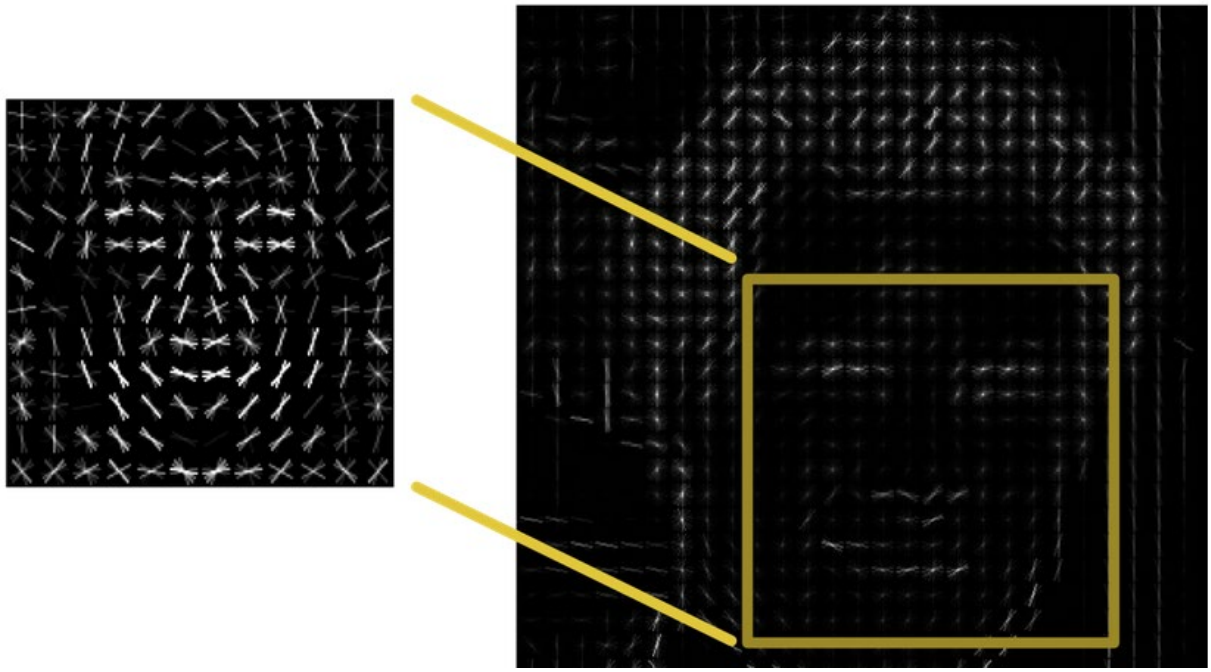


Рисунок 2.5. Пошук регіону обличчя за відомим шаблоном[53]

На другому етапі знайдене обличчя викривляється, щоб очі та губи були на одних і тих самих місцях. Для цього використовується алгоритм, який називається оцінкою орієнтиру обличчя. Є багато способів зробити це, але в цьому методі використано підхід, винайдений у 2014 році Вахідом Каземіта Жозефіною Салліван (Vahid Kazemi and Josephine Sullivan). Основна ідея полягає в тому, що створюється 68 точок (які називаються орієнтирами), які існують на кожному обличчі - верхній частині підборіддя, зовнішній край кожного ока, внутрішній край кожної брови тощо. Потім алгоритм буде натреновано так, щоб можна було знайти ці 68 точок на будь-якому обличчі.

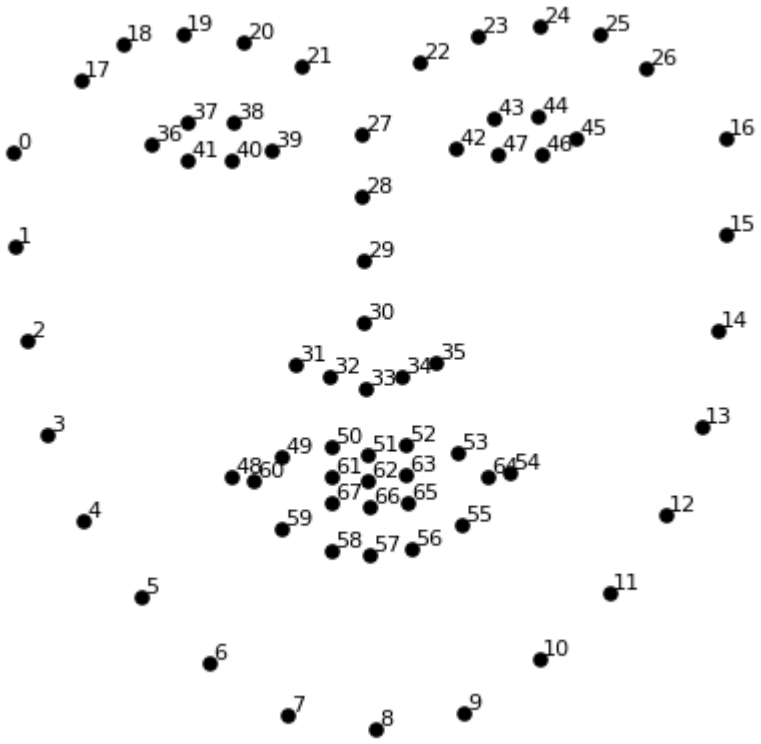


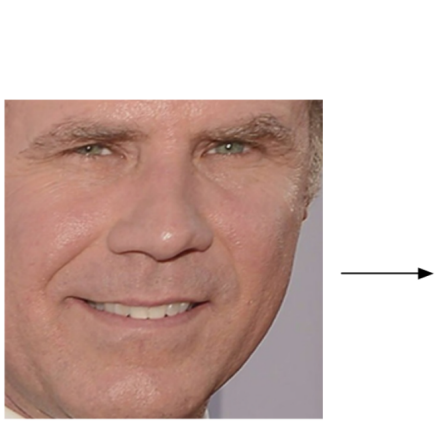
Рисунок 2.6. 68 орієнтирів, що знаходяться на кожному обличчі[53]

Знаючи де ці точки, можна виконати масштабування та інші афінні перетворення для їх найкращого розташування. Тому незважаючи на позицію обличчя, завжди можна розташувати обличчя так, щоб рот і очі були на однаковому місці на зображенні.

На третьому етапі відбувається виділення ключових особливостей за допомогою обраного для цього підходу. Можна, звичайно порівнювати точки, знайдені на попередньому кроці, але цей підхід дуже не ефективний, адже вимагає багато обчислень для порівняння в великій базі облич. Потрібен спосіб отримати кілька основних вимірювань з кожного обличчя, щоб можна було виміряти невідоме обличчя і знайти найближче вимірювання відомого обличчя.

Наприклад, можна виміряти розмір кожного вуха, відстань між очима, довжину носа тощо. Дослідники виявили, що найточніший підхід полягає в тому, щоб дозволити комп'ютеру самостійно з'ясувати ці виміри. Глибоке навчання справляється краще, ніж люди, коли вони з'ясовують, які частини обличчя важливі для вимірювання. Рішення полягає в тому, щоб навчити

згорткову нейронну мережу генерувати 128 вимірювань для кожного обличчя. Навчання такої мережі було описано у попередньому розділі. На виході отримуємо вектор ознак.



0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Рисунок 2.7. Генерація вектору ознак з обличчя[53]

Далі відбувається аналіз зображень кожної особи та ідентифікація особи на основі наявної бази даних. Для порівняння векторів використовується простий лінійний SVM класифікатор.

2.2 Архітектура системи аутентифікації за допомогою розпізнавання обличчя

Система аутентифікації за допомогою розпізнавання обличчя складається з 5 основних модулів:

1. модуль аутентифікації
2. модуль розпізнавання облич
3. бази даних
4. WEB-API
5. інтерфейс адміністратора

Рисунок 2.8 ілюструє архітектуру системи аутентифікації за допомогою розпізнавання облич та можливу схему інтеграції з іншими системами.

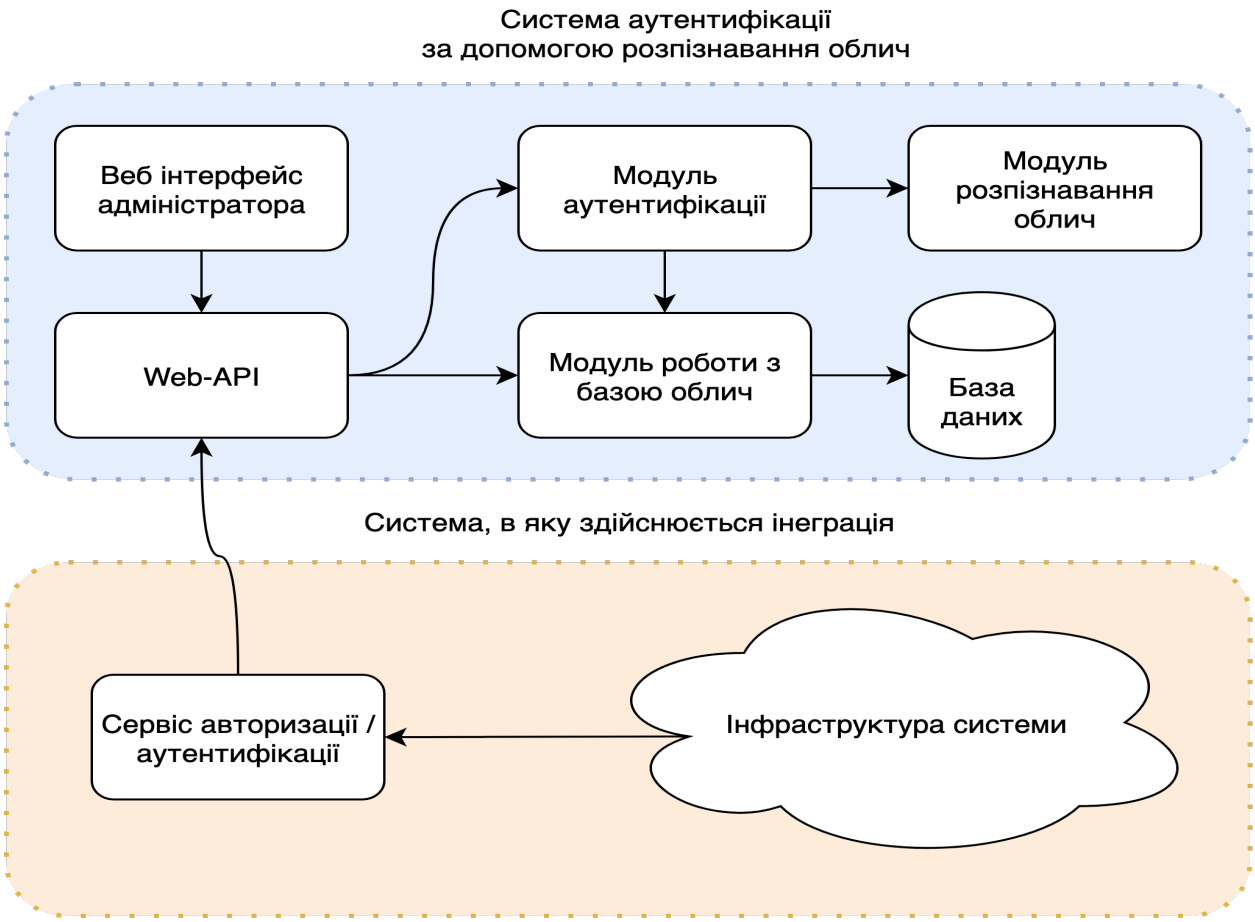


Рисунок 2.8. Архітектура системи аутентифікації за допомогою розпізнавання обличчя

Модуль розпізнавання облич отримує на вхід зображення, виконує його обробку відповідно до вище описаного алгоритму та повертає вектор ознак фіксованого розміру, що унікально ідентифікує особу.

Модуль роботи з базою облич реалізує CRUD сервіс. Він виконує пошук облич, їх реєстрацію, оновлення та видалення.

Модуль аутентифікації використовує два попередні модулі. На вхід отримується ідентифікатор особи, яку потрібно аутентифікувати та зображення її обличчя. Спочатку виконується відбувається виділення ключових особливостей з вхідного зображення. Потім отриманий вектор ознак порівнюється зі збереженим вектором ознак для заданого ідентифікатора користувача.

Web-API поєднує функціонал модуля аутентифікації та модуля роботи з базою даних у вигляді API.

Веб інтерфейс адміністратора використовує API для візуального відображення та керування зареєстрованими обличчями - створення, оновлення, видалення.

Інтеграція системи в існуючі рішення можлива за допомогою використання API системи, яке містить функціонал, аналогічний до того, що використовується в традиційних системах аутентифікації типу “логін-пароль”.

Всі користувачі в системі можуть бути двох видів: адміністратори та звичайні користувачі. Адміністратори існують лише для керування звичайними користувачами, створенням груп і пулів та додавання користувачів до цих груп і пулів. Пули існують для розмежування множини користувачів між собою. Можна, наприклад, створити окремі пули користувачів для різних сервісів системи, в яких будуть різні користувачі. Це корисно для розмежування доступу і для незалежної системи реєстрації в різних сервісах. Звичайні користувачі можуть входити до однієї або більше груп. Групи налаштовуються окремо для кожного пулу. Кількість та назви груп налаштовуються адміністратором та не мають ніяких обмежень. Наприклад, їх можна використовувати для надання різного рівня доступу до ресурсів при авторизації: читання та редагування - групи reader та editor.

На наступному рисунку зображено процес обробки вхідного зображення системою.

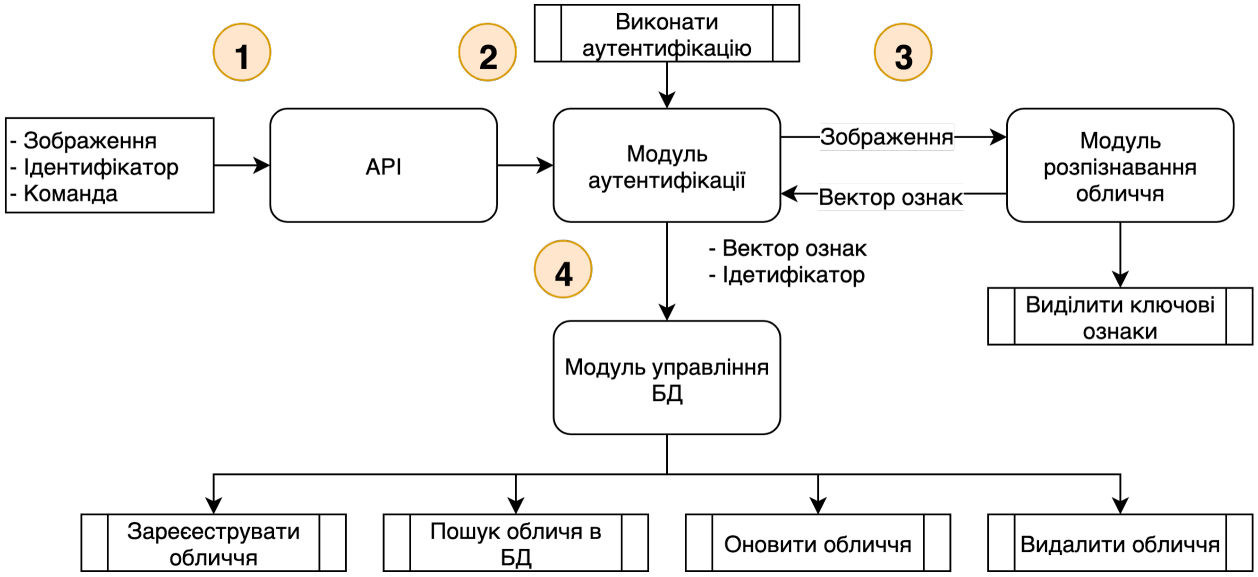


Рисунок 2.9. Зв'язки між модулями системи розпізнавання облич

1. На Web-API системи приходить зображення, ідентифікатор користувача та команда, яка вказує, що потрібно зробити з даним користувачем.
2. З Web-API ці дані передаються на модуль аутентифікації.
3. Модуль аутентифікації використовує модуль розпізнавання облич для перетворення вхідного зображення в вектор ознак.
4. Отриманий вектор ознак та ідентифікатор користувача передаються на модуль управління БД, де в залежності від вхідної команди користувач реєструється, проходить перевірку, оновлюється або видаляється.

2.3 Опис процесу реєстрації нового користувача

На наступній діаграмі описано процес реєстрації нового користувача.

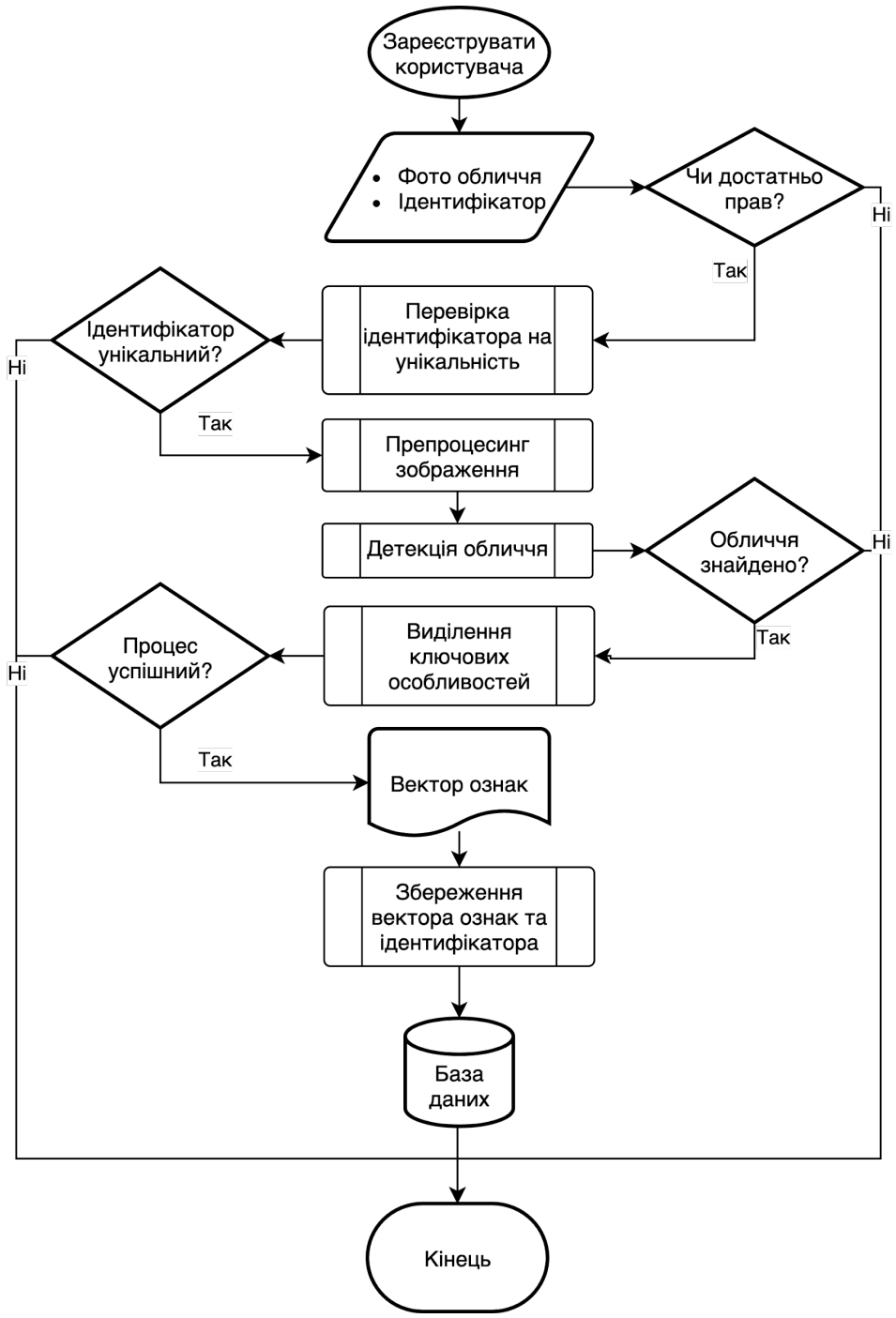


Рисунок 2.10. Процес реєстрації нового користувача

На модуль API передається ідентифікатор майбутнього користувача (електронна адреса, логін і т.д.) та фото його обличчя. В залежності від налаштувань системи, у користувачів є можливість зареєструватися самостійно, або це повинен зробити користувач з більш високими привілегіями (так званий адміністратор), який додається в систему при її встановленні. Тому наступним кроком є перевірка доступності реєстрації. Якщо реєстрацію проводить

адміністратор або анонімний користувач і дозволена самостійна реєстрація, то така реєстрація дозволяється і весь процес переходить на наступний крок. В іншому випадку генерується помилка і процес припиняється.

Наступним кроком перевіряється унікальність ідентифікатора. Зазвичай роль ідентифікатора грає електронна пошта або логін. Якщо такий ідентифікатор вже існує - генерується помилка.

Далі вхідне зображення обробляється відповідно до процесу, описаного у попередньому розділі (конвертація, зменшення розмірності і тд).

На оптимізованому зображенні відбувається пошук обличчя. Якщо облич не знайдено, або знайдено більше, ніж 1 - генерується помилка.

З отриманого обличчя генерується вектор ознак. Якщо цей процес пройшов успішно, ідентифікатор користувача та відповідний вектор ознак вносяться до бази даних і процес реєстрації нового користувача успішно закінчується.

2.4 Опис процесу аутентифікації користувача

На наступній діаграмі описано процес проходження аутентифікації користувачем.

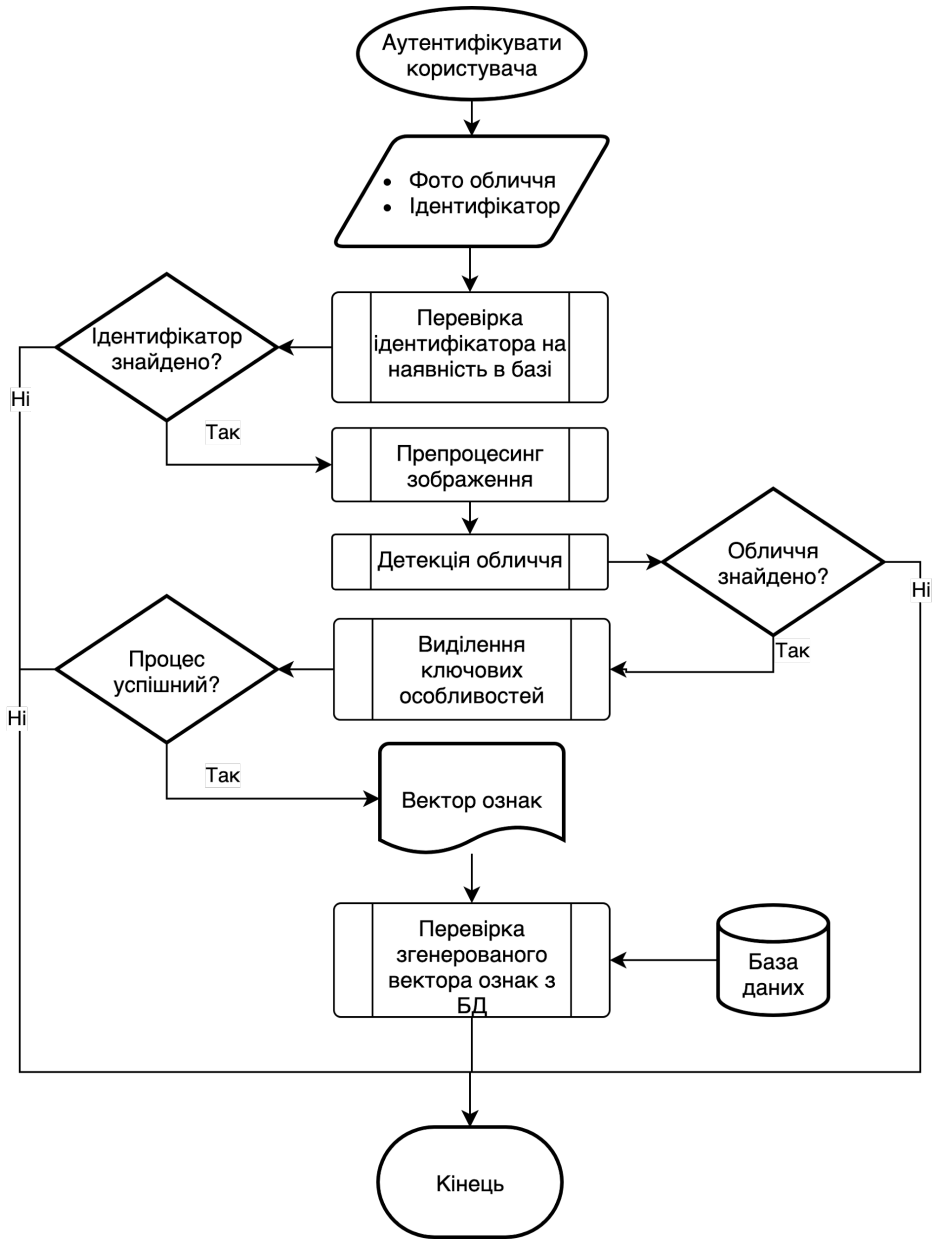


Рисунок 2.11. Процес проходження аутентифікації

Процес аутентифікації схожий з реєстрацією, але тепер на вхід в систему приходить існуючий ідентифікатор та фото обличчя. Після виділення вектору ознак з фото, він порівнюється з існуючим в базі даних за допомогою Евклідової відстані та встановленого значення порогу.

Процес оновлення зображення обличчя та видалення користувача використовує комбінацію попередніх кроків.

Висновки до розділу

Процес розпізнавання обличчя включає в себе серію послідовних кроків, які значно збільшують швидкість та якість роботи нейронної мережі:

- оптимізація вхідного зображення пришвидшує його обробку нейронною мережею
- заміна пікселів градієнтами робить алгоритм стійким до змін освітлення
- афінні трансформації над точками-орієнтирами роблять алгоритм стійким до змін положення, поворотів та нахилу обличчя.
- генерація одновимірного вектору ознак з точок-орієнтирів зменшує складність порівняння таких векторів між собою, порівняно з точками.

Архітектура системи є оптимальною для легкої інтеграції в існуючі системи. Також розділення на модулі і використання мікросервісної архітектури додає гнучкості при розгортанні системі на сервері та надає можливість незалежного використання кожного окремого модуля.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ АУТЕНТИФІКАЦІЇ ТА КОНТРОЛЯ ДОСТУПУ ЗА ДОПОМОГОЮ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ

3.1 Архітектура обраної мережі

У грудні 2015-го, приблизно в той же час, як була представлена архітектура Inception v3, відбулася революція - опублікували ResNet. У ній закладені прості ідеї: подаємо вихідні дані двох успішних згорткових шарів і обходимо вхідні дані для наступного шару (рисунок 1.11).

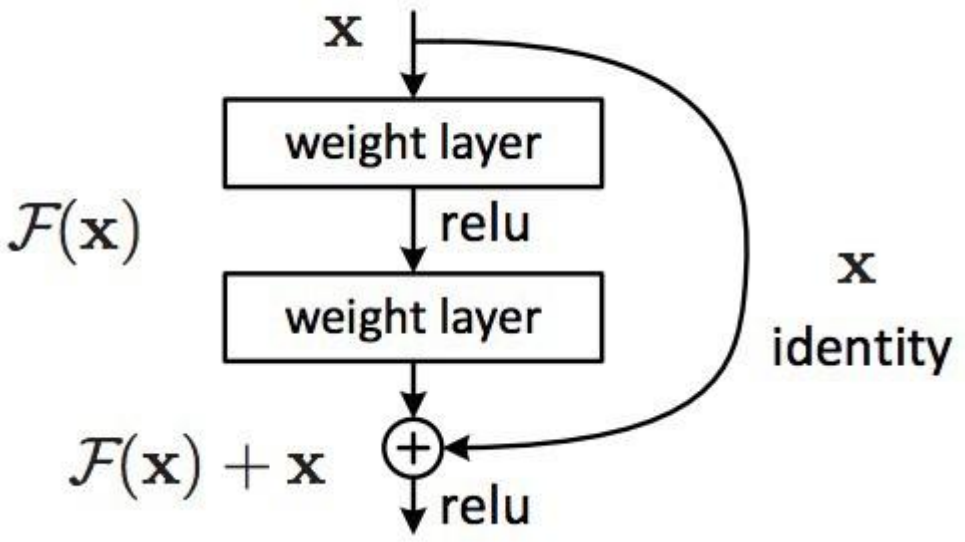


Рисунок 1.11. – Фундаментальний блок мережевої архітектури ResNet[54]

Обхід одного шару не дає особливої вигоди, а обхід двох - ключова знахідка. Це можна розглядати як маленький класифікатор, як мережа-в-мережі.

У багатошаровій ResNet застосували bottleneck-блок (блок, що містить 3 операції), який зменшує кількість властивостей в кожному шарі, спочатку використовуючи згортку 1x1 з меншим виходом (зазвичай чверть від входу), потім йде шар 3x3, а потім знову згортка 1x1 в більшу кількість властивостей. Це дозволяє економити обчислювальні ресурси, зберігаючи множин комбінацій властивостей. В якості фінального класифікатора в ResNet використовується pooling-шар з softmax.

Кожен шар ResNet складається з декількох блоків. Це тому, що коли ResNets заглиблюється, це робиться, збільшуючи кількість операцій всередині блоку, але кількість загальних шарів залишається однаковою - 4.

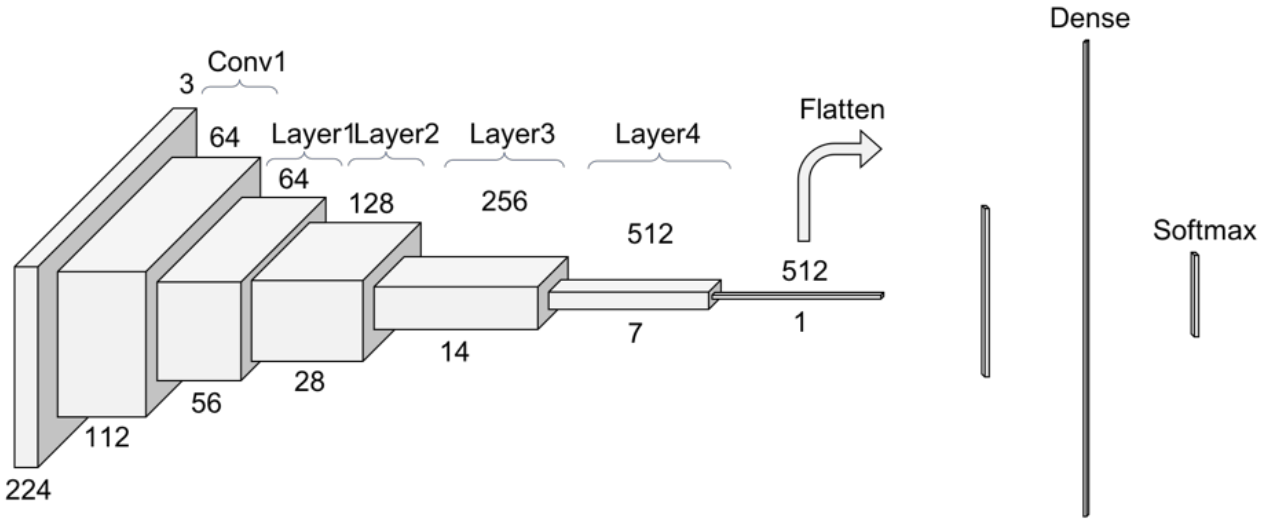


Рисунок 1.12. Шари мережі ResNet з 34 шарами[54]

Розрізняють блоки, що включають 2 операції - основний блок - та блоки, що включають 3 операції - ботлнек - блок. Зазвичай кожна з цих операцій називається шаром, але ми використовуємо термін шар для групи блоків.

3.2 Обґрунтування вибору мови програмування

При виборі мови програмування основними критеріями були:

- швидкість розробки
- висока продуктивність роботи додатку
- підтримка необхідних бібліотек
- кросплатформність, оскільки система буде встановлюватися на серверах клієнта, які можуть мати різні операційні системи

Для розробки програмних систем такого рівня існує три мови, які відповідають вимозі універсальності: Java[37], C#[38] та Python[39].

Java - мова програмування, спочатку розроблена Sun Microsystems і випущена в 1995, отримала значну частину свого синтаксису від C та C ++, але має більш просту об’єктну модель та менше можливостей низького рівня.

Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від архітектури комп'ютера.

C# - мова програмування загального призначення, була розроблена близько 2000 року Microsoft в рамках своєї екосистеми .NET, а згодом затверджена як міжнародний стандарт.

Python - інтерпретована мова програмування високого рівня, загального призначення. Використання пробілу робить код більш лаконічним та чітким, а архітектурні рішення і стандартна бібліотека допомагають програмістам писати писати код будь-якої складності, який легко читається без значних зусиль та затрат в часі.

Основним недоліком мови C# перед Java та Python є її пропрієтарна ліцензія та можливість використання всіх її можливостей в повному обсязі лише на ОС Windows. Оскільки розроблювана система є серверною та повинна витримувати досить високі навантаження бажаною є повна підтримка UNIX - подібних операційних систем, зокрема GNU/Linux. Тому реальним є вибір між Java та Python.

Швидкість розробки програмного забезпечення мовою Python є на порядок вищою у порівнянні із Java завдяки його динамічній типізації. Крім того Python є інтерпретованою мовою, що полегшує внесення змін та налаштування програмної системи на етапі впровадження. Важливим аргументом в сторону Python також є велика кількість підтримуваних бібліотек машинного навчання, зокрема різноманітні реалізації нейронних мереж для різних сфер застосування. Java значно поступається Python в цьому напрямку. Тому основною мовою програмування обрано мову Python.

Мова Python має різноманітні застосування в компаніях, що займаються розробкою програмного забезпечення, таких як ігри, веб-фреймворки і додатки, розробка мов, створення прототипів, додатки для графічного дизайну і т.д. Це забезпечує більш широке поширення Python в порівнянні з іншими мовами програмування, що використовуються в галузі. Деякі з його переваг:

- Великий набір вбудованих бібліотек
- Висока інтегрованість
- Висока швидкість розробки
- Легкий в читанні та написанні код

До недоліків можна віднести:

- Складність переходу на інші мови
- Слабка підтримка мобільних платформ
- Відносна повільність через інтерпретованість
- Помилки виявляються в рантаймі через динамічну типізацію

Всі вказані недоліки не є критичними і не матимуть негативного впливу на розроблену систему.

Для реалізації клієнтської частини веб - інтерфейсу обрано мову програмування JavaScript, мову розмітки HTML та мову задання стилів CSS по причині відсутності у них аналогів.

Для розробки серверної частини використано фреймворк Flask. Flask - це легкий WSGI фреймворк для розробки веб-додатків, призначений для швидкого і легкого початку роботи з можливістю масштабування до складних додатків. Він починався як проста оболонка навколо Werkzeug і Jinja і став однією з найпопулярніших платформ веб-додатків Python. Він класифікується як мікрофреймворк, оскільки не потребує конкретних інструментів чи бібліотек. У нього немає шару абстракції для бази даних, перевірки форм або будь-яких інших компонентів, які вже є реалізовані в існуючих сторонніх бібліотеках. Однак Flask підтримує розширення, які можуть додавати новий функціонал так, ніби він були реалізований в самому Flask. Розширення існують для об'єктно-реляційних мапперів (ORM), перевірки форм, обробки завантажень, різних технологій відкритої аутентифікації та іншого.

Для клієнтської частини використовувалися фреймворки jQuery та Bootstrap.

jQuery[40] - це легка бібліотека JavaScript, мета якої - полегшити використання JavaScript на веб-сайті. jQuery також спрощує багато складних речей з JavaScript, як AJAX виклики та маніпуляції з DOM. Станом на травень 2019 року jQuery використовується на 73% з 10 мільйонів найпопулярніших веб-сайтів. [41]

Bootstrap [42] - це вільний і відкритий вихідний код CSS, спрямований на чуйну мобільну розробку інтернету. Він містить шаблони дизайну на основі CSS та (необов'язково) JavaScript для типографії, форм, кнопок, навігації та інших компонентів інтерфейсу.

Bootstrap використовує сучасні напрацювання в області CSS і HTML, тому необхідно бути уважним при підтримці старих браузерів.

3.3 Обґрунтування вибору СУБД

Бази даних роблять структуроване сховище надійним і швидким. Бази даних - це концепція з багатьма реалізаціями, включаючи PostgreSQL[43], MySQL[44] і SQLite[45]. Нереляційні бази даних, звані сховищами даних NoSQL, також існують.

Абстракція сховища бази даних, яка найбільш часто використовується в веб-розробці Python - це набори реляційних таблиць.

Реляційні бази даних зберігають дані в серії таблиць. Взаємозв'язки між таблицями задаються як зовнішні ключі. Зовнішній ключ - це унікальне посилання від одного рядка у реляційній таблиці до іншого рядка в таблиці, який може бути тією ж таблицею, але найчастіше це інша таблиця.

Реалізації сховищ баз даних різняться за складністю. SQLite, база даних, що входить до Python, створює єдиний файл для всіх даних у базі даних. Інші бази даних, такі як PostgreSQL, MySQL, Oracle і Microsoft SQL Server, мають більш складні схеми збереження, пропонуючи додаткові розширені функції,

корисні для зберігання даних веб-додатків. Ці розширені функції включають, але не обмежуються:

1. реплікація даних між основною базою даних та одним або декількома дочірніми екземплярами, які знаходяться тільки режимі читання
2. розширені типи стовпців, які можуть ефективно зберігати напівструктуровані дані, такі як JavaScript Object Notation (JSON)
3. шардинг, який надає горизонтальне масштабування декількох баз даних
4. моніторинг, статистика та інша корисна інформація для схем та таблиць бази даних

Зазвичай веб-додатки починають з одного екземпляра бази даних, наприклад, PostgreSQL з простою схемою. З часом схема бази даних перетворюється на більш складну структуру з використанням міграцій схем, а додаткові функції, такі як реплікація, шардинг та моніторинг, стають все більш корисними, оскільки використання бази даних збільшується з ростом потреб користувачів додатку.

PostgreSQL і MySQL - це дві найпоширеніші бази даних з відкритим кодом для зберігання даних веб-додатків.

SQLite - це база даних, яка зберігається в одному файлі на диску. SQLite вбудований у Python, але побудований лише для доступу в єдиному потоці. Тому дуже рекомендується не запускати готовий веб-додаток із SQLite.

PostgreSQL - рекомендована реляційна база даних для роботи з веб-додатками в Python. Набір функцій PostgreSQL, активна розробка та стабільність сприяють її використанню в якості основи для мільйонів прикладних програм сьогодні.

MySQL - ще одна реалізація бази даних з відкритим кодом для додатків Python. MySQL має дещо простішу “криву початкового входження”, ніж PostgreSQL, але не настільки багата на можливості.

NoSQL - термін, що позначає ряд підходів, спрямованих на реалізацію сховищ баз даних, що мають суттєві відмінності від моделей, що використовуються в традиційних реляційних СКБД з доступом до даних засобами мови SQL.

До NoSQL відносяться такі СУБД як Redis, Riak, Memcached, CouchDB, MongoDB та інші.

В системі аутентифікації за допомогою розпізнавання всі дані строго структуровані та мають реляції, тому нереляційні рішення не розглядалися. Для реалізації системи ідеально підходять, як PostgreSQL, так і MySQL, тому вибір було зупинено на PostgreSQL по причині більшого досвіду роботи з даною СУБД. В розробленій системі робота з базою даних реалізована через шар ORM, що реалізується бібліотекою SQLAlchemy[46]. ORM надають високорівневу абстракцію на реляційну базу даних, яка дозволяє розробнику писати Python код замість SQL для створення, зчитування, оновлення та видалення даних і схем у своїй базі даних. Розробники можуть використовувати мову програмування, з якою їм зручно, для роботи з базою даних, замість написання SQL-операторів або збережених процедур.

Можливість написання коду Python замість SQL може пришвидшити розробку веб-додатків, особливо на початку проекту. Потенційне збільшення швидкості розробки відбувається через те, що не потрібно переходити з коду Python до написання декларативної парадигми SQL-операторів. Хоча деякі розробники програмного забезпечення не проти перемикання між мовами, зазвичай простіше зробити прототип або запустити веб-додаток за допомогою однієї мови програмування.

ORM також теоретично дозволяє перемикати додаток між різними реляційними базами даних. Наприклад, розробник може використовувати SQLite

для локальної розробки та MySQL на ‘бойовому’ сервері. Програма може бути переключена з MySQL на PostgreSQL з мінімальними модифікаціями коду.

Використання шару ORM є досить важливим рішенням, оскільки розроблювана система має інтегруватися з існуючими системами, які можуть використовувати абсолютно різні СУБД і можливість перемикатися між ними без модифікації коду є необхідністю. Звичайно можна налаштувати окрему базу даних виключно під систему аутентифікації, але більшості випадків це пуста трата ресурсів.

SQLAlchemy - це не просто ORM, бібліотека також надає SQLAlchemy Core - функціонал для виконання роботи з базою даних, який абстрагує від написання читсого SQL і від відмінностей кінцевої СУБД, будь то PostgreSQL, SQLite чи інша база даних. В певній мірі ORM є бонусом до Core, який автоматизує загально необхідні операції створення, читання, оновлення та видалення.

SQLAlchemy можна використовувати з або без функцій ORM. Будь-який проект може вибрати просто використання SQLAlchemy Core або як Core, так і ORM. В системі аутентифікації використовується комбінація SQLAlchemy ORM та SQLAlchemy Core.

Для розробки бази даних, по перше, необхідно побудувати концептуальну модель системи . Для цього виділяємо з предметної області, сутності з їх атрибутами та зв’язки між ними. Концептуальна модель подана у вигляді діаграми Чена.

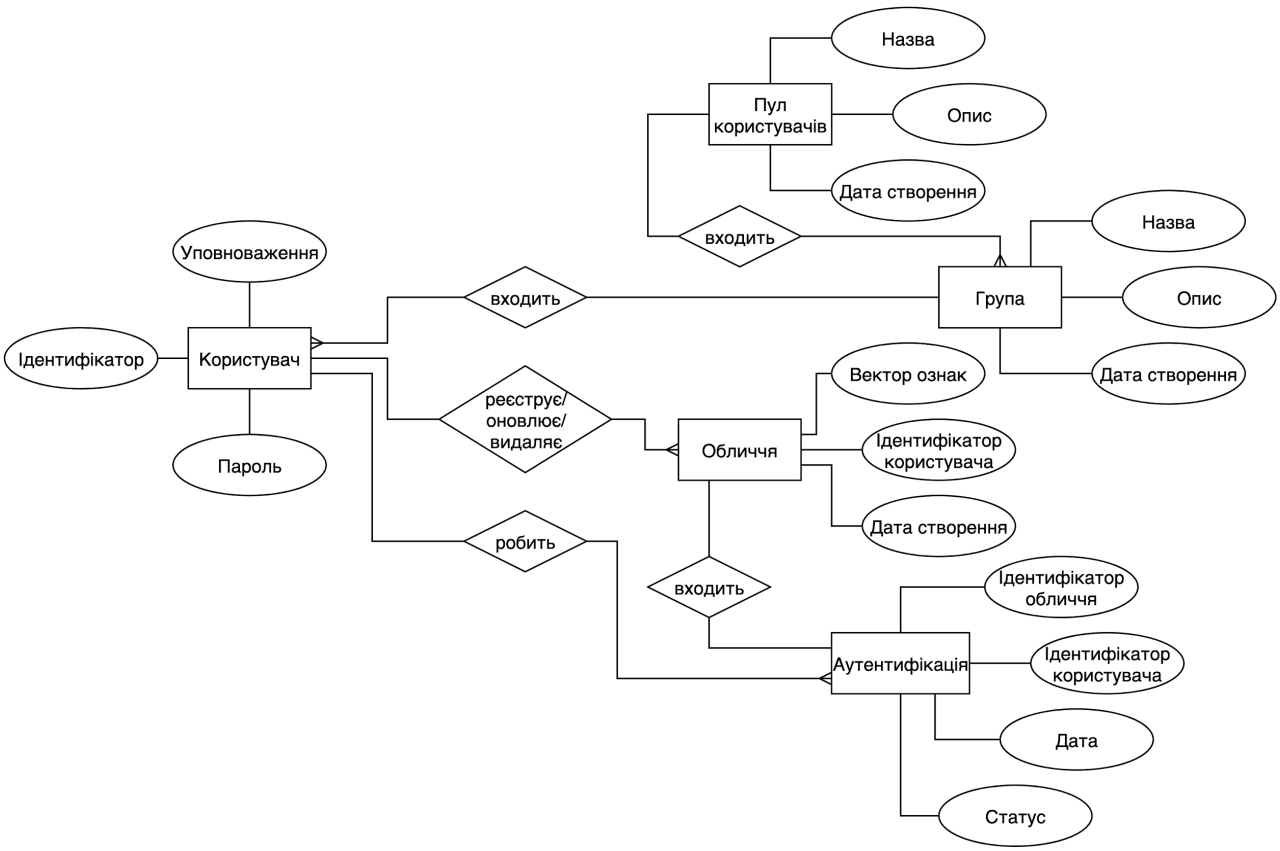


Рисунок 3.1. Концептуальна модель системи аутентифікації за допомогою обличчя

Розглянемо сутності, приведені на діаграмі:

1. Сутність “Користувач” характеризується наступними атрибутами:

- Уповноваження;
- Ідентифікатор;
- Пароль - для користувачів, що входять в групу ‘Адміністратор’;

2. Сутність “Пул користувачів” характеризується наступними атрибутами:

- Назва;
- Опис;
- Дата створення;

3. Сутність “Група” характеризується наступними атрибутами:

- Назва;
- Опис;
- Дата створення;

4. Сутність “Аутентифікація” характеризується наступними атрибутами:

- Ідентифікатор користувача;
- Ідентифікатор обличчя;
- Дата - дата проходження аутентифікації;
- Статус - результат проходження аутентифікації (успішно / невдало);

Таблиця 3.1 - Зв’язки між сутностями

Назва сутності	Назва зв’язку	Назва сутності	Тип зв’язку
користувач	входить	група	M : 1
користувач	реєструє/ оновлює/видаляє	обличчя	1 : M
користувач	робить	аутентифікація	1 : M
група	входить	пул	M : 1
обличчя	входить	аутентифікація	1 : 1

3.4 Даталогічна модель бази даних

На концептуальній моделі присутні лише зв’язки один до багатьох, які реалізовуються в даталогічній моделі у вигляді звичайного відношення Primary\Foreign key. Даталогічна модель наведена на рисунку 3.2.

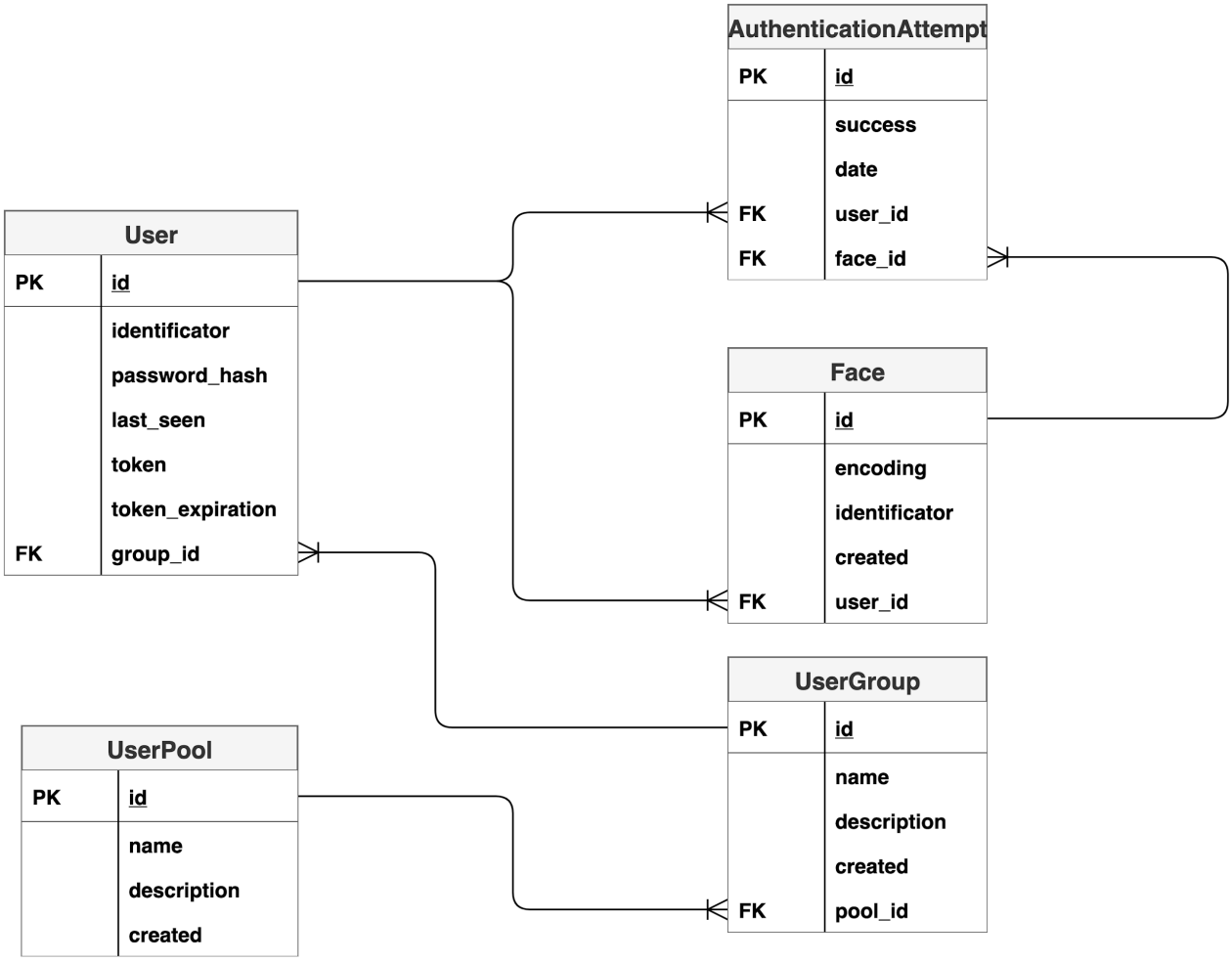


Рисунок 3.2. Даталогічна модель системи аутентифікації за допомогою обличчя

3.5 Фізична модель бази даних

Таблиця 3.2 - Структура таблиці User (Користувач)

Назва поля в таблиці	Назва	Тип поля	Ознака ключа
id	Спец. номер	Integer	Первинний
identificator	Унікальний ідентифікатор	Varchar(256)	
password_hash	Хеш пароля (Адміністратори)	Varchar(256)	
token	Токен авторизації (Адміністратори)	Varchar(256)	
token_expiration	Кінець дії токена (Адміністратори)	Datetime	
group_id	Група, до якої належить користувач	Integer	Вторинний

Таблиця 3.3 - Структура таблиці UserPool (Пул користувачів)

Назва поля в таблиці	Назва	Тип поля	Ознака ключа
id	Спец. номер	Integer	Первинний
name	Назва	Varchar(256)	
description	Опис	Varchar(256)	

created	Дата створення	Datetime	

Таблиця 3.4 - Структура таблиці UserGroup (Група)

Назва поля в таблиці	Назва	Тип поля	Ознака ключа
id	Спец. номер	Integer	Первинний
name	Назва	Varchar(256)	
description	Опис	Varchar(256)	
created	Дата створення	Datetime	
pool_id	Пул, до якої належить група	Integer	Вторинний

Таблиця 3.5 - Структура таблиці AuthenticationAttempt (Спроба аутентифікації)

Назва поля в таблиці	Назва	Тип поля	Ознака ключа
id	Спец. номер	Integer	Первинний
status	Статус аутентифікації	Boolean	
date	Дата аутентифікації	Datetime	
face_id	Обличчя користувача	Integer	Вторинний

user_id	Користувач	Integer	Вторинний

Таблиця 3.6 - Структура таблиці Face (Спроба аутентифікації)

Назва поля в таблиці	Назва	Тип поля	Ознака ключа
id	Спец. номер	Integer	Первинний
encoding	Вектор ознак	JSON	
name	Ідентифікатор	Varchar(256)	
created	Дата створення	Datetime	
user_id	Користувач	Integer	Вторинний

3.6 Опис методів REST API

Система аутентифікації за допомогою розпізнавання обличчя побудована у вигляді веб-додатку, що реалізує клієнт-серверну архітектуру.

Особливістю архітектури клієнт-сервер є передача обчислювального навантаження на сервер і максимальне звільнення клієнта від обчислювальної роботи, а також значне підвищення захисту даних як від зловмисних, так і просто некоректних змін.

Переваги архітектури "клієнт-сервер":

1. мережевий трафік зменшується через те, що сервер пов'язує з клієнтом тільки ті дані, які він запитує;

2. більшість обчислювальних процесів відбувається на сервері, тим самим знижуючи обчислювальні вимоги клієнтського комп'ютера;

3. різні рівні ізоляції транзакцій дозволяють визначати поведінку сервера в ситуаціях одночасної зміни даних.

4. спрощує збільшення обчислювальної потужності в контексті розробки програмного забезпечення і зростання оброблюваних даних;

5. сервер реалізує управління транзакціями і запобігає спробам одночасно змінити одні й ті ж дані;

В якості схеми взаємодії з сервісом використовується REST, дані передаються у форматі JSON. REST, або REpresentational State Transfer, являє собою архітектурний стиль для забезпечення стандартів між комп'ютерними системами в Інтернеті, який полегшує взаємодію систем один з одним. REST-сумісні системи, часто звані RESTful-системами, характеризуються тим, що вони не мають стану і поділяють інтереси клієнта і сервера.[47].

В мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (зазвичай «GET» або «POST»; такий запит називають «REST-запит»), а необхідні дані передаються в якості параметрів запиту[48] [49].

Для веб-служб, побудованих з урахуванням REST (тобто, що не порушують накладених обмежень), застосовують термін «RESTful».

На відміну від веб-сервісів (веб-служб) на основі SOAP, не існує «офіційного» стандарту для RESTful веб-API. Справа в тому, що REST є архітектурним стилем, в той час як SOAP є протоколом. Незважаючи на те, що REST не є стандартом сам по собі, більшість RESTful-реалізацій використовують такі стандарти, як HTTP, URL, JSON і XML.

В наступній таблиці наведено методи REST API, що реалізують функціонал системи аутентифікації за допомогою розпізнавання облич.

Таблиця 3.7 - Методи REST API

Запит	Опис	Відповідь

GET api/users	Отримати список зареєстрованих	200 OK
		{
		"_links": {
		"next":
		"/api/users?page=2&per_page
		=8",
		"prev": null,
		"self":
		"/api/users?page=1&per_page
		=8"
		},
		"_meta": {
		"page": 1,
		"per_page": 8,
		"total_items": 10,
		"total_pages": 2
		},
		"items": [
		{
		"created": "2020-03-
		29T21:23:48",
		"id": 3,
		"name": "Me"
		}
		]
		}

POST api/users
{
"identificator": "someid",
"image": "base64image"
}
	Зареєструвати нового користувача	201 CREATED

PUT api/users/<id>
{
"image": "base64image"
}
	Оновити фото користувача з заданим ідентифікатором	201 CREATED

GET api/users/<id>	Отримати дані користувача з заданим ідентифікатором	200 OK
		{
		"created": "2020-03-29T21:23:48",
		"id": 3,
		"name": "Me"
		}

DELETE api/users/<id>	Видалити користувача з заданим ідентифікатором	201 CREATED

POST api/users/auth
{
"identificator": "someid",
"image": "base64image"
}
	Провести аутентифікацію користувача	200 OK
		{
		“success”: true
		}

3.7 Інтеграція розробленого програмного забезпечення в існуючу інфраструктуру

Цільовими платформами для роботи даної системи є ОС сімейства Linux, Windows та macOS.

Можливо два шляхи розгортання системи на машині: встановивши всі необхідні залежності напряму на систему або застосувати docker-compose[50].

Розглянемо перший варіант розгортання системи шляхом прямого встановлення на гостьову операційну систему.

Для роботи системи необхідно встановити:

- інтерпретатор Python 3.7

- веб - сервер Nginx

- веб - сервер Gunicorn

- PostgreSQL Server або аналог, що підтримується в SqlAlchemy

- необхідні Python бібліотеки (з requirements.txt)

Nginx і Gunicorn- це веб сервери. Nginx потрібен для віддачі статичних даних і проксування запитів користувача до Gunicorn. А Gunicorn здійснює запити до веб - додатку та отримує від нього відповіді. Python спілкується з веб-сервером по протоколу WSGI. WSGI - це інтерфейс шлюзу веб-сервера. Це специфікація, яка описує, як веб-сервер спілкується з веб-додатками та як веб-додатки можуть бути пов'язані разом для обробки одного запиту.

Інтерпретатор Python є складовою частиною Debian GNU/Linux та macOS, а тому не потребує встановлення. Для встановлення Python на Windows необхідно слідувати інструкції з офіційного сайту Python. Все інше необхідне програмне забезпечення доступне із офіційних репозиторіїв ОС Linux та macOS

і встановлюється за допомогою стандартних системних утиліт apt або brew. Для Windows можна використати пакетний менеджер chocolatey.

Для керування залежностями в Python існує утиліта pip. Для встановлення всіх залежностей потрібно виконати pip install -r requirements.txt.

Після встановлення всіх залежной, можна запустити систему, виконавши наступні команди:

- flask db upgrade
- gunicorn 'app:create_app()' -c configs/gunicorn.py -b ${HOST}:${PORT}

Недоліком даного підходу є встановлення всіх залежностей на основну систему, що може конфліктувати з існуючим програмним забезпеченням, ускладнює видалення та оновлення компонентів.

Набагато простішим способом встановлення є використання docker-compose. Docker - це інструмент, призначений для спрощення створення, розгортання та запуску програм за допомогою контейнерів. Контейнери дозволяють розробнику пакувати додаток з усіма необхідними йому частинами, такими як бібліотеки та інші залежності, і розгортати його як один пакет. Тим самим, завдяки контейнеру, розробник може бути впевнений, що програма буде працювати на будь-якій іншій машині незалежно від налаштувань, які можуть відрізнятися від машини, що використовується для написання та тестування коду. Спочатку використовував можливості LXC, з 2015 року застосовував власну бібліотеку, що абстрагує віртуалізаційних можливості ядра Linux - libcontainer. З появою Open Container Initiative почався перехід від монолітної до модульної архітектури. Контейнери відокремлені один від одного та поєднують власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватися між собою через чітко визначені канали. Усі контейнери управляються одним ядром операційної системи і тому використовують менше ресурсів, ніж віртуальні машини.

Docker Compose використовується для одночасного управління декількома контейнерами, що входять в склад додатка. Цей інструмент надає ті ж можливості, що і Docker, але дає змогу працювати з більш складними додатками.

Перевагою docker-compose є:

- Можливість створення складних контейнерів, що містять декілька програм
- Ізоляція на рівні файлової системи
- Ізоляція ресурсів
- Використання легковагих контейнерів для ізоляції процесів від інших процесів і основної системи.
- Підтримка роботи на будь-якому комп'ютері на базі архітектури x86_64
- Всі компоненти вже встановлені і налаштовані всередині контейнерів

Після встановлення docker-compose, для запуску системи необхідно виконати наступну команду:

docker-compose up -d

Web інтерфейс буде доступний за адресою <IP машини>:5000

3.8 Аналіз результатів роботи програми

Приклад роботи REST API було наведено у попередньому розділі, тепер розглянемо WEB-інтерфейс адміністратора. В поточній реалізації WEB-інтерфейс підтримує додавання нових користувачів до бази даних, їх оновлення, видалення та аутентифікації особи по фото.

При вході на сайт користувач бачить головне вікно, яке виглядає наступним чином:

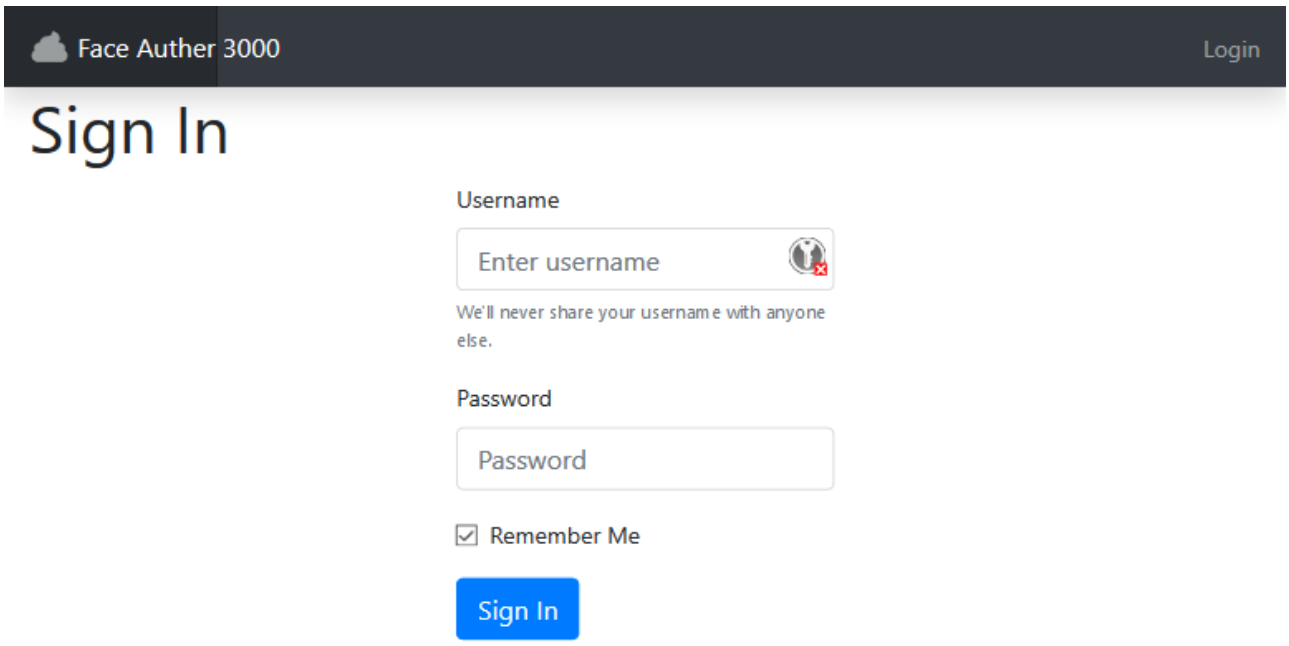


Рисунок 3.3. Головна сторінка веб-сайту

Після успішного проходження етапу автентифікації та авторизації користувачу відкривається доступ до решти розділів сайту, де можна додати користувачів (рисунок 3.4) і керувати всією базою користувачів через зручний інтерфейс (рисунок 3.5).

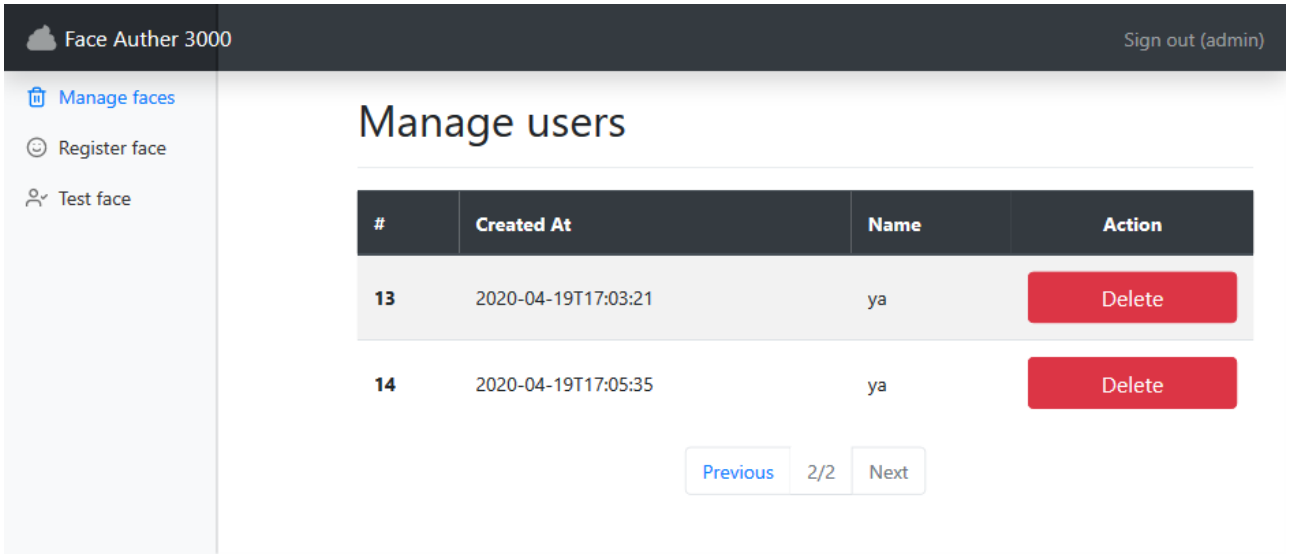


Рисунок 3.4. Сторінка керування базою користувачів

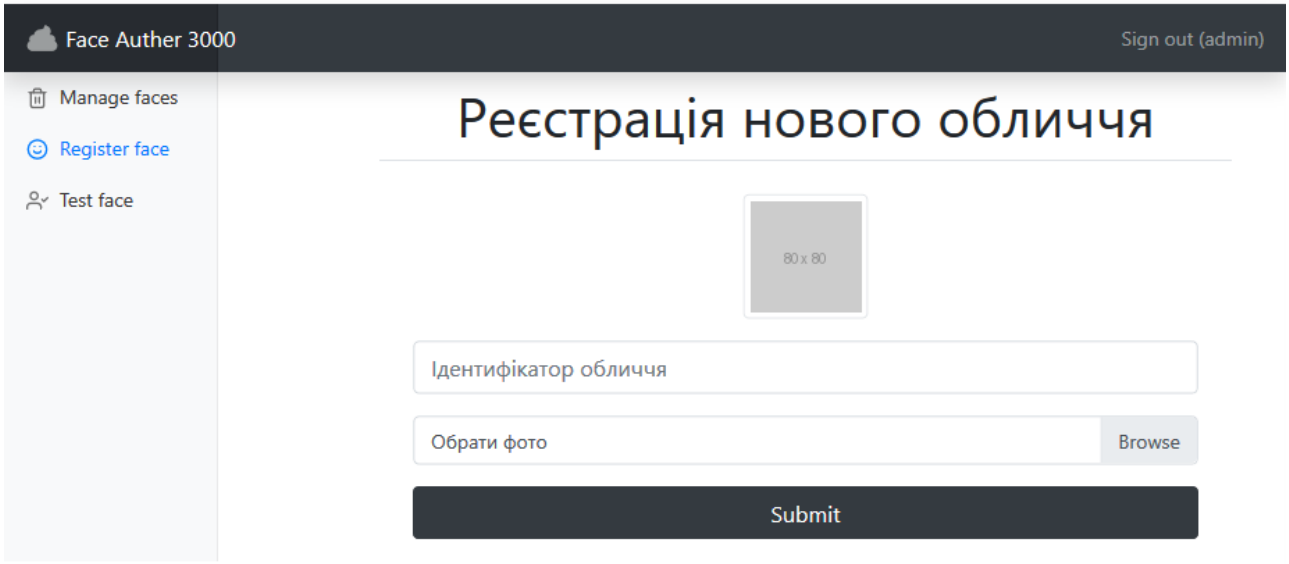


Рисунок 3.5. Сторінка додавання користувача

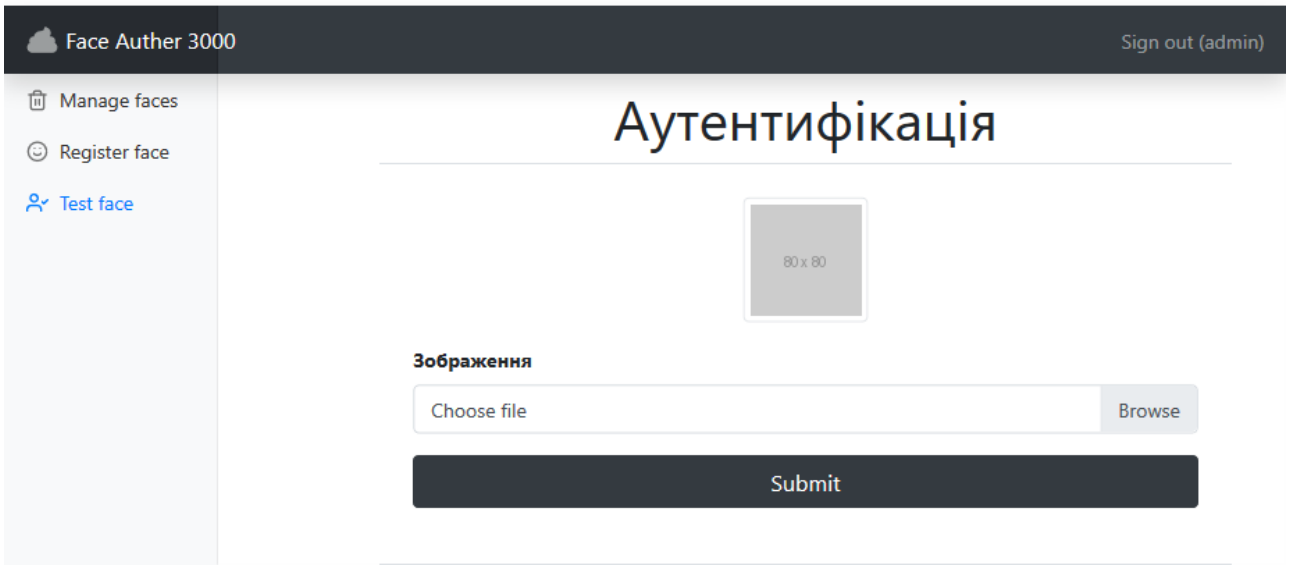


Рисунок 3.6. Сторінка аутентифікації особи по фото

Продемонструємо процес реєстрації на аутентифікації користувача. Для прикладу візьмемо наступне фото:

Рисунок 3.7. Тестове фото для реєстрації обличчя

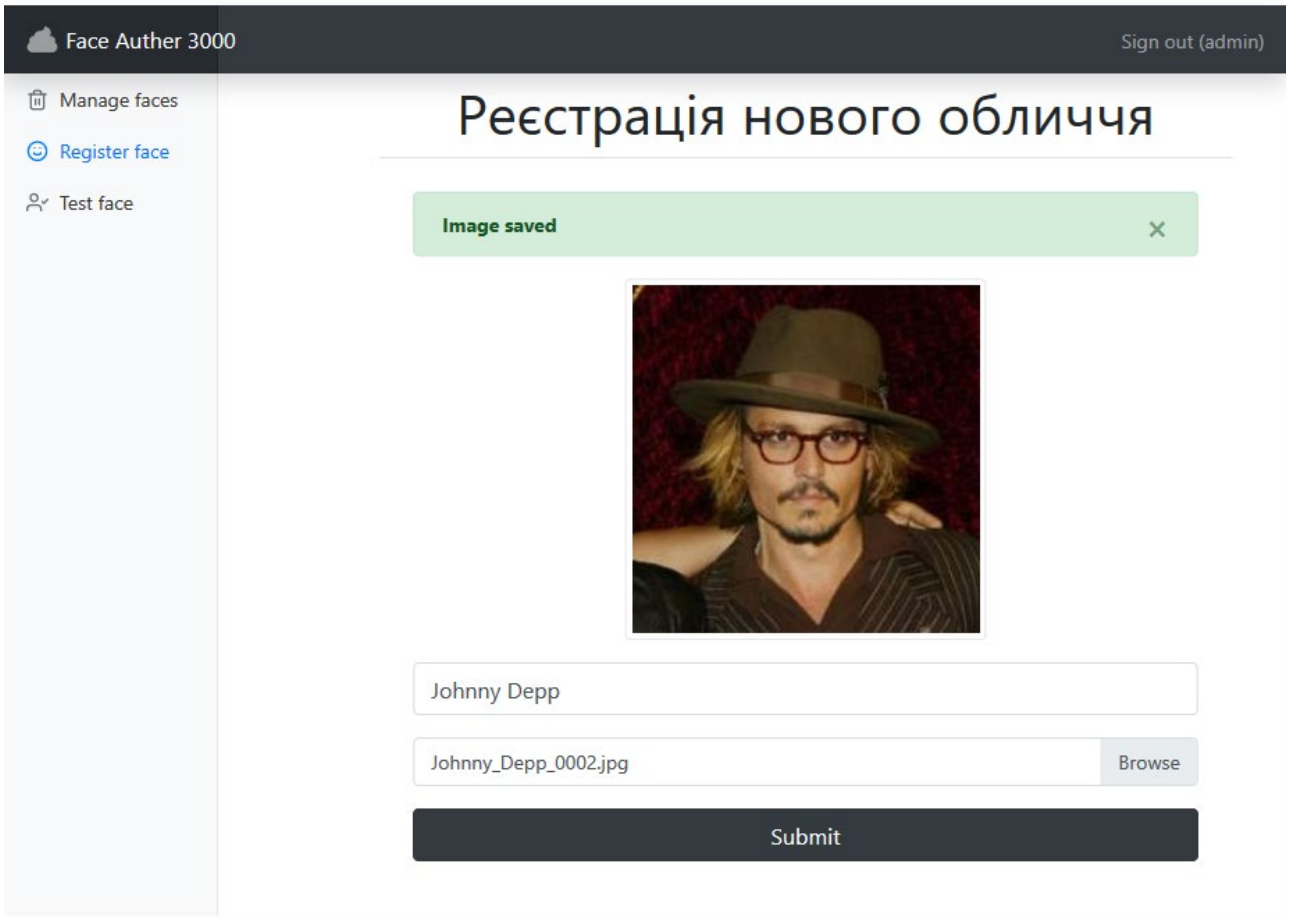


Рисунок 3.8. Реєстрація обличчя пройшла успішно

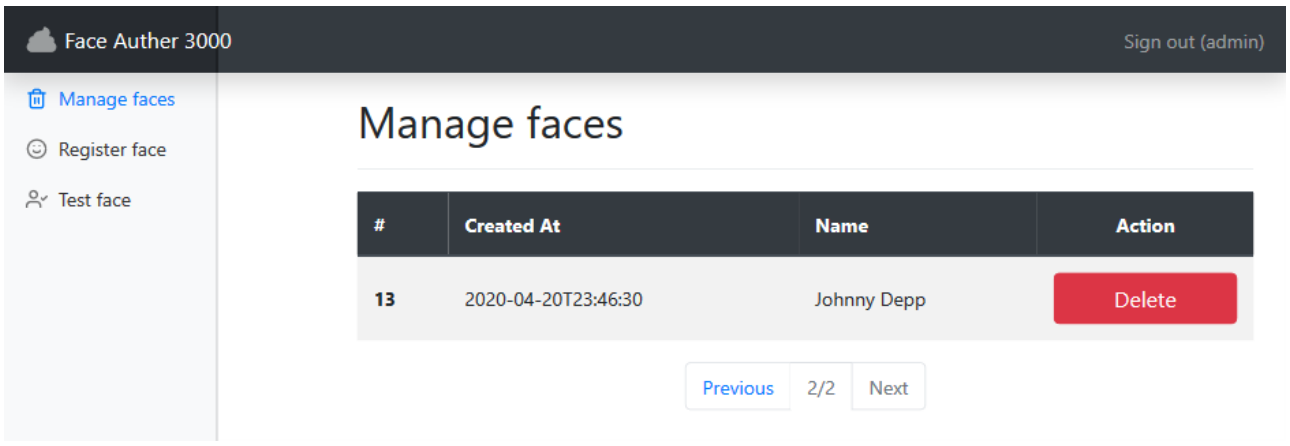


Рисунок 3.9. Обличчя додано до бази даних

Фото було успішно зареєстровано з унікальним ідентифікатором користувача. Тепер спробуємо провести аутентифікацію, використовуючи інше фото тієї ж особи.

Рисунок 3.10. Тестове фото обличчя для аутентифікації

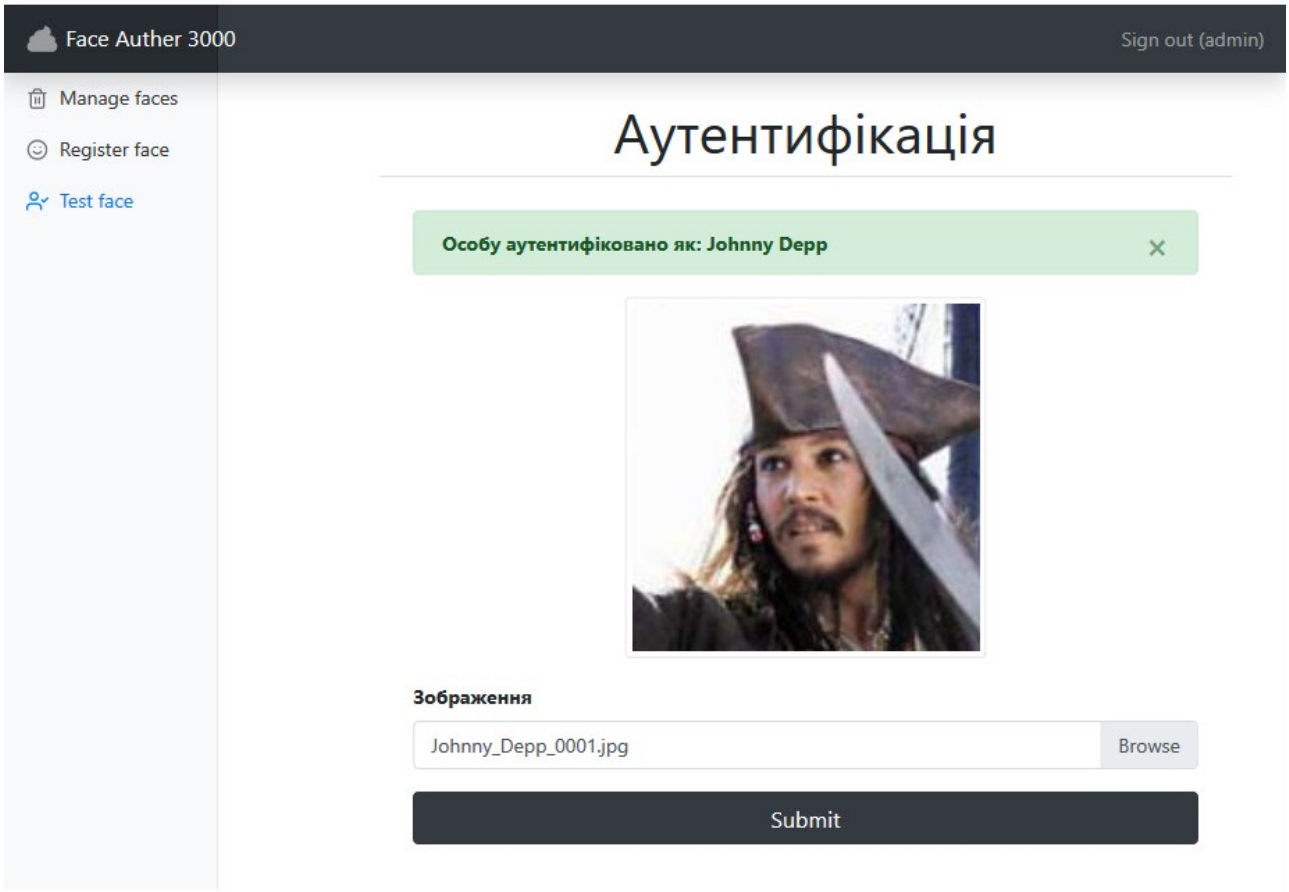


Рисунок 3.11. Особу успішно аутентифіковано

Тепер спробуємо пройти аутентифікацію, використовуючи фото невідомої особи:

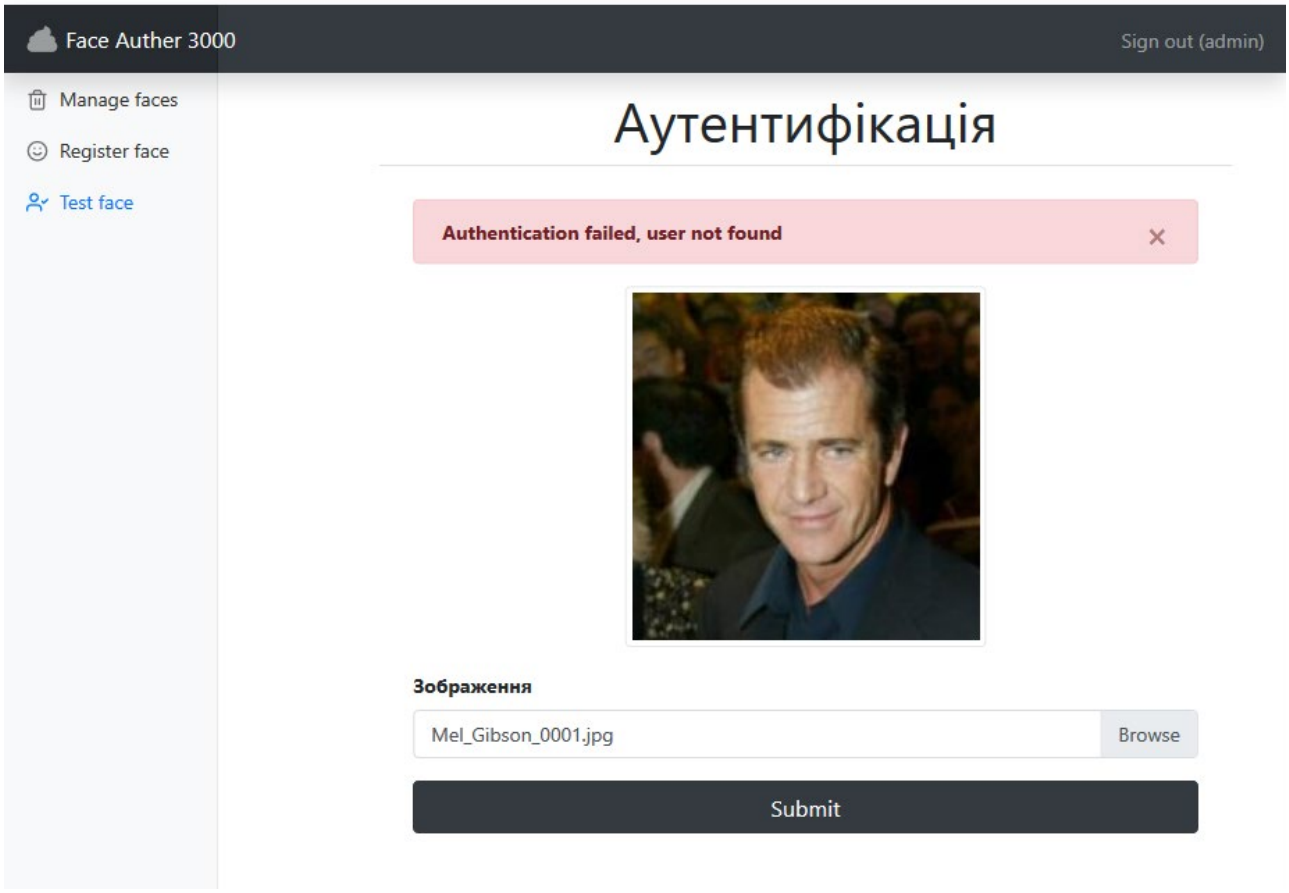


Рисунок 3.12. Аутентифікація невідомої особи пройшла невдало

Як видно з прикладів вище, система успішно аутентифікує користувачів, незважаючи на сторонні аксесуари (капелюх та окуляри).

Висновки до розділу

В даному розділі було описано обрану нейронну мережу, її архітектуру та реалізацію. Мовою програмування обрано Python 3.7.1, в якості СУБД використовувався PostgreSQL. Була описана схема бази даних, її фізична та даталогічна моделі. Були описані схеми REST API запитів та відповідей. Показані можливі способи інтеграції системи та приклади її роботи.

ВИСНОВКИ

На даному етапі в системі реалізовано ідентифікацію особи по зображенню обличчя, система успішно справляється зі змінами рис обличчя, наявністю сторонніх предметів (одяг, борода, капелюх, окуляри тощо), зміною положення та освітлення.

Розглянуто основні засоби розробки, доведено і обґрунтовано правильність вибору цих засобів для проектування. Зроблено аналіз існуючих аналогів. На основі проведеного аналізу встановлено, що оптимальною мовою програмування є Python, оскільки дозволяє швидко та ефективно розробляти додатки будь-якої складності.

Було проведено аналіз існуючих програмних продуктів, зроблено оцінку використаних та досліджених підходів, визначено основні переваги та недоліки цих продуктів, які були враховані при розробці власної системи.

В результаті виконання даної дипломної роботи було розроблено систему, яка мала повноцінний функціонал для аутентифікації користувачів за допомогою розпізнавання обличчя. Систему має зручний веб інтерфейс з розмежованим рівнем доступу для адміністраторів та REST API, що реалізує основний функціонал системи. Адміністратори мають змогу додавати нових користувачів, оновлювати існуючу базу користувачів та видаляти користувачів ви з системи. Користувачі мають змогу самостійної реєстрації, якщо це дозволено системою, керувати своїм обліковим записом (оновлювати / видаляти) та проходити аутентифікацію.

Готова система побудована у вигляді WEB-сервісу, використовуючи клієнт-серверну архітектуру. В якості схеми взаємодії з сервісом використовується REST, дані передаються у форматі JSON. Такий підхід сприяє легкій інтеграції в існую системні рішення як сторонній модуль, що не залежить від цільової операційної системи та мови програмування.

В роботі також розглянуто два можливі сценарії інтеграції системи: пряме встановлення на цільову систему, або використовуючи систему контейнеризації Docker та docker-compose.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Введение в оценку биометрических систем / П. Дж. Филлипс, Э. Мартин, С. Л. Уилсон, М. Пржибоски // Открытые Системы, №03, 2000. http://www. osp. ru/os/2000/03/ (дата об ращения: 17. 06. 2013)

2. Delac K., Grgic M. Face Recognition. I-Tech, 2007.

3. Kong S., Heo J., Abidi B. et al. // Computer Vision and Image Understanding. 2005. Vol. 97, № 1. P. 103-135.

4. Yang M.H., Kriegman D.J., Ahuja N. // IEEE Trans. on Pattern Analysis and Machine Intelligence. 2002. Vol. 24, № 1. P. 34-58.

5. Форсайт Д.А., Пойнс Ж. Компьютерное зрение. Современный подход. М., 2004.

6. Шапиро Л., Стокман Дж. Компьютерное зрение. М., 2006.

7. Openbiometrics - [Електронний ресурс] - Режим доступу: http://openbiometrics.org/

8. Center for Machine Perception @ CTU in Prague - [Електронний ресурс] - Режим доступу: https://github.com/uricamic/flandmark

9. OpenFaceTracker - [Електронний ресурс] - Режим доступу: https://openfacetracker-lib3.readthedocs.io/en/latest/

10. Open Biometrics Initiative - [Електронний ресурс] - Режим доступу: http://www.openbiometricsinitiative.org/

11. Bioenabletech - [Електронний ресурс] - Режим доступу: https://www.bioenabletech.com/bioenable-vface/iface

12. Bioenabletech - [Електронний ресурс] - Режим доступу: https://www.bioenabletech.com/bioenable-vface/vface

13. FacePlusPlus - [Електронний ресурс] - Режим доступу: https://www.faceplusplus.com/

14. DeepFace - [Електронний ресурс] - Режим доступу: https://deepface.ir/

15. Deep Learning For Face Recognition: A Critical Analysis [Електронний ресурс] - Режим доступу: https://arxiv.org/ftp/arxiv/papers/1907/1907.12739.pdf

16. Savvides, M. Face Verification using Correlation Filters [Text] / M. Savvides, B. V. K. V. Kumar, P. Khosla // CMU Electrical & Computer Engineering. [Електронний ресурс]. - Режим доступу: http://www.ece.cmu.edu/~kumar/Biometrics_AutoID.pdf

17. Li, S. Z. Handbook of Face Recognition [Text] / S. Z. Li, A. K. Jain. - London: Springer, 2011. - 699 p. doi:10.1007/978-0-85729-932-1 https://doi.org/10.1007/978-0-85729-932-1

18. Viola, P. Rapid object detection using a boosted cascade of simple features [Text] / P. Viola, M. Jones // Proceedingsofthe 2001 IEEE Computer Society Conference On Computer Vision and Pattern Recognition. CVPR 2001. - Kauai, Hawaii, USA, 2001. - Vol. 1. - P. 511-518. doi:10.1109/cvpr.2001.990517 https://doi.org/10.1109/cvpr.2001.990517

19. S-H Yooa, S-K Oha, Witold Pedrycz," Optimized face recognition algorithm using radial basis function neural networks and its practical applications", International journal on Neural Networks, volume 69, (2015), pp. 111-125.

20. Выбор функции активации обучения нейронной сети [Електронний ресурс]. - Режим доступу: https://monographies.ru/ru/book/section?

21. A Roy Chowdhury Tsung-Yu Lin Subhranshu Maji Erik Learned-Miller," Face Identification with Bilinear CNNs", Computer vision and pattern recognition, (2015).

22. Хемант Синг Миттал, Гарприт Каур Распознавание с использованием метода главных компонент и нейросети. - 2013.[Електронний ресурс]. - Режим доступу: http://www.ijese.org/attachments/File/v1i6/F0266 041613.pdf

23. M.Nandini, P.Bhargavi, G.Raja Sekhar,"Face Recognition Using Neural Network", International Journal of Scientific and Research Publications, vol. 3, no. 3, (2013), pp. 1-5.

24. Выбор функции активации обучения нейронной сети [Електронний ресурс]. - Режим доступу: https://monographies.ru/ru/book/section?id=2465.

25. H A. Rowley, Student Member," Neural Network-Based Face

26. Manisha M. Face Recognition Using Neural Network: A Review// Manisha M.Kasar1, Debnath Bhattacharyya1 and Tai-ho Kim2. [Електронний ресурс]. - Режим доступу: http://www.sersc.org/journals/IJSIA/vol10_no3_2016/8.pdf

27. H A. Rowley, S Baluja, T Kanade," Rotation Invariant Neural Network

28. Выбор функции активации обучения нейронной сети [Електронний ресурс]. - Режим доступу: https://monographies.ru/ru/book/section?

29. Recognize and manipulate faces from Python or from the command line - [Електронний ресурс] - Режим доступу: https://github.com/ageitgey/face_recognition

30. MTCNN face detection implementation for TensorFlow - [Електронний ресурс] - Режим доступу: https://github.com/ipazc/mtcnn

31. Deep learning-based Face detection using the YOLOv3 algorithm- [Електронний ресурс] - Режим доступу: https://github.com/sthanhng/yoloface

32. CPU Real Time face detection using Deep Learning - [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/faced-cpu-real-time-face-detection-using-deep-learning-1488681c1602

33. 1MB lightweight face detection model - [Електронний ресурс] - Режим доступу: https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB

34. He, Zhang, Ren i Sun - Deep Residual Learning for Image Recognition 2015

35. DLIB - [Електронний ресурс] - Режим доступу: http://dlib.net/

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

37. Java [Електронний ресурс] - Режим доступу: https://www.java.com/

38. C# [Електронний ресурс] - Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/

39. Python [Електронний ресурс] - Режим доступу: https://www.python.org/

40. Jquery [Електронний ресурс] - Режим доступу: https://jquery.com/

41. "Usage of JavaScript libraries for websites" [Електронний ресурс] - Режим доступу: W3Techs.

42. Bootstrap - [Електронний ресурс] - Режим доступу: https://getbootstrap.com/

43. Postgresql - [Електронний ресурс] - Режим доступу: https://www.postgresql.org/

44. Mysql - [Електронний ресурс] - Режим доступу: https://www.mysql.com/

45. Sqlite - [Електронний ресурс] - Режим доступу: https://www.sqlite.org/index.html

46. Sqlalchemy - [Електронний ресурс] - Режим доступу: https://www.sqlalchemy.org/

47. Машнин Тимур Сергеевич. Технология Web-сервисов платформы Java. — БХВ-Петербург, 2012. — С. 115. — 560 с.

48. Chapter 5 of Roy Fielding’s dissertation «Representational State Transfer (REST)»

49. Fielding discussing the definition of the REST term. [Електронний ресурс] - Режим доступу: Tech.groups.yahoo.com.

50. Docker - [Електронний ресурс] - Режим доступу: https://docs.docker.com/compose/

51. Міжнародний науковий симпозіум «Інтелектуальні рішення» - [Електронний ресурс] - Режим доступу: http://www.nas.gov.ua/UA/Messages/news/Pages/View.aspx?MessageID=4954

52. Міжнародна науково-практична конференція «Інформаційні технології та взаємодії» - [Електронний ресурс] - Режим доступу: http://iti.fit.univ.kiev.ua/uk/

53. Machine Learning is Fun! Part 4 - [Електронний ресурс] - Режим доступу: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

54. Understanding and visualizing ResNets - [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

55. РЕФЕРАТ на магістерську дисертацію - [Електронний ресурс] - Режим доступу: http://cad.kpi.ua/attachments/093_2017dm_Savelyev.pdf

ДОДАТКИ

Додаток А Фрагмент коду системи аутентифікації

```
import io
import os

import face_recognition
import numpy as np
import requests
from PIL import Image, ImageOps
from flask import request, current_app as app
from flask.views import MethodView
from flask_login import current_user

from app import db
from app.api import bp, response
from app.models import Face


def resize_image(img: bytes) -> io.BytesIO:
        basewidth = 800

        image = Image.open(io.BytesIO(img))
        img_width = image.size[0]
        if img_width > basewidth:
        wpercent = basewidth / float(image.size[0])
        hsize = int((float(image.size[1]) * float(wpercent)))
        image = image.resize((basewidth, hsize), Image.ANTIALIAS)

        # rotate according to exif
        image = ImageOps.exif_transpose(image)

        img_file_obj = io.BytesIO()
        image.save(img_file_obj, format="PNG", optimize=True, quality=95)
        img_file_obj.seek(0)
        app.logger.info(
        f"orig: {len(img)}, new: {len(img_file_obj.getvalue())} - ratio({len(img) / len(img_file_obj.getvalue())})"
        )
        return img_file_obj


class FaceApi(MethodView):
        def get(self, face_id):
        if face_id is None:
        page = request.args.get("page", 1, type=int)
        per_page = min(request.args.get("per_page", 8, type=int), 20)
        data = Face.to_collection_dict(
        Face.query,
        page=page,
        per_page=per_page,
        endpoint="api.face_api",
        id=face_id,
        )
        else:
        data = Face.query.get_or_404(face_id).to_dict()
        return response.success(data)

        def post(self):
        if "file" not in request.files:
```

```
        app.logger.error("Face image is required")
        return response.error("Face image is required.")

        file = request.files["file"]

        if file.mimetype not in app.config["FILE_ALLOWED"]:
        app.logger.error("File extension is not allowed")
        return response.error("We only allow upload file with *.png , *.jpg")

        # get name in form data
        name = request.form["name"]
        app.logger.info("Information of that face: %s", name)
        image = file.read()
        face_encodings = face_recognition.face_encodings(
        face_recognition.load_image_file(resize_image(image))
        )
        if len(face_encodings) != 1:
        msg = f"{len(face_encodings)} faces found instead of 1"
        app.logger.warning(msg)
        return response.error(msg)

        # let start save file to our storage
        face = Face(
        user_id=current_user.id, encoding=list(face_encodings[0]), name=name
        )
        db.session.add(face)
        db.session.commit()
        app.logger.info("Face saved in %s", face)

        return response.success({"id": face.id,})

        def delete(self, face_id):
        deleted = Face.query.filter(Face.id == face_id).delete()
        Face.query.session.commit()
        if deleted > 0:
        return response.success("Deleted", 201)
        return response.error("Not found", 404)


# router for recognize a unknown face
@bp.route("/faces/recognize", methods=["POST"])
def recognize():
        if "file" not in request.files:
        return response.error("Image is required")

        file = request.files["file"]
        # file extension valiate
        if file.mimetype not in app.config["FILE_ALLOWED"]:
        return response.error("File extension is not allowed")
        image = file.read()
        face_encoding_to_check = face_recognition.face_encodings(
        face_recognition.load_image_file(resize_image(image))
        )
        if len(face_encoding_to_check) != 1:
        msg = f"{len(face_encoding_to_check)} faces found instead of 1"
        app.logger.warning(msg)
        return response.error(msg)

        user_faces = list(current_user.faces)
        encodings = [np.array(face.encoding) for face in user_faces]

        matches = face_recognition.compare_faces(encodings, face_encoding_to_check[0])
```

```
        found = [face.name for face, match in zip(user_faces, matches) if match][:5]

        if found:
        return response.success({"message": ", ".join(found)})
        return response.error(
        "Sorry we can not found any people matched with your face image, try another image",
        404,
        )


@bp.route("/login", methods=["GET", "POST"])
def login():
        if current_user.is_authenticated:
        return redirect(url_for("main.index"))
        form = LoginForm()
        if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user is None or not user.check_password(form.password.data):
        flash("Invalid username or password")
        return redirect(url_for("auth.login"))
        login_user(user, remember=form.remember_me.data)
        next_page = request.args.get("next")
        if not next_page or url_parse(next_page).netloc != "":
        next_page = url_for("main.index")
        return redirect(next_page)
        return render_template("auth/login.html", title="Sign In", form=form)


@bp.route("/logout")
def logout():
        logout_user()
        return redirect(url_for("main.index"))


@bp.before_app_request
def before_request():
        if current_user.is_authenticated:
        current_user.last_seen = datetime.utcnow()
        db.session.commit()


@bp.route("/", methods=["GET"])
@login_required
def index():
        return render_template("index.html", title="Home")
```

Додаток Б Фрагмент коду порівняння для порівняння способів розпізнавання обличчя

```
import logging
import os
import time
from abc import ABCMeta, abstractmethod
from dataclasses import dataclass
from pathlib import Path
from typing import List, Tuple

import cv2
import numpy

from tester.config import SAVE_IMAGES_DIR

log = logging.getLogger(__name__)


@dataclass
class FaceBox:
    x1: int
    x2: int
    y1: int
    y2: int

    @property
    def pointa(self) -> Tuple[int, int]:
    return self.x1, self.y1

    @property
    def pointb(self) -> Tuple[int, int]:
    return self.x2, self.y2


class BaseModel(metaclass=ABCMeta):
    scale = 1

    @property
    @abstractmethod
    def model_name(self):
    pass

    def __init__(self):
    self.total_detection_time = 0
    self.detected_faces_num = 0

    def load_image(self, path: str) -> numpy.ndarray:
    return cv2.imread(path)

    def save_image(self, image_data: numpy.ndarray, filename: str):
    filename = os.path.join(
    SAVE_IMAGES_DIR,
    f"{self.__class__.__name__.lower()}_{self.scale}_{filename}",
    )
    cv2.imwrite(
    filename, image_data,
    )

    def add_text_to_image(self, image_data: numpy.ndarray, text: str):
    font = cv2.FONT_HERSHEY_DUPLEX
```

```
            cv2.putText(image_data, text, (20, 20), font, 0.5, (255, 255, 255), 1)

            def add_face_boxes(self, image_data, boxes: List[FaceBox]):
            for box in boxes:
            cv2.rectangle(
            image_data, box.pointa, box.pointb, (0, 255, 0), 2,
            )

            def test_detection(self, image_path: Path):
            image_data = self.load_image(image_path.as_posix())

            start_time = time.time()
            face_boxes = self.detect_faces(image_data)
            duration = time.time() - start_time
            self.total_detection_time += duration
            self.detected_faces_num += len(face_boxes)
            model = self.__class__.__name__.lower()
            text = f"{model} took {duration} to get {len(face_boxes)} faces in {image_path.name}"

            # log.info(text)

            self.add_text_to_image(image_data, text)
            self.add_face_boxes(image_data, face_boxes)
            self.save_image(image_data, image_path.name)

            @abstractmethod
            def detect_faces(self, image_data: numpy.ndarray) -> List[FaceBox]:
            pass


import csv
import warnings
from typing import Tuple

warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.simplefilter(action="ignore", category=UserWarning)

import logging
import sys
from pathlib import Path

from tester.config import DATA_DIR
from tester.detection.model import (
            FaceRecognition,
            FaceD,
            MTCNN,
            UltraLight,
            YOLOV3,
)
from tester.recognition.model import FaceRecognitionDet

DETECTION_MODELS = [FaceRecognition, FaceD, MTCNN, UltraLight, YOLOV3]
RECOGNITION_MODELS = [FaceRecognitionDet]

log = logging.getLogger(__name__)


def setup_logging():
            handlers = [logging.StreamHandler(sys.stdout)]
            logging.basicConfig(
            handlers=handlers,
            format=(
```

```
        "{asctime:^} | {levelname: ^8} | "
        "{filename: ^14} {lineno: <4} | {message}"
        ),
        style="{",
        datefmt="%d.%m.%Y %H:%M:%S",
        level=logging.DEBUG,
        )


def get_detection_dataset(max=None):
        for i, img in enumerate(Path(DATA_DIR, "TestImg", "lfw").rglob("*.jpg")):
        if max and i >= max:
        return img
        yield img


def format_image(name, id):
        return f"{name}_{id.zfill(4)}.jpg"


def parse_image_pair(string: str) -> Tuple[Path, Path, bool]:
        image_dir = Path(DATA_DIR, "TestImg", "lfw")
        elems = string.strip().split("\t")
        if len(elems) == 3:
        name, id1, id2 = elems
        first_image = image_dir.joinpath(name, format_image(name, id1))
        second_image = image_dir.joinpath(name, format_image(name, id2))
        elif len(elems) == 4:
        name1, id1, name2, id2 = elems
        first_image = image_dir.joinpath(name1, format_image(name1, id1))
        second_image = image_dir.joinpath(name2, format_image(name2, id2))
        else:
        raise ValueError(f"Invalid number of elements: {elems}")

        return first_image, second_image, len(elems) == 3


def get_recognition_dataset(max=None):
        pairs_file = Path(DATA_DIR, "TestImg", "lfw", "pairs.txt")
        with pairs_file.open("r", encoding="UTF-8") as f:
        sets, pairs = map(int, f.readline().split("\t"))
        for i, line in enumerate(f):
        if max and i >= max:
        return parse_image_pair(line)
        yield parse_image_pair(line)
        # for _ in range(sets):
        #         for _ in range(pairs):
        #         yield parse_image_pair(f.readline()), True
        #         for _ in range(pairs):
        #         yield parse_image_pair(f.readline()), False


def test_detection():
        models = [model() for model in DETECTION_MODELS]
        # csvfile = open("detection.csv", "a", newline="")
        # spamwriter = csv.writer(csvfile)
        for detector in models:
        images_num = 0
        for image in ["data\TestImg\crowd2.png"]:#  get_detection_dataset():
        for i in range(5):
        detector.test_detection(Path(image))
        images_num += 1
        # spamwriter.writerow(
```

```
        #         [
        #         detector.model_name,
        #         detector.total_detection_time,
        #         detector.detected_faces_num,
        #         images_num,
        #         detector.detected_faces_num / images_num,
        #         ]
        # )
        log.info(
        f"{detector.model_name}\t"
        f"{detector.total_detection_time / 5}\t"
        f"{detector.detected_faces_num / 5}"
        )
        # csvfile.close()


def test_recognition():
        models = [model() for model in RECOGNITION_MODELS]
        csvfile = open("recognition.csv", "a", newline="")
        spamwriter = csv.writer(csvfile)
        for recognizer in models:
        images_num = 0
        for imagea, imageb, match in get_recognition_dataset():
        if match:
        recognizer.test_match(imagea, imageb)
        else:
        recognizer.test_mismatch(imagea, imageb)
        images_num += 2
        spamwriter.writerow(
        [
        "model_name",
        "total_recognition_time",
        "recognized_faces_num",
        "match_pairs_success",
        "match_pairs_total",
        "match_pairs_success / match_pairs_total",
        "mismatch_pairs_success",
        "mismatch_pairs_total",
        "mismatch_pairs_success / mismatch_pairs_total",
        "images_num",
        "recognized_faces_num / images_num",
        ]
        )
        row = [
        recognizer.model_name,
        recognizer.total_recognition_time,
        recognizer.recognized_faces_num,
        recognizer.match_pairs_success,
        recognizer.match_pairs_total,
        recognizer.match_pairs_success / recognizer.match_pairs_total if recognizer.match_pairs_total else 0,
        recognizer.mismatch_pairs_success,
        recognizer.mismatch_pairs_total,
        recognizer.mismatch_pairs_success / recognizer.mismatch_pairs_total if recognizer.mismatch_pairs_total else
0,
        images_num,
        recognizer.recognized_faces_num / images_num,
        ]

        spamwriter.writerow(row)
        log.info("\t".join(map(str, row)))
        csvfile.close()
```

```
def main():
        setup_logging()
        test_detection()
        # test_recognition()


if __name__ == "__main__":
        main()
```