Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____Олена ОЛЬХОВСЬКА

(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ »

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Різник Нікіта Сергійович

_____«__»_____202_р.

(підпис)

Науковий керівник к.ф.-м.н.,доц.,_____

_____«__»_____202_р.

(підпис)

Рецензент

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Програмна реалізація застосунку для розпізнавання обличчя»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Різник Нікіта Сергійович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні відеоматеріали, технічна документація.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Переваги та недоліки систем розпізнавання облич
2.2. Огляд сучасного програмного забезпечення для розпізнавання облич
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Основні принципи роботи систем розпізнавання облич
3.2. Алгоритми розпізнавання облич
4. ПРАКТИЧНА ЧАСТИНА
4.1. Алгоритм роботи додатка для розпізнавання облич
4.2. Обґрунтування вибору технологій та мови програмування
4.3. Програмна реалізація застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
Перелік графічного матеріалу: Необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична реалізація	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.
Здобувач вищої освіти Різник Нікіта Сергійович
Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на
_____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Програмна реалізація застосунку для розпізнавання обличчя»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
ступеня бакалавра
Різник Нікіта Сергійович
Прізвище, ім'я, по батькові

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Переваги та недоліки систем розпізнавання облич
2.2. Огляд сучасного програмного забезпечення для розпізнавання облич
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Основні принципи роботи систем розпізнавання облич
3.2. Алгоритми розпізнавання облич
4. ПРАКТИЧНА ЧАСТИНА
4.1. Алгоритм роботи додатка для розпізнавання облич
4.2. Обґрунтування вибору технологій та мови програмування
4.3. Програмна реалізація застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ Нікіта РІЗНИК
«____» _____ 202_ р.

АНОТАЦІЯ

Записка: 64 сторінки, літературних джерел – 14, рисунків – 44.

Мета роботи – програмна реалізація самостійного застосунку з графічним інтерфейсом для розпізнавання облич.

Об'єкт розробки – застосунок для розпізнавання обличчя.

Методи дослідження та інформаційне забезпечення – аналіз числених джерел з навчальними матеріалами, серед яких присутні відеоматеріали, статті, технічні документації від розробників, подібні розроблені системи, тощо.

Результати дослідження. Розроблено самостійний додаток для розпізнання облич. Для додатку було розроблено графічний інтерфейс користувача, який містить всю необхідну інформацію для управління системою. Присутні відповідні кнопки управління для початку та зупинки процесу, вибір камери користувача, на випадок якщо їх декілька, та кнопки для управління зображеннями, з можливістю їх видалення та додавання до застосунку. Також додано список для зручного спостерігання в який час було розпізнане те чи інше додане до системи обличчя. Реалізований журнал змін які відбуваються в системі в окремому полі, в якому звітуються всі події з точним часом коли вони були здійснені. Для зручного використання користувачем, систему було зібрано до самостійного додатка, без необхідності встановлення численних бібліотек та середи розробки.

Рекомендації щодо використання результатів дослідження. Додаток на остаточний момент розробки можна використовувати в багатьох напрямках для спрощення щоденних задач. Одним з видів використання може стати спосіб моніторингу відвідування занять студентами. Необхідно завантажити одне фото, яке містить обличчя студента, встановити камеру на вході до аудиторії, та при його розпізнанні системою, буде в списку буде відмічено точний час коли студент відвідав лекцію. Іншим способом може бути встановлення системи на вході до підприємства з великою кількістю співробітників, таким чином на пункті пропуску, охоронець може перевіряти чи є людина перед ним, співробітником та має допуск.

Ключові слова: МОВА ПРОГРАМУВАННЯ, ЗАСТОСУНОК, РОЗПІЗНАННЯ ОБЛИЧЧЯ, САМОСТІЙНИЙ ДОДАТОК, ГРАФІЧНИЙ ІНТЕРФЕЙС, PYTHON, FACE RECOGNITION, OPENCV.

ЗМІСТ

ВСТУП

Системи розпізнавання обличчя є одним із найактуальніших напрямків досліджень у галузі комп'ютерного зору та біометрії. Їх широке застосування в таких сферах, як безпека, маркетинг, соціальні мережі та робототехніка, зумовлене постійним зростанням обсягів візуальної інформації та необхідністю забезпечення надійної ідентифікації особи.

Необхідність у системах розпізнання облич зумовлена розвитком технологій та потреб людства. Вже звичний спосіб перевірки, такі як паролі чи більш нові, такі як відбитки, повільно стають менш зручними чи актуальними, через можливу неточність, швидкість своєї роботи тощо. Винахід та розвиток нових систем безпеки та ідентифікації є цілком логічним та очікуваним кроком в еволюції технологій чи суспільства. Потреба у швидкій та точні ідентифікації користувача, значно спрощує та пришвидшує будь-які сфери в якому використовуються нові технології. Та для повного розвитку систем розпізнання облич, щоб можна було повністю покластись, потребін час та практика. Метою кваліфікаційної роботи є програмна реалізація доступної системи, з використанням бібліотек наявних у відкритому використанні, для розпізнавання облич.

Об’єктом розробки є самостійний додаток для розпізнання облич.

Предмет розробки — реалізація застосунку для розпізнання облич з використанням бібліотек у відкритому доступі.

Робота складається з чотирьох розділів. У першому розділі розглянуто постановку задачі. У другому розділі розглянуто інформаційний огляд питання. У третьому розділі представлено теоретичний опис роботи. У четвертому показано практичне виконання роботи.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Головною задачею кваліфікаційної роботи є програмна реалізація самостійного застосунку для розпізнавання облич, на мові Python з використання бібліотек у відкритому доступі яка матиме наступні функціональні можливості:

1. Мати можливість завантажити свої зображення обличчя для подальшої ідентифікації, порівняння, та знаходження обличчя яке система отримує в певний момент.
2. Розпізнання за допомогою бібліотек у відкритому доступі, таких як OpenCV та face_recognition, облич наявних на завантаженому зображенні, щоб мати можливість їх ідентифікувати пізніше.
3. Критично важливим, окрім бібліотек, є етап захоплення облич в реальному часі, для чого використовується наявна вебкамера користувача, підключена до ПК.
4. Ідентифікація отриманих попередньо зображень, опрацьованих та доданих до бази, та порівняння з аналізованими актуальними зображеннями користувача отриманими за допомогою вебкамера.
5. Порівняння зображень отриманих з вебкамери та попередньо завантажених до бази облич системи.
6. Вивід результату роботи системи з демонстрацією у реальному часі на основі отриманих та опрацьованих попередньо даних

Завданням кваліфікаційної роботи з системи розпізнання облич, є програмна реалізація сучасної системи з розпізнання у реальному часі, з використанням наявних у вільному доступі ресурсів для розробки роботиу.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Переваги та недоліки систем розпізнавання облич

Переваги систем розпізнавання облич:

- Висока точність і ефективність

Сучасні алгоритми розпізнавання облич, які базуються на методах машинного навчання та штучного інтелекту, здатні досягати високої точності в ідентифікації осіб. Багато сучасних систем використовують нейронні мережі глибокого навчання для покращення точності аналізу зображень. Наприклад, алгоритми можуть ідентифікувати обличчя навіть у великих натовпах або при поганих умовах освітлення. Така ефективність важлива для сфери безпеки, контролю доступу та спостереження [3,4].

- Безконтактна ідентифікація

Однією з ключових переваг систем розпізнавання облич є те, що вони не потребують фізичного контакту з користувачем. Це робить їх особливо корисними в умовах пандемій або в місцях з високим ризиком інфекцій. На відміну від технологій зчитування відбитків пальців або карткових систем, користувачу не потрібно взаємодіяти з будь-яким обладнанням, що значно знижує ризик поширення вірусів [4].

- Швидкість обробки

Системи розпізнавання облич працюють дуже швидко, що дозволяє їх використовувати в реальному часі. Це є критично важливим фактором для систем відеоспостереження та контролю на різних об'єктах, таких як аеропорти, вокзали та стадіони. Завдяки швидкості обробки, система може ідентифікувати підозрілих осіб або порушників у режимі реального часу, що дозволяє оперативно реагувати на потенційні загрози [3,4].

- Інтеграція з іншими системами безпеки

Системи розпізнавання облич легко інтегруються з іншими технологіями безпеки, такими як системи відеоспостереження, бази даних

правоохоронних органів, банківські системи тощо. Це дозволяє створити комплексні рішення для підвищення безпеки в різних середовищах. Також їх можна використовувати для контролю доступу в будівлі або для персоналізації послуг, як-от у банківських установах чи готелях [3,4].

- Підвищення рівня зручності та автоматизації

Розпізнавання облич дозволяє автоматизувати процеси аутентифікації та ідентифікації. Це значно зменшує потребу в використанні паролів, кодів або фізичних ідентифікаторів (як картки). У результаті користувачі можуть отримати доступ до своїх пристроїв або акаунтів простим поглядом на камеру. Ця технологія також може використовуватися для розумних домівок, персоналізованих сервісів та маркетингових платформ, що значно підвищує рівень зручності для користувачів [3,4].

Недоліки систем розпізнавання облич:

- Проблеми з приватністю та конфіденційністю

Одним із найбільших викликів є занепокоєння щодо конфіденційності даних. Системи розпізнавання облич здатні збирати та зберігати біометричні дані, що викликає побоювання щодо зловживань цими даними. У багатьох країнах ще не розроблені належні правові рамки для захисту таких даних, що може призводити до потенційних порушень приватності. Наприклад, зібрані зображення можуть використовуватися без згоди осіб або навіть передаватися третім сторонам без їхнього відома [3,4].

- Вразливість до зламів та підробок

Незважаючи на високу точність, системи розпізнавання облич можуть бути вразливими до атак. Зокрема, кіберзлочинці можуть використовувати підроблені фотографії, відео або навіть технології DeepFake для обману системи. Також існує ризик несанкціонованого доступу до баз даних, де зберігаються біометричні дані користувачів. Це може призвести до серйозних наслідків, включаючи фінансові втрати та порушення конфіденційності [3,4].

- Проблеми з точністю у специфічних умовах

Хоча системи розпізнавання облич можуть працювати дуже добре за оптимальних умов, вони можуть мати труднощі з точністю в умовах недостатнього освітлення, низької якості зображення або при зміненому зовнішньому вигляді користувача (наприклад, борода, окуляри, маски). Це обмежує їхнє використання в певних ситуаціях, наприклад, під час масових заходів або в екстремальних умовах [3,4].

- Етичні питання та масове спостереження

Використання систем розпізнавання облич в публічних місцях часто викликає етичні суперечки. Масове спостереження за допомогою таких систем може призводити до зловживань владою, порушення прав людини та встановлення надмірного контролю за громадянами. У деяких країнах технологія вже використовується для тотального спостереження, що викликає критику з боку правозахисних організацій та суспільства [3].

2.2. Огляд сучасного програмного забезпечення для розпізнавання облич

1. Face++, платформа для розпізнавання облич. Її відмінність від інших рішень полягає в тому, що вона здатна не тільки розпізнавати обличчя, але й визначати емоційний стан, вік, стать та навіть аналізувати ключові риси обличчя. Face++ інтегрується через API, що робить її ідеальним рішенням для розробників, які хочуть додати функції розпізнавання облич до своїх програм.

Face++ вирізняється серед інших високою точністю навіть за умов низької якості зображень або слабкого освітлення. Окрім цього, платформа надає можливості для відстеження емоцій у реальному часі, що є корисним у маркетингу та дослідженні споживчої поведінки [4].

2. Amazon Rekognition від Amazon Web Services (AWS) — хмарне рішення, яке, окрім розпізнавання облич, може аналізувати зображення та відео для виявлення інших об'єктів і сценаріїв. Його перевага полягає в тому, що система дозволяє не лише ідентифікувати обличчя, але й порівнювати їх із величезною базою даних та створювати власні колекції осіб. Це робить Rekognition

потужним інструментом для великих компаній та державних установ, що потребують високої масштабованості та швидкість обробки.

На відміну від інших систем, Amazon Rekognition інтегрується з широким спектром хмарних сервісів AWS, що дозволяє будувати складні системи для обробки та аналізу великих обсягів даних. Це робить платформу популярною серед корпорацій, які вже використовують екосистему AWS [4].

3. Microsoft Azure Face API є частиною Microsoft Cognitive Services і забезпечує високий рівень точності при розпізнаванні облич. Ця платформа дозволяє визначати вікові характеристики, стать, емоційні стани осіб, а також створювати бази даних для подальшого порівняння. Її перевага полягає у високій інтеграції з іншими сервісами Microsoft, що дає змогу легко впроваджувати функції розпізнавання облич у корпоративні рішення.

Azure Face API використовує передові алгоритми машинного навчання, що дозволяє йому досягати високої точності при роботі в реальному часі. Це ідеальне рішення для бізнесів, що шукають надійні інструменти для масштабованого використання, зокрема у системах контролю доступу, автоматизації клієнтських взаємодій або маркетингових дослідженнях [4].

4. FaceFirst — Система працює в реальному часі та може інтегруватися з чинними системами відеоспостереження. Вона здатна швидко аналізувати відеопотоки, зіставляючи обличчя з базами даних для ідентифікації відомих правопорушників, відстеження підозрілих осіб чи надання персоналізованого сервісу клієнтам. Обслуговує клієнтів у таких галузях, як роздрібна торгівля, правоохоронні органи, транспорт і охорона об'єктів [4].

5. OpenCV — це бібліотека з відкритим кодом, яка надає розробникам інструменти для роботи з комп'ютерним зором, включаючи розпізнавання облич. OpenCV є найпоширенішою серед розробників завдяки своїй безкоштовності та гнучкості. Вона підтримує широкий спектр мов програмування, таких як Python, C++ і Java, що робить її універсальним інструментом для створення робіт з використанням комп'ютерного зору.

OpenCV використовується в багатьох наукових та комерційних роботах, оскільки надає гнучкі можливості для модифікації, дозволяючи розробникам будувати власні алгоритми для різних застосувань, включаючи мобільні застосунки та системи реального часу [1].

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Основні принципи роботи систем розпізнавання облич

Системи розпізнавання облич є складовою сучасних технологій, що поєднують алгоритми машинного навчання, обробки зображень та статистичного аналізу для ідентифікації осіб за їхніми обличчями. Основою роботи таких систем є поетапна обробка вхідних зображень, що включає виявлення облич, витягнення ключових ознак та їх подальше зіставлення з базами даних [3,4].

Етап збору зображень передбачає отримання даних з різних джерел, таких як камери спостереження, мобільні пристрої або вебкамера. Важливою складовою цього процесу є попередня обробка, що включає нормалізацію яскравості, корекцію контрасту та масштабування. Дані операції забезпечують одноманітність і підвищують якість подальшого аналізу, зменшуючи вплив факторів, як-от неоднорідне освітлення або перешкоди [5].

Виявлення облич на зображеннях здійснюється за допомогою алгоритмів, що дозволяють ефективно виділити області з потенційними обличчями. Найпоширеніші методи:

- Haar-класифікатори – метод, використовує каскадні класифікатори для швидкої детекції облич. Цей підхід базується на обчисленні інтенсивностей пікселів та виявленні особливих рис [1].
- Метод HOG (Histogram of Oriented Gradients) – алгоритм, що аналізує напрямки градієнтів для визначення контурів облич [1,2].
- Глибокі нейронні мережі – CNN (Convolutional Neural Networks), що застосовуються для виявлення облич завдяки своїм потужним можливостям у виявленні складних структурних елементів [1,2,7].

Етап вирівнювання передбачає корекцію позиції обличчя для забезпечення однакового масштабу та орієнтації. Цей процес включає аналіз ключових точок обличчя (очі, ніс, рот) і виконання афінних перетворень. Вирівнювання

підвищує точність розпізнавання, оскільки дозволяє стандартизувати дані та забезпечити зменшення впливу різних ракурсів [2,3].

Виділення ознак є центральним етапом, який визначає ефективність і точність розпізнавання облич. Традиційні методи, такі як Local Binary Patterns (LBP) [2], використовують гістограми для опису текстурних характеристик облич. Проте сучасні системи дедалі частіше застосовують глибокі нейронні мережі, які здатні витягувати багатовимірні ознаки та забезпечувати стійкість до змін освітлення, експресії та часткових перекриттів.

На етапі порівняння ознак використовуються метрики відстаней, наприклад, евклідова відстань або косинусна подібність, для зіставлення векторів ознак зі зразками у базі даних [1,2,6,7]. Системи можуть здійснювати ідентифікацію в режимі 1:1 (верифікація) або 1 (ідентифікація), де обличчя порівнюється з усіма записами бази даних для визначення найкращого збігу. Для забезпечення високої точності системи можуть використовувати додаткові методи валідації. Наприклад, застосування алгоритмів класифікації, таких як Support Vector Machine (SVM) [2,7], дозволяє зменшити кількість хибно позитивних і хибно негативних результатів, забезпечуючи кращу адаптивність до змін у зовнішності людини та зовнішніх умов.

Системи розпізнавання облич широко використовуються в галузях безпеки, доступу до пристроїв, контролю кордонів, маркетингу та індивідуалізації користувацьких досвідів. Однак існують численні виклики, включно з проблемами конфіденційності, етичними питаннями та труднощами в ідентифікації при змінах у зовнішності (наприклад, з віком, зміною зачіски чи носінням окулярів) [3].

3.2. Алгоритми розпізнавання облич

Алгоритми розпізнавання облич є ключовим елементом, що забезпечують точне визначення та ідентифікацію осіб у цифрових зображеннях. Принцип роботи змінився від класичних статистичних підходів

до сучасних глибоких нейронних мереж, які можуть ефективно обробляти великі обсяги даних і витягувати складні ознаки. Ранні алгоритми базувалися на простих методах аналізу зображень, таких як:

- Метод головних компонент (PCA, або метод власних облич): один із перших алгоритмів розпізнавання облич, запропонований у 1990-х роках. PCA перетворює зображення облич на набір ознак, відомих як власні обличчя, використовуючи для цього принцип зниження розмірності та лінійної алгебри. Алгоритм ефективний для розпізнавання за стандартних умов освітлення, однак має обмеження при зміні виразу обличчя або кута нахилу [2].
- Метод розпізнавання за характеристиками (Feature-based): алгоритми цього типу шукають специфічні ознаки обличчя, такі як очі, ніс, рот, та створюють з них набір дескрипторів. Вони демонструють більшу стійкість до змін освітлення, але менш ефективні у випадках складних ракурсів [2].

Алгоритм HOG [2] базується на аналізі градієнтів яскравості пікселів для визначення контурів об'єктів. Він генерує вектор ознак, що використовується для класифікації зображень. Використання HOG у поєднанні з методами машинного навчання, такими як метод опорних векторів (SVM) [2], дозволяє розпізнавати обличчя з високою точністю. Зі стрімким розвитком обчислювальних потужностей і доступом до великих обсягів даних алгоритми глибокого навчання стали основним підходом у розпізнаванні облич. Вони дозволяють витягати складні ознаки та розпізнавати обличчя з високою точністю незалежно від умов освітлення чи виразів обличчя.

Конволюційні нейронні мережі (CNN)**:** CNN показали високі результати в задачах класифікації та детекції об'єктів, включно з обличчями. Ці мережі складаються з кількох шарів згорток, що дозволяють витягувати різнорівневі ознаки зображень, від простих країв до складних структур [2, 6].

Алгоритми витягнення ознак включають використання технік машинного навчання для створення унікальних векторів, що характеризують обличчя,

наприклад Dlib. Dlib's Face Recognition: бібліотека, що використовує глибокі навчальні моделі для створення 128-вимірних векторів, які слугують основою для порівняння облич [2].

Для прикладу, пропонується розглянути алгоритм роботи принципу Histogram of Oriented Gradients (HOG) на основі статті яка аналізує дослід на цей метод.

Першим кроком, імпортується зображення розміром 128x64 та конвертується у градацію сірого(рис. 3.1), задля оптимізації ресурсів та визначення бажаного предмету на фото[8].

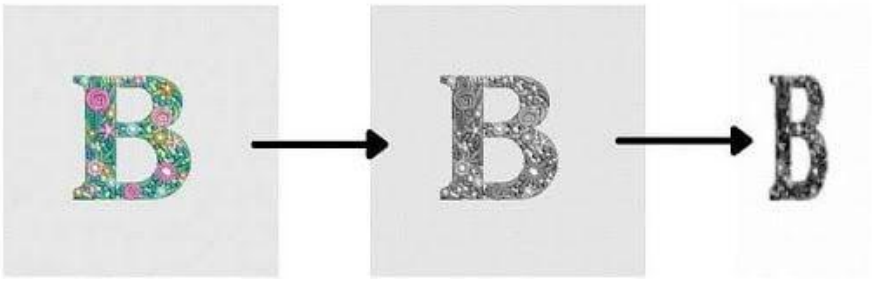

Рисунок 3.1 – Спрощення зображення

Після, градієнт зображення обчислюється шляхом комбінування величини та кута зображення. Розглянемо блок із 3x3 пікселів, спочатку для кожного пікселя обчислюються Gx та Gy, використовуючи наступну формулу для кожного значення пікселя, де r та c є рядком та стовпцем пікселя: [8].

$$G_x(r,c) = I(r,c+1) - I(r,c-1) G_y(r,c) = I(r-1,c) - I(r+1,c)$$

Після обчислення Gx та Gy, за наступною формулою обчислюється величина та кут кожного пікселя: [8]. Дві ці формули необхідні для визначення величини яскравості та напрямку зміни пікселів, для коректного розпізнання форми об’єкта.

$$\text{Величина}(\mu) = \sqrt{G_x^2 + G_y^2}\ \text{Кут}(\theta) = \left|\tan^{-1}\left(G_y/G_x\right)\right|$$

Візуалізація величини та кута зображень відповідно(рис. 3.2).

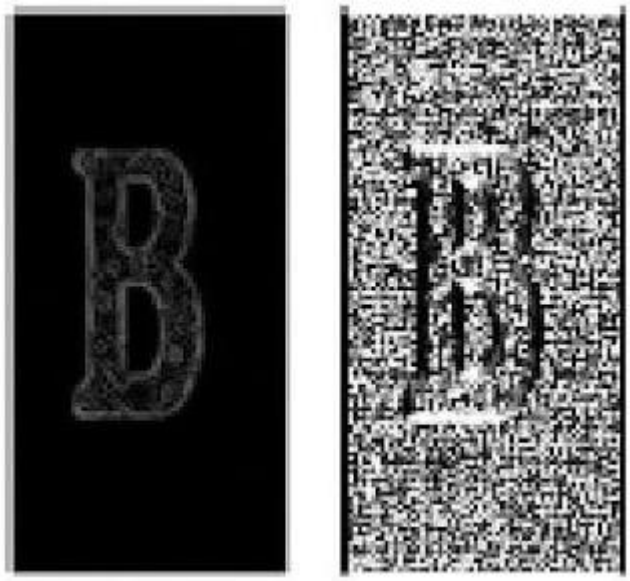

Рисунок 3.2 – Візуалізація величини та кута

Наступним кроком є розподіл значень по окремих осередках або клітинах зображення. Зазвичай зображення розбивається на клітини розміру 8x8 пікселів(рис. 3.3). Це дозволяє обробляти інформацію про градієнти в локальних ділянках зображення, а також отримати детальнішу інформацію про контури в кожному регіоні. Для кожного блоку обчислюється гістограма з 9 точок(далі біни), кожна з яких відповідає певному діапазону кутів (по 20 градусів для кожного)(рис. 3.4). Кожне з цих значень представляє інтенсивність градієнтів, що потрапляють в цей діапазон [8].

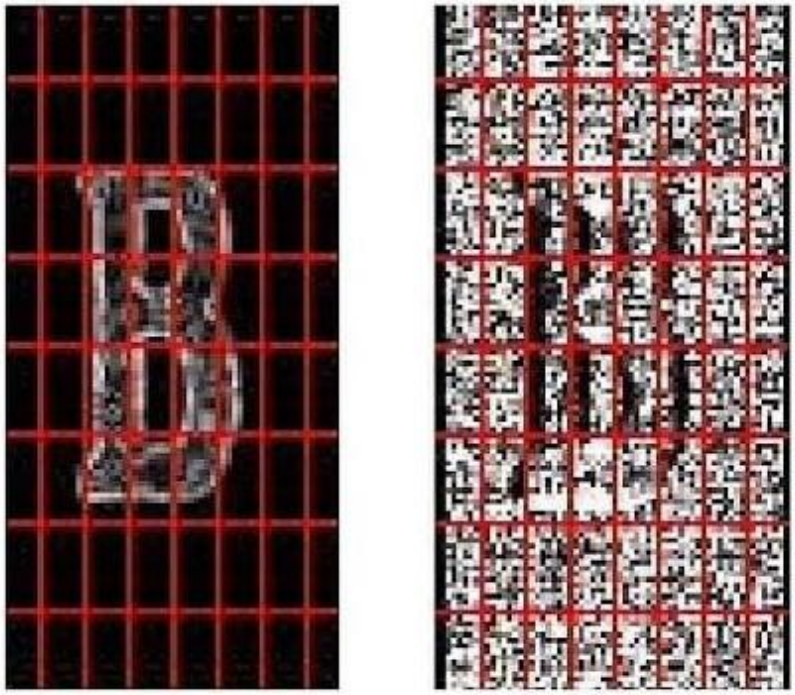

Рисунок 3.3 – Розбивка на сітку

Значення									
Біни	0	20	40	60	80	100	120	140	160

Рисунок 3.4 – Гістограма пікселю

Розбите раніше зображення на 8x8, має 64 різних клітин, і наступні обрахунки будуть виконані для всіх 64 значень глибини та куту. Всі клітини додадуть свої значення Vj і Vj+1 до j-го та (j+1)-го індексів відповідно [8]. На наступній формулі зображено принцип по якому розбиваються біни:

$$\text{Кількість бінів} = 9(\text{від } 0° \text{ до } 180°)$$
$$\text{Розмір кроку}(\Delta 0) = 180° / \text{Кількість бінів по } = 20°$$

Треба визначити, до якого саме біну належить градієнт певного пікселя. Порівнюючи орієнтацію пікселя з лівою і правою межами біна, можна визначити, в який бін його слід помістити за наступною формулою: [8].

$$\Delta\theta \cdot j, \Delta\theta \cdot (j+1)$$

Формула яка дозволяє визначити точне розташування центру кожного біна в гістограмі орієнтацій градієнтів. Це важливо для того, щоб правильно розподілити орієнтації пікселів по бінах і отримати точну гістограму: [8].

$$C_j = \Delta\theta(j + 0.5)$$

Для кожної клітинки в блоці ми спочатку обчислюємо індекс j-го біна, а потім значення, яке буде додано до j-го та (j+1)-го бінів відповідно. Значення обчислюється за наступними формулами: [8].

$$j = \left\lfloor \left(\frac{\theta}{\Delta\theta} - \frac{1}{2} \right) \right.$$

$$V_j = \mu \cdot \left[\frac{\theta}{\Delta\theta} - \frac{1}{2} \right]$$

$$V_{j+1} = \mu \cdot \left[\frac{\theta - C_j}{\Delta\theta} \right]$$

Після того, як обчислення гістограм завершено для всіх блоків, чотири блоки об'єднуються разом, утворюючи новий блок розміром 2x2, переміщуючи зліва направо, обчислюючи та закриваючи всі клітини [8]. Щоб закрити все зображення знадобиться 7x7 рухів нового блоку. При закритті кожної клітини, обраховується 9 бінів, це має наступний вигляд (рис. 3.5).

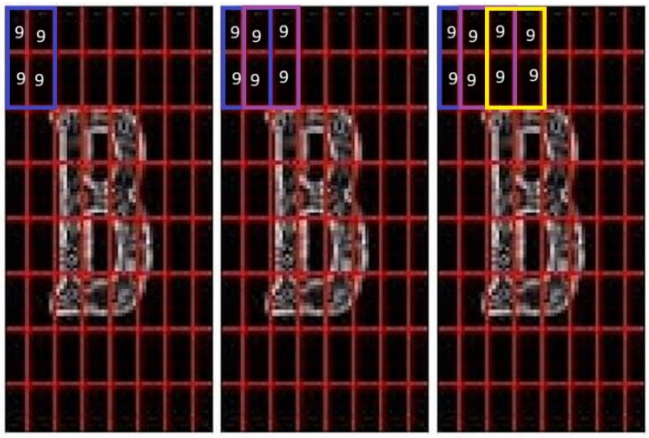

Рисунок 3.5 – Візуалізація розрахунку 2x2 блоку

В результаті отримується наступний HOG вектор: 7x7x36=1764 (де 36 – сума 2x2 блоку в якому міститься 9 бінів в кожній клітинці).

Також на основі гістограми бінів обрахованих раніше, можна візуалізувати наступне зображення, яке показує вектор зображення(рис 3.6) [8].

Рисунок 3.6 – Візуалізація вектору

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Алгоритм роботи додатка для розпізнавання облич

Система розпізнавання облич ґрунтується на порівнянні закодованих зображень відомих облич із зображеннями, отриманими в режимі реального часу.

На етапі ініціалізації система створює порожні контейнери для зберігання закодованих зображень відомих осіб та їхніх імен. Також встановлюється коефіцієнт масштабування для зменшення розміру кадру з метою підвищення продуктивності. Наступний етап - завантаження зображень відомих осіб. Система шукає всі файли зображень у визначеному каталозі. Для кожного зображення виконується ряд операцій:

- Зображення завантажується в пам'ять.
- Зображення конвертується з колірного простору BGR в RGB, оскільки бібліотека face_recognition очікує дані саме в такому форматі.
- З кожного зображення виділяється область зацікавленості - обличчя. Для цього використовується функція face_recognition.face_locations.
- З виділеної області обличчя обчислюється вектор ознак за допомогою функції face_recognition.face_encodings.
- Отриманий вектор ознак та ім'я файлу зображення (яке вважається ім'ям особи на зображенні) додаються до відповідних контейнерів.

Після завантаження зображень відомих осіб система переходить до етапу виявлення та розпізнавання обличчя в кадрі відеопотоку. Для цього з потоку захоплюється кадр, який потім масштабується за допомогою встановленого раніше коефіцієнта. Масштабування дозволяє зменшити розмірність даних та прискорити обчислення. Після масштабування кадр також конвертується в формат RGB. На масштабованому кадрі шукаються області, що містять обличчя. Знайдені області разом із самим кадром передаються функції face_recognition.face_encodings для обчислення векторів

ознак цих облич. Для кожного обличчя в кадрі система порівнює його вектор ознак із векторами відомих осіб, що зберігаються. Порівняння здійснюється за допомогою функції face_recognition.compare_faces.

Для визначення найбільш схожої відомої особи обчислюються відстані між векторами ознак поточного обличчя та всіх відомих облич. Відстань розраховується функцією face_recognition.face_distance. Менша відстань означає більшу схожість. Таким чином, мінімальне значення відстані вказує на індекс найбільш схожої відомої особи. Якщо для поточного обличчя знайдено схоже обличчя серед відомих, то ім'я цієї особи із контейнера відомих імен додається до масиву імен. В іншому випадку, до масиву додається значення "Невідомий".

Після визначення імен для всіх облич в кадрі система виконує візуалізацію результатів. На копії кадру для кожного обличчя малюється прямокутник, що позначає область розташування обличчя. Також біля кожного прямокутника підписується ім'я розпізнаної особи. Отримане зображення із намальованими прямокутниками та підписами подається на екран за допомогою функції cv2.imshow. Після зупинення циклу виконується вивільнення ресурсів, зайнятих відеозахопленням та вікном відображення.

4.2. Обґрунтування вибору технологій та мови програмування

Для роботи було використано ряд бібліотек, присутніх у відкритому доступі для завантаження:

1. OpenCV — найпопулярніша бібліотека, яка містить широкий набір функцій для обробки зображень та відео, таких як фільтрація, виявлення контурів, виділення ключових точок, сегментація, трекінг та багато іншого.

Бібліотека доволі проста у використанні, з великою кількістю доступних навчальних матеріалів у вільному доступі. Через популярність, має багато прикладів, документації та форумів для підтримки розробників [10].

2. Face Recognition – бібліотека у відкритому доступі на GitHub, яка спрощує написання коду для розпізнавання облич. Як і інші бібліотеки, використовує відомі методи обробки зображення для ідентифікації вмісту. Легко інтегрується з OpenCV, для спрощення розробки [9].

2. Numpy – фундаментальна бібліотека для роботи з математичними функціями та багатовимірними масивами даних і науковими обчисленнями. Вона використовується для обробки великих обсягів числових даних, які утворюються під час роботи з пікселями зображень або матрицями перетворень [11].

3. Matplotlib – використовується для візуалізації даних, таких як графіки, гістограми, або ж просто для відображення зображень.

4. Dlib — це багатофункціональна бібліотека, орієнтована на машинне навчання та розробку комплексного програмного забезпечення. Є критично важливою для роботи іншої бібліотеки, Face Recognition, оскільки та написана на її основі [12].

5. Glob та OS – використовуються для роботи з операційною системою та з масивами файлів [13, 14].

Для реалізації роботи була обрана мова програмування Python. Це універсальна, динамічна, високоабстрактна мова, яка стала стандартом у багатьох областях, зокрема в машинному навчанні та комп'ютерному зорі.

Зрозумілий синтаксис дозволяє швидко писати та тестувати код, різноманітність бібліотек доступних саме на цю мову, які значно спрощують написання будь-якої роботи. Доступ до численних ресурсів, прикладів, документації та форумів.

Вибір саме цієї мови та бібліотек, зумовлений простотою їх використання, доступністю, та популярністю. На інших мовах та за відсутністю бібліотек, схожий результат, зайняв би критично більше часу. Завдяки цьому, код програми можна написати швидко та легко.

4.3. Програмна реалізація застосунку

Перед написанням коду, необхідно мати Python версії 3.12.x для коректної роботи бібліотеки ряду бібліотек, який можна завантажити з офіційного сайту Python. Dlib необхідна для face_recognition, потребує numpy старішої версії, тому для розв'язання проблеми необхідно відкрити термінал з правами адміністратора й видалити наявну версію та встановити необхідну наступними командами(рис 4.1).

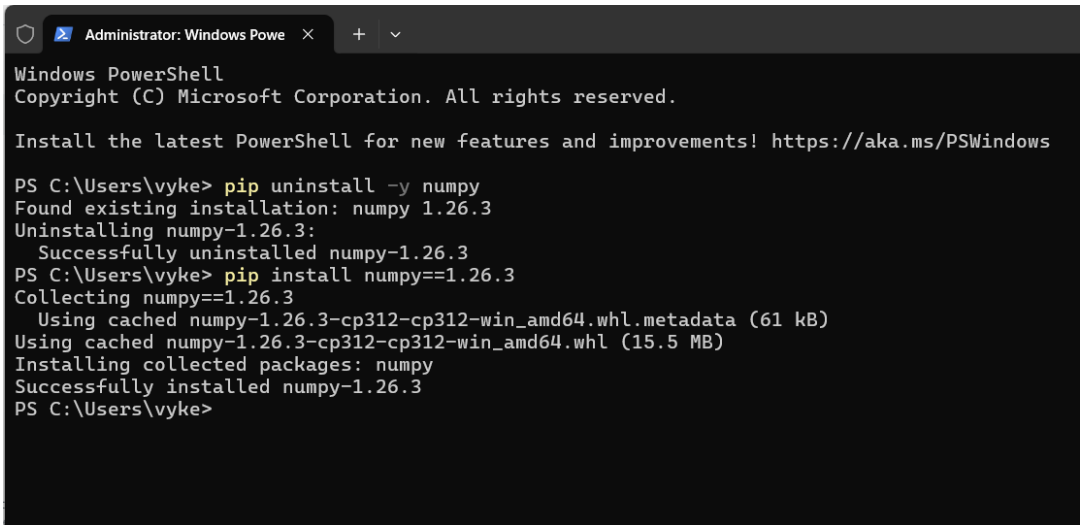

Рисунок 4.1 – Встановлення numpy

Після встановлення numpy, наступним кроком буде встановлення dlib. Для цього потрібно завантажити потрібну версію з GitHub, розпакувати файл на ПК, та в термінали вказати шлях до нього, після чого встановити(рис. 4.2). Також, для коректної роботи цієї бібліотеки, критично необхідно встановити файл shape_predictor_68_face_landmarks.dat, це попередньо навчена модель для бібліотеки dlib, яка використовується для визначення на обличчі 68 ключових точок, таких як: контур обличчя, брови, очі, ніс, рот(рис. 4.3).

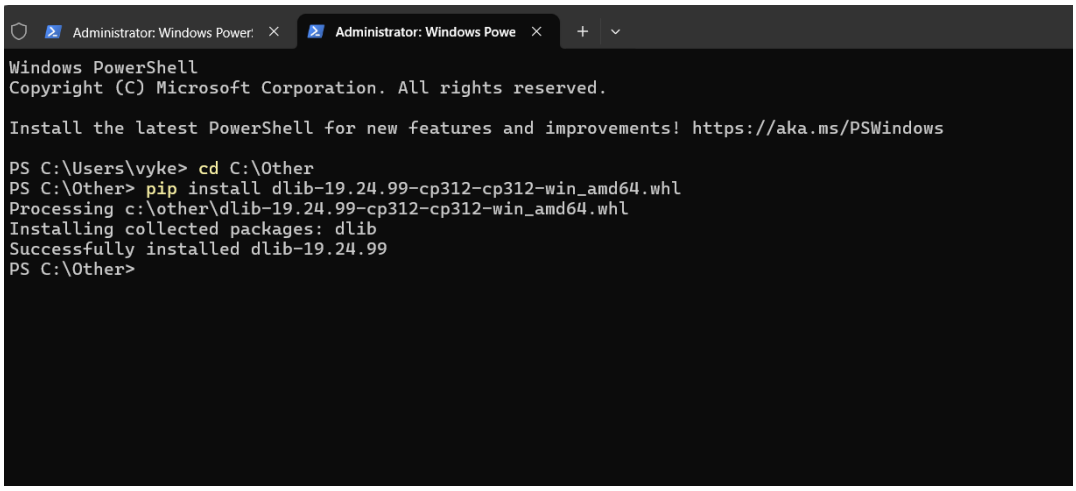

Рисунок 4.2 – Встановлення dlib

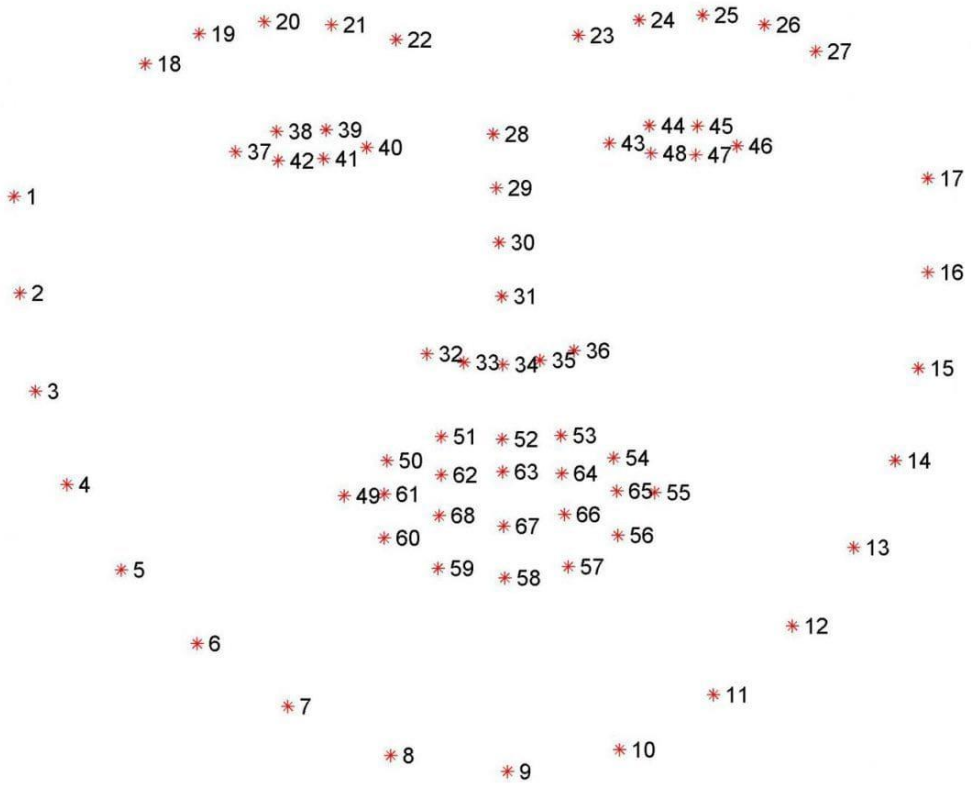

Рисунок 4.3 – Ключові точки на обличчі

На цьому етапі, нічого не заважає встановити бібліотеку face_recognition та OpenCV, що робиться за наступною командою у терміналі(рис. 4.4).

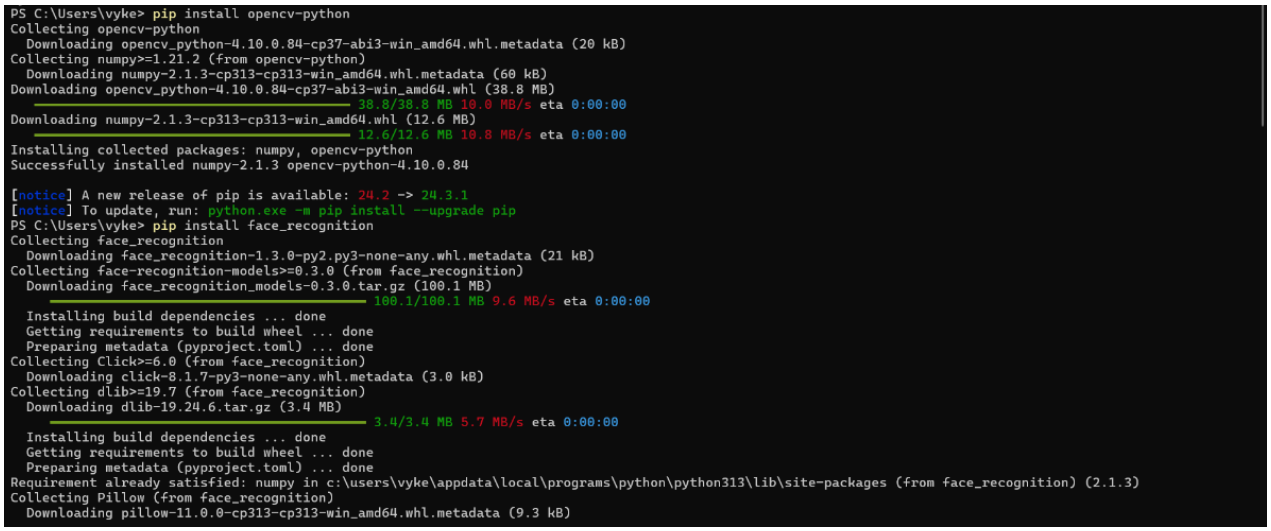

Рисунок 4.4 – Встановлення face_recognition та OpenCV

Після встановлення особливо важливих бібліотек для роботи програми, необхідно встановити середу для компіляції exe-додатків, cx_Freeze. Окрім попередньо зазначеної бібліотеки, також існує більш популярна серед розробників PyInstaller, проте останній не була віддана перевага, через проблему, що під час збирання додатка не вдавалось вмістити до переліку необхідних файлів - shape_predictor_68_face_landmarks.dat, що викликало помилку під час запуску. Для встановлення cx_Freeze необхідно в консолі написати запит, після чого вона автоматично встановиться(рис. 4.5).

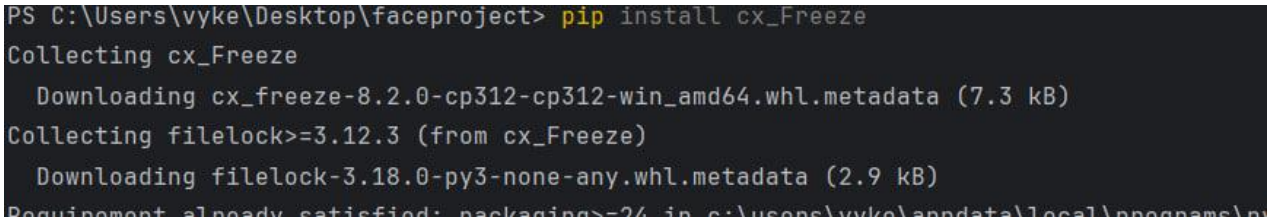

Рисунок 4.5 – Встановлення Cx_Freeze

Бібліотека, яка буде відповідати за візуальну частину застосунку, має назву PyQt. Ця бібліотека має більшу гнучкість і можливостей у порівнянні з Tkinter, хоч і більш вимоглива з огляду на опанування. Встановлюється, за схожою командою у консолі(рис. 4.6).

```
PS C:\Users\vyke\Desktop\Нова папка (2)> pip install PyQt5
Collecting PyQt5
  Downloading PyQt5-5.15.11-cp38-abi3-win_amd64.whl.metadata (2.1 kB)
Collecting PyQt5-sip<13,>=12.15 (from PyQt5)
  Downloading PyQt5_sip-12.17.0-cp312-cp312-win_amd64.whl.metadata (492 bytes)
Collecting PyQt5-Qt5<5.16.0,>=5.15.2 (from PyQt5)
  Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl.metadata (552 bytes)
Downloading PyQt5-5.15.11-cp38-abi3-win_amd64.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 10.6 MB/s eta 0:00:00
Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.1/50.1 MB 11.5 MB/s eta 0:00:00
Downloading PyQt5_sip-12.17.0-cp312-cp312-win_amd64.whl (58 kB)
Installing collected packages: PyQt5-Qt5, PyQt5-sip, PyQt5
Successfully installed PyQt5-5.15.11 PyQt5-Qt5-5.15.2 PyQt5-sip-12.17.0
```

Рисунок 4.6 – Встановлення PyQt

Перед написанням безпосередньо програми, якщо IDE не зробила це автоматично, необхідно обрати в інтерпретатор версію Python, що можна зробити в налаштуваннях та матиме наступний приблизний вигляд(рис. 4.6).

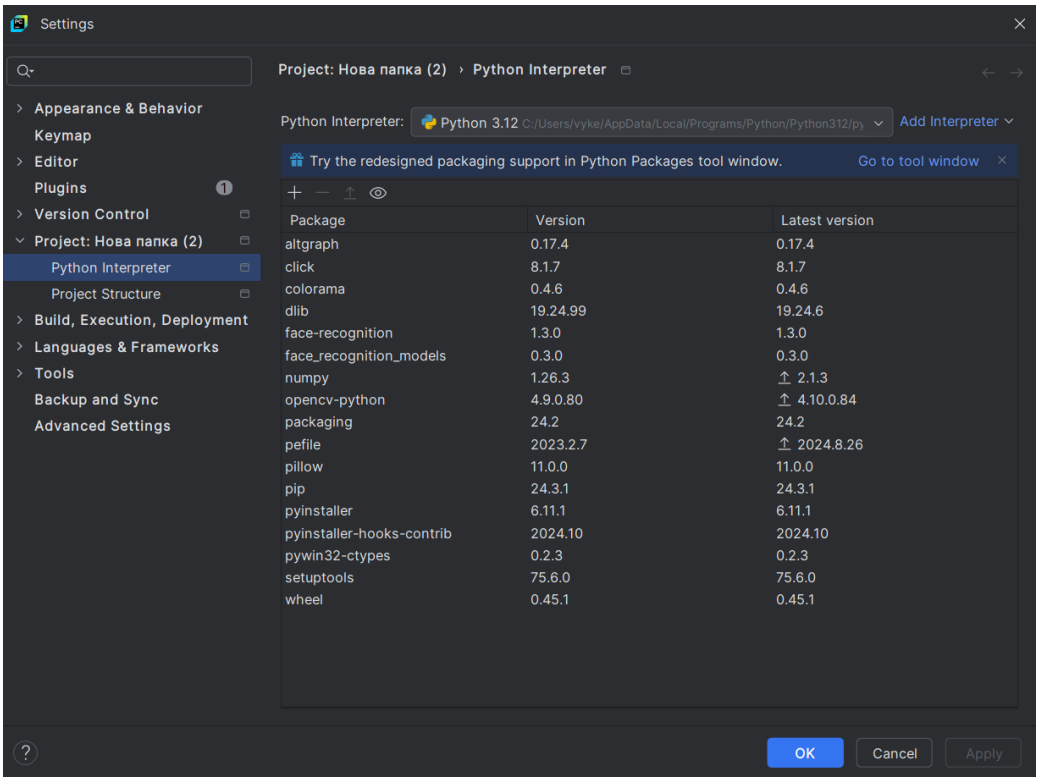

Рисунок 4.6 – Визначення інтерпретатора

Для подальшої розробки застосунку, необхідно імпортувати потрібні бібліотеки, серед яких присутні ті які було встановлено раніше, та вбудовані у Python попередньо. Потрібно імпортувати до середи розробки бібліотеку OS, що дозволяє програмі взаємодіяти з файловою системою комп'ютера. Додатково рекомендується імпортування бібліотеки Glob, яка дозволяє програмі швидко знаходити файли, що відповідають певному типу. Наприклад, знайти зображення у форматі .jpg або .png у заданій директорії. Ще однією важливою бібліотекою є Shutil, яка дозволяє програмі переміщати, копіювати та видаляти файли або цілі папки разом із їхнім вмістом. Ці бібліотеки допоможуть у роботі безпосередньо із фотографіями облич, які будуть використані для розпізнання. Щоб ці фотографії можна було зручно оновити, видалити, додати під час використання застосунку, і необхідні вище зазначені бібліотеки.

Черговим імпортованим елементом є datetime, який забезпечує отримання та обробку поточної дати та часу. Він застосовується для формування міток часу, які використовуються для визначення точного часу, коли було виконано розпізнання.

Окремо варто згадати бібліотеку sys, яка виконує допоміжну роль, але є важливою для організації запуску та завершення роботи програми.

Далі надано зображення, на якому всі необхідні бібліотеки імпортовано(рис. 4.7).

```
import sys                                                                    ⚠10 ⚠19
import cv2
import os
import glob
import numpy as np
import face_recognition
import shutil
from PyQt5.QtWidgets import (QApplication, QWidget, QPushButton, QVBoxLayout, QLabel,
                             QHBoxLayout, QTextEdit, QSplitter, QFileDialog, QListWidget,
                             QMessageBox, QGridLayout, QGroupBox, QDialog, QCheckBox,
                             QComboBox)
from PyQt5.QtGui import QImage, QPixmap, QFont
from PyQt5.QtCore import QTimer, Qt
from datetime import datetime
```

Рисунок 4.7 – Імпортовані бібліотеки

Клас FaceRecognition, в якому знаходиться логіка взаємодії із зображеннями, а також для першої обробки облич. В ньому знаходиться конструктор класу __init__, головне завдання якого — ініціалізувати основні властивості об’єкта та забезпечити правильну структуру файлової системи для подальшої роботи програми, тобто створює списки та перевіряє папку images.

self.known_face_encodings та self.known_face_names створюються два списки, перший для збереження числових векторів (векторів ознак) облич осіб, зображення яких є в базі, а другий створює список, що зберігає імена осіб, чиї вектори ознак вже збережені в попередньому списку

self.frame_resizing встановлює масштаб зменшення зображення, яке буде подаватись на вхід алгоритму розпізнавання. Значення 0.25 означає, що обробка буде здійснюватися на кадрі, зменшеному у 4 рази, що значно зменшує обчислювальні витрати й пришвидшує процес розпізнавання.

self.images_path визначає шлях до директорії, де зберігаються всі зображення. Назва папки є умовною та може змінюватися, однак у рамках цієї програми вона фіксована.

В кінці знаходиться перевірка - це захисна конструкція, яка перевіряє, чи існує папка images/ на момент запуску програми. Якщо вона відсутня, програма автоматично створює її. Що дозволяє уникнути різноманітних помилок при першому запуску програми на новому комп’ютері, або за якоїсь причини вона була видалена(рис 4.8).

```
class FaceRecognition:  1 usage
    def __init__(self):
        self.known_face_encodings = []
        self.known_face_names = []
        self.frame_resizing = 0.25
        self.images_path = "images/"

        if not os.path.exists(self.images_path):
            os.makedirs(self.images_path)
```

Рисунок 4.8 – Конструктор

Метод load_encoding_images(self) відповідає за підготовку бази даних відомих облич, які пізніше використовуватимуться для розпізнавання. Виконується зчитування всіх наявних зображень з певної папки, обробляє їх, виявляє обличчя та створює унікальні числові представлення.

Спочатку, перед тим як завантажити нові зображення, списки очищуються. Це забезпечує оновлення даних при кожному виклику методу, щоб не зберігалися застарілі або повторні значення.

За допомогою модуля glob збираються всі файли з будь-якими розширеннями в папці images/. Це дозволяє обробляти зображення у форматах .jpg, .png, .jpeg тощо. Також виводиться кількість знайдених зображень.

Далі починається цикл, в якому кожне зображення відкривається за допомогою OpenCV. Якщо файл за якоїсь причини не вдалося прочитати, програма виводить повідомлення про помилку та переходить до наступного файлу. Це рішення дозволяє уникнути миттєвого закінчення виконання через один проблемний файл. Після, кожне зображення зчитується та конвертується у формат RGB, оскільки під час зчитування OpenCV потребує BGR, та отримує свою назву, базуючись на назві файлу, позбувшись його формату.

Вбудована функція бібліотеки face_recognition починає знаходити обличчя на зображеннях. Якщо нічого не було виявлено, файл ігнорується. Це

також захист, на випадок коли користувач помилково завантажив не те фото, або застосунок не зможе з ним працювати.

Якщо обличчя знайдено, для нього обчислюється вектор ознак, що чисельно описують форму, риси обличчя тощо. Так, за допомогою бібліотеки, риси обличчя розбивається на масив чисел, та мають унікальні значення, за якими будуть проходити подальші порівняння для ідентифікації. Ці вектори зберігаються до self.known_face_encodings та self.known_face_names, на цей момент, ці обличчя вже відомі й зможуть бути ідентифіковані далі.

В кінці методу, знаходиться обробка помилок, та повідомлення, якщо все пройшло успішно(рис. 4.9).

```
def load_encoding_images(self):  3 usages
    self.known_face_encodings = []
    self.known_face_names = []

    images_path = glob.glob(os.path.join(self.images_path, "*.*"))
    print("{} зображень знайдено.".format(len(images_path)))

    for img_path in images_path:
        try:
            img = cv2.imread(img_path)
            if img is None:
                print(f"Помилка завантаження зображення: {img_path}")
                continue

            rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            basename = os.path.basename(img_path)
            (filename, ext) = os.path.splitext(basename)

            face_locations = face_recognition.face_locations(rgb_img)
            if not face_locations:
                print(f"Обличчя не знайдено на зображенні: {img_path}")
                continue

            img_encoding = face_recognition.face_encodings(rgb_img, face_locations)[0]

            self.known_face_encodings.append(img_encoding)
            self.known_face_names.append(filename)
        except Exception as e:
            print(f"Помилка обробки зображення {img_path}: {e}")

    print("Завантаження завершено")
    return len(self.known_face_names)
```

Рисунок 4.9 – Метод обробки зображень

Метод detect_known_faces приймає отриманий кадр з вебкамери, проводить його обробку та визначає чи присутні на ньому збережені раніше обличчя.

Спочатку зменшується до чверті кадр від початкового розміру, що прискорює процес обробки, та після знову конвертується у RGB формат, після того, як OpenCV отримав його у форматі BGR.

У отриманому кадрі, face_locations знаходить обличчя, після чого face_encodings його кодує до 128 вимірного унікального масиву. Для кожного обличчя, знайденого в кадрі, програма починає порівняння з вже відомими обличчями. Порожній список face_names буде наповнений іменами користувачів або позначками "unknown", якщо відповідного не було знайдено.

Функція compare_faces() порівнює обличчя з усіма, що були збереженими, й повертає список який вказує на наявність збігів. Якщо збігу немає, ім’я за замовчуванням стає - "unknown".

Щоб визначити найкращий варіант з усіх знайдених, виконується обчислення відстані між векторами ознак. Чим менше значення, тим більша ймовірність що обличчя схожі. Далі np.argmin() знаходить індекс найменшої відстані. Якщо за цим індексом значення у matches дорівнює True, то знайдене ім’я користувача записується в змінну name.

Оскільки вхідне зображення кадру з вебкамери користувача було зменшено для пришвидшення обробки, координати знайдених облич тепер масштабуються назад до вихідного розміру. Це необхідно для правильного відображення області поверх оригінального кадру, а не зменшеної копії зображеного кадру. Наостанок, для подальшої візуалізації результатів, дані повертаються(рис. 4.10).

```
def detect_known_faces(self, frame):  1 usage
    small_frame = cv2.resize(frame, dsize: (0, 0), fx=self.frame_resizing, fy=self.frame_resizing)
    rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)
    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

    face_names = []
    for face_encoding in face_encodings:
        matches = face_recognition.compare_faces(self.known_face_encodings, face_encoding)
        name = "unknown"

        if len(self.known_face_encodings) > 0:
            face_distances = face_recognition.face_distance(self.known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = self.known_face_names[best_match_index]
        face_names.append(name)

    face_locations = np.array(face_locations) if face_locations else np.array([])
    if len(face_locations) > 0:
        face_locations = face_locations / self.frame_resizing
    return face_locations.astype(int), face_names
```

Рисунок 4.10 – Метод розпізнання відомих облич

Метод get_image_list(self) повертає список усіх зображень, що зберігаються у раніше визначеній директорії images/. Потрібно передати перелік наявних файлів які присутні, для подальшої роботи.

Метод get_face_names(self) використовується для отримання імен облич, які були закодовані раніше, назв файлів без розширень. Ці імена надалі використовуються як теги або підписи для розпізнаних осіб.

Метод delete_image(self, image_name) відповідає за повне видалення зображення користувача із внутрішньої директорії програми. Цей метод може бути необхідний у випадку, коли користувач більше не хоче зберігати певне обличчя у базі відомих облич, або в разі помилки при додаванні. Щоб уникнути помилок під час видалення небажаного зображення, перевіряється, чи існує такий файл є у системі.

Метод add_image(self, source_path, name) потрібен для безпосереднього додавання нового зображення обличчя від користувача до директорії де зберігаються всі фото. На початку назва файлу автоматично присвоюється на основі назви оригінального файлу. Далі додається розширення, витягнуте з вихідного шляху. Потім створюється цільовий шлях для збереження

зображення в робочій директорії images, після чого виконується копіювання файлу за допомогою функції shutil.copy2.

Вигляд вище розібраних методів у середі розробки на наступному зображені(рис. 4.11).

```
def get_image_list(self):  1 usage
    images = glob.glob(os.path.join(self.images_path, "*.*"))
    return [os.path.basename(img) for img in images]

def get_face_names(self):  1 usage
    images = glob.glob(os.path.join(self.images_path, "*.*"))
    return [os.path.splitext(os.path.basename(img))[0] for img in images]

def delete_image(self, image_name):  1 usage
    image_path = os.path.join(self.images_path, image_name)
    if os.path.exists(image_path):
        os.remove(image_path)
        return True
    return False

def add_image(self, source_path, name):  1 usage
    if not name:
        name = os.path.basename(source_path)

    file_name, file_ext = os.path.splitext(name)
    if not file_ext:
        _, source_ext = os.path.splitext(source_path)
        name = f"{name}{source_ext}"

    target_path = os.path.join(self.images_path, name)

    shutil.copy2(source_path, target_path)
    return name
```

Рисунок 4.11 – Блок методів по роботі із зображеннями

Наступним кроком буде написання нового класу, CheckListDialog, яке відповідає за створення вікна, в якому знаходиться список облич, що потім будуть розпізнаватись. У цьому вікні користувач повинен бачити імена відомих системі осіб у вигляді чек боксів, а також актуальну інформацію щодо точного часу коли була виконана ідентифікація.

Цей клас є спадкоємцем QDialog з бібліотеки PyQt, тобто представляє вікно, яке може бути використане для виведення словнику розпізнаних осіб. Конструктор всередині класу приймає три аргументи: батьківський елемент, список імен та словник із вже розпізнаними обличчями, після чого задаються

базові параметри діалогового вікна: заголовок вікна, початкова позиція і розміри. Наступним, перевіряється чи було передано список розпізнаних облич, якщо ні, створюється порожній словник. Таким чином забезпечується стабільна робота системи навіть у випадку, коли цей параметр не був наданий за якоїсь причини. Наостанок створюється вертикальний макет QVBoxLayout, який використовуватиметься для розміщення усіх елементів у вікні. Далі створюються два словники, один для зберігання чек боксів за іменами, а другий для міток часу, які відображають час розпізнавання кожного обличчя(рис. 4.12).

```
class CheckListDialog(QDialog):  1 usage
    def __init__(self, parent=None, face_names=None, recognized_faces=None):
        super().__init__(parent)
        self.setWindowTitle("Список розпізнавання")
        self.setGeometry(200, 200, 400, 400)
        self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)

        self.recognized_faces = recognized_faces if recognized_faces else {}

        layout = QVBoxLayout()

        self.checkboxes = {}
        self.time_labels = {}
```

Рисунок 4.12 – Новий клас та метод вікна

В частині конструкції face_names, забезпечується створення інтерфейсу для кожної особи, яка передається у список. Перевіряється, чи не порожній список face_names, який містить імена осіб, що були виявлені та збережені під час розпізнавання. Якщо список порожній, то далі код не виконується, і інтерфейс не буде заповнений елементами для розпізнавання. Якщо ж у списку є елементи (імена), тоді виконуються наступні кроки для їх відображення.

Цикл для кожного імені в списку face_names, перебирає кожне ім’я в списку, де кожен елемент є рядком, що містить ім’я особи, для якої було збережено розпізнане обличчя. В межах саме цього циклу будуть створені елементи інтерфейсу. Для кожного імені створюється новий горизонтальний

макет QHBoxLayout(), який буде містити два основні елементи, поле для відображення стану, та позначка відображення точного часу коли ця особа була розпізнана. Також встановлене вирівнювання, для того, щоб елементи були вирівняні по лівому верхньому куту.

Для кожної особи створюється чек бокс QCheckBox, текст на якому відповідає імені цієї особи. Поряд з чек боксом створюється мітка, що спочатку відображає текст "Не розпізнано". Задається шрифт та його розмір за бажанням для обох цих елементів.

Наступним відбувається перевірка на те чи є ім’я особи в словнику. Якщо особа була розпізнана, то чек бокс буде автоматично відмічений і стає недоступним для подальших змін, після чого мітка часу оновлюється, щоб показати точний час, коли ця особа була розпізнана.

Тільки після того, як чек бокс і часова налаштовані, вони додаються до макета. Між елементами додається простір для зручності читання.

Чек бокс додається в словник, що дозволяє йому надалі звертатися до нього за іменем особи(рис. 4.13).

```
if face_names:
    for name in face_names:
        hbox = QHBoxLayout()
        hbox.setAlignment(Qt.AlignLeft | Qt.AlignTop)
        checkbox = QCheckBox(name)
        checkbox.setFont(QFont('Arial', 12))

        time_label = QLabel("Не розпізнано")
        time_label.setFont(QFont('Arial', 10))
        self.time_labels[name] = time_label

        if name in self.recognized_faces:
            checkbox.setChecked(True)
            checkbox.setEnabled(False)
            time_label.setText(f"Розпізнано о {self.recognized_faces[name]}")

        hbox.addWidget(checkbox)
        hbox.addStretch(1)
        hbox.addWidget(time_label)

        self.checkboxes[name] = checkbox
        layout.addLayout(hbox)
```

Рисунок 4.13 – Інтерфейс для списку

Загальне призначення для наступного методу є оновлення інтерфейсу діалогу в реальному часі, коли під час роботи програми було успішно ідентифіковано обличчя певної особи, і відобразити факт розпізнавання прямо в діалоговому вікні без необхідності його перевідкривати, оскільки без нього програма терміново закінчить свою роботу

Спочатку метод гарантує, що дійсно є чек бокс для обличчя з іменем. Якщо такого немає, то подальше оновлення не виконується, що запобігає помилкам. Чек бокс відповідної особи встановлюється як позначений, що вказує на те що особу було розпізнано. Після чого цей чек бокс деактивується, щоб користувач не міг змінити його вручну, що зроблено для того, щоб позначити що ідентифікація відбулась системою автоматично, а не за втручанням користувача.

Наостанок, змінюється текст біля імені користувача, для того, щоб показати точний час, коли його було розпізнано. Це робить інформацію наочнішою та дозволяє вести облік розпізнання безпосередньо в самому інтерфейсі програми(рис. 4.14).

```
def update_recognized_face(self, name, time_str):  1 usage (1 dynamic)
    if name in self.checkboxes:
        self.checkboxes[name].setChecked(True)
        self.checkboxes[name].setEnabled(False)

        self.recognized_faces[name] = time_str

        if name in self.time_labels:
            self.time_labels[name].setText(f"Розпізнано о {time_str}")
```

Рисунок 4.14 – Оновлення списку імен з часовою міткою

Наступний метод, ініціалізує головні змінні програми для розпізнавання облич, визначає доступні камери та запускає побудову графічного інтерфейсу.

Першим в програмі викликається конструктор батьківського класу QWidget, що забезпечує коректну ініціалізацію базового компонента графічного користувацького інтерфейсу. На початку роботи додатку, камера не активна та буде відкриватись лише при старті розпізнавання за окремою кнопкою. Створюється таймер, який кожні 30 мс викликатиме метод update_frame для оновлення зображення з камери. Створюється порожня множина для збереження імен розпізнаних облич у поточному кадрі та логічна змінна, яка вказує, чи активне зараз розпізнавання, а також словник з іменами та часом. Обирається камера яка стоїть за замовчуванням в системі та список всіх доступних камер. Викликається метод для створення графічного інтерфейсу користувача.

Таким чином, конструктор виконує початкову підготовку в створенні об’єктів, які відповідають за камеру, розпізнавання облич, графічний інтерфейс, а також налаштовує таймер і список доступних камер(рис. 4.15).

```
class FaceRecognitionApp(QWidget):  1 usage
    def __init__(self):
        super().__init__()
        self.face_detector = FaceRecognition()
        self.camera = None
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_frame)
        self.detected_faces = set()
        self.recognition_active = False
        self.recognized_faces = {}
        self.checklist_dialog = None
        self.current_camera_index = 0
        self.available_cameras = self.get_available_cameras()

        self.initUI()
```

Рисунок 4.15 – Налашатування та створення елементів

Метод get_available_cameras необхідний для перевірки на те, які камери підключені до системи та повертає список їхніх індексів. Створюється порожній список для збереження індексів знайдених камер, після чого створюється цикл в якому перевіряються порти наявних в системі камер з індексами від 0 до 9. За допомогою бібліотеки OpenCV, намагаємося підключитися до камери з різними індексами. Після чого якщо камера успішно відкрилась, її індекс додається до списку та закривається підключення до камери і метод повертає список індексів доступних камер(рис. 4.16).

```
def get_available_cameras(self):  1 usage
    available_cameras = []
    for i in range(10):
        cap = cv2.VideoCapture(i)
        if cap.isOpened():
            available_cameras.append(i)
            cap.release()
    return available_cameras
```

Рисунок 4.16 – Пошук доступних камер

Метод initUI є доволі об’ємним оскільки створює та налаштовує весь основний графічний інтерфейс головного вікна програми. Він формує такі елементи як кнопки, списки, текстові поля, розміщення тощо.

Встановлюється заголовок вікна та розміри, розміщення макетів, випадаючий список зі знайденими камерами, такі головні кнопки управління як початок чи припинення роботи вебкамери та відкриття списку з обличчями(рис.4.17).

```
def initUI(self):  1 usage
    self.setWindowTitle('Розпізнавання облич')
    self.setGeometry(100, 100, 1280, 720)

    main_layout = QHBoxLayout()

    left_layout = QVBoxLayout()

    control_layout = QHBoxLayout()

    camera_label = QLabel("Камера:")
    self.camera_combo = QComboBox()
    self.populate_camera_list()

    self.start_button = QPushButton('СТАРТ', self)
    self.start_button.setFixedSize(150, 50)
    self.start_button.clicked.connect(self.start_recognition)

    self.stop_button = QPushButton('СТОП', self)
    self.stop_button.setFixedSize(150, 50)
    self.stop_button.clicked.connect(self.stop_recognition)
    self.stop_button.setEnabled(False)

    self.checklist_button = QPushButton('Список', self)
    self.checklist_button.setFixedSize(150, 50)
    self.checklist_button.clicked.connect(self.show_checklist)
```

Рисунок 4.16 – Головне вікно та кнопки зі списками

Далі елемент камери та випадаючий список камер вирівнюються по горизонталі та вирівнюються до лівого краю. Потім три кнопки управління розміщуються поруч у ряд, забезпечуючи зручний доступ до основних функцій керування. Створюється місце, де буде виводитись відео з вебкамери користувача. Задається мінімальний розмір і потрібна рамка. Ці елементи вміщені у ліву частину вікна(рис.4.17).

```
camera_layout = QHBoxLayout()
camera_layout.addWidget(camera_label)
camera_layout.addWidget(self.camera_combo)
camera_layout.addStretch(1)

control_buttons_layout = QHBoxLayout()
control_buttons_layout.addWidget(self.start_button)
control_buttons_layout.addWidget(self.stop_button)
control_buttons_layout.addWidget(self.checklist_button)

self.video_label = QLabel(self)
self.video_label.setAlignment(Qt.AlignCenter)
self.video_label.setMinimumSize(640, 480)
self.video_label.setStyleSheet("border: 1px solid gray;")

left_layout.addLayout(camera_layout)
left_layout.addLayout(control_buttons_layout)
left_layout.addWidget(self.video_label)
```

Рисунок 4.17 – Ліва частина вікна застосунку

Наступним створюється права частина вікна застосунку, де будуть знаходитись лог розпізнан та поле операцій із зображеннями. Створюється секція з QTextEdit, у якій відображається журнал подій з можливістю тільки читання і бажаним шрифтом та його розміром, а після секція с переліком всіх зображень які додані. Додаються такі функціональні кнопки роботи із зображеннями як їх додавання, видалення, та оновлення списку(рис. 4.18).

```
right_layout = QVBoxLayout()

log_box = QGroupBox("Лог розпізнавань")
log_layout = QVBoxLayout()

self.log_text = QTextEdit()
self.log_text.setReadOnly(True)
self.log_text.setStyleSheet("font-family: Arial; font-size: 12px;")

log_layout.addWidget(self.log_text)
log_box.setLayout(log_layout)

image_box = QGroupBox("Управління зображеннями")
image_layout = QVBoxLayout()

self.image_list = QListWidget()
self.image_list.setStyleSheet("font-family: Arial; font-size: 12px;")

image_buttons_layout = QHBoxLayout()

self.add_image_button = QPushButton('Додати зображення')
self.add_image_button.clicked.connect(self.add_image)

self.delete_image_button = QPushButton('Видалити зображення')
self.delete_image_button.clicked.connect(self.delete_image)

self.refresh_button = QPushButton('Оновити')
self.refresh_button.clicked.connect(self.refresh_image_list)
```

Рисунок 4.18 – Лог та кнопки операцій над зображеннями в правій частині

Раніше створені кнопки операцій над зображеннями додаються на вікно. Розташовуються поля з логом та наявних зображень, та встановлюється відношенням яке будуть займати ці елементи.

До поля логу подій, додається інформація щодо точного часу коли програма була запущена, після перевіряється та виводиться статус знайдених камер, та якщо жодної не було знайдено про це також надається повідомлення та кнопка початку розпізнання деактивується. Також оновлюється список всіх доступних для розпізнання зображень(рис. 4.19).

```
image_buttons_layout.addWidget(self.add_image_button)
image_buttons_layout.addWidget(self.delete_image_button)
image_buttons_layout.addWidget(self.refresh_button)

image_layout.addWidget(self.image_list)
image_layout.addLayout(image_buttons_layout)
image_box.setLayout(image_layout)

right_layout.addWidget(log_box)
right_layout.addWidget(image_box)

main_layout.addLayout(left_layout, 7)
main_layout.addLayout(right_layout, 3)

self.setLayout(main_layout)

current_time = datetime.now().strftime("%H:%M:%S")
self.log_text.append(f"<b>[{current_time}]</b> Програму запущено")
if len(self.available_cameras) > 0:
    self.log_text.append(f"<b>[{current_time}]</b> Знайдено {len(self.available_cameras)} камер(и)")
else:
    self.log_text.append(f"<b>[{current_time}]</b> Камери не знайдено")
    self.start_button.setEnabled(False)

self.refresh_image_list()
```

Рисунок 4.19 – Розміщення кнопок та звітування

Наступний метод populate_camera_list, займається заповненням випадаючого списку камер доступними камерами на комп'ютері. Спочатку метод самостійно очищає список камер, на випадок якщо цей метод викликається повторно. Якщо по якійсь причині список доступних камер порожній, то у випадаючому списку з’являється повідомлення "Камери не знайдено", і сам список деактивується. У разі якщо присутні наявні камери, кожну з них додають до випадаючого списку. Одразу після того як всі доступні камери було додано, автоматично встановлюється перший елемент списку як поточний за замовчуванням.

Таким чином, цей метод займається коректним відображенням і взаємодією з випадаючим списком камер, а також оновлює інтерфейс залежно від наявності камер на комп'ютері(рис 4.20).

```
def populate_camera_list(self):  1 usage
    self.camera_combo.clear()
    if not self.available_cameras:
        self.camera_combo.addItem("Камери не знайдено")
        self.camera_combo.setEnabled(False)
    else:
        for i, cam_index in enumerate(self.available_cameras):
            self.camera_combo.addItem(f"Камера {cam_index}")
        self.camera_combo.setCurrentIndex(0)
```

Рисунок 4.20 – Оновлення списку камер

Метод show_checklist отримує імена облич, викликавши метод з об'єкта, який в свою чергу повертає список імен облич, які зберігаються для розпізнавання. Після, перевіряється чи вже відкрите вікно для перегляду списку, якщо так, то воно закривається. Далі створюється новий об'єкт CheckListDialog, який є діалоговим вікном для перегляду розпізнаних осіб. В кінці викликається для показу цього діалогового вікна на екрані.

Метод забезпечує відкриття або оновлення вікна зі списком осіб, що було виявлено і розпізнано, щоб користувач міг переглядати та взаємодіяти з цим списком(рис. 4.21).

```
def show_checklist(self):  1 usage
    face_names = self.face_detector.get_face_names()

    if self.checklist_dialog and self.checklist_dialog.isVisible():
        self.checklist_dialog.close()

    self.checklist_dialog = CheckListDialog(self, face_names, self.recognized_faces)
    self.checklist_dialog.show()
```

Рисунок 4.21 – Вікно списка осіб

Метод refresh_image_list оновлює список зображень, відображених в інтерфейсі, і в подальшому виводить це оновлення у лог для зручності

користувача. Першим кроком очищається список зображень, це необхідно для того, щоб перед додаванням нових елементів у списку не було старих даних. Далі викликається, та отримується список зображень, що були завантажені або доступні для розпізнавання. Цей список зображень зберігається в змінній images. Ці зображення в подальшому додаються в список на графічному інтерфейсі. Таким чином, відображаються всі доступні зображення, які програма може обробляти. Потім цей метод записує в лог повідомлення про те, скільки зображень було знайдено, також використовуючи актуальну часову мітку(рис. 4.22).

```
def refresh_image_list(self):  4 usages
    self.image_list.clear()
    images = self.face_detector.get_image_list()
    self.image_list.addItems(images)

    current_time = datetime.now().strftime("%H:%M:%S")
    self.log_text.append(f"<b>[{current_time}]</b> Знайдено {len(images)} зображень")
```

Рисунок 4.22 – Оновлення списку зображень та звітування

Метод add_image, створює об'єкт діалогового вікна, який надає користувачу можливість вибрати самостійно файл. Виклик відкриває вікно для вибору файлу з системи. Після, відбувається перевірка на те, чи був вибраний файл, у разі якщо користувач не вибрав файл, то значення буде порожнім, і нічого не відбудеться. Якщо файл все-таки було обрано, проходить обробка додавання зображення. Як зображення було додано, викликається метод, який відповідає за оновлення списку зображень на інтерфейсі користувача. Цей метод очищає список зображень та додає нові елементи, щоб відображати всі зображення, включаючи те, яке було щойно додано.

Якщо режим розпізнавання був активний, то викликається метод, щоб завантажити всі закодовані зображення, які будуть використовуватися для порівняння під час розпізнавання облич. Це потрібно для того, щоб після додавання нового зображення програма могла оновити дані для розпізнавання.

Також до логу додається новий запис, який в свою чергу вказує час додавання зображення і назву доданого зображення. Якщо при додаванні зображення виникає помилка, також виводиться повідомлення про це(рис. 4.23).

```
def add_image(self):  1 usage
    file_dialog = QFileDialog()
    file_path, _ = file_dialog.getOpenFileName(
        self, "Виберіть зображення", "",
        "Зображення (*.jpg *.jpeg *.png *.bmp)"
    )

    if file_path:
        try:
            name = self.face_detector.add_image(file_path, name: "")

            self.refresh_image_list()

            if self.recognition_active:
                self.face_detector.load_encoding_images()

            current_time = datetime.now().strftime("%H:%M:%S")
            self.log_text.append(f"<b>[{current_time}]</b> Додано зображення: {name}")

        except Exception as e:
            QMessageBox.critical(self, "Помилка", f"Не вдалося додати зображення: {e}")
```

Рисунок 4.23 – Вікно додавання зображення

Наступний метод delete_image, відповідає за видалення зображень з програми. У користувача запитується на підтвердження видалення, обробляє сам процес видалення, оновлює інтерфейс, синхронізує дані про розпізнані обличчя та веде журнал про виконані дії.

На початку отримується список елементів, які були вибрані користувачем у списку зображень. Якщо жодне зображення не було обрано, програма відображає повідомлення для користувача, яке сповіщає його, що необхідно вибрати зображення для видалення. У випадку, коли зображення вибрано, береться елемент і отримується його текстове значення. Для користувача показується текстове вікно діалогу із підтвердженням його дії, після чого на вибір дається вибір з так/ні. Якщо ім’я зображення є в словнику,

то цей запис видаляється. Це необхідно зробити для того, щоб синхронізувати стан списку розпізнаних облич із змінами, які вже відбулися після видалення зображення. Після, відбувається оновлення відображеного списку зображень в інтерфейсі. Якщо розпізнавання було активно, викликається метод для завантаження всіх закодованих зображень наново. Також у логу відбувається звітування, що містить інформацію про час видалення зображення та його ім’я. На випадок, якщо зображення не вдалося видалити, показується діалогове вікно з повідомленням про помилку(рис. 4.24).

```
def delete_image(self):  1 usage
    selected_items = self.image_list.selectedItems()

    if not selected_items:
        QMessageBox.information(self, "Інформація", "Виберіть зображення для видалення")
        return

    selected_image = selected_items[0].text()

    reply = QMessageBox.question(
        self, "Підтвердження",
        f"Ви впевнені, що хочете видалити зображення {selected_image}?",
        QMessageBox.Yes | QMessageBox.No, QMessageBox.No
    )

    if reply == QMessageBox.Yes:
        if self.face_detector.delete_image(selected_image):
            name_without_ext = os.path.splitext(selected_image)[0]

            if name_without_ext in self.recognized_faces:
                del self.recognized_faces[name_without_ext]

            self.refresh_image_list()

            if self.recognition_active:
                self.face_detector.load_encoding_images()

            current_time = datetime.now().strftime("%H:%M:%S")
            self.log_text.append(f"<b>[{current_time}]</b> Видалено зображення: {selected_image}")
        else:
            QMessageBox.critical(self, "Помилка", "Не вдалося видалити зображення")
```

Рисунок 4.24 – Коректне видалення зображення

Метод start_recognition відповідає за ініціалізацію та запуск процесу розпізнавання облич з камери. Перевіряючи наявність закодованих зображень

для порівняння, вибір камери та її відкриття, оновлює інтерфейс, щоб вказати на активний стан розпізнавання.

Спочатку викликається та дізнається кількість успішно завантажених та готових для порівняння зображення, але якщо не було завантажено жодного зображення, відображається інформаційне повідомлення, яке сповіщає користувача, що наразі немає доступних зображень для розпізнавання.

Визначається та перевіряється наявність обраної користувачем вебкамери. Якщо індекс вибраної камери є некоректним, з'являється повідомлення, яке сповіщає користувача про проблему. Камера відкривається за допомогою бібліотеки OpenCV, і якщо камера успішно відкрилась, програма починає отримувати кадри. Якщо камера не змогла відкритися відображається відповідне повідомлення з інформацією для користувача.

Під час коректної роботи відеопотоку, кнопка "СТАРТ" стає неактивною, а кнопка "СТОП" навпаки активною, що дає користувачеві можливість зупинити процес розпізнавання. Також перестає працювати список із камерами. Відображається запис в лог з поточним часом, що розпізнавання запущено, і вказується яка саме камера буде використовуватись для розпізнавання(рис. 4.25).

```
def start_recognition(self):  1 usage
    count = self.face_detector.load_encoding_images()

    if count == 0:
        QMessageBox.information(self, "Інформація",
                                "Немає зображень для розпізнавання. Додайте зображення спочатку.")
        return

    selected_index = self.camera_combo.currentIndex()

    if selected_index < 0 or selected_index >= len(self.available_cameras):
        QMessageBox.critical(self, "Помилка", "Виберіть камеру з доступних")
        return

    camera_index = self.available_cameras[selected_index]

    self.camera = cv2.VideoCapture(camera_index)
    if not self.camera.isOpened():
        QMessageBox.critical(self, "Помилка", f"Неможливо відкрити камеру {camera_index}")
        return

    self.start_button.setEnabled(False)
    self.stop_button.setEnabled(True)
    self.camera_combo.setEnabled(False)
    self.recognition_active = True

    current_time = datetime.now().strftime("%H:%M:%S")
    self.log_text.append(f"<b>[{current_time}]</b> Розпізнавання запущено (Камера {camera_index})")

    self.timer.start(30)
```

Рисунок 4.25 – Обрання камери та запуск процесу розпізнання

Метод stop_recognition зупиняє процес розпізнавання облич, звільняючи камеру, оновлюючи інтерфейс для наступного запуску та очищаючи відображення відео.

Якщо камера була відкритою, вона звільняється та припиняє використання і позначає це. Таймер, який використовується для регулярного оновлення кадрів, зупиняється, тим самим також зупинивши обробку відеопотоку. Кнопка "СТАРТ" стає активною, дозволяючи користувачу запустити розпізнавання знову, коли кнопка "СТОП" стає неактивною. Також активний стає випадаючий список камер. На екрані очищується те що відображає відеопотік, оскільки розпізнавання зупинене і більше не надходитиме нових кадрів. Очищається набір, що містить імена виявлених облич. Це робиться для того, щоб при наступному запуску розпізнавання не

залишалося попередніх даних. Також до логу звітується те, що розпізнавання було зупинене, разом з часом коли це відбулось(рис. 4.26).

```
def stop_recognition(self):  2 usages
    if self.camera is not None:
        self.camera.release()
        self.camera = None
    self.timer.stop()

    self.start_button.setEnabled(True)
    self.stop_button.setEnabled(False)
    self.camera_combo.setEnabled(True)
    self.recognition_active = False

    self.video_label.clear()

    self.detected_faces = set()

    current_time = datetime.now().strftime("%H:%M:%S")
    self.log_text.append(f"<b>[{current_time}]</b> Розпізнавання зупинено")
```

Рисунок 4.26 – Зупинка процесу розпізнання

Метод update_frame відповідає за обробку кожного отриманого кадру з відеопотоку. Він включає виявлення та розпізнавання облич, малювання відповідних позначок на кадрі, а також відображення результату на візуальному інтерфейсі.

Починається з перевірки двох умов, за випадку тільки якщо обидві умови працюють. Наявність камери та активність розпізнання.

Якщо все добре, то спочатку відбувається спроба зчитати кадр із камери, у разі помилки виводиться відповідне повідомлення і зупиняється робота, якщо проблеми відсутні, то поточний кадр передається для обробки та повертає список координат облич на кадрі, та список імен розпізнаних осіб, що відповідають знайденим обличчям.

Створюється порожній набір, який буде використовуватися для зберігання імен облич, що були розпізнані на поточному кадрі. Починається цикл який перебирає кожну пару з координат обличчя та його імені. Після чого

додається текст з ім'ям ідентифікованої людини на, та малює прямокутник навколо обличчя, та додається до набору поточних облич.

Цикл перевіряє, чи не було нового обличчя вже виявлено раніше, якщо ім'я обличчя відсутнє в наборі знайдених облич, тоді це повідомлення записується в лог про нове розпізнане обличчя. Якщо ім'я обличчя не є "невідомим" і воно ще не було зареєстровано, це ім'я додається до словника разом із поточним часом розпізнавання. Якщо вікно зі списком розпізнаних облич відкрите викликається метод для оновлення часу розпізнавання в цьому вікні.

Змінна виявленого обличчя оновлюється набором поточного обличчя, щоб зберігати список облич, розпізнаних на поточному кадрі. Це дозволяє уникнути повторного виведення одних і тих самих осіб.

Кадр спочатку перетворюється з формату BGR, після того, як його використав OpenCV, на RGB, визначаються розміри зображення, створюється об'єкт QImage з даних кадру для подальшого використання і відображенні в PyQt, QPixmap створює та отримує від QImage дані, і відображає відео у вікні програми(рис. 4.27).

```
def update_frame(self):  1 usage
    if not self.camera or not self.recognition_active:
        return

    ret, frame = self.camera.read()
    if not ret:
        self.log_text.append("<b>Помилка отримання кадру</b>")
        self.stop_recognition()
        return

    face_locations, face_names = self.face_detector.detect_known_faces(frame)

    current_faces = set()

    for face_loc, name in zip(face_locations, face_names):
        y1, x2, y2, x1 = face_loc[0], face_loc[1], face_loc[2], face_loc[3]

        cv2.putText(frame, name, org: (x1, y1 - 10), cv2.FONT_ITALIC, fontScale: 1, color: (0, 0, 200), thickn
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 200), 4)

        current_faces.add(name)

    for face in current_faces:
        if face not in self.detected_faces:
            current_time = datetime.now().strftime("%H:%M:%S")
            self.log_text.append(f"<b>[{current_time}]</b> Розпізнано: <b>{face}</b>")
            self.log_text.verticalScrollBar().setValue(
                self.log_text.verticalScrollBar().maximum()
            )

            if face != "unknown" and face not in self.recognized_faces:
                self.recognized_faces[face] = current_time

                if self.checklist_dialog and self.checklist_dialog.isVisible():
                    self.checklist_dialog.update_recognized_face(face, current_time)

    self.detected_faces = current_faces

    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    h, w, ch = rgb_image.shape
    bytes_per_line = ch * w
    qt_image = QImage(rgb_image.data, w, h, bytes_per_line, QImage.Format_RGB888)
    pixmap = QPixmap.fromImage(qt_image)

    self.video_label.setPixmap(pixmap.scaled(self.video_label.width(), self.video_label.height(),
                                             Qt.KeepAspectRatio))
```

Рисунок 4.27 – Обробка кадрів

Метод closeEvent потрібен для того, щоб перед закриттям програми всі ресурси, пов'язані з камерою, таймером і діалогами, були коректно звільнені.

Перевіряється наявність відкритої камери, і якщо камера була відкрита, то метод release() викликається, щоб правильно закрити доступ до камери.

Виклик stop() зупиняє таймер. Якщо вікно списку існує і є видимим, то метод close() викликається для його закриття перед тим, як програма повністю завершиться. Виклик event.accept() дозволяє завершити подію закриття вікна, що фактично закриває саму програму(рис. 4.28).

```
def closeEvent(self, event):
    if self.camera is not None:
        self.camera.release()
    self.timer.stop()

    if self.checklist_dialog and self.checklist_dialog.isVisible():
        self.checklist_dialog.close()

    event.accept()
```

Рисунок 4.28 – Закриття програми

Остання частина коду, є способом запуску програми. Першим виступає умовний блок, який гарантує, що код у ньому буде виконуватись тільки в тому випадку, якщо цей скрипт запускається безпосередньо, а не імпортується. Створюється екземпляр програми PyQt, створюється саме вікно програми і показується, викликається цикл подій для реагування на події та коректне завершення роботи(рис. 4.29).

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = FaceRecognitionApp()
    window.show()
    sys.exit(app.exec_())
```

Рисунок 4.29 – Запуск програми

На цей момент, написання повного коду програми завершено, ця система може використовуватись у різних цілях за бажанням. Проте, для

спрощення розповсюдження цієї програми, зручніше буде скомпілювати її у самостійний виконуваний exe-файл, без необхідності встановлення середи розробки та чисельних бібліотек для нового користувача. Для опанування цієї мети, буде використовуватись раніше зазначена бібліотека cx_Freeze, яка дозволить стиснути цю програму до розміру папки, в якій будуть зберігатись необхідні файли та присутній exe-файл який буде спроможній запускати систему розпізнання.

Для того, щоб спрацювала cx_Freeze, необхідно в директорії роботи створити новий .py файл, із назвою setup, в якій будуть записані всі необхідні налаштування для успішного білду програми(рис. 4.30).

Рисунок 4.30 – Скрипт файл

У словнику build_exe_options вказано, які пакети (os, sys, face_recognition) та зовнішні файли потрібно включити у збірку. Параметр optimize=2 означає, що Python-код буде зкомпільовано з максимальною оптимізацією, видаляючи коментарі та оператори assert. Далі, за допомогою об'єкта Executable описується конфігурація майбутнього .exe файлу, основний скрипт - main.py, тип застосунку - Win32GUI, а ім'я виконуваного файлу - face_recognition.exe. Функція setup() використовує наявну інформацію для створення роботи(рис. 4.31).

```python
from cx_Freeze import setup, Executable

build_exe_options = {
    "packages": ["os", "sys", "face_recognition"],
    "optimize": 2,
}

exe = Executable(
    script="main.py",
    base="Win32GUI",
    target_name="face_recognition.exe",
)

setup(
    name="FR",
    version="1.0",
    description="Face Rec",
    options={"build_exe": build_exe_options},
    executables=[exe],
)
```

Рисунок 4.31 – Вміст setup.py

Після введення однієї команди у консоль і очікування, результатом є повністю готовий до запуску .exe файл, що працює без встановленого Python, включаючи всі залежності(рис. 4.32).

```
PS C:\Users\vyke\Desktop\faceproject> python setup.py build
running build
running build_exe
creating directory C:\Users\vyke\Desktop\faceproject\build\exe.win-amd64-3.12
copying C:\Users\vyke\AppData\Local\Programs\Python\Python312\Lib\site-packages\cx_Freeze\b
n-amd64-3.12\face_recognition.exe
copying C:\Users\vyke\AppData\Local\Programs\Python\Python312\python312.dll -> C:\Users\vyk
copying C:\Users\vyke\AppData\Local\Programs\Python\Python312\Lib\site-packages\cx_Freeze\i
win-amd64-3.12\frozen_application_license.txt
Stamped: C:\Users\vyke\Desktop\faceproject\build\exe.win-amd64-3.12\face_recognition.exe
creating directory C:\Users\vyke\Desktop\faceproject\build\exe.win-amd64-3.12\lib
copying data from package PIL...
creating directory C:\Users\vyke\Desktop\faceproject\build\exe.win-amd64-3.12\lib\PIL
copying C:\Users\vyke\AppData\Local\Programs\Python\Python312\Lib\site-packages\PIL\py.type
```

Рисунок 4.32 – Збір додатку

У директорії роботи, в папці build, з’являється папка з готовою до запуску або розповсюдженню програма. Після запуску face_recognition.exe в перший раз, автоматично створиться папка із зображеннями, як було написано в коді раніше(рис. 4.33).

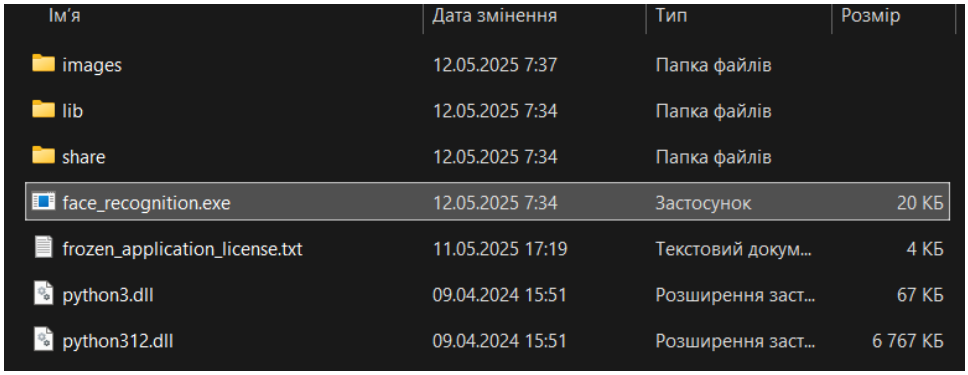

Рисунок 4.32 – Папка із зібраним додатком

Перейдемо до огляду який програми на свій фінальний білд.

В правій частині вікна маємо "Лог розпізнавань", в якому під час першого запуску виведена інформація про те що програма була запущена, знайдена певна кількість камер, та кількість знайдених зображень. Нижче, логу знаходиться "Управління зображеннями", в якому можна додати, видалити, або ж оновити список із зображеннями які потрібні для роботи додатка. В лівій частині вікна присутній випадаючий список із доступними на системі користувача камерами, та далі перелік кнопок "СТАРТ", "СТОП" необхідних для запуску та припинення роботи додатка, та "Список", який відкриває чек ліст для наявних зображень. Нижче кнопок знаходиться поле в якому буде знаходитись відеопотік отриманий з вебкамери(рис. 4.33).

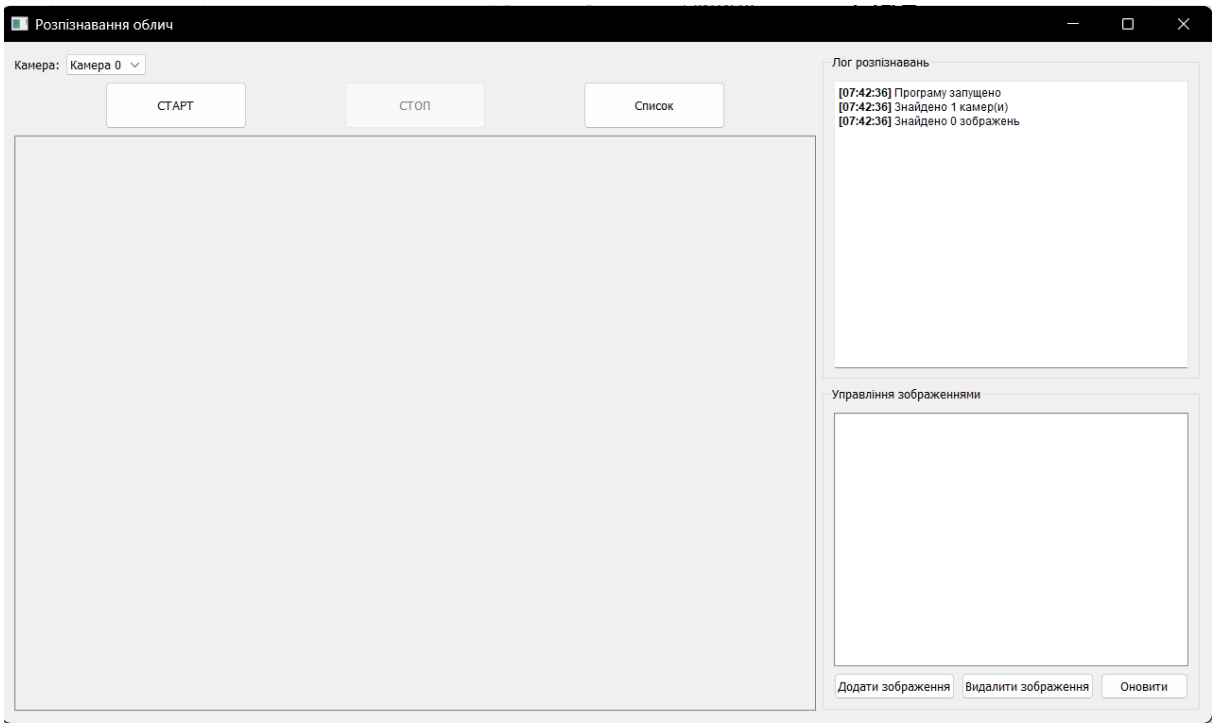

Рисунок 4.33 – Вигляд додатку під час першого запуску

Для перевірки коректної роботи додатку, додамо зображення людини використовуючи кнопку в блоці управління зображеннями. Після натискання, з’являється вікно вибору зображення із системи. Для прикладу, було обрано британського актора Кіта Герінґтона(рис. 4.34). Після успішного додавання, у логу надходить повідомлення, що зображення додано(рис. 4.35).

Рисунок 4.34 – Зображення актора

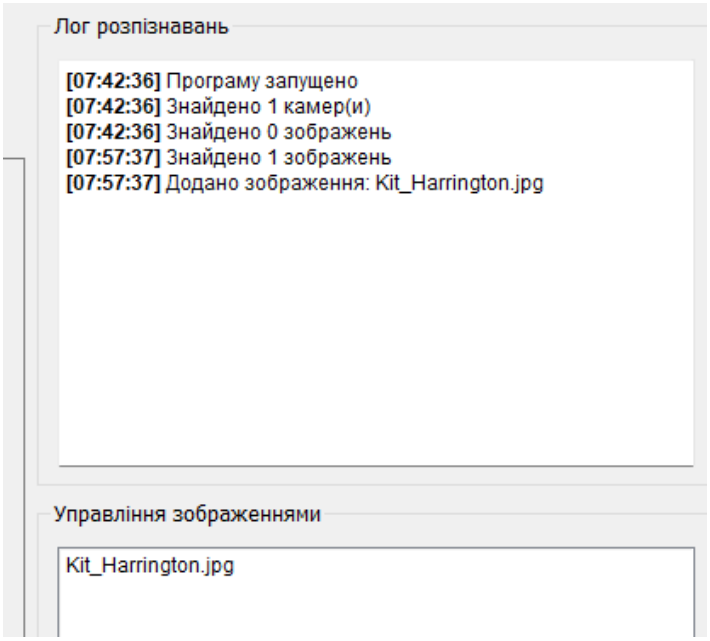

Рисунок 4.35 – Успішно додано зображення

Після вибору потрібної камери й натискання на кнопку "СТАРТ", у головному вікні запускається відеопотік в якому відбувається накладання на кадр з обличчям і з’являється нове повідомлення з інформацією про те що розпізнавання почалось, та з якої камери. Для зручності тестування, пропонується також відкрити "Список" для перевірки його також.

При наведені обличчя актора до камери, система ідентифікує та сповіщає про це. До лог надходить повідомлення про успішне розпізнання з точним часом, також це фіксується у списку. Можна зупинити додаток натиснувши на кнопку "СТОП"(рис. 4.36).

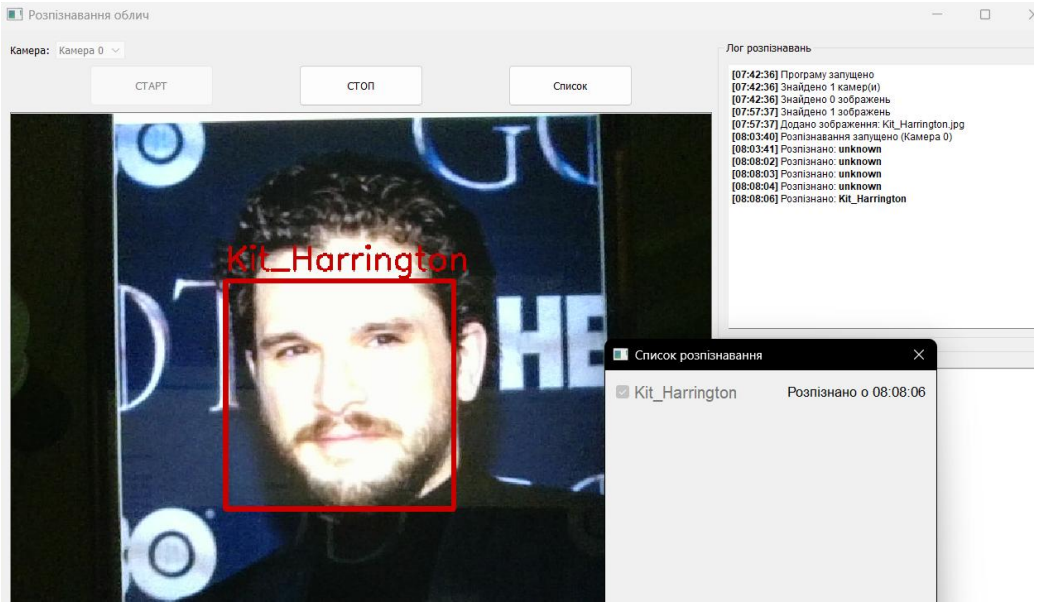

Рисунок 4.36 – Ідентифікація обличчя

При спробі видалення зображення актора, з’являється попередження з необхідністю підтвердити операцію(рис. 4.37). Після підтвердження дії, зображення зникає з переліку та надходить повідомлення в логу(рис 4.38).

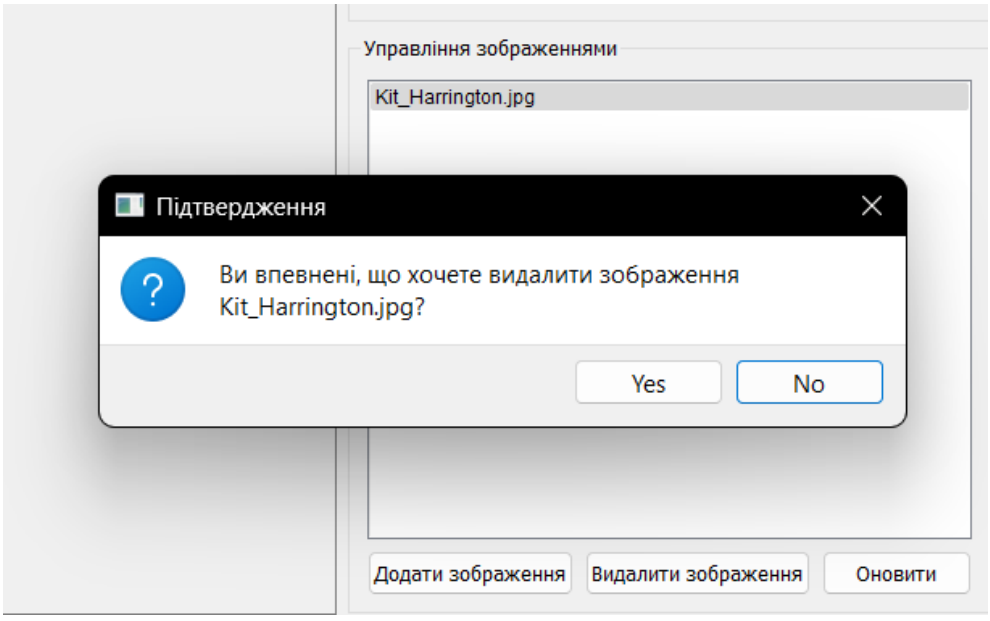

Рисунок 4.37 – Перевірка видалення зображення

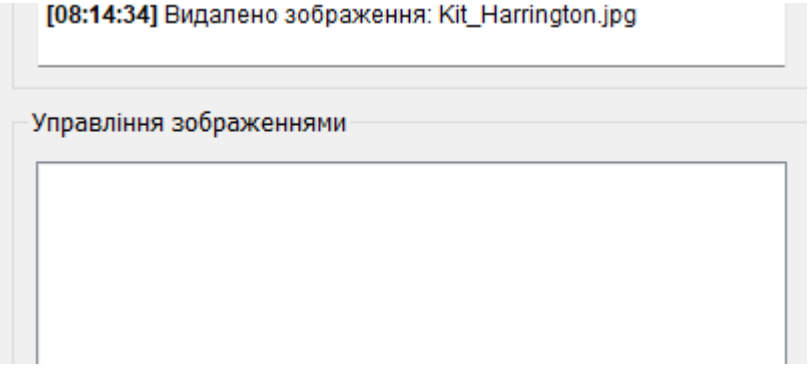

Рисунок 4.38 – Наслідки видалення

Таким чином, застосунок було перевірено на справність роботи її функцій і визнано задовільною для використання. Можливості використання застосунку, доволі значні. Наприклад як інструмент для перевірки відвідуваності студентів на заняттях, коли встановлюється камера на вході до аудиторії для фіксації який студент та о котрій годині був помічений у приміщенні, або як допоміжний інструмент для правоохоронних сил, які можуть завантажити обличчя людей які перебувають у розшуку, чи використовувати замість ключ-пропусків на виробництвах, якщо завантажити зображення співробітників на пропускному пункті.

В своєму вигляді застосунок має широкі можливості, та для певних задач може потребувати певних покращень.

ВИСНОВКИ

У процесі виконання даної кваліфікаційної роботи було досягнуто мети – розробка застосунку для розпізнавання облич з використанням мови програмування Python та суміжних бібліотек. Під час роботи над застосунком було ознайомлено та опановано базовий функціонал ряду бібліотек на мові Python. За час розробки, для знайомства з критично необхідними інструментами розробки, було використано відкриті джерела, такі як: відеоматеріали від розробників та досвідчених користувачів, статті, наукові роботи тощо. Також, було оглянуто сучасні системи розпізнання, їх принципи роботи, можливості імплементації у сучасне життя, переваги та недоліки, в чому вони відрізняються від схожих систем, можливі та фактичні способи використання.

У цій кваліфікаційній роботі було створено та показано, самостійний додаток з розпізнавання облич, який в свою чергу може використовуватись в багатьох напрямках, та за необхідних вимог, модифікуватись розробниками для виконання більш вузьких задач за бажанням користувача.

Розроблений застосунок демонструє легкість в опануванні та перспективи сучасних джерел які можуть бути використані для створення самостійних додатків та розвитку навичок.

За матеріалом роботи, опубліковано тези до XLVIII Міжнародної наукової студентської конференції «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті»

СПИСОК ЛІТЕРАТУРИ

1. Sefik Ilkin Serengil. Introduction to Face Recognition in Deep Learning. 2020. URL: https://sefiks.com/2020/05/01/a-gentle-introduction-to-face-recognition-in-deep-learning/.

2. Kortli, Yassin, Maher Jridi, Ayman Al Falou, and Mohamed Atri. Face Recognition Systems: A Survey. 2020. Sensors 20, no. 2: 342. URL: https://doi.org/10.3390/s20020342.

3. Ben Lutkevich. Pros and cons of facial recognition. 2024. URL: https://www.techtarget.com/whatis/feature/Pros-and-cons-of-facial-recognition.

4. David Gargaro, Rene Millman. The pros and cons of facial recognition technology. 2024. URL: https://www.itpro.com/security/privacy/356882/the-pros-and-cons-of-facial-recognition-technology.

5. Xue Lv, Mingxia Su, Zekun Wang. Face Recognition Method under Adaptive Image Matching and Dictionary Learning Algorithm. Computational Intelligence and Neuroscience. 2023. 1-8. URL: https://doi.org/10.1155/2023/8225630.

6. Guo Jiahui, Ma Feilong, Matteo Visconti di Oleggio Castello, Samuel A. Nastase, James V. Haxby, M. Ida Gobbini. Modeling naturalistic face processing in humans with deep convolutional neural networks. Proceedings of the National Academy of Sciences. 2023. Vol. 120. № 43. URL: https://www.pnas.org/doi/10.1073/pnas.2304085120.

7. Katharina Dobs, Joanne Yuan, Julio Martinez, Nancy Kanwisher. Behavioral signatures of face perception emerge in deep neural networks optimized for face recognition. Proceedings of the National Academy of Sciences. 2023. Vol. 120. № 32. URL: https://www.pnas.org/doi/10.1073/pnas.2220642120.

8. Mrinal Tyagi. Histogram of Oriented Gradients: An Overview. 2024. URL: https://builtin.com/articles/histogram-of-oriented-gradients.

9. Face Recognition – Режим доступу: https://face-recognition.readthedocs.io/en/latest/face_recognition.html

10. OpenCV – Режим доступу: https://docs.opencv.org/4.x/index.html

11. NumPy – Режим доступу: https://numpy.org/doc/2.2/reference/module_structure.html

12. Dlib – Режим доступу: https://dlib.net/python/

13. Glob – Режим доступу: https://docs.python.org/uk/3.9/library/glob.html

14. Os – Режим доступу: https://docs.python.org/uk/3.13/library/os.html