МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри
_____ Ігор КОПЛИК
(підпис)
____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

зі спеціальності 113 Прикладна математика,
освітньо-професійної програми Наука про дані та моделювання складних систем
на тему: «Система розпізнавання обличчя, що ґрунтується на використанні нейронних мереж»

Здобувача групи ПМ.м-21 Жигамовського Нікіти Анатолійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Нікіта ЖИГАМОВСЬКИЙ
(підпис)

Керівник канд. фіз.- мат. наук, доцент, Уляна ШВЕЦЬ _____
(підпис)

Суми – 2023

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**
Кафедра **прикладної математики та моделювання складних систем**
Рівень вищої освіти другий (магістр)

Галузь знань **11 Математика та статистика**
Спеціальність **113 Прикладна математика**
Освітня програма **освітньо-професійна «**Наука про дані та моделювання складних систем**»**

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМтаМСС
Ігор КОПЛИК_____
«___»_____2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Жигамовський Нікіта Анатолійович

1. Тема роботи Система розпізнавання обличчя, що ґрунтується на використанні нейронних мереж
Керівник роботи кандидат фізико-математичних наук, доцент, Швець Уляна Станіславівна
затверджую наказом по факультету ЕлІТ від «07» листопада 2023 р. № 1237-VI

2. Термін подання роботи студентом «16» грудня 2023 р.

3. Вихідні данні до роботи: навчальна та тестова вибірки бази даних зображень різних людей, що містять обличчя

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): літературний огляд алгоритмів виявлення та розпізнавання облич, створення моделі розпізнавання облич, яка здатна розпізнати обличчя людини

5. Перелік графічного матеріалу

1) Компоненти нейрона.

2) Архітектура багатошарової нейронної мережі.

3) Демонстрація роботи алгоритму Віола-Джонс.

4) Блок-схема алгоритму власних граней.

5) Приклад ознак Хаара.

6) Оригінальне чорно-біле зображення.

7) Застосування ознак Хаара.

8) Застосування каскадних класифікаторів ознак Хаара.

9) Розділення обличчя на клітинки 7×7 однакового розміру.

10) Результат обчислення LBP для кожної клітинки.

11) Блок-схема функції додавання обличчя у словник.

12) Блок-схема функції пошуку обличчя у словнику.

13) Блок-схема модуля сканування нового користувача.

14) Блок-схема модуля тренування моделі.

15) Блок-схема модуля розпізнавання обличчя.

16) Порівняння різних класифікаторів.

17) Порівняння кількості точок вибірки.

18) Порівняння задання порогового значення.

19) Порівняння зміни масштабу зображення.

20) Порівняння зміни параметру кількості сусідів.

21) Результат оптимізації системи.

22) Результат роботи системи до оптимізації.

23) Результат роботи системи після оптимізації.

6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

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

7. Дата видачі завдання «06» листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1	Ознайомлення з теоретичними основами архітектури нейронних мереж, огляд алгоритмів і методів виявлення та розпізнавання обличчя на зображеннях	06.11.2023 – 18.11.2023	
2	Формування вимог та побудова архітектури системи	19.11.2023 – 23.11.2023	
3	Розробка системи розпізнавання обличчя, що ґрунтується на використанні нейронних мереж	24.11.2023 – 02.12.2023	
4	Оптимізація системи та її тестування на виявлення помилок	03.12.2023 – 10.12.2023	
5	Написання тексту кваліфікаційної роботи	11.12.2023 – 16.12.2023	

Здобувач вищої освіти _____ Нікіта ЖИГАМОВСЬКИЙ

Керівник роботи _____ Уляна ШВЕЦЬ

АНОТАЦІЯ

Звіт містить: 62 с., 23 рис., 10 джерел, 5 додатків.

Мета роботи: створення ефективної моделі, яка здатна точно розпізнати обличчя конкретної людини через камеру.

Об’єкт досліджень: процес розпізнавання облич.

Предмет дослідження: алгоритми та методи розпізнавання та виявлення облич у кадрі.

Розроблено систему розпізнавання обличчя, що ґрунтується на використанні нейронних мереж. У роботі для розпізнавання був використаний алгоритм Віоли Джонса за ознаками Хаара, який базується на використанні зображення в інтегральному представленні. У роботі був застосований метод локальних бінарних шаблонів для навчання моделі та прогнозування результатів, що полягав у діленні області обличчя на невеликі області, з яких одержувалися гістограми локальних бінарних шаблонів та поєднувалися в єдину гістограму просторово розширених ознак, що ефективно подавало зображення обличчя.

Ключові слова: НЕЙРОННІ МЕРЕЖІ, ВИЯВЛЕННЯ ОБЛИЧ, РОЗПІЗНАВАННЯ ОБЛИЧ, ЗОБРАЖЕННЯ, АЛГОРИТМ ВІОЛІ-ДЖОНСА, ЛОКАЛЬНІ БІНАРНИ ШАБЛОНИ, ОЗНАКИ ХААРА

ЗМІСТ

ВСТУП

У сучасному світі нейронні мережі вирішують проблеми, які потребують розпізнавання образів. Наприклад, нейронну мережу можна навчити розпізнавати рукописні цифри. Іншим прикладом є безпілотний автомобіль Google, навчений класично розпізнавати собаку, вантажівку чи автомобіль. Вони підходять для розпізнавання образів, класифікації та оптимізації.

Це включає рішення для розпізнавання рукописного тексту, розпізнавання обличчя, розпізнавання мовлення, перекладу тексту, виявлення шахрайства з кредитними картками, медичної діагностики та величезних обсягів даних. Його можна використовувати для пошуку зв’язків між шаблонами, перетворення одного типу даних в інший і створення асоціацій або узагальнень між різними сутностями.

Ця робота зосереджена на створенні системи, що застосовує передові технології виявлення та розпізнавання облич.

У контексті дослідження, нейронні мережі – це потужний інструмент для отримання бажаного результату. Існують різні алгоритми та методи виявлення та розпізнавання облич, проте у кожного з них є свої переваги та недоліків. У цій роботі приділяється увага, окрім теоретичних аспектів, ще й практичному застосуванню.

1. НЕЙРОННІ МЕРЕЖІ

1.1 Структура та компоненти

Нейронні мережі є функціональними одиницями глибокого навчання та, як відомо, імітують поведінку людського мозку для вирішення складних завдань, заснованих на даних. Вхідні дані обробляються різними шарами штучних нейронів, об'єднаних разом для отримання бажаного результату. Нейронні мережі використовуються в усьому: від розпізнавання мови та особи до охорони здоров'я та маркетингу.

Архітектура нейронної мережі складається з окремих одиниць, званих нейронами, які імітують біологічну поведінку мозку [1]. Різні компоненти нейрона наведено на рис 1.1.

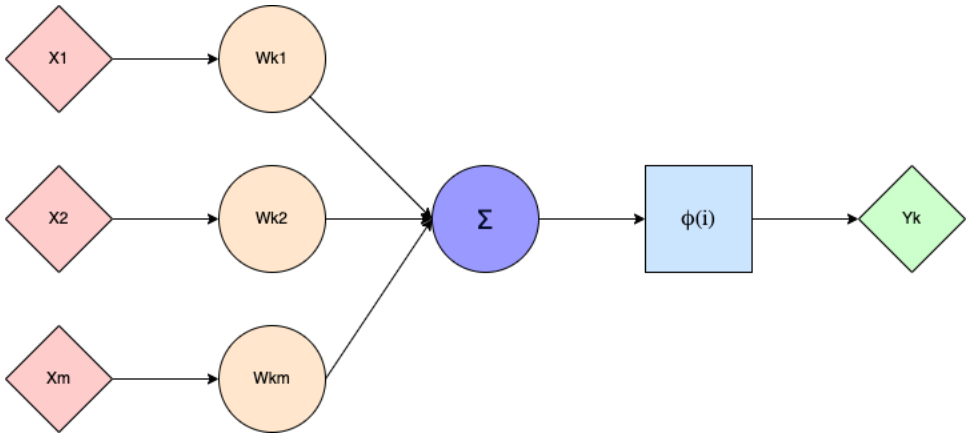

Рис 1.1 – Компоненти нейрона

Input – це набір функцій, введених у модель процесу навчання. Наприклад, введенням для пошуку об'єкта може бути масив значень пікселів, пов'язаних із зображенням.

Weight – головне завдання надавати значення завданням, які найбільш зручні для навчання. Це досягається введенням скалярного множення між вхідним значенням і ваговою матрицею. Наприклад, негативне слово вплине на рішення моделі аналізу настроїв більше, ніж пара нейтральних слів.

Transfer function – суть функції передачі полягає в об’єднанні кількох входів в одне вихідне значення, щоб можна було реалізувати функцію активації. Це сума всіх вхідних даних функції передачі.

Activation function – вносить нелінійність у роботу перцептронів, щоб врахувати різну лінійність із входами. Без цього вихід буде просто лінійною комбінацією вхідних значень і не вноситиме нелінійності в мережу.

Bias – роль зміщення полягає у зсуві значення, створеного функцією активації. Його роль подібна до константи в лінійній функції.

Коли кілька нейронів укладаються разом у ряд, вони утворюють шар, і кілька шарів, розташованих один на одному, називаються багатошаровою нейронною мережею. Багатошарова нейронна мережа наведена на рис 1.2.

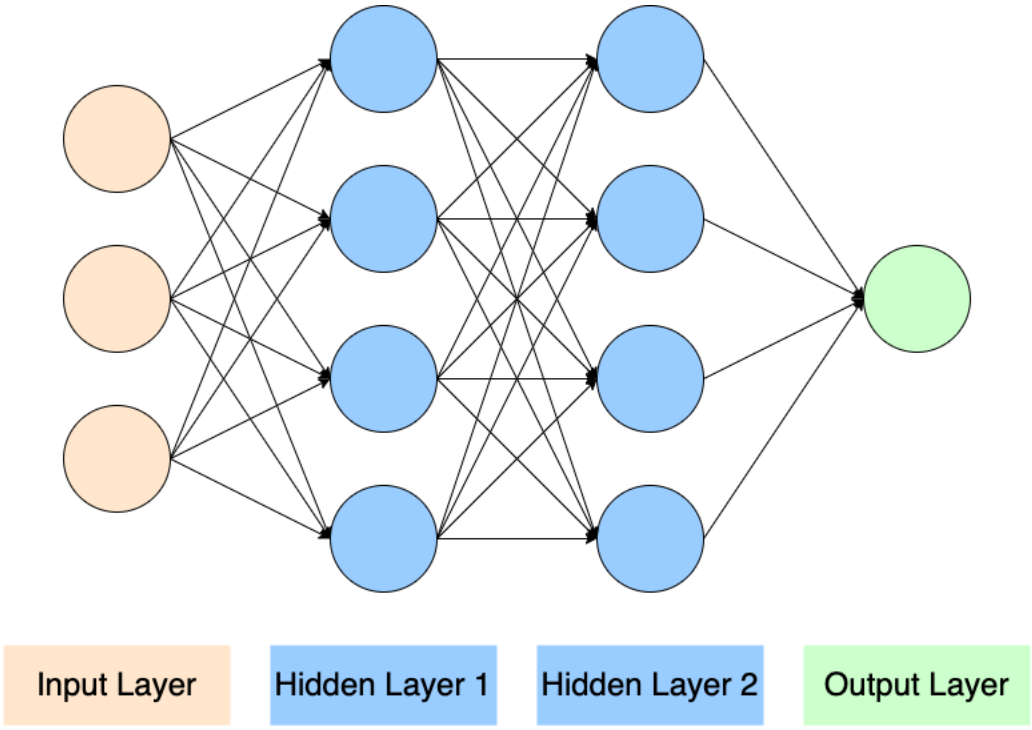

Рис 1.2 - Архітектура багатошарової нейронної мережі

Input Layer – дані, які подаються в модель, завантажуються на вхідний рівень із зовнішніх джерел, таких як файл CSV або веб-служба. Це єдиний оптичний рівень у повній архітектурі нейронної мережі, який надає повну інформацію із зовнішнього світу без будь-яких обчислень.

Hidden Layers – це те, що робить глибоке навчання таким, яким воно є сьогодні. Вони є проміжними рівнями, які виконують усі обчислення та витягують характеристики з даних. Дані можуть мати кілька взаємопов'язаних прихованих шарів, які призводять до пошуку різних прихованих функцій. Наприклад, під час обробки зображень перші приховані шари відповідають за високорівневі функції, такі як краї, форми чи межі. З іншого боку, пізніші приховані шари виконують більш складні завдання, такі як виявлення цілих об'єктів (автомобіль, будівля, людина).

Output Layer – отримує вхідні дані з попередніх прихованих шарів і приводить до остаточного прогнозу на основі навчання моделі. Це найважливіший шар, де ми отримуємо кінцевий результат. У моделях класифікації або регресії вихідний рівень зазвичай має один вузол. Однак це повністю залежить від конкретної проблеми і залежить від того, як побудована модель.

1.2 Роль нейромереж у повсякденному житті

Нейронні мережі вирішують проблеми, які потребують розпізнавання образів. Наприклад, нейронну мережу можна навчити розпізнавати рукописні цифри. Іншим прикладом є безпілотний автомобіль Google, навчений класично розпізнавати собаку, вантажівку чи автомобіль. Вони підходять для розпізнавання образів, класифікації та оптимізації. Це включає рішення для розпізнавання рукописного тексту, розпізнавання обличчя, розпізнавання мовлення, перекладу тексту, виявлення шахрайства з кредитними картками, медичної діагностики та величезних обсягів даних. Його можна використовувати для пошуку зв'язків між шаблонами, перетворення одного типу даних в інший і створення асоціацій або узагальнень між різними сутностями.

Нейронні мережі являють собою клас алгоритмів машинного навчання, які мають багато застосувань. Одними з найпопулярніших застосувань нейронних мереж є комп'ютерне бачення, розпізнавання мови та обробка природної мови. Сьогодні нейронні мережі використовуються в широкому діапазоні програм і

користуються великою увагою дослідницького співтовариства. Штучні нейронні мережі можна використовувати для вирішення багатьох складних проблем, які постають сьогодні. Вони використовуються як компонент більшої системи або можуть використовуватися на етапі попередньої обробки складних нелінійних методів.

Найбільша проблема нейронних мереж полягає в тому, що вони не надто точні, здебільшого тому, що мають відносно повільну криву навчання. І проблема є не лише у точності, а й в ефективності. Нейронні мережі можуть працювати надзвичайно повільно, оскільки вони часто покладаються на зворотний зв'язок від попередніх обчислень до наступних. Простий спосіб вирішити це – видалити один із багатьох рівнів мережі, щоб уникнути такого зворотного зв'язку, але це може фактично пошкодити точність мережі. Іншим рішенням є використання паралельних обчислень, які можна застосовувати для розподілу робочого навантаження та усунення проблем зі швидкістю.

1.3 Тонкощі навчання нейромережі

Існує три основні типи навчання в нейронних мережах: кероване навчання, некероване навчання та навчання з підкріпленням.

Кероване навчання – це тип машинного навчання, у якому моделі навчаються за допомогою добре "позначених" навчальних даних, а самі моделі прогнозують результати на основі цих даних. Позначені дані означають, що деякі вхідні дані вже позначені правильними вихідними даними.

У керованому навчанні навчальні дані, надані системам, діють як супервізор, який навчає систему точно прогнозувати результат. Це стосується того, що учень навчається під наглядом учителя.

Кероване навчання – це процес надання точних вихідних і вхідних даних для моделі машинного навчання. Мета контрольованого алгоритму навчання полягає в тому, щоб знайти функцію відображення для відображення вхідної змінної (x) із вихідною змінною (y).

У реальному світі контрольоване навчання можна використовувати для оцінки ризиків, класифікації зображень, виявлення шахрайства, фільтрації спаму тощо.

Некероване навчання – це метод машинного навчання, який використовує неконтрольовані моделі з використанням навчального набору даних. Натомість моделі самі виявляють приховані закономірності та ідеї з наданих даних. Це можна порівняти з навчанням, яке відбувається в людському мозку під час вивчення нового. На відміну від контрольованого навчання, неконтрольоване навчання не можна безпосередньо застосувати до проблеми регресії чи класифікації, оскільки у нас є вхідні дані, але немає відповідних вихідних даних. Мета неконтрольованого навчання полягає в тому, щоб виявити базову структуру набору даних, згрупувати ці дані за подібністю та представити цей набір даних у стисненому форматі.

Деякі з основних причин, які описують важливість навчання без нагляду:

- допомагає отримувати корисну інформацію з даних
- дуже схоже на те, як людина вчиться думати через свій досвід, наближаючи його до справжнього штучного інтелекту
- на немаркованих і несекретних даних, що робить неконтрольоване навчання ще більш важливим
- у реальному світі ми не завжди маємо вхідні дані з відповідними вихідними даними, тому для вирішення таких випадків нам потрібне некероване навчання

Навчання з підкріпленням – це техніка машинного навчання на основі зворотного зв’язку, за якої агент вчиться поводитися в середовищі, виконуючи дії та бачачи результати дій. За кожну хорошу дію агент отримує позитивний відгук, а за кожну погану дію агент отримує негативний відгук або покарання.

У навчанні з підкріпленням агент навчається автоматично, використовуючи зворотний зв’язок, без позначених даних, на відміну від навчання під наглядом. Оскільки маркованих даних немає, агент зобов’язаний навчатися лише на власному досвіді.

Ця техніка вирішує певний тип задач, такий як ігри, робототехніка тощо, де прийняття рішень є послідовним, а мета є довгостроковою.

Агент взаємодіє з середовищем і досліджує його самостійно. Основною метою агента навчання з підкріпленням є покращення ефективності шляхом максимізації позитивних винагород.

Агент вчиться в процесі ударів і експериментів, а на основі досвіду вчиться виконувати роботу кращим чином. Тому навчання з підкріпленням - це метод машинного навчання, за якого інтелектуальний агент (комп'ютерна програма) взаємодіє з навколишнім середовищем і вчиться діяти в ньому. Прикладом навчання з підкріпленням є те, як собака-робот вивчає рух рук.

Це фундаментальна частина штучного інтелекту (ШІ), і всі агенти ШІ працюють над концепцією навчання з підкріпленням. Тут нам не потрібно попередньо програмувати агента, оскільки він навчається на власному досвіді без втручання людини.

На прикладі в середовищі лабіринту є агент штучного інтелекту, метою якого є знайти алмаз. Агент взаємодіє з навколишнім середовищем, виконуючи певні дії, і на основі цих дій змінюється стан агента, і він отримує винагороду або покарання як зворотний зв'язок. Агент продовжує виконувати ці три дії (діяти, змінювати стан або залишатися в тому самому стані та отримувати зворотний зв'язок), і, виконуючи ці дії, він вивчає та досліджує середовище. Агент дізнається, які дії призводять до позитивного відгуку або винагороди, а які до покарання за негативний відгук. В якості позитивної винагороди агент отримує позитивний бал, а в якості покарання негативний бал.

1.4 Глибоке навчання

Глибоке навчання являє собою тип машинного навчання та штучного інтелекту, який імітує те, як люди отримують певні типи знань. Моделі глибокого навчання можна навчити виконувати завдання класифікації та розпізнавати шаблони в зображеннях, тексті, аудіо та багатьох інших даних. Він також

використовується для автоматизації завдань, які зазвичай потребують людського інтелекту, таких як опис зображень або транскрибування аудіофайлів.

Глибоке навчання є важливим аспектом науки про дані, включаючи статистику та прогнозоване моделювання. Це надзвичайно корисно для спеціалістів із обробки даних, які займаються збором, аналізом та інтерпретацією великих обсягів даних. Глибоке навчання робить цей процес швидшим і легшим.

Хоча людський мозок має мільйони взаємопов'язаних нейронів, які працюють разом, щоб вивчати інформацію, глибоке навчання включає нейронні мережі, що складаються з кількох шарів програмних вузлів, які працюють разом. Моделі глибокого навчання навчаються за допомогою великого набору даних міток і архітектури нейронної мережі [2].

Глибоке навчання дозволяє комп'ютеру навчатися на прикладі. Щоб зрозуміти глибоке навчання, потрібно подумати про перше слово дитини як про собаку. Малюк дізнається, що таке собака, а що ні, вказуючи на предмети та вимовляючи слово собака. Батьки кажуть: "Так, це собака" або "Ні, це не собака". Коли малюк продовжує вказувати на предмети, він починає краще усвідомлювати особливості всіх собак. Маленька дитина несвідомо пояснює складну абстракцію: поняття собаки. Вони роблять це шляхом побудови ієрархії, в якій кожен рівень абстракції створюється на основі знань, отриманих з попереднього рівня ієрархії.

Комп'ютерні програми, які використовують глибоке навчання, проходять подібний процес до того, як дитина вчиться розпізнавати собаку.

Програми глибокого навчання мають кілька рівнів взаємопов'язаних вузлів, кожен рівень побудований на останньому для вдосконалення й оптимізації прогнозів і класифікацій. Глибоке навчання виконує нелінійні перетворення на своїх вхідних даних і використовує отримані дані як вихідні дані для створення статистичної моделі. Ітерація триває, доки результат не досягне прийнятної точності. Кількість шарів обробки, через які повинні проходити дані, визначає глибину мітки.

У традиційному машинному навчанні процес навчання контролюється, і програміст повинен бути надзвичайно конкретним, повідомляючи комп'ютеру, на

що слід звернути увагу, щоб визначити, містить зображення собаку чи ні. Це трудомісткий процес, відомий як вилучення функцій, і рівень успіху комп'ютера повністю залежить від здатності програміста правильно визначити набір функцій для собаки. Перевага глибокого навчання полягає в тому, що програма створює набір функцій самостійно без нагляду.

Спочатку комп'ютерна програма може надати тренувальні дані, такі як набір зображень, які не стосуються собак, із людським позначенням кожного зображення собакою або метатегом. Програма використовує інформацію з тренувальних даних, щоб створити набір функцій для собаки та створити прогнозовану модель. У цьому випадку модель, яку спочатку створює комп'ютер, може передбачити, що все на зображенні з чотирма лапами та хвостом має бути позначено собакою. Насправді ця програма не знає ні про чотири ноги, ні про хвіст мітки. Він просто шукає шаблони пікселів у цифрових даних. З кожною ітерацією модель прогнозування стає складнішою та точнішою.

На відміну від малюка, якому потрібні тижні або місяці, щоб зрозуміти концепцію собаки, комп'ютерна програма, яка використовує алгоритми глибокого навчання, може продемонструвати навчальний набір, правильно ідентифікувати мільйони зображень із собаками та сортувати мільйони зображень. хвилин.

Щоб досягти прийнятного рівня точності, програми глибокого навчання вимагають доступу до величезних обсягів навчальних даних і потужності обробки, які були недоступні програмістам до ери великих даних і хмарних обчислень. Оскільки програмування глибокого навчання може створювати складні статистичні моделі, воно здатне створювати точні прогнозні моделі з великих обсягів немаркованих, неструктурованих даних.

2. МЕТОДИ ВИЯВЛЕННЯ ТА РОЗПІЗНАВАННЯ ОБЛИЧ

Виявлення обличчя - це перший і найважливіший етап процесу розпізнавання обличчя, який використовується для ідентифікації облич на фотографіях або відео. Це функція розпізнавання об'єктів, яка має застосування в різних сферах, включаючи безпеку, біометрію, правоохоронні органи, розваги та особисту безпеку. Існує багато алгоритмів машинного навчання, які можна вибрати для вирішення задачі.

2.1 AdaBoost

Алгоритм AdaBoost є технікою підвищення, яка використовується як метод ансамблю в машинному навчанні. Це називається адаптивним посиленням, оскільки вага перепризначається кожному екземпляру, де неправильно класифікованим екземплярам призначається більша вага [3].

Цей алгоритм створює модель і надає однакову вагу всім точкам даних. Потім він призначає більшу вагу неправильно класифікованим балам. Тепер усі пункти з більшою вагою набувають більшого значення в наступній моделі. Він зберігає навчальні моделі, доки не буде знайдено низьку помилку.

Є декілька важливих кроків цього алгоритму.

Важливий крок це підрахунок ваги зразка за формулою:

$$\omega(x_i, y_i) = \frac{1}{N}, i = 1, 2, \ldots . n. \tag{2.1}$$

Формула для підрахунку ваги зразка, де N - число, яке являє собою кількість точок даних. Далі обчислюється "значущість", "важливість" або "вплив" під час класифікації точок даних за формулою:

$$\frac{1}{2}\log\frac{1-Total\ Error}{Total\ Error}. \tag{2.2}$$

Загальна помилка є нічим іншим, як сумою всіх ваг вибірки неправильно класифікованих точок даних.

Неправильні передбачення набувають більшої ваги, а правильні – меншої. Після оновлення ваги, при створенні моделі, більша перевага надаватиметься важчим точкам. Після визначення значущості класифікації та загальної похибки оновлюється вага за формулою:

$$new\ sample\ weight = old\ weight * e^{\pm Amount\ of\ say\ (\alpha)}. \quad (2.3)$$

Коефіцієнт (альфа) буде негативним, якщо зразок правильно відсортовано і буде позитивнім, якщо вибірка неправильно класифікована.

Повторювання цих кроків, призведе до низької похибки навчання.

2.2 SNW

Алгоритм SNW (Sparse Network of Winnows) – це багатокласовий класифікатор, спеціально розроблений для великомасштабних навчальних завдань і доменів fpr. Він вивчає розріджену мережу лінійних функцій, де цільові поняття представлені як лінійні функції в загальному просторі функцій.

У SNW можна використовувати кілька правил оновлення, такі як Winnow, Perceptron і naive Bayes. Архітектура навчання SNW успадковує правило оновлення, яке використовує властивості узагальнення. Таким чином, коли використовується Winnow, це ефективний алгоритм навчання функцій, який лінійно масштабується з кількістю відповідних функцій і лінійно з кількістю функцій, що працюють у домені.

Проте є деякі відмінності, на які слід звернути увагу порівняно з простим використанням базового правила оновлення:

- Змінний розмір введення через нескінченний домен атрибутів

- Більший вираз, ніж основне правило Winnow. Базове правило оновлення Winnow використовує лише позитивні ваги. Стандартне розширення, за допомогою трюку дублювання, неможливе у просторі великої розмірності
- Особливості способів кадрування
- Механізм прогностичного переконання
- Функція на основі даних і сегментація посилань
- Механізм підтримки прийняття рішень
- Інтеграція з механізмом вилучення реляційних функцій (FEX), який забезпечує можливість гнучкого включення зовнішніх джерел інформації (функцій)

SNW успішно використовувався в різних широкомасштабних навчальних завданнях у сфері природної мови, а останнім часом і в області візуальної обробки. Він може вчитися та узагальнювати на невеликій кількості прикладів, таким чином добре адаптуючись до нових умов.

2.3 Support Vector Machine

Support Vector Machine (SVM) – це керований алгоритм машинного навчання, який використовується як для класифікації, так і для регресії. Основна мета алгоритму це знайти оптимальну гіперплощину в N-вимірному просторі, яка може розділяти точки даних різних класів у просторі ознак. Гіперплощина намагається забезпечити якомога більший запас між найближчими точками різних класів. Розмірність гіперплощини залежить від кількості ознак [4]. Якщо кількість вхідних елементів дорівнює двом, гіперплощина є просто лінією. Якщо кількість вхідних елементів дорівнює трьом, гіперплощина стає двовимірною площиною.

Якщо розглянути задачу двійкової класифікації з двома класами, позначеними +1 і -1, то припускається, що є навчальний набір даних, що

складається з вхідних векторів ознак X і відповідних їм міток класу Y. Тож рівняння для лінійної гіперплощини:

$$w^T x + b = 0. \tag{2.4}$$

Вектор W є вектором нормалі до гіперплощини. Тобто напрямок, перпендикулярний до гіперплощини. Параметр b у рівнянні – зсув або відстань гіперплощини від початку координат вздовж нормалі вектора w.

Відстань між точкою даних x_i та межею прийняття рішення можна обчислити як:

$$d_i = \frac{\omega^T x_i + b}{||w||}, \tag{2.5}$$

де $||w||$ представляє Евклідову норму ваги вектору w. Евклідова норма нормалі вектору w для лінійних SVM класифікаторів:

$$\gamma = \begin{cases} 1 : \omega^T x_i + b \geq 0 \\ 0 : \omega^T x_i + b < 0 \end{cases}. \tag{2.6}$$

Виходячи з характеру межі прийняття рішень, SVM можна розділити на дві основні категорії, такі як: лінійні SVM та нелінійні SVM.

Лінійні SVM використовує лінійне порогове значення для поділу точок даних на різні класи. Якщо дані строго лінійно розділені, ідеальним є лінійний SVM. Це означає, що одна пряма (у 2D) або гіперплощина (у вищих вимірах) може повністю розділити точки даних на відповідні класи. Межа прийняття рішень – це гіперплощина, яка максимізує запас між класами.

Нелінійні SVM можна використовувати для класифікації даних, якщо їх неможливо розділити на два класи прямою лінією. Використовуючи функції ядра, нелінійні SVM можуть обробляти нелінійно розділені дані. Ці функції ядра

перетворюють вихідні вхідні дані у високовимірний простір ознак, де точки даних можуть бути розділені лінійно. Лінійний SVM використовується для визначення межі нелінійного рішення в цьому модифікованому просторі.

2.4 Метод Віолі-Джонса

Алгоритм Віоли-Джонса починається зі створення великого набору характеристик, ознак Хаара [5], які обчислюються в різних масштабах і місцях зображення. Потім алгоритм використовує AdaBoost для вибору найкращих функцій із набору та навчання класифікатора за допомогою цих функцій. Навчений класифікатор використовується для ковзання вікна по зображенню та оцінки присутності об’єкта в поточному вікні. Якщо об’єкт виявлено, розмір вікна змінюється, і процес повторюється, доки об’єкт не буде розміщено правильно.

Демонстрація роботи алгоритму Віола-Джонс зображено на рис 2.1.

Рис 2.1 – Демонстрація роботи алгоритму Віола-Джонс

Головною перевагою алгоритму Віоли-Джонса є його швидкість. Алгоритм використовує цілісне представлення зображення, що дозволяє дуже швидко обчислювати такі функції, подібних до ознак Хаара. Алгоритм навчання AdaBoost дає змогу навчити потужний класифікатор із невеликою кількістю функцій, що скорочує час обчислення.

Метод Віоли-Джонса розпізнавання об'єктів на обличчі містить три техніки:

- Інтегральні об‘єкти зображення, для вилучення ознак Хаара, мають прямокутний тип і отримуються з інтегрального зображення
- AdaBoost – це метод машинного навчання для розпізнавання обличчя, слово “boost” означає, що класифікатори є складними на кожному етапі каскаду, і вони створюються з базових класифікаторів за допомогою одного з чотирьох методів підвищення.
- Каскадний класифікатор використовується для ефективного поєднання багатьох функцій. Слово “каскад” у назві класифікатора означає, що результуючий класифікатор складається з кількох.

Прості класифікатори застосовуються до обраного регіону до тих пір, поки кандидат не буде відхилено на якомусь етапі, або всі етапи пройдено. Нарешті, модель може отримати область без обличчя та область з обличчям після каскадування кожного сильного класифікатора.

2.5 Eigenfaces

Основою методу власних граней є аналіз головних компонент. Власні обличчя та PCA були використані Л. Сировичем та М. Кірбі для ефективного представлення зображень обличчя. Вони почали з набору оригінальних зображень обличчя та розрахували найкращу векторну систему для стиснення зображень. Пізніше М. А. Тюрк і А. П. Пентланд використовували метод власних граней для вирішення проблеми розпізнавання.

Аналіз головних компонент – це метод проектування на підпростір, який широко використовується в розпізнаванні образів. Метою PCA є заміна корельованих векторів великих розмірів некорельованими векторами малих розмірів. Іншою метою є обчислення основи для набору даних. Основними перевагами PCA є його низька чутливість до шуму, знижені вимоги до пам'яті та ємності, а також підвищена ефективність завдяки роботі в невеликому розмірі.

Стратегія методу власних граней полягає у виділенні характерних рис обличчя та представленні відповідного обличчя як лінійної комбінації так званих «власних граней», отриманих у процесі виділення ознак. Розраховано головні компоненти граней навчальної множини [6]. Розпізнавання досягається проектуванням обличчя на простір, утворений власними гранями. Проводиться порівняння евклідової відстані власних векторів власних фаз і власних фаз досліджуваного зображення. Якщо ця відстань досить мала, людину впізнають. З іншого боку, якщо відстань занадто велика, вважається, що зображення належить людині, якій потрібно навчити систему.

Блок-схема алгоритму показана на рис 2.2.

В якості відправної точки зчитуються навчальні зображення розмірів $N \times N$ і вони перетворюються на розміри $N^2 \times 1$. Таким чином створюється навчальний набір розмірів $N^2 \times M$, де M є кількістю зразків зображень. Середнє значення набору зображень обчислюється як:

$$\psi = \frac{1}{M}\sum_{i=1}^{M} \Gamma_i, \tag{2.7}$$

де ψ – середньо-статистичне зображення; M – число зображень; Γ_i – вектор зображення.

Власні грані, що відповідають найвищим власним значенням, зберігаються. Ці власні грані визначають простір граней. Власний простір створюється проектуванням зображення на простір граней, утворений власними гранями. Таким чином обчислюються вагові вектори. Розміри зображення налаштовуються відповідно до специфікацій, а зображення покращується на етапах попередньої

обробки виявлення. Порівнюються вектор ваги зображення та вектор ваги облич у базі даних.

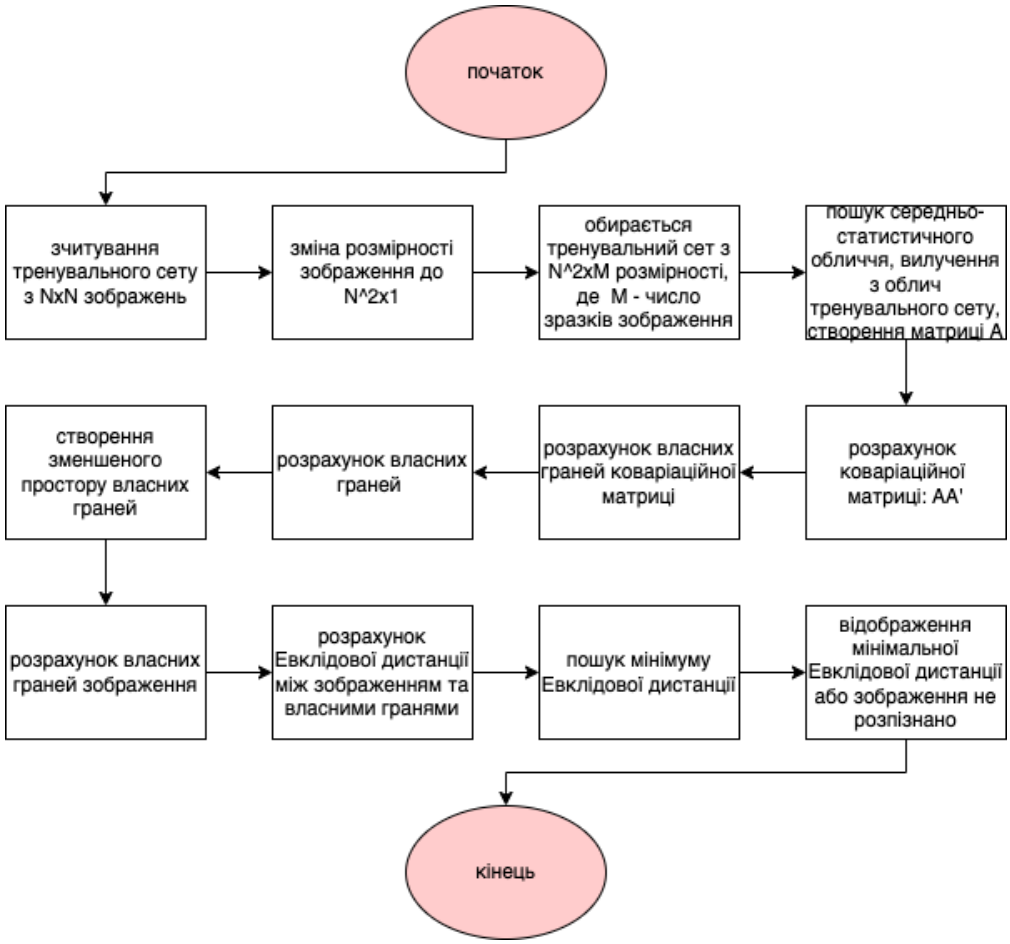

Рис 2.2 – Блок-схема алгоритму власних граней

Середнє обличчя обчислюється та віднімається від кожного обличчя в навчальному наборі. Матриця (A) формується за результатами операції віднімання. Різниця між кожним зображенням і середнім зображенням обчислюється за формулою:

$$\phi_i = \Gamma_i - \psi, i = 1,2,\ldots\ldots,M, \tag{2.8}$$

де ϕ – різниця між зображенням та середнім зображенням.

Менші власні значення відповідають меншим змінам власних векторів коваріаційної матриці. Риси обличчя збережені. Кількість власних векторів

залежить від точності, з якою визначена база даних, і може бути оптимізована. Вибраний набір власних векторів називаються власними гранями. Після отримання власних граней зображення в базі даних проектуються в простір власних граней, а ваги зображень зберігаються в цьому просторі. Для визначення ідентичності та зображення власні коефіцієнти порівнюються з власними коефіцієнтами в базі даних.

Формується власне обличчя відповідної фігури. Розраховується евклідова відстань між власними гранями зображення та попередньо збереженими власними гранями.

Особа, про яку йдеться, визначається як особа, евклідова відстань якої менша за мінімальне значення в базі даних власних граней. Якщо всі розраховані евклідові відстані перевищують порогове значення, зображення не можна розпізнати.

2.6 LBP

Локальні бінарні шаблони (LBP) – це візуальний дескриптор, який використовується для класифікації в комп'ютерному зорі. LBP є окремим випадком моделі Texture Spectrum, запропонованої в 1990 році та вперше був описаний у 1994 році. Відтоді було виявлено, що це потужна функція для класифікації текстур. Крім того, було визначено, що коли LBP поєднується з гістограмою дескрипторів орієнтованого градієнта (HOG), це значно покращує продуктивність розпізнавання деяких наборів даних. У 2015 році Е. Сільва та інші провели порівняння кількох покращень раннього LBP у сфері зменшення фону [7].

3. ОПЕРАТОРИ ЛОКАЛЬНИХ БІНАРНИХ ШАБЛОНІВ

3.1 Принцип роботи

Вектор ознак LBP у своїй найпростішій формі будується наступним чином:

- розділення визначеного вікна на клітинки (наприклад, 16×16 пікселів для кожної клітинки);
- для кожного пікселя в комірці порівнюється піксель із 8 його сусідами: ліворуч угорі, ліворуч у центрі, ліворуч унизу, праворуч угорі, тощо. Слідкування за пікселями вздовж кола, тобто за або проти годинникової стрілки;
- якщо значення центрального пікселя більше, ніж значення сусіднього, пишеться «0». В іншому разі пишеться «1». Це повертає 8-значне двійкове число, яке зазвичай перетворюється на десяткове для зручності;
- обчислюється гістограма по комірці частоти кожного числа, яке виникає, тобто кожної комбінації пікселів, менших і більших за центр. Цю гістограму можна розглядати як 256-вимірний вектор ознак;
- додаткове нормалізування гістограми;
- зведена гістограма всіх комірок, що повертає вектор ознак для всього вікна.

Тепер вектор ознак можна обробляти за допомогою опорних векторних машин, екстремально навчальних машин або інших алгоритмів машинного навчання для класифікації зображень. Такі класифікатори можна використовувати для розпізнавання облич або аналізу текстури.

Для оператора LBP використовується така нотація:

$$LBP_{P,R}^{u2}. \tag{3.1}$$

Нижній індекс означає використання оператора в околі (P, R). Верхній індекс $u2$ означає використання лише однакових візерунків і позначення всіх шаблонів, що залишилися, однією міткою. Після отримання зображення $f_l(x, y)$, позначеного LBP, гістограма LBP може бути визначена як:

$$H_i = \sum_{x,y} I\{f_l(x, y) = i\}, i = 0, \ldots, n-1, \quad (3.2)$$

де n – кількість різних міток, створених оператором LBP, а $I\{A\}$ дорівнює 1, якщо A – істина, і 0, якщо ні.

Якщо ділянки зображення, гістограми яких потрібно порівняти, мають різні розміри, гістограми необхідно нормалізувати, щоб отримати узгоджений опис:

$$N_i = \frac{H_i}{\sum_{j=0}^{n-1} H_j}. \quad (3.3)$$

3.2 Розширення

Існує декілька розширень операторів локальних бінарних шаблонів:

- OCLBP
- tLBP
- dLBP

Надповні локальні бінарні шаблони (OCLBP) – це варіант LBP, який покращує загальну ефективність перевірки обличчя. На відміну від LBP, OCLBP перекриває сусідні блоки. Формально конфігурація OCLBP позначається як $S: (a, b, v, h, p, r)$: зображення з вертикальним перекриттям v і горизонтальним перекриттям h ділиться на блоки $a \times b$, тоді рівномірний шаблон LBP $(u2, P, R)$ витягуються з усіх блоків. Крім того, OCLBP складається з кількох різних конфігурацій [8].

Загалом використовуються три конфігурації: S:(10, 10, 12, 12, 8, 1), (14, 14, 12, 12, 8, 2), (18, 18, 12, 12, 8, 3). Три конфігурації враховують три розміри блоку:

10×10, 14×14, 18×18 і коефіцієнт половинного перекриття вздовж вертикального та горизонтального напрямків. Ці конфігурації об'єднані, щоб сформувати 40877-вимірний вектор ознак для зображення розміром 150×80.

Локальні бінарні шаблони переходу (tLBP) являє собою варіант LBP, де бінарне значення кодованого переходу LBP складається з порівняння сусідніх пікселів за годинниковою стрілкою для всіх пікселів, крім центрального [8].

Локальні бінарні шаблони з направленим кодуванням (dLBP) – це варіант LBP, суть якого полягає у кодуванні зміни інтенсивності вздовж чотирьох основних напрямків через центральний піксель двома бітами [8].

4. ЕТАПИ ОБРОБКИ КАДРІВ

4.1 Виявлення обличчя методом Віоли-Джонса

Цей метод був розроблений і представлений Полом Віолою та Майклом Джонсом у 2001 році. Однак на сьогодні цей метод є основним способом пошуку об’єктів на зображенні в реальному часі.

Основні принципи, на яких базується робота цього методу:

- Інтегральне представлення зображень
- Пошук облич за допомогою ознак Хаара
- Каскадна класифікація із застосуванням бустингу

Щоб розрахувати яскравість прямокутної області зображення використовується інтегральне представлення, яке дуже часто використовується в більшості алгоритмів комп’ютерного зору. Воно дозволяє досить швидко обчислювати яскравість довільного прямокутника на певному зображенні, де швидкість обчислення не залежить від площі прямокутника

Цим інтегральним представленням зображення є матриця. Розмір матриці дорівнює розміру вхідного зображення. Кожний елемент матриці відповідає сумі інтенсивностей пікселів, які розташовані ліворуч і вище цього елемента. Самі елементи матриці розраховуються за формулою:

$$I(x,y) = \sum_{x \le x, y \le y} i(x,y), \tag{4.1}$$

де $I(x,y)$ – це значення точки (x,y) інтегрального зображення, а $i(x,y)$ – це значення інтенсивності вхідного зображення. Це дозволяє розрахувати однотипні ознаки, проте з різними геометричними параметрами. Обчислення матриці інтегрального представлення займає час, пропорційний кількості пікселів на зображенні.

4.2 Ознаки Хаара

Основним аспектом розпізнавання обличчя є розпізнавання відповідних рис людського обличчя, таких як очі, брови, ніс, губи. За допомогою ознак Хаара можна виявити ці риси в реальному часі.

Ознаки Хаара – це послідовність перемасштабованих квадратичних функцій форми, запропонована Альфредом Хааром у 1909 році. Вони подібні до згорткових ядер зі згорткових нейронних мереж. Ці ознаки застосовуються до всіх відповідних частин обличчя, для розпізнавання обличчя людини.

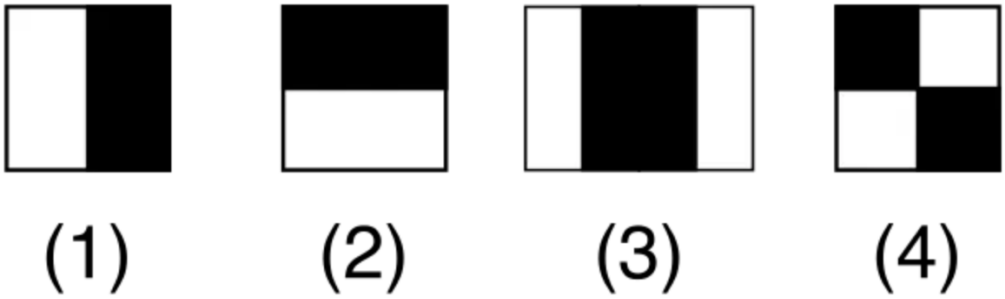

Рис 4.1 – Приклад ознак Хаара

Можна побачити на рис 4., що є кутові елементи (1 і 2) і лінії (3). Це білі та чорні піксельні зображення (значення 0 або значення 1). Але зазвичай є сіре або кольорове зображення (від 0 до 255), тому припустимо ідеальний сценарій, що є чорно-біле зображення.

Рис 4.2 – Оригінальне чорно-біле зображення

Для виявлення брів використовуються ознаки Хаара, оскільки лоб і брови утворюють світлі пікселі, а темні пікселі схожі на зображення. Подібним чином використовуються ознаки світло-темно-світлих пікселів для виявлення губ. Щоб виявити ніс, використовуються ознаки темно-світлих пікселів і так далі, як показано на рис 4.3.

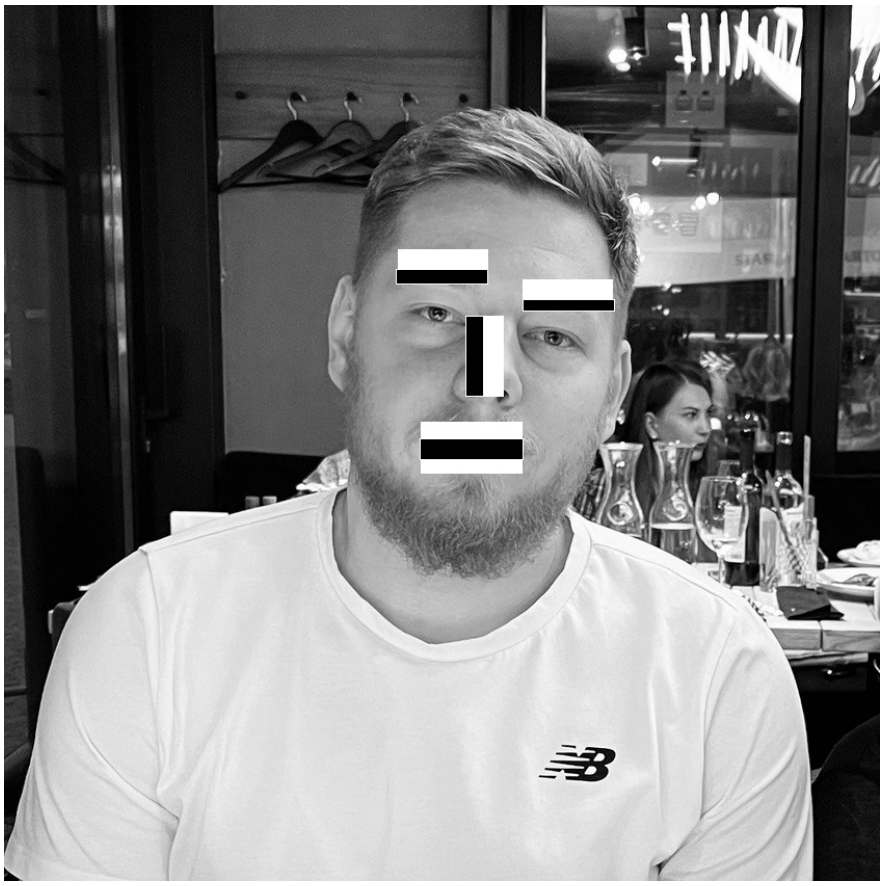

Рис 4.3 – Застосування ознак Хаара

Відповідно до алгоритму Віоли-Джонса, щоб виявити на зображенні подібну до Хаара рису, наступна формула повинна дати результат, близький до 1. Чим ближче значення до 1, тим більша різниця у виявленні ознаки Хаара на зображенні [9]:

$$\Delta = dark - white = \frac{1}{n}\sum_{dark}^{n} 1(x) - \frac{1}{n}\sum_{white}^{n} 1(x). \quad (4.2)$$

Каскадні класифікатори ознак Хаара є серією класифікаторів або ознак, які використовують для ідентифікації об’єкта на зображенні. Використання ковзних вікон і кількості ознак Хаара (збільшується зі збільшенням кількості етапів) зрештою призводить до розпізнавання обличчя чи ні. Всього для методу Віоли-Джонса визначено 38 етапів. Залежно від розміру ковзного вікна та положення

обличчя, кількості рис обличчя можна розпізнати на певному етапі, як показано на рис 4.4.

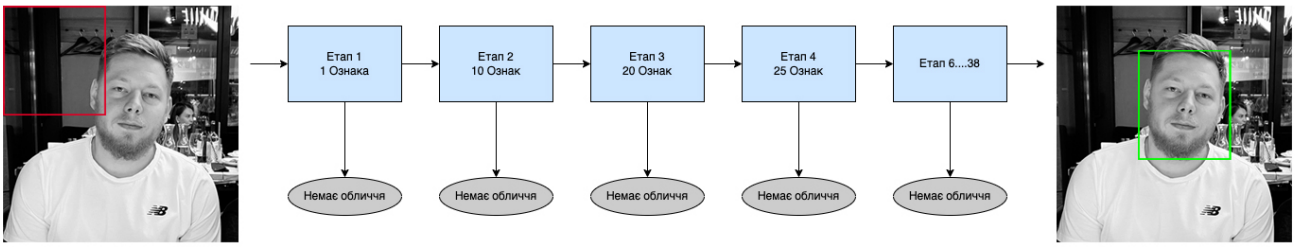

Рис 4.4 – Застосування каскадних класифікаторів ознак Хаара

4.3 LBP перетворення

Якщо у наборі даних є обличчя, першим кроком алгоритму є розділення обличчя на клітинки 7 × 7 однакового розміру, як показано на рис 4.5.

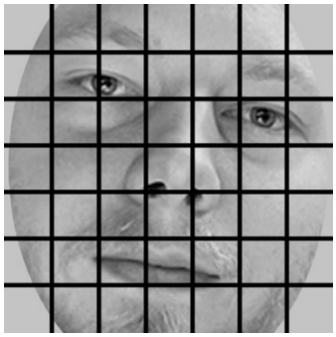

Рис 4.5 – Розділення обличчя на клітинки 7×7 однакового розміру

Потім для кожної з цих клітинок обчислюється локальна бінарна гістограма. За визначенням, гістограма відкидає всю просторову інформацію про те, як візерунки орієнтовані один відносно одного. Однак, обчислюючи гістограму для кожної клітинки, можна закодувати рівень просторової інформації про очі, ніс, рот тощо.

Це просторове кодування дозволяє по-різному зважувати гістограми з кожної клітинки, надаючи більшу здатність розрізняти найвиразніші риси обличчя, як показано на рис 4.6.

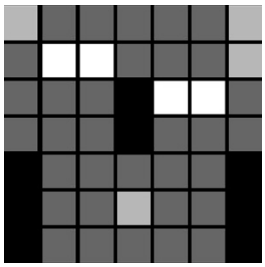

Рис 4.6 – Результат обчислення LBP для кожної клітинки

Схема зважування для кожної комірки:

- Гістограма LBP для білих клітин у 4 рази важча за інші клітини. Це просто означає, що ми беремо гістограми LBP з областей білих клітин і множимо їх на 4 (з урахуванням будь-якого масштабування або нормалізації гістограм).
- Світло-сірі клітини сприяють у 2 рази більше.
- Темно-сірі клітини сприяють лише 1×.
- Нарешті, чорні клітини повністю ігноруються і мають вагу 0×.

4.4 Метод найближчого сусіда

Метод найближчого сусіда – це простий алгоритм, який зберігає всі доступні випадки та класифікує нові дані або випадки на основі показника подібності. Його часто використовують для класифікації точки даних на основі того, як класифікуються її сусіди [10].

Виявлення обличчя виконується за допомогою класифікатора відстані x^2 і найближчого сусіда:

- Системі представлено обличчя.
- LBP витягуються, зважуються та комбінуються подібним чином до даних навчання.

- k-NN (з $k = 1$) виконується з відстанню x^2, щоб знайти найближче обличчя в навчальних даних.
- Ім’я особи, пов’язане з обличчям із найменшою відстанню x^2, вибирається як остаточний класифікатор.

5. РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ

5.1 Створення словника облич

Важливим моментом для розпізнавання обличчя є те, що потрібно зберігати ідентифікатори та імена користувачів, які було додано до системи. Для цього потрібно створити словник облич. Він повинен мати щонайменше 2 важливі функції:

- Додавання нового обличчя у словник.
- Пошук обличчя у словнику за ідентифікатором.

Перед додаванням обличчя у словник, спочатку перевіряється присутність обличчя с таким ідентифікатором у словнику. І тільки у випадку, якщо обличчя з таким ідентифікатором немає у словнику, воно додається. Блок-схема функції додавання обличчя у словник наведена на рис 5.1.

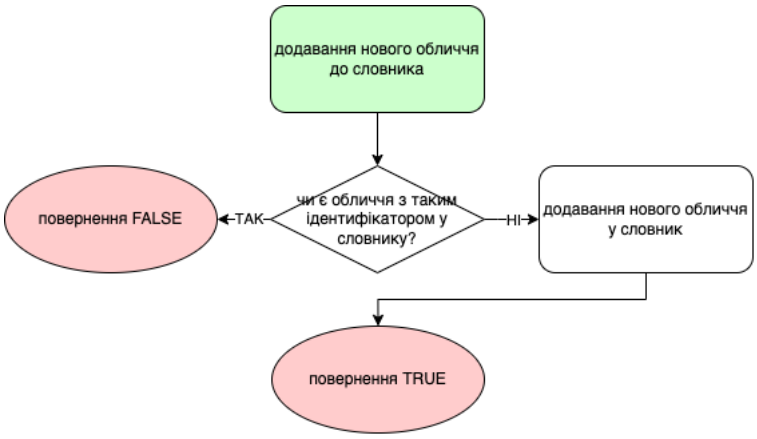

Рис 5.1 – Блок-схема функції додавання обличчя у словник

Функція додавання обличчя використовує іншу важливу функції, а саме пошук обличчя за ідентифікатором у словнику. В ній потрібно перебрати весь словник, і порівнювати кожен елемент у ньому на співпадіння з шуканим ідентифікатором обличчя. У випадку, якщо обличчя у словнику вже є, вертаємо його екземпляр, в іншому випадку вертаємо None, що буде означати, що обличчя з таким

ідентифікатором у словинку відсутнє. Блок-схема функції пошуку обличчя у словнику наведена на рис 5.2.

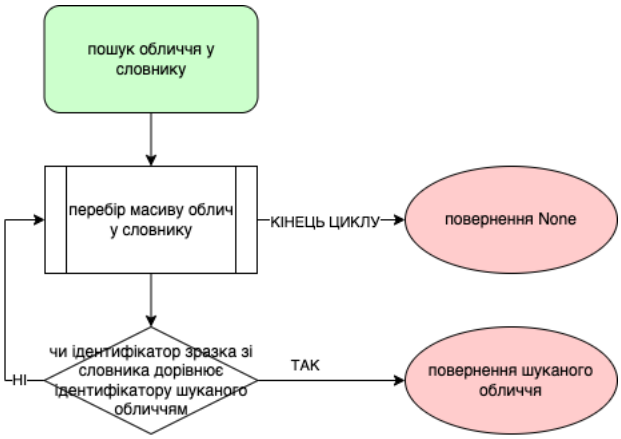

Рис 5.2 – Блок-схема функції пошуку обличчя у словнику

5.2 Сканування нового обличчя

Наступним етапом є сканування нового користувача. Перш за все для цього потрібно створити запит на введення ідентифікатору та імені користувача. Саме за цим ідентифікатором і буде розпізнаватися обличчя. Це значення чітко ідентифікує обличчя у словнику.

Наступним кроком є відкриття камери для запису відеопотоку. Саме через відеопотік з камери і отримуються зображення користувача (обличчя), яке потрібно відсканувати та занести до словника, а також потрібно створити необхідну кількість зображень обличчя та зберегти їх у директорію dataset, для подальшого навчання системи на основі цих зображень.

Кількість зображень, яке потрібно зберегти, в даному випадку відіграє велику роль, бо чим більше буде зразків, тим з кращою точністю буде фінальний результат. Але є й інша сторона, бо чим більше зображень потрібно зберегти, тим більше часу займає сканування обличчя. У цьому разі 100 зображень достатньо для розпізнавання облич, проте для покращення роботи системи потрібно розглянути зберігання 1 000 або 10 000 зображень.

Після виявлення кадру з відеопотоку камери, виявляється обличчя за допомогою алгоритму Віоли-Джонса та ознак Хаара, перебираються прямокутники та обрізається зображення до прямокутника. Після чого це зображення зберігається у директорії dataset, та виноється наступна ітерація для наступного кадру. Це буде відбуватись доти, поки ми не отримаємо 100 зображень обличчя для конкретного користувача.

Після того як всі зразки обличчя були збережені у директорію dataset, виводиться запит на додавання нового користувача. При позитивній відповіді, виконуються всі ті самі кроки для нього. У випадку, якщо всі необхідні користувачі були додані, відбувається запуск модуля тренування моделі. Блок схема модуля сканування обличчя наведена на рис 5.3.

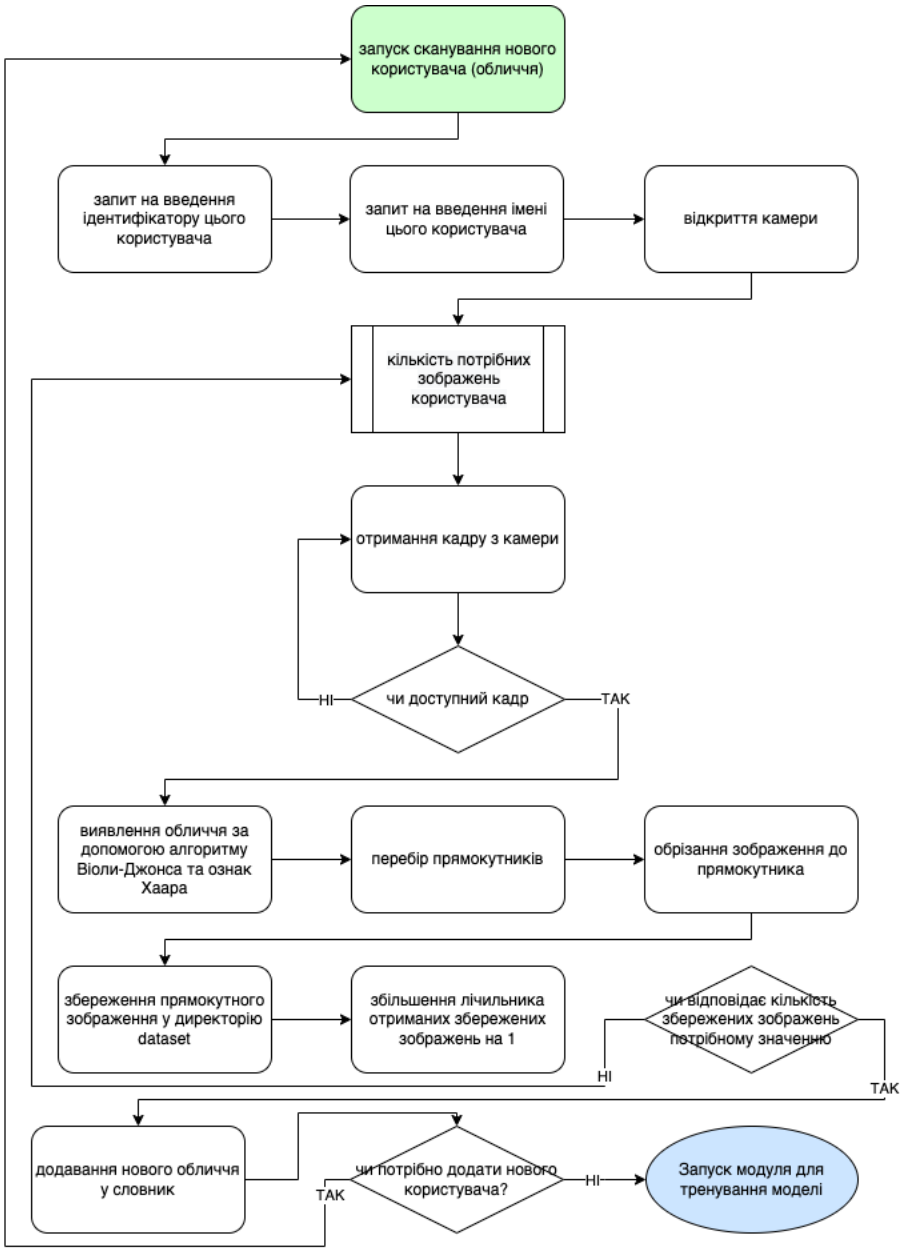

Рис 5.3 – Блок-схема модуля сканування нового користувача

5.3 Тренування моделі

Наступним етапом є тренування моделі. Перш за все створюється модель на основі LBP та класифікатор ознак Хаара для обличчя. Після цього отримуються всі шляхи до збережених зображень кожного користувача та перебираються, щоб отримати окремий шлях до зображення.

З поточного зображення вилучаються ідентифікатор та ім’я, та у випадку виявлення обличчя за допомогою алгоритму Віоли-Джонса та ознак Хаара,

перебираються прямокутники та додаються зображення, ідентифікатор та ім’я до масиву зразків облич, ідентифікаторів та імен. Після цього йде інша ітерація на іншому зображенні.

Коли всі збережені зображення були обробленні, відбувається тренування моделі на основі LBP алгоритму, з розставленням відповідних міток, для того щоб у подальшому кожне зображення було класифіковане і розпізнане як відповідне обличчя окремо взятого користувача.

Після цього перебирається масив вилучених ідентифікаторів, відбувається пошук обличчя у словнику облич за ідентифікатором, та у випадку якщо такого обличчя ще немає, воно додається до словника. Після тренування модель записується у файл trainer.yml.

Після цього є запит на запуск модуля розпізнавання обличчя. У випадку позитивної відповіді, він запускається. Якщо ж відповідть негативна – система закінчує роботу.

Блок-схема модуля тренування моделі наведена на рис 5.5.

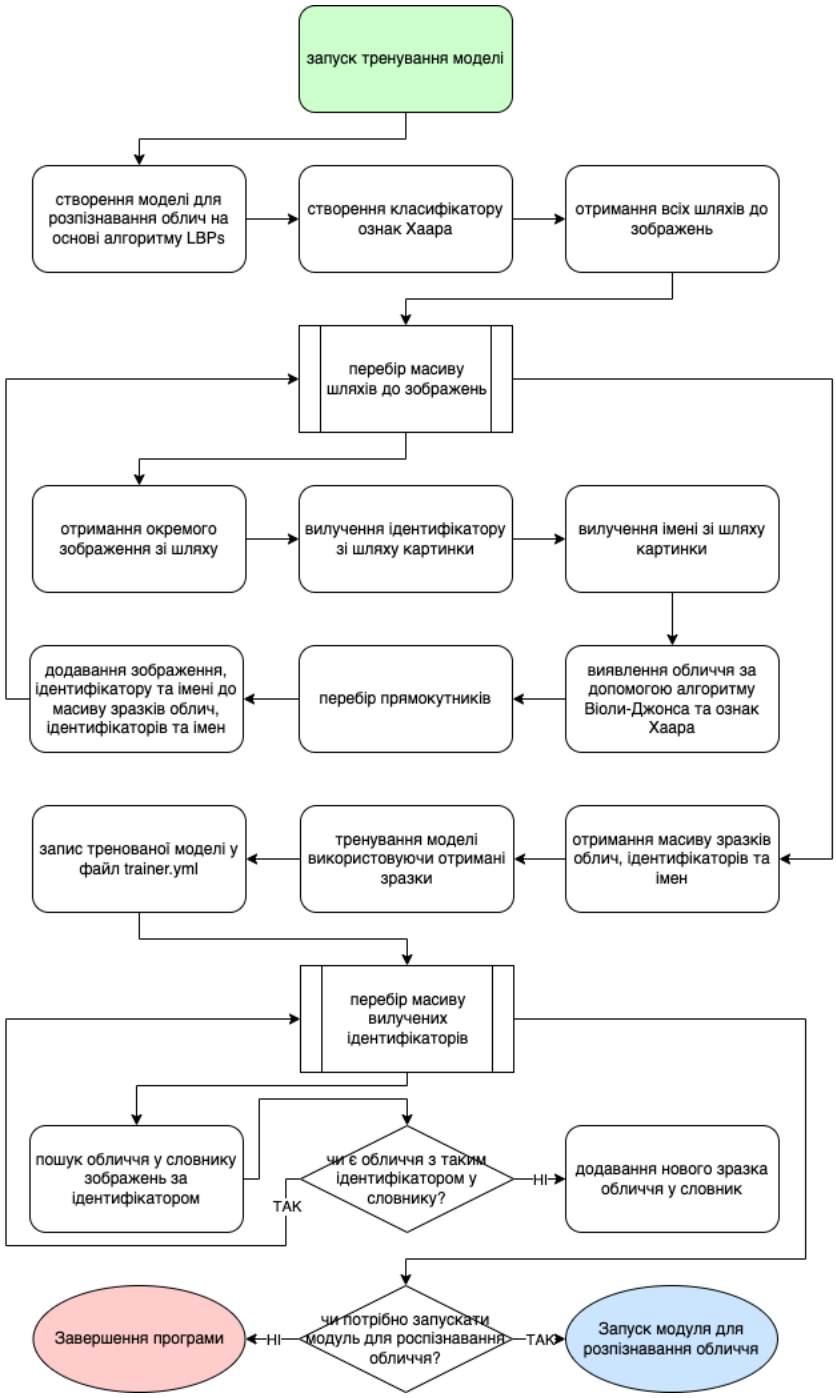

Рис 5.4 – Блок-схема модуля тренування моделі

5.4 Розпізнання обличчя

Наступним етапом є безпосереднє розпізнавання обличчя. Перш за все створюється модель на основі LBP. Після цього завантажується тренована модель з файлу trainer.yml після минулого етапу.

Для того щоб почати розпізнавати обличчя в реальному часі з відео потоку, відкривається камера. Далі береться кадр з відео потоку, використовується для виявлення обличчя за допомогою алгоритму Віоли-Джонса та ознак Хаара, відбувається обрізання зображення до прямокутника, перебір, розпізнавання до якого ідентифікатору належить зображення та отримання впевненості у співпадінні.

Мінімально допустиме значення для впевненості у співпадінні було задано як 50. Тобто якщо впевненість у розпізнаному обличчі нижче ніж 50, то вважається що обличчя не розпізнано, а якщо вище, то можна стверджувати, що обличчя розпізнано. У негативному випадку виводиться форматований заголовок із відповідним повідомленням.

Якщо значення впевненості більше ніж мінімально допустиме, далі відбувається пошук у словнику облич за розпізнаним ідентифікатором. У випадку, якщо такий ідентифікатор у словнику є, знаходиться і відповідне ім`я і виводиться форматований заголовок з ним.

Все це повторюється до тих пір, доки користувач не натисне на кнопку ESC. Тобто система у реальному часі розпізнає обличчя людини і виводить її ім’я, якщо обличчя розпізнано, доки не буде натиснута кнопка ESC.

Блок-схема модуля розпізнавання обличчя наведена на рис 5.5, а результат роботи системи наведений на рис 5.6.

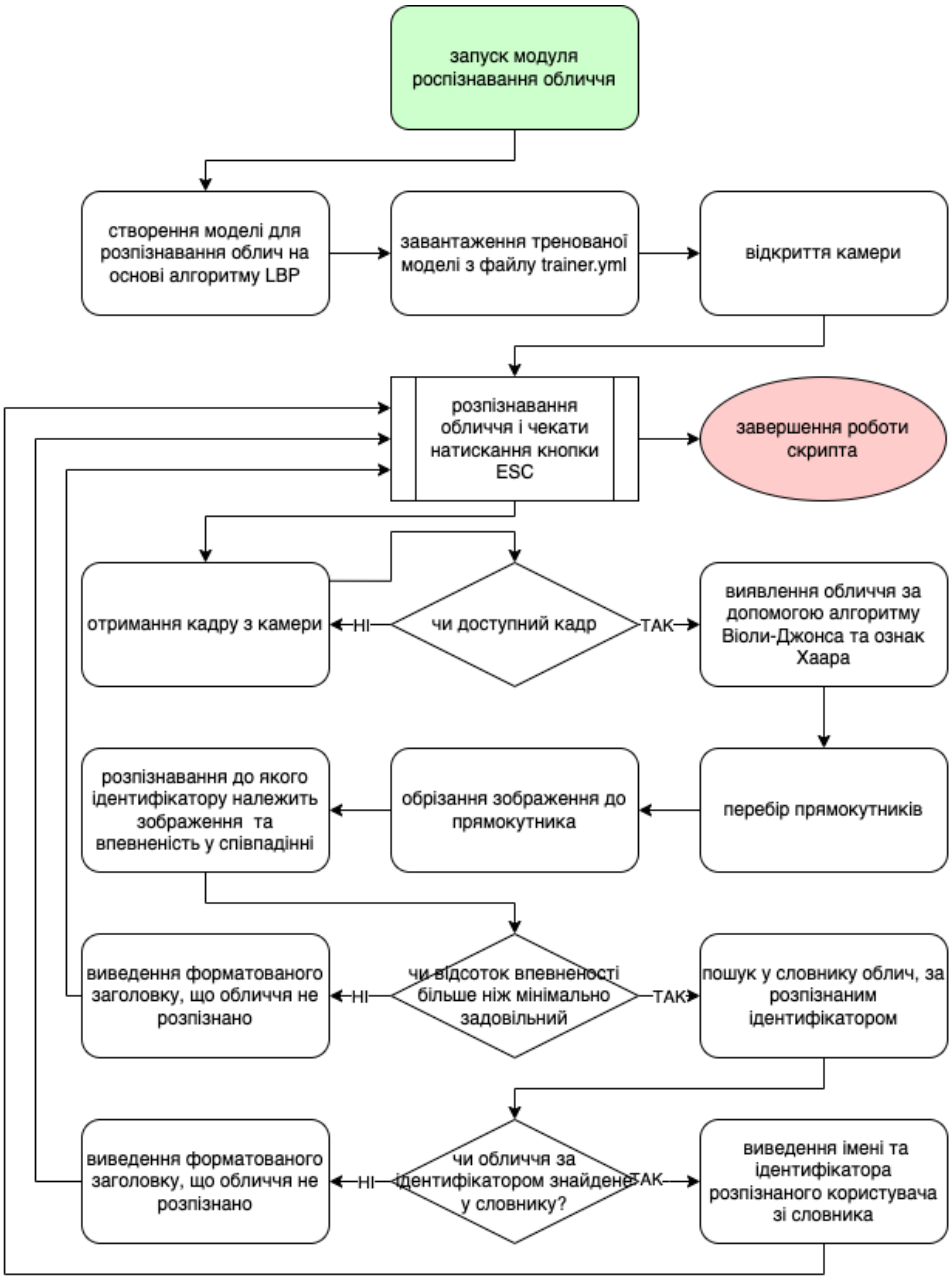

Рис 5.5 – Блок-схема модуля розпізнавання обличчя

5.5 Оптимізація системи

Останнім етапом є оптимізація системи. За допомогою зміни деяких параметрів можна нормалізувати якість розпізнання, зменшити час виявлення та збереження 100 зразків облич і час навчання системи, при цьому покращити відсоток виявлення різниці між обличчами різних людей.

Для початку було розглянуто існуючі класифікатори ознак Хаара для обличчя, та яка між ними різниця. Загалом вони відрізняються розміром ядра, який використовується у алгоритмі AdaBoost та іншими параметрами. В результаті отримуємо більше співпадінь з меншою якістю співпадінь.

У цьому разі найкращим варіантом було виявлено використання haarcascade_frontalface_alt2, що зменшує час виявлення та збереження 100 зразків зображень облич для одного користувача на 2 секунди та зменшує у два рази час навчання системи з незначною втратою якості розпізнання. Результат порівнянь наведений на рис 5.6.

класифікатор	час розпізнання 100 зразків облич (сек)	якість розпізнання (%)	час навчання (сек)
haarcascade_frontalface_default	12.37489376547834	95	2.9125749588012695
haarcascade_frontalface_alt	11.14719815254211	94	2.2784300804138184
haarcascade_frontalface_alt2	10.41787791252136	92	1.5462489032745361

Рис 5.6 – Порівняння різних класифікаторів

Наступним кроком в оптимізації системи була зміна точок вибірки для будування кругового локального бінарного шаблону. Зміна цього параметру дозволяє збільшити якість виявлення різниці між різними обличчями із втратою якості розпізнавання.

У цьому разі найкращим значенням кількості точок було виявлено 8, що зменшує кількість хибних розпізнавань із незначним зменшенням якості розпізнавання та незначним збільшенням часу навчання. Результат порівнянь наведений на рис 5.7.

кількість точок вибірки	якість розпізнання (%)	якість виявлення різниці (%)	час навчання (сек)
2	92	80	1.5462489032745361
3	91	79	1.5594689846038818
4	89	71	1.6152259540557861
5	87	68	1.6472823619842531
6	85	66	1.9173948764801025
7	77	48	1.5202207565307617
8	71	44	1.7581388950347934
9	59	25	2.2068548679351807

Рис 5.7 – Порівняння кількості точок вибірки

Ще один крок в оптимізації системи це задання порогового значення, яке використовується для прогнозу. Будь-яка відстань між тестом та найближчим знайденим зображенням ігнорується, якщо вона більше за цей поріг. Задання цього параметру дозволяє збільшити якість виявлення різниці між різними обличчями із незначною втратою якості розпізнання та незначним збільшенням часу навчання. Результат порівнянь наведений на рис 5.8.

застосування порогу відстані	якість розпізнання (%)	якість виявлення різниці (%)	час навчання (сек)
-	71	44	1.7581388950347934
123.0	70	31	2.0454671382904053

Рис 5.8 – Порівняння задання порогового значення

Наступним кроком оптимізації було зміна параметру, що визначає, наскільки розмір зображення зменшується при кожному масштабі зображення. Де значення 1 – це розмір зображення у масштабі один до одного, а значення 1.1 означає зменшення розміру на 10 відсотків у кожному масштабі зображення. Зміна цього параметру дозволяє зменшити час виявлення та збереження 100 зразків зображень облич для одного користувача та суттєво зменшити час навчання.

У цьому разі найкращим значенням для зміни розміру було виявлено 1.3, що означає зменшення розміру на 30 відсотків у кожному масштабі зображення із незначним збільшенням якості розпізнавання, зменшенням часу виявлення та

збереження 100 зразків зображень облич на 3 секунди і зменшенням часу навчання у 4 рази. Результат порівнянь наведений на рис 5.9.

масштабування зображення	якість розпізнання (%)	час розпізнання 100 зразків облич (сек)	час навчання (сек)
1.1	70	10.31329512596134	2.045467138290405
1.2	72	7.848832130432129	1.130454063415527
1.3	73	6.948884963989258	0.563546895980835
1.4	50	395.9130692481994621	5.045467138290405

Рис 5.9 – Порівняння зміни масштабу зображень

Останнім кроком в оптимізації системи була зміна параметру кількості сусідів, який повинен мати кожен прямокутник-кандидат, щоб зберегти його. Зміна цього параметру дозволяє збільшити якість розпізнання обличчя із незначним зменшенням якості виявлення різниці між різними обличчями.

У цьому разі найкращим значенням для кількості сусідів було виявлено 5, із незначним збільшенням кількості хибних розпізнавань та незначним збільшенням якості розпізнавання. Результат порівнянь наведений на рис 5.10.

кількість точок вибірки	якість розпізнання (%)	якість виявлення різниці (%)	час розпізнання 100 зразків облич (сек)	час навчання (сек)
8	73	25	6.948884963989258	0.563546895980835
7	74	31	10.57259178161621	0.723245436793345
6	75	36	9.759320735931396	0.835319042205815
5	76	39	8.509862899780273	0.713090181350708
4	84	65	7.089170932769775	0.639847538495798

Рис 5.10 – Порівняння зміни параметру кількості сусідів

Результатом оптимізації системи є нормалізація якості розпізнання обличчя, зменшення часу виявлення та збереження 100 зразків облич на 4 секунди, зменшення часу навчання системи у 4 рази та збільшення якості виявлення різниці між обличчами різних людей у 2 рази. Результат порівнянь наведений на рис 5.11. Результат роботи системи до та після оптимізації наведений на рис 5.12 та рис 5.13.

результат оптимізації	якість розпізнання (%)	якість виявлення різниці (%)	час розпізнання 100 зразків облич (сек)	час навчання (сек)
до	95	80	12.37489376547834	2.912574958801269
після	76	39	8.509862899780273	0.713090181350708

Рис 5.11 – Результат оптимізації системи

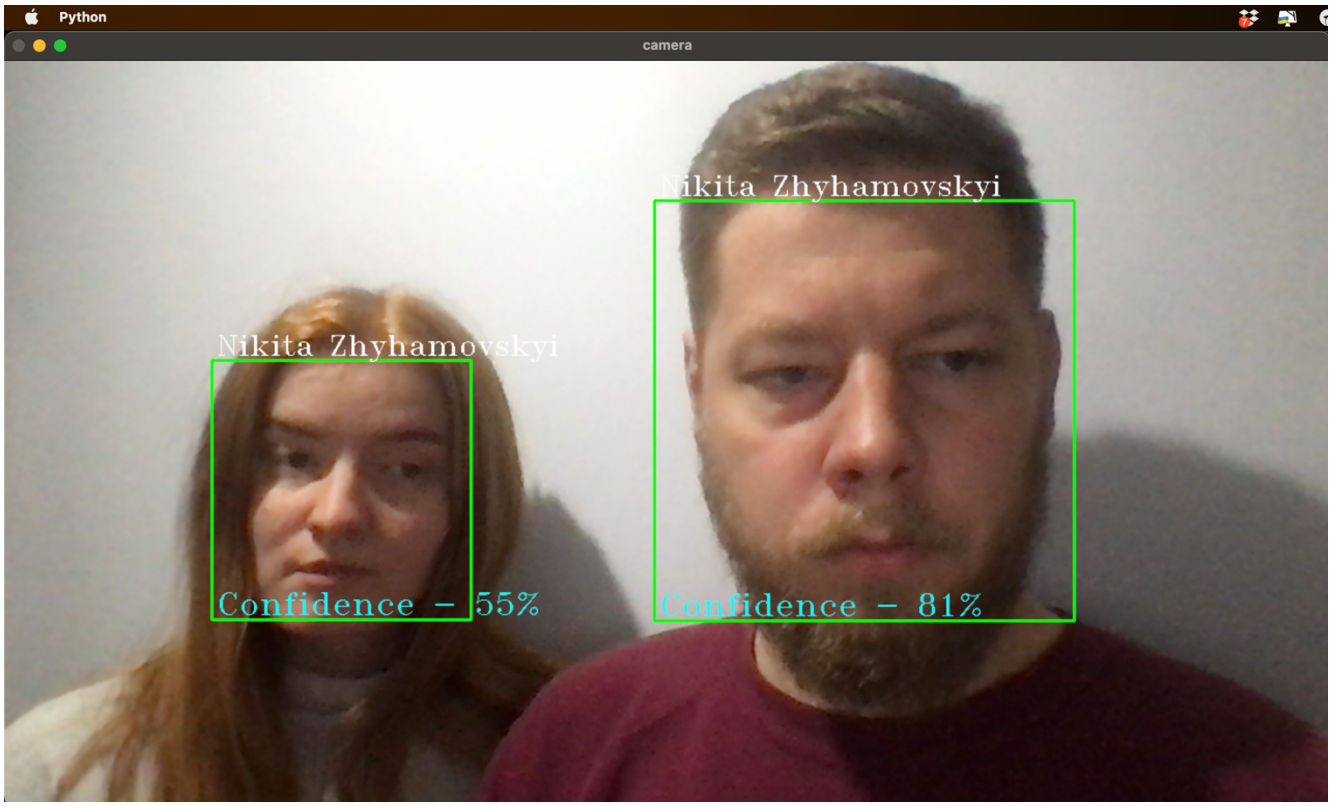

Рис 5.12 – Результат роботи системи до оптимізації

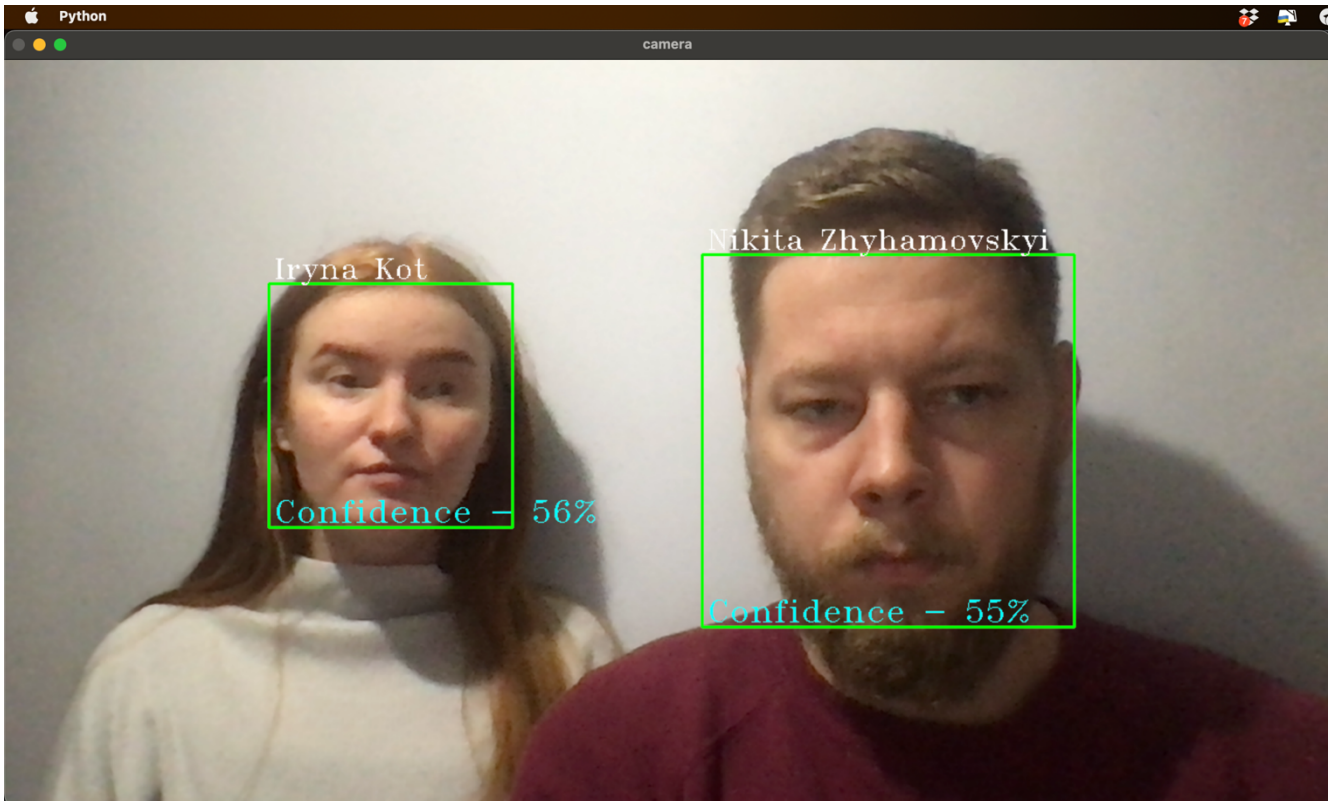

Рис 5.13 – Результат роботи системи після оптимізації

ВИСНОВКИ

Для проведених експериментів з виявлення та розпізнавання обличчя встановлено, що комбінація алгоритму Віоли-Джонса та каскадних класифікаторів ознак Хаара дає задовільний результат, і на цій основі можна будувати сучасні моделі розпізнавання обличчя.

Це має безліч сфер застосувань, в тому числі й у сфері безпеки. Якщо взяти звичайну IT компанію, та застосувати розроблену систему до неї з точки зору безпеки, то можна подумати про великий спектр можливостей, що дає ця система. Якщо кожен співробітник пройде сканування обличчя у системі, його обличчя з ідентифікатором та іменем одразу потрапить до словника або певної бази даних співробітників. У такому разі, якщо встановити камери у всьому приміщенні офісу, то можна з вхідних зображень/кадрів відео потоку в реальному часі одразу розпізнавати кожного співробітника, та у випадку, якщо система не змогла розпізнати обличчя, можна занести це обличчя до бази, як невідоме. Далі служба безпеки офісу може переглянути цю базу з нерозпізнаних облич та проаналізувати її на предмет небажаних людей, або невідомих людей.

Важливо пам’ятати, що кількість зразків зображень облич на вході повинна бути максимально-можливою. Чим більше є зразків, тим краще система буде давати результат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Нейромережа [Електронний ресурс] – 2019 – Режим доступу: https://termin.in.ua/neyromerezha
2. Вчені записки ТНУ [Електронний ресурс] – 2018 – Режим доступу: http://www.tech.vernadskyjournals.in.ua/journals/2018/5_2018/part_2/17.pdf
3. AdaBoost Illustrated [Electronic resource] – 2023 – Mode of access: https://medium.com/ai-made-simple/adaboost-illustrated-3084183a2086
4. Guide on Support Vector Machine [Electronic resource] – 2023 – Mode of access: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
5. What are Haar Features used in Face Detection? [Electronic resource] – 2019 – Mode of access: https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b
6. OpenCV eigenfaces for face recognition [Electronic resource] – 2021 – Mode of access: https://pyimagesearch.com/2021/05/10/opencv-eigenfaces-for-face-recognition/
7. Local Binary Pattern and its variants [Electronic resource] – 2019 – Mode of access: https://univ-rennes.hal.science/hal-02924534/document
8. Local Binary Patterns [Electronic resource] – 2022 – Mode of access: https://en.wikipedia.org/wiki/Local_binary_patterns
9. What are Haar Features used in Face Detection? [Electronic resource] – 2019 – Mode of access: https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b
10. A simple introduction on K-Nearest algorithm [Electronic resource] – 2019 – Mode of access: https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e

ДОДАТОК А

Модуль словника облич

```
# створення класу для створення облич
class face:
  # конструктор класу
  def __init__(self, id: int, name: str):
    # присвоювання переданого ідентифікатора
    self.id = id
    # присвоювання переданого імені
    self.name = name
# масив зі збережних облич
faces: list[face] = []
# функція додавання нового обличчя
def addNewFace(id: int, name: str):
  # перевірка, якщо обличчя з таким ідентифікатором вже було додано до масиву облич
  if getFace(id):
    # повернення False якщо обличчя не було додано но масиву
    return False
  # перевірка, якщо обличчя з таким ідентифікатором ще немає
  else:
    # додавання нового екземпляру обличчя до масиву облич
    faces.append(face(id, name))
    # повернення True якщо обличчя було додано
    return True
# функція пошуку обличчя по ідентифікатору у масиві облич
def getFace(id: int):
  # перебірка облич у масиві облич
  for f in faces:
    # перевірка на співпадіння обличчя по ідентифікатору
    if f.id == id:
      # повернення шуканого обличчя
      return f
```

```
  # обробка кейсу, якщо обличчя не було знайдено
  else:
    # повернення None у випадку, якщо обличчя з таким ідентифікатором не було знайдено
    return None
# функція видалення обличчя по ідентифікатору у масиві облич
def removeFace(id: int):
  # пошук обличчя за ідентифікатором
  f = getFace(id)
  # перевірка, чи є обличчя екземпляром масиву
  if isinstance(f, face):
    # видалення обличчя з масиву облич
    faces.remove(f)
    # повернення True у випадку, якщо обличчя з таким ідентифікатором було видалено
    return True
  # перевірка, чи обличчя не є екземпляром масиву
  else:
    # повернення False у випадку, якщо обличчя з таким ідентифікатором не було знайдено
    return False
```

ДОДАТОК Б

Модуль сканування нового обличя

```
# імпортуємо OpenCV для обробки зображень
import cv2
# імпортуємо для роботи з файлами/директоріями
import os
# імпортуємо модуль словника облич
import dictionary
# імпортуємо модуль для тренування моделі
import training
# імпортуємо модуль конфігурацій
import configuration
# імпортуємо модуль для роботи з часом
import time
# функція для сканування обличчя
def runScanFace():
  # інфо лог
  print("\n [INFO] Ініціалізація захоплення обличчя")
  # запуск захоплення обличчя
  cam = cv2.VideoCapture(0)
  # задання ширини відео
  cam.set(3, 1000)
  # задання висоти відео
  cam.set(4, 1000)
  # виявлення об'єктів у відео потоці використовуючи ознаки Хаара
  face_detector = cv2.CascadeClassifier(configuration.VJConfig.classifier)
  # задання унікального ідентифікатора для кожної людини (обличчя)
  face_id = int(
    input("\n [INPUT] Задайте ідентифікатор (id: int) цього користувача -> ")
  )
  # задання імені для кожної людини (обличчя)
  face_name = input("\n [INPUT] Задайте ім'я (name: str) цього користувача -> ")
```

```
# каунтер для підрахунку облич індивідуальної вибірки
face_count = 0
# максимальна кількість ітерацій (скільки фото вибірок потрібно зробити)
max_iterations = 100
# інфо лог
print("\n [INFO] Завантаження камери ...")
# починаємо відлік часу
start_time_execution = time.time()
# початок циклу
while True:
  # захоплення кадру відео
  ret, img = cam.read()
  # перевірка, чи доступна зараз камера
  if ret:
    # перетворення кадру на градації сірого
    image_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # виявлення обличчя за допомогою алгоритму Віоли-Джонса
    faces = face_detector.detectMultiScale(
      # задання зображення
      image = image_gray,
      # задання невеликого кроку для зміни розміру,
      # для збільшення ймовірності того,
      # що буде знайдено відповідний розмір моделі для виявлення
      scaleFactor = configuration.VJConfig.scaleFactor,
      # задання кількості сусідів, для якіснішого виявлення з меншою кількістю виявлень
      minNeighbors = configuration.VJConfig.minNeighbors,
    )
    # цикл для прямокутників
    for x, y, w, h in faces:
      # обрізання зображення до прямокутника
      cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

```
      # збільшення каунтеру підрахунку облич
      face_count += 1
      # збереження захопленого зображення у папку dataset
      cv2.imwrite(
        # передача сформованого імені файлу для збереження у наш dataset
        filename = "dataset/{0}.{1}.{2}.jpg".format(
          face_name, str(face_id), str(face_count)
        ),
        # передача зображення
        img = image_gray[y : y + h, x : x + w],
      )
      # відображення кадру з прямокутником на обличчі
      cv2.imshow("image", img)
    # припинення відео при натисканні ESC
    if (cv2.waitKey(10) & 0xFF) == 27:
      # вихід з циклу
      break
    # припинення відео при досягненні максимально потрібної кількості операцій
    elif face_count >= max_iterations:
      # інфо лог
      print("\n [INFO] Завершення захоплення обличчя")
      # вихід з циклу
      break
  # перевірка, якщо камера не доступна
  else:
    # інфо лог
    print("\n [WARN] Камера недоступна, перепідключення...")
# закінчуємо відлік часу
end_time_execution = time.time()
# додавання нового обличчя до словника (масиву облич)
dictionary.addNewFace(face_id, face_name)
```

```
  # інфо лог
  print("\n [INFO] Припинення відео. ", str(end_time_execution - start_time_execution), 's')
  # припинення відео
  cam.release()
  # інфо лог
  print("\n [INFO] Закриття вікна")
  # закриття всіх відкритих вікон
  cv2.destroyAllWindows()
  # інфо лог
  print("\n [INFO] Завершення виконання скрипта зі створення датасету користувача")
  # завит у користувача, чи потрібно додати нове обличчя
  additionalFace = input(
    "\n [INPUT] Потрібно додати додаткового користувача? (y/n) -> "
  )
  # перевірка на випадок позитивної відповіді
  if additionalFace == "y":
    # рекурсивний виклик функції сканування нового обличчя
    runScanFace()
    # завершення роботи поточного екземпляру функції
    return
  # перевірка на випадок негативної відповіді
  else:
    # інфо лог
    print("Користувачів було додано -", str(len(dictionary.faces)))
    # виклик функціїї для тренування моделі
    training.runTrainingModel()
# runScanFace()
```

ДОДАТОК В

Модуль тренування моделі

```
# імпортуємо OpenCV для обробки зображень
import cv2
# імпортуємо numpy для матричних обчислень
import numpy as np
# імпортуємо Python Image Library для роботи з зображеннями
from PIL import Image
# імпортуємо для роботи з файлами/директоріями
import os
# імпортуємо модуль словника облич
import dictionary
# імпортуємо модуль розпізнавання облич
import recognition
# імпортуємо модуль для роботи з часом
import time
# імпортуємо модуль конфігурацій
import configuration
# функція для навчання моделі
def runTrainingModel():
  # шлях до директорії з зображеннями
  path = "dataset"
  # інфо лог
  print("\n [INFO] Створення LBPH для розпізнавання облич")
  # створення LBPH моделі для роспізнавання облич
  recognizer = cv2.face.LBPHFaceRecognizer_create(
    # радіус, який використовується для будування кругового локального бінарного шаблону
    radius = configuration.LBPConfig.radius,
    # кількість точок вибірки для будування кругового бінарного шаблону
    neighbors = configuration.LBPConfig.neighbors,
    # кількість комірок по горизонталі
    grid_x = configuration.LBPConfig.grid_x,
```

```
    # кількість комірок по вертикалі
    grid_y = configuration.LBPConfig.grid_y,
    # поріг, який застосовується у прогнозі
    threshold = configuration.LBPConfig.threshold,
  )  # type: ignore
  # інфо лог
  print(
    "\n [INFO] Використання попередньо створеної тренувальної Haar Cascade моделі для виявлення облич"
  )
  # створення класифікатору за допомогою ознак Хаара
  detector = cv2.CascadeClassifier(configuration.VJConfig.classifier)
  # функція для перевірки чи файл зображенням
  def is_image(path: str):
    # розбивка шляху файла по сепаратору
    dotted_parts = os.path.split(path)[-1].split(".")
    # перевірка, чи містить файл всі необхідні значення
    if len(dotted_parts) < 4:
      # повернення False, у випадку якщо ні
      return False
    # отримання розширення файлу
    extension = dotted_parts[3]
    # перевірка, чи озширення є розширенням зображення
    if extension == 'jpg':
      # повернення True, у випадку якщо так
      return True
    # перевірка, якщо розширення не є зображенням
    else:
      # повернення False, у випадку якщо ні
      return False
  # початок відліку часу
  start_time_execution = time.time()
```

```
# функція для отримання зображень та даних міток
def getImagesAndLabels(path):
  # інфо лог
  print("\n [INFO] Отримання шляхів до зображень")
  # отримання всіх шляхів до зображень
  imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
  # фільтрація масиву шляхів щоб отримати тільки зображення
  filteredImagePaths = list(filter(is_image, imagePaths))
  # ініціалізація зразків обличчя
  faceSamples = []
  # ініціалізація ідентифікаторів
  ids: list[int] = []
  # ініціалізація імен
  names: list[str] = []
  # інфо лог
  print("\n [INFO] Конвертація зображень")
  # цикл для перебору шляхів до зображень
  for imagePath in filteredImagePaths:
    # отримання зображення та конвертація до градацій сірого
    PIL_img = Image.open(imagePath).convert("L")
    # створення numpy масиву картинки
    img_numpy = np.array(PIL_img, "uint8")
    # отримання ідентифікатору користувача зі шляху зображення
    id = int(os.path.split(imagePath)[-1].split(".")[1])
    # отримання імені користувача зі шляху зображення
    name = str(os.path.split(imagePath)[-1].split(".")[0])
    # виявлення обличчя за допомогою алгоритму Віоли-Джонса
    faces = detector.detectMultiScale(
      # задання зображення
      image = img_numpy,
      # задання невеликого кроку для зміни розміру,
```

```
        # для збільшення ймовірності того,
        # що буде знайдено відповідний розмір моделі для виявлення
        scaleFactor = configuration.VJConfig.scaleFactor,
        # задання кількості сусідів, для якіснішого виявлення з меншою кількістю виявлень
        minNeighbors = configuration.VJConfig.minNeighbors,
      )
      # цикл для прямокутників
      for x, y, w, h in faces:
        # додавання зображення до масиву зразків облич
        faceSamples.append(img_numpy[y : y + h, x : x + w])
        # додавання ідентифікатору до масиву ідентифікаторів
        ids.append(id)
        # додавання ім'я до масиву імен
        names.append(name)
    # вертаємо масив зразків облич та масив ідентифікаторів
    return faceSamples, ids, names
# інфо лог
print("\n [INFO] Тренування моделі на зразках зображень облич")
# отримання зразків облич та масиву ідентифікаторів
faces, ids, names = getImagesAndLabels(path)
# тренування моделі використовуючі отримані дані (зразки облич та ідентифікатори)
recognizer.train(faces, np.array(ids))
# збереження моделі
recognizer.write("trainer/trainer.yml")
# закінчення відліку часу
end_time_execution = time.time()
# кількість унікальних облич
unique_faces = len(np.unique(ids))
# функція для оновлення словника облич (масиву облич)
def updateDictionary():
    # перебір ідентифікаторів у циклі
```

```
    for currentID in ids:
      # пошук обличчя у словнику збережених облич
      faceFromDictionary = dictionary.getFace(currentID)
      # обробка кейсу, якщо обличчя з таким ідентифікатором вже є у словнику
      if isinstance(faceFromDictionary, dictionary.face):
        # вихід з поточної ітерації циклу
        break
      # обробка кейсу, якщо обличчя з таким ідентифікатором немає у словнику
      else:
        # отримання імені користувача по ідентифікатору
        currentName = names[currentID]
        # додавання нового обличчя до словника (масиву облич)
        dictionary.addNewFace(currentID, currentName)
  # виклик функції для оновлення словника облич (масиву облич) - синхронзація зі словником
  updateDictionary()
  # інфо лог
  print(
    "\n [INFO] Тренування моделі на {0} облич завершено. {1} s".format(unique_faces,
str(end_time_execution - start_time_execution))
  )
  # запит у користувача, чи потрібно запускати скрипт для розпізнавання обличчя
  shouldContinue = input(
    "\n [INPUT] Запустити скрипт розпізнавання обличчя? (y/n) -> "
  )
  # перевірка на випадок позитивної відповіді
  if shouldContinue == "y":
    # виклик функції для розпізнавання обличчя
    recognition.runFaceRecognition()
  # перевірка на випадок негативної відповіді
  else:
    # інфо лог
    return print("\n [INFO] Завершення роботи програми")
```

ДОДАТОК Г

Модуль розпізнання обличчя

```
# імпортуємо OpenCV для обробки зображень
import cv2
# імпортуємо numpy для матричних обчислень
import numpy as np
# імпортуємо для роботи з файлами/директоріями
import os
# імпортуємо модуль словника облич
import dictionary
# імпортуємо модуль конфігурацій
import configuration
# мінімальний задовільний відсоток впевненості
minSatisfyConfidence = 50
# функція для розпізнавання обличчя
def runFaceRecognition():
  # створення LBPH для роспізнавання облич
  recognizer = cv2.face.LBPHFaceRecognizer_create(
    # радіус, який використовується для будування кругового локального бінарного шаблону
    radius = configuration.LBPConfig.radius,
    # кількість точок вибірки для будування кругового бінарного шаблону
    neighbors = configuration.LBPConfig.neighbors,
    # кількість комірок по горизонталі
    grid_x = configuration.LBPConfig.grid_x,
    # кількість комірок по вертикалі
    grid_y = configuration.LBPConfig.grid_y,
    # поріг, який застосовується у прогнозі
    threshold = configuration.LBPConfig.threshold,
  )  # type: ignore
  # завантаження тренованої моделі
  recognizer.read("trainer/trainer.yml")
  # створення класифікатору за допомогою ознак Хаара
```

```
faceCascade = cv2.CascadeClassifier(configuration.VJConfig.classifier)
# задання шрифту
font = cv2.FONT_HERSHEY_TRIPLEX
# запуск захоплення обличчя
cam = cv2.VideoCapture(0)
# задання ширини відео
cam.set(3, 1000)
# задання висоти відео
cam.set(4, 1000)
# визначення мінімальної ширини вікна, яке буде розпізнано як обличчя
minWidth = 0.1 * cam.get(3)
# визначення мінімальної висоти вікна, яке буде розпізнано як обличчя
minHeight = 0.1 * cam.get(4)
# початок циклу
while True:
 # захоплення кадру відео
 ret, img = cam.read()
 # перевірка, чи доступна зараз камера
 if ret:
  # перетворення кадру на градації сірого
  image_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # виявлення обличчя за допомогою алгоритму Віоли-Джонса
  faces = faceCascade.detectMultiScale(
   # задання зображення
   image = image_gray,
   # задання невеликого кроку для зміни розміру,
   # для збільшення ймовірності того,
   # що буде знайдено відповідний розмір моделі для виявлення
   scaleFactor = configuration.VJConfig.scaleFactor,
   # задання кількості сусідів, для якіснішого виявлення з меншою кількістю виявлень
   minNeighbors = configuration.VJConfig.minNeighbors,
```

```
        # визначення мінімального розміру вікна, яке буде розпізнано як обличчя
        minSize = (int(minWidth), int(minHeight)),
    )
    # цикл для кожного обличчя
    for x, y, w, h in faces:
        # обрізання зображення до прямокутника
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        # розпізнавання, до якого ідентифікатору належить обличчя та впевненість у співпаданні
        id, confidence = recognizer.predict(image_gray[y : y + h, x : x + w])
        # форматований відсоток співпадіння розпізнаного обличчя
        formattedConfidence = round(100 - confidence)
        # заголовок за замовчуванням
        displayTitle = ""
        # форматований заголовок впевненості у співпадінні розпізнаного обличчя
        displayConfidence = "Confidence - {0}%".format(formattedConfidence)
        # перевірка, якщо відсоток впевненості більше ніж мінімальний задовільний відсоток впевненості
        if formattedConfidence > minSatisfyConfidence:
            # отримання повного екземляру людини по розпізнаному обличчю за ідентифікатором
            face = dictionary.getFace(int(id))
            # перевірка, якщо обличчя знайдене у словнику
            if isinstance(face, dictionary.face):
                # форматований фінальний заголовок з інформацією про людину
                displayTitle = face.name
            # перевірка, якщо обличчя не знайдене у словнику
            else:
                # форматований фінальний заголовок, що обличчя не розпізнано
                displayTitle = "Face didn't recognized"
        # перевірка, якщо відсоток впевненості менше ніж мінімальний задовільний відсоток впевненості
        else:
            # форматований фінальний заголовок, що обличчя не розпізнано
```

```
  displayTitle = "Face didn't recognized"
# відображення фінального заголовку з інформацією про розпізнану людину
cv2.putText(
  # задання зображення
  img = img,
  # задання самого заголовку
  text = str(displayTitle),
  # задання координат заголовку
  org = (x + 5, y - 5),
  # задання шрифту
  fontFace = font,
  # задання розміру шрифту
  fontScale = 0.9,
  # задання кольору шрифту
  color = (255, 255, 255),
  # задання товщини шрифту
  thickness = 1
)
# відображення фінального заголовку впевненості у співпадінні розпізнаного обличчя
cv2.putText(
  # задання зображення
  img = img,
  # задання самого заголовку
  text = str(displayConfidence),
  # задання координат заголовку
  org = (x + 5, y + h - 5),
  # задання шрифту
  fontFace = font,
  # задання розміру шрифту
  fontScale = 1,
  # задання кольору шрифту
```

```
        color = (255, 255, 0),
        #задання товщини шрифту
        thickness = 1,
      )
    # відображення кадру з прямокутником на обличчі
    cv2.imshow("camera", img)
    # припинення відео при натисканні ESC
    if (cv2.waitKey(10) & 0xFF) == 27:
      # вихід з циклу
      break
# інфо лог
print("\n [INFO] Припинення відео")
# припинення відео
cam.release()
# інфо лог
print("\n [INFO] Закриття вікна")
# закриття всіх відкритих вікон
cv2.destroyAllWindows()
# інфо лог
print("\n [INFO] Завершення виконання скрипта")
```

ДОДАТОК Ґ

Модуль конфігурацій

```
# об'єкт конфігурацій для застосування у будуванні локального бінарного шаблону
class LBPConfig:
  # радіус, який використовується для будування кругового локального бінарного шаблону
  radius = 1
  # [2] - кількість точок вибірки для будування кругового бінарного шаблону
  neighbors = 8
  # кількість комірок по горизонталі
  grid_x = 8
  # кількість комірок по вертикалі
  grid_y = 8
  # [3] - поріг, який застосовується у прогнозі
  threshold = 123.0
# об'єкт конфігурацій для застосування у алгоритмі Віоли-Джонса
class VJConfig:
  # [1] - шляху до обраного класифікатору ознак Хаара
  classifier = 'classifiers/haarcascade_frontalface_alt2.xml'
  # [4] - задання невеликого кроку для зміни розміру
  scaleFactor = 1.3
  # [5] - задання кількості сусідів, для якіснішого виявлення з меншою кількістю виявлень
  minNeighbors = 5
```