АНОТАЦІЯ

бакалаврської дипломної роботи Сарапулова Віктора Сергійовича

на тему «Eigenfaces алгоритми комп’ютерного зору з використанням OpenCV бібліотеки»

Дана дипломна робота присвячена розробці програмного продукту для систем автоматичного управління. Метою роботи є дослідження певного алгоритму комп’ютерного зору.

В роботі розглянуто Eigenface алгоритм, розроблено програми розпізнавання обличь у середовищі Matlab та на мові програмування C++, проведено тестування алгоритму. Наведено мінімальні та оптимальні параметри для запуску програмного забезпечення.

Загальний обсяг роботи: 74 сторінка, 23 рисунків, 6 таблиць, 9 бібліографічних посилань.

Ключові слова: комп’ютерний зір, розпізнавання обличь, обробка зображень, біометрика, власні вектори.

АННОТАЦИЯ

бакалаврской дипломной работы Сарапулова Виктора Сергеевича

на тему «Eigenfaces алгоритмы компьютерного зрения с использованием OpenCV библиотеки»

Данная дипломная работа посвящена разработке программного продукта для систем автоматического управления. Целью работы является исследование определенного алгоритма компьютерного зрения.

В работе рассмотрен Eigenface алгоритм, разработана программа распознавания лиц в среде Matlab и на языке программирования C++, проведено тестирование алгоритма. Приведены минимальные и оптимальные параметры для запуска программного обеспечения.

Общий объем работы: 74 страница, 23 рисунков, 6 таблиц, 9 библиографических ссылок.

Ключевые слова: компьютерное зрение, распознавание лиц, обработка изображений, биометрика, собственные вектора.

ANNOTATION

a bachelor’s degree work of Viktor Sarapulov

entitled "Eigenfaces algorithms using OpenCV library"

This project is devoted to the research of software tools for automatic control systems. The aim is to research some algorithm of computer vision.

The project covers the following: basic Eigenface algorithm model; developed face recognition program in Matlab and C++; efficiency tests for Eigenface algorithm; minimal and optimal launch parameters for developed program.

Total volume of work: 74 pages, 23 figures, 6 tables, 9 bibliographic links.

Keywords: computer vision, face recognition, image processing, biometrics, eigen vectors.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	Програмне забезпечення
МГК	Метод головних компонент
КЗ	Комп’ютерний зір
ВВ	Власні вектори
ВЗ	Власні значення
НМ	Нейронні мережі

ВСТУП

Комп’ютерний зір - це вид діяльності, у якому для вилучення даних використовуються статичні методи та використовують моделі, що побудовані за допомогою геометрії, фізики та теорії навчання. Комп’ютерний зір використовується досить широко, як у відносно старих сферах (таких як управління мобільними роботами, військові додатки, промислові засоби стеження), так і у відносно нових сферах (взаємодія людина\комп’ютер, пошук зображення у бібліотеках, аналіз медичних зображень та реалістична передача змодельованих сцен у комп’ютерній графіці).

Відмінна риса комп’ютерного зору - це вилучення опису зображення або послідовності зображень. Наприклад, такий аспект комп’ютерного зору, як визначення структури по руху, дозволяє із серії зображень отримати представлення щодо того, як рухається камера і що зображено на малюнку. В індустрії розваг подібні методи використовуються для відсіювання руху та побудови тривимірних комп’ютерних моделей будівлі зі збереженням їх структури. За допомогою невеликої кількості фотографій можна отримати хороші, прості, точні та зручні моделі. Розглянемо іншу ситуацію: люди, що бажають контролювати працю мобільних роботів. У такій ситуації інформація щодо області, де працює робот, зазвичай не представляє жодного інтересу, важливо лише розташування робота у цій області. Таким чином, тут відсіюється інформація щодо структури та відстежується рух, що дозволяє визначити точне місцезнаходження робота.

Існує також цілий ряд інших важливих областей застосування комп’ютерного зору. Це, наприклад, робота із медичними зображеннями: створення програмних систем, що можуть покращувати набір зображень, визначати на них важливі моменти або події, або візуалізувати інформацію, отриману із зображення. Друга важлива область - це різні технічні перевірки, коли по зображенню об’єктів визначається, чи відповідають об’єкти специфікації. Третя сфера застосування комп’ютерного зору - інтерпретація

фотографій, зроблених зі супутника, як у військових цілях, так і у цивільних. Четверта сфера – це впорядковані та структуровані колекції картин. Обробка бібліотек статичних або рухомих зображень на сьогодення має ряд серйозних нерозв’язаних питань.

На даний момент комп’ютерний зір знаходиться у спеціальній точці свого розвитку. Не так давно з’явилася можливість створення практичних програм, що використовували б ідеї комп’ютерного зору. Швидкість сучасних цифрових пристроїв та можливість використання паралельних обчислень дають можливість реалізації багатьох алгоритмів роботи з бібліотеками цифрових зображень. Таким чином, проводити серйозні досліди та вирішувати численні повсякденні задачі тепер можна за допомогою методів комп’ютерного зору.[1]

Як технологічна дисципліна, комп'ютерний зір прагне застосувати свою теорію та модель до створення систем комп'ютерного бачення. Прикладами застосування таких систем можуть бути:

1) Системи управління процесами (промислові роботи, автономні транспортні засоби).

2) Системи відеоспостереження.

3) Системи організації інформації (наприклад, для індексації баз даних зображень).

4) Системи моделювання об'єктів або навколишнього середовища (аналіз медичних зображень, топографічне моделювання).

5) Системи взаємодії (наприклад, пристрої введення для системи людино-машинного взаємодії).

1. РОЗПІЗНАВАННЯ ОБЛИЧЬ

У даній главі представлено загальний опис задачі розпізнавання обличь, існуючих підвидів задач. Наведено приклади застосування різних алгоритмів для розпізнавання обличь.

1.1 Класифікація розв’язуваних задач у розпізнаванні людини по зображенню обличчя

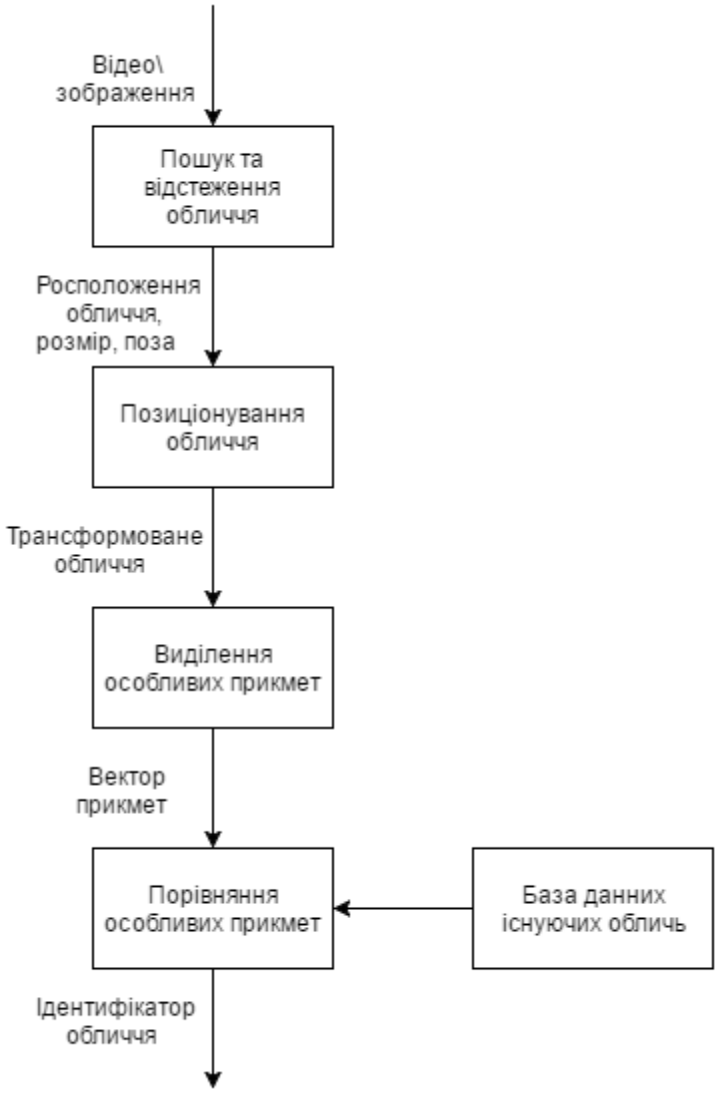

Рисунок 1.1 – Загальний алгоритм розпізнавання обличь.

В наш час все більшого розповсюдження отримують біометричні системи ідентифікації людини. Традиційні системи ідентифікації потребують знання

пароля, наявності ключа, ідентифікаційної картки або іншого ідентифікаційного документа, що можна забути, загубити або підробити. На відміну від них, біометричні системи основані на унікальності біологічних характеристиках, що важко підробити та які однозначно визначають конкретну людину. До таких характеристик відносять відбитки пальців, форма долоні, візерунок радужної оболонки, зображення сітківки ока. Обличчя, голос і запах кожної людини також індивідуальні.

Розпізнавання людини по зображенню обличчя виділяється серед біометричних систем тим, що, по-перше, не потребує спеціального коштовного обладнання. Для більшості додатків достатньо персонального комп'ютера або звичайної відеокамери. По-друге, фізичний контакт людини з приладами відсутній. Не треба ні до чого торкатися або спеціально зупинятися і чекати реакції системи. У більшості випадків достатньо просто пройти повз або затриматись перед камерою на декілька секунд.

До недоліків розпізнавання людини по зображенню обличчя слід віднести те, зо сама по собі така система не дає 100%-ової гарантії ідентифікації. Там, де необхідна висока надійність, застосовують комбінування декількох біометричних методів.

На даний момент проблемі розпізнаванню людини по зображенню обличчя присвячена велика кількість наукових робіт, однак в цілому вона ще далека від вирішення. Основні труднощі пов'язані з тим, щоб розпізнати людину по зображенню обличчя незалежно від зміни ракурсу та умов освітлення при зйомці, а також при різних змінах, що зв'язані зі зростом, зачіскою і т.д.

Розпізнавання зображень перетинаються з розпізнаванням образів. Такі задачі не мають точного аналітичного розв'язку. При цьому необхідне виділення ключових факторів, що характеризують зоровий образ, визначення відносної важливості факторів шляхом вибору їх вагових коефіцієнтів та врахування взаємозв'язків між ними. Спершу ці задачі вирішувались людиною-експертом вручну, шляхом експериментів, що займало багато часу та не

гарантувало якості. У нових методах виділення ключових ознак здійснюється шляхом автоматичного аналізу навчальної вибірки , але тим не менш більша частина інформації щодо ознак вводиться вручну. Для автоматичного застосування таких аналізаторів вибірка має бути досить великою та охоплювати усі можливі ситуації.

Нейронномережеві методи пропонують інший підхід до вирішення задач розпізнавання образів. Архітектура та функціонування нейронних мереж мають біологічні праобрази. Ваги нейронної мережі не вираховуються шляхом аналітичних рівнянь, а підбиваються різноманітними локальними методами при навчанні. Навчаються нейронні мережі на наборі навчальних прикладів. У процесі навчання НМ проходить автоматичне вилучення ключових факторів, визначення їх важливості та побудова взаємозв'язків між ними. Навчена НМ може успішно застосовувати отриманий досвід на невідомих образах за рахунок хороших узагальнюючих особливостей.

Таким чином використання НМ для задач розпізнавання людини по зображенню обличчя являється досить перспективним напрямком. Тим не менш існує також багато інших напрямків вирішення таких задач.

Задачі розпізнавання людини по зображенню обличчя поділяються на три великих класи:

- пошук у великих базах даних
- контроль доступу
- контроль фотографій у документах.

Вони поділяються як по вимогам, що висунуто до системи розпізнавання, так і по способам розв'язку і саме тому поділяються на окремі класи.

Вимоги також досить різноманітні, що застосовуються до помилок першого и другого роду для таких класів. Помилкою першого роду (тип I помилки, misdetection) називається ситуація, коли об'єкт заданого класу не розпізнається або пропускається системою. Помилка другого роду (тип II помилки, хибні тревоги) виникає, коли об'єкт заданого класу приймається за об'єкт другого класу.

Слід також зазначити різноманітні означення верифікацій і розпізнавання (ідентифікації). У задачі верифікації невідомий об’єкт заявляє, що він належить до певного відомого системі класу. Система підтверджує або спростовує цю заяву. У системах верифікації помилкою першого роду являється ситуація, коли об’єкт, що належить до відомого системі класу, приймається за об’єкт, що відноситься до невідомого системі класам і у доступі йому відмовлено. Помилка другого роду виникає, коли об’єкт невідомого класу приймається за об’єкт, що відноситься до відомим системі класам і йому дозвіл дозволено. При розпізнаванні необхідно віднести об’єкт до одного з N відомих класів або відати заключення про те, що даний об’єкт не належить до відомих класів.

1.1.1 Пошук зображень у великих базах даних

Порівняння типу «один до багатьох». Високі вимоги до помилки першого роду – система повинна знаходити зображення, що відповідає даній людині, не пропустивши жодного з таких зображень. При цьому допустимо, якщо у результуючій виборці буде присутня невелика кількість інших людей.

Зазвичай у великій базі (10τ-10^7зображень) необхідно знайти зображення, найбільш схоже на задане. Пошук має бути здійснено за певний час. Одним з рішенн даної проблеми є збереження у базі даних невеликих наборів завчасно вилучених ключових факторів, максимально характеризуючих зображення. При цьому вимоги до точності не так критичні, як у задачах контроля доступу і документного контролю.

До даного класу перш за все відносять метод головних компонент (метод «власних обличь», Eigenface). Коефіцієнти, отримані розкладанням вхідного зображення на головні компоненти, використовувались для порівняння зображень шляхом вирахування Евклідової відстані. Саме про цей метод піде мова у наступних розділах.

1.1.2 Задача контролю доступу

Порівняння типу «один до декількох». Критичними являються умови до помилок другого роду. Розпізнавальна система не повинна розпізнавати невідомих людей як відомих, можливо навіть за рахунок збільшення помилок першого роду (відмов у доступі відомим людям).

Маємо невеличку групу обличь (5-50 чоловік), яких система має розпізнавати по зображенню обличчя та відкривати їм доступ до певного приміщення. Людей, що не входять в цю групу, система не повинна пропускати. Можливі варіанти, коли необхідно встановити конкретну особу по зображенню обличчя. При цьому від системи вимагається висока достовірність розпізнавання, можливо навіть за рахунок збільшення кількості відмов на знайомі об’єкти.

У якості тренувальних зображень зазвичай для кожної людини доступні декілька зображень обличчя, що було отримано при різноманітних умовах. Це можуть бути, наприклад, зміни ракурсу, умов освітленості, зачіски, міміки, наявності або відсутності окулярів і т.д.

Система повинна працювати в реальному масштабі часу, а процес налаштування може займати більше часу і виконуватись попередньо. У процесі експлуатації система повинна підлаштовуватись і навчатись під нові вхідні зображення якомога швидше.

Обмежень на використовування різних методів тут нема, але всі методи збігаються у тому, що наявний навчальний набір зображень обличь заданої групи людей (допускаються різні умови зйомки). До цього набору система звертається у процесі розпізнавання або налаштування на нього у процесі навчання.

Одним з найвідоміших підходів до рішення такої задачі є використання НМ, які після навчання отримують хорошу узагальнюючу здібність.[2]

1.1.3 Задача контролю фотографій у документах

Порівняння типу «один до одного». Формулювати вимоги до помилок першого и другого роду як до системи верифікації або розпізнавання тут буде некоректним, адже система розпізнавання ніколи не мала справу з поступаючими на вхід класами. Але бажано, щоб система не здійснювала помилок при порівнянні.

Необхідно порівняти зображення обличчя людини, що було отримано у даний момент із фотографією з певного документа. Системі необхідно дати відповідь на питання, чи належать ці два обличчя одній і тій самій людині. Даний клас задач найбільш складний, адже, по-перше, система ніколи раніше не працювала із зображенням обличчя даної людини. Система порівнює завжди відмінні зображення, облік усіх можливих відмінностей у процесі навчання або налаштування системи досить складним.

По-друге, великий вплив здійснюють вікові та інші зміни обличчя. По-третє, якість і контраст відсканованої фотографії, як правило, гірше, ніж зображення обличчя, що було сфотографоване на камеру.

Більшість методів для даного класу задач неможливо використовувати без спеціальної адаптації.[2]

1.2 Методи розпізнавання обличь, що реалізовані у OpenCV

OpenCV — бібліотека комп’ютерного зору з відкритим кодом, що була спроектована компанією Itseez для полегшення дослідження алгоритмів комп’ютерного зору та розробки програм, що вирішуватимуть проблеми даної сфери. На сьогоднішній день являється найвідомішою і найрозвиненішою бібліотекою даної сфери. Її оновлення виходять кожні півроку.

Дана бібліотека являється відкритим інструментарієм з великою кількістю реалізованих методів та алгоритмів обробки зображень, відео, пошуку певних об’єктів на даній сцені і т.д. Щодо проблеми розпізнавання людського обличчя наявні наступні методи: Eigenface, Fisherface та локальні бінарні шаблони (LBP).[2]

1.3 Основні результати та висновки з розділу 1

1. Проведено аналіз проблеми розпізнавання обличь.
2. Розглянуто типи проблем розпізнавання обличь.
3. Розглянуто варіанти рішення проблем, що ускладнюють задачу розпізнавання обличь.
4. Розглянуто бібліотеку OpenCV в контексті задачі розпізнавання обличчя.

2. АЛГОРИТМ РОЗПІЗНАВАННЯ ОБЛИЧЬ EIGENFACE

У даній главі наводиться опис алгоритму комп'ютерного зору Eigenface: його побудова, архітектура, способи покращення алгоритму. Наводиться опис методу головних компонент, який є невід'ємною частиною Eigenface алгоритму[4].

2.1 Побудова алгоритму Eigenface

В онову алгоритму покладено використання фундаментальних статичних характеристик: середніх (математичне очікування) та коваріаційної матриці; використання метода головних компонент. Як і будь-який інший алгоритм сфери компьютерного навчання (machine learning), його необхідно спершу навчити первинній виборці (training set), яка складається з певної кількості зображення обличь, які ми хочемо розпізнавати. Як тільки модель стане навченою, слід подати на вхід деяке зображення і в результаті отримаємо відповідь на питання: якому зображенню із загальної виборки з найбільшою вірогідністю відповідає дане та чи належить дане зображення виборці взагалі.

Головний принцип алгоритму – представити вхідні зображення у вигляді однієї спільної матриці, що складатиметься із суми базисних компонент зображень:

$$\Phi_i = \sum_{j=1}^{K} w_j u_j \qquad (1)$$

Де Φ_i – відцентроване і-те зображення обличчя, w_j - ваги, u_j – власні вектори.

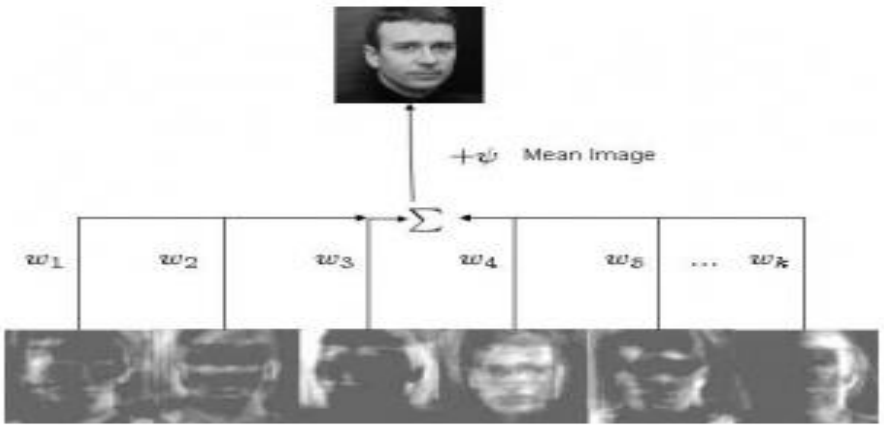

Рисунок 2.1 – Отримання вихідного обличчя.

На рисунку 2.1 ми отримуємо вихідне зображення обличчя зваженою сумою власних векторів та додаванням середнього значення. Інакше кажучи, маючи w та u ми можемо відтворити будь-яке вихідне зображення.

Навчальну виборку необхідно спроектувати у новий простір, де кожна розмірність даватиме певний вклад у спільне представлення. Метод головних компонент дозволяє знайти базис нового простору таким чином, щоб данні у ньому розташовувались, у певному сенсі, оптимально. Щоб це зрозуміти, достатньо уявити, що у новому просторі певні розмірності будуть нести більше спільної інформації, тоді як інші нестимуть лише специфічну інформацію. Як правило, розмірності більш високого порядку несуть набагато менше корисної інформації аніж перші розмірності, що відповідають найбільшим власним значенням. Залишаючи розмірності лише з корисною інформацією, ми отримуємо простір ознак, у якому кожне зображення вихідної виборки представлено у спільному вигляді. У цьому, дуже стисло, і полягає ідея алгоритму.

Надалі, маючи певні зображення, ми можемо відобразити його на створений раніше простір та визначити, до якого зображення навчальної виборки наш приклад відноситься більш за все. Якщо він знаходиться на досить великій відстані від всіх даних, тоді це зображення з найбільшою вірогідністю, не належить до зображень з нашої бази.

2.2 Архітектура алгоритму Eigenface

У цьому розділі розглянемо, як побудовано алгоритм, як проходить навчання алгоритму та як проходить розпізнавання обличчя.

2.2.1 Навчання алгоритму

Алгоритм Eigenface поділяється на певні етапи. На першому етапі всі зображення з бази необхідно представити у вигляді вектору. Так як на вхід подаються напівтонові зображення, значення вектору відповідатимуть

значенням рівня сірого кольору відповідних пікселів (див. рисунок 2.2). Так як зображення має розмір M*K, розмір вектору також дорівнюватиме M*K.

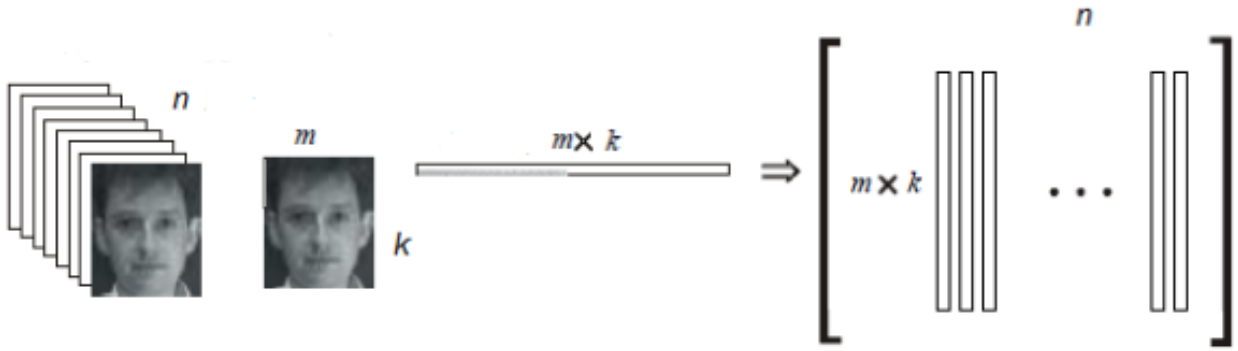

Рисунок 2.2 – Представлення вхідного зображення у векторному вигляді.[6]

Отримані вектори у сукупності утворюють матрицю розміром (M*K) x N, де N – кількість вхідних зображень.

Наступним етапом являється знаходження середнього значення матриці та його віднімання від кожного вектору:

$$\mu_i = \frac{1}{n} \sum_{j=1}^{n} x_{ij} \quad (2)$$

$$x_{ij} = x_{ij} - \mu_i \quad (3)$$

Ця процедура необхідна для того, щоб відкинути спільні ознаки обличь та залишити лише індивідуальні. Якщо зконвертувати вектор середніх значень у зображення, отримаємо «спільне обличчя»:

Рисунок 2.3 – Приклад «спільного обличчя».

Рисунок 2.4 – Віднімання «спільного обличчя». На другій стрічці ми чітко бачимо особливості кожного обличчя, що залишилися після віднімання.

Після отримання нормалізованої матриці, вираховуємо матрицю коваріацій C, власні вектори (eigenvectors – від того походить назва алгоритму) u та ваги w для кожного зображення з виборки:

$$C = X * X^T \quad (4)$$

$$w = A * u \quad (5)$$

Алгоритм знаходження власних векторів (в загальному випадку):

1. Складаємо характеристичний многочлен матриці:
$$\Delta_A(x) = \det(A - x * E) \quad (6)$$
2. Знаходимо всі відмінні корені характеристичного рівняння
$$\Delta_A(x) = 0 \quad (7)$$
3. Для кореня x – x_1 знайти фундаментальну систему $\varphi_1, \varphi_{2, \ldots,} \varphi_{n-r}$ рішень однорідної системи рівнянь
$$(A - x_1 * E) * x = 0, \text{ де } r = rang(A - x_1 * E) \quad (8)$$
4. Записати лінійно незалежні власні вектори матриці A, що відповідають власному значенню x_1: $s_1 = C_1\varphi_1$, $s_2 = C_2\varphi_2$, … , $s_{n-r} = C_{n-r}\varphi_{n-r}$, де $C_1, C_2, \ldots, C_{n-r}$ – відмінні від нуля довільні сталі.
5. Повторити пункти 3, 4 для інших власних значень.

Слід відзначити, що власні значення не мають принципово важливого значення для алгоритму, але вони можуть надати важливу інформацію. Власні значення показують дисперсію щодо кожної осі головних компонент (кожній осі відповідає одна розмірність у просторі). Таким чином ми можемо дізнатись, які з головних компонент являються важливими і формують основну частину інформації, а які – ні і якими можна знехтувати.

2.2.2 Розпізнавання обличчя

Після того, як алгоритм було навчено, можна подавати на вхід різні зображення, які в свою чергу будуть проходити майже той самий шлях, що і навчальні зображення: вирахування середнього значення, додавання до спільного простору компонент, знаходження ваги. Далі, коли ваги було розподілено, слід визначити, з яким об’єктом з виборки вхідний об’єкт найбільш схожий. Для цього використовується або пошук евклідової відстані, або відстані Махалонобіса. Той об’єкт, з яким відстань буде мінімальна, і вважається шуканим. Очевидно, що якщо подати на вхід обличчя, що не приймало участі при навчанні, алгоритм його «розпізнає». Для того, щоб алгоритм міг відсіювати зайві об’єкти, що не були у навчальній виборці, слід задати максимальне допустиме значення відстані.

2.3 Метод головних компонент

Метод головних компонент – один з основних способів зменшити розмірність даних, втративши найменшу кількість інформації. Був винайдений Карлом Пірсоном у 1901 році. Іноді МГК називають перетворенням Кархунена-Лоева або перетворенням Хотеллінга.

Даний метод складається з лінійного перетворення вхідного вектору x розмірності N у вихідний вектор y розмірності M, N > M. При цьому компоненти вектору y являються некорельованими і, відповідно, спільна дисперсія після перетворення лишається незмінною. Матриця X складається з усіх прикладів зображення навчального набору. Вирішив рівняння

$$\Lambda = \Phi^{T}\Sigma\Phi \quad (9)$$

отримуємо матрицю власних векторів Φ, де Σ – коваріаційна матриця для x, а Λ – діагональна матриця власних чисел. Вибрав з Φ підматрицю $\Phi_м$, що відповідає M найбільшим власним числам, отримаємо, що перетворення

$$y = \Psi_M^T \tilde{x} \quad (10),$$

де $$\tilde{x} = x - \bar{x} \quad (11)$$

- нормалізований вектор з нульовим математичним очікуванням,

характеризує велику частину спільної дисперсії та відображає найбільші існуючі зміни x. Вибір перших M головних компонент розбиває векторний простір на головний (власний) простір:

$$F = \{\Phi_i\}_{i=1}^{M} \quad (12),$$

що містить головні компоненти, та його ортогональне доповнення:

$$F = \{\Phi_i\}_{i=M+1}^{N} \quad (13).$$

В якості індикаторів приналежності у методі головних компонент використовують:

- відстань від образу аналізуємого зображення у власному просторі до еталонного образу;
- відстань від представлення аналізуємого зображення у просторі спостереження до проекції еталона у власному просторі.[3]

2.4 Основні результати та висновки до розділу 2

1. Представлена будова алгоритму Eigenface. Алгоритм складається з методу головних компонент та пошуку Евклідової відстані.

2. Представлена архітектура алгоритму, згідно з якою надалі відбуватиметься розробка програми. Алгоритм складається з двох частин: навчання моделі та розпізнавання зображення. Для розпізнавання необхідно отримати значення власних векторів, відсортованих за спаданням власних значень.

3. Метод головних компонент – метод, що допомагає знайти власні вектори та зменшує розмір вхідних даних за рахунок відкидання неважливої зайвої інформації.

3. РЕАЛІЗАЦІЯ ПРОГРАМИ РОЗПІЗНАВАННЯ ОБЛИЧЬ ЗА ДОПОМОГОЮ EIGENFACE АЛГОРИТМУ

У даному розділі буде використано алгоритм Eigenface для реалізації програми розпізнавання обличь. Програму буде реалізовано у середовищі Matlab та на мові програмування C++ з використанням бібліотеки OpenCV. Реалізація програми на Matlab необхідна для кращого засвоєння принципу роботи алгоритму а також для можливості оцінювання, як добре та зручно реалізовано даний алгоритм у бібліотеці OpenCV.[5]

3.1 Реалізація алгоритму у Matlab середовищі

3.1.1 Навчання алгоритму

На вхід функції load_images подаємо список адрес зображень at.txt (дивись додаток А). Після виклику функції, у терміналі з’являється інформація щодо кількості зображень та інформацію, на якому етапі проходить навчання в даний момент (дивись рисунок 3.1)

```
>> [Imgs,w,h,Vecs,Vals,Psi] = load_images('at.txt');
loading PGM file C:/test/at/s13/2.pgm
loading PGM file C:/test/at/s13/7.pgm
loading PGM file C:/test/at/s13/6.pgm

    • • •

loading PGM file C:/test/at/s38/8.pgm
loading PGM file C:/test/at/s38/1.pgm
Read 399 Imgs.
Computing average vector and vector differences from avg...
Calculating L=A'A
Calculating eigenVecs of L...
Sorting eVecs/Vals...
Computing eigenVecs of the real covariance matrix..
Warning: numvecs is 399; only 398 exist.
```

Рисунок 3.1 – Термінал Matlab після виклику функції load_images (навчання моделі)

Після закінчення роботи функції, всі дані зберігаються у відповідних їм змінних (дивись рисунок 3.2):

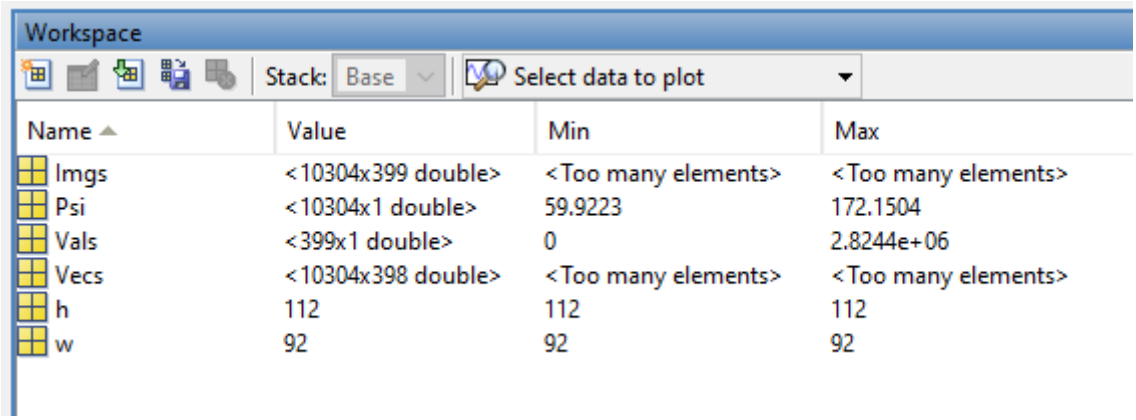

Рисунок 3.2 – Навчена модель, що необхідна для розпізнавання

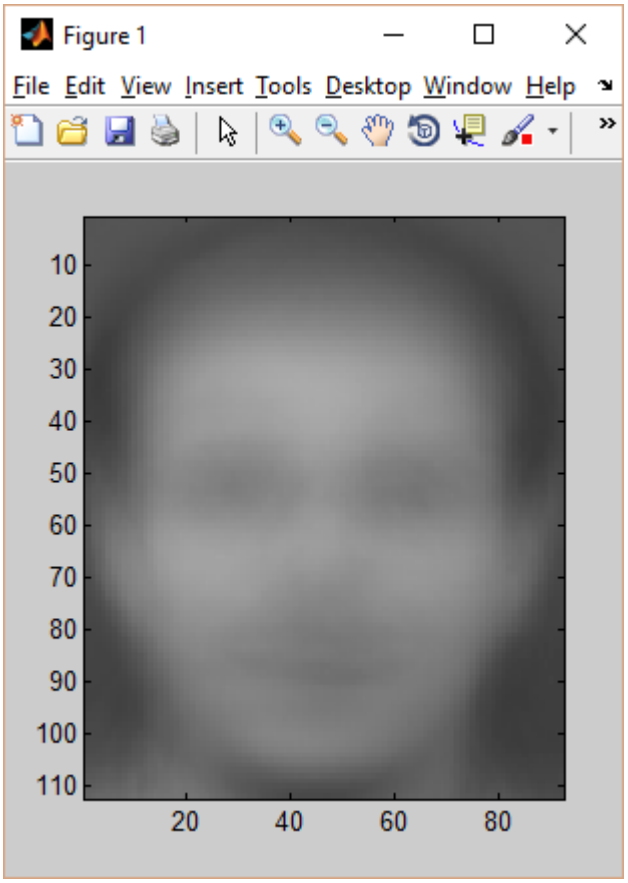

Рисунок 3.3 – Отримане «спільне обличчя»

Також, була розроблена додаткова функція spectrum, що візуалізувала значення власних значень. Вона була розроблена саме для того, щоб зрозуміти, якими головними компонентами ми можемо знехтувати.

Результат роботи цієї функції можемо побачити на рисунку 3.4

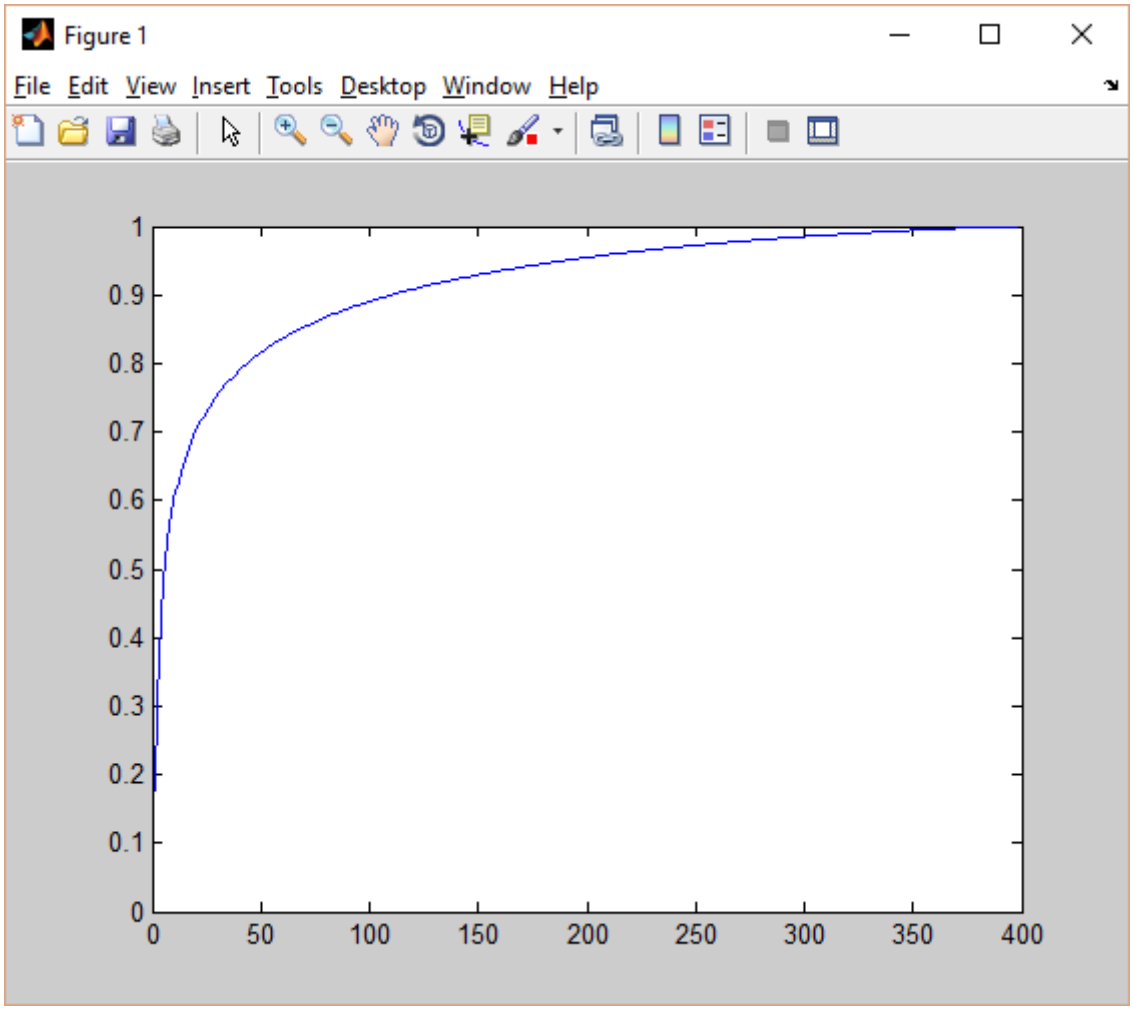

Рисунок 3.4 – Нормалізовані значення власних значень.

Згідно з графіком, ми чітко бачимо, що перші 150 компонентів містять 90% всієї інформації. Іншими компонентами можна знехтувати, адже на точність розпізнавання вони мають занадто низький вплив, а швидкість роботи алгоритму при цьому збільшиться.

3.2 Реалізація алгоритму на мові C++ з використанням OpenCV

Так як Eigenface алгоритм вбудовано в дану бібліотеку, основна частина коду прикрита від очей користувачів. З точки зору зручності використання, це надає велику перевагу над самостійною реалізацією, адже 1 стрічка коду даної

бібліотеки заміняє 10-15 стрічок коду власної реалізації. Однак є певні моменти, які стосуються саме внутрішньої частини алгоритму. Наприклад, якщо ми хочемо прискорити процес навчання нашої моделі, окрім як прописувати код вручну, іншого варіанту не буде. Розглянемо написання програми розпізнавання обличчя за допомогою OpenCV бібліотеки.

3.2.1 Навчання алгоритму

За допомогою бібліотеки OpenCV, навчити модель Eigenface можна використавши всього лише дві стрічки коду:

```
Ptr<FaceRecognizer> model = createEigenFaceRecognizer();

model->train(images, labels);
```

Перша стрічка ініціалізує модель, у відповідності з якою потім проходитиме перевірка вхідного зображення. В якості аргументів можна задати два параметри. Перший параметр відповідає за кількість зображень у виборці, до яких буде застосовано метод головних компонент. Другий – за максимальне значення відстані між значеннями вагів зображень. Він необхідний для того, щоб задавати певний критерій визначення належності вхідного зображення до навченої виборки.

Друга стрічка отримує вектори зображень та їх ідентифікаторів. Ідентифікатори необхідні для того, щоб програма під час розпізнавання могла вказати, до якого зображення вхідне зображення являється найбільш схожим.

Після проходження цих стрічок, отримуємо ось таку навчену модель алгоритму комп'ютерного зору (дивись рисунок 3.5):

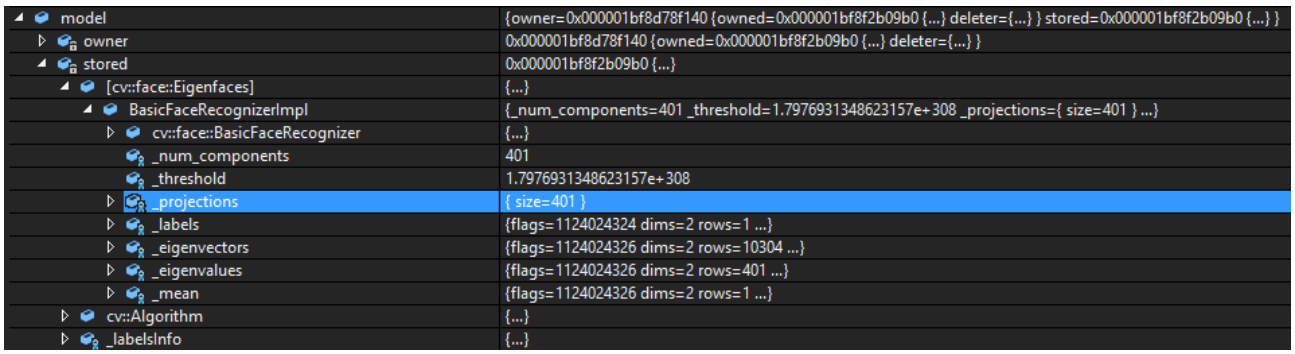

Рисунок 3.5 – Створена навчена модель алгоритму Eigenface на C++

3.2.2 Розпізнавання обличчя

Розпізнавання, як і навчання, також реалізується дуже просто. Однієї стрічки буде достатньо для визначення, якому зображенню відповідає дане:

```
int prediction = model->predict(face_resized);
```

У якості аргументу у функцію передається відцентроване обличчя людини.

3.3 Основні результати та висновки до розділу 3

1. Eigenface – досить простий у розумінні і опануванні алгоритм. Це було продемонтсровано на прикладі реалізації програми розпізнавання обличчя у Matlab середовищі. Завдяки засобам та інструментарію, доступному в цьому середовищі, можна було легко відстежити які з векторів слід враховувати при розпізнаванні, а якими можна було б знехтувати.

2. У випадку OpenCV, розробник не може у повному обсязі оцінити, наскільки складний або простий алгоритм за рахунок закритого коду. З іншого боку, тести (про які піде мова у наступних розділах) показали, що OpenCV надає все необхідне для того, щоб використовувати Eigenface (та багато інших алгоритмів) у повному обсязі. Найголовнішою перевагою OpenCV у контексті задачі розпізнавання є те, що за допомогою засобів бібліотеки можна автоматизувати процес знаходження обличчя на сцені.

4. ПОВЕДІНКА EIGENFACE АЛГОРИТМУ НА СТАТИЧНОМУ ЗОБРАЖЕННІ

В наш час існує неймовірна кількість різноманітних задач розпізнавання обличь. Деякі з них потребують ідентифікації на статичних зображеннях, а деякі – на відеопотоці. У даному розділі було розглянуто поведінку алгоритму на статичних зображеннях.

Для заощадження часу була запозичена база обличь “AT&T Laboratories, Cambridge”, що складається з 40 різних осіб. На кожну особу відведено по 10 фотографій, що були зроблені в однакових умовах освітлення і приміщення (дивись рисунок 4.1).

Під час навчання використовувалися не всі фотографії з бази. Деякі з них були вилучені спеціально для того, щоб побачити, як поводитиме себе алгоритм при різних умовах:

1. наявність ідентичної фотографії у базі (повний збіг);
2. наявність в базі даних даної людини, але відсутність вхідної фотографії у тренувальній виборці;
3. повна відсутність даної людини\об’єкту у базі;

На вхід подаватиметься певна картинка. На виході буде отримано наступну інформацію:

- Чи було розпізнано дану людину?
- Яка людина найбільше підходить, згідно з головними компонентами?
- Якою є мінімальна відстань між вагами?

На кожне тестування відводилося по 20 різних фотографій.

Рисунок 4.1 – Первинна навчальна виборка

4.1 Умова: вхідне фото присутнє у первинній виборці

Серед усіх тестів, всі зображення були розпізнані. Такий результат обумовлений тим, що головні компоненти вхідного зображення і відповідного зображення з виборки були однаковими і, в результаті, відстань між цими двома зображеннями дорівнювала нулю. Точність складала 100%. Інакше кажучу, система автоматичного керування може мати ідеальну систему розпізнавання особистості притримуючись двох простих правил: 1. камера буде мати постійне освітлення сталої скравості; 2. зображення певних розмірів буде подаватись на одне й те ж саме місце. Насправді, 2-ге правило не є критичним. Знову ж таки, можна використовувати спеціальні інструментарії, які самостійно оброблюватимуть фото, відцентровуватимуть їх і визначатимуть належність вхідного зображення до існуючих, як це було з OpenCV. Приклад тестування можна подивитися на рисунку 4.2:

Рисунок 4.2 – Результат тестування умови №1

4.2 Умова: вхідне фото належить людині, що є в базі, але дане фото відсутнє у виборці

Серед усіх тестів, всі зображення були розпізнані. Такий результат обумовлений тим, що головні компоненти вхідного зображення і відповідного зображення з виборки були однаковими і, в результаті, відстань між цими двома зображеннями дорівнювала нулю. На цей раз алгоритм видав один хибний результат. Точність алгоритму склала 95% (19 з 20). Щоб збільшити точність, необхідно збільшити кількість фотографій данної особи (якщо алгоритм відшукав людину в базі, але не ту, що треба), або змінити значення максимальної мінімальної відстані між вагами (якщо алгоритм не відшукав людину взагалі). Досить непоганий результат, враховуючи те, що його досі можна покращити. Приклад тетсування можна подивитися на рисунку 4.3:

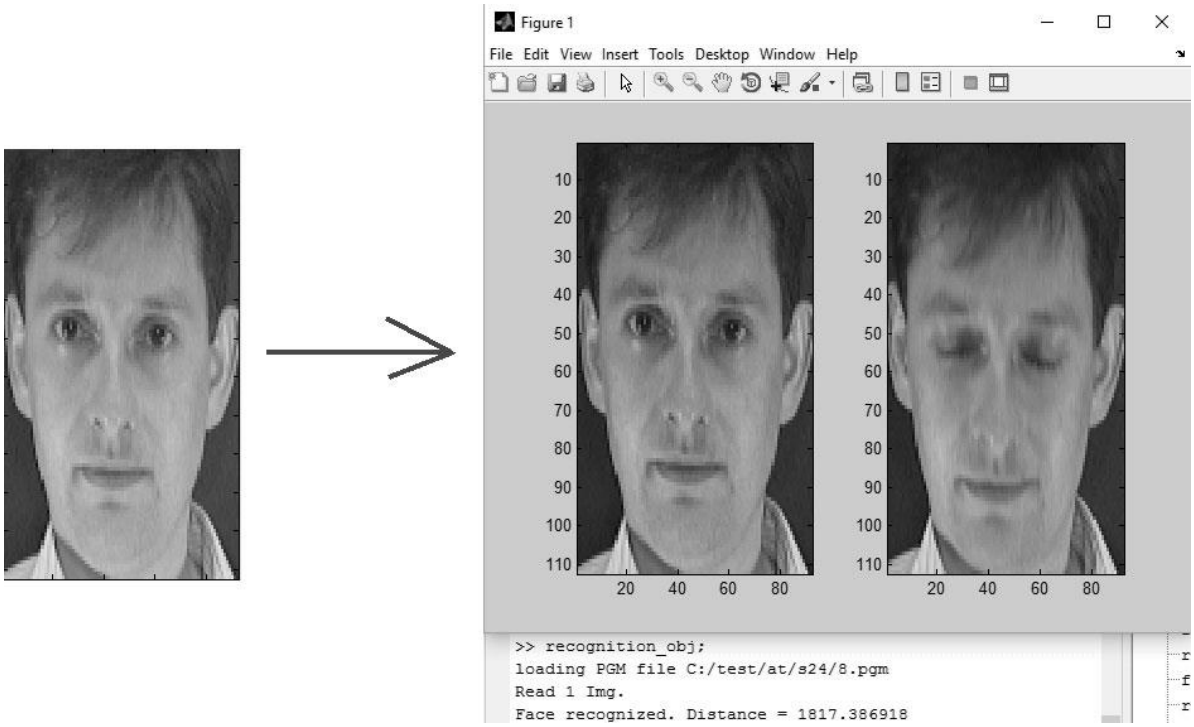

Рисунок 4.3 – Результат тестування умови №2

4.3 Умова: вхідне зображення зовсім відсутнє у виборці

На третій умові тестування результат погіршився. 16 з 20, що дає нам точність у 80%. Знову ж таки, це обумовлено тим, що мксимальне значення мінімальної відстані, можливо, було підібрано не досить точно. Слід збільшувати кількість тестування для того, щоб визначити максимальну допустиму відстань між векторами. Приклад тетсування можна подивитися на рисунку 4.4:

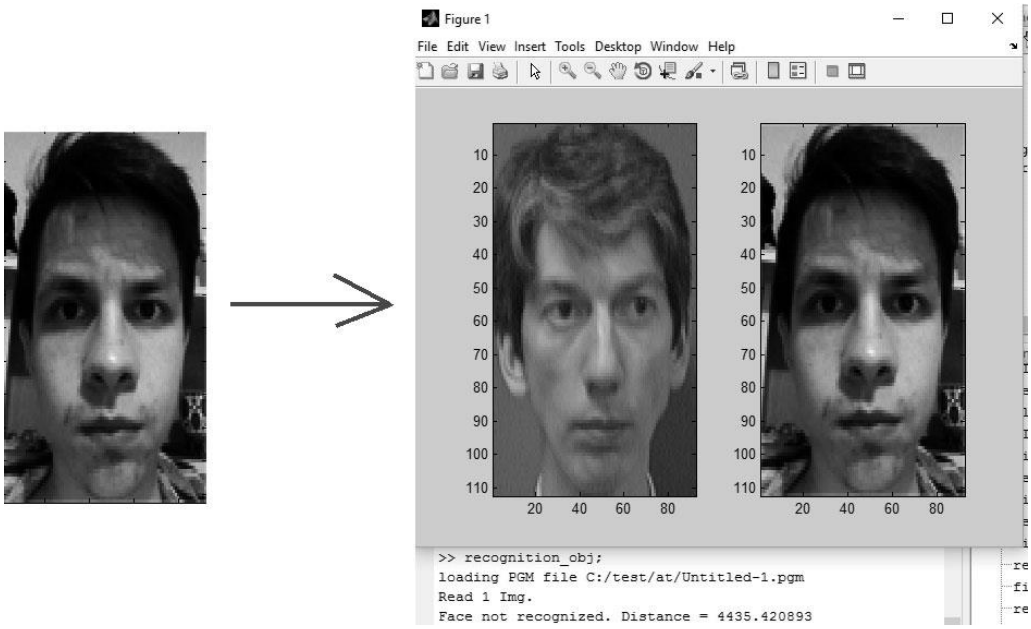

Рисунок 4.4 – Результат тестування умови №3

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

4.4 Перевірка на зіпсовану базу зображень

Також, на статиці було проведено такий тест: якщо у первинний набір додати зображення не людського обличчя, сама модель розпізнавання не зіпсується. Основа моделі будується на більшості зображень, які входять до

первинного набору. Після нормування та усереднення зображень тренувального набору, ми отримали таку модель:

Рисунок 4.5 – Модель роспізнавання, що утворилася із нелюдських зображень.

4.5 Висновки до розділу 4

Було проведено ряд тестів на статичних зображеннях. Після тестування виявилося, що чим більше різноманітних зображень однієї людини приймають участь у даному наборі, тим краще буде роспізнавання. Також, для кращого виявлення максимального значення мінімальної евклідової відстані, необхідно як можна більше тестів провести і вирахувати середнє мінімальне значення. Після ряду тестувань було виявленно, що вибірка, що містить помилкові зображення, не псує свою основу, а отже алгоритм Eigenface є досить стійким. Також було виявленно, що даний алгоритм чудово підійде не тільки для розпізнавання людських обличь, а й різних жестів, читанні по губам та іншим «группам», які можна певним чином візуально об’єднувати між собою.

5. ПОВЕДІНКА EIGENFACE АЛГОРИТМУ НА ВІДЕОПОТОЦІ

У даному розділі було розглянуто поведінку алгоритму на відеопотоці. Більшість задач ідентифікації людини пов’язані саме з відеопотоком.

Як вже було зазначено вище, на кожну людину відведено по 10 зображень різноманітних зображеннях. В якості моделі розпізнавання використовуватимо модель, що була навчена у попередньому розділі. Лістинг програми дивись у додатку Б.

5.1 Результаті роботи програми

Для відстежання обличчя були використані каскади Хаара, що імплементовані у бібліотеці OpenCV версії 3.0.

Під час тестування програми результати значно погіршилися. Після ряду тестувань, була виявлена помилка у реалізації, а саме: при центруванні знайденого зображення, вбудована функція пошуку робить вирізання обличчя за статичними розмірами, які не збігаються з розмірами зображень первинної виборки. Після корегування коду, результати покращилися. Тим не менш, тестування виявило, що Eigenface вразливий до невеличких відмінностей між первинною вибіркою та вхідним зображенням. Було проведено ряд тестувань із різними первинними виборками та вхідними зображеннями. На вхід зображення подавались під різним кутом нахилу, різним освітленням та різними особливостями обличчя людини. Було виявлено максимальне мінімальне значення відстані між власними векторами. Результати можна спостерігати на рисунках 5.1- 5.3

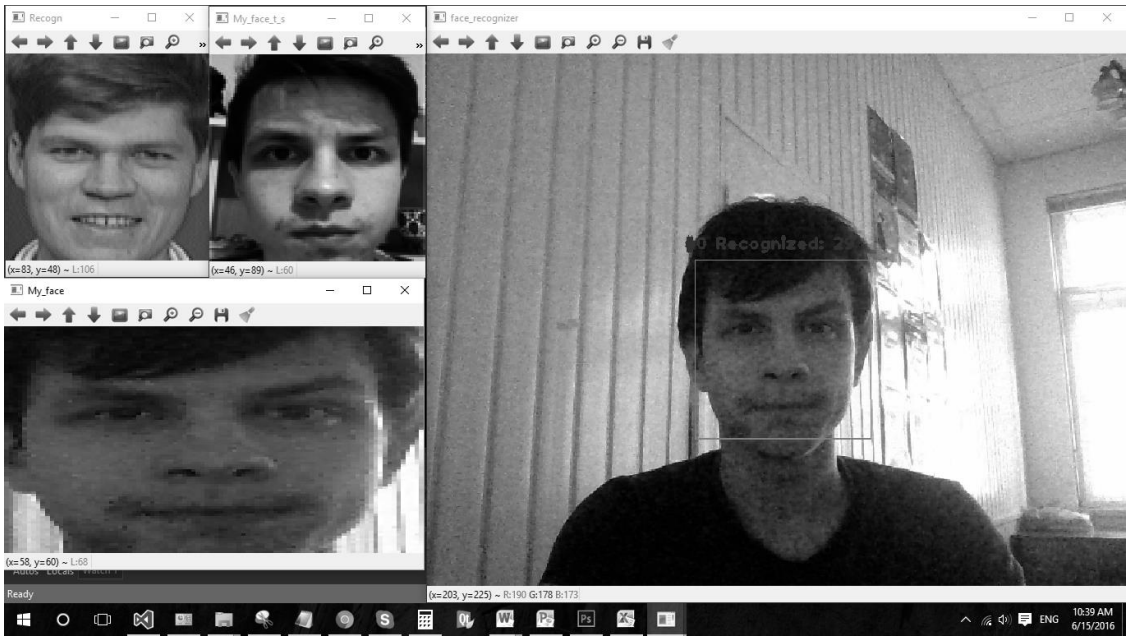

Рисунок 5.1 – Результат роботи програми без накладання умови максимального мінімуму. Обличчя розпізнано невірно.

Рисунок 5.2 – Результат роботи програми без накладання умови максимального мінімуму. Обличчя розпізнано вірно.

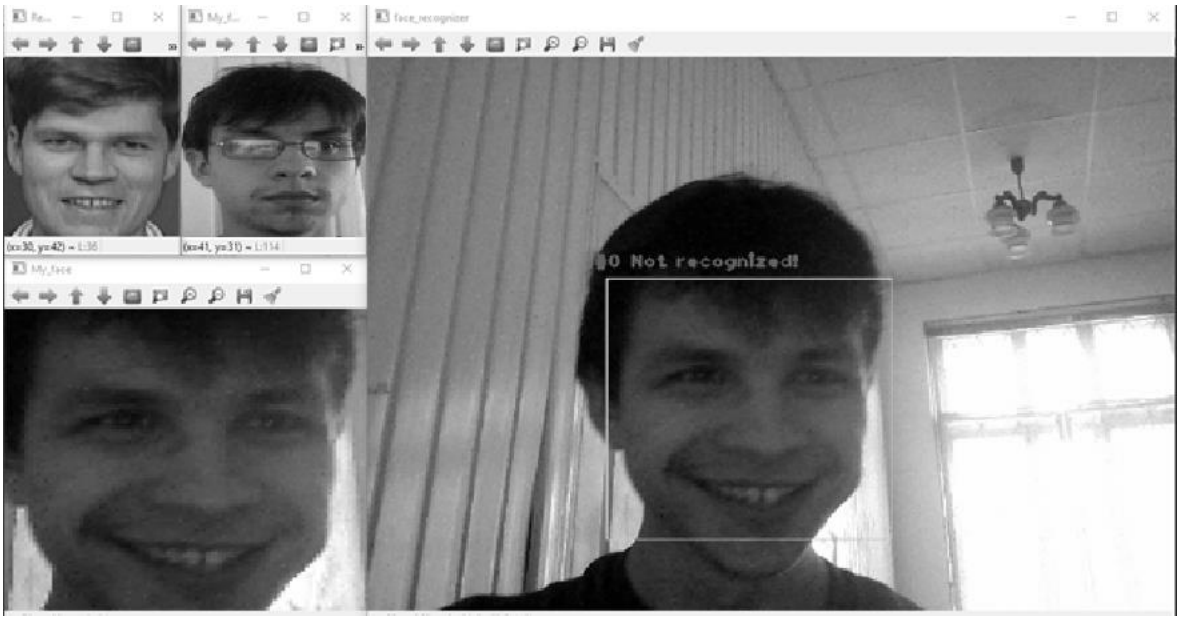

Рисунок 5.3 – Результат роботи програми з накладанням умови максимального мінімуму. Обличчя не розпізнано. Результат вірний.

5.2 Висновки до розділу 5

Було проведено ряд дослідів. Після тестування роботи програми на відеопотоці було виявлено, що алгоритм чутливий до незначних змін зображення людини. Також, було підтверджено, що центрування тестової вибірки та вхідного зображення – найголовніша умова точного розпізнавання. Максимальне значення допустимої мінімальної відстані для вирішення задачі розпізнавання за допомогою даної навченної моделі дорівнювало 0,256.

6. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для покращення якості зображень, використовуючи еволюційні обчислення. Програмний продукт був розроблений за допомогою мови програмування C++ у середовищі розробки Visual Studio Community 2015. Інтерфейс користувача створений за допомогою технології WindowsForms.

Програмний продукт є крос-платформним та рекомендується для використання на персональних комп'ютерах під управлінням операційних систем Windows та Linux.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

6.1 Постановка задачі

У роботі застосовується метод ФВА. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для вибору методу прогнозування місцезнаходження об’єктів у контекстно-залежних системах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

6.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який будує набір моделей прогнозу та перевіряє їх точність на певному наборі даних. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – використання готових бібліотек;

F_3 – вибір фреймворку.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування C++;

б) мова програмування Java;

Функція F_2:

а) написання алгоритмів вручну;

б) використання готових бібліотек;

Функція F_3:

а) Qt;

б) Miscrosoft Windows Studio Community 2015.

6.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 6.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 6.1).

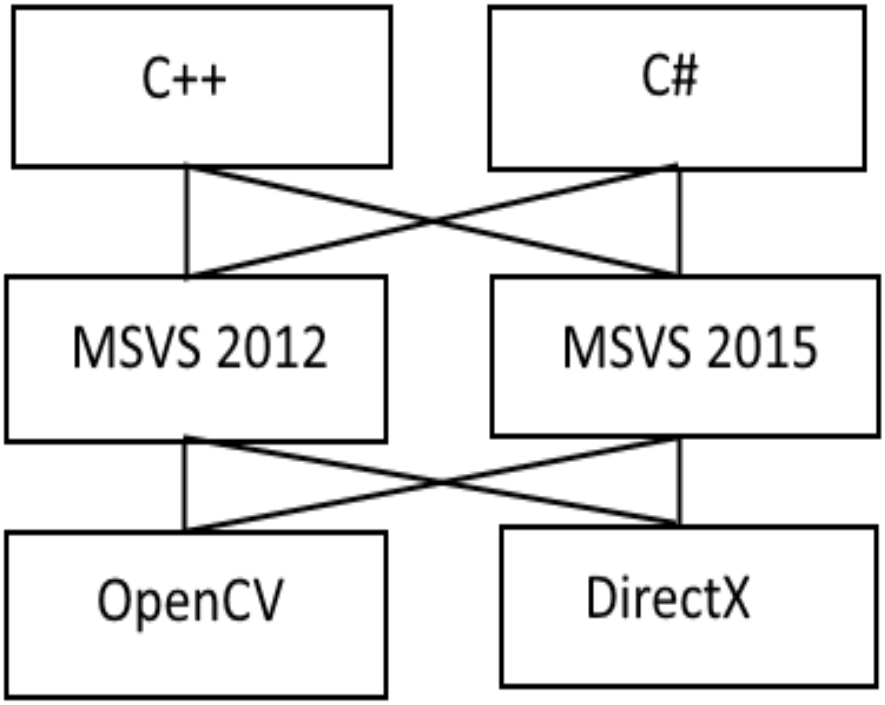

Рисунок 6.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 6.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*A*	Кросплатформений. Швидкість.	Немає.

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*Б*	Швидкість написання коду.	Не кросплатформений. Швидкість.
F2	А	Наявність безкоштовної ліцензії на професійну версію.	Деякі можливості нових стандартів не включені .
	Б	Більше можливостей за рахунок більш нового компілятора.	Новий компілятор може давати збої. Необхідність купувати.
F3	*А*	Простота створення. Кросплатформеність.	Немає.
	Б	Простота створення. На Windows швидкість більше.	Відсутність кросплатформеності.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Функція *F1:*

Оскільки програма має буди швидкою і підтримувати великі об’єми даних, то варіант «Б» має бути відкинутий.

Функція *F2:*

Вибрана IDE з представлених варіантів не буде давати помітної різниці, тому зупиняємось на варіанті «А», як на менш затратному.

Функція *F3:*

Обираємо варіант А, тому що в планах на майбутнє закладена ідея кросплатформенності.

6.2 Обґрунтування системи параметрів ПП

6.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

6.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 6.2.

Таблиця 6.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	2 000	11 000	19 000
Об’єм пам’яті для збереження даних	X2	Мб	1000	420	60
Час обробки даних алгоритмом	X3	мс	2000	1500	1000
Потенційний об’єм програмного коду	X4	кількість строк коду	512	1024	2048

За даними таблиці 6.2 будуються графічні характеристики параметрів – рис. 6.2 – рис. 6.5.

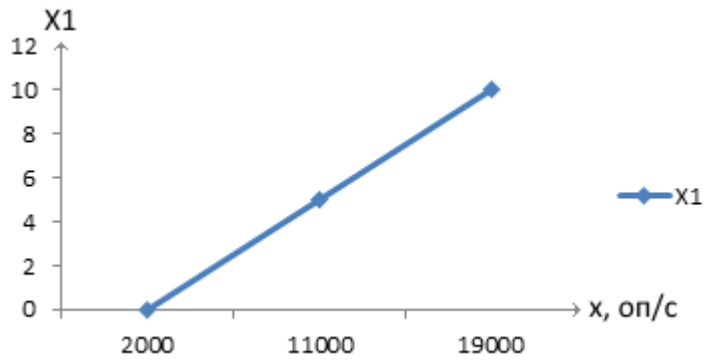

Рисунок 6.2 – X1, швидкодія мови програмування

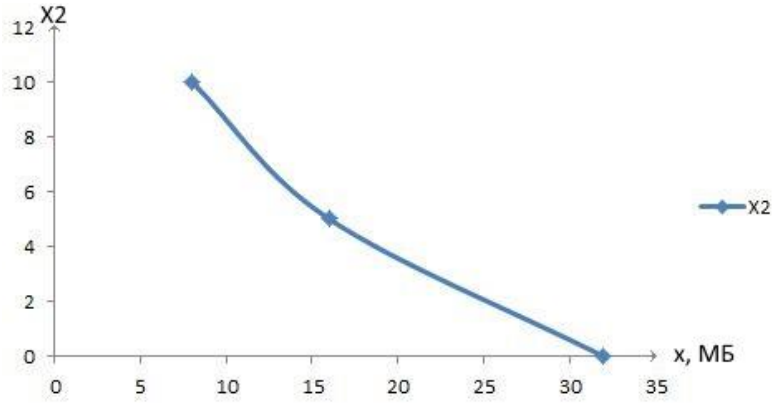

Рисунок 6.3 – X2, об’єм пам’яті для збереження даних

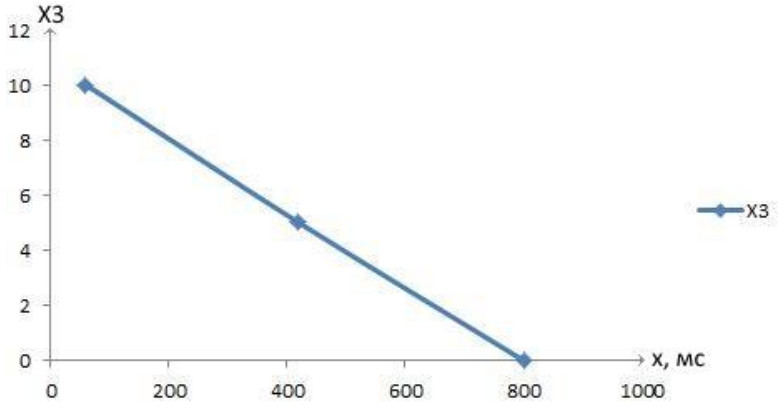

Рисунок 6.4 – X3, час обробки даних алгоритмом

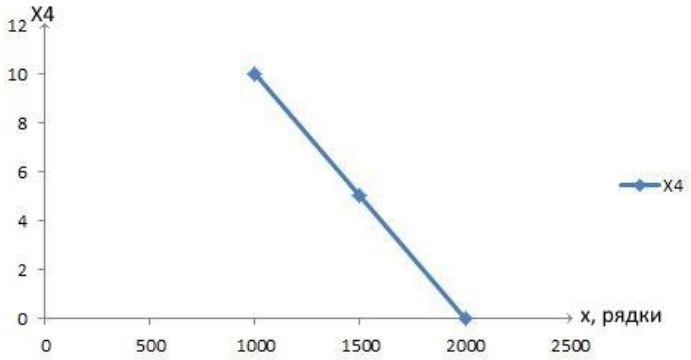

Рисунок 6.5 – X4, потенційний об’єм програмного коду

6.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні

параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 6.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	4	4	3	4	4	4	25	2.5	6.25
X2	Об’єм пам’яті для збереження даних	Мб	1	3	1	2	1	2	3	13	-9.5	90.25

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X3	Час обробки даних алгоритмом	Мс	5	6	6	4	5	6	4	36	13.5	182.25
X4	Потенційний об'єм програмного коду	кількість строк коду	2	2	2	4	2	2	2	16	-6.5	42.25
	Разом		10	15	13	13	12	14	13	90	0	321

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 90,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 22.5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 321$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 321}{7^2(5^3 - 5)} = 1{,}31 > W_k = 0.67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.4.

Таблиця 6.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1.5
X1 і X3	<	<	<	<	<	<	<>	<	0.5
X1 і X4	<>	>	>	<	>	>	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	>	<	<	<	<>	>	<	0.5
X3 і X4	>	>	>	<>	>	>	>	>	1.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 6.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X1	1	1.5	0.5	1.5	4.5	0.18	16.25	0.275
X2	0.5	1	0.5	0.5	2.5	0.1	9.25	0.157
X3	1.5	1.5	1	1.5	5.5	0.22	21.25	0.36

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
X4	0.5	1.5	0.5	1	3.5	0.14	12.25	0.208
Всього:					16	0.64	59	1

6.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об'єм пам'яті для збереження даних) *X1* (швидкодія мови програмування) та *X3* відповідають технічним вимогам умов функціонування даного ПП.1

Абсолютне значення параметра *X4 (*потенційний об'єм програмного коду) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1800 або варіанту б) 1200.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 6.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 6.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	11000	3,6	0,215	0,774
F2(X2)	А	16	3,4	0,283	0,962
F3(X3,X4)	А	800	2,4	0,348	0,835
	Б	80	1	0,154	0,154

За даними з таблиці 6.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0{,}774 + 0{,}962 + 0{,}835 = 2{,}57$

$K_{K2} = 0{,}774 + 0{,}962 + 0{,}154 = 1{,}89$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

6.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

При цьому варіант 3 має додаткове завдання:

1. Реалізація методів аналізу;

А варіант 4 має інше додаткове завдання:

2. Обробка інтерфейсу готових бібліотек.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (6.1)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці

першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90·1.7·0.8 = 122.4 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм другої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$Т_2$ = 27 · 0.9 · 0.8 = 19.44 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (122.4 + 19.44 + 4.8 + 19.44) · 8 = 1328,64 людино-годин;

$Т_{II}$ = (122.4 + 19.44 + 6.91 + 19.44) · 8 = 1345.52 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20 000 грн., один фінансовий аналітик з окладом 15 500 грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$СЧ = \frac{20\,000 + 20\,000 + 15500}{3 \cdot 21 \cdot 8} = 110 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д},$$

де $С_{ч}$– величина погодинної оплати праці програміста;$Т_{i}$ – трудомісткість відповідного завдання; $К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 110 *1 328,64 * 1.2 = 175 380,48 грн.

II. $С_{ЗП}$ = 110 * 1 345,52 * 1.2 = 177 608,64 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22$ = 175 380,48 * 0,22 = 38 583,7 грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22$ = 177 608,64 * 0,22 = 39 073,9 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_{М}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 6000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_{Г} = 12 \cdot М \cdot К_{З}$ = 12 * 20 0000 * 0,2 = 48 000 грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_{Г} \cdot (1 + К_{З})$ = 48 000* (1 + 0.2) = 57 600 грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22$ = 17280 · 0.22 = 3801.6 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 50 000 грн.

$С_{А} = К_{ТМ} \cdot К_{А} \cdot Ц_{ПР}$ = 1.15 · 0.25 · 50 000 = 14 375 грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 50\,000 \cdot 0.05 = 2\,875$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) * t_З * К_В = (365 – 104 – 8 – 16) * 8 * 0.9 = 1706.4$ годин,

годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} * N_С * К_З * Ц_{ЕН} = 1706{,}4 * 0{,}156 * 0{,}9733 * 1{,}506 = 390{,}20$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} * 0.67 = 50\,000 ; 0{,}67 = 33\,500$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 57\,600 + 12\,672 + 14\,375 + 2\,875 + 390{,}20 + 33\,500 = 121\,412{,}20$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М-Г} = C_{ЕКС} / T_{ЕФ} = 121\ 412{,}20 / 1\ 706{,}4 = 71{,}15$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot T$$

I. $C_М = 71{,}15 * 1\ 328{,}64 = 94\ 534{,}16$ грн.;

II. $C_М = 71{,}15 * 1\ 345.52 = 95\ 733{,}75$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} * 0{,}67$$

I. $C_Н = 175\ 380{,}48 * 0{,}67 = 117\ 504{,}92$ грн.;

II. $C_Н = 177\ 608{,}64 * 0{,}67 = 118\ 997{,}78$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н$$

I. $C_{ПП} = 175\ 380{,}48 + 38\ 583{,}7 + 94\ 534{,}16 + 117\ 504{,}92 = 426\ 003{,}26$ грн.;

II. $C_{ПП} = 177\ 608{,}64 + 39\ 073{,}9 + 95\ 733{,}75 + 118\ 997{,}78 = 431\ 414{,}07$ грн.;

6.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 2,57 / 426\,003,26 = 6 \cdot 10^{-6}$;

$К_{ТЕР2} = 1,89 / 431\,414,07 = 4,3 \cdot 10^{-6}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 6 \cdot 10^{-6}$.

6.6 Висновки до розділу 6

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

ВИСНОВКИ

У даній роботі було проведено аналіз алгоритму Eigenface. Проаналізовано ефекстивність та швидкість роботи у порівнянні з іншими алгоритмами розпізнавання обличчя, та протестовано даний алгоритм на практиці. Була реалізована програма розпізнавання статичних зображень обличчя у середовищі Matlab та відеопотоку на мові програмування C++ за допомогою бібліотеки OpenCV. Досліди показали, що алгоритм показує високий відсоток правильної ідентифікації при умові, що всі обличчя правильно відцентровані, первинна виборка є досить великою та різноманітною, у якій згруповані комбінації облич між відповідними людьми.

На сьогодні алгоритм являється актуальним тому, що його можна використовувати як для вирішення задачі розпізнавання обличь, так і інших образів, що певним чином можна поділяти на підмножини (розпізнавання жестів, «читання по губам»).

Для покращення даного методу було реалізовано розпізнавання в декілька потоків. Інакше кажучи, якщо на вхід одночасно подається декілька зображень, кожне зображення переходить до відведенного йому потоку, який оброблює, перевіряє та ідентифікує приналежність зображення до первинної виборки. Такий підхід значно прискорив роботу алгоритму при обробці відеопотоку.

Серед тестувань було перевірено наступні комбінації вхідних значень та первинної виборки:

- Зображення належить до підмножини навченої моделі, та серед первинної виборки вхідне зображення присутнє;
- Зображення належить до підмножини навченої моделі, але серед первинної виборки дане зображення відсутнє;
- Зображення не належить до підмножини навченої моделі, але присутнє зображення схожої людини

- Зображення зовсім не належить до підмножини навченої моделі.

Тестування обробки статичних зображень показали набагато кращі результати, ніж тестування обробки відеопотоку. Дане явище обумовлене багатьма причинами: погана відеокамера, рухомість об’єктів, що піддаються розпізнаванню, значна різниця між освітленням.

Реалізованна обробка відеопотоку для вирішення задачі розпізнавання людських обличь не давала жодних перебоїв, було отримано певні результати, що допомогли скласти певне бачення та розуміння алгоритму та краще ознайомитися з класифікаторами Хаара та методом головних компонент, що являється ядром алгоритму eigenface.

Алгоритм підходить для вирішення певних задач розпізнавання, але для відеопотоку він зазнає певних втрат ефективності. Буде вестись подальше дослідження алгоритму для вирішення проблеми втрати єфективності шляхом його покращення з середини, не впливаюси на первиний набір та вхідного зображення.

Для використання алгоритму в контектсі інших задач, необхідно використовувати алгоритми пошуку певних об’єктів на зображенні. У даному випадку ми використовували каскади Хаара. У інших випадках можна використовувати або каскади Хаара, або один з наступних дескрипторів: SURF, SIFT, FAST, ORB, BRISK, що використовуються для відстежання різних об’єктів по характеристичним точкам.

Частина результатів та дослідів даної дипломної роботи було викладено у Міжнародному Науковому Журналі 2016 року, 6-го випуску. У публікації описані основні принципи роботи алгоритму, реалізація алгоритму у Matlab середовищі, та задачі, для яких доцільно буде використовувати даний алгоритм.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сарапулов, В. Дослідження алгоритму розпізнавання обличь та його реалізація у Matlab середовищі / В.Сарапулов – Режим доступу: http://www.inter-nauka.com/issues/2016/5/1221 Дата доступу - 1.06.2016.
2. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. - М.: Техносфера, 2005. - 1072 с.
3. Белоусов, А.А. Применение генетических алгоритмов и вейвлет-преобразований для повышения качества изображений/ А.А. Белоусов, В.Г. Спицын, Д.В. Сидоров // Известия Томского политехнического университета, Т. 309. № 7. - 2006. - С. 21-26.
4. Каширина, И.Л. Введение в эволюционное моделирование. Учебное пособие/ И.Л. Каширина. Воронеж: ВГУ, 2007.
5. Курейчик, В.М. Параллельный генетический алгоритм. Модели и проблемы построения/ В.М. Курейчик, Кныш Д.С. - Таганрог: Технологический ин-т ЮФУ, 2010.
6. Курейчик В. М. Генетические алгоритмы и их применение/ В.М. Курейчик. - Таганрог: Изд-во ТРТУ, 2002.
7. Baeck, T. Evolutionary computation/ T. Baeck, D. Fogel, Z. Michalewicz. - Berlin Heidelberg: Springer-Verlag, 2000.
8. De Jong, K.A. An analysis of the behavior of a class of genetic adaptive systems: Unpublished PhD thesis / K. De Jong. - University of Michigan, Ann Arbor, 1975. - Also University microfilms No. 76-9381 – Режим доступу: http://www.cs.gmu.edu/~eclab Дата доступу – 30.04.2016.
9. Holland, J.H. Adaptation in natural and artificial systems/ J.H. Holland. - Michigan: The University of Michigan Press, 1975.

ДОДАТОК А

Лістинг програми середовища Matlab

```
function [Imgs,w,h,Vecs,Vals,Psi] = load_images(filelist,downscale_f)
%       Imgs – матриця зображень
%     w    - ширина зображення
%     h    - довжина зображення
%     Vecs – власні вектори зображень
%     Vals – власні значення зображень
%       Psi  - середнє значення
 if nargin < 1 || nargin > 2
  error('usage: load_Imgs(filelist[,downscale_f]');
 end;
 if nargin == 1
  downscale_f = 1.0;
 end;
 Imgs = []; old_w = 0; old_h = 0; w=0; h=0;
 numimgs = linecount(filelist);
 fid = fopen(filelist,'r');
 if fid < 0 || numimgs < 1
  error(['Cannot get list of Imgs from file "' filelist, '"']);
 end;
 % Get the Imgs
 for i = 1:numimgs
  imgname = fgetl(fid);
  if ~isstr(imgname)
   break;                    % вихід з циклу, якщо EOF
  end;
  fprintf(1,'loading PGM file %s\n',imgname);
```

```
   Img = readpgm(imgname);       % Окрема функція зчитування у
двопросторовий              масив
   if i==1                     % Якщо це перше зображення,
     old_w = size(Img,2);       % тоді визначити довжину, ширину та масштаб
     old_h = size(Img,1);
     if downscale_f <= 1.0
   w = old_w; h = old_h;
     else
   w = round(old_w/downscale_f); h = round(old_h/downscale_f);
     end;
     Imgs = zeros(w*h,numimgs);   % - передвстановлення значень матриці
   end;
   if downscale_f > 1.0
     Img = im_resize(Img,w,h);      % зміна масштабування
   end;
   Imgs(1:w*h,i) = reshape(Img',w*h,1);   % зі стрічки зробити стовбець
 end;
 fclose(fid);                  % закрити список зображень після закінчення
 fprintf(1,'Read %d Imgs.\n',numimgs);
[Vecs,Vals,Psi] = pc_evectors(Imgs,numimgs); % визначення власних векторів

function lc = linecount(filename)
fid = fopen(filename,'r');
 if fid < 0
   lc = 0;
 else
   lc = 0;
   while 1
     ln = fgetl(fid);
     if ~isstr(ln) break; end;
```

```
      lc = lc + 1;
    end;
    fclose(fid);
  end;

function image = readpgm(filename)
fid = fopen(filename,'r');
A = fgets(fid);
if strcmp(A(1:2),'P5') ~= 1
  error('File is not a raw PGM');
end;
A = fgets(fid);
sizes = sscanf(A,'%d');
w = sizes(1);
h = sizes(2);
A = fgets(fid);
max = sscanf(A,'%d');
tlength = w*h;
if max ~= 255
 error('Cannot handle anything but 8-bit graymaps');
end;
 [v,count] = fread(fid,inf,'uchar');
if count ~= tlength
 error('File size does not agree with specified dimensions.');
end;
image = reshape(v,w,h)';
fclose(fid);


function [Vecs,Vals,Psi] = pc_evectors(A,numvecs)
```

```
if nargin ~= 2
  error('usage: pc_eVecs(A,numvecs)');
end;
nexamp = size(A,2);
fprintf(1,'Computing average vector and vector differences from avg...\n');
Psi = mean(A')';
% Вирахування середнього значення з усіх векторів
for i = 1:nexamp
  A(:,i) = A(:,i) - Psi;
end;
% Знаходження матриці коваріацій
fprintf(1,'Calculating L=A"A\n');
L = A'*A;
% Підрахунок власних векторів(стовбці Vecs) та власних значень (диагоналі
Vals)
fprintf(1,'Calculating eigenVecs of L...\n');
[Vecs,Vals] = eig(L);
% Сортування Джеймса Джавурек-Хаміга по зменшенню значень власних
значень
fprintf(1,'Sorting eVecs/Vals...\n');
[Vecs,Vals] = sortem2(Vecs,Vals);
% конвертування власних векторів з A'*A у власні вектори A*A'
fprintf(1,'Computing eigenVecs of the real covariance matrix..\n');
Vecs = A*Vecs;
Vals = diag(Vals);
Vals = Vals / (nexamp-1);
% Нормалізація власних веторів та відкидання «поганих»
num_good = 0;
for i = 1:nexamp
  Vecs(:,i) = Vecs(:,i)/norm(Vecs(:,i));
```

```
    if Vals(i) < 0.00001
      % Set the vector to the 0 vector; set the value to 0.
      Vals(i) = 0;
      Vecs(:,i) = zeros(size(Vecs,1),1);
    else
      num_good = num_good + 1;
    end;
  end;
  if (numvecs > num_good)
    fprintf(1,'Warning: numvecs is %d; only %d exist.\n',numvecs,num_good);
    numvecs = num_good;
  end;
  Vecs = Vecs(:,1:numvecs);
Omega = Vecs'*A;

function [vectors values] = sortem2(vectors, values)
%this error message is directly from Matthew Dailey's sortem.m
if nargin ~= 2
 error('Must specify vector matrix and diag value matrix')
end;
vals = max(values); %create a row vector containing only the eigenvalues
[svals inds] = sort(vals,'descend'); %sort the row vector and get the indicies
vectors = vectors(:,inds); %sort the vectors according to the indicies from sort
values = max(values(:,inds)); %sort the eigenvalues according to the indicies from
sort
values = diag(values); %place the values into a diagonal matrix


plot(Vals);
CVals = zeros(1,length(Vals));
CVals(1) = Vals(1);
```

```
for i = 2:length(Vals)
CVals(i) = CVals(i-1) + Vals(i);
end;
CVals = CVals / sum(Vals);
plot(CVals);
ylim([0 1]);

Img = load_single(file);
%дана функція – копія функції load_images для однієї картинки
Projection = Vecs'*(Img - Psi);
d = repmat(Projection, 1, M) - Omega;
dist = zeros(M,1);
for i=1:M
      dist(i,1)=norm(d(:,i));
end;
index=IndexOfMinimum(dist);
%функція визначення індексу елементу з найменшою відстаню
subplot(1,2,1);
back_convertion(Imgs,w,h,index);
%функція конвертації вектора у картинку та її відображення
subplot(1,2,2);
back_convertion(Img,w,h,1)
```

Функція оберненої конвертації:

```
function y = back_convertion(Imgs,w,h,img_num);
L = Imgs(:,img_num);
L = reshape(L,w,h)';
pgm_show(L);
```

Функція пошуку відповідного елементу в масиві:

```
function minIndex=IndexOfMinimum(x)
minIndex=[];
if size(x,2)>1
   fprintf('Give a column vector as an input\n')
else
min_element=min(x);
for i=1:size(x,1)
   if x(i,1) == min_element
      minIndex=i;
      break;
   end
end
end
if min_element/1000 < 3.8
%критерій знаходження обличчя. Підбирається вручну
   fprintf(1,'Face recognized. Distance = %f\n', min_element);
else
   fprintf(1,'Face not recognized. Distance = %f\n', min_element);
end
```

ДОДАТОК Б

Лістинг програми на мові C++ з використанням Opencv

```
#include "opencv2/core.hpp"
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/objdetect.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
using namespace cv;
using namespace cv::face;
using namespace std;
static void read_csv(const string& filename, vector<Mat>& images, vector<int>&
labels, char separator = ';') {
	std::ifstream file(filename.c_str(), ifstream::in);
	if (!file) {
		string error_message = "No valid input file was given, please check the
given filename.";
		CV_Error(CV_StsBadArg, error_message);
	}
	string line, path, classlabel;
	while (getline(file, line)) {
		stringstream liness(line);
		getline(liness, path, separator);
		getline(liness, classlabel);
		if (!path.empty() && !classlabel.empty()) {
```

```
                images.push_back(imread(path, 0));
                labels.push_back(atoi(classlabel.c_str()));
            }
        }
}
int main(int argc, const char *argv[]) {
        string fn_haar, fn_csv;
        int deviceId;
        if (argc != 4) {
            cout << "usage: " << argv[0] << " </path/to/haar_cascade>
</path/to/csv.ext> </path/to/device id>" << endl;
            cout << "\t </path/to/haar_cascade> -- Path to the Haar Cascade for face
detection." << endl;
            cout << "\t </path/to/csv.ext> -- Path to the CSV file with the face
database." << endl;
            cout << "\t <device id> -- The webcam device id to grab frames
from.\n\nDefault settings activated!" << endl;
            fn_haar = "C:\\Users\\da222\\Desktop\\Opencv-master\\Opencv-
master\\haarcascade\\haarcascade_frontalface_alt2.xml";
            fn_csv = ".\\at.csv";
            deviceId = 0;

        }
        else{
            fn_haar = string(argv[1]);
            fn_csv = string(argv[2]);
            deviceId = atoi(argv[3]);
        }
        vector<Mat> images;
        vector<int> labels;
```

```
        try {
                read_csv(fn_csv, images, labels);
        }
        catch (cv::Exception& e) {
                cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg <<
endl;
                exit(1);
        }

        int im_width = images[0].cols;
        int im_height = images[0].rows;

        CascadeClassifier haar_cascade;
        haar_cascade.load(fn_haar);
        for (int i = 0; i < images.size(); i++){
                vector< Rect_<int> > train_faces;
                haar_cascade.detectMultiScale(images[i], train_faces);
                Rect roi(0, 10, im_width, im_width);

                images[i] = images[i](roi);
        }
        im_width = images[0].cols;
        im_height = images[0].rows;
        Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
        model->train(images, labels);
        VideoCapture cap(deviceId);
        if (!cap.isOpened()) {
                cerr << "Capture Device ID " << deviceId << "cannot be opened." <<
endl;
                return -1;
```

```
        }
        Mat frame;
        for (;;) {
                cap >> frame;
                Mat original = frame.clone();
                Mat gray;
                cvtColor(original, gray, CV_BGR2GRAY);
                vector< Rect_<int> > faces;
                haar_cascade.detectMultiScale(gray, faces);
                for (int i = 0; i < faces.size(); i++) {
                        Rect face_i = faces[i];
                        Mat face = gray(face_i);
                        Mat face_resized;
                        cv::resize(face, face_resized, Size(im_width, im_height), 1.0, 1.0,
INTER_CUBIC);
                        namedWindow("My_face", CV_WINDOW_KEEPRATIO);
                        namedWindow("My_face_t_s", CV_WINDOW_KEEPRATIO);
                        imshow("My_face", face_resized);
                        imshow("My_face_t_s", images.back());
                        int prediction = model->predict(face_resized);
                        rectangle(original, face_i, CV_RGB(0, 255, 0), 1);
                        string box_text;
                        if (prediction == 0){
                                box_text = format("#%i Not recognized!", i);
                        }
                        else{
                                box_text = format("#%i Recognized: %d", i, prediction);
                                if (prediction == 40){
                                        cout << "Viktor!!!\n";
                                }
```

```
                }
                string out = to_string(i);
                namedWindow("Recogn", CV_WINDOW_KEEPRATIO);
                int pos_x = std::max(face_i.tl().x - 10, 0);
                int pos_y = std::max(face_i.tl().y - 10, 0);
                putText(original, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(100, 100, 0), 2.0);
                if (prediction > 0){
                        imshow("Recogn", images[((prediction) * 10) + 1]);
                }
        }
        imshow("face_recognizer", original);
        char key = (char)waitKey(20);
        if (key == 27)
                break;
    }
    return 0;
}
```