Хмельницький національний університет
Факультет програмування та комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерних наук та інформаційних технологій

ДИПЛОМНА РОБОТА
МАГІСТРА

на тему *Моделювання ознак зображення для задач розпізнавання*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Виконав: студент *2* курсу, група *КНм-19-1* Підпис *С.О. Іващенко* Ініціали, прізвище

Керівник: *д.т.н., проф. кафедри КНІТ* Підпис *О.В. Бармак* Ініціали, прізвище

Нормоконтроль: *к.т.н., доцент кафедри КНІТ* Підпис *Р.О. Багрій* Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КНІТ, д.т.н., професор Підпис *О.В. Бармак* Ініціали, прізвище

7 *12* 2020 р.

Хмельницький 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *програмування та комп'ютерних і телекомунікаційних систем*

Кафедра *комп'ютерних наук та інформаційних технологій*

Освітній ступінь *магістр*

Галузь знань *12 – Інформаційні технології*

Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Завідувач кафедри *комп'ютерних наук та інформаційних технологій*

(підпис)

д.т.н., професор О.В. Бармак

«*7*» *09* *2020* року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ МАГІСТРА

1. Тема дипломної роботи магістра: *«Моделювання ознак зображення для задач розпізнавання»*

2. Завдання видано студенту *Іващенку Сергію Олександровичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *д.т.н., професор Бармак Олександр Володимирович*
(прізвище, ім'я, по батькові)

4. Затверджені наказом університету від «*9*» *09* *2020* р. № *22*

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Мета роботи – дослідження математичних моделей для задач розпізнавання, з використанням гіперплощиної класифікації, розробка програмних засобів для дослідження. Запропонований підхід базується на геометричних особливостях обличчя під час різних емоційних станів. Результатом виконання є модель, підхід до розпізнавання та інформаційна система для дослідження.

Реферат

Дипломна робота магістра присвячена створенню та дослідженню моделі мімічних проявів емоційного стану на обличчі людини, розробці інформаційної системи для розпізнавання мімічних проявів емоцій на обличчі людини.

Актуальність теми. Сьогодні більшість нашого часу в повсякденному житті витрачається на взаємодію з комп’ютерами та мобільними телефонами завдяки прогресу технологій та їхньому розповсюдженню. Вони відіграють важливу роль у нашому житті, і величезна кількість існуючих програмних інтерфейсів є невербальними, примітивними та лаконічними. Включення розпізнавання емоційних виразів для фіксації відчуттів та емоційного стану користувачів може різко покращити взаємодію людини та комп'ютера. Human–computer interaction (HCI) вважається однією з найпривабливіших і швидко зростаючих галузей.

Розпізнавання емоцій по ознакам обличчя є однією з технологій, що найбільш динамічно розвивається. Розпізнавання виразів обличчя може широко застосовуватися в різних областях досліджень, таких як діагностика психічних захворювань та виявлення соціальної та фізіологічної взаємодії людини.

Незважаючи на те, що лабораторно керовані системи Facial Expression Recognition (FER) досягають дуже високої точності, близько 97%, технічний перехід від лабораторії до реальних застосувань стикається з великим бар'єром дуже низької точності, приблизно 50%. Тому розробка такого застосування є актуальним та затребуваним для практичного використання. Адже збільшення точності розпізнавання важливий аспект дослідження.

Мета і задачі роботи. Метою роботи є створення і дослідження моделі мімічних проявів емоційного стану людини, розробка інформаційної системи для розпізнавання мімічних проявів емоцій.

Для досягнення поставленої мети визначені наступні задачі дослідження:

– створення та дослідження моделі мімічних проявів емоцій;

– дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;

– розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;

– валідація та верифікація моделі, підходу та інформаційної системи.

Об’єкт дослідження - процес розпізнавання мімічних проявів емоційних станів із використанням інформаційних технологій.

Предмет дослідження - моделі, методи, підходи та засоби інформаційної технології автоматизованого розпізнавання емоційних станів.

Методи дослідження, використані для вирішення поставлених завдань: для виявлення обличчя – метод Хаара, для виявлення контрольних точок обличчя – facemarkLBF, для багатовимірного шкалювання – класичний MDS та SMACOF, для визначення приналежності до певного емоційного стану – метод гіперплощинної класифікації.

Наукова новизна одержаних результатів. В результаті проведених досліджень були отримані наступні результати:

– вдосконалено модель емоційного стану обличчя людини, досліджено її валідність для ідентифікації різних емоційних станів;

– у рамках розробленої моделі, вдосконалено підхід до ідентифікації мімічних проявів емоційних станів;

– розроблено застосування, що за допомогою вдосконалених моделі і підходу ідентифікує мімічні прояви емоцій на обличчі людини.

Практичне значення одержаних результатів. Запропоновані модель, підхід до розпізнавання та інформаційна системи на їх базі дозволяють використовувати емоційну складову для багатьох практичних задач: контроль стану водія транспортного засобу, контроль за станом оператора складного виробництва тощо.

Апробація результатів дипломної роботи магістра та публікації. Основні наукові та практичні результати за магістерською роботою

оприлюднювались на міжнародній конференції IEEE International Conference on Advanced Trends in Information Theory (ATIT), та опубліковані:

1. Svirnevskyi M., Barmak O., Ivaschenko S., Krak Iu. Face Image Transformations for Correct Recognition Problems Solving // IEEE International Conference on Advanced Trends in Information Theory (ATIT). 2019,P. 415-419 (Scopus)

2. Іващенко С.О., Калита О.Д., Бармак О.В., Скрипник Т.К. Інформаційна технологія визначення характерних ознак на обличчі для розпізнавання емоційних проявів // Журнал Computer Systems And Information Technologies. Том 1 № 2 (2020).

Структура та обсяг роботи. Дипломна робота магістра складається з завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 36 найменувань та 1 додатку. Загальний обсяг дипломної роботи магістра становить 98 сторінок, з них 71 сторінки основного тексту та 27 сторінок додатків. У роботі наведено 51 рисунків та 6 таблиць.

Ключові слова: моделювання ознак зображення, розпізнавання, класифікація, емоції.

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
FER	Facial Expression Recognition
MDS	Multidimensional scaling
ЛДО	Локальні двійкові ознаки
FACS	Facial Action Coding System
GUI	Graphical user interface
HCI	Human–computer interaction
SMACOF	Scaling by MAjorization of a COmplicated Function

Вступ

Актуальність теми. Сьогодні більшість нашого часу в повсякденному житті витрачається на взаємодію з комп’ютерами та мобільними телефонами завдяки прогресу технологій та їхньому розповсюдженню. Вони відіграють важливу роль у нашому житті, і величезна кількість існуючих програмних інтерфейсів є невербальними, примітивними та лаконічними. Включення розпізнавання емоційних виразів для фіксації відчуттів та емоційного стану користувачів може різко покращити взаємодію людини та комп'ютера. Human–computer interaction (HCI) вважається однією з найпривабливіших і швидко зростаючих галузей.

Розпізнавання емоцій по ознакам обличчя є однією з технологій, що найбільш динамічно розвивається. Розпізнавання виразів обличчя може широко застосовуватися в різних областях досліджень, таких як діагностика психічних захворювань та виявлення соціальної та фізіологічної взаємодії людини.

Незважаючи на те, що лабораторно керовані системи Facial Expression Recognition (FER) досягають дуже високої точності, близько 97%, технічний перехід від лабораторії до реальних застосувань стикається з великим бар'єром дуже низької точності, приблизно 50%. Тому розробка такого застосування є актуальним та затребуваним для практичного використання. Адже збільшення точності розпізнавання важливий аспект дослідження.

Метою роботи є створення і дослідження моделі мімічних проявів емоційного стану людини, розробка інформаційної системи для розпізнавання мімічних проявів емоцій.

Для досягнення поставленої мети визначені наступні задачі дослідження:

– створення та дослідження моделі мімічних проявів емоцій;

– дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;

– розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;

– валідація та верифікація моделі, підходу та інформаційної системи.

Об’єкт дослідження - процес розпізнавання мімічних проявів емоційних станів із використанням інформаційних технологій.

Предмет дослідження - моделі, методи, підходи та засоби інформаційної технології автоматизованого розпізнавання емоційних станів.

Методи дослідження, використані для вирішення поставлених завдань: для виявлення обличчя – метод Хаара, для виявлення контрольних точок обличчя – facemarkLBF, для багатовимірного шкалювання – класичний MDS та SMACOF, для визначення приналежності до певного емоційного стану – метод гіперплощинної класифікації.

Практичне значення одержаних результатів. В результаті проведених досліджень були отримані наступні результати:

– вдосконалено модель емоційного стану обличчя людини, досліджено її валідність для ідентифікації різних емоційних станів;

– у рамках розробленої моделі, вдосконалено підхід до ідентифікації мімічних проявів емоційних станів;

– розроблено застосування, що за допомогою вдосконалених моделі і підходу ідентифікує мімічні прояви емоцій на обличчі людини.

Запропоновані модель, підхід до розпізнавання та інформаційна системи на їх базі дозволяють використовувати емоційну складову для багатьох практичних задач: контроль стану водія транспортного засобу, контроль за станом оператора складного виробництва тощо.

Апробація результатів дипломної роботи магістра та публікації. Основні наукові та практичні результати за магістерською роботою оприлюднювались на міжнародній конференції IEEE International Conference on Advanced Trends in Information Theory (ATIT), та опубліковані:

1. Svirnevskyi M., Barmak O., Ivaschenko S., Krak Iu. Face Image Transformations for Correct Recognition Problems Solving // IEEE International Conference on Advanced Trends in Information Theory (ATIT). 2019,P. 415-419 (Scopus)

2. Іващенко С.О., Калита О.Д., Бармак О.В., Скрипник Т.К. Інформаційна технологія визначення характерних ознак на обличчі для розпізнавання емоційних проявів // Журнал Computer Systems And Information Technologies. Том 1 № 2 (2020).

Структура та обсяг роботи. Дипломна робота магістра складається з завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 36 найменувань та 1 додатку. Загальний обсяг дипломної роботи магістра становить 98 сторінок, з них 71 сторінки основного тексту та 26 сторінок додатків. У роботі наведено 51 рисунків та 6 таблиць.

Розділ 1
Аналіз сучасного стану у розпізнаванні мімічних проявів емоцій

1.1. Внутрішня та зовнішня природа емоцій

Емоції - це переживання людиною свого особистого ставлення до дійсності, його реакції на вплив внутрішніх і зовнішніх подразників. Емоційне переживання - це суб'єктивне відображення об'єктивної реальності, його виникнення наперед визначене впливом на нервову систему людини об'єктивних явищ (рис. 1.1). Виникнення емоцій пов'язане із задоволенням чи незадоволенням потреб людини. Ті або інші явища можуть викликати почуття задоволення чи невдоволення, можуть порадувати чи засмутити, подобатися чи не подобатися. Необхідно зазначити, що емоції істотно впливають на діяльність людини [1].

Рисунок 1.1 – Емоції

Існує різноманіття емоцій (емоційних переживань). Виникнення різних емоцій обумовлене не лише властивостями впливів, але і суб'єктивними особливостями самого індивіда, його індивідуально-психологічними особливостями, станом здоров'я, потребами, прагненнями, бажаннями тощо.
Емоційні переживання можуть бути класифіковані. Так, розрізняють позитивні і негативні емоції (характеризуються наявністю відтінків задоволення або невдоволення). Емоції відрізняються також за ступенем збудження чи заспокоєння.

Є три класи емоцій (рис. 1.2):

– Настрої – стійкі переживання будь-яких емоцій. Вони є найбільш поширеним видом емоційних станів людини.

– Афекти – переживання, що вирізняються різко вираженою інтенсивністю і відносною короткочасністю.

– Почуття – переживання, що характеризуються відносною стійкістю і виділяють ті чи інші явища, що мають стабільну мотиваційну значущість.

Рисунок 1.2 – Види емоцій [1]

Є три види вищих почуттів (рис. 1.3):

– моральні (відображається ставлення індивідуума до вимог моралі): почуття обов'язку, дружби, людської гідності тощо;

– інтелектуальні (пов'язані з пізнавальною сферою діяльності людини): почуття здогадки, упевненості, сумніву, подиву тощо;

– естетичні (викликані красою чи потворністю явищ чи об'єктів, що сприймаються).

Рисунок 1.3 – Види почуттів [1]

Виникнення і розвиток емоцій обумовлений впливом таких чинників: спосіб життя індивідуума; особливості його діяльності; рівень інтелектуального

розвитку; характер виховання; здатність переживати різні емоції. Емоції людини відображаються в різних емоційних станах. Емоційні стани (радість, страх, гнів, сум та ін.) зовнішньо проявляються у зміні подиху міміці, пантоміміці (рис. 1.4) та ін.

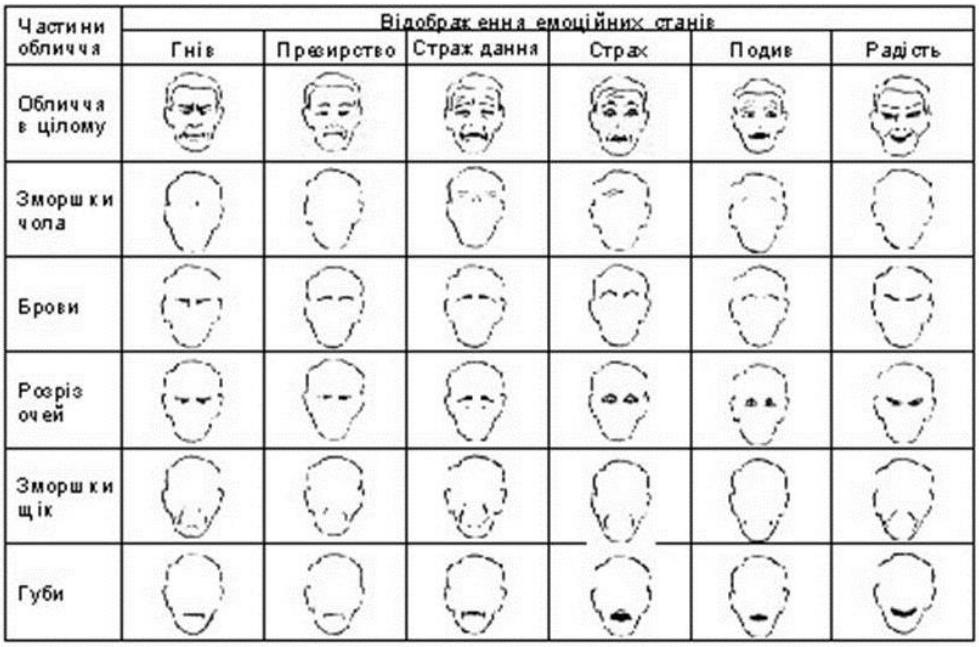


Рисунок 1.4 – Міміка обличчя при різних емоційних станах [1]

Фундаментальні емоції забезпечуються вродженими нейронними програмами [2]. Міміка покликана приховувати або заміщати вроджені типи виразів емоцій і надзвичайно різна у представників різних соціальних верств.

К. Ізард виділив [2] такі основні, «фундаментальні емоції»:

– Зацікавленість - позитивний емоційний стан, що сприяє розвитку навичок і умінь, набуття знань, мотивуюче навчання;

– Радість – позитивний емоційний стан з можливістю повністю задовольнити актуальні потреби, ймовірність чого до даного моменту була невелика або невизначена;

– Здивування – не має чіткого вираженого позитивної або негативної ознаки емоційної реакції на раптово виникнувши обставини;

– Страждання – негативний емоційний стан, зв’язаний з отриманням достовірної інформації про неможливість задовольнити життєві потреби;

– Гнів – негативний емоційний стан, як правило, протікає в стані афекту і викликане раптовою перешкодою на шляху до виключно важливої для суб’єкта потреби;

– Відраза – негативний емоційний стан, визваний об’єктами, які вступають в різке противоріччя з принципами і установками суб’єкта;

– Зневага – негативний емоційний стан, що виникає в взаєминах і породжений неузгодженістю життєвих позицій, поглядів і поведінки суб'єкта з життєвими позиціями, поглядами і поведінкою об'єкта почуття.

– Страх – негативний емоційний стан, виникає при отриманні інформації про можливу загрозу;

– Сором – негативний стан, що виражається в усвідомленні невідповідності власних помислів, вчинків і зовнішності не тільки очікуванням оточуючих, а й власним уявленням про належну поведінку і зовнішній вигляд.

Макро вираз - прояви емоцій на обличчі, які тривають від пів секунди секунди до приблизно 4 секунд. Ці емоції чітко видно на обличчі людини., Виражені через макро вирази, включають гнів, шок, смуток, щастя та хвилювання [3].

Мікро вираз - це мимовільне прояв обличчям справжньої емоції, яку набагато важче побачити, оскільки вона триває лише частку секунди, іноді так швидко, як 1/25 секунди. Людина, яка виразила мікровираз, зазвичай не знає, що вони проявили цю емоцію через мікровираз.

Здивування, яке викликане чимось раптовим чи несподіваним. При здивуванні брови підняті, але проявляють більше вигинання, ніж видно зі страху (рис. 1.5) [4]. Верхні повіки та щелепи також більш розслаблені, коли висловлюють здивування. Повіки відкриті, біла частина очей показана зверху і знизу [5].

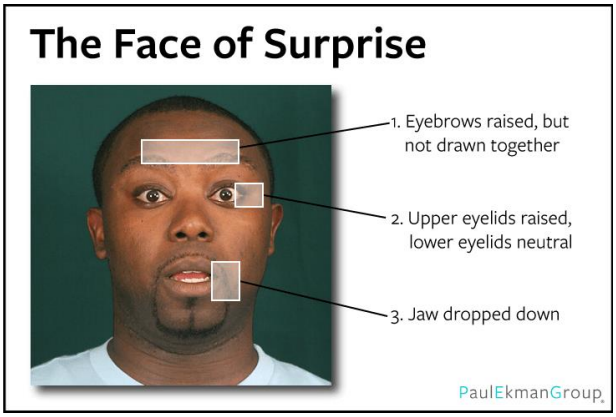


Рисунок 1.5 – Вираз обличчя під час здивування [5]

Страх - це головна емоція, яку викликають конкретні загрози в оточенні людей. Думка про машину, що швидко наближається, павук, який повзає по руці, будь-яке відчуття небезпеки, все це може викликати страх у людини [6].

Вираз обличчя (рис. 1.6) під часу страху часто плутають із подивом. Хоча обидва вирази демонструють чітко підняті брови, брови виразу страху є більш прямими і горизонтальними, тоді як з подивом вони підняті і вигнуті. Верхня повіка також піднімається вище від страху, ніж від здивування, оголюючи білу частину ока. Губи напружуються і розтягуються від страху, але більш відкриті і мляві від здивування [7].

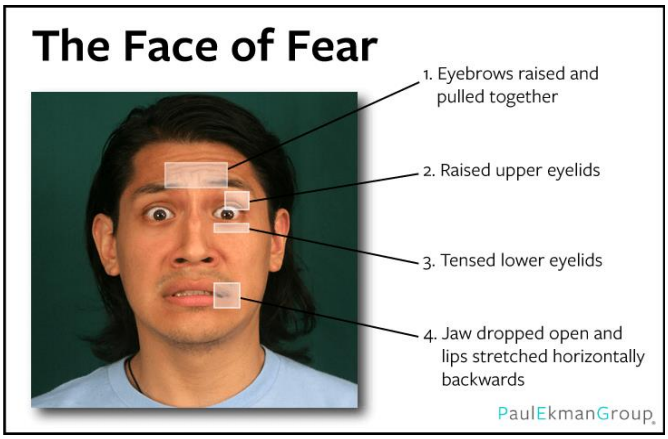


Рисунок 1.6 – Вираз обличчя під час страху [7]

Гнів – це емоція, коли хтось зробив щось погане, що шкодило або образило людину. Наприклад, хтось поширює чутки про вас, хтось навмисно

штовхає вас у метро, або хтось розмовляє з вами поблажливо. Зрозуміло, що слово "погано" в цьому реченні може означати ряд речей, відповідно, зашкодити репутації, завдати фізичної шкоди або образити. Один фактор завжди присутній: людина звинувачуєте іншу людину в тому, що сталося. У більшості випадків це відбувається тому, що вони навмисно мали на меті образити або нашкодити. Тільки якщо відчувається, що дія зовсім поза провиною, наприклад, якщо гальма раптом не спрацювали, то не відчувався б гнів [8].

Під час гніву (рис. 1.7) брови опущені і з’єднуються разом, з’являються вертикальні лінії між бровами, спостерігається звуження куточків губ, ніздрі можуть бути розширеними [9].

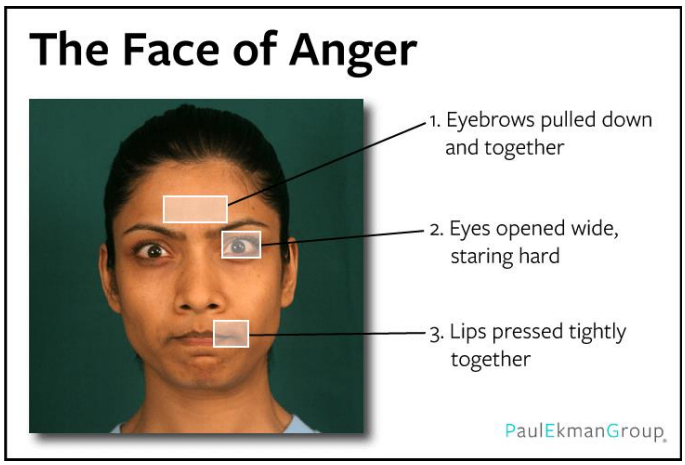


Рисунок 1.7 – Вираз обличчя під часу злості [9]

Радість - це специфічна емоція, яку відчувають люди, коли трапляються добрі речі. Вона включає почуття приємності разом із помірним рівнем збудження. Крім того, емоція часто поєднується з конкретним виразом обличчя: посмішкою [10]. Під час радості кутики губ в положенні назад і вгору (рис. 1.8), зморшки проходить від зовнішнього носа до зовнішньої губи, щоки підняті [11].

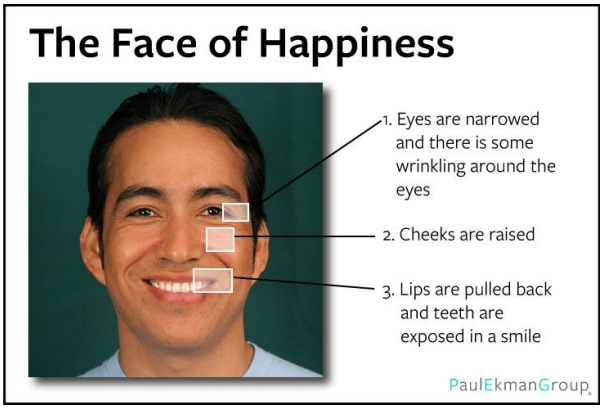


Рисунок 1.8 – Вираз обличчя під час радості [11]

Смуток - це негативна емоція і часто сприймається як протилежність щастя. Все, що ми отримуємо в житті, ми в кінцевому рахунку також втрачаємо [12]. Ми сумуємо, коли втрачаємо щось важливе, що може бути матеріальними цінностями (наприклад, втрата улюбленого взуття), фінансові активи (наприклад, втрата грошей), відносини (наприклад, через розпад), соціальні статус (наприклад, перестати захоплюватися) або задоволення (наприклад, відмова від куріння).

Один з дуже сильних і надійних ознак смутку - це накручування внутрішніх куточків брів (рис. 1.9). Куточки губ направлені вниз, нижня губа виділяється [13].

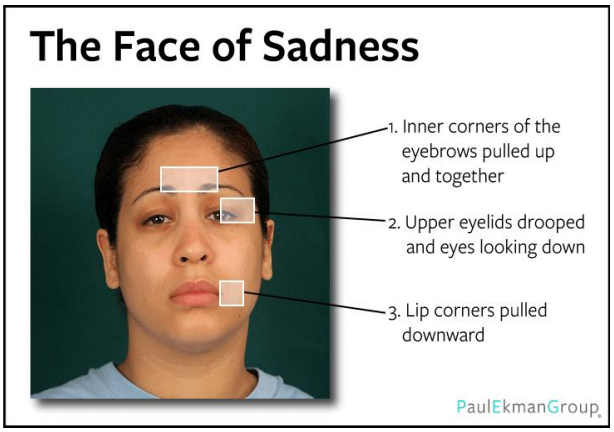


Рисунок 1.9 – Вираз обличчя під час смутку [13]

1.2. Розпізнавання елементів обличчя на зображенні

Основною елементарною одиницею в задачі розпізнавання є зображення. Комп'ютер інтерпретує зображення за допомогою матриці чисел (0…255). Для зображення у градаціях сірого це означає, що кожен піксель представляє певну яскравість (рис. 1.10).

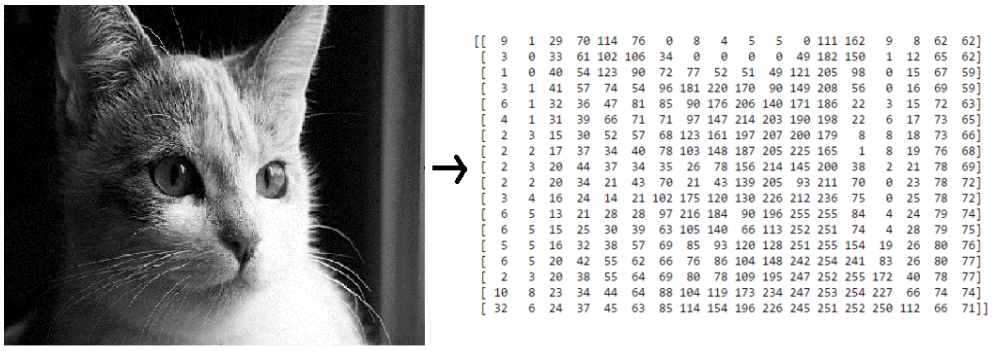


Рисунок 1.10 – Представлення зображення

У сірих зображеннях кожен піксель являє собою інтенсивність лише одного кольору. Іншими словами, він має один канал. У той час як у кольорових зображеннях (рис. 1.11) маємо 3 канали RGB (червоний, зелений, синій).

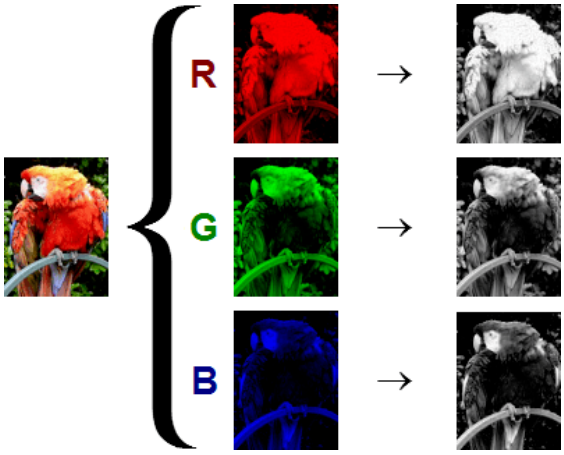


Рисунок 1.11 – RGB представлення

У цьому випадку ми матимемо 3D – матрицю. У нас є одна матриця для кожного каналу. Наприклад, перша матриця являє собою червоний канал, тоді кожен елемент цієї матриці являє собою інтенсивність червоного кольору у цьому пікселі, також зелений і синій. Загалом, кожен піксель у кольоровому зображенні має три числа (від 0 до 255), пов'язані з ним. Ці числа представляють інтенсивність червоного, зеленого та синього кольорів у цьому конкретному пікселі.

Здебільшого, розпізнавання виразів обличчя в основному виконується в три основні етапи (рис. 1.12) [14]. Спочатку відбувається виявлення обличчя, а потім виявлення особливостей отриманої області зображення, і як результат маємо класифікацію на основі отриманих даних.

Основна потреба системи розпізнавання обличчя - це модуль, який використовується для виявлення обличчя. Далі визначаються функції, які використовуються для вибору та вилучення відповідних особливостей, таких як очі, брови, ніс і губи з обличчя. Останнім кроком є класифікація міміки, яка працює на основі отриманих відповідних ознак.

Існують різні методи отримання особливостей, такі як метод на основі зовнішнього вигляду, метод на основі геометричних даних, метод на основі текстури, тощо.

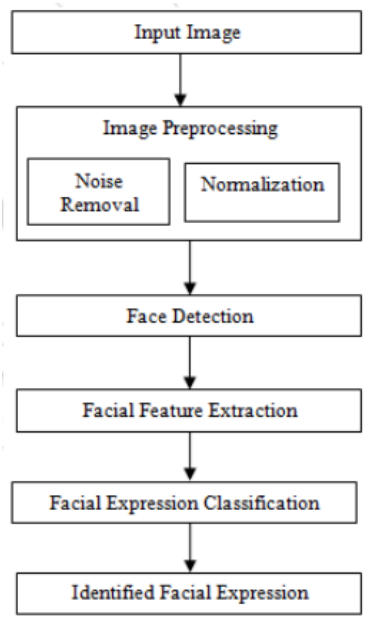


Рисунок 1.12 – Етапи у системі розпізнавання виразів обличчя [14]

Мажундер А. Бехера Л. Субраманій, В. К. та ін. представили систему розпізнавання обличчя, засновану на функції зовнішнього вигляду, використовуючи карту самоорганізації Kohonen (KSOM). Особливості зовнішнього вигляду витягуються за допомогою рівномірних локальних бінарних шаблонів (LBP) з однаково розділених блоків, нанесених на зображення обличчя [15].

Джічжен Ій; Ся Мао; Ліцзян Чень; Юлі Сюе; Порівняйте, А. та ін. запропонували алгоритм FER, використовуючи структурні характеристики та текстурну інформацію. По-перше, особливості балів були позначені моделлю активної зовнішності. По-друге, для усунення відмінностей між людьми були запропоновані три риси обличчя, які є коефіцієнтом співвідношення точкових відстаней, коефіцієнтом відношення кута з'єднання та параметром деформації шкіри. Нарешті, нейромережа радіальної основи функціонувала як класифікатор для FER [16].

Лі Ся запропонував метод класифікації виразів на основі SVM для дефектів традиційних методів класифікації. Він реалізує швидку класифікацію за допомогою відносно невеликої комбінації підкласифікаторів, зменшуючи похибку класифікації [17].

Мюнхун Сук, Прабхакаран, Б. та ін. запропонували систему, яка використовує набір SVM для класифікації 6 основних емоцій та нейтрального вираз обличчя, а також перевірки стану губ. Особливості виразу обличчя для розпізнавання емоцій отримувалися за допомогою Active Shape Model (ASM), що визначали орієнтири на обличчі, а потім динамічно генерували переміщення між нейтральним виразом та основними емоціями [18].

Урваші Бакші, Рохіт Сінгал запровадили методику розпізнавання людського обличчя штучним шляхом за допомогою нейтральної мережі DCT, PCA та SOM. Аналіз основних компонентів (PCA) - класичний і успішний метод зменшення розмірів. Дискретна косинальна трансформація (DCT) - добре

відома техніка стиснення, і карта самоорганізації (SOM) виступає в якості класифікатора і використовується для представлення простору обличчя.

Системи FER мають широке застосування в різних областях, таких як комп'ютерна взаємодія, системи охорони здоров’я та соціальний маркетинг. Однак аналіз виразів обличчя є надзвичайно складним через тонкі та швидкоплинні рухи людей, складне середовище зображення в реальних образах, відео. Існують три основні проблеми, викликані різницею освітленості, залежністю від предмета та зміною постави голови; вони широко впливають на продуктивність системи FER (рис. 1.13). Сучасні методи в системах FER ефективні для використання в контрольованих лабораторних умовах, але не для застосування в реальних ситуаціях [19].

Рисунок 1.13 – Приклад проблемних зображень

Вилучення особливостей є ключовим етапом у системі FER для виявлення емоцій. Гарне представлення функцій може гарантувати ефективний та точний процес розпізнавання. Існує два типи методів розпізнавання міміки виразів обличчя: геометричні та зовнішні. Перша стосується систем FER, які витягують локальні риси обличчя, включаючи форму, положення та кути між різними елементами обличчя, тобто вухом, оком, ротом та носом, і вектор зображених зображень ґрунтується на геометричних залежностях. Друге стосується систем FER, які описують зовнішній вигляд і використовують

текстурну інформацію обличчя як особливість вектора. Підходи, що базуються на зовнішньому вигляді, можуть отримати більш високий показник розпізнавання та користуються більшою популярністю, ніж методи, засновані на геометрії, оскільки це складне завдання знайти ефективні та належні геометричні особливості в не обмежених середовищах та реальних програмах.

Базуючись на попередньому аналізі об’єктом дослідження є зображення. Предмет дослідження - моделі, методи, підходи та засоби для розпізнавання емоційних станів.

1.3 Аналіз інформаційного забезпечення для розпізнавання емоцій

На даний момент існує велика кількість застосувань для розпізнавання емоцій.

FaceReader - це інструмент, здатний автоматично аналізувати міміку, надаючи користувачам об'єктивну оцінку емоцій людини (рис. 1.14). Даний застосунок надає змогу виявляти: радість, сум, злість, здивування, страх, огиду, зневагу та спокій. Також розділяє емоції на позитивні та негативні. Дає визначення стану частин обличчя: очі (відкриті, закриті), рот (відкритий, закритий), брови (опущені, нейтральні, підняті). Визначає погляд (ліворуч, вперед або праворуч), допомагає визначити увагу. Дає характеристики обличчя, а саме: стать, вік та наявність окулярів, бороди та вусів [20].

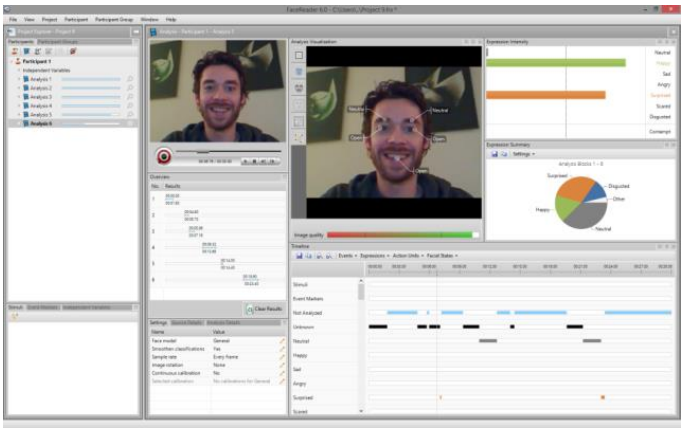


Рисунок 1.14 – Вигляд програми FaceReader [20]

Sightcorp - це AI з Амстердамського університету (UvA), що спеціалізується на аналізі та розпізнаванні облич (рис. 1.15). Завдяки комп'ютерному зору та глибокому навчанню програма може аналізувати обличчя у зображеннях, відео та реальних умовах життя.

Аналіз обличчя включає такі функції, як розпізнавання емоцій, виявлення віку, виявлення статі, уваги та відстеження поглядів очей [21].

Програмне забезпечення може визначити 7 основних емоцій на основі положення та руху м'язів обличчя. Кожен аналіз дає оцінку різних типів емоцій: гнів, огида, страх, радість, здивування, горе та нейтральність. Додаток виявляє різні вирази обличчя з точністю до 75%. Загальний настрій можна оцінити з точністю до 92% [22].

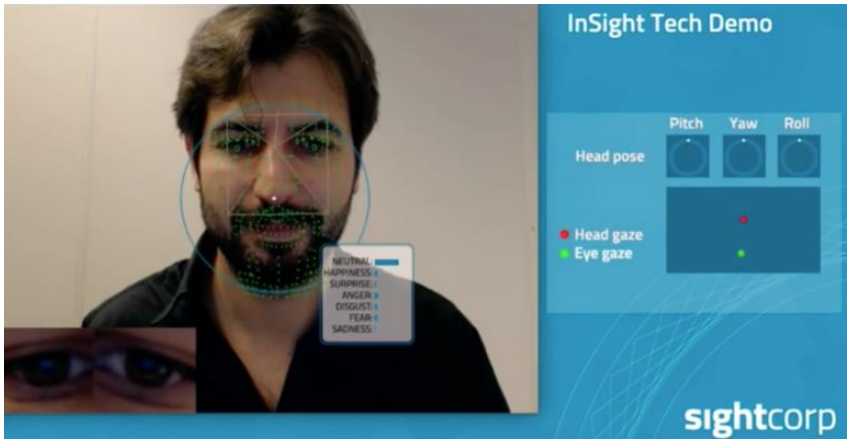


Рисунок 1.15 – Вигляд програми Sightcorp [22]

Під час розробки програми потрібно враховувати переваги і недоліки вже існуючого програмного забезпечення. Серед недоліків даних систем можна виділити:

– складні, громіздкі математичні моделі;
– велика обчислювальна складність;
– низька ефективність при застосуванні систем в реальних умовах.

1.4 Постановка задачі

Метою роботи є створення і дослідження моделі мімічних проявів емоційного стану людини, розробка інформаційної системи для розпізнавання мімічних проявів емоцій.

Для досягнення поставленої мети визначені наступні задачі дослідження:

– створення та дослідження моделі мімічних проявів емоцій;

– дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;

– розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;

– валідація та верифікація моделі, підходу та інформаційної системи.

Створення моделі полягає у пошуку максимально простого, але в той же час максимально ефективного відображення емоційного стану у вигляді параметрів моделі, за допомогою яких можливо розв’язати задачу розпізнавання емоційношо стану за обличчям людини. Проведені дослідження, валідація та веріфікація моделі повтні показати спроможність у запропонованій моделі розв’язати поставлену залачу.

Дослідження підходів полягає у підборі такого підходу, який надає найкраще розділення даних, та створенні необхідних модулів для дослідження.

Розробка інформаційної системи полягає у створенні системи, яка підтверджує обрані модель та підхід.

Валідація визначає відповідність між очікуваними та отриманими, рнзультатами. Верифікація – перевірка компонентів системи на задекларовані вимоги.

1.5 Висновки до розділу 1

Задача розпізнавання емоцій є актуальним напрямком досліджень. В існуючих підходах використовуються складні моделі, які потребують значних обчислювальних потужностей, а також підготовки зображень під певні умови, а в реальних умовах отримати достатній відсоток коректного розпізнавання доволі складно. Тому, дослідження в даному напрямку є актуальним, особливо якщо враховувати прості моделі для швидкої роботи.

Розділ 2
Розробка інформаційної технології розпізнавання емоцій

2.1 Загальний опис інформаційної технології розпізнавання емоцій

Для вирішення поставлених завдань використовуються наступні методи:

- метод Віоли - Джонса;
- facemarkLBF;
- метод отримання вектора ознак;
- Multidimensional scaling;
- SMACOF;
- метод верифікації емоції.

Метод Віоли - Джонса – алгоритм, що дозволяє виявити об'єкти на зображенні в реальному часі. Його запропонували Паул Віола і Майкл Джонс в 2001 році. Хоча алгоритм може розпізнавати різні класи зображень, основним завданням при його створенні було виявлення облич. У методу є безліч реалізацій, в тому числі в складі бібліотеки комп'ютерного зору OpenCV. Алгоритм знаходить обличчя з високою точністю і низьким числом помилкових спрацьовувань.

За ознаки для алгоритму розпізнавання авторами були запропоновані ознаки Хаара.

В задачі розпізнавання обличчя, використовується загальне спостереження, що серед усіх облич область очей темніша за область щік. В підході використовуються маски (рис. 2.1), що складаються з світлих і темних областей. Кожна маска характеризується певним розміром світлих і темних областей, пропорціями, а також мінімальним розміром.

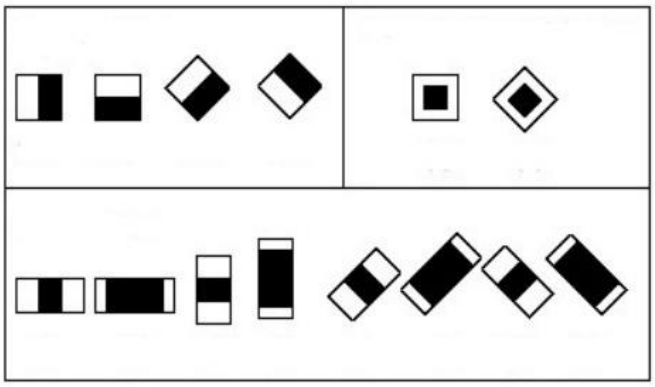


Рисунок 2.1 – Маски для каскаду Хаара [23]

Ознаки Хаара дають точкове значення перепаду яскравості по осі X і Y. Тому загальна ознака Хаара для розпізнавання осіб представляє набір двох суміжних прямокутників, які лежать вище очей і на щоках [23]. Ознака обчислюється за формулою:

$$F = X - Y, \tag{2.1}$$

де X – сума значень яскравості точок закритих світлою частиною, а Y – сума значень яскравості точок закритих темною частиною.

Для прискорення обрахунків в методі використовується інтегральне представлення зображення. Надає змогу швидко обрахувати суму пік селів довільного прямокутника. Інтегральне представлення зображення – це матриця, яка збігається розмірами з вхідним зображенням. Кожний елемент матриці містить суму інтенсивностей всіх пікселів, що знаходяться лівіше і вище даного елемента. Елементи матриці розраховуються за формулою:

$$L(x, y) = \sum_{i=0,j=0}^{i\leq x, j\leq y} I(i, j), \tag{2.2}$$

де $I\ (i, j)$ – яскравість пікселя вихідного зображення.

Кожен елемент матриці $L(x, y)$ є сумою пікселів в прямокутнику від $(0,0)$ до (x, y), тобто значення кожного пікселя (x, y) дорівнює сумі значень усіх

пікселів лівіше і вище даного пікселя *(x, y)*. Розрахунок матриці займає лінійний час, пропорційне числу пікселів в зображенні, тому інтегральне зображення прораховується за один прохід. Розрахунок матриці можливий за формулою:

$$L(x,y) = l(x,y) - L(x-1,y-1) + L(x,y-1) + L(x-1,y), \quad (2.3)$$

За такою інтегральною матрицею можна дуже швидко вирахувати суму пікселів довільного прямокутника(рис. 2.2), довільної площі. Нехай в прямокутнику *ABCD* є цікавий для нас об'єкт *D*:

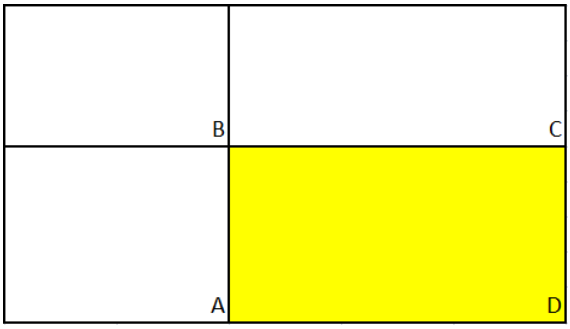


Рисунок 2.2 – Довільний прямокутник [23]

Суму всередині прямокутника можна виразити через суми і різниці суміжних прямокутників за формулою:

$$S(ABCD) = L(A) + L(C) - L(B) - L(D), \quad (2.4)$$

Узагальнена схема алгоритму навчання виглядає наступним чином. Є тестова вибірка зображень. Розмір тестової вибірки близько 10 000 зображень. Приклад навчальних зображень облич наведено на рис. 2.3. Алгоритм навчання працює з зображеннями в відтінках сірого.

Рисунок 2.3 – Обличчя із тестової вибірки [23]

При розмірі тестового зображення 24 на 24 пікселя кількість конфігурацій однієї ознаки близько 40 000 (залежить від мінімального розміру маски). Сучасна реалізація алгоритму використовує близько 20 масок. Для кожної маски, кожної конфігурації тренується такий слабкий класифікатор, який дає найменшу помилку на всій тренувальній базі. Він додається в базу даних. Таким чином алгоритм навчається. І на виході алгоритму виходить база даних з Т слабких класифікаторів. Узагальнена схема алгоритму навчання (рис. 2.4):

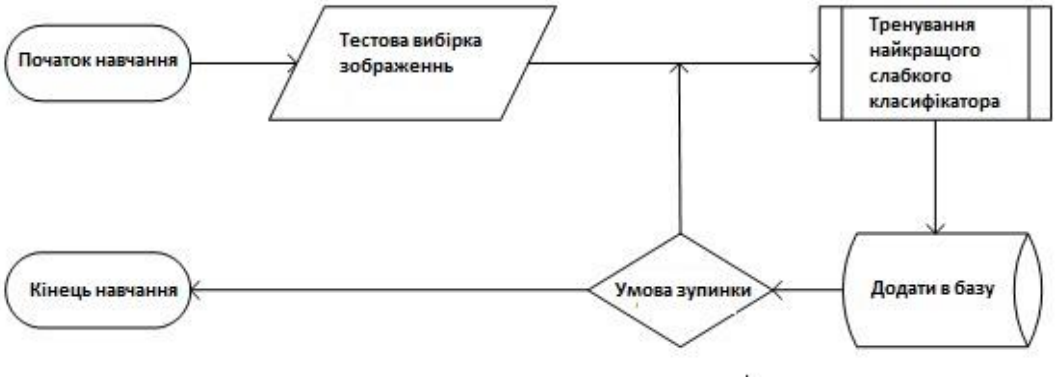


Рисунок 2.4 – Схема алгоритму навчання [23]

Для алгоритму необхідно заздалегідь підготувати тестову вибірку з l зображень, що містять шуканий об'єкт і n що не містять. Тоді кількість всіх тестових зображень буде:

$$n = l + m \tag{2.5}$$

$$X = \{x_1, x_2, \dots, x_n\}, \tag{2.6}$$

де X - множина всіх тестових зображень, де для кожного заздалегідь відомо чи присутній шуканий об'єкт чи ні і відображено в множині Y.

$$Y = \{y_1, y_2, \dots, y_n\}, \tag{2.7}$$

де $y_i = \begin{cases} 1, \text{об}'\text{єкт присутній на зображенні } x_i \\ 0, \text{інакше} \end{cases}$

Під ознакою j мається на увазі структура виду:

$$j = \{\text{маска}, \text{положення}, \text{розмір}\} \tag{2.8}$$

Тоді відгуком ознаки буде $f_j(x)$, який обчислюється як різниця інтенсивностей пікселів у світлій і темній областях. Слабкий класифікатор має вигляд:

$$h_j(x) = \begin{cases} 1, p_j, f_j(x) < p_j \Theta_j \\ 0, \text{інакше} \end{cases} \tag{2.9}$$

Завдання слабкого класифікатора - вгадувати присутність об'єкта в більше ніж 50% випадків. Створення дуже сильного класифікатора складається з T слабких класифікаторів і має вигляд:

$$H(x) = \begin{cases} 1, \sum_{t=1}^{T} a_t \; h_{j(t)}\,(x) \geq \frac{1}{2} \sum_{t=1}^{T} a_t, \\ 0, \text{інакше} \end{cases} \quad \text{и} \tag{2.10}$$

Цільова функція навчання має наступний вигляд:

$$T, h_{j(1)} h_{j(2)}, \dots, a_1 \dots, , a_T = argmin \sum_{i=1}^{n} |H(x_i, T, h_{j(1)}, , h_{j(T)}, a_1, \dots, a_T)| \tag{2.11}$$

Перед початком навчання ініціалізуються ваги *w (q, i)*, де *q* - номер ітерації, *i*-номер зображення.

$$w_{1,i} = \begin{cases} \frac{1}{2l}, y_i = 1 \\ \frac{1}{2m}, y_i = 0 \end{cases} \quad (2.12)$$

Після процедури навчання вийде *T* слабких класифікаторів і *T* значень.

$$p_j = \{1, -1\}, \Theta_j = [\Theta_{min}, \Theta_{max}] \quad (2.13)$$

На кожній ітерації циклу відбувається оновлення ваг так, що їх сума буде дорівнює 1. Далі для всіх можливих ознак відбувається підбір таких значень *p, θ, j* що значення помилки *e_j* буде мінімально на цій ітерації. Отримана ознака *J (t)* (на кроці *t*) зберігається в базу слабких класифікаторів, оновлюються ваги і обчислюється коефіцієнт *a_t*.

Після навчання на тестовій вибірці є навчена база знань з Т слабких класифікаторів. Для кожного класифікатора відомі: ознака Хаара, що використовується в цьому класифікаторі, його положення всередині вікна розміром 24х24 пікселя і значення порога *E*.

На вхід алгоритму надходить зображення *I (r, c)* розміром *W x H*, де *I (r, c)* - яскравість зображення. Результатом роботи алгоритму є безліч прямокутників *R (x, y, w, h)*, що визначають положення осіб в оригінальному зображенні *I*.

Алгоритм сканує зображення I на декількох масштабах, починаючи з базової шкали: розмір зображення 24х24 пікселя і 11 масштабів, при цьому кожен наступний рівень в 1.25 рази більше попереднього.

При обробці геометричних особливостей обличчя важливим фактором є положення голови. Як було описано у статті «Face image transformations for correct recognition problems solving»[24]: некоректне положення голови здатне

значно змінювати позиції ключових точок обличчя, і як наслідок відстань між ними. Тому фільтрація обличчя перед обробкою ключових точок є важливим фактором, адже дозволяє вилучати погані екземпляри вибірки, що дає змогу формувати більш точні дані. Також даний підхід може бути застосований для фільтрації обличь в реальному часі, для уникнення надлишкової обробки некоректних зображень.

Фільтрація реалізується за допомогою перевірки ознак, які описують коректність відображення обличчя (рис. 2.5).

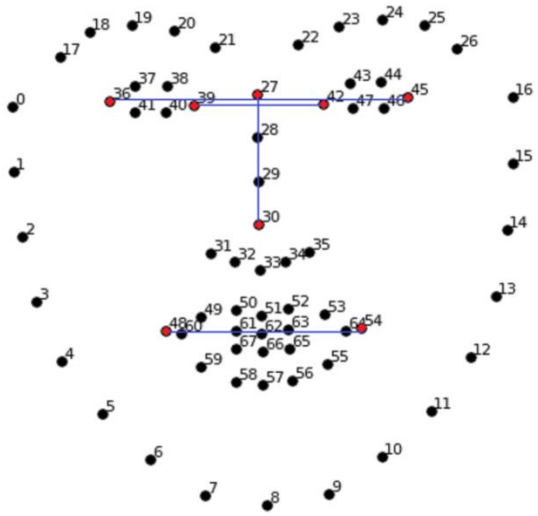


Рисунок 2.5 – Взаємозв'язок між точками обличчя [24]

Ознака перпендикулярності ліній носа і очей, а також симетрії кутових точок очей. Лінії носа визначаються точками 27 і 30, якщо в КСК координати цих точок $X_{27} = X_{30} = 0$. Тобто лінія носа співпадає з віссю Y.

Ознака паралельності ліній очей і рота. Визначається точками 48 і 54. Ознака виконується, якщо в КСК $Y_{48} - X_{54} = 0$.

Ознака симетрії кутових точок губ. Визначається точками 48 і 54. Виконується, якщо в КСК $X_{48} + X_{54} = 0$

Ознака за якої кутові точки очей знаходяться на одній прямій. Прямі визначаються точками 36 і 45, 39 і 42. За умови виконання ознаки перпендикулярності ліній носа і очей, достатньо визначити в КСК ознаку $Y_{36} - Y_{39} = 0$.

Оскільки абсолютної рівності нулю не буває, вибирається мінімально допустима величина.

FaceLandmarksLBF

Спочатку визначається(навчається) функція локального відображення для генерації локальних двійкових особливостей (ЛДО). Далі отримані ЛДО об'єднуються в функцію Φ^t. Потім визначається(навчається) матриця W^t за допомогою лінійної регресії [25]. Даний процес повторюється в каскадному режимі (рис. 2.6).

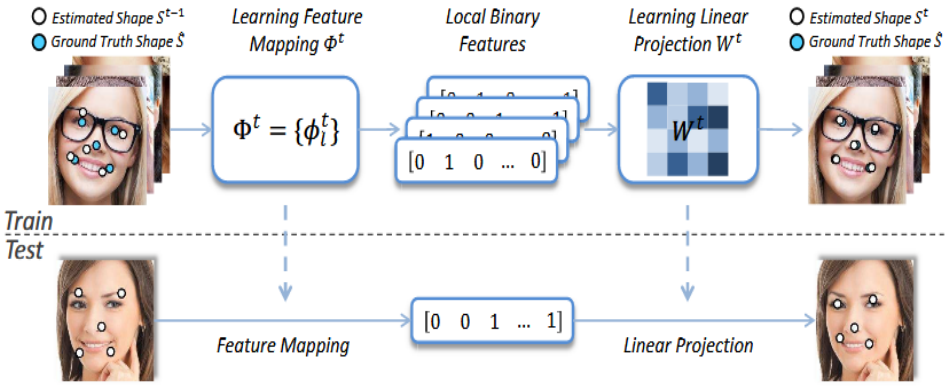


Рисунок 2.6 – Схема алгоритму навчання [25]

Визначення ЛДО.

Функція Φ^t складається з ЛДО і має вигляд $\Phi^t = [\phi_1^t, \phi_2^t, \dots, \phi_L^t]$.

Кожна ознака визначається окремо. ϕ_1^t є приростом основної форми ΔS^t:

$$min_{w^t,\phi_1^t} \sum_{i=1} \left|\left|\pi_l \, o \, \Delta S^t - \, w_l^t \phi_l^t\left(I_l, S_i^{t-1}\right)\right|\right|_2^2 , \qquad (2.14)$$

де i перебирає всі зразки навчання, оператор π_l отримує 2 елементи $(2l-1, 2l)$ з вектору ΔS^t, $\pi_l \, o \, \Delta S^t$ є головним 2D зміщенням орієнтиру в і ітерації навчання.

Для навчання функції ϕ_1^t використовується метод машинного навчання Random forest.

Під час тестування зразок проходить по деревах, поки не досягне одного вузла для кожного дерева. Виходом Random forest є підсумовування виходів, що зберігаються в цих вузлах. Припустимо, що загальна кількість листових вузлів дорівнює *D*, вихідна інформація може бути переписана як:

$$w_l^t \phi_l^t\left(I_{l,} S_i^{t-1}\right), \tag{2.15}$$

де w_l^t це $2 * D$ в якій кожен стовпець є 2D вектор, що зберігається у відповідному листовому вузлі, ϕ_l^t це є D - мірний двійковий вектор. Для кожного виміру у ϕ_l^t його значення дорівнює 1, якщо тестовий зразок досягає відповідного вузла та 0 якщо ні. Тому ϕ_l^t є дуже розріджений двійковий вектор. Кількість не нульових елементів у ϕ_l^t дорівнює кількості дерев, якщо набагато менше за *D*.

ϕ_l^t (рис. 2.6) кодує певну область у двійкову особливість. Всі ЛДО об'єднуються у двовимірні двійкові особливості.

Random forest використовується, як функція локального відображення.

Кожна отримана ЛДО вказує чи містить вхідне зображення деякі локальні шаблони.

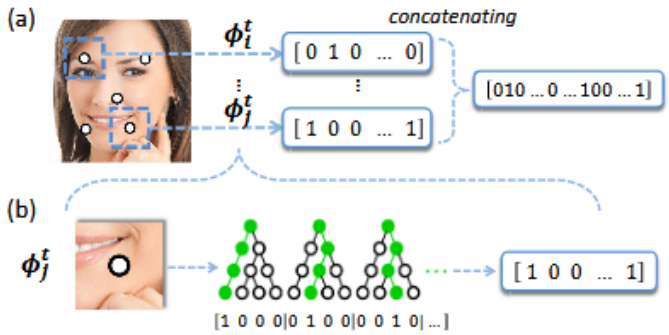


Рисунок 2.7 – Схема ЛДО [25]

Визначення W_t

Після виконання random forest отримуємо w_l^t. І відкидаємо отриманий локальний вихід w_l^t. Замість цього ми об'єднаємо двійкові функції в глобальну

функцію відображення об'єктів ϕ^t і визначає W^t шляхом мінімізації наступної цільової функції:

$$min_{W^t} \sum_{i=1}^{N} ||\Delta S_i^t - W^t \Phi_t\left(l_i, S_i^{t-1}\right)||_2^2 + \lambda|| W_t ||_2^2, \qquad (2.16)$$

де перший член є регресійною ціллю, другий член - регуляризація $L2$ в W_t, λ - контролює силу регуляризації. Регуляризація необхідна, оскільки розмірність особливостей дуже висока. Для 68 контрольних точок розмірність Φ_t є близько 100’000. Без регуляризації ми спостерігаємо істотне пристосування. Оскільки бінарні риси дуже розріджені, використовується метод подвійного координатного спуску, щоб мати справу з такою великомасштабною розрідженою лінійною системою. Оскільки цільова функція квадратична по відношенню до Φ_t можна досягти глобального оптимуму.

Глобальне “перенавчання” або “передача навчань” значно покращує продуктивність. З одного боку, локально отриманий вихідний лісовий маркер (random forest) є шумним, оскільки кількість навчальних проб в листовому вузлі може бути недостатньою. З іншого боку, глобальна регресія може ефективно забезпечити глобальну фігуру і зменшити локальні помилки, зумовлені оклюзією та неоднозначним місцевим виглядом.

Принцип локальності

Використовується два важливих методу регуляції в вивченні ознак, як принцип керованості локальності: 1) навчаємо ліс для кожного орієнтира незалежно; 2) ми розглянемо лише особливості пікселів у локальній області орієнтира.

Використання локального регіону. Припустимо, ми хочемо передбачити зсув f з орієнтиром $single$ і вибираємо особливості з локальної області з радіусом. Інтуїтивно оптимальним радіусом слід вважати розподіл. Оскільки всі навчальні зразки розкидані широко, ми повинні використовувати більше.

Для вивчення взаємозв'язку між розподілом і оптимальним радіусом для орієнтира синтезуються райони тренування і випробування зразків, для яких виконується розподіл Гаусса з різними стандартними відхиленнями. Для кожної дистрибації експериментально визначаємо оптимальні області радіуса (в термінах випробувальної похибки) за допомогою тренувальної регресії в різних радіусах. Використовується ті ж параметри лісу (перегляд і кількість дерев), як і в каскадному навчанні. Повторюється для всіх орієнтирів і береться середній радіус оптимального регіону.

Результати трьох розподілів, які були встановлені. - 0,05, 0,1 і 0,2 (нормалізована відстань за розміром прямокутника) (рис. 2.8). Оптимальними радіусами є 0,12, 0,21 і 0,39.

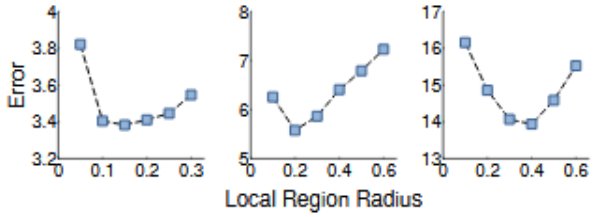


Рисунок 2.8 – Розподіли

Враховуючи обмеженість обчислень більш ефективним є розгляд лише особливостей кандидатів у локальному регіоні замість глобального зображення обличчя. У каскадному навчанні, на кожному етапі, шукається найкращий радіус регіону (з 10 дискретних значень) шляхом перехресної перевірки на наборі перевірок. Найкращі радіуси регіонів, знайдені на етапах 1, 3 і 5 (рис. 2.9). Очікується, що радіус поступово зменшується з ранніх стадій етапу, оскільки зміна форми обличчя зменшується під час каскаду.

Рисунок 2.9 – Результат кроків

В результаті роботи методу FacemarkLBF отримуються 68 специфічних точок обличчя (рис. 2.10).

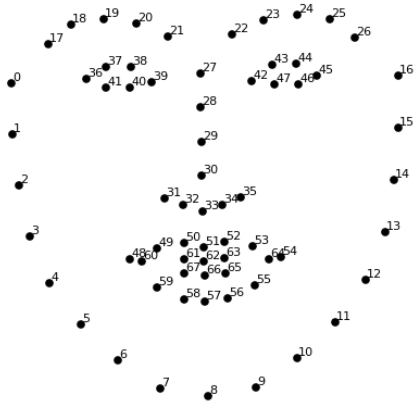


Рисунок 2.10 – Специфічні точки обличчя

Визначення 2D каркасу

Для зручності визначення ознак та нормалізації геометрії обличчя використовуємо користувацьку систему координат (КСК), вісь X якої проходить через відрізок між центрами очей, а вісь Y - перпендикулярна відрізку через його середину, в напрямку вгору. Координати КСК (від -1 до +1) нормалізовані - співвіднесені з відстанню між середніми точками очей (рис. 2.11).

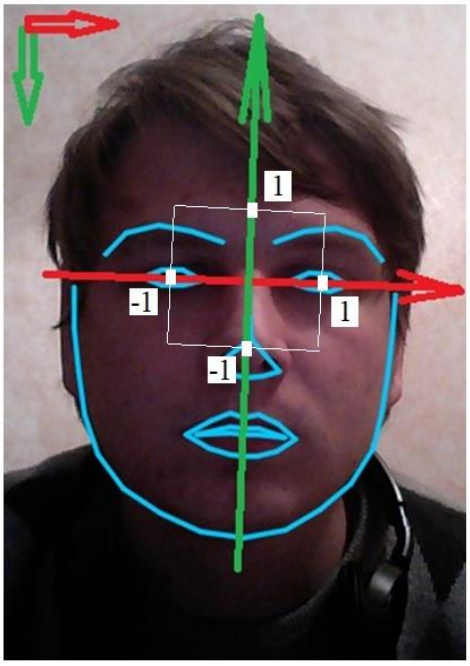


Рисунок 2.11 – Система координат на основі обличчя [24]

Координати середніх точок лівого і правого очей (рис. 2.12).

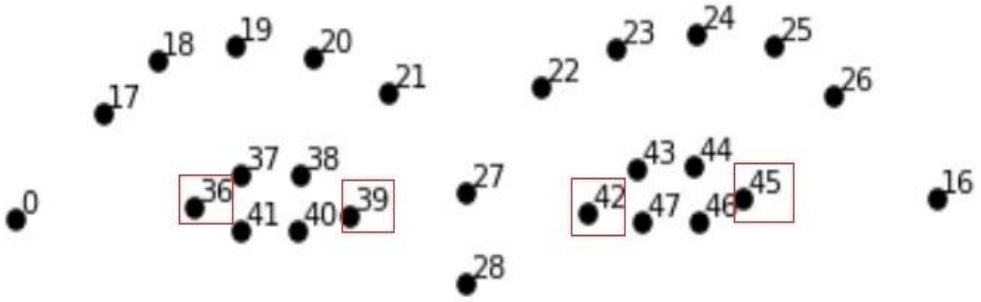


Рисунок 2.12 – Точки обличчя [24]

$$XL = \frac{(X45+X42)}{2} \quad (2.17)$$

$$YL = \frac{(X45 + X42)}{2} \quad (2.18)$$

$$\mathrm{XR} = \frac{\mathrm{X39+X36}}{2} \quad (2.19)$$

$$\mathrm{YR} = \frac{\mathrm{Y39+Y36}}{2} \quad (2.20)$$

Початок КСК (рис. 2.13):

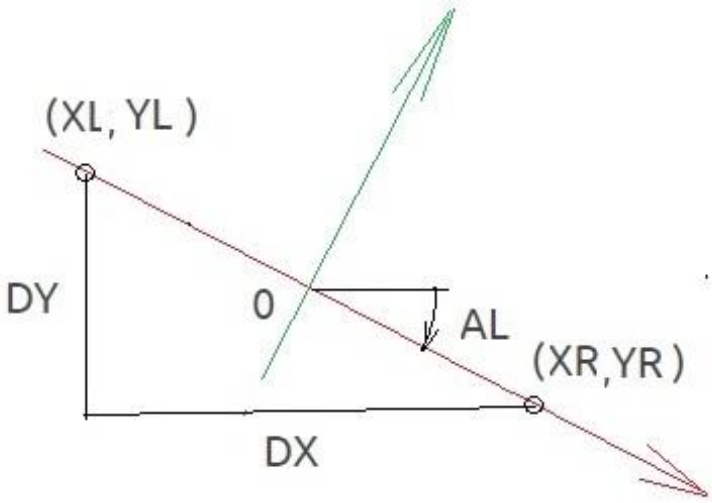


Рисунок 2.13 – Початок КСК [24]

$$X0 = \frac{(XL+XR)}{2} \quad (2.21)$$

$$Y0 = \frac{(YL+YR)}{2} \quad (2.22)$$

Відстані між середніми точками очей уздовж осей *X* і *Y*:

$$DX = XR - XL \tag{2.23}$$

$$DY = YR - YL \tag{2.24}$$

Відстань між центрами очей:

$$L = sqrt(DX^2 + DY^2) \tag{2.25}$$

Тригонометричні функції кута повороту КСК:

$$\sin_{AL} = \frac{DY}{L} \tag{2.26}$$

$$\cos_{AL} = \frac{DX}{L} \tag{2.27}$$

Переходимо від координат віконної СК до координат в КСК, використовуючи параметри *X0*, *Y0*, *L*, sin_{AL}, cos_{AL}:

$$X_{User0} = \frac{2(X_{window} - X0)}{L} \tag{2.28}$$

$$Y_{User0} = -\frac{2(Y_{window} - Y0)}{L} \tag{2.29}$$

$$X_{User} = X_{User0} * cos_{AL} - Y_{User0} * sin_{AL} \tag{2.30}$$

$$Y_{User} = X_{User0} * sin_{AL} - Y_{User0} * cos_{AL} \tag{2.31}$$

Отримання вектора ознак

Основні частини обличчя, які використовуються для визначення емоцій: брови, очі, губи. В результаті маємо вектор ознак: $v = \{x_1, x_2, x_3\}$.

Для визначення відстані між точками використовуємо формулу:

$$distance = \sqrt{(p1x - p2x)^2 + (p1y - p2y)^2} \tag{2.32}$$

Визначення значення ока:

$$pointA = (flm37 + flm38)/2\ , \tag{2.33}$$

де flm37 – 37 точка на обличчі, flm38 – 38 точка на обличчі.

$$pointB = (flm40 + flm41)/2\ , \tag{2.34}$$

де flm40 – 40 точка на обличчі, flm41 – 41 точка на обличчі.

$$\text{eye}fd = distance(pointA, pointB) \tag{2.34}$$

$$eyesd = distance(flm36, flm39), \tag{2.35}$$

де flm36 – 36 точка на обличчі, flm39 – 39 точка на обличчі.

$$eyevalue = \frac{eyefd}{eyesd}/EYEMAX, \tag{2.36}$$

де EYEMAX – максимальне доступне значення відстані для ока.

Визначення значення брів:

$$\text{eye}browfd = distance(eyesd, flm19), \tag{2.37}$$

де flm19 – 19 точка на обличчі.

$$\text{eye}browsd = distance(flm36, flm45), \tag{2.38}$$

де flm36 – 36 точка на обличчі, flm45 – 45 точка на обличчі.

$$eyebrowvalue = \frac{eyefd}{eyesd}/EYEBROWMAX, \tag{2.39}$$

де $EYEBROWMAX$ – максимальне доступне значення відстані для брів.

Визначення значення губ:

$$\text{lips}sd = distance(flm48, flm54), \tag{2.40}$$

де flm48 – 48 точка на обличчі, flm54 – 54 точка на обличчі.

$$lipsvalue = \frac{\frac{sd}{fd}}{\text{LIPSMAX}}, \tag{2.41}$$

де LIPSMAX - максимальне доступне значення відстані для губ.

Перетворення значення до значення моделі відбувається за наступними формулами:

$$\begin{cases} eyeval = (eyevalue\ -\ 0.0) * \frac{(0.2-0.0)}{(0.55-0.0)} +\ 0.0, \text{якщо } eyevalue < 0.55 \\ eyeval = (eyevalue - 0.55) * \frac{(0.6-0.4)}{(0.75-0.55)} +\ 0.4, \text{якщо } eyevalue < 0.75 \\ eyeval = (eyevalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} +\ 0.8, \text{якщо } eyevalue < 1.0 \end{cases} \tag{2.42}$$

$$\begin{cases} eyebv = (eyebrowvalue\ -\ 0.0) * \frac{(0.2-0.0)}{(0.65-0.0)} +\ 0.0, \text{якщо } eyebrowvalue < 0.55 \\ eyebv = (eyebrowvalue - 0.65) * \frac{(0.6-0.4)}{(0.75-0.65)} +\ 0.4, \text{якщо } eyebrowvalue < 0.75 \\ eyebv = (eyebrowvalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} +\ 0.8, \text{якщо } eyevbrowvalue < 1.0 \end{cases} \tag{2.43}$$

$$\begin{cases} lipsval = (lipsvalue\ -\ 0.0) * \frac{(0.2-0.0)}{(0.65-0.0)} +\ 0.0, \text{якщо } lipsvalue < 0.65 \\ lipsval = (lipsvalue - 0.65) * \frac{(0.6-0.4)}{(0.75-0.65)} +\ 0.4, \text{якщо } lipsvalue < 0.75 \\ lipsval = (lipsvalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} +\ 0.8, \text{якщо } lipsvalue < 1.0 \end{cases} \tag{2.44}$$

Multidimensional scaling

Багатовимірне масштабування - техніка, що часто використовуються в інформаційній візуалізації для дослідження схожості та відмінності даних. Дані, що використовуються для багатовимірного масштабування, є відмінностями між парами об'єктів. Головною метою MDS є представити ці відмінності як відстані між точками в низькому розмірному просторі, такі, щоб відстані максимально відповідали відмінностям [26].

MDS використовує той факт, що координатну матриця *X* можна отримати за допомогою розкладання власного значення від *B=XX'*. Матриця *B* може бути обрахована з матриці *D* за допомогою подвійного центрування.

Спочатку встановлюється квадратна матриця близькості $D^{(2)} = [d_{ij}^2]$. Далі застосовується подвійне центрування

$$B = -\frac{1}{2} J D^{(2)} J \tag{2.44}$$

$$J = I - \frac{1}{n} 11', \tag{2.45}$$

де n –кількість обє'ктів.

Наступним кроком визначаються найбільші власні значення $\lambda_1, \lambda_2, \dots, \lambda_m$ матриці *B,* а також відповідні власні вектори $e_1, e_2, \dots, e_m$. Де m – розмірність простору.

Тепер обраховується результуюча матриця

$$X = E_m \Lambda_m^{1/2}, \text{ де} \tag{2.46}$$

E_m- матриця власних векторів, Λ_m - діагональна матриця власних значень матриці *B.*

SMACOF

Мажоризація стресу - це стратегічна оптимізація, що використовується в багатомірному шкалюванні, де для набору з n елементів розмірності m шукається конфігурація X n точок в r - розмірному просторі, який мінімізує функцією мажоризації $\sigma(X)$. Зазвичай r дорівнює 2 або 3, тобто (n x r) матриця X перераховує точки в 2- або 3-мірному евклідовому просторі, так що результат може бути відображений візуально. Функція σ є функцією втрат, яка вимірює квадрат різниці між ідеальною (m-мірною) відстанню в r-мірному просторі.

Вхідні дані MDS, як правило, є n × n матриця відмінностей на основі спостережуваних даних. Задача, яка вирішується – знайти $\{i, j\} = 1, ..., n$ точки в маломірному евклідовому просторі таким чином, що відстані між точками наближаються до заданих відмінностей δ_{ij} [27]. Таким чином необхідно знайти n × p матрицю X така де $d_{ij}(X) \approx \delta_{ij}$, де

$$d_{ij}(X) = \sqrt{\sum_{s=1}^{p}(x_{is} - x_{js})^2} \tag{2.47}$$

Індекс s = 1, ..., P позначає кількість вимірів в евклідовому просторі. Елементи X називаються конфігураціями об'єктів. Таким чином, кожен об'єкт масштабується в p-мірному просторі таким чином, щоб відстані між точками в просторі збігалися, а також спостерігалися відмінності. Графічно представлені результати конфігурації демонструють координати на графіку конфігурації.

Тепер ми робимо задачу оптимізації більш точною, визначаючи стрес $\sigma(X)$ наступним чином:

$$\sigma(X) = \sum_{i<j} w_{ij} \left(\delta_{ij} - d_{ij}(X)\right)^2 \tag{2.48}$$

W відома, як матриця ваг, розмірності n × n. Визначається наступним чином:

$$\sum_{i<j} w_{ij}\delta_{ij} = n(n-1)/2 \quad (2.49)$$

Візуалізація вхідних даних у двомірний простір: $R^m \to R^2$.

За допомогою SMACOF отримуємо множину точок: $x'(i) \in R^2, i = 1, n$.

Після мануального розподілення отримуємо множину точок: $x'(i,j) \in R^2, i = 1, l$, j = 1,m , де l – кількість ліній.

Пошук точки на лінії

Маємо $StartPoint = \{x_0, y_0\}$, як початок лінії та $EndPoint = \{x_1, y_1\}$, як кінець лінії. Визначається відстань d_t від нової точки до початку лінії.

Довжина лінії обчислюється наступним чином:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad (2.50)$$

Знаходимо співвідношення відстаней:

$$t = \frac{d_t}{d} \quad (2.51)$$

Тоді координата *x* нової точки має значення:

$$x_t = (1-t)x_0 + tx_1 \quad (2.52)$$

Координата *y* нової точки має значення:

$$y_t = (1-t)y_0 + ty_1 \quad (2.53)$$

Отримуємо $NewPoint = \{x_t, y_t\}$.

Обернений SMACOF надає: $x^L(k,j) \in R^m, i = 1, m;\ k = 1, l$. За множиною точок $x^L(k,j) \in R^m, i = 1, m;\ k = 1, l$, які розташовані на відповідних гіперплощинах, для визначення коефіцієнти цих гіперплощин,

формуємо системи лінійних алгебраїчних рівнянь. Таких систем буде стільки, скільки ліній у R^2 . Для і-ї гіперплощини система рівнянь матиме вигляд:

$$\left\{\begin{array}{c} w_1 x_1^L(i,1) + w_2 x_2^L(i,1) + \ldots + w_m x_m^L(i,1) + b = 0; \\ w_1 x_1^L(i,2) + w_2 x_2^L(i,2) + \ldots + w_m x_m^L(i,2) + b = 0; \\ \ldots \\ w_1 x_1^L(i,m) + w_2 x_2^L(i,m) + \ldots + w_m x_m^L(i,m) + b = 0. \end{array}\right\} \quad (2.54)$$

$$w_1 = (-1^{1+1}) \begin{pmatrix} x_2^L(i,1) & x_3^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_2^L(i,2) & x_3^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_2^L(i,m) & x_3^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_1 \quad (2.55)$$

$$w_2 = (-1^{1+2}) \begin{pmatrix} x_1^L(i,1) & x_3^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_3^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_1^L(i,m) & x_3^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_2 \quad (2.56)$$

$$w_k = (-1^{1+k}) \begin{pmatrix} x_1^L(i,1) & x_{k-1}^L(i,1) & x_{k+1}^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_{k-1}^L(i,2) & x_{k+1}^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & & \\ x_1^L(i,m) & x_{k-1}^L(i,m) & x_{k+1}^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_k \quad (2.57)$$

$$b = (-1^{2+m}) \begin{pmatrix} x_1^L(i,1) & x_2^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_2^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_1^L(i,m) & x_2^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} \quad (2.58)$$

2.2 Модель емоційного стану

Найбільш вираженими частинами обличчя є: брови, очі, губи [28] (рис. 2.14).

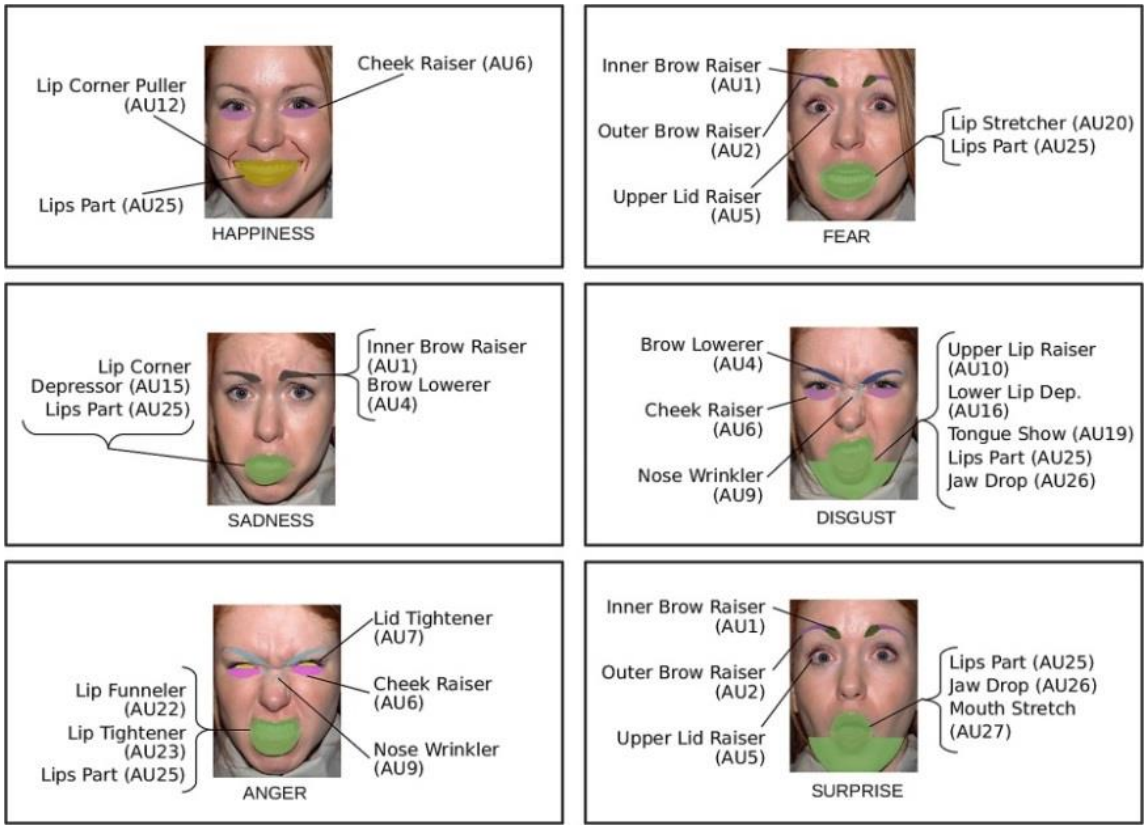


Рисунок 2.14 – Сприяння частини обличчя для розпізнавання [28]

У роботі було досліджено дрібний рівень, на фізичні особливості якого найбільше покладаються при розшифровці міміки. В експерименті окремі обличчя, що виражають основні емоції за Екманом, були заховані за маскою з 48 плиток, яку послідовно розкривали. Лицьові частини, що мають найвищу діагностичну цінність для ідентифікації емоційного стану, зазвичай розташовуються в областях, відповідних одиницям дії з системи кодування обличчя (FACS).

Система кодування рухів обличчя - система, що базується на спостереженнях людини, розроблена для того, щоб визначити тонкі зміни риси обличчя (рис. 2.15) [29].

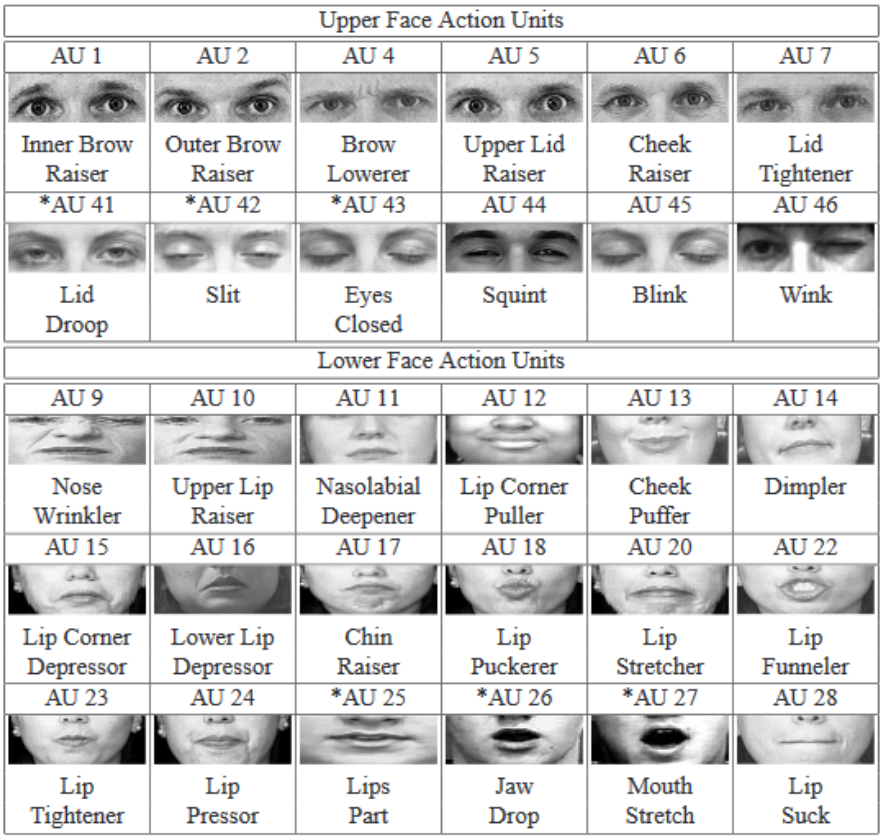


Рисунок 2.15 – Система кодування рухів обличчя [29]

Виділені частини обличчя можуть перебувати у трьох станах [30] (рис. 2.16):

– брови (підняті, опущені, нормальні);
– очі (розплющені, примружені, нормальні);
– губи (розтягнуті, зжаті, нормальні).

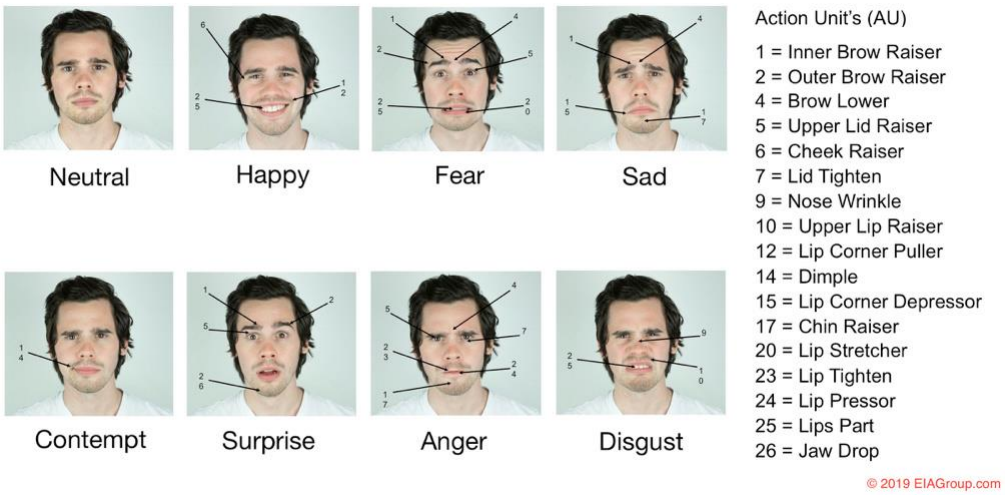


Рисунок 2.16 – Система кодування рухів обличчя [30]

Звідси формується вектор ознак, який описує емоційний стан: $\{x_1, x_2, x_3\}$. $x_1, x_2, x_3 \in [0,\ 1]$, причому $x_1 \in [0,\ 0.2]$ – для примружених очей; $x_1 \in [0.4,\ 0.6]$ – для нормальних очей; $x_1 \in [0.8,\ 1.0]$ – для розплющених очей; $x_2 \in [0,\ 0.2]$ – для зжатих губ; $x_2 \in [0.4,\ 0.6]$ – для нормальних губ; $x_2 \in [0.8,\ 1.0]$ – для розтягнутих губ; $x_3 \in [0,\ 0.2]$ – для опущених брів; $x_3 \in [0.4,\ 0.6]$ – для нормальних брів; $x_3 \in [0.8,\ 1.0]$ – для піднятих брів (рис. 2.17).

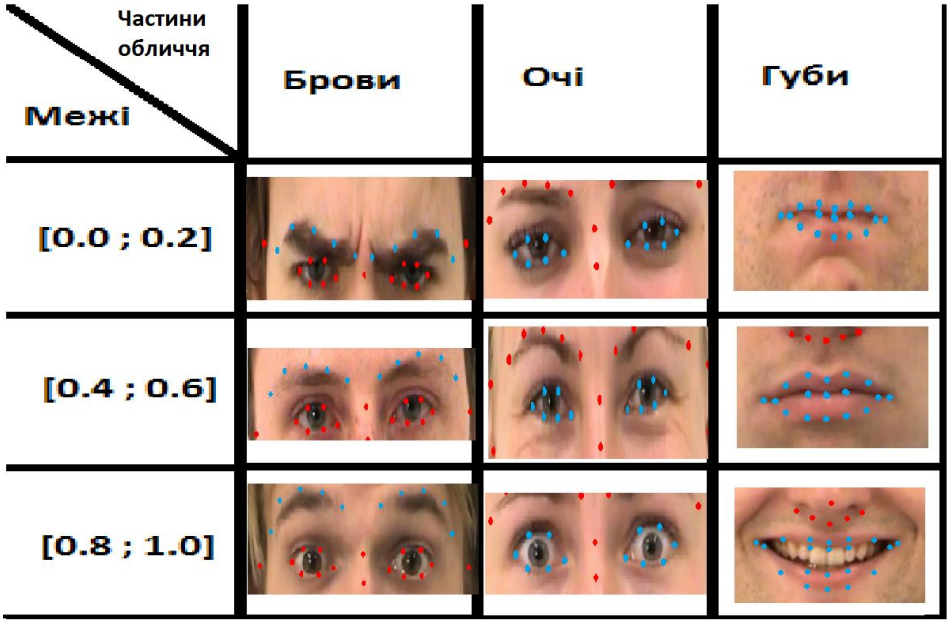


Рисунок 2.17 – Межі параметрів та частини обличчя

2.3 Алгоритм дослідження

Можна виділити наступні кроки: підготовка векторів емоційних станів, візуалізація вхідних даних у двомірний простір, мануальне розділення на класи, обернене перетворення до n-мірного простору, отримання коефіцієнтів гіперплощин.

Для формування вхідних даних для дослідження за допомогою модуля OpenCV відбувається отримання зображення, виявлення обличчя за допомогою

методу Віоли – Джонса, FaceLandmarksLBF використовується для отримання специфічних точок обличчя.

За допомогою запропонованого алгоритму формуються вектори. В результаті отримано 48 векторів емоційних станів.

Крок 1. За допомогою методу *SMACOF* відбувається перехід від n-мірних даних (рис. 2.18) до двомірної системи (рис. 2.19).

```
100.00|:0.57:0.44:0.83
200.00|:0.51:0.50:0.18
300.00|:0.20:0.91:0.19
400.00|:0.83:0.91:0.16
101.00|:0.47:0.51:0.59
201.00|:0.55:0.82:0.16
301.00|:0.18:0.89:0.17
401.00|:0.83:1.00:0.17
102.00|:0.44:0.49:0.83
202.00|:0.40:0.90:0.17
302.00|:0.16:0.88:0.16
402.00|:0.00:0.00:0.00
103.00|:0.56:0.56:0.58
203.00|:0.44:0.90:0.18
303.00|:0.15:0.86:0.17
403.00|:0.00:0.00:0.00
104.00|:0.48:0.19:0.89
204.00|:0.52:0.54:0.19
304.00|:0.15:0.47:0.19
404.00|:0.94:0.92:0.40
```

Рисунок 2.18 – Вхідні вектори

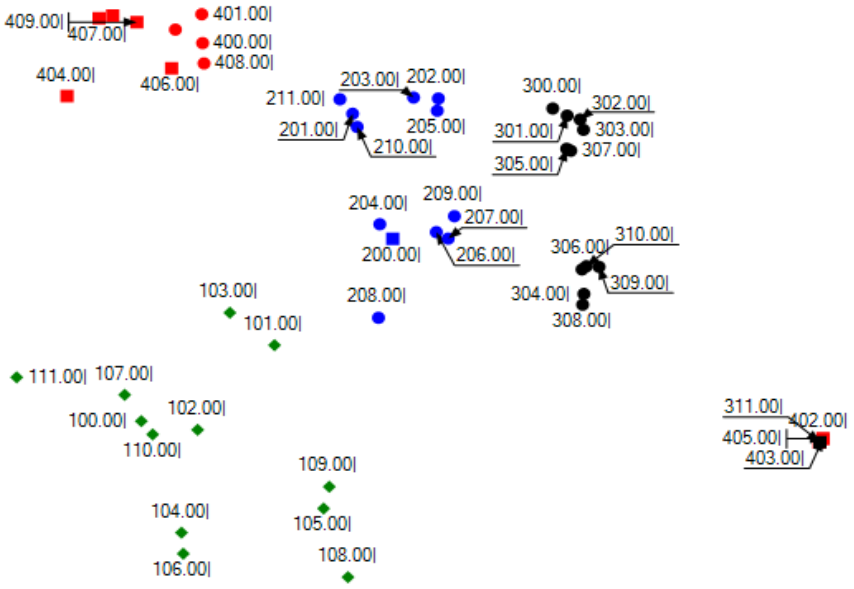


Рисунок 2.19 – Результат багатовимірного шкалювання

Крок 2. Отримані масиви точок мануально розділяються на групи (рис. 2.20)

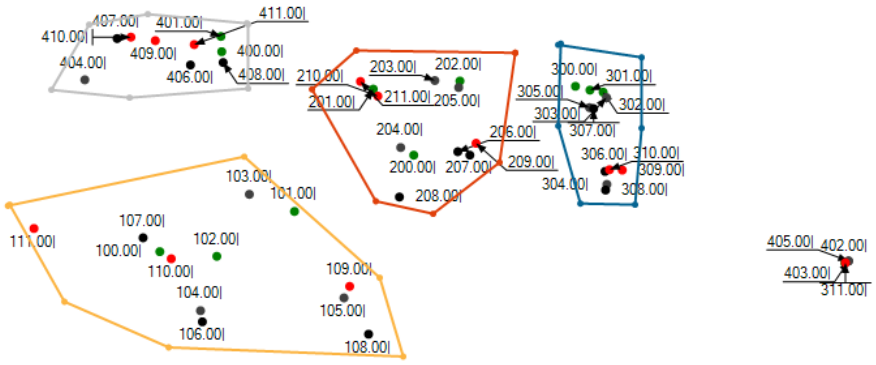


Рисунок 2.20 – Результат мануального розділення

Для того щоб мати достатньо даних для перетворення до N мірного простору відбувається визначення додаткової точки (рис. 2.21).

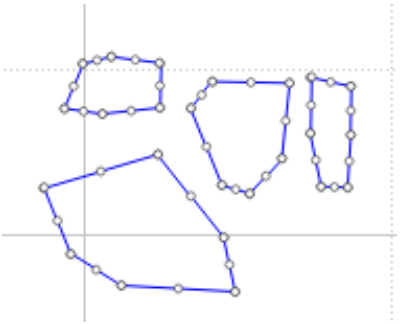


Рисунок 2.21 – Визначені точки ліній

Крок 3. При використанні SMACOF існує ймовірність потрапляння в локальні мінімуми, особливо при малій розмірності. Також важливою складовою є визначення початкових координат [31]. Тому, поки всі коефіцієнти гіперплощини не будуть визначені (а також вони будуть різні) для відповідних векторів продовжується робота алгоритму SMACOF (рис. 2.22). Наприклад, якщо всі вектори страху при перевірці принадлежності до даної множини видають однакові патерни (‘+’ або ‘-’) – коефіцієнти гіперлощини можна прийняти. Якщо дана умова коректна для всіх емоційних станів – рішення знайдено.

Рисунок 2.22 – Алгоритм пошуку коректних коефіцієнтів гіперплощини

Було отримано наступні патерни для емоційних станів:

$Happy = \{+, +, +, +, -, +\}$, $Sad = \{-, -, +, +, -, +\}$, $Anger = \{+, +, +, +, -, -\}$, $Fear = \{+, +, -, -, +, +\}$.

Маємо наступну схему дослідження (рис. 2.23).

Рисунок 2.23 – Схема дослідження

Під час дослідження використовувалася база ADFES [32], яка надає 12 зображень (7 чоловіків, 5 жінок), як вираження 6-и основних емоційних станів(гнів, страх, радість, сум, відраза, здивування) та 3-х комплексних станів (презирство, гордість, збентеження), а також нейтральність (рис. 2.24).

Рисунок 2.24 – Приклад зображень [32]

Після дослідження були виявлені проблемні зображення (рис. 2.25). Для більш коректного розуміння було побудовано схему значень різних емоцій для брів (рис. 2.26).

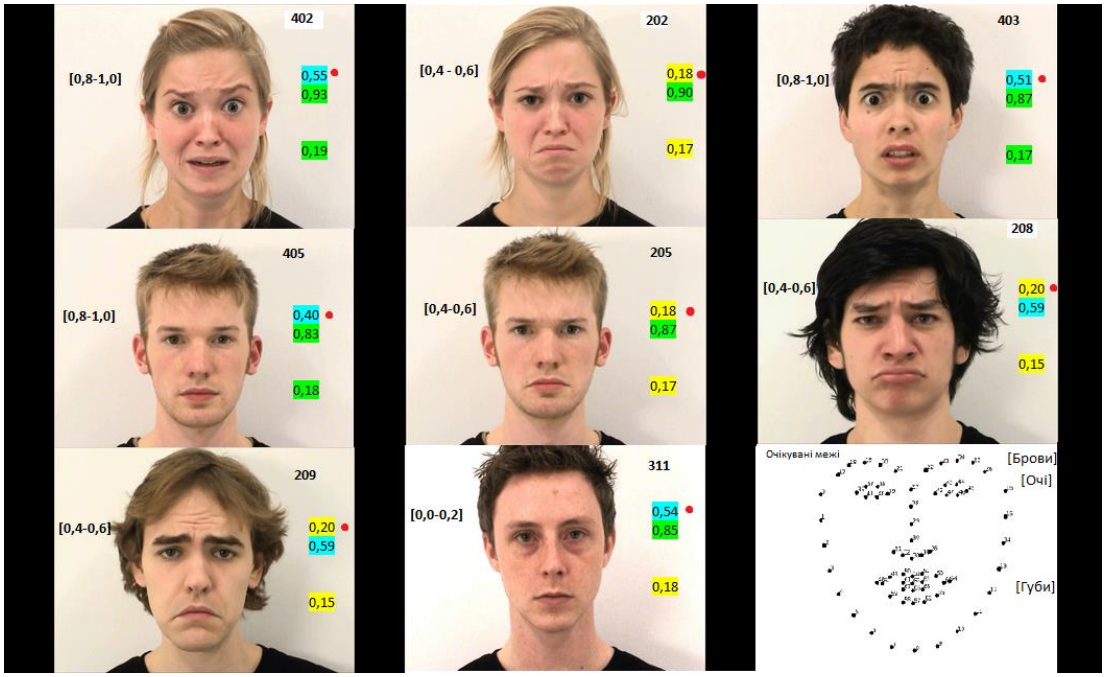


Рисунок 2.25– Проблемні зображення

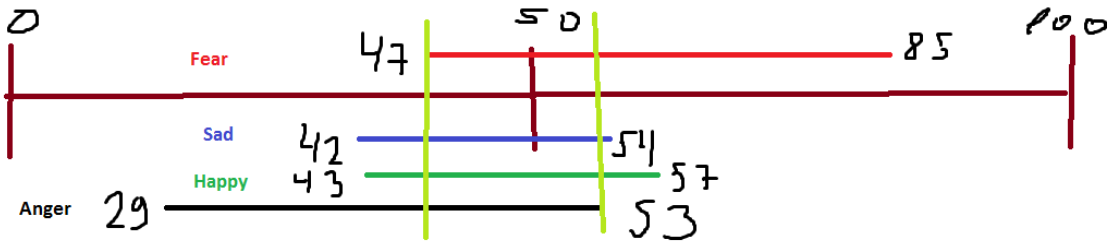


Рисунок 2.26 – Межі значень брів для різних емоцій

Для зображень «202», «205», «208», «209» спостерігається накладання меж різних емоцій. Після перевірки приналежності значення брів до іншої межі, були отримані коректні результати і зображення були долучені до вибірки.

Зображення «311» та «405» були відкинуті так як не проявляють задекларованих емоцій.

Зображення «402», «403» являються такими, які на даному етапі не можуть бути використаними, так як проявляють коректні емоції, але не може бути отримані коректні дані про них.

Для покращення роздільності даних було вирішено збільшити кількість даних у векторі емоційного стану та використовувати новий метод для нормалізації геометрії обличчя людини за допомогою 2D каркасу.

Для збільшення кількості необхідних відстаней використовувався FACS. Для початку були відібрані основні відстані, які описують стан брів, очей та губ (таблиця 2.1) (рис. 2.27) [33] . На основі Action Unit [34][35] 1, 2, 4, 5, 6, 12, 17, 20, 23, 24, 25 для відповідних емоцій: радість, сум, гнів, страх (таблиця 2.2).

Таблиця 2.1 – Відстані між точками і певні частини обличчя

	Брови	Очі	Губи
Відстані між точками	P17 і P36 P18 і P36 P18 і P37 P19 і P36 P19 і P37 P19 і P38	P37 і P41 P38 і P40	P60 і P64 P50 і P58 P51 і P57

	P20 і P37 P20 і P38 P20 і P39 P21 і P38 P21 і P39		P52 і P56

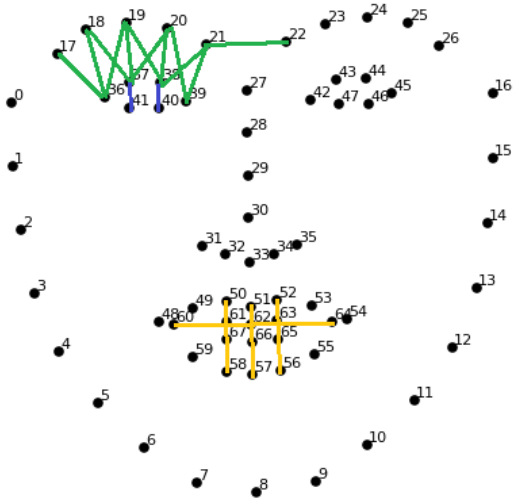


Рисунок 2.27 – Основні відстані для дослідження

Таблиця 2.2 – Відстані між точками і відповідні Action Unit

Action Unit	Відстані між точками
AU 1  Inner Brow Raiser	P20 і P37 P20 і P38 P20 і P39 P21 і P38 P21 і P39
AU 2  Outer Brow Raiser	P17 і P36 P18 і P36 P18 і P37
AU 4  Brow Lowerer	P21 і P22
AU 5  Upper Lid Raiser	P37 і P41 P38 і P40

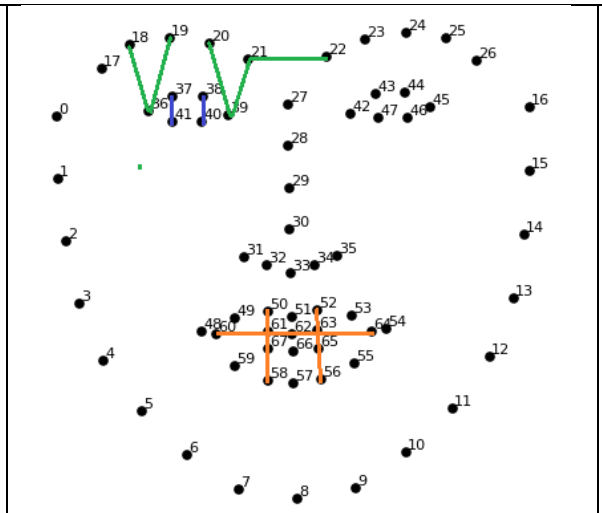
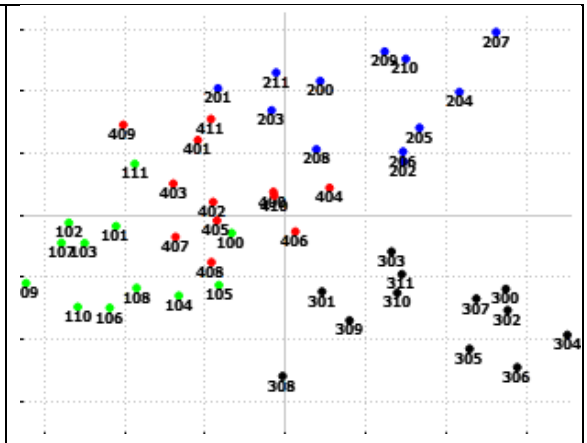
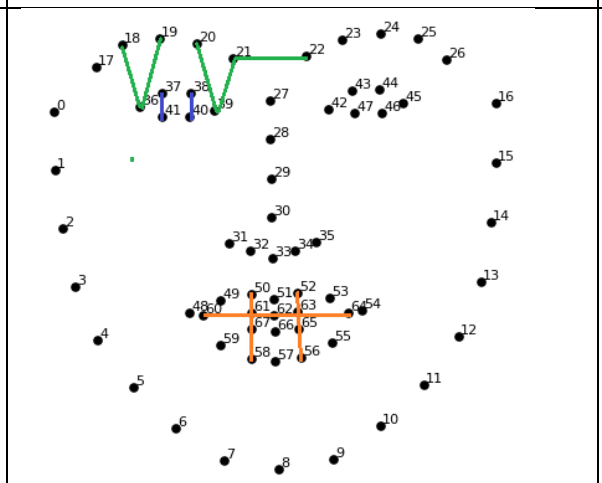
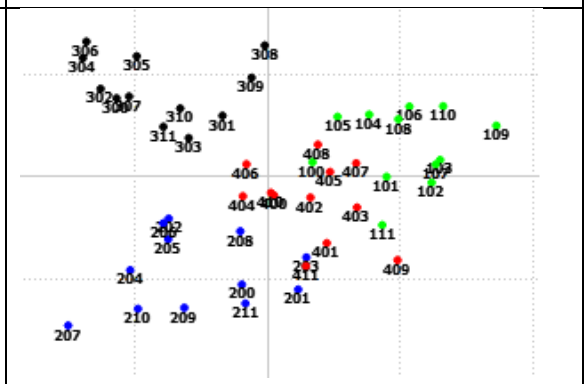
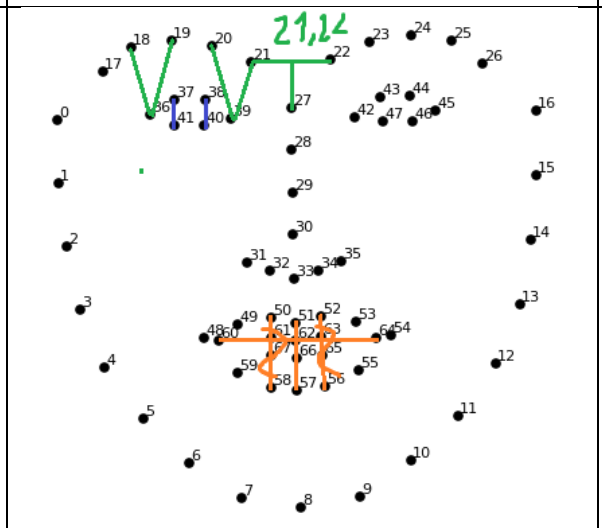
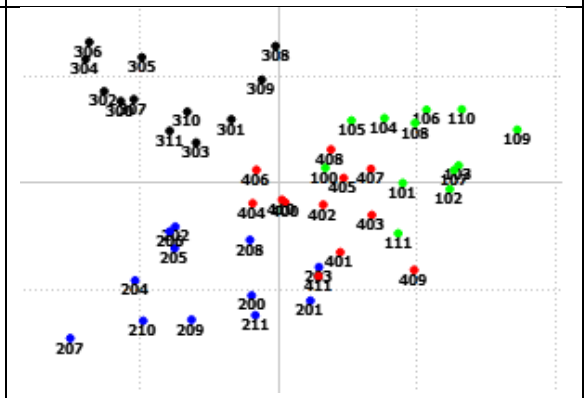
AU 6  Cheek Raiser	P37 і P41 P38 і P40
AU 12  Lip Corner Puller	P60 і P64
AU 17  Chin Raiser	P50 і P58 P51 і P57 P52 і P56
AU 20  Lip Stretcher	P60 і P64
AU 23  Lip Tightener	P60 і P64
*AU 25  Lips Part	P50 і P58 P51 і P57 P52 і P56

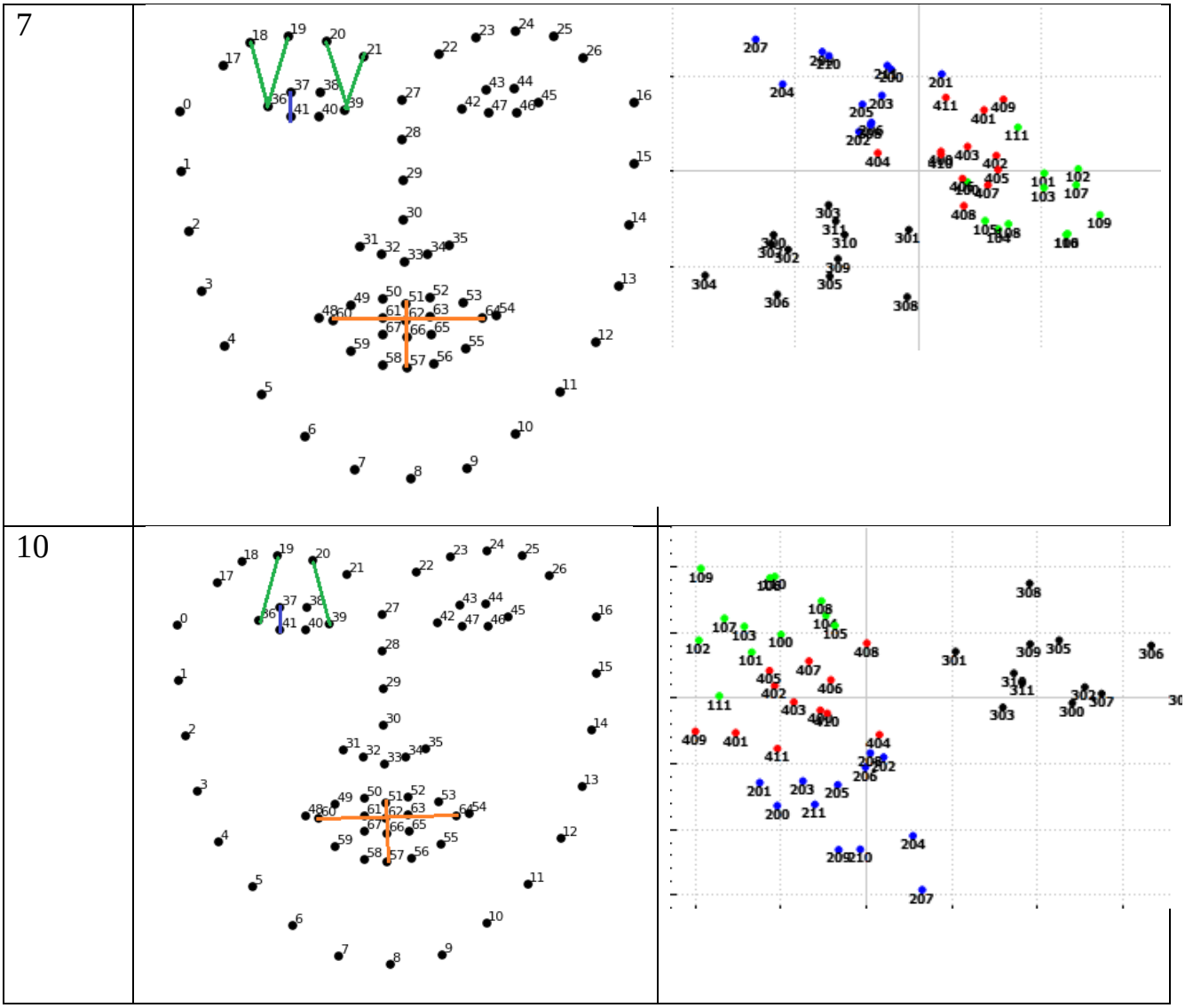
Під час дослідження були сформовані різні комбінації відстаней між точками, деякі з них наведені в наступній таблиці (таблиця 2.3).

Через незначний або поганий вплив на результати були виключені наступні відстані: P50 і P58, P52 і P56, P36 і P40, P[21,22] і P27, P21 і P22. Відстані між точками на верхівці ока і бровами були виключені, бо не є коректними під час страху і злості.

Таблиця 2.3 – Результати дослідження

Номер тесту	Відстані між точками	Результат візуалізації

4		
5		
6		



Було отримано наступний фінальний результат (рис. 2.28, 2.29).

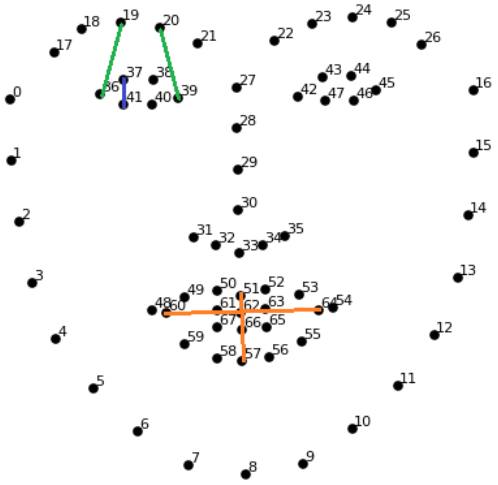


Рисунок 2.28 – Результуючі відстані

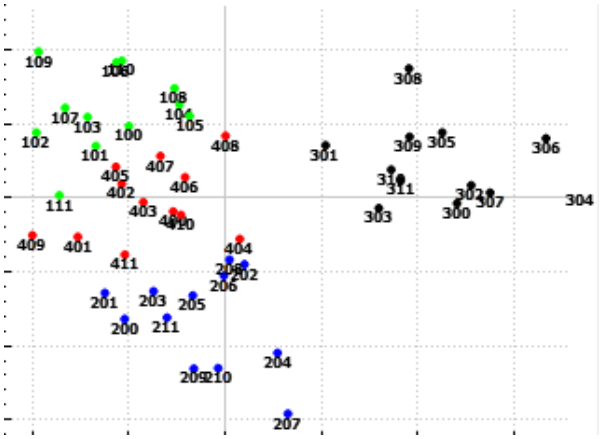


Рисунок 2.29 – Візуалізація емоційних станів

В результаті маємо вектор ознак: $v = \{x_1, x_2, x_3, x_4, x_5\}$, де

$x_1 =$ відстань між $P19$ і $P36$, $x_2 =$ відстань між $P20$ і $P39$,

$x_3 =$ відстань між $P37$ і $P41$, $x_4 =$ відстань між $P51$ і $P57$,

$x_5 =$ відстань між $P60$ і $P64$.

2.4 Висновки до розділу 2

Розглянувши алгоритми для розпізнавання та візуалізації, було враховано їхні особливості та спроектовано математичну модель придатну для візуалізації даних та розпізнавання емоційних станів.

Використання бібліотеки OpenCV (методи Віоли-Джонса і facemarkLBF), можливо використати для розпізнавання обличчя і точок-особливостей відповідно. Також дана бібліотека може використовуватись для отримання відео кадру. Бібліотека Qt використовується для можливості програмування GUI та взаємодії з OpenCV.

Відбувається отримання відео кадру, обробка зображення, виявлення обличчя, виявлення контрольних точок, отримання вектору обличчя, визначення емоційного стану.

Для максимальної сумісності, варто використовувати C++, як рідну мову бібліотеки OpenCV. Розробка відбуватиметься в Visual Studio, з використанням графічної бібліотеки Qt.

Розділ 3

Розробка методів та компонентів для інформаційної технології

3.1 Структура і функціональне призначення модулів системи

Для дослідження та перевірки математичної моделі було створено наступні модулі (рис. 3.1):

– «EmotionDetection» призначений для обчислення векторів емоційних станів;

– «MDS&SMACOF» призначений для багатовимірного шкалювання;

– «MDS_QT» призначений для мануального розділення на класи;

– «NPoints» призначений для пошуку точок на лінії;

– «det» призначений для пошуку дискримінантів;

– «Dist» призначений для формування матриці відстаней;

– «er_qt» призначений для розпізнавання емоційного стану обличчя на основі зображення.

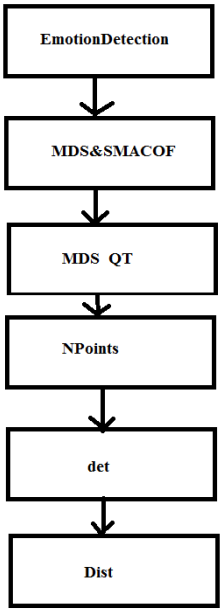


Рисунок 3.1 – Взаємодія модулів системи

Взаємодія модулів відбувається наступним чином, модуль «EmotionDetection» на основі набору зображень формує вектори емоційних станів. Модуль «MDS&SMACOF» з отриманих векторів формує точки в 2-мірному просторі, а модуль MDS_QT надає змогу мануального розділення, після чого записує отримані лінії у файл. Модуль «NPoints» шукає N – 2 (де N – розмірність вектора емоції) точок на лінії.

Модуль «det» з векторів у N-мірному просторі обчислює детермінанти матриць NxN, які описують кожну лінії у 2-мірному просторі.

Модуль «er_qt» з отриманих патернів(детермінантів) надає змогу виявлення емоційного стану з зображення або відео-потоку.

3.2 Розробка модуля обчислення векторів емоційного стану

Для реалізації функціоналу обчислення векторів емоційного стану було створено модуль «EmotionDetection» (рис. 3.2).

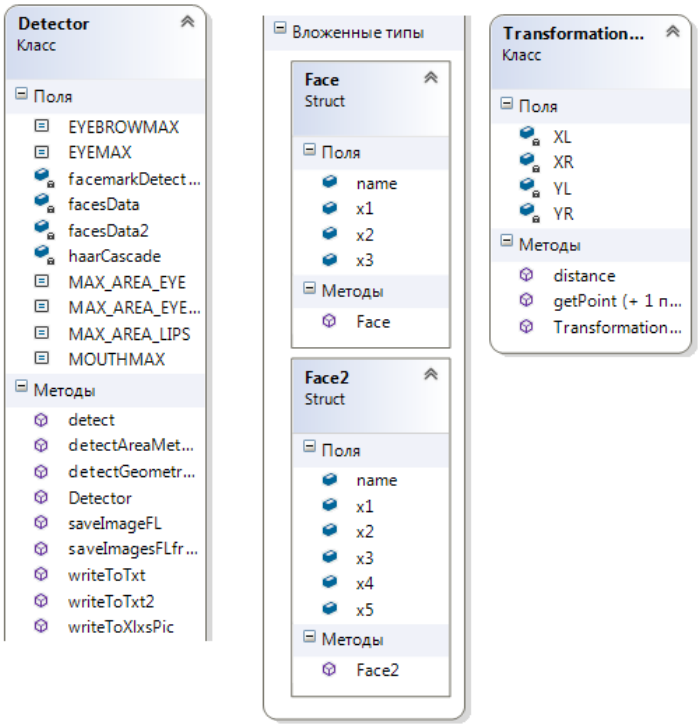


Рисунок 3.2 – Діаграма класів модуля «EmotionDetection»

Модуль містить наступні класи:

– Detector – призначений для виявлення обличчя, специфічних точок обличчя, отримання вектору обличчя.

– Transformation2D – призначений для формування КСК, та отримання нормалізованих точок обличчя.

Модуль зчитує фото з певної папки, проводить виявлення обличчя та контрольних точок, а також на їхній основі формує КСК, обраховує вектори емоційних станів і записує їх у файл.

3.3 Розробка модуля багатовимірного шкалювання

Багатовимірне шкалювання виконується за допомогою модуля “MDS&SMACOF” (рис. 3.3).

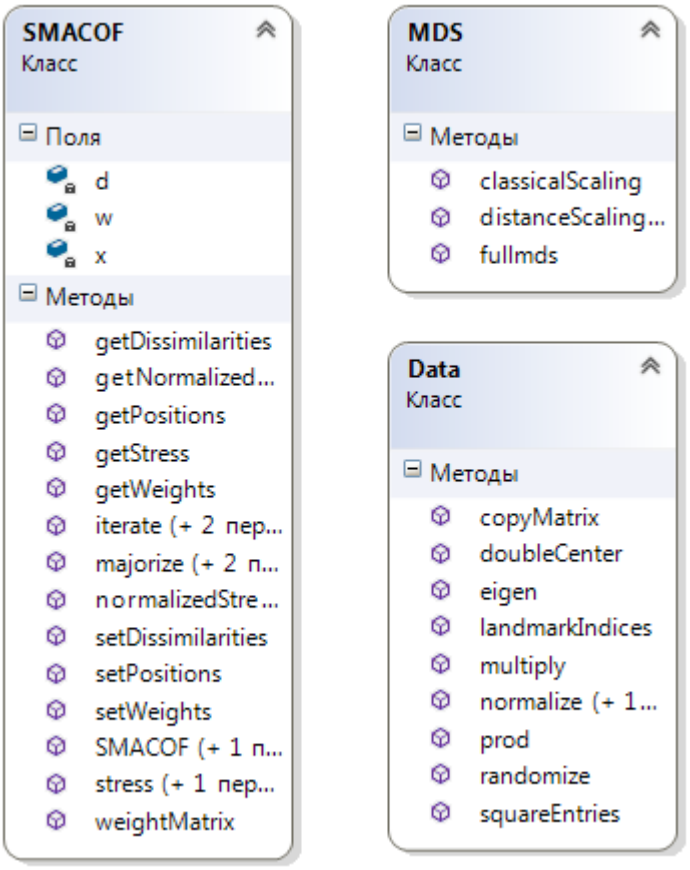


Рисунок 3.3 – Діаграма класів модуля «MDS&SMACOF»

MDS&SMACOF складається з наступних класів:

– Data призначений для виконання класичного багатовимірного шкалювання;

– MDS призначений для взаємодії класів MDS та Data;

SMACOF застосовується для мажоризації стресу та нормалізації отриманих з MDS даних

3.4 Розробка модуля для розпізнавання емоцій

Модуль “er_qt” використовується для розпізнавання емоційних станів (рис. 3.4).

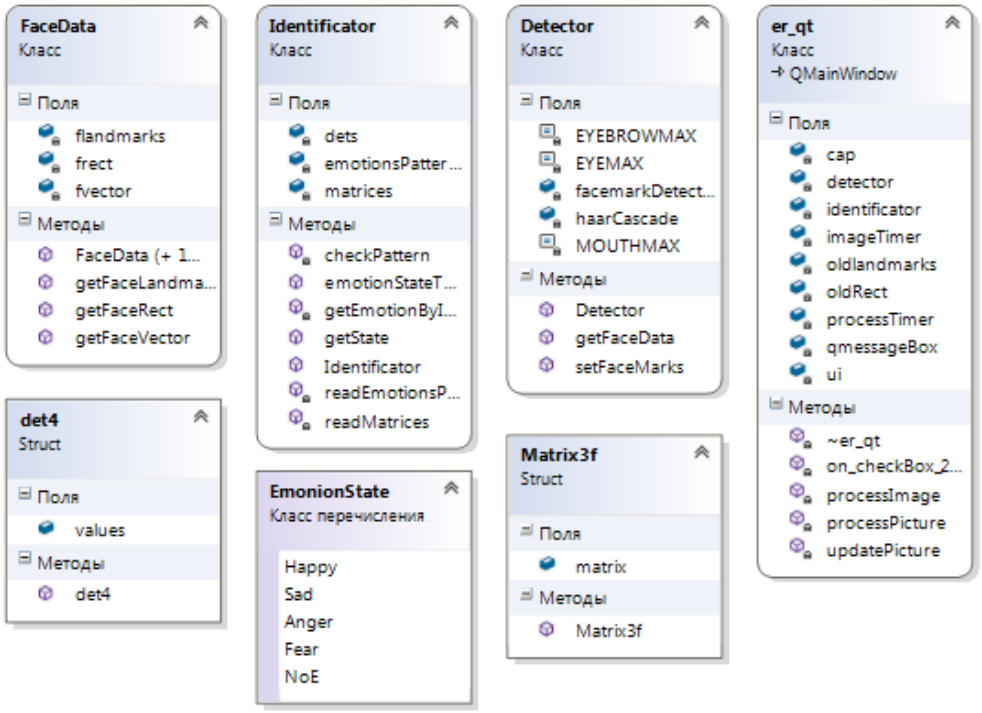


Рисунок 3.4 – Діаграма класів модуля «er_qt»

«er_qt» реалізується за допомогою наступних класів:

– Detector призначений для виявлення обличчя, специфічних точок обличчя, отримання вектору обличчя;

– Identificator призначений для визначення емоційного стану;

– det4 описує результати обробки матриць – детермінанти;

– Matrix3d описує матрицю розмірністю 3x3;

– FaceData містить основу інформацію про обличчя: вектор значень, координати розташування, координати специфічних точок обличчя.;

– er_qt описує головне вікно програми та взаємодію компонентів.

Вся взаємодія з програмою відбувається за допомогою GUI, який надає змогу завантаження зображення або отримання кадру з відео-камери, а також розпізнавання емоцій (рис. 3.5).

Метод “UpdatePicture” класу “er_qt” використовується для отримання кадру з відео-потоку і його оновлення на компоненті GUI.

Метод «processImage» класу «er_qt» оброблює отримане зображення, а саме проводить пошук обличчя, контрольних точок, а також визначення емоційного стану.

Метод «readEmotionsPatterns» класу «Identificator» зчитує з файлу патерни емоційних станів.

Метод «checkPattern» класу «Identificator» перевіряє приналежність отриманого значення до одного з емоційних станів.

Метод «getFaceData» класу «Detector» обчислює вектор емоційного стану.

Метод «getState» класу «Identificator» повертає значення емоційного стану.

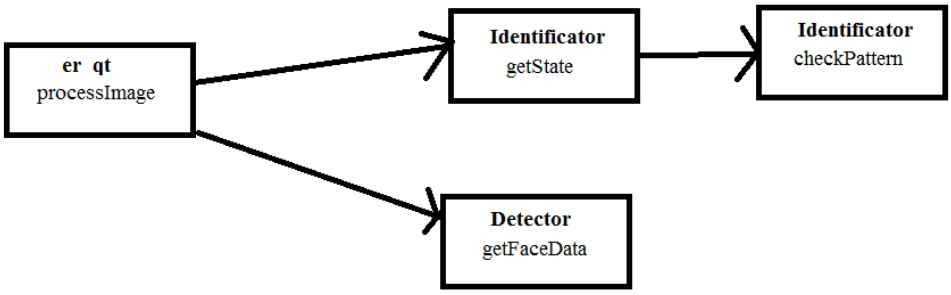


Рисунок 3.5 – Взаємодія основних методів системи

3.5 Висновки до розділу 3

Після детального розгляду алгоритмів та математичної моделі, було спроектовано та реалізовано модулі системи, які реалізують необхідний

функціонал описаний в розділі №2. Розглянуто їхній взаємозв’язок, а також нюанси роботи. За допомогою яких відбувається формування векторів, перехід в 2-мірний простір, візуалізація та розділення даних.

Створені модулі повністю реалізують функції описані у постановці завдання.

Розділ 4

Дослідження ефективності інформаційної технології розпізнавання емоційних станів

4.1 Розробка тестових випадків для інформаційної технології

Відповідно до завдань було проведено тестування модулів системи. Для тестування основних функцій було розроблено набір тест-кейсів.

Тест-кейс - це тестовий випадок, який полягає у виконанні кількох дій, необхідних для перевірки певної функціональності розробленої програмної системи [36].

У таблиці 4.1 демонструється тест-кейс для перевірки формування векторів.

Таблиця 4.1 –Тест-кейс «Формування векторів емоцій»

TC ID/Priority	**0001**	**1**
IDEA: Перевірка операції «Формування емоцій векторів» **ADDITIONAL INFO:** Вхідний дані «Зображення з папки testPhotos»		
Revision History		
Created on: 06.11.2020		Новий тест-кейс
Execution part		
PROCEDURE		**EXPECTED RESULT**
1. Запустити програму «EmotionDetection»		Вектори емоцій записано у файл «res.txt».

Таблиця 4.2 містить тестовий випадок для перевірки формування точок в 2D просторі.

Таблиця 4.2 –Тест-кейс «Формування точок в 2D просторі»

TC ID/Priority	**0002**	**1**
IDEA: Перевірка операції «Формування точок в 2D просторі» **ADDITIONAL INFO:** Вхідний файл «res.txt»		
Revision History		
Created on: 06.11.2020		Новий тест-кейс
PROCEDURE		**EXPECTED RESULT**
1. Запустити програму «MDS&SMACOF»		Координати точок записано у файл «2d.txt».

Для тестування мануального розділення на класи використовувався відповідний тест (таблиця 4.3).

Таблиця 4.3 –Тест-кейс «Мануальне розділення на класи»

TC ID/Priority	**0003**	**1**
IDEA: Перевірка операції «Мануальне розділення на класи» **ADDITIONAL INFO:** Вхідний файл «2d.txt»		
Revision History		
Created on:		Новий тест-кейс
PROCEDURE		**EXPECTED RESULT**
1. Запустити програму «MDS_QT» 2. Натиснути кнопку «виділити область» 3. Виділити необхідну область 4. Натиснути кнопку «Завершити виділення» 5. Виконати пункти 2-4 необхідну кількість разів		Координати ліній записано у файл «lines.txt»

Перевірка фільтрації здійснюється за допомогою наступного тест-кейсу (таблиця 4.4).

Таблиця 4.4 –Тест-кейс «Перевірка фільтрації»

TC ID/Priority	**0004**	**1**
IDEA: Перевірка операції «Перевірка фільтрації» **ADDITIONAL INFO:** Вхідний файл «data.txt»		
Revision History		
Created on:		Новий тест-кейс
PROCEDURE		**EXPECTED RESULT**
1. Запустити програму «Filter»		Отримане повідомлення: «Положення голови коректне»

Після виконання тест-кейсів можна пересвідчитися у коректності роботи модулів програми та формуванні вихідних даних. Також відстежується вплив вихідних даних одного модуля, як вхідних даних іншого, що дозволяє уникати помилок на ранніх етапах розробки.

4.2 Тестування системи. Формування результатів

Для перевірки системи було обрано базу емоцій «ADFES», серед яких відібрано 12 обличь, по 4 емоційні стани кожне.

З початку відбувається запуск модуля «evdetection_x64_release.exe», який з відповідної папки зчитує кожне зображення і формує на основі геометричних особливостей обличчя вектори емоційних станів. Дані записуються у файл (рис. 4.1).

```
0.32 0.28 0.17 0.09 0.65
0.4 0.39 0.18 0.36 0.58
0.37 0.29 0.12 0.41 0.94
0.34 0.34 0.13 0.26 0.68
0.3 0.28 0.17 0.15 0.65
0.37 0.42 0.2 0.25 0.6
0.34 0.3 0.12 0.22 0.87
0.35 0.35 0.15 0.24 0.58
0.27 0.27 0.16 0.17 0.58
0.39 0.36 0.19 0.36 0.72
0.3 0.32 0.12 0.4 0.96
0.28 0.33 0.17 0.24 0.64
0.25 0.29 0.16 0.2 0.63
0.34 0.4 0.15 0.4 0.6
0.37 0.33 0.13 0.33 0.88
0.31 0.35 0.17 0.24 0.66
0.26 0.23 0.12 0.28 0.7
0.46 0.4 0.18 0.38 0.73
0.34 0.29 0.1 0.48 1.05
0.35 0.34 0.14 0.34 0.72
0.29 0.23 0.15 0.26 0.72
0.41 0.34 0.18 0.38 0.65
0.31 0.26 0.1 0.36 0.95
0.3 0.27 0.16 0.26 0.63
0.28 0.21 0.13 0.16 0.68
0.42 0.35 0.15 0.3 0.71
0.33 0.3 0.08 0.32 1.07
0.34 0.29 0.14 0.22 0.71
0.28 0.29 0.14 0.15 0.62
0.49 0.51 0.18 0.34 0.74
0.35 0.32 0.14 0.38 0.99
0.3 0.29 0.14 0.22 0.69
0.29 0.28 0.12 0.19 0.73
0.39 0.4 0.17 0.31 0.72
0.3 0.28 0.09 0.2 0.98
0.29 0.34 0.12 0.2 0.77
0.25 0.24 0.12 0.19 0.59
0.46 0.43 0.17 0.36 0.64
0.32 0.28 0.12 0.36 0.92
0.3 0.38 0.13 0.26 0.52
0.23 0.25 0.12 0.18 0.68
0.46 0.45 0.18 0.29 0.69
0.34 0.31 0.1 0.29 0.91
0.34 0.32 0.14 0.25 0.69
0.32 0.34 0.17 0.21 0.58
0.39 0.49 0.19 0.25 0.66
0.36 0.36 0.11 0.26 0.93
0.34 0.45 0.16 0.23 0.63
```

Рисунок 4.1 – Вектори емоційних станів

За формування векторів відповідає код наведений у «Додаток А» у модулі «EmotionalDetection», а саме метод «detectGeometryMethod».

Багатовимірним шкалюванням даних до 2-мірної системи займається модуль «MDS&SMACOF». Результати роботи програми записуються у файл (рис. 4.2).

```
100 0.114455 -0.125507
200 0.115573 0.177251
300 -0.249106 0.0139737
400 0.0544143 0.0194837
101 0.14385 -0.0741256
201 0.133314 0.130228
301 -0.109326 -0.079046
401 0.157797 0.0533411
102 0.201646 -0.0884846
202 -0.0294574 0.0961324
302 -0.263726 -0.014423
402 0.110356 -0.0179132
103 0.149373 -0.111094
203 0.0881725 0.151021
303 -0.159662 0.0174698
403 0.0867701 0.00613269
104 0.0426398 -0.140983
204 -0.0586441 0.211189
304 -0.382851 -0.00475247
404 -0.0195727 0.0570031
105 0.0343148 -0.115888
205 0.0317759 0.142201
305 -0.233615 -0.0905857
405 0.118571 -0.0405614
106 0.114881 -0.189226
206 -0.000612474 0.105637
306 -0.341107 -0.0863456
406 0.0387564 -0.029787
107 0.172114 -0.123775
207 -0.0711461 0.305034
307 -0.280308 -0.00343273
407 0.0666034 -0.0557066
108 0.051232 -0.157003
208 -0.00612892 0.0832137
308 -0.197718 -0.191821
408 -0.00718043 -0.0865664
109 0.20369 -0.20927
209 0.032891 0.235297
309 -0.198853 -0.0874581
```

Рисунок 4.2 – Координати точок в 2-мірній системі

Перетворення з N-мірної в 2-мірну систему здійснюється за рахунок коду наведеного в «Додатку А».

Для візуалізації та мануального розділення використовується модуль «MDS_QT». За допомогою кнопки «Виділити область» надається змога формування ліній для виділення певної області точок (рис. 4.3).

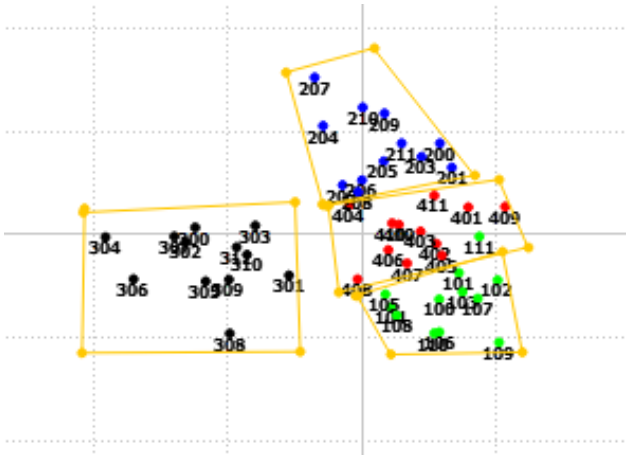


Рисунок 4.3 – Виділені області точок

За формування лінії відповідає код наведений в «Додатку А» у модулі «MDS_QT».

4.3 Висновки до розділу 4

Після розробки основних компонентів системи було перевірено їхнє функціонування та взаємозв’язок за допомогою тест-кейсів. Також були продемонстровано роботу модулів системи. Було отримано очікувані результати тестування, які задовольняють початковим потребам.

Загальні висновки

Проблема розпізнавання емоційних станів вирішувалася багатьма методами, які з часом покращувалися. Кожен підхід має свої переваги і недоліки, кожен має свої межи оптимальної роботи.

Результатом дипломної роботи є система розпізнавання емоційних станів за допомогою геометричних особливостей обличчя.

В результаті було виконані наступні задачі:

– створення та дослідження моделі мімічних проявів емоцій;

– дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;

– розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;

– валідація та верифікація моделі, підходу та інформаційної системи.

Для подальшого розвитку програми можна додати ще декілька ключових значень обличчя, сформувати точніший механізм отримання значень, а також алгоритм розпізнавання, який враховує емоційний стан через одиничні особливості.

Дана програма може використовуватися для швидкого розпізнавання, а також розпізнавання швидкого переходу з одного емоційного стану до іншого, що є важливим в критичних ситуаціях.

Основні наукові та практичні результати за магістерською роботою оприлюднювались на міжнародній конференції IEEE International Conference on Advanced Trends in Information Theory (ATIT), та опубліковані:

1. Svirnevskyi M., Barmak O., Ivaschenko S., Krak Iu. Face Image Transformations for Correct Recognition Problems Solving // IEEE International Conference on Advanced Trends in Information Theory (ATIT). 2019,P. 415-419 (Scopus)
2. Іващенко С.О., Калита О.Д., Бармак О.В., Скрипник Т.К. Інформаційна технологія визначення характерних ознак на обличчі для розпізнавання

емоційних проявів // Журнал Computer Systems And Information Technologies. Том 1 № 2 (2020).

Перелік посилань

1. Емоції і почуття [Електронний ресурс]. – Режим доступу: http://library.nlu.edu.ua/POLN_TEXT/KNIGI/1_DISKI/UR_PSIX/html/12.htm

2. Фундаментальные эмоции (по К. Изарду) [Електронний ресурс]. – Режим доступу: https://psyera.ru/fundamentalnye-emocii-po-k-izardu_7878.htm

3. Micro Expressions: Definition & Examples [Електронний ресурс]. – Режим доступу: https://study.com/academy/lesson/micro-expressions-definition-examples.html

4. Surprise [Електронний ресурс]. – Режим доступу: http://psychology.iresearchnet.com/social-psychology/emotions/surprise/

5. What is surprise? [Електронний ресурс]. – Режим доступу: https://www.paulekman.com/universal-emotions/what-is-surprise/

6. Fear [Електронний ресурс]. – Режим доступу: https://emotiontypology.com/typology/list/fear

7. What is Fear? [Електронний ресурс]. – Режим доступу: https://www.paulekman.com/universal-emotions/what-is-fear/

8. Anger [Електронний ресурс]. – Режим доступу: https://emotiontypology.com/typology/list/anger

9. What is anger? [Електронний ресурс]. – Режим доступу: https://www.paulekman.com/universal-emotions/what-is-anger/

10. Happiness [Електронний ресурс]. – Режим доступу: http://psychology.iresearchnet.com/social-psychology/emotions/happiness/

11. What is enjoyment? [Електронний ресурс]. – Режим доступу: https://www.paulekman.com/universal-emotions/what-is-enjoyment/

12. Sadness [Електронний ресурс]. – Режим доступу: https://emotiontypology.com/typology/list/sadness

13. What is Sadness? [Електронний ресурс]. – Режим доступу: https://www.paulekman.com/universal-emotions/what-is-sadness/

14. Swati Mishra1, Avinash Dhole, “A SurveyonFacial Expression Recognition Techniques”, 2015

15. A. Majumder, L. Behera, V.K. Subramanian, "Local binary pattern based facial expression recognition using Self-organizing Map," 2014

16. Jizheng Yi; Xia Mao; Lijiang Chen; Yuli Xue; A. Compare, "Facial expression recognition considering individual differences in facial structure and texture," IET Computer Vision, vol.8, no.5, pp. 429-440, October 2014

17. Li Xia, "Facial Expression Recognition Based on SVM," 2014 7th International Conference on Intelligent Computation Technology and Automation (ICICTA), pp. 256-259, 25-26 Oct. 2014

18. M. Suk, B. Prabhakaran, "Real-Time Mobile Facial Expression Recognition System – A Case Study," 2014IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 132-137, 23-28 June 2014.

19. A Review on Automatic Facial Expression Recognition Systems Assisted by Multimodal Sensor Data [Електронний ресурс]. – Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6514576/

20. FaceReader [Електронний ресурс]. – Режим доступу: http://www.vicarvision.nl/products/facereader/

21. Sightcorp [Електронний ресурс]. – Режим доступу: https://builders.intel.com/ai/membership/sightcorp

22. Emotion Recognition Software [Електронний ресурс]. – Режим доступу: https://sightcorp.com/emotion-recognition/

23. Метод распознавания лиц Виолы-Джонса (Viola-Jones) [Електронний ресурс]. – Режим доступу: https://oxozle.com/2015/04/11/metod-raspoznavaniya-lic-violy-dzhonsa-viola-jones//

24. Svirnevskyi M., Barmak O., Ivaschenko S., Krak Iu. Face Image Transformations for Correct Recognition Problems Solving // IEEE International Conference on Advanced Trends in Information Theory (ATIT). 2019,P. 415-419 (Scopus)

25. Shaoqing Ren, Xudong Cao, Yichen Wei, Jian Sun. Face Alignment at 3000 FPS via Regressing Local Binary Features, 2014

26. Patrick J.F. Groenen, Ingwer Borg, The Past, Present, and Future of Multidimensional Scaling, 2013

27. Jan De Leeuw, Patrick Mair, “Multidimensional Scaling using majorization: SMACOF in R”, August 2009

28. Mapping the emotional face. How individual face parts contribute to successful emotion recognition [Електронний ресурс]. – Режим доступу: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177239

29. Chapter 11. Facial Expression Analysis [Електронний ресурс]. – Режим доступу: https://www.cs.cmu.edu/~cga/behavior/FEA-Bookchapter.pdf

30. Facial Expressions [Електронний ресурс]. – Режим доступу: https://www.eiagroup.com/knowledge/facial-expressions/

31. Ingwer Borg, Patrick Mair, “The Choice of Initial Configurations inMultidimensional Scaling: Local Minima, Fit, andInterpretability”, February 2017

32. Validation of the Amsterdam Dynamic Facial Expression Set – Bath Intensity Variations (ADFES-BIV): A Set of Videos Expressing Low, Intermediate, and High Intensity Emotions [Електронний ресурс]. – Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4718603/

33. S. Du, Y. Tao, A. M. Martinez, Compound facial expressions of emotion, February 2014

34. Facial Action Coding System (FACS) – A Visual Guidebook [Електронний ресурс]. – Режим доступу: https://imotions.com/blog/facial-action-coding-system/

35. Michael Valstar, Timing is everything A spatio-temporal approach to the analysis of facial actions, 2008

36. How to Write Test Cases: Sample Template with Examples [Електронний ресурс]. – Режим доступу: https://www.guru99.com/test-case.html

ДОДАТКИ

Додаток А

Програмні коди

EmotionDetection

Detector.h

```
#pragma once

#include <string>
#include <vector>

#include <opencv2\face.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2\highgui.hpp>
#include <opencv2\imgproc.hpp>
#include <opencv2/core/core.hpp>

class Detector
{
	struct Face
	{
		std::string name;
		float x1;
		float x2;
		float x3;

		Face(std::string name, float x1, float x2, float x3) :name(name), x1(x1), x2(x2), x3(x3) {}
	};

	struct Face2
	{
		std::string name;
		float x1;
		float x2;
		float x3;
		float x4;
		float x5;
		//float x6;
		//float x7;

		Face2(std::string name, float x1, float x2, float x3, float x4, float x5) :name(name), x1(x1), x2(x2), x3(x3),x4(x4), x5(x5){}
	};

	std::vector<Face> facesData;

	std::vector<Face2> facesData2;


	cv::CascadeClassifier haarCascade;
	cv::Ptr<cv::face::FacemarkLBF> facemarkDetector;

public:

	Detector();

	void detect(std::string folderName);
	void writeToXlxsPic(std::string saveFilename);
	void writeToTxt(std::string saveFilename);
	void writeToTxt2(std::string saveFilename);

	void saveImageFL(std::string fileName);
	void saveImagesFLfromFolder(std::string folderName);
	void detectAreaMethod(std::string folderName);
	void detectGeometryMethod(std::string folderName);

	static const float EYEMAX;
	static const float MOUTHMAX;;
	static const float EYEBROWMAX;

	static const float MAX_AREA_EYE;
	static const float MAX_AREA_LIPS;
	static const float MAX_AREA_EYEBROW;
};
```

Detector.cpp

```
#include "Detector.h"

#include "Transformation2D.h"

#include <iostream>

#include <boost\filesystem.hpp>

#include <libxl\libxl.h>

using namespace boost::filesystem;

static float distance(cv::Point2f a, cv::Point2f b)
{
	return std::sqrtf(std::powf(a.x - b.x, 2.f) + std::powf(a.y - b.y, 2.f));
}

Detector::Detector() : haarCascade("haarcascade_frontalface_alt2.xml")
{
	facemarkDetector = cv::face::FacemarkLBF::create();
	facemarkDetector->loadModel("lbfmodel.yaml");
}

void Detector::detect(std::string folderName)
{
	path p("testPhoto/");

	if (!exists(p))
	{
		std::cout << "testPhoto folder doesn't found" << std::endl;
		return;
	}

	directory_iterator end_itr;

	std::vector<size_t> index;

	std::vector<std::string> dirs;
	for (directory_iterator itr(p); itr != end_itr; ++itr)
	{
		if (is_directory(itr->path())) {
			dirs.push_back(itr->path().string());
		}
	}

	std::vector<cv::Rect> faces;
	std::vector<std::vector<cv::Point2f>>faces Landmarks;
```

```
	std::string fname;

	size_t cnt = std::count_if(
		directory_iterator(p),
		directory_iterator(),
		static_cast<bool(*)(const
path&)>(is_regular_file));

	size_t top = 1;

	directory_iterator di(p);
	directory_iterator di_end;

	while (di != di_end)
	{
		fname = p.string() + di-
>path().filename().string();
		cv::Mat image, gray;

		image = cv::imread(fname,
cv::IMREAD_COLOR);
		if (image.empty())
		{
			std::cerr << fname << " :
Image not found" << std::endl;
			++di;
			continue;
		}
		cvtColor(image, gray,
cv::COLOR_BGR2GRAY);
		haarCascade.detectMultiScale(gray,
faces);
		facemarkDetector->fit(gray, faces,
facesLandmarks);

		for (size_t i = 0; i <
faces.size(); i++)
		{
			//Eye
			cv::Point2f a =
(facesLandmarks[i][37] + facesLandmarks[i][38]) /
2;
			cv::Point2f b =
(facesLandmarks[i][40] + facesLandmarks[i][41]) /
2;

			float fd = distance(a, b);
			float sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][39]);

			float res = fd / sd /
EYEMAX;

			float eyeVal = 0.f;
			if (res < 0.55f)
				eyeVal = (res -
0.f) * (0.2f - 0.0f) / (0.55f - 0.0f) + 0.0f;
			else  if (res < 0.75f)
				eyeVal = (res -
0.55f) * (0.6f - 0.4f) / (0.75f - 0.55f) + 0.4f;
			else
				eyeVal = (res -
0.75f) * (1.0f - 0.8f) / (1.0f - 0.75f) + 0.8f;

			if (eyeVal > 1.f)
				eyeVal = 1.f;

			//EyeBrow
			a = (facesLandmarks[i][36]
+ facesLandmarks[i][39]) / 2;

			fd = distance(a,
facesLandmarks[i][19]);
			sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][45]);

			res = fd / sd /
EYEBROWMAX;
			float eyeBrowVal = 0.f;

			if (res < 0.65f)
				eyeBrowVal = (res
- 0.f) * (0.2f - 0.0f) / (0.65f - 0.0f) + 0.0f;
			else  if (res < 0.75f)
				eyeBrowVal = (res
- 0.65f) * (0.6f - 0.4f) / (0.75f - 0.65f) + 0.4f;
			else
				eyeBrowVal = (res
- 0.75f) * (1.0f - 0.8f) / (1.0f - 0.75f) + 0.8f;

			if (eyeBrowVal > 1.f)
				eyeBrowVal = 1.f;

			//Lips

			fd =
distance(facesLandmarks[i][36],
facesLandmarks[i][45]);
			sd =
distance(facesLandmarks[i][48],
facesLandmarks[i][54]);

			res = sd / fd / MOUTHMAX;
			float mouthVal = 0.f;
			if (res < 0.65f)
				mouthVal = (res -
0.0f) * (0.2f - 0.0f) / (0.65f - 0.f) + 0.0f;
			else  if (res < 0.75f)
				mouthVal = (res -
0.65f) * (0.6f - 0.4f) / (0.75f - 0.65f) + 0.4f;
			else
				mouthVal = (res -
0.75f) * (1.0f - 0.8f) / (1.f - 0.75f) + 0.8f;

			if (mouthVal > 1.f)
				mouthVal = 1.f;


		facesData.emplace_back(fname, eyeBrowVal,
eyeVal, mouthVal);
		}

		++di;

		std::cout << "Processed[" << top++
<< "/" << cnt << "]" << std::endl;
	}
}

void Detector::detectAreaMethod(std::string
folderName)
{
	path p(folderName + std::string("/"));

	if (!exists(p))
	{
		std::cout << folderName << "
folder not found" << std::endl;
		return;
	}

	directory_iterator end_itr;

	std::vector<size_t> index;

	std::vector<std::string> dirs;
	for (directory_iterator itr(p); itr !=
end_itr; ++itr)
	{
		if (is_directory(itr->path())) {
			dirs.push_back(itr-
>path().string());
		}
```

```
	}

	std::vector<cv::Rect> faces;
	std::vector<std::vector<cv::Point2f>>faces
Landmarks;

	std::string fname;

	size_t cnt = std::count_if(
		directory_iterator(p),
		directory_iterator(),
		static_cast<bool(*)(const
path&)>(is_regular_file));

	size_t top = 1;

	directory_iterator di(p);
	directory_iterator di_end;

	while (di != di_end)
	{
		fname = p.string() + di-
>path().filename().string();
		cv::Mat image, gray;

		image = cv::imread(fname,
cv::IMREAD_COLOR);
		if (image.empty())
		{
			std::cerr << fname << " :
Image not found" << std::endl;
			++di;
			continue;
		}
		cvtColor(image, gray,
cv::COLOR_BGR2GRAY);
		haarCascade.detectMultiScale(gray,
faces);
		facemarkDetector->fit(gray, faces,
facesLandmarks);

		for (size_t i = 0; i <
faces.size(); i++)
		{
			Transformation2D
tr2d(facesLandmarks[i][36], facesLandmarks[i][39],
facesLandmarks[i][42], facesLandmarks[i][45]);
			//Eye
			cv::Point2f
a(tr2d.getPoint(facesLandmarks[i], 36).x,
tr2d.getPoint(facesLandmarks[i], 37).y);
			cv::Point2f
b(tr2d.getPoint(facesLandmarks[i], 39).x,
tr2d.getPoint(facesLandmarks[i], 40).y);
			//cv::Point2f
b(facesLandmarks[i][39].x,
facesLandmarks[i][40].y);

		/*	float fd = distance(a, b);
			float sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][39]);*/
			float eyearea =
std::fabsf(a.x - b.x) * std::fabsf(a.y - b.y);
			eyearea /= MAX_AREA_EYE;
			//float res = fd / sd /
EYEMAX;
			std::cout << fname << "
:eyearea: " << eyearea << std::endl;
			//float eyeVal = 0.f;
			if (eyearea < 0.60f)
				eyearea = (eyearea
- 0.f) * (0.2f - 0.0f) / (0.60f - 0.0f) + 0.0f;
			else  if (eyearea < 0.70f)
				eyearea = (eyearea
- 0.60f) * (0.6f - 0.4f) / (0.70f - 0.60f) + 0.4f;
			else
				eyearea = (eyearea
- 0.70f) * (1.0f - 0.8f) / (1.0f - 0.70f) + 0.8f;

			if (eyearea > 1.f)
				eyearea = 1.f;

			//EyeBrow
			a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i],
17).x, tr2d.getPoint(facesLandmarks[i], 19).y);
			b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i],
21).x, tr2d.getPoint(facesLandmarks[i], 39).y);
			//a =
cv::Point2f(facesLandmarks[i][17].x,
facesLandmarks[i][19].y);
			//b =
cv::Point2f(facesLandmarks[i][21].x,
facesLandmarks[i][39].y);
			//fd = distance(a,
facesLandmarks[i][19]);
			//sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][45]);
			float eyeabrowarea =
std::fabsf(a.x - b.x) * std::fabsf(a.y - b.y);
			eyeabrowarea /=
MAX_AREA_EYEBROW;
			//float res = fd / sd /
EYEMAX;
			std::cout << fname << "
:eyebrowarea: " << eyeabrowarea << std::endl;
			//res = fd / sd /
EYEBROWMAX;
			//float eyeBrowVal = 0.f;

			if (eyeabrowarea < 0.625f)
				eyeabrowarea =
(eyeabrowarea - 0.f) * (0.2f - 0.0f) / (0.625f -
0.0f) + 0.0f;
			else  if (eyeabrowarea <
0.65f)
				eyeabrowarea =
(eyeabrowarea - 0.625f) * (0.6f - 0.4f) / (0.65f -
0.625f) + 0.4f;
			else
				eyeabrowarea =
(eyeabrowarea - 0.65f) * (1.0f - 0.8f) / (1.0f -
0.65f) + 0.8f;

			if (eyeabrowarea > 1.f)
				eyeabrowarea =
1.f;


			//Lips

			a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i],
60).x, tr2d.getPoint(facesLandmarks[i], 50).y);
			b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i],
64).x, tr2d.getPoint(facesLandmarks[i], 57).y);

			//a =
cv::Point2f(facesLandmarks[i][60].x,
facesLandmarks[i][50].y);
			//b =
cv::Point2f(facesLandmarks[i][64].x,
facesLandmarks[i][57].y);

			float lipsarea =
std::fabsf(a.x - b.x) * std::fabsf(a.y - b.y);
			//float res = fd / sd /
EYEMAX;
			lipsarea /= MAX_AREA_LIPS;
			std::cout << fname << "
:lipsarea: " << lipsarea << std::endl;
```

```
                        //float mouthVal = 0.f;
                        //res = sd / fd /
MOUTHMAX;
                        if (lipsarea < 0.30f)
                                lipsarea =
(lipsarea - 0.0f) * (0.2f - 0.0f) / (0.30f - 0.f)
+ 0.0f;
                        else  if (lipsarea <
0.45f)
                                lipsarea =
(lipsarea - 0.30f) * (0.6f - 0.4f) / (0.45f -
0.30f) + 0.4f;
                        else
                                lipsarea =
(lipsarea - 0.45f) * (1.0f - 0.8f) / (1.f - 0.45f)
+ 0.8f;

                        if (lipsarea > 1.f)
                                lipsarea = 1.f;


        facesData.emplace_back(fname,
eyeabrowarea, eyearea, lipsarea);
                }

                ++di;

                std::cout << "Processed[" << top++
<< "/" << cnt << "]" << std::endl;
        }

}

void Detector::saveImageFL(std::string fileName)
{
        cv::Mat image, gray;

        image = cv::imread(fileName,
cv::IMREAD_COLOR);
        if (image.empty())
        {
                std::cerr << fileName << " : Image
not found" << std::endl;
                return;
        }

        std::vector<cv::Rect> faces;
        std::vector<std::vector<cv::Point2f>>faces
Landmarks;

        cvtColor(image, gray, cv::COLOR_BGR2GRAY);
        haarCascade.detectMultiScale(gray, faces);
        facemarkDetector->fit(gray, faces,
facesLandmarks);

        for (size_t i = 0; i <
facesLandmarks.size(); i++)
        {
                for (size_t j = 0; j <
facesLandmarks[i].size(); j++)
                {
                        cv::circle(image,
facesLandmarks[i][j], 1, cv::Scalar(0, 0, 255),
3);
                }
        }

        cv::imwrite(fileName, image);
        std::cout << "Image saved" << std::endl;
}

void Detector::saveImagesFLfromFolder(std::string
folderName)
{
        path p(folderName + std::string("/"));

        if (!exists(p))
        {
                std::cout << folderName << "
folder not found" << std::endl;
                return;
        }

        directory_iterator end_itr;

        std::vector<size_t> index;

        std::vector<std::string> dirs;
        for (directory_iterator itr(p); itr !=
end_itr; ++itr)
        {
                if (is_directory(itr->path())) {
                        dirs.push_back(itr-
>path().string());
                }
        }

        std::vector<cv::Rect> faces;
        std::vector<std::vector<cv::Point2f>>faces
Landmarks;

        std::string fname;

        size_t cnt = std::count_if(
                directory_iterator(p),
                directory_iterator(),
                static_cast<bool(*)(const
path&)>(is_regular_file));

        size_t top = 1;

        directory_iterator di(p);
        directory_iterator di_end;

        while (di != di_end)
        {
                fname = p.string() + di-
>path().filename().string();
                cv::Mat image, gray;

                image = cv::imread(fname,
cv::IMREAD_COLOR);
                if (image.empty())
                {
                        std::cerr << fname << " :
Image not found" << std::endl;
                        ++di;
                        continue;
                }
                cvtColor(image, gray,
cv::COLOR_BGR2GRAY);
                haarCascade.detectMultiScale(gray,
faces);
                facemarkDetector->fit(gray, faces,
facesLandmarks);

                for (size_t i = 0; i <
faces.size(); i++)
                {
                        saveImageFL(fname);
                }

                ++di;

                std::cout << "Processed[" << top++
<< "/" << cnt << "] " << fname << std::endl;
        }
}

void Detector::detectGeometryMethod(std::string
folderName)
{
        path p(folderName + std::string("/"));

        if (!exists(p))
        {
```

```
                std::cout << folderName << "
folder not found" << std::endl;
                return;
        }

        directory_iterator end_itr;

        std::vector<size_t> index;

        std::vector<std::string> dirs;
        for (directory_iterator itr(p); itr !=
end_itr; ++itr)
        {
                if (is_directory(itr->path())) {
                        dirs.push_back(itr-
>path().string());
                }
        }

        std::vector<cv::Rect> faces;
        std::vector<std::vector<cv::Point2f>>faces
Landmarks;

        std::string fname;

        size_t cnt = std::count_if(
                directory_iterator(p),
                directory_iterator(),
                static_cast<bool(*)(const
path&)>(is_regular_file));

        size_t top = 1;

        directory_iterator di(p);
        directory_iterator di_end;

        while (di != di_end)
        {
                fname = p.string() + di-
>path().filename().string();
                cv::Mat image, gray;

                image = cv::imread(fname,
cv::IMREAD_COLOR);
                if (image.empty())
                {
                        std::cerr << fname << " :
Image not found" << std::endl;
                        ++di;
                        continue;
                }
                cvtColor(image, gray,
cv::COLOR_BGR2GRAY);
                haarCascade.detectMultiScale(gray,
faces);
                facemarkDetector->fit(gray, faces,
facesLandmarks);

                float eyebrow1, eyebrow2,
eyebrow3, eyebrow4, eye1, lips1, lips2;

                for (size_t i = 0; i <
faces.size(); i++)
                {
                        Transformation2D
tr2d(facesLandmarks[i][36], facesLandmarks[i][39],
facesLandmarks[i][42], facesLandmarks[i][45]);

                        //Eyebrow
                        // #1 p18-p36

                        cv::Point2f
a(tr2d.getPoint(facesLandmarks[i][18]));
                        cv::Point2f
b(tr2d.getPoint(facesLandmarks[i][36]));

                        eyebrow1 = distance(a, b);

                        // #2 p19-p36
                        a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][19]));
                        b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][36]));

                        eyebrow2 = distance(a, b);

                        // #3 p20-p39
                        a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][20]));
                        b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][39]));

                        eyebrow3 = distance(a, b);

                        // #4 p21-p39
                        a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][21]));
                        b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][39]));

                        eyebrow4 = distance(a, b);


                        //Lips
                        //#1 p51-p57
                        a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][51]));
                        b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][57]));

                        lips1 = distance(a, b);

                        // #2 p60-p64
                        a =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][60]));
                        b =
cv::Point2f(tr2d.getPoint(facesLandmarks[i][64]));

                        lips2 = distance(a, b);


        facesData2.emplace_back(fname, eyebrow2,
eyebrow3, eye1, lips1, lips2);
                }

                ++di;

                std::cout << "Processed[" << top++
<< "/" << cnt << "] " << fname << std::endl;
        }


}

void Detector::writeToTxt(std::string
saveFilename)
{
        if (facesData.empty())
                return;

        std::ofstream ofst("res.txt");

        size_t columnIndex = 1;
        size_t i1 = 0;
        for (size_t exIndex = 4; exIndex <
facesData.size() + 4; columnIndex++, exIndex += 4,
i1++)
        {
                ofst << 100 + i1 << "| " << "" <<
':' << int(facesData[exIndex - 2].x1 * 100 + 0.5f)
/ 100.f << ':';
                ofst << int(facesData[exIndex -
2].x2 * 100 + 0.5f) / 100.f << ':';
                ofst << int(facesData[exIndex -
2].x3 * 100 + 0.5f) / 100.f;
                ofst << '\n';
```

```
                ofst << 200 + i1 << "| " << "" <<
':' << int(facesData[exIndex - 1].x1 * 100 + 0.5f)
/ 100.f << ':';
                ofst << int(facesData[exIndex -
1].x2 * 100 + 0.5f) / 100.f << ':';
                ofst << int(facesData[exIndex -
1].x3 * 100 + 0.5f) / 100.f;
                ofst << '\n';
                ofst << 300 + i1 << "| " << "" <<
':' << int(facesData[exIndex - 4].x1 * 100 + 0.5f)
/ 100.f << ':';
                ofst << int(facesData[exIndex -
4].x2 * 100 + 0.5f) / 100.f << ':';
                ofst << int(facesData[exIndex -
4].x3 * 100 + 0.5f) / 100.f;
                ofst << '\n';
                ofst << 400 + i1 << "| " << "" <<
':' << int(facesData[exIndex - 3].x1 * 100 + 0.5f)
/ 100.f << ':';
                ofst << int(facesData[exIndex -
3].x2 * 100 + 0.5f) / 100.f << ':';
                ofst << int(facesData[exIndex -
3].x3 * 100 + 0.5f) / 100.f;
                ofst << '\n';

                std::cout << "Add[" << exIndex - 3
<< "/" << facesData.size() << "]" << std::endl;
                std::cout << "Add[" << exIndex - 2
<< "/" << facesData.size() << "]" << std::endl;
                std::cout << "Add[" << exIndex - 1
<< "/" << facesData.size() << "]" << std::endl;
                std::cout << "Add[" << exIndex <<
"/" << facesData.size() << "]" << std::endl;
        }

        ofst.close();
        std::cout << "Txt file saved" << '\n';
}

void Detector::writeToTxt2(std::string
saveFilename)
{
        if (facesData2.empty())
                return;

        std::ofstream ofst("res.txt");

        size_t columnIndex = 1;
        size_t i1 = 1;
        for (size_t exIndex = 0; exIndex <
facesData2.size(); exIndex++)
        {
                ofst << 100 * i1 + (exIndex / 4)
<< ' ';
                ofst << int(facesData2[exIndex].x1
* 100 + 0.5f) / 100.f << ' ';
                ofst << int(facesData2[exIndex].x2
* 100 + 0.5f) / 100.f << ' ';
                ofst << int(facesData2[exIndex].x3
* 100 + 0.5f) / 100.f << ' ';
                ofst << int(facesData2[exIndex].x4
* 100 + 0.5f) / 100.f << ' ';
                ofst << int(facesData2[exIndex].x5
* 100 + 0.5f) / 100.f;
                ofst << '\n';

                ++i1;
                if (i1 == 5)
                        i1 = 1;
        }

        ofst.close();
        std::cout << "Txt file saved" << '\n';
}

void Detector::writeToXlxsPic(std::string
saveFilename)
{
        if (facesData.empty())
                return;

        libxl::Book* book = xlCreateBook();
        assert(book);

        libxl::Format* format = book->addFormat();
        format->setAlignH(libxl::ALIGNH_CENTER);
        format->setAlignV(libxl::ALIGNV_CENTER);

        libxl::Sheet* facesSheet = book-
>addSheet("Faces");
        assert(facesSheet);
        int imagePos = 21;
        int imagePosOld = 1;
        for (size_t i = 0; i < facesData.size();
i++, imagePos += 20)
        {
                int id = book-
>addPicture(facesData[i].name.c_str());
                facesSheet-
>setPicture(imagePosOld, 0, id, 0.5);

                facesSheet->setMerge(imagePosOld,
imagePos, 10, 10);
                facesSheet->setMerge(imagePosOld,
imagePos, 11, 11);
                facesSheet->setMerge(imagePosOld,
imagePos, 12, 12);
                facesSheet->setMerge(imagePosOld,
imagePos, 13, 13);

                imagePos += 2;

                facesSheet->writeStr(imagePosOld,
10, facesData[i].name.c_str(), format);
                facesSheet->writeNum(imagePosOld,
11, facesData[i].x1, format);
                facesSheet->writeNum(imagePosOld,
12, facesData[i].x2, format);
                facesSheet->writeNum(imagePosOld,
13, facesData[i].x3, format);

                std::cout << "Add[" << i + 1 <<
"/" << facesData.size() << "]" << std::endl;

                imagePosOld = imagePos;
        }

        book->save((saveFilename +
std::string(".xls")).c_str());
        book->release();

        std::cout << "xlsx file saved" <<
std::endl;
}

const float Detector::EYEMAX = 0.45f;
const float Detector::MOUTHMAX = 0.96f;
const float Detector::EYEBROWMAX = 0.4f;

const float Detector::MAX_AREA_EYE = 0.09f;
const float Detector::MAX_AREA_LIPS = 0.55f;
const float Detector::MAX_AREA_EYEBROW = 0.45f;
```

Transformation2D.h

```
#ifndef _2D_H_
#define _2D_H_

#include <vector>

#include "opencv2/highgui/highgui.hpp"

class Transformation2D
{
        float XL;
```

```
        float YL;
        float XR;
        float YR;
public:
        Transformation2D(cv::Point2f point36,
cv::Point2f point39, cv::Point2f point42,
cv::Point2f point45);
        //get point
        cv::Point2f getPoint(const cv::Point2f &
point);

        static cv::Point2f getPoint(const
std::vector<cv::Point2f> & points, size_t
pointIndex);
        static float distance(cv::Point2f a,
cv::Point2f b);
};

#endif // !_2D_H_
```

Transformation2D.cpp

```
#include "Transformation2D.h"

Transformation2D::Transformation2D(cv::Point2f
point36, cv::Point2f point39, cv::Point2f point42,
cv::Point2f point45)
{
        XL = (point45.x + point42.x) / 2.0;
        YL = (point45.y + point42.y) / 2.0;
        XR = (point39.x + point36.x) / 2.0;
        YR = (point39.y + point36.y) / 2.0;
}

cv::Point2f Transformation2D::getPoint(const
cv::Point2f & point)
{
        float X0 = (XL + XR) / 2.0;
        float Y0 = (YL + YR) / 2.0;

        float DX = XR - XL;
        float DY = YR - YL;

        float L = std::sqrtf(std::powf(DX, 2) +
std::powf(DY, 2));

        float sin_AL = DY / L;
        float cos_AL = DX / L;

        float X_User = 0;
        float Y_User = 0;

        auto getPointXY = [&X0, &Y0, L, &sin_AL,
&cos_AL, &X_User, &Y_User](float x1, float y1)
        {
                float X_User_0 = (x1 - X0) / L;
                float Y_User_0 = -(y1 - Y0) / L;
                X_User = X_User_0 * cos_AL -
Y_User_0 * sin_AL;
                Y_User = X_User_0 * sin_AL -
Y_User_0 * cos_AL;
        };

        getPointXY(point.x, point.y);

        return cv::Point2f(X_User, Y_User);
}

cv::Point2f Transformation2D::getPoint(const
std::vector<cv::Point2f> & points, size_t
pointIndex)
{
        float XL = (points[45].x + points[42].x) /
2.0;
        float YL = (points[45].y + points[42].y) /
2.0;
        float XR = (points[39].x + points[36].x) /
2.0;
        float YR = (points[39].y + points[36].y) /
2.0;

        float X0 = (XL + XR) / 2.0;
        float Y0 = (YL + YR) / 2.0;

        float DX = XR - XL;
        float DY = YR - YL;

        float L = std::sqrtf(std::powf(DX, 2) +
std::powf(DY, 2));

        float sin_AL = DY / L;
        float cos_AL = DX / L;

        float X_User = 0;
        float Y_User = 0;

        auto getPointXY = [&X0, &Y0, L, &sin_AL,
&cos_AL, &X_User, &Y_User](float x1, float y1)
        {
                float X_User_0 = (x1 - X0) / L;
                float Y_User_0 = -(y1 - Y0) / L;
                X_User = X_User_0 * cos_AL -
Y_User_0 * sin_AL;
                Y_User = X_User_0 * sin_AL -
Y_User_0 * cos_AL;
        };

        getPointXY(points[pointIndex].x,
points[pointIndex].y);

        return cv::Point2f(X_User, Y_User);
}

float Transformation2D::distance(cv::Point2f a,
cv::Point2f b)
{
        return sqrtf(std::powf(a.x - b.x, 2) +
std::powf(a.y - b.y, 2));
}
```

Source.cpp

```
#include "Detector.h"

#include<iostream>

int main()
{
        Detector detector;
        detector.detectGeometryMethod("testPhoto1"
);
        detector.writeToTxt2("res");

        std::cin.get();

        return 0;
}
```

SMACOF&MDS

Data.h

```
#pragma once

#include <vector>

class Data
{
public:
```

```
        static void
doubleCenter(std::vector<std::vector<double>>&
matrix);

        static void
multiply(std::vector<std::vector<double>>& matrix,
double factor);

        static void
squareEntries(std::vector<std::vector<double>>&
matrix);

        static void
normalize(std::vector<std::vector<double>>& x);

        static double
normalize(std::vector<double>& x);

        static double prod(std::vector<double>& x,
std::vector<double>& y);

    static void
eigen(std::vector<std::vector<double>>& matrix,
std::vector<std::vector<double>>& evecs,
std::vector<double>& evals);

        static void
randomize(std::vector<std::vector<double>>&
matrix);

        static std::vector<int>
landmarkIndices(std::vector<std::vector<double>>
matrix);

        static std::vector<std::vector<double>>
copyMatrix(std::vector<std::vector<double>>
matrix);
};
```

Data.cpp

```
#include "Data.h"

#include <cmath>
#include <ctime>
#include <algorithm>

#include <iostream>

void
Data::doubleCenter(std::vector<std::vector<double>
>& matrix)
{
        int n = matrix[0].size();
        int k = matrix.size();

        for (int j = 0; j < k; j++) {
                double avg = 0.0;
                for (int i = 0; i < n; i++) avg +=
matrix[j][i];
                avg /= n;
                for (int i = 0; i < n; i++)
matrix[j][i] -= avg;
        }

        for (int i = 0; i < n; i++) {
                double avg = 0.0;
                for (int j = 0; j < k; j++) avg +=
matrix[j][i];
                avg /= matrix.size();
                for (int j = 0; j < k; j++)
matrix[j][i] -= avg;
        }
}

void
Data::multiply(std::vector<std::vector<double>>&
matrix, double factor)
{
        for (int i = 0; i < matrix.size(); i++) {
                for (int j = 0; j <
matrix[0].size(); j++) {
                        matrix[i][j] *= factor;
                }
        }
}

void
Data::squareEntries(std::vector<std::vector<double
>>& matrix)
{
        int n = matrix[0].size();
        int k = matrix.size();
        for (int i = 0; i < k; i++) {
                for (int j = 0; j < n; j++) {
                        matrix[i][j] =
std::pow(matrix[i][j], 2.0);
                }
        }
}

void
Data::normalize(std::vector<std::vector<double>>&
x)
{
        for (int i = 0; i < x.size(); i++)
                normalize(x[i]);
}

double Data::normalize(std::vector<double>& x)
{
        double norm = std::sqrt(prod(x, x));
        for (int i = 0; i < x.size(); i++)
                x[i] /= norm;

        return norm;
}

double Data::prod(std::vector<double>& x,
std::vector<double>& y)
{
        double result = 0.0;
        int length = x.size() < y.size() ?
x.size() : y.size(); // min(x.size(), y.size());
        for (int i = 0; i < length; i++)
                result += x[i] * y[i];

        return result;
}

void Data::eigen(std::vector<std::vector<double>>&
matrix, std::vector<std::vector<double>>& evecs,
std::vector<double>& evals)
{
        //
        double maxr = 0.0;
        //

        int d = evals.size();
        int k = matrix.size();
        double d1 = 1.e-05;
        double r = 0.0;
        for (int m = 0; m < d; m++) {
                double eps;
                evals[m] =
Data::normalize(evecs[m]);
        }
        int iterations = 0;
        std::vector<std::vector<double>>
tempOld(d, std::vector<double>(k, 0.0)); //[d][k];

        while (r < 0.9999900000000001) {
```

```
                for (int m = 0; m < d; m++) {
                        for (int i = 0; i < k;
i++) {
                                tempOld[m][i] =
evecs[m][i];
                                evecs[m][i] = 0.0;
                        }
                }

                for (int m = 0; m < d; m++) {
                        for (int i = 0; i < k;
i++)
                                for (int j = 0; j
< k; j++)

        evecs[m][j] += matrix[i][j] *
tempOld[m][i];
                }
                for (int m = 0; m < d; m++) {
                        for (int p = 0; p < m;
p++) {
                                double fac =
Data::prod(evecs[p], evecs[m]) /
Data::prod(evecs[p], evecs[p]);
                                for (int i = 0; i
< k; i++) evecs[m][i] -= fac * evecs[p][i];
                        }
                }

                for (int m = 0; m < d; m++)
                {
                        evals[m] =
Data::normalize(evecs[m]);
                }

                r = 1.0;
                for (int m = 0; m < d; m++)
                {
                        r =
std::min(std::abs(Data::prod(evecs[m],
tempOld[m])), r);
                }

                iterations++;

                if (r > maxr)
                        maxr = r;


                if (iterations % 10000 == 0) {
                        std::cout << iterations <<
std::endl;

                        std::cout << evals[2] << "
" << evals[3] << " " << evals[4] << std::endl;
                }
                if (iterations == 250000)
                        break;


        }
        std::cout << "Max r : " << maxr <<
std::endl;
        for (size_t i = 0; i < evals.size(); i++)
        {
                std::cout << "d" << i <<": "<<
evals[i] << std::endl;
        }
}


void
Data::randomize(std::vector<std::vector<double>>&
matrix)
{
        srand(time(nullptr));

        for (int i = 0; i < matrix.size(); i++) {
                for (int j = 0; j <
matrix[0].size(); j++) {
                        matrix[i][j] = rand() /
(RAND_MAX + 1.0); // 0...1
                }
        }
}

std::vector<int>
Data::landmarkIndices(std::vector<std::vector<doub
le>> matrix)
{
        int k = matrix.size();
        int n = matrix[0].size();
        std::vector<int> result(k, 0);
        for (int i = 0; i < k; i++) {
                for (int j = 0; j < n; j++) {
                        if (matrix[i][j] == 0.0) {
                                result[i] = j;
                        }
                }
        }
        return result;
}

std::vector<std::vector<double>>
Data::copyMatrix(std::vector<std::vector<double>>
matrix)
{
        std::vector<std::vector<double>>
copy(matrix.size(),
std::vector<double>(matrix[0].size(), 0.0));
        for (int i = 0; i < matrix.size(); i++) {
                for (int j = 0; j <
matrix[0].size(); j++) {
                        copy[i][j] = matrix[i][j];
                }
        }
        return copy;
}
```

MDS.h

```
#pragma once

#include <vector>

class MDS
{
public:

        static std::vector<std::vector<double>>
fullmds(std::vector<std::vector<double>> d, int
dim);

        static std::vector<std::vector<double>>
classicalScaling(std::vector<std::vector<double>>
distances, int dim);

        static std::vector<std::vector<double>>
distanceScaling(std::vector<std::vector<double>>
d, int dim);

        static std::vector<std::vector<double>>
distanceScaling(std::vector<std::vector<double>>
d, int dim, int iter, int threshold, int timeout);
};
```

MDS.cpp

```
#include "MDS.h"
#include "Data.h"
#include "SMACOF.h"

#include <fstream>
```

```
std::vector<std::vector<double>>
MDS::fullmds(std::vector<std::vector<double>> d,
int dim)
{
	std::vector<std::vector<double>>
result(dim, std::vector<double>(d.size(), 0.0));

	Data::randomize(result);
	std::vector<double> evals(result.size(),
0.0);
	Data::squareEntries(d);
	Data::doubleCenter(d);
	Data::multiply(d, -0.5);
	Data::eigen(d, result, evals);

	for (int i = 0; i < result.size(); i++) {
		evals[i] = std::sqrt(evals[i]);
		for (int j = 0; j <
result[0].size(); j++) {
			result[i][j] *= evals[i];
		}
	}

	return result;
}

std::vector<std::vector<double>>
MDS::classicalScaling(std::vector<std::vector<doub
le>> distances, int dim)
{
	std::vector<std::vector<double>> d =
Data::copyMatrix(distances);
	return fullmds(d, dim);
}

std::vector<std::vector<double>>
MDS::distanceScaling(std::vector<std::vector<doubl
e>> d, int dim)
{
	std::vector<std::vector<double>> x =
classicalScaling(d, dim);
	SMACOF smacof(d, x);
	smacof.iterate(100, 3, 10 * 60000);
	return smacof.getPositions();
}

std::vector<std::vector<double>>
MDS::distanceScaling(std::vector<std::vector<doubl
e>> d, int dim, int iter, int threshold, int
timeout)
{
	std::vector<std::vector<double>> x =
classicalScaling(d, dim);
	SMACOF smacof(d, x);
	smacof.iterate(iter, threshold, timeout);
	return smacof.getPositions();
}
```

SMACOF.h

```
#pragma once

#include <vector>
#include <string>

class SMACOF
{
	std::vector<std::vector<double>> x;
	std::vector<std::vector<double>> d;
	std::vector<std::vector<double>> w;

public:

	/**
	* Construct a new SMACOF instance.
	* @param d distance matrix
	* @param x initial coordinate matrix
	* @param w weights matrix
	*/
	SMACOF(std::vector<std::vector<double>> d,
std::vector<std::vector<double>> x,
std::vector<std::vector<double>> w);

	/**
	* Construct a SMACOF instance without
weights.
	* @param d distance matrix
	* @param x initial coordinate matrix
	*/
	SMACOF(std::vector<std::vector<double>> d,
std::vector<std::vector<double>> x);

	std::vector<std::vector<double>>
getDissimilarities();

	std::vector<std::vector<double>>
getWeights();

	std::vector<std::vector<double>>
getPositions();

	void
setDissimilarities(std::vector<std::vector<double>
> d);

	void
setWeights(std::vector<std::vector<double>> w);

	void
setPositions(std::vector<std::vector<double>> x);

	/**
	* Perform 1 majorization iteration using
this SMACOF instance.
	* @return report
	*/
	std::string iterate();

	/**
	* Perform n majorization iterations using
this SMACOF instance.
	* @param n number of iterations
	* @return report
	*/
	std::string iterate(int n);

	/**
	* Perform majorization iterations until
the maximum number of iterations
	* is reached, the maximum runtime has
elapsed, or the change in
	* normalized stress falls below the
threshold, whichever comes first.
	* @param iter maximum number of iterations
	* @param threshold threshold for change in
normalized stress
	* @param timeout maximum runtime in
milliseconds
	* @return report
	*/
	std::string iterate(int iter, int
threshold, int timeout);

	/**
	* Compute the absolute stress for this
SMACOF instance.
	* @return stress
	*/
	double getStress();

	/**
	* Compute the normalized stress for this
SMACOF instance.
	* @return normalized stress
	*/
```

```
        double getNormalizedStress();

        /**
        * Element-wise matrix exponentiation for
self-weighting of distances.
        * @param D distance matrix or initial
weights
        * @param exponent power to raise each
element the matrix
        * @return exponentiated weights
        */
        static std::vector<std::vector<double>>
weightMatrix(std::vector<std::vector<double>> D,
double exponent);

        /**
        * SMACOF algorithm (weighted).
        * @param x coordinates matrix
        * @param d distance matrix
        * @param w weights matrix
        * @param iter maximum iterations
        * @param threshold halting threshold for
change in normalized stress
        * @param timeout maximum runtime in
milliseconds
        * @return report
        */
        static std::string
majorize(std::vector<std::vector<double>>& x,
std::vector<std::vector<double>>& d,
                std::vector<std::vector<double>>&
w, int iter, int threshold, int timeout);

        /**
        * SMACOF algorithm (unweighted).
        * @param x coordinates matrix
        * @param d distance matrix
        * @param iter maximum iterations
        * @param threshold halting threshold for
change in normalized stress
        * @param timeout maximum runtime in
milliseconds
        * @return report
        */
        static std::string
majorize(std::vector<std::vector<double>>& x,
std::vector<std::vector<double>>& d,
                int iter, int threshold, int
timeout);

        /**
        * Bare SMACOF algorithm. Convenient for
reading the algorithm.
        * @param x coordinates matrix
        * @param d distance matrix
        * @param w weights matrix
        * @param iter number of iterations
        */
        static void
majorize(std::vector<std::vector<double>>& x,
std::vector<std::vector<double>>& d,
                std::vector<std::vector<double>>&
w, int iter);

        /**
        * Compute the absolute stress between a
weighted distance matrix and a configuration of
coordinates.
        * @param d distance matrix
        * @param w weights matrix
        * @param x coordinates matrix
        * @return stress
        */
        static double
stress(std::vector<std::vector<double>> &d,
std::vector<std::vector<double>>& w,
std::vector<std::vector<double>>& x);

        /**
        * Compute the absolute stress between a
distance matrix and a configuration of
coordinates.
        * @param d distance matrix
        * @param x coordinates matrix
        * @return stress
        */
        static double
stress(std::vector<std::vector<double>>& d,
std::vector<std::vector<double>>& x);

        /**
        * Compute the normlized stress between a
weighted distance matrix and a configuration of
coordinates.
        * @param d distance matrix
        * @param w weights matrix
        * @param x coordinate matrix
        * @return normalized stress
        */
        static double
normalizedStress(std::vector<std::vector<double>>&
d, std::vector<std::vector<double>>& w,
                std::vector<std::vector<double>>&
x);

        /**
        * Return the normlized stress between a
distance matrix and a configuration of
coordinates.
        * @param d distance matrix
        * @param x coordinate matrix
        * @return normalized stress
        */
        static double
normalizedStress(std::vector<std::vector<double>>&
d, std::vector<std::vector<double>>& x);

};
```

SMACOF.cpp

```
#include "SMACOF.h"

#include <chrono>

SMACOF::SMACOF(std::vector<std::vector<double>> d,
std::vector<std::vector<double>> x,
std::vector<std::vector<double>> w)
{
        this->x = x;
        this->d = d;
        this->w = w;
}

SMACOF::SMACOF(std::vector<std::vector<double>> d,
std::vector<std::vector<double>> x)
{
        this->x = x;
        this->d = d;
}

std::vector<std::vector<double>>
SMACOF::getDissimilarities()
{
        return this->d;
}

std::vector<std::vector<double>>
SMACOF::getWeights()
{
        return this->w;
}

std::vector<std::vector<double>>
SMACOF::getPositions()
```

```
{
	return this->x;
}

void
SMACOF::setDissimilarities(std::vector<std::vector
<double>> d)
{
	this->d = d;
}

void
SMACOF::setWeights(std::vector<std::vector<double>
> w)
{
	this->w = w;
}

void
SMACOF::setPositions(std::vector<std::vector<doubl
e>> x)
{
	this->x = x;
}

std::string SMACOF::iterate()
{
	return iterate(1);
}

std::string SMACOF::iterate(int n)
{
	if (!this->w.empty())
		return majorize(this->x, this->d,
this->w, n, 0, 0);
	return majorize(this->x, this->d, n, 0,
0);
}


std::string SMACOF::iterate(int iter, int
threshold, int timeout)
{
	if (!this->w.empty())
		return majorize(this->x, this->d,
this->w, iter, threshold, timeout);
	return majorize(this->x, this->d, iter,
threshold, timeout);
}

double SMACOF::getStress()
{
	if (!this->w.empty())
		return stress(this->d, this->w,
this->x);
	return stress(this->d, this->x);
}

double SMACOF::getNormalizedStress()
{
	if (!this->w.empty())
		return normalizedStress(this->d,
this->w, this->x);
	return normalizedStress(this->d, this->x);
}

std::vector<std::vector<double>>
SMACOF::weightMatrix(std::vector<std::vector<doubl
e>> D, double exponent)
{
	int n = D[0].size();
	int k = D.size();
	std::vector<std::vector<double>> result(k,
std::vector<double>(n, 0.0));
	for (int i = 0; i < k; i++)
		for (int j = 0; j < n; j++)
			if (D[i][j] > 0.0)
				result[i][j] =
std::pow(D[i][j], exponent);
	return result;
}

std::string
SMACOF::majorize(std::vector<std::vector<double>>
&x, std::vector<std::vector<double>> &d,
	std::vector<std::vector<double>>& w, int
iter, int threshold, int timeout)
{
	std::string report = "";
	int n = x[0].size();
	int k = d.size();
	int dim = x.size();

	std::vector<double> wSum(n, 0.0);
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < k; j++) {
			wSum[i] += w[j][i];
		}
	}
	double eps = std::pow(10.0, -threshold);
	auto start =
std::chrono::steady_clock::now();
	if (iter == 0)
		iter = 10000000;
	for (int c = 0; c < iter; c++) {
		double change = 0.0;
		double magnitude = 0.0;
		for (int i = 0; i < n; i++) {
			std::vector<double>
xnew(dim, 0.0);
			for (int j = 0; j < k;
j++) {
				double inv = 0.0;
				for (int m = 0; m
< dim; m++) {
					inv +=
std::pow(x[m][i] - x[m][j], 2.0);
				}
				if (inv != 0.0)
inv = std::pow(inv, -0.5);
				for (int m = 0; m
< dim; m++) {
					xnew[m] +=
w[j][i] * (x[m][j] +

	d[j][i] * (x[m][i] - x[m][j]) * inv);
				}
			}
			if (wSum[i] != 0.0) {
				for (int m = 0; m
< dim; m++) {
					change +=
std::pow(xnew[m] / wSum[i] - x[m][i], 2.0);
					magnitude
+= std::pow(x[m][i], 2.0);
					x[m][i] =
(xnew[m] / wSum[i]);
				}
			}
		}
		change = std::sqrt(change /
magnitude);
		std::chrono::duration<double>
elapsed_seconds = std::chrono::steady_clock::now()
- start;
		long timediff =
elapsed_seconds.count() / 1000000L;

		if ((timeout > 0) && (timediff >
timeout)) {
			return std::to_string(c +
1) + std::string(" iterations, ") +

	std::to_string(timediff) + std::string("
```

```
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
                if ((threshold > 0) && (change <
eps)) {
                        return std::to_string(c +
1) + std::string(" iterations, ") +

        std::to_string(timediff) + std::string("
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
                if ((iter > 0) && (c >= iter - 1))
{
                        report = std::to_string(c
+ 1) + std::string(" iterations, ") +

        std::to_string(timediff) + std::string("
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
        }
        return report;
}

std::string
SMACOF::majorize(std::vector<std::vector<double>>
&x, std::vector<std::vector<double>> &d,
        int iter, int threshold, int timeout)
{
        std::string report = "";
        int n = x[0].size();
        int k = d.size();
        int dim = x.size();

        double eps = std::pow(10.0, -threshold);
        auto start =
std::chrono::steady_clock::now();
        if (iter == 0)
                iter = 10000000;
        for (int c = 0; c < iter; c++) {
                double change = 0.0;
                double magnitude = 0.0;
                for (int i = 0; i < n; i++) {
                        std::vector<double>
xnew(dim, 0.0);
                        for (int j = 0; j < k;
j++) {
                                double inv = 0.0;
                                for (int m = 0; m
< dim; m++) {
                                        inv +=
std::pow(x[m][i] - x[m][j], 2.0);
                                }
                                if (inv != 0.0)
inv = std::pow(inv, -0.5);
                                for (int m = 0; m
< dim; m++) {
                                        xnew[m] +=
(x[m][j] +

        d[j][i] * (x[m][i] - x[m][j]) * inv);
                                }
                        }
                        for (int m = 0; m < dim;
m++) {
                                change +=
std::pow(xnew[m] / n - x[m][i], 2.0);
                                magnitude +=
std::pow(x[m][i], 2.0);
                                x[m][i] = (xnew[m]
/ n);
                        }
                }
                std::chrono::duration<double>
elapsed_seconds = std::chrono::steady_clock::now()
- start;
                long timediff =
elapsed_seconds.count() / 1000000L;

                if ((timeout > 0) && (timediff >
timeout)) {
                        return std::to_string(c +
1) + std::string(" iterations, ") +

        std::to_string(timediff) + std::string("
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
                if ((threshold > 0) && (change <
eps)) {
                        return std::to_string(c +
1) + std::string(" iterations, ") +

        std::to_string(timediff) + std::string("
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
                if ((iter > 0) && (c >= iter - 1))
{
                        report = std::to_string(c
+ 1) + std::string(" iterations, ") +

        std::to_string(timediff) + std::string("
milliseconds, ") + std::to_string(change) +
std::string(" relative change");
                }
        }
        return report;
}

void
SMACOF::majorize(std::vector<std::vector<double>>
&x, std::vector<std::vector<double>> &d,
        std::vector<std::vector<double>> &w, int
iter)
{
        int n = x[0].size();
        int dim = x.size();
        std::vector<double> wSum(n, 0.0);
        for (int i = 0; i < n; i++) for (int j =
0; j < n; j++) wSum[i] += w[i][j];
        for (int c = 0; c < iter; c++)
                for (int i = 0; i < n; i++) {
                        std::vector<double>
xnew(dim, 0.0);
                        for (int j = 0; j < n;
j++) {
                                double inv = 0.0;
                                for (int k = 0; k
< dim; k++)
                                        inv +=
std::pow(x[k][i] - x[k][j], 2.0);
                                if (inv != 0.0)
                                        inv =
std::pow(inv, -0.5);
                                for (int k = 0; k
< dim; k++)
                                        xnew[k] +=
w[i][j] * (x[k][j] + d[i][j] * (x[k][i] - x[k][j])
* inv);
                        }
                        if (wSum[i] != 0.0)
                                for (int k = 0; k
< dim; k++)
                                        x[k][i] =
(xnew[k] / wSum[i]);
                }
}

double
SMACOF::stress(std::vector<std::vector<double>>
&d, std::vector<std::vector<double>> &w,
std::vector<std::vector<double>> &x) {
        double result = 0.0;
```

```
	int n = x[0].size();
	int k = d.size();
	int dim = x.size();

	for (int i = 0; i < k; i++) {
		for (int j = i + 1; j < n; j++) {
			double dist = 0.0;
			for (int m = 0; m < dim;
m++)
				dist +=
std::pow(x[m][i] - x[m][j], 2.0);
			result += w[i][j] *
std::pow(d[i][j] - std::sqrt(dist), 2.0);
		}
	}
	return result;
}

double
SMACOF::stress(std::vector<std::vector<double>>
&d, std::vector<std::vector<double>> &x) {
	double result = 0.0;
	int n = x[0].size();
	int k = d.size();
	int dim = x.size();

	for (int i = 0; i < k; i++) {
		for (int j = i + 1; j < n; j++) {
			double dist = 0.0;
			for (int m = 0; m < dim;
m++)
				dist +=
std::pow(x[m][i] - x[m][j], 2.0);
			result += std::pow(d[i][j]
- std::sqrt(dist), 2.0);
		}
	}
	return result;
}

double
SMACOF::normalizedStress(std::vector<std::vector<d
ouble>> &d, std::vector<std::vector<double>> &w,
	std::vector<std::vector<double>> &x) {
	double result = 0.0;
	int n = x[0].size();
	int k = d.size();
	int dim = x.size();

	double sum = 0.0;
	for (int i = 0; i < k; i++) {
		for (int j = i + 1; j < n; j++) {
			double dist = 0.0;
			for (int m = 0; m < dim;
m++)
				dist +=
std::pow(x[m][i] - x[m][j], 2.0);
			result += w[i][j] *
std::pow(d[i][j] - std::sqrt(dist), 2.0);
			sum += w[i][j] *
std::pow(d[i][j], 2.0);
		}
	}
	return result / sum;
}

double
SMACOF::normalizedStress(std::vector<std::vector<d
ouble>> &d, std::vector<std::vector<double>> &x) {
	double result = 0.0;
	int n = x[0].size();
	int k = d.size();
	int dim = x.size();

	double sum = 0.0;
	for (int i = 0; i < k; i++) {
		for (int j = i + 1; j < n; j++) {
			double dist = 0.0;
			for (int m = 0; m < dim;
m++)
				dist +=
std::pow(x[m][i] - x[m][j], 2.0);
			result += std::pow(d[i][j]
- std::sqrt(dist), 2.0);
			sum += std::pow(d[i][j],
2.0);
		}
	}
	return result / sum;
}
```

Source.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <iterator>
#include <sstream>

#include "MDS.h"

std::vector<std::vector<double>>
getDistances(std::vector<std::vector<double>>
points)
{
	std::vector<std::vector<double>>
distances(points.size(),
std::vector<double>(points.size()));
	for (int i = 0; i < points.size(); i++)
	{
		//std::cout << "i:" << i <<
std::endl;
		for (int j = i + 1; j <
points.size(); j++)
		{
			//std::cout << "j:" << j
<< std::endl;

			double temp = 0;
			for (int k = 0; k <
points[j].size(); k++)
			{
				//std::cout <<
"k:" << k << std::endl;

				double t =
points[i][k] - points[j][k];
				temp +=
std::pow(points[i][k] - points[j][k], 2);
			}
			distances[i][j] =
std::sqrt(temp);
			distances[j][i] =
std::sqrt(temp);
		}
	}

	return distances;
}

int funcRows(size_t dim)
{
	std::vector<std::vector<double>> points;

	std::ifstream ifile("NPLines.txt");

	if (ifile.fail()) {

		std::cout << "txt file doesn't
exist" << std::endl;
		std::cin.get();

		return 1;
	}
	std::string line;
```

```
	size_t index = 0;
	while (!ifile.eof())
	{

		std::getline(ifile, line);

		std::istringstream iss(line);
		std::vector<std::string>
res((std::istream_iterator<std::string>(iss)),

	std::istream_iterator<std::string>());

		if (res.empty())
			continue;


	points.push_back(std::vector<double>(res.s
ize()));
		std::cout << index << " " <<
res.size() << std::endl;

		for (size_t i = 0; i < res.size();
i++)
		{
			points[index][i] =
std::stod(res[i]);
		}
		++index;
	}
	ifile.close();

	auto distances = getDistances(points);

	auto points2d =
MDS::distanceScaling(distances, dim);

	std::ofstream ofile("5d.txt");

	for (size_t i = 0; i < points2d[0].size();
i++)
	{
		for (size_t j = 0; j <
points2d.size(); j++)
		{
			ofile << points2d[j][i] <<
" ";
		}
		ofile << "\n";
	}

	ofile.close();

	return 0;
}

int funcColumns()
{
	std::vector<std::vector<double>> points;

	std::ifstream ifile("gordon-
2002_database.txt");

	if (ifile.fail()) {

		std::cout << "txt file doesn't
exist" << std::endl;
		std::cin.get();

		return 1;
	}
	std::string line;
	size_t index = 0;

	std::cout << "Reading file..." <<
std::endl;

	//
	std::getline(ifile, line);

	std::istringstream iss(line);
	std::vector<std::string>
res((std::istream_iterator<std::string>(iss)),

	std::istream_iterator<std::string>());

	if (res.empty()) {
		std::cout << "txt file is empty"
<< std::endl;
		std::cin.get();

		return 2;
	}
	points.assign(res.size() - 1,
std::vector<double>());

	for (size_t i = 1; i < res.size(); i++)
	{
		points[i -
1].push_back(std::stod(res[i]));
	}
	//
	while (!ifile.eof())
	{
		std::getline(ifile, line);

		std::istringstream iss(line);
		std::vector<std::string>
res((std::istream_iterator<std::string>(iss)),

	std::istream_iterator<std::string>());

		if (res.empty())
			continue;

		++index;

		//std::cout << index << " " <<
res.size() << std::endl;

		for (size_t i = 1; i < res.size();
i++)
		{
			points[i -
1].push_back(std::stod(res[i]));
		}
	}
	ifile.close();

	std::cout << "Computing distances..." <<
std::endl;

	auto distances = getDistances(points);

	std::cout << "Distances scaling..." <<
std::endl;

	auto points2d =
MDS::distanceScaling(distances, 2);

	std::ofstream ofile("2d.txt");

	for (size_t j = 0; j < points2d[0].size();
j++)
	{

		ofile << points2d[0][j] << " " <<
points2d[1][j] << "\n";
	}

	ofile.close();

	return 0;
}

int main()
{
```

```
        if (int code = funcRows(5) != 0)
                return code;

        std::cout << "Done" << std::endl;
        std::cin.get();
        return 0;
}
```

MDS_QT

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVector>

namespace Ui {
class MainWindow;
}

struct Point2d
{
    double x;
    double y;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

    void calculateDistance();
    QVector<QVector<double>>
GetDistances(QList<QString> lines);
    void drawPoints();

private slots:
    void on_pushButton_clicked();
    void on_mouse_clicked(QMouseEvent*);
private:
    Ui::MainWindow *ui;

    QVector<Point2d> lpoints;
    QVector<Point2d> points;
    bool isDrawing = false;
};

#endif // MAINWINDOW_H
```

MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "smacof.h"
#include "data.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->customplot->xAxis->setRange(-1, 2);
    ui->customplot->yAxis->setRange(-1, 2);

    ui->customplot-
>setInteraction(QCP::iRangeDrag, true);
    ui->customplot-
>setInteraction(QCP::iRangeZoom, true);

    ui->label->setText(QString::number(ui-
>customplot->xAxis->range().upper));
    ui->label_2->setText(QString::number(ui-
>customplot->xAxis->range().lower));

    //calculateDistance();
    drawPoints();

    connect(ui->customplot,
SIGNAL(mouseRelease(QMouseEvent*)), this,
SLOT(on_mouse_clicked(QMouseEvent*)));
}

void MainWindow::drawPoints()
{
    int dim = 2;

    QVector<QVector<double>> input; //=
GetValues(lines);
    QVector<QString> ids; //= GetValues(lines);

    QFile file("2d.txt");

    if(!file.open(QIODevice::ReadOnly))
    {
        qDebug() << "error opening file: " <<
file.error();
        return;
    }

    QTextStream in(&file);
    while(!in.atEnd())
    {

        QString line = in.readLine();
        QStringList slist = line.split(' ');
        QVector<double> tempData;
        ids.push_back(slist.at(0));
////////////////////////////////IDS
        for(size_t
i=1;i<slist.size();i++)//////////////////////////I
DS
        {

tempData.push_back(slist.at(i).toDouble());
            qDebug()<<slist.at(i)<<'\n';
        }

        input.push_back(tempData);
        //  qDebug()<<input.back().size()<<'\n';
    }

    file.close();

    QVector<double> xcoord;
    QVector<double> ycoord;

    QBrush qb[9]{Qt::green, Qt::blue, Qt::black,
Qt::red, Qt::yellow, Qt::darkCyan, Qt::darkGray,
Qt::darkMagenta, Qt::darkBlue};
////////////////color
    int t = -1;

    // qDebug()<<xycoord.size()<<'\n';

    for(size_t i=0;i<input.size();i++)
    {
        xcoord.push_back(input[i][0]);
        ycoord.push_back(input[i][1]);
        qDebug()<<input[i][0] << ' ' <<
input[i][1] << '\n';

        QCPGraph *graph =  ui->customplot-
>addGraph();
        // if(i > 30)
```

```
        //   t=1;
        ++t;
        if(t==4)
            t=0;

        graph-
>setScatterStyle(QCPScatterStyle(QCPScatterStyle::
ssCircle, QPen(qb[t], 1), qb[t], 4));
        graph->setData(xcoord, ycoord);
        //
        if(1){
        QCPItemText *caption = new QCPItemText(ui-
>customplot);
        //caption->setPen(Qt::SolidLine);  // draw
the box round the text
        caption->setFont(QFont("Sans", 6,
QFont::Bold));
        caption->setText(ids.at(i));
        caption-
>setTextAlignment(Qt::AlignCenter);    //centre
text within its rectangle
        caption-
>setPositionAlignment(Qt::AlignHCenter |
Qt::AlignTop);    // use the centre top of the
rectangle to position it
        caption->position-
>setType(QCPItemPosition::ptPlotCoords);
        caption->position-
>setCoords(input[i][0],input[i][1]);
        caption->setClipToAxisRect(false);
        //
        }

        xcoord.clear();
        ycoord.clear();
    }

    ui->customplot->replot();
}

MainWindow::~MainWindow()
{
    QFile file("lpoints.txt");
    if(!file.open(QIODevice::WriteOnly |
QIODevice::Text))
    {
        qDebug() << "error opening file: " <<
file.error();
        return;
    }
    QTextStream out(&file);
    for(size_t i=0;i<points.size();i++)
    {
        qDebug()<< i<<' '<<points[i].x << ' '<<
points[i].y<<'\n';
        out<< points[i].x << ' '  << points[i].y
<< '\n';
    }
    delete ui;
}

void MainWindow::on_mouse_clicked(QMouseEvent*
mouseEvent)
{
    if(isDrawing == false)
        return;

    QPoint point = mouseEvent->pos();

    double newX = ui->customplot->xAxis-
>pixelToCoord(point.x());
    double newY = ui->customplot->yAxis-
>pixelToCoord(point.y());

    QCPGraph *graph =  ui->customplot->addGraph();

    QVector<double> x = {newX};
    QVector<double> y = {newY};
    graph-
>setScatterStyle(QCPScatterStyle(QCPScatterStyle::
ssCircle, QPen(QColor(255, 200, 0), 1),
QColor(255, 200, 0), 4));
    graph->setData(x, y);

    lpoints.push_back({newX, newY});

    size_t s = lpoints.size();

    if(s > 1)
    {
        QCPItemLine *line = new QCPItemLine(ui-
>customplot);

        line->start->setCoords(lpoints[s - 2].x,
lpoints[s - 2].y);
        line->end->setCoords(lpoints[s - 1].x,
lpoints[s - 1].y);
        line->setPen(QPen(QColor(255, 200, 0)));
    }

    ui->customplot->replot();
}

void MainWindow::on_pushButton_clicked()
{

    ui->label->setText(QString::number(ui-
>customplot->xAxis->range().upper) + QString(" ")
+ QString::number(ui->customplot->xAxis-
>range().lower));
    ui->label_2->setText(QString::number(ui-
>customplot->yAxis->range().upper) + QString(" ")
+ QString::number(ui->customplot->yAxis-
>range().lower));


    isDrawing = !isDrawing;

    if(isDrawing)
        ui->pushButton->setText(QString("Завершити
виділення"));
    else
        ui->pushButton->setText(QString("Виділити
область"));

    for(int i = 0; i < lpoints.size() - 1;i++)
        points.push_back(lpoints[i]);

    lpoints.clear();
}
```

NPoints

Source.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>
#include <iterator>

struct Line2f
{
        struct Point2f
        {
                float x;
                float y;
        };

        Point2f p1; //start point
        Point2f p2; // end point
        std::vector<Point2f> points;
```

```
	Line2f(float x1, float y1, float x2, float y2) :p1{ x1, y1 }, p2{ x2, y2 } {}

	float getLenght()const
	{
		return std::sqrtf(std::powf(p2.x - p1.x, 2.f) + std::powf(p2.y - p1.y, 2.f));
	}

	void addNewPoint(Point2f newPoint)
	{
		points.push_back(newPoint);
	}
};

int main()
{
	std::ifstream ifile("lines.txt");
	/*
	lines.txt
	line1X1 line1Y1 line1X2 line1Y2
	...
	*/

	if (ifile.fail()) {

		std::cout << "lines.txt file doesn't exist" << std::endl;
		std::cin.get();

		return 1;
	}

	std::vector<Line2f> lines;
	std::string line;

	while (!ifile.eof())
	{

		std::getline(ifile, line);

		std::istringstream iss(line);
		std::vector<std::string> res((std::istream_iterator<std::string>(iss)),

	std::istream_iterator<std::string>());

		if (res.size() != 4)
		{
			std::cout << "Line number " << lines.size() + 1 << " doesn't have 4 values" << std::endl;
			continue;
		}


	lines.emplace_back(std::stof(res[0]), std::stof(res[1]), std::stof(res[2]), std::stof(res[3]));
	}

	ifile.close();

	size_t pointsc = 3;

	for (size_t i = 0; i < lines.size(); i++)
	{
		float ilineLen = lines[i].getLenght();
		float dt = ilineLen / (pointsc + 1);

		for (size_t j = 1; j <=pointsc; j++)
		{
			float t = dt * j / ilineLen;
			float newX = (1.f - t) * lines[i].p1.x + t * lines[i].p2.x;
			float newY = (1.f - t) * lines[i].p1.y + t * lines[i].p2.y;

			lines[i].addNewPoint({ newX, newY });
		}
	}

	std::ofstream ofile("NPLines.txt");

	for (size_t i = 0; i < lines.size(); i++)
	{
		ofile << lines[i].p1.x << ' ' << lines[i].p1.y<< ' ';
		for (size_t j = 0; j < lines[i].points.size(); j++)
		{
			ofile << lines[i].points[j].x << ' ' << lines[i].points[j].y <<' ';
		}
		ofile << lines[i].p2.x << ' ' << lines[i].p2.y << '\n';
	}
	ofile.close();


	std::cout << "+" << std::endl;

	std::cin.get();
	return 0;
}
```

Dist

Source.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <iterator>
#include <cassert>

using doubleMatrix = std::vector<std::vector<double>>;

doubleMatrix readDoubleMatrixFromFile(std::string filename)
{
	doubleMatrix tempmatrix;

	std::ifstream ifile(filename);

	if (ifile.fail()) {

		std::cout << "txt file doesn't exist" << std::endl;
		std::cin.get();

		//return 1;
	}

	std::string line;
	size_t index = 0;
	while (!ifile.eof())
	{
		std::getline(ifile, line);

		std::istringstream iss(line);
		std::vector<std::string> res((std::istream_iterator<std::string>(iss)),
```

```
        std::istream_iterator<std::string>());

                if (res.empty())
                        continue;


        tempmatrix.push_back(std::vector<double>(r
es.size()));
                std::cout << index << " " <<
res.size() << std::endl;

                for (size_t i = 0; i < res.size();
i++)
                {
                        tempmatrix[index][i] =
std::stod(res[i]);
                }
                ++index;
        }
        ifile.close();

        return tempmatrix;
}

static doubleMatrix getDistances(doubleMatrix&
points)
{
        doubleMatrix distances(points.size(),
std::vector<double>(points.size()));
        for (int i = 0; i < points.size(); i++)
        {
                //std::cout << "i:" << i <<
std::endl;
                for (int j = i + 1; j <
points.size(); j++)
                {
                        //std::cout << "j:" << j
<< std::endl;

                        double temp = 0;
                        for (int k = 0; k <
points[j].size(); k++)
                        {
                                //std::cout <<
"k:" << k << std::endl;

                                double t =
points[i][k] - points[j][k];
                                temp +=
std::pow(points[i][k] - points[j][k], 2);
                        }
                        distances[i][j] =
std::sqrt(temp);
                        distances[j][i] =
std::sqrt(temp);
                }
        }

        std::ofstream ofile("distances.txt");

        for (size_t i = 0; i < distances.size();
i++)
        {
                for (size_t j = 0; j <
distances[i].size(); j++)
                {
                        ofile << distances[i][j]
<< ' ';
                }
                ofile << '\n';
        }

        ofile.close();

        std::cout << "distance matrix has written"
<< std::endl;
        return distances;
}

static doubleMatrix compSMatrix(doubleMatrix &
fmatrix, doubleMatrix & smatrix)
{
        size_t len = fmatrix.size();
        assert(len == smatrix.size());
        double tempval = 0.0;
        doubleMatrix resmatrix(len,
std::vector<double>(len, 0.f));

        for (size_t i = 0; i < len; i++)
        {
                for (size_t j = 0; j < len; j++)
                {
                        resmatrix[i][j] =
std::fabsf(fmatrix[i][j] - smatrix[i][j]);
                }
        }

        return resmatrix;
}

int main()
{
        /*doubleMatrix smatrix =
compSMatrix(readDoubleMatrixFromFile("2Ddistances.
txt"),
readDoubleMatrixFromFile("all5Ddistances.txt"));

        std::ofstream ofile("distances.txt");

        for (size_t i = 0; i < smatrix.size();
i++)
        {
                for (size_t j = 0; j <
smatrix[i].size(); j++)
                {
                        ofile << smatrix[i][j] <<
' ';
                }
                ofile << '\n';
        }

        ofile.close();

        std::cout << "distance matrix has written"
<< std::endl;*/
    doubleMatrix distances =
getDistances(readDoubleMatrixFromFile("test.txt"))
;


        std::cin.get();
        return 0;
}
```

FaceFilter

FaceFilter.h

```
#ifndef _FACE_FILTER_H_
#define _FACE_FILTER_H_

#include <vector>

struct Point2f
{
        float x;
        float y;
};

class FaceFilter
{
```

```
        static const float admissibleValue;

public:

        static bool filter(const
std::vector<Point2f> & points);

};

#endif // !_FACE_FILTER_H_
```

FaceFilter.cpp

```
#include "FaceFilter.h"

bool FaceFilter::filter(const std::vector<Point2f>
& points)
{
        float XL = (points[45].x + points[42].x) /
2.0;
        float YL = (points[45].y + points[42].y) /
2.0;
        float XR = (points[39].x + points[36].x) /
2.0;
        float YR = (points[39].y + points[36].y) /
2.0;

        float X0 = (XL + XR) / 2.0;
        float Y0 = (YL + YR) / 2.0;

        float DX = XR - XL;
        float DY = YR - YL;

        float L = std::sqrtf(std::powf(DX, 2) +
std::powf(DY, 2));

        float sin_AL = DY / L;
        float cos_AL = DX / L;

        float X_User = 0;
        float Y_User = 0;

        auto getPointXY = [&X0, &Y0, L, &sin_AL,
&cos_AL, &X_User, &Y_User](float x1, float y1)
        {
                float X_User_0 = (x1 - X0) / L;
                float Y_User_0 = -(y1 - Y0) / L;
                X_User = X_User_0 * cos_AL -
Y_User_0 * sin_AL;
                Y_User = X_User_0 * sin_AL -
Y_User_0 * cos_AL;
        };

        float x1, y1, x2, y2;
        //1 filter
        getPointXY(points[27].x, points[27].y);
        x1 = X_User; // 27x
        y1 = Y_User;  // 27y
        getPointXY(points[30].x, points[30].y);
        x2 = X_User;   //30x
        y2 = Y_User;  //30y

        float filter1 = x1 - x2;
        if (filter1 >= admissibleValue)
                return false;

        //2 filter
        getPointXY(points[48].x, points[48].y);
        x1 = X_User;   // 48x
        y1 = Y_User;  // 48y
        getPointXY(points[54].x, points[54].y);
        x2 = X_User;   //54x
        y2 = Y_User;  //54y

        float filter2 = y1 - y2;
        if (filter2 >= admissibleValue)
                return false;

        //3 filter
        float filter3 = x1 + x2;
        if (filter3 >= admissibleValue)
                return false;

        //4 filter
        getPointXY(points[36].x, points[36].y);
        x1 = X_User;   // 48x
        y1 = Y_User;  // 48y
        getPointXY(points[39].x, points[39].y);
        x2 = X_User;   //54x
        y2 = Y_User;  //54y

        float filter4 = y2 - y1;
        if (y2 - y1 >= admissibleValue)
                return false;

        return true;
}

const float FaceFilter::admissibleValue = 0.2;
```

er_qt

Detector.h

```
#pragma once

#include <opencv2\opencv.hpp>
#include <opencv2\face.hpp>

#include "FaceData.h"

class Detector
{
        cv::CascadeClassifier haarCascade;
        cv::Ptr<cv::face::FacemarkLBF>
facemarkDetector;

        const float Detector::EYEMAX = 0.45f;
        const float Detector::MOUTHMAX = 0.96f;
        const float Detector::EYEBROWMAX = 0.4f;

public:
        Detector();

        FaceData getFaceData(const cv::Mat &
inputMat);

        static void setFaceMarks(cv::Mat
&inputMat, const std::vector<cv::Point2f>
&faceLandmarks);
};
```

Detector.cpp

```
#include "Detector.h"

Detector::Detector() :
haarCascade("haarcascade_frontalface_alt2.xml")
{
        facemarkDetector =
cv::face::FacemarkLBF::create();
        facemarkDetector-
>loadModel("lbfmodel.yaml");
}
```

```
static float distance(cv::Point2f a, cv::Point2f
b)
{
	return std::sqrtf(std::powf(a.x - b.x, 2)
+ std::powf(a.y - b.y, 2));
}

FaceData Detector::getFaceData(const cv::Mat &
inputMat)
{
	std::vector<cv::Rect> faces;
	std::vector<std::vector<cv::Point2f>>faces
Landmarks;

	cv::Mat gray;

	cvtColor(inputMat, gray,
cv::COLOR_BGR2GRAY);
	haarCascade.detectMultiScale(gray, faces);

	if (faces.empty())
		return FaceData();

	facemarkDetector->fit(gray, faces,
facesLandmarks);

	std::vector<float> fvector;

	for (size_t i = 0; i <
facesLandmarks.size(); i++)
	{
		//Eye
		cv::Point2f a =
(facesLandmarks[i][37] + facesLandmarks[i][38]) /
2;
		cv::Point2f b =
(facesLandmarks[i][40] + facesLandmarks[i][41]) /
2;

		float fd = distance(a, b);
		float sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][39]);

		float res = fd / sd / EYEMAX;

		float eyeVal = 0.f;
		if (res < 0.55f)
			eyeVal = (res - 0.f) *
(0.2f - 0.0f) / (0.55f - 0.0f) + 0.0f;
		else  if (res < 0.75f)
			eyeVal = (res - 0.55f) *
(0.6f - 0.4f) / (0.75f - 0.55f) + 0.4f;
		else
			eyeVal = (res - 0.75f) *
(1.0f - 0.8f) / (1.0f - 0.75f) + 0.8f;

		if (eyeVal > 1.f)
			eyeVal = 1.f;

		//Eyebrow
		a = (facesLandmarks[i][36] +
facesLandmarks[i][39]) / 2;

		fd = distance(a,
facesLandmarks[i][19]);
		sd =
distance(facesLandmarks[i][36],
facesLandmarks[i][45]);

		res = fd / sd / EYEBROWMAX;
		float eyeBrowVal = 0.f;

		if (res < 0.65f)
			eyeBrowVal = (res - 0.f) *
(0.2f - 0.0f) / (0.65f - 0.0f) + 0.0f;
		else  if (res < 0.75f)
			eyeBrowVal = (res - 0.65f)
* (0.6f - 0.4f) / (0.75f - 0.65f) + 0.4f;
		else
			eyeBrowVal = (res - 0.75f)
* (1.0f - 0.8f) / (1.0f - 0.75f) + 0.8f;

		if (eyeBrowVal > 1.f)
			eyeBrowVal = 1.f;

		//Lips

		fd =
distance(facesLandmarks[i][36],
facesLandmarks[i][45]);
		sd =
distance(facesLandmarks[i][48],
facesLandmarks[i][54]);

		res = sd / fd / MOUTHMAX;
		float lipsVal = 0.f;
		if (res < 0.65f)
			lipsVal = (res - 0.0f) *
(0.2f - 0.0f) / (0.65f - 0.f) + 0.0f;
		else  if (res < 0.75f)
			lipsVal = (res - 0.65f) *
(0.6f - 0.4f) / (0.75f - 0.65f) + 0.4f;
		else
			lipsVal = (res - 0.75f) *
(1.0f - 0.8f) / (1.f - 0.75f) + 0.8f;

		if (lipsVal > 1.f)
			lipsVal = 1.f;

		fvector.push_back(eyeBrowVal);
		fvector.push_back(eyeVal);
		fvector.push_back(lipsVal);
	}

	return FaceData(fvector, faces.front(),
facesLandmarks.front());
}

void Detector::setFaceMarks(cv::Mat &inputMat,
const std::vector<cv::Point2f> &faceLandmarks)
{
	if (faceLandmarks.empty())
		return;

	for (size_t j = 17; j <= 26; j++) //
eyebrows points
		cv::circle(inputMat,
faceLandmarks[j], 2, cv::Scalar(0, 0, 255), 2);

	for (size_t j = 36; j <= 47; j++) // eyes
points
		cv::circle(inputMat,
faceLandmarks[j], 2, cv::Scalar(0, 0, 255), 2);

	for (size_t j = 48; j <= 60; j++) // lips
points
		cv::circle(inputMat,
faceLandmarks[j], 2, cv::Scalar(0, 0, 255), 2);
}
```

Identificator.h

```
#pragma once

#include <vector>
#include <string>
#include <array>
#include <map>

struct Matrix3f
{
	std::array<std::array<float, 3>, 3>
matrix;
```

```
        Matrix3f(float arr[3][3]);
};

struct det4
{
        std::array<float, 4> values;

        det4(float values_[4]);
};

enum class EmonionState : size_t
{
        Happy,
        Sad,
        Anger,
        Fear,
        NoE
};

class Identificator
{
        std::vector<Matrix3f> matrices;
        std::vector<det4> dets;
        std::vector<std::vector<float> >
emotionsPatterns; // Happy Sad Anger Fear

        bool readMatrices();
        bool readEmotionsPatterns();
        bool checkPattern(const std::vector<float>
&ipattern, const std::vector<float>
&cpattern)const;

        inline EmonionState
getEmotionByIndex(size_t)const;

public:
        Identificator();

        static std::string
emotionStateToString(EmonionState);

        std::vector<EmonionState>
getState(std::vector<float>)const;
};
```

Identificator.cpp

```
#include "Identificator.h"

#include <fstream>
#include <string>
#include <iterator>
#include <sstream>

Matrix3f::Matrix3f(float arr[3][3])
{
        for (size_t i = 0; i < 3; i++)
                for (size_t j = 0; j < 3; j++)
                        matrix[i][j] = arr[i][j];
}

det4::det4(float values_[4])
{
        for (size_t i = 0; i < 4; i++)
                values[i] = values_[i];
}

static float calcDet(Matrix3f matrix, size_t
index)
{
        float det = matrix.matrix[0][0] *
matrix.matrix[1][1] * matrix.matrix[2][2];
        det += matrix.matrix[0][1] *
matrix.matrix[1][2] * matrix.matrix[2][0];
        det += matrix.matrix[0][2] *
matrix.matrix[1][0] * matrix.matrix[2][1];
        det -= matrix.matrix[0][2] *
matrix.matrix[1][1] * matrix.matrix[2][0];
        det -= matrix.matrix[0][1] *
matrix.matrix[1][0] * matrix.matrix[2][2];
        det -= matrix.matrix[0][0] *
matrix.matrix[1][2] * matrix.matrix[2][1];

        return std::powf(-1.f, 1 + index) * det;
}

Identificator::Identificator()
{
        readMatrices();
        readEmotionsPatterns();
}

bool Identificator::readMatrices()
{
        std::ifstream ifile("points.txt");

        if (ifile.fail()) {
                return false;
        }

        std::string line;

        while (!ifile.eof())
        {
                float tempmatrix[3][3];

                for (size_t i = 0; i < 3; i++)
                {
                        std::getline(ifile, line);
                        if (line.empty())
                                break;

                        std::istringstream
iss(line);
                        std::vector<std::string>
res((std::istream_iterator<std::string>(iss)),

        std::istream_iterator<std::string>());

                        if (res.size() != 3)
                        {
                                continue;
                        }

                        tempmatrix[i][0] =
std::stof(res[0]);
                        tempmatrix[i][1] =
std::stof(res[1]);
                        tempmatrix[i][2] =
std::stof(res[2]);

                }

                matrices.push_back(tempmatrix);
        }

        ifile.close();

        float dtm[4]{ 0.f, 0.f, 0.f, 0.f };

        for (size_t t = 0; t < matrices.size();
t++)
        {
                float localMatrix[3][3];

                size_t index = 0;

                localMatrix[0][2] = 1.f;
                localMatrix[1][2] = 1.f;
                localMatrix[2][2] = 1.f;
                // 2x 3x
                for (size_t i = 0; i < 3; i++)
                {
```

```
                        localMatrix[i][0] =
matrices[t].matrix[i][1];
                        localMatrix[i][0] =
matrices[t].matrix[i][1];
                        localMatrix[i][0] =
matrices[t].matrix[i][1];

                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                }

                dtm[0] = calcDet(localMatrix,
index + 1);
                // 1x 3x
                for (size_t i = 0; i < 3; i++)
                {
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];

                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                        localMatrix[i][1] =
matrices[t].matrix[i][2];
                }

                dtm[1] = calcDet(localMatrix,
index + 2);

                // 1x 2x
                for (size_t i = 0; i < 3; i++)
                {
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];

                        localMatrix[i][1] =
matrices[t].matrix[i][1];
                        localMatrix[i][1] =
matrices[t].matrix[i][1];
                        localMatrix[i][1] =
matrices[t].matrix[i][1];
                }

                dtm[2] = calcDet(localMatrix,
index + 3);

                // 1x 2x 3x
                for (size_t i = 0; i < 3; i++)
                {
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];
                        localMatrix[i][0] =
matrices[t].matrix[i][0];

                        localMatrix[i][1] =
matrices[t].matrix[i][1];
                        localMatrix[i][1] =
matrices[t].matrix[i][1];
                        localMatrix[i][1] =
matrices[t].matrix[i][1];

                        localMatrix[i][2] =
matrices[t].matrix[i][2];
                        localMatrix[i][2] =
matrices[t].matrix[i][2];
                        localMatrix[i][2] =
matrices[t].matrix[i][2];
                }


                dtm[3] = calcDet(localMatrix,
index + 4);

                dets.push_back(det4(dtm));
        }

        return true;
}

bool Identificator::readEmotionsPatterns()
{
        std::ifstream ifile("patterns.txt");

        if (ifile.fail()) {
                return false;
        }

        std::string line;
        size_t index = 0;
        while (!ifile.eof())
        {
                std::getline(ifile, line);
                if (line.empty())
                        continue;

                std::istringstream iss(line);
                std::vector<std::string>
res((std::istream_iterator<std::string>(iss)),
std::istream_iterator<std::string>());


        emotionsPatterns.push_back(std::vector<flo
at>());

                for (size_t i = 0; i < res.size();
i++)

        emotionsPatterns[index].push_back(std::sto
f(res[i]));

                ++index;
        }

        ifile.close();

        return true;
}

bool Identificator::checkPattern(const
std::vector<float> &ipattern, const
std::vector<float> &cpattern)const
{
        if (ipattern.size() != cpattern.size())
                return false;

        for (size_t i = 0; i < ipattern.size();
i++)
        {
                if (ipattern[i] < 0 != cpattern[i]
< 0) // same sign
                        return false;
        }

        return true;
}

std::vector<EmonionState>
Identificator::getState(std::vector<float>
inValues)const
{
        const size_t ls = 6;
```

```
	std::vector<float> inputPattern(ls, 0.f);

	std::vector<EmonionState> ems;

	size_t eIndex = 0;

	/*
	eIndex 0: Happy
	eIndex 1: Sad
	eIndex 2: Angry
	eIndex 3: Fear
	*/

	for (size_t t = 0; t < dets.size(); t += 
ls, eIndex++)
	{
		for (size_t i = t, j = 0; i < t + 
ls; i++, j++)
		{
			float value = inValues[0] 
* dets[i].values[0];
			value += inValues[1] * 
dets[i].values[1];
			value += inValues[2] * 
dets[i].values[2];
			value += 
dets[i].values[3];

			inputPattern[j] = value;
		}

		if (checkPattern(inputPattern, 
emotionsPatterns[eIndex]))
		{

	ems.push_back(getEmotionByIndex(eIndex));
		}
	}
	if (ems.empty())
		ems.push_back(EmonionState::NoE);

	return ems;
}

inline EmonionState
Identificator::getEmotionByIndex(size_t 
index)const
{
	return static_cast<EmonionState>(index);
}

std::string
Identificator::emotionStateToString(EmonionState 
emotion)
{
	switch (emotion)
	{
	case EmonionState::Happy:
		return "Happy";
	case EmonionState::Sad:
		return "Sad";
	case EmonionState::Anger:
		return "Anger";
	case EmonionState::Fear:
		return "Fear";
	case EmonionState::NoE:
		return "NoE";
	default:
		break;
	}
	return "";
}
```

FaceData.h

```
#pragma once

#include <vector>
#include <opencv2\opencv.hpp>

class FaceData
{
	std::vector<float> fvector; // face vector
	cv::Rect frect; // face rectangle
	std::vector<cv::Point2f> flandmarks; // 
face marks

public:
	FaceData() = default;
	FaceData(std::vector<float>, cv::Rect, 
std::vector<cv::Point2f>);

	std::vector<float> getFaceVector()const;
	cv::Rect getFaceRect()const;
	std::vector<cv::Point2f> 
getFaceLandmarks()const;
};
```

FaceData.cpp

```
#include "FaceData.h"

FaceData::FaceData(std::vector<float> fvector, 
cv::Rect frect, std::vector<cv::Point2f> 
flandmarks)
	:fvector(fvector), frect(frect), 
flandmarks(flandmarks)
{

}

std::vector<float> FaceData::getFaceVector()const
{
	return fvector;
}

cv::Rect FaceData::getFaceRect()const
{
	return frect;
}

std::vector<cv::Point2f> 
FaceData::getFaceLandmarks()const
{
	return flandmarks;
}
```

er_qt.h

```
#ifndef ER_QT_H
#define ER_QT_H

#include <QtWidgets/QMainWindow>
#include <QMessageBox>
#include <QTimer>

#include "ui_er_qt.h"

#include "Detector.h"
#include "Identificator.h"

class er_qt : public QMainWindow
{
	Q_OBJECT

public:
	er_qt(QWidget *parent = 0);
	~er_qt() = default;

	private slots:
	void on_pushButton_clicked();
```

```
        void on_checkBox_2_stateChanged();
        void updatePicture();
        void processPicture();

private:
        Ui::er_qtClass ui;
        QMessageBox qmessageBox;

        std::unique_ptr<QTimer> imageTimer;
        std::unique_ptr<QTimer> processTimer;

        cv::Rect oldRect;
        std::vector<cv::Point2f> oldlandmarks;

        cv::VideoCapture cap;

        Detector detector;
        Identificator identificator;

        void processImage(cv::Mat & image);
};

#endif // ER_QT_H
```

er_qt.cpp

```
#include "er_qt.h"

#include <QFileDialog>

#include <opencv2\opencv.hpp>

er_qt::er_qt(QWidget *parent) :
QMainWindow(parent)
{
        imageTimer =
std::make_unique<QTimer>(this);
        processTimer =
std::make_unique<QTimer>(this);

        connect(imageTimer.get(),
SIGNAL(timeout()), this, SLOT(updatePicture()));
        connect(processTimer.get(),
SIGNAL(timeout()), this, SLOT(processPicture()));

        ui.setupUi(this);
}

void er_qt::updatePicture()
{
        cv::Mat capImage;

        cap >> capImage;

        cv::rectangle(capImage, oldRect,
cv::Scalar(0, 0, 255), 2);

        if (ui.checkBox->isChecked())
                Detector::setFaceMarks(capImage,
oldlandmarks);

        QImage qimage =
QImage((uchar*)capImage.data, capImage.cols,
capImage.rows, capImage.step,
QImage::Format_RGB888).rgbSwapped();

        ui.label-
>setPixmap(QPixmap::fromImage(qimage).scaled(ui.la
bel->width(), ui.label->height(),
Qt::KeepAspectRatio));
}

void er_qt::processPicture()
{
        cv::Mat capImage;

        cap >> capImage;

        processImage(capImage);
}

void er_qt::on_pushButton_clicked()
{
        QString fileName =
QFileDialog::getOpenFileName(this, tr("Open
Image"), "C://", tr("Image Files (*.png *.jpg
*.bmp)"));

        if (fileName.isEmpty())
                return;

        cv::Mat fImage =
cv::imread(fileName.toStdString().c_str(),
cv::IMREAD_COLOR);

        processImage(fImage);
}

void er_qt::on_checkBox_2_stateChanged()
{
        if (ui.checkBox_2->isChecked())
        {
                imageTimer->start(20);
                processTimer->start(500);
                cap.open(0);
                ui.pushButton->setEnabled(false);
        }
        else
        {
                imageTimer->stop();
                processTimer->stop();
                cap.release();
                ui.pushButton->setEnabled(true);
                ui.label->clear();
                ui.label_2->clear();
        }
}

void er_qt::processImage(cv::Mat & image)
{
        QLabel * imageLabel = ui.label;

        FaceData fdata =
detector.getFaceData(image);

        std::vector<float> fvector =
fdata.getFaceVector();

        if (fvector.empty())
                return;

        std::vector<EmonionState> estates =
identificator.getState(fvector);

        QString result;
        for (size_t i = 0; i < estates.size();
i++)
        {
                result +=
QString(Identificator::emotionStateToString(estate
s[i]).c_str()) + QString(" / ");
        }

        ui.label_2->setText(result);

        auto flandmarks =
fdata.getFaceLandmarks();
        oldlandmarks = flandmarks;
        if (ui.checkBox->isChecked())
                detector.setFaceMarks(image,
flandmarks);

        cv::Rect frect = fdata.getFaceRect();
        oldRect = frect;
        cv::rectangle(image, frect, cv::Scalar(0,
0, 255), 2);
```

```
        QImage qimage = QImage((uchar*)image.data,
image.cols, image.rows, image.step,
QImage::Format_RGB888).rgbSwapped();

        imageLabel-
>setPixmap(QPixmap::fromImage(qimage).scaled(image
Label->width(), imageLabel->height(),
Qt::KeepAspectRatio));
}
```

main.cpp

```
#include "er_qt.h"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
        QApplication a(argc, argv);
        a.setWindowIcon(QIcon("icon.png"));

        er_qt w;
        w.show();
        return a.exec();
}
```

Face image transformations for correct recognition problems solving

Mykola Svirnevskyi
Department of Computer Science and Information Technology
Khmelnitskyi National University
Khmelnitskyi, Ukraine
mykola.svirnevskyi@gmail.com

Iurii Krak
Department. of Theoretical Cybernetics,
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine,
krak@univ.kiev.ua

Olexander Barmak
Department of Computer Science and Information Technology
Khmelnitskyi National University
Khmelnitskyi, Ukraine
alexander.barmak@gmail.com

Sergiy Ivaschenko
Department of Computer Science and Information Technology
Khmelnitskyi National University
Khmelnitskyi, Ukraine
serguy1998ukraine@gmail.com

***Abstract* – The main problem in face recognition is associated with changes in lighting, head position and facial expression. The article focuses on improving the performance of face recognition algorithms by solving this problem. An approach for face recognition with image conversion is proposed. For this approach, face alignment algorithms through 2D transformations and 3D reconstruction of anthropometric points of the face were developed. Testing of the approach showed that facial recognition quality improved.**

Keywords – identification, recognition, face, facemark, alignment, transformations, 3D reconstruction, image.

I. INTRODUCTION AND FORMULATION OF THE PROBLEM

Face recognition is one of the most promising methods of biometric contactless identification of a person. Most often it is used in video surveillance and access control systems.

The face recognition system can be described as a process of comparing faces caught in the camera lens with a database of previously saved face images.

There are a lot of face recognition algorithms today, including 2D and 3D recognition models [1]. Recognition of 2D images is one of the most popular technologies at the moment. This is due to the fact that the databases of identified individuals accumulated in the world are two-dimensional. In addition, the basic equipment already installed around the world is also for 2D recognition.

2D face recognition technology is based on flat two-dimensional images. In face recognition algorithms are used anthropometric landmarks on the face (facial landmark points, FLP), as well as images represented by a specific set of physical or mathematical features [2–9]. Proper use of anthropometric parameters on the face is especially important for recognizing the emotional state of the face in many applications [10-12].

Most recognition systems work by determining the distances between the feature points of the face. The main problem of such systems is that the points may not be displayed correctly, and the distances may be distorted depending on various factors [13, 14].

Objective factors – the position of the face relative to the camera (Fig. 1).

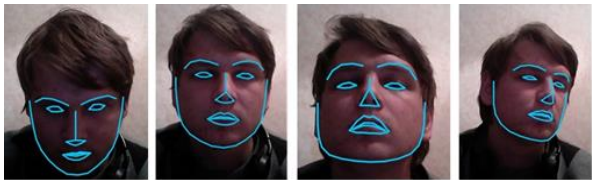


Fig. 1. Possible face position relative to the camera

Subjective factors – uneven or poor lighting, facial distortion due to emotions, squinting eyes, etc. In these cases, the points may be torn off the face (Fig. 2).

Fig. 2. Incorrect display of face points

It is proposed to solve this problem: 1) by aligning the position of the face and 2) by discarding the incorrect images remaining after that.

It should be noted that these methods are interconnected. With effective face alignment, the number of incorrect images can be reduced, since the incorrectness of images in this case is determined only by subjective factors (lighting, face distortion, etc.).

Face alignment is suggested by 2D transformations and 3D model restoration.

II. FACE RECOGNITION BASED ON THE 2D MODEL

We describe the proposed face recognition approach with face image alignment and discarding incorrect images.

In exploring the possibilities of the proposed approach (Fig. 3), the Viola-Jones method [15] was used to identify the face area in the image, and Active Shape Models (ASM) [16] for recognition.

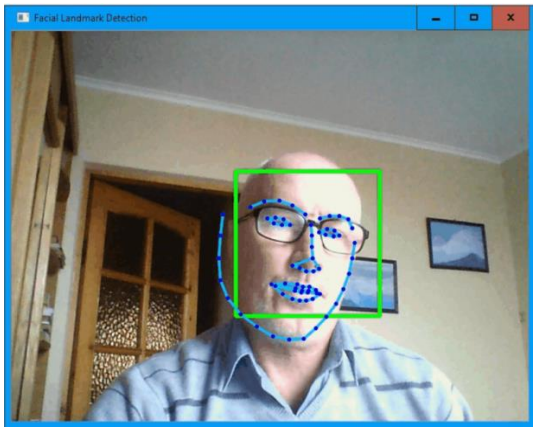


Fig. 3. Face and FLP identification

The main idea of ASM is to take into account the statistical relationships between the locations of the anthropometric points of the face.

On each face image, the points are numbered in the same order (Fig. 4).

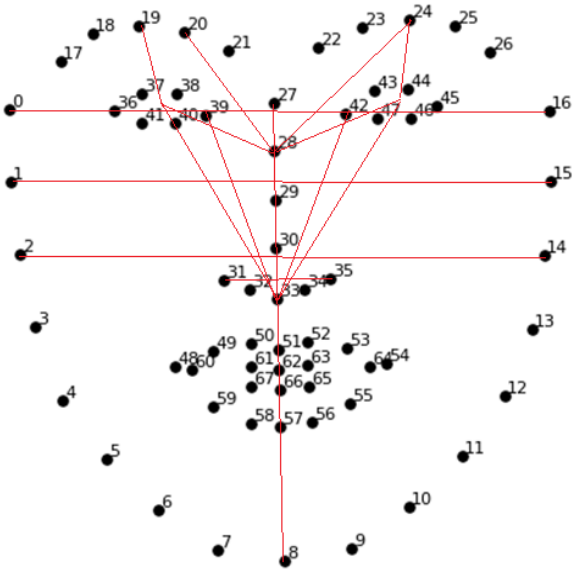


Fig. 4. Features for face recognition based on distances between anthropometric points of the face

According to their mutual arrangement, a comparison of persons is carried out. The distance between the points marked in Fig.4 was taken as features for recognition.

Features are normalized with respect to the distance between the centers of the eyes. The degree of similarity between the subject and each of the applicants is determined by the sum of the points scored on all grounds.

The coordinates of the face points are initially set in the frame coordinate system (FCS), which is attached to the upper left point of the window (Fig. 5). *Y*-axis down, *X*-axis right.

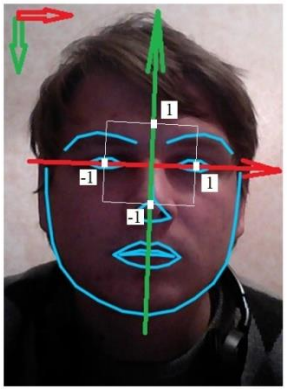


Fig. 5. The relative position of FCS and UCS

For the convenience of determining the distance between points, we use a user coordinate system (UCS), the *X* axis of which passes through the centers of the eyes, and the *Y* axis passes through a point in the middle between the centers of the eye (Fig.6). The UCS coordinates are normalized by dividing by the distance between the centers of the eyes.

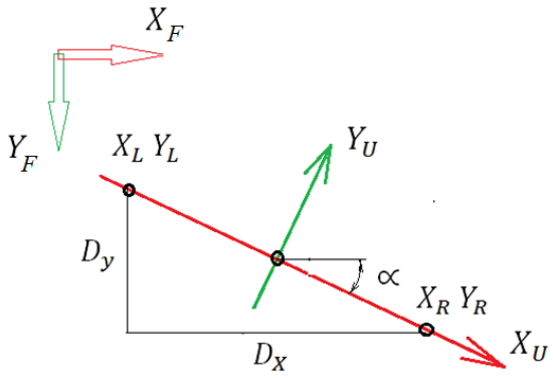


Fig. 6. Scheme for determining UCS relative to FCS

The sequence of transition from FCS coordinates to UCS coordinates:

1. The coordinates of the midpoints of the left and right eyes:

$$X_L = \frac{(X_{45}+X_{42})}{2};\quad Y_L = \frac{(X_{45}+X_{42})}{2};$$

$$X_R = \frac{X_{39}+X_{36}}{2};\quad Y_R = \frac{Y_{39}+Y_{36}}{2}.$$

2. UCS Origin:

$$X_0 = \frac{(X_L+X_R)}{2};\quad Y_R = \frac{Y_{39}+Y_{36}}{2}.$$

3. Distances between midpoints along the *X* and *Y* axes:

$$D_X = X_R - X_L;\quad D_y = Y_R - Y_L.$$

4. Actual distance between midpoints of the eyes:

$$L = \sqrt{(D_X^2 + D_Y^2)}\,.$$

5. UCS trigonometric angle functions:

$$\sin\alpha = \frac{D_Y}{L};\quad \cos\alpha = \frac{D_X}{L}.$$

6. The equations of transition from FCS coordinates to UCS coordinates (Fig.6):

$$X_U = \frac{2(X_F - X_0)}{L} \cos\propto + \frac{2(Y_F - Y_0}{L} \sin\propto;$$

$$Y_U = \frac{2(X_F - X_0)}{L} \sin\propto - \frac{2(Y_F - Y_0)}{L} \cos\propto.$$

We define the features by which incorrect images will be discarded. These features are formalized through the conditions of parallelism and perpendicularity of the lines that pass through the FLP, as well as the symmetry of the points (Fig.7).

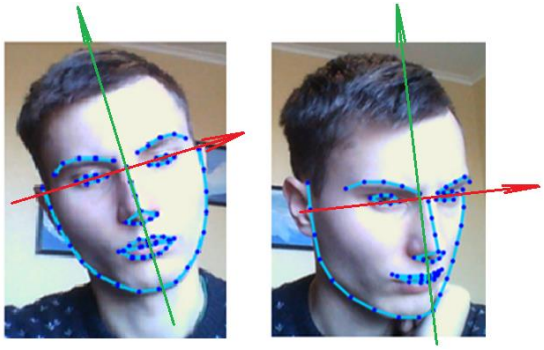


Fig. 7. Correct (left) and incorrect (right) images

For example, the image shown in the figure on the right (Fig. 7) is discarded provided that the *Y*-axis does not coincide with the nose line. In UCS, this condition can be formalized very simply: $X_{U27} = X_{U30} = \mathbf{0}$.

III. JUSTIFICATION OF 3D RECONSTRUCTION OPPORTUNITY

Obviously, the distances between the anthropometric points are projected with different distortions – depending on how the face is facing the camera (Fig.8).

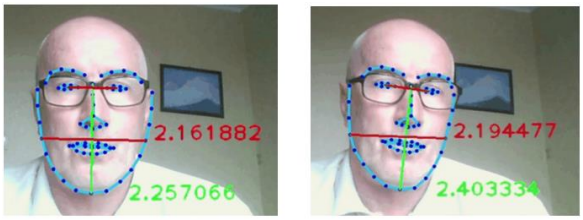


Fig. 8. Change of the distances when the face turns

Note that when turning the head, the horizontal distance changes less than the vertical. The fact is that the horizontal line is parallel to the line between the middle of the eyes, and the distortion of parallel lines is the same (a property of affine transformations). Therefore, the position of the face practically does not affect the change in the magnitude of this distance.

In fact, we are not dealing with an affine transformation (parallel projection), but with a projective transformation (central projection), in which parallelism is not preserved. But deviations from parallelism are insignificant if the object is far enough from the camera. Moreover, the differences in distortions of parallel segments are also small. Therefore, they can be neglected.

From the above reasoning, it follows that the distances between points of a face should be normalized by the distances between points on parallel lines. To normalize, you can select points on three mutually perpendicular (in 3D space) lines that define the projection of the spatial coordinate system attached to the face in the image.

IV. FACE RECOGNITION BASED ON 3D MODEL

Based on FLP, you can create a spatial coordinate system. Knowing the direction of the axes, as well as unit segments on them, it is possible to determine the normalized three-dimensional coordinates of any of the points. The characteristic points that determine the coordinate system are selected from the conditions of mutual perpendicularity of the coordinate axes, as well as the symmetry of the FLP pairs with respect to the *yz* plane (Fig.9).

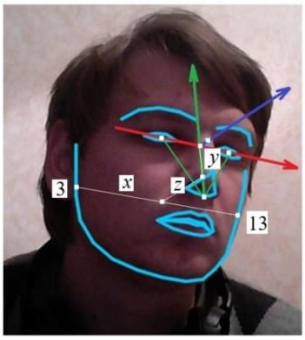


Fig. 9. Reconstruction of the spatial UCS

Fig. 9 shows a spatial UCS model whose axes pass through FLP:

- The axis *x* passes through the centers of the eyes (determined based on points 36, 39, 42 and 45).
- The *y*-axis passes through point 33 and the midpoint between the centers of the eyes.
- The *z*-axis passes through point 27 and the midpoint between the centers of the eyes.

Fig. 10 shows an example of how to determine the coordinates of symmetrically located FLPs 3 and 13. Connect the points with a line and define its middle. Through the middle, draw a line parallel to the *z*-axis until it intersects with the *y*-axis. Segments from point 3 to the origin show the *x* coordinate for point 3.

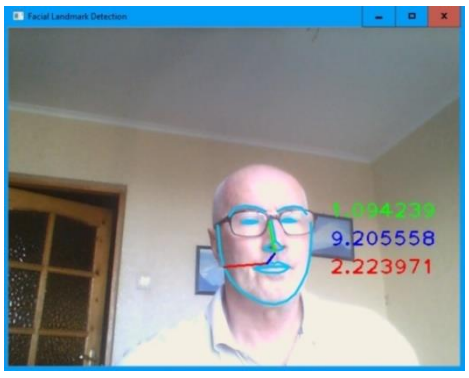


Fig. 10. An example of displaying 3D coordinates

The coordinates are normalized by the distances on the coordinate axes:

- between the middle of the eyes – for *x*;
- from the middle between the eyes to point 33 – for *y*;

- from the middle between the eyes to point 27 – for *z*.

Let us consider in more detail the algorithm for determining 3D coordinates. First of all, move the FCS origin to a point in the middle between the eyes (Fig. 11).

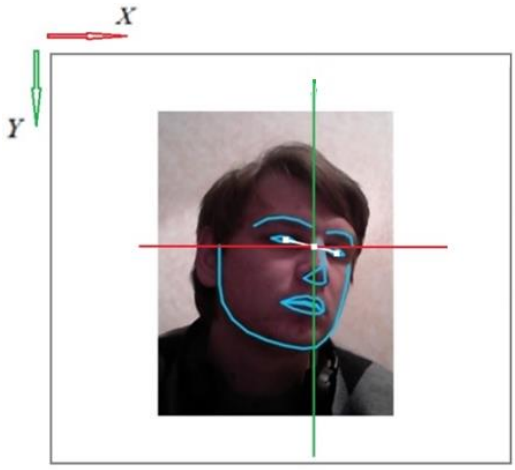


Fig. 11. Change of origin

The coordinates of this point are determined based on FLP 36, 39, 42 и 45 (Fig.4).

$$X_L = \frac{(X_{36}+X_{39})}{2};\quad Y_L = \frac{(Y_{36}+Y_{39})}{2};$$
$$X_R = \frac{(X_{42}+X_{45})}{2};\quad Y_R = \frac{(Y_{42}+Y_{45})}{2};$$
$$X_0 = \frac{(X_L+X_R)}{2};\quad Y_0 = \frac{(Y_L+Y_R)}{2}.$$

Relative to the new origin, we determine the direction of the projections of the axes of the spatial UCS through the angular coefficients of the line ($Y = KX + B$) at points R (middle of the right eye), 33 and 27 (Fig.12).

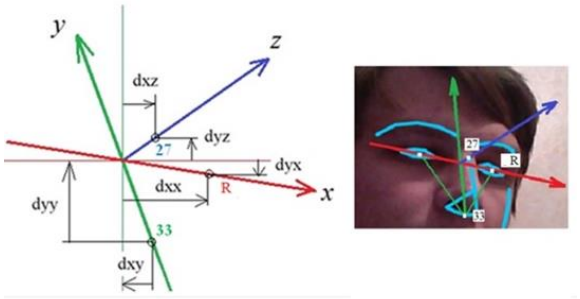


Fig. 12. Scheme for determining the projections of the axes of UCS

$$K_x = \frac{d_{yx}}{d_{xx}};\quad K_y = \frac{d_{yy}}{d_{xy}};\quad K_z = \frac{d_{yz}}{d_{xz}},$$

where the displacements ($d...$) are determined through the coordinates of the points *R*, 33 и 27:

$$d_{xx} = X_R - X_0;\quad d_{yx} = Y_R - Y_0;$$
$$d_{xy} = X_{33} - X_0;\quad d_{yy} = Y_{33} - Y_0;$$
$$d_{xz} = X_{27} - X_0;\quad d_{yz} = (Y_{27} - Y_0).$$

Next, we determine the projections of the segments of the coordinate polygonal line (*xzy*) for each pair of symmetric FLP. Consider this as an example of point's 3 and 13 (Fig.9).

1. Determine the distance between a pair of points:

$$D_X = |X_{13} - X_3|;$$
$$D_Y = |Y_{13} - Y_3|;$$
$$D = \sqrt{D_X^2 + D_Y^2}.$$

2. Determine the module of the projection of the *x* coordinates of points 13 и 3:

$$x_{13p} = x_{3p} = D/2.$$

3. Determine the projection modulus of the coordinates of *y* and *z* points. To do this, first find the middle of the segment between a pair of points:

$$X_{mid} = X_3 + \frac{(X_{13}-X_3)}{2};\quad Y_{mid} = Y_3 + \frac{(Y_{13}-Y_3)}{2}.$$

Move origin to point (X_0, Y_0) (Fig.11):

$$X_{mid} = X_{mid} - X_0;\quad Y_{mid} = Y_{mid} - Y_0.$$

Through a point with coordinates (*Xmid*, *Ymid*) we draw a straight line parallel to the projection of the *z*-axis until it intersects with the projection of the *y*-axis. Find the coordinates of the intersection point of these lines:

$$X_{int} = \frac{(Y_{mid} - K_z X_{mid})}{(K_y - K_z)};$$
$$Y_{int} = K_y X_{int}.$$

Now you can define the projection modules of the *z* and *y* coordinates for points 3 and 13:

$$y_{3p} = y_{13p} = \sqrt{X_{int}^2 + Y_{int}^2};$$
$$z_{3p} = z_{13p} = \sqrt{(X_{int} - X_{mid})^2 + (Y_{int} - Y_{mid})^2}.$$

4. Move from projections to normalized coordinates:

$$x_{13} = \frac{x_{13p}}{L_x};\ x_3 = -x_{13};$$
$$y_3 = y_{13} = -\frac{y_{3p}}{L_y};$$
$$z_3 = z_{13} = -\frac{z_{3p}}{L_z}.$$

Here *Lx*, *Ly* and *Lz* are the segments on the axes, cut off by points *R*, 33 и 27 (Fig.12):

$$L_x = \sqrt{d_{yx}^2 + d_{xx}^2};$$
$$L_y = \sqrt{d_{yy}^2 + d_{xy}^2};$$
$$L_z = \sqrt{d_{xz}^2 + d_{yz}^2}.$$

The values of the normalized coordinates should be approximately the same in modulus. Otherwise, the distance-features between the points will depend only on the values of the largest coordinate. For example, Fig. 10 shows that the value of the *z* coordinate of point 3 is significantly larger compared to the *x* and *y* coordinates. It is necessary to reduce the *z* coordinate by 3 times.

Obviously, when using the 3D model, the number of incorrect images will decrease compared to the 2-D model discussed above. For example, in Fig. 7, the image on the right can be considered correct for the 3D model - in contrast to the 2D model. To discard incorrect images in the 3D model, it is necessary to leave only those features that are determined by subjective factors (lighting, face distortion, etc.). It is enough to consider only features of parallelism of

lines that connect symmetrical points, for example, lines between the corner points of the mouth and lines between the corner points of the eyes.

V. ASSESSMENT OF QUALITY OF FACE IDENTIFICATION ALGORITHMS

For testing, the FEI Face [17] database was selected. For experiments, the initial weight of each feature was taken at one point. The choice of the winner from the set of applicants was carried out according to the highest amount of points scored.

Confidence intervals of the features of each applicant were determined as a result of processing the values of the characteristics obtained for the set of images that remained after discarding incorrect ones.

When experimenting with a 2D model, the proportion of incorrect images was about 20%. When testing a 3D model, the percentage of incorrect images decreased to 10%.

To assess the quality of face recognition models, FRR (False Reject Rate) и FAR (False Acceptance Rate) [18] metrics were used. The results are shown in the table I:

TABLE I.

Recognition model	**Quality criteria**	
	FRR	***FAR***
2D (all images)	38%	42%
2D (correct)	40%	40%
3D (all images)	43%	37%
3D (correct)	46%	34%

VI. CONCLUSIONS

The main problems of face recognition methods and the possibilities of overcoming them were identified.

An approach for facial recognition with the conversion and discarding of incorrect images is proposed. An information technology for face recognition was developed, with the help of which studies of the proposed approach were conducted. To do this, methods have been developed for face alignment using 2D transformations and 3D reconstruction of anthropometric points of the face.

The quality of the proposed approach was evaluated. When discarding irregular images and aligning the face by three-dimensional reconstruction of the model, as can be seen from table 1, the results improve by about 5%.

In the conducted studies, the problem of determining the similarity was considered in a simplified form. For example, the weight of each trait was estimated at one point. However, the weight of the symptoms may vary depending on a number of factors. Further research that can improve face recognition results is related to solving this problem.

REFERENCES

[1] Yu.Krak, A.Barmak, E.Baraban, “Usage of NURBS-approximation for construction of spatial model of human face”, Journal of Automation and Information Sciences, Vol. 43(2), pp. 71-81, 2011.

[2] Mac Brennan. Facial Key Point Detection, (2018). URL: https://medium.com/datadriveninvestor/facial-key-point-detection-88ccfaeaf9ee

[3] D.Chen, S.Ren, Y.Wei, X.Cao, J.Sun, “Joint cascade face detection and alignment”, In Proc. ECCV, 7,2014.

[4] C.Lu, X.Tang, “Surpassing human-level face verification performance on LFW with gaussianface”, CoRR, 1: abs/1404.3840, 2014.

[5] M.Galvánek, K.Furmanová, I.Chalás, J.Sochor, “Automated facial landmark detection, comparison and visualization Proceedings of the 31st spring conference on computer graphics”, Comenius University, Bratislava, Slovakia, p. 21–8, 2015.

[6] J.Wang, Y.Song, T.Leung, C.Rosenberg, J.Wang, J.Philbin, B.Chen, Y.Wu, “Learning fine-grained image similarity with deep ranking“, CoRR, 2: abs/1404.4661, 2014.

[7] S.Liang, J.Wu, S.M.Weinberg, L.G.Shapiro, “Improved detection of landmarks on 3D human face data”, Engineering in medicine and biology society (EMBC), 35th annual international conference of the IEEE, 2013.

[8] L.Wolf, T.Hassner, I.Maoz, “Face recognition in unconstrained videos with matched background similarity”, In IEEE Conf. on CVPR, 5, 2011.

[9] M.Evison, I.Dryden, N.Fieller, X.Mallett, L.Morecroft, D.Schofield, RV.Bruegge, “Key parameters of face shape variation in 3D in a large sample”, J Forensic Sci, Vol. 55(1), pp. 159-162, 2010.

[10] V.Kuznetsov, Iu.Krak,, O.Barmak, A.Kulias, “Facial Expressions Analysis for Applications in the Study of Sign Language”. CEUR Workshop Proceedings 2353, CEUR-WS.org., pp.159-172, 2019.

[11] I.Kryvonos, I.Krak, “Modeling human hand movements, facial expressions, and articulation to synthesize and visualize gesture information”, Cybernetics and Systems Analysis, Vol.47(4), pp. 501-505, 2011

[12] I.Kryvonos, I.Krak, O.Barmak, A.Ternov, V.Kuznetsov, “Information Technology for the Analysis of Mimic Expressions of Human Emotional States”, Cybernetics and Systems Analysis, Vol.51, pp. 25-33, 2015

[13] A.S.Jackson, A.Bulat, V.Argyriou, G.Tzimiropoulos, “Large Pose 3D Face Reconstruction from a Single Image via Direct Volumetric CNN Regression”, ICCV, 2017, URL: http://aaronsplace.co.uk/papers/jackson2017recon/

[14] L.Zhang, D.Tjondronegoro, V.Chandran, “Geometry vs. Appearance for Discriminating between Posed and Spontaneous Emotions”, NIP, 2011

[15] P.A.Viola, M.J.Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, IEEE CVPR, 1, 2001.

[16] D.Zhou, D.Petrovska-Delacretaz, B.Dorizzi, “Automatic Landmark Location with a Combined Active Shape Model”, BTAS, 2009.

[17] FEI Face Database. URL: https://fei.edu.br/~cet/facedatabase.html

[18] W.Bao, H.Li, N.Li, W.Jiang, “A liveness detection method for face recognition based on optical flow field”, In: Image Analysis and Signal Processing. IASP 2009, International Conference on, pp. 233–236, 2009.

УДК 004.93’1
DOI: 10.31891/CSIT-2020-2-8

ІВАЩЕНКО С. О., КАЛИТА О. Д.,
БАРМАК О. В., СКРИПНИК Т. К.
Хмельницький національний університет

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЗНАЧЕННЯ ХАРАКТЕРНИХ ОЗНАК НА ОБЛИЧЧІ ДЛЯ РОЗПІЗНАВАННЯ ЕМОЦІЙНИХ ПРОЯВІВ

У роботі наведено та експериментально досліджено технологію визначення характерних ознак на обличчі для розпізнавання емоційних проявів. Наведені кроки технології дозволяють отримати достатньо простим, обчислювально нескладним способом модель подання емоційних станів на обличчі за допомогою векторів характерних ознак згрупованих за класами емоцій та визначити коефіцієнти гіперплощин, які опукло обмежують ці класи і можуть бути використанні для наступного розпізнавання довільного емоційного виразуобличчя.

Ключові слова: Facial Expression Recognition Systems

IVASCHENKO S., KALYTA O., BARMAK O., SKRYPNYK T.
Khmelnytskyi National University, Ukraine

INFORMATION TECHNOLOGY FOR DETERMINATION OF CHARACTERISTICS ON THE FACE FOR EMOTIONAL RECOGNITION

One of the ways to process an image presented in the form of a set of pixels, in order to further identify, classify the objects present on it is to display the specified set in the form of sets of certain features. Such features are not universal in nature, but rather significantly depend on the tasks under consideration. For certain classes of problems, such features (model) are selected that best allow the application of appropriate methods to solve the problem. The paper considers a class of problems for recognizing the emotional state on a person's face.

In, a convolutional neural network (CNN) is used to detect emotions. CNN differs from multilayer perceptron (MLP) in that they have hidden layers called convolutional layers. The proposed method is based on a two-tier CNN system. At the first level, the background of the image is removed to better reflect emotions. A standard CNN network module is used to obtain the primary expression vector (EV). EV is formed by tracking the relevant important points of the face. EV is directly related to changes in facial expression. EV is obtained using a basic perceptron unit plotted on a face image with the background removed. In the proposed model at the last stage there is a nonconvolutionary perceptron layer. Each of the convoluted layers receives input data (images), converts them, and then takes them to the next level. After detecting a face, the CNN filter of the second part captures parts of the face, such as eyes, ears, lips, nose and cheeks. The authors agree that the method has some limitations, and especially requires high computing power when setting up CNN.

The technology of determination of characteristic features on the face for recognition of emotional manifestations is presented and experimentally investigated.

These steps of technology allow to obtain a fairly simple, computationally simple model of representation of emotional states on the face with the help of characteristic vectors grouped by classes of emotions and determine the coefficients of hyperplanes that convexly limit these classes and can be used for subsequent recognition of arbitrary emotional facial expressions.

Keywords: Facial Expression Recognition Systems

Вступ

Одним із шляхів обробки зображення, поданого у вигляді множини пікселів, з метою подальшої ідентифікації, класифікації об’єктів, що присутні на ньому є відображення зазначеної множини у вигляді наборів певних ознак[1]. Такі ознаки несуть не універсальний характер, а досить суттєво залежать від задач, що розглядаються. Для певних класів задач підбираються такі ознаки (модель), які найкращим чином дозволяють застосувати відповідні методи для вирішення поставленої задачі. У роботі розглянуто клас задач по розпізнаванню емоційного стану на обличчі людини.

Аналіз досліджень та публікацій

У роботі [2] для виявлення емоцій використовується конволюційна нейронна мережа (CNN). CNN відрізняється від багатошарового персептрона (MLP) тим, що вони мають приховані шари, які називаються згортковими шарами. Запропонований метод базується на дворівневій системі CNN. На першому рівні видаляється фон зображення для кращого відображення емоцій. Для отримання первинного експресійного вектора (EV) використовується стандартний мережевий модуль CNN. EV формується шляхом відстеження відповідних важливих точок обличчя. EV безпосередньо пов'язаний зі змінами в виразі обличчя. EV отримують за допомогою базової перцептронної одиниці, нанесеної на зображення обличчя з вилученим фоном. У запропонованій моделі на останній стадії присутній неконволюційний шар перцептрону. Кожен із згорткових шарів отримує вхідні дані (зображення), перетворює їх, а потім переводить їх на наступний рівень. Після виявлення обличчя фільтр CNN другої частини вловлює частини обличчя, такі як очі, вуха, губи, ніс і щоки. Автори погоджуються, що метод має деякі обмеження, а особливо потребує високу обчислювальну потужність під час налаштування CNN.

Метою роботи [3] є досягнення кращої точності розпізнавання емоцій на обличчі та класифікації їх з обмеженої множини тренувальних зразків при різній інтенсивності освітлення. Запропоновано відповідний метод. Глобальні та локальні особливості зображень виразів обличчя були ідентифіковані за допомогою Haar Wavelet Transform (HWT) та вейвлетів Габора. Розмірність знайдених ознак зменшується за допомогою методу нелінійних головних компонент (NLPCA). Для з'єднання глобальних та локальних особливостей використовувались об'єднувальні методи синтезу. Щоб розпізнати та класифікувати шість емоцій (радість,

сюрприз, страх, огиду, гнів і смуток) з міміки, використовувався метод опорних векторів. Запропонований метод оцінюється за розширеним набором даних Кона-Канаде. Середні показники розпізнавання 97,3% і 98% досягаються при двох версіях запропонованого способу, забезпечуючи кращу точність розпізнавання порівняно з існуючими методами.

У роботі [4] використовували схему перехресних перевірок K-Fold для експериментів щодо розміру бази даних. За кожною базою даних навчальні та тестові набори відокремлюються через 3-кратну перехресну перевірку. У триразовій перехресній валідації вся база даних поділяється на три рівні набори зображень випадковим чином, потім два набори з них поєднуються для тренування, а ще один для тестування. Експерименти виконуються на монохромних зображеннях шляхом зміни розміру у всій базі даних. Дана робота в основному була зосереджена на належному виявленні обличчя та частин обличчя за допомогою існуючих алгоритмів Віоли Джонса, і після багатьох експериментів було помічено, що на деяких зображеннях алгоритм Віоли Джонса не належно визначає обличчя та частини обличчя через проблеми з освітленням і варіацією. Алгоритм Віоли Джонса дуже чутливий до зміни форми та інтенсивності. У модифікованому алгоритмі Віоли Джонса, інтенсивність зображення та розмір (елементи Хаара) різноманітні.

Автори підходу [5] для незалежного розпізнавання виразів обличчя людини використовують глибинні відеодані як вхід до системи, де в кожному кадрі інтенсивність пікселів розподіляється за відстанями до камери. У цій роботі застосовується новий процес вилучення функцій, який називається "локальний шаблон спрямованої позиції" (LDPP). У LDPP, після вилучення локальних напрямків для кожного пікселя, таких як застосований у типовому локальному режимі спрямованості (LDP), верхні позиційні положення сили розглядаються у двійковій формі разом із бітами знаку їх сили.

Системи розпізнавання емоційної складової на обличчі (Facial Expression Recognition Systems, FER) мають широке застосування в різних областях, таких як комп'ютерна взаємодія, системи охорони здоров’я та соціальний маркетинг. Однак аналіз виразів обличчя є надзвичайно складним через тонкі та швидкоплинні рухи людей, складне середовище зображення в реальних образах, відео. Існують три основні проблеми, викликані різницею освітленості, залежністю від предмета та зміною постави голови; вони широко впливають на продуктивність системи FER. Сучасні методи в системах FER ефективні для використання в контрольованих лабораторних умовах, але не для застосування в реальних ситуаціях [6].

Для розпізнавання емоційної складової на обличчі необхідним кроком є виявлення ділянки обличчя на зображенні. Одним із методів виявлення обличчя, який показав достойні результати, є каскадний класифікатор Хаара [7]. Даний підхід базується на вейвлетах Хаара, техніці аналізу пікселів в певних областях зображення (рис.1). Для виявлення обличчя людини особливості Хаара є найважливішою частиною класифікатора. Характеристики Хаара застосовуються для виявлення існування ознак у даному зображенні. Кожна особливість, як правило, обробляє одне значення. Воно обчислюється відніманням кількості пікселів через білий прямокутник від кількості пікселів через чорний прямокутник. Характеристики Хаара існують у вигляді прямокутних особливостей.

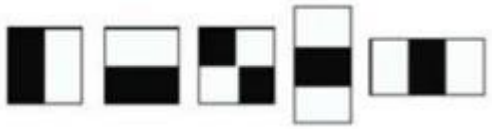


Рис. 1. Особливості Хаара

Для виявлення специфічних точок досить ефективним є підхід FaceLandmarksLBF [8]. У роботі представлений високоефективний, точний регресійний підхід. Цей підхід має дві нові компоненти: набір локальних бінарних особливостей та принцип локальності для вивчення цих особливостей. Принцип локальності сприяє аналізу набору локальних бінарних ознак для кожного орієнтира обличчя незалежно. Отримані локальні бінарні ознаки використовуються для спільного аналізу лінійної регресії для кінцевого результату. Крім того, оскільки отримання та регресія локальних бінарних функцій обчислюється дуже дешево, система набагато швидша, ніж попередні методи.

Основні підходи для розпізнавання емоцій мають складну модель і потребують високої обчислювальної потужності. Спрощення моделі з урахуванням основних факторів, які виражають емоції, їх ефективне об’єднання, а також швидкодія є важливою складовою у системах, які повинні реагувати миттєво.

Прояв емоцій у реальних критичних ситуаціях є максимально природнім. Виявлення основних специфічних ознак емоційного стану з урахуванням тривалості макровиразів та особливостей ситуації потребує швидкого обчислення.

Отже метою роботи є інформаційна технологія отримання характерних ознак за зображенням обличчя людини при відображенні на ньому емоційних станів у вигляді моделі, придатної для швидкої класифікації.

Технологія визначення ознак на обличчі для розпізнавання емоційних проявів

Базовою одиницею розпізнавання є зображення людського обличчя. У роботі [9] встановлено, що найбільш вираженим частинами обличчя під час емоцій є: брови, очі, губи (рис.2). У роботі було досліджено

дрібний рівень, на фізичні особливості якого найбільше покладаються при розшифруванні міміки. В експерименті окремі обличчя, що виражають основні емоції за Екманом [10], були заховані за маскою з 48 плиток, яку послідовно розкривали. Лицьові частини, що мають найвищу діагностичну цінність для ідентифікації емоційного стану, зазвичай розташовуються в областях, відповідних одиницям дії з системи кодування обличчя (FACS)[10].

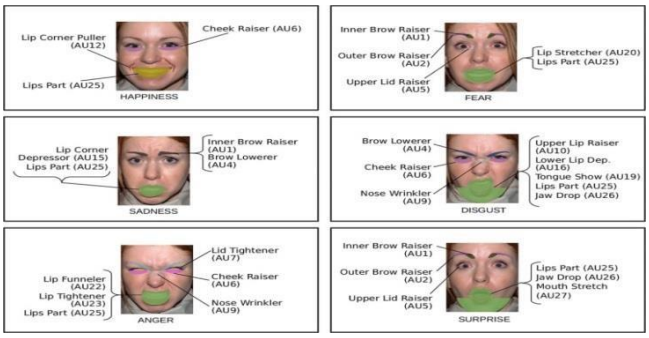


Рис. 2. Частини обличчя під час різних емоційних станів

Рис. 3. Приклад зображень

Під час дослідження використовувалася база ADFES [11], яка надає 12 зображень (7 чоловіків, 5 жінок), як вираження 6-и основних емоційних станів(гнів, страх, радість, сум, відраза, здивування) та 3-х комплексних станів (презирство, гордість, збентеження), а також нейтральність (рис.3).

Виділені частини обличчя можуть перебувати у трьох станах [12]:

- брови (підняті, опущені, нормальні);
- очі (розплющені, примружені, нормальні);
- губи (розтягнуті, зжаті, нормальні).

Звідси формується вектор ознак, який описує емоційний стан: $\{x_1, x_2, x_3\}$. $x_1, x_2, x_3 \in [0,\ 1]$, причому $x_1 \in [0,\ 0.2]$ – для примружених очей; $x_1 \in [0.4,\ 0.6]$ – для нормальних очей; $x_1 \in [0.8,\ 1.0]$ – для розплющених очей; $x_2 \in [0,\ 0.2]$ – для зжатих губ; $x_2 \in [0.4,\ 0.6]$ – для нормальних губ; $x_2 \in [0.8,\ 1.0]$ – для розтягнутих губ; $x_3 \in [0,\ 0.2]$ – для опущених брів; $x_3 \in [0.4,\ 0.6]$ – для нормальних брів; $x_3 \in [0.8,\ 1.0]$ – для піднятих брів (рис.4).

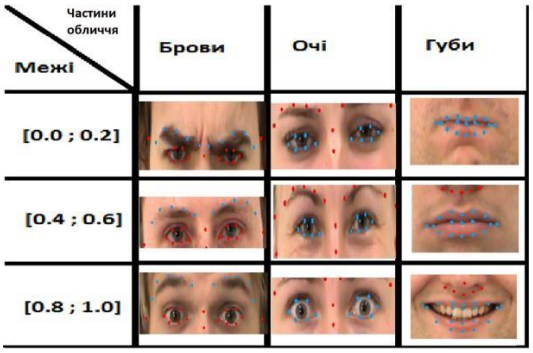


Рис. 4. Межі параметрів та частини обличчя

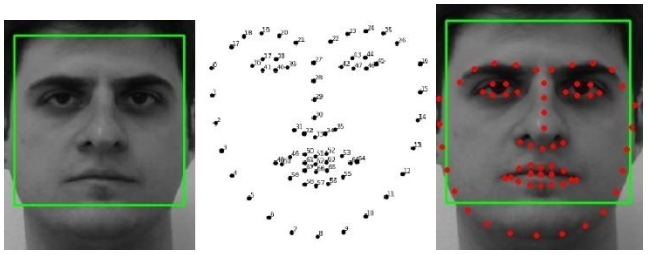


Рис. 5. Виявлення обличчя та специфічних точок

Для визначення області обличчя використовуються Haarcascade, для визначення специфічних точок обличчя використовується FaceLandmarksLBF, в результаті отримуємо 68 точок (рис.5).

Основні частини обличчя, які використовуються для визначення емоцій: брови, очі, губи. В результаті маємо вектор ознак: $v = \{x_1, x_2, x_3\}$.

Для визначення відстані між точками використовуємо формулу:

$$distance = \sqrt{(p1x - p2x)^2 + (p1y - p2y)^2} \quad (1)$$

Визначення значення ока:

$$pointA = (flm37 + flm38)/2\ , \quad (2)$$

де flm37 – 37 точка на обличчі;
flm38 – 38 точка на обличчі.

$$pointB = (flm40 + flm41)/2\ , \quad (3)$$

де flm40 – 40 точка на обличчі;
flm41 – 41 точка на обличчі.

$$\text{eye}fd = distance\,(pointA, pointB) \quad (4)$$

$$eyesd = distance(flm36, flm39), \quad (5)$$

де flm36 – 36 точка на обличчі;
flm39 – 39 точка на обличчі.

$$eyevalue = \frac{eyefd}{eyesd} / EYEMAX, \quad (6)$$

де EYEMAX – максимальне доступне значення відстані для ока.

Визначення значення брів:

$$eyebrowfd = distance(eyesd, flm19), \quad (7)$$

де flm19 – 19 точка на обличчі.

$$\text{eye}browsd = distance(flm36, flm45), \quad (8)$$

де flm36 – 36 точка на обличчі;
flm45 – 45 точка на обличчі.

$$eyebrowvalue = \frac{eyefd}{eyesd} / EYEBROWMAX, \quad (9)$$

де $EYEBROWMAX$ – максимальне доступне значення відстані для брів.

Визначення значення губ:

$$\text{lips}sd = distance(flm48, flm54), \quad (10)$$

де flm48 – 48 точка на обличчі;
flm54 – 54 точка на обличчі.

$$lipsvalue = \frac{\frac{sd}{fd}}{\text{LIPSMAX}}, \quad (11)$$

де LIPSMAX - максимальне доступне значення відстані для губ.

Перетворення фізичних значень до значень моделі відбувається за наступними формулами:

$$\begin{cases} eyeval = (eyevalue - 0.0) * \frac{(0.2-0.0)}{(0.55-0.0)} + 0.0, \text{якщо } eyevalue < 0.55 \\ eyeval = (eyevalue - 0.55) * \frac{(0.6-0.4)}{(0.75-0.55)} + 0.4, \text{якщо } eyevalue < 0.75 \\ eyeval = (eyevalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} + 0.8, \text{якщо } eyevalue < 1.0 \end{cases} \quad (12)$$

$$\begin{cases} eyebv = (eyebrowvalue - 0.0) * \frac{(0.2-0.0)}{(0.65-0.0)} + 0.0, \text{якщо } eyebrowvalue < 0.55 \\ eyebv = (eyebrowvalue - 0.65) * \frac{(0.6-0.4)}{(0.75-0.65)} + 0.4, \text{якщо } eyebrowvalue < 0.75 \\ eyebv = (eyebrowvalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} + 0.8, \text{якщо } eyevbrowvalue < 1.0 \end{cases} \quad (13)$$

$$\begin{cases} lipsval = (lipsvalue - 0.0) * \frac{(0.2-0.0)}{(0.65-0.0)} + 0.0, \text{якщо } lipsvalue < 0.65 \\ lipsval = (lipsvalue - 0.65) * \frac{(0.6-0.4)}{(0.75-0.65)} + 0.4, \text{якщо } lipsvalue < 0.75 \\ lipsval = (lipsvalue - 0.75) * \frac{(1.0-0.8)}{(1.0-0.75)} + 0.8, \text{якщо } lipsvalue < 1.0 \end{cases} \quad (14)$$

Для аналізу отриманих даних доцільно використати метод SMACOF для багатовимірного шкалювання (MDS), за допомогою якого можна візуалізувати отримані результати.

Основною у методі SMACOF є мажоризація стресу – стратегічна оптимізація, що використовується в багатомірному шкалюванні, де для набору з *n* елементів розмірності *m* шукається конфігурація *Xn* точок в *r*-мірному просторі, який минімізує функцією мажоризації $\sigma(X)$. Зазвичай *r* дорівнює 2 або 3, тобто (*n* × *r*) матриця *X* перераховує точки в 2- або 3-мірному евклідовому просторі, так що результат може бути відображений візуально. Функція $\boldsymbol{\sigma}$ функцією втрат, яка вимірює квадрат різниці між ідеальною (*m*-мірною) відстанню в *r*-мірному просторі.

Вхідні дані MDS, як правило, є *n* × *n* матриця відмінностей на основі спостережуваних даних. Задача, яка вирішується – знайти *{i, j} = 1, ..., n* точки в маломірному евклідовому просторі таким чином, що відстані між точками наближаються до заданих відмінностей δ_{ij} [12]. Таким чином необхідно знайти *n* × *p* матрицю *X* така де $d_{ij}(X) \approx \delta_{ij}$, де

$$d_{ij}(X) = \sqrt{\sum_{s=1}^{p}(x_{is} - x_{js})^2} \quad (15)$$

Для пошуку точки на лінії: маємо $StartPoint = \{x_0, y_0\}$, як початок лінії та $EndPoint = \{x_1, y_1\}$, як кінець лінії. Визначається відстань d_t від нової точки до початку лінії.

Довжина лінії обчислюється наступним чином:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad (16)$$

Знаходимо співвідношення відстаней:

$$t = \frac{d_t}{d} \quad (17)$$

Тоді координата *x* нової точки має значення:

$$x_t = (1 - t)x_0 + tx_1 \quad (18)$$

Координата *y* нової точки має значення:

$$y_t = (1-t)y_0 + ty_1 \tag{19}$$

Отримуємо $NewPoint = \{x_t, y_t\}$.

Обернений метод SMACOF надає: $x^L(k,j) \in R^m, i = 1,m;\ k = 1,l$. За множиною точок $x^L(k,j) \in R^m, i = 1,m;\ k = 1,l$, які розташовані на відповідних гіперплощинах, для визначення коефіцієнти цих гіперплощин, формуємо системи лінійних алгебраїчних рівнянь,. Таких систем буде стільки, скільки ліній у R^2 . Для *i*-ї гіперплощини система рівнянь матиме вигляд:

$$\left\{\begin{array}{l} w_1x_1^L(i,1) + w_2x_2^L(i,1) + \ldots + w_mx_m^L(i,1) + b = 0; \\ w_1x_1^L(i,2) + w_2x_2^L(i,2) + \ldots + w_mx_m^L(i,2) + b = 0; \\ \ldots \\ w_1x_1^L(i,m) + w_2x_2^L(i,m) + \ldots + w_mx_m^L(i,m) + b = 0. \end{array}\right\} \tag{20}$$

$$w_1 = (-1^{1+1})\begin{pmatrix} x_2^L(i,1) & x_3^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_2^L(i,2) & x_3^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_2^L(i,m) & x_3^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_1 \tag{21}$$

$$w_2 = (-1^{1+2})\begin{pmatrix} x_1^L(i,1) & x_3^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_3^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_1^L(i,m) & x_3^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_2 \tag{22}$$

$$w_k = (-1^{1+k})\begin{pmatrix} x_1^L(i,1) & x_{k-1}^L(i,1) & x_{k+1}^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_{k-1}^L(i,2) & x_{k+1}^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & & \\ x_1^L(i,m) & x_{k-1}^L(i,m) & x_{k+1}^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} x_k \tag{23}$$

$$b = (-1^{2+m})\begin{pmatrix} x_1^L(i,1) & x_2^L(i,1) & \ldots & x_m^L(i,1) & 1 \\ x_1^L(i,2) & x_2^L(i,2) & \ldots & x_m^L(i,2) & 1 \\ & & \ldots & & \\ x_1^L(i,m) & x_2^L(i,m) & \ldots & x_m^L(i,m) & 1 \end{pmatrix} \tag{24}$$

Можна виділити наступні кроки: підготовка векторів емоційних станів, візуалізація вхідних даних у двомірний простір, мануальне розділення на класи, обернене перетворення до *n*-мірного простору, отримання коефіцієнтів гіперплощин.

Для формування вхідних даних для дослідження за допомогою модуля OpenCV відбувається отримання зображення, виявлення обличчя за допомогою методу Віоли – Джонса, FaceLandmarksLBF використовується для отримання специфічних точок обличчя.

За допомогою запропонованого алгоритму формуються вектори. В результаті отримано 48 векторів емоційних станів.

Далі наведено кроки необхідні для визначення коефіцієнтів гіперплощини.

Крок 1. За допомогою методу *SMACOF* відбувається перехід від n-мірних даних до двомірної системи (рис. 6).

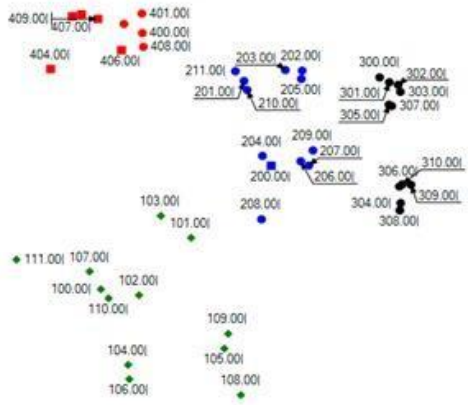


Рис. 6. Результат методу SMACOF

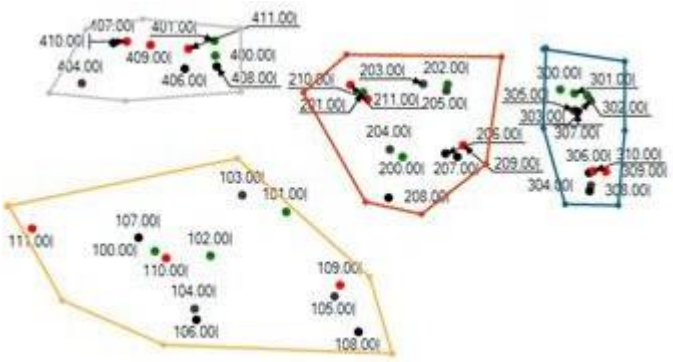


Рис. 7. Результат мануального розділення

Крок 2. Отримані масиви точок мануально розділяються на класи (рис. 6)

Для того щоб мати достатньо даних для перетворення до *N*-мірного простору відбувається визначення додаткової точки (рис. 8).

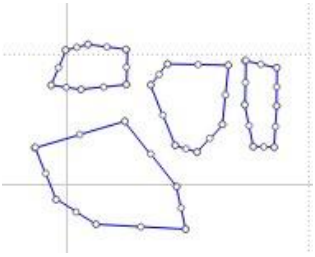


Рис. 8. Визначені точки ліній

Рис. 9. Алгоритм пошуку коректних коефіцієнтів гіперплощини

Крок 3. При використанні SMACOF існує ймовірність потрапляння в локальні мінімуми, особливо при малій розмірності. Також важливою складовою є визначення початкових координат [13]. Тому, поки всі коефіцієнти гіперплощини не будуть визначені (а також вони будуть різні) для відповідних векторів продовжується робота алгоритму SMACOF (рис. 9). Наприклад, якщо всі вектори страху при перевірці принадлежності до даної множини видають однакові патерни (‘+’ або ‘-’) – коефіцієнти гіперлощини можна прийняти. Якщо дана умова коректна для всіх емоційних станів – рішення знайдено.

Було отримано наступні патерни для емоційних станів:

$Happy = \{+,+,+,+,-,+\}, Sad = \{-,-,+,+,-,+\}, Anger = \{+,+,+,+,-,-\}, Fear = \{+,+,-,-,+,+\}$.

Маємо наступну схему дослідження (рис. 10).

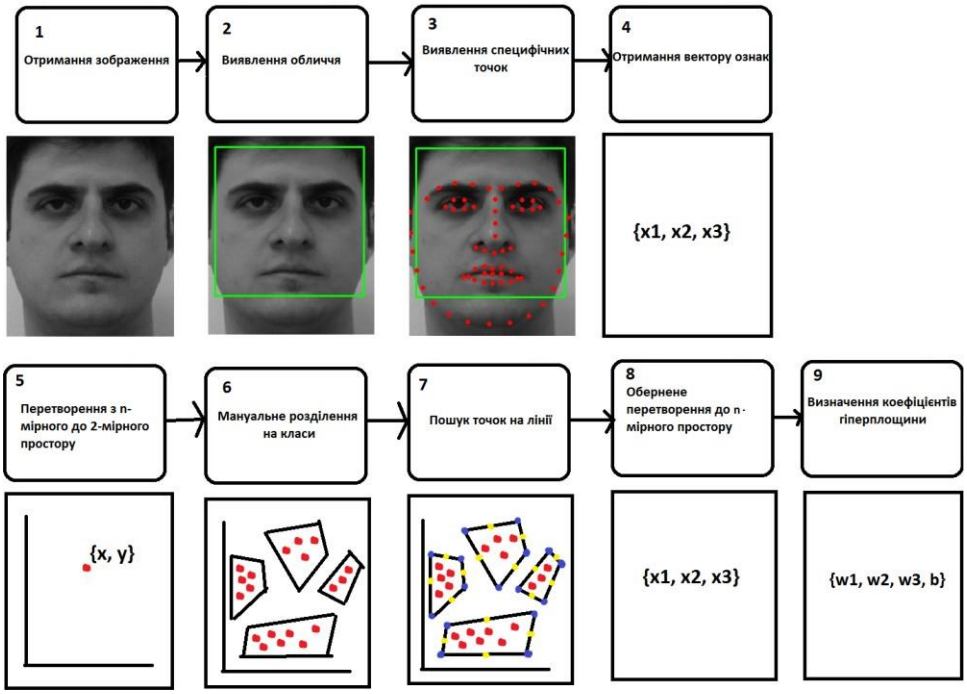


Рис. 10. Схема дослідження

Висновки

У роботі наведено та експериментально досліджено технологію визначення характерних ознак на обличчі для розпізнавання емоційних проявів. Наведені кроки технології дозволяють отримати достатньо простим, обчислювально нескладним способом модель подання емоційних станів на обличчі за допомогою векторів характерних ознак згрупованих за класами емоцій та визначити коефіцієнти гіперплощин, які опукло обмежують ці класи і можуть бути використанні для наступного розпізнавання довільного емоційного виразу обличчя.

Подальші дослідження направлені на створення за допомогою наведеної інформаційної технології застосування для розпізнавання основних емоційних станів на обличчі. Таке застосування планується використовувати у системах які потребують контроля емоційного стану особи.

Література

1. Ahdid R., Taifi K., Safi S., Manaut B. A Survey on Facial Feature Points DetectionTechniques and Approaches. World Academy of Science, Engineering and Technology. Version 10005826.
2. Mehendale N. Facial emotion recognition using convolutional neural networks (FERC). SN Applied Sciences, February 2020.
3. Chirra V. R., Uyyala S. R., Kolli V. K. Facial Emotion RecognitionUsing NLPCA and SVM. Traitement du Signal, February, 2019.
4. Kuldeep Y., Joyeeta S., Facial expression recognition using modified Viola-John's algorithm and KNN classifier. Springer Science+Business Media, January 2020
5. Uddin, M.Z.; Hassan, M.M.; Almogren, A.; Alamri, A.; Alrubaian, M.; Fortino, G. Facial expression recognition utilizing local direction-based robust features and deep belief network. IEEE Access 2017
6. Najmeh S., Guangyan H., Borui C., Wei L., Chi-Hung C., Yong X., Jing H. A Review on Automatic Facial Expression Recognition. Systems Assisted by Multimodal Sensor Data, 2019 Apr.
7. Thin T.S. Detection of Faces from Images Using Haar Cascade Classifier, IRE Journals, JUN 2020
8. Shaoqing R., Xudong C., Yichen W., Jian S. Face Alignment at 3000 FPS via Regressing Local Binary Features, 2014
9. Martin W., Maria V., Berna K., Julia S., Johanna K., Mapping the emotional face. How individual face parts contribute to successful emotion recognition, PLoS ONE, 2017 May
10. Ekman P., Friesen W. V., Hager J. C.: The Facial Action Coding System. Salt Lake City, UT Research Nexus eBook (2002)
11. Tanja S. H. W., Chris A., Mark B., Validation of the Amsterdam Dynamic Facial Expression Set – Bath Intensity Variations (ADFES-BIV): A Set of Videos Expressing Low, Intermediate, and High Intensity Emotions, PLoS One, 2016 Jan
12. Facial Expressions [Електронний ресурс]. – Режим доступу: https://www.eiagroup.com/knowledge/facial-expressions/
13. Ingwer Borg, Patrick Mair, "The Choice of Initial Configurations inMultidimensional Scaling: Local Minima, Fit, andInterpretability", February 2017

Надійшла / Paper received: 11.09.2020
Надрукована / Paper Printed : 03.11.2020

ДИПЛОМНА РОБОТА МАГІСТРА «Моделювання ознак зображення для задач розпізнавання»

Виконав студент групи КНм-19
Іващенко Сергій

АКТУАЛЬНІСТЬ

Сьогодні більшість нашого часу в повсякденному житті витрачається на взаємодію з комп'ютерами та мобільними телефонами завдяки прогресу технологій та їхньому розповсюдженню. Вони відіграють важливу роль у нашому житті, і величезна кількість існуючих програмних інтерфейсів є невербальними, примітивними та лаконічними. Включення розпізнавання емоційних виразів для фіксації відчуттів та емоційного стану користувачів може різко покращити взаємодію людини та комп'ютера. HCI вважається однією з найпривабливіших і швидко зростаючих галузей.

Розпізнавання емоцій по ознакам обличчя є однією з технологій, що найбільш динамічно розвивається. Розпізнавання виразів обличчя може широко застосовуватися в різних областях досліджень, таких як діагностика психічних захворювань та виявлення соціальної та фізіологічної взаємодії людини.

Незважаючи на те, що лабораторно керовані системи FER досягають дуже високої точності, близько 97%, технічний перехід від лабораторії до реальних застосувань стикається з великим бар'єром дуже низької точності, приблизно 50%. Тому розробка такого застосування є актуальним та затребуваним для практичного застосування. Адже збільшення точності розпізнавання важливий аспект дослідження.

Мета ДРМ

Метою роботи є створення і дослідження моделі мімічних проявів емоційного стану людини, розробка інформаційної системи для розпізнавання мімічних проявів емоцій.

Для досягнення поставленої мети визначені наступні задачі дослідження:

- створення та дослідження моделі мімічних проявів емоцій;
- дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;
- розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;
- валідація та верифікація моделі, підходу та інформаційної системи.

Методи використані під час дослідження

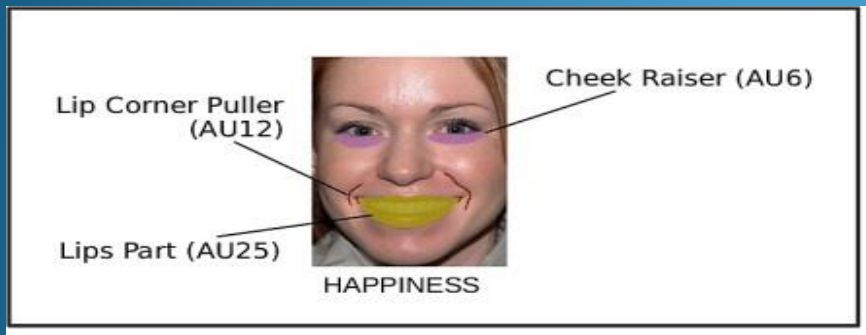
- метод Віоли - Джонса;
- facemarkLBF;
- метод форм. вектора;
- Multidimensional scaling;
- SMACOF;
- метод верифікації емоції.

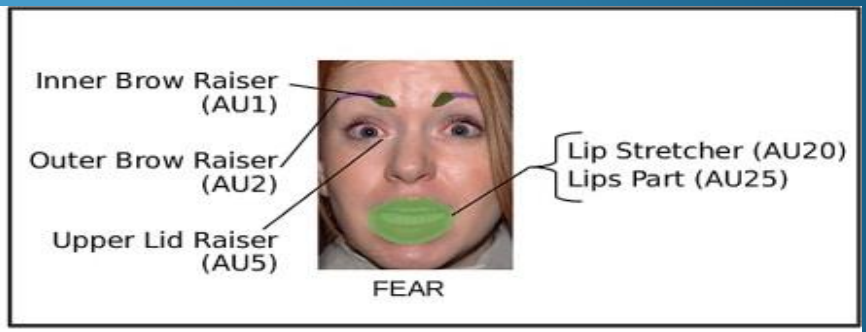
База даних для дослідження

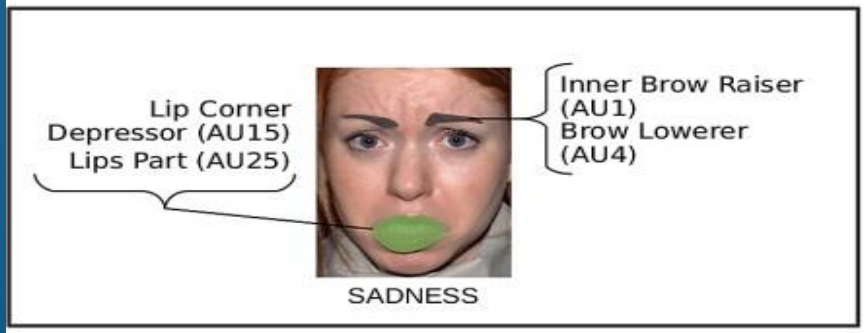
Робота відбувалася з базою зображень ADFES, яка надає 12 зображень (7 чоловіків, 5 жінок), як вираження 6-и основних емоційних станів(гнів, страх, радість, сум, відраза, здивування) та 3-х комплексних станів (презирство, гордість, збентеження), а також нейтральність.

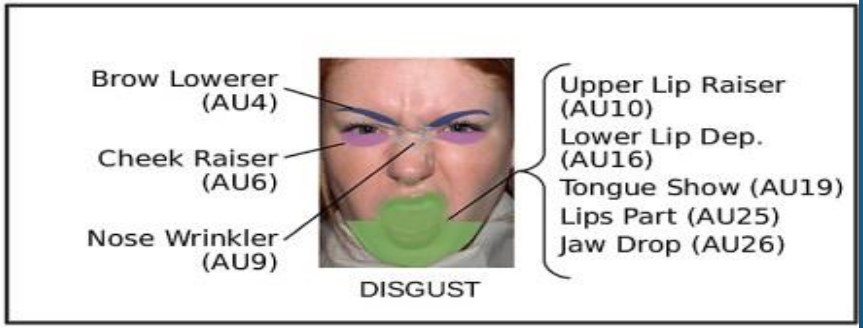
Модель емоційного стану

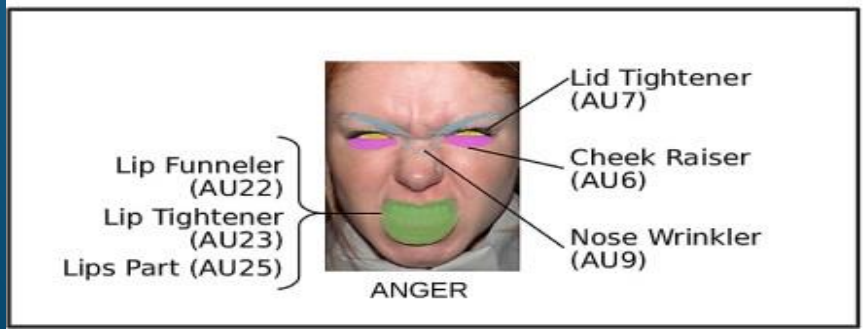
Проаналізувавши статтю «Mapping the emotional face. How individual face parts contribute to successful emotion recognition» було визначено, що найбільш вираженими частинами обличчя є: брови, очі та губи.

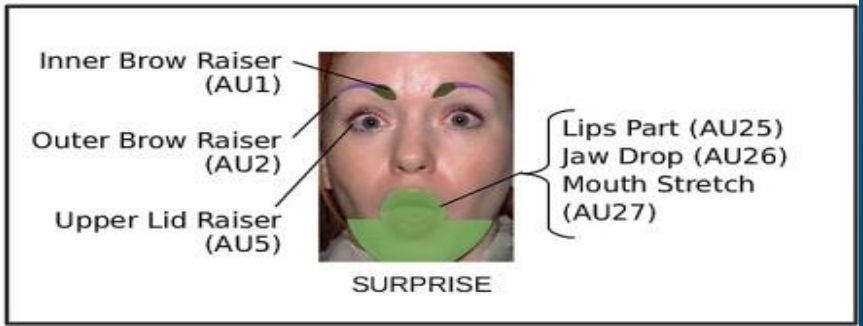




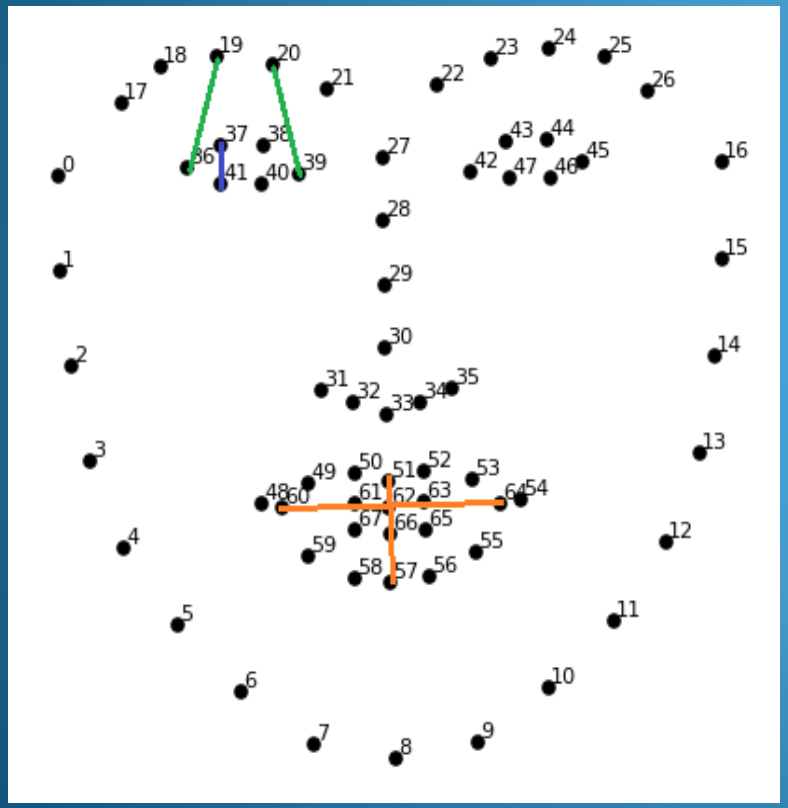


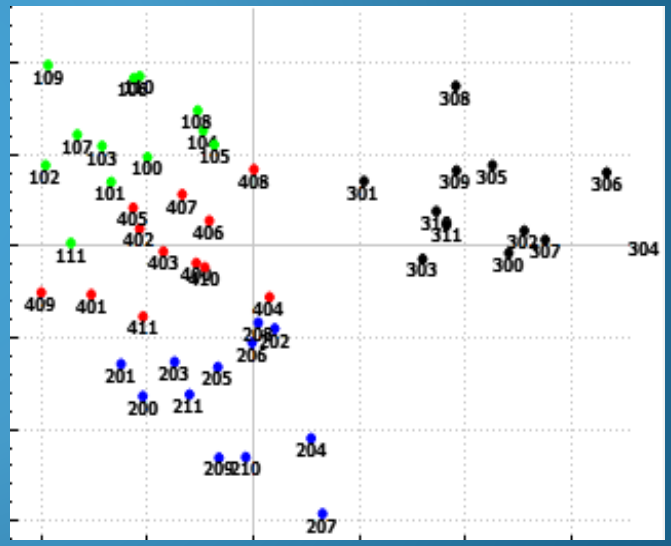






Оптимальні значення мат. моделі





Висновки

Проблема розпізнавання емоційних станів вирішувалася багатьма методами, які з часом покращувалися. Кожен підхід має свої переваги і недоліки, кожен має свої межи оптимальної роботи.

Результатом дипломної роботи є система розпізнавання емоційних станів за допомогою геометричних особливостей обличчя.

В результаті було виконані наступні задачі:

створення та дослідження моделі мімічних проявів емоцій;

дослідження підходів для розпізнавання мімічних проявів емоцій у рамках запропонованої моделі;

розробка інформаційної системи для розпізнавання емоцій за запропонованими моделлю та підходом;

валідація та верифікація моделі, підходу та інформаційної системи.

Для подальшого розвитку програми можна додати ще декілька ключових значень обличчя, сформувати точніший механізм отримання значень, а також алгоритм розпізнавання, який враховує емоційний стан через одиничні особливості.

Дякую за увагу!

Sun Nov 29 14:25:37 EET 2020, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

	Документ		Сумарний збіг по Базі Даних	
ID: 81551 Назва: Моделювання ознак зображення для задач розпізнавання Додано в БД: 2020-11-29 Автора: С.О. Іващенко Керівники: О.В. Бармак Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	48457	474	1112 (2%)	13 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

РІШЕННЯ КАФЕДРИ **КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ** ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: **Моделювання ознак зображення для задач розпізнавання**
Автор: **Іващенко С.О.**
Спеціальність: **122 Комп'ютерні науки**
Науковий керівник: **д.т.н., проф. Бармак О.В.**

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	-
3	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	-
4	Інше:	-

Підтвердження: Виявленні запозичення не є плагіатом т.я. розміщенні в розділах, які не описують безпосередньо авторське дослідження, складають 5% та мають посилання на приведений список літературних джерел

01.11.2020
Дата

Підпис керівника

Підпис завідувача кафедри

Хмельницький національний університет

Кафедра комп'ютерних наук та інформаційних технологій

ВІДГУК ОПОНЕНТА

на дипломну роботу магістра

Магістра *гр. КНм-19-1 Іващенка Сергія Олександровича*

На тему: *Моделювання ознак зображення для задач розпізнавання.*

1. Актуальність і значення теми

З розвитком інформаційних технологій задачі розпізнавання набували все ширшого застосування. Пошук оптимальних рішень в даній сфері триває досі, адже досягнути високих результатів розпізнавання в реальних умовах доволі складно.

Вирішення проблеми роздільності даних потребує низки досліджень пов'язаних з математичною моделлю, методами формування та методами обробки об'єктів дослідження.

Інформаційні технології побудовані на основі задач розпізнавання мають широкий спектр застосувань. Вони можуть бути застосовані, як системи ідентифікації користувачів, контролю стану робітників, дослідження вподобань клієнтів, класифікація медичних даних, тощо.

2.Оцінка якості та достовірності проведених досліджень.

Проведені дослідження в повній мірі співвідносяться з результатами наукових робіт. Достовірність отриманих результатів була підтверджена наявним дослідженням.

3. Оцінка запропонованих заходів та пропозицій, практичної цінності та ефективності.

Запропонований підхід повністю вирішує поставлену задачу. Дослідження проведені в рамках дипломної роботи представляють наукову цінність, а саме дослідження в цілому є ефективним. Результати дослідження можуть бити використані для покращення систем розпізнавання.

4. Загальний висновок та оцінка

Дипломна робота магістра виконана в повному обсязі. Оформлення пояснювальної записки здійснено згідно відповідних норм. Структура, практичність, мета та рішення поставленої задачі відповідають вимогам, що пред'являются до освітньо-кваліфікаційного рівня «магістр», а її автор Іващенко С.О. заслуговує присвоєння кваліфікації магістра з комп'ютерних наук та інформаційних технологій.

Робота заслуговує на оцінку « відмінно ».

Опонент Дихо О.В., д.т.н., проф., зав.каф. ТАМ ХНУ