БАКАЛАВРСЬКА РОБОТА

ДРБ. ІП - 34.00.00.000 ПЗ

Група ІП-23-1К

Галяс Максим

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Галяс Максим Сергійович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка та реалізація системи глибокого навчання у веб-браузерах

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Галяс М.С.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Вовк Роман Богданович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Галясу Максиму Сергійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Розробка та реалізація системи глибокого навчання у веб-браузерах ”

керівник проекту (роботи) Вовк Р.Б., к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області використання глибокого навчання у веб-технологіях
2. Теоретичні основи глибокого навчання та нейронних мереж у задачах розпізнавання облич
3. Моделі глибинного навчання виявлення облич та архітектура розроблюваної системи
4. Архітектура розроблюваної системи глибокого навчання у веб-браузерах
5. Реалізація системи глибокого навчання у веб-браузерах для задач розпізнавання облич

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Системи глибокого навчання моделюються за зразком нейронних мереж (рис. 1.1)
2. Вузли нейронних мереж (рис. 1.2)
3. Базова ідея зворотного поширення (рис. 1.3)
4. Концепціч глибокої розділюваної згортки (рис. 1.4)
5. Архітектура глибокої нейронної мережі типу DenseNet (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області використання глибокого навчання у веб-технологіях	04.05.2025	виконано
2	Теоретичні основи глибокого навчання та нейронних мереж у задачах розпізнавання облич	13.05.2025	виконано
3	Моделі глибинного навчання виявлення облич та архітектура розроблюваної системи	23.05.2025	виконано
4	Архітектура розроблюваної системи глибокого навчання у веб-браузерах	30.05.2025	виконано
5	Реалізація системи глибокого навчання у веб-браузерах для задач розпізнавання облич	04.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 79 сторінок, 39 рисунків, список використаних джерел із 39 найменуваннями, 1 додаток.

Мета роботи - розробити та реалізувати систему розпізнавання облич на основі глибокого навчання, що працює у веб-браузері за допомогою сучасних JavaScript-бібліотек, з використанням попередньо навчених моделей.

Об'єкт дослідження - системи глибокого навчання для розпізнавання облич.

Предмет дослідження - застосування моделей глибокого навчання у веб-браузерах для розпізнавання облич у режимі реального часу.

В першому розділі проведено теоретичний аналіз глибокого навчання та нейромереж, актуальних моделей і бібліотек для реалізації розпізнавання облич у браузері.

В другому розділі визначено архітектуру системи, підібрано моделі виявлення облич і обґрунтовано вибір технологічного стеку для реалізації в веб-середовищі.

В третьому розділі реалізовано систему розпізнавання облич у браузері, яка успішно виконує виявлення, обробку та ідентифікацію облич у режимі реального часу

Висновок: сформовано архітектуру системи на основі технологій Node.js, TensorFlow.js та face-api.js. Реалізовано повний цикл обробки зображень у браузері: від завантаження моделі та обробки відеопотоку до побудови дескрипторів і розпізнавання осіб за евклідовою метрикою

КЛЮЧОВІ СЛОВА: ГЛИБОКЕ НАВЧАННЯ, ВЕБ-БРАУЗЕР, РОЗПІЗНАВАННЯ ОБЛИЧ, TENSORFLOW.JS, FACE-API.JS, SSD MOBILENET V1, MTCNN, TINY FACE DETECTOR.

ANNOTATION

The bachelor's thesis contains 79 pages, 39 figures, a list of used sources with 39 names, 1 appendix.

The purpose of the work is to develop and implement a face recognition system based on deep learning, working in a web browser using modern JavaScript libraries, using pre-trained models.

The object of the study is deep learning systems for face recognition.

The subject of the study is the application of deep learning models in web browsers for real-time face recognition.

The first section provides a theoretical analysis of deep learning and neural networks, current models and libraries for implementing face recognition in a browser.

The second section defines the system architecture, selects face detection models and justifies the choice of a technological stack for implementation in a web environment.

In the third section, a face recognition system in the browser is implemented, which successfully performs detection, processing and identification of faces in real time.

Conclusion: the system architecture is formed based on Node.js, TensorFlow.js and face-api.js technologies. The full cycle of image processing in the browser is implemented: from loading the model and processing the video stream to building descriptors and recognizing faces using the Euclidean metric

KEYWORDS: DEEP LEARNING, WEB BROWSER, FACE RECOGNITION, TENSORFLOW.JS, FACE-API.JS, SSD MOBILENET V1, MTCNN, TINY FACE DETECTOR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CNN – Convolutional Neural Network (Згорткова нейронна мережа)

DOM – Document Object Model (Об'єктна модель документа)

FPS – Frames Per Second (Кадри на секунду)

MTCNN – Multi-task Cascaded Convolutional Networks (Багатозадачні каскадні згорткові мережі)

PCA – Principal Component Analysis (Метод головних компонент)

SSD – Single Shot MultiBox Detector (Однокадровий детектор з багатьма обмежувальними рамками)

WebGL – Web Graphics Library (Веб-графічна бібліотека)

ВСТУП

У сучасну епоху цифрової трансформації зростає потреба в розумних і безпечних веб-застосунках, здатних виконувати складні обчислювальні завдання безпосередньо в браузері. Одним із найбільш перспективних напрямів є інтеграція технологій глибокого навчання у веб-середовище для виконання задач комп'ютерного зору, зокрема — виявлення та розпізнавання облич. Зважаючи на стрімкий розвиток клієнтських технологій, зростання продуктивності пристроїв користувачів, а також актуальні вимоги до приватності персональних даних, реалізація таких інтелектуальних функцій у веб-браузерах є надзвичайно актуальною. Додатково, можливість запуску нейронних мереж без залучення серверної інфраструктури значно знижує затрати на обслуговування та забезпечує високу масштабованість. У цьому контексті розробка ефективної системи глибокого навчання у веб-браузері для задач розпізнавання облич має як наукову, так і прикладну значущість.

Інформаційні технології стрімко розвиваються у напрямку децентралізованих обчислень та інтеграції інтелектуальних функцій безпосередньо в користувацьке середовище. Серед таких функцій особливої популярності набуває розпізнавання облич — ключова технологія, яка використовується у сфері безпеки, контролю доступу, біометричної ідентифікації, маркетингу, відеоаналітики та інтерфейсів людина-машина. Традиційно реалізація таких систем вимагала потужних серверів і спеціалізованого програмного забезпечення, однак сучасні технології дозволяють інтегрувати подібні алгоритми безпосередньо в браузер за допомогою JavaScript-бібліотек і платформ на кшталт TensorFlow.js та face-api.js.

Водночас глибоке навчання, що лежить в основі більшості сучасних систем розпізнавання, демонструє високу ефективність у задачах комп'ютерного зору. Реалізація нейронних мереж у браузерах відкриває нові

можливості для побудови автономних, кросплатформених та захищених систем обробки зображень без надсилання даних на зовнішні сервери.

Актуальність роботи

З розвитком веб-технологій зростає попит на інтерактивні та інтелектуальні рішення, які можуть працювати безпосередньо в браузері без необхідності у серверних обчисленнях. Однією з перспективних галузей є використання глибокого навчання для задач комп'ютерного зору, зокрема розпізнавання облич у реальному часі. Це відкриває нові можливості у сферах безпеки, автентифікації, персоналізації контенту та створення адаптивних інтерфейсів користувача. Оскільки більшість сучасних додатків орієнтовані на веб-платформи, дослідження інтеграції моделей глибокого навчання у браузерне середовище є вкрай актуальним. Додатково, активна розробка JavaScript-бібліотек на кшталт tensorflow.js та face-api.js сприяє впровадженню складних моделей безпосередньо у клієнтське середовище, знижуючи залежність від серверних рішень і забезпечуючи вищу приватність даних користувача.

У цій роботі розглядається розробка повнофункціональної системи, здатної виконувати виявлення та розпізнавання облич у режимі реального часу в браузерному середовищі. На основі аналізу існуючих моделей глибокого навчання було обрано три найбільш релевантні архітектури — SSD MobileNet V1, Tiny Face Detector та MTCNN — які були імплементовані та протестовані у веб-застосунку. Реалізована система демонструє високу ефективність розпізнавання, підтримує інтерактивну взаємодію з користувачем, а також забезпечує захист приватних даних шляхом обробки даних локально.

Таким чином, розробка подібних систем є актуальним напрямом досліджень, що має вагоме практичне значення для подальшої інтеграції технологій штучного інтелекту у повсякденні веб-рішення.

Мета роботи - розробити та реалізувати систему розпізнавання облич на основі глибокого навчання, що працює у веб-браузері за допомогою сучасних JavaScript-бібліотек, з використанням попередньо навчених моделей.

Завдання дослідження:

1. Проаналізувати сучасні методи глибокого навчання для задач розпізнавання облич.

2. Дослідити можливості реалізації нейронних мереж у браузерному середовищі.

3. Обрати та адаптувати моделі глибокого навчання для задачі розпізнавання облич.

4. Розробити архітектуру системи, яка забезпечує виявлення, витягнення ознак і розпізнавання облич.

5. Реалізувати користувацький інтерфейс для інтерактивної роботи з системою.

6. Провести тестування та оцінювання ефективності реалізованої системи.

Об'єкт дослідження - системи глибокого навчання для розпізнавання облич.

Предмет дослідження - застосування моделей глибокого навчання у веб-браузерах для розпізнавання облич у режимі реального часу.

Методи дослідження

- Теоретичний аналіз наукових джерел з глибокого навчання та нейронних мереж.

- Моделювання та експериментальна реалізація моделей у веб-середовищі.

- Порівняльний аналіз ефективності різних моделей розпізнавання облич.

- Програмна реалізація інтерфейсу та логіки обробки відеопотоку в браузері.

Наукова новизна

Розроблено практичну реалізацію системи, що дозволяє використовувати моделі глибокого навчання для розпізнавання облич безпосередньо у веб-браузері, що знижує потребу у серверних обчисленнях та забезпечує захист особистих даних користувачів.

Практичне застосування

Розроблена система може бути використана в освітніх, комерційних і безпекових веб-додатках, де необхідне швидке та приватне розпізнавання користувачів за обличчям без передачі даних на сторонні сервери.

Бакалаврська робота містить 79 сторінок, 39 рисунків, 3 розділи список використаних джерел із 39 найменуваннями, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИКОРИСТАННЯ ГЛИБОКОГО НАВЧАННЯ У ВЕБ-ТЕХНОЛОГІЯХ

1.1. Глибоке навчання у контексті реалізації в браузерному середовищі

Протягом останнього десятиліття глибоке навчання (Deep Learning) зазнало стрімкого розвитку, демонструючи провідні результати в різноманітних сферах, зокрема в комп'ютерному зорі, обробці природної мови, медичній діагностиці та автоматизованому прийнятті рішень. В умовах епохи великих даних (Big Data) процес трансформації масивів інформації у значущі знання є ключовим завданням сучасних обчислювальних систем. Особливу актуальність набуває розробка ефективних алгоритмів глибокого навчання, які можуть функціонувати безпосередньо у веб-браузерах, що відкриває нові можливості для створення інтерактивних, доступних і платформо-незалежних інтелектуальних систем.

У межах цієї роботи основну увагу приділено алгоритмам розпізнавання облич, які вже тривалий час є предметом активних наукових досліджень і демонструють високу ефективність у практичних застосуваннях. Ми проаналізували сучасні підходи на основі глибоких нейронних мереж, зокрема згорткових нейронних мереж (Convolutional Neural Networks, CNN), глибинно роздільних згорток (Depthwise Separable Convolutions) та щільно з'єднаних архітектур (Dense Convolutional Networks, DenseNet), акцентуючи увагу на їх придатності до реалізації у середовищі веб-браузера з використанням таких технологій, як TensorFlow.js та ONNX.js.

З метою надання систематизованого огляду, дослідження було структуровано за типами архітектур глибокого навчання, що використовуються для задач класифікації, детекції та верифікації облич. Для кожного підходу наводяться короткі характеристики, оцінюється його

продуктивність, обчислювальна складність, вимоги до апаратного забезпечення, а також ступінь адаптації до веб-середовища. Окрему увагу приділено оптимізаційним технікам, що дозволяють зменшити розмір моделі та прискорити її інференс без значної втрати точності, що є критично важливим для інтерактивних застосунків у браузері.

Ця робота має на меті не лише узагальнення поточних досягнень у сфері глибокого навчання, а й формування теоретичного та практичного підґрунтя для подальших досліджень, зокрема проєктів у галузі комп'ютерної інженерії та прикладної інформатики. Представлений аналіз сприяє розумінню можливостей та обмежень сучасних браузерних платформ для реалізації інтелектуальних систем на основі глибокого навчання, відкриваючи шлях до розробки нових інноваційних застосунків, доступних з будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення.

1.2. Постановка завдання дипломної роботи щодо застосування глибокого навчання для браузерного розпізнавання облич

Зі стрімким розвитком цифрового суспільства та постійно зростаючою потребою в ефективних засобах автоматичної ідентифікації особи біометричні технології на сьогодні є одним із ключових напрямів у сфері інформаційної безпеки. З безперервним розширенням суспільства та негайною потребою у швидкій та ефективній автоматичній перевірці особистості, біометрична технологія швидко розвивалася в останні роки. Серед них розпізнавання обличчя посідає одне з провідних місць завдяки своїй ненав'язливості, зручності використання та широкому спектру застосувань — від контролю доступу до аналітики відеоспостереження, цифрової ідентифікації та форензіки.

Значне підвищення точності систем розпізнавання облич стало можливим завдяки стрімкому розвитку методів машинного навчання, особливо глибокого навчання, яке за останні роки перетворилося на провідну технологію в обробці зображень та відео. Від своїх витоків у 1960-х роках до сучасних архітектур глибоких нейронних мереж, включаючи згорткові та резидуальні мережі, еволюція глибокого навчання стала потужним рушієм інновацій у комп'ютерному зорі.

1.2.1. Мета і цілі роботи

Метою даної роботи є аналіз теоретичних засад і практичних аспектів реалізації системи розпізнавання обличчя на основі глибокого навчання у веб-браузерах, а також оцінка ефективності таких систем для використання в реальному часі. Робота охоплює ключові напрями розвитку, методологічні підходи та перспективи застосування вбудованих обчислювальних ресурсів для досягнення максимальної продуктивності.

У межах роботи ставляться такі основні цілі:

- Розробити ефективну систему розпізнавання обличчя, орієнтовану на запуск у веб-додатку з урахуванням часових обмежень, пов'язаних із виконанням в режимі реального часу, з подальшою оптимізацією для вбудованих платформ.

- Провести порівняльний аналіз сучасних моделей розпізнавання обличчя за такими критеріями, як точність, швидкість обробки та обчислювальні ресурси, необхідні для роботи у браузері.

- Застосувати глибокі нейронні мережі для побудови дескрипторів облич, що є векторним представленням характерних ознак обличчя, які використовуються для порівняння та ідентифікації.

- Реалізувати алгоритм зіставлення обличчя шляхом обчислення евклідової відстані між дескрипторами, з урахуванням порогового значення для визначення схожості.

1.2.2. Застосування та практичне використання пропонованого рішення

Системи розпізнавання обличчя на основі глибокого навчання демонструють високу ефективність у різних практичних сценаріях, зокрема:

- у пристроях реального часу, таких як цифрові камери, смартфони або веб-додатки для відеоконференцій, де необхідна швидка обробка та ідентифікація користувачів.

- у задачах розпізнавання шаблонів на відео- та фотозображеннях, включаючи ідентифікацію осіб, транспортних засобів, виявлення аномалій у медичних зображеннях, а також виявлення об’єктів у великих масивах даних.

- у багатокористувацьких системах розпізнавання облич (Multi-Face Recognition, MFR), що є критично важливими для таких галузей, як банківська безпека, контроль доступу в аеропортах, та в системах громадської безпеки.

- у спеціалізованих галузях, таких як криміналістика, де технологія розпізнавання обличчя застосовується для верифікації особи, виявлення підозрюваних та ідентифікації в архівах відеодоказів.

Дана дипломна робота зосереджена на реалізації систем глибокого навчання у браузерному середовищі, робить вагомий внесок у розуміння технічних і прикладних аспектів впровадження інтелектуальних систем у доступні, масштабовані та кросплатформенні рішення майбутнього.

1.3. Теоретичні основи глибокого навчання та нейронних мереж у задачах розпізнавання облич

Цей розділ присвячено ключовим концепціям, необхідним для розуміння принципів роботи сучасних систем розпізнавання облич. Особлива увага приділяється архітектурам глибокого навчання, нейронним мережам, а також алгоритмам навчання, які лежать в основі цих підходів. Наведені

концепції становлять теоретичну базу для подальшої реалізації систем глибокого навчання у веб-середовищі.

1.3.1. Глибоке навчання

Глибоке навчання (Deep Learning) — це підгалузь машинного навчання, яка оперує багаторівневими нейронними архітектурами для вивчення представлень даних на різних рівнях абстракції. Його фундаментальна ідея ґрунтується на біологічних принципах роботи людського мозку, зокрема на здатності обчислювальних одиниць (аналогів нейронів) адаптивно змінювати свої параметри на основі отриманих вхідних сигналів.

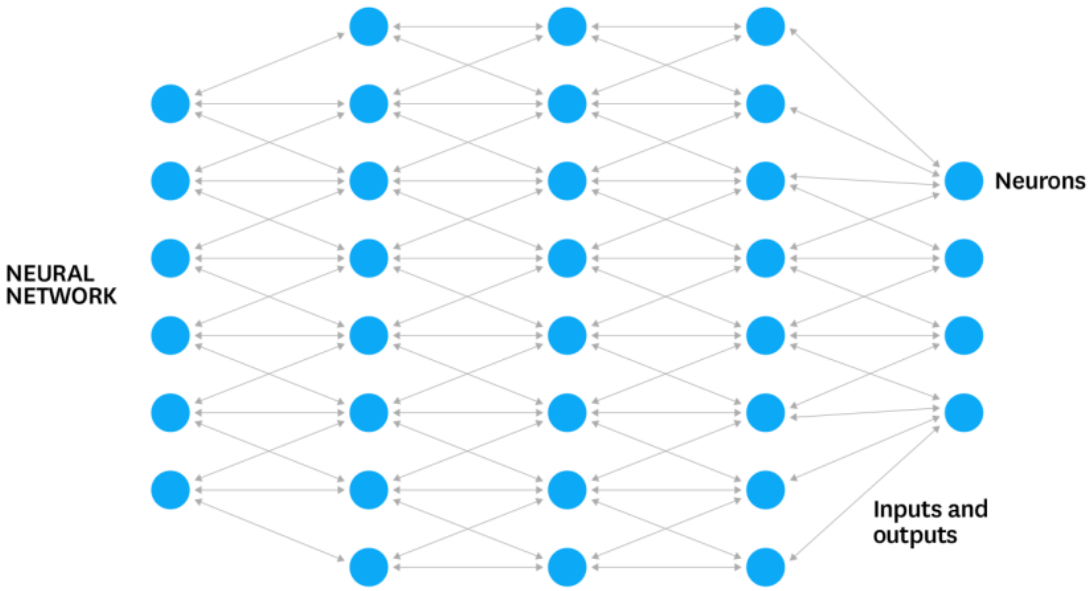

Рисунок 1.1 - Системи глибокого навчання моделюються за зразком нейронних мереж

Системи глибокого навчання складаються з багатьох рівнів нелінійних перетворень, що дозволяють моделювати складні залежності між вхідними та вихідними даними. Дані, як правило, подаються у вигляді багатовимірних

структур — матриць або тензорів, які передаються через послідовні шари мережі. На кожному етапі ці шари виконують виявлення й узагальнення ознак, починаючи з базових (наприклад, контури чи грані) до високорівневих (наприклад, структура обличчя чи емоційні вирази).

Глибоке навчання значно розширило можливості комп’ютерного зору, зокрема в задачах класифікації, сегментації, детекції об’єктів та розпізнавання облич. Завдяки глибоким згортковим нейронним мережам (CNN), стало можливим досягати високої точності без потреби в ручному витягу ознак, що традиційно вимагалося у класичних підходах до машинного навчання.

1.3.2. Нейронні мережі

Нейронні мережі (Artificial Neural Networks, ANN) — це обчислювальні моделі, що імітують структуру та функціонування біологічного мозку. Вони складаються з вузлів (нейронів), об'єднаних у шари: вхідний, приховані та вихідний. Основне призначення таких мереж — розпізнавання шаблонів та закономірностей у даних, представлених у числовій формі.

На початковому етапі розвитку нейронних мереж (1950–1970-ті роки) моделі були обмежені за складністю й глибиною, що істотно обмежувало їх практичне застосування. Однією з важливих концепцій, що призвела до прориву, стало використання прихованих шарів, які дозволили моделювати складні функції через багатокрокову обробку вхідних даних. Кожен прихований шар трансформує дані, виявляючи більш складні особливості, на основі яких мережа формує остаточне рішення.

Наприклад, у задачах розпізнавання облич перші шари мережі можуть фіксувати прості геометричні фрагменти, такі як краї, кути, текстури, тоді як наступні — об’єднують їх у вищі абстракції, такі як очі, ніс, рот або форма обличчя. Такий підхід забезпечує ефективне та узагальнене представлення

зображень, що значно покращує точність розпізнавання навіть за наявності шуму, часткових перекриттів або зміни освітлення.

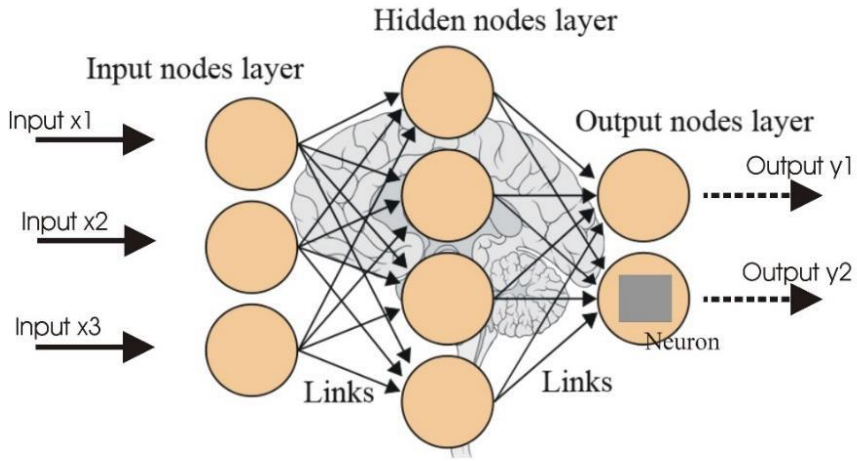

Рисунок 1.2 - Вузли нейронних мереж

У минулому ефективність машинного навчання значною мірою залежала від випереджального витягу ознак (feature extraction), який виконувався вручну. Проте сучасні нейронні мережі здатні самостійно вивчати релевантні ознаки без необхідності в ручному втручанні, що стало можливо завдяки покращеним алгоритмам навчання та збільшенню обчислювальної потужності.

1.3.3 Алгоритм зворотного поширення помилки

Один із ключових чинників успіху багатошарових нейронних мереж — алгоритм зворотного поширення помилки (Backpropagation), який дозволяє ефективно оновлювати вагові коефіцієнти мережі в процесі навчання. Ідея полягає в обчисленні градієнта функції втрат по відношенню до кожного параметра мережі та коригуванні ваг за допомогою градієнтного спуску.

Перші концептуальні підходи до зворотного поширення були сформульовані у 1960-х роках, зокрема у роботах Сеппо Ліннайнмаа. У 1974 році Пол Вербос у своїй докторській дисертації запропонував використання

цього методу для навчання нейронних мереж, відкривши шлях до побудови ефективних багатошарових моделей. Його ідеї базувалися на прагненні моделювати людський інтелект, що заклало основу для подальших досліджень у сфері штучного інтелекту.

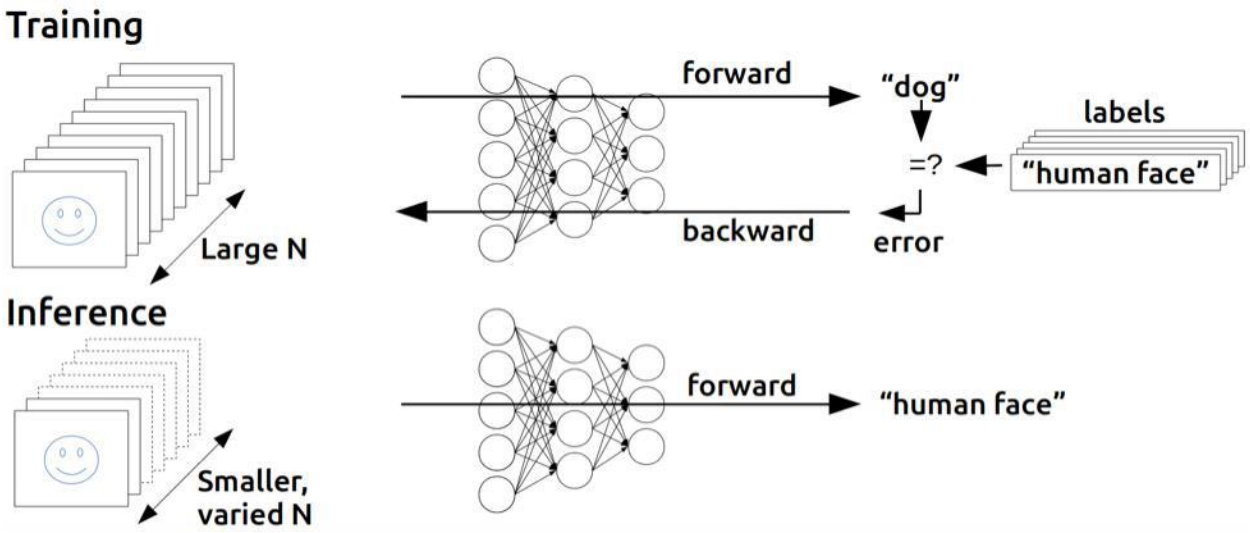

Рисунок 1.3 - Базова ідея зворотного поширення

В загальному, рисунок 1.3 ілюструє процеси навчання та висновку в нейронній мережі.

Етап Навчання (Training):

- У лівій частині етапу навчання зображено стопку зображень, що символізують великий набір даних (Large N), який використовується для тренування моделі.

- У центральній частині показано архітектуру нейронної мережі, що складається з кількох шарів взаємопов'язаних вузлів (нейронів).

- Стрілка "forward" (вперед) вказує напрямок проходження даних через мережу під час навчання. Вхідні дані (зображення) подаються на вхідний шар, обробляються прихованими шарами, і мережа генерує вихідний результат (наприклад, передбачення "dog").

- Вихідний результат порівнюється з правильними мітками (labels) (наприклад, "human face").

- Символ "=" зі знаком питання та стрілка, що йде до "error" (помилка), показують процес порівняння передбачення мережі з істинною міткою та обчислення помилки.

- Стрілка "backward" (назад) ілюструє процес зворотного поширення помилки. Обчислена помилка використовується для коригування ваг зв'язків у мережі, щоб зменшити помилку при наступних ітераціях. Цей процес повторюється багато разів на великому наборі даних.

Етап Висновку (Inference):

- У лівій частині етапу висновку також зображено зображення, але стопка менша і позначена як "Smaller; varied N", що символізує нові, раніше не бачені дані, на яких мережа має зробити передбачення.

- Центральна частина знову показує навчену нейронну мережу.

- Стрілка "forward" (вперед) вказує на проходження нових даних через навчену мережу. На цьому етапі мережа вже не коригує свої ваги.

- Вихідний результат є передбаченням мережі для нових вхідних даних (наприклад, "human face").

Отже, рисунок 1.3 візуалізує:

- Навчання - процес подачі великого обсягу даних з відомими мітками через мережу, обчислення помилки та коригування внутрішніх параметрів мережі (ваг) за допомогою зворотного поширення.

- Висновок - процес використання навченої мережі для обробки нових даних та генерації передбачень без подальшого коригування ваг.

1.4. Глибинно роздільні згортки та щільно з’єднані згорткові мережі

1.4.1. Глибинно роздільні згортки

Глибинно роздільні згортки (depthwise separable convolutions) є оптимізованим варіантом класичних згорткових операцій у згорткових

нейронних мережах (CNN), який значно зменшує обчислювальну складність моделі при незначній втраті точності. Ця ідея ґрунтується на припущенні, що просторові (ширина та висота зображення) та глибинні (кількість каналів) виміри можна обробляти окремо, що і дало назву цьому типу згортки.

Замість того, щоб виконувати згортку над усіма каналами одночасно, як у традиційних CNN, глибинно роздільна згортка виконує два окремі кроки:

1. Глибинна згортка (depthwise convolution) — застосовується окремо до кожного каналу вхідного зображення.

2. Згортка точка-за-точкою (pointwise convolution) — виконується згортка 1×1, що об'єднує інформацію з усіх каналів.

Таке розділення призводить до значного скорочення кількості параметрів і обчислень, що особливо корисно при розробці моделей для пристроїв з обмеженими ресурсами, включаючи мобільні та вбудовані системи, а також для розгортання у веб-браузерах. Однією з найбільш відомих архітектур, що використовують глибинно роздільні згортки, є MobileNet, яка показала ефективність у задачах класифікації, детекції облич та інших задачах комп’ютерного зору.

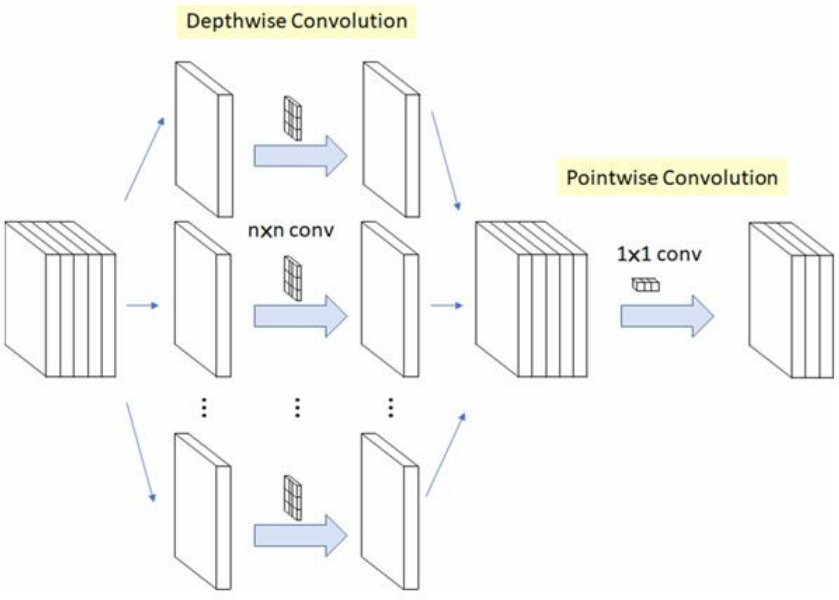

Рисунок 1.4 - Концепціч глибокої розділюваної згортки

На рисунку 1.4 зображено концепцію глибокої розділюваної згортки (Depthwise Separable Convolution), яка є ефективним методом для зменшення обчислювальної складності згорткових нейронних мереж, особливо в мобільних та вбудованих системах.

Ця операція розділена на два етапи:

1. Поканальна згортка (Depthwise Convolution):

- У лівій частині рисунка показано вхідний тензор даних (наприклад, зображення з кількома каналами).

- Цей етап передбачає застосування окремого просторового фільтра (розміром n x n, де n - розмір ядра) до кожного вхідного каналу незалежно.

- На рисунку це проілюстровано тим, що кожен вхідний шар (канал) обробляється окремим ядром n x n conv, генеруючи вихідний шар з тією ж кількістю каналів, що й вхідний.

2. Точкова згортка (Pointwise Convolution):

- Результати поканальної згортки об'єднуються і подаються на вхід другого етапу.

- На цьому етапі застосовується згортка ядром розміром 1x1 до вихідних даних поканальної згортки.

- Ядро 1x1 згортки проходить через усі канали вихідного тензора поканальної згортки, обчислюючи лінійну комбінацію вихідних карт ознак з попереднього етапу.

- Це дозволяє створити нові комбінації ознак між каналами.

Глибока розділювана згортка замінює традиційну згортку, яка одночасно виконує просторову та канальну фільтрацію, на два окремі кроки. Спочатку виконується просторова фільтрація для кожного каналу окремо (поканальна згортка), а потім виконується комбінування каналів за допомогою згортки 1x1 (точкова згортка).

Такий підхід значно зменшує кількість параметрів та обчислювальну складність порівняно зі стандартною згорткою, зберігаючи при цьому значну

частину ефективності. Це робить глибокі розділювані згортки популярними в архітектурах ефективних нейронних мереж (наприклад, MobileNet).

Через свою високу обчислювальну ефективність глибинно роздільні згортки отримали широке поширення в наукових дослідженнях і практичних застосуваннях, пов'язаних із розпізнаванням образів у режимі реального часу, зокрема в браузерних середовищах, де обмеження обчислювальних ресурсів є критичними.

1.4.2. Щільно з'єднані згорткові мережі

Щільно з'єднані згорткові мережі (Dense Convolutional Networks, або DenseNet) є ще однією важливою архітектурною інновацією у сфері глибокого навчання. Основна ідея DenseNet полягає у введенні щільних з'єднань між шарами, де кожен шар приймає на вхід не лише вихід попереднього шару, але й усі виходи з попередніх шарів у мережі.

Така структура сприяє:

- кращому поширенню градієнтів у процесі навчання;
- зменшенню ризику зникнення градієнта (vanishing gradient problem);
- ефективнішому повторному використанню ознак;
- зменшенню кількості параметрів за рахунок відсутності потреби в дублюванні інформації.

DenseNet демонструє високу ефективність у задачах розпізнавання образів, зокрема — облич, де точність, надійність і глибина мережі є критичними факторами. Вона дозволяє моделі вивчати складніші та глибші представлення зображень, що підвищує загальну якість розпізнавання, навіть за умов обмеженої навчальної вибірки або складного фону.

З погляду біологічної інспірації, DenseNet можна порівняти з організацією нейронних зв'язків у головному мозку, де міжнейронні взаємодії не обмежуються лише послідовними рівнями, а є щільно

переплетеними, що дозволяє реалізовувати складну поведінку навіть на базі відносно простих елементів.

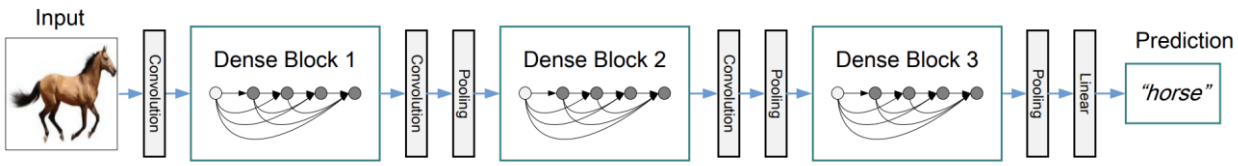

Рисунок 1.5 - Архітектура глибокої нейронної мережі типу DenseNet

На рисунку 1.5 зображено архітектуру глибокої нейронної мережі типу DenseNet, яка складається з кількох щільних блоків (Dense Blocks).

Розглянемо основні компоненти архітектури:

1. Input (Вхід): Початкове зображення (у даному випадку, зображення коня), яке подається на вхід мережі.

2. Convolution (Згортка): Початковий згортковий шар, який обробляє вхідне зображення перед подачею його до першого щільного блоку.

3. Dense Blocks (Щільні Блоки): Це основні будівельні блоки DenseNet. У межах кожного щільного блоку кожен шар отримує на вхід карти ознак від усіх попередніх шарів у цьому ж блоці. Це сприяє повторному використанню ознак та зменшує проблему зникаючого градієнта. На рисунку кожен блок показано як послідовність вузлів (шарів), де вихід кожного вузла з'єднується з усіма наступними вузлами в блоці.

4. Transition Layers (Перехідні Шари): Шари, розташовані між щільними блоками. Вони виконують операції згортки (Convolution) та пулінгу (Pooling) для зменшення розмірності карт ознак та кількості каналів, готуючи дані для наступного щільного блоку. На рисунку вони позначені як блоки "Convolution" та "Pooling" між щільними блоками.

5. Linear (Лінійний шар): Повнозв'язний шар, який зазвичай розташовується наприкінці мережі після останнього щільного блоку та перехідних шарів. Він відповідає за формування остаточного передбачення.

6. Prediction (Передбачення): Вихід мережі, що є результатом класифікації вхідного зображення (у даному випадку, передбачення "horse" - кінь).

Особливості DenseNet, проілюстровані на рисунку 1.5:

- З'єднання всіх зі всіма в блоці: На рисунку видно, як вихід кожного шару в щільному блоці (представлені як вузли) з'єднується з усіма наступними шарами в тому ж блоці. Це відрізняє DenseNet від традиційних згорткових мереж, де кожен шар отримує вхід лише від попереднього шару.

- Перехідні шари для зміни розмірності: Показано, що між щільними блоками знаходяться шари, які зменшують розмірність даних перед подачею їх до наступного блоку.

У контексті веб-орієнтованих систем розпізнавання облич такі архітектури, як DenseNet, можуть бути адаптовані до браузерного середовища через використання легковагових варіацій або шляхом попереднього навчання моделі на потужних серверах з подальшим розгортанням оптимізованої версії у клієнтському оточенні.

РОЗДІЛ 2. МОДЕЛІ ГЛИБИННОГО НАВЧАННЯ ВИЯВЛЕННЯ ОБЛИЧ ТА АРХІТЕКТУРА РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1. Моделі виявлення облич

Метою даної роботи є здійснення виявлення облич на основі методів глибокого навчання. З метою спрощення реалізації передбачається використання попередньо навчених нейронних мереж, які будуть детально описані у подальших розділах. Для забезпечення можливості розпізнавання конкретних осіб здійснюється завантаження одного або кількох зображень кожного суб'єкта, що виконуватимуть роль еталонних даних. Надалі здійснюється порівняння вхідного зображення з цими еталонними зразками. Крім того, передбачено можливість оцінювання схожості, а також визначення статі, віку та емоційного стану об’єкта.

Система, реалізована на платформі Node.js, передбачає використання трьох моделей для виявлення облич, першою з яких є SSD MobileNet V1.

2.1.1. Модель SSD MobileNet V1

Однією з ключових моделей для детектування облич у розробленій системі є SSD (Single Shot Multibox Detector), побудована на основі згорткової нейронної мережі MobileNet V1. Ця архітектура поєднує високу ефективність із помірними обчислювальними витратами, що робить її придатною для використання в системах реального часу на обмежених апаратних ресурсах.

На відміну від базової структури MobileNet, модель SSD включає додаткові шари для передбачення обмежувальних рамок (bounding boxes), які накладаються поверх основної мережі. Завдяки цим додатковим шарам мережа здатна ідентифікувати місця розташування облич на зображенні та повертати координати обмежувальних рамок разом із відповідними оцінками

ймовірності. Особливістю даної моделі є орієнтація на досягнення високої точності у локалізації облич, навіть за умов складного фону, варіацій освітлення або поз.

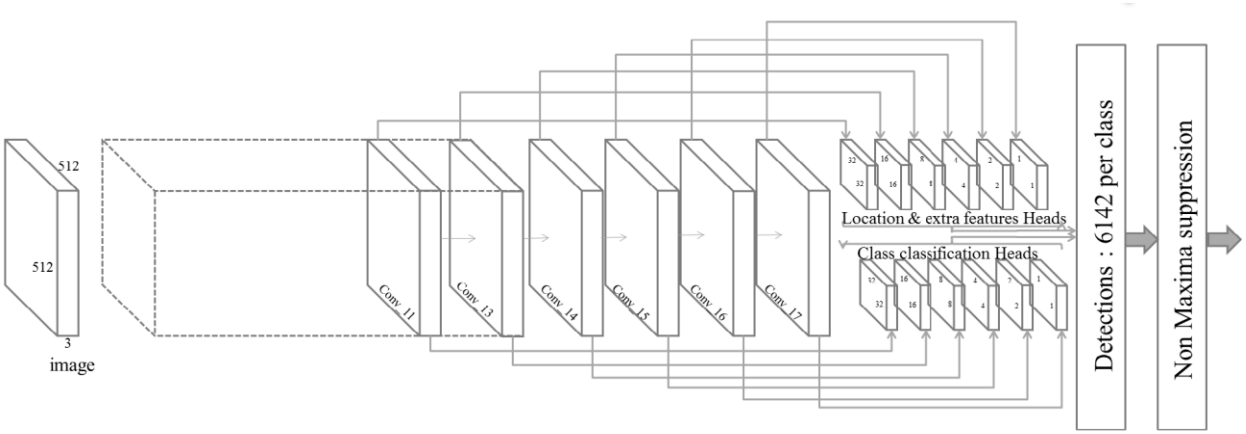

Рисунок 2.1 – Архітектура Single Shot MultiBox Detector

На рисунку 2.1 подана архітектура SSD (Single Shot MultiBox Detector), призначена для виявлення об'єктів. Наведемо детальний опис її компонентів та потоку даних:

1. Вхідний шар (Input Layer)

Мережа приймає на вхід зображення розміром 512x512 пікселів з 3 колірними каналами (RGB).

2. Основна мережа (Backbone Network)

Вона складається з послідовності згорткових блоків, позначених як Conv. 11, Conv. 13, Conv. 14, Conv. 15, Conv. 16, Conv. 17. Це згорткові шари, які витягують ієрархічні ознаки з вхідного зображення. Номери шарів можуть вказувати на походження цих блоків, наприклад, з мережі MobileNet V1, як було зазначено в метаданих зображення.

Напрямок стрілок вказує на послідовну обробку даних від одного згорткового шару до іншого, зменшуючи просторові розміри та збільшуючи кількість каналів.

3. Витягування ознак для детектування (Feature Extraction for Detection)

Ознакові карти для детектування витягуються з різних етапів основної мережі. Стрілки від Conv. 11, Conv. 13, Conv. 14, Conv. 15, Conv. 16 та Conv. 17 вказують на те, що багаторівневі ознаки використовуються для виявлення об'єктів. Це дозволяє мережі детектувати об'єкти різних масштабів: великі об'єкти на ранніх (більш високих роздільних здатність) ознакових картах, а малі об'єкти на пізніших (менших роздільних здатність) ознакових картах.

4. Голови детектування (Detection Heads)

З кожної витягнутої ознакової карти відгалужуються дві паралельні "голови":

- Location & extra features Heads (Голови регресії місцезнаходження та додаткових ознак):

Ці голови відповідають за регресію координат обмежувальних рамок.

Кожна голова, схоже, складається з кількох згорткових шарів, що зменшують кількість каналів (наприклад, 32, 16, 8, 4, 2, 1). Ці шари обробляють ознакову карту, щоб передбачити точне місцезнаходження об'єктів. У контексті попереднього опису, замість 4 вихідних каналів (x,y, width, height), тут може використовуватися 8 вихідних каналів для (x,y) чотирьох кутів.

- Class classification Heads (Голови класифікації класів):

Ці голови паралельно відповідають за класифікацію об'єктів, тобто визначення, до якого класу належить виявлений об'єкт (наприклад, "автомобіль", "пішохід", "паркувальне місце" тощо).

Вони також складаються з послідовності згорткових шарів, що обробляють ту ж ознакову карту, що й голови регресії, для передбачення ймовірності належності до кожного класу.

5. Результати детекції (Detections)

Вихідні дані з усіх "Location & extra features Heads" та "Class classification Heads" об'єднуються, формуючи загальну кількість 6142 детекції

на клас. Це означає, що мережа генерує до 6142 потенційних обмежувальних рамок разом з їхніми відповідними оцінками впевненості для кожного класу.

6. Придушення немаксимумів (Non Maxima Suppression - NMS)

Нарешті, застосовується етап Non Maxima Suppression (NMS). Цей пост-процесинг є критично важливим для видалення дубльованих детекцій. NMS обирає найвпевненішу обмежувальну рамку серед тих, що сильно перекриваються та позначають один і той же об'єкт, тим самим забезпечуючи, що кожен об'єкт виявляється лише один раз.

SSD MobileNet V1 була попередньо навчена на відкритому датасеті WIDER FACE, який містить широкий спектр зображень облич у різноманітних умовах (кут зору, розмір, положення тощо), що забезпечує хорошу узагальнювальну здатність моделі. Завдяки цьому модель є ефективною для задач виявлення облич у реальних сценаріях, зокрема для побудови систем розпізнавання, моніторингу або інтерпретації емоційних станів.

2.1.2. Модель Tiny Face Detector

Tiny Face Detector є однією з моделей, реалізованих у складі API для розпізнавання облич, та становить собою високоефективний і оптимізований детектор облич, орієнтований на швидкодію і низьке споживання ресурсів. На відміну від традиційних згорткових нейронних мереж, дана модель використовує глибинно роздільні згортки (depthwise separable convolutions), що дозволяє істотно зменшити обчислювальні витрати та прискорити процес виявлення.

Хоча Tiny Face Detector поступається моделі SSD MobileNet V1 за точністю, він демонструє значно вищу продуктивність у контексті швидкості обробки, компактності та ефективного використання ресурсів. Це робить його особливо придатним для використання у веб-застосунках та мобільних пристроях, де обмежені обчислювальні потужності та обсяг пам’яті.

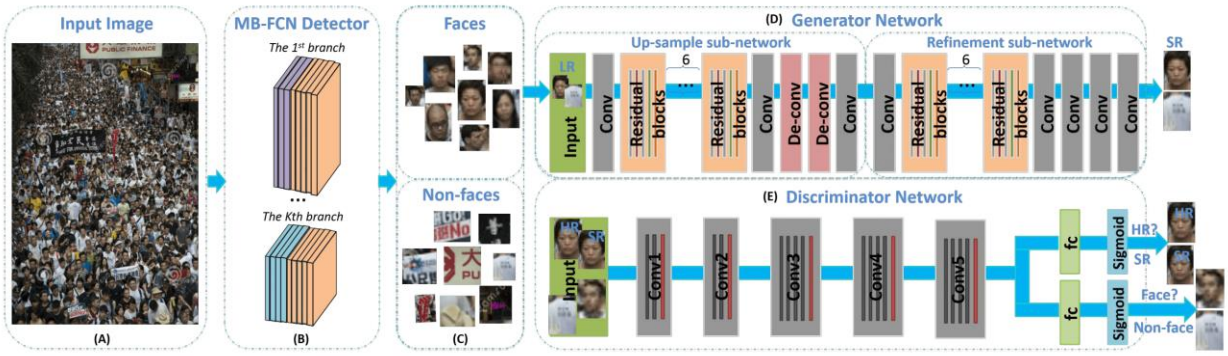

Рисунок 2.2 – Архітектура системи призначеної для виявлення крихітних облич та їх подальшого розширення

Ця архітектура, як показано на рис. 2.2, є гібридною системою, призначеною для виявлення крихітних облич та їх подальшого суперрозширення (збільшення роздільної здатності) за допомогою генеративно-змагальної мережі (GAN). Опишемо її.

1. Вхідне зображення (Input Image) (A)

Система приймає на вхід велике зображення (A), яке може містити безліч людей, включаючи потенційно дуже маленькі та розмиті обличчя.

2. Детектор MB-FCN (Multi-Branch Fully Convolutional Network Detector) (B)

Це перший етап системи. Детектор MB-FCN (B) обробляє вхідне зображення для ідентифікації регіонів, що містять обличчя.

Він використовує кілька гілок (наприклад, "1st branch", "Kth branch"), що дозволяє йому працювати на різних масштабах і виявляти обличчя різного розміру та положення на зображенні.

Результатом роботи детектора є виділення численних патчів (фрагментів) з зображення, які класифікуються як "обличчя" або "не-обличчя".

3. Вихід детектора: Обличчя та Не-обличчя (C)

Обличчя (Faces): Це виявлені фрагменти зображення, які, імовірно, містять обличчя. Ці патчі часто є малими та розмитими, що є вихідною

"проблемою" для їх суперрозширення. Саме ці "обличчя" передаються в Генераторну мережу.

Не-обличчя (Non-faces): Це інші фрагменти зображення, які не були ідентифіковані як обличчя.

4. Генераторна мережа (Generator Network) (D)

Ця мережа відповідає за перетворення малих, розмитих патчів облич у чіткі обличчя високої роздільної здатності. Вона складається з двох основних підмереж:

1) Підмережа для підвищення роздільної здатності (Up-sample sub-network):

- Приймає на вхід виявлені "обличчя" (які є низької роздільної здатності).

- Містить згорткові шари (Conv), кілька блоків залишків (Residual blocks) (показано 6) для збереження інформації та запобігання зникненню градієнтів, а також дезгорткові шари (De-conv) для підвищення просторової роздільної здатності зображення.

2) Підмережа уточнення (Refinement sub-network):

- Отримує вихідні дані з підмережі підвищення роздільної здатності.

- Містить додаткові згорткові шари та блоки залишків (також 6) для подальшого уточнення деталей, зменшення артефактів та покращення візуальної якості зображення.

На виході генератор генерує суперрозширене (SR) обличчя, яке має бути чітким і мати високу роздільну здатність.

5. Дискримінаторна мережа (Discriminator Network) (E)

Роль дискримінатора полягає у відрізненні справжніх облич високої роздільної здатності (HR) від згенерованих (SR) Генератором облич.

Це ключовий компонент GAN-архітектури, що забезпечує реалістичність згенерованих зображень.

Він приймає як справжні зображення облич високої роздільної здатності, так і згенеровані генератором суперрозширені обличчя.

Архітектура складається з послідовності згорткових шарів (Conv1-Conv4), які витягують ознаки із вхідного зображення. За ними йдуть повнозв'язні шари (fc).

Завершується шаром Sigmoid, який видає ймовірність того, що вхідне зображення є "справжнім" (високої роздільної здатності). Чим ближче значення до 1, тим більше Дискримінатор вважає зображення справжнім.

Ця архітектура працює як генеративно-змагальна система:

1. Детектор MB-FCN знаходить потенційні обличчя на великому зображенні.

2. Ці малі, розмиті патчі облич подаються на вхід генератору.

3. Генератор намагається створити реалістичні, високоякісні версії цих облич.

4. Дискримінатор навчається розрізняти справжні високоякісні обличчя від тих, що були згенеровані генератором.

5. Під час навчання генератор намагається "обманути" Дискримінатора, створюючи все більш реалістичні зображення, тоді як Дискримінатор намагається покращити свою здатність розрізняти справжнє від підробленого. Цей змагальний процес призводить до того, що генератор з часом виробляє дуже якісні та реалістичні суперрозширені обличчя.

Модель була навчена на спеціалізованому наборі даних, що містить понад 14 тисяч зображень, на яких вручну позначено обмежувальні рамки облич.

Крім того, модель було додатково оптимізовано для визначення орієнтирів обличчя (facial landmarks), тобто ключових точок (очі, ніс, рот тощо), що дозволяє їй досягати вищої точності в завданнях локалізації рис обличчя порівняно з SSD MobileNet V1.

Таким чином, Tiny Face Detector виступає як ефективне рішення для задач реального часу, де критичними є швидкість виконання та економія апаратних ресурсів, при збереженні прийнятного рівня точності.

2.1.3. Модель виявлення облич MTCNN (Multi-task Cascaded Convolutional Neural Networks)

MTCNN (Multi-task Cascaded Convolutional Neural Networks) — ще одна модель для виявлення облич, реалізована в межах системи переважно для експериментальних досліджень. Ця архітектура являє собою багатозадачну каскадну згорткову нейронну мережу, яка забезпечує одночасне виявлення облич та визначення ключових орієнтирів.

Модель MTCNN складається з трьох послідовно з'єднаних згорткових підмереж, кожна з яких виконує конкретний етап обробки: початкове виявлення потенційних облич, уточнення меж обмежувальних рамок та високоточне локалізування разом із виявленням п'яти ключових точок обличчя (очі, ніс, кути рота). Завдяки каскадній структурі модель здатна детектувати обличчя різного масштабу та в умовах часткового перекриття чи складного фону.

На виході MTCNN повертає координати обмежувальних рамок для кожного знайденого обличчя, а також відповідні оцінки ймовірності належності (у діапазоні від 0.00 до 1.00). Ці оцінки можуть використовуватися як порогові значення для фільтрації результатів, що дозволяє підвищити надійність системи розпізнавання.

В [12] запропонували одну з перших глибоких моделей для детекції облич, засновану на каскаді згорткових нейронних мереж. Запропонований каскад CNN працює з кількома роздільними здатностями, швидко відхиляючи фонові ділянки на швидких етапах низької роздільної здатності та ретельно оцінюючи невелику кількість кандидатів на останньому етапі високої роздільної здатності. Щоб покращити ефективність локалізації та

зменшити кількість кандидатів на пізніших етапах, вони впровадили етап калібрування на основі CNN після кожного з етапів детекції в каскаді. Запропонований метод працює зі швидкістю 14 кадрів/с на одному ядрі CPU для зображень з роздільною здатністю VGA та 100 кадрів/с з використанням GPU. Архітектура 12-шарової мережі CNN-Cascade показана на рисунку 2.3. Вона призначена для бінарної класифікації, зокрема для визначення, чи містить вхідне зображення обличчя чи ні. Ця компактна архітектура є частиною більшої каскадної системи для детекції облич, як це часто буває в моделях, що працюють з різними роздільними здатностями та етапами відхилення.

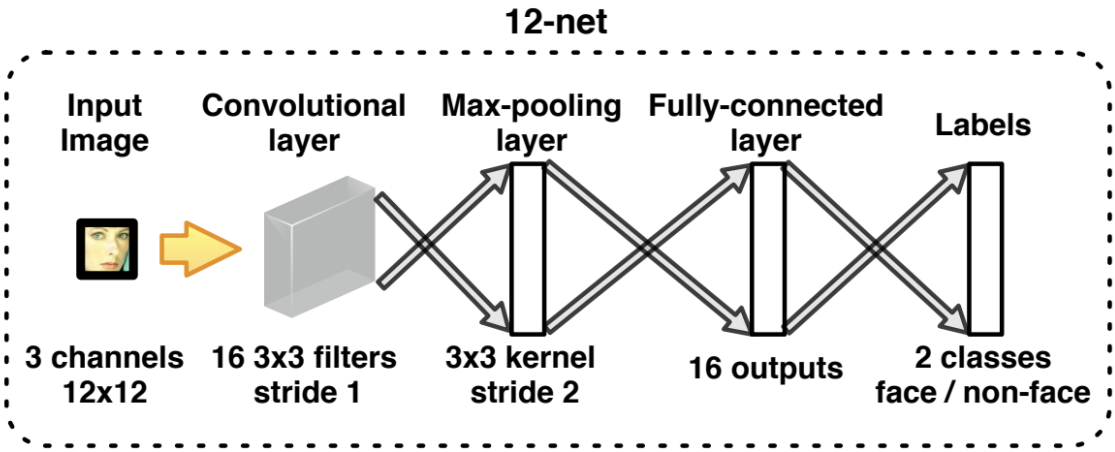

Рисунок 2.3 - Архітектура 12-шарової моделі Cascade-CNN

Мережа приймає на вхід зображення розміром 12x12 пікселів з 3 каналами (RGB). Першим шаром є згортковий шар. Він використовує 16 фільтрів розміром 3x3 пікселі. Крок (stride) застосування фільтра становить 1. Цей шар відповідає за витягування базових ознак з вхідного зображення.

Після згорткового шару йде шар макс-пулінгу. Він використовує ядро (kernel) розміром 3x3 пікселі. Крок (stride) застосування пулінгу становить 2, що призводить до зменшення просторових розмірів ознакових карт.

Далі йде повнозв'язний шар. Він має 16 вихідних нейронів. Цей шар агрегує ознаки, витягнуті попередніми згортковими та пулінговими шарами, готуючи їх до класифікації.

Останній шар генерує вихід для 2 класів: "обличчя" або "не-обличчя". Це бінарна класифікація, яка вказує на присутність чи відсутність обличчя на вхідному зображенні.

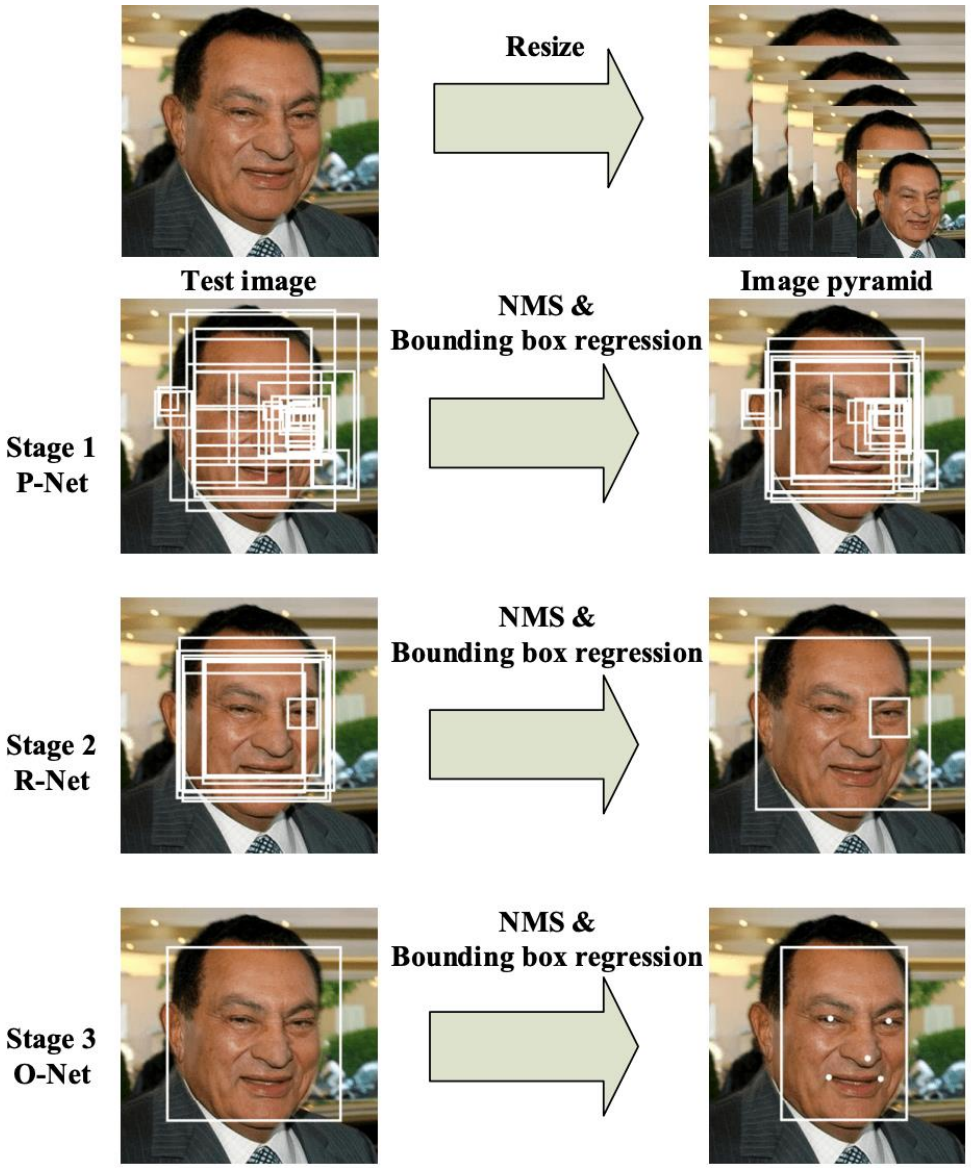

Рисунок 2.4 - Архітектура багатозадачної каскадної CNN

Цей підхід був розширений для спільної детекції та вирівнювання облич. Тому використовують каскадну архітектуру з трьома етапами

ретельно розроблених глибоких згорткових мереж для передбачення місцезнаходження обличчя та опорних точок (landmarks) за принципом від грубого до точного. Вони також запропонували нову стратегію добування складних прикладів (online hard sample mining) в режимі онлайн для подальшого підвищення точності. Високорівнева архітектура цієї моделі показана на рисунку 2.4.

Водночас слід зазначити, що модель MTCNN є ресурсомісткою та не повністю оптимізованою для середовищ із обмеженими обчислювальними можливостями, таких як мобільні пристрої або веббраузери. Незважаючи на це, її висока точність у поєднанні з можливістю одночасного виявлення орієнтирів облич робить її цінним інструментом для тестування та порівняльного аналізу результатів у контексті розпізнавання облич.

2.2. Архітектура розроблюваної системи глибокого навчання у веб-браузерах

Ми будемо використовувати фреймворк MVC, тому вихідний код програмного забезпечення буде розділений на три шари, як показано нижче на рисунку 2.5.

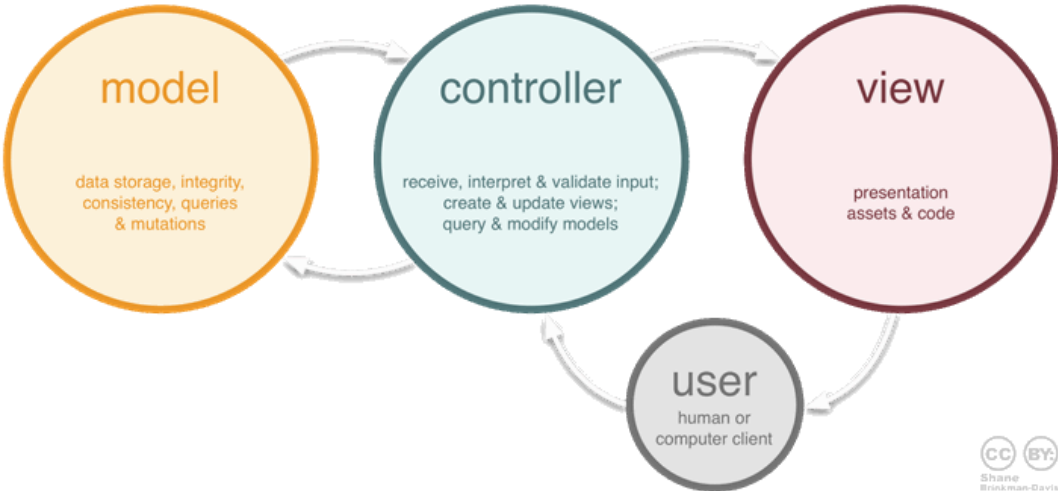

Рисунок 2.5 – MVC-фреймворк

Реалізація має фронтенд та бекенд додаток, як показано на наступному рисунку 2.6.

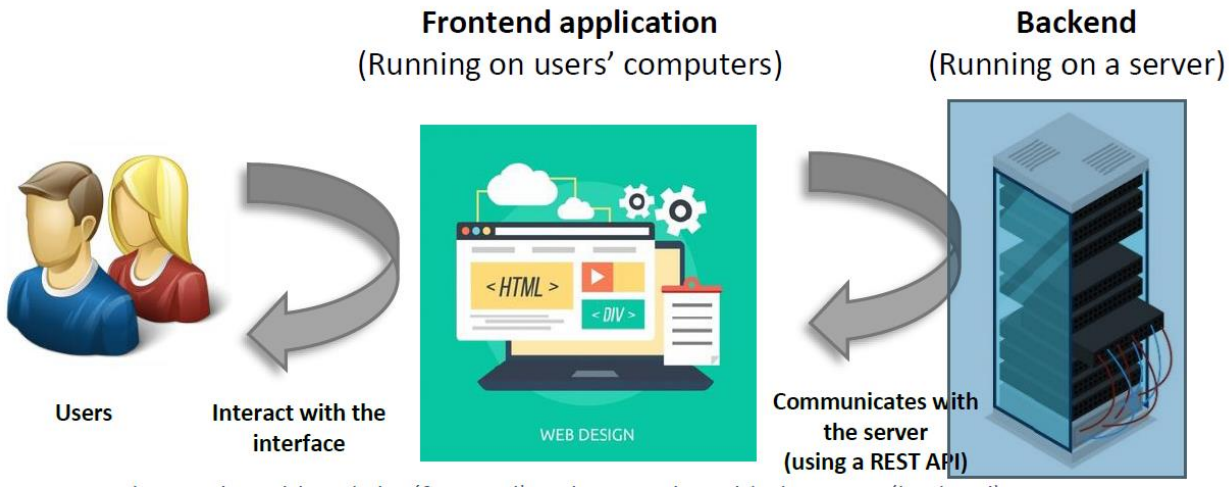

Рисунок 2.6 - Взаємодія між користувачем, фронтенд-застосунком та бек-ендом у веб-архітектурі

На рисунку 2.6 представлена взаємодія між користувачем, фронтенд-застосунком (клієнтською частиною) та бек-ендом (серверною частиною) у типовій веб-архітектурі.

Все починається з користувачів, які є кінцевими споживачами послуг. Користувачі взаємодіють безпосередньо з інтерфейсом вебсайту або веб-застосунку. Фронтенд-застосунок — це та частина вебсайту, яку бачить і з якою взаємодіє користувач. Вона працює безпосередньо на комп'ютерах користувачів (у їхньому браузері або як десктопний чи мобільний додаток).

На рисунку фронтенд представлений екраном ноутбука з елементами вебдизайну (HTML, CSS), що підкреслює його роль у відображенні контенту та наданні можливості взаємодії. Коли фронтенд-застосунку потрібні дані (наприклад, для завантаження сторінки, надсилання форми, збереження інформації) або необхідно виконати складні обчислення, він спілкується з бек-енд сервером.

Ця комунікація зазвичай відбувається за допомогою REST API (Application Programming Interface), що є стандартизованим набором правил для обміну даними між різними програмними компонентами. Стрілки вказують на двосторонній обмін даними.

Бек-енд — це "мозок" веб-застосунку. Він працює на сервері (представленому серверною стійкою) і відповідає за зберігання та управління даними (бази даних), обробку запитів від фронтенду, виконання бізнес-логіки, взаємодію з іншими сервісами. Користувачі не взаємодіють з бек-ендом безпосередньо; всі їхні запити проходять через фронтенд.

2.3. Опис платформ для розробки та алгоритму виявлення, витягнення та розпізнавання облич

2.3.1. Node.js

Для даної роботи ми використовуємо RESTful платформу під назвою Node.JS. Node.js — це відкритий код, крос-платформний та виконує код JavaScript поза браузером. Він використовується для виконання надійного розпізнавання та виявлення облич за допомогою бібліотек для отримання високоточних результатів.

Для виявлення облич ми можемо використовувати глибоку нейронну мережу для виявлення облич або простий фронтальний розпізнавач облич для ефективного та менш надійного виявлення.

Модуль розпізнавання облич використовує глибоку нейронну мережу для обчислення унікальних описів облич. Цей модуль розпізнавання облич може бути навчений з позначеними зображеннями облич, і пізніше він може передбачити мітку будь-якого зображення з вхідних даних користувача. Використовуючи алгоритми глибокого навчання та навчальні дані, ми будемо будувати моделі та робити передбачення на основі найкращої моделі.

Використовуючи комп'ютерний зір, ми, люди, навчаємо машини розпізнавати та виявляти речі так, як це робимо ми.

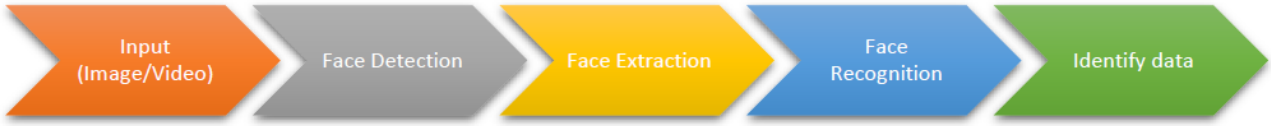

Рисунок 2.7 - Етапи, використані для виявлення облич, витягнення облич і розпізнавання облич

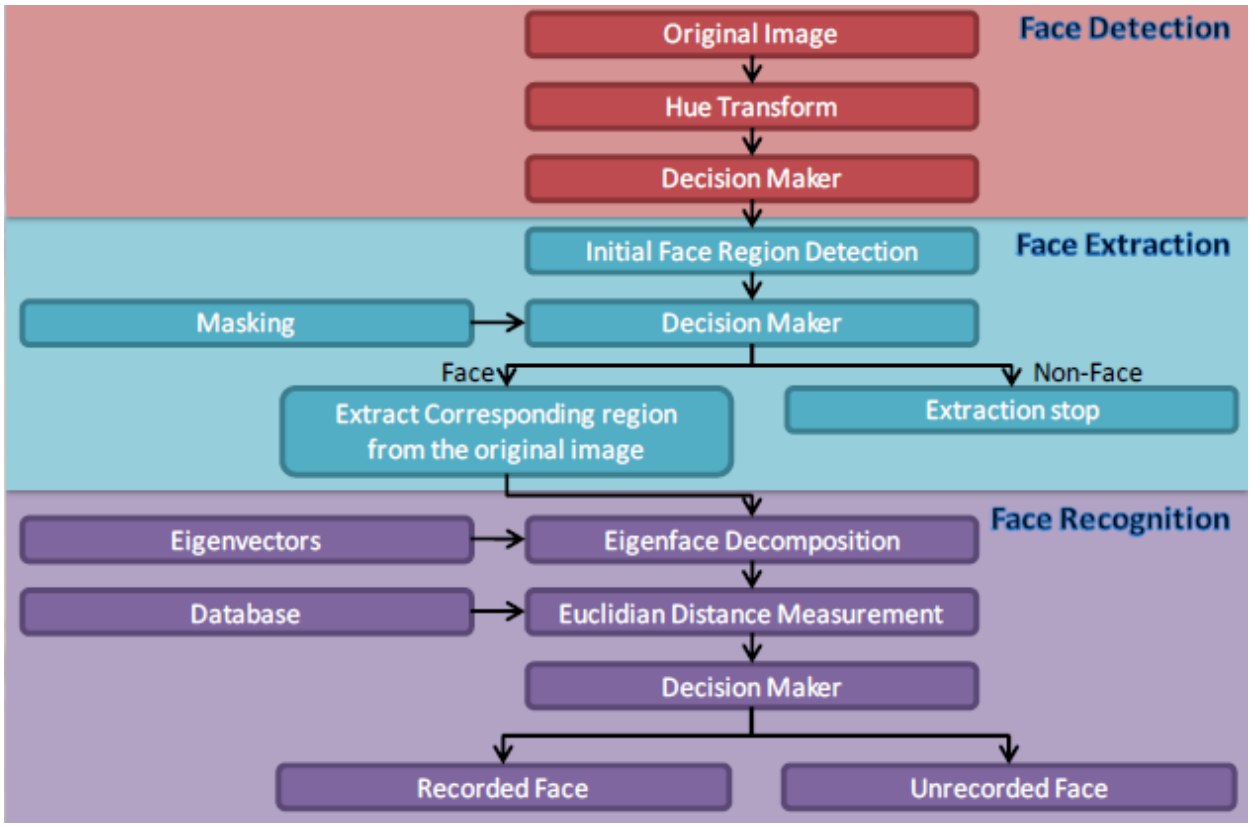

Рисунок 2.8 - Модель виявлення, витягнення та розпізнавання облич

На рисунку 2.8 зображено гібридну модель для виявлення та розпізнавання облич, яка складається з трьох послідовних етапів: виявлення облич (Face Detection), виділення облич (Face Extraction) та розпізнавання облич (Face Recognition).

1. Етап Виявлення Облич (Face Detection)

Початковим входом для системи є оригінальне зображення. На першому кроці до оригінального зображення застосовується перетворення відтінку. Цей етап спрямований на нормалізацію кольорових компонентів або виділення регіонів, які потенційно містять шкіру. Після перетворення відтінку відбувається початкове прийняття рішень, яке визначає потенційні області, що можуть містити обличчя, передаючи їх на наступний етап.

2. Етап Виділення Облич (Face Extraction)

Отримані на попередньому етапі потенційні регіони обличчя є входом для цього кроку. До виявлених регіонів застосовується операція маскування, яка може полягати у відсіканні або виділенні конкретних ділянок для подальшої обробки. Блок Decision Maker виконує більш точне рішення, класифікуючи вимасковані регіони як "Обличчя" (Face) або "Не-обличчя" (Non-Face). Якщо регіон класифікується як "Обличчя", відбувається витягування відповідного регіону з оригінального зображення (Extract Corresponding region from the original image). Це означає, що піксельні дані фактичного обличчя витягуються для подальшої обробки. Якщо регіон класифікується як "Не-обличчя", процес виділення для цього регіону зупиняється (Extraction stop).

3. Етап Розпізнавання Облич (Face Recognition)

Витягнуті регіони обличчя з попереднього етапу є входом для розпізнавання. Власні вектори (Eigenvectors) представляють "власні обличчя" (Eigenfaces) – основу для простору облич, отриману за допомогою методу головних компонент (PCA). Вони використовуються як еталон для розкладання. База даних містить попередньо збережені представлення (наприклад, власні обличчя або інші вектори ознак) відомих осіб. Витягнуте обличчя проектується на простір власних облич, що дозволяє отримати вектор ознак, який характеризує це обличчя. Потім вектор ознак вхідного обличчя порівнюється з векторами ознак, що зберігаються в базі даних, за

допомогою метрики Евклідової відстані. Менша відстань вказує на більшу схожість.

На основі результатів вимірювання відстані приймається фінальне рішення: якщо знайдено відповідність з високим ступенем подібності до запису в базі даних, обличчя класифікується як "Записане Обличчя" (Recorded Face). В іншому випадку, обличчя класифікується як "Незаписане Обличчя" (Unrecorded Face), що означає, що воно не було знайдено в базі даних відомих осіб.

2.3.2. Бібліотека tensorflow.js

Tensorflow.js — це бібліотека JavaScript з відкритим вихідним кодом з апаратним прискоренням для визначення, навчання та запуску моделей машинного навчання в браузерах, використовуючи JavaScript та API високого рівня. TensorFlow.js є відкритою JavaScript-бібліотекою для розробки та розгортання моделей машинного навчання (МН) безпосередньо у веб-браузері або в середовищі Node.js. Вона представляє собою адаптацію популярного фреймворку TensorFlow, розширюючи його функціональні можливості на клієнтську сторону та серверні JavaScript-середовища.

TensorFlow.js надає повний набір API для побудови, навчання та тестування нейронних мереж. Це включає можливість створення моделей з нуля за допомогою API, подібного до Keras (високорівневий tf.layers API), або ж маніпулювання тензорами на низькому рівні за допомогою tf.Operations API. Підтримується широкий спектр архітектур та типів шарів, що дозволяє реалізовувати різноманітні моделі глибокого навчання.

Бібліотека забезпечує ефективне завантаження та виконання моделей, навчених у TensorFlow (Python), які були конвертовані у формат TensorFlow.js. Це дозволяє використовувати складні, ресурсомісткі моделі, навчені на сервері, для інференсу безпосередньо на пристрої користувача.

Для оптимізації продуктивності, особливо при роботі з великими тензорами та складними моделями, TensorFlow.js використовує апаратне прискорення. У браузері це реалізується за допомогою WebGL для використання графічного процесора (GPU) клієнта, а в Node.js – через прив'язки до нативних бібліотек TensorFlow.

Система автоматичної диференціації (tf.Gradients) є фундаментальною для навчання нейронних мереж, дозволяючи ефективно обчислювати градієнти функції втрат щодо параметрів моделі, що є основою алгоритмів оптимізації, таких як градієнтний спуск.

2.3.3. Бібліотека face-api.js

Face-api.js — це модуль JavaScript, побудований на основі ядра tensorflow.js, і він реалізує кілька згорткових нейронних мереж (CNN) для виконання виявлення облич та розпізнавання облич, і він був оптимізований для роботи на веб- та мобільних пристроях.

Бібліотека може точно визначати місцезнаходження облич на зображеннях або у відеопотоці. Вона надає кілька моделей детекції, включаючи оптимізовані для мобільних пристроїв та "крихітних облич" (наприклад, SSD MobileNetV1 Face Detector, Tiny Face Detector), що дозволяє обирати компроміс між швидкістю та точністю залежно від конкретного застосування.

Після виявлення обличчя, face-api.js може визначити ключові анатомічні точки обличчя (наприклад, куточки очей, ніс, рот, контур щелепи). Це зазвичай робиться за допомогою моделей, що передбачають 68 або 5 ключових точок. Ці точки є критично важливими для подальшого вирівнювання обличчя та аналізу.

Бібліотека дозволяє генерувати так звані "описи облич" (face descriptors) – компактні числові вектори, які унікально характеризують ідентичність особи. Ці дескриптори можна порівнювати для визначення, чи

належать два обличчя одній і тій самій людині, або для ідентифікації особистості серед раніше зареєстрованих облич.

face-api.js може аналізувати вирази обличчя та класифікувати їх за основними емоціями (наприклад, щастя, смуток, злість, здивування, нейтральний, огида, страх). Додатково, бібліотека може надавати оцінку віку та статі для виявлених облич.

face-api.js розроблена з акцентом на простоту інтеграції. Вона надає високо рівневий API, що дозволяє розробникам додавати функціональність розпізнавання облич до своїх веб-застосунків лише кількома рядками коду, не заглиблюючись у складність моделей машинного навчання.

Обробка відбувається на стороні клієнта, конфіденційні біометричні дані користувача не залишають його пристрій, що значно підвищує рівень приватності та безпеки. Бібліотека постачається з попередньо навченими моделями, оптимізованими для JavaScript-середовища, що дозволяє швидко почати розробку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ГЛИБОКОГО НАВЧАННЯ У ВЕБ-БРАУЗЕРАХ ДЛЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБЛИЧ

3.1. Налаштування програмного середовища для розробки фронтенд- та бек-енд-застосунків

У цьому розділі буде викладено принципи реалізації фронтенд- та бек-енд-компонентів застосунків, зокрема з використанням бібліотеки face-api.js.

Для коректної роботи системи необхідно інсталювати актуальну версію платформи Node.js (рекомендовано останню версію з довгостроковою підтримкою (LTS), яка включає npm 6.12.0).

Створення робочої директорії для проєкту є обов'язковим. Наприклад, в операційних системах Windows це виконується командою md face_recognition, а в середовищах Linux – sudo mkdir /home/user/face_recognition.

Інсталяція необхідних залежностей здійснюється за допомогою пакетного менеджера npm. Для встановлення фреймворку Express та бібліотеки face-api.js виконуються наступні команди:

```
npm i express
npm i face-api.js
```

Для значного підвищення продуктивності обчислень рекомендується встановлення пакету @tensorflow/tfjs-node (опціонально), який забезпечує прискорення за рахунок компіляції та використання нативної бібліотеки TensorFlow на C++. Інсталяція здійснюється командою:

```
npm i face-api.js canvas @tensorflow/tfjs-node
```

Подальшим кроком є модифікація середовища (monkey-patching) для інтеграції необхідних поліфілів, що є критично важливим для коректної роботи face-api.js у Node.js. Це включає надання реалізацій об'єктів HTMLCanvasElement, HTMLImageElement та ImageData, які зазвичай доступні лише у браузерному середовищі

```
// Імпорт прив'язок Node.js до нативної бібліотеки TensorFlow. Це необов'язково,
// але значно прискорює обчислення, вимагаючи встановленого Python.
import '@tensorflow/tfjs-node';

// Імпорт реалізацій обгорток для HTMLCanvasElement, HTMLImageElement, ImageData
// у середовищі Node.js.
import * as canvas from 'canvas';
import * as faceapi from 'face-api.js';

// Модифікація глобального середовища Node.js шляхом надання реалізацій
// об'єктів Canvas, Image та ImageData. Це необхідно для коректної
// роботи face-api.js, зокрема, реалізація ImageData потрібна при використанні MTCNN.
const { Canvas, Image, ImageData } = canvas;
faceapi.env.monkeyPatch({ Canvas, Image, ImageData });
```

Ця процедура забезпечує сумісність функціоналу, що очікує браузерне DOM-середовище, з Node.js.

3.2. Завантаження моделей виявлення та розпізнавання

Доступ до попередньо визначених моделей нейронних мереж у бібліотеці face-api.js здійснюється через глобальний об'єкт faceapi.nets. Цей об'єкт інкапсулює різноманітні архітектури, призначені для виявлення та розпізнавання облич, а також для оцінки пов'язаних атрибутів, таких як вік, стать та експресії. Перелік доступних моделей включає:

```
console.log(faceapi.nets);
// ageGenderNet
// faceExpressionNet
// faceLandmark68Net
// faceLandmark68TinyNet
// faceRecognitionNet
// ssdMobilenetv1
// tinyFaceDetector
// mtcnn
// tinyYolov2
```

Для успішного завантаження моделей необхідно забезпечити їхню доступність за вказаним URI (Uniform Resource Identifier). Це передбачає розміщення файлів маніфесту (.json) та відповідних файлів ваг моделі (шардів) у відповідній публічній директорії веб-сервера або за визначеним маршрутом. Метод loadFromUri() асинхронно завантажує файли моделі із зазначеного URL-шляху.

Приклад завантаження моделі ssdMobilenetv1 продемонстровано нижче:

```
await faceapi.nets.ssdMobilenetv1.loadFromUri('/models');
// Аналогічно для інших моделей:
// await faceapi.nets.faceLandmark68Net.loadFromUri('/models');
// await faceapi.nets.faceRecognitionNet.loadFromUri('/models');
// ...
```

Альтернативним методом завантаження моделей є безпосереднє звернення до локального файлового сховища. Метод loadFromDisk() дозволяє завантажувати моделі з локальної директорії. Це особливо актуально для застосунків, що виконуються в середовищі Node.js, де доступ до файлової системи є прямим:

```
await faceapi.nets.ssdMobilenetv1.loadFromDisk('./models');
```

Окрім використання глобальних екземплярів, існує можливість явного інстанціювання окремих нейронних мереж. Це дозволяє більш гнучко управляти конфігурацією та життєвим циклом моделі. Приклад створення та завантаження екземпляра SsdMobilenetv1 наведено:

```
const net = new faceapi.SsdMobilenetv1();
await net.loadFromUri('/models');
```

3.3. Вхідні дані для обробки

Бібліотека face-api.js надає гнучкий інтерфейс для обробки візуальних даних, приймаючи як вхідні параметри стандартні елементи HTML-документа, що містять зображення або відеопотік. Зокрема, підтримуються елементи <img/>, <video/> та <canvas/>. Ідентифікація цільового елемента може здійснюватися шляхом передачі його унікального ідентифікатора (id) або безпосередньо посилання на DOM-об'єкт.

Приклади визначення вхідних елементів:

```
<img id="myImg" src="images/example.png" />
<video id="myVideo" src="media/example.mp4" />
<canvas id="myCanvas" />
```

Програмний доступ до вхідних даних:

```
const input = document.getElementById('myImg');
// Альтернативні варіанти:
// const input = document.getElementById('myVideo');
// const input = document.getElementById('myCanvas');
// Або спрощений синтаксис:
// const input = 'myImg';
```

3.4. Методи виявлення облич

face-api.js пропонує функціональність для виявлення облич на основі попередньо навчених нейронних мереж.

3.4.1. Виявлення всіх облич

Для ідентифікації всіх облич, присутніх на вхідному зображенні або відеокадрі, використовується асинхронна функція faceapi.detectAllFaces(). Ця

функція повертає масив об'єктів, кожен з яких представляє виявлене обличчя та містить інформацію про його обмежувальну рамку.

```
const detections = await faceapi.detectAllFaces(input);
```

3.4.2. Виявлення одного обличчя

У випадках, коли необхідно ідентифікувати лише одне обличчя (як правило, те, що має найвищий коефіцієнт впевненості), застосовується функція faceapi.detectSingleFace(). Вона повертає єдиний об'єкт детекції або undefined, якщо обличчя не виявлено.

```
const detection = await faceapi.detectSingleFace(input);
```

3.4.3. Конфігурація детектора облич

За замовчуванням, функції faceapi.detectAllFaces() та faceapi.detectSingleFace() використовують модель SSD MobileNet V1 для детекції облич. Однак, бібліотека дозволяє явно вказувати бажаний детектор шляхом передачі відповідного об'єкта опцій як другого аргументу:

```
const detections1 = await faceapi.detectAllFaces(input, new faceapi.SsdMobilenetv1(
const detections2 = await faceapi.detectAllFaces(input, new faceapi.TinyFaceDetect(
const detections3 = await faceapi.detectAllFaces(input, new faceapi.MtcnnOptions())
```

3.4.4. Виявлення опорних точок обличчя (Landmarks)

Для отримання інформації про ключові анатомічні точки обличчя (наприклад, розташування очей, носа, рота) після детекції обличчя, використовується метод .withFaceLandmarks(). Цей метод може бути ланцюгованим після функції детекції:

Виявлення всіх облич з опорними точками:

```
const detectionsWithLandmarks = await faceapi.detectAllFaces(input).withFaceLar
```

Виявлення одного обличчя з опорними точками:

```
const detectionWithLandmarks = await faceapi.detectSingleFace(input).withFaceLa
```

3.4.5. Використання "Малої" моделі для опорних точок

Існує також можливість використання оптимізованої, меншої моделі для виявлення опорних точок, що може бути корисним для підвищення продуктивності на менш потужних пристроях. Це досягається шляхом передачі булевого значення true до методу withFaceLandmarks():

```
const useTinyModel = true;
const detectionsWithLandmarks = await faceapi.detectAllFaces(input).withFaceLandma
```

3.5. Виявлення опорних точок обличчя та обчислення дескрипторів

Для отримання деталізованої інформації про ключові анатомічні точки на обличчі (наприклад, розташування очей, носа, рота та контур щелепи), після детекції обличчя застосовується метод .withFaceLandmarks(). Цей метод дозволяє отримати масив об'єктів, що містять координати 68 або 5 ключових точок для кожного виявленого обличчя.

Виявлення всіх облич з опорними точками:

```
const detectionsWithLandmarks = await faceapi.detectAllFaces(input).withFaceLar
```

Виявлення одного обличчя з опорними точками (з найвищим коефіцієнтом впевненості):

```
const detectionWithLandmarks = await faceapi.detectSingleFace(input).withFaceLa
```

Для завдань розпізнавання та ідентифікації облич використовується обчислення дескрипторів обличчя. Дескриптор обличчя — це компактний числовий вектор (embedding), який унікально представляє ідентичність особи та може бути використаний для порівняння з іншими дескрипторами для визначення схожості або належності до певної особи. Ця функціональність реалізується методом .withFaceDescriptors(), який зазвичай застосовується після виявлення обличчя та його опорних точок.

Виявлення всіх облич, опорних точок та обчислення дескрипторів:

```
const results = await faceapi.detectAllFaces(input).withFaceLandmarks().withFac
```

Виявлення одного обличчя, опорних точок та обчислення дескриптора:

```
const result = await faceapi.detectSingleFace(input).withFaceLandmarks().withFa
```

3.6. Виконання розпізнавання облич шляхом порівняння дескрипторів

Для здійснення операції розпізнавання облич бібліотека face-api.js надає клас faceapi.FaceMatcher. Цей клас дозволяє порівнювати векторні дескриптори відомих облич (довідкові дані) з дескрипторами облич, отриманих під час запиту, з метою ідентифікації.

Ініціалізація екземпляра FaceMatcher вимагає надання набору довідкових дескрипторів. Ці дескриптори можуть бути попередньо обчислені з довідкового зображення, що містить одне або кілька облич.

Процедура отримання цих довідкових дескрипторів включає детекцію облич, виявлення опорних точок та генерацію дескрипторів:

```
const results = await faceapi
    .detectAllFaces(referenceImage)
    .withFaceLandmarks()
    .withFaceDescriptors();

if (!results.length) {
    return; // Обробка випадку, коли обличчя не виявлено в довідковому зображенні
}

// Створення екземпляра FaceMatcher з автоматично призначеними мітками
// на основі результатів детекції довідкового зображення.
const faceMatcher = new faceapi.FaceMatcher(results);
```

Після ініціалізації FaceMatcher здійснюється процес ідентифікації обличчя в запиті (наприклад, queryImage1). Для одного обличчя процедура полягає в детекції, виявленні опорних точок та отриманні дескриптора, після чого виконується порівняння з довідковими даними:

```
const singleResult = await faceapi
    .detectSingleFace(queryImage1)
    .withFaceLandmarks()
    .withFaceDescriptor();

if (singleResult) {
    // Пошук найбільш відповідного дескриптора серед довідкових даних
    const bestMatch = faceMatcher.findBestMatch(singleResult.descriptor);
    console.log(bestMatch.toString()); // Виведення результату порівняння
}
```

Аналогічно, FaceMatcher може бути використаний для розпізнавання множинних облич у одному зображенні (queryImage2). У цьому випадку для кожного виявленого обличчя обчислюється дескриптор, який потім порівнюється з довідковими даними:

```
const results = await faceapi
    .detectAllFaces(queryImage2)
    .withFaceLandmarks()
    .withFaceDescriptors();

results.forEach(fd => {
    // Для кожного виявленого дескриптора обличчя знайти найкращу відповідність
    const bestMatch = faceMatcher.findBestMatch(fd.descriptor);
    console.log(bestMatch.toString()); // Виведення результату для кожного обличчя
});
```

Ключовою функціональністю FaceMatcher є можливість асоціювати дескриптори з експліцитно визначеними мітками (іменами осіб). Це забезпечує семантичну ідентифікацію розпізнаних облич. Довідкові дескриптори можуть бути організовані за допомогою класу faceapi.LabeledFaceDescriptors:

```
const labeledDescriptors = [
    new faceapi.LabeledFaceDescriptors(
        'obama', // Мітка для обличчя
        [descriptorObama1, descriptorObama2] // Масив дескрипторів для цієї мітки
    ),
    new faceapi.LabeledFaceDescriptors(
        'trump', // Мітка для обличчя
        [descriptorTrump]
    )
];

const faceMatcher = new faceapi.FaceMatcher(labeledDescriptors);
```

3.7. Параметри моделей виявлення облич

Бібліотека face-api.js надає декілька моделей для виявлення облич, кожна з яких має специфічні параметри конфігурації, що впливають на продуктивність та точність.

SsdMobilenetv1Options

Ці параметри застосовуються до детектора облич, що базується на архітектурі SSD (Single Shot MultiBox Detector) з MobileNetV1 як основою.

minConfidence?: number

Мінімальний пороговий показник достовірності виявлення. Детекції з оцінкою нижче цього порогу відхиляються.

Значення за замовчуванням: 0.5

maxResults?: number

Максимальна кількість об'єктів облич, що підлягають поверненню.

Значення за замовчуванням: 100

Приклад використання:

```
const options = new faceapi.SsdMobilenetv1Options({ minConfidence: 0.8 });
```

TinyFaceDetectorOptions

Дані параметри призначені для легковагової моделі детектора облич, оптимізованої для високої продуктивності.

inputSize?: number

Розмір вхідного зображення для обробки. Зменшення цього параметра прискорює обробку, але знижує точність виявлення менших облич. Значення повинні бути кратні 32. Рекомендовані розміри для відстеження облич з веб-камери: 128, 160. Для виявлення дрібних облич рекомендуються більші розміри, такі як 512, 608.

Приклад використання:

```
const options = new faceapi.TinyFaceDetectorOptions({ inputSize: 320 });
```

Після виявлення обмежувальних рамок існує можливість витягування сегментів зображень, сфокусованих виключно на обличчі. Ця операція є критично важливою перед подальшою передачею даних до мережі розпізнавання облич, оскільки вона суттєво підвищує точність розпізнавання та якість кінцевих результатів.

API для розпізнавання облич включає архітектуру згорткової нейронної мережі, призначену для визначення 68 опорних точок (landmarks) на кожному виявленому обличчі. Координати цих опорних точок можуть бути використані для точного вирівнювання та центрування обмежувальної рамки навколо обличчя.

Рисунок 3.1 - Результати виявлення облич (зліва) у порівнянні з центрованим зображенням обличчя з використанням точкових орієнтирів обличчя (справа)

Після етапів детекції та вирівнювання, оброблені зображення облич передаються до мережі розпізнавання облич. Ця мережа заснована на архітектурі, подібній до ResNet-34, яка є реалізацією, запозиченою з бібліотеки dlib. Метою навчання даної нейронної мережі є формування відображення складних візуальних ознак людського обличчя у високорозмірний числовий вектор, відомий як дескриптор обличчя (face descriptor) або вбудовування обличчя (face embedding). Цей вектор, що зазвичай складається зі 128 значень, інкапсулює унікальні ідентифікаційні характеристики особи. Важливо зазначити, що модель демонструє високу генералізуючу здатність, дозволяючи розпізнавати обличчя осіб, які не були присутні в початковому навчальному наборі даних.

Процедура порівняння двох облич ґрунтується на аналізі відстані між їхніми відповідними дескрипторами. Зокрема, обчислюється Евклідова відстань між двома векторами дескрипторів. На основі цього значення та попередньо визначеного порогового значення (threshold), алгоритм визначає ступінь схожості між обличчями. Вибір оптимального порогового значення може залежати від характеристик вхідних зображень, таких як їхня роздільна

здатність. Наприклад, порогове значення 0.6 часто вважається прийнятним для зображень з роздільною здатністю приблизно 150x150 пікселів.

3.8. Метрика Евклідової відстані

Евклідова відстань є метрикою, що застосовується для кількісної оцінки подібності або несхожості між точками (зразками) у багатовимірному геометричному просторі. Її широке використання в обчислювальних та аналітичних завданнях обумовлене відносною простотою розрахунку. Завдяки цій метриці, Евклідів простір набуває властивостей метричного простору, де визначено поняття "відстані" між будь-якими двома елементами.

Евклідова відстань для n-вимірного простору визначається як

$$d(p,q) = \sqrt{\sum_{i=0}^{n} (p_i - q_i)^2} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

Евклідова відстань не враховує взаємозв'язок кожного виміру вектора, і кожен вимір є однаково важливим, що впливає на ефективність та сферу його використання. Вагова евклідова відстань використовується для забезпечення точності ідентифікації обличчя.

Наступна функція використовується для обчислення евклідової відстані між двома дескрипторами обличчя

```
// Використовується для обчислення Евклідової відстані між двома векторами
const dist = faceapi.euclideanDistance([0, 0], [0, 20]);
console.log(dist); // Виведе: 20
```

Для візуалізації концепції порівняння облич за допомогою Евклідової відстані між їхніми дескрипторами, наведено приклади, що демонструють процес знаходження подібності.

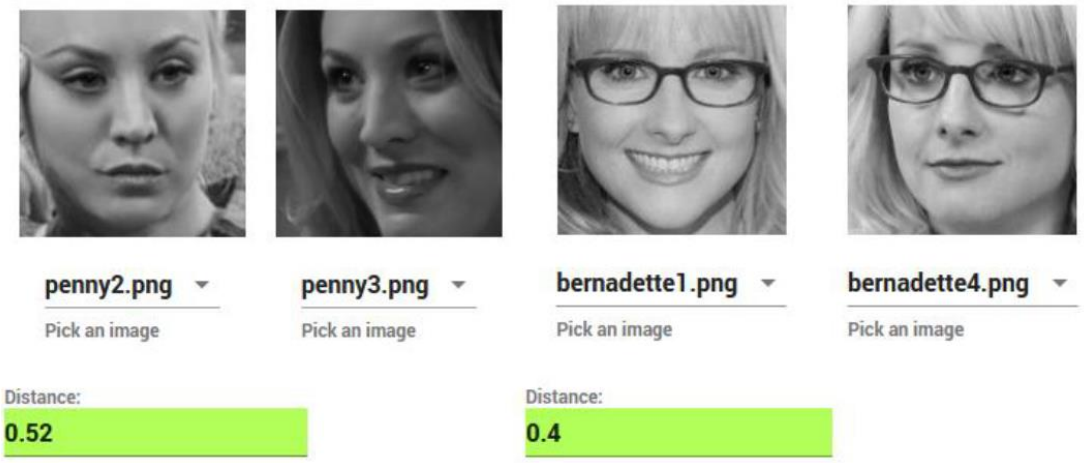

Рисунок 3.2 - Задовільні результати схожості облич з використанням евклідової відстані

Рисунок 3.3 - Незадовільні результати схожості облич з використанням евклідової відстані

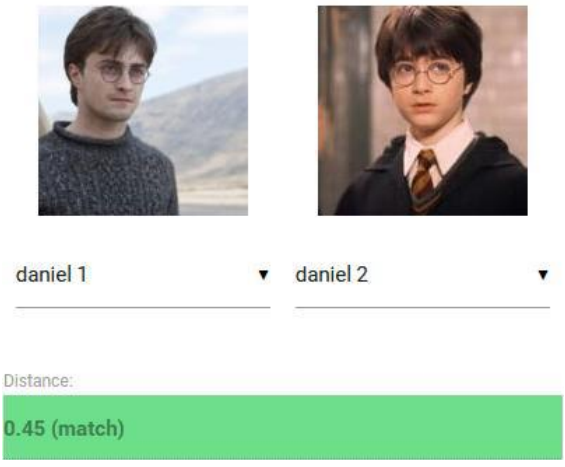

Рисунок 3.4 - Деніел Редкліфф (вік 28 і вік 11)

Рисунок 3.4 - Леонардо ДіКапріо (чисте та нечисте обличчя)

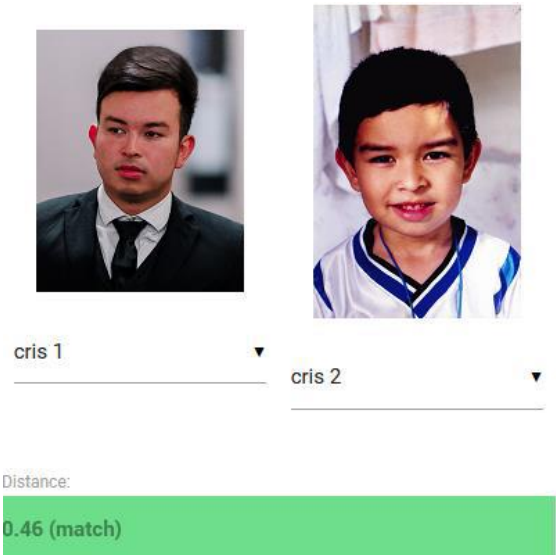

Рисунок 3.5 - Крістіан Еспіноса, вік 24 і вік 4

3.9. Реалізація інтерфейсу користувача та представлення результатів роботи системи розпізнавання

На рисунку 3.6 подано інтерфейс, що демонструє веб-додаток, розроблений для інтерактивного використання бібліотеки face-api.js, що спеціалізується на функціях розпізнавання облич.

Інтерфейс розділений на дві основні частини:

- Бічна панель навігації (ліворуч) - містить список функціональних можливостей та демонстрацій, доступних у face-api.js.

- Основна область вмісту (праворуч) - відображає результат обробки зображень та елементи керування для налаштування параметрів детекції.

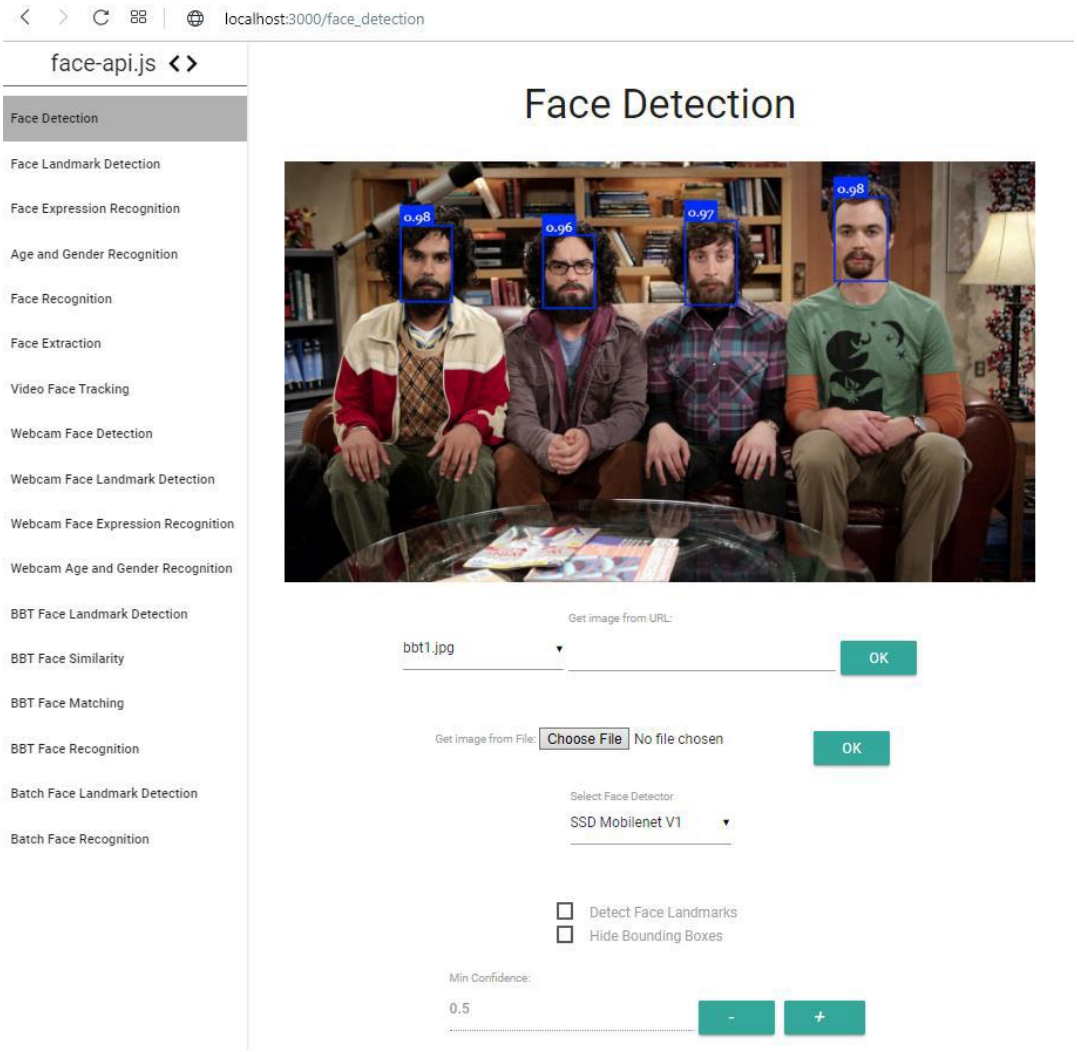

Рисунок 3.6 - Приклад роботи додатку

Навігаційна панель надає швидкий доступ до різних демонстрацій та інструментів бібліотеки face-api.js. Вона включає такі категорії:

- Face Detection (Виявлення обличчя)

-Face Landmark Detection (Виявлення опорних точок обличчя)

- Face Expression Recognition (Розпізнавання виразів обличчя)

- Age and Gender Recognition (Розпізнавання віку та статі)

- Face Recognition (Розпізнавання обличчя)

- Face Extraction (Виділення обличчя)

- Video Face Tracking (Відстеження обличчя у відео)

- Webcam Face Detection/Landmark Detection/Expression Recognition/Age and Gender Recognition (Функції для веб-камери)

Розділи, пов'язані з "BBT" ("Big Bang Theory"), такі як "BBT Face Landmark Detection", "BBT Face Similarity", "BBT Face Matching", "BBT Face Recognition", "Batch Face Landmark Detection", "Batch Face Recognition".

У центрі розташовано зображення чотирьох осіб (персонажів з "Теорії великого вибуху"), на якому сині рамки обмежують виявлені обличчя. Над кожною рамкою відображається числовий показник впевненості (наприклад, 0.98, 0.96, 0.97), що вказує на достовірність виявлення обличчя.

Опції завантаження зображень:

- "Get image from URL" - поле введення (input field) для вставки URL-адреси зображення та кнопка "OK" для його завантаження.

- "Get image from File" - кнопка "Choose File" для вибору локального файлу зображення з комп'ютера користувача.

Розкривний список "Select Face Detector", що дозволяє вибрати одну з доступних моделей детекції (наприклад, "SSD Mobilenet V1", як показано на скріншоті).

Два прапорці (checkboxes) для керування відображенням та функціональністю:

- "Detect Face Landmarks" (Виявити опорні точки обличчя).

- "Hide Bounding Boxes" (Приховати обмежувальні рамки).

Мінімальний поріг впевненості (Min Confidence) - поле введення з числовим значенням (за замовчуванням 0.5), яке можна регулювати за допомогою кнопок "+" та "-" для встановлення мінімального порогу достовірності детекції.

Розглянемо результати роботи системи для виявлення кількох облич (з використанням Face-api.js).

Рисунок 3.7 - Виявлення облич та обмежувальні рамки для облич

Рисунок 3.8 - Виявлення облич та виявлення орієнтирів обличчя

Модель розпізнавання виразів обличчя є легкою, швидкою та забезпечує реалістичну точність. Модель використовує глибинно-сепарабельні згортки та щільно пов’язані блоки.

Модель була навчена на різноманітних зображеннях із загальнодоступних наборів даних та веб-ресурсів.

Рисунок 3.9 - Розпізнавання виразів обличчя з використанням SSD MobileNet v1 та мінімальним коефіцієнтом довіри 0.5. Приклад №1

Рисунок 3.10 - Розпізнавання виразів обличчя з використанням SSD MobileNet v1 та мінімальним коефіцієнтом довіри 0.5. Приклад №2

Рисунок 3.11 - Сумне обличчя, яке, можливо, до певного рівня може бути сердитим

Рисунок 3.12 - Сердитий Х'ю Джек ман

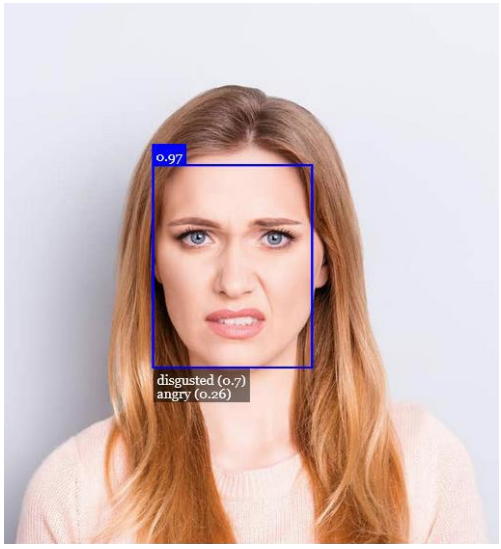

Рисунок 3.13 - Обличчя з відразою (70%) та сердите (0.26)

Модель розпізнавання віку та статі використовує багатозадачну мережу, яка включає шар вилучення ознак, шар регресії віку та класифікатор статі. Ця архітектура подібна до Xception.

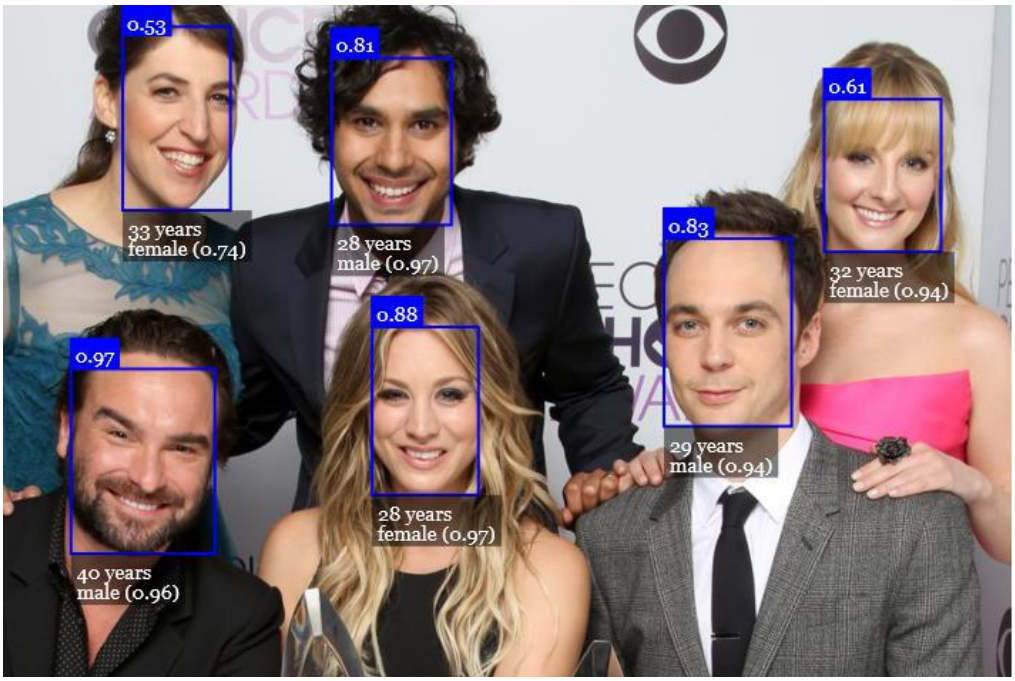

Рисунок 3.14 - Виявлення віку та статі на шести різних обличчях з використанням face-api.js

Рисунок 3.15 - Виявлення віку та статі на 5 різних обличчях з використанням face-api.js

На рисунку 3.16 показано процес витягнення обличчя.

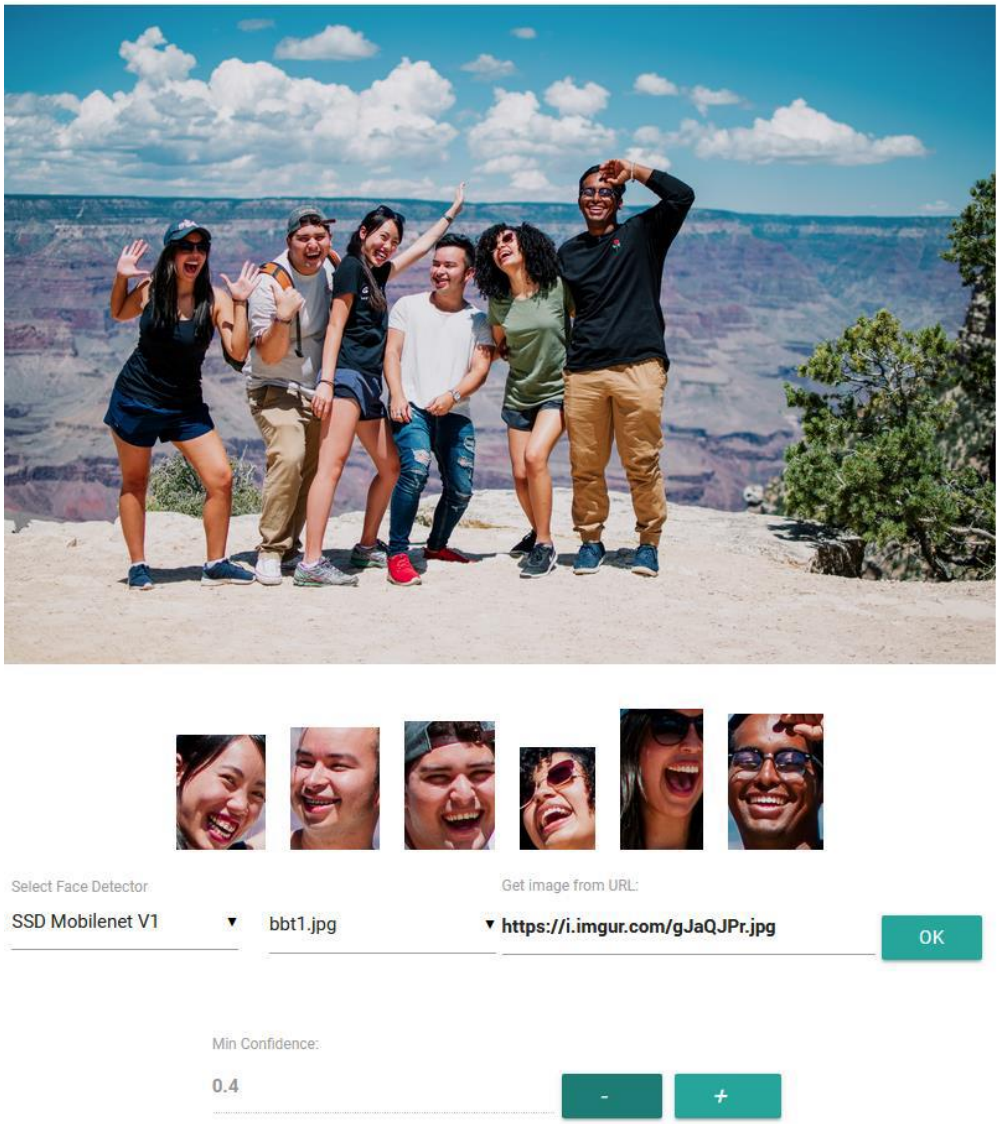

Рисунок 3.16 - Виконання витягнення обличчя з використанням face-api.js

Порівняння швидкості виявлення облич у відеопотоці з використанням бібліотеки face-api.js є важливим аспектом для оптимізації продуктивності застосунків реального часу. Цей процес вимагає систематичного підходу до вимірювання та аналізу ефективності різних конфігурацій детектора.

Основна мета полягає у визначенні найбільш оптимальної моделі виявлення облич та її параметрів (наприклад, розміру вхідного зображення, порогу впевненості) для конкретних умов відеопотоку, що дозволяє досягти балансу між швидкістю обробки (вимірюється в кадрах на секунду, FPS) та точністю детекції.

Виконаємо порівняння швидкості виявлення при відстеженні облич на відео (з використанням Face-api.js)

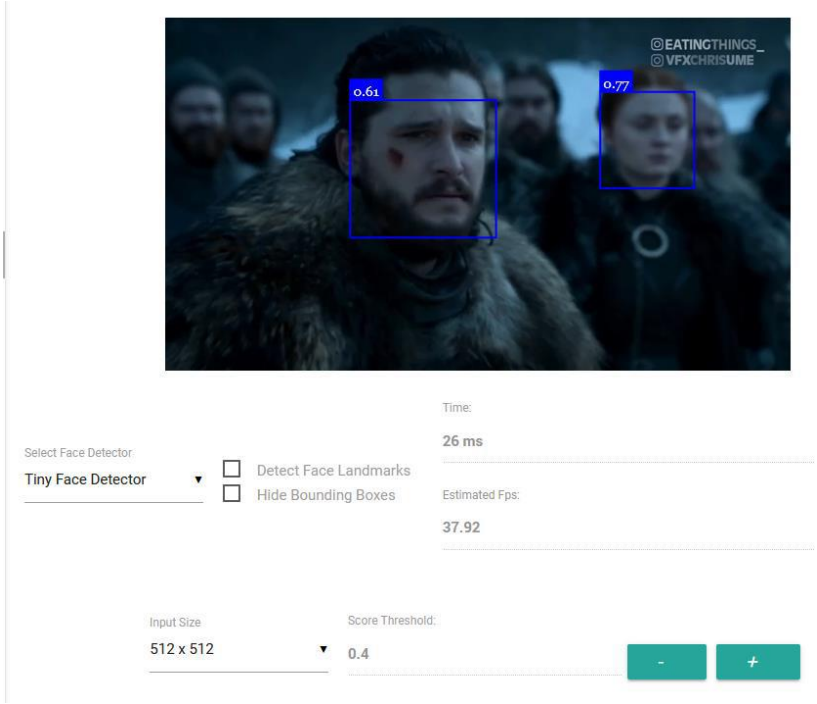

Рисунок 3.17 - Виявлення облич, виконане з використанням Tiny Face Detector як нейронної мережі

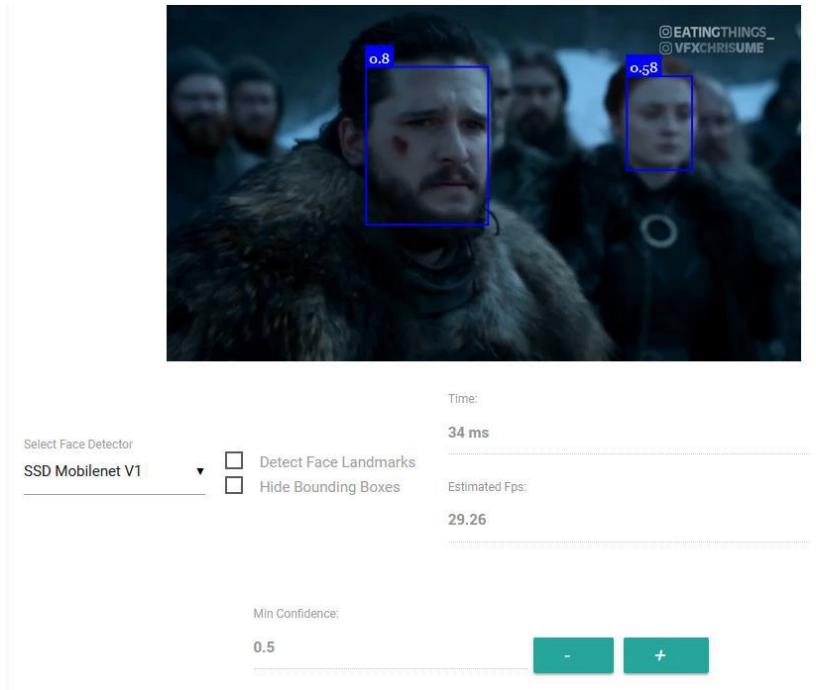

Рисунок 3.18 - Виявлення облич, виконане з використанням SSD MobileNet v1 як нейронної мережі

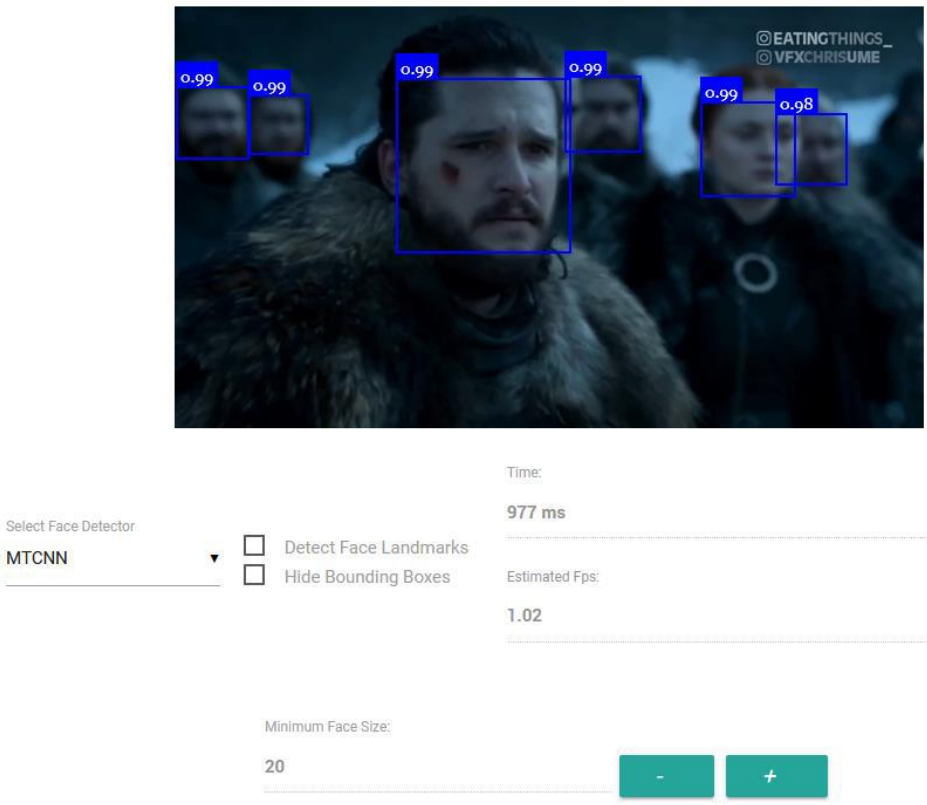

Рисунок 3.19 - Виявлення облич, виконане з використанням MTCNN як нейронної мережі

На рисунку 3.20 у вигляді таблиці представлені результати порівняння продуктивності трьох різних нейронних мереж для виявлення облич: Tiny Face Detector, SSD MobileNet v1 та MTCNN. Порівняння базується на трьох ключових метриках: затримці (Latency), оціненій частоті кадрів (Estimated FPS) та кількості виявлених облич (Faces Detected).

Neural Network	Latency	Estimated FPS (Frames per second)	Faces Detected
Tiny Face Detector	26 ms	37.92	2
SSD MobileNet v1	34 ms	29.26	2
MTCNN	977 ms	1.02	6

Рисунок 3.20 - Результати порівняння продуктивності трьох нейронних мереж для виявлення облич

Проведемо аналіз результатів виявлення облич різними мережами.

1. Tiny Face Detector

Затримка (Latency) - ця модель демонструє найнижчу затримку серед представлених, яка становить 26 мс.

Оцінена частота кадрів (Estimated FPS) - відповідно, Tiny Face Detector забезпечує найвищу оцінену частоту кадрів — 37.92 FPS, що вказує на її високу швидкість обробки.

Виявлені обличчя (Faces Detected) - модель успішно виявила 2 обличчя. Це свідчить про те, що вона є найефективнішою за швидкістю для сценаріїв, де пріоритетом є висока частота кадрів.

2. SSD MobileNet v1

Затримка (Latency) - цей детектор показує затримку у 34 мс, що є дещо вищим показником порівняно з Tiny Face Detector.

Оцінена частота кадрів (Estimated FPS) - його продуктивність становить 29.26 FPS, що є нижчим за показник Tiny Face Detector, але все ще прийнятним для багатьох застосувань у реальному часі.

Виявлені обличчя (Faces Detected) - модель також виявила 2 обличчя. SSD MobileNet v1 є добрим компромісом між швидкістю та потенційною точністю, хоча в даному тесті кількість виявлених облич збігається з Tiny Face Detector.

3. MTCNN

Затримка (Latency) - MTCNN демонструє значно вищу затримку — 977 мс, що робить її найповільнішою з трьох моделей.

Оцінена частота кадрів (Estimated FPS) - це безпосередньо відбивається на дуже низькій частоті кадрів — лише 1.02 FPS, що робить її непридатною для більшості застосувань реального часу.

Виявлені обличчя (Faces Detected) - незважаючи на низьку швидкість, MTCNN виявила 6 облич, що є найбільшою кількістю порівняно з іншими моделями в цьому тесті. Це може вказувати на її потенційно вищу точність

або чутливість до виявлення менших чи складніших облич, хоча це відбувається за рахунок значного збільшення часу обробки.

Отримані результати підкреслюють очевидні компроміси між швидкістю та можливістю виявлення більшої кількості облич (що може корелювати з точністю) для різних моделей. Tiny Face Detector є оптимальним вибором для застосунків, що вимагають високої частоти кадрів. SSD MobileNet v1 також пропонує добру продуктивність, яка є балансом між швидкістю та точністю. MTCNN, незважаючи на здатність виявляти більше облич у цьому конкретному тесті, є значно повільнішою, що обмежує її використання в сценаріях реального часу, де необхідна висока частота кадрів.

Подальший розвиток системи розпізнавання облич може включати інтеграцію додаткових функціональних можливостей. Доцільним є впровадження розпізнавання об'єктів для ідентифікації різноманітних об'єктів на зображеннях або у відеопотоках. Окрім цього, перспективним напрямком є додавання незалежного модуля для детекції простих жестів рук з відео в реальному часі. Хоча ця функціональність не претендує на ідеальне вирішення проблеми розпізнавання жестів рук, вона покликана продемонструвати підхід до реалізації подібних завдань.

Отже, було показано застосування глибоких нейронних мереж для задач виявлення та розпізнавання облич. Була розроблена та реалізована прототипна система виявлення та розпізнавання облич з використанням Node.js та TensorFlow.js core API, що спирається на можливості існуючої бібліотеки face-api.js, Ця бібліотека оптимізована для ефективної роботи в браузерному середовищі. Генерація навчальних даних виходила за межі цього дослідження; проте існують загальнодоступні набори даних, які можуть бути використані для завантаження в додаток з метою виконання розпізнавання облич. Розроблений API дозволив провести низку експериментів, де користувачі мали можливість визначати емоції, стать та вік

людини. За останні роки технології розпізнавання облич досягли значного прогресу, що дозволяє створювати додатки з високою точністю ідентифікації облич. Особливий інтерес у рамках цього експерименту становить пошук нових корисних застосувань для цієї технології.

В умовах сучасної епохи "великих даних" складні системи вимагають все більшого обсягу інформації. Різноманітні навчальні дані повинні бути репрезентативними щодо даних, з якими нейронна мережа зіткнеться в реальному світі. Поряд з цим, розуміння ваг, які нейронна мережа навчилася, залишається складним завданням. Зокрема, при використанні багатошарових архітектур виведення чітких висновків щодо ролі окремих ваг у прийнятті рішень може бути вкрай важким.

ВИСНОВКИ

У дипломній роботі проведено дослідження та практичну реалізацію системи розпізнавання облич у браузерному середовищі з використанням сучасних методів глибокого навчання. На основі аналізу предметної області та доступних інструментів запропоновано архітектурне рішення, яке поєднує ефективні моделі комп'ютерного зору, програмні платформи та бібліотеки для забезпечення роботи нейронних мереж безпосередньо у веб-браузері без потреби у серверних обчисленнях.

У першому розділі було проаналізовано сучасний стан застосування глибокого навчання у веб-технологіях. Особливу увагу приділено можливостям реалізації нейронних мереж у браузерах, що дозволяє створювати розподілені системи штучного інтелекту, зменшуючи затримки, підвищуючи приватність користувача та мінімізуючи залежність від серверної інфраструктури. Сформульовано мету, завдання та сфери практичного застосування запропонованого підходу до розпізнавання облич.

Другий розділ присвячено огляду моделей глибокого навчання для виявлення облич та обґрунтуванню вибору трьох моделей: SSD MobileNet V1, Tiny Face Detector та MTCNN. Було охарактеризовано особливості кожної моделі, її архітектуру, переваги та недоліки. SSD MobileNet V1 забезпечує високу точність локалізації облич, Tiny Face Detector є оптимізованим рішенням для швидкодії та використання в обмежених середовищах, тоді як MTCNN надає розширену функціональність для роботи з орієнтирами облич, попри високе навантаження на ресурси. На основі цього проведено структурний опис архітектури запропонованої системи, що охоплює компоненти обробки даних, детектування, візуалізації та взаємодії з користувачем.

У третьому розділі розглянуто процес реалізації системи на основі платформ Node.js, TensorFlow.js та face-api.js, які забезпечують

кросплатформеність, інтеграцію з веб-середовищем та підтримку моделей глибокого навчання. Було реалізовано повний цикл роботи системи — від завантаження моделей і вхідних зображень до виявлення облич, побудови дескрипторів та виконання розпізнавання на основі евклідової відстані. Також забезпечено можливість визначення орієнтирів обличчя, оцінювання віку, статі та емоційного стану.

Важливим досягненням є те, що вся система працює у браузері без потреби надсилати зображення на сервер, що значно підвищує безпеку персональних даних. Результати розпізнавання представлені через інтерфейс користувача, що дозволяє візуалізувати обмежувальні рамки, ідентифіковані особи та додаткову аналітичну інформацію.

Підсумовуючи, у дипломній роботі:

- проведено теоретичний аналіз та класифікацію моделей глибокого навчання для задач виявлення облич;

- реалізовано функціональну веб-систему, здатну ефективно розпізнавати обличчя на основі попередньо навчених моделей у реальному часі;

- забезпечено масштабованість, автономність та інтерактивність рішення, придатного до подальшого розширення (наприклад, для біометричної автентифікації або візуального моніторингу);

- продемонстровано можливість повноцінної інтеграції глибокого навчання у клієнтське середовище веб-браузера.

Отримані результати можуть бути корисними для подальших досліджень у галузі комп’ютерного зору, веб-інтелектуальних систем, а також при розробці безпечних і приватних технологій розпізнавання в браузерах без сторонніх серверів.

В роботі доводиться, що використання глибоких нейронних мереж та сучасних методик може значно покращити точність прогнозування для задач виявлення, розпізнавання та виділення облич. Зокрема, глибоке розуміння та

взаємодія з нейронними мережами під час експериментів призвели до цінних спостережень.

Ця робота охопила ключові концепції, що застосовуються в глибокому навчанні, а саме механізми внутрішніх обчислень, що призводять до підвищення точності моделей для виявлення та розпізнавання облич. Висловлюємо подяку дослідникам у цій галузі, чиї відкриття та знахідки сприяли розвитку справжньої потужності нейронних мереж.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. T. Zhang, R. Wang, J. Ding, X. Li and B. Li, "Face Recognition Based on Densely Connected Convolutional Networks," 2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM), Xi'an, 2018, pp. 1-6.
2. Viola, P., & Jones, M. J. (2004). Robust real-time face detection. International Journal of Computer Vision, 57(2), 137-154.
3. Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91-110.
4. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
5. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (NIPS 2012), 25, 1097-1105.
6. Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In International Conference on Learning Representations (ICLR 2015).
7. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), 770-778.
8. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), 779-788.
9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In European Conference on Computer Vision (ECCV 2016), 21-37.

10. Howard, A. G., Zhu, M., Chen, B., Wang, D., Chen, T., Tan, M., ... & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.
11. Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), 4700-4708.
12. Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. IEEE Signal Processing Letters, 23(10), 1499-1503.
13. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), 815-823.
14. King, D. E. (2009). Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research, 10, 1755-1758.
15. Van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). scikit-image: Image processing in Python. PeerJ, 2, e453.
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
17. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 265-283.
18. Smilkov, D., Thorat, N., Assar, M., Abernethy, B., Viégas, F. B., & Wattenberg, M. (2019). TensorFlow.js: Machine Learning for the Web and Beyond. Journal of Open Source Software, 4(39), 1735.

19. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools, 25(11), 120-126.
20. Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), 1701-1708.
21. Sun, Y., Wang, X., & Tang, X. (2014). Deep Learning Face Representation by Joint Identification-Verification. In Advances in Neural Information Processing Systems (NIPS 2014), 27, 1989-1997.
22. Zhou, X., Han, X., Zhang, H., Cui, J., & Li, Z. (2019). Rethinking BiSeNet for Real-time Semantic Segmentation. arXiv preprint arXiv:1908.00767.
23. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (AISTATS 2010), 249-256.
24. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
25. Gehler, P. V., & Nowozin, S. (2009). On Feature Combination for Multiclass Object Detection. In IEEE 12th International Conference on Computer Vision (ICCV 2009), 221-228.
26. Donahue, J., Jia, Y., Hoffman, J., Darrell, T., & Saenko, K. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In International Conference on Machine Learning (ICML 2014), 647-655.
27. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009), 248-255.

28. Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 1, 886-893.
29. Cao, X., Shen, L., Xie, D., Park, S., & Li, Z. (2018). VGG-Face2: A Dataset for Recognising Faces Across Pose and Age. In 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018), 67-74.
30. Raghavendra, R., Raja, K. B., Busch, C., & Roli, S. (2017). Face Recognition in the Wild: A Review. IEEE Access, 5, 23724-23740.
31. Geman, S., & Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6(6), 721-741.
32. Eigen, D., Puhrsch, D., & Fergus, R. (2014). Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In Advances in Neural Information Processing Systems (NIPS 2014), 27, 2366-2374.
33. Wu, C., Zhang, N., Li, X., Huang, Y., & Li, B. (2015). Deep Learning for Face Attribute Prediction. arXiv preprint arXiv:1506.01235.
34. Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
35. Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), 1800-1807.
36. Denton, A., Chien, E., & Sutton, S. (2017). A Case Study of Model Compression for Deep Neural Networks. arXiv preprint arXiv:1710.09282.
37. Zhou, L., Shi, X., Zhang, M., & Yang, B. (2020). Real-Time Face Detection and Recognition Based on TensorFlow.js. In 2020 12th International Conference on Computational Intelligence and Communication Networks (CICN 2020), 405-409.

38. Bartlett, M. S., Movellan, J. R., & Sejnowski, T. J. (2002). Face recognition by independent component analysis. IEEE Transactions on Neural Networks, 13(6), 1450-1464.
39. Belongie, S., Malik, J., & Perona, P. (2002). Shape matching and object recognition using shape contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(4), 509-522.

ДОДАТКИ

Додаток А

Фрагменти програмних кодів

-- Фрагмент вихідного коду розпізнавання віку та статі

```
<!DOCTYPE html>
<html>
<head>
    <script src="face-api.js"></script>
    <script src="js/commons.js"></script>
    <script src="js/faceDetectionControls.js"></script>
    <script src="js/imageSelectionControls.js"></script>
    <link rel="stylesheet" href="styles.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.
</head>
<body>
    <div id="navbar"></div>
    <div class="center-content page-container">
        <div class="progress" id="loader">
            <div class="indeterminate"></div>
        </div>
        <div style="position: relative" class="margin">
            <img id="inputImg" src="" style="max-width: 800px;" />
            <canvas id="overlay" />
        </div>
        <div class="row side-by-side">
            <!-- image_selection_control -->
            <div id="selectList"></div>
            <div class="row">
                <label for="imgUrlInput">Отримати зображення з URL:</label>
                <input id="imgUrlInput" type="text" class="bold">
            </div>
            <button class="waves-effect waves-light btn" onclick="loadImageFromUrl()">
                Ok
            </button>
            <!-- image_selection_control -->
        </div>
        <div class="row side-by-side">
            <!-- face_detector_selection_control -->
            <div id="face_detector_selection_control" class="row input-field" style="margin-rig
                <select id="selectFaceDetector">
                    <option value="ssd_mobilenetv1">SSD Mobilenet V1</option>
                    <option value="tiny_face_detector">Tiny Face Detector</option>
                    <option value="mtcnn">MTCNN</option>
                </select>
                <label>Виберіть детектор обличчя</label>
            </div>
            <!-- face_detector_selection_control -->
        </div>
        <!-- ssd_mobilenetv1_controls -->
        <span id="ssd_mobilenetv1_controls">
            <div class="row side-by-side">
                <div class="row">
                    <label for="minConfidence">Мін. впевненість:</label>
                    <input disabled value="0.5" id="minConfidence" type="text" class="bold">
                </div>
                <button class="waves-effect waves-light btn" onclick="onDecreaseMinConfidence()
                    <i class="material-icons left">-</i>
                </button>
                <button class="waves-effect waves-light btn" onclick="onIncreaseMinConfidence()
                    <i class="material-icons left">+</i>
                </button>
```

-- Фрагмент вихідного коду виявлення обличчя

```
<!DOCTYPE html>
<html>
<head>
    <script src="face-api.js"></script>
    <script src="js/commons.js"></script>
    <script src="js/faceDetectionControls.js"></script>
    <script src="js/imageSelectionControls.js"></script>
    <link rel="stylesheet" href="styles.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.
</head>
<body>
    <div id="navbar"></div>
    <div class="center-content page-container">
        <div class="progress" id="loader">
            <div class="indeterminate"></div>
        </div>
        <div style="position: relative" class="margin">
            <img id="inputImg" src="" style="max-width: 800px;" />
            <canvas id="overlay" />
        </div>
        <div class="row side-by-side">
            <!-- image_selection_control -->
            <div id="selectList"></div>
            <div class="row">
                <label for="imgUrlInput">Отримати зображення з URL:</label>
                <input id="imgUrlInput" type="text" class="bold">
            </div>
            <button class="waves-effect waves-light btn" onclick="loadImageFromUrl()">
                Ok
            </button>
            <!-- image_selection_control -->
        </div>
        <!-- Image selection -->
        <div class="row side-by-side">
            <!-- image_selection_control -->
            <div id="selectList"></div>
            <div class="row">
                <label for="imgUrlInput">Отримати зображення з файлу:</label>
                <input id="myFileUpload" type="file" onchange="uploadImage()" accept=".jpg, .jp
            </div>
            <button class="waves-effect waves-light btn" onclick="loadImageFromUrl()">
                Ok
            </button>
            <!-- image_selection_control -->
        </div>
        <div class="row side-by-side">
            <!-- face_detector_selection_control -->
            <div id="face_detector_selection_control" class="row input-field" style="margin-rig
                <select id="selectFaceDetector">
                    <option value="ssd_mobilenetv1">SSD Mobilenet V1</option>
                    <option value="tiny_face_detector">Tiny Face Detector</option>
                    <option value="mtcnn">MTCNN</option>
                </select>
                <label>Виберіть детектор обличчя</label>
            </div>
            <!-- face_detector_selection_control -->
        </div>
        <!-- check boxes -->
```

-- Фрагмент вихідного коду розпізнавання виразу обличчя

```
<!DOCTYPE html>
<html>
<head>
    <script src="face-api.js"></script>
    <script src="js/commons.js"></script>
    <script src="js/faceDetectionControls.js"></script>
    <script src="js/imageSelectionControls.js"></script>
    <link rel="stylesheet" href="styles.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.
</head>
<body>
    <div id="navbar"></div>
    <div class="center-content page-container">
        <div class="progress" id="loader">
            <div class="indeterminate"></div>
        </div>
        <div style="position: relative" class="margin">
            <img id="inputImg" src="" style="max-width: 800px;" />
            <canvas id="overlay" />
        </div>
        <div class="row side-by-side">
            <!-- image_selection_control -->
            <div id="selectList"></div>
            <div class="row">
                <label for="imgUrlInput">Отримати зображення з URL:</label>
                <input id="imgUrlInput" type="text" class="bold">
            </div>
            <button class="waves-effect waves-light btn" onclick="loadImageFromUrl()">
                Ok
            </button>
            <!-- image_selection_control -->
        </div>
        <div class="row side-by-side">
            <!-- image_selection_control -->
            <div id="selectList"></div>
            <div class="row">
                <label for="imgUrlInput">Отримати зображення з файлу:</label>
                <input id="myFileUpload" type="file" onchange="uploadImage()" accept=".jpg, .jp
            </div>
            <button class="waves-effect waves-light btn" onclick="loadImageFromUrl()">
                Ok
            </button>
            <!-- image_selection_control -->
        </div>
        <div class="row side-by-side">
            <!-- face_detector_selection_control -->
            <div id="face_detector_selection_control" class="row input-field" style="margin-rig
                <select id="selectFaceDetector">
                    <option value="ssd_mobilenetv1">SSD Mobilenet V1</option>
                    <option value="tiny_face_detector">Tiny Face Detector</option>
                    <option value="mtcnn">MTCNN</option>
                </select>
                <label>Виберіть детектор обличчя</label>
            </div>
            <!-- face_detector_selection_control -->
        </div>
        <!-- check boxes -->
        <div class="row" style="width: 220px;">
            <input type="checkbox" id="hideBoundingBoxesCheckbox" onchange="onChangeHideBoundin
```

-- Фрагмент вихідного коду виявлення обличчя з веб-камери

```
<!DOCTYPE html>
<html>
<head>
    <script src="face-api.js"></script>
    <script src="js/commons.js"></script>
    <script src="js/faceDetectionControls.js"></script>
    <link rel="stylesheet" href="styles.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.
</head>
<body>
    <div id="navbar"></div>
    <div class="center-content page-container">
        <div class="progress" id="loader">
            <div class="indeterminate"></div>
        </div>
        <div style="position: relative" class="margin">
            <video onloadedmetadata="onPlay(this)" id="inputVideo" autoplay muted playsinline><
            <canvas id="overlay" />
        </div>
        <div class="row side-by-side">
            <!-- face_detector_selection_control -->
            <div id="face_detector_selection_control" class="row input-field" style="margin-rig
                <select id="selectFaceDetector">
                    <option value="ssd_mobilenetv1">SSD Mobilenet V1</option>
                    <option value="tiny_face_detector">Tiny Face Detector</option>
                    <option value="mtcnn">MTCNN</option>
                </select>
                <label>Виберіть детектор обличчя</label>
            </div>
            <!-- face_detector_selection_control -->
            <!-- fps_meter -->
            <div id="fps_meter" class="row side-by-side">
                <div>
                    <label for="time">Час:</label>
                    <input disabled value="-" id="time" type="text" class="bold">
                    <label for="fps">Оцінка FPS:</label>
                    <input disabled value="-" id="fps" type="text" class="bold">
                </div>
            </div>
            <!-- fps_meter -->
        </div>
        <!-- ssd_mobilenetv1_controls -->
        <span id="ssd_mobilenetv1_controls">
            <div class="row side-by-side">
                <div class="row">
                    <label for="minConfidence">Мін. впевненість:</label>
                    <input disabled value="0.5" id="minConfidence" type="text" class="bold">
                </div>
                <button class="waves-effect waves-light btn" onclick="onDecreaseMinConfidence()
                    <i class="material-icons left">-</i>
                </button>
                <button class="waves-effect waves-light btn" onclick="onIncreaseMinConfidence()
                    <i class="material-icons left">+</i>
                </button>
            </div>
        </span>
        <!-- ssd_mobilenetv1_controls -->
        <!-- tiny_face_detector_controls -->
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Розробка та реалізація системи глибокого навчання у веб-браузерах ”

Обсяг пояснювальної записки: 79 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____