МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів оптимізації алгоритмів розпізнавання обличчя для систем з обмеженими ресурсами

Виконав:
Студент 2 курсу групи ІПЗм-20-3
Садовников Б.І.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова
Керівник доц. Мазурова О.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ проф. Дудар З.В.

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студента_ Садовникова Бориса Ігоровича _

(Прізвище, ім'я, по батькові студентів)

1. Тема роботи Дослідження методів оптимізації алгоритмів розпізнавання обличчя для систем з обмеженими ресурсами.

затверджена наказом університету від 24.03.2022 № 412СТ

2. Термін здачі студентом закінченої роботи „_17__”___05_____ 2022 р.

3. Вихідні дані до проекту: Технічне завдання, календарний план, методичні вказівки –

4. Зміст розрахунково-пояснювальної записки: Вступ, аналіз проблемної області, аналіз аналогів, постановка задачі, дослідження алгоритмів розпізнавання облич, моделювання задачі розпізнавання при роботі у системах з обмеженими ресурсами, порівняння алгоритмів розпізнавання облич, аналіз методів оптимізації алгоритмів розпізнавання облич, опис програмної реалізації, опис експериментів, рекомендації по використанню досліджених оптимізацій ,висновок.

5. Перелік графічного матеріалу: *13 рисунків, 12 таблиці, 2 формули*.

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблемної області дослідження	20.01.2022	виконано
2	Розробка постановки задачі	06.02.2022	виконано
3	Дослідження методів розпізнавання облич	19.02.2022	виконано
4	Моделювання задачі розпізнавання в системах з обмеженими ресурсами	05.03.2022	виконано
5	Порівняння алгоритмів розпізнавання облич	07.03.2022	виконано
6	Аналіз методів оптимізації алгоритмів розпізнавання облич	18.03.2022	виконано
7	Програмна реалізація	21.03.2022	виконано
8	Експериментальне дослідження	06.04.2022	виконано
9	Підготовка пояснювальної записки	16.04.2022	виконано
10	Підготовка презентації та доповіді	05.05.2022	виконано
11	Попередній захист	12.05.22	виконано
12	Нормконтроль, рецензування	15.05.22	виконано
13	Занесення диплома у електроний архів	17.05.22	виконано
14	Допуск до захисту у зав. кафедри	22.05.22	виконано

Дата видачі завдання 17.01.2022 р.

Керівник доц. _____(Мазурова О.О.)

Завдання прийняв до виконання_____ (Садовников Б.І.)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 90 стор, 13 рис., 2 формули, 12 таблиці, 27 джерел.

EIGENFACES, FACENET, FISHERFACES, KERAS, NUMPY, PYTHON, TENSORFLOW, НЕЙРОННА МЕРЕЖА, ОПТИМІЗАЦІЯ, РОЗПІЗНАВАННЯ.

Об’єкт дослідження – існуючі методи розпізнавання облич, методи їх оптимізації.

Мета роботи – дослідити алгоритми для подальшої оптимізації для роботи на приладах з обмеженими ресурсами, дослідити самі методи оптимізації та провести експериментальне дослідження оптимізованих алгоритмів.

У результаті роботи був проведений аналіз методів розпізнавання облич та шляхів їх оптимізації, програмно реалізовані оптимізовані версії алгоритмів та проведені експерименти з ними, сформовано рекомендації щодо застосування оптимізацій.

EIGENFACES, FACENET, FISHERFACES, KERAS, NEURAL NETWORK, NUMPY, OPTIMIZATION, PYTHON, RECOGNITION, TENSORFLOW.

The object of research is the existing methods of face recognition and methods of its optimization.

The purpose of the work is to choose the best algorithms for further optimization to work on devices with limited resources, investigate optimization methods and conduct experiments.

As a result, the face recognition methods and optimization techniques were analyzed, implemented optimized versions of algorithms and performed experiments with it, created recommendation regarding usage of optimizations.

Я, Садовников Борис Ігорович, студент гр. ІПЗм-20-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів оптимізації алгоритмів розпізнавання обличчя для систем з обмеженими ресурсами», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Біометричні методи аутентифікації, такі як розпізнавання за обличчям, відбитком пальця, голосом набувають все ширшої популярності у сучасному світі. Нажаль, сучасні епідеміологічні реалії вимагають підвищення санітарних норм і шляхи аутентифікації, які потребують фізичного контакту стають менш кращими за безконтактні.

Метою цієї роботи є дослідження методів розпізнавання обличчя з метою вибору максимально ефективних для подальшої оптимізації і проведення експериментів, а саме перевірка точності, швидкості та надійності при роботі на пристроях з обмеженими ресурсами, таких як вбудовані пристрої.

Також необхідно дослідити методи оптимізації процесу розпізнавання облич та провести серії експериментів з метою визначити різницю між оптимізованими версіями, розгорнутими на вбудованому пристрої, та версіями без оптимізацій.

У останній час технології набули швидкого темпу розвитку, нажаль їх вплив на освіту залишився досить малим, що можна вважати істотним недоліком, оскільки це може створити відставання сфери освіти від ішних сфер. Гарною спробою це компенсувати є створення та використання комп’ютерних систем для автоматизації та підвищення якості освіти.

Практичною метою даного дослідження використання отриманих результатів експериментів з ціллю перевірка доцільності використання алгоритмів розпізнавання на IoT приладах та додавання отриманого модуля до вже існуючої системи контролю присутності та тестування працездатності в умовах максимально наближених до реальних.

Під час виконання магістерської кваліфікаційної роботи був проведений аналіз алгоритмів розпізнавання облич на основі публікацій світових та вітчизняних фахівців в проблемній області (див. додаток А), розглянуті основні існуючі методи розпізнавання облич, шляхи їх оптимізації, а також обрані найбільш актуальні методи для проведення дослідження. У результаті була

сформована постановка задачі, а також спроектовано весь процес проведення експериментального дослідження, включаючи реалізації оптимізацій обраних алгоритмів. Крім того, були описані умови та обмеження проведення експерименту, сформовані та обґрунтовані метрики, за якими порівнювалася ефективність роботи обраних алгоритмів після оптимізації .

Робота пройшла успішну перевірку на академічну доброчесність (див. додаток Б).

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В). Також за результатами роботи були написані тези доповіді на участі у науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» (див. додаток Г). Також були подані тези на міжнародну науково-технічну конференцію «Наукові досягнення та відкриття сучасної молоді»(див. додаток Г).

Також робота відповідає всім критеріям нормоконтролю (див. додаток Д).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області

Зараз освіта має ключову роль у існуванні суспільства, її якість є найважливішим фактором у подальшому розвитку та процвітання людства. Для того щоб зробити такі висновки, слід лише поглянути на темп розвитку технологій: поява машин з автопілотом, велика кількість космічних місій, ряд досліджень в галузях природних наук таких як фізика, біологія, хімія, екологія та інші. Нові покоління мають бути більш освіченим за попередників для того щоб і надалі рушити прогрес уперед.

Нажаль, таких, можна сказати, революційних покращень не спостерігається у освіті. Порівнюючі процеси навчання які були до приходи ери комп'ютерів з поточними, можна знайти дуже мало відмінностей. Перш за все це поява спеціальностей, пов'язаних з комп'ютерами. Але це ніяким чином не впливає на якість освіти загалом. Це лише розширення можливих напрямків навчання, що безумовно є покращенням, але кількісним оскільки з'являється велика кількість нових областей знань, досліджень, методологій. Тобто, нові знання дають новий матеріал на базі якого може грунтуватися навчання. Має сенс задуматися над покращення саме якісних характеристик освіти за допомогою нових технологій.

На даному етапі вплив технологій на освіту мінімальний: все лімітується використання комп'ютерів інколи для перевірки знань та демонстраційних занять з проектором. На мою думку, існує ряд областей, які ми можемо покращити за допомогою комп'ютерних систем та технологій.

Першим кроком до покращення освіти є поява онлайн платформ, які надають доступ до навчальних курсів, гарними прикладами є «Интуит»[1], «Coursera»[2], «Stepik»[3]. Ці ресурси надають велику кількість курсів, але їх слід класифікувати більше як додаткову освіту, оскільки деякі питання можуть бути розглянуті досить поверхнево і слабкий зв'язок з викладачами або його повна відсутність не дають можливості назвати це фундаментальною, основною освітою.

Іншим підходом до покращення освіти є автоматизації перевірки знань. Класичним прикладом такої системи є «OpenTEST»[4]. Цей підхід набув досить великої популярності і використовується у багатьох вищих навчальних закладах. Головною перевагою такого роду систем є можливість досить значної економії часу викладачів за рахунок відсутності необхідності перевіряти контрольні роботи студентів власноруч. І як приємний бонус студентам - вони одразу знають оцінку за таку роботу. Іншою перевагою є можливість додавати такого роду роботи до електронного документообігу без будь яких додаткових зусиль.

Але до оптимізації навчальних процесів можна підійти з іншого боку, а саме автоматизувати процес контролю відвідувань занять. Так чи інакше, відвідування є важливою метрикою особливо для студентів молодших курсів. Перевірка відвідування може забирати суттєву кількість часу занять, відволікати викладача від основної думки лекції, порушувати концентрацію студентів. Виходячи з цього, доцільно подумати про можливі варіанти вирішення проблеми.

Ключовим компонентом системи контролю відвідувань занять студентами є механізм розпізнавання хто саме прийшов на заняття.

Перше питання, на яке ми маємо відповісти це наскільки метод має бути надійним. Банальним рішенням є використання rfid карток або будь чого подібного, але таке рішення має обмеження у вигляді що робити, якщо забув або втратив картку. Іншим цікавим обходом є один студент, який принесе окрім своєї картки, картку друга і зареєструє відвідування студента, якого фактично немає на занятті. Аналогічним чином, свого роду соціальна інженерія, можна обійти досить багато систем, які базуються на використанні ключових предметів. Цей недолік можна виправити, якщо уважно слідкувати хто і як відмічається, але це іде на спротив ідеї автоматизації і навіть ускладнює процес.

Виходячи з попередніх міркувань, ми маємо розпізнавати присутніх за допомогою чогось, що неможливо передати будь кому. Зараз набула популярності ідея використання біометричних даних для аутентифікації у мобільних і не тільки приладах: комп’ютери які розблоковуються за допомогою відбитку пальця, телефони, які з фронтальної камери отримують зображення хто саме хоче

використати прилад і базуючись на розпізнаванні обличчя приймають рішення. Підхід з відбитком пальця є досить надійним, але існує декілька важливих недоліків. Перше це необхідність кожному студенту підійти до приладу та за допомогою відбитку відмітитися. Прилад може спрацьовувати не з першої спроби, що також збільшує витрати часу. Другий недолік - усі студенти мають доторкнутися до сканеру, що з огляду на поточну ситуацію у світі з пандемією не є добрим рішенням.

Наступний метод, на мою думку, найбільш підходящий - розпізнавання обличчя[5]. Зараз існує досить багато рішень які дозволяють виконати цю операцію, які ми розглянемо детальніше далі. Основним моментом є досить великі вимоги до ресурсів від такого роду алгоритмів. Звичайно під такі задачі виділяється сервер[6] з високими апаратними можливостями і на ньому централізовано виконуються всі обчислення(розпізнавання). Головним недоліком є обмеження у масштабованості такого рішення: з ростом кількості студентів, більша кількість спроб розпізнавання за одиницю часу, більша кількість пар, що у свою чергу теж збільшує навантаження на сервер. У кінці все може упертися у два фактору: неможливо підвищувати потужність серверу безкінечно та дуже велика вартість за таку потужність, яку бюджет навчальних закладів може не витримати.

Іншою проблемою є велике навантаження на мережу від постійної передачі зображень на сервер, що може негативним чином вплинути на затримки у отриманні даних з сервера та на процес навчання у цілому. Також завжди існує можливість викрадення фото студентів зловмисниками.

Обхідним методом є вертикальне масштабування - збільшення кількості обробників інформації, а не їх потужності. Дивлячись глибше, можна навіть реалізувати розпізнавання прямо на місці - обчислення виконуються приладом, який отримав фото. Таким шляхом можна одразу вирішити проблеми з навантаженням на мережу, підвищити безпеку системи завдяки зменшенню кількості інформації, яка передається на сервер та досить дешево адаптувати систему до високих навантажень. Оскільки алгоритми потребують досить велику кількість ресурсів, далеко не кожен вбудований пристрій може працювати з ними.

Головною метою цього дослідження є оптимізація алгоритмів розпізнавання облич до рівня можливості використання на IoT приладі. Важливо пам’ятати, що в обмін на ресурси, може знижуватися точність та швидкість роботи, тому необхідно знайти оптимальний баланс як у реалізації алгоритмів, так і тому, який прилад використовувати. Очевидно, що використання IoT приладу з мінімумом можливих ресурсів не принесе ніяких цінних результатів.

1.2 Аналіз аналогів

Спочатку розглянемо найпоширеніші рішення для розпізнавання облич - хмарні провайдери з їх сервісами обробки зображень. Наприклад, Amazon надає сервіс Amazon Rekognition який можна навчити розпізнавати визначену множину облич і потім відправляти зображення на обробку, у результаті якої ми маємо ідентифікатор того, кто на фото, якщо система знає цю людину.

При використанні хмарних сервісів ми підходимо до обмежень у вигляді необхідності передачі великої кількості зображень по мережі та досить високих цін за такі послуги. Іншим важливим недоліком є залежність від поточного стану сервісу: якщо щось сталось і сервіс не працює, ми ніяк не можемо вплинути на це.

Як перевагу, можна виділити відсутність необхідності займатися маштабуванням або будь якими діями стосовно менеджменту ресурсів для серверу розпізнавання. Усе буде виконано з боку хмарного провайдера. Наступним позитивним моментом є досить висока ефективність - алгоритми постійно покращуються і компанії, які надають такого роду сервіси, мають постійно покращувати свої реалізації.

Нажаль, варіант з хмарою не дозволяє перенести обчислення на IoT прилад, лише винести його на окремі віртуальні машини, про деталі роботи яких ми нічого не знаємо. Крім хмарних рішень, існують образи, утіліти, сервіси які можна

розвертати на своєму залізі, а вони у свою чергу виконують розпізнавання. Гарним прикладом такого рішення є Facebox [7] (див. рис. 1.1).

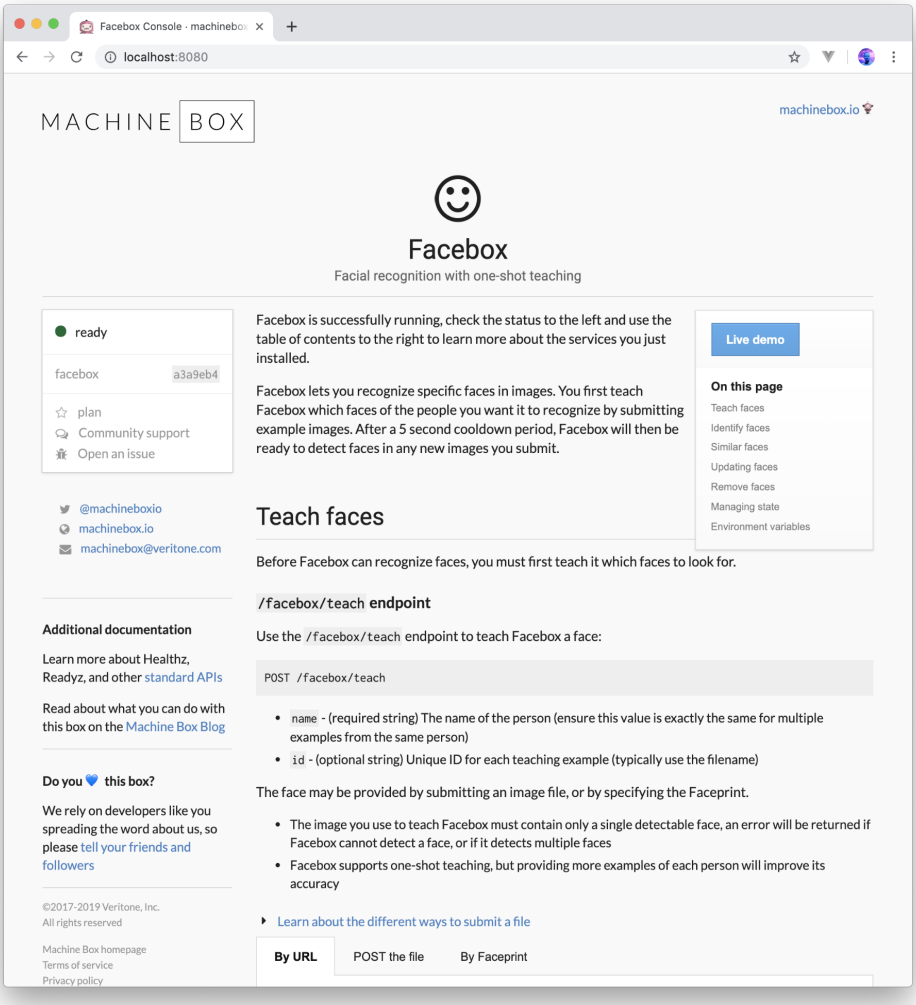

Рисунок 1.1 - Консоль керування Facebox

Суть рішення полягає у наданні користувачеві контейнеру, який автономно працює на виділеному порті та надає REST [8] API [9] для використання його функціоналу. Фактично, це чорний ящик якому подаються на вхід дані, а на виході маємо висновок щодо людини на фото.

З точки зору інтеграції такого роду сервісу до системи, маємо два важливих ендпоинти - для навчання розпізнаванню конкретного обличчя та обробка фото на предмет знайомих облич.

На відміну від хмарних рішень, ми можемо розгорнути контейнер на вбудованому приладі і використати його ресурси для обчислень, але такого роду сервіси мають вимоги до системи для коректного функціонування. Наприклад сам Facebox вимагає як мінімум 4 ГБ оперативної пам’яті що досить значна цифра для IoT приладу.

Тут можна побачити основний недолік використання вже готового контейнеру - ми ніяк не можемо кастомізувати алгоритм, змінити співвідношення точність-ресурси-швидкість або тюнити модель під нашу задачу.

Іншим недоліком є необхідність самостійно оновлювати версію контейнера на кожному девайсі, де він встановлений. Оскількі існує вірогідність, що поточна реалізація, наприклад, має даякі суттєві недоліки і у разі необхідності терміногового оновлення система може перестати працювати деякий час.

Для практичної частини роботи буде доцільно перейняти практику ізоляції сервіса розпізнавання у контейнері заради спрощення встановлення на цільові прилади. Таким чином, в результаті роботи маємо отримати контейнер з реалізацією алгоритму розпізнавання облич оптимізований для роботи на IoT приладі.

1.3 Постановка задачі

Метою роботи є дослідження алгоритмів розпізнавання обличчя та методів їх оптимізації для роботи у системах з обмеженими ресурсами.

Як результат дослідження, буде обрано кращі алгоритми, проведена їх оптимізація та серія експерементів, яка продемострує різницю у точності, використанні ресурсів між звичайними та оптимізованими версіями алгоритмів.

Перед нами стоїть задача, провести математичне моделювання алгоритму розпізнавання облич, розробленого для роботи у системах з обмеженими ресурсами та провести його дослідження, дослідити вже існуючі алгоритми та

обрати з них кращі за сукупністю критеріїв. Після цього, проаналізувати методи оптимізації, які можна використати для цих алгоритмів. Практичною частиною дослідження будуть експерименти з отриманими алгоритмами. Результатом роботи будуть вихідні дані експериментів за якими можна зробити висновки стосовно ефективності досліджених оптимізацій при їх використанні на методах розпізнавання облич та доцільності використання оптимізованих алгоритмів на вбудованих приладах.

Отже, під час магістерського дослідження необхідно вирішити наступні задачі:

- провести аналіз існуючих сервісів та алгоритмів розпізнавання облич; обрати найкращі для подальшого дослідження;
- провести математичне моделювання задачі розпізнавання при роботі у системах з обмеженими ресурсами;
- провести аналіз методів оптимізації алгоритмів розпізнавання облич;
- розробити оптимізовані версії обраних алгоритмів та програмно реалізувати їх;
- провести експериментальні дослідження щодо ефективності нових алгоритмів без обмежень у ресурсах та в умовах обмежених ресурсів
- за результатами експериментів розробити рекомендації, щодо використання розглянутих методів оптимізації до алгоритмів розпізнавання облич.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз алгоритмів розпізнавання облич

Розглянемо кроки, які необхідно виконати для розпізнавання обличчя. Перш за все необхідно отримати фото з зображеним на ньому обличчям. Для цього використовується зовнішній відео модуль на IoT приладі або будь яка вбудована камера. Для нашого дослідження буде використовуватися частково дані з камери IoT приладу, частково вже готові датасети з обличчями, такі як, наприклад, Flickr-Faces-HQ Dataset, VGGFace2 (див. рис. 2.1), CASIA-Webface.

Рисунок 2.1 - Приклад зображень з датасету VGGFace2

Важливо позначити, що для максимальної ефективності роботи у реальних умовах, обличчя у датасеті мають бути у різних позах, при різному рівні освітлення та позах.

Наступним кроком є обробка зображення.

2.1.1 Обробка зображення

Спочатку необхідно виділити з фото саме обличчя людини. Для цього доцільно використовувати метод Віоли-Джонса розроблений у 2001 році. Головними перевагами цього методу є дуже високий відсоток істинно позитивних розпізнань, з низьким числом хибно позитивних спрацювань, спрямованість на використання у системах реального часу й обробці облич на відео. Метод Віоли-Джонса[10] розроблений спеціально для пошуку облич на зображенні, його основною метою є виділення облич поміж інших об’єктів, ніяких дій щодо розпізнавання конкретної особи у алгоритмі немає. Серед головних недоліків слід позначити досить низьку ефективність при розпізнаванні великої кількості облич на фото, але це не вплине на поточне дослідження.

Реалізація алгоритму Віоли-Джонса складається з двох кроків: підрахунку інтегрального уявлення зображення та знаходження ознак Хаару на ньому. У алгоритмі необхідно досить часто підраховувати суму пікселів зображення, оскільки обличчя знаходиться за допомогою пошуку темних та світлих частин обличчя. Інтегральне уявлення допомагає значно зменшити кількість обчислень оскільки, дозволяє швидко обчислювати суму пікселів випадкових прямокутних регіонів зображення. Таким чином, цей метод вже є досить оптимізованим для використання на вбудованому пристрою.

Ознаки Хаара[11] - ефективні маркери, які позначають наявність обличчя на фото. Вона складається з декількох суміжних прямокутних регіонів. Вони позицюються за зображенні, потім сумуються інтенсивності пікселів у цих областях, після чого обчислюється різниця між цими сумами. Ця різниця і є ознакою Хаара.

Ключовою особливістю цього алгоритму є саме його швидкість. Через використання інтегральної форми зображення, ознаки Хаара можуть бути підраховані практично за константний час для зображення будь якого розміру.

На рисунку 2.2 можемо побачити як деякі з цих ознак розташовуються на обличчі.

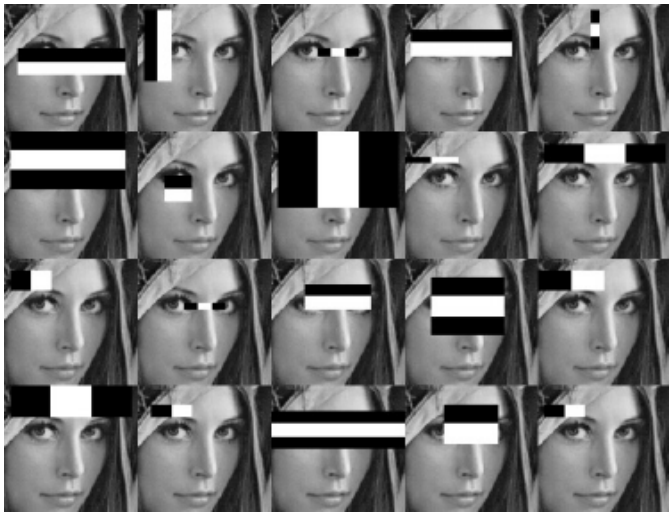

Рисунок 2.2 - Ознаки Хаара на обличчі

Слід окремо виділити метод навчання AdaBoost та каскадну архітектуру класифікаторів які є частиною реалізації методу Віоли-Джонса.

AdaBoost - алгоритм машинного навчання, який використовує множину слабких класифікаторів для підвищення їх ефективності. Підсилення відбувається за рахунок об’єднання класифікаторів у групи, і кожна наступна група базується на об’єктах, які неправильно були класифіковані попередньою групою.

Каскадна архітектура класифікаторів дозволяє значно прискорити роботу алгоритму за рахунок зменшення кількості операцій класифікації. У каскадній архітектурі множина класифікаторів або фільтрів застосовуються до об’єкта один за одним, у випадку якщо на поточні ітерації класифікатор не знайшов необхідну особливість, обробка зображення припиняється. Таким чином, більша частина часу витрачається на зображеннях де потенційно може бути обличчя, і зменшуються обчислювальні витрати на випадки, де з самого початку очевидна відсутність обличчя.

Крім алгоритму Віоли-Джонса існує інший досить поширений метод пошуку та виділення обличча з зображення – згорткова нейрона мережа MTCNN[12]. MTCNN має дві основні задачі – пошук обличча та його виділення з вирівнюванням.

MTCNN виконує пошук та виділення зображення у три кроки, кожен з яких виконується за допомогою окремої згорткової нейронної мережі. Спочатку вхідні дані обробляються, а саме будується піраміда зображень з копій вхідного фото різних розмірів. На першому кроці згорткова модель знаходить можливі місця, де може бути обличчя та передає ці данні. Наступна модель більш детально обробляє області, які знайшла перша модель, відкидає хибно-позитивні області, формує рамки для знайдених облич. Третя модель виконує аналогічну другій функцію, але ще з більшій точністю та знаходить такі деталі як позиції очей, носа, роту.

За звичай, область обличча MTCNN відрізняється від результата алгоритму Віоли-Джонса (див. рис.2.3).

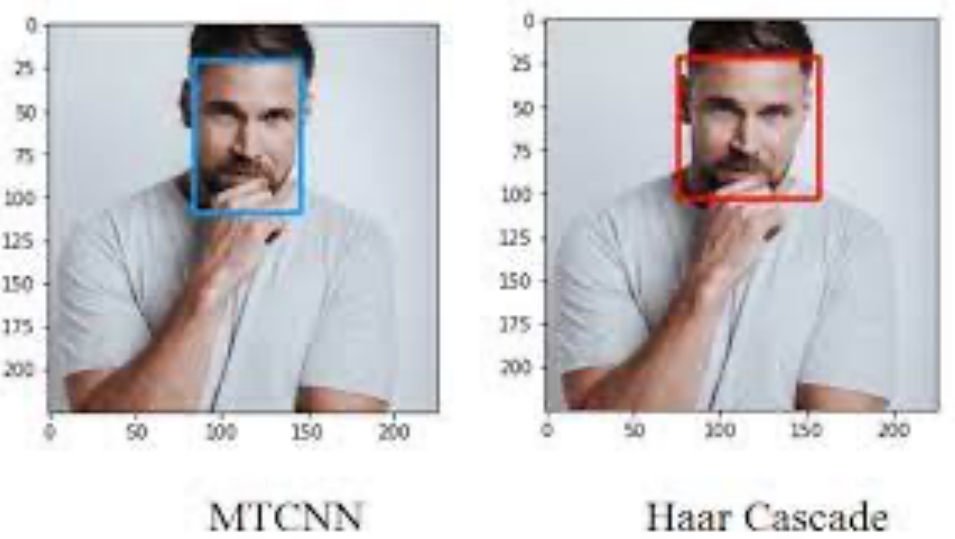

Рисунок 2.3 – Різниця між MTCNN та алгоритмом Віоли-Джонса

Різниця у формі області обличча може змінити кінцеву точність алгоритму, тому має сенс робити вимірювання точності з використанням обох алгоритмів.

Однак MTCNN використовує більше ресурсів системи та працює довше за алгоритм Віоли-Джонса.

Після отримання зображення з обличчям опціональною трансформацією[13] є зміна розміру, її необхідність обумовлюється у першу чергу вимогами подальших кроків алгоритму. Далі отримане фото нормалізується, приводиться до відтінків сірого для спрощення подальших обчислень, оптимізації використання CPU та пам’яті.

Розглянемо етап безпосереднього розпізнавання обличчя на фото. Існує декілька підходів до цієї проблеми які відрізняються тим, як саме вони аналізують зображення, що на ньому шукають та на основі чого базується їх висновок щодо схожості людей на фото:

- цілісний підхід;
- на основі особливостей;
- змішаний підхід.

Розглянемо кожне сімейство окремо.

2.1.2 Цілісний підхід до розпізнавання обличчя

Перше сімейство методів використовує цілісний підхід для розпізнавання облич. Його ще називають базуючимся на зображенні. Суттю цього шляху є використання всього регіону обличчя для розпізнавання. У цих методах обчислюється ряд особливостей для кожного окремого обличчя базуючись на значень пікселів по усій площині. Отже, обличчя представляється як вектор особливостей. Для розпізнавання ми маємо порівняти вектора особливостей двох облич та в залежності від відстані між ними зробити висновок про схожість. Таким чином, усе обличчя є вводом для алгоритму. Головним недоліком такого підходу є можливість обробляти лише невеликі за розміром зображення, оскільки чим більший розмір, тим більша кількість пікселів, які приймають участь у обчисленні особливостей та можуть бути шумом. Досить мала кількість особливостей, які можуть бути залежні від усього зображення, призводить до

зменшення кількості відмін між обличчями різних людей, що у свою чергу призводить до збільшення числа хибно-позитивних результатів.

Розглянемо алгоритм розпізнавання FisherFaces (див. рис. 2.4), який відноситься до родини методів з цілісним підходом.

Рисунок 2.4 - Обличчя після трансформації у FisherFaces

Цей метод є базується на алгоритмі EigenFaces (див. рис. 2.5), і є його прокращеною версією. У EigenFaces використовується метод головних компонентів для видалення інформації, яка не буде корисною для диференціації облич, тобо проводиться зменшення розмірності вхідних даних та їх більш ефективне уявлення. В той час, у FisherFaces ту саму задачу виконують за допомогою лінійного дискримінантного аналізу, який більше підходить для задачі класифікації оскільки намагається сформувати лінійну комбінацію ознак таким чином, що буде описувати або розділяти два або більше класів об’єктів. Таким

чином, досягається більша точність у випадках з різним рівнем освітлення та виразами обличчя.

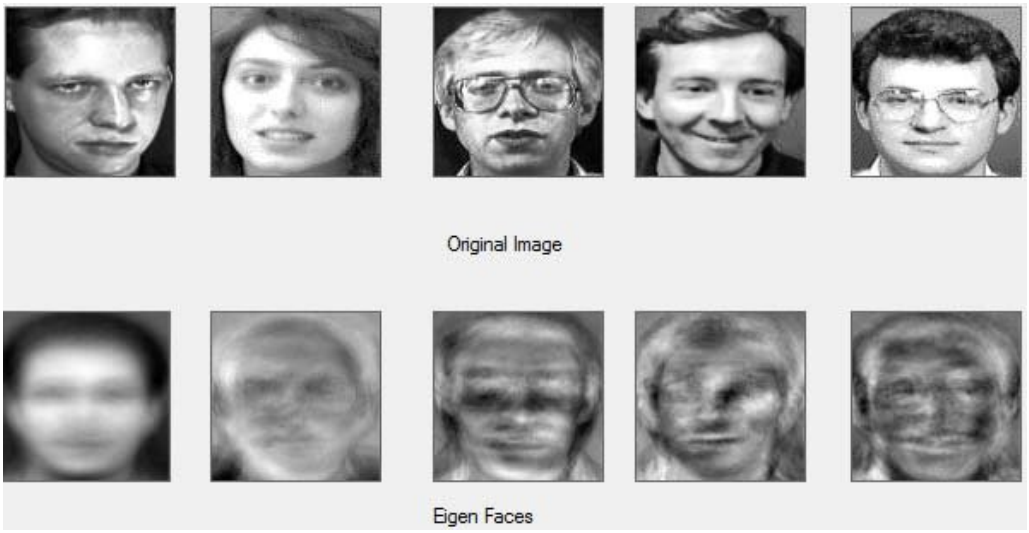

Рисунок 2.5 - Порівняння оригінального та трансформованого до EigenFaces зображення

2.1.3 Розпізнавання на основі особливостей обличчя

Друге сімейство методів - розпізнавання на основі особливостей обличчя. На відміну від першого підходу, аналізуються лише деякі ділянки обличчя, такі як очі, нос, область біля роту, відстань між очима, форма частин обличчя і так далі. Беручи до уваги ці відміни, можна розділити ці методи на статистичні та структурні, де статистичні беруть за основу аналіз особливостей всього обличчя, а структурні - обробляють форму та структуру деяких обраних ділянок, які як правило можуть суттєво відрізнятися у різних людей.

Під час розпізнавання за допомогою особливостей, спочатку на зображення накладаються фільтри щоб знайти регіони, які слід аналізувати, далі проводиться обчислення числових значень особливостей. Оскільки кількість підрахунків мінімізується через фільтрацію, ці алгоритми досить швидко та ефективно

виконують задачу, але у випадку, коли не вдалося знайти необхідні ділянки обличчя точність може значно впасти.

Найбільш простим прикладом алгоритма розпізнавання облич на основі особливостей є локальні бінарні шаблони. Метод базується на розділенні зображення на малі регіони, наприклад 16x16 пікселів і далі порівнювати пікселі з сусідніми. Якщо значення пікселя більше сусіднього, записується 0, інакше 1. У результаті таких трансформацій отримується бінарне число, яке використовується у подальшій обробці, наприклад обчислення гістограму по частоті появи кожного числа і використати як вектор ознак конкретного обличчя(див. рис. 2.6). Як варіацію, можна шукати конкретні регіони, такі як область очей, носа та підраховувати шаблони там. Завдяки такому підходу з’являється можливість не лише отримувати інформацію про особливості, а і їх точне положення на обличчі.

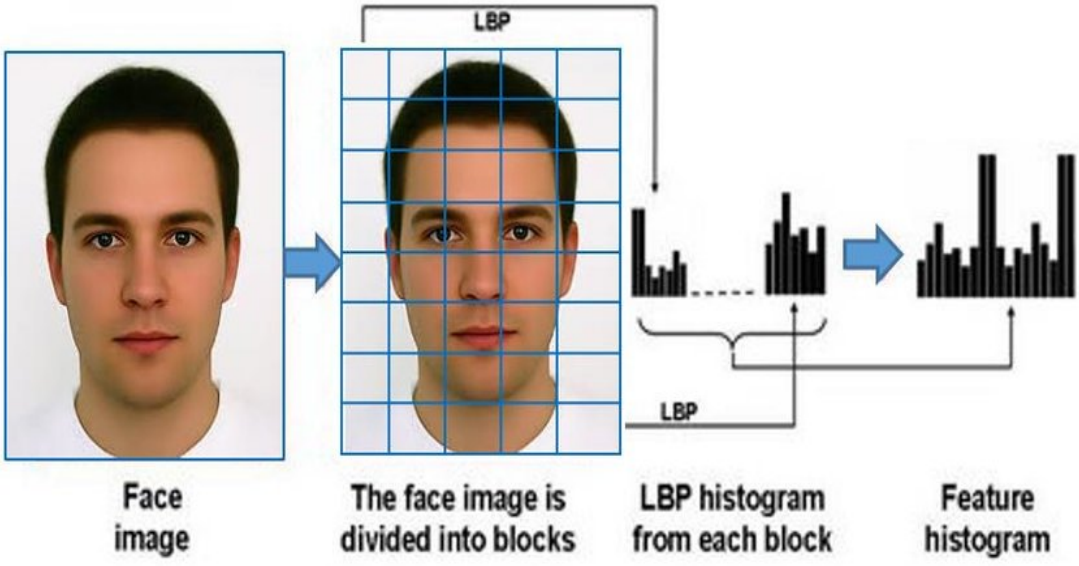

Рисунок 2.6 - Приклад аналізу зображення за допомогою методу локальних шаблонів

Серед недоліків слід зазначити вплив шумів на ефективність розпізнавання. Оскільки значення особливості залежить від конкретного значення пікселю та його оточення, будь яка зміна значень у регіони може змінити остаточне значення особливості. Для боротьби з цією проблемою був розроблений метод локальних

тернарних шаблонів, який ділить значення пікселів на 3 категорії: 1,0 та -1, таким чином отримана гістограма буде більшою за розміром, але більш інформативною. Іншим недоліком є неможливість захопити великі, глобальні особливості обличчя, які за розміром більше за блок, на який ділиться зображення. Цей недолік можна обійти, використовуючи пошук конкретних регіонів.

2.1.4 Змішаний підхід до розпізнавання обличчя

Наступним шляхом до розпізнавання є змішаний підхід, який комбінує два попередніх. Тобто частина особливостей обчислюється на рівні локальних, а інша частина - статистичні дані щодо всього обличчя. Найбільш актуальними представниками цього сімейства методів на сьогодні є згорткові нейроні[14] мережі.

Згорткові нейроні мережі – клас глибинних штучних нейронних мереж у яких за основу використовується схема з’єднання нейронів зорової кори тварин. Цей клас нейронних мереж демонструє свої кращі результати у задачах пов’язаних з аналізом зображень.

Розглянемо принципи роботи згорткових[15] нейронних мереж та спробуємо порівняти їх з класичними алгоритмами комп’ютерного зору. Почнемо з такого поняття як лінійна фільтрація - обчислення лінійної комбінації[16] значень пікселів у деякому просторі(вікні) з врахуванням матриці вагів фільтра або маски. Під час виконання цієї операції область фільтрації переміщується по усій площині зображення і таким чином зображення обробляється цілком. У результаті виконання операції отримуємо згортку зображення. Ця операція є основною у багатьох алгоритмах обробки зображень, головними відмінами є розмір та ваги фільтрів, кроки попередньої обробки зображень. У свою чергу, основним блоком згорткової нейронної мережі є шар згортки, який використовує ядро згортки або згортковий фільтр для обробки вхідних даних шляхом скалярного добутку. Таким

чином, базові концепції цих підходів збігаються, але існують деякі важливі відміни, які зіграли вирішальну роль у популярності та перспективності цих методів роботи з зображеннями.

Роботу згорткової нейронної мережі можна інтерпретувати як виділення з зображення максимально конкретних особливостей, потім інших, більш абстрактних деталей, і так далі до тих пір, поки не отримаємо задовільний результат. Результатом роботи мережі можуть бути деякі аналітичні дані, такі як що саме знаходиться на зображені, згенерований опис зображення елементами на ньому та подібне. Крім таких досить високорівневих результатів, можна отримати, наприклад, числове уявлення зображення, яке обчислюється за допомогою особливостей вилучених за допомогою згорткової мережі. У випадку з розпізнаванням облич таке уявлення ще називається векторним виглядом обличчя, вектором обличчя. Досить часто, саме особливості зображення використовуються як його числове уявлення. Також можливі деякі математичні трансформації.

Не залежно від того, який результат повертає модель, спочатку її слід навчити на навчальному наборі даних. Навчання є одним із найважливіших факторів, від якого залежить чи буде мережа виконувати свої функції, на яких даних вона буде функціонувати більш ефективно, яка попередня обробка зображення необхідна для використання на ньому мережі.

Розглянемо як відбувається процес навчання моделі. Під час тренування ваги моделі змінюються та підлаштовуються для виконання конкретної задачі, яка ставиться за допомогою відповідного навчального датасету[17]. У випадку зі згортковими нейронними мережами, ваги виконують роль значень фільтрів для кожного шару згортки. Таким чином, підходимо до важливої відміни згорткових нейронних мереж від класичних алгоритмів комп’ютерного зору: у випадку з алгоритмами значення всіх фільтрів визначається людиною, яка виконує програмну реалізацію, у той час як нейрона мережа самостійно знаходить значення фільтрів, які дозволять максимально ефективно обробляти вхідні дані. Під ефективністю мається на увазі можливість виділяти такі дані з оброблюємих зображень, що дозволить зробити точний висновок(класифікувати, описати,

знайти об’єкт) стосовно них. Завдяки цьому згорткові нейроні мережі знайшли дуже широке використання, оскільки відкривають можливість перевикористовувати одну архітектуру моделі для вирішення широкого спектру задач. Єдиним недоліком є необхідність повторного навчання на нових даних для кожної нової задачі та безпосередньо формування датасетів під задачі. Слід позначити, що модель можна донавчати або корегувати для нової задачі, якщо вхідні дані не сильно відрізняються від попередніх.

Подивимось на процес навчання на більш високому рівні. Метою навчання є, фактично, проаналізувати множину даних для тренування щоб знайти фактори, зв’язки, особливостями за якими зображення можна однозначно класифікувати один між одним. У випадку з обчисленнями, такими як вектор обличчя – знайти такі особливості, які для одного обличчя будуть максимально схожі на будь-якому зображенні, але відмінні від якогось іншого обличчя. Таким чином модель підлаштовується до вхідних даних, що і є головною відміною від класичних алгоритмів.

Для задачі розпізнавання облич як варіант з числовим уявленням, так і варіант з лейблом конкретної людини на фото є прийнятними з точки зору теорітичної можливості використання. З практичної сторони оптимальним варіантом є отримання числового уявлення, оскільки така модель повертає вектор особливостей обличча не залежно від того, хто саме на зображенні. Якщо модель повертає лейбл особи на фото, то вона має мати можливість розпізнати хто саме на фото. Для цього, перш за все, у процесі навчання необхідно надати множину людей, яких модель має розпізнавати. У противному випадку, модель просто не матиме інформації стосовно який лейбл привласнити особі. При додаванні нового обличча до моделі постає необхідність донавчати модель оскільки необхідно щоб вона мала можливість повернути на один лейбл більше. Також, суттєвим недоліком такого шляху є необхідність у досить великій кількості зображень кожної людини для того щоб навчити модель її розпізнавати. У випадку з моделями, які повертають числове уявлення, таких проблем немає, оскільки у цих двох конкретних ситуаціях мережі мають дещо різні задачі: одна з моделей має

знайти особливості та залежності на зображенні, які є максимально унікальними та вилучити їх; у той час як задача іншої моделі – навчитись вирізняти об’єкти серед множини подібних об’єктів. Тобто перша задача є більш загальною і має мінімальну прив’язку до навчальних даних: знайдені залежності і особливості можна використовувати для будь-яких зображень, але результат може бути не задовільним якщо тестові дані ніяк не пов’язані з навчальною вибіркою. Наприклад, модель для вилучення особливостей облич може показати низьку точність якщо її застосувати на зображеннях автомобілів або тварин.

Зазвичай, для роботи з зображеннями за допомогою алгоритмів комп’ютерного зору необхідно досить серйозні кроки передобробки. Причому для кожного окремого алгоритму ці кроки можуть відрізнятися. При практичному використанні можуть виникнути труднощі пов’язані з цим, якщо, наприклад, потрібно застосувати низку алгоритмів і кожен потребує якось по іншому передоброблене зображення. Нейроні мережі допомагають обійти таке обмеження оскільки можуть пристосовуватись до вхідних даних і самостійно, під час тренування, підбирати такі значення вагів для фільтрів, що складна попередня обробка не потрібна. За звичай, для використання у нейронній мережі достатньо змінити розмір зображення та у разі необхідності видалити зайві канали або перевести до градацій сірого.

Кожна згорткова нейрона мережа складається з шарів, у кожному шарі знаходиться певна кількість нейронів. Результат роботи одного шара передається наступному шару, тобто шари деяким чином з’єднані один між одним. Сукупність такого роду рішень як кількість шарів, нейронів у шарі, зв’язками між шарами формують архітектуру мережі. Програмна реалізація будь якої архітектури можна називати моделлю.

Для кожного класу задач існує модель, яка надає найбільш точний результат. Можливо використовувати одну модель для будь якої задачі, але у такому випадку точність може варіюватися від задачі і у деяких випадках, модель не зможе виконувати коректно поставлену задачу.

Окрім задачі, для рішення якої спроектовано модель, існує низка інших відмінностей. Кожна архітектура в залежності від кількості шарів, нейронів у шарах, шляху з’єднання шарів, розміру вхідних даних, вимог до результату роботи потребує різний час для навчання, кількість тренувальних даних, ресурси для роботи, розмір моделі. Наприклад, модель для бінарної класифікації того, що на зображенні, може вимагати 200 МБ пам’яті для роботи. У той час як більш складний класифікатор – 2 ГБ.

Прикладом архітектури для вирішення задачі розпізнавання обличчя є FaceNet. Існує навчена модель FaceNet, яка отримує на вхід зображення, а на виході видає вектор його особливостей. Важливою перевагою саме цієї моделі є те, що вона навчається лише раз, і її можна використовувати без обмежень. Приблизний час навчання такої моделі на спеціалізованому обладнанні – 30-40 годин. Як датасет можна використовувати, наприклад, VGGFace2. Це досить велика за розміром модель, яка вимагає для своєї роботи приблизно 4 ГБ оперативної пам’яті, що досить велике число для вбудованого пристрою. Використати такого роду мережу на IoT приладі можна за умови її попередньої оптимізації. Ця модель є прикладом підходу до розпізнавання з використанням числового уявлення обличчя.

Наступним варіантом є класифікатор[18] який базується на згортковій нейронній мережі. Спочатку його слід навчити розпізнавати множину облич, навчання має проходити на зображеннях людей, яких у майбутньому класифікатор має розпізнавати. Важливо зазначити, що при додаванні нового обличчя, класифікатор треба навчати заново або донавчити, що не є оптимальним, оскільки у цей час система не працює і навчання потребує ресурсів. Такого роду класифікатор є другим розглянутим вище підходом, нейроні мережі які класифікують і надають лейбл вхідним даним.

Слід позначити, що у випадку системи контролю відвідувань, нові студенти мають з’являтися раз у рік, що значно зменшує витрати на перенавчання, плюс є можливість навчати класифікатор поза учбовим процесом.

2.2 Вибір алгоритмів розпізнавання обличчя для подальшого дослідження

Для порівняння методів розпізнавання обличчя доцільно використати метод лінійної адитивної згортки (2):

$$Z^* = max \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}, \quad (1)$$

де α_j – нормуючі множники, β_j – вагові коефіцієнти.

Розглянемо крітерії за якими будемо порівнювати алгоритми:

а) точність розпізнавання обличчя, оцінка від 1 до 5;

б) стійкість до шумів(різне освітлення, положення обличчя, кути фото, вирази), оцінка від 1 до 5;

в) необхідність навчання:

1) не треба навчати – 5 бали;
2) навчати лише один раз – 3 бали;
3) навчати при додаванні нового обличчя – 1 бал.

г) потенціал до оптимізації – який простір існує в алгоритмі для внесення змін націлених на зменшення використання ресурсів при умові працездатності алгоритма, оцінка від 1 до 5.

У порівнянні розглядаємо як методи з використанням нейронних мереж, так і класичні алгоритми. Отримаємо наступну таблицю (див. табл. 1):

Усі дані знаходяться у одній шкалі, тому нормалізувати їх немає сенсу. Далі проаналізуємо які крітерії для нас біль важливі на основі аналізу оберемо вагові коефіцієтнти. Для цього використаэмо пропорційний метод.

Таблиця 1 – Алгоритми розпізнавання облич з критеріями порівняння

	Точність	Стійкість	Необхідність навчання	Потеціал до оптимізації
EigenFaces	3	2	5	3
FisherFaces	3	3	5	3
Локальні бінарні шаблони	3	2	5	4
Локальні тернарні шаблони	4	3	5	4
FaceNet	5	5	3	2
Класифікатор на основі CNN	3	3	1	3

Як пригадувалось раніше, навчання раз або два у рік не є дуже високою ціною, тому слід зменшити вплив цього фактору з 25% до 15%. У свою чергу, точність і стійкість впливають на результат роботи системи, але частково ми можемо створити умови з мінімальною кількістю шумів, наприклад у аудиторіях завжди достатній рівень освітлення для роспізнання. З огляду на це, можемо до точності додати ще 5% з необхідності навчання і отримати вплив цього фактору 30%. Крім того, важливим є можливість оптимізації алгоритму, тому додамо до цього фактору ще 5% і отримаємо 30% впливу, як у точності. Побудуємо таблицю з урахуванням вагових коефіцієнтів та виконаємо згортку(див. табл. 2):

Як бачимо з порівняння, найбільш цікавими є метод базуючийся на використанні нейронної мережі FaceNet.

Таблиця 2 – Алгоритми розпізнавання облич з вагами та результатом згортки

	Точність	Стійкість	Необхідність навчання	Потеціал до оптимізації	Z^*
EigenFaces	3	2	5	3	3.05
FisherFaces	3	3	5	3	3.3
Локальні бінарні шаблони	4	3	5	4	3.65
Локальні тернарні шаблони	4	4	5	4	3.9
FaceNet	5	5	3	3	4.1
Класифікатор на основі CNN	3	3	1	3	2.7
Вагові коеф	0.3	0.25	0.15	0.3	

2.3 Моделювання задачі розпізнавання при роботі у системах з обмеженими ресурсами

Розглянемо загальні кроки необхідні для розпізнавання людини на фото за допомогою нейронної мережі FaceNet. На вхід ми отримуємо оброблене зображення: нормалізоване, необхідного нам розміру та містить лише обличчя, яке ми маємо розпізнати. В залежності від вхідного зображення змінюється результат функції, причому для одного аргументу завжди має бути однаковий результат. З цього випливає, що модель є детермінованою.

Після отримання зображення, ми маємо виділити особливості(наприклад, векторне уявлення обличчя), за якими будемо визначати хто на фото. Існує ряд алгоритмів, які дозволяють це робити, але у випадку нашої математичної моделі, спростимо конкретні деталі реалізації до загального алгоритму підрахунку числових значень особливостей обличчя.

Останнім кроком алгоритму є порівняння отриманої числової інформації про обличчя з даними, відомих нашій системі облич, та пошуку співпадіння. Для цього необхідно використовувати алгоритм порівняння, гарним прикладом якого є Евклідова відстань. Згідно цієї метрики, чим менше її значення, тим більше два обличчя схожі один на одного. Існує ряд інших алгоритмів, які можна спробувати у експериментах для пошуку найшвидшого та точного рішення, але на етапи моделювання має сенс спростити та уніфікувати модель, використовуючи абстрактний алгоритм пошуку відстані між обличчями.

Алгоритм буде працювати у операційній системі з постійною апаратною конфігурацією, яка не буде змінюватися протягом роботи алгоритму, з цього випливає стаціонарний режим функціонування.

Розглянувши основні кроки алгоритму, можна побудувати рівняння для математичної моделі.

Нехай – E – функція для трансформації зображення обличчя до числового вигляду, C – функція пошуку відстані між двома числовими уявленнями обличчя, маємо наступне рівняння:

$$f(x) = min\ (C(E(x), m1\ ), C(E(x), m2\ ), \dots, C(E(x), mn\ ), p) \qquad (2)$$

де mn – числовий вигляд відомого обличчя;

n – кількість відомих обличь;

p – порогове значення, яке дозволяє визначити, що обличчя невідоме.

З даного рівняння випливає, що швидкість роботи алгоритму залежить від кількості відомих обличь у базі даних[19] приладу. Тобто, чим менше кількість

відомих обличь, тим швидше буде працювати алгоритм, оскільки менше операцій обчислень відстані та порівнянь. Слід зазначити, для кожної пари методів обчислення числового уявлення обличчя та відстані між двома обличчями своє співвідношення.

Виділимо предмет дослідження з навколишнього середовищем та дослідимо їх точки взаємодії (див. рис. 2.7).

Рисунок 2.7 – Модель предмету дослідження у вигляді «чорного ящика»

Одразу виділимо чінники крім вхідних даних, які можуть впливати на нашу модель. Перш за все це може бути операційна система, на якій працює додаток. Існує досить багато шляхів, які можуть змінити поведінку алгоритму: версії встановлених у системі бібліотек, наявність та доступ до необхідних для коректної роботи API, наявність необхідних привілеїв для використання ресурсів.

Іншим важливим фактором є апаратна складова приладу, на якому система буде працювати. Тут є головним розрядність процесору та підтримка їм додаткових наборів інструкцій, наявність дискретної або інтегрованої відеокарти. Ці фактори можуть як змінити ефективність нашого алгоритму, так і зробити неможливим його функціонування на даній платформі.

Далі розглянемо ресурси зовнішнього середовища, які необхідні моделі для коректного функціонування. Перш за все, це обчислювальна здатність

центрального процесору. Слід позначити, чим потужніший процесор тим швидше будуть виконуватися кроки алгоритму, оскільки деякі етапи залежать лише від ЦП.

Наступним чинником є доступ до ресурсів відеокарти. Деякі з алгоритмів трансформації обличчя до числових значень можна виконувати за допомогою GPU, що може значно збільшити ефективність роботи. Тут, аналогічно до процесору, ми будемо залежати від характеристик обладнання. На відміну від процесору, відеокарта на є обов'язковою, вона лише може прискорити у деяких випадках роботу алгоритму.

Останнім ресурсом системи є оперативна пам'ять. Великий об'єм ОЗУ дозволяє виконувати складні обчислення та, наприклад, використовувати більші та потужніші нейронні мережі, як складові частини нашого алгоритму.

Також важливо позначити, що дані які використовуються у алгоритмі мають знаходитися в оперативній пам'яті. Тому через обмеження оперативної пам'яті з'являються і обмеження у максимальній кількості облич, які, наприклад, класифікатор прилад може розпізнати, обмежується розмір вагів нейронних мереж. Іншим можливим варіантом є зменшення розміру вагів моделі за допомогою оптимізацій, що дозволить використовувати більш складні мережі.

За результатами проведеного моделювання виділимо фактори конфігурації системи, які впливають на можливість та якість використання моделі FaceNet на приладі. Головним чинником є оперативна пам’ять приладу. У разі недостатнього об’єму оперативної пам’яті, не буде можливості завантажити модель. Крім того, якщо пам’яті буде лише для моделі, то не буде можливості працювати з зображеннями, оскільки їх трансформації теж вимагають деяку кількість оперативної пам’яті. Слід позначити наявність файлу підкачки, який розширює оперативну пам’ять за рахунок постійної пам’яті, але у випадку його використання, швидкість значно нижча ніж у звичайної оперативної пам’яті, що вплине на час виконання розпізнавання. Однією з цілей оптимізацій має бути зменшення вимог моделі до об’єму оперативної пам’яті.

Інший фактор це швидкість виконання, яка має залежність від процесору та визначає наскільки ефективно використання обраного алгоритму у реальному часі. Якщо розпізнавання занадто довге, то с практичної точки зору це марно, тому має сенс оптимізувати час виконання одного розпізнавання. Це можна зробити шляхом зменшення кількості операцій, або їх пришвидшенням, наприклад, змінивши тип даних вагів на той, який швидше виконується на обраній апаратній конфігурації.

2.4 Аналіз методів оптимізації алгоритмів розпізнавання обличь на основі нейронних мереж

Головною проблемою використання нейронних мереж на вбудованих пристроях є високі вимоги до оперативної пам’яті та процесору. Якщо менш потужний процесор дозволить моделі працювати, але з меншою швидкістю, то при нестачі пам’яті немає можливості навіть запустити мережу. Розглянемо методи, які дозволять зменшити як використання процесору, так і зроблять модель менше за розміром.

Як відомо, розмір моделі залежить від архітектури яка лежить у її основі. Чим складніша, більша, глибша архітектура мережі, тим більш складні задачі вона може вирішувати та тим точніша вона. Разом з цим, росте розмір моделі через велику кількість вагів, які підлаштовуються під тренувальні дані.

Для використання на приладах з обмеженими ресурсами, такими як мобільні телефоні, існують спеціальні архітектури мереж, які мають значно нижчій розмір вагів, наприклад сімейство архітектур MobileNet, моделі яких можуть важити від 16 до 60 МБ. Існування таких видів архітектури не дає можливості використовувати їх для вирішення будь-якої задачі на вбудованому пристрої оскільки їх потужність обмежена і у таких складних задачах, як розпізнавання

обличчя, вони не зможуть продемонструвати задовільний результат. Причиною цього є недостатня глибина для пошуку складних залежностей.

Метод дистиляції знань дозволяє обійти ці обмеження за допомогою використання більш потужної моделі для підрахунку вагів. Основна суть полягає у тому, що існує деяка потужна модель-вчитель, яка вже має ваги, готові до використання з метою вирішення поточної задачі. Через великий розмір, ця модель не може бути застосована на IoT приладі, тому вводиться додаткова менша модель-студент, яку вже можливо використати на цільовому пристрої. Далі відбувається навчання моделі-студента за допомогою моделі-вчителя, тобто початковий датасет для задачі був оброблений потужною моделлю, яка знайшла необхідні залежності і сформувала валідні ваги, після чого менша за розміром модель на основі цих вагів сформувала свої ваги. Таким чином, отримуємо меншу модель з відповідною ефективністю. Зауважимо, не у всіх задачах такий підхід буде однаково гарно працювати і можливі втрати у точності порівняно з моделлю-вчителем.

Наступним можливим методом зменшення розміру моделі є квантування. Процес квантування моделі складається у зменшені точності чисел, які зберігають ваги. Головна ідея складається у тому, що не обов’язково ваги моделі мають відображатися 32 бітним числом, можливо зменшити цей розмір до 16 або 8 біт при збереженні працездатності моделі. У результаті операції розмір моделі зменшується, та можливе зниження точності. Передбачити наскільки впаде точність неможливо, але за звичай різниця досить не велика та не впливає на працездатність моделі. Окремим випадком квантування є бінарізація, де розмір вагів зменшується до 2 біт на значення. Це найефективніший метод оптимізації, оскільки дає значний приріст швидкості роботи разом зі зменшенням використаної пам’яті, але значно знижує точність моделі, що може зробити неможливим її подальше використання. Також існує варіант квантизації з динамічним діапазоном, який квантує ваги моделі, але усі операції залишаються у числах з плаваючою крапкою. Таким чином квантована модель має тенденцію до

стрибкових підвищень використання пам’яті через часткову конвертацію вагів до float уявлення під час обчислень, але надає мінімальні втрати у точності.

Існує три методи квантування: динамічне, статичне та тренування з врахуванням квантування.

Динамічне квантування складається у тому, що квантування відбувається під час запуску моделі і підбирається такий масштаб, щоб максимально зберегти точність. Такий вид оптимізації не завжди спрацьовує і залежить від архітектури мережі.

Статичне квантування на відміну від динамічного використовується одноразово після завершення навчання, після квантування відбувається процес калібрування на підмножині тренувального датасету. На практиці, за допомогою цього методу можливо зменшити розмір моделі у 2-3 рази та пришвидшити роботу близько в 1.5 разів, в залежності от архітектури мережі.

Під час тренування з врахуванням квантування модель одразу робить округлення вагів до необхідної точності. Всі обчислення вагів проходять у вигляді чисел з плаваючою точкою та округлюється лише результат. Цей метод надає модель з найбільшою точністю, але вимагає навчання нової моделі з нуля. Тобто неможливо таким чином оптимізувати вже існуючу, натреновану модель.

За звичай, у нейронних мережах кожний нейрон, канал та шар мають різний вплив на кінцевий результат. Можлива ситуація, коли нейрон має дуже малий вплив і фактично, мережа може працювати без нього з незмінним результатом. У таких випадках доцільно використати техніку обрізки мережі, під час якої видаляються нейрони або шари, які не приносять достатнього впливу на результат. Таким чином можна одночасно зменшити розмір моделі та прискорити її роботу, оскільки чим менше нейронів – тим менше обчислювальних операцій за одну операцію розпізнавання.

Кожний нейрон у мережі має своє значення вагів, яке він використовує під час розпізнавання. У випадку зі згортковими нейронними мережами, використовуються фільтри, які проходять по зображенню. За звичай, розмір фільтра менший за розмір зображення, тому один і той же фільтр

використовується на одному зображенні декілька разів і кожен раз створює нові ваги, для кожного зі свого положень. Для економії використаного моделлю місця можна використовувати одні ваги для кожного з можливих положень фільтра. Ця техніка називається сумісне використання вагів і дозволяє зменшити як розмір мережі, так і час її навчання.

У дослідженні має сенс використати наступні оптимізації: статичне квантування після тренування, обрізку вагів та дистиляцію знань. Динамічне квантування не підходить через необхідність постійно повторювати його, сумісне використання вагів потребує внесення змін до архітектури моделі, що може вплинити на швидкість, точність та спроможність моделі працювати взагалі.

2.5 Планування експериментів

Для порівняння ефективності використання ресурсів та точності розпізнавання оптимізованих та звичайних методів проведемо серію експериментів. Введемо наступні передумови:

- порівнюються оптимізований та звичайний варіант алгоритму між собою;
- для порівняння витрат пам’яті має використовуватись один і той же обчислювальний пристрій, оскільки через відмінності у архітектурі можливі деякі відміни у використанні пам’яті;
- для порівняння швидкості роботи є допустимим використання різних обчислювальних пристроїв, оскільки деякі оптимізації можуть привести до зменшення швидкості роботи через зміну цільової архітектури;
- для порівняння точності можна використовувати різні пристрої;
- усі заміри мають використовувати один і той же датасет та алгоритм пошуку і виділення обличчя з зображення;
- експерименти мають бути проведені з використанням алгоритму Віоли-Джонса та мережі MTCNN;

- обчислення вектору обличча має бути за виконано за допомогою нейронної мережі FaceNet;

- для витрат пам’яті та часу виконання необхідно отримати як абсолютну різницю, так і відносну.

Для експериментів було обрано тестову вибірку датасету Labeled Faces in the Wild[20], яка складається з 1000 пар зображень. На разі, цей датасет досить часто використовується з метою визначення точності моделі через те що він не великий за розміром та зображення на ньому мають реалістичне навколишнє середовище, що імітує використання алгоритму у реальних умовах. Крім того, у датасеті є вже сформовані тренувальні пари зображень які доцільно використати при перевірці точності розпізнавання.

Метою експериментів є отримання наступних даних:

- використання пам’яті, швидкість роботи та точність наступними версіями тренованої моделі FaceNet:

1) звичайною;

2) статично квантованої з динамічним діапазоном;

3) статично квантованої до float16;

4) статично квантованої до uint8;

5) з обрізаними вагами;

- використання пам’яті, швидкість роботи та точність меншої моделі, яка була навчена методом дистиляції знань з тренованої моделі FaceNet як вчителя;

- аналіз можливості використання таких моделей на вбудованих пристроях.

Розглянемо апаратну конфігурацію обчислювальних пристроїв, на яких будет проводитися дослідження. Важливо мати на увазі, що архітектура процесорів вбудованих пристроїв відрізняється від десктопних комп’ютерів. За звичай IoT прилади мають ARM або ARM64 архітектури процесорів, тому важливо використовувати у тестах прилад з подібною архітектурою.

Оскільки оптимізаційні фреймворки можуть бути налаштовані для роботи на ARM-подібних архітектурах то при виконанні оптимізованих моделей, наприклад, на процесорі з x64 архітектурою, час виконання може збільшитися. С першого

погляду це виглядає як песимізація, але ситуація може бути іншою на приладі з архітектурою, для якої оптимізували.

Комп’ютер з x64 архітектурою має наступні характеристики:

- операційна система Windows 10;

- частота процесора – 2.5 ГГц;

- 8 ядер та 16 потоків, 30 МБ кешу;

-12 ГБ оперативної пам’яті з частотою 2400 МГц;

- 8 ГБ відео пам’яті з розрядністю шини у 256 біт.

Комп’ютер з ARM архітектурою має наступні характеристики:

- операційна система MacOS 11;

- частота процесора – 3.2 ГГц;

- 8 ядер та 8 потоків, 320 КБ кешу першого рівня на кожне ядро, та 12 МБ кешу другого рівня для усіх ядер;

-16 ГБ оперативної пам’яті;

- оперативна пам’ять використовується як відео пам’ять, теоретичний максимум 16 ГБ, але через використання RAM іншими процесами та системою, точне поточне значення передбачити неможливо.

Слід наголосити, що поточна конфігурація ARM приладу значно потужніша ніж переважна більшість вбудованих приладів, тому має сенс дивитись не на абсолютні величини часу виконання, а відносні. Також це дозволить дослідити працездатність та ефективність складних та великих моделей на ARM архітектурі.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Вибір мови програмування для роботи з нейронними мережами

Визначимо мову програмування за допомогою якої будуть розроблятися експериментальні прототипи.

Наразі досить популярним виборами для роботи з нейронними мережами є такі мови як Python, Java, C++. Розглянемо інструменти які надають ці мови та їх переваги і недоліки.

Код на C++ виконується досить швидко і найважливішим є можливість контролювати усі значимі операції у коді самостійно. Відомий фреймворк для машинного навчання від Google – Tensorflow написаний на саме на C++ і має відповідний API. На разі, за допомогою C++ та Tensorflow створюються State of the art нейроні мережі. Головним недоліком є залежність коду від платформи та необхідність перекомпілювати усе, включаючи додаткові бібліотеки на кожну платформу окремо. Крім того, такий ступінь контролю виконання вимагає досить значних об’ємів коду, що ускладнює дослідження.

Java – також досить популярний мова для дата саєнсу, але спектр інструментів який вона надає для глибокого навчання досить вузький. На відміну від C++, код на Java працює на будь-якій платформі де є JWM, але ціною цього є нижча швидкість виконання.

Останнім варіантом є Python. Для цієї мови існує досить значна кількість фреймворків та бібліотек для машинного навчання, включаючи API для Tensorflow. Головною перевагою є необхідність у досить малої кількості коду та велике число інструментів для роботи з даними, що робить цю мову потужним засобом для розробки прототипів та досліджень. Існує багато наукових робот і досліджень, де прототипи розроблялись на Python з використанням C++ фреймворків. Головним недоліком є низька швидкість виконання, що не дає можливості створювати State of the art моделі. Це можна значною мірою нівелювати використанням C++ бібліотеками, які надають Python інтерфейс.

Беручі до уваги мету дослідження, оптимальним вибором буде мова Python, оскільки вона надає широкий вибір фреймворків і значний простір для прототипування через відсутність необхідності писати код не пов’язаний з дослідженням. Також цей код можна буде досить просто портувати на C++ у випадку якщо використовувалась Python обгортка C++ фреймворку.

3.2 Вибір фреймворку для прототипування

Python надає три великі фреймворки для роботи з нейроними мережами: Keras[21], PyTorch[22], Tensorflow[23]. Проаналізуємо доцільність використання кожного з них.

Tensorflow надає як низькорівневе, так і високорівне API для створення моделей. Важливими перевагами цього фреймворку є висока швидкість роботи, можливість максимальної кастомізації кожного кроку роботи мережі, потужна система для побудови потоків вхідних даних до моделей, яка дозволяє працювати з гігантськими датасетами, та широкий спектр вбудованих датасетів. Єдиний недолік – досить висока складність та довгий час розробки моделей через велику кількість опцій які потрібно враховувати та додавати на кожному кроці реалізації, що не є підходящім для експериментального дослідження.

Навколо Tensorflow існує фреймворк-обгортка, який надає більш високорівневий та простий інтерфейс – Keras. Ця бібліотека використовує у середині низькорівневі компоненти Tensorflow, що робить моделі досить ефективними. Крім цього, Keras є вдалими вибором для розробки прототипів, оскільки за його допомогою можна досить швидко будувати моделі. Цей інструмент повністю розроблений за допомогою Python, звідки випливає його основний недолік – низька швидкість роботи. Компоненти пов’язані з імпортом ресурсів не таки потужні як у Tensorflow, що не дає можливості ефективно працювати з великими за розміром датасетами, обмежуючись малими або

середніми. У рамках поточного дослідження це обмеження не є критичним, оскільки датасет LFW обраний для експериментів не великий за розміром і його можна досить ефективно використовувати з Keras.

Остання можлива опція – PyTorch. Цей фреймворк дуже схожий на Tensorflow, але на відміну від нього не надає високорівневого інтерфейсу. Тому він ще менше підходить для прототипування.

Оптимальним фреймворком для проведення експериментів є Keras, оскільки він поєднує достатній рівень ефективності, спектр функціоналу та швидкість і простоту розробки і прототипування.

3.3 Огляд додаткових бібліотек

Оскільки у дослідженні будуть використовуватись згорткові нейронні мережі для розпізнавання облич на фото, необхідні інструменти для роботи з зображеннями. Оптимальними варіантами є модулі OpenCV та PIL.

OpenCV містить у собі багато алгоритмів для роботи з зображеннями, у дослідженні нас цікавить реалізація алгоритму Віоли-Джонса та методи масштабування зображення з мінімальними спотвореннями.

PIL доцільно використовувати для завантаження зображень з файлів оскільки ця бібліотека надає інтеграцію з numpy що зменшує кількість трансформацій даних для передачі у нейрону мережу.

Математична бібліотека numpy необхідна для роботи з багатовимірними даними. Оскільки згорткові нейроні моделі за звичай сприймають дані у вигляді 4 вимірному масиву: перший вимір – зображення, другий висота, третій ширина та четвертий містить канали. Цей фреймворк надає спектр статистичних алгоритмів для використання на масивах що значно спрощує нормалізації вхідних та вихідних даних моделі.

Бібліотека MTCNN надає реалізацію відповідної нейронної мережі для пошуку та вилучення облич на фото.

TensorFlow Lite – бібліотека яка є частиною фреймвоку TensorFlow, вона надає компоненти та моделі нейронних мереж для роботи на мобільних приладах. Головною відміною від самого фреймворку TensorFlow є велика кількість оптимізацій для роботи на ARM архітектурах. Дана бібліотека має опції для оптимізації тренованих моделей, такі як квантизація, підрізання моделей та трансформація для використання TensorFlow Lite як бекенду, що дозволяє підвищити

3.4 Огляд архітектур нейронних мереж та моделей використаних у дослідженні

Для дослідження необхідно обрати дві моделі, які як з теоретичної точки зору, так і з практичною підходять для задачі розпізнавання облич. Одна з моделей має бути побудованою на основі глибокої архітектури, без будь яких обмежень по розміру та оптимізацій для вбудованих приладів. Важливим є висока точність моделі в розпізнаванні моделей, максимальна наближеність до точності найкращих існуючих рішень[24], оскільки ця модель буде використана як вчитель у дослідженні оптимізації за допомогою дистиляції знань. Крім цього, цікаво дослідити як оптимізації можуть змінити вимоги до ресурсів, час роботи та точність моделі, яка може успішно використовуватись на практиці.

На разі, таким рішенням є модель FaceNet описана у дослідженні використання нейронних мереж для розпізнавання облич[25]. Згідно результатів дослідження, підходящою архітектурою мережи є Inception-ResNet. Це сімейство архітектур спеціалізоване на класифікації зображень. Різниця між версіями складається головним чином у точності і використанні ресурсів. Має сенс зупинитися на першій версії, оскільки саме з нею FaceNet демонструє

максимальну точність. Inception-ResNet v1 складається з 27 шарів. Нижче наведено архітектуру мережі (див. рис. 3.1):

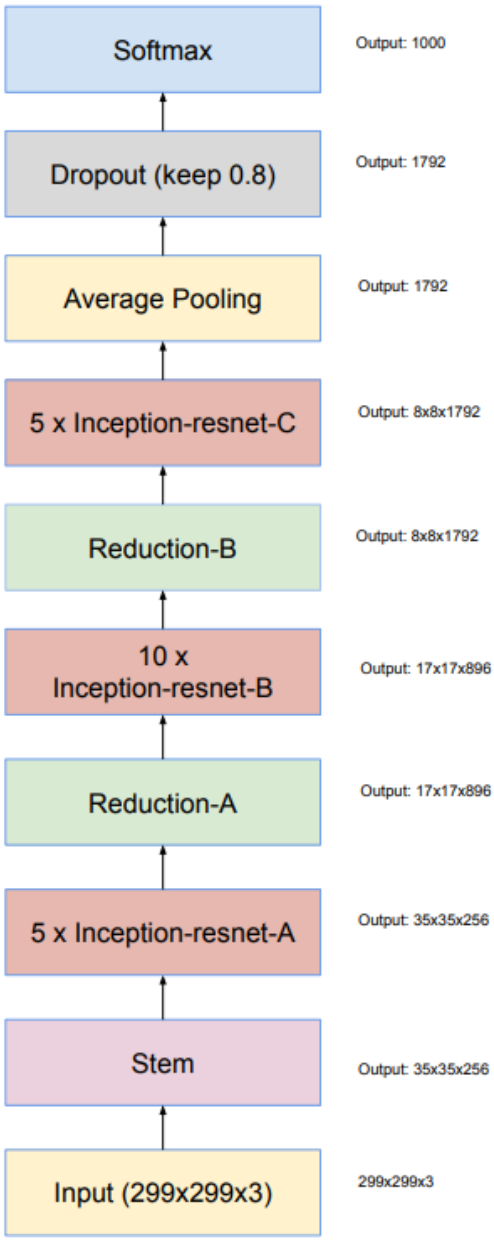

Рисунок 3.1 – Архітектура Inception-ResNet v1

Головною особливістю є її Inception-resnet шари які складаються з декількох згорткових шарів різних розмірів які працюють разом у одному шарі. Головною ідеєю цих шарів є те, що якщо спочатку обробити вхід шаром з меншим розміром, а потім передати далі, можна досягти більшої ефективності обчислень через

зниження розмірності даних. Завелике зменшення розмірності може привести до деградації точності мережі через втрату вхідних даних.

Inception-resnet шари між собою відрізняються розмірами згорткових шарі в середені.

Крім архітектури мережі важливим чинником є метод тренування. У задачі розпізнавання облич це є ключовим фактором і йому надається багато уваги. Модель FaceNet використовує спеціальну функцію підрахунку втрат – Triplet loss. Основною ідеєю є зробити так, щоб модель мінімізувала дистанцію між векторами облич однієї людини, та максимізувала якщо це різні люди. Для цього під час навчання передається одразу три зображення – якірне, позитивне та негативне. З цих зображень формується дві пари, у кожній парі знаходиться якірне та позитивне або негативне зображення. Далі підраховуються вектора облич для кожного зображення і знаходяться відстані між якірнім і позитивним, та якірним і негативним зображенням. На основі відстаней підраховується втрата. Схематично це зображено на рисунку 3.2.

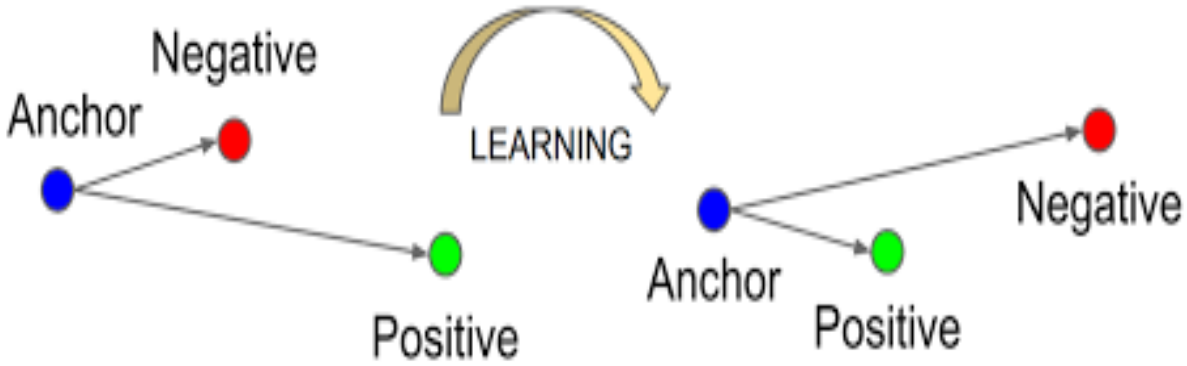

Рисунок 3.2 – Принцип роботи Triplet loss

Існує інша робота [26] по дослідженню розпізнавання облич, у якій запропоновано альтернативну функцію підрахунку втрат – Cluster Loss. Цей підхід взяв натхнення з алгоритму кластеризації K-Means, а саме намагатися створювати для кожного окремого обличча кластер максимально віддалений від інших облич.

На відміну від Triplet loss, тренування за допомогою функції Center Loss не потребує спеціальної підготовки даних, такого як формування триплетів зображень, але вимагає використання батчів тренувальних зображень розміром більше за одне зображення на батч. У кожному батчі формуються кластери з векторі облич кожної унікальної люди, знаходиться відстань між ними та знаходиться значення втрати як середнє значеннях квадратів відстаней.

Візуально результат роботи мережі тренованої за допомогою Center Loss виглядає як на рисунку 3.3:

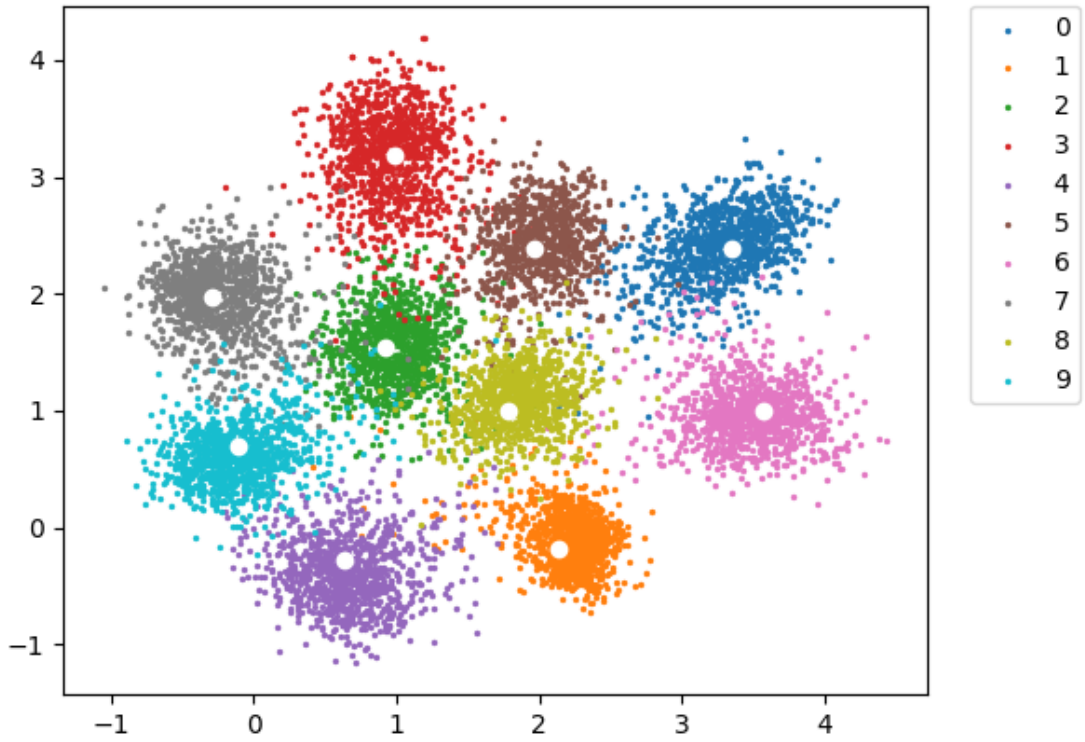

Рисунок 3.3 – Результат тренування з Cluster Loss

Через досить велику глибину моделі та обмеження фреймворку Keras для роботи з великими датасетами, має сенс використати треновано модель FaceNet навчену за одним з вказаних вище алгоритмів.

Далі розглянемо модель, в яку будемо дистилювати знання з моделі-вчителя, тобто модель-студента. На відміну від першої моделі, ми маємо ряд

обмежень стосовно доступних ресурсів, оскільки ідеєю дистиляції знань є навчання меншої моделі, яка може виконуватися на цільовому приладі. Гарним вибором буде сімейство архітектур MobileNet. Ці моделі розроблені для роботи на мобільних приладах, якими можна вважати і вбудовані пристрої. Існує декілька версій моделі, ми зупинимося на другій, MobileNet v2, оскільки перша версія має трохи більший розмір та меншу точність. З архітектурної точки зору MobileNet v1 простіша за другу версію і має меншу кількість шарів. Розмір моделі MobileNet v2 складає приблизно 14 МБ. На рисунку 3.4 наведено архітектуру мережі MobileNet v2.

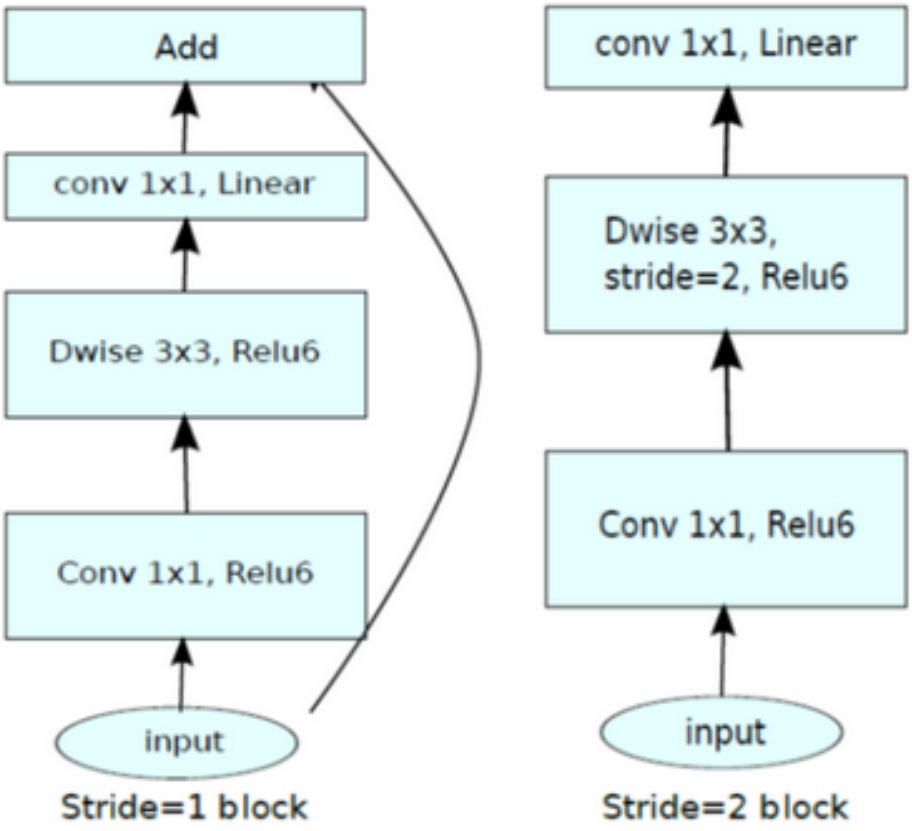

Рисунок 3.4 – Архітектура MobileNet v2

Ключовою особливістю цієї мережі є наявність блоків Depthwise Convolution, які дозволяють зробити модель більш компактною. Відмінність роботи цього блока від звичайних згорткових шарів у тому, Depthwise Convolution блок обробляє кожен канал зображення фільтром окремо, а після обробки збирає

результат роботи фільтрів у один вихід. У той час, як звичайний згортковий шар обробляє усі канали зображення одночасно. З точки зору роботи з даними маємо наступну відміну – звичайні згорткові шари беруть до уваги не лише інформацію про канал, а і деяку міжканальну інформацію, що створює додаткові ваги. У Depthwise Convolution блоці робиться припущення, що якщо обробити інформацію про канали та міжканальну інформацію окремо, послідовно, втрати даних або не буде, або вона буде незначною.

Для використання дистиляції знань необхідно обрати функцію підрахунку втрат для моделі-студента. Слід використовувати таку саму функцію, яка використовувалась для навчання моделі-вчителя. У нашому випадку це або Triplet Loss, або Center Loss.

3.5 Реалізація оптимізованих версій моделі FaceNet

Розглянемо деталі програмної реалізації оптимізованих алгоритмів. Процес оптимізації починається з завантаження тренованої моделі, існує два способи зробити це.

Перший варіант це одразу завантажити збережену модель з файлу спеціального формату. Нажаль, цей метод не досить надійний у контексті Keras, оскільки зі змінами у бібліотеці з версіями збережена модель може стати не сумісної і доведеться підбирати старшу версію моделі. Крім того, можлива проблема з завантаженням моделі на різній платформі.

Альтернативним методом є завантаження лише вагів, а не усієї моделі. Файл вагів має стандартизований вигляд, тому проблем з платформами та версіями не виникає, оскільки у ньому не зберігається інформації щодо шарів. Але для завантаження вагів необхідно до цього створити модель, а це потребує наявності реалізації архітектури моделі.

Розглянемо фрагмент коду, який створює модель та завантажує у неї відповідні ваги:

```
facenet_model = interception_resnetv1.InceptionResNetV1()
facenet_model.load_weights(filepath='models/facenet_keras_weights.h5'
)
```

Після того, як завантажили ваги моделі, тим самим повністю її ініціалізував, необхідно зберегти її як модель, щоб оптимізатор міг її завантажити. У конкретно цьому випадку можемо не зважати на недоліки збереження і завантаження саме моделі оскільки у подальшому для роботи з моделлю буде використовуватися одна і таж платформа. Наведемо код збереження моделі:

```
facenet_model.save("models/facenet_keras_quantization.h5")
```

Після отримання файлу з моделлю, наступним шляхом буде її квантизація. Для цього використаємо бібліотеку TensorFlow Lite, і разом з цим, збережемо модель у форматі TensorFlow Lite. Виконання моделі за допомогою TensorFlow Lite більш ефективне на вбудованих приладах оскільки компоненти моделі оптимізовані під мобільні прилади. Маємо наступний метод для квантизації:

```
def post_training_quantization_dynamic(model_path, result_path):
    converter=tf.lite.TFLiteConverter.from_keras_model_file
(model_path)
    converter.post_training_quantize = True
    tflite_model = converter.convert()
    open(result_path, "wb").write(tflite_model)
```

У наведеному фрагменті коду створюється конвертор, який конвертує нашу модель до вигляду TensorFlow Lite з опціональними оптимізаціями. У поточному випадку, включаємо квантування після тренування, що значить що модель буде квантована під час конвертації. За замовчуванням модель квантується у динамічному діапазоні значень, тобто ваги шарів до float16, а ваги функцій активацій залишаються без змін.

Далі розглянемо код для повної квантизації до float16:

```
def post_training_quantization_float16(model_path, result_path):
   converter=tf.lite.TFLiteConverter.from_keras_model_file
(model_path)
    converter.post_training_quantize = True
    converter.target_spec.supported_types = [tf.float16]
    tflite_model = converter.convert()
    open(result_path, "wb").write(tflite_model)
```

Головною відміною є наявність цільового типу даних, до якого має бути квантизація, у поточному фрагменті коду це float16.

Наприклад, при квантизації до int8, матимемо наступний цільовий тип:

```
converter.target_spec.supported_types = [tf.int8]
```

Наступна оптимізація, код якої має сенс розглянути – дистиляція знань. Для виконання цієї оптимізації створимо клас, успадкований від keras.Model та реалізуємо власні функції для навчання. Нижче наведено код конструктора класу для дистиляції знань:

```
class Distiller(keras.Model):
    def __init__(self, student, teacher, class_num):
        super(Distiller, self).__init__()
        self.teacher = teacher
        self.student = student
        self.prelogits_output=keras.Model(student.input,
student.output)
        logits_layer = keras.layers.Dense(class_num,
kernel_regularizer=tf.keras.regularizers.l2(l2=5e-4))(student.output)
        self.logits_model = keras.Model(student.input, logits_layer)
```

Для ініціалізації потрібно дві моделі - вчитель та учень. Це особливий випадок реалізації, оскільки будемо працювати з моделлю, яка повертає вектор особливостей і для підрахунку втрат необхідно використовувати додатковий шар з ненормалізованим результатом. Крім того, щоб модель-учень повертала не результат класифікації, а набір особливостей, необхідно ініціалізувати її наступним чином:

```
test_model=keras.applications.MobileNetV2(include_top=False,
input_shape=(160, 160) + (3,), pooling="avg")
```

Також необхідно привести вихідні дані для такого ж формату як і має модель вчитель, тому додамо додаткові шари до виходу моделі за допомогою наступного методу:

```
def add_embeddings_to_model(model):
    x = layers.Dropout(0.2, name='Dropout')(model.output)
    x = layers.Dense(128, use_bias=False, name='Bottleneck')(x)
    x   =   layers.BatchNormalization(momentum=0.995,   epsilon=0.001,
scale=False,
                                  name="BatchNormLastBottleneck")(x)
    return Model(model.input, x, name="Embedding_mobinet")
```

У результаті виклика цього методу, повертається модель яка повертає вектор особливостей розміром 128 значень.

Далі розглянемо найважливіший крок дистиляції знань – функцію навчання моделі. Стандартна реалізація не підходить, оскільки не враховує передбачення вчителя під час навчання. Тому наведемо окрему реалізацію:

```
        def train_step(self, data):
            x, y = data
            teacher_predictions = self.teacher(x, training=False)
            with tf.GradientTape() as tape:
                student_predictions = self.student(x, training=True)
                student_logits = self.logits_model(x, training=True)
                student_loss_entropy = self.student_cross_entropy_loss_fn
(logits=student_logits, labels=y)
                cross_entropy_mean = tf.reduce_mean(student_loss_entropy)
                student_loss_center=self.center_loss_fn(y,
student_predictions)*0.5
                student_loss=tf.add_n
([cross_entropy_mean]+student_loss_center)
                distillation_loss = self.distillation_loss_fn(
                    tf.nn.softmax(teacher_predictions / self.temperature,
axis=1),
                    tf.nn.softmax(student_predictions / self.temperature,
axis=1))
                loss= self.alpha * student_loss + (1 - self.alpha) *
distillation_loss
            trainable_vars = self.student.trainable_variables
            gradients = tape.gradient(loss, trainable_vars)
            self.optimizer.apply_gradients(zip(gradients,
trainable_vars))
            self.compiled_metrics.update_state(y, student_predictions)
            results = {m.name: m.result() for m in self.metrics}
            results.update(
                {"student_loss": student_loss, "distillation_loss":
distillation_loss}
            )
            return results
```

Важливою частиною дистиляції знань є підрахунок втрат, базуючись на передбаченні вчителя. Крім того, для підрахунку втрат моделі яка навчається використовується одразу дві функції і вже їх сумісний результат є кінцевим значенням. Необхідність цього обумовлюється специфікою навчання моделі для розпізнавання облич, яку розглядали вище. Крім того, такий же шлях використовувався для навчання моделі-вчителя. Далі отримане значення втрати учня об’єднуємо з передбаченням вчителя, це і є кінцевим значень втрати, яке і корегує ваги.

Частина членів класу ініціалізується під час виклику методу компіляції моделі, який має деякі спеціальні для нашої реалізації параметри. Розглядати реалізації немає сенсу, оскільки вона виконує функції дещо схожі за конструктором класу.

4 ОПИС ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

4.1 Опис алгоритму ходу експериментів

Розглянемо загальні кроки експерименту на рисунку 4.1.

Рисунок 4.1 – Блок-схема кроків експерименту

Спочатку необхідно завантажити модель мережі над якій будемо проводити експеримент у пам’ять, оскільки у випадку помилки буде можливість одразу визначити її і не втрачати час на завантаження або відкриття датасету.

Наступним кроком є завантаження датасету у пам’ять. З одного боку, таким чином збільшується використання оперативної пам’яті, а з іншого під час обробки зображень немає необхідності кожен раз завантажувати файли.

Коли завантажили датасет, вибирається пара зображень для розпізнавання. Пари зображень разом з правильним висновком щодо подібності сформовані заздалегідь та надаються разом з датасетом.

Для вилучення обличчя з зображення є два алгоритми тому експеримент буде проводитися з кожними алгоритмом окремо. Тобто для кожної моделі проводиться два експерименти.

Далі починається відлік часу та виконання обробки зображення за допомогою нейронної мережі. Після виконання операції підрахунку вектору обличчя, записується час, який це зайняло та робиться передбачення стосовно чи одна це людина базуючись на Евклідовій[27] відстані між векторами. Евклідова відстань досить просто та ефективно обчислюється, тому буде доцільно обрати саме її.

Після того як отримані передбачення для всіх елементів можна підрахувати метрики поточної моделі: середній час обчислення, мінімальний час обчислення, максимальний час обчислення, розмір моделі у пам’яті та точність. Відносні значення будуть підраховані окремо. Крім того, існують статична метрика - розмір моделі на диску, яку можна подивитися окремо у файловій системі.

Не залежно від виду оптимізації моделі, яка досліджується, кроки алгоритму не будуть змінюватись оскільки деталі вигляду яка модель зараз активна ніяк не впливають на хід експерименту.

Оптимізації моделі будуть виконуватися окремо від експериментів, таким чином перед експериментальним дослідженням маємо набір оптимізованих нейронних мереж.

4.2 Експериментальне дослідження моделі FaceNet без оптимізацій

Для отримання результатів стосовно ефективності використаних оптимізацій необхідно мати інформацію щодо базової версії моделі. Проведемо серію експериментів на базовій версії моделі (див. табл. 4.1).

Таблиця 4.1 – Метрики FaceNet без оптимізацій на ARM процесорі

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	96.1%	196	30.1	29.8	792.3	92.5
Метод Віоли-Джонса	92.5%	196	31.9	29.8	809.6	92.5

Аналогічні експерименти на x64 архітектурі наведено у таблиці 4.2.

Таблиця 4.2 – Метрики FaceNet без оптимізацій на x64 процесорі

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	96.1%	196	91.5	83.2	1964.4	92.5
Метод Віоли-Джонса	92.5%	196	91.3	84.1	1970	92.5

За результатами видно, що при використанні MTCNN маємо більш високу точність у порівнянні з методом Віоли-Джонса.

Важливо позначити, під час дослідження час виконання експерименту з використанням MTCNN помітно вищій, тому зробимо заміри часу, який використовує кожний алгоритм для вилучення обличчя з зображення. Ця інформація дасть нам можливість краще розуміти скільки усього займає часу процедура розпізнавання та яка частка з нього йде на пошук обличчя, а яка на безпосередньо розпізнавання. Результати дослідження у таблиці 4.3.

Таблиця 4.3 – Порівняння часу роботи алгоритмів вилучення обличчя на ARM архітектурі

	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс
MTCNN	259.26	149.7	520.1
Метод Віоли-Джонса	3.8	2.5	45.5

З таблиці видно, що метод Віоли-Джонса приблизно у 68 разів швидше за поточну реалізацію MTCNN. Але потрібно мати на увазі, що оба алгоритми працюють на CPU у поточному виконанні та середовищі .

4.3 Експериментальне дослідження моделі FaceNet оптимізованої за допомогою статичного квантування

Для оптимізації моделі будемо виконувати статичне квантування на вже тренованій моделі. Усього буде розглядатися три квантовані моделі, які

відрізняються точністю: динамічний діапазон, до float16, до int8. Важливо позначити, що квантування відбувається за допомогою бібліотеки TensorFlow Lite, та модель буде трансформована для більш ефективного використання на мобільних архітектурах. Таким чином, можливе погіршення часу виконання на архітектурі x64.

Маємо наступні результати вимірювань для динамічного діапазону квантування(див. табл. 4.4):

Таблиця 4.4 – Метрики FaceNet з динамічним діапазоном квантування на ARM процесорі

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
dMTCNN	96.0%	68	27.1	26.9	49.7	23.7
Метод Віоли-Джонса	92.7%	68	27.6	27	37.4	23.7

Аналогічні експерименти на x64 архітектурі наведено у таблиці 4.5.

Можемо побачити, що розмір моделі зменшився приблизно у чотири рази, оскільки самі ваги квантовані до int8, у той час як ваги функцій активації залишились без змін.

Стосовно середнього часу розпізнавання, він зменшився у на 13.5-14% на ARM архітектурі. У той час як на x64 архітектурі можемо спостерігати значну песимізацію. Це можна пояснити тим що моделі, які виконуються за допомогою TensorFlow Lite мають оптимізації для обчислень на ARM процесорах та мають значно нижчу на серверних рішеннях. Іншим можливим випадком є відсутність

підтримки процесором деяких команд. З цього можна зробити висновок, що подальші заміри моделей, які оптимізовані та виконуються за допомого TensorFlow Lite не має сенсу робити на поточній конфігурації машини за x64 архітектурою.

Таблиця 4.5 – Метрики FaceNet з динамічним діапазоном квантування на x64 процесорі

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	96.0%	68	13156	12815	13600	23.7
Метод Віоли-Джонса	92.7%	68	13149	12823.5	13590.65	23.7

Зміни у точності лежать у межах 0.1-0.2%, що є досить незначним.

Проведемо експерименти з моделю квантованою до float16(див. табл. 4.6).

Таблиця 4.6 – Метрики FaceNet з квантуванням до float16 на ARM процесорі

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	96.1%	98	25.1	21.3	150.7	45.7
Метод Віоли-Джонса	92.0%	98	24.8	22.8	78	45.7

Оскільки початково модель зберігала свої ваги у float32, то при квантуванні до float16 маємо зменшення рівно у два рази. На відміну від попереднього експерименту, квантовані були усі ваги.

Середній час розпізнавання зменшився на 24-25% що є досить гарним результатом.

Точність у випадку MTCNN не змінилась, а при використанні метода Віоли-Джонса стала меншою на 0.5%.

Розглянемо модель, ваги якої квантовані до діапазону значень типу int8. Маємо наступні результати (див. табл. 4.7).

Таблиця 4.7 – Метрики FaceNet з квантуванням до int8 на ARM процесорі

	Точність	Розмір у пам’яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	96.0%	49	27.6	26.8	38.2	23.7
Метод Віоли-Джонса	92.3%	49	27.6	26.9	32.41	23.7

У випадку с квантуванням до int8 бачимо зниження розміру у пам’яті рівно у чотири рази.

Прискорення аналогічне до моделі яка була квантована з динамічним діапазоном.

Головна відмінність складається у погіршені точності при використанні алгоритму Віоли-Джонса на 0.5%. Таку відміну можна пояснити тим, що через квантування вагів функцій активації модель стала менш чутливою що й привело до таких відмін.

З усіх досліджених варіантів квантизації найбільше прискорення, на 24-25%, надає квантизація до точності float16. Це можна пояснити тим, що виконання операцій з таким типом даних найбільш ефективний з запропонованих. Також таке квантування призводить до мінімальних втрат точності. У той час як квантування до int8 максимально зменшує вимоги до пам’яті, а саме у 4 рази. Але разом з цим погіршує точність у діапазоні 0.2-0.5%. Квантизація з дінамічним діапазоном зменшує вагу моделі менше ніж квантизація до int8, але і втрати точності менші – 0.1-0.2%. Швидкість виконання цих моделей аналогічна.

Також було експериментально отримано, що моделі оптимізовані та конвертовані до формату TensorFlow Lite виконуються повільніше на поточній тестовій конфігурації з процесором архітектурі x64.

4.4 Експериментальне дослідження моделі FaceNet оптимізованої за допомогою обрізання вагів

Для обрізання вагів необхідно обрати дані, на яких модель буде донавчатись та перевірятись з метою видалення вагів які мають найменший вплив на кінцевий результат. Крім того, необхідна функція підрахунку втрат. Такою функцією може виступати Center Loss, оскільки оптимізуєма модель FaceNet навчалась саме за нею.

Бібліотека TensorFlow Lite надає інструментарій для обрізання вагів мережі, тому буде доцільно разом з оптимізацією трансформувати модель до цього вигляду. Таким чином, буде доцільно проводити експеримент лише на обчислювальному приладі з ARM архітектурою.

Результати експериментів наведені у таблиці 4.7

Таблиця 4.8 – Метрики FaceNet після обрізання вагів на ARM процесорі

	Точність	Розмір у пам’яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	66.1%	55	22.21	19.3	86	25.7
Метод Віоли-Джонса	62.0%	55	23.8	20.8	82.5	25.7

З результатів можна побачити, що точність моделі значно погіршилась у результаті оптимізації. Важливим моментом є те, що 50% тестів є негативним. У випадку якщо модель зовсім не працює, вона може мати точність 50% або близькі к ньому значення. Тобто можна зробити висновок, що точність моделі не задовільна.

Основний фактор, який міг вплинути на точність це видалення забагато вагів. Причиною такої поведінки може бути недостатня кількість тренувальних даних, які використовувались під час обрізання вагів, не підходящий оптимізатор або функція підрахунку втрат. Питання функції підрахунку втрат особливо актуальне, оскільки процес тренування є ключовим для коректної роботи FaceNet.

4.5 Експериментальне дослідження моделі MobileNet v2 навченної за допомогою дистиляції знань з моделі FaceNet

Спочатку перевіримо, як демонструє себе модель MobileNet v2 тренована на датасеті ImageNet у задачі розпізнавання облич. Оскільки модель без вагів буде потребувати значно більше даних для тренування. У поточному випадку, буде

проводитися процес донавчання та корегування вагів моделі, що теж можна вважати дистиляцією знань.

Метрики наведені у таблиці 4.8

Таблиця 4.9 – Метрики MobileNet v2 до дистиляції знань

	Точність	Розмір у пам'яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	65.4%	52.2	21.5	19.2	332	16
Метод Віоли-Джонса	64.5%	52.2	20.7	19.1	387	16

Зараз мережа надає досить низьку точність, як і у попередньому випадку. За часом обчислень теж у цієї моделі є перевага через її простоту. Після дистиляції знань, ці характеристики мають лишитися на приблизном тому ж рівні. Можливі зміни у розмірі через корекцію вагів.

Для дистиляції знань будемо використовувати дві функції підрахунку втрат – Softmax, Center Loss, причому результат роботи Center Loss буде помножатись на 0.5, щоб знизити вплив на ваги від цієї функції. Такий вибір обумовлений тим, що модель-вчитель тренувалась з використанням такого самого набору та коефіцієнтів.

Для донавчання з моделлю-вчителем будемо використовувати 5400 фото з датасету LFW, у батчах по 70 зображень на протязі 40 епох.

Стосовно параметрів дистиляції знань, будемо використовувати альфа – коефіцієнт того, наскільки сильно передбачення вчителя впливають на отриманий спільні ваги, як 3.

Функцією, яка підраховує втрату на основі втрат моделі-студента та моделі-вчителя, буде Розходження Кульбака — Лейблера.

У таблиці 4.9 наведено заміри після дистиляції знань.

Таблиця 4.10 – Метрики MobileNet v2 після дистиляції знань

	Точність	Розмір у пам’яті, МБ	Середній час обчислення, мс	Мінімальний час обчислення, мс	Максимальний час обчислення, мс	Розмір на диску, МБ
MTCNN	77.3%	54.2	21.1	19.3	360	16
Метод Віоли-Джонса	73.5%	54.2	20.8	19.5	358.7	16

Перш за все, значно піднялася точність моделі у порівнянні з початковим станом, а саме на 9-12%. Цікавим спостереженням є не пропорційне підвищення точності при використанні MTCNN. Можливою причиною є те, що під час донавчання використовувався саме цей алгоритм, та модель-вчитель тренувалась на обличчях, оброблених цим алгоритмом.

Отримана модель працює на 40-45% швидше за модель вчитель та використовує у 3.5-3.6 разів менше оперативної пам’яті.

Існують опції для покращення результатів. Перш за все, використання більшого за розміром датасету у сукупності з іншим набором параметрів тренування, таких як швидкість навчання. У поточному експерименті використовувались значення за замовчуванням. Також інший набор функцій підрахунку втрат може привести до підвищення точності, або розробка якоїсь нової функції підрахунку втрат спеціально для цієї задачі.

4.6 Аналіз результатів та рекомендації щодо досліджених оптимізацій

Розглянемо оптимізацію, яка призвела до найгірших результатів – обрізання вагів мережі. Після використання цієї оптимізації модель втратила можливість працювати правильно хоча і мала значні покращення у використанні ресурсів. FaceNet досить складна модель через те, що результатом її роботи є не клас, а вектор особливостей, що значно ускладнює як процес тренування і валідації, так і вибір функції підрахунку втрат.

Для більш простої моделі, наприклад, класифікатору, значно простіше підібрати необхідні параметри, тому метод обрізання вагів має сенс використовувати з такою моделлю яка: видає значення класу, до якого належить об’єкт та не має специфічних умов тренування подібних до Triplet Loss, оскільки у такому випадку стандартні інструменти оптимізації будуть не ефективними. Також використання більш потужних фреймворків машинного навчання, з більшим спектром контролю над усіма операціями, таким як TensorFlow, може покращити результати.

Середній результат продемонстрував метод дистиляції знань. З одного боку, ми отримали значне зниження необхідної оперативної пам’яті для роботи мережі та суттєве прискорення. Але зниження різниця у точності між оригінальною моделлю та моделлю-студентом досить значна. Аналогічно до попереднього випадку, основні складнощі виникають через нетривіальний процес тренування оригінальної моделі, що значно ускладнює донавчання моделі-студента.

Статична квантизація після тренування моделі надала дуже гарний результат, причому у кожному з можливих варіантів. Головною перевагою статичної квантизаці є відсутність необхідності дотреновувати модель або тренувати її з початку. Це значить, що є можливість оптимізувати таким чином будь яку модель існуючу, не маючи відповідного датасету для її навчання. Також це позбавляє від необхідності підбору функції підрахунку втрат та параметрів

тренування, як у випадку з дистиляцією знань або обрізкою мереж. З практичної точки зору, квантування самий простий метод оптимізації.

З перспективи точності, втрати незначні та не можуть сильно вплинути на кінцевий результат. Але слід мати на увазі, що такий результат варіюється від моделі до моделі, тому не має сенсу виконувати квантизацію кожної моделі без попереднього дослідження впливу на точність.

Квантована модель працює швидше за звичайний варіант через те, що через зменшення розміру змінних, які зберігають ваги, з'являється доступ більш продуктивних операцій над масивами даних. Наприклад, якщо ми маємо floa32, та регістр розміром 64 біта, ми можемо використати тільки два таких числа при обчисленні, а чисел типу float16 – чотири. Що дозволить зменшити необхідний час для обчислення всіх чисел. Таким чином, логічно передбачити, що модель квантована до int8 має бути швидша ніж та що до float16. Експериментальне дослідження продемонструвало, що це не так. Найбільш вірогідною причиною є краща апаратна підтримка обчислень з float16, ніж з int8. Модель, квантована до float16 виявилась самою швидкою.

Стосовно зменшення використовуємої оперативної пам'яті після квантування знайшлась чітка залежність – модель зменшується рівно у стількі разів, у скільки поточний тип даних більше за майбутній. Це правило не зовсім коректне по відношенню до моделей, квантованих з динамічним діапазоном, оскільки ваги функцій активацій залишаються такими саме, що приводить до збільшення розміру квантованої моделі. З точки зору точності, квантування з динамічним діапазоном продемонструвало кращу точність ніж просте квантування до int8, при трохи більшому використанні пам'яті.

Таким чином, для оптимального використання статичного квантування, необхідно квантувати модель до всіх можливих точностей, перевірити їх характеристики та обрати оптимальну точність для поточної задачі.

Також алгоритм вилучення обличчя може вплинути на кінцеву точність, згідно з дослідження, при використанні MTCNN ми отримуємо більшу точність, але швидкість виконання значно нижча ніж у алгоритму Віоли-Джонса.

ВИСНОВКИ

У результаті кваліфікаційної роботи магістра було проведено аналіз існуючих методів розпізнавання обличь та обрано найбільш ефективний для використання у подальших експериментах – нейрона мережа FaceNet. Також проаналізовано методи оптимізації нейронних мереж, відібрані ті, які можна застосувати до обраної моделі.

Створена теоретична база стосовно методів розпізнавання обличь, методів оптимізації нейронних мереж.

Спочатку було проведено аналіз предметної області практичної задачі і можливі варіанти її вирішення. Далі ми провели аналог існуючих сервісів, які надають функціонал розпізнавання облич, виділили їх особливості, переваги та недоліки і порівняли з нашим підходом. В результаті аналізу ми провели постановку завдання.

На основі аналізу сформовані вимоги до інтерфейсів майбутнього модулю розпізнавання облич.

Для проведення експериментів була обрана мова програмування Python, як фреймворк машинного навчання використовувався Keras. Деякі оптимізації проводилися за допомогою бібліотеки TensorFlow Lite.

У ході виконання кваліфікаційної роботи магістра було зроблене наступне:

- проведено аналіз існуючих сервісів та алгоритмів розпізнавання облич; обрано найкращій варіант – FaceNet, для подальшого дослідження;

- проведене математичне моделювання задачі розпізнавання при роботі у системах з обмеженими ресурсами;

- проведено аналіз методів оптимізації обраного алгоритму та обрані оптимальні;

- розроблено та виконано програмну реалізацію оптимізованих версій обраного алгоритму розпізнавання облич;

- сплановано та проведено експериментальне дослідження стосовно ефективності нових алгоритмів;

- сформовано рекомендації стосовно використання методів оптимізації нейронних мереж для задачі розпізнавання обличча.

У результаті проведення експериментального дослідження та аналізу його результатів було розроблено презентацію (див. додаток В).

Були написані тези доповіді на участі у науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» (див. додаток Г). Також були подані тези на міжнародну науково-технічну конференцію «Наукові досягнення та відкриття сучасної молоді».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Национальный Открытый Университет «ИНТУИТ» . - URL: https://www.intuit.ru/ (дата звернення: 07.03.2022).

2. Coursera | Build Skills with Online Courses from Top Institutions. - URL: https://www.coursera.org/ (дата звернення: 07.03.2022).

3. Catalog - Stepik. - URL: https://stepik.org/catalog (дата звернення: 05.03.2022).

4. OpenTEST – программа тестирования знаний. - URL: http://opentest.com.ua/ (дата звернення: 08.03.2022).

5. Berle, I. Face Recognition Technology: Compulsory Visibility and Its Impact on Privacy and the Confidentiality of Personal Identifiable Images. – Berlin: Springer, 2020 – 228 p.

6. Фаулер, М. Шаблоны корпоративных приложений. - М.: Диалектика-Вильямс, 2019. - 544 с.

7. Facebox [Електронний ресурс] – URL: https://machinebox.io/docs/facebox (дата звернення: 10.03.2022)

8. REST API Tutorial [Електронний ресурс] – URL: https://www.restapitutorial.com/ (дата звернення: 12.03.2022).

9. Webber, J. Parastatidis S., Robinson I. REST in Practice – Себастопол: O'Reilly, 2010 – 448 с.

10. Szeliski, R. Computer Vision: Algorithms and Applications. — Berlin: Springer, 2010. — 812 p.

11. OpenCV: OpenCV Tutorials [Електронний ресурс] – URL: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (дата звернення: .05.2022)

12. Zhang K., Zhang Z., Li Z., Qiao Y., Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, Arxiv, 2016, URL: https://arxiv.org/abs/1604.02878

13. Simon J. D. Prince, Computer Vision: Models, Learning, and Inference. – Cambridge: Cambridge University Press, 2012. – 581 p.

14. Smelyakov, K., Chupryna, A., Bohomolov, O., Hunko, N. The Neural Network Models Effectiveness for Face Detection and Face Recognition 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences, eStream 2021 - Proceedings, 2021, 9431476 DOI 10.1109/eStream53087.2021.9431476

15. Charu C. Aggarwal, Neural Networks and Deep Learning: A Textbook. - Berlin: Springer, 2018 – 520 p.

16. Anton, H. Elementary Linear Algebra with Applications. – Hoboken: Wiley, 2005. – 386 p.

17. Stevens E., Deep Learning with PyTorch: Build, train, and tune neural networks using Python tools. – New York: Manning, 2020 – 520 p.

18. Steve R. Gunn, Support Vector Machines for Classification and Regression. – URL: http://ce.sharif.ir/courses/85-86/2/ce725/resources/root/LECTURES/SVM.pdf (дата звернення: 17.03.2022)

19. Hernandez, M. Database Design For Mere Mortals – Boston: Addison-Wesley Professiona, 2013 – 614 c.

20. LFW Face Database: Main . - URL: http://vis-www.cs.umass.edu/lfw/ (дата звернення: 20.03.2022).

21. Keras: The Python deep learnig API. - URL: https://keras.io/ (дата звернення: 22.03.2022).

22. PyTorch . - URL: https://pytorch.org/ (дата звернення: 22.03.2022).

23. TensorFlow . - URL: https://www.tensorflow.org/ (дата звернення: 23.03.2022).

24. A. Arsenov, I. Ruban, K. Smelyakov, A. Chupryna, Evolution of Convolutional Neural Network Architecture in Image Classification Problems // Selected Papers of the XVIII International Scientific and Practical Conference on Information Technologies and Security (ITS 2018). – CEUR Workshop Processing. – Kyiv, Ukraine, November 27, 2018. – Pp. 35-45.

25. Schroff F., Kalenichenko D., Philbin J., FaceNet: A Unified Embedding for Face Recognition and Clustering, Arxiv, 2015, URL: https://arxiv.org/abs/1503.03832

26. Qi C., Su F., Contrastive-center loss for deep neural networks, Arxiv, 2017, URL: https://arxiv.org/abs/1707.07391

27. Dokmanic I., Parhizkar R., Ranieri J., Vetterli M., Euclidean Distance Matrices: Essential Theory, Algorithms and Applications, Arxiv, 2015, URL: https://arxiv.org/abs/1502.07541