Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ

завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____» _____2023 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ІДЕНТИФІКАЦІЇ ОСОБИ ЗА БІОМЕТРІЄЮ ВУХА

Данило КУЛЬБАЧИНСЬКИЙ _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Виконав: студент групи МІТ-41
Керівник: доцент кафедри мережевих та інтернет технологій

к.т.н., доцент Оксана ГЕРАСИМЕНКО _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ-2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____» _____2023 року

Кафедра мережевих та інтернет технологій

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Кульбачинському Данилу Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Дослідження технологій ідентифікації особи за біометрією вуха

затверджена на засіданні кафедри МІТ «07» грудня 2022 р. протокол №5
2. Термін здачі закінченої роботи «30» травня 2023 р.
3. Вихідні дані до проекту (роботи)

Ear Dataset 28412 Ear images for 164 people

IIT Delhi Ear Database version 1.0

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)

Вступ

1. Аналіз та постановка задачі дослідження технологій біометричної ідентифікації осіб

1.1 Аналіз технологій та засобів біометричної ідентифікації особи

1.2 Математичні основи технологій нейронних мереж

1.2.1 Модель нейрона

1.2.2 Одношарова нейронна мережа

1.2.3 Багатошарова нейронна мережа

1.2.4 Аналіз методів навчання нейронних мереж

1.3 Застосування нейронних мереж для біометричної ідентифікації особи

1.4 Аналіз засобів реалізації задачі біометричної ідентифікації особи за біометрією вуха за допомогою згорткових нейронних мереж

1.5 Постановка задачі

Висновки по розділу I

2. Створення нейромережі для ідентифікації особи за біометрією вуха

2.1 Вибір інструментів реалізації задачі біометричної ідентифікації особи за біометрією вуха

2.2 Вибір типу нейронної мережі

2.3 Визначення архітектури нейронної мережі

2.4 Підготовка навчального набору даних
2.5 Навчання нейронної мережі
Висновки по розділу II
3. Аналіз показників якості створеної нейромережі на різних наборах даних та рекомендації щодо покращення запропонованої моделі
3.1 Аналіз показників якості навчання нейронної мережі
3.2 Шляхи покращення існуючої моделі
3.3 Приклади застосування створеного продукту
Висновки по розділу III
Висновок
Перелік посилань
Додатки

Дата видачі завдання 30.01.2023

Керівник роботи Оксана ГЕРАСИМЕНКО

(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Данило КУЛЬБАЧИНСЬКИЙ

(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	01.02.2022	
2	Розділ 1	01.03.2022	
3	Розділ 2	01.04.2022	
4	Розділ 3	15.05.2022	
5	Доповідь та слайди	30.05.2022	
6	Пояснювальна записка	30.05.2022	

Здобувач вищої освіти _____ Данило КУЛЬБАЧИНСЬКИЙ

(підпис)

Керівник _____ Оксана ГЕРАСИМЕНКО

(підпис)

РЕФЕРАТ

Пояснювальна записка: 67 с., 9 рис., 3 табл., 2 додатки, 19 джерел.

Об'єкт дослідження: процес ідентифікації особи за біометрією вуха.

Предмет дослідження: нейронні мережі як засіб визначення приналежності зображення вуха певній особі.

Мета роботи (проекту): спроектувати та реалізувати нейромережу для ідентифікації особи за біометрією вуха.

Методи дослідження: системний підхід, порівняльний аналіз, алгоритми навчання нейромереж.

У спеціальній частині розглянуто сучасний стан і особливості ідентифікації людей за їх біометричними даним.

У роботі проведено аналіз різних технологій нейронних мереж та алгоритмів навчання нейронних мереж.

Досліджено використання згорткової нейронної мережі та сіамської нейронної мережі (власної конфігурації та конфігурації зі стандартної бібліотеки) для ідентифікації особи за біометрією вуха.

Розроблено програмний код на мові програмування Python3 із використанням бібліотек машинного навчання Pytorch, TensorFlow для реалізації моделі згорткової нейронної мережі та сіамської нейронної мережі для ідентифікації особи за біометрією вуха.

Практична цінність роботи полягає у тому, що розроблено схему аутентифікації особи з використанням розробленої згорткової нейромережі.

Галузь використання – кібербезпека, авторизація.

Результати здійснених у дипломному проекті досліджень можуть бути використані для перевірки особи при авторизації на інтернет ресурси, виявлення потенційно небезпечних осіб у людному місці (вулиці, громадські місця) та інше.

Напрямки подальших досліджень полягають у вдосконаленні мережі, комбінування з іншими технологіями розпізнавання, що підвищить загалом точність розпізнавання особи, зміні способу отримування даних, наприклад: надати можливість ідентифікувати особу за допомогою камери відеоспостереження.

Ключові слова: ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, СІАМСЬКА НЕЙРОННА МЕРЕЖА, ІДЕНТИФІКАЦІЯ ОСОБИ ЗА БІОМЕТРИЧНИМИ ДАНИМИ, РОЗПІЗНАВАННЯ ОБРАЗІВ, КЛАСИФІКАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЗНМ Згорткова нейронна мережа

СНМ Сіамська нейронна мережа

ВСТУП

Машинне навчання та розпізнавання образів – дві сучасні галузі кібернетики, які мають високий потенціал для зростання. У зв'язку з потребою автоматизувати процеси в інтелектуальних системах останніми роками зріс інтерес до вирішення більш складних проблем ідентифікації об'єктів. Таким чином, надзвичайно важливо покращити реалізацію комп'ютерними системами розпізнавання об'єктів на фотографіях. Застосування штучного інтелекту і нейронних мереж, найдосконаліших інструментів, доступних для завдання розпізнавання образів, є одним із найперспективніших підходів до вирішення цієї проблеми.

У розпізнаванні об'єктів зображення було досягнуто значного прогресу завдяки зростанню популярності згорткових нейронних мереж. Їх ефективність і швидкий розвиток є результат гібридного підходу до архітектури нейронних мереж, використанням нових методів навчання.

Своєю перевагою згорткові нейронні мережі завдячують спільним вагам у згорткових шарах мережі, тобто певний фільтр використовується для кожного пікселя в шарі. У даній роботі створено модель згорткової нейронної мережі для ідентифікації особи за біометрією вуха.

Дане дослідження ставить за мету спроектувати та реалізувати нейромережу для ідентифікації особи за біометрією вуха. Ідентифікація особи за біометрією вуха у сукупності з іншими технологіями розпізнавання має потенціал бути надійним і зручним рішенням для ідентифікації. Дана задача є актуальною на сьогоднішній день, оскільки має багато сфер застосування: від ідентифікації особи серед групи людей до забезпечення доступу до конфіденційних даних у інформаційній системі чи на цифровому пристрої. Аналізуючи різні сценарії, це дослідження прагне дати практичне уявлення про потенціал біометрії вуха для використання у практичних задачах, зокрема автентифікація користувача при вході в інформаційну систему. Основними задачами дослідження є: дослідити можливості нейронних мереж по ідентифікації особи за біометрією вуха, розробити нейронну мережу для ідентифікації особи за біометрією вуха,

спроектувати схему використання розробленої мережі для аутентифікації особи при вході в інформаційну систему.

І. АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ ОСІБ

І.1 Аналіз технологій та засобів біометричної ідентифікації особи

У повсякденному житті люди не задумуються, що чимало технологій працюють заради їхньої безпеки. Можливістю швидко спіймати правопорушника та передбачити майбутні злочинні дії завдячуємо сьогоднішньому прогресу, який дозволяє швидко ідентифікувати особу. Події 11 вересня 2001 року привели до стрімкого розвитку технологій, що відповідають за безпеку.

Біометричні технології набувають все більшого значення як одні з технологій 21-го століття, що швидко розвиваються. Використовуючи біометричну інформацію можна достовірно ідентифікувати та верифікувати фізичних осіб. Значення терміну «біометрія» позначає лише автоматизовані методи встановлення чи розпізнавання будь-якої особистості за біологічними характеристиками чи проявами.

Біометрія – це систематичний та автоматизований спосіб ідентифікації осіб на основі їхніх фізіологічних або поведінкових характеристик. Прикладами фізіологічних характеристик є форма рук, форма обличчя, відбитки пальців, райдужна оболонка ока тощо. Також включають унікальні риси чи характеристики, набуті або розвинуті пізніше, наприклад: голос, письмо. Біометрія – це унікальні характеристики, які можна застосувати до особи для її автоматичної ідентифікації. Слово «автоматизована» означає, що біометрична система повинна ідентифікувати або верифікувати людину майже миттєво. Ідентифікація за допомогою біометричної системи передбачає порівняння вихідного біометричного зразка з уже записаними біометричними даними.

Біометрія розвивається вже більше п'яти десятиліть. Стандарти та тести вдосконалюють системи безпеки. Біометричні методи зараз є важливою складовою розвитку інформаційних технологій. Мільйони людей у всьому світі використовують біометричні технології ідентифікації та автентифікації. Значні

проекти реалізують як державні, так і комерційні структури. Використання біометричних технологій має вирішальне значення для успіху та конкурентоспроможності в суспільному житті. Це стосується приватних осіб, компаній і держав.

Нині існує безліч методів біометричної ідентифікації, які поділяються на дві групи: статичні та динамічні. До статичних можна віднести: відбиток пальця, форма долоні, райдужна оболонка ока тощо. До динамічних відносяться: почерк, динаміка клавіатурного набору, голос тощо.

Одним із прийомів, який набуває все більшої популярності, є розпізнавання обличчя людини. Люди без особливих зусиль сприймають один одного, однак автоматизувати подібне не так легко. Фото, необхідне для ідентифікації, отримується шляхом фотозйомки, використанням актуальних фотографій або відеозаписів камер спостереження. На відміну від інших біометричних технологій розпізнавання обличчя не вимагає контакту з людиною, особа якої встановлюється.

Елегантність ідентифікації людей за рисами обличчя полягає в тому, що вона ближче до природнього розпізнавання людей. Зростання мультимедійних технологій, зокрема все ширше використання відеокамер у громадських місцях таких як аеропорт, автобусна станція, освітні заклади та в різних місцях масового скупчення людей, привели до розвитку цього напрямку.

Існує багато підходів до розпізнавання людей за їх обличчям. Вони охоплюють наступні:

- техніка власних характеристик – аналіз зображення в градаціях сірого для сприйняття точних параметрів;
- техніка виявлення відмінностей – є найпопулярнішою, адаптована до модифікації міміки;
- методика автоматизованої обробки фотографій – визначення відстаней і співвідношення відстаней між рисами обличчя, які легко помітити – рекомендовано використовувати при поганому освітленні;

– методика, що базується переважно на нейронних мережах, – оцінка за допомогою унікальних точок, зокрема, які використовуються для ідентифікації фотографії обличчя людей у складних ситуаціях для відеоспостереження.

Також як додаткове джерело інформації можна використати біометрію вух. Рідко можна знайти інформацію про біометричні рішення, засновані на технології ідентифікації людини за формою вуха. Із зростанням можливостей цієї технології ця система також починає привертати увагу.

Експерти стверджують, що ідентифікація за формою вух має велике значення, оскільки ця особливість є однією з найбільш незмінних рис людини. Вуха розвиваються через 56 днів після зачаття, не змінюються протягом життя і зберігають свою форму після смерті, на відміну від інших носіїв біометричних характеристик людини. За винятком мочки вух, які можуть змінюватися під впливом певних чинників, форма вух зберігається довше, ніж форма обличчя чи візерунки пальців. Цей факт робить вухо особливо цінним інструментом для ідентифікації [1]. Його можна використовувати для розпізнавання в багатьох ситуаціях. У деяких випадках відбитки пальців не можуть бути використані для ідентифікації через вроджені дефекти, захворювання або травми. У таких випадках можна встановити особу загиблого за його фотографією шляхом порівняння зображення вуха.

Знімок вуха робиться наступним чином: спеціальний пластик прокатують на вусі вверх та вниз, зберігаючи лінії від вуха, які потім скануються за допомогою комп'ютерної програми і перетворюються на цифровий знімок, що зберігається у базі даних. Процес пошуку правильного зображення вуха не займає багато часу і вимагає чіткого зображення вушної раковини на фото та ретельного аналізу. Але, незважаючи на таку складну систему, деякі правоохоронні органи використовують цю технологію ідентифікації. Оскільки можна зробити знімок вуха на відстані, метод ефективний і в інших ситуаціях. Якщо відеокамера

зафіксує в кадрі зображення злочинця, то за зображенням вуха можна його впізнати.

Даний процес можна значно прискорити. Завдяки сучасним інформаційним технологіям, особливо, нейронним мережам, можливо побудувати програму, яка автоматизує збір інформації щодо вуха, а також співставлення нових даних та даних збережених в базі.

І.2 Математичні основи технологій нейронних мереж

І.2.1 Модель нейрона

Нейронні мережі – це технологія імітації нервових клітин мозку людини, які здійснюють обробку та передачу інформації.

Штучні нейрони – це базові одиниці, які складають основу будь-якої нейронної мережі. Функціональний нейрон можна назвати простим прикладом біологічного нейрона. Штучні нейронні мережі створюються в результаті поєднання штучних нейронів. Штучно створений нейрон, як і його природний прототип, має синапс (вхід), який з'єднаний з виходом попереднього нейрона, і вихідний канал цього нейрона – аксон, через який сигнал рухається до синапсів інших нейронів.

Штучний нейрон складається з двох компонентів – вагового суматора і нелінійного перетворювача. Вхід штучного нейрона, який отримує кожен сигнал, є виходом іншого нейрона (за вийнятком першого шару: він отримує дані ззовні). Щоб визначити рівень активності нейрона, кожен вхідний сигнал множиться на однакову величину і всі продукти множення додаються разом.

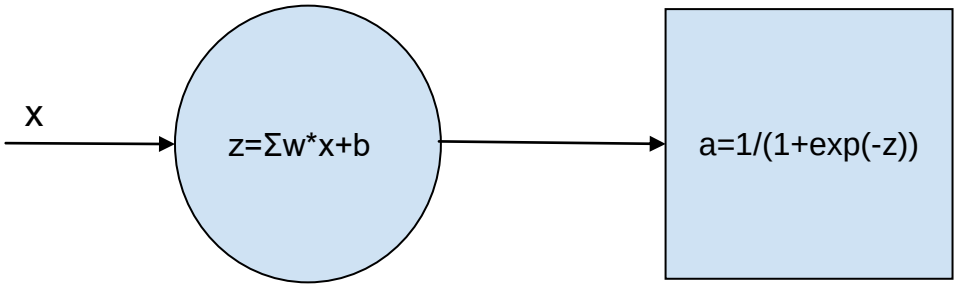

Рисунок 1.1 – Модель штучного нейрона

На рисунку 1.1 зображено схематичне представлення штучного нейрона, що складається з лінійної функції та функції активації, який приймає на вхід сигнал x.

I.2.2 Одношарова нейронна мережа

Одношарова нейронна мережа, також відома як перцептрон, є найпростішим типом функціональної нейронної мережі. Вона має вбудований шар нейронів, де кожен нейрон підключений до всіх вхідних сигналів. Цей тип мережі можна вважати лінійною моделлю, оскільки вона виконує лінійне множення вхідних сигналів для прийняття рішення.

В одношаровій нейронній мережі кожен нейрон має вагу, яка визначає його чутливість до вхідного сигналу. Кожен вхідний сигнал множиться на відповідний коефіцієнт, а результат додається. Після цього застосовується функція активації, яка трансформує в необхідну форму вихідний сигнал нейрона.

Функція активації в одношаровій нейронній мережі може бути буть-якою, наприклад, сигмоїдою, релу, софтмакс. Сигмоїда приймає значення від 0 до 1 і допомагає нейрону моделювати нелінійний базис даних. Порогова функція(релу) виводить 1, якщо значення вхідних даних перевищує один поріг, і 0 в іншому випадку.

Одношарову нейронну мережу можна використовувати для бінарної класифікації, оскільки вона ділить дані на дві категорії. Наприклад, у задачі розпізнавання зображення мережа може класифікувати зображення як собаку або кота.

Однак цей тип мережі обмежений у своїх можливостях і не може моделювати складні, нелінійні залежності. Це пояснюється тим, що одношарова нейронна мережа має лінійний характер і не може вирішувати проблеми, які задають складні задачі.

Застосування однорівневих нейронних мереж також може включати прості задачі регресії, де мережа намагається знайти зв’язок між вхідними характеристиками та вихідними значеннями.

Важливо відзначити, що одношарові нейронні мережі не можуть вирішити проблеми із складними наборами даних, такими як зображення та текст, або з тими, що вимагають складної взаємодії між об’єктами. Такі завдання вимагають комплексної організації нейронних мереж, таких як багатошарові перцептрони (багатошарові нейронні мережі), які можуть самостійно створювати складні нелінійні залежності та вирішувати складні завдання.

На рисунку 1.2 зображено шар, що складається з трьох нейронів, які приймають на вхід вектор сигналів X.

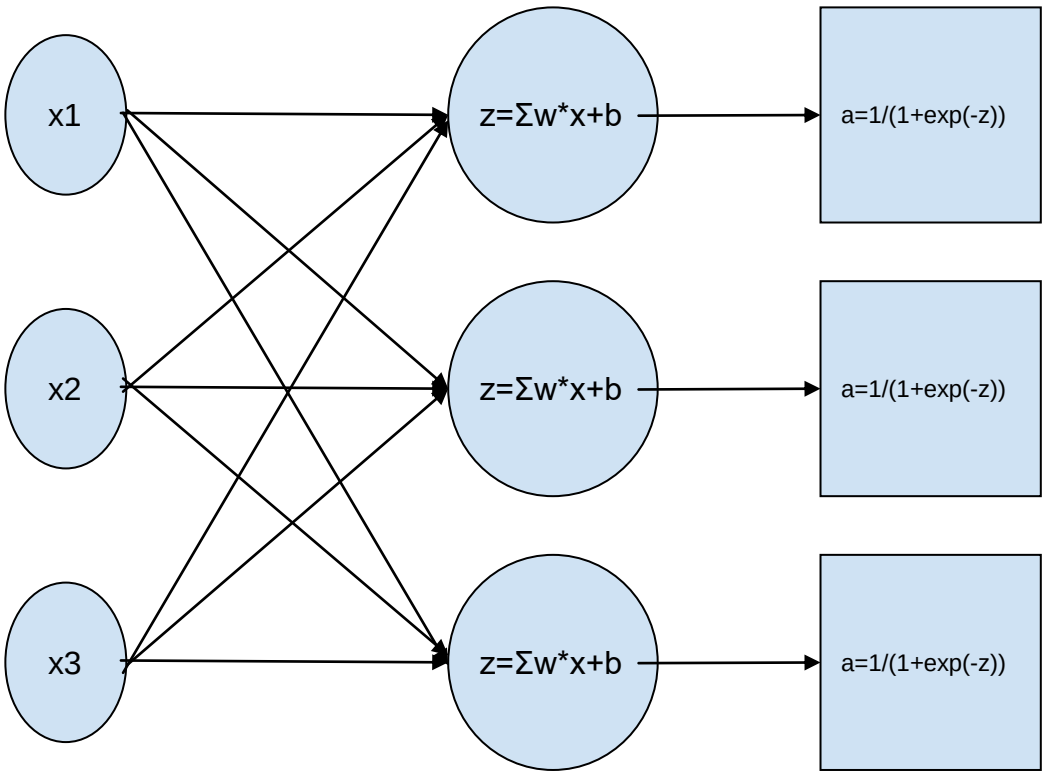

Рисунок 1.2 – Модель першого шару нейронної мережі

І.2.3 Багатошарова нейронна мережа

Багатошарова нейронна мережа — це нейронна мережа, яка має вхідний рівень, вихідний рівень і один або кілька прихованих шарів нейронів між ними. Багатошарова нейронна мережа може виконувати завдання практично будь-якої складності, а кількість рівнів і кількість елементів у кожному шарі визначають складність завдання. Важливе значення при проектуванні мережі має кількість проміжних шарів і кількість елементів у них.

У багатошарових нейронних мережах існує три основні класи нейронних мереж:

- мережі прямого поширення;
- мережі зворотнього поширення;
- повнозв’язні мережі.

Нейронні мережі прямого поширення припускають наявність декількох рівнів нейронів з зв’язками. Всі зв'язки спрямовані від вхідного нейрона до вихідного. Такі мережі також називаються багатошаровими персептронами і дещо схожі на звичайний персептрон, розроблений Розенблатом, який має лише один прихований шар.

Мережі зворотнього поширення припускають, що між нейронами різних шарів і між нейронами одного шару існують зворотні зв'язки. Ці нейронні мережі можуть мати властивості, подібні до короткочасної пам’яті людини. Сигнали в таких мережах повертаються на входи нейронів вхідного рівня від вихідних нейронів або нейронів прихованого шару.

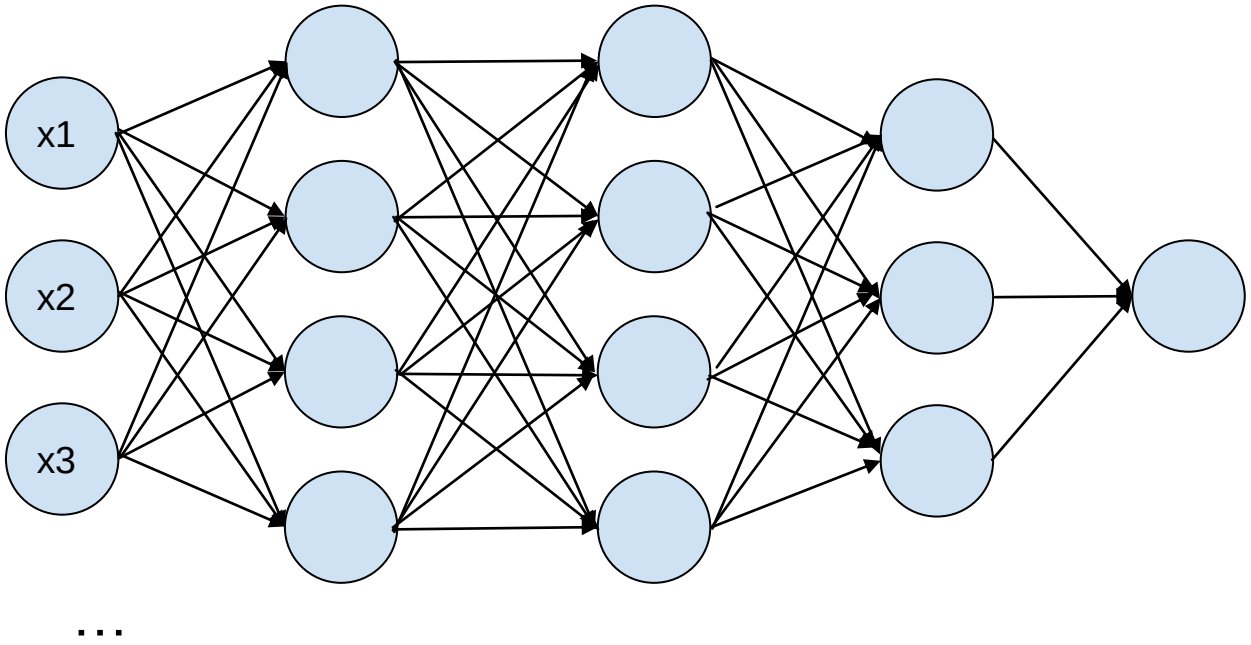

Рисунок 1.3 – Багатошарова нейронна мережа

Повнозв’язна нейронна мережа має унікальну особливість: вона має зв’язки між усіма нейронами. Найбільш відомим типом мережі є мережа Хопфілда. У такій мережі кожен нейрон має двонаправлені зв’язки з іншими нейронами мережі. Загалом мережа Хопфілда має симетричну кругову структуру, і приховані нейрони неможливо ідентифікувати, оскільки немає єдиного напрямку поширення.

Рисунок 1.3 зображує модель багатошарової нейронної мережі.

І.2.4 Аналіз методів навчання нейронних мереж

Навчання нейронних мереж – головна причина, чому дана технологія отримала таку популярність. Можливість ваг адаптуватись до вхідних даних завдяки їх стимуляції, дає свободу застосовувати нейронні мережі у різних сферах.

Це є вагомою перевагою перед традиційними алгоритмами. Однак і методи навчання можуть відрізнятися. Наведемо найбільш популярні з них:

– контрольоване навчання – під час контрольованого навчання модель навчається завдяки позначеним даним, з відомим, бажаним результатом.

Мережа коригує параметри в залежності від похибки між прогнозованим та бажаним виходом;

- неконтрольоване навчання – не надає помічені дані, мережа вчиться шукати закономірності, без будь-якого контролю;
- навчання з підкріпленням – мережа навчається через взаємодію з оточуючим середовищем, отримуючи в якості зворотнього зв’язку винагороду чи покарання. Модель вчиться виконувати дії, які максимізують сукупну винагороду;
- передавальне навчання – передбачає уже існування навченої моделі нейронної мережі яка використовувалась для відповідного завдання як відправної точки для нового завдання. Попередньо навчена мережа навчилася корисним функціям із великого набору даних і ці функції можна перенести та налаштувати для нового завдання з меншим набором даних, що призведе до швидшого та ефективнішого навчання;
- онлайн навчання – мережа оновлюється в міру появи нових даних, що адаптує мережу в режимі реального часу. Це підходить для сценаріїв, коли дані надходять потоками, і мережі потрібно поступово оновлювати свої знання;
- пакетне навчання – передбачає навчання мережі на фіксованому наборі даних, де весь набір даних обробляється як один пакет. Мережа обчислює помилки для всіх зразків та відповідно оновлює свої ваги. Використовується, коли набір даних може помістится в пам’ять.

Ці методи можна поєднувати та змінювати під вимоги завдання чи наданих даних. Вибір методу навчання залежить від наявної проблеми та наявних даних і ресурсів.

І.3 Застосування нейронних мереж для біометричної ідентифікації особи

Нейронні мережі часто використовуються для біометричної ідентифікації особи, оскільки вони здатні виявляти та розпізнавати унікальні біометричні характеристики людини, такі як обличчя, відбиток пальця, райдужна оболонка ока, голос тощо.

Використання нейронних мереж для цих цілей має багато переваг:

- точність розпізнавання: нейронні мережі здатні вчитися на великих даних і визначати складні закономірності, досягаючи високої точності;
- гнучкість: нейронні мережі можна навчити розпізнавати різні біометричні характеристики та адаптуватися до цих змін, наприклад, зачіски, рис обличчя, освітлення тощо;
- захист від шахрайства: нейронні мережі можуть виявляти спроби шахрайства, такі як використання фотографій, масок, або інші методи, які намагаються обманути системи біометричної ідентифікації.
- швидкість обробки: використання швидкого обчислювального обладнання, такого як графічний процесор (GPU), дозволяє нейронним мережам обробляти біометричні дані ефективно та в режимі реального часу.

Нижче наведено приклади застосування нейронної мережі для біометричної ідентифікації:

- розпізнавання обличчя: нейронні мережі можна використовувати для ідентифікації людей за зображеннями обличчя. Вони вчаться розпізнавати унікальні вирази обличчя та можуть краще порівнювати дані з баз даних;
- ідентифікація відбитків пальців: нейронні мережі можна використовувати для аналізу та порівняння відбитків пальців. Вони можуть навчитися розпізнавати унікальні характеристики відбитків пальців і швидко ними оперувати;
- розпізнавання райдужної оболонки: нейронні мережі можуть аналізувати унікальні особливості райдужної оболонки (наприклад, малюнок або

текстуру вен), щоб ідентифікувати людину. Їх можна навчити розпізнавати певні шаблони та використовувати їх для ідентифікації людини на основі класифікації райдужної оболонки ока;

– розпізнавання голосу: нейронні мережі можуть аналізувати акустичні характеристики голосів, щоб ідентифікувати людей. Їх можна навчити розпізнавати певні характеристики голосу, такі як висота, частота, інтонація тощо. Програми нейронної мережі для біометричної ідентифікації особистості постійно розвиваються та вдосконалюються, щоб підвищити точність, швидкість і безпеку системи ідентифікації.

Автоматична ідентифікація людей за зображеннями вух є активною областю досліджень у спільноті біометриків. Подібно до інших біометричних даних, вухо має велику кількість унікальних характеристик, які дозволяють ідентифікувати особу. Людське вухо є ідеальним джерелом даних для ідентифікації людей, оскільки воно не включає людську взаємодію, а структура вуха не надто змінюється з часом. Доступ до людського вуха також легкий, оскільки вухо видно навіть у випадку носіння маски. Біометричні характеристики вуха можуть доповнювати інші біометричні технології автоматизованих систем ідентифікації людини та забезпечувати ідентифікацію, коли інформація з інших систем є ненадійною або навіть недоступною.

І.4 Аналіз засобів реалізації задачі біометричної ідентифікації особи за біометрією вуха за допомогою згорткових нейронних мереж

Згорткові нейронні мережі є ключовими інструментами для ефективного розпізнавання зображень, розпізнавання облич і фотографій, використовуються в різних областях. Вони забезпечують стійкість до змін масштабу, поворотів, перспективи та інших спотворень.

Характеристики згорткових нейронних мереж:

- мають фіксовані вхідні значення та генерують однакові вихідні значення;
- мережа зворотнього поширення;
- використовується для обробки зображень і відео.

Мережа отримала свою назву від функції, що називається згорткою. У цій нейронній мережі використовуються три різні типи шарів: згортковий шар, шар об’єднання і повнозв’язний шар. Порядок цих рівнів визначає якість мережі.

ЗНМ призначена для вивчення та вилучення важливих функцій із вхідного зображення за допомогою серії згорткових і об’єднувальних рівнів. Ось детальний опис архітектури ЗНМ:

- вхідний шар: вхідний шар ЗНМ отримує необроблені значення пікселів зображення. Вхідне зображення зазвичай виражається як тривимірний тензор із розмірами (висота, ширина, канал). Довжина та ширина відповідають розмірам зображення, тоді як канали представляють кольорові канали (наприклад, червоний, зелений та синій у зображеннях RGB);
- згорткові шари: згорткові рівні є головною особливістю ЗНМ. Вони складаються з кількох фільтрів, які ковзають по вхідному зображенню та виконують поелементні операції множення та підсумовування. Кожен фільтр знаходить конкретну ознаку, наприклад висоту, текстуру або форму. Результатом згорткового шару є карта ознак, яка представляє функцію фільтра через просторові виміри вхідних даних;
- функція активації: після кожного згорткового шару поелементно застосовується функція активації для введення нелінійності в мережу. Функція активації, яка зазвичай використовується в ЗНН, – випрямлений лінійний вузол (ReLU), яка встановлює від’ємним значенням нуль і залишає додатні значення незмінними. ReLU забезпечує складну взаємодію між функціями;
- шар об’єднання: використовуються для зменшення просторових розмірів карт образів, зберігаючи найважливішу інформацію. Найпопулярнішою

операцією об’єднання є максимальне об’єднання (max-pooling), яка ділить карту образів та вибирає максимальне значення в кожній області. Максимальне об’єднання допомагає зменшити просторову роздільну здатність і контролювати перенавчання;

- повнозв’язний шар: також відомий як щільний шар, розміщується після згортки та об’єднання. Ці шари з’єднують кожен нейрон попереднього шару з нейронами поточного шару. Вони відіграють важливу роль у вивченні розширених представлень і створенні остаточних прогнозів. Кожен нейрон Повнозв’язного шару отримує вхідні дані від усіх нейронів попереднього шару;
- вихідний шар: вихід ЗНМ залежить від конкретного завдання, яке виконується. Для обробки зображень зазвичай використовується шар softmax, який генерує ймовірності для кожного класу. Клас з найвищою ймовірністю вважається прогнозованим класом. Для інших завдань, таких як виявлення об’єктів або класифікація, вихідний шар може відрізнятися;
- виключення: Щоб уникнути перенавчання, під час тренування виключення часто розміщують на повнозв’язних шарах. Виключення випадково встановлює частку спрацьовування нейрона на нуль під час кожного тренувального сеансу. Цей метод спонукає мережу вчитися виконувати більш складні та масштабовані завдання, зменшуючи залежність від конкретних нейронів;
- функція втрат: вимірює різницю між прогнозованим виходом і істинним виходом. Для класифікації зображень зазвичай використовується, категорійна перехресна ентропія, яка обчислює втрати на основі прогнозованих імовірностей класу та справжніх міток класу;
- оптимізація: для навчання ЗНМ використовується алгоритм оптимізації для мінімізації функції втрат. Популярним вибором є стохастичний градієнтний спуск (SGD) або його варіанти, такі як Adam або RMSprop. Ці алгоритми

коригують ваги та зміщення мережі на основі градієнтів функції втрат щодо параметрів мережі.

І.5 Постановка задачі

Завдання полягає в аналізі різних методів ідентифікації людей на основі біометрії вуха за допомогою згорткових нейронних мереж. Біометричне розпізнавання вуха — це метод автоматичного розпізнавання людини на основі унікальних характеристик вуха кожної людини, таких як форма, розмір, контур і розташування різноманітних анатомічних особливостей.

Метою дипломного проекту є пректування та реалізація нейромережі для ідентифікації особи за біометрією вуха.

Основними задачами дослідження є:

- дослідити можливості нейронних мереж по ідентифікації особи за біометрією вуха;
- розробити нейронну мережу для ідентифікації особи за біометрією вуха;
- розглянути можливості використання розробленої мережі для аутентифікації особи при вході в інформаційну систему.

Для досягнення мети необхідно виконати такі завдання:

- збір даних: необхідно зібрати великий набір даних із зображеннями вуха кожної людини. Ці дані можна отримати з різних джерел, таких як бази даних, набори даних або фотографії.
- попередня обробка: потрібно провести попередню обробку даних, таку як масштабування, згладжування, усунення шумів або нормалізація, щоб покращити якість вхідного зображення та забезпечити однорідність.
- вибір архітектури згорткової нейронної мережі: вибір відповідної архітектури згорткової нейронної мережі для аналізу зображення вуха. Це може включати різні комбінації згорткових шарів, шарів об’єднання, повністю зв’язаних шарів і шарів активації.

- навчання моделі: згорткові нейронні мережі навчаються на підготовлених даних для вивчення кореляцій і визначення корисних функцій, які допомагають ідентифікувати людей за допомогою біометрії вуха.
- оцінка та порівняння результатів: оцінка результатів розпізнавання та порівняння результатів роботи моделей. Це може включати порівняння точності, часу виконання, складності моделі та інших показників, щоб знайти найефективніший метод.

Висновки по розділу І

Біометричні технології мають велике значення для сучасного світу. Задачі ідентифікації виникають регулярно і, відповідно, потрібно знаходити рішення, що пришвидчать їх реалізацію. Використання ЗНМ є важливим засобом у сфері ідентифікації осіб за різними за біометричними характеристиками. Задача ідентифікації особи забіометрією вуха вимагає збору великого набору даних зображень вуха, їх попередньої обробки та використання згорткових нейронних мереж для розпізнавання особи на основі унікальних характеристик вуха.

Результати дослідження можуть відрізнятися в залежності від обраної архітектури згорткової нейронної мережі, методу обробки даних і показників оцінки. Оцінка результатів зазвичай включає точність розпізнавання, час виконання та складність моделі. Перевірка та тестування моделі на незалежних даних також є важливим етапом.

Дослідження в цій галузі мають значення для розробки біометричних систем, які мають широкий спектр застосувань, включаючи безпеку, контроль доступу, відновлення даних та інші сфери, де ідентифікація є важливим елементом.

ІІ. СТВОРЕННЯ НЕЙРОМЕРЕЖІ ДЛЯ ІДЕНТИФІКАЦІЇ ОСОБИ ЗА БІОМЕТРІЄЮ ВУХА

ІІ.1 Вибір інструментів реалізації задачі біометричної ідентифікації особи за біометрією вуха

Розвиток програмного забезпечення з відкритим вихідним кодом значно спростив застосування машинного навчання в мережах і на окремих пристроях за допомогою різноманітних мов програмування. Бібліотеки C++, Java, JavaScript, Python, R та багатьох інших мов мають приклади реалізації засобів для машинного навчання. Однак найчастіше, спеціалісти з області машинного навчання використовують мову програмування Python,та її бібліотеки.

З ряду причин Python часто вважають однією з найкращих мов програмування для побудови нейронних мереж, а саме:

- читабельність і простота: Python дуже читабельний і простий для розуміння завдяки чіткому та нескладному синтаксису. Це чудовий варіант як для новачків, так і для досвідчених розробників, оскільки він зосереджений на зручності читання коду;
- велика та активна спільнота: Python має значну та активну спільноту розробників, яка сприяє його розширенню та просуванню. Python підходить для різноманітних сфер, включаючи веб-розробку, аналіз даних, машинне навчання та наукові обчислення, завдяки великій кількості документації, навчальних посібників і великого вибору модулів і фреймворків;
- Python — це гнучка мова, яка переноситься та підходить для різноманітних програм. Кросплатформна сумісність дозволяє розробникам створювати код один раз і виконувати його без змін в інших системах. Використання Python у веб-розробці, сценаріях, автоматизації, наукових обчисленнях, штучному інтелекті та інших сферах демонструє його адаптивність;

- великі бібліотеки та фреймворки: Python має гарну екосистему бібліотек і фреймворків, які спрощують складні завдання та прискорюють розробку. Маніпуляції та аналіз даних стають можливими завдяки таким бібліотекам, як NumPy, Pandas і Matplotlib. Веб-розробка полегшується такими фреймворками, як Django і Flask, а проекти машинного та глибокого навчання — TensorFlow і PyTorch. Розробники можуть заощадити час і зусилля, використовуючи ці легкодоступні потужні інструменти;
- здатність Python взаємодіяти з іншими мовами програмування дозволяє розробникам використовувати вже існуючі ресурси та код. Його можна використовувати як мову сценаріїв у великих системах, його легко інтегрувати в програми C/C++. Python також має повну підтримку API, що робить можливим інтеграцію з іншими службами та системами;
- Python є надзвичайно зручною мовою для створення прототипів і швидкої розробки через те, наскільки вона проста і виразна. Завдяки стислому синтаксису розробники можуть швидко створювати код, тестувати концепції та швидко виконувати ітерації. Ця можливість дуже корисна в гнучких підходах до розробки, де дуже важливі короткий час виконання та негайний зворотний зв’язок.

Коротко розглянемо найпопулярніші бібліотеки Python для нейронних мереж:

- Pandas — одна з найпопулярніших бібліотек обробки та аналізу даних. Він надає структури даних, такі як DataFrames, які полегшують читання, маніпулювання та аналіз табличних даних. Pandas також має вбудовану підтримку для виконання операцій об’єднання, фільтрації, групування та агрегації даних;
- NumPy — це основна бібліотека для наукових обчислень на Python. Він надає потужні структури даних і функції для лінійної алгебри, статистики, генерації випадкових чисел тощо. NumPy використовується як основа для

багатьох інших бібліотек, включаючи Pandas і бібліотеки глибокого навчання;

- Matplotlib — це бібліотека візуалізації даних Python. Вона надає широкий спектр можливостей для створення графіків, діаграм, гістограм, діаграм розсіювання та інших типів візуалізації. Matplotlib дозволяє налаштовувати зовнішній вигляд графіків і додавати до них маркери, легенди, заголовки та інші елементи;
- SciPy — це колекція наукових бібліотек, які збільшують можливості NumPy. Він включає модулі, які оптимізують, обробляють сигнали, зображення, статистику та інше;
- TensorFlow є одним із найпопулярніших фреймворків для глибокого навчання та нейронних мереж. Він пропонує високоякісні API для створення, навчання та розгортання різних нейронних мереж, включаючи згорткові мережі, рекурентні мережі та генеративні моделі;
- PyTorch — це наукова, обчислювальна бібліотека, яка забезпечує потужну структуру для створення нейронних мереж. Він простий за інтерфейсом і підтримує різноманітні алгоритми;
- Scikit-learn — популярний пакет машинного навчання на Python. Він пропонує різноманітні алгоритми для класифікації, регресії, кластеризації та вимірювання важливості ознак, а також інші інструменти обробки та аналізу даних;
- OpenCV — це бібліотека комп’ютерного зору, яка надає багато функцій для створення зображень і відео. Зазвичай використовується для обробки даних перед подачею їх у нейронні мережі, включаючи розпізнавання об’єктів, розпізнавання облич та інші комп’ютерні програми.

ІІ.2 Вибір типу нейронної мережі

Вибір моделі нейронної мережі для розпізнавання людини та біометрії вуха залежить від вимог і характеристик програми. Розглянемо кілька типів нейронних мереж, які можна використовувати для цього завдання:

- згорткові нейронні мережі (CNN, ЗНН) широко використовуються в обробці зображень і добре працюють у розпізнаванні образів. Дана мережа може визначити форму вуха, наприклад форму, розташування ключових точок, частин тощо;
- рекурентна нейронна мережа (RNN) придатна для обробки послідовних даних, тобто вхідних сигналів, які показують зміни з часом;
- сіамська нейронна мережа (SNN, СНН) використовується для порівняння двох об’єктів. У випадку ідентифікації особи за біометрією вуха можна мати два зображення вуха для порівняння: вхідне зображення та те, що зберігається в базі даних. SNN може порівняти два зображення і відповісти на питання, чи належать вуха на них вони одній людині;
- автоенкодер — це нейронна мережа, яка вчиться відтворювати вхідні дані як вихідні. Щоб відтворити форму вуха, можна скористатися автоенкодер. Потім можна порівняти вхідне вухо з відтвореним вухом та визначити, наскільки добре вони відповідають один одному.

Це лише приклади типів нейронних мереж, які можна використовувати для ідентифікації людей за допомогою біометрії вуха. Вибір нейронної мережі залежить від програми, доступних даних і швидкодії.

Завдяки унікальній структурі ЗНМ надзвичайно корисні в розпізнаванні зображень. Вони чудово справляються з виявленням об’єктів, класифікацією зображень та розпізнаванням зображень.

Розпізнавання зображень – це технологія, яка використовується для розпізнавання та ідентифікації об’єктів на основі їхніх зображень. Воно базується

на аналізі унікальних рис, контурів, форми, розмірів, текстури та інших характеристик.

Класифікація зображень — це процес призначення категорій або міток зображенням на основі їхнього вмісту. Метою класифікації зображень є розподіл зображень на групи чи класи, що дозволяє ідентифікувати об’єкти чи елементи, представлені на зображенні.

Навчання нейронної мережі може зайняти багато часу, однак згорткові нейронні мережі досягли прогресу в цій області.

Основна ідея ЗНМ полягає в тому, щоб використовувати просторову структуру та локальні залежності, наявні в зображеннях.

Згорткові шари в ЗНМ застосовують фільтри до вхідного зображення, захоплюючи різні візуальні моделі, такі як краї, текстури або форми. Ці фільтри вивчаються в процесі навчання, коли мережа коригує свої параметри, щоб мінімізувати різницю між прогнозованими та справжніми мітками. Шари об’єднання зменшують розмірність, зберігаючи найважливішу інформацію. Нарешті, повнозв’язні шари об’єднують отримані ознаки та приймають остаточне рішення щодо класифікації.

Для навчання ЗНМ зазвичай потрібен великий набір даних із мітками. Нейромережі подаються вхідні зображення разом із відповідними мітками, а ваги мережі ітеративно коригуються за допомогою процесу, який називається зворотним поширенням. Мета полягає в тому, щоб оптимізувати параметри мережі, і тим самим підвищити точність передбачення правильних міток.

Коли ЗНМ навчено, її можна використовувати для класифікації зображень, подаючи на вхід нові зображення без міток. Мережа обробляє зображення та обчислює розподіл ймовірностей за різними класами. Клас із найвищою ймовірністю вважається прогнозованою міткою для цього зображення.

Основна ідея сіамської нейронної мережі полягає в тому, що дві або більше ідентичних нейронних підмережі обробляють різні вхідні дані та подібним чином вивчають їх представлення. Ці підмережі мають однакові набори параметрів,

тобто вони мають ту саму архітектуру та ваги. Після того, як вхідні дані проходять через кожну підмережу, вони порівнюються з використанням певної метрики подібності, наприклад, евклідової відстані чи косинусної подібності.

Процес навчання СНМ включає подачу пари вхідних даних, таких як два зображення, у вхідні шари кожної підмережі. Параметри обох підмереж змінюються під час навчання, з метою мінімізації відстані між представленнями вхідних даних однакових об'єктів та максимізації відстані між представленнями вхідних даних різних об'єктів. Це допомагає забезпечити, що подібні вхідні дані матимуть близькі представлення, тоді як неподібні дані матимуть віддалені представлення.

Наведені вище два типи нейронних мереж в подальшому будуть використані як базові для побудови моделей ідентифікації особи за зображенням вуха.

ІІ.3 Визначення архітектури нейронних мереж

Архітектура нейронної мережі визначає структуру та організацію нейронів та їх зв'язків у мережі. Це включає в себе кількість шарів, кількість нейронів у кожному шарі, тип активаційних функцій, спосіб зв'язків між нейронами та інші параметри.

Загалом архітектуру будь-якої нейронної мережі можна описати наступним чином: блок згортки, шар вирівнювання, повнозв’язний блок та вихідний шар.

Таблиця 2.1 – Архітектура моделі ЗНМ

Шар (тип)	**Параметри моделі**	**Вихідна форма**	**Треновані параметри**
Вхідні дані	-	180, 50, 32	0
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	90, 25, 32	320
Шар підвибірки	розмір пулу - 2, 2	90, 25, 32	0
Нормалізація	-	90, 25, 32	128

Шар (тип)	Параметри моделі	Вихідна форма	Треновані параметри
пакету			
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	45, 12, 32	9248
Шар підвибірки	розмір пулу - 2, 2	45, 12, 32	0
Нормалізація пакету	-	45, 12, 32	128
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	45, 12, 32	9248
Шар підвибірки	розмір пулу - 2, 2	22, 6, 32	0
Нормалізація пакету	-	22, 6, 32	128
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	22, 6, 32	18496
Шар підвибірки	розмір пулу - 2, 2	11, 3, 64	0
Нормалізація пакету	-	11, 3, 64	256
Шар вирівнювання	-	2112	0
Вихідний шар (Softmax)	-	222	469086
			Загалом: 507614

Код реалізації ЗНМ, описаної у таблиці 2.1, на мові Python матиме вигляд:

```
def model(in_1=32, in_2=32, in_3=32, in_4=64):
""" Function for returning schedule of accuracy and loss

    Parameters:
    accuracy (list): result of training
    loss (list): result of training
    val_accuracy (list): result of training
    val_loss (list): result of training

    Returns:
    None: -
    """
    model = Sequential()
   model.add(Conv2D(filters=in_1,kernel_size=3, strides=1, input_shape=(64,64,1), padding='same', activation =
'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(BatchNormalization())
```

```
model.add(Conv2D(filters=in_2,kernel_size=3, strides=1, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=in_3,kernel_size=3, strides=1, padding ='same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=in_4,kernel_size=3, strides=1, padding ='same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units = 222 , activation = 'softmax'))

model.compile( optimizer='Adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
return model
```

Вхідні дані – це фотографії вух, розміром 180x50, конвертовані в чорно-білий. Практично в усіх шарах використовується Relu функція активації, за вийнятком останнього. Кожен шар згортки має розмір фільтру (або ядра) 3x3 та має 32 фільтри на трьох шарах та 64 на останньому. Також кожен шар згортки має заповнення (padding), яке зберігає розмір вихідних даних. Перед вихідним шаром здійснюється вирівнювання, а саме перетворення зображення з двохвимірного масиву в одновимірний. На виході розташований шар, що містить 222 нейрони. Функція активації останнього шару – Softmax. Варто відмітити, що усі параметри можна регулювати заради найкращого результату. Для цього була реалізована відповідна функція на Python, за допомогою якої і були визначені параметри нейромережі (таблиця 2.1).

Дана модель здатна здійснювати розпізнавання осіб, на чиїх даних відбулось навчання. Однак її можна покращити, щоб мати можливість використовувати передбачення для людей, чиї дані не були в тренувальному наборі.

Змінивши вихідний шар з 222 нейронами та Softmax активацією на шар з одним нейроном та без функції активації, отримаємо модель, що повертає тільки тензор. Застосувавши функцію Евклідової відстані до тензорів (точки в просторі) з фотографій з відомим власником та без власника, зможемо знайти відстань між ними. Якщо відстань менша за певний поріг, то фотографії мають спільного власника, якщо ні, то їх власник різний.

Таким чином можна використовувати дану ЗНМ для ідентифікації особи. Наприклад, при реєстрації, зберігати значення (тензор), яке повертає модель, в базу даних і кожного разу при автентифікації порівнювати дані, надані при вході, з тими що є в базі даних.

Однак можна спробувати і інший підхід. Наприклад, використати сіамські нейронні мережі, котрі мають на вході два канали даних та зазвичай містять дві підмережі.

Таблиця 2.2 – Архітектура моделі СНМ

Шар (тип)	**Параметри моделі**	**Вихідна форма**	**Треновані параметри**
Вхідні дані 1	-	64, 64, 32	0
Вхідні дані 2	-	64, 64, 32	0
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	64, 64, 32	320
Шар підвибірки	розмір пулу - 2, 2	32, 32, 32	0
Нормалізація пакету	-	32, 32, 32	128
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	32, 32, 32	9248
Шар підвибірки	розмір пулу - 2, 2	16, 16, 32	0
Нормалізація пакету	-	16, 16, 32	128
Шар згортки (Relu)	розмір фільтра - 3 крок - 1	16, 16, 32	9248
Шар підвибірки	розмір пулу - 2, 2	8, 8, 32	0
Нормалізація пакету	-	8, 8, 32	128
Шар згортки	розмір фільтра - 3	8, 8, 64	18496

Шар (тип)	Параметри моделі	Вихідна форма	Треновані параметри
(Relu)	крок - 1		
Шар підвибірки	розмір пулу - 2, 2	4, 4, 64	0
Нормалізація пакету	-	4, 4, 64	256
Шар вирівнювання	-	1024	0
Виключення	швидкість виключення - 0.1	1024	0
Повнозв’язний шар (Tanh)	-	512	1049088
Вихідний шар (Sigmoida)	-	1	513

Код реалізації СНМ, описаної у таблиці 2.2, на мові Python матиме вигляд:

```
def siamese_model(in_1=32, in_2=32, in_3=32, in_4=64):
""" Function for creating siamese neural network model

    Parameters:
    in_1 (int): filters amount
    in_2 (int): filters amount
    in_3 (int): filters amount
    in_4 (int): filters amount

    Returns:
    model (model): neural network model
    """
    input_shape = (64,64, 1)
    input_1 = Input(shape=input_shape)
    input_2 = Input(shape=input_shape)


    shared_layers = [
        Conv2D(filters=in_1, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_2, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_3, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_4, kernel_size=3, strides=1, padding='same', activation='relu'),
```

```
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Flatten(),
    Dropout(0.1)
]

fc_layers = [
    Dense(units=512, activation='tanh'),
    Dense(units=1, activation='sigmoid')]
output_1 = input_1
output_2 = input_2
for layer in shared_layers:
    output_1 = layer(output_1)
    output_2 = layer(output_2)

concatenated = tf.keras.layers.Concatenate()([output_1, output_2])
for layer in fc_layers:
    concatenated = layer(concatenated)

siamese_model = Model(inputs=[input_1, input_2], outputs=concatenated)
siamese_model.compile(optimizer=Adam(), loss=contrastive_loss, metrics=['accuracy'])
return siamese_model
```

Сіамські нейронні мережі чудово підходять для вирішення задач класифікації чи порівняння.

Даний тип мереж містять у собі ідентичні нейронні підмережі, що обробляють різні вхідні дані. Завдяки цьому є можливість здійснити перевірку подібності фотографій та не перебудовувати мережу.

Під час тренування на вхід мережі подаються 2 зображення, кожне з яких проходить крізь усі рівні блоку згортки та шару вирівнювання. Опісля іде об’єднання двох вихідних тензорів в один і його подальша передача на повнозв’язний рівень та вихідний рівень. На вихідному рівні тензор проходить через функцію активації, що визначає, чи належать дані зображення одній людині, чи ні. Також варто зауважити, для даної моделі було використано дві різні функції втрат. Ця функція визначає різницю між прогнозованими значеннями моделі і правильними (очікуваними) значеннями вихідних даних. Зазвичай при

класифікації використовують функцію binary crossentropy, однак у цьому проєкті використано ще й функцію contrastive loss (контрастна втрата). Контрастна втрата використовується в задачах навчання без нагляду для навчання моделі розрізняти між парами схожих та несхожих прикладів.

Загалом дана модель є простішою та швидшою за рахунок більшого використання пам’яті.

Варто зауважити, що не обов’язково будувати усю мережу самостійно. Можна використати уже заздалегідь архітектурно визначені та треновані моделі, котрі є у вільному доступу, наприклад, у таблиці 2.2 представлено характеристики однієї з таких моделей від Google.

Таблиця 2.3 – Архітектура моделі СНМ з тренованим блоком MobilenetV3small

Шар (тип)	**Параметри моделі**	**Вихідна форма**	**Треновані параметри**
Вхідні дані 1	-	64, 64, 32	0
Вхідні дані 2	-	64, 64, 32	0
Тренований блок (MobilenetV3small)	вхідні дані - 64,64,3 ваги - “imagenet” виключити верхній шар	2, 2, 576	939120
Шар вирівнювання	-	1024	0
Повнозв’язний шар (Tanh)	-	512	2359808
Вихідний шар (Sigmoida)	-	1	513

Код реалізації СНМ, описаної у таблиці 2.3, на мові Python матиме вигляд:

```
def siamese_model2(in_1=32, in_2=32, in_3=32, in_4=64):
""" Function for creating siamese neural network model

    Parameters:
    in_1 (int): filters amount
    in_2 (int): filters amount
    in_3 (int): filters amount
    in_4 (int): filters amount
```

```
    Returns:
    model (model): neural network model
    """
    input_shape = (64,64, 3)
    input_1 = Input(shape=input_shape)
    input_2 = Input(shape=input_shape)

    base_model = tf.keras.applications.MobileNetV3Small(weights='imagenet', include_top=False,
input_shape=input_shape)

    fc_layers = [
        Dense(units=512, activation='tanh'),
        Dense(units=1, activation='sigmoid')
    ]



    output_1 = base_model(input_1)
    output_2 = base_model(input_2)

    flattened_1 = Flatten()(output_1)
    flattened_2 = Flatten()(output_2)

    concatenated = tf.keras.layers.Concatenate()([flattened_1, flattened_2])
    for layer in fc_layers:
        concatenated = layer(concatenated)

    siamese_model = Model(inputs=[input_1, input_2], outputs=concatenated)
    siamese_model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return siamese_model
```

Дана архітектура має у собі заздалегідь треновану модель MobilenetV3small, що є однією з моделей глибокого навчання, яку можна використовувати за допомогою бібліотеки TensorFlow. “ImageNet” вказує, що модель завантажує навчені ваги, які були попередньо треновані. Використовуючи цю модель можна здійснювати передбачення на зображеннях для розпізнавання об'єктів. Вона надає готовий інтерфейс для завантаження моделі та виконання передбачень без необхідності вручну налаштовувати всю архітектуру моделі. У додатках А та Б представлено повний код програм нейромереж.

Окремо варто виділити модель, побудовану на бібліотеці Pytorch.

Процес створення ЗНМ на основі бібліотеки Pytorch:

```
class CNN(nn.Module):
    def __init__(self,out_1=2,out_2=4, out_3=4):
        """ Function for creating convolutional neural network model

        Parameters:
        out_1 (int): filters amount
        out_2 (int): filters amount
        out_3 (int): filters amount

        Returns:
        None: -
        """
        super(CNN,self).__init__()
        self.cnn1=nn.Conv2d(in_channels=1,out_channels=out_1,kernel_size=2,padding=0)
        self.maxpool1=nn.MaxPool2d(kernel_size=2 ,stride=1)

        self.cnn2=nn.Conv2d(in_channels=out_1,out_channels=out_2,kernel_size=2,stride=1,padding=0)
        self.maxpool2=nn.MaxPool2d(kernel_size=2 ,stride=1)

        self.cnn3=nn.Conv2d(in_channels=out_2,out_channels=out_3,kernel_size=4,stride=2,padding=0)
        self.maxpool3=nn.MaxPool2d(kernel_size=4 ,stride=2)

        self.fc1=nn.Linear(out_3 * 13 * 13,164)

    def forward(self,x):
        x=torch.relu(self.cnn1(x))
        x=self.maxpool1(x)
        x=torch.relu(self.cnn2(x))
        x=self.maxpool2(x)
        x=torch.relu(self.cnn3(x))
        x=self.maxpool3(x)
        x=x.view(x.size(0),-1)
        x=self.fc1(x)
        return x
```

Дана модель відрізняється від попередніх своєю простотою та значно меншою кількістю фільтрів у кожному шарі. Незважаючи на, здавалось би, малу кількість фільтрів, вона є надзвичайно повільною під час тренування і вкрай неефективною. Процес навчання займав 5 годин, маючи лиш 10 ядер. Тому

вирішено було зосередитися тільки на роботі з бібліотекою TensorFlow, що виявилась швидшою.

II.4 Підготовка навчального набору даних

Будь-яка нейронна мережа вимагає набір даних, зазвичай, чим більший – тим краще. Навчальні дані використовуються для ініціалізації ваг мережі, які, власне, і будуть здійснювати обчислення для майбутнього передбачення.

Набір даних містить у собі зображення вух, які розподілені по директоріям, що в свою чергу також є мітками для зображень. Подібна ієрархія комфортна і дозволяє вільно користуватися інструментами бібліотеки TensorFlow для підготовки даних до тренування.

Даний етап завдав найбільше проблем, оскільки знайти якісний датасет вух, надзвичайно складно. Під час виконання роботи знайдено було тільки два:

- перший, який має надзвичайно велику кількість зображень 164 осіб (28412 фотографій), але чиї зображення не є фільтрованими та містять велику кількість «сміття»;
- другий містить чисті, обрізані та чіткі фотографії, але їх на 221 осіб тільки 754, що надзвичайно мало для чотирьох і більше -шарової ЗНМ. Мережа вже через 10 ітерацій зазнавала перетренування. Однак завдяки аугментації зображень та додавання шуму даний датасет став потужним джерелом тренувальних даних, що дозволили виконати близько 300 ітерацій без перетренування.

Код підготовки тренувальних даних має вигляд:

```
train_folder = "/home/jovyan/dataset2/train"
test_folder = "/home/jovyan/dataset2/test"
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range=0.2, zoom_range=0.2,
width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory(train_folder,
                                                 target_size = IMAGE_SIZE,
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
test_set = test_datagen.flow_from_directory(test_folder,
                                            target_size = IMAGE_SIZE,
                                            batch_size = 16,
                                            class_mode = 'categorical')
```

У даному фрагменті коду варто звернути увагу на:

- train_folder і test_folder – шляхи до директорій з даними;
- train_datagen та test_datagen – екземпляри класу ImageDataGenerator, який надає функціональність для генерації партій зображень з директорій та аугментації за допомогою налаштувань;
- training_set та test_set – є генераторами даних і можуть бути використані для навчання моделі. Кожна ітерація надасть партію зображень та міток з тренувального набору даних.

Для СНМ підготовка датасету дещо відрізняється. Так як на вхід сіамської моделі подається два зображення для перевірки їх подібності, то, відповідно, датасетів повинно бути два.

Код підготовки тренувальних даних для СНМ має вигляд:

```
def build_siamese_dataset(train_folder, mode='grayscale'):
    """ Function for creating siamese neural network model

    Parameters:
    train_folder (str): path to folder with images
    mode (str): determines color mode: rgb or grayscale

    Returns:
    x_train1 (list): train data
    x_train2 (list): train data
    """
    train_datagen = ImageDataGenerator(rescale=1./255)
    training_set = train_datagen.flow_from_directory(train_folder,
                                                     target_size=(64,64),
                                                     batch_size=32,
                                                     class_mode='categorical',
                                                     color_mode=mode,
                                                     shuffle=True,
                                                     seed=42,
                                                     subset='training')
    x_train1, x_train2 = [], []
    for i in range(len(training_set)):
```

```
        images, labels = training_set[i]
        num = len(images) -1 if len(images) % 2 != 0 else len(images)
        for o in range(num):
            if o % 2 == 0:
                x_train1.append(images[o])
            else:
                x_train2.append(images[o])
    return x_train1, x_train2

x_train1, x_train2 = build_siamese_dataset("/home/jovyan/dataset1/dataset2/train")
x_train2 = x_train2[:len(x_train2)+1] + x_train2[:len(x_train2):-1]
train_labels1 = [1] * (len(x_train2)//2) + [0] * (len(x_train2)//2) + [0]
x_train1 = np.array(x_train1)
x_train2 = np.array(x_train2)
train_labels1 = np.array(train_labels1)

x_test1, x_test2 = build_siamese_dataset("/home/jovyan/dataset1/dataset2/test")
x_test2 = x_test2[:len(x_test2)+1] + x_test2[:len(x_test2):-1]
test_labels1 = [1] * (len(x_test2)//2) + [0] * (len(x_test2)//2) + [0]
x_test1 = np.array(x_test1)
x_test2 = np.array(x_test2)
test_labels1 = np.array(test_labels1)
```

Даний код, крім використання класу ImageDataGenerator, будує 6 масивів даних: 3 тренувальних та 3 тестових (2 масиви зображень та 1 міток). Для моделі з MobileNetV3Small потрібно змінити параметр color_mode з 'grayscale' на 'rgb', оскільки MobileNetV3Small приймає зображення тільки з трьома каналами. Також виникла необхідність скоротити кількість екземплярів датасету через недостатню потужність комп’ютера.

II.5 Навчання нейронної мережі

Вище було представлено моделі, які будуть використані для тренування. Найскладніше під час тренування було підібрати оптимальні параметри. Тому було створено функцію, яка перебирала ці параметри, повертала результати та затрачений час. Код функції має вигляд:

```
def model_and_best_hyperparameters(nums):
    """ Function for training model

    Parameters:
    nums (list): list of filters' amount numbers

    Returns:
    execution time (int): how long the function took to execute
    """
    start_time1 = time.time()
    for i in nums:
        start_time2 = time.time()


        ready_model = model(i,i,i,i)        # change order
        trained_model = ready_model.fit(training_set, steps_per_epoch = 32, epochs = EPOCHS,
                                validation_data = test_set, validation_steps=16)


        print("for one training time taken: {:.2f} seconds".format(time.time() - start_time2)) # how much time one
training take


        SAVED_MODEL_BEST_ACCURACY_RESULTS.append(trained_model.history['accuracy'])
        SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS.append(trained_model.history['val_accuracy'])
    return time.time() - start_time1 # how much time all training take
```

Фінальний код тренування також дозволив зберегти ваги після тренування на постійному носії і має вигляд:

```
def final_model():
    ready_model = model()
    call_back = ModelCheckpoint(f"/home/jovyan/weights.hdf5", monitor='val_accuracy', verbose=0,
save_best_only=True, save_weights_only=False, mode='auto')
    trained_model = ready_model.fit(training_set, steps_per_epoch = 16 , epochs = EPOCHS,
                                validation_data = test_set, validation_steps=8 , callbacks = [call_back])
    SAVED_MODEL_BEST_ACCURACY_RESULTS2.append(trained_model.history['accuracy'])
    SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS2.append(trained_model.history['val_accuracy'])
    return trained_model, ready_model
history, trained_model = final_model()
```

Для CHM тренування мало чим відрізняється від попереднього, за вийнятком датасету, який передається, та кількість епох – 15 (у попередньому 300). Така різниця обумовлена великою швидкістю тренування CHM.

```
history = model1.fit([x_train1, x_train2],
```

```
train_labels1,
steps_per_epoch = 180,
epochs=15,
validation_data = ([x_test1,x_test2], test_labels1),
validation_steps=60,
shuffle=True)
```

СНМ з MobilenetV3small має 60 епох, заради більш якісного тренування.

Висновки по розділу II

Для реалізації задачі ідентифікації особи за біометрією вуха були обрані такі інструменти як: Python, Tensorflow, NumPy, Matplotlib, частково Pytorch.

При виборі інструментів для реалізації цієї задачі важливо враховувати функціональні можливості, а також наявність необхідних бібліотек і фреймворків для розробки нейромереж.

Для вибору типу нейронної мережі було розглянуто різні архітектури, такі як згорткові нейронні мережі, сіамські нейронні мережі або комбінацію різних типів мереж.

Архітектура нейронної мережі для ідентифікації особи за біометрією вуха повинна бути ретельно визначена, враховуючи специфіку завдання і можливості обраного типу мережі. Для виконання даного проєкту було вирішено зупинити вибір на ЗНМ, та двох СНМ, одна з яких розроблена власними зусиллями, а інша побудована на базі відкритих для доступу бібліотек.

Підготовка навчального набору даних є важливим етапом у створенні нейромережі. Для успішного тренування мережі необхідно мати достатню кількість чітких, фільтрованих зображень. У роботі було використано два набори даних, котрі знаходяться у вільному доступі.

Після підготовки навчального набору «IIT Delhi Ear Database version 1.0» можна перейти до навчання нейронних мереж на основі ЗНМ та СНМ. Цей процес включає передачу даних через мережу, підрахунок помилок і використання оптимізаційних алгоритмів для покращення роботи мережі.

ІІІ. АНАЛІЗ ПОКАЗНИКІВ ЯКОСТІ СТВОРЕНОЇ НЕЙРОМЕРЕЖІ НА РІЗНИХ НАБОРАХ ДАНИХ ТА РЕКОМЕНДАЦІЇ ЩОДО ПОКРАЩЕННЯ ЗАПРОПОНОВАНОЇ МОДЕЛІ

ІІІ.1 Аналіз показників якості навчання нейронної мережі

Точність моделі залежить від багатьох параметрів, однак є ті, вплив яких значно більший. Наприклад, кількість епох. Епоха – це один повний цикл проходження всього навчального набору через модель. ЗНМ (рис. 3.1, 3.2) при 300 епохах показала точність 91% на тренувальному наборі, та 92% на тестовому. Модель працює добре як із навчальними даними, так із тестовими.

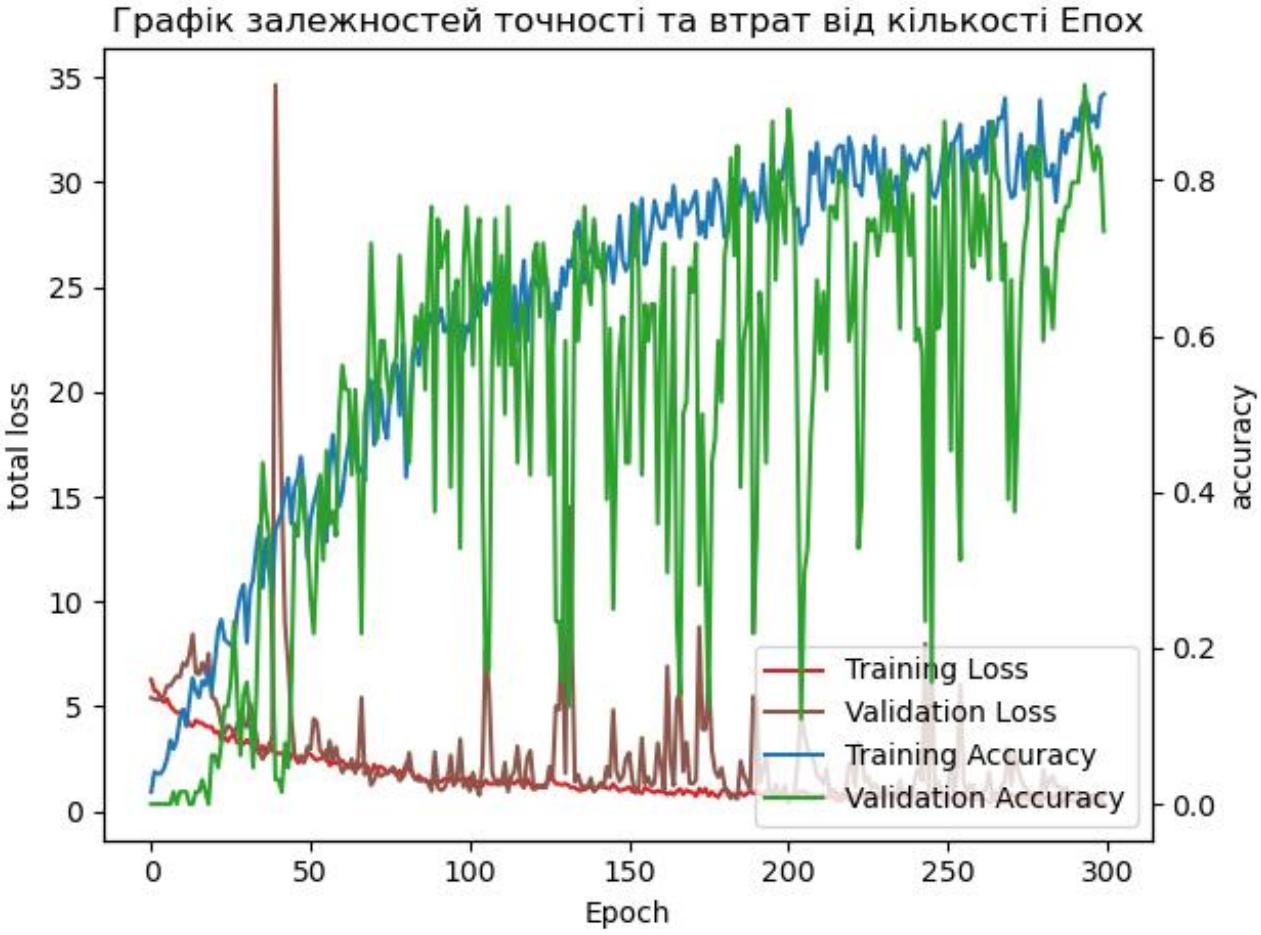

Рисунок 3.1 – Показники ЗНМ у залежності від номеру епохи

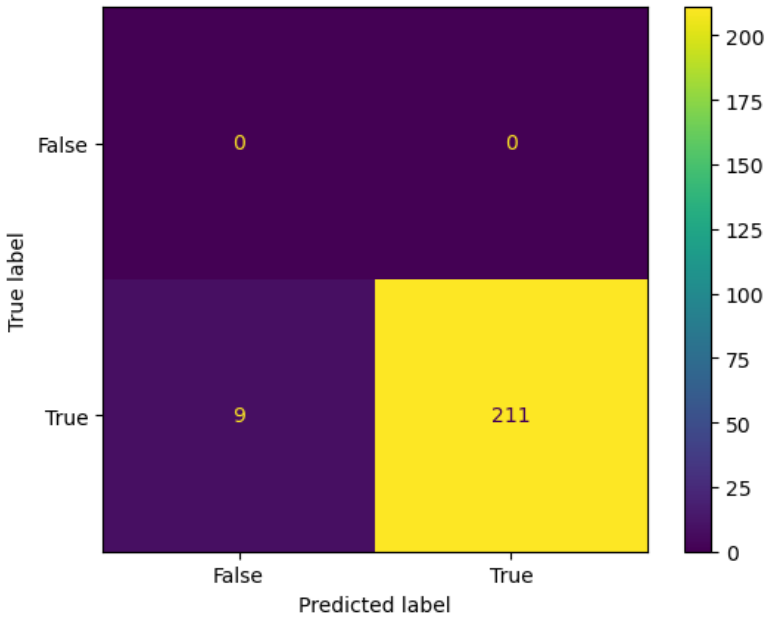

Рисунок 3.2 – Матриця невідповідностей ЗНМ

СНМ (рис. 3.3) показала результат гірший, 97% точності на тренувальному наборі та 50% на тестовому.

Виходячи з результатів, можна стверджувати, що проблема полягає у архітектурі моделей, а саме у СНМ. Наприклад, СНМ може вимагати чимало обчислювальних ресурсів, через природу своєї архітектури.

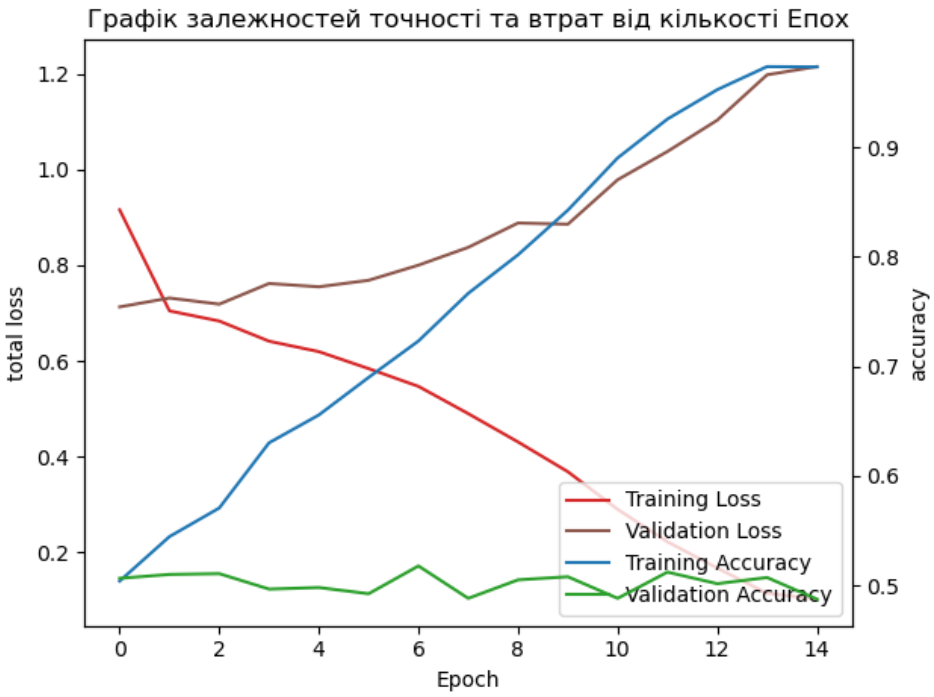

Рисунок 3.3 – Показники СНМ у залежності від номеру епохи

Застосування другого датасету дає аналогічні значення: кількість зображень надто мала, хоч він і є не засміченим, він надто швидко перетреновується, що унеможливлює його ефективне використання.

Варто зауважити, що СНМ з MobilenetV3small (рис. 3.4) має дещо кращий результат, однак у кінцевому рахунку, тестові дані показують точність також на рівні 50%.

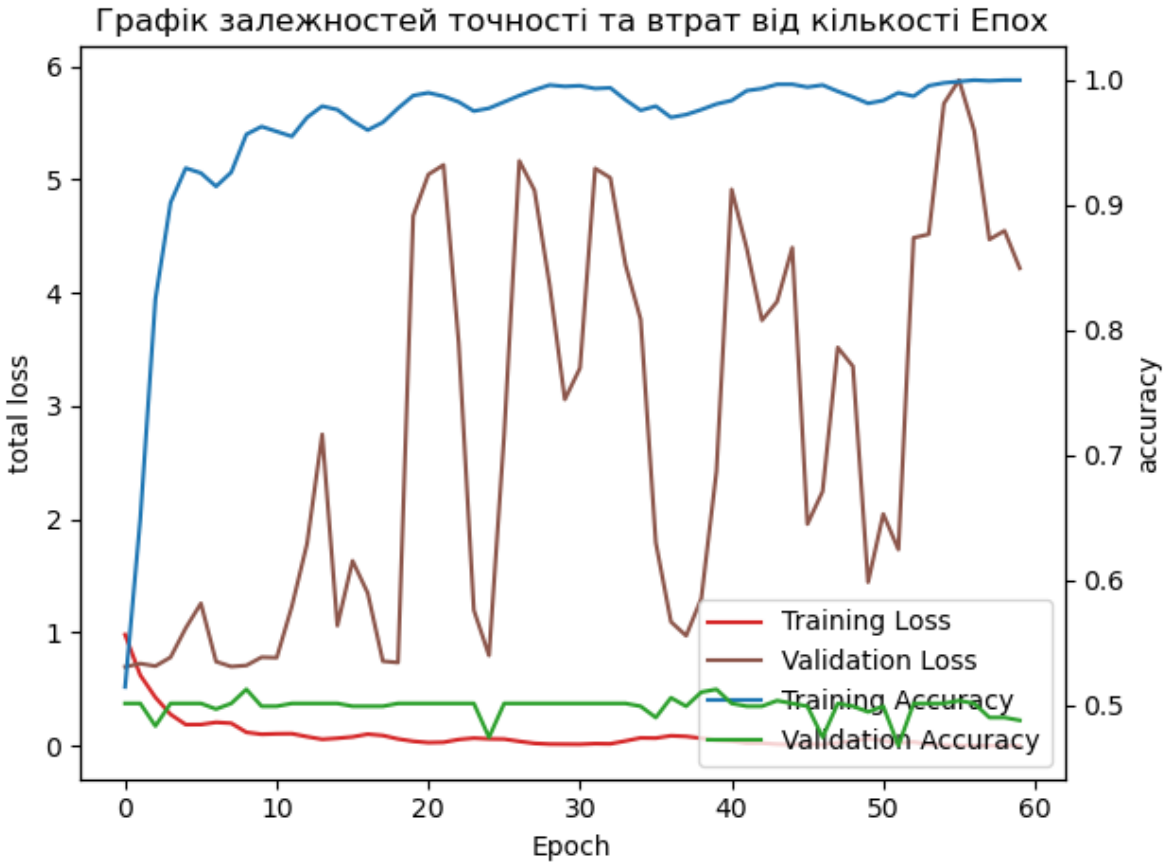

Рисунок 3.4 – Графік результатів СНМ з блоком MobileNetV3Small

На рисунку 3.4 представлено код, який використовується для визначення подібності між зображеннями з використанням заданої у якості параметра моделі.

Як результат, можна бачити, що розроблену ЗНМ можна використовувати для ідентифікації особи. Хоч і є вірогідність отримання помилки на рівні 10%,

тим не менше шляхом завантаження і перевірки кількох зображень ці помилки можна нівелювати.

```python
def distance_metric(x1, x2):
    return tf.sqrt(tf.reduce_sum(tf.square(x1 - x2), axis=1))

def predict(model, known_data_path, unknown_data_path):
    """ Function for training model

    Parameters:
    model (): neural network model
    known_data_path (str): path to image
    unknown_data_path (str): path to image

    Returns:
    prediction (bool): identification
    distance (float): how far are examples from each other
    """
    known_data = get_image_and_transform(known_data_path)
    unknown_data = get_image_and_transform(unknown_data_path)

    known_data = model.predict(known_data)
    unknown_data = model.predict(unknown_data)
    distance = distance_metric(unknown_data, known_data)

    threshold = 1.2
    return  bool(distance < threshold), float(distance[0])

predict(predict_model, '/home/jovyan/r1.jpg', '/home/jovyan/12.jpg')
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 15ms/step
(True, 0.4879112243652344)
```

Рисунок 3.4 – Використання моделі задля передбачення та його результат

III.2 Шляхи покращення існуючих моделей

Дані моделі мають спільні недоліки – навчальні дані, обчислювальні потужності, підбір параметрів.

Один із способів покращення існуючих моделей полягає в розширенні набору даних, на яких вони навчаються. Більший за обсягом та різноманітніший набір даних може допомогти моделі зрозуміти більш широкий спектр сценаріїв та патернів, що впливають на її роботу. Додавання нових даних може забезпечити кращу універсальність та загальну репрезентативність моделі. Також можна

провести фільтрування фотографій з першого датасету, щоб відокремити сміття від якісних фотографій.

Вручну проведене фільтрування даних може вимагати часу і зусиль, але дозволяє впевнено визначити, які фотографії є якісними, а які можуть бути виключені з навчального набору. Такий підхід буде особливо корисним, якщо потрібно забезпечити високу якість фотографій.

Алгоритми кластеризації можуть також бути використані для автоматичного розподілу фотографій на різні групи. Кластеризація дозволяє групувати схожі дані разом, ґрунтуючись на їхніх характеристиках. Наприклад, алгоритм може виділити групи, що містять якісні фотографії з різних ракурсів, з різною освітленістю тощо, і окремі групи, які містять фотографії з низькою якістю або «сміття». Такий підхід може допомогти забезпечити, що навчальні дані матимуть більшу якість та репрезентативність.

Ще одним шляхом покращення є вдосконалення методів навчання моделей. Використання більш ефективних алгоритмів оптимізації, які дозволяють швидше знаходити оптимальні параметри моделі, може призвести до покращення її якості. Також можуть бути використані різні методи регуляризації, які допомагають уникнути перенавчання та покращити узагальнюючу здатність моделі.

Загалом, комбінація цих підходів та збільшення обчислювальних потужностей може допомогти покращити якість існуючих моделей і забезпечити їх кращу продуктивність у завданнях обробки зображень. Кінцевим результатом буде ефективна система ідентифікації особи. Вуха є унікальними для кожної людини, а тому при високій точності моделі даний підхід зможе змагатися з іншими біометричними даними, такі як обличчя, долоні, сітківка ока, голос тощо.

ІІІ.3 Приклади застосування створеного продукту

Технологія ідентифікації за біометрією вуха людини використовує унікальні характеристики вуха для ідентифікації особи. Ця технологія може мати різноманітні застосування в різних сферах. Розглянемо кілька прикладів застосування.

Безпека і контроль доступу: системи контролю доступу можуть використовувати технологію ідентифікації за біометрією вуха для перевірки особи перед наданням доступу до обмежених приміщень або ресурсів. Це може бути особливо корисно в сферах, де важлива висока точність ідентифікації, наприклад, у фінансових установах або лабораторіях з обмеженим доступом.

Аутентифікація на пристроях: технологія ідентифікації за біометрією вуха може бути використана для аутентифікації особи на різних пристроях, таких як смартфони, планшети або ноутбуки. Замість введення пароля або використання відбитку пальця, пристрій може сканувати разом з обличчям, або ж навіть замість нього, вуха користувача, для підвищення рівня безпеки.

У рамках виконання роботи розроблено схему аутентифікації користувача з використанням розробленої ЗНМ, яка представлена на рисунку 3.5. Вона може бути базовою для перевірки особи при вході в інформаційну систему.

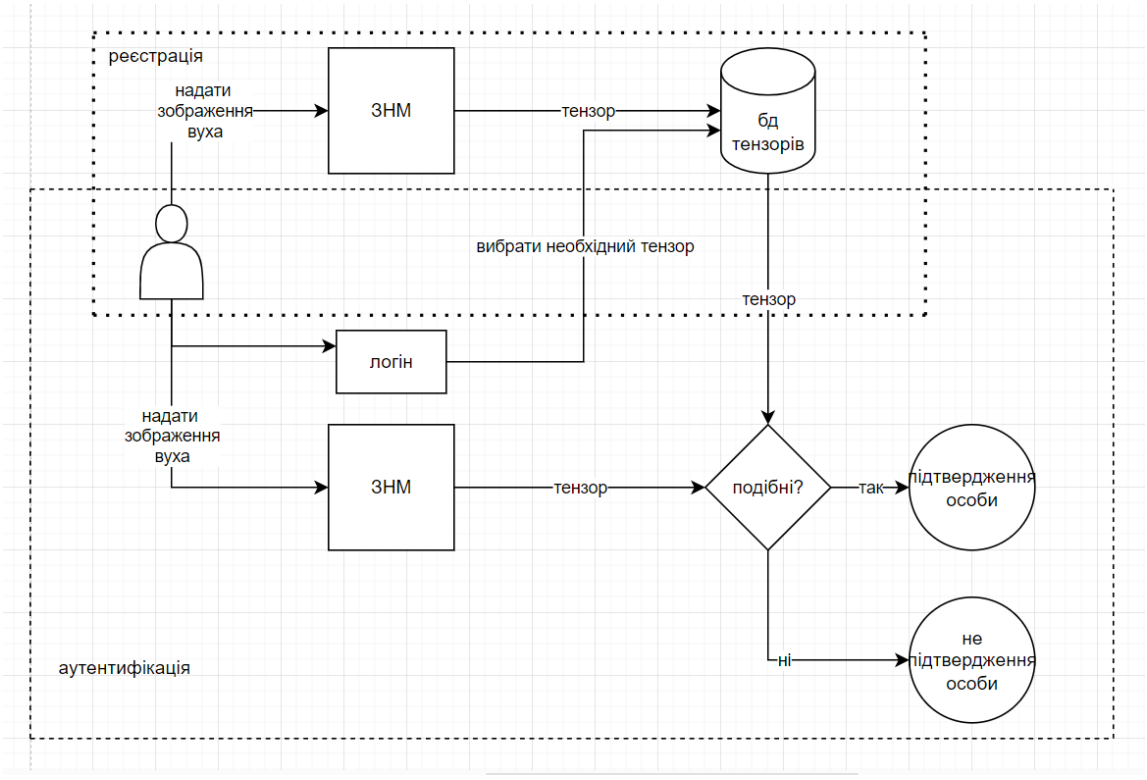

Рисунок 3.5 – Схема використання ЗНМ для аутентифікації користувача при вході в інформаційну систему

Масові заходи безпеки: технологія може бути використана для ідентифікації людей на масових заходах безпеки, таких як аеропорти, стадіони або концертні зали. Вона може допомогти управляти безпекою, швидко розпізнаючи осіб, які можуть нести загрозу.

Ці приклади демонструють, що технологія ідентифікації за біометрією вуха людини має широкі застосування і може бути корисною для поліпшення безпеки, зручності та ефективності у різних галузях.

Висновки по розділу III

Аналіз показників якості навчання нейронної мережі показав, що точність моделі ЗНМ є задовільною та можливою для використання в задачі ідентифікації особи. ЗНМ при 300 епохах показала точність 91% на тренувальному наборі та 92% на тестовому. Модель працює добре з навчальними даними та справляється з тестовими.

СНМ показала результат гірший (при 15 та 60 епохах), 97% точності на тренувальному наборі та 50% на тестовому, що говорить про необхідність її подальшого дослідження і вдосконалення.

Підсумовуючи, можна стверджувати, що проблема полягає у архітектурі моделей, а саме у СНМ. Наприклад, СНМ може вимагати чимало обчислювальних ресурсів через природу своєї архітектури. Оскільки робота виконувалась на персональному комп’ютері, то для цієї мережі довелось зменшити розмір датасету, що також суттєво вплинуло на результат. Для прикладного використання необхідно вдосконалити модель для досягнення вищої точності. Одним із шляхів покращення може бути збільшення обсягу та якості навчальних даних, що використовуються для тренування моделі.

Вуха є унікальними для кожної людини, тому ефективна модель ідентифікації за біометрією вуха може бути конкурентоспроможною порівняно з іншими біометричними даними, такими як обличчя, долоні або сітківка очей.

Потенційні приклади застосування створеного продукту можуть включати безпеку та контроль доступу, аутентифікацію на пристроях та застосування на масових заходах безпеки. У цьому розділі як приклад представлено базову схему використання розробленої ЗНМ для аутентифікації користувача при вході в інформаційну систему.

ВИСНОВОК

Біометричні технології, зокрема технології біометричної ідентифікації осіб, відіграють важливу роль у сучасному світі. Задачі ідентифікації виникають регулярно і вирішення цих задач є надзвичайно важливим для забезпечення швидкості та точності ідентифікації. Одним із підходів до біометричної ідентифікації є використання нейронних мереж.

Аналіз технологій та засобів біометричної ідентифікації особи дозволяє оцінити різні підходи та методи, які використовуються для виявлення та розпізнавання біометричних характеристик. Дослідження математичних основ нейронних мереж дає змогу зрозуміти принципи їх функціонування, зокрема модель нейрона, одношарові та багатошарові нейронні мережі.

Особлива увага у роботі приділяється аналізу методів навчання нейронних мереж, оскільки ефективність та точність нейромережевих моделей залежить від правильного підбору методу навчання. Застосування нейронних мереж для біометричної ідентифікації особи відкриває широкі перспективи в розробці систем, які забезпечують високу точність та надійність ідентифікації.

Для реалізації ідентифікації особи за допомогою біометрії вуха було використано згорткові нейронні мережі. Ця задача передбачає збір великого набору даних зображень вуха, їх попередню обробку та використання згорткових нейронних мереж для виявлення та розпізнавання унікальних ознак на зображеннях вуха.

Згорткові нейронні мережі є потужним інструментом у глибинному навчанні, оскільки вони можуть автоматично виявляти та виділяти різноманітні характеристики об'єктів на зображеннях. Також вони використовуються для виявлення різних шаблонів та ознак на зображеннях, зокрема контурів, текстур, особливостей форми тощо.

Аналіз показників якості нейронної мережі показав, що точність моделі ЗНМ, розробленої під час дипломного проєктування, є задовільною та дозволяє її використання в задачі ідентифікації особи. ЗНМ при 300 епохах показала точність

91% на тренувальному наборі та 92% на тестовому. Як приклад прикладного застосування розробленої моделі ЗНМ у роботі представлено базову схему її використання для аутентифікації користувача при вході в інформаційну систему.

СНМ показала гірший результат – 97% точності на тренувальному наборі та 50% на тестовому, що говорить про необхідність її подальшого дослідження і вдосконалення. Можна стверджувати, що проблема полягає у архітектурі моделей, СНМ може вимагати чимало обчислювальних ресурсів через природу своєї архітектури. Оскільки робота виконувалась на персональному комп'ютері, то для цієї мережі довелось зменшити розмір датасету, що також суттєво вплинуло на результат. У роботі запропоновано шляхи покращення даної моделі.

Загалом, розвиток технології ідентифікації за біометрією вуха відкриває можливості для покращення безпеки, зручності та ефективності в різних сферах, але потребує подальших досліджень та вдосконалення моделей для досягнення високої точності і надійності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Захаров В. П., Рудешко В. І. Біометричні технології в XXI столітті та їх використання правоохоронними органами: посібник. – 2-ге вид., доп. / В. П. Захаров, В. І. Рудешко. – Львів: ЛьвДУВС, 2015. – 492 с.
2. R. Ahila Priyadharshini, S. Arivazhagan, M. Arun. A deep learning approach for person identification using ear biometrics [Електронний ресурс]. – Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7594944/
3. Practical Deep Learning [Електронний ресурс]. – Режим доступу: https://course.fast.ai/
4. Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD [Електронний ресурс]. – Режим доступу: https://course.fast.ai/Resources/book.html
5. Машинне навчання [Електронний ресурс]. – Режим доступу: https://apps.prometheus.org.ua/learning/course/course-v1:IRF+ML101+2016_T3/home
6. Machine Learning with Python [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/machine-learning-with-python?specialization=ai-engineer
7. Introduction to Deep Learning & Neural Networks with Keras [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/introduction-to-deep-learning-with-keras
8. Introduction to Computer Vision and Image Processing [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/introduction-computer-vision-watson-opencv
9. Deep Neural Networks with PyTorch [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/deep-neural-networks-with-pytorch
10. Building Deep Learning Models with TensorFlow [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/building-deep-learning-models-with-tensorflow

11. Професійна сертифікація 'IBM AI Engineering' [Електронний ресурс]. – Режим доступу: https://www.coursera.org/professional-certificates/ai-engineer#courses
12. Kaggle [Електронний ресурс]. – Режим доступу: https://www.kaggle.com/
13. NumPy Tutorial [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/python/numpy/default.asp
14. SciPy Introduction [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/python/scipy/scipy_intro.php
15. Matplotlib Tutorial [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/python/matplotlib_intro.asp
16. Pandas Tutorial [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/python/pandas/default.asp
17. A friendly introduction to Siamese Networks [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942
18. Siamese networks with Keras, TensorFlow, and Deep Learning [Електронний ресурс]. – Режим доступу: https://pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/
19. Comparing images for similarity using siamese networks, Keras, and TensorFlow [Електронний ресурс]. – Режим доступу: https://pyimagesearch.com/2020/12/07/comparing-images-for-similarity-using-siamese-networks-keras-and-tensorflow/

ДОДАТОК А

РЕАЛІЗАЦІЯ ЗНМ ЗАСОБАМИ PYTHON

```
import libraries

import tensorflow as tf
import numpy as np
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pylab as plt
from keras.preprocessing import image
from tensorflow.keras import utils
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
import shutil
import splitfolders
import time
# !unzip /home/jovyan/dataset2/221.zip -d /home/jovyan/dataset2/

# !rm -r /home/jovyan/dataset2/dataset2

print(tf.__version__)

functions

SAVED_MODEL_BEST_ACCURACY_RESULTS = []
SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS = []
SAVED_MODEL_BEST_ACCURACY_RESULTS2 = []
SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS2 = []
```

```
IMAGE_SIZE = (64,64)
EPOCHS = 170 # try different # 170 for dataset1
def get_image_and_transform(path):
    data = utils.load_img(path, color_mode='grayscale', target_size=IMAGE_SIZE)
    data = utils.img_to_array(data)
    data = np.expand_dims(data, axis=0)
    # data = data / 255.0
    return data
def build_dataset(dataset_dir): # build 2 datasets for training and testing
    x_train, y_train = [], []
    classes = sorted(os.listdir(dataset_dir))
    for i in range(len(classes)):
        class_dir = os.path.join(dataset_dir, classes[i])
        images = os.listdir(class_dir)
        for image in images:
            full_image_path = os.path.join(class_dir, image)
            if 'ipynb_checkpoints' in full_image_path:
                continue
            x_train.append(get_image_and_transform(full_image_path))
            y_train.append(i)
    return x_train, y_train
def get_examples_and_classes(dataset_dir, num): # return persons' names and persons' ears
    classes = sorted(os.listdir(dataset_dir))
    classes.remove('.ipynb_checkpoints') if '.ipynb_checkpoints' in classes else None
    examples = []
    for i in range(len(classes)):
        class_dir = os.path.join(dataset_dir, classes[i])
        images = os.listdir(class_dir)
        examples.append(os.path.join(class_dir, images[num]))

    return classes, examples
def plot(accuracy, loss, val_accuracy, val_loss): # return schedule of accuracy and loss
    fig, ax1 = plt.subplots()
    ax1.set_xlabel('epoch')

    ax1.set_ylabel('total loss')
    ax1.plot(loss,color='tab:red')
    ax1.plot( val_loss, color='tab:brown')
    ax1.tick_params(axis='y')

    ax2 = ax1.twinx()
```

```
    ax2.set_ylabel('accuracy')
    ax2.plot( accuracy, color='tab:blue')
    ax2.plot( val_accuracy, color='tab:green')
    ax2.tick_params(axis='y')
    fig.tight_layout()
def plot_many(title='title', data=[]): # return schedules of accuracy
    for i,o in enumerate(data):
        plt.figure(i)
        plt.plot([i for i in range(EPOCHS)],o, label=title)
        plt.legend(loc='upper left')
def save_lists(): # save lists of accuracy and loss
    t = time.ctime(time.time())[:-5]
    with open(f'accuracy {t}.txt', 'w') as f:
        for item in SAVED_MODEL_BEST_ACCURACY_RESULTS:
            f.write("%s\n" % item)
    with open(f'val_accuracy {t}.txt', 'w') as f:
        for item in SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS:
            f.write("%s\n" % item)
def read_list_from_file(filename): # read accuracy or loss data  from files
    lst = []
    with open(filename, 'r') as file:
        for line in file.readlines():
            if line in ('', ' ','\n'):
                break
            line = line[1:-2].split(', ')
            lst.append([float(i.strip()) for i in line])
    return lst
# file = 'accuracy Sat May  6 11:19:23.txt'
# l = read_list_from_file(file)
# print("sorted list of max  epochs' values for accuracy  ", show_best_res(l))
def show_best_res(list_of_data): # return max value in every list of accuracy
    max_val = {}
    for i, o in enumerate(list_of_data):
        max_val[i] = max(o)
    return dict(sorted(max_val.items(), key=lambda item: item[1], reverse=True))
def create_new_dataset(path):
    images = sorted(os.listdir(path))
    for image in images:
        if 'ipynb_checkpoints' in image:
            continue
        old_path = path + '/' + image
```

```
        new_path = f"/home/jovyan/dataset2/dataset2/{image[:3]}"

        if os.path.exists(new_path):
            shutil.copyfile(old_path, new_path+'/'+image)
        else:
            os.mkdir(new_path)
            shutil.copyfile(old_path, new_path+'/'+image)
dataset

train_folder = "/home/jovyan/dataset1/dataset2/train"
test_folder = "/home/jovyan/dataset1/dataset2/test"
image = "/home/jovyan/dataset1/dataset2/test/001.ALI_HD/001 (14).jpg"
image = utils.load_img(image)
image.size
(74, 122)
# data_folder = "/home/jovyan/work/ear/raw"
# create_new_dataset(data_folder)
# train_datagen = "/home/jovyan/dataset2/dataset2"
# splitfolders.ratio(train_datagen, output="/home/jovyan/dataset2/dirty_set", seed=1337, ratio=(.7, 0.3))
# train_folder = "/home/jovyan/dataset2/train"
# test_folder = "/home/jovyan/dataset2/test"
train_datagen = ImageDataGenerator(rescale = 1./255)#, shear_range = 0.2, zoom_range = 0.2, horizontal_flip
= True)
test_datagen = ImageDataGenerator(rescale = 1./255)


training_set = train_datagen.flow_from_directory(train_folder,
                                                 target_size = IMAGE_SIZE,
                                                 batch_size = 32,
                                                 class_mode = 'categorical',
                                                 color_mode='grayscale')

test_set = test_datagen.flow_from_directory(test_folder,
                                            target_size = IMAGE_SIZE,
                                            batch_size = 16,
                                            class_mode = 'categorical',
                                            color_mode='grayscale')

model

def model(in_1=32, in_2=32, in_3=32, in_4=64):
```

```
    model = Sequential()
    model.add(Conv2D(filters=in_1,kernel_size=3, strides=1, input_shape=(64,64,1), padding='same', activation
= 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=in_2,kernel_size=3, strides=1, padding = 'same', activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=in_3,kernel_size=3, strides=1, padding ='same', activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=in_4,kernel_size=3, strides=1, padding ='same', activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(units = 512 , activation = 'tanh')) # try delete
    model.add(Dense(units = 164 , activation = 'softmax'))
    model.compile( optimizer='Adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
    return model
summary_example = model()
summary_example.summary()
def model_and_best_hyperparameters(nums):
    start_time1 = time.time()
    for i in nums:
        start_time2 = time.time()
        ready_model = model(i,i,i,i)        # change order
        trained_model = ready_model.fit(training_set, steps_per_epoch = 32, epochs = EPOCHS, validation_data
= test_set, validation_steps=16)
        print("for one training time taken: {:.2f} seconds".format(time.time() - start_time2)) # how much time one
training take

        SAVED_MODEL_BEST_ACCURACY_RESULTS.append(trained_model.history['accuracy'])
SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS.append(trained_model.history['val_accuracy'])
    return time.time() - start_time1 # how much time all training take
all_time = model_and_best_hyperparameters([32])
print("sorted list of max  epochs' values for accuracy  ",
show_best_res(SAVED_MODEL_BEST_ACCURACY_RESULTS))
print("sorted list of max  epochs' values for val_accuracy  ",
show_best_res(SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS))
print('all training take {:.2f} minutes'.format(all_time/60))
# save_lists()
```

```
def final_model():
    ready_model = model(64,64,64,64)
    call_back = ModelCheckpoint(f"/home/jovyan/weights.hdf5", monitor='val_accuracy', verbose=0,
save_best_only=True, save_weights_only=False, mode='auto')
    trained_model = ready_model.fit(training_set, steps_per_epoch = 32 , epochs = EPOCHS, #len(training_set)
                                    validation_data = test_set, validation_steps=16 , callbacks = [call_back]) #len(test_set)
SAVED_MODEL_BEST_ACCURACY_RESULTS2.append(trained_model.history['accuracy'])
SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS2.append(trained_model.history['val_accuracy'])
    return trained_model, ready_model
history, trained_model = final_model()
if trained_model:
    print("sorted list of max  epochs' values for accuracy  ",
show_best_res(SAVED_MODEL_BEST_ACCURACY_RESULTS2))
    print("sorted list of max  epochs' values for val_accuracy  ",
show_best_res(SAVED_MODEL_BEST_VAL_ACCURACY_RESULTS2))
    plot(history.history['accuracy'], history.history['loss'], history.history['val_accuracy'], history.history['val_loss'])
predict_model = Sequential()
ready_model2 = model(32,64,64,32)
for layer in ready_model2.layers[:-1]:
    predict_model.add(layer)
predict_model.add(Dense(units=1, activation=None))
predict_model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
predict_model.summary()
predict_model.load_weights('weights.hdf5', skip_mismatch=True, by_name=True)
def distance_metric(x1, x2):
    return tf.sqrt(tf.reduce_sum(tf.square(x1 - x2), axis=1))
def predict(model, known_data_path, unknown_data_path):
    known_data = get_image_and_transform(known_data_path)
    unknown_data = get_image_and_transform(unknown_data_path)
    known_data = model.predict(known_data)
    unknown_data = model.predict(unknown_data)
    distance = distance_metric(unknown_data, known_data)
    threshold = 1
    return  bool(distance < threshold), float(distance[0])
predict(predict_model, '/home/jovyan/22.jpg', '/home/jovyan/11.jpg')

choose best threshold

def return_lists_of_classes_and_images():
    test_folder = "/home/jovyan/dataset1/dataset2/test"
    classes, examples1 = get_examples_and_classes(test_folder,1) # every first image
```

```
    classes, examples2 = get_examples_and_classes(test_folder,2) #every second image
    examples3 = examples2[:len(examples2)//2]
    examples3 += examples2[len(examples2)//2::-1] # second part of list is reversed second part of examples2
list
    colors = ['blue' for i in range(len(examples2)//2)] + ['red' for i in range(len(examples2)//2)]
    del examples2
    return classes, examples1, examples3, colors

def find_threshold():
    classes, examples1, examples3, colors = return_lists_of_classes_and_images()
    y_predict_tensors = []
    y_predict_labels = []
    for i in range(len(examples1)):  # try build histogram
        predict_label, predict_tensor = predict(predict_model, examples1[i], examples3[i])
        y_predict_tensors.append(predict_tensor) # distance list
        y_predict_labels.append(int(predict_label)) # bool list

    true_labels = [1 for i in range(len(examples3)//2)] + [0 for i in range(len(examples3)//2)]
    return y_predict_tensors, y_predict_labels, true_labels, colors
y_predict_tensors, y_predict_labels, true_labels, colors = find_threshold()
plt.scatter(y_predict_tensors, y_predict_labels, c=colors)
plt.show()
confusion_matrix = confusion_matrix(true_labels, y_predict_labels)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

ДОДАТОК Б

РЕАЛІЗАЦІЯ СНМ ЗАСОБАМИ PYTHON

```
import libraries

import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense,
Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
from tensorflow.keras import utils
import matplotlib.pyplot as plt
import tensorflow.keras.backend as K
import random
functions

def plot(accuracy, loss, val_accuracy, val_loss): # return schedule of accuracy and loss
    fig, ax1 = plt.subplots()
    ax1.set_xlabel('epoch')
    ax1.set_ylabel('total loss')
    ax1.plot(loss,color='tab:red')
    ax1.plot( val_loss, color='tab:brown')
    ax1.tick_params(axis='y')
    ax2 = ax1.twinx()
    ax2.set_ylabel('accuracy')
    ax2.plot( accuracy, color='tab:blue')
    ax2.plot( val_accuracy, color='tab:green')
    ax2.tick_params(axis='y')
    fig.tight_layout()
def get_image_and_transform(path, dims=False, mode='grayscale'):

    data = utils.load_img(path, color_mode=mode, target_size=(64,64))
    data = np.array(data)
    if dims:
        data = np.expand_dims(data, axis=0)
```

```
    data = data / 255.0
    return data
def contrastive_loss(y, preds, margin=0.01):
        # explicitly cast the true class label data type to the predicted
        # class label data type (otherwise we run the risk of having two
        # separate data types, causing TensorFlow to error out)
        y = tf.cast(y, preds.dtype)
        # calculate the contrastive loss between the true labels and the predicted labels
        squaredPreds = K.square(preds)
        squaredMargin = K.square(K.maximum(margin - preds, 0))
        loss = K.mean(y * squaredPreds + (1 - y) * squaredMargin)
        # return the computed contrastive loss to the calling function
        return loss
def build_dataset(dataset_dir, label, dims=False, mode='grayscale'): # build 2 list with array for training and 1 list
with labels
    x_train1, x_train2, y_train = [], [], []
    classes = sorted(os.listdir(dataset_dir))
    for i in range(len(classes)):
        class_dir = os.path.join(dataset_dir, classes[i])
        images = os.listdir(class_dir)
        if len(images) == 1:
            continue
        if len(images) % 2 != 0:
            del images[len(images) - 1]
        for image in range(len(images)):
            full_image_path = os.path.join(class_dir, images[image])
            if 'ipynb_checkpoints' in full_image_path:
                continue
            if image < len(images)//2:
                y_train.append(label)
                x_train1.append(get_image_and_transform(full_image_path, dims, mode))
            else:
                x_train2.append(get_image_and_transform(full_image_path, dims, mode))
    return x_train1, x_train2, y_train
# train_data = "/home/jovyan/dataset1/dataset2/train"
# train_wrong_data = "/home/jovyan/dataset1/dataset2/test"

# train_data_1, train_data_2, train_labels1 = build_dataset(train_data, 1)
# # train_data_3, train_data_4, train_labels2 = build_dataset(train_wrong_data, 0)

# print(len(train_data_1), len(train_data_2), len(train_labels1))
```

```
# # print(len(train_data_3), len(train_data_4), len(train_labels2))
# # random.shuffle(train_data_3)
# # random.shuffle(train_data_4)

# # train_data_1 = train_data_1 + train_data_3
# # train_data_2 = train_data_2 + train_data_4
# # train_labels1 = train_labels1 + train_labels2

# train_data_1 = np.array(train_data_1)
# train_data_2 = np.array(train_data_2)
# train_labels1 = np.array(train_labels1)

# print(
#     len(train_data_1),
#     len(train_data_2),
#     len(train_labels1)
#     )
def build_siamese_dataset(train_folder, mode='grayscale'):
    train_datagen = ImageDataGenerator(rescale=1./255)
    training_set = train_datagen.flow_from_directory(train_folder,
                                                     target_size=(64,64),
                                                     batch_size=32,
                                                     class_mode='categorical',
                                                     color_mode=mode,
                                                     shuffle=True,
                                                     seed=42,
                                                     subset='training')
    x_train1, x_train2 = [], []
    for i in range(len(training_set)):
        images, labels = training_set[i]
        # images, labels = images[len(images)//2:], labels[len(labels)//2:]
        num = len(images) -1 if len(images) % 2 != 0 else len(images)
        for o in range(num):
            if o % 2 == 0:
                x_train1.append(images[o])
            else:
                x_train2.append(images[o])
    return x_train1, x_train2
x_train1, x_train2 = build_siamese_dataset("/home/jovyan/dataset1/dataset2/train")#, 'rgb')
x_train2 = x_train2[:len(x_train2)+1] + x_train2[:len(x_train2):-1]
train_labels1 = [1] * (len(x_train2)//2) + [0] * (len(x_train2)//2) + [0]
```

```
x_train1 = np.array(x_train1)
x_train2 = np.array(x_train2)
train_labels1 = np.array(train_labels1)
x_test1, x_test2 = build_siamese_dataset("/home/jovyan/dataset1/dataset2/test")#, 'rgb')
x_test2 = x_test2[:len(x_test2)+1] + x_test2[:len(x_test2):-1]
test_labels1 = [1] * (len(x_test2)//2) + [0] * (len(x_test2)//2) + [0]

x_test1 = np.array(x_test1)
x_test2 = np.array(x_test2)
test_labels1 = np.array(test_labels1)

first implementation of siamese CNN
def siamese_model(in_1=32, in_2=32, in_3=32, in_4=64):
    input_shape = (64,64, 1)

    input_1 = Input(shape=input_shape)
    input_2 = Input(shape=input_shape)

    shared_layers = [
        Conv2D(filters=in_1, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_2, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_3, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=in_4, kernel_size=3, strides=1, padding='same', activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Flatten(),
        Dropout(0.1)
    ]

    fc_layers = [
        Dense(units=512, activation='tanh'),
        Dense(units=1, activation='sigmoid')
    ]

    output_1 = input_1
```

```
    output_2 = input_2
    for layer in shared_layers:
        output_1 = layer(output_1)
        output_2 = layer(output_2)

    concatenated = tf.keras.layers.Concatenate()([output_1, output_2])
    for layer in fc_layers:
        concatenated = layer(concatenated)

    siamese_model = Model(inputs=[input_1, input_2], outputs=concatenated)
    siamese_model.compile(optimizer=Adam(), loss=contrastive_loss, metrics=['accuracy'])

    return siamese_model
model1 = siamese_model()
model1.summary()
history = model1.fit([x_train1, x_train2],
                     train_labels1,
                     steps_per_epoch = 180,
                     epochs=15,
                     validation_data = ([x_test1,x_test2], test_labels1),
                     validation_steps=60,
                     shuffle=True)
plot(history.history['accuracy'], history.history['loss'],history.history['val_accuracy'],history.history['val_loss'])
def predict_siamese_model(siamese_model, data_1, data_2):
    predictions = siamese_model.predict([data_1, data_2])
    return predictions, float(predictions) > 0.5
img1 = get_image_and_transform('/home/jovyan/21.jpg', True)
img2 = get_image_and_transform('/home/jovyan/11.jpg', True)
predict_siamese_model(model1,img1, img2)

second implementation of siamese CNN

def siamese_model2(in_1=32, in_2=32, in_3=32, in_4=64):
    input_shape = (64,64, 3)
    input_1 = Input(shape=input_shape)
    input_2 = Input(shape=input_shape)
    base_model = tf.keras.applications.MobileNetV3Small(weights='imagenet', include_top=False,
input_shape=input_shape)
    fc_layers = [
        Dense(units=512, activation='tanh'),
        Dense(units=1, activation='sigmoid')
```

```
    ]
    output_1 = base_model(input_1)
    output_2 = base_model(input_2)
    flattened_1 = Flatten()(output_1)
    flattened_2 = Flatten()(output_2)
    concatenated = tf.keras.layers.Concatenate()([flattened_1, flattened_2])
    for layer in fc_layers:
        concatenated = layer(concatenated)
    siamese_model = Model(inputs=[input_1, input_2], outputs=concatenated)
    siamese_model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return siamese_model
model2 = siamese_model2()
model2.summary()
history = model2.fit([x_train1, x_train2],
                     train_labels1,
                     steps_per_epoch = 32,
                     epochs=60,
                     validation_data = ([x_test1,x_test2], test_labels1),
                     validation_steps=60,
                     shuffle=True)
plot(history.history['accuracy'], history.history['loss'],history.history['val_accuracy'],history.history['val_loss'])
def predict_siamese_model(siamese_model, data_1, data_2):
    predictions = siamese_model.predict([data_1, data_2])
    return predictions, float(predictions) > 0.5

img1 = get_image_and_transform('/home/jovyan/11.jpg', True, 'rgb')
img2 = get_image_and_transform('/home/jovyan/22.jpg', True, 'rgb')

predict_siamese_model(model2,img1, img2)
```