НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки6.050103 “ Програмна інженерія “

на тему: Web-система прискореного пошуку зображень з використанням нейронних мереж

Виконав: студент 4 курсу, групи ТВ-51

Здор Костянтин Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., Шалденко О.В _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050103 “ Програмна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту
_____Здор Костянтин Андрійович_____
(прізвище, ім’я, по батькові)

1.Тема роботи Web-система прискореного пошуку зображень з використанням нейронних мереж
керівник роботи Шалденко Олексій Вікторович, к.т.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 2019р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: Web-система прискореного пошуку зображень з використанням нейронних мереж

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)проаналізувати задачу створення web-систему по прискореному пошуку зображень з використанням нейронних мереж на прикладі завантажуємого зображення та знаходження відповідного в базі данних.

5. Перелік ілюстративного матеріалу: схеми архітектури додатку, знімки екранних форм, діаграма прецедентів системи, діаграма класів, діаграма структури системи, зразки розробленого інтерфейсу додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”6”_листопада_____2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	1 Захист програмного продукту		
7.	2 Передзахист		
8.	Захист		

Студент _____ Здор К.А._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Шалденко О. В.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці web-системи прискореного пошуку зображень за допомогою нейронних мереж на прикладі завантаженого зображення та знаходження відповідного в базі даних.

Метою роботи є створення системи для прискореного пошуку зображень в базі даних на основі інтернет-магазину.

Для досягнення мети були вирішені наступні задачі:

- Проаналізовано методи обробки зображень.
- Спроектована та реалізована система пришвидшеного пошуку зображень.
- Проведено попереднє завантаження зображення на сервер.
- Розробка нейронної мережі для кодування зображень.
- Збереження та оптимізація нейронної мережі для використання у Web-системі.
- Використання математичних алгоритмів для порівняння закодованих зображень.
- Пошук відповідного зображення в базі даних та відображення результатів.

Ключові слова: сіамські нейронні мережі, розпізнавання зображень, класифікація.

Обсяг звіту становить 83 сторінок, міститься 27 ілюстрацій, 3 додатки. Загалом опрацьовано 18 джерел.

ABSTRACT

This thesis is devoted to the development of a web-system for the accelerated image search by neural network for searching related image in the database.

The purpose of the work is to search image in the database based on internet shop.

To achieve the goal, the following tasks were solved:

1. Preloaded image on the server.
2. Development of the neural network for image encoding.
3. Neural network saving and optimization for Web-system using.
4. Encoded images comparison by mathematical algorithms.
5. Related image search in the database and result visualization.

Keywords: Siamese Neural Network, image recognizing, classification.

Scope of the report is 83 pages contain 27 illustrations, 3 annexes. Generally 18 sources have been processed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NN	–	Neural network
DFD	–	Data Flow Diagram
SNN	–	Siamese Neural Network
CNN	–	Convolutional Neural Network
DB	–	Data Base
ED	–	Euclidean Distance
UI	–	User interface

ВСТУП

Останні десятиріччя технологія комп’ютерного зору набуває все більшої популярності та вдосконалюється з часом. Перші кроки у цій сфері були зроблені у 50-х роках 20 сторіччя. Однак, саме в останній час комп’ютерний зір почав стрімко розвиватися. Це відбувається завдяки розвитку комп’ютерної техніки, що веде за собою збільшення обчислювальних потужностей, а також розвитку штучного інтелекту, що дозволяє поліпшити і прискорити алгоритми аналізу зображень.

Цифрова обробка зображень та їх аналіз зосереджені в основному на роботі з покращенням зображень, у той час як комп’ютерний зір дозволяє отримати атрибути з аналізу зображень.

Одним з актуальних завдань в комп'ютерному зорі є завдання порівняння зображень та знаходження зображень одного типу. Значна частина цієї області присвячена практичному застосуванню цих методів, такі як “SIFT”, “SURF”, “BRIEF” та SNN.

Застосування комп’ютерного зору є дуже різноманітним, воно охоплює системи безпеки, контроль виготовлення об’єктів, системи візуального контролю та управління, контроль транспортних засобів. В товарно-орієнтованій промисловості системи машинного зору використовуються для пошуку схожих зображень та їх класифікації.

Успішне застосування машинного зору на потребує багато знань і навичок в різних областях комп’ютерних наук, таких як бази даних, мережеві системи та інше.

Задачі пошуку зображень можуть розглядатися на прикладі інтернет магазину який дозволяє по фотографії знаходити відповідний товар в інтернет магазині. Для отримання цих результатів була розроблена web-система, яка дозволяє вирішити поставлені задачі.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ WEB-СИСТЕМИ ПРИСКОРЕНОГО ПОШУКУ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ

Метою даної дипломної роботи є створення web-системи прискореного пошуку зображень за допомогою нейронних мереж для покушу відповідного зображення в базі даних по зображенню користувача.

При розробленні відповідного забезпечення потрібно розв’язати наступні завдання:

1. Провести попереднє завантаження зображення на сервер.
2. Створити та навчити нейрону мережу.
3. Зберегти та оптимізувати нейрону мережу для використання її в web-системі.
4. Підготовка зображення для загрузки його в нейронну мережу.
5. Використання нейронної мережі для кодування зображення.
6. Пошук відповідного, закодованого зображення у базі даних.
7. Вивести зображення та інформацію яка відноситься до цього зображення та зберігається в базі даних.

Для завантаження зображення на сервер спочатку було отримано абсолютне посилання до web-додатку, потім визначений шлях до папки, в якій буде збережено файл, створено папку, за умови що її не існує. Після цього було реалізовано запис файлу у вибраний каталог за допомогою методу handle_uploaded_file, що представляє частину або форму елементу, що був отриманий у запиті POST з множинними частинами або формами даних.

Для розробки нейронної мережі був вибраний фреймворк TensorFlow. Створення та тренування мережі проводилося на Google Colab. Нейронна мережа має архітектуру SNN з Triplet loss[14].

Для збереження та оптимізації було використано можливість TensorFlow обрізати частину графу та зберегти його значення як константи після чого результат був збережений у бінарний серіалізований файл.

За допомогою методів PIL та NumPy проводиться аналіз зображення та перетворення його до потрібного вигляду. Після цього класс NNManager кодує зображення.

Скориставшись классом ProductManager було отримано закодовані значення з бази даних та проведено пошук по ним. Як результат повертається зображення та інформація яка закріплена за цим зображенням.

За допомогою класу HistoryManager отримані результати були записані у базу даних (MongoDB).

Оброблені дані були повертаються користувачеві на UI.

2 АНАЛІЗ МЕТОДІВ ПОРІВНЯННЯ ЗОБРАЖЕНЬ

Для процесів порівняння зображень були протестовані наступні методи, основна ідея яких виділення признаків.

В машинному навчанні, розпізнаванні образів та в обробці зображень виокремлення ознак починається з первинного набору даних вимірювань, і будує похідні значення, покликані бути інформативними та ненадлишковими, полегшувати наступні кроки навчання та узагальнення, і в деяких випадках вести до кращого тлумачення людьми. Виділення ознак пов'язане зі зниженням розмірності. Коли вхідні дані алгоритму є занадто великими, щоби їх можливо було обробити, і підозрюються на надлишковість, тоді їх може бути перетворено на скорочений набір ознак. Цей процес називається виділянням ознак. Очікується, що виділені ознаки містять доречну інформацію з вхідних даних, так що бажане завдання може бути виконано із застосуванням цього скороченого представлення замість повних первинних даних.

У різних галузях часто виникає потреба отримання скороченого представлення даних. Наприклад коли потрібно зберігати великі обсяги даних та порівнювати їх.

На даний момент не існує універсальних алгоритмів порівняння зображень, тому необхідно створити спеціальні алгоритми, параметри яких підбираються користувачем, виходячи з конкретних умов.

2.1 Метод «SIFT»

Метод SIFT означає Scale-Invarant Feature Transform[5] і вперше був представлений у 2004 році Д. Лоу, Університет Британської Колумбії. SIFT - інваріантність до масштабу зображення і обертання. Цей алгоритм запатентований, тому його включено в модуль Non-free в OpenCV.

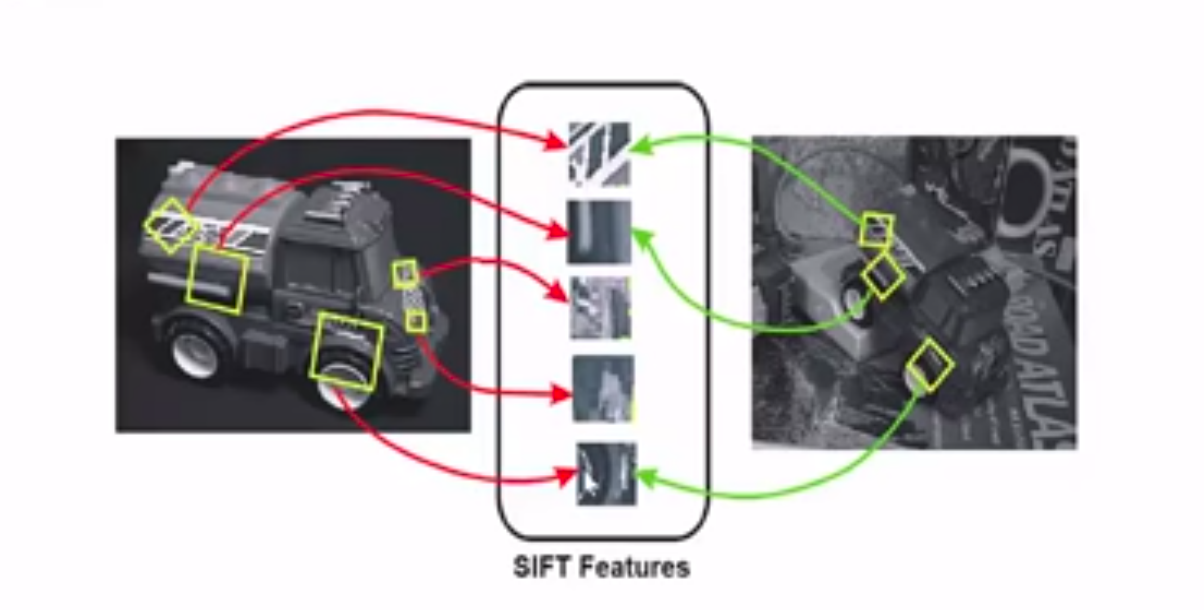

Рисунок 1.1 –Приклад роботи методу «SIFT»

Основними перевагами SIFT є:

- Місцевість: особливості локальні, стійкі до перешкод.
- Відмінність: індивідуальні особливості можуть бути узгоджені з великою базою даних об'єктів.
- Чисельність: багато функцій можуть генеруватися навіть для невеликих об'єктів.
- Ефективність: близька до продуктивності в реальному часі.
- Розширюваність: легко розширюється на широкий діапазон різних типів функцій, при цьому кожна додає надійність.

Простір масштабу зображення - це функція L (x, y, σ), яка виробляється зі згортки гауссового ядра на різних масштабах з вхідним зображенням[1]. Масштаб-простір розділяється на октави, а кількість октав і масштабу залежить від розміру вихідного зображення. Таким чином, ми генеруємо кілька октав вихідного зображення. Розмір зображення кожної октави становить половину попереднього.

Розмиті зображення використовуються для створення іншого набору зображень - Difference of Gaussians (DoG)[6]. Ці DoG зображення чудово підходять для пошуку цікавих ключових точок на зображенні. Ці точки виконується для різних октав зображення в Гауссовій піраміді.

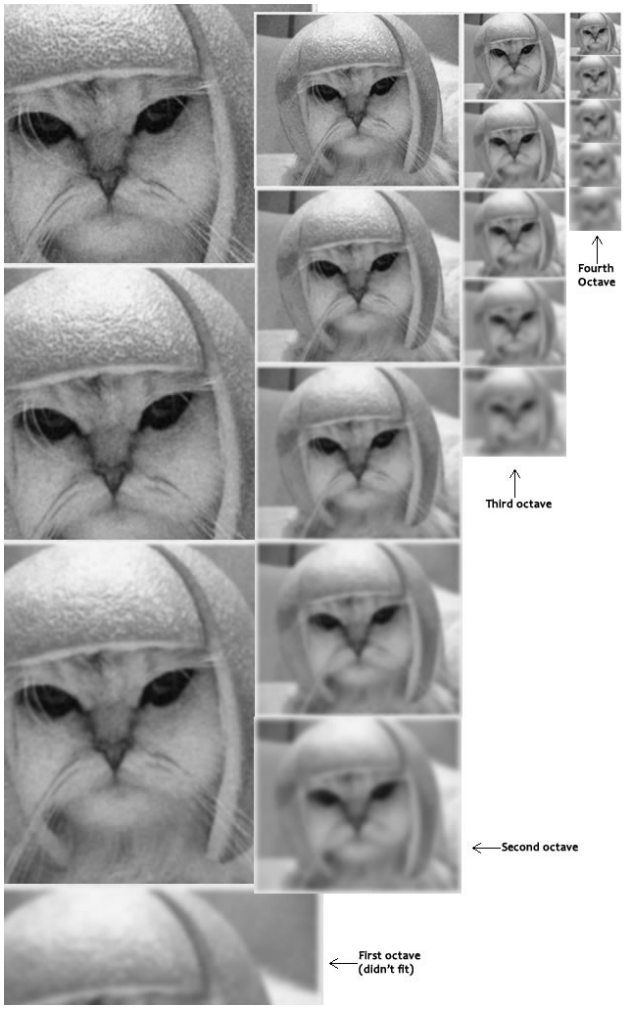

Рисунок 1.2 –Приклад масштабування та розмиття зображення

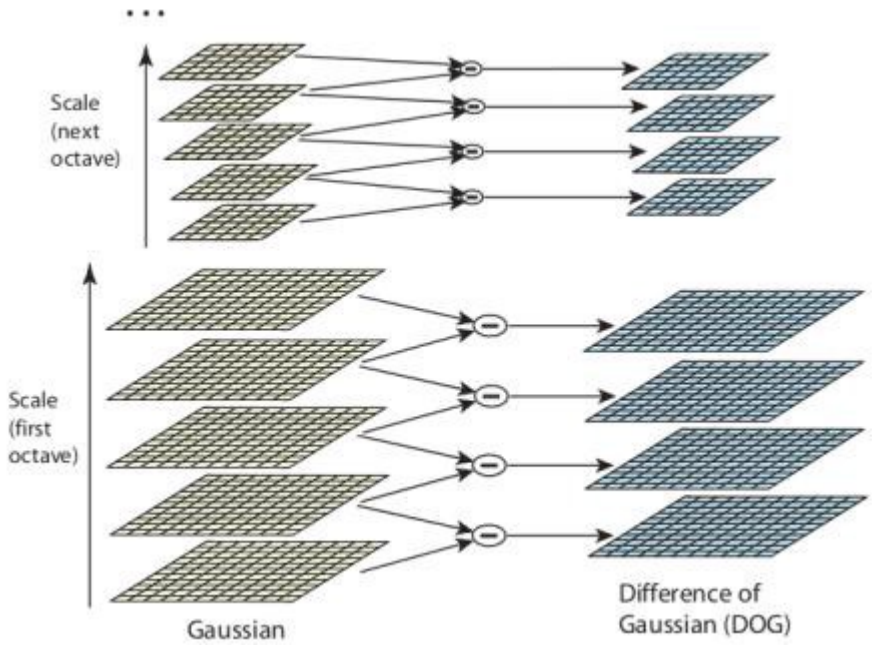

Рисунок 1.3 –Приклад Гаусової піраміди

Потім вони використовуються для обчислення лапласиану гаусових наближень, які є інваріантними за шкалою. Один піксель зображення порівнюється з 8 сусідів, а також 9 пікселів у наступній шкалі і 9 пікселів у попередніх масштабах. Таким чином, проводиться 26 перевірок. Якщо це локальні екстремуми, це потенційна ключова точка. В основному це означає, що ключова точка краще представлена в такому масштабі.

Ключові точки, сформовані на попередньому кроці, створюють багато ключових точок. Деякі з них лежать уздовж краю, або вони не мають достатньої контрастності. В обох випадках вони не настільки корисні, як ознаки. Тому ми позбавляємося від них. Підхід подібний до того, який використовується в кутовому детекторі Harris для усунення особливостей краю. Для функцій з низькою контрастністю ми просто перевіряємо їх інтенсивність.

Тепер у нас є легітимні ключові точки. Вони були перевірені, щоб бути стабільними. Ми вже знаємо масштаб, в якому було виявлено ключову точку (це те саме, що масштаб розмитого зображення). Тому ми маємо масштабну інваріантність. Околиці взяті навколо розташування ключових точок залежно від масштабу, і величина і напрямок градієнта обчислюються в цьому регіоні. Створюється гістограма орієнтації з 36 бункерами, що охоплюють 360 градусів. Як тільки ви зробите це для всіх пікселів навколо ключової точки, гістограма матиме пік в певний момент. Найвищий пік в гістограмі та будь-який пік вище 80% також враховується для обчислення орієнтації. Він створює ключові точки з однаковим розташуванням і масштабом, але різними напрямками. Це сприяє стабільності відповідності.

У цей момент кожна ключова точка має розташування, масштаб, орієнтацію. Далі слід обчислити дескриптор для локальної області зображення про кожну ключову точку, яка є надзвичайно відмінною та інваріантною, наскільки це можливо до змін, таких як зміни точки зору та освітлення.

Ключові точки між двома зображеннями узгоджуються, ідентифікуючи їх найближчих сусідів.

2.2 Метод SURF

Метод SURF (Speeded Up Robust Features) - це швидкий і надійний алгоритм локального інваріантного подання та порівняння зображень. Головний інтерес підходу SURF полягає в його швидкому обчисленні операторів, що використовують фільтри паралельно, що дозволяє в реальному часі відстежувати та розпізнавати об'єкти.

Таблиця інтегрального зображення або підсумкова область була введена в 1984 році[4]. Інтегральне зображення використовується як швидкий та ефективний спосіб обчислення суми значень у даному зображенні - або прямокутної підмножини сітки. Він також використовується для розрахунку середньої інтенсивності в межах даного зображення.

Це дозволяє швидко обчислювати згорткові фільтри. Вступ інтегрального зображення $I_{\Sigma}(x)$ в місці x = (x, y) ^{T} являє собою суму всіх пікселів у вхідному зображенні I в прямокутній області, утвореної походженням та x.

$$I_{\Sigma}(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \tag{1}$$

При розрахунку $I_{\Sigma}(x)$ потрібно лише чотири доповнення для обчислення суми інтенсивностей над будь-якою прямокутною областю, незалежно від її розміру[9].

Surf використовує матрицю Гессена через його хорошу продуктивність у часі обчислення та точності. Замість того, щоб використовувати іншу міру для вибору місця та масштабу, наприклад детектора Гессе-Лапласа[2], Surf спирається на детермінант матриці Гессе. З огляду на піксель, гессіан цього пікселя має такий вигляд:

$$H(f(x,y)) = \begin{bmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta x \delta y} & \frac{\delta^2 f}{\delta y^2} \end{bmatrix} \tag{2}$$

Для адаптації до будь-якого масштабу зображення фільтруються гаусовим ядром, тому з урахуванням точки X = (x, y) матриця Гессена H(x,σ) в точці x при шкалі σ визначається як:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \quad (3)$$

де $L_{xx}(x, \sigma)$ - згортка гаусової похідної другого порядку з зображенням *I* в точці x, і аналогічно для $L_{xy}(x, \sigma)$ і $L_{yy}(x, \sigma)$. Гаусіани є оптимальними для аналізу масштабу, але на практиці їх необхідно дискретизувати і обрізати. Це призводить до втрати повторюваності під обертаннями зображення навколо непарних кратних значень π / 4. Ця слабкість дотримується в цілому для детекторів на основі Гессе. Тим не менш, детектори все ще працюють добре, і незначне зниження продуктивності не переважає перевагу швидких згортків, принесених дискретизацією і обрізанням.

Для обчислення детермінанта матриці Гессе спочатку необхідно застосувати згортку з гаусовим ядром, потім похідною другого порядку. Після успіху Лоу з наближеннями LoG (SIFT), SURF висуває апроксимацію (як згортку, так і похідну другого порядку) ще далі за допомогою віконних фільтрів. Ці приблизні гаусівські похідні другого порядку можуть бути оцінені за дуже низькою обчислювальною вартістю за допомогою інтегральних зображень і незалежно від розміру, і це є частиною причини швидкого SURF.

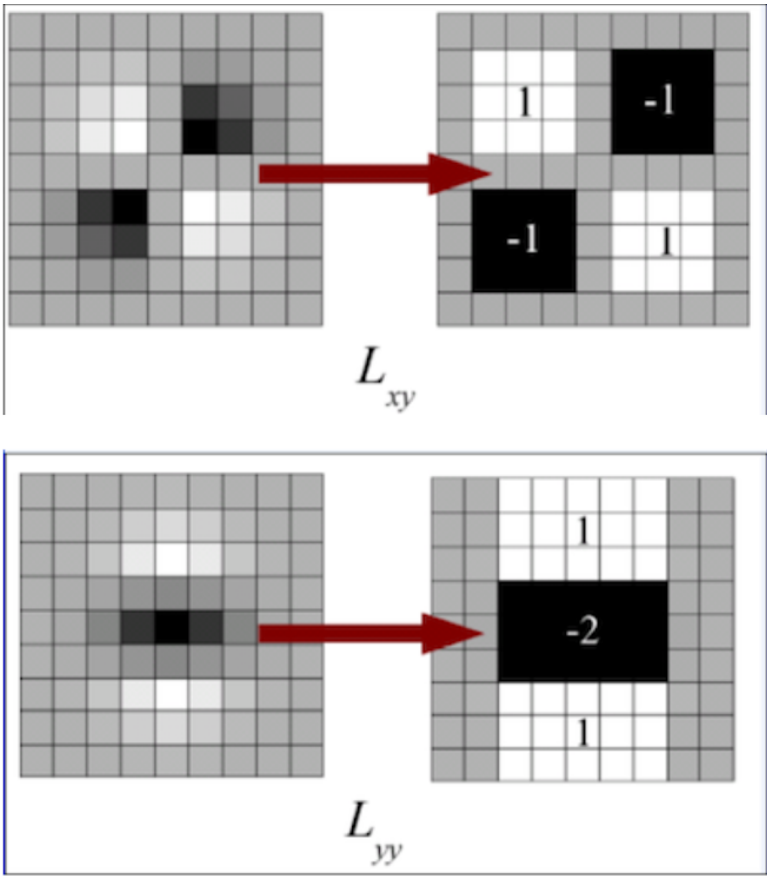

Рисунок 2.1 –Приклад гаусівської частинної похідної

Фільтри 9 × 9 у вищенаведених зображеннях є наближеннями для гауссових похідних другого порядку з σ = 1.2. Позначимо ці наближення за Dxx, Dyy і Dxy. Тепер ми можемо представляти детермінант гессенського (наближеного) як:

$$det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2 \quad (4)$$

Масштабні розміри зазвичай реалізуються як зображення пірамід. Зображення неодноразово згладжуються гаусовими і згодом суб-вибірковими функціями для того, щоб досягти більш високого рівня піраміди. Завдяки використанню віконних фільтрів і інтегральних зображень, SURF не повинен ітераційно застосовувати один і той же фільтр до виходу попередньо відфільтрованого шару, а замість цього може застосовувати такі фільтри будь-якого розміру на однаковій швидкості безпосередньо на вихідному зображенні навіть паралельно. Таким чином, простір масштабу аналізується шляхом збільшення масштабу фільтра (9 × 9 → 15 × 15 → 21 × 21 → 27 × 27 і т.д.), а не через ітеративне зменшення розміру зображення. Для кожної нової

октави збільшення розміру фільтра подвоюється одночасно, інтервали вибірки для вилучення точок інтересу (σ) також можна подвоїти, що дозволяє підвищувати масштаб фільтра при постійних витратах. Для локалізації точок інтересу в зображенні і по масштабах застосовується nonmaximum suppression в розмірі 3 × 3 × 3.

Створення дескриптора SURF відбувається у два етапи. Перший крок складається з фіксації відтворюваної орієнтації на основі інформації з кругової області навколо ключової точки. Потім будується квадратна область, вирівняна до обраної орієнтації, і витягується з неї дескриптор SURF.

Для того, щоб бути інваріантним до обертання, surf намагається визначити відтворювальну орієнтацію для точок інтересу.

Спочатку потрібно побудувати квадратну область, центровану навколо ключової точки, і орієнтовану вздовж орієнтації.

Потім область ділиться регулярно на менші квадратні субрегіони. Для кожної субрегіону обчислюється кілька простих функцій на регулярно розташованих точках вибірки. З метою спрощення, ми називаємо dx вейвлет-відповідь Хаара в горизонтальному напрямку і dy вейвлет-відповідь Хаара у вертикальному напрямку. Щоб підвищити стійкість до геометричних деформацій і помилок локалізації, відповіді dx і dy спочатку зважують за допомогою гауссової функції з центром у ключовій точці.

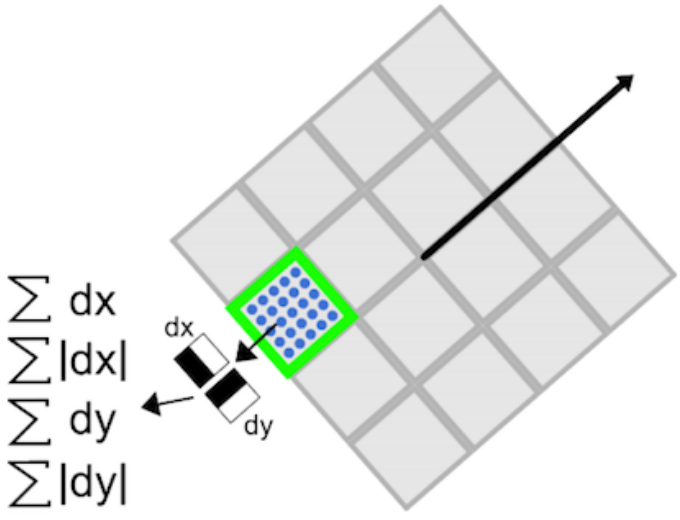

Рисунок 2.2 –Приклад роботи дескриптора

Потім вейвлет-відповіді dx і dy підсумовуються по кожному субрегіону і утворюють перший набір записів до вектора ознак. Для того, щоб ввести інформацію про полярність змін інтенсивності, витягується сума абсолютних значень відповідей, | dx | і | dy |. Отже, кожен субрегіон має чотиривимірний вектор дескриптора v для своєї основної структури інтенсивності $V = (\Sigma\, dx, y, dy, d\,|dx|, d\,|dy|)$. Це призводить до дескрипторного вектора для всіх субрегіонів.

2.3 Метод BRIEF

BRIEF дуже швидкий, як для збірки, так і при роботі[3]. BRIEF легко перевершує інші швидкі дескриптори, такі як SURF і SIFT з точки зору швидкості розпізнавання.

Після виявлення ключової точки ми продовжуємо обчислювати дескриптор для кожної з них. Дескриптори функцій кодують інформацію в ряд цифр і виступають як свого роду числовий «відбиток», який можна використовувати для диференціації однієї функції від іншої. Визначене сусідство навколо пікселя відоме як патч, який є квадратом ширини і висоти пікселя.

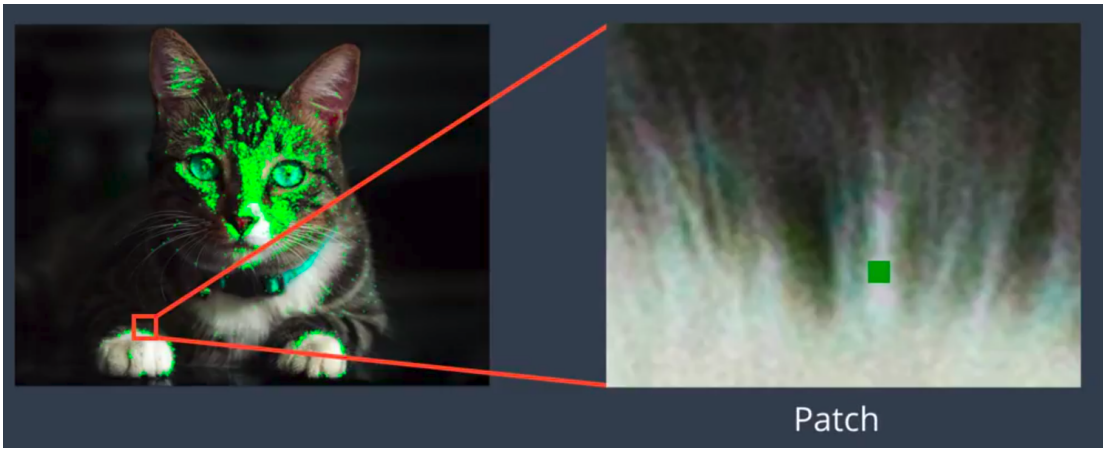

Рисунок 3.1 –Приклад патчу

Зразки зображень можуть бути ефективно класифіковані на основі порівняно невеликого числа парних порівнянь інтенсивності. Короткі перетворення патчів

зображень у бінарний вектор ознак робляться таким чином, що вони разом можуть представляти об'єкт. Вектор бінарних ознак також відомий як дескриптор бінарних ознак містить лише 1 і 0. Кожна ключова точка описується вектором ознак, який є рядком 128–512 біт.

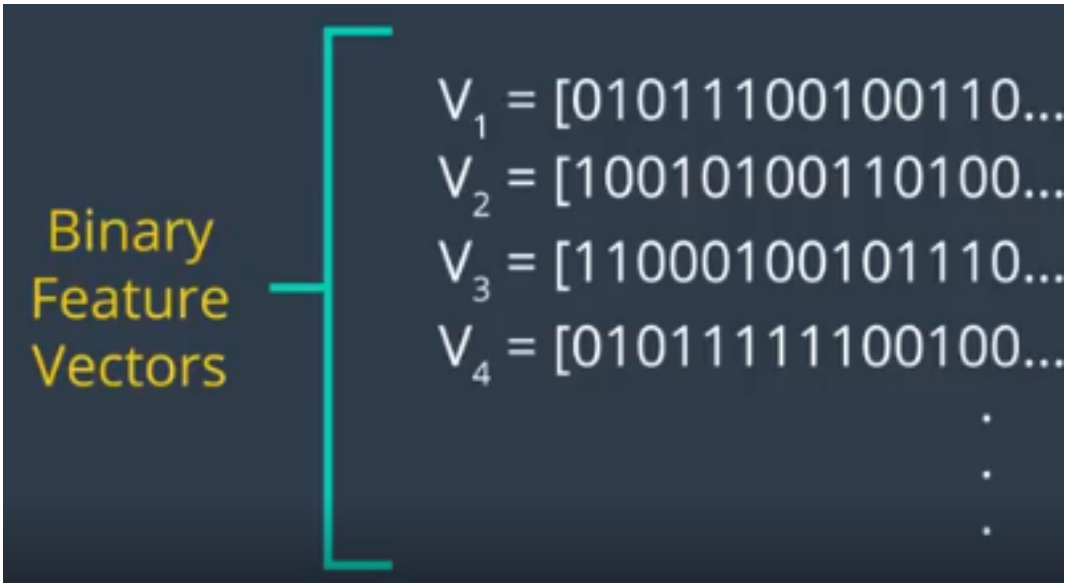

Рисунок 3.2 –Приклад векторів ознак

Метод BRIEF описує зображення на рівні пікселів, тому він дуже чутливий до шуму. Шляхом попереднього згладжування патча ця чутливість може бути зменшена, таким чином збільшуючи стабільність і повторюваність дескрипторів[15]. З тієї ж самої причини зображення необхідно згладжувати до того, як вони можуть бути змістовно диференційовані при пошуку країв.

Рисунок 3.3 –Приклад згладжування зображення

Далі потрібно створити векторний бінарний елемент з цього патча. Створюється бінарний вектор характеристик двійкових тестових (τ) відповідей. Двійковий тест τ визначається:

$$\tau(p;x,y)=\begin{cases}1 & :P(x)<P(y)\\ 0 & :p(x)\geq p(y)\end{cases} \quad (4)$$

В цій формулі p (x) - інтенсивність p в точці x. Вибір множини n (x, y) - локаційних пар однозначно визначає набір бінарних тестів. Де n - довжина вектора бінарних ознак і може бути 128, 256 і 512.

Генерування довжини n бітового вектора залишає безліч варіантів для вибору тестових місць ((x, y) пари). Пара (x, y) також називається випадковою парою, яка знаходиться всередині патча. Усього ми повинні вибрати n тест (випадкова пара) для створення вектора бінарних ознак, і ми повинні вибрати цей n тест з одного з п'яти підходів (Sampling Geometries)[13].

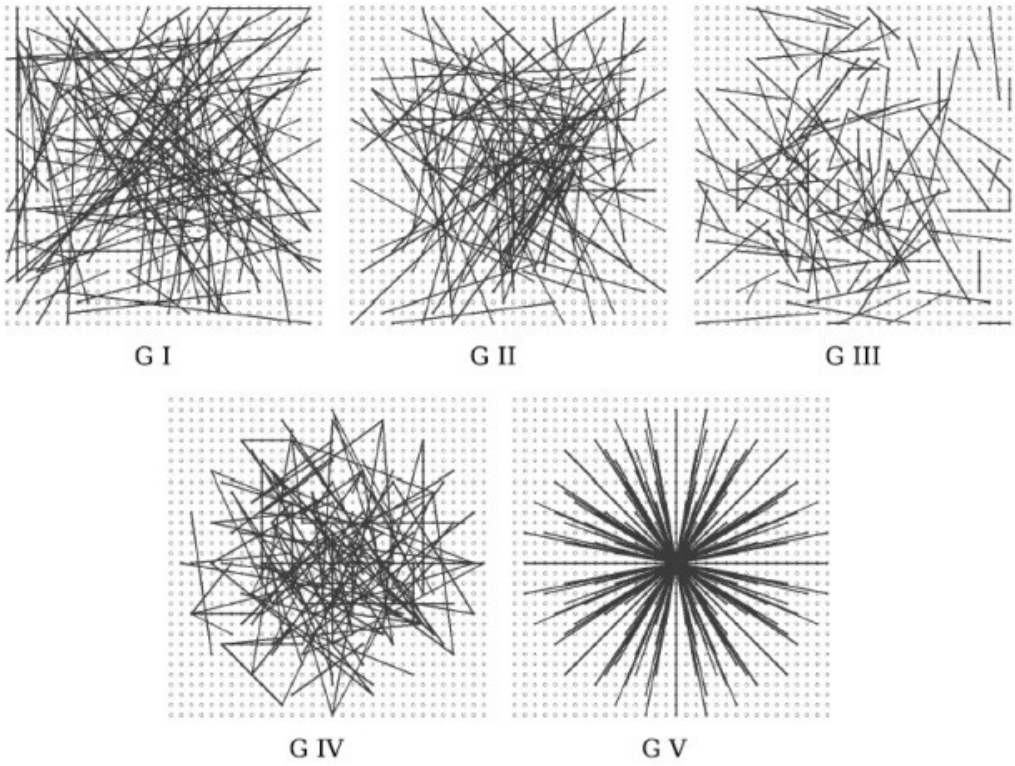

Рисунок 3.4 – Візуалізація п’яти підходів для знаходження пар

Підходи для знаходження пар:

1. Уніформа (G I): Обидва пікселя x та y у випадковій парі витягуються з Unifrom distribution або поширення S / 2 навколо ключової точки. Пара (тест) може лежати поблизу кордону.

2. Гауссова (G II): Пікселі x та y витягуються у випадковій парі з розподілу Гаусса або поширення 0,04 * S^2 навколо ключової точки.

3. Гауссова (G III): Перший піксель (x) у випадковій парі витягується з розподілу Гауса, центрованого навколо ключової точки з багатовимірним відхиленням або поширенням 0,04 * S^2. Другий піксель (y) у випадковій парі витягується з розподілу Гауса з центром навколо першого пікселя (x) зі стандартним відхиленням або поширенням 0,01 * S^2. Це змушує тест бути більш локальним. Тестові місця поза патчем притискаються до краю патчу.

4. Груба полярна сітка (G IV): Обидва пікселя x та y відбираються у випадковій парі з дискретних розташувань грубої полярної сітки, вводячи просторове квантування.

5. Груба полярна сітка (G V): Перший піксель (x) у випадковій парі знаходиться на (0, 0), а другий піксель (y) у випадковій парі витягується з дискретних розташувань грубої полярної сітки.

Дескриптор BRIEF виглядає так:

$$f_{n_d}(p) := \sum_{1 \le i \le n_d} 2^{i-1} \tau(p; x_i, y_i) \qquad (5)$$

2.4 Метод SNN

Нейронна мережа - це мережа нейронів, які використовуються для обробки інформації. Щоб створити їх, вчені подивилися на найсучаснішу машину обробки даних у той час - мозок. Наш мозок обробляє інформацію за допомогою мереж нейронів. Вони отримують вхід, обробляють його і відповідно виводять електричні

сигнали до нейронів, з якими він з'єднаний. Використовуючи біо-мімікрію, ми змогли застосувати архітектуру наших мізків для подальшого розвитку галузі штучного інтелекту[16].

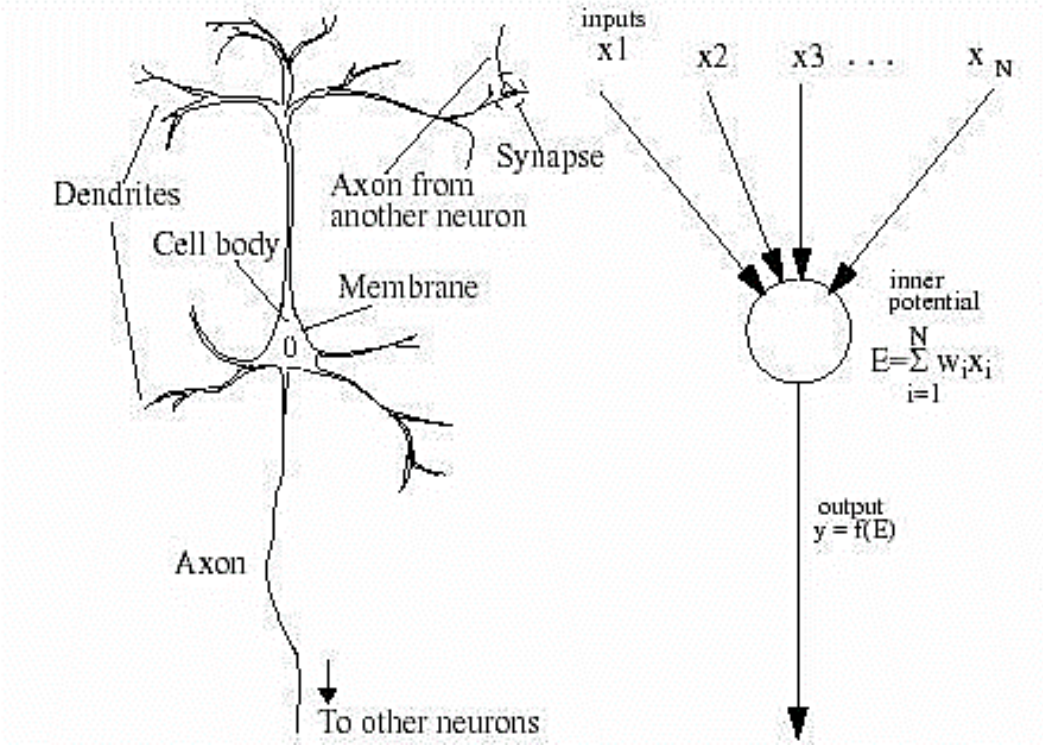

Рисунок 4.1 – Порівняння роботи нейронів мозку та штучних нейронів

Штучні нейронні мережі відтворюють структуру людських нейронів для обробки інформації, що призводить до набагато більш точних результатів, ніж раніше використовувані моделі регресії.

Нейронна мережа складається з 3 основних частин - вхідний шар, приховані шари, вихідний шар.

Вхідний шар це буквально шар, який вводить інформацію для нейронної мережі для обробки. Це може бути що завгодно. Це може бути квадратний метр вашого будинку для програми прогнозування ціни на будинок, або значення пікселя на екрані для програми комп'ютерного зору.

Приховані шари - це шари, які виконують всю обробку для нейронних мереж. Їх можете бути скільки завгодно. Чим більше шарів у вас є, тим точніше буде нейронна мережа. Кожен шар складається з вузлів, які імітують нейрони нашого

мозку. Ці вузли отримують інформацію з вузлів попереднього шару, множать її на вагу, а потім додають до неї зміщення.

Вихідний шар це шар, який просто об'єднує інформацію з останнього прихованого шару мережі для виведення всієї необхідної інформації з програми.

Нейронні мережі часто починають з випадкових вагів і здвигів, але потім тренують себе знову і знову, поки не досягнуть максимальної продуктивності. Вони роблять це, обчислюючи кількість помилок, яку вони наразі мають. Це називається похибкою нейронної мережі. Це обчислюється шляхом знаходження різниці між прогнозом мережі та бажаним результатом.

Сіамська нейронна мережа - це особливий тип нейронної мережі, в якій ми спочатку тренуємо зображення з послідовністю згорткових шарів, об'єднуючих шарів і повністю з'єднаних шарів, у результаті чого з'являється вектор ознак f(x1)[7]. Потім тренуємо інше зображення в тій же послідовності, щоб отримати інший вектор ознак f(x2). Тепер обчислимо d, яка буде відстанню між кожною з точок у векторі ознак f (x1) з вектором ознак f (x2). Якщо d є малим, ми можемо сказати, що обидва зображення є такими ж, якщо d велика, і навпаки.

Основною причиною для використання цього методу є відсутність даних. Сучасні алгоритми машинного навчання працюють дуже добре, коли існує величезна кількість даних, але вони можуть провалитися, якщо є дефіцит даних. У цьому методі модель повинна зробити правильний прогноз з огляду лише на один приклад у кожному класі навчального набору.

Сіамська нейронна мережа складається з двох мереж, які приймають два різних входи з енергетичною функцією у верхній частині. Параметри, які використовуються в цій нейронній мережі, пов'язані один з одним. Цей метод зважування ваги гарантує, що два подібні зображення, коли вони відображаються відповідними мережами, будуть знаходитися в одному і тому ж місці в просторі функцій, оскільки кожна мережа обчислює ту ж саму функцію.

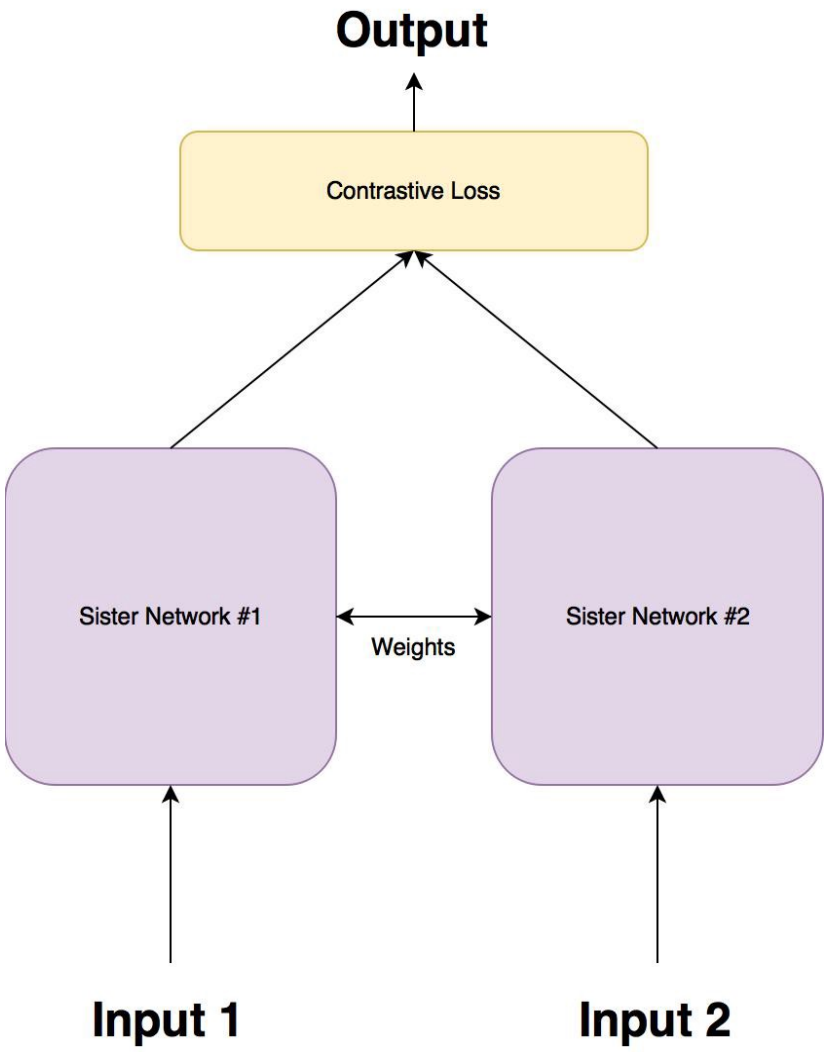

Рисунок 4.2 – Приклад SNN

Є дві основні функції похибки для SNN - contrastive loss та triplet loss[10]. Для contrastive loss використовуються позитивні і негативні пари навчальних точок даних. Для однакових пар дистанція має бути нулем, в той час як для негативної пари дистанція має наближатися до заданого значення. Якщо r_0 і r_1 є парними елементами, а y є двійковим прапором, рівним 0 для негативної пари, і 1 для позитивної пари, а відстань d - евклідовою відстанню, то функцію похибки можна записати:

$$L(r_0, r_1, y) = y||r_0 - r_1|| + (1 - y)max(0, m - ||r_0 - r_1||) \quad (6)$$

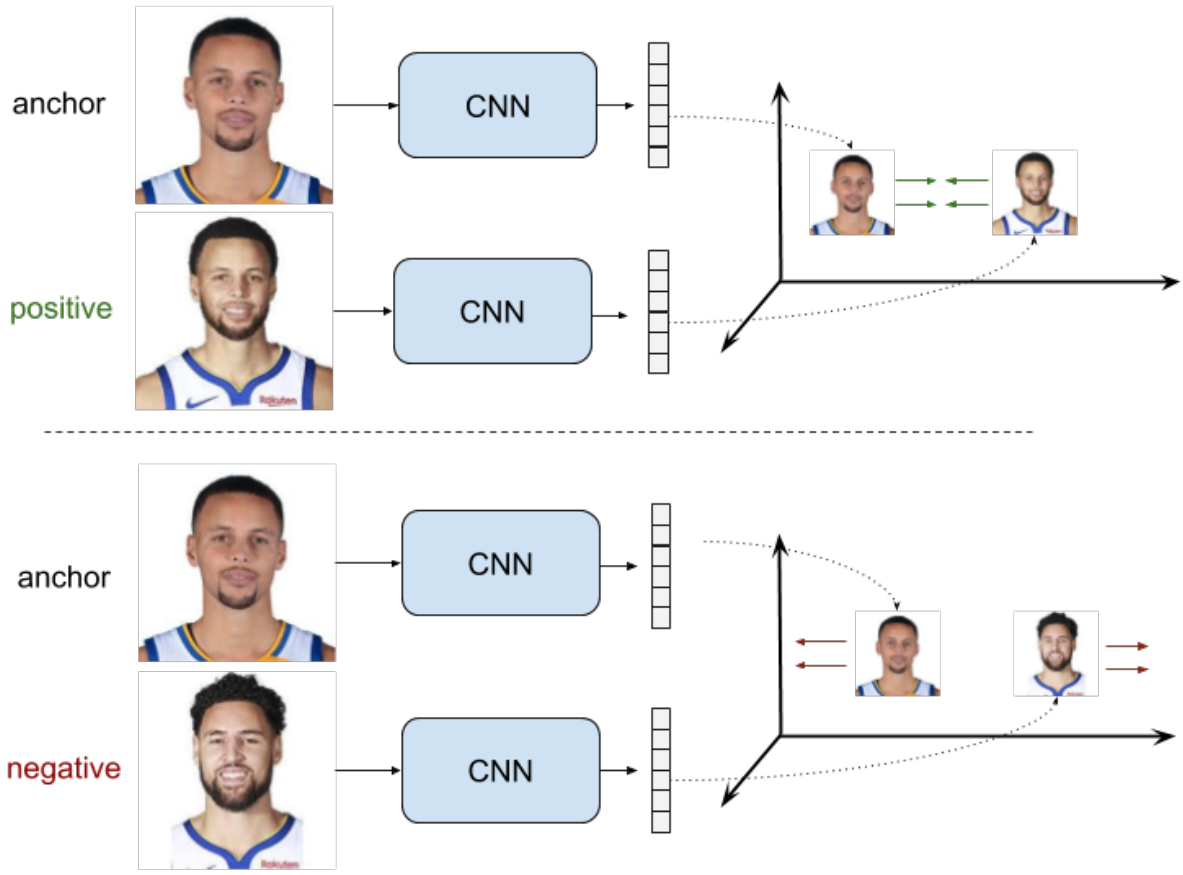

Рисунок 4.3 – Приклад роботи contrastive loss

Функція похибки triplet loss[17] перевершує першу, використовуючи замість пар триплети зразків навчальних даних. Триплети утворені зразковим прикладом x_a, позитивною пробою x_p і негативним зразком x_n. Мета полягає в тому, щоб відстань між зразком прикладом і негативним зразком d (r_a,r_n) була більше, ніж відстань між зразком і позитивними поданнями d (r_a, r_p)[8]. З такою ж нотацією ми можемо написати:

$$L(r_a, r_p, r_n) = max(0, m + d(r_a, r_p) - d(r_a, r_n)) \quad (7)$$

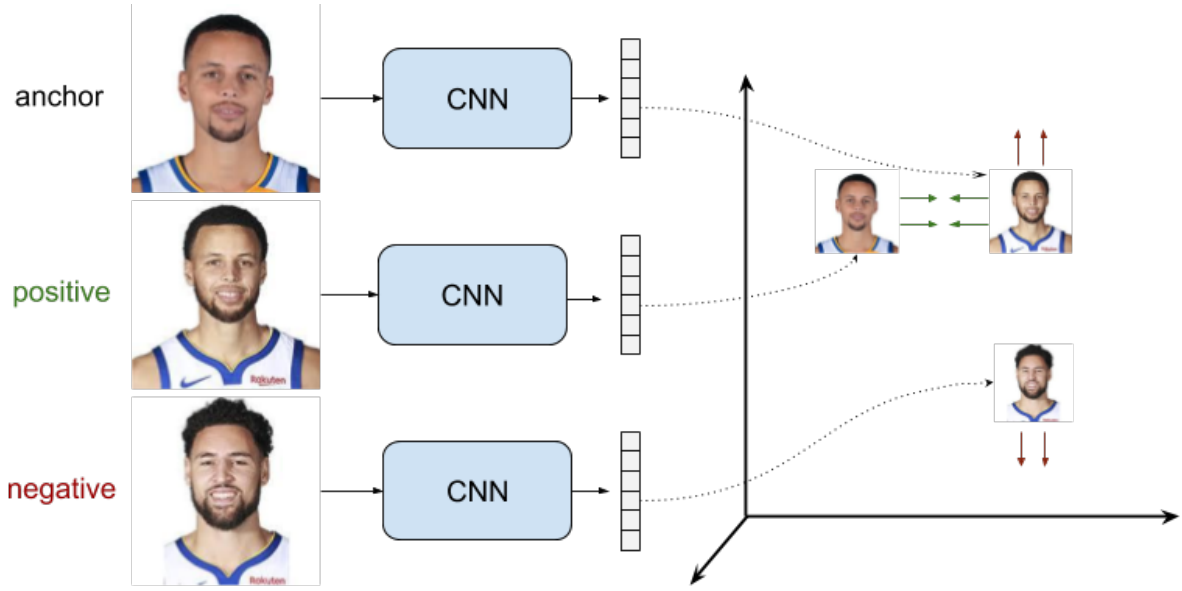

Рисунок 4.4 – Приклад роботи triplet loss

Потрібно вибирати триплети таким чином, щоб відстань позитивна була мінімальною, а відстань негативна - великою, такі триплети допомагатимуть моделі швидше збігатися і більш ефективно навчатися.

2.5 Висновки до розділу

У розділі розглянуті основні методи та алгоритми для порівняння зображень.

Це завдання низького рівня часто є першим та найважливішим кроком при проектуванні системи. Від вибору алгоритму буде залежати точність, швидкодія та витрачені ресурси при роботі.

3 БІБЛІОТЕКА TENSORFLOW ТА ПОБУДОВА НЕЙРОННОЇ МЕРЕЖІ

Для виконання поставленої задачі була використана бібліотека TensorFlow та її класи Graph, Layer.

3.1 Бібліотека TensorFlow

Бібліотека (в програмуванні) - це файл або набір файлів, що містять підпрограми, функції, які використовуються для розробки програмного забезпечення. Різні мови програмування мають свій набір бібліотек.

Виділяють:

Стандартні бібліотеки - поставляються з мовою програмування.

Призначені для користувача бібліотеки - створюються користувачами або програмістами.

TensorFlow - це обчислювальна бібліотека для побудови моделей машинного навчання. Це система другого покоління від Google Brain на чолі з Джеффом Діном[11]. Розпочатий на початку 2017 року, TensorFlow змінив світ машинного навчання шляхом залучення обчислювальних можливостей від масштабованості до побудови готових моделей.

TensorFlow забезпечує безліч різних наборів інструментів, які дозволяють писати код на бажаному рівні абстракції. Наприклад, можно написати код в Core TensorFlow (C++) і викликати цей метод з коду Python. Можно визначити архітектуру, на якій повинен працювати ваш код (CPU, GPU і т.д.). Найнижчий рівень, на якому можно написати код, це C++ або Python. Ці два рівні дозволяють писати числові програми для вирішення математичних операцій і рівнянь. Хоча це не рекомендується для побудови моделей машинного навчання, він пропонує широкий спектр бібліотек

математики, які полегшують ваші завдання. Наступний рівень, на якому можно написати свій код, використовує специфічні абстрактні методи TF, які високо оптимізовані для компонентів моделі. Наприклад, за допомогою абстрактного методу tf.layers можна грати з шарами нейронної мережі. Можна побудувати модель і оцінити продуктивність моделі за допомогою методу tf.metrics. Найбільш широко використовуваним є tf.estimator API, який дозволяє легко створювати (тренувати і прогнозувати) готові моделі. API estimator є надзвичайно простим у використанні і добре оптимізований. Незважаючи на те, що вона забезпечує меншу гнучкість, вона має все необхідне для навчання та тестування вашої моделі.

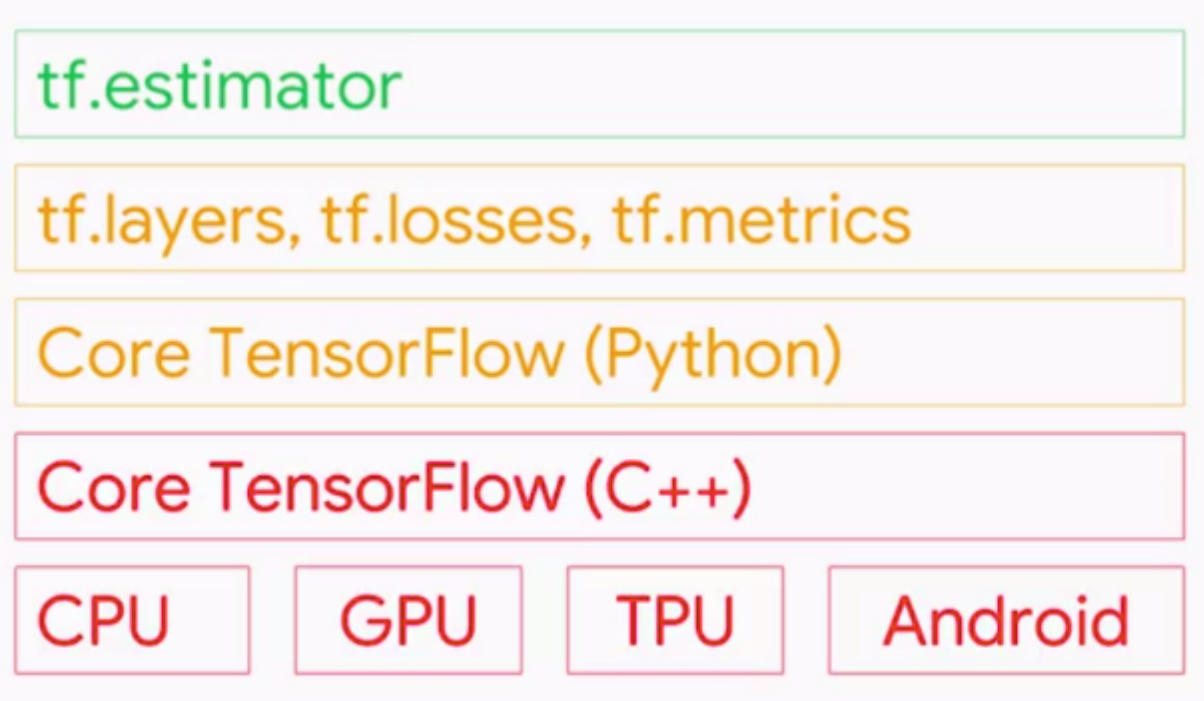

Рисунок 3.1 – Рівні абстракції TensorFlow

Основним типом даних у цій структурі є тензор. Тензор - це N-вимірний масив даних. Наприклад, можно викликати скаляр як тензор 0-розмірності. Вектор - це 1-мірний тензор, а матриця - двовимірний тензор.

3.2 Робота з Graph в TensorFlow

TensorFlow використовує граф потоку даних для представлення обчислень у термінах залежностей між окремими операціями. Це призводить до низькорівневої моделі програмування, в якій спочатку визначається графік потоку даних, а потім створюється сеанс TensorFlow для запуску частин графіка через набір локальних і віддалених пристроїв.

Високорівневі API, такі як tf.estimator.Estimator і Keras, приховують деталі графів і сесій від кінцевого користувача.

Dataflow - це загальна модель програмування для паралельних обчислень[12]. На графу потоку даних вузли являють собою одиниці обчислення, а ребра представляють дані, що споживаються або виробляються обчисленням. Наприклад, в графі TensorFlow операція tf.matmul відповідає одному вузлу з двома вхідними ребрами (матрицями, які потрібно множити) і одним вихідним краєм (результатом множення).

Dataflow має ряд переваг, які TensorFlow використовує при виконанні програм:

● Паралелізм - використовуючи явні ребра для представлення залежностей між операціями, системі легко визначити операції, які можуть виконуватися паралельно.

● Розподілене виконання - використовуючи явні ребра для представлення значень, що протікають між операціями, TensorFlow може розділити вашу програму на декілька пристроїв (процесорів, графічних процесорів і ТПУ), приєднаних до різних машин. TensorFlow вставляє необхідні комунікації і координацію між пристроями.

● Компіляція - компілятор XLA TensorFlow може використовувати інформацію у вашому графіку потоку даних для генерації більш швидкого коду, наприклад, шляхом злиття суміжних операцій.

● Переносимість - графік потоку даних не залежить від мови представлення коду вашої моделі. Ви можете побудувати графік потоку даних в Python, зберегти його в SavedModel і відновити в програмі C ++ для низьких затримок.

Вузли і ребра графа, що вказують, як окремі операції складаються разом, але не прописують, як вони повинні бути використані. Структура графіка схожа на код збірки: перевірка може передати деяку корисну інформацію, але вона не містить всього корисного контексту, який передає вихідний код.

TensorFlow забезпечує загальний механізм для зберігання колекцій метаданих в tf.Graph. Функція tf.add_to_collection дозволяє асоціювати список об'єктів з ключем (де tf.GraphKeys визначає деякі стандартні ключі), а tf.get_collection дозволяє шукати всі об'єкти, пов'язані з ключем. Багато частин бібліотеки TensorFlow використовують цю функцію: наприклад, коли створюється tf.Variable, вона додається за замовчуванням до колекцій, що представляють "глобальні змінні" і "тренувальні змінні". Пізніше при створенні tf.train.Saver або tf.train.Optimizer, змінні в цих колекціях використовуються як аргументи за замовчуванням.

Більшість програм TensorFlow починаються з фази побудови графіка потоку даних. На цій фазі ви викликаєте функції TensorFlow API, які будують нові об'єкти tf.Operation (node) і tf.Tensor (edge) і додають їх до tf.Graph. TensorFlow надає граф за замовчуванням, який є неявним аргументом для всіх функцій API в одному контексті. Виклик tf.constant (42.0) створює єдину tf.Operation, яка виробляє значення 42.0, додає її до графіка за замовчуванням, і повертає tf.Tensor, що представляє значення константи. Виклик tf.matmul (x, y) створює єдину операцію tf.Operation, яка помножує значення об'єктів tf.Tensor x і y, додає її до графіка за замовчуванням і повертає tf.Tensor, який представляє результат множення. Виконання v = tf.Variable (0) додає до графа tf.Operation, яке зберігатиме значення тензора, яке зберігається між викликами tf.Session.run. Об'єкт tf.Variable обертає цю операцію і може використовуватися як тензор, який зчитує поточне значення збереженого значення. Об'єкт tf.Variable також має такі методи, як tf.Variable.assign і tf.Variable.assign_add, які створюють об'єкти tf.Operation, які при виконанні оновлюють збережені значення. Виклик tf.train.Optimizer.minimize додасть операції і тензори до графіка за замовчуванням, який обчислює градієнти, і повертає tf.Operation, який при запуску застосує ці градієнти до набору змінних.

3.3 Створення нейронної мережі за допомогою модуля Layers.

У машинному навчанні модель є функцією з параметрами, які можна вивчати, що відображає вхід на вихід. Оптимальні параметри отримують шляхом навчання моделі за даними. Добре підготовлена модель забезпечує точне відображення від входу до потрібного виходу.

У TensorFlow існує два способи створення моделі машинного навчання:

- Використовуючи API "Layers", де створюється модель за допомогою шарів.
- Використовуючи API Core з операціями нижчого рівня, такими як tf.matMul (), tf.add () і т.п.

Основними шарами для нейронної мережі для порівняння зображень є :

- Шари згортки які виділяють особливі шаблони на зображенні.
- Шари зменшення розміру для зменшення розмірності зображення та прискорення роботи.
- Шари нормалізації для пришвидшення роботи та стабільності моделі.
- Шари активації для нелінійності функції.
- Повнозвязні шари для представлення закодованого вектору зображення.

Також була використана особливість шарів яка дозволяє використовувати однакові ваги для різних шарів.

3.4 Розробка та тренування нейронної мережі

Створення нейронної мережі дуже довгий процес, який складається з багатьох частин. Перш за все необхідно визначити яку задачу треба вирішити та з якими даними потрібно буде працювати. Після цього проводиться пошук даних серед відкритих датасетів. Якщо потрібного датасету не існує то потрібно створити його

власноруч. Для даної задачі датасет було створення через парсинг сайтів та збереження зображень з анотаціями до них.

Наступним кроком був вибір архітектури нейронної мережі. Є три основні підходи для побудови сіамських нейронних мереж.

Перший, це дві однакові нейронні мережі які мають спільні ваги. Вони кодують значення в вектор значень, потім ці вектори об'єднуються і новий вектор передається в класифікатор який визначає чи є подані на вхід дані однаковими. Недоліком цього методу є те, що потрібно весь час використовувати класифікатор. Для навчання використовується крос-ентропійна функція втрат. Це витрачає багато ресурсів та часу, тому цей метод не є оптимальним для нас.

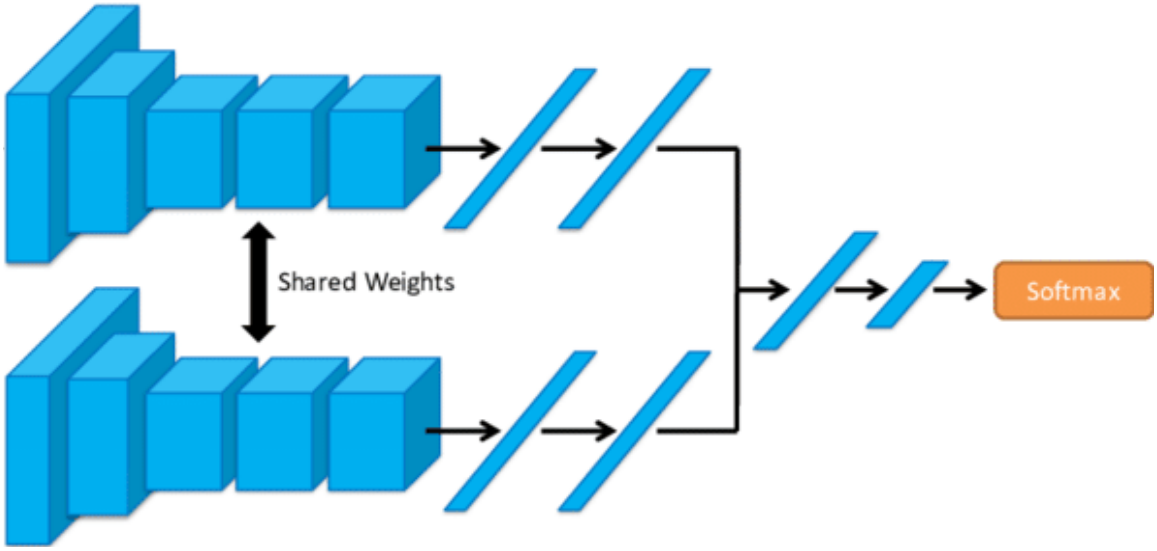

Рисунок 3.2 – Приклад SNN з классифікатором

Другий, це підхід в якому классифікатор не використовується. Для визначення схожості використовується евклідова відстань. Перевагою цієї архітектури є те, що після навчання частину нейронної мережі можно відрізати та залишити лише одну вітку для кодування. Таким чином зображення можно закодувати та зберігти результати. Після цього якщо зображення потрібно буде порівняти з новим, то потрібно буде лише закодувати нове зображення та порівняти їх через евклідову відстань.

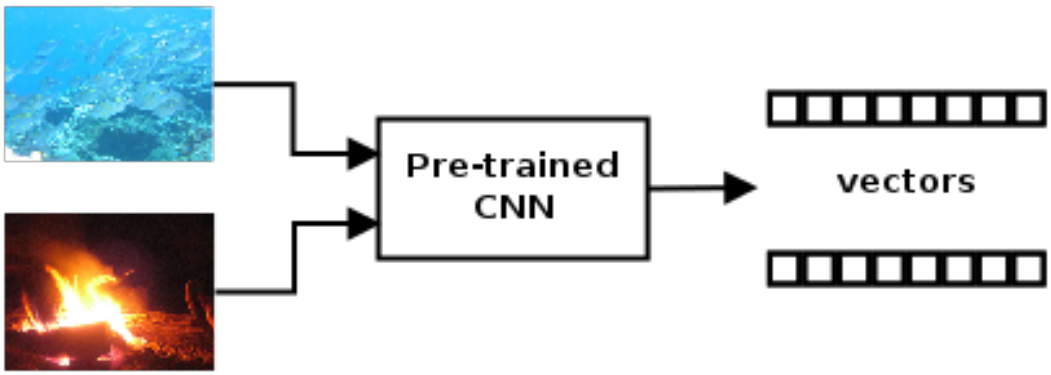

Рисунок 3.3 – Приклад SNN без класифікатора

Третій, покращення архітектури без класифікатора - Triplet SNN. В ній використовується відразу три копії нейронної мережі для кодування. На вхід подається завжди три зображення, два зображення з яких належать до одного класу, а третій до іншого. Таким чином нейронна мережа завжди отримує однакову кількість позитивних та негативних прикладів, що дуже добре впливає не результат навчання.

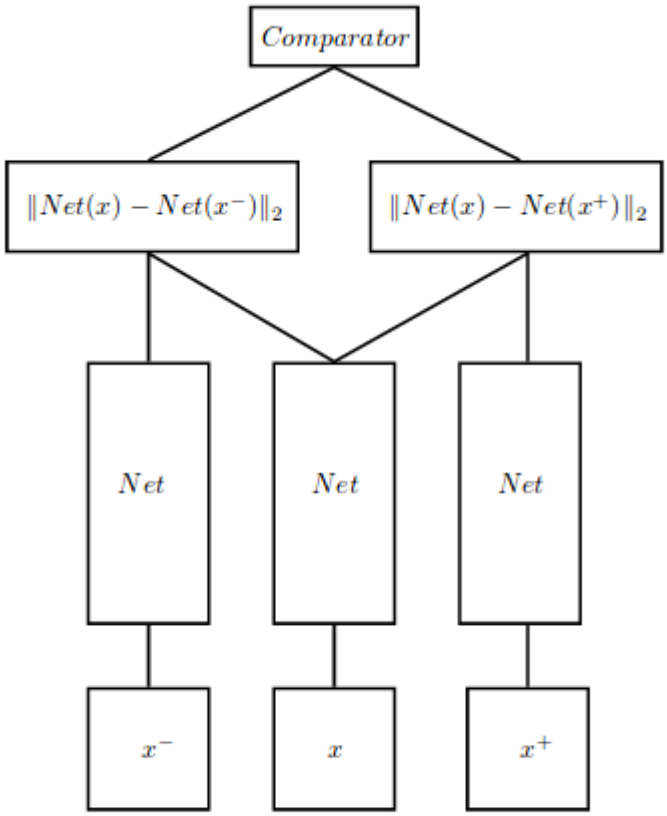

Рисунок 3.4 – Приклад triplet SNN

Так як задача є не тривіальною функції яка описувала би точність не написана в стандартній бібліотеці TensorFlow. Тому була написана власна функція для підрахунку точності моделі:

```
accuracy = tf.div( (tf.reduce_sum(tf.cast(positive_distance<0.5, tf.float32))
+tf.reduce_sum(tf.cast(negative_distance>0.5, tf.float32)) ),
tf.cast(tf.shape(positive_distance)[0], tf.float32)*2.)
```

Так як при навчанні градієнт протікає три рази по вагам нейронної мережі його прийшлося значно знизити, щоб не отримати saturated gradient.

Для такої нейронної мережі знадобилось розробити особливий клас для групування та підготовки зображень перед подачею їх на вхід нейронної мережі.

Для покращення результатів навчання була використана бібліотека OpenCV для аугментації даних. При подачі зображення до нейронної мережі застосовувалася випадкова функція яка змінювала зображення: віддзеркалювала його, трохи крутила, обрізала, міняла контраст, робила зображення яскравішим чи темнішим або нічого з ним не робила.

Після навчання нейрона мережа зберігається в стані, що навчання можно продовжити. Після цього граф обрізався таким чином, щоб залишився один вхід та один вихід. Потім вся значення переписувалися в константи та зберігалися в бінарний серіалізований файл. В такому вигляді модель дуже зручно використовувати в системах які потребують нейронні мережі.

3.5 Висновки до розділу

У цьому розділі було розглянуто роботу фреймворку TensorFlow.

Також було ознайомлено з основними принципами та методами на яких будуються нейронні мережі.

Було розглянуто роботу модуля Layers та Graph які дозволяють швидко та ефективно побудувати нейронну мережу будь якої складності.

4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Web-система призначена для двох акторів. Користувача, що бажає загрузити зображення та отримати найбільш схоже в базі даних та адміністратора яких може загрузити нові зображення в базу даних.

Для того, щоб мати можливість користуватися системою користувач повинен зареєструватися або увійти у систему.

4.1 Структура програми

Програмне забезпечення для розв'язання поставленої задачі спроектовано з дотриманням парадигм об'єктно-орієнтованого програмування.

Розглянемо схему роботи розробленого программного продукту на рисунку 4.1.

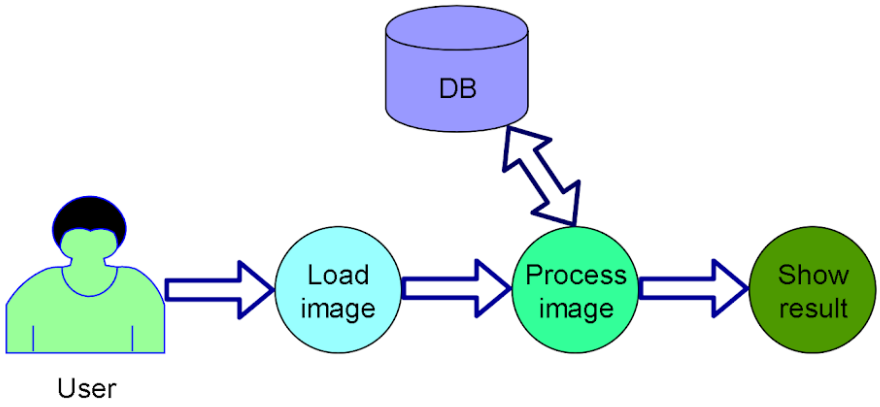

Рисунок 4.1 – Схема роботи розробленого програмного продукту

Спочатку користувач повинен пройти реєстрацію або увійти у систему, потім завантажити зображення на сервер для подальшої обробки. Отримані результати записуються в базу даних і повертаються користувачеві. Користувач отримує зображення з бази даних та всю інформацію яка належить до цього зображення

4.2 Опис розроблених алгоритмів

Для розробки даного програмного продукту по пришвидшеному пошуку зображень за допомогою нейронних мереж у вигляді web-системи були використані такі технології як бібліотеки TensorFlow, OpenCV, Pillow, NumPy для роботи з нейронною мережею та зображеннями.

Для збору датасету була використана бібліотека BeautifulSoup4. За допомогою цієї бібліотеки було написано парсер який ходить по сайту, зберігає зображення та всю інформацію, що відноситься до цього зображення[18].

Для реалізації серверної частини використовувались такі технології як Python, Django, MongoDB, MongoDB Compass, Visual Studio Code.

Були виконані такі кроки:

1. Проведено попереднє завантаження зображення на сервер.
2. Збір даних для датасету через парсинг сайтів.
3. Розробка нейронної мережі для кодування зображень.
4. Збереження та оптимізація нейронної мережі для використання в Web-системі.
5. Використання математичних алгоритмів для порівняння закодованих зображень.
6. Пошук відповідного зображення в базі даних та відображення результатів.

Для кращого розуміння продемонстрована діаграма класів для бізнес логіки на рисунку 4.2. На ній можна поділити систему класів на дві основні, такі як класи сутності для зв`зку з базою даних та логика програми.

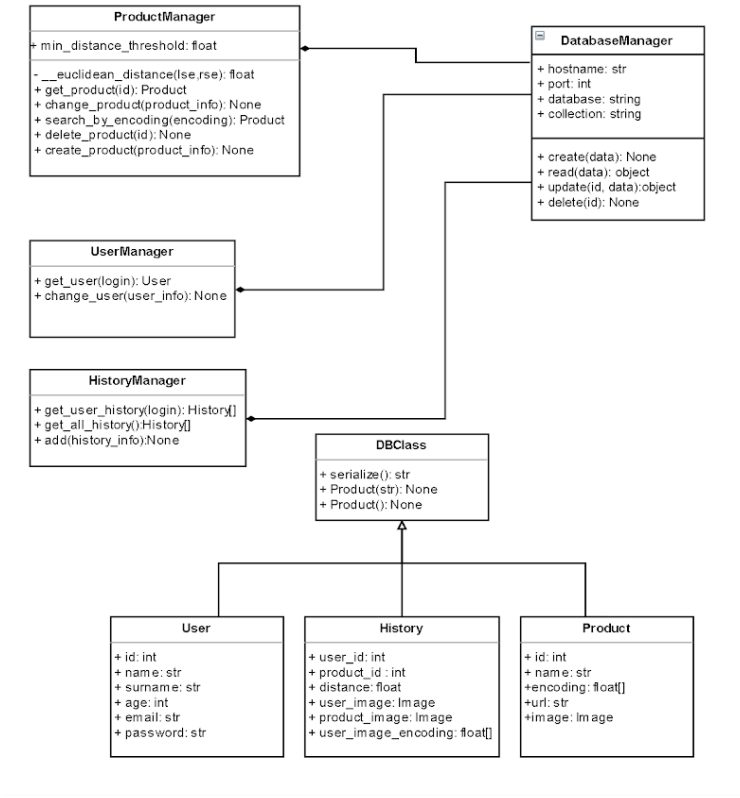

Рисунок 4.2–Діаграма класів програмного продукту

Для використання нейронної мережі був створений окремий модуль. Спочатку створюється нова сесія TensorFlow. Потім модуль він зчитує бінарний файл за допомогою спеціального класу TensorFlow GFile. Після десеріалізації можно получити доступ до нашого графу з обученою нейронною мережею. Потім стандартній граф сесії замінюється на зчитаний граф.

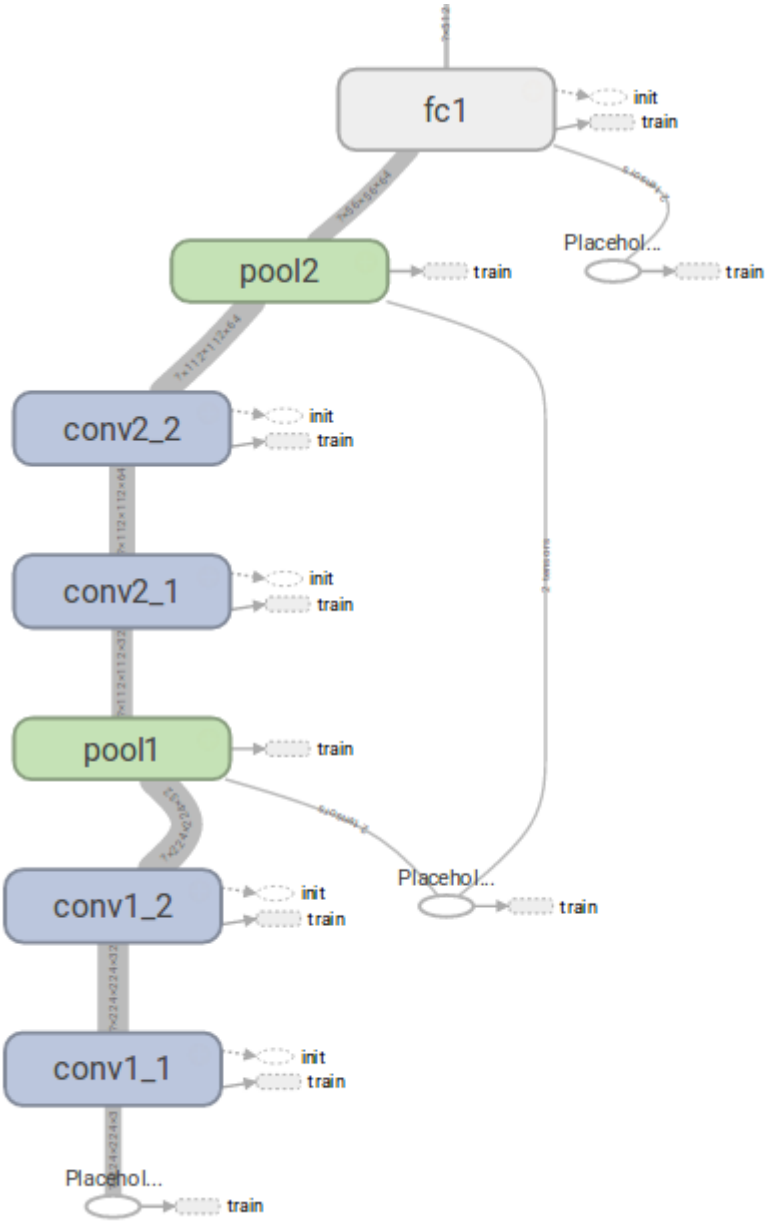

Рисунок 4.3 Приклад графу

Далі необхідно знайти та забрати тензори які відповідають за вхідні дані та за результат виконання нейронної мережі. Останнім етапом є підготовка зображень для подання їх до нейронної мережі. Для кожного зображення змінюється розмір, воно приводиться до RGB формату та додається додатковий, четвертий простір. Після цього зображення можно подавати до нейронної мережі. На виході ми отримуємо закодоване представлення зображення, яке спочатку потрібно перетворити з вигляду матриці до вигляду масиву і після цього отриманий результат можно використовувати для пошуку по зображенням.

4.3 Висновки до розділу

У розділі було розкрито структуру програмного продукту, методи, що були використані при розробці та керівництво користувача, де були описані весь функціонал.

Головні сторінки - це сторінка завантаження зображення користувача на сервер та пошуку за допомогою нейронної мережі найближчого зображення в базі даних.

Було сформоване детальне керівництво користувача для полегшення у користуванні програмним продуктом. Керівництво користувача містить необхідні інструкції для швидкого ознайомлення із усім функціоналом системи. Також присутні screenshot’и кожної сторінки web-системи.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

Для того, щоб скористатись функціоналом обробки та сегментації динамічних зображень користувач повинен зареєструватися, рисунок 5.1, або увійти в систему, рисунок 5.2.

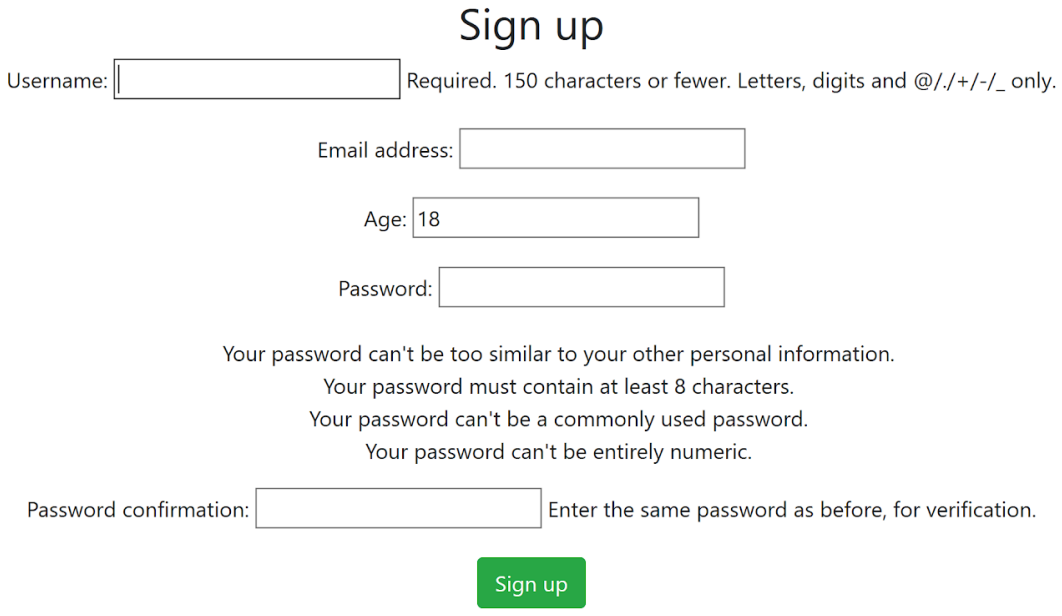

Рисунок 5.1–Форма реєстрації нового користувача.

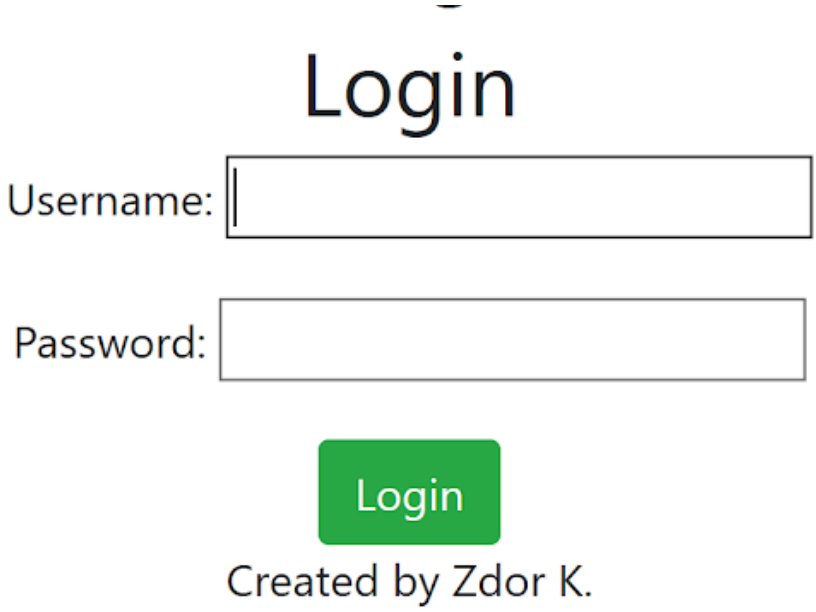

Рисунок 5.2–Форма входу у систему існуючого користувача

Для завантаження зображення на сервер користувач має зайти на головну сторінку, рисунок 5.3.

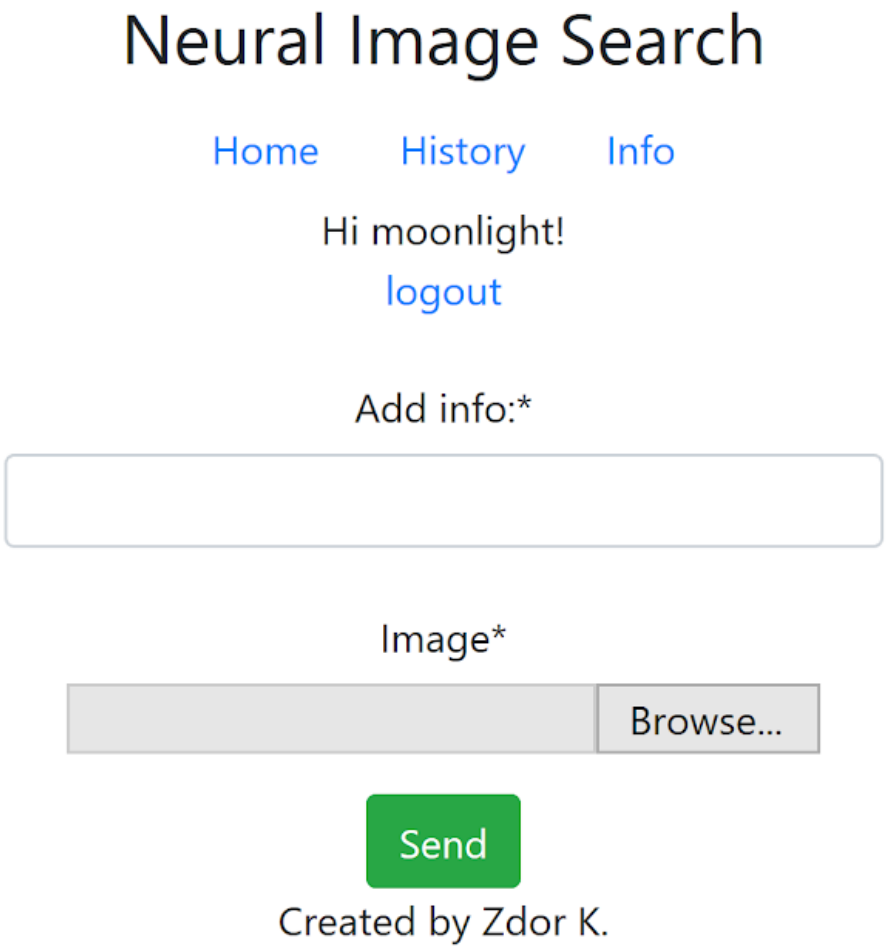

Рисунок 5.3–Сторінка завантаження зображення на сервер

На цій сторінці користувач може ввести інформацію про зображення, та має вибрати зображення. Після відправлення даних користувач отримує сторінку з результатами пошуку зображеними на рисунку 5.4.

Рисунок 5.4–Приклад результату пошуку по зображенню

Користувач може перейти в вкладку History для того щоб переглянути результати своєї історії. В результаті він отримає сторінку зображену на рисунку 5.5.

Date	Image	Image info	Server image	Server info	Distance	User login
12.06.2019 18:12:00	cup.jpeg		qscoaobnaodb.jpeg	Cup;buy_cup.com	0	moonlight
12.06.2019 18:13:00	apple.png	Red apple	asdf1r1d1sc.jpeg	Red apple; Fruit	0.17	moonlight

Рисунок 5.5 – Приклад історії користувача

Якщо перейти в вкладку Info то можно подивитися на архітектуру нейронної мережі, рисунок 5.6.

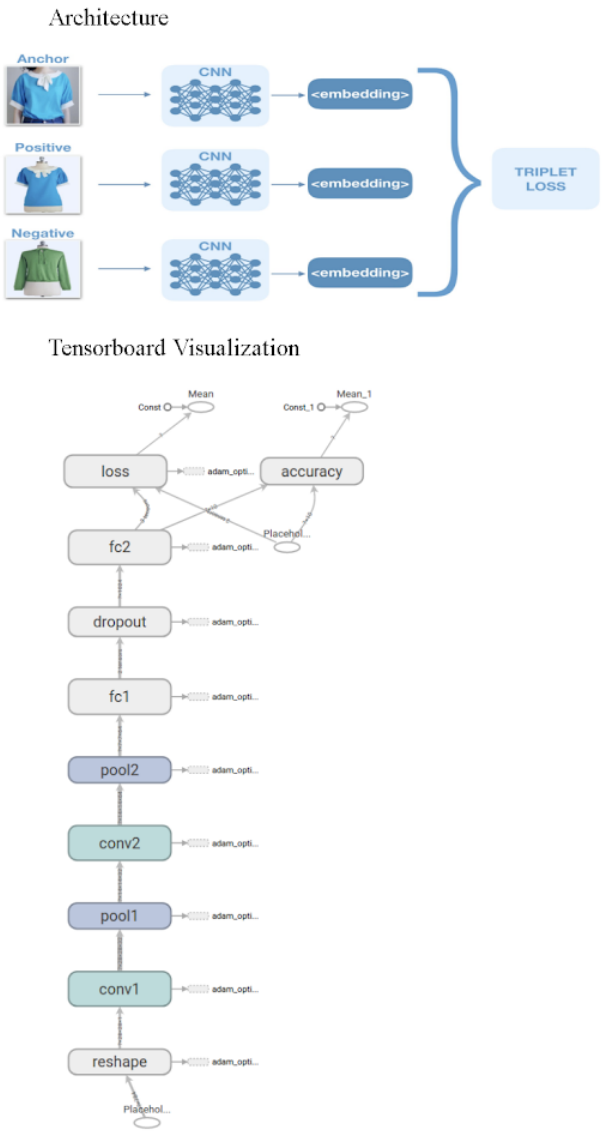

Рисунок 5.6 – Сторінка Info

ВИСНОВКИ

В результаті виконання роботи було досліджено методи пошуку зображень за допомогою нейронних мереж, а також розроблена web-система для зручного користування та подальшої інтеграції в інтернет-магазини.

Під час виконання було проаналізовано проблему прискореного пошуку зображень за допомогою нейронних мереж.

Було спроектовано та розроблено програмне забезпечення у вигляді web-системи по пошуку зображень в базі даних за допомогою нейронної мережі.

Проведено огляд методів і засобів тренування нейронних мерез та зосереджена увага на ознайомлені з бібліотекою TensorFlow.

Реалізовано функціонал кодування зображень за допомогою сімської нейронної мережі. Для порівняння результатів розпізнавання та подальшого покращення нейронної мережі було реалізоване збереження результатів пошуку.

Система спроектована з урахуванням можливості розширення функціоналу в майбутньому.

Виконано усі задачі, необхідні для досягнення мети роботи, а саме пришвидшений пошук зображень за допомогою нейронних мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lukyanitsa A. A.Digital processing of images / A.A. Lukyanitsa, A.G. Shishkin - Moscow: ISSC Press, 2009. – 518c.

2. Yilmaz A. Object detection: A survey /A.Yilmaz, O. Javed, M. Shah. - ACM Computing Surveys, 2006, vol. 38, no. –94c.

3. Martínez-Martín E. Robust image detection in real-life scenarios./ E. Martínez-Martín, A. Pobil– Springer-Verlag London, 2012. – 108 c.

4. SURF. [Електронний ресурс]/Режим доступу:https://medium.com/software-incubator/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e

5. Novoseltsev IV Siamese Neural Networks / IV Novoseltsev., N.G. Aksak -M: Information Processing Systems, 2007. - Vip. 3. – 64 c.

6. Комп`ютерний зір. [Електронний ресурс]/Режим доступу:https://www.amazon.com/Computer-Vision-Modern-Approach-2nd/dp/013608592X

7. Pratt W. Neural Networks / W. Pratt - M .: Mir, 1982. - 480 c.

8. Siamese Neural Network for One-Shot learning. [Електронний ресурс]/Режим доступу:https://medium.com/@subham.tiwari186/siamese-neural-network-for-one-shot-image-recognition-paper-analysis-44cf7f0c66cb

9. Soifer V.A. Methods of computer image processing / V.A. Soifer - Moscow: FIZMATLIT, 2003.— 784 c.

10. Contrastive and Triplet losses. [Електронний ресурс]/Режим доступу: https://gombru.github.io/2019/04/03/ranking_loss/

11. Gonzalez R. Digital image processing / R. Gonzalez, R. Woods - M .: Technosphere, 2012. — 1104 c.

12. Anisimov B. V. Recognition and digital image processing // BV Anisimov, VD Kurganov, VK Zlobin - M .: Vyssh., 1983. — 295 c.

13. Shapiro L. Computer vision // L. Shapiro, D. Stockmann - M .: Binom. Laboratory of knowledge, 2006. — 752 c.

14. Yu. V. Vizilter. Processing and analysis of images in computer vision problems. // Yu. V.Visilter, S. Yu. Zheltov, - Moscow: Fizmatkniga, 2010. — 689 c.

15. Zhuravlev Yu.I. Image recognition and image recognition // Yu. I. Zhuravlev, I.B. Gurevich - Moscow: Science, 1989. –72 c.

16. Janet B. Digital image processing / B. Yane - M .: Technosphere, 2007. — 584 c.

17. Oppenheim A. Digital signal processing/A. Oppenheim, V. Shafer - M .: Technosphere, 2012. - 1048 c.

18. Forsyth D. Computer Vision: A Modern Approach (2nd Edition) / D. Forsyth, J. Ponce - Pearson, 2011. - 792 p.

ДОДАТОК А

Web-система прискореного пошуку зображень за допомогою нейронних мереж

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ42136_17Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ4 2136_17Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ4 2136_17Б 12-1	ProductManager.py	Основний компонент
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ4 2136_17Б 12-2	SiameseNeuralNetwork.py	Компонент створення нейронної мережі

ДОДАТОК Б

Web-система прискореного пошуку зображень за допомогою нейронних мереж

Текст програми

УКР.НТУУ "КПІ". ТВ42136_17Б 12-1

Аркушів 23

2019

product_manager.py

```
import numpy as np

class ProductManager:

  def __init__(self, db_manager):
    self.db_manager = db_manager
    self.min_distance_threshold = 0.5

  def __euclidean_distance(self, lse, rse):
    lse = np.asarray(lse)
    rse = np.asarray(rse)
    return np.sum(np.square(lse-rse))

  def get_product(self, id):
    return self.db_manager.get_product().objects(id=id)

  def change_product(self, product_info):
    product = self.db_manager.get_product().objects(id=product_info['id'])[0]
    product.name = product_info['name']
    product.url = product_info['url']
    product.image = product_info['image']
    product.encoding = product_info['encoding']
    product.save()

  def search_by_encoding(self, encoding):
    products = self.db_manager.get_product().objects
    min_distance = self.min_distance_threshold+0.1
    match = None
    for product in products:
      distance = self.__euclidean_distance(encoding, product['encoding'])
      if distance < min_distance:
        match = product
```

```
                min_distance = distance
        return match, min_distance

    def delete_product(self, id):
        product = self.db_manager.get_product().objects(id=id)[0]
        product.delete()

    def create_product(self, **product_info):
        #warning
        product = self.db_manager.get_product()()
        product.name = product_info['name']
        product.url = product_info['url']
        product.image.put(  open(product_info['image'], 'rb'), content_type = 'image/jpeg')
        #image = ImageField()
        product.encoding = product_info['encoding'].tolist()
        #product = self.db_manager.get_product()(**product_info)
        product.save()
```

nn_manager.py

```
import tensorflow as tf
from tensorflow.python.platform import gfile
import numpy as np
import os
from PIL import Image

class NNManager:
    def __init__(self, path2pb='siamese_model/Graph.pb'):
        self.input_ph = 'import/inputs/anchor_ph:0'
        self.output_ph = 'import/siamese/output/Sigmoid:0'
        self.sess = tf.Session()
        with tf.gfile.GFile(path2pb, 'rb') as f:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(f.read())
```

```
            self.sess.graph.as_default()
            tf.import_graph_def(graph_def)
        self.input = self.sess.graph.get_tensor_by_name(self.input_ph)
        self.output = self.sess.graph.get_tensor_by_name(self.output_ph)

    def prepare_image(self, image):
        img = image.copy()
        img = img.convert('LA')
        img = img.resize((28, 28), Image.ANTIALIAS)
        return img

    def eval(self, data):
        data = data.resize((28,28),  Image.ANTIALIAS)
        data = np.asarray(data)
        try:
            data = data.reshape(1,28,28,1)
        except:
            data = data.reshape(1,28,28,3)
            data = data[:,:,:,0]
        #data = np.expand_dims(data, axis=0)
        data = data.reshape(1,28,28,1)
        predictions = self.sess.run(self.output, feed_dict={self.input: data})
        predictions = np.squeeze(predictions)
        return predictions

if __name__=='__main__':
    data = np.random.rand(28, 28, 1).astype(np.float32)
    nn = NNManager('WebSearch\\managers\\Graph.pb')

    prediction = nn.eval(data)
    print(prediction)
```

history_manager.py

```
class HistoryManager:

  def __init__(self, db_manager):
    self.db_manager = db_manager

  def get_user_history(self, user_login):
    return self.db_manager.get_history().objects(user_login=user_login)

  def get_all_history(self):
    return self.db_manager.get_history().objects

  def add(self, **history_info):
    #warning
    history = self.db_manager.get_history()(**history_info)
    history.save()
```

user_manager.py

```
class UserManager:

  def __init__(self, db_manager):
    self.db_manager = db_manager

  def get_user(self, login):
    return self.db_manager.get_user().objects(login=login)

  def change_user(self, user_info):
    user = self.db_manager.get_user().objects(login=user_info['login'])[0]
    user.name = user_info['name']
    user.surname = user_info['surname']
    user.age = user_info['age']
    user.email = user_info['email']
```

```
        user.password = user_info['password']
        user.save()

db_manager.py
from mongoengine import *
from WebSearch.domain import *

class DBManager:
    def __init__(self, db='NeuralImageSearch', host='localhost', port=27017):
        connect(db)

    def get_user(self):
        return User

    def get_history(self):
        return History

    def get_product(self):
        return Product
domain.py
from mongoengine import *

class History(Document):
    user_login = StringField()
    product_id = StringField()
    distance = FloatField()
    user_image = ImageField()
    product_image = ImageField()
    user_image_encoding = ListField(FloatField())

class Product(Document):
    #id = StringField()
    name = StringField()
```

```
    url = StringField()
    image = FileField()
    #image = ImageField()
    encoding = ListField(FloatField())
    meta = {'strict': False}
class User(Document):
    login = StringField()
    name = StringField()
    surname = StringField()
    age = IntField()
    email = EmailField()
    password = StringField()
```

views.py

```
from django.http import HttpResponse
from django.shortcuts import render
from .forms import UserForm, ProductForm
from .managers import DBManager, ProductManager, UserManager, NNManager
import os
from PIL import Image
import numpy as np
import io
import base64
db = DBManager()
nn_manager = NNManager()
user_manager = UserManager(db)
product_manager = ProductManager(db)


def handle_uploaded_file(name, file):
    name = f'IMAGES/{name}'
    #os.mkdir('IMAGES')
    with open(name, 'wb+') as destination:
```

```
        for chunk in file.chunks():
            destination.write(chunk)
    return name

def index(request):
    db = DBManager()
    if request.method == "POST":
        name = request.POST.get("info")
        image = request.POST.get("image")
        image_name = handle_uploaded_file(str(request.FILES['image']),
request.FILES['image'])
        image = Image.open(image_name)
        encoding = nn_manager.eval(image)
        product, distance = product_manager.search_by_encoding(encoding)
        if product is not None:
            product_name = product.name
            product_url = product.url
            product_image = product.image.read()
            product_image_id = f'images\\{str(product.image._id)}.jpeg'
            product_image = Image.open(io.BytesIO(product_image))

            product_image.save('static\\'+product_image_id, 'JPEG')
            data = {'success':True, 'name':product_name, 'url':product_url,
'image':product_image_id, 'distance':distance}
        else:
            data = {'success':False}
        return render(request, "home/search_result.html", context = data)
    else:
        data = {'form' : UserForm()}
        return render(request, "home/home.html", context=data)
```

```
def upload_product(request):
  db = DBManager()
  if request.method == "POST":
     name = request.POST.get("name")
     url = request.POST.get("url")
     #image = request.POST.get("image")
     image_name = handle_uploaded_file(str(request.FILES['image']),
request.FILES['image'])

     image = Image.open(image_name)
     encoding = nn_manager.eval(image)
     data = {'name':name, 'url':url, 'image':image_name, 'encoding':encoding}
     product_manager.create_product(**data)

  data = {'form' : ProductForm()}
  return render(request, "home/upload_product.html", context=data)
```

siamese_network.py

```
# -*- coding: utf-8 -*-
"""Siamese Network.ipynb

Automatically generated by Colaboratory.

Original file is located at
  https://colab.research.google.com/drive/1FUa0onQMb55BVlt4kgSDrECY0WfxCfQV
"""

import numpy as np
import keras
from keras.datasets import fashion_mnist
from keras.utils.np_utils import to_categorical
```

```
import tensorflow as tf
import threading
import queue
import matplotlib.pyplot as plt

BATCH_SIZE=64
EPOCHS = 25
VAL_SPLIT = 0.85
DROPOUT_RATE = 0.3

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
#(y_train, y_test) = map(to_categorical, [y_train, y_test])
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.
split = int(len(x_train)*VAL_SPLIT)
x_val = x_train[split:]
y_val = y_train[split:]
x_train = x_train[:split]
y_train = y_train[:split]

train_groups = [(x_train[np.where(y_train==i)[0]],i) for i in np.unique(y_train)]
val_groups = [(x_val[np.where(y_val==i)[0]],i)  for i in np.unique(y_val)]
test_groups = [(x_test[np.where(y_test==i)[0]],i)  for i in np.unique(y_train)]
print('train groups:', [x[0].shape[0] for x in train_groups])
print('val groups:', [x[0].shape[0] for x in val_groups])
print('test groups:', [x[0].shape[0] for x in test_groups])

class GroupLoader:
 def __init__(self, group, shuffle=True):
   self.x = np.asarray(group)
   self.len = len(self.x)
   self.index = 0
   self.shuffle = shuffle
```

```
  def __len__(self):
   return self.len
  def get(self, count):
   if self.index + count > self.len:
    self.index = 0
    if self.shuffle:
     np.random.shuffle(self.x)
   batch_x = self.x[self.index:self.index+count]
   self.index += count
   return batch_x

class DatasetLoader:
 def __init__(self, dataset, batch_size=BATCH_SIZE, shuffle=True):
   self.data = {y:GroupLoader(x, shuffle) for x,y in dataset}
   self.keys = np.asarray([int(x) for x in self.data.keys()])
   self.batch_size = batch_size
   self.out_shape = (self.batch_size, 28,28,1)

 def __len__(self):
   return sum([len(x) for x in self.data.values()])//self.batch_size*3
  def get(self):
   anchor_keys = np.random.choice(self.keys, self.batch_size)
   negative_keys = np.random.choice(self.keys, self.batch_size)
   negative_keys[anchor_keys==negative_keys] +=1
   negative_keys[negative_keys==len(self.keys)] = 0
   anchor = np.asarray([self.data[key].get(1) for key in
anchor_keys]).reshape(self.out_shape)
   positive = np.asarray([self.data[key].get(1) for key in
anchor_keys]).reshape(self.out_shape)
   negative = np.asarray([self.data[key].get(1) for key in
negative_keys]).reshape(self.out_shape)
   return anchor, positive, negative
```

```
class BackgroundDatasetLoader(threading.Thread, DatasetLoader):
 def __init__(self, dataset, batch_size=BATCH_SIZE, shuffle=True):
   DatasetLoader.__init__(self, dataset, batch_size, shuffle)
   threading.Thread.__init__(self)
   self.queue = queue.Queue(1)
   self.daemon = True
   self.start()

 def run(self):
   while True:
     self.queue.put(self.get())
   self.queue.put(None)

 def next(self):
   next_item = self.queue.get()
   if next_item is None:
     print('Queue is empty.')
     next_item = self.get()
     if next_item is None:
       raise StopIteration
   return next_item

train_loader = DatasetLoader(train_groups, BATCH_SIZE)
val_loader = DatasetLoader(val_groups, BATCH_SIZE)
test_loader = DatasetLoader(test_groups, BATCH_SIZE)

print(len(train_loader))

class TrainTriplet:
 def __init__(self, loaders):
   self.train_loader, self.val_loader, self.test_loader = loaders
   self.build()
  def _euclidean_distance(self, first_input, second_input, name):
```

```
  return tf.reduce_sum(tf.square(first_input-second_input), axis=1, keepdims=True)#tf.sqrt(, name=name)

 def _triplet_loss(self, positive_distance, negative_distance):

  margin = 0.4
  loss = tf.reduce_mean(tf.maximum(0.0, positive_distance -
                negative_distance + margin), name='triplet_loss')  # , axis=0, keepdims=True
  return loss
 def __conv_block(self, x, filters, filter_size=(3,3), pool_size=(2,2), pool_strides=(2,2), activation=tf.nn.relu, reuse=False, name='conv_block_1_'):
  x = tf.layers.conv2d(x, filters, kernel_size=filter_size, activation=activation, padding='same', reuse=reuse, name=name+'conv1')
  x = tf.layers.conv2d(x, filters, kernel_size=filter_size, activation=activation, padding='same', reuse=reuse, name=name+'conv2')
  x = tf.layers.max_pooling2d(x, pool_size, pool_strides, padding='same', name=name+'mp')
  x = tf.layers.batch_normalization(x, reuse=reuse, name=name+'bn')
  x = tf.layers.dropout(x, self.dropout_rate, name=name+'dr')
  return x

 def __fc_block(self, x, units, activation=tf.nn.relu, reuse=False, name='fc_block_1_'):
  x = tf.layers.dense(x,units, activation=activation, reuse=reuse, name=name+'dense')
  x = tf.layers.batch_normalization(x, reuse=reuse, name=name+'bn')
  x = tf.layers.dropout(x, self.dropout_rate, name=name+'dr')
  return x
 def _siamese_model(self, input_tensor, reuse):
  with tf.name_scope('conv_blocks'):
   x = self.__conv_block(input_tensor, 32, reuse=reuse, name='conv_block_1_')
   x = self.__conv_block(x, 64, reuse=reuse, name='conv_block_2_')
   x = self.__conv_block(x, 128, reuse=reuse, name='conv_block_3_')
```

```
  x = tf.layers.flatten(x)

  with tf.name_scope('fc_blocks'):
    x = self.__fc_block(x, 512, reuse=reuse, name='fc_block_1_')
  output = tf.layers.dense(x,128, activation=tf.nn.sigmoid, reuse=reuse, name='output')
  return output
 def build(self):
  tf.reset_default_graph()
  with tf.name_scope('inputs'):
    self.anchor = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28, 1],
name='anchor_ph')
    self.positive = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28, 1],
name='positive_ph')
    self.negative = tf.placeholder(dtype=tf.float32, shape=[None, 28, 28, 1],
name='negative_ph')
    self.lr = tf.placeholder_with_default(0.0002, [])
    self.dropout_rate = tf.placeholder_with_default(1., [])

  with tf.name_scope('siamese'):
    self.anchor_output = self._siamese_model(self.anchor, False)
    self.positive_output = self._siamese_model(self.positive, True)
    self.negative_output = self._siamese_model(self.negative, True)

  with tf.name_scope('distances'):
    self.positive_distance = self._euclidean_distance(self.anchor_output,
self.positive_output, 'positive_distance')
    self.negative_distance = self._euclidean_distance(self.anchor_output,
self.negative_output, 'negative_distance')

  with tf.name_scope('train'):
    self.loss = self._triplet_loss(self.positive_distance, self.negative_distance)
    self.accuracy = tf.div( (tf.reduce_sum(tf.cast(self.positive_distance<0.5, tf.float32))
+tf.reduce_sum(tf.cast(self.negative_distance>0.5, tf.float32)) )
```

```
                    , tf.cast(tf.shape(self.positive_distance)[0], tf.float32)*2.)
    self.train = tf.train.AdamOptimizer(learning_rate=self.lr).minimize(self.loss)

 def _eval(self, loader):
    total_loss=[]
    total_acc=[]
    for step in range(len(loader)):
      a, p, n = loader.get()
      _loss, acc, = self.sess.run([self.loss, self.accuracy], feed_dict={self.anchor: a, self.positive: p, self.negative:n})
      total_loss.append(_loss)
      total_acc.append(acc)
    loss = sum(total_loss)/len(total_loss)
    acc = sum(total_acc)/len(total_acc)
    return (loss, acc)

 def evaluate(self, data):
   return self.sess.run(self.anchor_output, feed_dict={self.anchor:data})

 def train_model(self, epochs=EPOCHS, lr=0.0002, decay=1.):
   self.sess = tf.Session()
   self.sess.run(tf.global_variables_initializer())
   self.sess.run(tf.local_variables_initializer())
   for epoch in range(epochs):
    total_train_loss=[]
    total_train_acc=[]
    for step in range(len(self.train_loader)):
      a, p, n = self.train_loader.get()
      _, _loss, _acc, = self.sess.run([self.train, self.loss, self.accuracy], feed_dict={self.anchor: a, self.positive: p, self.negative:n ,self.lr:lr, self.dropout_rate:DROPOUT_RATE})
      total_train_loss.append(_loss)
      total_train_acc.append(_acc)
```

```
    lr *= decay
    val_loss, val_acc = self._eval(self.val_loader)
    print('epoch:',epoch+1,'  train loss:', sum(total_train_loss)/len(total_train_loss), ' train acc:',sum(total_train_acc)/len(total_train_acc) ,'  val loss:', val_loss, ' val acc:',val_acc, '  lr:', lr)

  test_loss, test_acc = self._eval(self.test_loader)
  print('test loss:', test_loss, ' test acc:',test_acc )

train = TrainTriplet((train_loader, val_loader, test_loader))

train.train_model(3, 0.0002, 0.95)

nodes = [n.name for n in tf.get_default_graph().as_graph_def().node]
for node in nodes:
 print(node)

from google.colab import drive
 drive.mount('/content/gdrive')

def save(self):
  saver = tf.train.Saver()
  saver.save(self.sess,'./tensorflowModel.ckpt')
  tf.train.write_graph(self.sess.graph.as_graph_def(), '.', 'tensorflowModel.pbtxt', as_text=True)
  from tensorflow.python.tools import freeze_graph
  freeze_graph.freeze_graph('tensorflowModel.pbtxt', "", False,
                    './tensorflowModel.ckpt', "siamese/output/Sigmoid",
                     "save/restore_all", "save/Const:0",
                     'frozentensorflowModel.pb', True, ""
                    )
  inputGraph = tf.GraphDef()
```

```
  with tf.gfile.Open('frozentensorflowModel.pb', "rb") as f:
    data2read = f.read()
  inputGraph.ParseFromString(data2read)
  f = tf.gfile.FastGFile('/content/gdrive/My Drive/Graph.pb', "w")
  f.write(inputGraph.SerializeToString())
  from tensorflow.python.tools import optimize_for_inference_lib
  outputGraph = optimize_for_inference_lib.optimize_for_inference(
        inputGraph,
        ["inputs/anchor_ph"], # an array of the input node(s)
        ["siamese/output/Sigmoid"], # an array of output nodes
        tf.float32.as_datatype_enum)

  # Save the optimized graph'test.pb'
  f = tf.gfile.FastGFile('/content/gdrive/My Drive/OptimizedGraph.pb', "w")
  f.write(outputGraph.SerializeToString())

save(train)

def euclidean_distance(first_input, second_input):
 return np.sum(np.square(first_input-second_input), axis=1, keepdims=True)#tf.sqrt(,
name=name)

a, p, n = test_loader.get()
a = a[:8]
p = p[:4]
n = n[4:8]
pn = np.concatenate((p,n), axis=0)
enc_a = train.evaluate(a)
enc_pn = train.evaluate(pn)
dist = euclidean_distance(enc_a, enc_pn).reshape(-1)

fig, m_axs = plt.subplots(2, a.shape[0], figsize = (12, 4))
for c_a, c_b, c_d, (ax1, ax2) in zip(a, pn, dist, m_axs.T):
```

```
ax1.imshow(c_a[:,:,0])
ax1.set_title('Image A')
ax1.axis('off')
ax2.imshow(c_b[:,:,0])
ax2.set_title('Image B\n%3.0f%%' % (c_d*100))
ax2.axis('off')
```

ДОДАТОК В

WEB-система прискореного пошуку зображень за допомогою нейроних мереж

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ АПЕПС_ТВ42103_18Б 13-1

Аркушів 10

Київ 2019

АНОТАЦІЯ

Додаток містить опис web-системи прискореного пошуку зображень за допомогою нейронних мереж, що виконує деякі із завдань, поставлених в розділі 1, а саме:

Для досягнення мети були вирішені наступні задачі:

- Проведено попереднє завантаження зображення на сервер.
- Розробка нейронної мережі для кодування зображень.
- Збереження та оптимізація нейронної мережі для використання в Web-системі.
- Використання математичних алгоритмів для порівняння закодованих зображень.
- Пошук відповідного зображення в базі даних та відображення результатів.

Додаток розроблений за допомогою мови програмування Python з використанням технології Django у середовищі програмування Visual Studio Code.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис основного компоненту web-системи прискореного пошуку зображень за допомогою нейронних мереж, що виконує деякі із завдань, поставлених в розділі 1. У додатку Б міститься програмний код компоненту.

Система для роботи потребує браузер, що підтримує JavaScript та Cookies.

Додаток розроблений за допомогою мови програмування Python з використанням технології Django у середовищі програмування Visual Studio Code.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Додаток надає функціонал для швидкого пошуку зображень по базі даних за допомогою сіамської нейронної мережі та перегляду історії пошуку.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з архітектурою, додаток складається з трьох головних модулів:

— шар контроллеру;

— шар бізнес логіки;

— шар зв’язку з базою даних.

Шар контроллеру містить в собі методи для взаємодії с користувачем.

Шар бізнес логіки містить в собі класи, що використовуються для моделювання сутностей, якими оперує додаток.

Шар зв’язку з базою даних містить класи з методами для створення, видалення та оновлення полів у базі.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання додатку користувач повинен мати персональний комп’ютер або мобільний пристрій з браузером, що підтримує JavaScript, а також підключення до мережі Інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програма не потребує інсталяції і доступна на сайті за посиланням http://localhost:8080/home/.

ВХІДНІ ДАНІ

Вхідна інформація для додатку:

1) дані для створення індивідуального облікового запису;
2) дані для обробки.

ВИХІДНІ ДАНІ

Вихідна інформація додатку - зображення з бази даних та інформація, що належить до цього зображення;