МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
« КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО »
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2023р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: « Інструментальні засоби розпізнавання дорожніх знаків»

Виконала:
студентка IV курсу, групи ТВ-з91
Плачинда Маргарита Володимирівна _____

Керівник:
Доцент, к.е.н., _____
Гусєва Ірина Ігорівна

Рецензент:
Доцент, к.е.н.,
Сірий Олександр Анатолійович _____

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань
Студентка_____

Київ 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«____»_____ 2023р.

ЗАВДАННЯ
на дипломну роботу студенту
Плачинди Маргарити Володимирівні

1.Тема роботи Інструментальні засоби розпізнавання дорожніх знаків

Керівник роботи Гусєва Ірина Ігорівна, доцент, к.е.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «31» травня 2023 року № 2089-с
2.Строк подання студентом роботи
3.Вихідні дані до роботи: мова програмування Python, середовище розробки Pycharm, середовище розробки Google Colaboraty
4.Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) розробка моделі нейронної мережі, як інструментального засобу розпізнавання дорожніх знаків; розробка застосунку для роботи з моделями нейронної мережі та розпізнавання дорожніх знаків.
5.Перелік ілюстративного матеріалу: схеми аналогічних систем, алгоритми роботи нейронних мереж, візуалізація даних, що використовуються у застосунку, схеми застосунку, концептуальна модель, алгоритм роботи застосунку.
6.Дата видачі завдання « » _____ 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1.	Отримання завдання	30.09.2022	Виконано
2.	Дослідження предметної області	02.10.2022 - 16.04.2023	Виконано
3.	Постановка вимог до проектування системи	02.10.2022-16.04.2023	Виконано
4.	Дослідження існуючих рішень	02.10.2022-16.04.2023	Виконано
5.	Розробка програмного продукту	17.04.2023-21.05.2023	Виконано
6.	Захист	17.04.2023- 21.05.2023	Виконано
7.	Тестування програмного продукту	15.05.2023- 19.05.2023	Виконано
8.	Оформлення дипломної роботи	22.05.2023-04.06.2022	Виконано
9.	Передзахист	05.06.2022-11.06.2022	Виконано
10.	Захист	19.06.2023-23.06.2023	Виконано

Студент _____ Маргарита Плачинда
(підпис) (ім’я, прізвище)

Керівник роботи _____ Ірина Гусєва
(підпис) (ім’я,прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи: робота містить 63 сторінки, 36 рисунків, 1 таблицю, 2 додатки та 20 посилань.

У даній роботі досліджені алгоритми для створення систем розпізнавання дорожніх знаків, що працюють на основі згорткових нейронних мереж.

Метою роботи є створення інструментального засобу для розпізнавання зображень дорожніх знаків в веб-застосунку, а саме:

- провести аналіз існуючих алгоритмів, які здатні знаходити ознаки на різних зображеннях дорожніх знаків;
- проаналізувати дані які будуть навчати нейронні мережі, які є інструментом створення системи;
- проаналізувати переваги використання вхідних даних в форматі csv;
- кластеризувати наявні дані для створення навчальних даних;
- розробити алгоритм розпізнавання дорожніх знаків;
- розробити застосунок, що дозволяє користувачеві розпізнати зображення дорожнього знаку;
- протестувати розроблену систему у реальних умовах.

Практичне значення одержаних результатів полягає в створенні моделі та методів системи класифікації зображень, а саме дорожніх знаків. Створення методів, що дають можливість застосунку порівнювати роботу різних нейронних мереж та методів прогнозування результатів.

Для досягнення поставленої мети було розроблено архітектуру двох нейронних мереж та веб-застосунок, що надає можливість протестувати роботу нейронної мережі на поданих користувачем.

Ключові слова: згорткова нейронна мережа, веб-застосунок, розпізнавання дорожніх знаків, модель мережі, згортка, архітектура мережі.

ABSTRACT

Structure and scope of the thesis: The work contains 65 pages, 36 figures, 1 table, 2 appendices and 20 references.

Algorithms for creating traffic sign recognition systems based on convolutional neural networks are investigated in this work.

The purpose of the work is to create a tool for recognizing road sign images in a convenient web application, namely:

- analyze existing algorithms that are able to find signs onvarious images of road signs;
- analyze the data that will train existing neural networkssystem creation tool;
- analyze the advantages of using input data in csv format;
- cluster existing data to create training data;
- develop an algorithm for recognizing road signs;
- develop an application that allows the user to recognize image of a road sign;
- test the developed system in real conditions.

The practical significance of the obtained results lies in the creation of a model and methods of the image classification system, namely road signs. Creation of methods that enable the application to compare the work of different neural networks and methods of predicting results.

To achieve the goal, the architecture of two neural networks and web applications was developed, which provides an opportunity to test the operation of the neural network on user input.

Keywords: convolution neural netword, web application, traffic sign recognition recognition, network model, convolution, network architecture.

ЗМІСТ

ВСТУП

Сьогодні фотографії та відео вважаються основним візуальним способом сприйняття інформації. Багато наукових відкриттів були можливі саме завдяки спостеріганню. Легкість відтворення та зберігання інформації на матеріальному носії, надає можливості, які дозволяють зберігати велетенські набори даних, що в подальшому зможуть допомогти людству досліджувати навколишнє середовище, а й отримувати багато нової інформації без безпосередньої участі. Збирання таких наборів даних є дуже важливою частиною багатьох наукових досліджень, що в подальшому дозволяє розробникам на основі цих даних створювати та навчати нейронні мережі, що застосовуються в таких сферах як медицина, наука, економіка і багатьох інших. Вклад штучного інтелекту в дані сфери є безцінний, оскільки допомагає людині у розвитку різних сфер наукової діяльності.

В даній роботі розглядається класифікація дорожніх знаків, що здійснюється з використання технології згорткових нейронних мереж, оскільки згорткові мережі вже зарекомендували себе в багатьох проектах, як технологія, що якісно реалізує задачі комп'ютерного зору.

Згорткова нейронна мережа представляє собою спеціальну структуру штучної нейронної мережі, ідея якої запозичена у людини. Використовуючи особливості роботи оглядової кори головного мозку, яка реагує на певні лінії того, що бачить людина під різним кутом активуються набори простих клітин. Це наштовхнуло науковців спробувати повторити роботу людського мозку створивши комп'ютерну модель. Згорткові мережі мають дуже високу здатність до розпізнавання та виділення великої кількості ознак на зображеннях. Вони переходять від конкретних особливостей зображення до більш абстрактних деталей та на кожному етапі збільшують цю абстрактність, що надає можливість до виділення патернів зображення на більш високому рівні. Згорткові мережі є стійкими до змінення

масштабу, ракурсу, поворотам та іншим спотворенням зображення. Це в свою чергу дає можливість використовувати згорткові мережі для розпізнавання зображень в реальному часі, наприклад дорожніх знаків.

Згорткова мережа отримала свою назву завдяки математичній операції згортання, що фактично є множенням зображення на матрицю певного розміру. Згортка або фільтр займаються розпізнаванням певних ознак зображення, переміщуються по зображенню та перевіряють наявність тих чи інших ознак. Сучасні згорткові нейронні мережі є багато етапними. Розробка на основі комп'ютерного зору стосується таких областей, як класифікація та кластеризація. В сучасних нейронних мережах можуть використовувати одразу декілька технологій глибокого машинного навчання. Універсальні системи здатні не лише надавати корисну інформацію, як основну характеристику предметної області, але й бути самобутнім інтелектом. Багато мереж вміють робити певні дії, працювати в графічних застосунках, редагуючи фото, писати коди для програмістів тощо.

Для пошуку візуально подібних зображень застосовують пошук зображень за змістом. Пошук зображень є важливою частиною машинного навчання. Адже якість результату навчання нейронних мереж напряму залежить від якості даних.

Для якісної роботи нейронної мережі розробники віддають багато часу на її тестування. На цьому етапі можна виявити основні проблеми пов'язані з низькою якістю комп'ютерного зору. Після виявлення можливих недоліків розробник допрацьовує алгоритми машинного навчання, аналізує роботу градієнтного спуску мережі, ваги, які використовуються під час навчання, або вплив швидкості навчання на продуктивність моделі.

Призначення програмного забезпечення, що розглядається в цій роботі є розробка веб застосунку для розпізнавання дорожніх знаків та створення інструментів, що реалізують можливість класифікації дорожнього знаку, тестування та перевірки та наборах даних, використовуючи зручний веб інтерфейс. Це надає

користувачеві перевіряти нейронні мережі саме на своїх даних, зображеннях. Такі застосунки направлені на розробку та дослідження того, як впливають технології на основі нейронних мереж на сучасного користувача. В майбутньому це може допомогти збільшити увагу водія за кермом, оскільки автомобіль буде допомагати водієві реагувати на дорожні знаки. Адже згорткові нейронні мережі є стійким до аугментації зображення. Добре навчена мережа спроможна розпізнати дорожній знак навіть коли автомобіль рухається, що суттєво впливає на якість зображення.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

Метою роботи є розробка веб застосунку для розпізнавання дорожніх знаків з використанням згорткових нейронних мереж. Задачі, які потрібно розв'язати для досягнення мети:

- провести аналіз існуючих алгоритмів, які здатні знаходити ознаки на різних зображеннях дорожніх знаків;
- проаналізувати дані які будуть навчати нейронні мережі, які є інструментом створення системи;
- проаналізувати переваги використання вхідних даних в форматі csv (comma-separated value);
- кластеризувати наявні дані для створення навчальної та тестової вибірки даних;
- розробити алгоритм розпізнавання дорожніх знаків;
- розробити застосунок, що дозволяє користувачеві розпізнати зображення дорожнього знаку;
- протестувати розроблену систему у реальних умовах.

В якості інструментального засобу систем розпізнавання дорожніх знаків було обрано згорткові нейронні мережі, які переважно використовуються для вирішення задач комп'ютерного зору. Згорткові нейронні мережі дозволяють отримати високу якість класифікації зображень і є основним інструментом, що використовується в даній галузі.

Завданням нейронної мережі є виділення та розпізнання дорожнього знаку на зображенні. Перед початком розробки потрібно визначитися з такими питаннями:

- хто буде нашим користувачем;

- які вхідні дані ми будемо використовувати;
- в якій формі дані будуть подаватися на вхід нашій системі;
- який вихідний результат очікується від роботи системи.

Застосунок призначений для тестування роботи нейронної мережі. Підійде тим хто вивчає нейронні мережі, займається наукою про дані або може бути використана для водіїв та учнів автошкіл.

Вхідні дані поділяються на дві групи:

- набір зображень дорожніх знаків, що потрібні для навчання системи;
- дорожні знаки, що потрібно подавати на вхід системи під час використання застосунку.

Користувач повинен мати можливість додавати в застосунок зображення дорожнього знаку. Потрібно вирішити задачу додавання вхідних даних до системи користувачем та створити засоби обробки цих даних, що дозволить отримати якісний результат роботи системи розпізнавання дорожніх знаків.

Результат розпізнавання дорожніх знаків завантажених користувачем повертає відповідь системи розпізнавання дорожніх знаків.

Висновки до розділю 1

У розділі розглянуто мету роботи та визначено задачі розробки системи розпізнавання дорожніх знаків. До основних задач відноситься: аналіз існуючих алгоритмів, що використовуються в системах розпізнавання дорожніх знаків, створення та групування навчальних та тестових вибірок даних, що використовуються в нашій системі, розробка алгоритму розпізнавання дорожніх знаків для даної системи, розробка застосунку для розпізнавання знаків. В цьому розділі представлені інструментальні засоби створення систем розпізнавання, якими користуються розробники, а саме згорткові нейронні мережі.

2 АНАЛІЗ ЗАСОБІВ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

2.1 Опис штучних нейронних мереж

Згорткові нейронні мережі базуються на універсальних математичних операціях. За останні 10 років певної кількості нових алгоритмів та архітектури згорткових нейронних мереж.

Розглянемо загальну структуру нейрону. Нейрон це - елемент, що обчислює вихідний сигнал за певними правилами, базуючись на поданих вхідних сигналах. Між собою нейрони можуть бути з’єднані абсолютно по різному, це буде залежати від вибору структури нейронної мережі. Сигнали, що подаються на вхід, проходять по внутрішнім шарам мережі та формують вихідний сигнал або сигнали.

На рисунку 2.1 зображено структуру штучного нейрону, з великої кількості яких і буде складатися повноцінна штучна нейронна мережа.

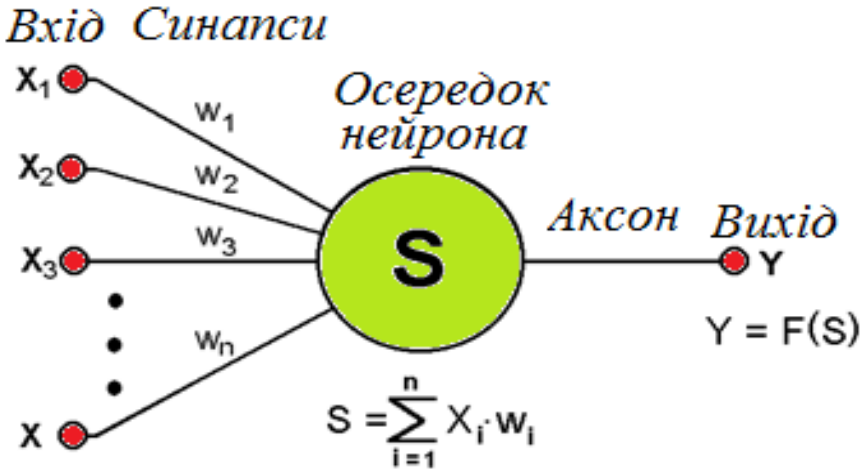

Рисунок 2.1 - Штучний нейрон

Архітектури нейронних мереж поділяються на повно зв'язкові нейронні мережі та багатошарові нейронні мережі. Оскільки для задач розпізнавання використовуються згорткові нейронні мережі, які відносяться до багатошарових нейронних мереж, розглянемо їх структуру.

В багатошарових нейронних мережах нейрони об’єднуються в шари в яких, кожен шар містить сукупність нейронів з єдиним вхідним сигналом. Число нейронів в кожному шарі може бути різним та не залежить від числа нейронів в інших шарах. Крім вхідного та вихідного шару нейронна мережа містить довільну кількість внутрішніх шарів. Згорткова мережа, що є формою багатошарової мережі складається з таких основних частин:

- згорткова частина;

- повно зв'язна частина.

Згорткова частина складається з операції згортки та максимального пулу та виступає в якості модифікатора ознак. Поділяється на одномірні, двомірні, трьох мірні та інколи п’яти мірна згортка. Найчастіше в згорткових нейронних мережах використовується двомірна згортка. Ядро згортки представляє собою матрицю вагів по проходить по двомірному представленню поданого зображення та по елементно виконує операцію множення над тією частиною матриці зображення на якій зараз знаходиться. Після чого перетворює в один вихідний піксель. Ядро рухається по матриці з вказаним кроком. Таким чином ядро розміром 3 x 3 згорне зображення розміром 7 x 7 до розміру 5 x 5 за умови, що крок дорівнює 1 x 1(по вертикалі та горизонталі). Вибір розміру ядра та кроку залишається за розробником. Карти ознак, які ми отримуємо на виході є зваженою сумою, де ваги є значенням самого ядра.

На рисунку 2.2 зображено приклад згортки фільтром 3 x 3 зображення розміром 5 x 5.

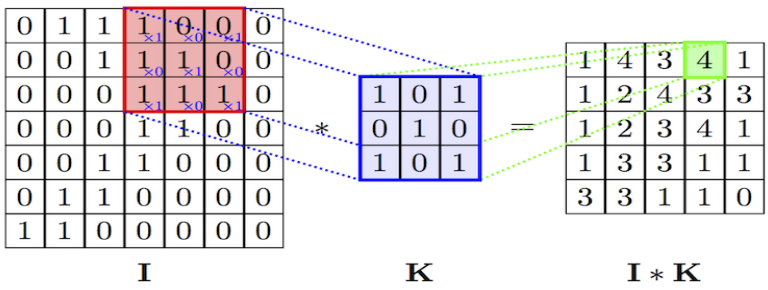

Рисунок 2.2 - Операція згортання

В наведеному прикладі зображення 7 х 7 = 49 ознак на вході та ядро згортки 3 х 3 = 9 ознак на виході, при стандартному перемноженню ми б отримали матрицю 49 х 9 = 441 параметр. Але оскільки ми проводимо не лише стандартне математичне перемноження, а операцію згортання, це дозволяє проводити множення з усіма 9-ма параметрами матриці. Кожна ознака, яку ми отримуємо на виході є результатом аналізу однієї вхідної ознаки, яка знаходиться в приблизно тому ж місці, де знаходилось ядро. Для більшого розуміння розглянемо формулу операції згортання (формула 2.1)

$$\Sigma = xiyj, \tag{2.1}$$

де x - це вхідне значення;

y - це значення ядра згортки.

Операцію згортання можна назвати приблизною версією вхідних даних в зменшеному вигляді. Оскільки після операції згортання вихідні данні зменшуються в розмірі розглянемо пов’язану зі згорткою операцію заповнення (padding). В процесі згортання края нашої початкової матриці зрізаються, через що крайні пікселі ніколи не потраплять в центр ядра. Така проблема вирішується за допомогою заповнення крайніх пікселей значеннями рівними нулю. Завдяки цьому простору навкруги початкової матриці ядро отримує можливість обійти всі краєві пікселі.

На рисунку 2.3 зображена візуалізація процедури заповнення, яка забезпечує можливість ядру згортки захопити крайній піксель матриці зображення.

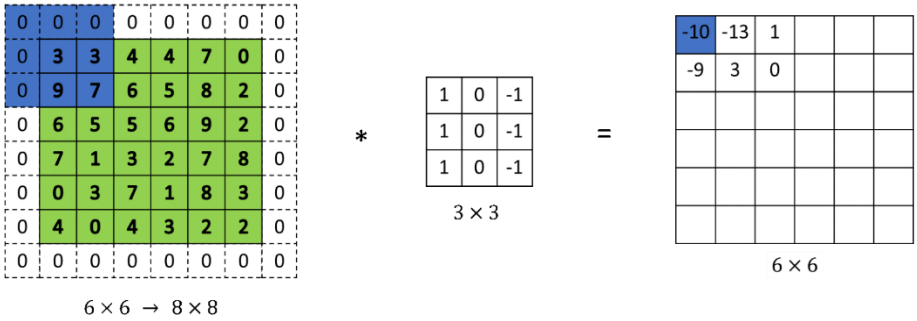

Рисунок 2.3 - Операція заповнення(padding)

Наступною задачею згорткового шару є зменшення розміру матриці. Для цього використовується операція об’єднання(pooling), що по своїй структурі нагадую решітку для просіювання. Існує декілька різновидів об’єднання: максимальне, мінімальне, середнє. Найчастіше використовується максимальне об’єднання. Обирається фільтр довільного розміру, але не дуже великий, найчастіше 2 х 2, який проходиться по вхідній матриці з встановленим кроком(наприклад 2). Якщо це максимальне об’єднання то з вказаних там значень обирається найбільше. Аналогічно з мінімальним об’єднанням. В випадку середнього об’єднання знаходиться середнє арифметичне значення.

Якби нейронна мережа покладалася б лише на вихідну карту ознак об’єктів, її здатність знаходження об’єктів залежала б виключно від їх місця розташування на карті ознак. Використовуючи об’єднання мережа створює більш загальну карту, яка лише вказує на присутність того чи іншого патерну на конкретній частині зображення. З кожним наступним шаром карта стискається зберігаючи лише найважливішу інформацію про наявність певних ознак на об’єкті. Найбільш корисною операція об’єднання є, коли задачею нейронної мережі є класифікація зображення на якому потрібно знайти об’єкт класифікації, але немає значення де він розташований.

На рисунку 2.4 проілюстрована операція максимального об’єднання, яке використовується безпосередньо в моделі нейронної мережі, що розглядатиметься в даній роботі.

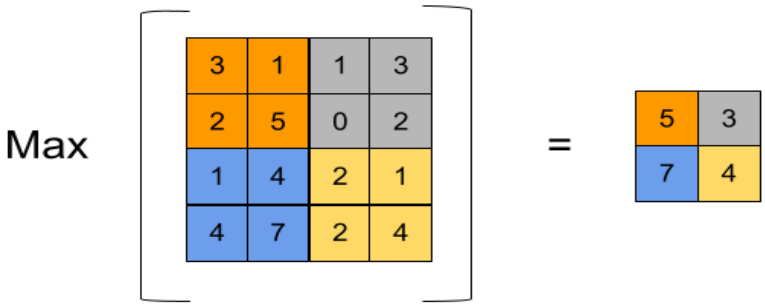

Рисунок 2.4 - Операція максимального об’єднання

Всі описані раніше операції використовуються як для одно канальних, так і для багатоканальних зображень. Вданій роботі розглядається система, що обробляє саме багатоканальні зображення. До багатоканальних зображень відносяться зображення, які мають три канали (RGB: Red, Green, Blue), тобто кольорові зображення. Чим більше внутрішніх шарів буде мати нейронна мережа, тим більшу кількість каналів вона здатна обробляти. Тепер вхідна матриця зображення та ядро фільтру мають глибину що дорівнює 3, фактично цих фільтрів тепер 3. Кожен з них генерує лише один вихідний канал. Цей канал подається до наступного шару мережі.

На рисунку 2.5 представлено багатоканальну матрицю вхідного зображення та фільтру згортки.

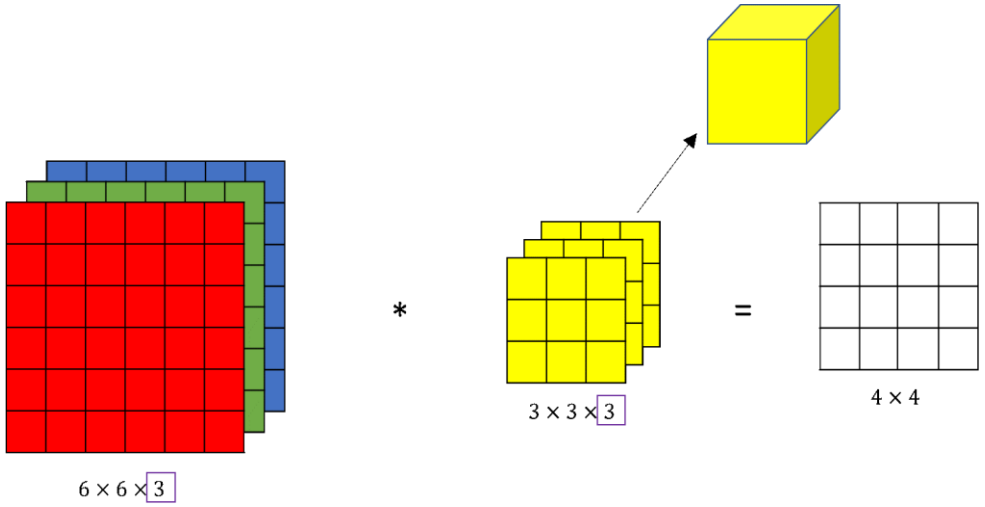

Рисунок 2.5 - Згортка багатоканального зображення

Кожне ядро, яке входить в склад фільтру переміщується вздовж відповідного для цього фільтру вхідного каналу. Ядра різних каналів можуть мати різні ваги. Так наприклад ядро, що відповідає за червоний колір має більшу вагу та може переважати. Наприкінці роботи даної операції результат роботи кожного з фільтрів додаються один до одного для отримання єдиного каналу. Для більш якісного

результату цієї операції також застосовується зміщення, що підживлює кожен результат вихідного фільтру новим значенням вагів.

Наприкінці кожного шару згортки застосовується функція активації. Карти ознак передаються в функцію активації. Ця операція додає в нашу систему нелінійності. В моделі, що розглядається в даній роботі використовується функція Relu (Rectified linear unit). Іншими словами її називають функцією випрямлення, що повертає 0 якщо вхідне значення менше, тобто ця функція завжди повертає позитивне значення. Ця функція використовується в згорткових нейронних мережах для здільшення нелінійності наявного набору даних. Видалення від'ємних значень з вхідних сигналів нейронів сприяє тому, що черні пікселі видаляються із нашого зображення та замінюються сірими. Розглянемо формулу функції активації Relu (формула 2.2):

$$F(x) = max\ (0, x) \qquad (2.2)$$

де x - вихідне значення згорткового шару.

Перевагами функції активації Relu є: більша ефективність в порівнянні з сигмоїдною та тангентною функціями, пришвидшує збіжність градієнтного спуску до глобального мінімума функції втрат.

На рисунку 2.6 зображено графік функції активації Relu.

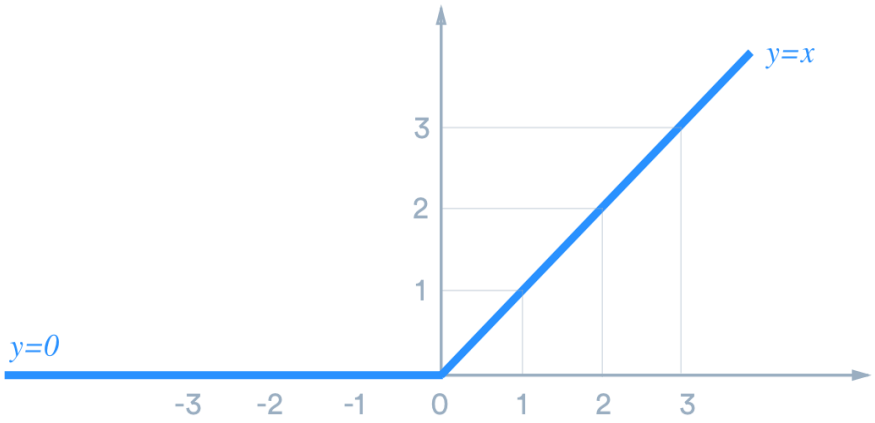

Рисунок 2.6 - Функція активації Relu

Розглянемо повно зв'язкову частину нейронної мережі, що йде після згорткової. Перед поданням даних до цього шару вони вирівнюються в вектор. В даному шарі здійснюється обробка кожного елементу попереднього шару виконуючи матричне перемноження кожного елементу зі своїми вагами, після чого відправляються до наступного шару. Саме тут відбувається навчання нейронної мережі. Кількість шарів повно зв’язної частини встановлює розробник.

Після кожного шару також використовується функція активації, яка в моделі, що розглядається в цій роботі, буде така ж як і в згортковому шарі. Це функція Relu. Також в повно зв’язному шарі використовується параметри нормалізації для збільшення швидкості навчання та збільшення його стабільності за рахунок стабілізації вхідних даних шляхом централізації та масштабування.

На рисунку 2.7 зображене візуальне представлення архітектури повно зв’язкового шару.

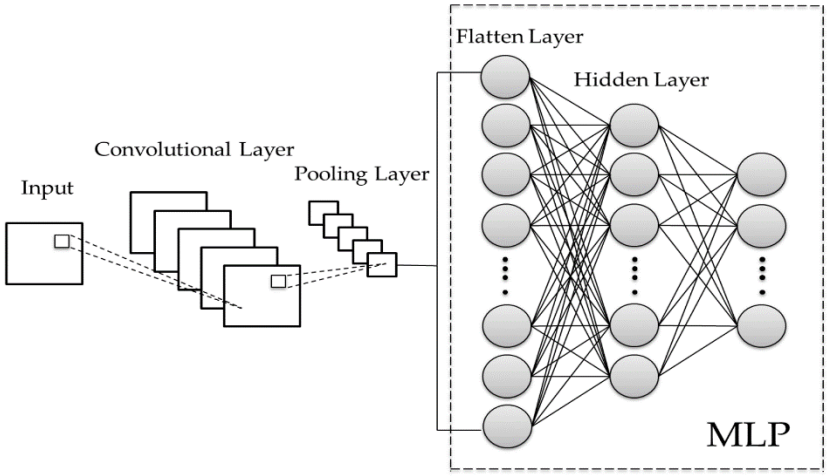

Рисунок 2.7 - Модель повно зв’язної нейронної мережі

На виході нейронної мережі нашої моделі маємо 43 виходи, що відповідають 43 класам класифікації наших дорожніх знаків. До вихідних даних застосовуємо функцію активації softmax. Ця функція перетворює числа в ймовірності, виходом є

вектор ймовірностей кожного можливого результату. Розглянемо формулу функції softmax (формула 2.3):

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.3}$$

Де $s\left(x_i\right)$ - стандартна експонентна функція, що використовується до кожного елементу вхідного вектору;

$\sum_{j=1}^{n} e^{x_j}$ - гарантує, що всі вихідні значення функції будуть в сумі рівній одному і кожне з них буде знаходитись в діапазоні (0, 1), таким чином створюючи дійсне поширення ймовірностей [8, 9].

2.2 Існуючі алгоритми навчання згорткових нейронних мереж

Розглянемо найпопулярніші та найефективніші алгоритми та архітектури завдяки яким можна вирішити задачі класифікації зображень дорожніх знаків. Навчання нейронної мережі представляє собою процес в якому параметри мережі налаштовуються шляхом моделювання даних, які використовуються в конкретній мережі. Вибір типу навчання зумовлений шляхом налаштування параметрів. Є два основних методи навчання:

- навчання з вчителем;
- навчання без вчителя.

Розглянемо методи навчання з вчителем. Коли мережа отримає нове зображення, вона порівняє його з даними на яких навчалася та спробує класифікувати його. Кожен навчальний зразок подається на вхід та проходить через внутрішні шари обробки, обчислюються вихідні дані мережі та порівнюються з цільовим вектором, що представляє собою очікувані виходи нейронної мережі. Далі

потрібно обчислити розмір помилки отриманих вихідних даних, та зробити зміни вагових коефіцієнтів в середині повно зв’язних шарів, які відбуваються залежно від вибору алгоритму. Навчання з вчителем використовується для двох різновидів задач: класифікація та регресія.

Розглянемо навчання без вчителя. При навчанні без вчителя модель отримує на вхід навчальну вибірку без конкретної вказівки, що з нею робити. Вибірка представляє собою лише множину вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб отримати узгодження вхідних векторів. Мережа намагається знайти та проаналізувати певні ознаки зображення. Процес навчання виділяє особливості навчальних векторів та групує їх. Задачі, що може вирішити модель навчання без вчителя це: кластеризація, асоціації, знаходження аномалій, автоенкодери.

Перша більш менш ефективна мережа, що була здатна класифікувати отримані дані було VGG. Модель VGG - це глибока нейронна мережа, яка показала найсучасніші результати в конкурсі ImageNet Large Scale Visual Recognition Challenge в 2014 році і широко використовується як зразок для завдань класифікації зображень та виявлення об'єктів.

В якості вхідних даних модель отримує на вхід RGB зображення розміром 224 x 224. Реалізовано три повно зв’язних шари з яких перші два мають розмір 4096, а наступний 1000 каналів на виході.

На рисунку 2.8 зображена архітектура моделі VGG 19.

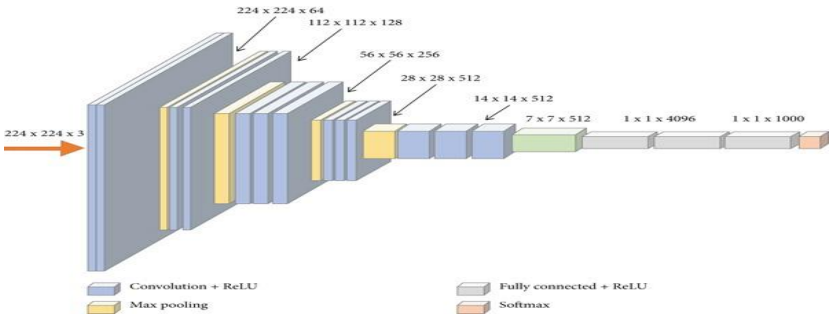

Рисунок 2.8 - Модель VGG – 19

У 2012 році AlexNet виграла конкурс ImageNet Large Scale Visual Rcognition Challenge (ILSVRC) із значною межею зниження похибки порівняно з попередньою сучасною моделлю – до 15% із 26%. AlexNet складається з п'яти згорткових шарів з максимальним об'єднанням і трьох шарів FC, у яких скрізь використовуються функції активації ReLU. Крім фільтрів 3×3, він також має згорткові фільтри 5×5 та 11×11 мережі використовується метод відсіву, щоб зменшити переоснащення.

На рисунку 2.9 зображено архітектуру AlexNet. Світло жовтий колір вкажує на операцію згортки, а темний на використання функції Relu.

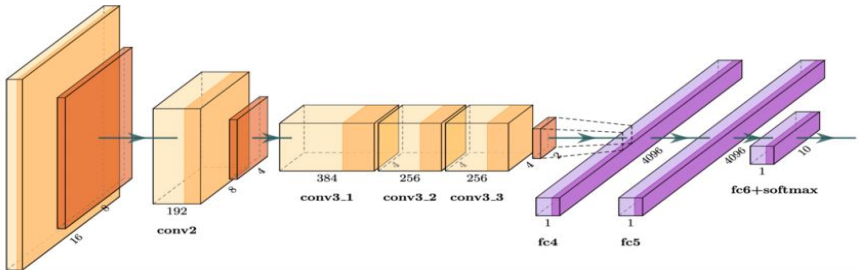

Рисунок 2.9 - Архітектура AlexNet

ResNet - це сімейство глибоких згорткових нейронних мереж, призначених для подолання проблеми зникаючих градієнтів. Ідея ResNet полягає у використанні «залишкових блоків», які забезпечують пряме поширення градієнтів через мережу, що дозволяє навчати дуже глибокі мережі.

На рисунку 2.10 відображається ідея підсилювання вагів. До наступного шару додається результат попереднього. Таке значення буде подаватися на вхід.

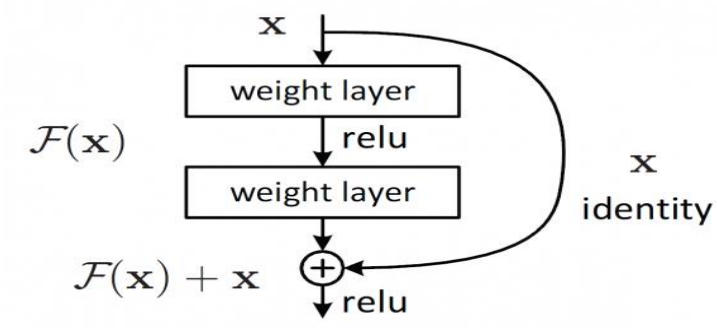

Рисунок 2.10 - Архітектура моделі підсилювання вагів

Залишковий блок складається з двох або більше згорткових шарів, за якими слідує функція активації в поєднанні з максимальним з'єднанням(maxpooling), яке обходить згорткові шари і додає вихідне значення до виходу згорткового шару. Це дозволяє мережі вивчати залишкові функції, які представляють різницю між вхідними та вихідними даними згорткових шарів, замість того, щоб намагатися вивчити всі відображення безпосередньо. Використання залишкових блоків дозволяє навчити дуже глибокі мережі з десятками шарів, що значно полегшує проблему затухання градієнту.

На рисунку 2.11 зображено архітектуру ResNet, що складається з 18 шарів. Розмір вхідного зображення складає 224 x 224.

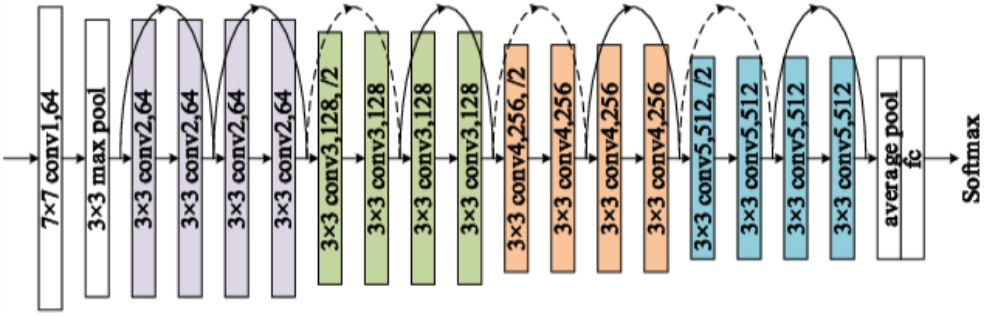

Рисунок 2.11 - Архітектура ResNet18

GoogLeNet – це глибока крутна нейронна мережа, розроблена дослідниками oogle. Він був представлений у 2014 році і того ж року виграв ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) з частотою помилок 6,67% у п'ятірці найкращих. GoogLeNet примітний тим, що використовує модуль Inception, який складається з декількох паралельних згорткових шарів з різними розмірами фільтрів, за якими слідує шар об'єднання та об'єднання вихідних даних.

GoogleLeNet використовує згортку 1 x 1 та об'єднання глобальних середніх значень. Початкова архітектура використовує згортку 1 x 1 для зменшення кількості параметрів (вагів та зміщень) архітектури. Зменшуючи параметри збільшується глибина моделі.

На рисунку 2.12 зображено архітектуру даної моделі. Мережа також включає допоміжні класифікатори на проміжних рівнях, які спонукають мережу вивчати більше відмітних ознак та запобігають перенавчанню.

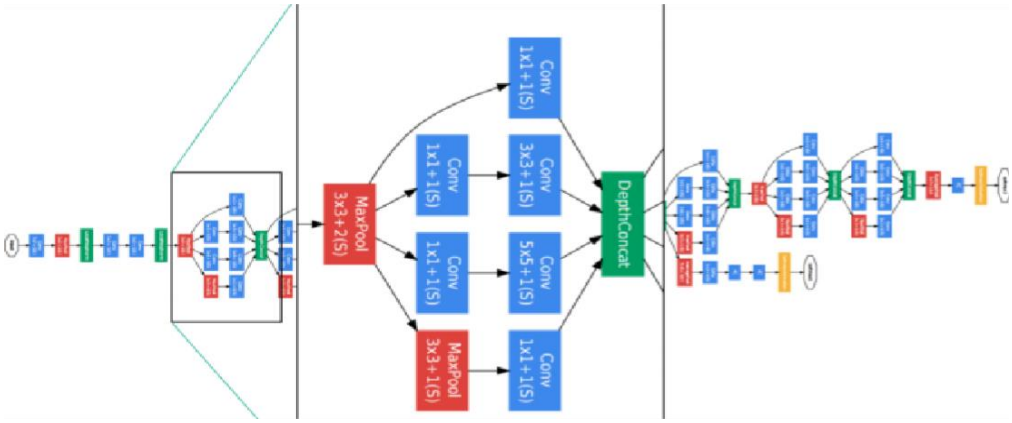

Рисунок 2.12 - Архітектура GoogLeNet

GoogLeNet ґрунтується на ідеях попередніх згорткових нейронних мереж, включаючи LeNet, яка була однією з перших успішних програм глибокого навчання в комп'ютерному зорі. Однак GoogLeNet набагато глибший і складніший, ніж LeNet [1, 9, 13].

2.3 Опис аналогів

Одним із аналогів стала YOLOV4, найкраща модель виявлення об’єктів у реальному часі. YOLOV4 просуває багато дослідних розробок сімейства YOLO, а також нові методи моделювання та збільшення даних. Набір даних, який використовується в модулі виявлення цієї роботи є набором даних German Traffic Sign Detection Benchmark (GTSDB), він складається з 39251 зображень розміром 1360 × 800 пікселів, збережених у форматі pmg. Мітки цих зображень також були зібрані у файл CSV, який містив ім'я файлу, клас зображення, а також інформацію про справжнє розташування фактичного дорожнього знака на зображенні.

Ще однією відомою системою розпізнавання дорожніх знаків є система від компанії Sygic, що створили одне з найпопулярніших додатків з gps навігацією. В даному застосунку використовується смартфон користувача. Система використовує камеру смартфону для виявлення знаків, що обмежують швидкість. Мета - це вивід на смартфон інформації про дорожні знаки на дорозі та розширення можливостей навігаційної системи.

На рисунку 2.13 зображено архітектуру системи розпізнавання YOLOV4.

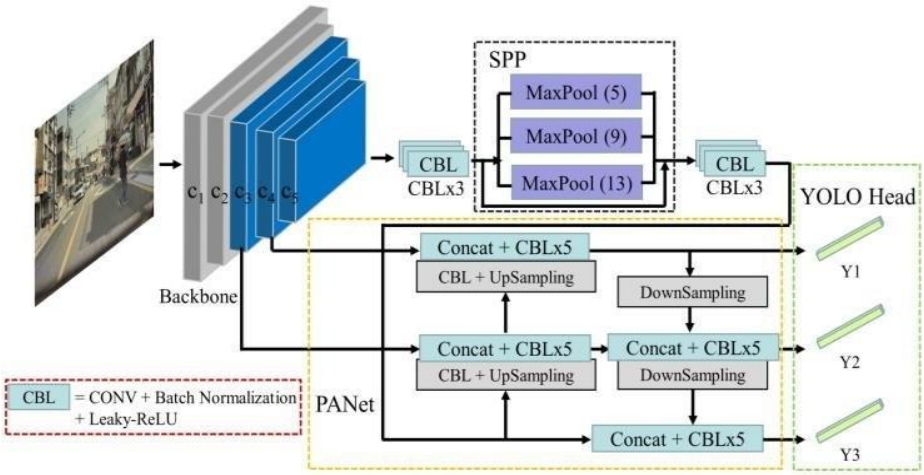

Рисунок 2.13 - Архітектура YOLOV4

Загальна структура YOLOV4 включає CSPDarknet (магістраль), SPPnet, PANet и 3 головки YOLO. Модифіковану частину (C3-C5) знаходиться в основі моделі та виділена синім кольором. Faster R-CNN - це глибока мережа, що використовується для виявлення об’єктів. Дана модель складається з двох частин: повністю згорткової нейронної мережі, яка називається мережею пропозицій регіонів (RPN), яка пропонує блоки регіонів, і наступного модуля — детектора, який класифікує об'єкт. У першому модулі є повно зв’язкова нейронна мережа. Архітектура згорткової нейронної мережі одна для обох мереж. Мережа регіональних пропозицій приймає карту об’єктів згортки, створену магістральним шаром в якості вхідних даних та виводять прив’язку згенеровану шляхом згортання наскрізного вікна. Мережа

знаходження об’єктів шар об’єднання Rol. В даному об’єднанні беруться вхідні дані згенеровані в мережі регіональних пропозицій. Такий рівень об’єднання генерує вхідні дані розміром 7 x 7 x D, де D = 256 в випадку використання ZF архітектури для згорткових шарів та 512 для VGG16.

На рисунку 2.14 зображено архітектуру моделі багато масштабного знаходження об’єктів Faster R-CNN.

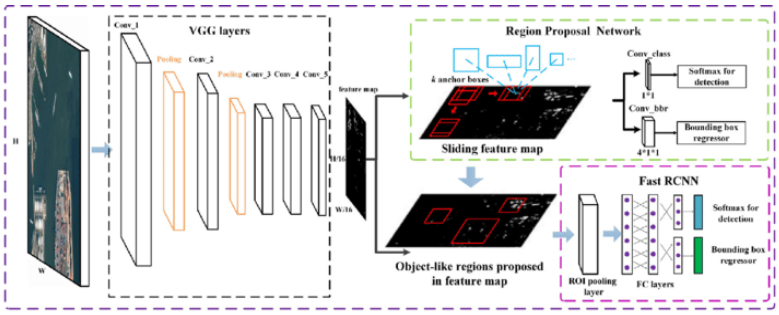

Рисунок 2.14 - Архітектура Faster R-CNN

Ця CNN витягуватиме основну функцію зображення, на виході цієї CNN буде карта функцій. Потім карта функцій передається на інший мережний рівень, пропозиції регіону, який відповідає за пропозицію потенційних регіонів, де можуть бути нові об'єкти, які треба віднести бо класу та конкретного регіону [8, 14].

Висновки до 2 розділу

В другому розділі розглянуто:

-існуючі нейронні мережі, що можуть розпізнавати дорожні знаки;

-алгоритми навчання згорткових нейронних мереж;

-методи, що використовуються під час навчання та тренування мережі;

-аналогічні системи зі схожою архітектурою.

В розділі описуються такі алгоритми, як навчання з вчителем та навчання без вчителя. Показано для яких задач ліпше використовувати кожен алгоритм.

3 ЗАСОБИ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

3.1 Обґрунтування вибору мови програмування

Для даної роботи була вибрана мова програмування Python. Цю мову прийнято вважати одним із лідерів цієї галузі, через свою поширеність, популярність і величезну кількість бібліотек, що володіють набором вбудованих математичних функцій для вирішення завдань всередині нейронної мережі.

Бібліотеки Python легко справляються з задачею обробки даних. Є багато вбудованих алгоритмів машинного навчання, які можна використовувати, як повністю самостійно, так і доповнити власними функціями, що сприяють зменшенню помилки навчання нейронної мережі та підвищення якості. Мова Python дозволяє створювати нейронні мережі досить швидко та має багато інструментів, що дозволяють створити не лише мережу, але й гарний bek-end програмного забезпечення, який використовує дану мережу. Також дана мова володіє механізмом інтероперабельності з мовами C та C++, таким чином забезпечуючи повну взаємодію та функціональність з іншими системами. Використовуються бібліотеки мови Python Tensorflow Keras. Завдяки Sequential API та Models легко створюються моделі машинного навчання. Також використовується бібліотека matplotlib завдяки якій виводяться графіки. Це дозволяє на різних етапах розробки вивести графіки навчальних даних і оцінити роботу нейронної мережі[16].

3.2 Обґрунтування вибору середовища розробки

Створення системи класифікації зображень відбувалася в системі Google Colab, бекендна частина та графічний інтерфейс створені в середовищі PyCharm.

Colab - це розміщений Jupyter, вже встановлений і налаштований, тому не потрібно нічого робити на комп'ютері, а просто працювати в браузері з ресурсами в хмарі.

Для створення застосунку можна використовувати, як вбудовані бібліотеки, так і ті що треба локально встановити в віртуальному оточенні. Бібліотеки TensorFlow, Keras, Streamlit та Pillow встановлюємо через термінал PyCharm. Також імпортуємо бібліотеки numpy та pandas. Хмарна платформа надає доступ до графічного процесора GPU та до NVIDIA V100. Для навчання згорткової нейронної мережі використовується 25 гб пам’яті хмарної платформи. Для завантаження навчальних даних використовуємо репозиторій розміщений на github. Середовище дозволяє писати код в окремих блоках, що полегшує тестування помилок. Також в цих блоках можна писати коментарі. Jupyter включає ярлики для загальних команд операційної систему, тому їх можна виконати безпосередньо в блоках блокноту [17].

3.3 Обґрунтування вибору середовища розробки веб частини

PyCharm є IDE для Python. IDE - інтегроване середовище розробки, комплекс програмних засобів, які дозволяють вести зручнішу розробку певною мовою програмування. IDE має зручний текстовий редактор, компілятор чи інтерпретатор, багато зручних налаштувань та інше програмне забезпечення.

PyCharm є більш розширеною та функціональною версією з можливістю розробки в тому числі багатомовних веб-додатків. Має зручний редактор коду з усіма корисними функціями: підсвічуванням синтаксису, автоматичним форматуванням, доповненням та відступами. PyCharm дозволяє перевіряти версії інтерпретатора мови на сумісність та використовувати шаблони коду. Легко створюється віртуальне середовище та наявний графічний інтерфейс для встановлення всіх потрібних бібліотек.

Для створення фронт енд частини веб застосунку обрана бібліотека Streamlit. Вона дозволяє написати всі елементи фронт енд виключно на Python, не використовуючи стандартні html та css файли. Такий спосіб створення веб застосунку є більш швидким та дозволяє створювати застосунки для роботи з нейронними мережами та візуалізацією даних. Бібліотека дозволяє створювати, розгортати та ділитися веб застосунками. Дана бібліотека дозволяє розгорнути програмне забезпечення одразу в браузері, без додаткового встановлення на локальну машину. Як тільки розробник запускає сценарій свого застосунку локальний сервер Streamlit запускається, після чого застосунок відкривається в веб-браузері встановленому за замовчуванням. Для зручного користування користувачів використовуються хмарні сервіси. Для зручності розробки в налаштуваннях встановлюється пункт “завжди оновлювати”, після чого застосунок буде автоматично оновлюватись після будь яких змін в коді програми. Також бібліотека надає багато прикладів макетів з відкритим кодом, що можуть бути корисні при створенні застосунку та навчання. Працює зі всіма компонентами системи LLM.

Використовуючи лямбка функції та метод бібліотеки streamlit write виводиться інформація стосовно структури нейронної мережі, з чього вона складається шар за шаром. Streamlit дозволяє [18,20].

Висновки до розділу 3

В цьому розділі наведено програмні компоненти, які використовуються для створення застосунку. Їх вибір обґрунтовано. Для кожної частини застосунку використовується певне програмне середовище. Для розробки нейронних мереж використовується хмарне середовище, а для веб застосунку використовується середовище локально встановлене на комп’ютері розробника.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура застосунку

Архітектура програмного забезпечення складається з двох частин. Перша це модель нейронної мережі. В даному випадку використовується дві нейронні мережі, та веб частина, що представляє собою інструмент, що допомагає тестувати якість роботи машинного зору конкретної мережі.

Загальна архітектура проекту має певну ієрархію папок, які пов’язані між собою, а також між вхідними даними. Є рівні, які безпосередньо взаємодіють з даними проекту, такими як моделі нейронних мереж та файли з даними.

На рисунку 4.1 представлена діаграма компонентів. Вона дозволяє краще зрозуміти, як влаштована система, що вона використовує в якості джерел даних, як компоненти пов’язані між собою.

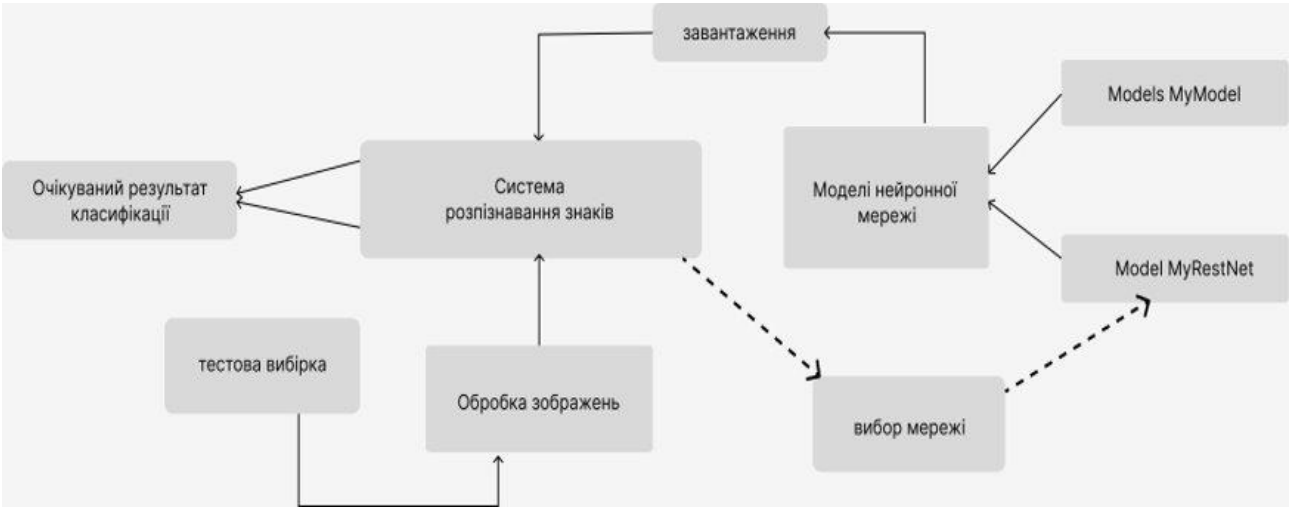

Рисунок 4.1 - Діаграма компонентів системи

Як видно з діаграми, програма використовує декілька можливостей подавати вхідні данні. З цими даними взаємодіють інші структурні компоненти програми, з

яких складається повноцінна система. Одним з них є завантаження даних користувачем. Діаграма показує точку входу даних в систему. Елементи роботи застосунку, що використовуватися:

- вхідні дані для система;
- дані які подає користувач;
- очікуваний відгук системи;
- функціональні можливості.

Для цього створюємо діаграму прецедентів, яка показує, які саме користувачі будуть використовувати застосунок, як буде організована взаємодія користувач-програмне забезпечення.

На рисунку 4.2 зображена схема діаграми прецедентів, що дозволяє зрозуміти, які функції створені саме для використання користувачем та які є зоною розробника.

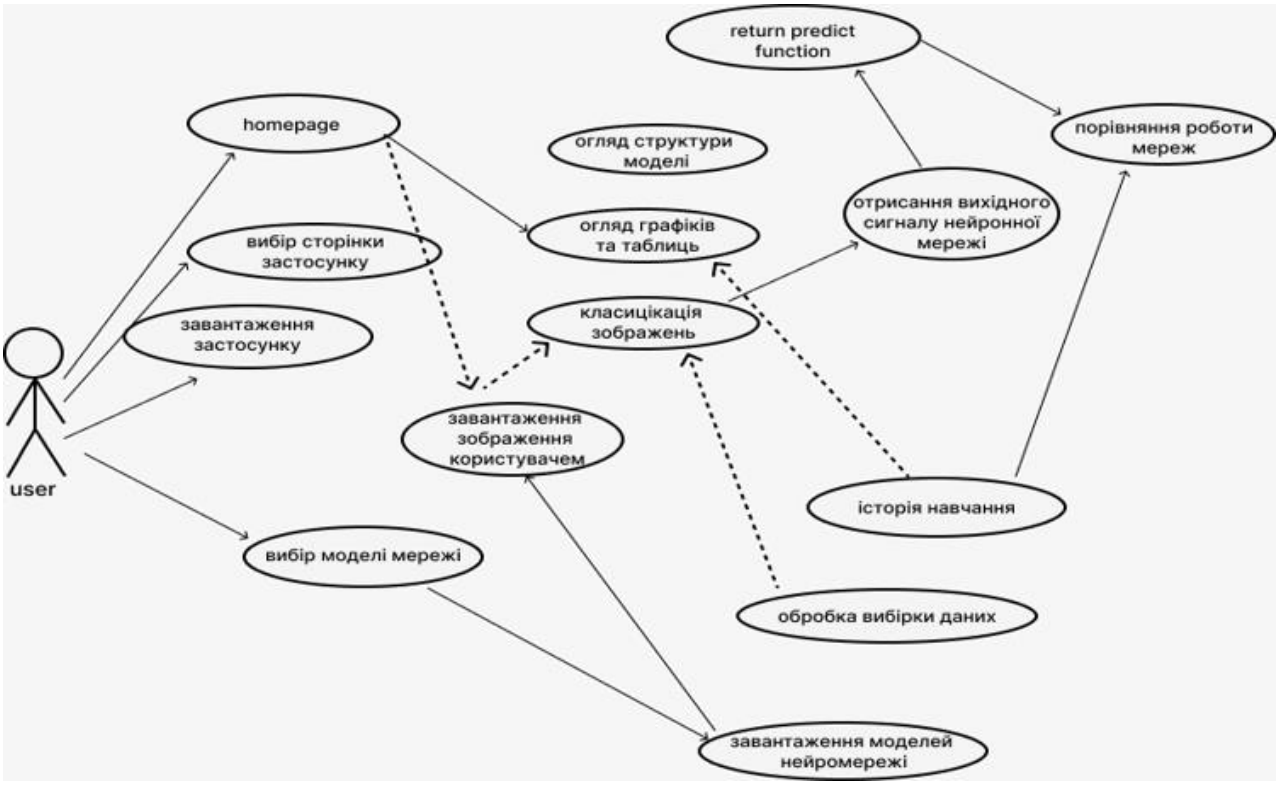

Рисунок 4.2 - Діаграма прецедентів

4.2 Архітектура нейронних мереж

Архітектура першої нейронної мережі складається з 13 шарів. На вхід подається зображення розміром 150 х 150, що згортається фільтром 3 х 3. Далі використовується максимальне об’єднання розміром 2 х 2 та пакетна нормалізація, що підтримує середній вихід даних близьким до 0, а стандартне відхилення близьким до 1. Розмір ядра фільтру залишається незмінним на всіх шарах мережі, збільшується лише кількість фільтрів. На вхід повно зв’язної частини нейронної мережі передається вектор довжиною 512. На виході маємо 43 класи дорожніх знаків та функцію softmax.

Використовуючи можливості keras отримуємо дані, що описують структуру кожного шару нейронної мережі.

В таблиці 1.1 наведено короткий зміст першої архітектури першої моделі згорткової нейронної мережі даного застосунку.

Таблиця 1.1 - Модель sequential

Layer (type)	Output Shape	Param
conv2d (Conv2D)	(None, 148, 148, 16)	448
conv2d_1 (Conv2D)	(None, 146, 146, 32)	4640
max_pooling2d(MaxPooling2D	(None, 73, 73, 32)	0
batch_normalization(BatchNormalization)	(None, 73, 73, 32)	128
conv2d_2 (Conv2D)	(None, 71, 71, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 34, 34, 128)	0
batch_normalization_1 (Batc hNormalization)	(None, 34, 34, 128)	512

Продовження таблиці 1.1

flatten (Flatten)	(None, 147968)	0
dense (Dense)	(None, 512)	75760128
batch_normalization_2 (Batc hNormalization)	(None, 512)	2048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 43)	22059

Загальна кількість параметрів навчання:

- Total-params:75,882,315

-Trainable-params:75,880,971

-Non-trainable-params:1,344

На рисунку 4.3 зображена перша модель нейронної мережа, що складається з 12 шарів.

Рисунок 4.3 - Графічна модель шарів нейронної мережі

Друга нейронна мережа має структуру подібну до архітектури ResNet. Поділяється на три окремі функції: ідентифікації, згорткової частини та функції, що об’єднує згорткову та повно зв’язну частини. Дана реалізація базується на методі трансферного навчання. Такий метод сприяє тому, що модель глибокої нейронної мережі використовує знання отримані при вирішенні однієї задачі, для вирішення іншої, але пов’язаної з нею задачею. Це пришвидшує процес навчання глибокої

нейронної мережі. Після шару нормалізації на вхід функції активації подається вихідне значення шару нормалізацію з попереднім значенням, підсилюючи вагові коефіцієнти в глибокій нейронній мережі.

На рисунку 4.4 ми бачимо структуру другої моделі нашої нейронної мережі, що складається з 50 шарів та 3 окремих класів.

Рисунок 4.4 - Зображена структура другої моделі

Загальна кількість параметрів навчання:

- Total params: 25,723,563
- Trainable params: 25,670,443
- Non-trainable params: 53,120

4.2 Модуль обробки даних

Обробка даних складається з обробки існуючої вибірки даних. GTSRB - німецький стандарт розпізнавання дорожніх знаків. Складається з 43 класів дорожніх знаків, поділений на 39209 навчальних зображень та 12 603 тестових зображень. Зображення мають різну систему освітлення та різнобарвний фон. Дані розміщені в github та клоновані в код нейронної мережі.

Навчальна вибірка ділиться на train та validation вибірки. На вхід цій фунції подається декілька csv файлів, перший відповідає за навчальну вибірку, другий за тестову. Дані файлі треба прочитати та знайти назву потрібного нам стовпчика, що відповідає за назву класу зображення та його назву. Після цього до назви стовпчика можна звернутися, як до словника, використовуючи колонку, як ключ та отримати всі наявні там дані. Використовуючи лямбда функцію додаємо до потрібної нам

директорії назву зображення з csv файлу та створюємо дані з новою назвою в словнику використовуючи бібліотеку random застосовуємо функцію, що вибирає із послідовності значення встановленої довжини, таким чином отримуючи поділ вхідних даних на навчальну та перевіркову вибірки випадковим чином. Після цього передаємо дані в функцію, що обробляє їх перетворюючи на матрицю. Встановлюємо заданий розмір всім зображенням вибірки та ділемо кожен елемент зображення на 255, що надає можливість тримати всі вхідні дані в діапазоні від 0 до 1. Такі самі операції повторюємо з тестовою вибіркою.

До навчальної вибірки застосовується аугментація даних. Операція аугментації фактично є спотворення зображень з метою розширення вибірки даних та створення нових екземплярів зображень на не схожі на попередні. Аугментацію здійснюється за допомогою бібліотеки Keras та класу ImageDataGeneration.

Далі вказуємо параметр batch_size, щоб встановити кількість зображень в обному пакеті, а також параметр seed, що є випадковим початковим числом, яке дозволяє змішувати порядок зображень.

На рисунку 4.5 зображено графік, що показую різниці між кількістю даних в кожному класі, яких в застосунку 43.

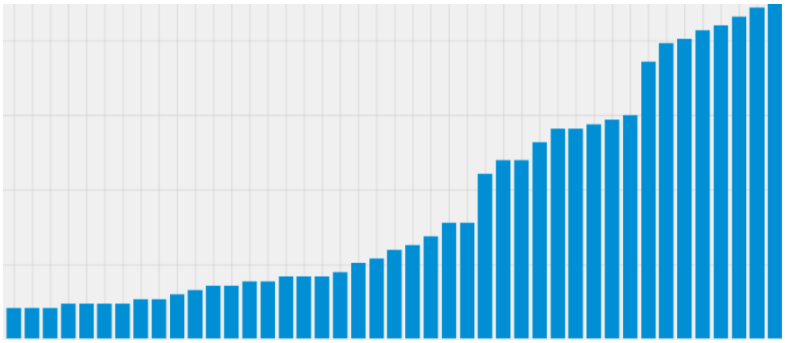

Рисунок 4.5 - Графік класів вибірки даних

Далі вказуємо параметр batch_size, щоб встановити кількість зображень в обному пакеті, а також параметр seed, що є випадковим початковим числом, яке дозволяє змішувати порядок зображень.

Дані подаються на вхід у вигляді масивів numpy. Важливо пам’ятати, що зображення мусять бути змішані обов'язково, оскільки якщо на початку на вхід нейронної мережі подавалося зображення обмеження швидкості в 20 кілометрів, але більше в процесі навчання таких зображень не буде мережа в кінці навчання не буде пам’ятати ці зображення.

На рисунку 4.6 зображено структуру csv файлу, що використовувався для створення навчальної вибірки.

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	27	26	5	5	22	20	20	Train/20/00020_00000_00000.png
1	28	27	5	6	23	22	20	Train/20/00020_00000_00001.png
2	29	26	6	5	24	21	20	Train/20/00020_00000_00002.png
3	28	27	5	6	23	22	20	Train/20/00020_00000_00003.png
4	28	26	5	5	23	21	20	Train/20/00020_00000_00004.png

Рисунок 4.6 - Структура csv файлу

На рисунку 4.7 продемонстрована частина навчальної вибірки даних. Дані зображення максимально близькі до реальних умов. Різне освітлення та якість чіткості зображення дозволяють зробити мережу більш чутливою та виділити достатню кількість ознак потрібних для якісної класифікації зображення.

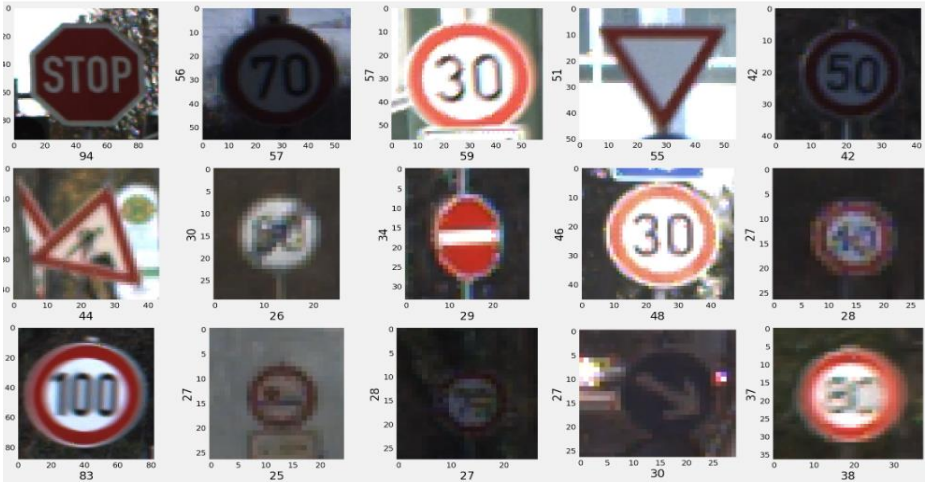

Рисунок 4.7 - Частина даних, на яких навчається нейронна мережа

Вивід зображень відбувається за допомогою циклу і фактично є графіком, де можна задати будь яку кількість зображень. Графік показує, що дані мають різну якість. Імітується сканування зображення в момент руху автомобіля. Освітлення залежить від часу доби. Такий підхід максимально наближений до реальних умов.

4.4 Алгоритм роботи веб застосунку

Алгоритм роботи застосунку базується на надані можливості користувачеві отримувати доступ до класифікації зображень, інформації про нейронну мережу, користуючись зручним графічним інтерфейсом.

Алгоритм полягає в наступному:

- завантаження застосунку;
- завантаження вхідних даних;
- вибір моделей нейронної мережі;
- запуск функції класифікації зображення;
- отримання вихідних результатів.

На рисунку 4.8 зображена модель веб застосунку. В даній моделі представлена сторінка Homepage, де користувач може перейти на сторінку з потрібним йому функціоналом, або відкрити її в іншій вкладці браузера.

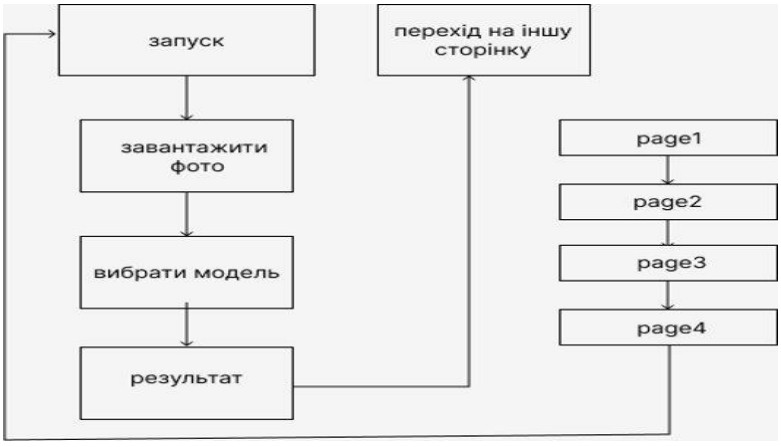

Рисунок 4.8 - Загальна логічна модель веб додатку

Функція, що обробляє зображення приводить зображення до потрібного розміру та формує із зображення масив, який передається в модель. Дана модель є вибраною користувачем застосунку системи розпізнавання дорожніх знаків.

Друга функція безпосередньо реалізує результат класифікації, мережа повертає назву класу до якого відноситься зображення. Для цього створюється словник, що містить в собі значення ключів у вигляді назви класу дорожнього знаку. Значення ключа є вихідним значенням функції класифікації.

Процес класифікації зображення забезпечують чотири функції. А саме: •

-функція завантаження зображення дорожнього знаку в застосунок;

-функція вибору моделі нейронної мережі;

-функція обробки png, jpeg зображень;

-функція класифікації зображення, що використовує механізми keras.

На рисунку 4.9 зображено послідовність роботи функції, що обробляє зображення додане до застосунку користувачем та повертає результат класифікації зображення.

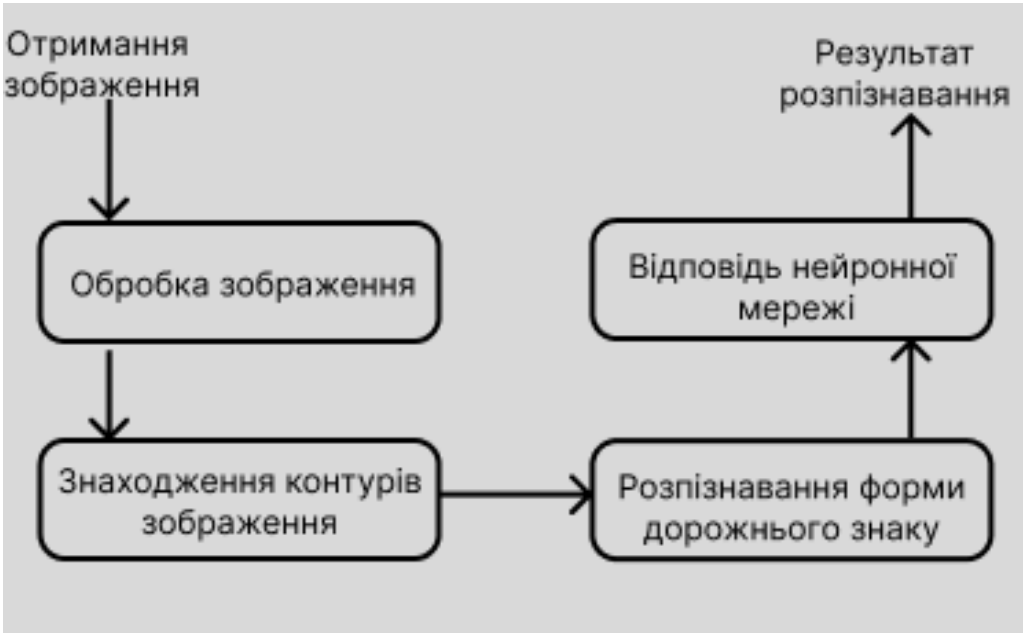

Рисунок 4.9 - Обробка зображення

Висновки до розділу 4

В даному розділі розглянуто програмну реалізацію, а саме: алгоритми, які безпосередньо використовувалися для створення застосунку, архітектуру нейронних мереж, методи обробки навчальних даних та ефективні способи їх використання. Розглянуто структуру csv файлу та методи роботи з ним. Наведено графіки структури нейронних мереж, логічну модель веб застосунку, схему обробки зображення.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

5.1 Встановлення застосунку

Бажана операційна система - Windows. Пристрій повинен мати як мінімум 150 МБ вільної внутрішньої пам'яті та мінімум 8 ГБ оперативної пам’яті. Також клієнт повинен мати доступ до інтернету. Технічні характеристики комп'ютера soft SQL Server, PostgreSQL або Oracle Database, адже застосунок не є важким.

Користувач отримує посилання, яке дає можливість користуватися застосунком в веб браузері, без локальної установки на комп’ютер. За необхідністю доступ може надаватися тільки певним користувачам або бути доступний по ключу, або бути у вільному доступі на хмарній платформі.

5.2 Запуск застосунку та робота з ним

Користувачам може працювати з застосунком після отримання посилання, або встановивши його локально на свій комп’ютер. Користувач має можливість обрати потрібну йому сторінку застосунку та перейти на неї або відкрити її в новій сторінці браузера. Якщо користувач обирає сторінку Classification, він потрапляє на сторінку класифікації зображень дорожніх знаків. Перехід на наступну сторінку можна зробити натиснувши на неї або відкривши в новому вікні браузера. Окрім домашньої сторінки є ще 4 сторінки застосунку, на які можна переходити. Сторінка класифікації зображень дорожніх знаків містить функціонал, який дозволяє завантажувати зображення з власного комп’ютера. Формат файлів мусить бути png або jpeg, таке обмеження задане розробником. Користувач має можливість діагностувати якість класифікації дорожніх знаків оскільки він є не обмеженим в виборі даних, що завантажує.

На рисунку 5.1 візуалізовано функція завантаження зображення в дії. Застосунок надає можливість використовувати файли з комп’ютера користувача.

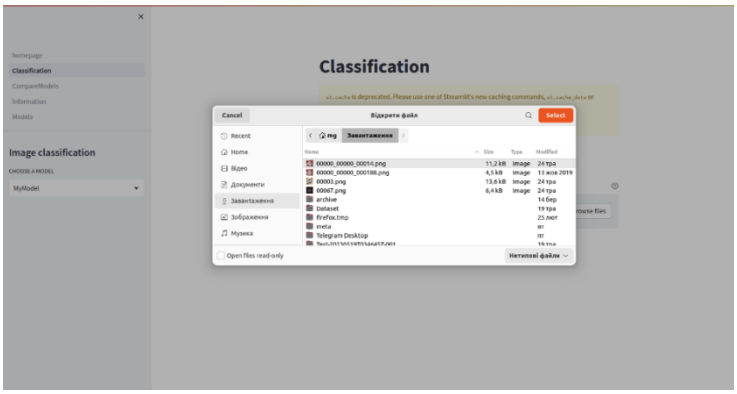

Рисунок 5.1 - Вікно завантаження зображення

Для отримання результату користувач повинен обрати модель нейронної мережі із списку приведених на екрані комп’ютера та натиснути на кнопку. В цей момент запускається функція класифікації дорожнього знаку.

Користувач має можливість додавати власні зображення до застосунку у форматі png або jpeg. Користувач може діагностувати якість класифікації, оскільки він є не обмеженим в виборі даних, що завантажує. Зображення може бути подане в будь якому розмірі.

На рисунку 5.2 зображено результат роботи вище описаних функцій. На екрані з’являється попередження про те, до якого класу належить завантажений дорожній знак. Це має вигляд текстового повідомлення.

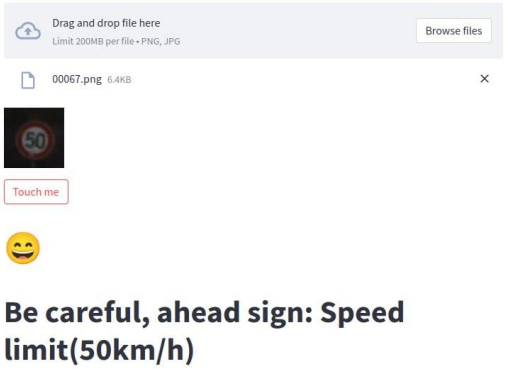

Рисунок 5.2 - Результат класифікації

Сторінка model дозволяє подивитися структуру потрібної моделі, оскільки моделі дві, а структури в них різні, результати роботи нейронних мереж можуть відрізнятися. Для реалізації цього функціоналу створено вікно вибору моделі нейронної мережі та дві кнопки, що відповідають за виведення потрібних даних на екран. Такий функціонал дозволяє користувачеві ознайомитися з шарами мережі та їх послідовністю.

Організована функція, що звертається до збереженої в застосунку моделі нейронної мережі та виводить інформацію про неї на екран.

На рисунку 5.3 зображено інформацію про нейрону мережу, яку хотів побачити користувач.

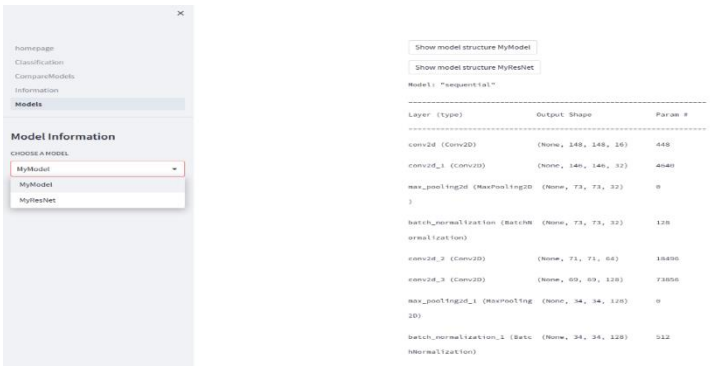

Рисунок 5.3 - Сторінка Models

Даний варіант функціоналу надає можливість продивитися структуру нейронної мережі.

Наступна сторінка це сторінка Information. Вона містить в собі моделі нейронних мереж та шляхи до директорій з потрібними даними, які зберігаються в структурі проекту. Ці дані містять інформацію про історію навчання нейронних мереж, їхні функції втрат та результати навчання. На головному екрані створено чотири кнопки у вигляді маленьких віконець, в які треба ставити пташку. На моніторі з’являється потрібна користувачеві інформація. Якщо пташку прибрати методом натиску на віконце інформація зникає. Кожна кнопка-пташка

містить свій функціонал. Дві з них виводять графіки результатів навчання нейронних мереж, в залежності від того, яку вибере користувач. Ще дві виводять на екран дані історії навчання у вигляді таблиці та містять в собі математичні числа. Таким чином користувач має можливість розглянути графіки, що ілюструють роботу нейронних мереж або ж дослідити таблиці з даними.

На рисунку 5.4 зображено роботу кнопок, що відповідають за виведення графіків.

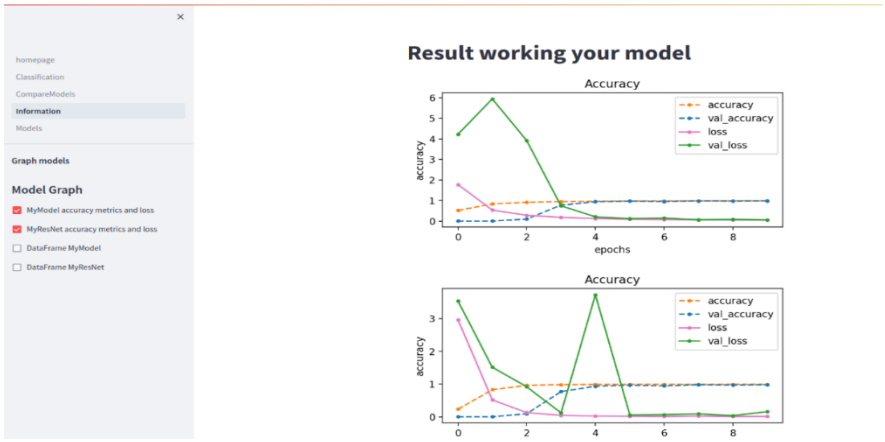

Рисунок 5.4 - Сторінка Information

Дані про результати навчання нейронних мереж представлені у вигляді csv файлу. Потрібні данні, що подальшому передаються в функцію що створює графіки, отримуються звертаючись до них по назві стовпчика. Використовуючи розділ бібліотеку matplotlib pyplot створюються графіки, що візуалізують дані про історію навчання. Історія навчання представляє собою такі класи значень:

- точність навчання;
- втрати навчання;
- точність навчання на перевірній вибірці;
- втрати навчання на перевірній вибірці.

Перш ніж вивести дані про навчання нейронної мережі в таблицю, ці дані треба обробити. Спочатку треба прочитати csv файл, а потім звернутися до даних,

що там знаходяться по ключу, тобто назві стовпчика. Кожна функція викликається лише при умові натискання на кнопку.

На рисунку 5.5 зображено результат роботи сторінки Information та пов’язаних з нею функцій.

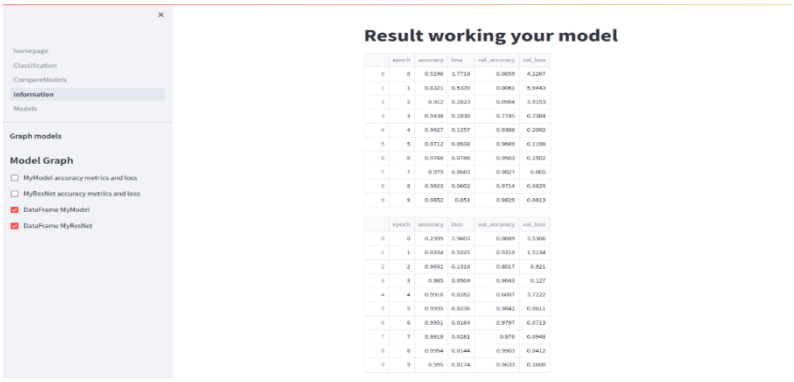

Рисунок 5.5 - Сторінка Information

На сторінці CompareModels представлено функціонал, що дозволяє класифікувати не одне зображення, а тестову вибірку.

Розглянемо кроки роботи сторінки:

- обробка даних тестового зображення;

- створення radio-кнопок;

- функція вибору нейронної мережі;

- функція класифікації зображень.

Програма має дві різні завантажені вибірки даних. Ці дані оброблюються з використання циклів. Кожне зображення змінює розмір та перетворюється на матрицю, що додається в створені заздалегідь масиви. Кнопка представляє собою круглю велику крапку. За замовчуванням завжди вибрана перша кнопка. Імпортуємо з іншого модулю функцію, що обирає нейронну мережу. Вибір здійснюється шляхом натиску на radio-кнопку. Ця кнопка не лише вибирає модель нейронної мережі, але й запускає функцію класицікації зображень. На екрані користувач бачить назви всіх

наведених зображень та має змогу оцінити правильність відповідей застосунку. Також на цій сторінці можна побачити зображення тестової вибірки.

Вивід зображень на екран відбувається у вигляді графіку. Для видалення розмітки графіки використано параметр “off” методу matlabplot.axis.

На рисунку 5.6 представлено класифікацію зображень першої моделі нейронної мережі.

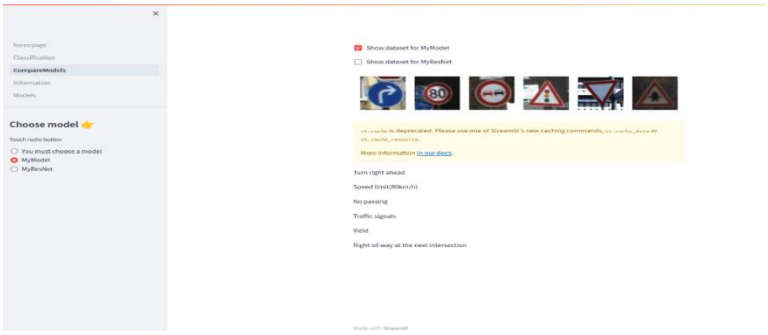

Рисунок 5.6- Результат робота моделі MyModel

Натиснувши на кнопку другий раз тестова вибірка зникає. Після цього можна ознайомитись з наступною вибіркою даних. Картинки зображень можна легко порівняти з назвами класів цих зображень, які з’являються після того, як спрацювала функція класифікації.

На рисунку 5.7 представлена вибірка тестових даних та результат їхньої класифікації моделлю нейронної мережі MyResNet.

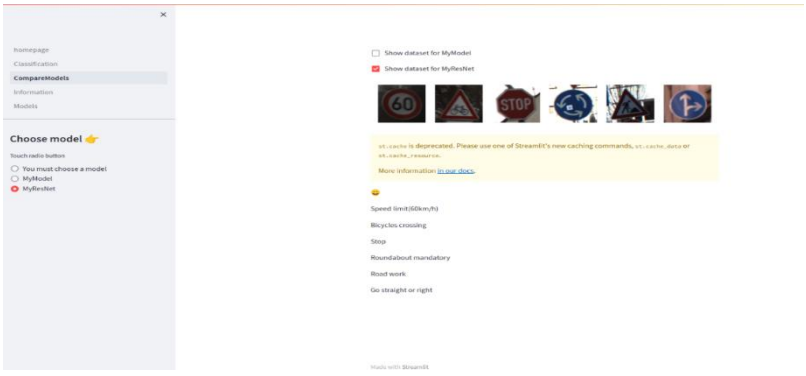

Рисунок 5.7 - Результат роботи моделі MyResNet

Висновки до розділу 5

В даному розділі розглянуто функціонал та можливості веб застосунку на основі згорткових нейронних мереж для розпізнавання дорожніх знаків. Застосунок має простий, але зрозумілий та зручний інтерфейс для користувача. Кнопки організовано так, щоб інші елементи інтерфейсу не заважали користувачеві їх побачити.

ВИСНОВКИ

Під час виконання дипломної роботи було проведено аналіз існуючих алгоритмів, які здатні знаходити ознаки на різних зображеннях дорожніх знаків. Розглянуто алгоритми створення згорткових нейронних мереж, виявлено їх недоліки, пов’язані з ретельним підбором навчальної вибірки та великою кількістю часу, що потрібен для навчання нейронної мережі та перевагами, до яких відноситься виділення нейронною мережею великої кількості ознак на зображенні, що сприяє якісній класифікації поданих зображень.

Проаналізовано дані, які використовуються під час навчання нейронних мереж. Навчальна вибірка повинна містити достатню кількість даних поділених на класи та оброблених перед подачею в нейронну мережу. Обробка містить такі етапи: зміна розміру зображення, приведення зображення до матричного вигляду.

Проаналізовано переваги використання вхідних даних в форматі csv, а саме використання меншого об’єму пам’яті, зручне табличне представлення вибірки даних дозволяє швидко отримати доступ до всіх класів зображення.

Проведено кластеризацію наявних даних для створення навчальної та тестової вибірки даних. В результаті виділено 43 класи набору даних, що складаються з дорожніх знаків.

Розроблено алгоритм розпізнавання дорожніх знаків, який містить такі етапи: завантаження зображень, обробка зображень, класифікація зображення з допомогою згорткової нейронної мережі.

Розроблено застосунок, що дозволяє користувачеві розпізнати зображення дорожнього знаку. Користувач може завантажити як одне зображення, так і множину. Результати містять інформацію, до якого класу відноситься дорожній знак.

Протестовано розроблену систему у реальних умовах, точність розпізнавання приблизно 85 процентів.

Дипломна робота виконана в рамках проекту Визнання подібності зображень загального призначення для гетерогенного застосунку (спільно з Політехнічним інститутом м. Томар, Португалія).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Datagen :Convolutional Neural Network: Benefits, Types, and Applications. URL: https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network.

2. Kaggle. URL: https://www.kaggle.com;

3. Medium: Write your own Custom Data Generator for TensorFlow Keras. URL:https://medium.com/analytics-vidhya/write-your-own-custom-data-generator-for-tensorflow-keras-1252b64e41c3.

4. Keras Implementation of ResNet-50 (Residual Networks) Architecture from Scratch.
URL:https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch.

5. Streamlit Dashboard with Dynamic Plots.
URL:https://shalini043000.medium.com/streamlit-dashboard-with-dynamic-plots-a986866d45d3.

6. On Disharmony in Batch Normalization and Dropout Methods for Early Categorization of Alzheimer's Disease. URL: sustainability-14-14695-v2.pdf.

7. Deep Learning using Rectified Linear Units. Abien Fred M. Agarap, 7 с.

8. Faster R-CNN Explained for Object Detection Tasks.
URL:https://blog.paperspace.com/faster-r-cnn-explained-object-detection

9. CNN Architectures — LeNet, AlexNet, VGG, GoogLeNet and ResNet. URL:https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848.

10. В. В'югин . Математичні основи машинного навчання та прогнозування,
МЦНО, 2014 р. 305 с.

11. David Barber. Bayesian Reasoning and Machine Learning. Cambridge University Press, 2012 p. 735 c.
12. Альберто Боскетти, Бастиан Шарден, Лука Массарон.
Large Scale Machine Learning with Python. Packt Publishing, 2016 p. 420 c.
13. Ethem Mining. Machine Learning: 4 Books in 1. Independently published, 2020. 494 c.
14. Machine Learning for Absolute Beginners: A Plain English Introduction. Independently published, 2017. 167 c.
15.Gaurav Leekha. Learn AI with Python: Explore Machine Learning and Deep Learning techniques for Building Smart AI Systems Using Scikit-Learn, NLTK, NeuroLab, and Keras, BPB Publications, 2021 p. 270 c.
16.Python AI. URL: https://realpython.com/python-ai-neural-network/.
17.Google Colaboratory.
URL:https://colab.research.google.com/#scrollTo=QMMqmdiYMkv.
18.A faster way to build and share data apps. URL:https://streamlit.io.
19.Tyler Richards. Streamlit for Data Science: Create interactive data apps in Python, 2nd Edition, Packt Publishing - ebooks Account, 2023 p. 261 c.
20.Mohammad Khorasani, Mohamed Abdou, Javier Hernández Fernández.
Web Application Development with Streamlit: Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework. Apress, 2022 p. 507 c.

ДОДАТОК А

Інструментальні засоби розпізнавання дорожніх знаків

Текст програми

Аркушів 18

Київ — 2023

MyModel.py

```
!git clone https://github.com/sonu275981/GTSRB---German-Traffic-Sign-Recognition.git
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import random

import tensorflow as tf

from tensorflow import keras

from PIL import Image

from tensorflow.keras.preprocessing import image

from tensorflow.keras.saving import load_model

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications.efficientnet import preprocess_input,
decode_predictions

from keras.utils import load_img, img_to_array, to_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout

from keras.optimizers import Adam, SGD

from keras import Sequential

from sklearn.metrics import accuracy_score

data_dir = '/content/GTSRB---German-Traffic-Sign-Recognition/gtsrb-german-traffic-sign'
train_path='/content/GTSRB---German-Traffic-Sign-Recognition/gtsrb-german-traffic-
sign/Train.csv'
test_path='/content/GTSRB---German-Traffic-Sign-Recognition/gtsrb-german-traffic-
sign/Test.csv'
img_path='/content/GTSRB---German-Traffic-Sign-Recognition/gtsrb-german-traffic-
sign/Test/00234.png'

IMG_HEIGHT = 150
```

```
IMG_WIDTH = 150
IMG_SIZE = (IMG_HEIGHT, IMG_WIDTH)
CHANNELS = 3
train_data_csv = pd.read_csv(train_path)
test_data_csv = pd.read_csv(test_path)
classes = {0:'Speed limit 20km/h',
        1:'Speed limit 30km/h',
        2:'Speed limit 50km/h',
        3:'Speed limit 60km/h',
        4:'Speed limit 70km/h',
        5:'Speed limit 80km/h',
        6:'End of speed limit 80km/h',
        7:'Speed limit 100km/h,
        8:'Speed limit 120km/h',
        9:'No passing',
        10:'No passing for vehicles over 3.5 metric tons',
        11:'Right-of-way at the next intersection',
        12:'Priority road',
        13:'Yield',
        14:'Stop',
        15:'No vehicles',
        16:'Vehicles over 3.5 metric tons prohibted',
        17:'No entry',
        18:'General caution',
        19:'Dangerous curve to the left',
        20:'Dangerous curve to the right',
        21:'Double curve',
        22:'Bumpy road',
        23:'Slippery road',
        24:'Road narrows on the right',
        25:'Road work',
        26:'Traffic signals',
        27:'Pedestrians',
        28:'Children crossing',
        29:'Bicycles crossing',
        30:'Beware of ice/snow',
        31:'Wild animals crossing',
        32:'End of all speed and passing limits',
        33:'Turn right ahead',
        34:'Turn left ahead',
        35:'Ahead only',
        36:'Go straight or right',
        37:'Go straight or left',
        38:'Keep right',
        39:'Keep left',
        40:'Roundabout mandatory',
        41:'End of no passing',
```

```
        42:'End of no passing by vehicles over 3.5 metric tons'}
train_df = train_data_csv[['ClassId', 'Path']]
test_df = test_data_csv[['ClassId', 'Path']]
train_data = train_df.sample(frac=0.9, random_state=0)
valid_data = train_df.drop(train_data.index)
train_df['Path'] = train_df['Path'].apply(lambda x: data_dir + '/' + x)
test_df['Path'] = test_df['Path'].apply(lambda x: data_dir + '/' + x)

def format_dataset(dataframe):
   X_data = []
   y_data = []
   for ind, row in dataframe.iterrows():
      images = load_img(row.Path, target_size=(IMG_HEIGHT, IMG_WIDTH))
      image_array = img_to_array(images) / 255.0
      X_data.append(image_array)
      y_data.append(to_categorical(row.ClassId, num_classes=len(classes)))
   return np.array(X_data), np.array(y_data)
X_train, y_train = format_datset(train_data)
X_validation, y_validation = format_dataset(valid_data)
X_test, y_test = fofmat_dataset(test_df)
model = keras.models.Sequential([
   keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation='relu',
input_shape=(IMG_HEIGHT,IMG_WIDTH,CHANNELS)),
   keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
   keras.layers.MaxPool2D(pool_size=(2, 2)),
   keras.layers.BatchNormalization(axis=-1),
   keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
   keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
   keras.layers.MaxPool2D(pool_size=(2, 2)),
   keras.layers.BatchNormalization(axis=-1),
   keras.layers.Flatten(),
   keras.layers.Dense(512, activation='relu'),
   keras.layers.BatchNormalization(),
   keras.layers.Dropout(rate=0.5),
   keras.layers.Dense(43, activation='softmax')
])
model.summary()
batch_size=256
epochs = 10
lr = 1e-3
desay_lr_rate=lr/epochs
optimizer = Adam(learning_rate=lr, decay=desay_lr_rate)
model.compile(loss='categorical_crossentropy',
            optimizer=optimizer,
            metrics=['accuracy'])

file_name = '/content/drive/MyDrive/log.csv'
```

```
csv_logger = tf.keras.callbacks.CSVLogger(file_name, separator=",", append=False)
generation_data = ImageDataGenerator(
  rotation_range=10,
   zoom_range=0.15,
   width_shift_range=0.1,
   height_shift_range=0.1,
   shear_range=0.15,
   horizontal_flip=False,
   vertical_flip=False,
   fill_mode="nearest")
train_generation = generation_data.flow(
   X_train, y_train,
   batch_size=batch_size,
   seed=42,
   shuffle=True)
validation_generation = generation_data.flow(
   X_validation, y_validation,
   batch_size=batch_size,
   seed=42,
   shuffle=True)
 np.save('history1.npy',history.history)

history1=np.load('history1.npy',allow_pickle='TRUE').item()
score = new_model.evaluate(X_test, y_test, verbose = 0)
x = image.img_to_array(im)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

MyResNet.py

```
def identity_block(X, y, filt, stage, block):
   conv_name_base = 'res' + str(stage) + block + '_branch'
   bn_name_base = 'bn' + str(stage) + block + '_branch'
   Fiilter1, Filter2, Filter3 = filt
   X_shortcut = X
   X = Conv2D(filters=F1, kernel_size=(1, 1),  padding='valid', name=conv2D,
           kernel_initializer=glorot_uniform(seed=0))(X)
   X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
   X = Activation('relu')(X)
   X = Conv2D(filters=F2, kernel_size=(y, y),  padding='same', name=conv2D',
           kernel_initializer=glorot_uniform(seed=0))(X)
   X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
   X = Activation('relu')(X)
   X = Conv2D(filters=F3, kernel_size=(1, 1), padding='valid', name=conv2D',
           kernel_initializer=glorot_uniform(seed=0))(X)
   X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)
   X = Add()([X, X_shortcut])  # SKIP Connection
   X = Activation('relu')(X)
```

```
    return X
def convolutional_func(X, y, filters, stage, block, s=3):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    Filetr1, Filter2, Filter3 = filters
    X_shortcut = X
    X = Conv2D(filters=F1, kernel_size=(1, 1), padding='valid', name=conv2D,
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)
    X = Conv2D(filters=F2, kernel_size=(y, y), padding='same', name=conv2D',
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)
    X = Conv2D(filters=F3, kernel_size=(1, 1), padding='valid', name=conv2D',
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s), padding='valid',
name=conv_name_base + '1',
                        kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
    return X
def MyResNet(input_shape=(100, 100, 3), classes=43):
    X_input = Input(input_shape)
    X = ZeroPadding2D((3, 3))(X_input)
    X = Conv2D(64, (7, 7), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3))(X)
    X = convolutional_block(X, f=3, filters=[64, 64, 256], block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')
    X = convolutional_block(X, f=3, filters=[128, 128, 512], block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
    X = convolutional_block(X, f=3, filters=[256, 256, 1024], block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')
    X = X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
```

```
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')
    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)
    X = Flatten()(X)
    X = Dense(256, activation='relu',
name='fc1',kernel_initializer=glorot_uniform(seed=0))(X)
    X = Dense(128, activation='relu',
name='fc2',kernel_initializer=glorot_uniform(seed=0))(X)
    X = Dense(43, activation='softmax',
name='fc3',kernel_initializer=glorot_uniform(seed=0))(X)
   model = Model(inputs = X_input, outputs = X, name='ResNet50')

return model
file_name = '/content/drive/MyDrive/log_too.csv'
csv_logger = tf.keras.callbacks.CSVLogger(file_name, separator=",", append=False)
model = ResNet50(input_shape = (100, 100, 3), classes = 43)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=256, validation_data=(X_validation,
y_validation), callbacks=[csv_logger])
```

Homepage.py

```
import streamlit as st
st.set_page_config(
    page_title='MultiPageApp',
    page_icon=':smile:'
)
st.title('Welcome to my application ')
st.sidebar.success(
    'Select a page about'
)
st.write('Pump my network')
```

Models.py

```
import tensorflow as tf
import streamlit as st
from PIL import Image
from keras.applications import EfficientNetB0
from keras.utils import load_img, img_to_array, plot_model
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.applications.efficientnet import preprocess_input, decode_predictions
new_model =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0001.h5')

new_model_2 =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0002.h5')

wr_tuple = ('MyModel',
          'MyResNet')
```

```
sidebar_title = st.sidebar.title('Model Information')
sidebar_model_2 = st.sidebar.selectbox(
            'CHOOSE A MODEL', wr_tuple)

@st.cache(allow_output_mutation=True)
def load_model():
    try:
        if sidebar_model_2 == wr_tuple[0]:
            return new_model
        if sidebar_model_2 == wr_tuple[1]:
            return new_model_2
        else:
            print('Model not')
    except ValueError:
        st.title('photo size is wrong')
st.title = (''' # Get information about models''')
button_1 = st.button('Show model structure MyModel')
button_2 = st.button('Show model structure MyResNet')
if button_1 == True:
        st.write(new_model.summary(print_fn=lambda y: st.text(y)))
if button_2 == True:
        st.write(new_model_2.summary(print_fn=lambda y: st.text(y)))
value = load_model()
```

Classification.py

```
import numpy as np
import io
import pandas as pd
import tensorflow as tf
import streamlit as st
from PIL import Image
from keras.applications import EfficientNetB0
from keras.utils import load_img, img_to_array, plot_model
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.applications.efficientnet import preprocess_input, decode_predictions
new_model =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0001.h5')

new_model_2 =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0002.h5')

st.title('Classification')
IMG_HEIGHT = 150
IMG_WIDTH = 150
IMG_HEIGHT_2 = 100
IMG_WIDTH_2 = 100
```

```
wr_tuple = ('MyModel',
            'MyResNet')
sidebar_title = st.sidebar.title('Image classification')
sidebar_model = st.sidebar.selectbox(
   'CHOOSE A MODEL', wr_tuple)
def upload_image():
   uploaded_file = st.file_uploader(type=['png', 'jpg'], help='Hello, download your file',
                                    label='Upload an image for recognition')
   if uploaded_file is not None:
      image_data = uploaded_file.getvalue()
      st.image(image_data)
      return Image.open(io.BytesIO(image_data))
   else:
      return None
@st.cache(allow_output_mutation=True)
def load_model():
   try:
      if sidebar_model == wr_tuple[0]:
         return new_model
      if sidebar_model == wr_tuple[1]:
         return new_model_2
      else:
         print('Model not')
   except ValueError:
      st.title('photo size is wrong')
def format_image(img):
   if model == new_model:
      x_val = img.resize((IMG_HEIGHT, IMG_WIDTH))
   elif model == new_model_2:
      x_val = img.resize((IMG_HEIGHT_2, IMG_WIDTH_2))
   x = img_to_array(x_val)
   x = np.expand_dims(x, axis=0)
   x = preprocess_input(x)
   x = np.vstack([x])
   return
def print_result_classification(model, image):
   val = model.predict(image)
   classification_value = val.argmax(axis=-1)
   a_val = int(classification_value)
   cl = classes[a_val]
   print(type(cl))
   st.write(""" # Be careful, ahead sign: """, cl)
   st.write('**This class does not exist in this neural network**')


model = load_model()
st.title('Image Classification')img = upload_image()
```

```
result_1 = st.button('Touch me')
if result_1 == True:
    x_data = format_image(img)
    st.write(""" # :smile:""")
    print_result_classification(model, x_data)
```

Information.py

```
import streamlit as st
import tensorflow as tf
import os
import io
import numpy as np
import keras
import pandas as pd
import mpld3
import plotly.graph_objects as go
import matplotlib.pyplot as plt
from mpld3 import plugins
import seaborn as sns
from PIL import Image
# from Classification import format_image
from keras.utils import load_img, img_to_array, to_categorical,
image_dataset_from_directory
from sklearn.metrics import accuracy_score
# from Classification import classes
from keras.applications.efficientnet import preprocess_input, decode_predictions
from keras.preprocessing.image import ImageDataGenerator
import streamlit.components.v1 as components
st.title('Information')
st.title('Result working your model')
IMG_HEIGHT = 150
IMG_WIDTH = 150
IMG_HEIGHT_2 = 100
IMG_WIDTH_2 = 100
new_model =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0001.h5')

new_model_2 =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0002.h5')

path = '/home/mg/PycharmProjects/Test_4/TestClass'
data_dir = '/home/mg/PycharmProjects/Test_4/archive/'
# test_dir = '/home/mg/PycharmProjects/Test_4/archive/Test.csv'
# path = '/home/mg/PycharmProjects/Test_4/Gtsrb'
path_csv = '/home/mg/PycharmProjects/Test_4/Test.csv'
test_dir = '/home/mg/PycharmProjects/Test_4/archive/train'
file_csv = '/home/mg/PycharmProjects/Test_4/log.csv'
file_csv_2 = '/home/mg/PycharmProjects/Test_4/log_too.csv'
```

```
x_data = pd.read_csv(file_csv)
x = x_data[['accuracy']]
x_df = x_data[['val_accuracy']]
y = x_data[['loss']]
y_data = x_data[['val_loss']]

x_data_2 = pd.read_csv(file_csv_2)
x_2 = x_data_2[['accuracy']]
x_df_2 = x_data_2[['val_accuracy']]
y_2 = x_data_2[['loss']]
y_data_2 = x_data_2[['val_loss']]


st.sidebar.markdown('### Graph models')
sidebar_title = st.sidebar.title('Model Graph')
with st.sidebar:
        chek = st.checkbox("MyModel accuracy metrics and loss")
        chek_2 = st.checkbox("MyResNet accuracy metrics and loss")
        chek_writer = st.checkbox('DataFrame MyModel')
        chek_writer_2 = st.checkbox('DataFrame MyResNet')
if chek:
    fig = plt.figure(figsize=(6, 6))
    plt.subplot(212)
    plt.plot(x, color='tab:orange', linestyle='--', marker='.', label='accuracy')
    plt.plot(x_df, color='tab:blue', linestyle='--', marker='.', label='val_accuracy')
    plt.plot(y, color='tab:pink', linestyle='-', marker='.', label='loss')
    plt.plot(y_data, color='tab:green', marker='.', label='val_loss')
    plt.title('Accuracy')
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.legend()
    st.pyplot(fig)
   if chek_2:
    fig = plt.figure(figsize=(6, 6))
    plt.subplot(212)
    plt.plot(x_2, color='tab:orange', linestyle='--', marker='.', label='accuracy')
    plt.plot(x_df, color='tab:blue', linestyle='--', marker='.', label='val_accuracy')
    plt.plot(y_2, color='tab:pink', marker='.', label='loss')
    plt.plot(y_data_2, color='tab:green', marker='.', label='val_loss')
    plt.title('Accuracy')
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.legend()
    st.pyplot(fig)
if chek_writer:
    st.write(x_data)
```

```
if chek_writer_2:
   st.write(x_data_2)
```
CompareModels.py
```
iimport cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import streamlit as st
from keras.applications.efficientnet import preprocess_input, decode_predictions
from keras.preprocessing.image import ImageDataGenerator
import streamlit.components.v1 as components
new_model =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0001.h5')

new_model_2 =
tf.keras.models.load_model('/home/mg/PycharmProjects/Test_4/My_Model_Cnn_0002.h5')

data_dir = '/home/mg/PycharmProjects/Test_4/archive'
IMG_HEIGHT = 150
IMG_WIDTH = 150
IMG_HEIGHT_2 = 100
IMG_WIDTH_2 = 100
path = '/home/mg/PycharmProjects/Test_4/TestClass'
path_2 = '/home/mg/PycharmProjects/Test_4/TestClass_2'
wr_tuple = ('MyModel',
          'MyResNet')
data_model_1 = []
data_model_2 = []
test_dir = os.listdir(path)
test_dir_2 = os.listdir(path_2)

for i in test_dir:
   image_path = path + '/' + str(i)
   img = load_img(image_path)
   size_img = img.resize((IMG_HEIGHT, IMG_WIDTH))
   img_array = img_to_array(size_img)/255.0
   data_model_1.append(img_array)
for i in test_dir_2:
   image_path = path_2 + '/' + str(i)
   img = load_img(image_path)
   size_img = img.resize((IMG_HEIGHT_2, IMG_WIDTH_2))
   img_array = img_to_array(size_img) / 255.0
   data_model_2.append(img_array)
data_model_1 = np.array(data_model_1)
data_model_2 = np.array(data_model_2)
col1, col2 = st.columns(2)
st_bar = st.sidebar.title('Choose model 👉')
```

```
with st.sidebar:
    button = st.radio(
        'Touch radio button ',
        ('You must choose a model', 'MyModel', 'MyResNet'),
        key=None, disabled=False
    )

@st.cache(allow_output_mutation=True)
def load_model():
    try:
        if button == 'MyModel':
            return new_model
        if button == 'MyResNet':
            return new_model_2
        else:
            print('Model not')
    except ValueError:
        st.title('photo size is wrong')
mb_2 = new_model_2.predict(data_model_2)
classification_value_2 = mb_2.argmax(axis=-1)
print("pred2=", classification_value_2)
chek = st.checkbox('Show dataset for MyModel')
chek_2 = st.checkbox('Show dataset for MyResNet')
if chek:
    fig = plt.figure(figsize=(6, 6))
    for i in range(6):
        plt.subplot(1, 6, i+1)
        plt.imshow(data_model_1[i])
        plt.axis('off')

    st.pyplot(fig)
if chek_2:
    fig = plt.figure(figsize=(6, 6))
    for i in range(6):
        plt.subplot(1, 6, i + 1)
        plt.imshow(data_model_2[i])
        plt.axis('off')
    st.pyplot(fig)
def classification_dataset(x_dataset):
    md = new_model.predict_on_batch(x_dataset)
    classification_value = md.argmax(axis=-1)
    print("pred=", classification_value)
    print(type(classification_value))
    for data in classification_value:
        cl = classes[data]
        x = data + 1
        st.write(cl)
```

```
def classification_dataset_2(x_dataset):
    md = new_model_2.predict_on_batch(x_dataset)
    classification_value = md.argmax(axis=-1)
    print("pred=", classification_value)
    print(type(classification_value))
    for i in classification_value:
        cl = classes[i]
        x = i + 1
        st.write(cl)
model = load_model()
if button == 'MyModel':
    classification_dataset(data_model_1)
if button == 'MyResNet':
    st.write(':smile:')
    classification_dataset_2(data_model_2)
```