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 і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Система розпізнавання зображень на основі нейромережі»

Виконала:
студентка IV курсу, групи ТВ-з91
Козленко Маргарита Андріївна _____

Керівник:
Старший викладач
Бандурка Олена Іванівна _____

Консультант:

_____

Рецензент:
Професор кафедри ЦТЕ, д.т.н.,
проф. Отрох С. І. _____

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

Київ — 2023 року

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 і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____»_____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту

Козленко Маргарита Андріївна
(прізвище, ім'я, по батькові)

1. Тема роботи __Система розпізнавання зображень на основі нейромережі____
_____
_____
керівник роботи _____Бандурка Олена Іванівна, ст. в._____
(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 12.06.2023 р._____
3. Вихідні дані до роботи _____веб-додаток, мова програмування Python, середовище розробки PyCharm, фреймворк Flask._____
_____
_____
_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) ___розробити нейромережеву модель для розпізнавання зображень; розробити веб-додаток для взаємодії активації моделі з користувачем; проаналізувати математичну складову роботи згорткових нейромереж; дослідити алгоритми для реалізації навчання нейромережі на розпізнавання об'єктів на зображеннях._____
_____

5. Перелік ілюстративного матеріалу актуальність роботи, мета та завдання роботи, аналіз математичних обчислень в системі, вибір інструментальних засобів розробки, архітектура системи, головна сторінка системи, висновок.

6. Консультанти розділів роботи

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

7. Дата видачі завдання «30» вересня 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	02.10.2022-31.12.2022	виконано
3	Постановка вимог до проектування системи	14.04.2023-16.04.2023	виконано
4	Дослідження існуючих рішень	17.04.2023-23.04.2023	виконано
5	Розробка програмного продукту	24.04.2023-14.05.2023	виконано
6	Тестування	01.05.2023-21.05.2023	виконано
7	Захист програмного продукту	16.05.2023	виконано
8	Оформлення дипломної роботи	22.05.2023-04.06.2023	виконано
9	Передзахист	06.06.2023	виконано
10	Захист	21.06.2023	виконано

Студент _____ Козленко М.А.
(підпис) (ім’я. прізвище)

Керівник роботи _____ Бандурка О.І.
(підпис) (ім’я. прізвище)

РЕФЕРАТ

Структура і обсяг дипломної роботи. Робота містить 52 сторінки, 26 рисунків, 1 додаток та 24 посилання.

Метою роботи є створення системи на основі нейромережі, що вміє розпізнавати різні образи на зображеннях і веб-додаток для графічного запуску роботи системи.

Для досягнення мети розроблено три нейромережеві моделі, систему генерації графіків функцій, веб-додаток для графічного запуску роботи системи та було протестовано застосунок на різних даних.

Практичне значення одержаних результатів полягає в отриманні готових моделей для розпізнавання та класифікації об’єктів на вхідних даних. Зроблено огляд існуючих рішень, які розв’язують подібні задачі. Визначено подальші вектори розвитку даного дослідження.

Ключові слова: веб-додаток, згорткова нейронна мережа, розпізнавання медичних масок, розпізнавання графіків функцій.

ABSTRACT

Structure and scope of the thesis. The work contains 52 pages, 26 figures, 1 appendice and 24 references.

The goal of operation is the creation of systems based on a neural network that can recognize different images on images and a web application for graphically launching the system of operation.

To achieve the goal, three neural network models, a system for generating graphs of functions, a web application for graphically launching the system were developed, and the application was tested on various data.

The practical significance of the obtained results was in obtaining ready-made models for recognition and classification of objects on the input data. An overview of existing solutions that solve similar problems is made. Further development vectors of this research have been determined.

Keywords: web application, convolutional neural network, recognition of medical masks, recognition of function graphs.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN	–	Convolution Neural Network (Згорткова нейронна мережа)
YOLO	–	You Only Look Once (Дивишся лише раз), алгоритм розпізнавання зображень
GPU	–	Graphics Processing Unit (Графічний процесор)
API	–	Application Programming Interface (Прикладний програмний інтерфейс)
RGB	–	Red Green Blue (червоний, зелений, синій), адитивна колірна модель, що описує спосіб синтезу кольору
CUDA	–	Compute Unified Device Architecture, програмно-апаратна архітектура паралельних обчислень

ВСТУП

Системи розпізнавання на основі нейромереж стають все більш актуальними в сучасному світі, оскільки з кожним днем кількість даних, що потребують обробки, зростає в геометричній прогресії. Нейромережі дозволяють ефективно і швидко обробляти великі обсяги даних і здійснювати розпізнавання з високою точністю.

Одним з основних застосувань систем розпізнавання на основі нейромереж є розпізнавання образів. Застосування цих систем може бути корисним у багатьох сферах, таких як медицина, автоматизоване виробництво, банківська справа, транспорт, безпека і багато інших. Наприклад, в медицині системи розпізнавання можуть допомогти виявляти різні захворювання та оцінювати стан пацієнта на основі зображень медичних обстежень.

Крім розпізнавання образів, нейронні мережі також можуть бути використані для розпізнавання мовлення, здійснення прогнозів та аналізу даних. У технічних системах нейромережі використовуються для автоматизації процесів та зменшення витрат на ручну працю.

Одним з найважливіших аспектів систем розпізнавання на основі нейромереж є їх здатність до самоорганізації та навчання. Нейромережі можуть навчатися на великих обсягах даних та постійно покращуватися, що дозволяє їм забезпечувати високу точність розпізнавання.

Загалом, системи розпізнавання на основі нейромереж є важливим інструментом в сучасному світі, який дозволяє ефективно обробляти великі обсяги даних та надавати відповідну до поставленої задачі відповідь.

На даний момент в світі спостерігаються тенденції до автоматизації великої кількості процесів, що раніше виконувалися людиною. Технологія розпізнавання об’єктів на зображеннях та відео є однією з частин подібної автоматизації, адже це дозволяє позбутися людського фактору, а також зекономити достатньо часу.

Тому було вирішено розробити систему, яка може визначати положення та ідентифікувати об’єкти на фотографіях та надавати користувачеві візуальний результат щодо розпізнавання.

Метою даної роботи є розробка системи, яка зможе швидко та точно визначати об’єкти на зображеннях. Система є актуальною, адже на даний момент існує великий попит на автоматизоване розпізнавання конкретних об’єктів на зображеннях та відео, визначення яких може бути трудомістким для людини, на відміну від натренованої автоматизованої системи.

Робота складається з п’яти розділів:

- перший – постановка задачі системи розпізнавання зображень на основі нейромережі;
- другий – аналіз нейромережевих алгоритмів, засоби їх реалізації та види алгоритмів що направлені на роботу з розпізнаванням зображень;
- третій – інформація про використані засоби розробки програмного продукту;
- четвертий – опис компонентів та структура програми;
- п’ятий – демонстрація роботи програми та системні вимоги, методика користування системою користувачем.

1 ПОСТАНОВКА ЗАДАЧІ СИСТЕМИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ НЕЙРОМЕРЕЖІ

Метою дипломної роботи є дослідження алгоритмів нейромереж та імплементацію найкращого алгоритму для реалізації розпізнавання об’єктів по фотографії або відео. Система є унікальною та необхідною для розвитку та автоматизації розпізнавання даних, що може бути використаним не тільки людьми, а в майбутньому систему можна розширювати і натреновувати додатковими даними для автоматизації розпізнавання об’єктів на зображеннях на виробництвах з відсутністю похибки людського фактору. Система може бути застосована як в приватних цілях – для ідентифікації об’єктів на окремих фотографіях чи відео звичайними користувачами – так і в виробничих цілях, де розпізнавання об’єктів є важливою складовою робочого процесу, який можна удосконалити системою, може ідентифікувати та візуалізувати результат розпізнавання. При цьому додаток має бути абсолютно доступний, зрозумілий у використанні, достатньо мати доступ до мережі інтернет та веб-браузер. Програмний продукт реалізовуватиметься в вигляді веб додатку з відкритим кодом на GitHub, щоб кожен міг отримати вихідний код та доповнювати або змінювати власноруч додаток за своїм бажанням.

Завдання даної дипломної роботи полягає в наступних пунктах:

- розробити систему, яка буде розпізнавати об’єкти на зображеннях на основі нейромережевих алгоритмів з інтерфейсом веб-додатку та можливістю активації системи користувачем;
- дослідити математичну складову роботи згорткової нейромережі;
- проаналізувати алгоритми згорткових нейромереж;
- порівняти проаналізовані алгоритми та визначити найкращий для реалізації програмного продукту дипломної роботи.

Основні вимоги до функціоналу програмного додатку:

- завантаження даних з пристрою;

- активація процесу обробки завантаженого файлу за допомогою інтерфейсу користувача системи;
- відображення проаналізованих даних;
- зберігання результату обробки в локальному сховищі.

Висновки до розділу 1

У даному розділі було представлено та розглянуто задачу системи розпізнавання зображень на основі нейромережі, визначено мету роботи, представлені задачі дипломної роботи а також основні вимоги до функціоналу програмного продукту.

2 АНАЛІЗ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

Згорткова нейронна мережа, також відома як CNN — це клас нейронних мереж, який спеціалізується на обробці даних, які мають сітчасту топологію, наприклад зображення. Цифрове зображення — це двійкове представлення візуальних даних. Воно містить серію пікселів, упорядкованих у вигляді сітки, яка містить значення пікселів, щоб позначити, наскільки яскравим і якого кольору повинен бути кожен піксель [1].

Згорткові нейронні мережі — це тип багаторівневої нейронної мережі, яка призначена для розпізнавання візуальних шаблонів із піксельних зображень. У CNN «згортка» називається математичною функцією. Це тип лінійної операції, у якій ви можете помножити дві функції, щоб створити третю функцію, яка виражає, як форма однієї функції може бути змінена іншою. Простіше кажучи, два зображення, які представлені у вигляді двох матриць, перемножуються, щоб забезпечити вихід, який використовується для вилучення інформації із зображення. CNN схожа на інші нейронні мережі, але оскільки вони використовують послідовність згорткових шарів, вони додають рівень складності до рівняння. CNN не може функціонувати без згорткових шарів [12].

Людський мозок обробляє величезну кількість інформації в ту секунду, коли ми бачимо зображення. Кожен нейрон працює у своєму власному рецептивному полі та пов’язаний з іншими нейронами таким чином, що вони покривають усе поле зору. Подібно до того, як кожен нейрон реагує на стимули лише в обмеженій області поля зору, яка називається сприйнятливим полем у системі біологічного зору, кожен нейрон у CNN обробляє дані лише у своєму сприятливому полі. Шари розташовані таким чином, щоб спочатку виявляти прості візерунки (лінії, криві), а далі – складніші (обличчя, об’єкти) [9].

У різноманітних задачах комп’ютерного зору штучні нейронні мережі CNN піднялися на вершину. Це зацікавило людей у різних сферах.

Згорткова нейронна мережа складається з численних шарів, таких як шари згортки, шари об’єднання та повністю зв’язані шари, і використовує алгоритм зворотного поширення для автоматичного й адаптивного вивчення просторових ієрархій даних [10].

2.1 Архітектура згорткової нейронної мережі

CNN зазвичай має три рівні: згортковий рівень (convolution layer), рівень об’єднання (pooling layer) та повністю зв’язаний рівень (fully-connected).

Згортковий рівень застосовує фільтри до вхідного зображення, щоб виділити особливості, рівень об’єднання зменшує дискретизацію зображення, щоб зменшити обчислення, а повністю підключений рівень робить остаточний прогноз. Мережа вивчає оптимальні фільтри за допомогою зворотного поширення та градієнтного спуску [5].

Повністю зв’язаний рівень, який використовує вихідні дані процесу згортання та передбачає клас зображення на основі ознак, виділених на попередніх етапах.

Робота CNN полягає в тому, щоб стиснути зображення у формат, який легше обробляти, зберігаючи елементи, важливі для отримання гідного прогнозу. Це критично важливо для розробки архітектури, яка здатна вивчати функції, а також масштабована до великих наборів даних.

Нижче представлені основні характеристики складу CNN:

- інструмент згортки, який розділяє та ідентифікує різні характеристики зображення для аналізу в процесі, що називається виділенням ознак (feature extraction);
- мережа виділення ознак складається з багатьох пар згорткових або об’єднуючих шарів;
- повністю зв’язаний рівень, який використовує вихідні дані процесу згортання та передбачає клас зображення на основі ознак, виділених

на попередніх етапах;

- ця модель виділення ознак CNN має на меті зменшити кількість ознак, присутніх у наборі даних. Вона створює нові функції, які узагальнюють існуючі функції, що містяться в оригінальному наборі функцій. Існує багато рівнів мережі, як показано на схемі архітектури CNN [13].

Візуальне представлення вище наведених ознак може побачити нижче (рисунок 2.1).

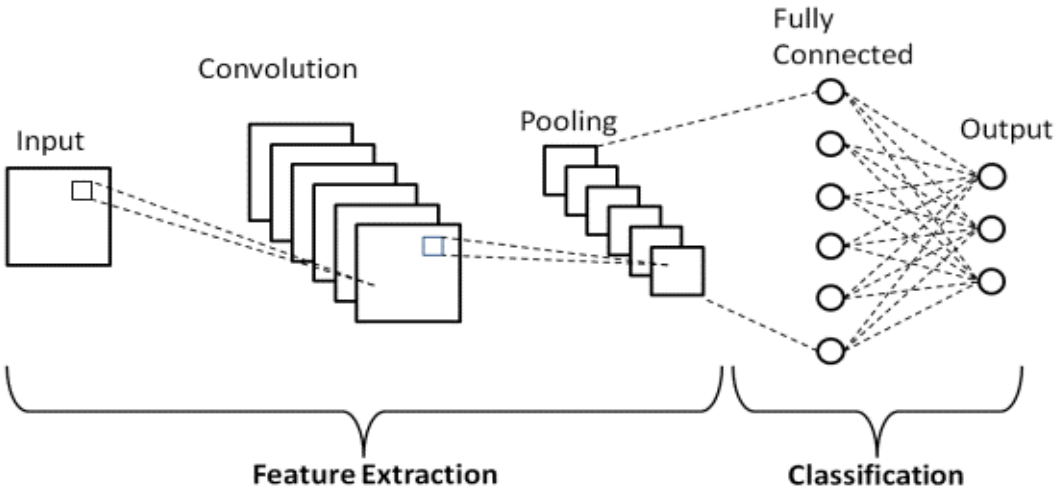

Рисунок 2.1 – Архітектура CNN [13]

2.2 Згортковий рівень

Рівень згортки є основним будівельним блоком CNN. Він несе основну частину обчислювального навантаження мережі.

Цей рівень виконує скалярний добуток між двома матрицями, де одна матриця є набором параметрів, які можна дізнатися, інакше відомих як ядро, а інша матриця є обмеженою частиною сприйнятливого поля. Ядро просторово менше, ніж зображення, але більш глибоке. Це означає, що якщо зображення складається з трьох (RGB) каналів, висота і ширина ядра будуть просторово малими, але глибина поширюється на всі три канали [15].

Під час прямого проходу ядро ковзає по висоті та ширині зображення,

створюючи представлення зображення цієї сприятливої області. Це створює двовимірне представлення зображення, відоме як карта активації, яка дає відповідь ядра на кожну просторову позицію зображення. Розмір ковзання ядра називається кроком [14].

Загалом, згортка — це математична операція над двома функціями, у якій два джерела інформації поєднуються для створення вихідної функції. Він використовується в широкому діапазоні застосувань, включаючи обробку сигналів, комп'ютерне зір, фізику та диференціальні рівняння. Хоча існує багато типів згорток, таких як безперервна, кругова та дискретна, ми зосередимося на останній, оскільки на згортковому рівні ми маємо справу з дискретними даними [14].

У комп'ютерному баченні згортка виконується між зображенням I і фільтром K, який визначається як мала матриця. Спочатку фільтр послідовно проходить через кожен піксель двовимірного вхідного зображення. На кожному кроці виконується поелементне множення між пікселями фільтра та відповідними пікселями зображення. Потім підсумовуються результати в один вихідний піксель. Після повторення цієї процедури для кожного пікселя отримується вихідну 2D матрицю функцій [14].

В даному розділі буде розглянуті на пояснені наступна важливі пункти щодо рівня згортки згорткової нейронної мережі:

- згорткові нейронні мережі застосовують фільтр до вхідних даних, щоб створити карту функцій, яка підсумовує наявність виявлених функцій у вхідних даних;
- фільтри можна створювати вручну, наприклад детектори ліній, але інновація згорткових нейронних мереж полягає в тому, щоб вивчати фільтри під час навчання в контексті конкретної проблеми прогнозування;
- як обчислити карту ознак для одно- та двовимірних згорткових шарів у згортковій нейронній мережі [14].

Після наведеної вище інформації, можна розглянути процедуру згортки на простому прикладі. У нас є зображення I 5 x 5 і фільтр K 3 x 3, які ми хочемо згорнути. Нижче (рисунок 2.2) можна побачити жирним шрифтом, які пікселі вхідного зображення використовуються на кожному кроці згортки:

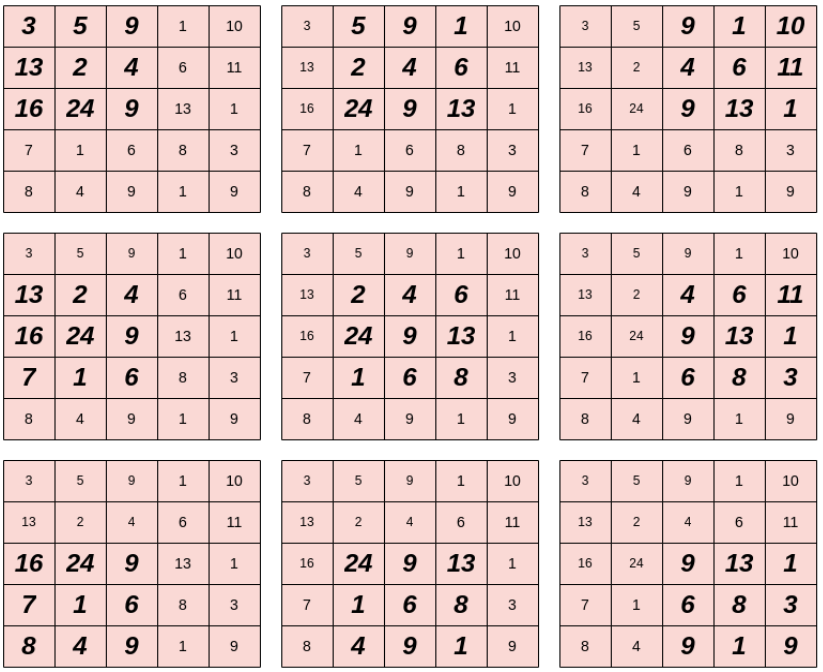

Рисунок 2.2 – Проходження фільтра по вхідній матриці [14]

Як згадувалося раніше, на кожному кроці відбувається процес множення значень вибраних пікселів (виділено жирним шрифтом) на відповідні значення фільтра та підсумовуються результати до єдиного виходу. У прикладі нижче (рисунок 2.3) обчислюється згортка в центральному пікселі (крок = 5)

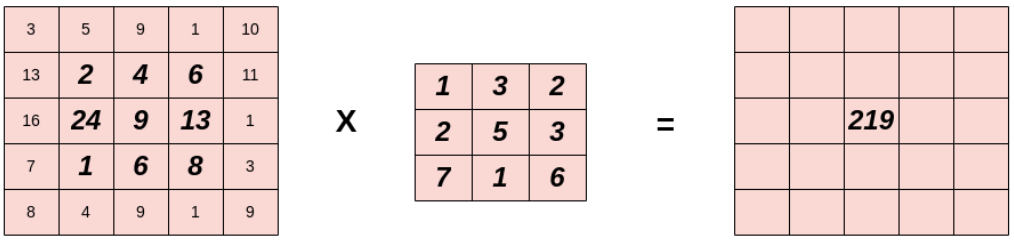

Рисунок 2.3 – Результат обчислення по центральному пікселю [14]

Спочатку обчислюється множення кожного пікселя фільтра на відповідний піксель зображення. Потім підсумовуються всі добутки: 2*1+4*3+6*2+24*2+5*9+13*3+1*7+6*1+8*6 = 219.

Отже, центральний піксель вихідної карти активації дорівнює 219. Ця процедура виконується для кожного пікселя вхідного зображення.

Згортковий рівень є основним будівельним блоком кожної згорткової нейронної мережі. У кожному шарі ми маємо набір вивчаємих фільтрів. Вхідні дані згортаються з кожним фільтром під час прямого поширення, створюючи вихідну карту активації цього фільтра. У результаті мережа вивчає активовані фільтри, коли на вхідному зображенні з'являються певні функції. Нижче можна побачити приклад оператора Прюітта, чіткого фільтра, який використовується для виявлення країв (рисунок 2.4):

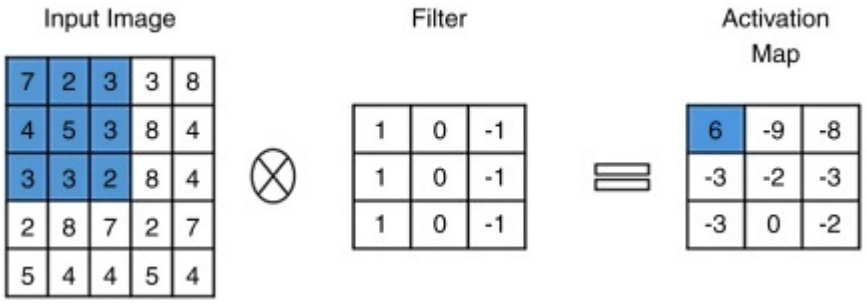

Рисунок 2.4 - Результат згортки по фільтру Прюітта [14]

Щоб сформулювати спосіб обчислення вихідного розміру згорткового шару, треба виділити два критичні гіперпараметри - крок (stride) та заповнення (padding) [14].

Під час згортки фільтр ковзає зліва направо та зверху вниз, доки не пройде через усе вхідне зображення. Ми визначаємо крок S як крок фільтра. Отже, коли ми хочемо зменшити дискретизацію вхідного зображення та отримати менший вихід, ми встановлюємо $S > 0$.

Нижче представлено матрицю, що була розглянута на рисунку 2.3, коли крок $S = 2$ (рисунок 2.5):

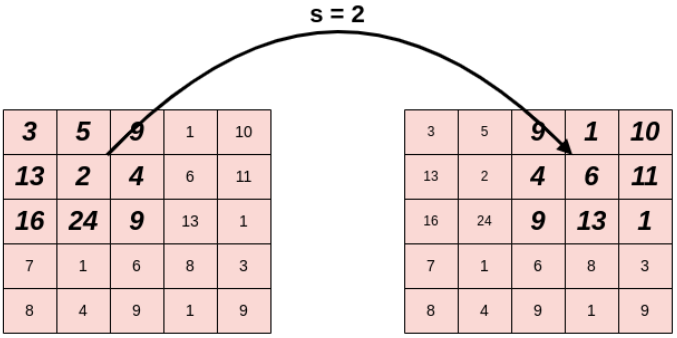

Рисунок 2.5 - Приклад проходження фільтру по матриці з кроком S = 2 [14]

У згортковому шарі ми спостерігаємо, що пікселі, розташовані по кутах і краях, використовуються набагато менше, ніж пікселі в середині. Наприклад, у наведеному нижче прикладі ми маємо вхідне зображення 5 x 5 і фільтр 3 x 3 (рисунок 2.6):

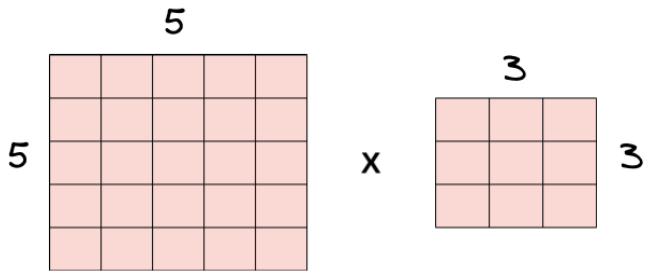

Рисунок 2.6 – Приклад розмірів вхідного зображення і фільтру [14]

Нижче можна побачити скільки разів використовується кожен піксель із вхідного зображення під час застосування згортки з S = 1 (рисунок 2.7):

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

Рисунок 2.7 – Кількість застосувань кожного пікселю при кроці S = 1 [14]

Видно, що піксель (0, 0) використовується лише один раз, тоді як центральний піксель (3, 3) використовується дев’ять разів. Загалом пікселі, що розташовані посередині, використовуються частіше, ніж пікселі по краях і кутах. Тому інформація на межах зображень зберігається не так добре, як інформація в середині [14].

Простим і потужним рішенням цієї проблеми є заповнення, яке додає рядки та стовпці нулів до вхідного зображення. Якщо ми застосовуємо заповнення P у вхідному зображенні розміром W × H, вихідне зображення матиме розміри (W+2P) x (H+2P) [14].

Нижче (рисунок 2.6) можна побачити приклад зображення до та після заповнення з P = 2. Як бачимо, розміри зросли з 5 x 5 до 9 x 9:

Рисунок 2.6 - Додавання заповнення до вхідного зображення [14]

Використовуючи заповнення (padding) у згортковому шарі, збільшується внесок пікселів по кутах і краях у процес навчання. Додавання заповнення важливо робити для того, щоб кожен з пікселів вхідного зображення був залучений у процес навчання якомога більшу кількість разів.

Тепер можна перейти до представлення формули для обчислення вихідного розміру згорткового шару. Для прикладу маємо наступні дані:

- зображення розміром W_{in} x H_{in};
- фільтр розміром K x K;

- крок S і заповнення P.

Вихідна карта активації матиме таку ширину (формула 2.1) і висоту (формула 2.2):

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1, \quad (2.1)$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1, \quad (2.2)$$

де W_{out}, H_{out} – ширина і висота вихідної карти активації відповідно;

W_{in}, H_{in} – ширина і висота вхідної карти активації відповідно;

K – розмір фільтра;

P – розмір заповнення (padding);

S – значення кроку [9].

Якщо вихідні розміри не є цілими числами, це означає, що крок S був встановлений неправильно.

Щодо представлених вище формул існують два виняткових випадки:

1. Якщо заповнень взагалі немає, вихідна карта матиме наступну ширину (формула 2.3) і висоту (формула 2.4) :

$$\frac{W_{in} - K}{S} + 1, \quad (2.3)$$

$$\frac{H_{in} - K}{S} + 1, \quad (2.4)$$

де W_{in}, H_{in} – ширина і висота вихідної карти активації відповідно;

K – розмір фільтра;

S – значення кроку [14].

2. Якщо потрібно залишити розмір вхідних даних незмінним після шару згортки – додається заповнення, що обчислюється наступним чином (формула 2.5):

$$P = \frac{K - S}{2}, \quad (2.5)$$

де P – значення заповнення;

K і S – значення розміру фільтра і кроку відповідно [14].

2.3 Рівень об’єднання

Рівень об’єднання зазвичай застосовується після шару згортки, щоб зменшити просторовий розмір вхідних даних. Він застосовується незалежно до кожного зрізу глибини вхідного обсягу. Глибина об’єму завжди зберігається під час операції об’єднання. Нижче (рисунок 2.7) до вхідного зображення застосований шар максимального об’єднання (max-pooling):

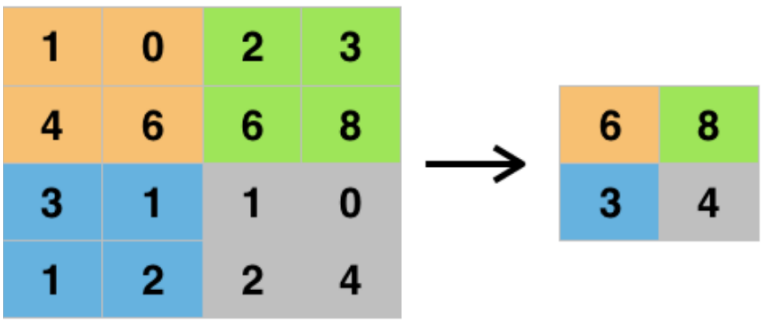

Рисунок 2.7 - Результат виконання максимального об’єднання

Припустимо, вхідні розміри попередньо згорнутого зображення мають ширину W_{in}, висоту H_{in} і глибину D_{in}. Рівень об’єднання вимагає 2 гіперпараметри: розмір фільтра K і крок S [16].

Після застосування шару об’єднання над вхідними розмірами, вихідне зображення матиме таку ширину (формула 2.6), висоту (формула 2.7) і глибину (формула 2.8):

$$W_{out} = \frac{W_{in} - K}{S} + 1, \tag{2.6}$$

$$H_{out} = \frac{H_{in} - K}{S} + 1, \tag{2.7}$$

$$D_{out} = D_{in}, \tag{2.8}$$

де W_{out}, H_{out}, D_{out} – ширина, висота і глибина вихідної карти відповідно;

W_{in}, H_{in}, D_{in} – ширина, висота і глибина вхідної карти відповідно;

K – розмір фільтра;

S – значення кроку [16].

Переваги застосування рівня об’єднання полягають в тому, що цей процес зменшує кількість параметрів навчання та вартість обчислень, таким чином контролюючи переобладнання і робить модель інваріантною до певних спотворень.

Існує декілька видів рівнів об’єднання:

- максимальне об’єднання (max-pooling): у цьому типі об’єднання максимальне значення кожного ядра в кожному сегменті глибини фіксується і передається на наступний рівень;
- мінімальне об’єднання (min-pooling): у цьому типі мінімальне значення кожного ядра в кожному сегменті глибини фіксується і передається на наступний рівень.
- середнє об’єднання (average pooling): у цьому типі обчислюється середнє значення ядра [16].

Висновки до розділу 2

У даному розділі було проаналізовано згорткові нейронні мережі зі сторони математичних операцій, що проводяться над зображеннями та способи покращення вихідних результатів для подальшої роботи над ними. Розуміння обчислень, що проводяться над кожним окремим пікселем зображення дуже важливе, адже саме це лягає в основу роботи окремих алгоритмів нейромереж.

На основі проаналізованої інформації, яка була представлена у цьому розділі, можна створити власний нейромережевий алгоритм, що буде проводити деякі операції над вхідних зображенням, та представляти вихідні дані в обробленому вигляді, тобто після операцій об’єднання на згортки.

3 ЗАСОБИ РОЗРОБКИ

Правильний вибір засобів розробки програмного забезпечення є важливою складовою процесу створення програм. Невірний вибір може призвести до погіршення якості коду, збільшення часу розробки та витрат, а також до проблем зі сумісністю та інтеграцією з іншими засобами.

Перш за все, потрібно звернути увагу на мову програмування, яка буде використовуватися. Різні мови програмування мають свої переваги та недоліки, тому вибір повинен базуватися потребах розробника та вимогах проекту. Наприклад, якщо потрібно розробити веб-додаток, то можна розглянути використання мов програмування, таких як PHP, Python або Ruby. В даному програмному продукті було вирішено використовувати мову програмування Python, адже саме вона та її бібліотеки дозволяють розробити найкращий на найшвидший програмний продукт на основі нейромережі.

Другим важливим кроком є вибір інтегрованого середовища розробки (IDE). IDE дозволяють зручно та ефективно писати, тестувати та налагоджувати код. При виборі IDE слід звернути увагу на такі критерії, як підтримка мов програмування, функціональні можливості, інтерфейс користувача та наявність додаткових інструментів. Було вирішено вибрати середовище розробки PyCharm.

Серед інших інструментів, використаних в розробці програмного продукту дипломної роботи можна виділити фреймворк Flask, який дозволяє розробляти веб-додати на мові програмування Python, а також такі бібліотеки як matplotlib, numpy, opencv, torch, torchvision, seaborn і pandas.

3.1 Середовище розробки PyCharm

PyCharm - це інтегроване середовище розробки для мови програмування Python, яке створено компанією JetBrains. PyCharm має безліч корисних функцій

та інструментів, що дозволяють ефективно працювати з Python-кодом [17].

Один з головних інструментів PyCharm - це дебагер. Він дозволяє відлагоджувати код, встановлювати точки зупинки, досліджувати змінні та інші параметри в процесі виконання програми. Також PyCharm має інспектор коду, який допомагає знайти та виправити помилки.

Ще один корисний інструмент в PyCharm - це можливість роботи з системою контролю версій Git. PyCharm дозволяє легко здійснювати коміти, пушити та пулити зміни, переглядати історію змін та розв’язувати конфлікти.

PyCharm також має вбудовану підтримку віртуальних середовищ Python та управління залежностями проекту, що дозволяє легко встановлювати та оновлювати залежності проекту, ізолювати середовище від інших проектів та робити його більш стабільним та незалежним.

За допомогою PyCharm можна створювати різні типи проектів на Python, такі як консольні, веб-додатки, GUI-додатки, наукоємні обчислення та багато інших. Крім того, PyCharm підтримує ряд інших мов програмування, таких як JavaScript, HTML, CSS, SQL, тому може використовуватись для розробки комплексних багатомовних проектів [17].

Загалом, PyCharm - це потужне та ефективне інтегроване середовище розробки для мови програмування Python, яке надає розробникам багато корисних інструментів для зручної та швидкої розробки високоякісних проектів.

3.2 Мова Python

Python - це високорівнева інтерпретована мова програмування, яка була створена в 1991 році Гвідо ван Россумом. Python має простий та зрозумілий синтаксис, який дозволяє швидко писати код та знижує кількість помилок [19].

Основними особливостями Python є:

- сильна підтримка об'єктно-орієнтованого програмування: Python

дозволяє створювати класи, об'єкти та інші об'єкти-орієнтовані конструкції, що дозволяє розробляти складні програми та підтримувати їх;

- інтерпретована мова: Python компілюється під час виконання, що дозволяє бути більш гнучким у виборі платформи та забезпечує швидку розробку;
- широкі можливості стандартної бібліотеки: Python має велику стандартну бібліотеку, яка містить багато корисних функцій та модулів для різноманітних задач, що дозволяє значно скоротити час розробки;
- динамічна типізація: Python не вимагає явного вказання типів змінних, що дозволяє швидко писати код та мінімізує кількість помилок;
- переносимість: Python може працювати на різних операційних системах та архітектурах, що дозволяє створювати крос-платформні програми.

Python використовується для створення різноманітних програм, від скриптів та інструментів до веб-додатків, машинного навчання та науки даних. Завдяки своїм особливостям та зручності, Python став популярним в середині розробників та знайшов застосування в багатьох галузях, що робить його однією з найбільш важливих мов програмування в сучасному світі [18].

3.3 Фреймворк Flask

Flask є одним з найбільш популярних мікрофреймворків для створення веб-додатків на мові програмування Python. Цей фреймворк дозволяє розробникам створювати веб-додатки різного рівня складності з використанням мінімальної кількості коду та максимальної швидкості [20].

Однією з головних переваг Flask є його простота і легкість використання. Flask не має складних конфігурацій та вимог до проекту, що дозволяє розробникам швидко почати роботу з цим фреймворком. Flask дає розробникам

волю для створення веб-додатків на свій смак, не нав'язуючи певної структури проекту.

Flask має широку підтримку розширень, які дозволяють розширювати функціональність веб-додатків. Flask-розширення можна встановлювати з використанням менеджера пакетів pip та використовувати їх з легкістю. Це дозволяє розробникам швидко додавати нові можливості та функції до своїх проектів [4].

Flask дуже гнучкий та легко інтегрується з іншими технологіями та сервісами. Flask підтримує багато різних типів баз даних, таких як MySQL, PostgreSQL, SQLite та інші, що дозволяє розробникам працювати з даними зі своїх додатків. Flask також має підтримку шаблонізаторів, таких як Jinja2, що дозволяє розробникам легко створювати шаблони для своїх веб-сторінок [20].

Flask має вбудовану підтримку REST API, що дозволяє створювати API для взаємодії з іншими додатками та сервісами. Flask також підтримує вбудований обробник форм, який дозволяє розробникам легко створювати та обробляти форми на своїх веб-сторінках [4].

Окрім того, Flask має вбудовану підтримку тестування, що дозволяє розробникам швидко та легко перевіряти функціональність своїх додатків. Flask також підтримує вбудовану систему логування, яка дозволяє вести журнал подій та помилок, що виникають у веб-додатках [4].

Крім того, Flask має велику спільноту розробників, яка допомагає іншим розробникам вирішувати проблеми та ділитися знаннями про цей фреймворк. Велика кількість ресурсів, таких як документація та онлайн-курси, також допомагає розробникам засвоїти Flask та почати роботу з ним.

Загалом, Flask є дуже потужним та гнучким фреймворком, який дозволяє розробникам швидко створювати веб-додатки та API з використанням мінімальної кількості коду та максимальної швидкості.

3.4 Алгоритм YOLO

Алгоритм YOLO - це потужний алгоритм комп'ютерного зору для об'єктного виявлення, що розроблений з метою забезпечення швидкої та точної роботи в режимі реального часу. YOLO є досить популярним алгоритмом, який використовується для візуального розпізнавання об'єктів в зображеннях та відео.

Алгоритм YOLO працює шляхом використання глибоких нейронних мереж з використанням згорткових шарів. Ці згорткові шари використовуються для здійснення операцій зі зменшенням розміру зображення, що дозволяє значно знизити кількість обчислень, необхідних для роботи алгоритму [2].

YOLO складається з декількох етапів. Спочатку зображення поділяється на сітку різного розміру, залежно від обраної конфігурації алгоритму. Кожен блок сітки відповідає за визначення тих областей зображення, де можуть знаходитися об'єкти [2].

Далі алгоритм використовує нейронну мережу для визначення важливих ознак кожної з областей зображення, які можуть бути пов'язані з об'єктами. Наприклад, мережа може шукати такі ознаки, як форма, розмір, текстура та колір об'єктів [2].

Після того, як мережа знайде важливі ознаки кожної з областей зображення, алгоритм використовує ці дані для прогнозування розташування та класифікації об'єктів на зображенні. Крім того, алгоритм YOLO може виконувати додаткові завдання, такі як визначення розміру та орієнтації об'єктів.

Однією з ключових переваг алгоритму YOLO є те, що він вміє визначати декілька об'єктів на одному зображенні одночасно. Це дозволяє забезпечити більш точне та повне виявлення об'єктів на зображенні [2].

Крім того, YOLO є досить швидким та ефективним алгоритмом, що дозволяє йому працювати в режимі реального часу на пристроях з обмеженими

обчислювальними ресурсами. Це робить його ідеальним вибором для застосування в системах відеоспостереження, автоматизованих системах управління транспортними потоками та інших подібних задачах.

Загалом, алгоритм YOLO - це потужний інструмент для об'єктного виявлення, який дозволяє швидко та ефективно визначати розташування та класифікацію об'єктів на зображеннях та відео. З його допомогою можна реалізувати різноманітні проекти, що потребують виявлення об'єктів, і забезпечити їхню надійність та точність.

Протягом останніх кількох років алгоритм був розроблений та підтримувався командою з Університету Вашингтона та Facebook AI Research. В наступних рядках ми опишемо всі версії алгоритму YOLO.

YOLOv1: Перша версія алгоритму YOLO була випущена в 2015 році. Ця версія використовує глибоку нейронну мережу для виявлення об'єктів на зображеннях. Однією з ключових переваг цієї версії є її швидкість - алгоритм може обробляти зображення з швидкістю до 45 кадрів за секунду. Однак, YOLOv1 має деякі недоліки, такі як низьку точність в порівнянні з іншими алгоритмами [21].

YOLOv2: Випущений у 2017 році, YOLOv2 був покращеною версією алгоритму. Він має більшу точність та надійність в порівнянні з YOLOv1, а також може працювати з відео зі швидкістю до 90 кадрів за секунду. YOLOv2 також використовує більш потужну нейронну мережу та використовує додаткові техніки для покращення виявлення об'єктів [21].

YOLOv3: Випущений у 2018 році, YOLOv3 є найбільш потужною та точною версією алгоритму до цього часу. Він використовує більш складну нейронну мережу, яка дозволяє досягнути точності виявлення об'єктів більш якісно, ніж YOLOv2. YOLOv3 може працювати з відео зі швидкістю до 30 кадрі за секунду, що може бути менш ефективним, ніж YOLO v2, але компенсує це вищою точністю виявлення об'єктів. В YOLO v3 також включено кілька додаткових функцій, таких як зменшення впливу невідомих факторів на процес

виявлення та покращення стійкості до перекладу та масштабування [21].

YOLO v4: Випущений в 2020 році, YOLO v4 не є останньою версією алгоритму. Він є ще більш потужним та точним, ніж його попередники. YOLOv4 включає більш потужні нейронні мережі та використовує додаткові техніки для покращення точності виявлення об'єктів. Наприклад, в YOLOv4 використовуються механізми уваги, що дозволяють нейронній мережі зосередитися на важливих деталях зображення. В результаті цього досягається ще вища точність та швидкість виявлення об'єктів на зображеннях та відео [21].

YOLOv5 є однією з найновіших та найбільш ефективних версій алгоритму для виявлення об'єктів на зображеннях та відео. Він був розроблений компанією Ultralytics і випущений в червні 2020 року. YOLOv5 використовує архітектуру мережі, яка базується на принципах детектора об'єктів на основі пропонованих рамок, що включає ефективний механізм пригнічення неактивних областей зображення. Це дозволяє YOLOv5 працювати значно швидше, ніж його попередники, і зменшує кількість помилок виявлення об'єктів. YOLOv5 також має декілька покращень порівняно з YOLO v4. Він включає нейронну мережу CSP (Cross Stage Partial), яка дозволяє швидше навчати модель і зменшити кількість параметрів, що необхідні для роботи моделі. Крім того, YOLOv5 має покращений механізм для виявлення малих об'єктів, що дозволяє збільшити точність виявлення таких об'єктів. Однією з ключових переваг YOLOv5 є те, що він має багато режимів роботи, які дозволяють використовувати його для різних задач. Наприклад, можна використовувати YOLOv5 для виявлення облич, розпізнавання номерних знаків, виявлення автомобілів і так далі [21].

Нещодавно були випущені більш нові версії розглянутого в даному підрозділі алгоритму: YOLOv6, YOLOv7, YOLOv8. Вони всі є більш довершеними, ніж п’ята версія алгоритму, тобто мають розширені можливості більш ефективного тренування і великий обсяг класів у стандартній моделі. Кожна наступна версія є загалом більш довершеною ніж попередня, що видно

по розвитку перших п’яти. Стрімкий розвиток даного алгоритму дозволить в майбутньому розробляти набагато ефективні системи з розпізнавання об’єктів на зображеннях на основі нейромережі.

3.5 Roboflow

Roboflow – це комплексна платформа комп’ютерного зору, яка спрощує процес створення, навчання та розгортання моделей комп’ютерного зору. Вона надає набір інструментів і ресурсів для оптимізації наскрізного робочого процесу, полегшуючи розробникам і дослідникам роботу з великомасштабними наборами даних зображень і навчання високоефективних моделей [22].

Однією з ключових особливостей Roboflow є її можливості керування наборами даних. Це дозволяє користувачам завантажувати, упорядковувати та коментувати свої набори даних зображень у зручному інтерфейсі. Платформа підтримує різні формати анотацій, що робить її гнучкою та сумісною з різними інструментами маркування та робочими процесами. За допомогою Roboflow можна ефективно обробляти великі обсяги даних, гарантуючи, що вони добре організовані та готові до навчання.

Іншим потужним аспектом Roboflow є його інтеграція з популярними фреймворками глибокого навчання, такими як TensorFlow, PyTorch і YOLO. Вона забезпечує повну сумісність і дозволяє експортувати анотовані набори даних у форматі, який безпосередньо сумісний із цими фреймворками. Це спрощує процес навчання моделей комп’ютерного зору, оскільки можна легко включити анотовані дані у бажане середовище глибокого навчання [7].

Крім того, Roboflow пропонує низку методів розширення даних для покращення набору даних і покращення продуктивності моделі. Вона надає різноманітні параметри розширення, включаючи зміну розміру зображення, обертання, перевертання, тощо. Ці доповнення допомагають збільшити

різноманітність і мінливість даних, дозволяючи моделям краще узагальнювати та добре працювати на невидимих зображеннях.

Розгортання моделей комп'ютерного зору стало легшим завдяки можливостям хостингу моделей Roboflow. Після того, як модель була навчена, Roboflow надає зручний спосіб розгорнути її як API, дозволяючи безперешкодно інтегрувати її у програми чи служби. Це дозволяє зробити висновок у реальному часі та полегшує використання можливостей комп'ютерного бачення у проектах.

Загалом Roboflow – це потужна платформа, яка спрощує весь робочий процес комп'ютерного зору, від керування набором даних до розгортання моделі. Його інтуїтивно зрозумілий інтерфейс, функції керування наборами даних та інтеграція з популярними фреймворками глибокого навчання роблять його цінним інструментом для дослідників, розробників і всіх, хто працює із завданнями комп'ютерного зору.

3.6 PyTorch

PyTorch – це платформа машинного навчання з відкритим вихідним кодом, яка надає динамічний обчислювальний графік для створення та навчання нейронних мереж. PyTorch, розроблена командою дослідження штучного інтелекту Facebook, завоював популярність серед дослідників і розробників завдяки своїй гнучкості, простоті використання та широкій підтримці алгоритмів глибокого навчання.

Однією з ключових особливостей PyTorch є її динамічний характер. На відміну від фреймворків статичних графів, таких як TensorFlow, PyTorch дозволяє розробникам визначати та змінювати обчислювальні графіки на льоту. Ця динамічна природа платформи полегшує налагодження та експерименти з моделями, оскільки графік будується та виконується більш інтуїтивно зрозумілим та інтерактивним способом. Це також дозволяє створювати більш

складні та динамічні мережеві архітектури, де потік керування може змінюватися залежно від вхідних даних [23].

PyTorch надає широкий спектр інструментів і модулів, які спрощують процес створення та навчання нейронних мереж. Він пропонує багатий набір попередньо визначених рівнів, функцій втрат і алгоритмів оптимізації, що полегшує побудову складних мережевих архітектур. Крім того, PyTorch легко інтегрується з популярними бібліотеками Python, дозволяючи користувачам використовувати широку екосистему наукових обчислювальних інструментів для попередньої обробки даних, візуалізації та оцінки [23].

Однією з визначних особливостей PyTorch є можливість автоматичної диференціації. Це дозволяє розробникам обчислювати градієнти тензорів і параметрів нейронної мережі щодо заданої функції втрат, полегшуючи виконання зворотного поширення та оновлення вагових коефіцієнтів моделі під час навчання. Ця функція автоматичного диференціювання в поєднанні з динамічним обчислювальним графіком PyTorch забезпечує потужну структуру для реалізації та експериментування з функціями втрат і методами оптимізації.

PyTorch також підтримує прискорення GPU, забезпечуючи ефективні обчислення на GPU NVIDIA. Вона забезпечує серверну частину CUDA, яка дозволяє переносити тензорні операції та мережеві обчислення на графічний процесор, що призводить до значного прискорення навчання та висновків. Це робить PyTorch придатним для завдань глибокого навчання, які вимагають великомасштабної паралельної обробки природної мови та генеративного моделювання. [24].

Крім того, PyTorch має яскраву та активну спільноту, яка сприяє її зростанню та розвитку. Природа PyTorch, керована спільнотою, призвела до створення численних бібліотек, інструментів і ресурсів, які ще більше розширюють її можливості. Це включає такі бібліотеки, як torchvision для завдань комп’ютерного зору, torchaudio для обробки звуку та трансформатори для обробки природної мови. [6].

Таким чином, PyTorch – це потужний і гнучкий фреймворк машинного навчання, який забезпечує динамічний обчислювальний графік, автоматичне розрізнення та прискорення GPU. Його інтуїтивно зрозумілий інтерфейс, широка бібліотечна підтримка та активна спільнота роблять його популярним вибором для дослідників і розробників, які працюють над широким спектром завдань машинного навчання.

Висновки до розділу 3

Загалом, в даному розділі було проаналізовано середовище розробки PyCharm, мову програмування Python і фреймворк Flask, що забезпечують підтримку продукту, структурне та візуальне оформлення системи. Також було розглянуто найголовнішу частину забезпечення розробки програмного продукту – алгоритм YOLO.

Отже, в даному розділі було досліджено та представлено вибрані засоби розробки, використані при створенні програмного продукту дипломної роботи та розглянуто різні алгоритми навчання нейромережі. Під час аналізу різних версій алгоритму YOLO, було вирішено використовувати YOLOv5 як алгоритм що ляже в основу програмного продукту дипломної роботи, адже саме він на даний момент є достатньо універсальним а також не вимагає великої кількості вільної пам’яті для тренування нейромережі як на центральному процесорі комп’ютера, так і на графічному.

Також було досліджені інші допоміжні системи для ефективної розробки програмного нейромережевого продукту і було обрано використовувати Roboflow для будування датасету і бібліотеку torchvision фреймворку PyTorch.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Для того, щоб звичайний користувач міг легко і швидко скористатися системою розпізнавання об’єктів було створено веб-додаток з можливістю завантаження картинок для розпізнавання та швидкого аналізу і виведення результату.

4.1 Структура програмного забезпечення

Файлова структура програмного забезпечення має просту структуру, яка представлена нижче на рисунку 4.1:

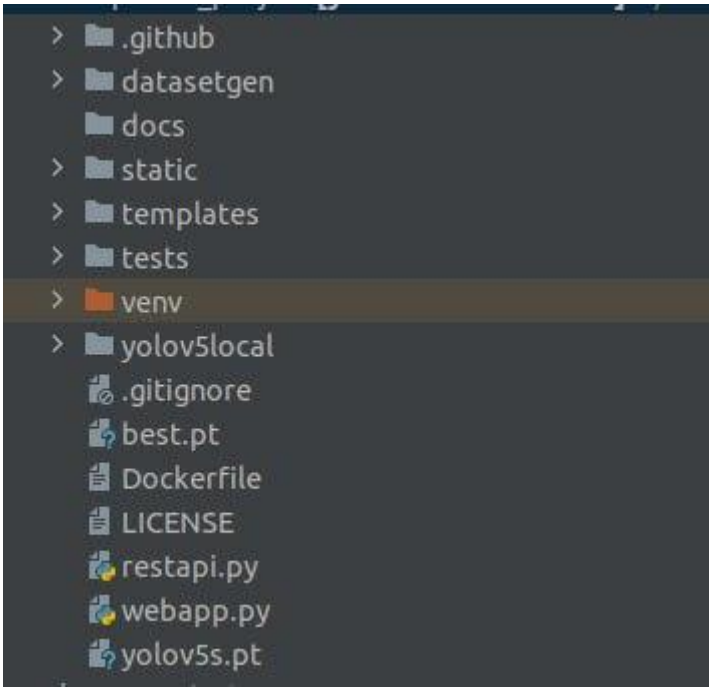

Рисунок 4.1 – Файлова структура програмного продукту

Основна частина програмного продукту реалізована в файлах webapp.py та restapi.py, запуск виконання яких запускає роботу проекту та дає можливість побачити його виконання, зайшовши у веб-браузер.

Директорія templates складається з файлів розмітки веб-сторінки, що забезпечують візуальне представлення програмного продукту.

Директорія static зберігає файли оброблених нейромережею картинок, що

були завантажені користувачем до веб-додатку.

Директорія test складається з файлів програмного коду, який відповідає за тестування системи на вразливості, який був розроблений в процесі налагодження роботи програмного продукту.

Директорія venv – це стандартна директорія будь-якого додатку, написаного на мові програмування Python. Всі зовнішні бібліотеки, які забезпечують роботу програми, завантажуються в неї та зберігаються в даній директорії.

Файл yolov5s – файл моделі алгоритму YOLO, який використовується при навчанні нейромережі.

Директорія datasetgen – це директорія що містить в собі файл з кодом, написаним для генерації датасету з картинок, на яких зображені різного роду графіки. Цей датасет використовується при тренуванні моделі нейромережі для розпізнавання графіків.

Директорія yolov5local містить в собі кастомну модель, попередньо натренованою на розпізнавання наявності та відсутності маски на людині.

Головна функція програми інтегрує в програмне забезпечення модель yolov5, для проведення розпізнавання об’єктів та завантажує віддалено розташовану натреновану модель за допомогою функції torch.hub.load() і запускає програму за прописаними критеріями.

4.2 Генерація датасету з картинок для розпізнавання графіків

При дослідженні моделі YOLOv5 як алгоритму для розпізнавання графіків на зображеннях, було визначено що для успішного тренування моделі необхідно мати достатню кількість чітко визначених картинок, з яких формуватиметься датасет для тренування нейромережевої моделі. Оскільки в мережі Інтернет не було достатньої кількості чітких та якісних зображень, які можна було б використати в якості датасету для тренування нейромережі, було

вирішено написати програмний код, що створить потрібну кількість зображень, з яких надалі формується датасет для тренування моделі нейромережі на розпізнавання графіків.

Програмний код генерації зображень написаний на мові програмування Python при залученні бібліотек matplotlib і numpy. Бібліотека numpy в даному випадку допомагає виконати математичні обчислення, необхідні для задання графіку а також визначити за допомогою влаштованих функцій вид графіку. Бібліотека matplotlib використовується для відображення графіку та зберіганням зображення у форматі png локально на комп’ютері.

Нижче представлені приклади згенерованих зображень за допомогою вище описаних методів (рисунок 4.2, рисунок 4.3):

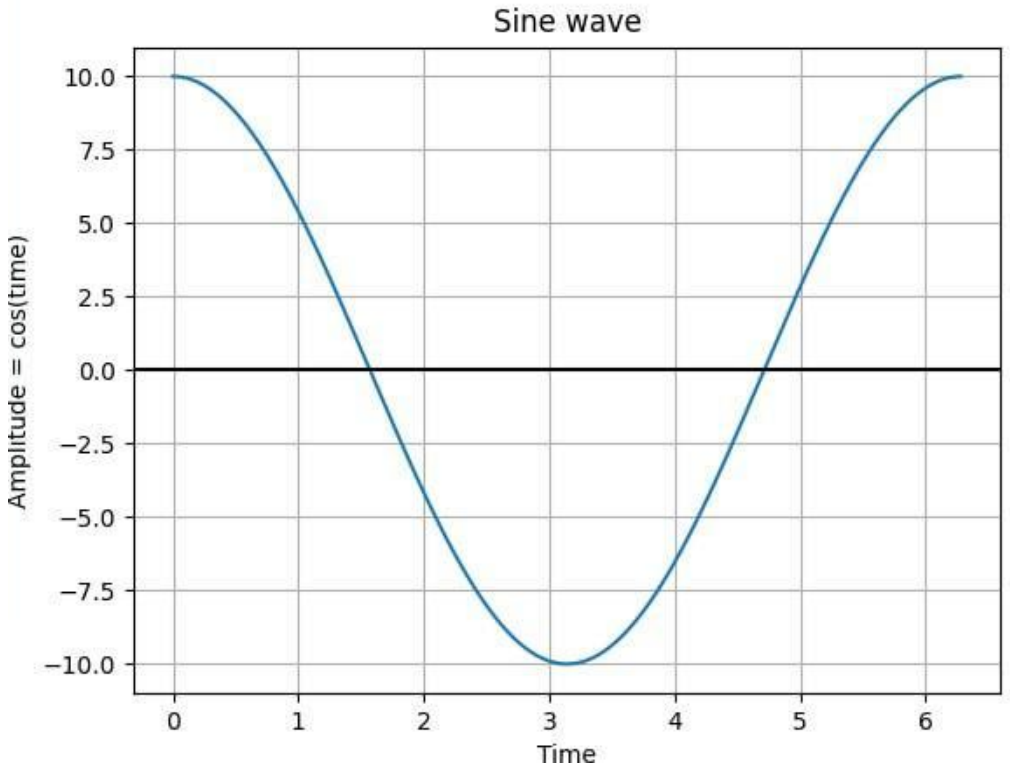

Рисунок 4.2 – Графік функції y=cos(x) з амплітудою рівною 1

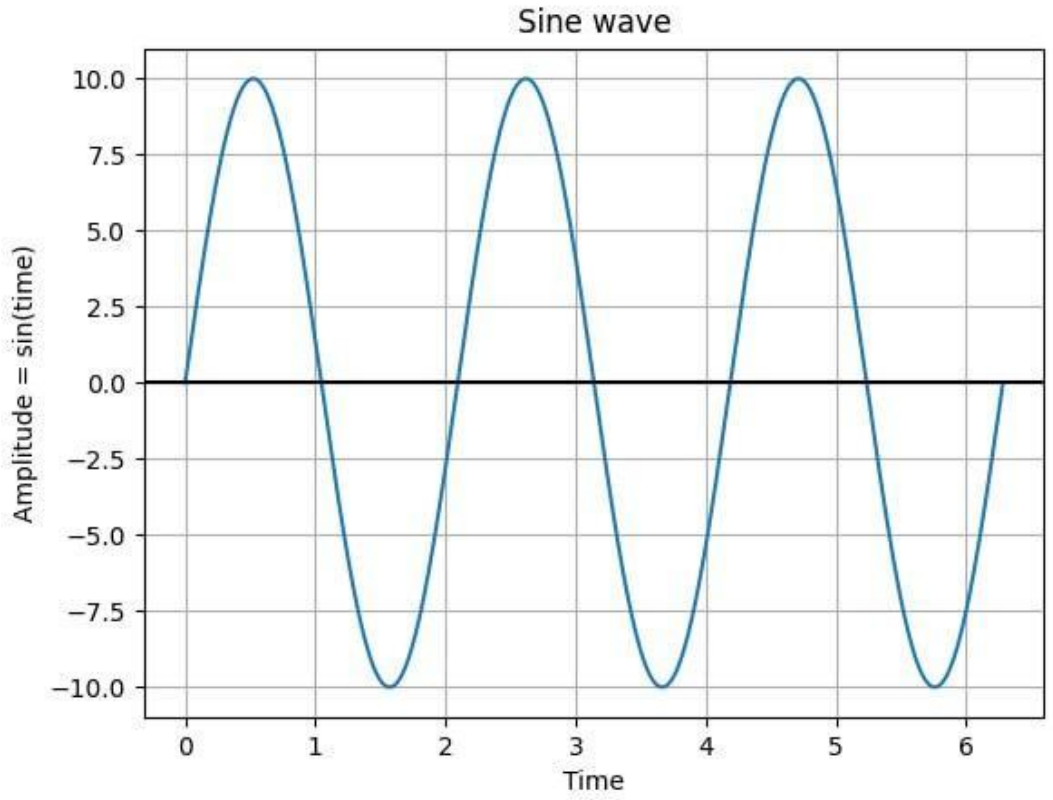

Рисунок 4.3 – Графік функції y=sin(x) з амплітудою рівною 3

Як можна побачити вище, при генерації картинок для датасету графіків функцій, були реалізовані не тільки розрахункові обчислення та виведення зображень, а також можливість збільшення або зменшення амплітуди таких графіків як синусоїда і косинусоїда для отримання якомога більшої кількості різноманітних картинок що формуватимуть датасет.

4.3 Формування датасету для тренування нейромережі

Для тренування згорткової нейронної мережі за алгоритмом YOLO необхідно створити датасет, який представляє собою пари з картинок та текстового файлу з координатами шуканого об’єкта на картинці.

Кожна така пара має однакову назву і розміщується в папках images для картинок і labels для текстових файлів.

Для автоматичного створення текстових файлів, відповідних до картинок датасету, був використаний ресурс Roboflow, що допомагає згенерувати пари

текстових файлів відповідно до вибраних координат на картинках.

Приклад створення датасету на ресурсі Roboflow представлений нижче (рисунок 4.4, рисунок 4.5).

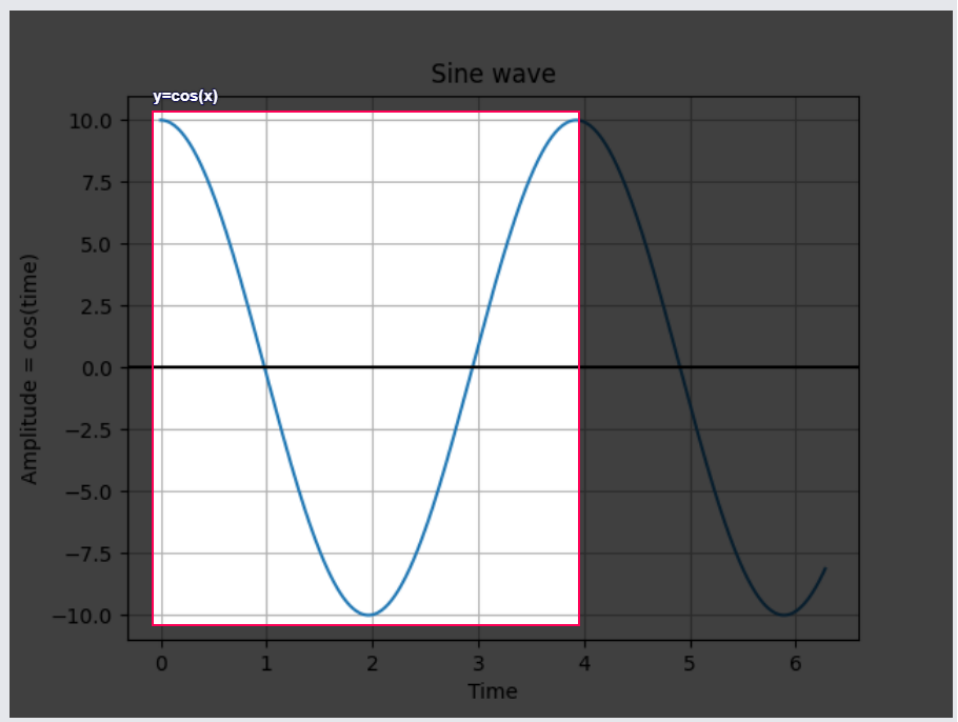

Рисунок 4.4 – Виділення на картинці шуканого елементу та його класифікація

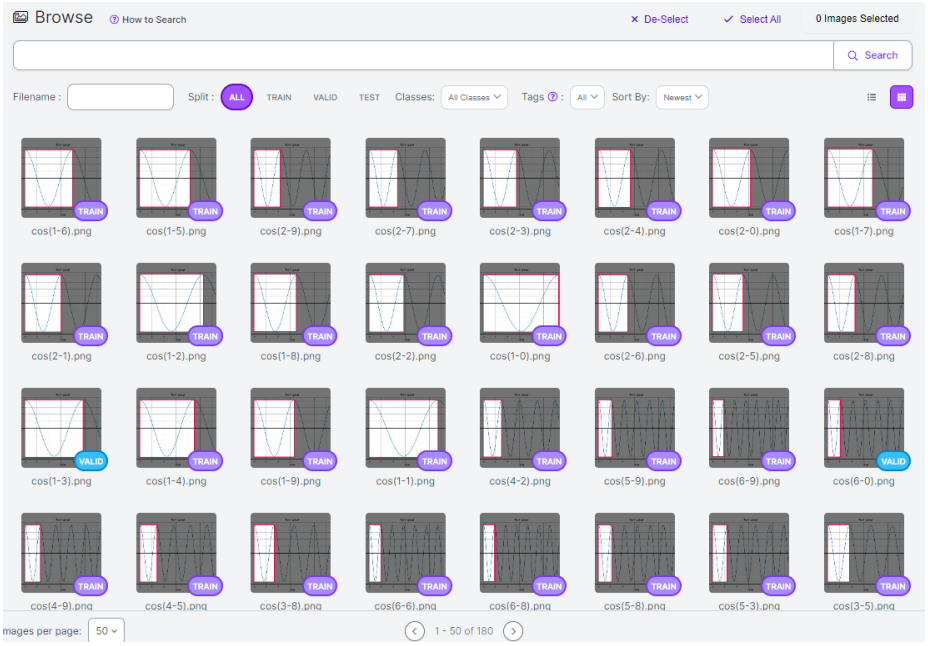

Рисунок 4.5 – Частина кінцевого результату після обробки кожного зображення та розподілення на test і valid вибірки

Можна побачити, що даний ресурс дозволяє не тільки автоматично створити пари з текстових файлів для кожної картинки датасету, а також поділити їх на вибірки що прийматимуть участь у навчанні нейромережі.

Останнім етапом підготовки датасету є створення файлу dataset.yaml, в якому прописані відносні шляхи до картинок (images) та їх назв, текстових файлів (labels). Також цей файл містить в собі назви і кількість класів, на розпізнавання яких має навчитися нейромережа. Назви класів відповідають назвам майбутніх розпізнаних об’єктів на вхідному зображенні.

В процесі розробки програмного продукту дипломної роботи, були використані і готові датасети, а також сформований власноруч, за допомогою ресурсу описаного в даному підрозділі.

4.4 Тренування нейромережі

Наступним етапом у розробці програмного забезпечення дипломної роботи є тренування нейромережі. Цей етап також має за собою достатню кількість підготовки, яку буде описано нижче в даному підрозділі.

За замовчуванням, обчислювальний процес тренування відбувається на процесорі комп’ютера, на якому було запущено тренування, але не кожна машина має достатньо потужний процесор навіть для тренування відносно невеликої моделі з малою кількістю епох. Тому було вирішено реалізувати навчання нейромережі використовуючи відеокарту комп’ютера.

Для того щоб запустити процес навчання нейромережі на відеокарті, необхідно налаштувати CUDA – програмно-апаратну архітектуру паралельних обчислень, яка націлена саме на збільшення обчислювальної продуктивності, тобто на використання графічних процесорів замість центрального процесора.

Тренування на відеокарті обчислювальної машини можна налаштувати дуже простим способом, а саме прописати додатковий параметр в команду запуску тренування через термінал (рисунок 4.6, рисунок 4.7).

Рисунок 4.6 – Запуск тренування на центральному процесорі комп’ютера

Рисунок 4.7 – Запуск тренування на графічному процесорі комп’ютера

Параметр device в команді терміналу відповідає кількості графічних процесорів, на яких буде запущено тренування. В даному випадку, значення 0 означає, що при тренування використовуватиметься тільки один графічний процесор. Тренування нейромережі можна налаштувати і на декілька графічних процесорів (за наявності), але при розробці програмного продукту було використано тільки один.

Наступним кроком тренування мережі є сам запуск тренування і автоматизовані розрахункові процеси моделі. Тренування мережі циклічно за 5 епох представлено на рисунку 4.8.

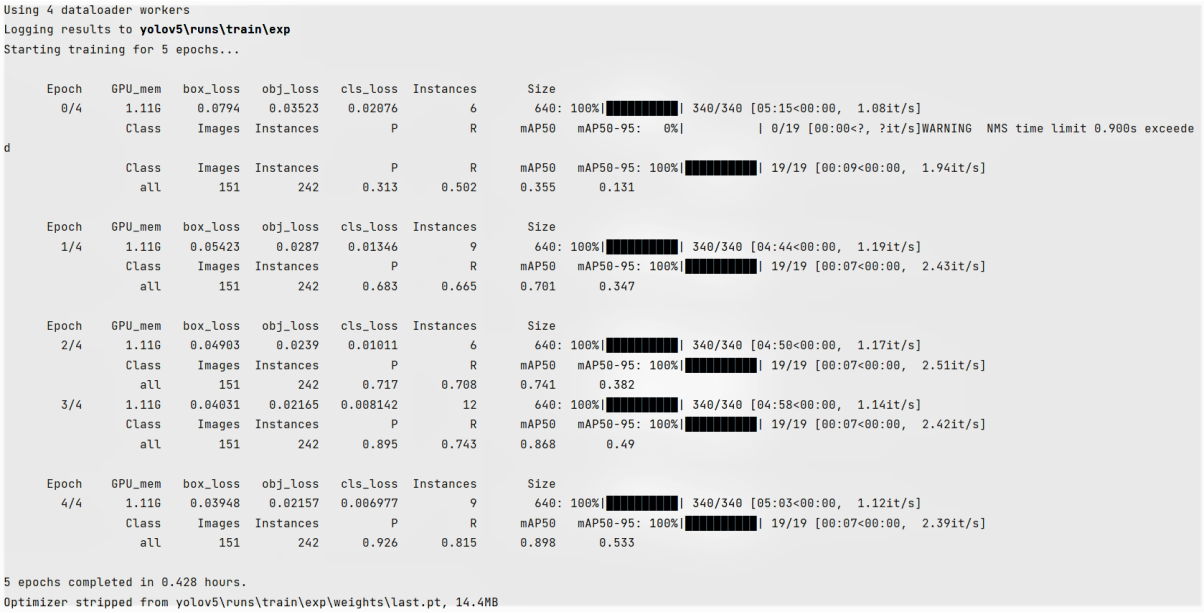

Рисунок 4.8 – Тренування нейромережі

Після завершення процесу тренування моделі, результати тренування що

будуть використані для розпізнавання об’єктів зберігаються в директорії runs\train\exp\weights\ в вигляді файлів best.pt і last.pt. Для розпізнавання зображень будуть задіяні результати, що були записані в файл best.pt, адже в нього записані відібрані найкращі результати тренування з найменшою похибкою.

4.5 Результати тренування

Візуальне представлення тренування нейромережі може бути представлено в вигляді графіків з залежністю номеру епохи від значення похибки.

Оскільки програмний продукт складається з двох мереж нетренованих на окремих датасетах, результати тренування є різними. Результати тренування мережі, що розпізнає відсутність та присутність маски на людині можна побачити на рисунку 4.9.

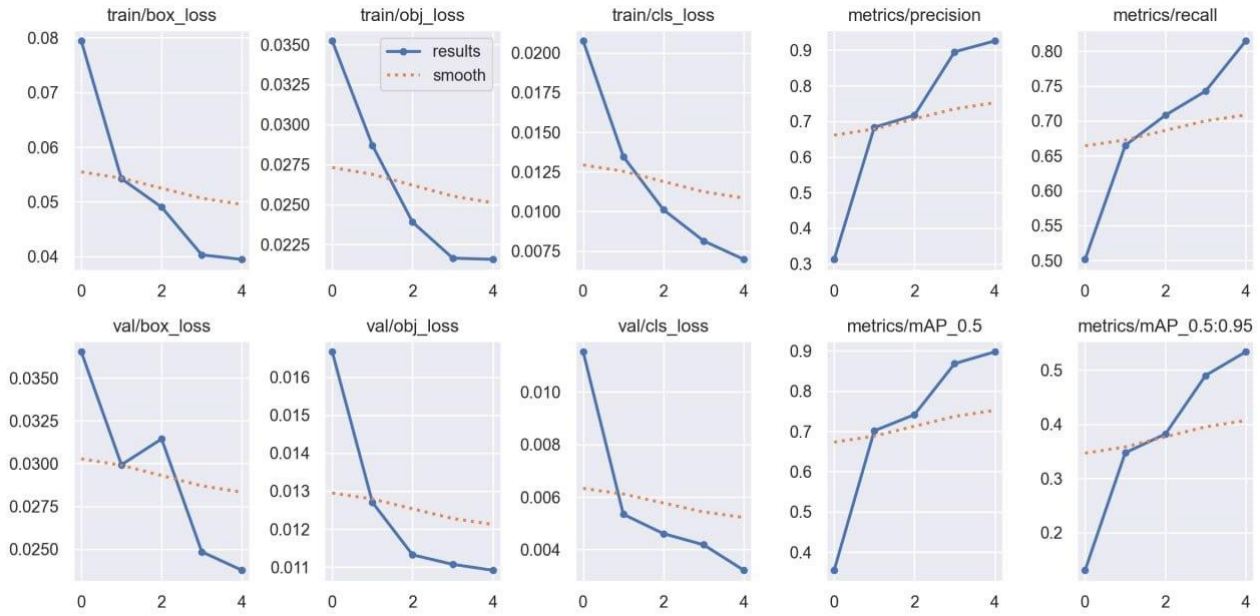

Рисунок 4.9 – Графічне представлення тренування мережі на розпізнавання масок за 5 епох

Можна побачити, як з кожною епохою похибка обчислень прямує до 0, а

точність розпізнавання прямує до 1. В даному випадку ці числа представляють ймовірність розпізнавання, тому найбільшим числом, що репрезентує ймовірність успішного розпізнавання може бути тільки число 1.

Наступне зображення, що репрезентує результати тренування нейромережі на відсутність або присутність медичних масок на людині представляє матрицю, що показує точність розпізнавання об’єктів після процесу тренування (рисунок 4.10).

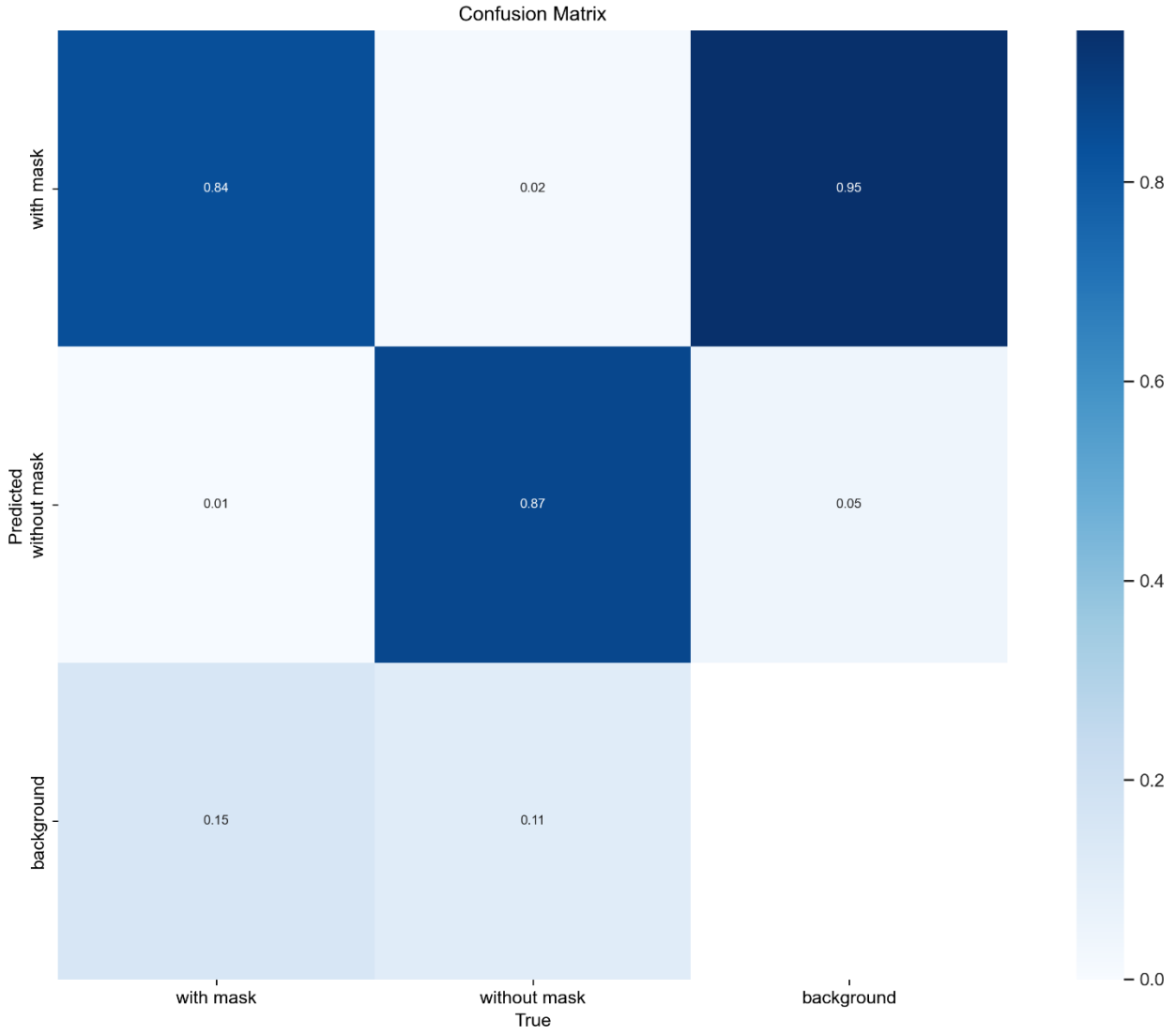

Рисунок 4.10 – Матриця плутанини

Отже за даними з вище наданої інфографіки можна побачити, що тренування нейромережі на розпізнавання масок пройшло успішно і можна

запустити процес розпізнавання. Ця процедура виконується за допомогою файлу best.py, що містить в собі найкращі результати тренування моделі, тобто ті результати що містять найменшу похибку.

Розглянемо другу модель нейромережі, яка тренувалася на розпізнавання двох видів графіків на зображеннях за допомогою власноруч згенерованого датасету з картинок. Нейромережа навчалася на двох графіках – синуса і косинуса. Нижче представлені аналогічні до попередньої моделі результати тренування в вигляді інфографіки (рисунок 4.11).

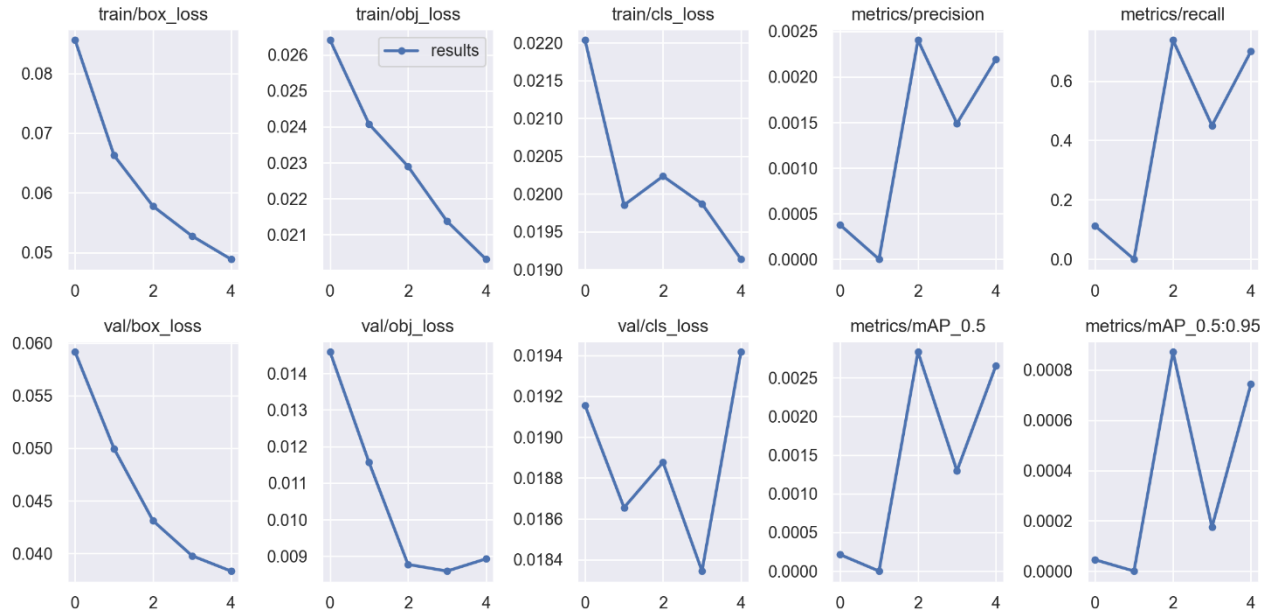

Рисунок 4.11 – Графічне представлення тренування мережі на розпізнавання графіків за 5 епох

Аналізуючи ці графіки, можна дійти висновку, що тренування другої мережі пройшло не дуже вдало, в порівнянні з тренуванням попередньої. Також після завершення тренування, системою не було створено матрицю плутанини, що могла б надати інформацію про ймовірність успішного розпізнавання синуса і косинуса на зображеннях.

Наступним кроком в реалізації програмного продукту є запуск процесу розпізнавання об’єктів на довільному зображенні.

Запуск розпізнавання визначається нижче наведеною командою з термінала (рисунок 4.12).

```
python detect.py --weights runs/train/exp/weights/best.pt --img 640 --conf 0.4 --source ../../downloads/mask1.jpg
```

Рисунок 4.12 – Команда запуску розпізнавання об’єктів на зображенні

Перевірка першої моделі, що була натренована на розпізнавання медичних масок на людині пройшла успішно і надала у відповідь нижче наведений результат (рисунок 4.13).

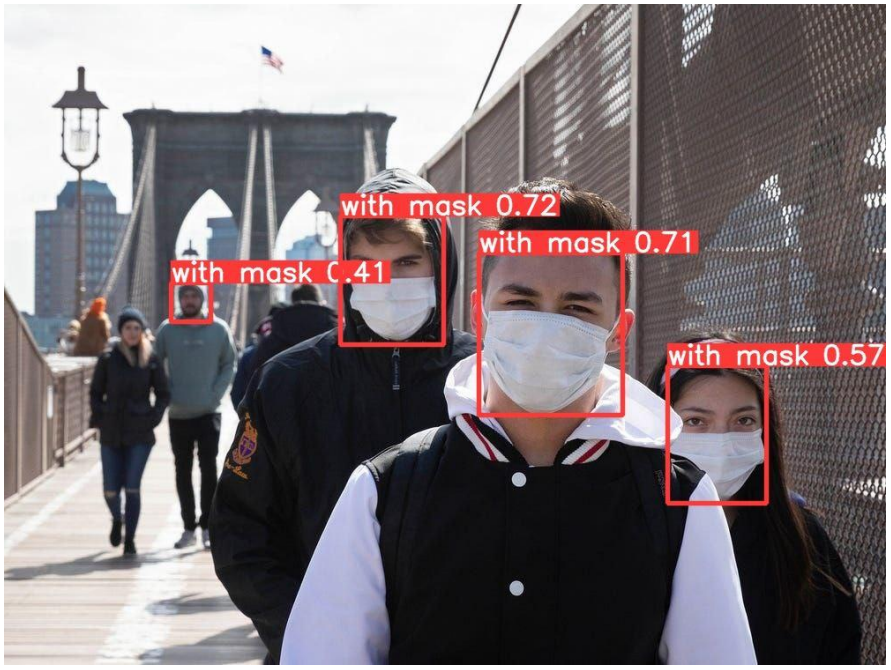

Рисунок 4.13 – Результат розпізнавання масок на людях на основі першої нетренованої моделі

Числа, що відображені біля назви об’єкта означають ймовірність успішного розпізнавання та відповідної класифікації.

Інші зображення, що були використані в тестування даної моделі, також пройшли успішне розпізнавання і збереглися в каталогову структуру моделі.

При перевірці роботи другої моделі, було визначено, що тренування не мало успіху і модель не виконує заданої роботи – розпізнавання синуса і косинуса на зображенні з графіками. Процес обробки запиту розпізнавання

представлений на рисунку 4.14.

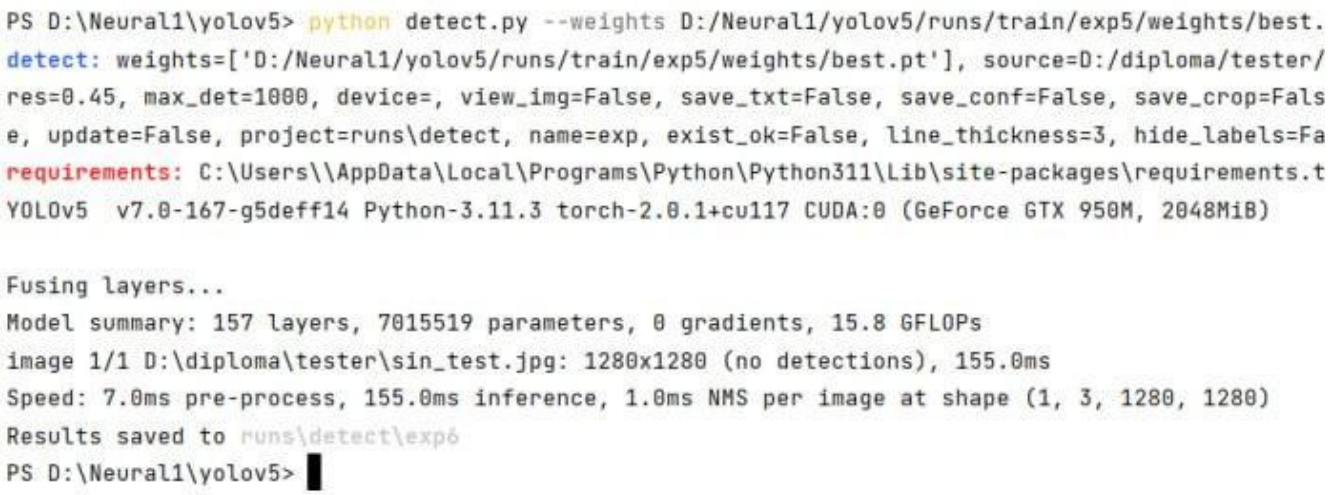

Рисунок 4.14 – Процес розпізнавання графіку на зображенні

Система надає повідомлення no detections, що означає відсутність успішного результату розпізнавання.

Під час аналізу даного результату було визначено, що розробити подібну систему для розпізнавання графіків не є можливим через відсутність достатньої кількості унікальних ознак у вхідних даних для датасету а також неможливість сформувати достатньо велику кількість даних для датасету.

4.6 Розробка візуальної частини додатку

Розробка візуальної частини додатку, що створена для комфортного користування системою була виконана за допомогою мови програмування Python, фреймворку Flask, мови розмітки HTML, таблиці каскадних стилів CSS і мови програмування JavaScript.

Візуальна частина, що формується за допомогою мови розмітки HTML зберігається в окремому каталозі програмного продукту викликається за допомогою мови програмування Python з інтеграцією можливостей фреймворку Flask.

При завантаженні зображення на сторінці додатку відбувається обробка

цього процесу та відповідні повідомлення при невдалому завантаженні файлу.

Додаток аналізує розмір вхідного зображення і повертає повідомлення про завеликий розмір зображення у випадку якщо він перевищує один мегабайт.

Проаналізоване зображення зберігається в каталоговій системі програмного продукту в директорії static, формуючи назву відповідно до дати і часу, коли зображення було проаналізоване.

Висновки до розділу 4

Таким чином, реалізовано структуру веб-додатку. Візуальна частина додатку має зрозумілий вигляд і просту структуру.

Було розроблено багато різних компонентів для трьох нейромережевих моделей і протестовано кожну з них. У ході тестування було виявлено, що модель з розпізнавання графіків не виявилася успішною і були приведені результати тренування даної моделі.

5 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

Для роботи з системою користувачеві необхідно мати доступ до мережі Інтернет та веб-браузер.

5.1 Системні вимоги

Програмний продукт не має особливих системних вимог, оскільки для роботи підходить будь-який веб браузер та операційна система, навіть можна отримати доступ до програми з браузеру телефона та використати завантаження даних з телефона.

5.2 Робота користувача з програмним продуктом

При запуску програми користувачеві відображається головне вікно програми (рисунок 5.1):

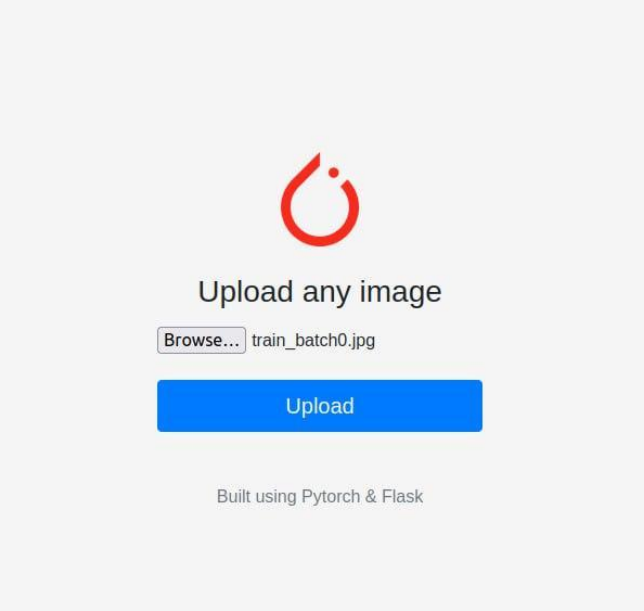

Рисунок 5.1 – Головне вікно програми

Для того, щоб скористатися системою, потрібно натиснути на кнопку «Browse..» та вибрати з носія інформації картинку в форматі jpg або png. При

успішному завантаженні, користувач має натиснути на кнопку «Upload», що переадресовує його на наступну сторінку та відобразить проаналізовану картинку (рисунок 5.2, рисунок 5.3):

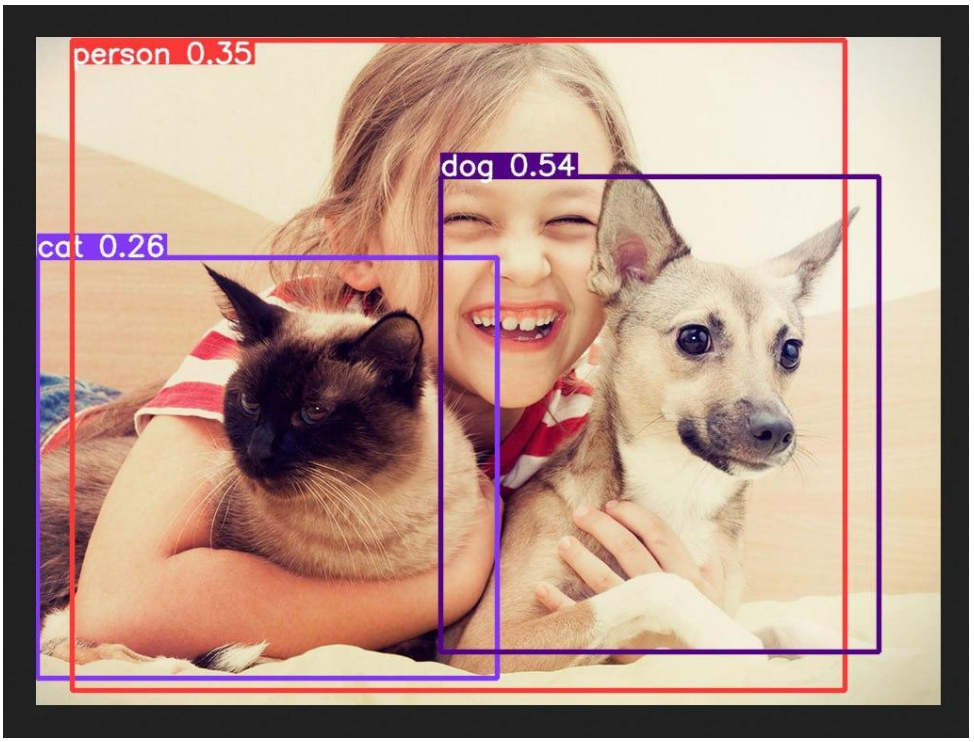

Рисунок 5.2 – Результат розпізнавання стандартної моделі

Рисунок 5.3 – Результат розпізнавання моделі з масками

Висновки до розділу 5

У даному розділі описані системи вимоги до запуску розробленого програмного продукту дипломної роботи – веб-додатку системи розпізнавання зображень на основі нейромережі. Також розглянуто інтерфейс розробленого додатку, описано процес роботи з додатком і наведено приклади роботи застосунку з активацією двох успішних нейромережевих моделей.

Описані вимоги для успішної роботи системи – визначено необхідний формат вхідних даних і візуальний процес обробки вхідних даних, тобто зображення що має бути проаналізовано.

ВИСНОВКИ

У ході виконання даної роботи було проаналізовано можливість розробки системи розпізнавання зображень на основі нейромережі. При створенні та тренуванні нейромережевої моделі для ідентифікування зображень, було розроблено програмний код, який генерує графіки різної частоти і амплітуди щоб мати достатньо різноманітний датасет для того, щоб модель успішно натренувалася. При виконанні тренування, було визначено, що графіки функцій не мають достатньо унікальних ознак для того, щоб створити нейромережеву систему а також неможливо створити достатньо велику кількість даних для успішного тренування нейромережі, тому виконати розпізнавання графіків неможливо.

Незважаючи на те, що розпізнавання графіків не виявилося можливим, створена система чудово розпізнає інші образи, такі як люди, тварини, транспортні засоби а також з високою точністю розпізнає медичні маски на людях – це було зроблено з метою показати, що натренувати нейромережу можливо з даними, які мають достатньо чіткі ознаки для розпізнавання.

При виконанні роботи було:

- розроблено систему, що розпізнає об’єкти на зображеннях на основі нейромережевих алгоритмів з інтерфейсом веб-додатку та можливістю активації системи користувачем;
- досліджено математичну складову роботи згорткової нейромережі;
- проаналізовано алгоритми згорткових нейромереж;
- порівняно проаналізовані алгоритми та визначено найкращий для реалізації програмного продукту дипломної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville Deep Learning. MIT Press, 2016р, 800с.

2. Опис алгоритму YOLO. URL: https://arxiv.org/pdf/1506.02640.pdf/. (дата звернення: 05.05.2023).

3. Васильєв О.М. Програмування мовою Python: Навчальна книга – Богдан, 2019 рік. 504 с.

4. Документація до Flask. URL: https://flask.palletsprojects.com/en/2.3.x/. (дата звернення 25.04.2023).

5. John Murphy. An Overview of Convolutional Neural Network Architectures for Deep Learning. URL: гіперпосилання (дата звернення 15.04.2023).

6. Документація по PyTorch. URL: https://pytorch.org/ (дата звернення 25.04.2023).

7. Документація по Roboflow. URL: https://docs.roboflow.com/ (дата звернення 23.04.2023).

8. A Comprehensive Guide to Convolutional Neural Networks. URL: гіперпосилання (дата звернення 10.04.2023).

9. Convolutional Neural Networks, Explained. URL: гіперпосилання (дата звернення 23.03.2023).

10. Introduction to Convolutional Neural Networks. URL: гіперпосилання (дата звернення 23.03.2023).

11. YOLO object detection using Opencv with Python. URL: гіперпосилання (дата звернення 05.05.2023).

12. CNN Architecture – Detailed Explanation. URL: гіперпосилання (дата звернення 10.04.2023).

13. Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network. URL: гіперпосилання (дата звернення 01.05.2023).

14. Calculating the output size of a Convolutional Layer. URL: гіперпосилання

(дата звернення 01.05.2023).
15. Convolutional Neural Networks. URL: гіперпосилання (дата звернення 23.03.2023).
16. Pooling Layer – Short and Simple. URL: гіперпосилання (дата звернення 25.03.2023).
17. What is PyCharm? URL: https://intellipaat.com/blog/what-is-pycharm/?US (дата звернення 26.03.2023).
18. What is Python Used For? A Beginner's Guide. URL: гіперпосилання (дата звернення 26.03.2023).
19. History of Python. URL: https://www.geeksforgeeks.org/history-of-python/ (дата звернення 26.03.2023).
20. What is Flask? URL: гіперпосилання (дата звернення 27.03.2023).
21. Introduction to the YOLO Family. URL: гіперпосилання (дата звернення 05.05.2023).
22. Roboflow: Converting Annotations for Object Detection. URL: гіперпосилання (дата звернення 23.04.2023).
23. What is PyTorch, and How Does It Work: All You Need To Know. URL: гіперпосилання (дата звернення 25.04.2023).
24. Working with CUDA in PyTorch. URL: гіперпосилання (дата звернення 25.04..2023).

ДОДАТОК А

Система розпізнавання зображень на основі нейромережі

Лістинг програми

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-з91

Аркушів 4

Київ – 2023

webapp.py
```
import argparse
import io
import os
from PIL import Image
import datetime

import torch
from flask import Flask, render_template, request, redirect

app = Flask(name)

DATETIME_FORMAT = "%Y-%m-%d_%H-%M-%S-%f"


@app.route("/", methods=["GET", "POST"])
def predict():
if request.method == "POST":
if "file" not in request.files:
return redirect(request.url)
file = request.files["file"]
if not file:
return

img_bytes = file.read()
img = Image.open(io.BytesIO(img_bytes))
results = model([img])

results.render() # updates results.imgs with boxes and labels
now_time = datetime.datetime.now().strftime(DATETIME_FORMAT)
img_savename = f"static/{now_time}.png"
Image.fromarray(results.ims[0]).save(img_savename)
return redirect(img_savename)

return render_template("index.html")

if name == "main":
parser = argparse.ArgumentParser(description="Flask app exposing yolov5
models")
parser.add_argument("--port", default=5001, type=int, help="port number")
args = parser.parse_args()

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True) 54
```

```
model.eval()
app.run(host="0.0.0.0", port=args.port)

index.html
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
<link rel="stylesheet"
href="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"
integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4i
Y Mj70gZWKYbI706tWS" crossorigin="anonymous">
<style>
.bd-placeholder-img {
font-size: 1.125rem;
text-anchor: middle;
}

@media (min-width: 768px) {
.bd-placeholder-img-lg {
font-size: 3.5rem;
}
}
</style>
<link rel="stylesheet" href="/static/style.css">

<title>yolov5 object detection</title>
</head>
<body class="text-center">
<form class="form-signin" method=post enctype=multipart/form-data>
<img class="mb-4" src="/static/pytorch.png" alt="" width="72"> <h1
class="h3 mb-3 font-weight-normal">Upload any image</h1> <input
type="file" name="file" class="form-control-file" id="inputfile"> <br/>
<button class="btn btn-lg btn-primary btn-block"
type="submit">Upload</button>
<p class="mt-5 mb-3 text-muted">Built using Pytorch & Flask</p>
</form>
<script src="//code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRv
H+ 8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"
```

```
integrity="sha384-wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5
W HdZYu1hA6ZOblgut" crossorigin="anonymous"></script>
<script src="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"
integrity="sha384-B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/m
kk aduKaBhlAXv9k" crossorigin="anonymous"></script>
<script type="text/javascript">
$('#inputfile').bind('change', function() {
let fileSize = this.files[0].size/1024/1024; // this gives in MB
if (fileSize > 1) {
$("#inputfile").val(null);
alert('file is too big. images more than 1MB are not allowed')
return
}

let ext = $('#inputfile').val().split('.').pop().toLowerCase();
if($.inArray(ext, ['jpg','jpeg']) == -1) {
$("#inputfile").val(null);
alert('only jpeg/jpg files are allowed!');
}
});
</script>
</body>
</html>

style.css
html,
body {
height: 100%;
}

body {
display: -ms-flexbox;
display: flex;
-ms-flex-align: center;
align-items: center;
padding-top: 40px;
padding-bottom: 40px;
background-color: #f5f5f5;
}

.form-signin {
width: 100%;
max-width: 330px;
```

```
padding: 15px;
margin: auto;
}

.form-signin .form-control {
position: relative;
box-sizing: border-box;
height: auto;
padding: 10px;
font-size: 16px;
}
```

stolengen.py

```
from pylab import *
import matplotlib.pyplot as plot
import matplotlib.animation as animation
import numpy as np

# Get x values of the sine wave
time = np.arange(0, 2 * pi, 0.01)

# Amplitude of the sine wave is sine of a variable like time
amp = 10
q = 1
for i in np.arange(1, 5, 0.5):

amplitude = np.sin(time * i)
amplifier = amplitude * amp
plot.plot(time, amplifier)
plot.title('Sine wave')
plot.xlabel('Time')
plot.ylabel('Amplitude = sin(time)')
plot.grid(True, which='both')
plot.axhline(y=0, color='k')
plot.savefig('exp(%1.1f).png' % i)
plot.show()
plt.clf()
```