МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

БУШМЕНЬОВ ВЛАДИСЛАВ ЄВГЕНІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
к.т.н., доцент
_____ О. В. Зелінська
«_____» _____ 2024 р.

АВТОМАТИЗАЦІЯ ПАРСИНГУ ФОТОГРАФІЙ З ПУБЛІЧНО ДОСТУПНИХ ДЖЕРЕЛ ІЗ ЗАСТОСУВАННЯМ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Потапова Надія Анатоліївна,
доцент кафедри інформаційних технологій,
доц., к.е.н.

_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Бушменьов В.Є. Автоматизація парсингу фотографій з публічно доступних джерел із застосуванням нейронних мереж. Спеціальність 122 «Комп'ютерні науки». Освітня програма Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця 2024.

У магістерській роботі представлена розробка системи парсингу даних на основі згорткової нейронної мережі, призначенням якої є отримання та обробка набору даних фотографій з публічно доступних джерел. Система відслідковує та класифікує фотографії за ознакою медичних масок, має авторську конфігурацію та операційний блок, призначена для використання в процесі машинного навчання для оцінки отриманих даних. При розробці була використана мова програмування Python, середовище розробки PyCharm та Jupyter Notebook, середовища сервінгу моделі tensorflow та docker.

Магістерська робота складається з вступу, трьох розділів, висновків та додатків. У вступі обґрунтовується актуальність теми, описується мета роботи та постановка завдань дослідження. У першому розділі висвітлюються питання теоретичних засад розробки нейронних мереж та процесів парсингу. У другому розділі висвітлюються питання технології розробки та реалізації моделей парсингу на основі згорткових нейронних мереж. Описуються класифікаційні ознаки датасету для експериментального машинного навчання, наведено математичне обґрунтування формалізації моделі. У третій частині висвітлено практичні результати роботи системи парсингу фотографій, які отримані з публічно доступних пошукових платформ.

118 с., 87 рис., 51 джерело.

Ключові слова: автоматизація, парсинг, нейронні мережі, машинне навчання, Python, JSON.

ABSTRACT

Bushmenov V.Y. Automation of Parsing Photographs from publicly accessible sources using neural networks. Specialty 122 "Computer Science". Educational Program: Data Science. Vasyl Stus Donetsk National University, Vinnytsia 2024.

This master's thesis presents the development of a data parsing system based on a convolutional neural network. Its purpose is to acquire and process a dataset of photographs from publicly accessible sources. The system tracks and classifies photographs based on the presence of medical masks and features a unique configuration and operational block, designed for use in machine learning processes to evaluate the gathered data. The development utilized the Python programming language, PyCharm and Jupyter Notebook development environments, and the tensorflow and docker model serving environments.

The master's thesis consists of an introduction, three chapters, conclusions, and appendices. The introduction justifies the relevance of the topic, describes the objectives of the work, and sets out the research tasks. The first chapter addresses the theoretical foundations of neural network development and parsing processes. The second chapter discusses the technology of development and implementation of parsing models based on convolutional neural networks. It describes the classification features of the dataset for experimental machine learning and provides a mathematical justification for the model's formalization. The third part highlights the practical results of the photo parsing system, obtained from publicly accessible search platforms.

118 pages, 87 pictures, 51 sources.

Keywords: automation, parsing, neural networks, machine learning, Python, JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DOM – (Document Object Model)

APP – додаток (Application)

API – інтерфейс прикладного програмування (Application Programming Interface)

XML – розширювана мова розмітки (Extensible Markup Language)

JSON – нотація об’єктів JavaScript (JavaScript Object Notation)

ML - машинне навчання (Machine Learning)

F1 - оцінка F1 (F1 score)

TF - (Tensorflow)

NP - (NumPy)

JN - (Jupyter Notebook)

CNN - згорткова нейронна мережа (Convolutional neural network)

IDE - інтегроване середовище розробки (Integrated development environment)

HTTP - (The Hypertext Transfer Protocol)

URL - (Uniform Resource Locator)

HTML - (HyperText Markup Language)

REST - (Representational state transfer)

UUID - (Universally unique identifier)

GPU - графічний процесор (Graphics Processing Unit)

AI - штучний інтелект (Artificial Intelligence)

CPU - центральний процесор (Central Processing Unit)

ВСТУП

Актуальність теми. В сучасному житті все більше і більше сфер застосовують штучні нейронні мережі. Проекти, що базуються на них, надають можливість автоматизувати або спростити завдання, які раніше вважались недосяжними для комп'ютерів і були виключною прерогативою людей. З появою практичного застосування нейронних мереж, людство отримало потужний інструмент для полегшення розпізнавання зображень, перекладу текстів, прогнозування різних явищ та багато інших завдань. Проте, незважаючи на те, що нейронні мережі стають все більш поширеними, є ще багато завдань, які потребують подальшого вирішення.

Мета магістерської роботи полягає у розробці та реалізації автоматизованої системи парсингу даних у вигляді фотографій з публічно доступних джерел на основі моделі нейронної мережі.

Завдання магістерської роботи. Для реалізації поставленої мети було поставлено ряд завдань:

- дослідити теоретичні засади створення та функціонування нейронних мереж, з метою розробки програмної моделі для розпізнавання наявності медичної маски на обличчі людини;
- визначити основні класифікаційні ознаки відбору даних при формуванні датасету, що мають бути отримані і можуть містити помилки візуалізації;
- провести алгоритмізацію процесу парсингу, з метою його подальшої реалізації для формування датасету вхідних даних отриманих з публічно доступних джерел;
- розробити модель нейронної мережі, використовуючи машинне навчання з учителем з метою покращення датасету;
- розробити та програмно реалізувати додаток парсингу фотографій;
- провести тренування моделі та оцінити результати на основі метрик: мінімальної втрати, максимальної користувацької точності для тренувального та валідаційного наборів даних, часу тренування.

Об’єкт дослідження – автоматизована система парсингу сайтів та моделі класифікації фотографій, отриманих із публічно доступних джерел за обраною тематикою.

Предмет дослідження – методи, засоби та програмні модулі, що дозволяють аналізувати візуальні дані за допомогою нейронних мереж, функціональні властивості парсингу фотографій та їх автоматизація з використанням проекту з відкритим програмним кодом Selenium.

Наукова новизна дослідження полягає в тому, що:

вперше отримано:

автоматизовану систему парсингу фотографій отриманих з публічно доступних джерел із застосуванням нейронних мереж, основною функціональною можливістю якої є реалізація формування датасету за категоризацією ознаки («маска на обличчі») з покращеними елементами усунення похибок розпізнавання.

вдосконалено:

алгоритмічні та програмні підходи щодо реалізації процесу парсингу фотографій покращенням вхідного датасету через деталізацію початкових класифікаційних ознак за результатами глибокого машинного навчання.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі висвітлено теоретичні засади побудови та використання нейронних мереж, проаналізовано топологію нейронних мереж, розглянуто основні елементи процесу вебскрапінгу та напрями його використання.

У другому розділі наведено аналіз технологічних рішень для проєктування моделі, а також її математичне обґрунтування.

У третьому розділі наведено результати розробки та експериментальної оцінки автоматизованої системи парсингу фотографій.

Практичне значення роботи полягає в:

– можливості використання системи при розпізнаванні образів людини в медичній масці в різних погодних та інших умовах;

– застосуванні моделі для аналізу масиву фотографій, у вигляді якого представлені знімки людей в лікарнях, чи різних публічних місцях під час епідемічних захворювань;

– можливості використання програмного забезпечення для здійснення збору та аналізу фотографій з публічних ресурсів за обраною тематикою.

Апробація результатів даного дослідження здійснена на II Міжнародній науково-практичній конференції «Прикладні аспекти сучасних міждисциплінарних досліджень» (м. Вінниця, 24 листопада 2023 року) та IV Всеукраїнській науково-практичній конференції «Комп'ютерні технології обробки даних» (м. Вінниця, 8 грудня 2023 року).

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ ПОБУДОВИ ТА ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ

1.1. Сутність та основні поняття теорії нейронних мереж

Перш ніж розглядати методи аналізу візуальних даних, важливо розібратися в особливостях використання нейронних мереж. Нейронна мережа є ключовою складовою програми для аналізу візуальних даних і відповідає за розмітку цих даних.

Штучні нейронні мережі - це обчислювальні системи, які були створені на основі біологічної природи нейронних мереж, які зустрічаються у мозку людей та багатьох тварин. В основі цих систем лежить ідея про наявність штучних нейронів, які окремо є простими і примітивними, але коли вони з'єднуються в мережу, вони можуть виконувати складні операції. Сигнал з одного нейрона може передаватись на тисячі інших нейронів у системі. Навчання штучних нейронних мереж відбувається через ітеративну активацію певних зв'язків між нейронами та коригування їх ваги. Цей процес навчання використовує зворотний зв'язок для коригування помилок і покращення результатів.

В нейроні найважливішою є активаційна функція, яка може функціонувати як вимикач з двома станами - 0 і 1 (або -1 і 1), залежно від вхідного сигналу. Ця функція імітує активність біологічного нейрону. Один із прикладів активаційної функції - сигмоїдна функція, яка показана на рисунку 1.1.

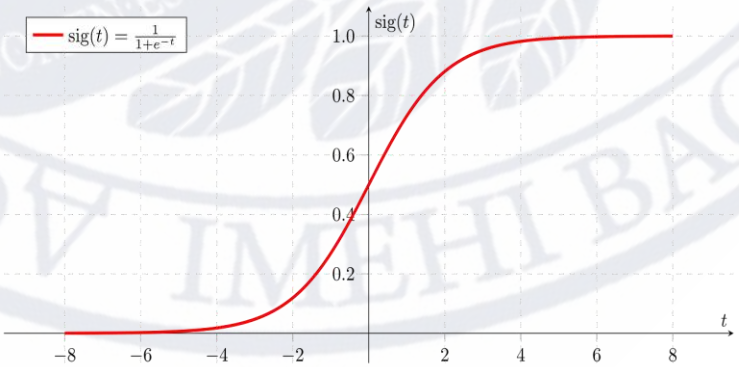


Рисунок 1.1 – Сигмоїдна функція

Нейрони (персептрони) отримують зважені входи, які сумуються і піддаються активаційній функції, після чого вони дають певний результат. Ваги нейронної мережі представляють собою числові коефіцієнти, які використовуються для зважування вхідних сигналів. Кожен нейрон має свої ваги, які відображають важливість кожного входу для розрахунку вихідного значення. Ваги в мережі коригуються під час навчання, щоб оптимізувати роботу мережі і досягти бажаного результату. Зважений вхід у вузол має наступний вигляд: x1*w1+x2*w2+x3*w3+b, де x - входи, w - їх ваги, а b - зсув. [1] На рисунку 1.2 та рисунку 1.3 проілюстровані ефекти від зміни ваг та зсуву.

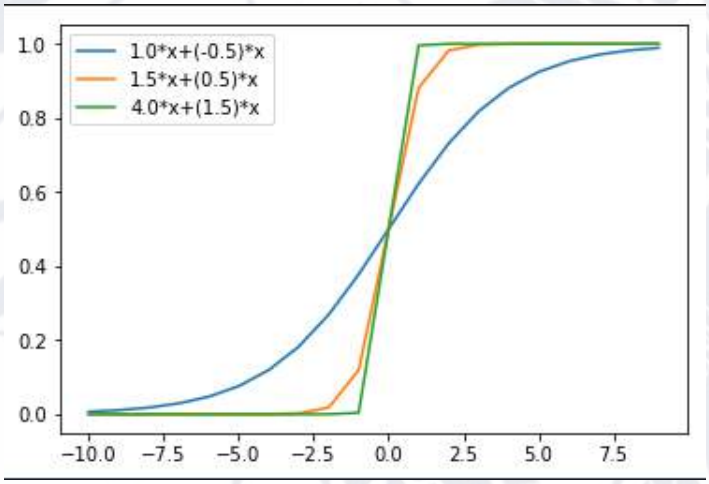


Рисунок 1.2 – Ефект від зміни ваг, де w1 набуває значень від 1 до 4, а w2 - від 0.5 до 1.5

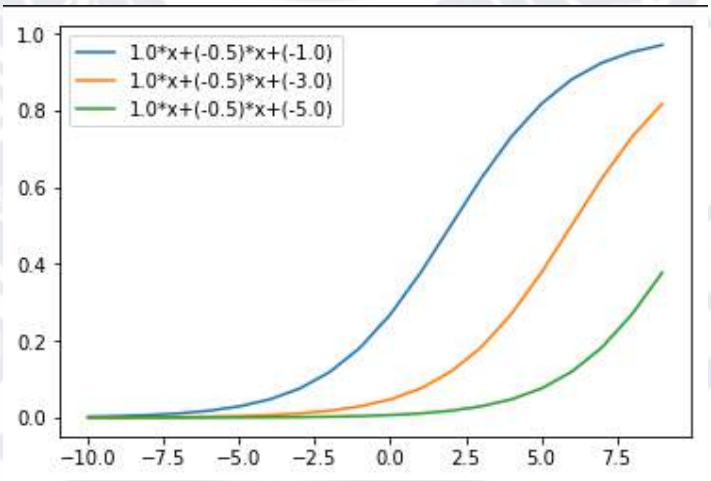


Рисунок 1.3 – Ефект від зміни зсуву, де b набуває значень від -1 до 5

Великі та складні штучні нейронні мережі володіють, як правило, і великими обчислювальними можливостями. шари в нейромережі – це групи нейронів, які працюють разом і відповідають за різні етапи обробки інформації. Вони забезпечують нейромережі можливість адаптуватися до різних задач. [2]

Характерною властивістю штучної нейронної мережі є її здатність до навчання, що полягає у виробленні правильної реакції на подані їй різні вхідні сигнали.

Штучний нейрон може бути представлений у такий спосіб на рисунку 1.4.

Рисунок 1.4 – Модель штучного нейрона

У процесі навчання ваги стають такими, що під час надходження вхідних сигналів мережа виробляє відповідні необхідні вихідні сигнали.

Для досягнення задовільного рівня точності, глибокі навчальні програми вимагають доступу до величезного обсягу навчальних даних та потужних обчислювальних ресурсів. Ці ресурси були важко доступними для програмістів до появи великих обсягів даних та хмарних обчислень. Завдяки можливості програмування глибокого навчання створювати складні статистичні моделі безпосередньо з ітеративного процесу, воно може створювати точні прогнозуючі моделі з великого обсягу непозначених неструктурованих даних. Це особливо важливо в контексті розширення Інтернету речей, оскільки більшість даних, які генеруються людьми та машинами, є неструктурованими та не мають міток.

Кількість необхідних даних також залежить від типу навчання, що використовується в нейронній мережі. З’єднані між собою нейрони утворюють

штучну нейронну мережу. Таким чином, штучна нейронна мережа – пара (M, V), де M – множина нейронів; V – множина зв'язків. Структура мережі задається у вигляді графа, у якому вершини є нейронами, а ребра являють собою зв'язки (з'єднання). Кожен нейрон мережі має вхідні ланцюги, причому їхня кількість є довільною для кожного нейрона. За кількістю шарів в структурі штучної нейронної мережі вони поділяються на одношарові та багатошарові.

Одношарова нейронна мережа - найпростіша мережа складається з групи нейронів, що утворюють шар, як показано на рисунку 1.5. Ліві вершини виконують функцію розподілу вхідних сигналів, але не здійснюють окремих обчислень, тому їх не можна вважати окремим шаром. Це позначено колами, щоб відрізняти їх від обчислювальних нейронів, позначених квадратами. Кожен елемент з множини входів X з'єднується з кожним нейроном з урахуванням окремої ваги, і кожен нейрон видає зважену суму вхідних сигналів. Всі з'єднання показані для узагальнення, але в конкретній мережі деякі з'єднання можуть бути відсутні. Можуть мати місце також з'єднання між виходами і входами нейронів в шарі. [3,4]

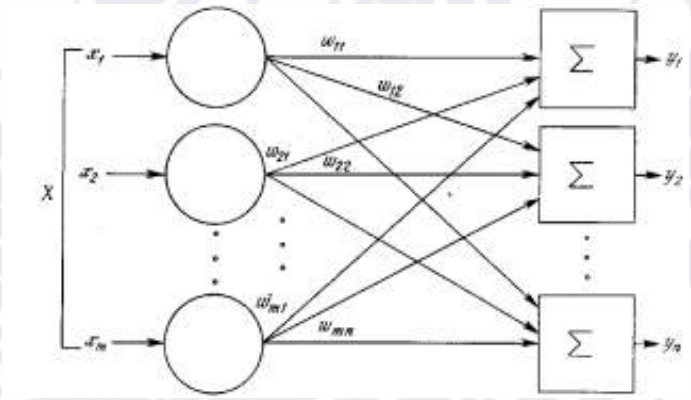


Рисунок 1.5 – Одношарова штучна нейронна мережі

Багатошарова нейронна мережа - більш велика та складна штучна нейронна мережа володіє, як правило, і великими обчислювальними можливостями. Хоча створені мережі всіх конфігурацій, які тільки можна собі уявити, пошарова організація нейронів копіює шаруваті структури певних відділів мозку. Виявилося, що такі багатошарові мережі мають більші

можливості, ніж одношарові, і в останні роки були розроблені алгоритми для їх навчання. Багатошарові мережі можуть утворюватися каскадами прошарків. Вихід одного шару є входом для наступного шару. [3]

Багатошарова штучна нейронна мережа включає в себе: [5]

1. Вхідний шар: Вхідний шар приймає дані ззовні, наприклад, зображення або текст, і передає їх у наступні шари. Вхідний шар не змінює дані, а лише служить точкою входу для них.

2. Приховані шари: шари звичайних нейронів, які передають сигнали від входу до виходу. Їх входом служить вихід попереднього шару, а вихід - входом наступного шару.

3. Вихідний шар: Вихідний шар формує результат, який нейромережа передбачає на основі вхідних даних. Результат може бути класифікацією, числовим значенням або іншою інформацією, залежно від типу задачі.[5]

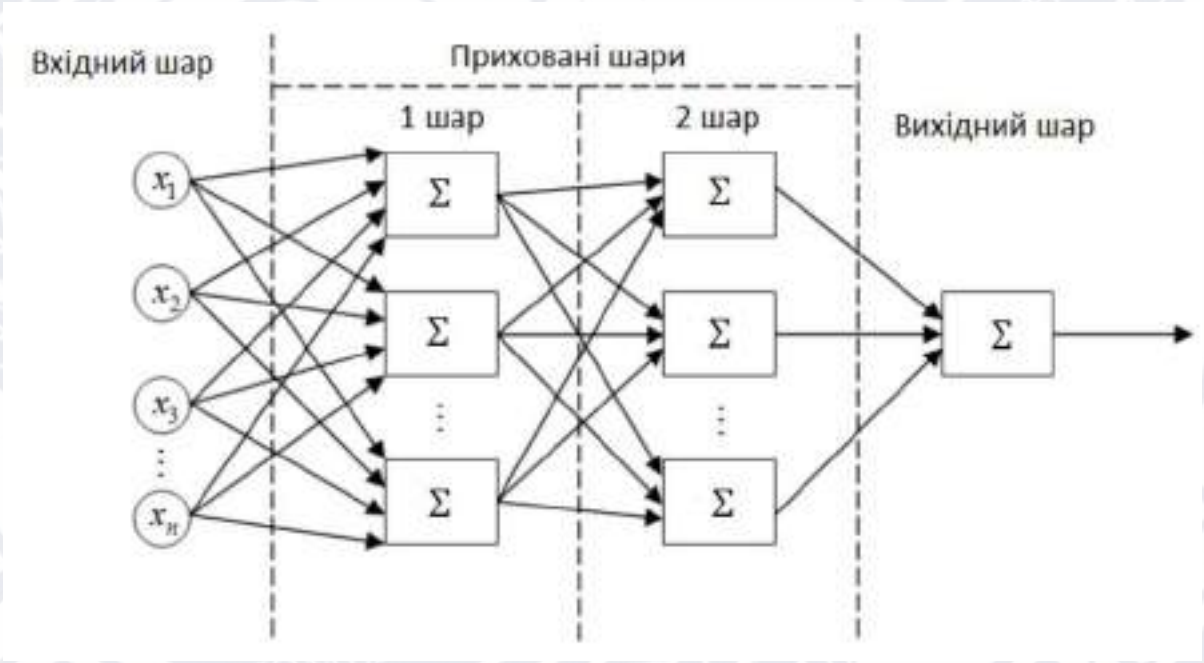


Рисунок 1.6 – Шари багатошарової нейронної мережі

Зв’язки скеровані від вхідних прошарків до вихідних називаються аферентними. Зв’язки в зворотному напрямку називаються еферентними. В більшості мереж кожен нейрон прихованого прошарку отримує сигнали від всіх нейронів попереднього прошарку чи від нейронів вхідного прошарку. Після

виконання операцій над сигналами, нейрон передає свій вихід до всіх нейронів наступних прошарків, забезпечуючи передачу вперед (feedforward) на вихід. При зворотному зв’язку, вихід нейронів прошарку скеровується до нейронів попереднього прошарку зображено на рисунку 1.7.

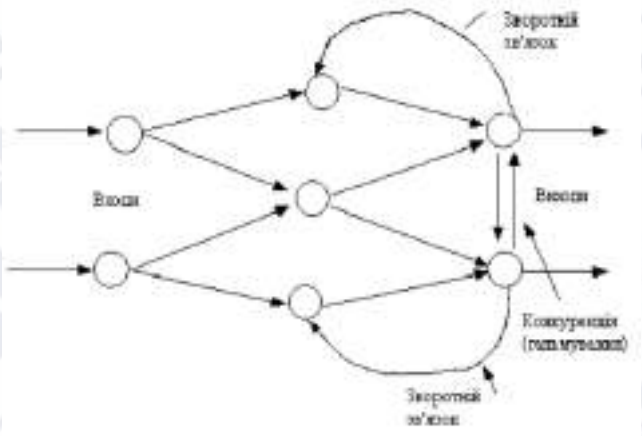


Рисунок 1.7 – Зворотній зв’язок у нейронній мережі

Напрямок зв’язків нейронів має значний вплив на роботу мережі. Більшість програмних нейромереж дозволяють користувачу додавати, вилучати та керувати з’єднаннями як завгодно. Корегуючи параметри, можна налаштувати зв’язки як на посилення так і на послаблення величини сигналів. [6]

1.2. Топологія нейронних мереж

Топологія характеризує організацію вузлів нейромереж. Основними типами є повнозв’язні мережі, багатошарові та нейромережі з локальними зв’язками. Розглянемо деякі можливі топології нейромереж: [8]

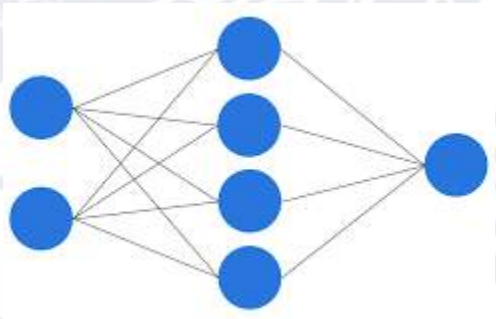


Рисунок 1.8 – Мережа прямого зв’язку

Нейронна мережа прямого зв’язку - це тип штучної нейронної мережі, де інформаційний потік перетікає лише в одному напрямку, від вхідних вузлів до вихідних. Нейронна мережа прямого зв’язку (FF) відповідає таким твердженням:

- всі вузли повністю з’єднані
- активація проходить від вхідного рівня до вихідного, без
- зворотних циклів
- існує один шар між входом і виходом (прихований шар)

Зазвичай, для навчання цього типу мереж використовується метод зворотного поширення. Один з її підвидів - RBF - використовує радіальну базисну функцію як функцію активації замість логістичної функції. Основна відмінність полягає в тому, що логістична функція відображає значення у діапазоні від 0 до 1, відповідаючи на питання "так чи ні". Це корисно для задач класифікації та прийняття рішень, але не ефективно для обробки неперервних значень. [8]

Автоенкодери - використовуються для класифікації, кластеризації та стиснення ознак. Під час навчання прямолінійних зворотних нейронних мереж для класифікації, в основному потрібно подавати X прикладів у Y категоріях і очікувати активацію однієї з Y вихідних комірок. Це називається "навчанням з учителем". З іншого боку, автоенкодери можуть бути навчені без нагляду. Їх структура - коли кількість прихованих комірок менша, ніж кількість вхідних комірок (і кількість вихідних комірок дорівнює кількості вхідних комірок) і після навчання автоенкодера вихідні дані якомога ближчі до вхідних - змушує автоенкодери узагальнювати дані і шукати загальні патерни.

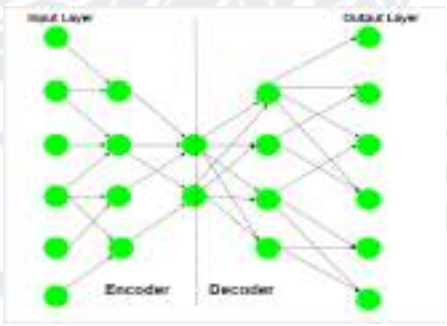


Рисунок 1.9 – Мережа AE

Мережа радіально базисних функцій (RBF) у математичному моделюванні – це штучна нейронна мережа, яка використовує радіальні базисні функції у якості функції активації. Виходом мережі є лінійна комбінація радіальних базисних функцій входу та параметрів нейрона. Мережі радіальних базисних функцій мають багато застосувань, зокрема, такі як апроксимацію функції, прогнозування часових рядів, задачі класифікації та керування системою. [9]

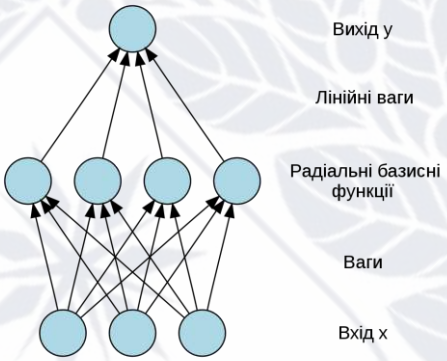


Рисунок 1.10 – Архітектура мережі радіальних базисних функцій

Нейронна мережа DFF - бере свій початок із 90-х років, є FF мережею з більш ніж одним прихованим шаром, довгий час їх навчання було майже неможливим через надто стрімке зростання часу навчання від кожного додаткового шару і вже ближче до 2000-х років розроблено підходи до навчання, які дозволили більш ефективно тренувати цей тип мереж, що вкупі з більш потужною апаратною частиною переросло в бум нейронних мереж. Оскільки DFF є, по суті, варіацією FF мережі – вона виконує ті самі цілі, однак робить це з набагато кращими результатами. [8]

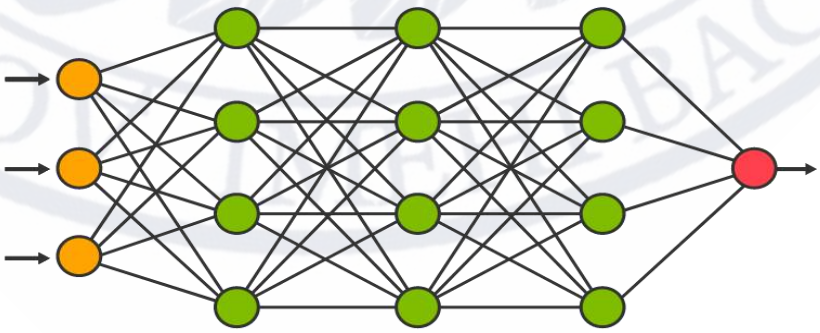


Рисунок 1.11 – Нейронна мережа DFF

Рекурентна нейронна мережа (RNN - Recurrent Neural Network) - це тип штучної нейронної мережі, яка використовується для обробки послідовних даних. У відміну від нейронних мереж прямого зв'язку, RNN має зворотні зв'язки між вузлами, що створює можливість передачі інформації з попередніх кроків обробки до наступних кроків. RNN широко використовуються для завдань, таких як машинний переклад, розпізнавання мови, генерація тексту, аналіз настрою, прогнозування часових рядів та інші завдання, де важлива взаємодія між елементами послідовних даних і залежності в часі.

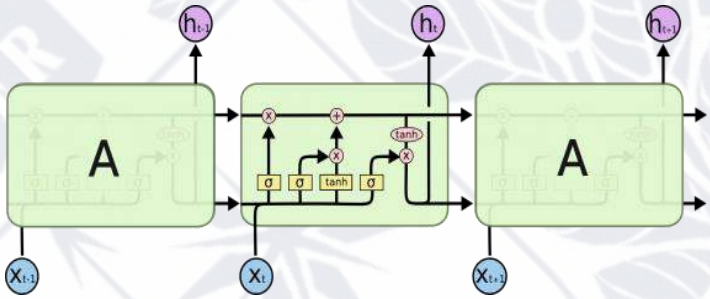


Рисунок 1.12 – Рекурентна нейронна мережа

LSTM (довготривала короткотривала пам'ять) - це тип рекурентної нейронної мережі, здатний навчатися довгостроковим залежностям. LSTM спеціально розроблені для усунення проблеми довгострокової залежності. Вони спеціалізуються на запам'ятовуванні інформації протягом тривалих періодів часу, тому практично не потребують навчання. Всі рекурентні нейронні мережі мають ланцюжкову структуру повторюваних модулів нейронної мережі. У стандартних РНС цей повторюваний модуль має просту структуру, наприклад, один шар tanh. [10]

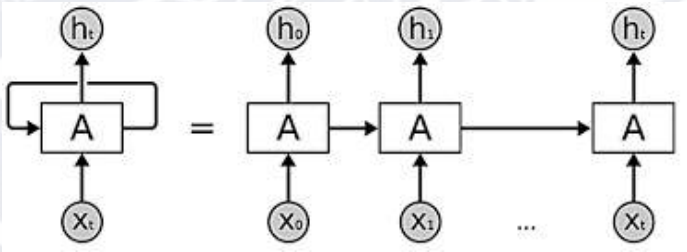


Рисунок 1.13 – LSTM

GAN (Generative Adversarial Network) - представляє велику родину

подвійних мереж, які складаються з генератора та дискримінатора. Вони постійно намагаються обдурити один одного - генератор намагається створити якісь дані, а дискримінатор, отримуючи зразки даних, намагається відрізнити згенеровані дані від зразків. Постійно еволюціонуючи, цей тип нейронних мереж може генерувати реалістичні зображення, якщо ви здатні підтримувати баланс тренування між цими двома мережами. [8]

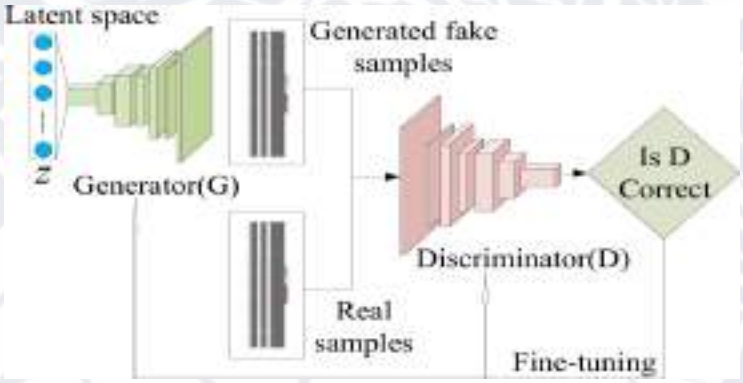


Рисунок 1.14 – Generative Adversarial Network

1.3. Процес вебскрапінгу для парсингу сайтів

Вебскрапінг представляє собою автоматизований процес вилучення великих обсягів даних з веб-сайтів. Більшість цих даних первинно представлена в неструктурованому форматі HTML, який подальше трансформується в структуровані дані, що можуть бути відображені у таблицях або базах даних для подальшого використання в різноманітних додатках. Екзистує множина методик реалізації вебскрапінгу для вилучення даних з вебсайтів, які включають використання онлайн-служб, специфічних API, або навіть розробку власного коду для вебскрапінгу від нуля. Багато великих вебплатформ, зокрема Google, Twitter, Facebook, StackOverflow та інші, пропонують API, які надають можливість отримання їхніх даних у структурованому форматі. Це вважається оптимальним варіантом, але існують інші сайти, які або не надають користувачам доступ до великого обсягу даних у структурованій формі, або не є настільки технологічно розвинутими. В таких випадках вебскрапінг стає вельми

ефективним інструментом для вилучення даних з вебсайту.

Для запуску процесу вебскрапінгу спочатку надаються URL-адреси цільових вебсайтів. Після цього скрапер завантажує повний HTML-код цих сайтів, а більш модернізований скрапер може навіть екстрагувати всі пов'язані CSS та Javascript елементи. Наступним кроком є витягування потрібних даних з HTML-коду та представлення цих даних у форматі, вказаному користувачем. Зазвичай дані представляються у вигляді таблиць Excel або файлів CSV, проте існує можливість збереження даних у інших форматах, наприклад, у файлах JSON. Отже, цей процес має значний науковий та практичний потенціал, надаючи можливість аналізувати великі обсяги інформації з веб-сайтів, що є невід'ємною складовою сучасних досліджень у галузі інформатики та аналізу даних. Вебскрапінг як технологічний процес має обширний спектр застосувань у різноманітних галузях промисловості та наукових дослідженнях. Розглянемо декілька ключових напрямків його використання:

- Моніторинг ціноутворення: веб-скрапінг дає змогу компаніям автоматично збирати дані про ціни на продукцію, як від власних виробничих ліній, так і від конкурентів. Аналіз цих даних допомагає формувати ефективні цінові стратегії з метою оптимізації прибутків.

- Маркетингові дослідження: використання веб-скрапінгу для масового збору високоякісних даних може бути неоціненним інструментом для аналізу ринкових тенденцій, споживчих уподобань та прогнозування стратегічного руху компанії в майбутньому.

- Моніторинг новин: скрапінг новинних веб-ресурсів може забезпечувати компанії актуальними даними про поточні події, які мають прямий чи непрямий вплив на їх діяльність. Це особливо важливо для організацій, для яких щоденний моніторинг новин є критично важливим.

- Аналіз сентиментів: здійснюється шляхом веб-скрапінгу даних з соціальних медіа-платформ та інших ресурсів з метою визначення загального настрою споживачів щодо продукції або бренду компанії. Це може підтримувати інформованість компанії та адаптацію її стратегій до споживчих уподобань.

- Електронний маркетинг : веб-скрапінг дозволяє здійснювати масовий збір контактних даних потенційних клієнтів з різних веб-ресурсів для подальшої реалізації маркетингових кампаній через електронну пошту.

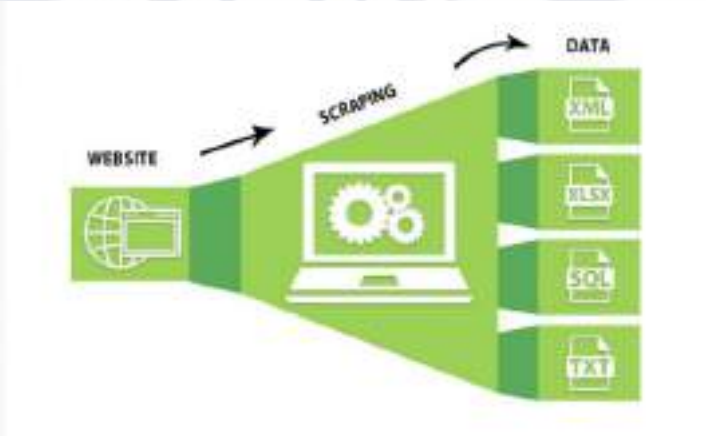


Рисунок 1.15 – Процес вебскрапінгу

Кожен з вищевказаних напрямків демонструє широкий спектр можливостей використання вебскрапінгу як інструменту для збору та аналізу даних у сучасному бізнес-середовищі та наукових дослідженнях. [20]

Вебскрапінг, у порівнянні з людиною, працює набагато швидше та ефективніше. Комп'ютерна програма-парсер:

- швидко обійде тисячі вебсторінок;
- акуратно відокремить технічну інформацію від «людської»;
- безпомилково відбере потрібне і відкине зайве; ефективно упакує кінцеві дані в необхідному вигляді.

Отже, вебскрапінг – це метод, який дозволяє автоматично отримувати велику кількість інформації з веб-сайту, що може заощадити час та зусилля. Використання подібних додатків є дуже актуальною темою сучасних розробок програмних засобів. Здається, що все діє досить просто, але є одна проблема. Вона полягає у тому, що близько 95% всіх вебсторінок або вебсайтів – це унікальні ресурси. Тому виникає задача по створенню унікальних додатків під кожний вебресурс. Але, з точки зору програміста, в унікальних додатках можливе часткове використання того-самого способу отримання даних або частково алгоритмів. Розглянемо, що собою являє програма або додаток для

вебскрапінгу. Як правило, це комп'ютерна програма, що імітує поведінку людини в Інтернеті, або з'єднуючись з вебсервером напряму по протоколу HTTP, або за допомогою керування повноцінним веббраузером. Webscraping включає в себе завантаження та вилучення. Спочатку завантажується сторінка (що робить браузер, коли ви переглядаєте сторінку), після цього можна добувати потрібну інформацію. Інформаційні технології та кібербезпека може бути проаналізовано, переформатовано, його дані скопійовані в електронну таблицю тощо. Веб-скрапери, як правило, беруть щось зі сторінки, щоб використати це для інших цілей та потреб. [21]

Існують методи, за допомогою яких певні веб-сайти намагаються запобігати вебскрапінгу. Наприклад, виявлення та заборона ботів для сканування (перегляду) своїх сторінок. У відповідь на це були розроблені системи, які спираються на використання методів аналізу об'єктної моделі документа, комп'ютерного бачення та обробки тексту на природній мові, щоб імітувати пошук людини, щоб дозволити збирати вміст веб-сторінок для автономного синтаксичного аналізу. Методи боротьби з вебскрапінг: [22]

- HTML-fingerprint - процедура фільтрації розпочинається з глибокого аналізу заголовків HTML. Заголовки можуть містити інформацію, яка дозволяє визначити, чи є відвідувач ботом чи людиною, а також чи є він небезпечним чи безпечним. Сигнатури заголовків порівнюються із безперервно оновленою базою даних, яка включає понад 10 мільйонів відомих варіантів.

- Репутація IP-адреси - збираються дані IP з усіх атак клієнтів сайту. Візити з IP-адрес, які мали історію використання в агресивних діях, розглядаються з підозрою та мають більшу ймовірність для подальшого детального вивчення.

- Аналіз поведінки - моніторинг способів взаємодії відвідувачів із веб-сайтом може виявити аномальні моделі поведінки, такі як підозріло агресивна частота запитів та нелогічні моделі перегляду. Це сприяє ідентифікації ботів, які виступають у ролі людських відвідувачів.

- Прогресивні виклики – використовуються набори викликів, включаючи

підтримку файлів cookie та виконання JavaScript, для фільтрації ботів та мінімізації хибнопозитивних результатів.

- CAPTCHA - виклик CAPTCHA може відсіяти ботів, які намагаються виступати у ролі людей. Представляє собою систему, яка розроблена для визначення, чи є користувачем сайту людина, чи комп'ютер. Це важливо для запобігання автоматизованим атакам зловмисників, таким як спам або зловживання веб-ресурсами.

З моменту народження Інтернету та виникнення безмежної кількості веб-ресурсів, навіть відносно недавно, здобути доступ до інформації на веб-сторінках стало ключовим завданням для багатьох сфер діяльності. З поглибленням інтернетізації усіх аспектів життя суспільства та розвитком онлайн-технологій, збільшилася необхідність у вилученні, аналізі та використанні даних з вебресурсів. Аналіз ефективності методів вебскрапінгу для парсингу сайтів є важливою та актуальною темою у світі інформаційних технологій. Вебскрапінг, або отримання даних з вебсторінок, стало одним із ключових інструментів для збору інформації з Інтернету. Цей процес дозволяє автоматично отримувати та структурувати дані зі власних або сторонніх веб-сайтів з мінімальною участю людини. Аналіз ефективності методів вебскрапінгу стає надзвичайно важливим для багатьох сфер застосування, таких як бізнес-аналітика, наукові дослідження, конкурентний аналіз, моніторинг ринку та багато інших. Залежно від обраного методу скрапінгу та його параметрів може бути суттєва різниця у якості отриманих даних, швидкості збору та витраті ресурсів.

У цьому розділі ми глибоко проаналізуємо різні методи вебскрапінгу та їхню ефективність, дослідимо основні виклики та обмеження, що виникають під час збору даних з веб-сайтів. Ми також дослідимо різноманітні сценарії застосування вебскрапінгу та з'ясуємо, які методи найкраще відповідають певним завданням та умовам. Проведення цього дослідження має великий практичний вагомість і може бути корисним як для бізнесу, який прагне отримати конкурентну перевагу на ринку, так і для наукової громадськості, що прагне отримати нові знання та відкриття на основі великих обсягів даних,

доступних в мережі Інтернет. Найпоширеніші техніки, які використовуються для вебскрапінг, включають: [23]

- Ручне копіювання та вставка: ручне копіювання та вставка даних із вебсторінки у текстовий файл або електронну таблицю представляють собою найбільш елементарну форму вебскрапінгу. Незважаючи на високий рівень розвитку технологій вебскрапінгу, інколи немає альтернативи ручному аналізу та копіюванню даних людиною, особливо у випадках, коли вебсайти явно забороняють використання машинної автоматизації для збору даних. Цей метод може бути відносно часомістким і піддається помилкам через людський фактор, але іноді єдиним можливим у випадках, коли автоматизовані методи заборонені або неефективні. Ручне копіювання та вставка, як правило, не вимагають специфічних технічних навичок або програмного забезпечення, і це може бути вигідним для окремих користувачів або малих проектів. Однак, цей підхід має обмежену масштабованість та ефективність, що робить його менш привабливим для великих датасетів та проектів вищого рівня.

- Порівняння текстових шаблонів: метод порівняння текстових шаблонів відіграє важливу роль у процесі вебскрапінгу. Зокрема, універсальні засоби пошуку текстових рядків, такі як команда UNIX grep, або можливості порівняння регулярних виразів, що надаються мовами програмування (наприклад, Perl або Python), можуть бути використані для витягування інформації з вебсторінок ефективним та гнучким способом. Даний підхід базується на визначенні специфічних шаблонів у текстовому контенті веб-сторінки та їх подальшому використанні для вилучення цільових даних. Технологія порівняння текстових шаблонів дозволяє визначати регулярні вирази, які відповідають певним шаблонам, та використовувати ці вирази для пошуку та вилучення інформації з вебресурсів. Специфічні алгоритми та інструменти, що використовують регулярні вирази для пошуку текстових шаблонів, можуть бути адаптовані до різноманітних задач вебскрапінгу. Наприклад, вони можуть бути використані для автоматичного вилучення даних з вебсайтів, що мають стандартну структуру, або для здійснення складного пошуку в розгалужених вебресурсах.

Проте, підхід порівняння текстових шаблонів може мати обмеження в контексті складноструктурованих або динамічно згенерованих вебсторінок, які вимагають більш складних методів аналізу та вилучення даних. Окрім того, необхідність розробки точних та ефективних регулярних виразів може вимагати високого рівня технічних знань та досвіду в області програмування.

- Програмування на рівні HTTP (HyperText Transfer Protocol) - є фундаментальним підходом до доступу до веб-ресурсів, що дозволяє отримувати як статичні, так і динамічні веб-сторінки. Цей метод передбачає використання сокет-програмування для створення та надсилання HTTP-запитів до віддаленого веб-сервера, що, в свою чергу, повертає відповідь, яка містить веб-контент. Основна ідея програмування HTTP полягає у формуванні та надсиланні запитів до веб-сервера з дотриманням специфікацій протоколу HTTP. Це включає в себе визначення методу запиту (наприклад, GET або POST), заголовків запиту, та інших параметрів, що вказують на те, як сервер повинен обробляти запит та повертати відповідь. У контексті веб-скрапінгу програмування HTTP може бути вкрай корисним для реалізації автоматичного доступу до веб-даних. Проте, його ефективність може бути обмеженою у випадках, коли веб-сторінки захищені від автоматичного доступу або вимагають взаємодії на клієнтській стороні для отримання повного контенту.

- Парсинг HTML - велика кількість веб-сайтів мають сторінки, які формуються динамічно зі структурованих даних, зазвичай із бази даних. Щоб отримати ці дані, необхідно розібрати HTML-код сторінок, що відображаються в браузері. HTML - це мова розмітки, яка визначає структуру та вигляд веб-сторінок. Створення обгортки (коду для парсингу): Для того, щоб автоматично збирати дані з подібних сторінок, потрібно створити програму або скрипт, яка визначає шаблони та правила для видобування інформації з цих сторінок. Ця програма називається обгорткою. Обгортки розробляються для розпізнавання структурованих даних на веб-сторінках та вилучення їх у певний формат, зазвичай у вигляді таблиць або бази даних. Після того, як обгортка визначає та вилучає дані з вебсторінок, ці дані можуть бути перетворені у реляційну форму,

щоб забезпечити зручний доступ та аналіз. Одним із способів цього перетворення є використання мов запитів, таких як XQuery та HTQL, які дозволяють аналізувати структуровані дані на сторінках HTML і здійснювати їх трансформацію. Обгортки передбачають, що сторінки мають загальні шаблони та можуть бути ідентифіковані за певною схемою URL. Це допомагає автоматизувати процес збору даних з багатьох сторінок.

- Парсинг DOM - це процес отримання та обробки динамічного вмісту вебсторінок, який генерується клієнтськими скриптами, використовуючи об'єктну модель документа (Document Object Model, DOM). DOM - це інтерфейс для доступу до структури та змісту веб-сторінок. Він представляє вебсторінку як дерево об'єктів, де кожен елемент сторінки, такий як текст, зображення або посилання, має свій об'єктний представник. DOM надає можливість програмам змінювати структуру та вміст сторінок, а також взаємодіяти з ними. Потім ми отримуємо динамічний зміст. Деякі вебсторінки генерують свій вміст за допомогою клієнтських скриптів, таких як JavaScript. Цей вміст може бути недоступним у початковому HTML-коді сторінки. Для отримання цього динамічного вмісту програми можуть використовувати веббраузери, такі як Internet Explorer або Mozilla Firefox, або спеціальні бібліотеки для керування браузерами. Після отримання сторінки веббраузер або програма, що керує браузером, створює структуру дерева DOM, яка представляє всі елементи сторінки та їхні взаємозв'язки. Програми можуть аналізувати це дерево для отримання конкретних частин сторінки, таких як текст, таблиці або дані з форм. Для навігації та видобування даних з DOM-дерева можна використовувати мови запитів, такі як Xpath. Xpath - це мова запитів, яка дозволяє точно вибирати елементи з DOM за допомогою шляхів або виразів. Вона дуже корисна для отримання конкретних даних зі складних структур вебсторінок. Отже, парсинг DOM - це процес отримання, аналізу та використання динамічного вмісту веб-сторінок за допомогою об'єктної моделі документа та мов запитів, таких як Xpath. Вертикальна агрегація - це підхід до збору даних та автоматизації, де декілька компаній створюють спеціалізовані платформи для збору і аналізу

інформації в певній галузі або вертикалі. Важливою особливістю цього підходу є використання "ботів" або автоматизованих програм, які працюють без прямої участі людини та без необхідності роботи з конкретними цільовими сайтами. Кілька компаній спільно розробляє спеціалізовану платформу для збору даних в певній галузі або галузі (вертикалі). Ця платформа має вбудовані інструменти та ресурси, призначені для збору, обробки та аналізу інформації. На цих вертикальних платформах створюються та керуються "ботами". Боти - це автоматизовані програми, які виконують завдання зі збору даних. Компанії програмують ці боти для виконання конкретних завдань, пов’язаних із збором інформації. Основною особливістю вертикальної агрегації є те, що цей процес автоматизований і не вимагає прямого втручання людини. Боти працюють самостійно, без потреби в постійному нагляді чи втручанні оператора. Важливою перевагою вертикальної агрегації є те, що вона дозволяє отримувати інформацію з різних джерел, не потребуючи прямого доступу до конкретних цільових сайтів. Це робить процес більш ефективним та менш залежним від змін в структурі цільових веб-ресурсів.

- Розпізнавання семантичних анотацій - це процес виявлення та використання метаданих, семантичних розміток та анотацій на вебсторінках для пошуку конкретних даних. Ця техніка має важливе значення в контексті веб-скрапінгу і дозволяє ефективно видобувати дані зі сторінок в більш інтелектуальний та структурований спосіб. Семантичні анотації та метадані - це додаткова інформація, яка може бути включена в HTML-код веб-сторінки. Вони надають структуру та значення даним на сторінці. Семантичні розмітки, такі як Microformats або RDFa, дозволяють вказувати, які саме дані знаходяться на сторінці і як вони пов’язані між собою. Вебскрапери можуть аналізувати HTML-код сторінок для виявлення семантичних анотацій та метаданих. Це включає в себе пошук певних тегів та атрибутів, які вказують на наявність семантичних розміток. Після розпізнавання семантичних анотацій вебскрапери можуть використовувати цю інформацію для більш точного та структурованого видобування даних. Наприклад, якщо сторінка містить інформацію про події,

семантичні анотації можуть допомогти визначити дати, місця та інші деталі кожного події. У деяких випадках семантичні анотації можуть бути збережені та керуватися окремо від вебсторінок. Це означає, що інформація про схему даних та інструкції для скраперів можуть бути збережені на рівні системи, і скрапери можуть отримати доступ до цих даних перед розпочатком видобування інформації зі сторінок. Розпізнавання семантичних анотацій робить процес веб-скрапінгу більш інтелектуальним та ефективним, оскільки дозволяє точніше розуміти структуру та значення даних на веб-сторінках. Такий підхід допомагає отримувати більш якісну та структуровану інформацію зі скраплених джерел. [23]

В сучасному світі, де обсяги даних невпинно зростають, збільшується і потреба у витягненні цінної інформації з вебсайтів. Комп'ютерний зір та парсинг HTML стали потужними інструментами для досягнення цієї мети. У цьому розділі ми докладно розглянемо ці методи та їхню ефективність у процесі аналізу вебсторінок. Комп'ютерний зір та його роль в аналізі вебсторінок: Комп'ютерний зір відіграє ключову роль у виявленні та інтерпретації візуальної інформації на вебсторінках. Завдяки алгоритмам машинного навчання та нейронним мережам, системи комп'ютерного зору можуть розпізнавати об'єкти, текст, зображення, кольори та інші візуальні атрибути на сторінках. Це дозволяє не лише видобувати текстову інформацію, але й аналізувати графічні компоненти, які можуть мати важливе значення для подальшого аналізу.

- Машинне навчання для автоматизованого аналізу вмісту - відкриває широкий спектр можливостей для розробки інтелектуальних алгоритмів, спроможних автоматично визначати та розуміти структури даних на веб-сторінках. Ці алгоритми, завдяки використанню машинного навчання, здатні розпізнавати різні структурні елементи, такі як таблиці, списки, форми тощо, та вилучати інформацію з них. Одним з ключових аспектів є аналіз тексту та зображень. Алгоритми машинного навчання можуть бути навчені розпізнавати та аналізувати текстовий вміст, витягаючи ключові дані та відносини між ними. Зображення, що розміщені на вебсторінках, також можуть бути оброблені з

використанням технік комп'ютерного зору, щоб виділити інформацію з графічних елементів. Використання машинного навчання у контексті аналізу вмісту дозволяє підвищити точність та ефективність процесу вилучення даних з веб-сайтів. Це важливо для різноманітних галузей, включаючи бізнес-аналітику, наукові дослідження, моніторинг ринків та багато інших областей, де діє потреба в обробці та аналізі інформації з великої кількості джерел.

- Нейромережеве навчання для визначення структури вебсторінок - основною є виявлення різних зон інформації на сторінці, таких як заголовки, текстові блоки, зображення та посилання. Це дозволяє автоматизовано визначати, де саме розміщена корисна інформація, та ефективно вилучати її. Застосування глибокого навчання у цій сфері важливе для різноманітних областей, включаючи аналітику даних, створення пошукових систем, бізнес-інтелект та інші. Використання згорткових нейронних мереж і RNN дозволяє автоматизувати і поліпшити процес визначення структури вебсторінок, що робить його більш точним та ефективним.

- Використання NLP для визначення контексту та вмісту - розпізнавання та інтерпретація природної мови (NLP) є важливою галуззю штучного інтелекту, яка знаходить широке застосування у визначенні та аналізі текстового вмісту вебсторінок. В даному контексті, NLP виступає в якості потужного інструменту для визначення корисної інформації, що міститься на веб-сторінках, та її подальшого аналізу. Використання NLP в аналізі веб-сторінок дозволяє здійснювати більш глибокий та контекстний розгляд текстового вмісту. За допомогою NLP можна визначати сутність тексту, розпізнавати ключові слова та фрази, аналізувати синтаксис та семантику речень. Це дозволяє точніше визначати значущі дані та взаємозв'язки між ними на веб-сторінках. NLP допомагає визначити контекст і зміст текстового вмісту веб-сторінок, що дає можливість зрозуміти та інтерпретувати інформацію на більш високому рівні. Це може включати визначення тематики сторінки, виявлення ключових понять, виділення важливих даних та встановлення взаємозв'язків між різними елементами інформації.

- Вебскрапінг з використанням хмарних платформ - В сучасному цифровому світі, де обсяги даних безперервно зростають, процес вебскрапінгу стає важливим завданням для багатьох галузей. Щоб забезпечити швидку та ефективну обробку великих обсягів вебданих, на допомогу приходять хмарні платформи. Хмарні платформи надають потужні обчислювальні ресурси, які можна використовувати для масштабування процесу вебскрапінгу. Це означає, що ви можете здійснювати збір та обробку вебданих на великому масштабі, навіть якщо ви працюєте з численними джерелами або великими обсягами інформації.

Переваги хмарного вебскрапінгу:

- Масштабованість: Хмарні платформи дозволяють легко масштабувати вебскрапінг на потребу, додавати обчислювальні ресурси за потреби та зменшувати їх, коли це необхідно.

- Висока швидкість: Завдяки розподіленій обробці та швидкому з’єднанню до мережі Інтернет, хмарні платформи забезпечують високу швидкість вебскрапінгу, що дозволяє збирати дані швидше та ефективніше.

- Автоматизація: Хмарні сервіси також можуть надавати інструменти для автоматизації вебскрапінгу, що полегшує процес збору даних та зменшує необхідність в ручному втручанні.

Використання хмарних платформ для вебскрапінгу особливо корисне при зборі великих обсягів даних з глобального вебу. Це може бути важливо для бізнес-аналізу, моніторингу трендів, аналітики соціальних мереж, а також наукових досліджень та інших областей, де доступ до великої кількості даних є ключовим.

- Гібридний вебскрапінг (оптимізація через комбінацію технологій). В сфері вебскрапінгу, де точність та ефективність є критичними факторами, гібридний підхід стає потужним інструментом для оптимізації процесу збору даних. Гібридний вебскрапінг включає в себе комбінацію різних методів, технологій та підходів з метою підвищення точності та ефективності збору веб-даних. Гібридний вебскрапінг дозволяє об’єднувати сили різних методів та

технологій для досягнення кращих результатів. Це особливо корисно, коли веб-сайти мають різні структури, та один підхід не може забезпечити необхідну точність та стабільність. Компоненти гібридного підходу:

- Традиційні методи: Гібридний підхід може включати в себе використання традиційних методів вебскрапінгу, таких як парсинг HTML-сторінок чи робота з API.

- Машинне навчання: Використання алгоритмів машинного навчання дозволяє автоматизувати процес визначення структури даних та підвищити точність вилучення інформації з вебсторінок.

- Комп'ютерний зір: Використання комп'ютерного зору для аналізу зображень на сторінках, що дозволяє визначати та виділяти інформацію з графічних елементів.

- Семантичний аналіз: Визначення семантичних анотацій та взаємозв'язків між елементами даних на сторінці.

- Гібридний вебскрапінг знаходить застосування в різних сферах, включаючи бізнес-аналітику, аналіз ринків, збір інформації для досліджень та розвідки конкурентів. Використання гібридного підходу дозволяє отримувати більш точні та надійні дані з вебсерверів, що є важливим для прийняття обґрунтованих рішень.

Таким чином, сутність задачі дослідження полягатиме в розробці програмної моделі, яка буде розпізнавати наявність медичної маски на обличчі людини використовуючи нейронні мережі, що навчаються з вчителем.

ВИСНОВКИ ДО РОЗДІЛУ 1

Сучасні підхід досліджень поведінки систем та об’єктів все більше спирається на біологізацію процесів, що передбачає використання моделей на інструментів здатних імітувати живі істоти. Саме в основі таких механізмів, покладено розробку та використання штучних нейронних мереж. Даний підхід обумовлений припущенням того, що штучна модель відрізняється від оригіналу і тільки оцінка та дослідження зможуть її наблизити до бажаних характеристик.

Сучасна теорія інтелектуального аналізу даних невід’ємно пов’язана із розробкою та використанням нейронних мереж різної топології, основними серед яких є: мережа прямого зв’язку, мережа автоенкодер, мережа радіальних базисних функцій, нейронна мережа DFF, рекурентна нейронна мережа.

Нейронна мережа використовується в процесі машинного навчання, що обгрунтовує використання технологій для їх накопичення та обробки, зокрема, вебскрапінгу для парсингу сайтів. Аналіз методів та технологій в процесі вебскрапингу для парсингу сайтів показав, що одними із ефективних є: парсинг DOM, парсинг HTML, машинне навчання для автоматизованого аналізу вмісту, нейромережеве навчання для визначення структури веб-сторінок, гібридний веб-скрапінг.

На сьогодні суспільно-соціальні процеси відрізняються проявами збурень та ризикових викликів, одними із яких, є епідемії та надзвичайні стани. В такий спосіб суспільство корегує свою поведінку шляхом набуття різних прямих та непрямих ознак для запобігання ризику. Однією із них при епідемії COVID стала необхідність носити медичну маску, тому задача відрізнити людину в масці та без маски набуває своєї актуальності, а розробка моделі для розпізнавання наявності медичної маски на обличчі потребує використання нейронної мережі, що навчається з вчителем.

РОЗДІЛ 2
ТЕХНОЛОГІЇ РОЗРОБКИ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПАРСИНГУ ФОТОГРАФІЙ

2.1. Особливості технологій парсингу

Під час реалізації поставленого завдання були використані наступні основні технології, які зображені на рисунку 2.1.

Рисунок 2.1 – Основні технології, які були використані при досягненні поставленої задачі

Реалізація нейронної мережі:

- Мова програмування Python
- Обчислювальні бібліотеки мови програмування Python : numpy, keras, tensorflow, scikit-learn
- Бібліотеки для візуалізації даних : matplotlib та seaborn
- Середовище розробки програмного коду : PyCharm та Jupyter Notebook
- Для прискорення математичних обчислень була використана технологія NVIDIA CUDA.

Реалізація парсингу:

- Мова програмування Python
- Парсингові бібліотеки мови програмування Python : selenium, webdriver
- Бібліотеки мови програмування Python для обробки фотографій : pillow

- Середовища сервінгу моделі : tensorflow, docker.
- Середовище розробки програмного коду : Jupyter Notebook

Обираючи мову програмування для вирішення практичного завдання я спирався на те, що вона повинна відповідати наступним вимогам:

1. Легкість у використанні: мати простий та зрозумілий синтаксис.

2. Багата екосистема: мати широку підтримку і велику кількість бібліотек, спеціалізованих на машинному навчанні та нейронних мережах, які надають потужні та ефективні інструменти для розробки моделей нейронних мереж.

3. Гнучкість: дозволяти легко експериментувати з різними архітектурами нейронних мереж та параметрами. Надавати можливість швидко створювати, змінювати та перевіряти моделі, щоб сприяти більш швидкому процесу розробки та прототипуванню.

4. Мати велику спільноту: велика спільнота надасть можливість знайти велику кількість ресурсів, підручників, документації та підтримки в разі потреби. За рахунок цього, можна швидко отримати відповіді на свої питання та розв’язати проблеми, які виникають під час розробки моделей нейронних мереж.

5. Інтеграція з іншими інструментами: мова програмування повинна легко інтегрується з іншими мовами програмування та інструментами.

6. Розширюваність: підтримка можливостей розширення функціональності за допомогою власних модулів та пакетів.

7. Велика кількість датасетів та популярних моделей: повинен існувати широкий вибір готових датасетів для навчання моделей нейронних мереж. Це надасть змогу знайти безліч популярних архітектур нейронних мереж, реалізованих у вигляді готових модулів, які можна використовувати у дослідженні.

8. Візуалізація та аналіз даних: потрібний великий вибір бібліотек для візуалізації та аналізу даних. Це дозволить легко відображати та аналізувати результати навчання моделей нейронних мереж, що сприяє візуальному розумінню даних.

9. Підтримка інтерактивного програмування: дозволить ефективно експериментувати, виконувати код по частинах та візуалізувати проміжні результати навчання моделей.

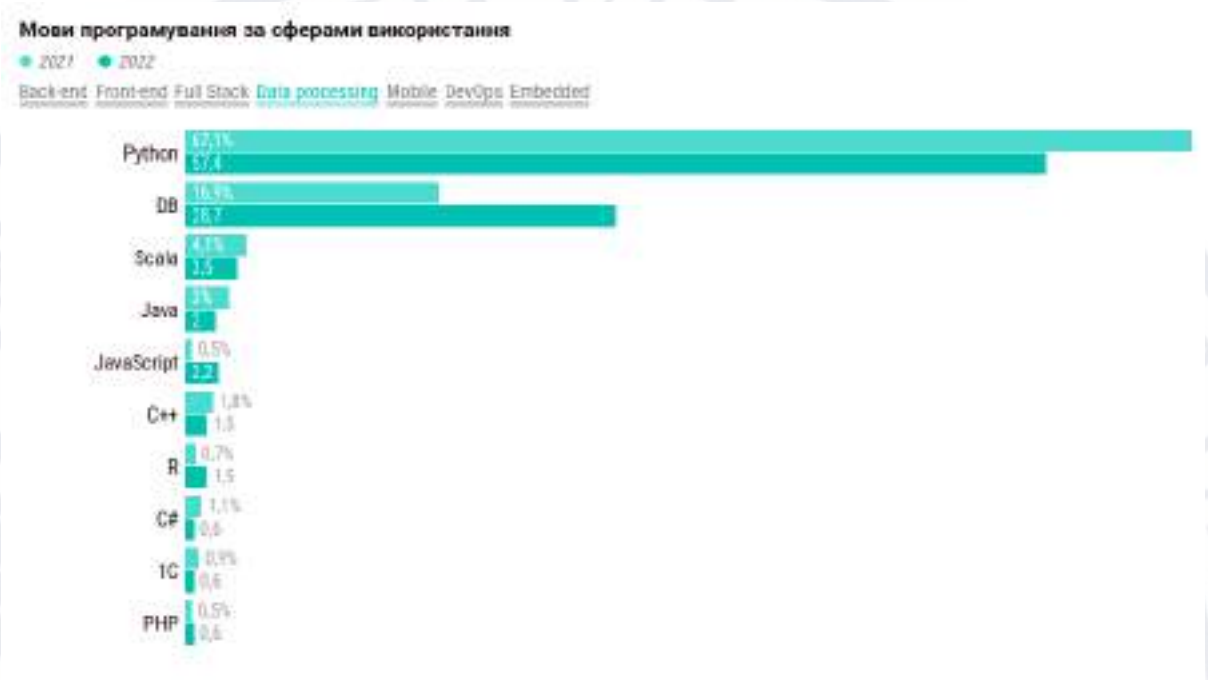


Рисунок 2.2 – Популярність мов програмування в сфері аналізу даних

Python - інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв’язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. [11]

Python має розширення та модулі для різних сфер програмування, що дозволяє розробникам використовувати готові рішення для широкого спектру завдань. При розробці була акцентована увага на використання бібліотек, підключення яких дозволило програмно реалізувати парсинг.

TensorFlow – це бібліотека для високопродуктивних числових обчислень. Він використовується в різних галузях науки. TensorFlow – це, по суті, структура

для визначення та виконання обчислень, які включають тензори, які є частково визначеними обчислювальними об’єктами, які зрештою видають результат. TensorFlow спрощує розробку та навчання моделей машинного навчання за допомогою API високого рівня, таких як Keras. Він також пропонує різні рівні абстракції, що дозволяє вибрати найкращий підхід для вашої моделі. TensorFlow також дозволяє розгортати моделі машинного навчання в різних середовищах, включаючи хмару, браузер і ваш пристрій. Переваги бібліотеки : [13, 14, 15, 16]

1. Гнучкість: TensorFlow може працювати з різноманітними типами даних, включаючи зображення, мовні дані тощо. Він підтримує багато типів моделей, включаючи глибокі нейронні мережі, рекурентні нейронні мережі, конволюційні нейронні мережі та багато інших.

2. Масштабованість: TensorFlow був розроблений з урахуванням розподіленого обчислення, що дозволяє йому виконувати обчислення на кластерах з багатьох машин. Це робить його великою платформою для тренування великих моделей на великих наборах даних.

3. Підтримка GPU: TensorFlow може використовувати графічні процесори (GPU) для прискорення обчислень, що є особливо важливим для тренування великих моделей глибокого навчання.

4. Візуалізація: TensorFlow включає TensorBoard, інструмент для візуалізації тренування моделей, включаючи графики втрат і точності, графики обчислень, візуалізації векторів слів та багато іншого.

5. Спільнота: TensorFlow має велику та активну спільноту розробників, яка постійно покращує бібліотеку та надає підтримку. Є багато доступних ресурсів для навчання та підтримки, включаючи офіційну документацію, навчальні матеріали та форуми.

6. Візуалізація даних: TensorFlow надає кращий спосіб візуалізації даних за допомогою графічного підходу. Він також дозволяє легко відлагоджувати вузли за допомогою TensorBoard, що зменшує необхідність перегляду всього коду і ефективно вирішує проблеми нейронної мережі.

7. Сумісність з Keras: TensorFlow сумісний з Keras, що дозволяє його

користувачам кодувати деякі секції з високорівневою функціональністю. Keras надає специфічну для системи функціональність TensorFlow, таку як конвеєрність, оцінювачі та жадібне виконання. Функціональний API Keras підтримує різноманітні топології з різними комбінаціями входів, виходів та шарів.

8. Сумісність з мовами програмування : TensorFlow сумісний з багатьма мовами, такими як C++, JavaScript, Python, C#, Ruby і Swift. Це дозволяє користувачу працювати в середовищі, в якому вони почуваються комфортно.

9. Паралелізм: TensorFlow використовується як бібліотека апаратного прискорення завдяки паралелізму робочих моделей. Він використовує різні стратегії розподілу в GPU та CPU системах. Користувач може вибрати, на якій з архітектур виконувати свій код, засновуючись на правилі моделювання. Система вибирає GPU, якщо не вказано інше. Цей процес зменшує використання пам’яті до певної міри.

Сфери використання Tensorflow [13, 14, 15, 16]: розпізнавання образів, обробка природної мови, рекомендаційні системи, аналітика даних, генерація зображень, автоматичне управління, медична діагностика, автономні автомобілі, фінансовий аналіз, голосовий асистент, аналітика соціальних мереж, аналіз настрою.

На сьогодні Tensorflow використовується великою кількістю компаній та організацій по всьому світу. Компанії, які використовують Tensorflow: Google, Uber, Airbnb, Nvidia, Twitter, Intel, PayPal. Мінусами бібліотеки Tensorflow є:

1. Високий поріг входу: Tensorflow має дещо крутіший поріг входу порівняно з іншими бібліотеками машинного навчання. Його складна структура та широкі можливості можуть ускладнити початкову роботу з ним.

2. Високі вимоги до апаратного забезпечення: Tensorflow може вимагати потужних обчислювальних ресурсів, зокрема графічних процесорів (GPU) або спеціалізованих пристроїв, для оптимальної роботи з великими моделями або обсягами даних.

3. Висока складність настройки: Налаштування Tensorflow, зокрема

оптимізація параметрів навчання та обробки даних, може бути складним процесом, що вимагає глибоких знань про бібліотеку та машинне навчання.

4. Помірна швидкість виконання: Хоча Tensorflow пропонує оптимізацію для прискорення виконання моделей, в порівнянні з деякими іншими бібліотеками швидкість роботи може бути помірною, особливо при великому масштабі даних.

5. Великий обсяг пам’яті: Використання Tensorflow може вимагати значних обсягів оперативної пам’яті, особливо при роботі з великими моделями або обсягами даних.

Приклад використання Tensorflow в поєднанні з мовою Python наведено на рисунку 2.3.

```
from tensorflow.keras.preprocessing.text import Tokenizer
def first_model(n_classes):
    model = tf.keras.models.Sequential([

            tf.keras.layers.Dense(2000, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.50),
            tf.keras.layers.Dense(20, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.50),
            tf.keras.layers.Dense(n_classes, activation='softmax')
            ])

    model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

    return model

def fit_print (X_train, X_test, y_train, y_test,n_classes):
    t = Tokenizer(num_words=2000)
    t.fit_on_texts(X_train)
    X_train = t.texts_to_matrix(X_train, mode='count')
    X_test = t.texts_to_matrix(X_test, mode='count')
    print (X_train.shape)
    model=first_model(n_classes)
    model.fit(X_train, y_train, epochs=5)
    results = model.evaluate(X_test,  y_test, verbose=2)
    print ('test loss: {0}, test acc: {1}'.format(results[0],results[1]))
    y_pred=model.predict_classes(X_test)
    con_mat = tf.math.confusion_matrix(labels=y_test, predictions=y_pred)
    print(con_mat.numpy())

fit_print (X_train_zab, X_test_zab,np.array(y_train_zab), np.array(y_test_zab),2)
fit_print (X_train, X_test, np.array(y_train)-1, np.array(y_test)-1,7)
```

Рисунок 2.3 - Приклад використання бібліотеки Tensorflow на мові Python

Розглянемо функцію first_model, в якій будуємо нашу першу модель. Аргумент n_classes – позитивне ціле число, вказує на кількість класів у вихідному шарі. Спочатку оголошуємо екземпляр класу tf.keras.models.Sequential, – що дозволяє лінійно вкладати шари нейронної мережі, і будуємо просту мережу, яка включає три повнозв'язні шари Dense (останній шар – вихідний, тому йому передається параметр n_classes), два шари BatchNormalization (нормалізує й масштабує входи для зменшення перенавчання) й два шари Dropout («відключає» частину нейронів у нейромережі, знову ж таки для зменшення перенавчання). Метод model.compile потрібен для компіляції моделі. Таким чином, перша нейронна мережа побудована і готова для використання.

Функція fit_print приймає навчальні й тестові вибірки, а також кількість класів. На початковому етапі оголошуємо клас Tokenizer і вказуємо максимальну кількість слів у вокабулярі. Метод fit_on_texts будує вокабуляр на навчальній вибірці, а метод texts_to_matrix перетворює текстові дані у матричний вигляд. Усе за аналогією до моделі «мішка слів», реалізованої в попередній частині. Далі навчаємо нашу модель і друкуємо результати. [17]

Для класифікації за забудованістю ділянок:

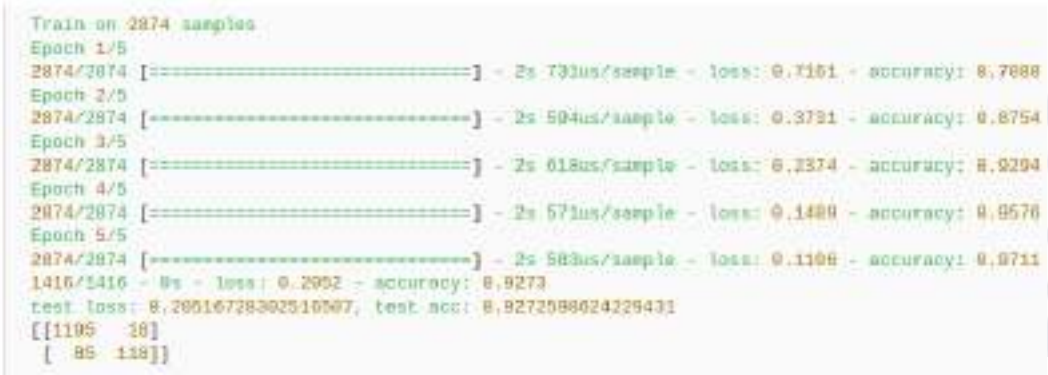


Рисунок 2.4 - Результати навчання моделі за забудованістю ділянок

Matplotlib - це бібліотека для мови програмування Python, призначена для візуалізації даних. Вона надає можливості для створення різних типів графіків, діаграм, гістограм, графіків розсіювання та інших видів візуалізацій. Дана

бібліотека дозволяє контролювати різні аспекти графіків, такі як заголовки, мітки осей, стилі ліній, кольори та шрифти. Бібліотека підтримує роботу з одновимірними та двовимірними даними, що дозволяє побудувати графіки залежностей між даними, відображення розподілів, трендів та ін.

```
Train on 2874 samples
Epoch 1/5
2874/2874 [==============================] - 2s 805us/sample - loss: 1.3955 - accuracy: 0.5612
Epoch 2/5
2874/2874 [==============================] - 2s 649us/sample - loss: 0.6525 - accuracy: 0.8381
Epoch 3/5
2874/2874 [==============================] - 2s 628us/sample - loss: 0.4022 - accuracy: 0.9123
Epoch 4/5
2874/2874 [==============================] - 2s 592us/sample - loss: 0.2985 - accuracy: 0.9315
Epoch 5/5
2874/2874 [==============================] - 2s 631us/sample - loss: 0.2377 - accuracy: 0.9482
1410/1410 - 0s - loss: 0.2796 - accuracy: 0.9301
test loss: 0.2796336193963633, test acc: 0.9300884764957428
[[883   2   5   3   0   0   0]
 [ 15  93   5   4   0   0   0]
 [ 10   2 270   0   0   0   0]
 [ 14   2   0  68   0   8   0]
 [ 11   0   0   1   1   9   0]
 [  0   1   0   0   0  12   0]
 [ 10   5   1   0   0   0  10]]
```

Для класифікації за типами ділянок:

Рисунок 2.5 - Результати навчання моделі за типами ділянок

Matplotlib є потужним інструментом для візуалізації даних у наукових, інженерних, статистичних та багатьох інших областях. Вона використовується як самостійна бібліотека або в поєднанні з іншими пакетами Python, такими як NumPy, Pandas та SciPy, для аналізу та візуалізації даних. Крім того, matplotlib надає різні режими роботи, включаючи режим сценаріїв, режим інтерактивної роботи та режим створення графіків у вікнах зображень. За допомогою Matplotlib можна зберігати графіки у різних форматах, таких як PNG, PDF, SVG та інші. Основною концепцією Matplotlib є об’єктно-орієнтований підхід до побудови графіків. Це означає, що графіки створюються за допомогою об’єктів, таких як фігури (Figures), підграфіки (Subplots) та елементи графіків (Artists), які можна налаштовувати та маніпулювати. Загалом, Matplotlib є потужним інструментом для візуалізації, власне тому я обрав до використання у даній роботі.

Scikit-learn, або sklearn, є високоякісною та повнофункціональною бібліотекою машинного навчання для мови програмування Python. Вона надає

широкий асортимент алгоритмів машинного навчання, які можна використовувати для класифікації, регресії, кластеризації, зменшення розмірності, виявлення аномалій та інших завдань. Scikit-learn побудована на базі інших потужних бібліотек Python, таких як NumPy, SciPy та Matplotlib, і пропонує однорідний інтерфейс для легкого впровадження алгоритмів машинного навчання. Вона надає широкі можливості для практичної реалізації складних концепцій машинного навчання, таких як хрещення даних, валідація моделей та налаштування гіперпараметрів.

В бібліотеці sklearn ви знайдете реалізації різноманітних алгоритмів, таких як лінійна регресія, логістична регресія, метод опорних векторів (SVM), рішітка рішень (Decision Tree), випадковий ліс (Random Forest), метод k-найближчих сусідів (KNN), кластеризація k-середніх (K-Means), метод головних компонент (PCA) та інші. Ці алгоритми базуються на передових дослідженнях у галузі машинного навчання та забезпечують ефективну та надійну реалізацію.

Крім алгоритмів машинного навчання, sklearn також надає засоби для попередньої обробки даних, включаючи масштабування, кодування категоріальних ознак, відбір ознак та обробку пропущених значень. NumPy (Numerical Python) - це бібліотека для мови програмування Python, яка надає підтримку для роботи з багатовимірними масивами і матрицями, а також велику колекцію математичних функцій, що дозволяють виконувати операції над цими структурами даних ефективно і зручно.

Основою NumPy є об’єкт `ndarray`, який представляє масиви багатьох розмірностей. `ndarray` дозволяє виконувати операції масової обробки даних, такі як арифметичні операції, індексація, зрізи та трансформації даних. Масиви NumPy є ефективними, оскільки вони забезпечують компактне зберігання даних та оптимізовані алгоритми для виконання операцій над ними.

Окрім роботи з масивами, NumPy також має вбудовані функції для виконання математичних операцій, таких як тригонометрія, логарифми, експоненціальні функції та інші. Вона надає зручний інтерфейс для векторизованої обробки даних, що дозволяє виконувати операції на цілих

масивах без необхідності використовувати цикли.

NumPy також має функції для роботи з лінійною алгеброю, випадковими числами, статистикою та іншими областями математики. Вона інтегрується з іншими бібліотеками машинного навчання, такими як scikit-learn, що дозволяє ефективно працювати з даними та виконувати розрахунки. NumPy є важливим інструментом для наукових обчислень та обробки даних в Python і використовується широко у спільноті.

Selenium, це потужний інструмент для контролювання веб-браузера через програму. Selenium WebDriver – це гнучкий інструмент для автоматизованого тестування веб-проєктів на базі набору бібліотек для різних мов програмування, таких як Java, .Net (C#), Python, Ruby, PHP, Perl, JavaScript. Даний інструмент підтримує роботу на базі Windows, macOS та Linux, а також найпоширеніші браузери Google Chrome, Firefox, Safari, Edge, Internet Explorer та навіть деякі браузери без графічного інтерфейсу. Використовується Selenium WebDriver найчастіше з такими видами тестування, як регресійне та функціональне. [25]

Архітектура Selenium WebDriver складається з 4-х компонентів: драйвер конкретного браузера; клієнтська бібліотека Selenium; браузер; протокол JSON Wire (JavaScript Object Notation).

Рисунок 2.6 - Архітектура Selenium WebDriver

Використовується webdriver браузера, який звертається до браузера, драйвер якого використано, та виконує послідовність дій через певний скрипт з бібліотеки Selenium. Взаємодія бібліотеки з веб-драйвером відбувається через JSON Wire Protocol. Основні сутності та методи Selenium WebDriver :

- WebDriver – це найважливіша сутність, яка відповідає за управління діями браузера.
- Webelement – друга важлива сутність, яка представляє собою абстракцію над конкретним веб-елементом (посиланням, кнопкою, полем для вводу та ін.).
- Локатор – це тип веб-елементу, який потрібно знайти та над яким виконуватиме дії веб-драйвер.
- By – це абстракція над локатором веб-елемента, клас необхідний для ідентифікації веб-елементів який має такий синтаксис By.локатор.

2.2 Характеристика та ознаки набору даних для навчання

В якості даних був зібраний власний датасет. Етап збору датасету був розділений на наступні етапи :

1. Визначення мети. На цьому етапі ми визначаємо для якої конкретної задачі потрібен датасет та фіксуємо її. Це може бути як розпізнавання об’єктів, класифікація, сегментація зображень. Ясно сформульована мета допомагає чітко визначити потрібні дані. Кінцева мета збору нашого датасету є - класифікація зображень. Класифікація зображень - це процес призначення мітки або категорії для зображення на основі його вмісту. Зібраний датасет в свою чергу надасть нам можливість автоматизувати подальший збір даних ще з більшою швидкістю та точністю, за рахунок участі нейронної мережі у цьому процесі, що і є кінцевою метор - тобто автоматизування зібрання даних.

2. Визначення джерела вхідних даних та типу даних. Визначення джерела вхідних даних - є дуже важливим кроком у процесі збору датасету. Якість та репрезентативність вхідних даних мають прямий вплив на якість та ефективність моделі машинного навчання. Використання надійних та достовірних джерел даних допомагає уникнути неточностей та помилок у датасеті. Під час вибору

джерел даних враховувалась доступність даних. Наприклад, якщо планується збирати дані самостійно, то потрібно врахувати доступні ресурси, час та зусилля, необхідні для отримання великого обсягу якісних даних. Джерелами вхідних даних в нашому видадку - це публічно доступні ресурси, а саме : Unsplash, Pexels, Instagram, Pixabay, Google (рис. 2.7).

Рисунок 2.7 - Публічно доступні джерела даних

Типом вхідних даних можуть бути фотографії, відео, текстові документи, аудіозаписи тощо. Залежно від мети, можуть знадобитися спеціальні датчики або засоби для збору даних. Тип вхідних даних - фотографії.

3. Визначення категорій. Однією з особливостей визначення категорій датасету є правильний підбір та організація категорій, що відображають вашу задачу класифікації.

- Репрезентативність категорій: Важливо, щоб категорії були репрезентативними для задачі та враховували всі можливі варіації об’єктів, які плануються класифікуватись. В нашому випадку ми хочемо отримати, як кінечний результат - нейронну мережу, яка здатна розпізнавати наявність медичної маски на лиці людини.Отже, нам потрібно врахувати такі варіації об’єктів:

- Людина в медичній масці
- Людина без маски
- Людина з прикритим лицем будь-яким іншим об’єктом, який не являється медичною маскою або частиною тіла
- Відсутність фізичного об’єкту
- Присутність тільки медичної маски при відсутності фізичного об’єкту
- Присутність тваринної істоти або фізичного об’єкту

- Присутність людини та медичної маски, але не в одягнутому положенні.

- Доступність даних для категорій: Варто враховувати, що для кожної категорії ви повинні мати достатню кількість зображень, щоб забезпечити ефективне тренування моделі. Якщо одна з категорій має дуже мало зображень, це може призвести до проблем з недостатньою репрезентативністю цієї категорії та поганою продуктивністю моделі. В нашому випадку, дані з людиною в одягнутій медичній масці будуть мати найбільший дефіцит - тобто, їхня кількість являється порівняно обмеженою в порівнянні з іншими типами фотографій. На відміну від цього типу фотографій - інші типи мають необмежену кількість прикладів, адже являються менш предметно специфікованими - тобто більш узагальненими та не потребуючими конкретних артефактів на фото.

- Баланс категорій: Важливо підтримувати баланс між категоріями у датасеті. Якщо одна категорія має значно більше зображень, ніж інші, це може призвести до перекосу в навчанні моделі та незадовільно впливати на результати моделі. В нашому випадку виходячи з висновку попереднього пункту про доступність даних для категорій зрозуміло, що баланс категорій при зборі може бути порушений через клас, який потребує особливого артефакту на фото - тобто медичної маски. Тому ми будемо орієнтуватись саме на цей клас - та введемо його зібрану кількість прикладів - як обмежуючу і для інших класів. В нашому випадку, це буде 3 категорії фотографій. Це будуть наступні три категорії:

- “Without mask”, людина в медичній масці.
- “With mask”, людина без медичної маски.
- “Without person”, зображення, де образ людини відсутній.

4. Збір зображень. Процес збереження зображення для кожної категорії з публічних ресурсів відбувається спочатку власноруч. Ціль цього кроку - зібрати достатню кількість зображень у кожній категорії, щоб забезпечити достатню репрезентативність датасету. Важливо, щоб датасет був репрезентативним для задачі. Вибір правильного джерела даних допомагає забезпечити, що ваш датасет включає різноманітність зображень, що покривають різні аспекти та варіації об’єктів, які ви хочете класифікувати. Нерепрезентативний датасет може

призвести до низької точності та поганої загальної продуктивності моделі.

Усі фотографії, які попали в набір для дослідження мають не випадковий розподіл. Розглянемо три класи:

- “Without Mask” - клас містить фотографії людей без медичних масок.
- “With Mask” - клас містить фотографії людей в медичних масках.
- “Without Person” - клас містить в собі фотографії, які не містять в собі силуети людей. Але, фотографії які знаходяться в цьому класі збирались спираючись на спеціальний розподіл, який буде відображено нижче.

Категорія “Without Person” - існує в моделі для того, щоб надати моделі поняття, що ж саме собою представляє собою фотографія без силуэта людина. Проблематика цього класу в тому, що якщо ми наприклад віддамо цьому класу тільки фотографії тварин та будівель, які по своїй природі не є людиною - призведе до того, що “Without Person” клас буде в собі містить тільки два характерні типи об’єктів. І модель замість того, щоб зрозуміти, як виглядає фотографія без людини - навчиться розуміти, як виглядає фотографія на якій є тварини та будівлі, що не є коректною поставленою задачею для заданого класу. Тому, ця категорія вміщує спеціальний розподіл фотографій за різними тематиками: "Тварини", "Рослини", "Транспорт", "Комікси та картини", "Декоративні предмети", "Предмети схожі на медичну маску", "Природні об’єкти", "Інтер’єри", "Будівлі", "Їжа", "Електронні ручні пристрої", "Іграшки".

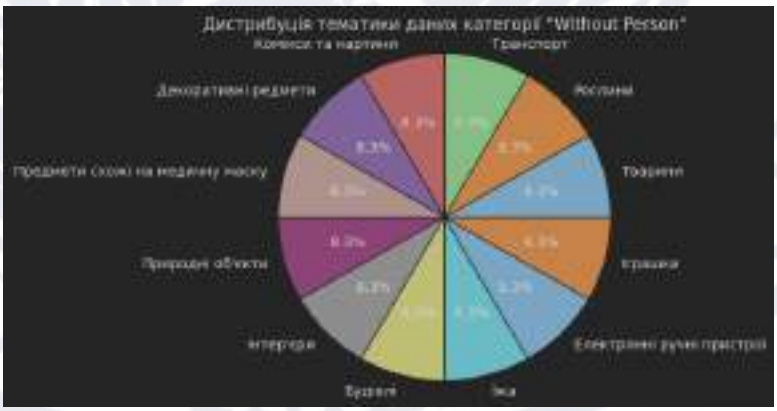


Рисунок 2.8 - Дистрибуція тематик фотографій категорії “Without Person”

5. Перевірка якості дані: необхідно переконатись, що дані мають високу якість. Перевірити наявність помилок, відсутність дублікатів та інших аномалій. Перед використанням датасету переконуюсь, що дані мають високу якість і є репрезентативними для кожної категорії. Видаляю некоректні або неправильно анотовані зображення, а також впевнююсь, що кожне зображення належить лише одній категорії. Для прикладу візьмемо фотографію нижче. Легко бачити, що ця фотографія відноситься саме до класу “Without Mask” і ніяк не може належати до інших категорій, по причинам: відсутність маски на фотографії відразу відкидує категорію “With Mask”, а наявність чоловіка на фотографії відразу відкидує категорію “”Without Person”. Отже, фотографії датасету мають чіткий розподіл між категоріями.

Рисунок 2.9 - Фотографія з датасету

Наступний приклад ілюструє зображення людини в масці на достатньо великій відстані з дещо викривленою формою зображення. Ця фотографія якісно задовольняє критерій якості датасету, оскільки на фотографії присутня людина в масці - незважаючи на невеликі викривлення на фотографії.

Рисунок 2.10 - Віддалена фотографія з датасету з викривленим зображенням

6. Анотування зображень: Для кожного зібраного зображення присвоюю йому відповідну категорію. Наприклад, якщо я бую зображення людини в маці - я позначаю це зображення відповідною міткою “Людина в медичній масці”. Цей процес виконується вручну.

Рисунок 2.11 - Фотографій категорії “With Mask”

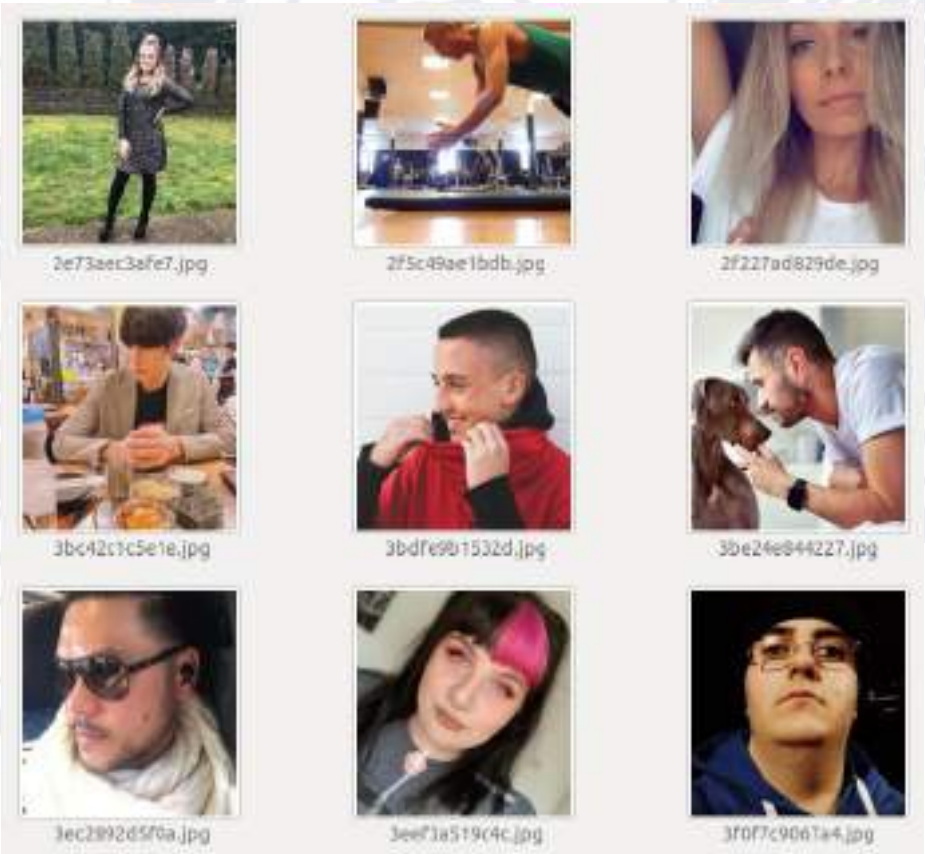


Рисунок 2.12 – Фотографії категорії “Without Mask”

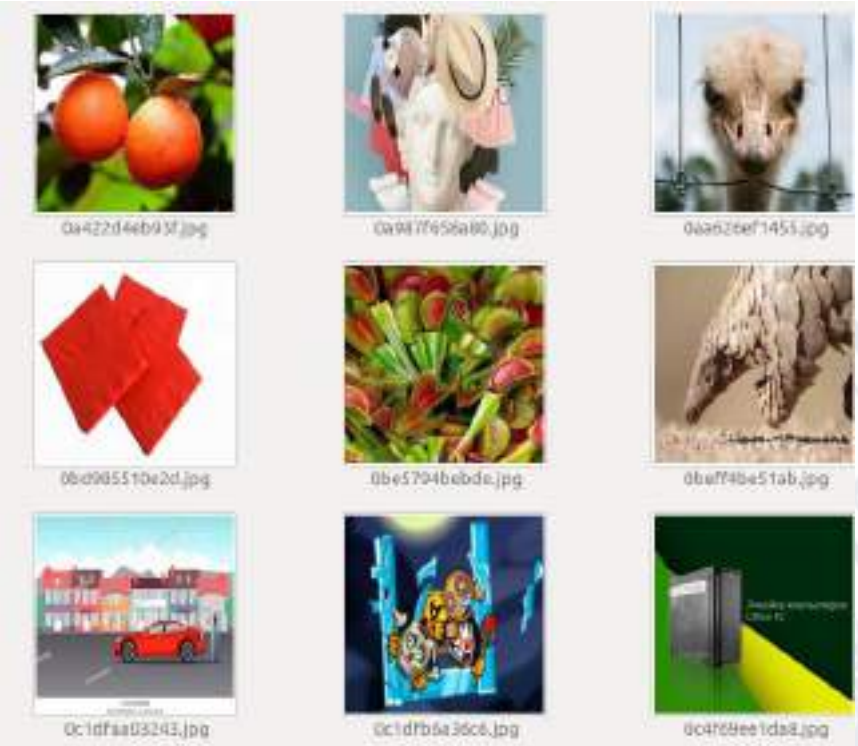


Рисунок 2.13 - Фотографії категорії “Without Person”

В результаті анотування зображень категорії будуть поділятись в зібраному датасеті наступним чином, як зображено на рисунку 2.14. З даного графіку ми бачимо, що категорія “Without mask” становить 5740 фотографій, що відповідає 41.5% загальної кількості даних; категорія “With mask” становить 3990 фотографій, що відповідає 28.9% загальної кількості даних; категорія “Without Person” становить 4090 фотографій, що відповідає 29.6% загальної кількості даних.

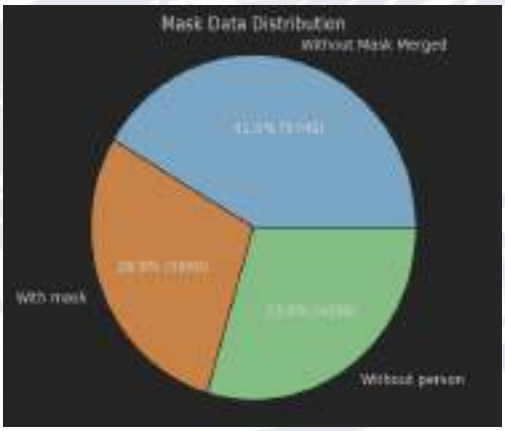


Рисунок 2.14 - Розбиття датасету на категорії

1. Розбиття датасету на тренувальну, валідаційну та тестову вибірки. Категорії будуть поділятись на тренувальній виборці наступним чином, як зображено на рисунку 2.16. З даного графіку ми бачимо, що категорія “Without mask” становить 4400 фотографій, що відповідає 41.5% загальної кількості даних; категорія “With mask” становить 3000 фотографій, що відповідає 28.3% загальної кількості даних; категорія “Without Person” становить 3200 фотографій, що відповідає 30.2 загальної кількості даних.

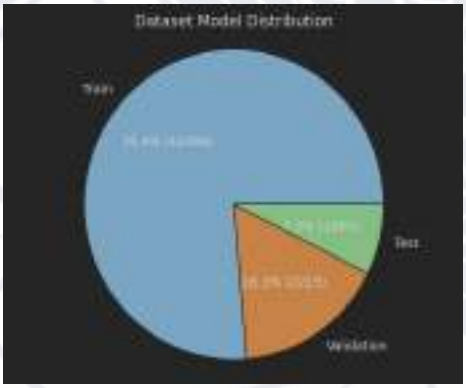


Рисунок 2.15 - Розподіл датасету на тренувальну, тестову та валідаційну вибірку

2.3 Автоматизація збору даних з публічно доступних джерел

Нами зроблено акцент на тому, що бібліотека Selenium WebDriver, спроектована для автоматизованої взаємодії з вебсайтами, може бути потужним інструментом для збору даних з різноманітних публічно доступних джерел. Selenium дозволяє створювати автоматизовані боти, які можуть переглядати сторінки, виконувати дії, отримувати і аналізувати дані на вебсайтах, і все це відбувається з великою швидкістю та точністю.

Selenium - це комплекс інструментів, широко використовуваних у тестувальній спільноті для перевірки сумісності з різними веб-браузерами. Selenium не призначений для автоматизації роботи з настільними додатками; він призначений лише для використання у браузерах. Він вважається одним із

найбільш популярних наборів інструментів для автоматизованого тестування веб-додатків, оскільки підтримує популярні веб-браузери, що робить його дуже потужним. [26]

Рисунок 2.16 - Python + Chrome + Selenium

Selenium підтримує ряд браузерів (Google Chrome 12+, Internet Explorer 7,8,9,10, Safari 5.1+, Opera 11.5, Firefox 3+) та операційних систем (Windows, Mac, Linux/Unix). Також Selenium забезпечує сумісність з різними мовами програмування - C#, Java, JavaScript, Ruby, Python, PHP. Тестувальники можуть вибирати мову програмування для створення тестових кейсів, що робить Selenium надзвичайно гнучким. Не обов'язково писати код Selenium на тій самій мові, що і додаток. Наприклад, якщо веб-сайт написаний на C#, код Selenium може бути написаний на PHP. Комплект інструментів Selenium складається з чотирьох основних компонент:

1. Selenium IDE (Інтегроване середовище розробки) - це, передусім, інструмент для запису та відтворення дій. Це додаток або розширення, доступне для браузерів Firefox і Chrome, яке швидко генерує тести за допомогою функції запису та відтворення. Вам не потрібно вивчати жодну мову для написання функціональних тестів.

2. Selenium RC - у випадку роботи з Selenium RC (Remote Control), потрібно мати хороше розуміння хоча б однієї мови програмування. Цей інструмент дозволяє вам розробляти тести для респонсивного дизайну на будь-якій мові скрипту, яку ви виберете. Сервер та клієнтські бібліотеки - два основні

компоненти Selenium RC. Його архітектура складна і має свої обмеження.

3. Selenium WebDriver - це розширена версія Selenium RC. Він був представлений на ринку для подолання обмежень, з якими стикається Selenium RC. Незважаючи на те, що це вдосконалена версія RC, його архітектура повністю відрізняється від RC. Так само, як і Selenium RC, Selenium WebDriver підтримує кілька платформ програмування, щоб надати більшу гнучкість і вимагає володіння хоча б однією мовою програмування.

4. Selenium Grid - це інструмент, який використовується для одночасного виконання тестових кейсів на різних браузерах, машинах і операційних системах. Цей інструмент робить тестування сумісності з різними браузерами надзвичайно простим. [26]

Selenium WebDriver - це вебфреймворк, який дозволяє вам виконувати тести, які сумісні з різними браузерами. Цей інструмент використовується для автоматизації тестування веб-додатків з метою перевірки їхньої правильної роботи. Архітектура Selenium WebDriver складається з 4 компонентів (рис. 2.18):

- Selenium Client library
- JSON wire protocol over HTTP
- Browser Drivers
- Browsers

Підтримка різних мовних обгорток в Селеніум - Селеніум забезпечує можливість роботи з різними бібліотеками або мовами програмування, такими як Ruby, Python, Java тощо, завдяки розробленим розробниками Селеніум мовним інтерфейсом. Наприклад, для використання драйвера браузера в мові Python, можна скористатися спеціальними інструментами, призначеними для Python. Роль JSON Wire Protocol у Селеніум. JSON Wire Protocol - це абревіатура від JavaScript Object Notation. Це відкритий стандарт, який створює механізм для обміну даними між клієнтом та сервером у Інтернеті. Він підтримує різноманітні структури даних, такі як масиви і об’єкти, що спрощує операції зчитування і запису даних у форматі JSON. JSON функціонує як REST API (Representational State Transfer), яке здійснює обмін інформацією між HTTP-серверами.

Докладніше вивчайте REST API для доступу до Селеніум.

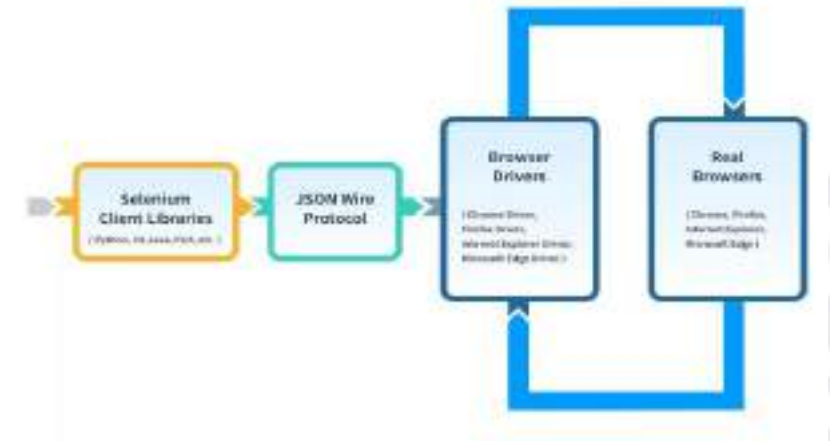


Рисунок 2.17 - Основні компоненти Архітектура Selenium WebDriver

- Браузерні драйвери у Селеніум - у Селеніум існують драйвери, які призначені для роботи з кожним окремим браузером. Вони взаємодіють з браузерами, не розголошуючи внутрішньої логіки функціональності браузера, завдяки чому можливе безпечне встановлення зв'язку. Ці браузерні драйвери також адаптовані до мов програмування, які використовуються для автоматизації тестових сценаріїв, таких як C#, Python, Java та інші.
-

Рисунок 2.18 - Selenium Browser Drivers

Webdriver працює наступним чином:

1. Для кожної команди Селеніум генерується HTTP-запит, який надсилається до драйвера браузера.

2. HTTP-запит отримується драйвером через HTTP-сервер. Тому WebDriver і Браузер обоє налаштовані таким чином, що вони слухають команди

HTTP сервера та вміють спілкуватися тільки через нього.

3. Всі кроки та інструкції, які потрібно виконати в браузері за допомогою драйвера, визначаються HTTP-сервером.

4. Після цього HTTP-сервер отримує зворотній статус виконання та надсилає його назад у сценарії автоматизації.

Рисунок 2.19 - Схема клієнт, WebDriver, Browser.

Базові етапи в скрипті для Selenium WebDriver. [26]

1. Створюється WebDriver екземпляр.

2. Відбувається навігація на веб-сторінку.

3. Використовуючи Локатори Selenium - знаходиться певний шуканий веб-елемент на веб-сторінці.

4. Виконуються вказані командоми бібліотеки селеніум дії на знайденому веб-елементі.

5. Завантажується відповідь браузера на конкретну дію.

6. Використовується відповідь браузера для конкретних цілей.

На своєму найменшому рівні WebDriver взаємодіє з веб-браузером через драйвер. Комунікація відбувається у двох напрямках: WebDriver передає команди веб-браузеру через драйвер і отримує інформацію назад по тому ж маршруту. Драйвер є специфічним для конкретного браузера, такого як ChromeDriver для браузера Google Chrome/Chromium, GeckoDriver для Mozilla Firefox і так далі. Драйвер запускається на тій же системі, що і браузер. [27]

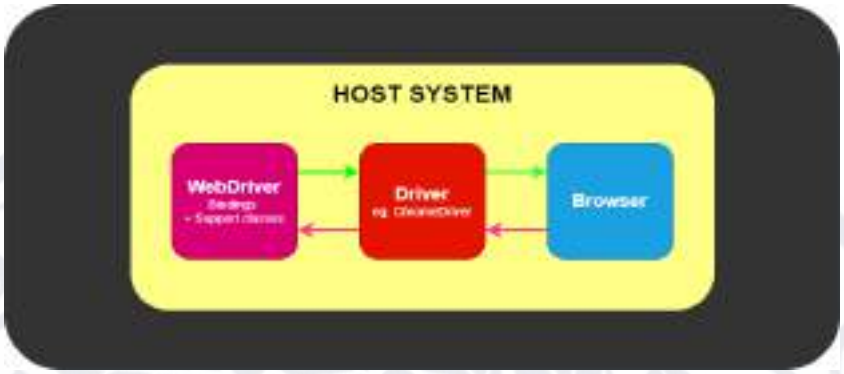


Рисунок 2.20 - Пряма комунікація на єдиній хост системі

Цей простий приклад вище є прямою комунікацією. Комунікація з браузером також може бути віддаленою за допомогою Selenium Server або RemoteWebDriver. RemoteWebDriver працює на тій же системі, що і драйвер та браузер. [27]

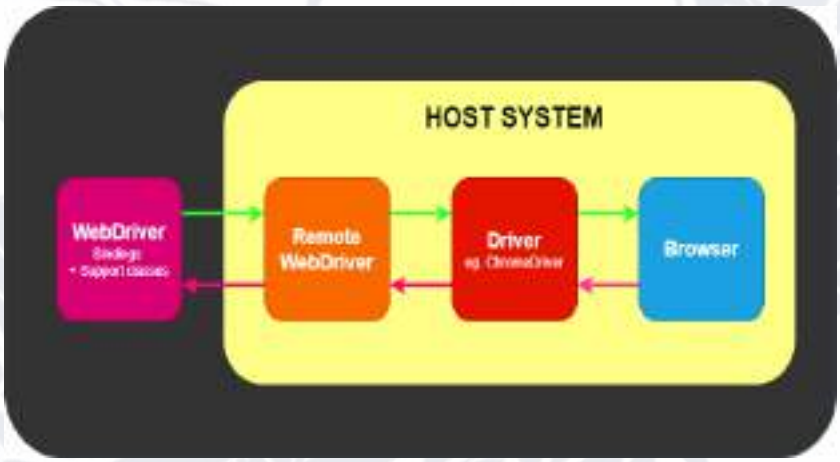


Рисунок 2.21 - Віддалена комунікація

2.4. Математичне обґрунтування застосовуваних технологій проектування нейронної мережі

Розглянемо архітектуру нейронної мережі MobileNet.

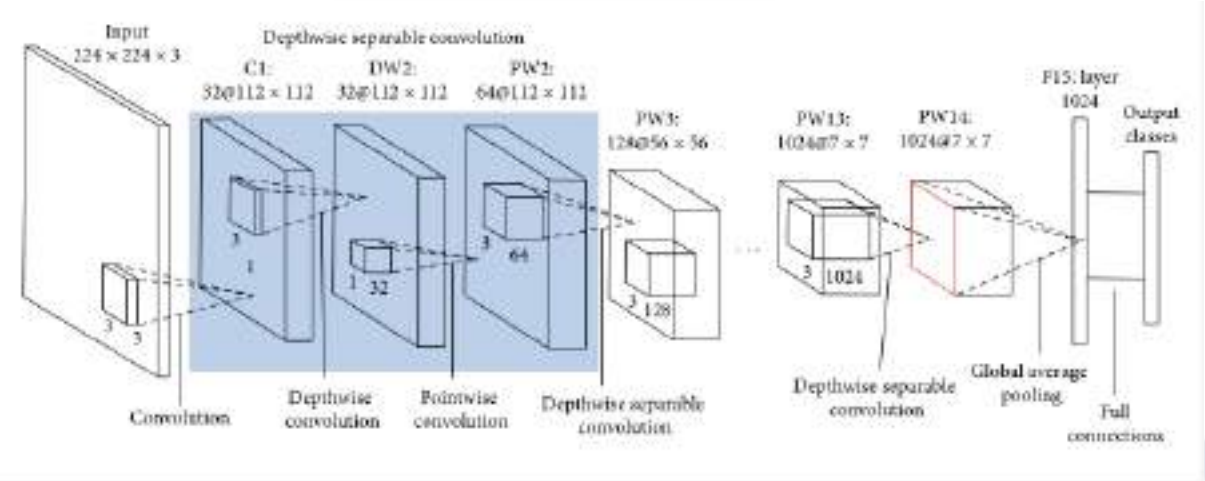


Рисунок 2.22 - Архітектура нейронної мережі “MobileNet”

Зліва: звичайний згортковий шар із batchnorm і ReLU слоями. Праворуч: згортки, що розділяються по глибині, з шарами по глибині та по точках, за якими слідують batchnorm та ReLU. [39]

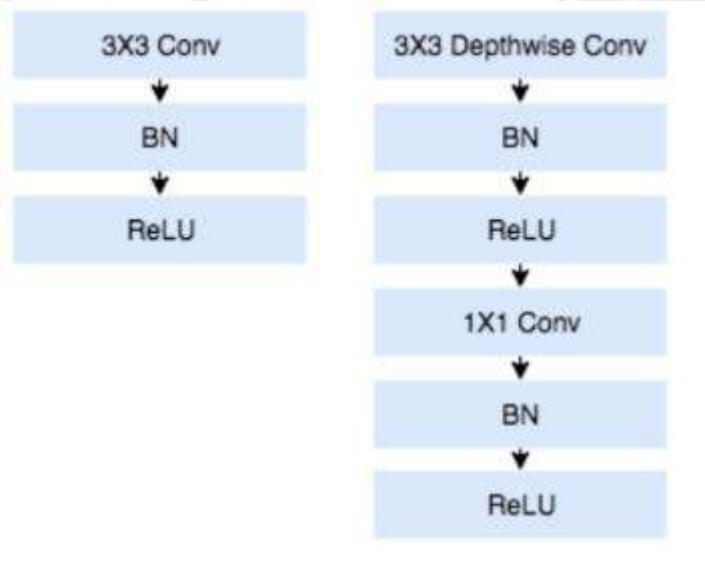


Рисунок 2.23 - Згортковий шар

Повна мережа MobileNets складається з 30 рівнів. Конструкція мережі досить проста: згортковий шар із кроком 2; шар по глибині; точковий шар, який подвоює кількість каналів; шар по глибині з кроком 2; точковий шар, який подвоює кількість каналів; шар по глибині; точковий шар; шар по глибині з

кроком 2; точковий шар, який подвоює кількість каналів. Після самого першого шару (звичайної згортки) чергуються глибинний і точковий шари. Іноді рівень глибини має крок 2, щоб зменшити ширину та висоту даних, які проходять через мережу. Іноді поточковий рівень подвоює кількість каналів у даних. Усі згорточні шари супроводжуються функцією активації ReLU. [42]

Це триває деякий час, поки вихідне зображення не зменшиться до пікселів, але тепер воно має 1024 канали. Після цього з'являється середній шар об'єднання, який працює над усім зображенням, щоб отримати зображення, яке є просто вектором із 1024 елементів (додаток А).

Модель MobileNet базується на глибоко віддільних згортках, які являють собою процедуру факторизованих згорток. Цей процес полягає у розкладанні звичайної згортки на глибоку згортку та згортку 1 × 1, яку називають точковою згорткою. Глибока згортка в MobileNets використовує один фільтр для кожного вхідного каналу. Після цього точкова згортка застосовує згортку 1 × 1 для об'єднання виходів глибокої згортки. Звичайна згортка одночасно фільтрує та об'єднує входи в новий набір виходів за один крок. Глибоко віддільна згортка розділяє цей процес на два шари: окремий шар для фільтрації та окремий шар для об'єднання. Така факторизація має ефект значного зниження обчислювальних витрат і розміру моделі. На рисунку показано, як звичайна згортка (додаток А, рис. А.2) розкладається на глибоку згортку (додаток А, рис. А.3) і точкову згортку 1 × 1 2(c). [44]

Отже, звичайні конволюційні фільтри замінюються двома шарами: глибокою згорткою та точковою згорткою для створення глибоко віддільного фільтра. [45] Звичайний конволюційний шар приймає на вході відображення ознак розміром DF * DF * M, позначене як F, і виробляє відображення ознак розміром DF * DF * N, позначене як G, де DF – просторова ширина та висота квадратного вхідного відображення ознак, M – кількість вхідних каналів (глибина входу), DG – просторова ширина та висота квадратного вихідного відображення ознак, а N – кількість вихідних каналів (глибина виходу). Звичайний конволюційний шар параметризується конволюційним ядром K

розміром Dk * Dk * M * N, де Dk – просторовий розмір ядра, яке припускається квадратним, а M і N – кількість вхідних та вихідних каналів відповідно, як було зазначено раніше.

Вартість обчислень для звичайної згортки залежить множинно від кількості вхідних каналів M, кількості вихідних каналів N, розміру ядра згортки DK * DK та розміру відображення ознак DF * DF. [46] Моделі MobileNet детально розглядають кожен з цих термінів та їхні взаємозв'язки. По-перше, вони використовують глибоко віддільні згортки для розриву зв'язків між кількістю вихідних каналів та розміром ядра. Звичайна операція згортки дає результат у вигляді фільтрації ознак, сформованих на ядрах згортки, та об'єднання ознак з метою створення нового відображення. Етапи фільтрації та об'єднання можна розділити на два кроки за допомогою використання факторизованих згорток, які називаються глибоко віддільними згортками, для істотного зменшення вартості обчислень.

Кількість множень в 1 операції згортання = розмір фільтра = Dk × Dk × M. Оскільки є N фільтрів і кожен фільтр ковзає вертикально та горизонтально Dp разів, загальна кількість множень стає N × Dp × Dp × (множення на згортку). Загальна кількість операцій множень обчислюється [47]:

$$TM = N \times Dp^2 \times Dk^2 \times M \qquad (2.1)$$

де TM – загальна кількість операцій множень;
N – кількість вихідних каналів;
Dk – просторовий розмір ядра згортки;
M – кількість вхідних каналів.

Глибоко віддільні згортки складаються з двох шарів: глибоких згорток та точкових згорток. Тоді глибокі згортки застосовують один фільтр до кожного вхідного каналу (глибина входу). Потім точкова згортка, яка є простою згорткою 1×1, використовується для створення лінійної комбінації виходу. Ця модель використовує як нормалізацію партій (batchnorm), так і ReLU нелінійності для

обох шарів. [31] Вартість обчислень глибоко-віддільної згортки:

$$CC = DF \times DF \times DK \times DK \times M \quad (2.2)$$

де DF×DF – розмір вихідного відображення ознак;

DK×DK – розмір ядра згортки;

M – кількість вхідних каналів.

Відмінність вартості обчислень глибокої згортки від звичайної згортки полягає в тому, що вона не залежить від кількості вихідних каналів N, оскільки глибока згортка виконує операції фільтрації окремо для кожного вхідного каналу. Це призводить до значного зменшення вартості обчислень, особливо коли кількість каналів велика.

У глибинних операціях згортка застосовується до одного каналу за раз, на відміну від стандартних CNN, у яких це робиться для всіх M каналів. Отже, тут фільтри/ядра матимуть розмір Dk × Dk × 1. Якщо у вхідних даних є M каналів, то таких фільтрів потрібно M. Вихід матиме розмір Dp × Dp × M. [49]

Щоб отримати такий самий ефект із звичайною згорткою, нам потрібно вибрати канал, зробити всі елементи фільтра нульовими, крім цього каналу, а потім виконати згортку. Нам знадобляться три різних фільтра – по одному для кожного каналу. Хоча параметри залишаються незмінними, ця згортка дає вам три вихідних канали лише з одним 3-канальним фільтром, тоді як вам знадобляться три 3-канальні фільтри, якщо ви використовуєте звичайну згортку.

Одна операція згортки вимагає множення Dk × Dk. Оскільки фільтр ковзає на Dp × Dp по всіх M каналах, загальна кількість множень у згортці (CM) обчислюється:

$$CM = Dp \times Dp \times D_K \times D_K \times M \quad (2.3)$$

де Dp×Dp – розмір вихідного каналу;

$D_K \times D_K$ – розмір ядра згортки;

M – кількість вихідних каналів.

Глибока згортка з одним фільтром на кожний вхідний канал (глибина входу) може бути представлена наступним чином:

$$G_{k,l,n} = \sum_{i,j} F_{k+i-1,l+j-1,n} \cdot K_{i,j,n} \quad (2.4)$$

де $G_{k,l,n}$ – значення в позиції (k, l) на n-му вихідному каналі вихідного відображення ознак;

F – вхідне зображення ознак;

K – ядро згортки;

n – індекс вхідного/вихідного сигналу

M – кількість вихідних каналів;

(i, j) – позиція в ядрі згортки;

(k, l) – просторова позиція на виході.

Тут глибока згортка здійснюється окремо для кожного вхідного каналу, і ядро згортки K має розміри, які відповідають просторовим розмірам вхідного відображення ознак, але діє окремо на кожному каналі. В результаті отримується відображення ознак, де кожен канал виходу сформований за допомогою фільтрації відповідного вхідного каналу окремим фільтром.

Поточкова згортка. Згортка з розміром ядра 1*1, яка просто поєднує функції, створені згорткою по глибині. Його обчислювальна вартість становить M×N×DF×DF. Оскільки згортка по глибині використовується лише для фільтрації вхідного каналу, вона не поєднує їх для отримання нових функцій. Таким чином створюється додатковий рівень, який називається шаром поточкової згортки, який обчислює лінійну комбінацію результату поглибленої згортки за допомогою 1*1 згортки. Вартість обчислення поточкової згортки [42] буде дорівнювати:

$$CC = DF \times DF \times N \times M \quad (2.5)$$

де DF×DF – розмір вихідного відображення ознак;

M – кількість вхідних каналів.

Наведений вище випадок застосовується, коли стрибок дорівнює 1. Поточкову згортку також можна застосувати, коли стрідер не дорівнює 1. У поточковій роботі операція згортки 1×1 застосовується до M каналів. Отже, розмір фільтра для цієї операції буде 1×1×M. Скажімо, ми використовуємо N таких фільтрів, вихідний розмір стає DP × DP ×N.

Вартість поточкової операції згортки. Одна операція згортки вимагає множень 1×M. Оскільки фільтр пересувається на Dp×Dp разів, загальна кількість множень дорівнює M×Dp×Dp×(кількість фільтрів). Кількість операцій множення поточкової згортки можна вирахувати формулою (CCp):

$$CCp = M \times Dp^2 \times N \qquad (2.6)$$

де Dp^2 – розмір вихідного каналу;

N – кількість фільтрів;

M – кількість вхідних каналів.

Розглянемо фільтр Sobel, який використовується в обробці зображень для виявлення країв. Ми можемо розділити висоту та ширину цих фільтрів. Фільтр Gx (додаток А, рис. А.9) можна розглядати як матричний добуток транспонування [1 2 1] з [-1 0 1]. [45]

Фільтр замаскувався. Він показує, що мав 9 параметрів, але насправді він має 6. Це стало можливим завдяки розділенню його розмірів висоти та ширини. Та саме, застосовується до відокремлення розміру глибини від горизонтального (ширина × висота), дає нам згортку, що розділяється по глибині, де ми виконуємо згортку по глибині, а після цього використовуємо 1*1 фільтр, щоб охопити розмір глибини рисунок 2.24. [42]

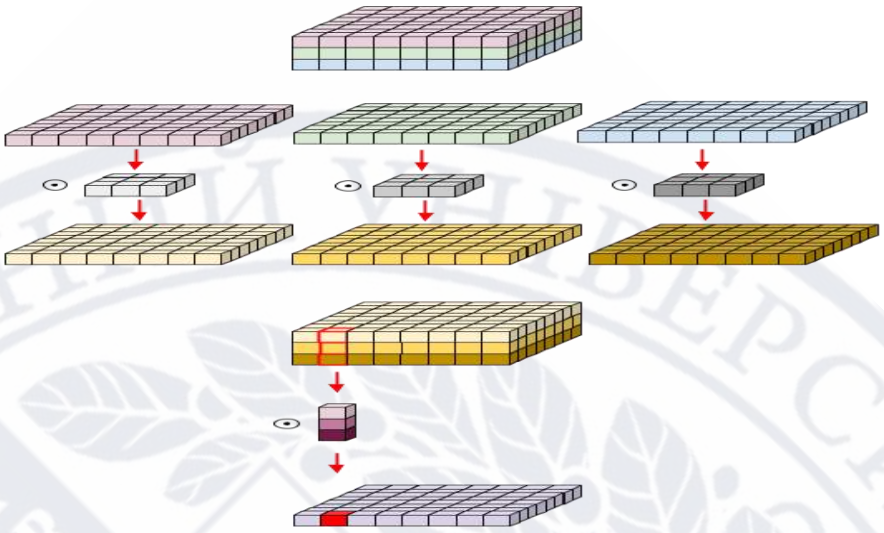


Рисунок 2.24 - Роздільна згортка по глибині

Варто відзначити кількість параметрів, яку було зменшено цією згорткою, щоб вивести те саме число каналів. Для створення одного каналу потрібно 3*3*3 параметрів для виконання згортки по глибині та 1*3 параметри для виконання подальшої згортки у вимірі глибини. Але якщо потрібно 3 вихідних канали, нам знадобляться лише 1*3 фільтри глибини, які дають нам загальну кількість параметрів 36 = 27 + 9, в той час як для тої ж кількості вихідних каналів у звичайній згортці, нам потрібні 3*3*3 фільтри, які дають нам загалом 81 параметр. Наявність занадто великої кількості параметрів змушує функцію запам’ятовувати, а не навчатися, і, отже, надмірно підлаштовуватися. Цю проблему вирішує роздільна згортка по глибині.

Таким чином, загальні обчислювальні витрати згорток, які можна розділити по глибині, можна розрахувати як: Dk× Dk×M×Df×Df+N×M×Df×Df.

Для операції роздільної згортки з глибиною загальна кількість операцій:

$$CCg = M \times Dp^2 \times (Dk^2 + N) \qquad (2.7)$$

де Dp^2 – розмір вихідного каналу;

N – кількість фільтрів;

M – кількість вхідних каналів.

Порівняння складності типів операцій згортки наведена в таблиці 2.1

Таблиця 2.1 – Порівняння складності типів операцій згортки

Тип згортки	Складність
Стандартна згортка	$N \times Dp^2 \times Dk^2 \times M$
Роздільна згортки по глибині	$M \times Dp^2 \times (Dk^2 + N)$

Коефіцієнт Ratio (R) визначається:

$$\text{Ratio (R)} = \frac{1}{N} + \frac{1}{Dk^2} \qquad (2.8)$$

Розглянемо N=100 і Dk = 512. Тоді Ratio (R) = 0.010004 Це означає, що мережа згортки, що розділяється на глибину, у цьому прикладі виконує в 100 разів менше множень порівняно зі стандартною структурною нейронною мережею. Це означає, що ми можемо розгортати швидші моделі нейронної мережі згортки без значної втрати точності. Незважаючи на те, що базова архітектура MobileNet вже невелика і не потребує великих обчислень, вона має два різних глобальних гіпер параметри для подальшого ефективного зниження витрат на обчислення. Один з них - множник ширини, інший - множник роздільної здатності.

Для подальшого зниження обчислювальних витрат був запроваджений параметр під назвою Width Multiplier - який позначається в вигляді α константи. Для кожного шару множник ширини α буде помножено на вхідний і вихідний канали (N і M), щоб звузити мережу. Отже, обчислювальні витрати з множником ширини стануть: Dk× Dk×αM×Df×Df+αN×αM×Df×Df. Припустимо, що α змінюватиметься від 0 до 1 із типовими значеннями [1, 0,75, 0,5 та 0,25]. Коли α = 1, що називається базовим рівнем MobileNet, і α < 1, яке називається скороченим MobileNet. Множник ширини має ефект зменшення обчислювальних витрат на α^2.

Другий параметр для ефективного зниження витрат на обчислення позначається. Для даного шару множник роздільної здатності буде помножено на вхідну карту функцій. Тепер ми можемо виразити обчислювальну вартість, застосовуючи множник ширини та множник роздільної здатності, як: Dk× Dk×αM×ρDf×ρDf+αN×αM×ρDf×ρDf.

MobileNet використовує функцію активації ReLU6, що подібно до ReLU, але він запобігає тому, щоб активації ставали занадто великими:

$$y = \min(\max(0, x), 6) \qquad (2.9)$$

ReLU6 більш надійний, ніж звичайний ReLU. Це також робить форму функції більш схожою на сигмоід.

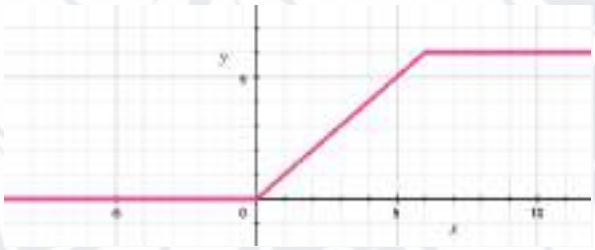


Рисунок 2.25 - Графік функції активації ReLu6

Функція Softmax, або ж нормована експоненційна функція - це узагальнення логістичної функції, що «стискує» *K*-вимірний вектор z із довільним значеннями компонент до *K*-вимірного вектора $\sigma(z)$ з дійсними значеннями компонент в області [0, 1] що в сумі дають одиницю. Функція задається формулою:

$$\sigma : \mathbb{R}^K \to [0,1]^K$$
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K. \qquad (2.10)$$

Функція софтмакс використовується в різних методах багатокласової

класифікації, таких, як наприклад мультиноміальна логістична регресія (також відома як софтмакс-регресія), багатокласовий лінійний розділювальний аналіз, наївний баєсів класифікатор, і штучні нейронні мережі. Зокрема, у мультиноміальній логістичній регресії та лінійному дискримінантному аналізі вхідними даними функції є результати K різних лінійних функцій, а прогнозована ймовірність для j-го класу з урахуванням вектора вибірки x і вектора ваги w є:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}} \tag{2.11}$$

Це можна розглядати як композицію K лінійних функцій зображену на Рисунку1 і функції softmax (де $\chi^\tau \omega$ позначає внутрішній добуток χ і ω). Операція еквівалентна застосуванню лінійного оператора, визначеного за допомогою ω до векторів χ , перетворюючи таким чином вхідний, багатовимірний, вектор аргументів на вектор у к-вимірному просторі:

$$\mathbf{x} \mapsto \mathbf{x}^\mathsf{T}\mathbf{w}_1, \ldots, \mathbf{x} \mapsto \mathbf{x}^\mathsf{T}\mathbf{w}_K \tag{2.12}$$

Функція softmax використовується в останньому шарі класифікаторів на основі нейронних мереж. Такі мережі зазвичай навчаються за допомогою перехресної ентропії, що дає нелінійний варіант поліноміальної логістичної регресії. Оскільки функція переводить вектор q і певний індекс в дійсне число, то похідна повинна враховувати ще й індекс:

$$\frac{\partial}{\partial q_k}\sigma(\mathbf{q}, i) = \sigma(\mathbf{q}, i)(\delta_{ik} - \sigma(\mathbf{q}, k)). \tag{2.13}$$

Геометрично функція softmax відображає векторний простір R^κ на межі (K-1)-вимірного симплекса, зменшуючи розмірність на одиницю (діапазоном значень стає (K-1)-вимірний симплекс в К-вимірному просторі), через лінійне

обмеження, що сума елементів вихідного вектору дорівнює 1, що означає, що він лежить на гіперплощині. По головній діагоналі (x, x, …, x), softmax стає просто рівномірним розподілом, (1/n, .., 1/n): рівна ваги даються рівні ймовірності.

Загалом, softmax є інваріантним щодо зсуву на одне й те саме значення в кожній координаті: додавання c = (c, … , c) до верктору вхідних значень Z дає $\sigma(z + c) = \sigma(z)$, оскільки softmax множить кожен показник на один і той же коефіцієнт, e^c, тобто співвідношення не змінюється:

$$\sigma(\mathbf{z}+\mathbf{c})_j = \frac{e^{z_j+c}}{\sum_{k=1}^{K} e^{z_k+c}} = \frac{e^{z_j} \cdot e^{c}}{\sum_{k=1}^{K} e^{z_k} \cdot e^{c}} = \sigma(\mathbf{z})_j. \quad (2.14)$$

Геометрично, softmax є постійним уздовж діагоналей: це відповідає тому, що вихідне значення softmax не залежить від зсуву вхідних значень.

В машинному навчанны в контексті завдань статистичної класифікації, існує важливий інструмент - "матриця невідповідностей" або "матриця помилок". Ця матриця представляє собою таблицю, яка дозволяє оцінювати продуктивність алгоритму класифікації, зазвичай в контексті навчання з учителем. У навчанні без учителя, аналогічний інструмент називається "матрицею допасованості". Кожен рядок в цій матриці відповідає прикладам, які були класифіковані відповідно до певного класу, тоді як кожен стовпець відповідає дійсним класам (або навпаки). Назва цієї матриці походить від її здатності відображати, чи виникають невідповідності між передбаченими та дійсними класами, наприклад, чи робить система помилкові класифікації. Можна вважати цю матрицю спеціальним варіантом таблиці спряженості з двома вимірами, де один вимір відображає "справжній" клас, а інший - "прогнозований" клас, і мають однаковий набір "класів" у обох вимірах, що робить кожну комбінацію виміру та класу змінною цієї таблиці спряженості.

Матриця невідповідностей зображена на рисунку 2.26, де :

- P - це кількість реальних позитивних випадків у даних;
- N - кількість реальних негативних випадків у даних;

		Прогнозований клас	
	Загальна кількість = P + N	Predicted Positive (PP)	Predicted Negative (PN)
Справжній клас	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Рисунок 2.26- Врахування індексу в обчисленнях

З матриці невідповідності ми можемо вивести формули точності та F1 міри. Точність – це частка правильних прогнозів (як істинно позитивних, так і істинно негативних) серед загальної кількості досліджених випадків. Таким чином, порівнюються оцінки ймовірності до та після. Формула для кількісного визначення бінарної точності наступна:

$$T = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.15)$$

де TP – істинно позитивне;
FP – хибно позитивне;
TN – істинно негативне;
FN – хибно негативне.

При обчисленні точності в багатокласовій класифікації, точність є часткою правильних класифікацій. Зазвичай виражається у відсотках. Наприклад, якщо класифікатор робить десять прогнозів і дев’ять з них правильні, то точність становить 90 %.

F-міра – це одна з мір точності тесту. Її обчислюють через влучність та повноту тесту, де влучність є числом правильно визначених позитивних результатів, поділеним на число всіх позитивних результатів, включно з

визначеними неправильно, а повнота є числом правильно визначених позитивних результатів, поділеним на число всіх зразків, які повинно було бути визначено як позитивні. Традиційна F-міра, або збалансована F-оцінка є середнім гармонійним влучності та повноти:

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{tp}}{2\mathrm{tp} + \mathrm{fp} + \mathrm{fn}} \quad (2.16)$$

Загальнішою F-мірою, Fβ, що використовує додатний дійснозначний коефіцієнт β, де β обирають так, що повноту вважають у β разів важливішою за влучність, є :

$$F_\beta = (1 + \beta^2) \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{(\beta^2 \cdot \mathrm{precision}) + \mathrm{recall}} \quad (2.17)$$

В термінах помилок першого і другого роду це стає:

$$F_\beta = \frac{(1 + \beta^2) \cdot \mathrm{true\ positive}}{(1 + \beta^2) \cdot \mathrm{true\ positive} + \beta^2 \cdot \mathrm{false\ negative} + \mathrm{false\ positive}} \quad (2.18)$$

Двома широко вживаними значеннями β є 2, яке надає повноті більшої ваги, ніж влучності, та 0,5, яке надає повноті меншої ваги, ніж влучності.

F-міру було виведено таким чином, що Fβ «вимірює ефективність пошуку з урахуванням користувача, який надає в β разів вищої важливості повноті, ніж влучності». Вона ґрунтується на мірі ефективності Van Rijsbergen's:

$$E = 1 - \left(\frac{\alpha}{p} + \frac{1 - \alpha}{r}\right)^{-1} \quad (2.19)$$

Cross Entropy - є мірою помилки для задачі багатокласової класифікації. Зазвичай «істинний» розподіл (той, якому намагається відповідати алгоритм

машинного навчання) виражається в термінах унітарного кодування. Перехресна ентропія двох розподілів і на тому самому ймовірнісному просторі визначається наступним чином. Для дискретного випадку і над одним і тим же носієм X це значить, що:

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x) \tag{2.20}$$

Для неперервного розподілу аналогічна ситуація. Ми припускаємо, що та абсолютно неперервні відносно деякої міри r. Нехай P та Q будуть функціями густини ймовірностей p та q будуть відносно r. Тоді:

$$H(p, q) = -\int_{\mathcal{X}} P(x) \log Q(x)\, dr(x) \tag{2.21}$$

Стохастичний градієнтний спуск – це ітераційний метод для оптимізації цільової функції з відповідними властивостями гладкості. Його можна розглядати як стохастичне наближення оптимізації градієнтного спуску, оскільки воно замінює фактичний градієнт (розрахований на основі всього набору даних) його оцінкою (розрахованою на основі випадково обраної підмножини даних). У стохастичному градієнтному спаді справжній градієнт апроксимується градієнтом в одній вибірці:

$$w := w - \eta \nabla Q_i(w). \tag{2.22}$$

Припустимо, ми хочемо вписати пряму лінію y=w1+w2x до навчального набору з спостереженнями (x1, x2, ... , xn) і відповідні оцінені відповіді (y1, y2, ... , yn) за допомогою найменших квадратів. Цільовою функцією, яку необхідно мінімізувати, є:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2. \tag{2.23}$$

Класичний стохастичний градієнтний спуск зазвичай чутливий до швидкості навчання η. Швидка конвергенція вимагає великих темпів навчання, але це може спричинити чисельну нестабільність. Проблему можна значною мірою вирішити шляхом розгляду неявних оновлень, за допомогою яких стохастичний градієнт оцінюється на наступній ітерації, а не на поточній:

$$w^{\text{new}} := w^{\text{old}} - \eta \nabla Q_i(w^{\text{old}}) \tag{2.24}$$

Це рівняння неявне, оскільки w з'являється з обох сторін рівняння. Це стохастична форма методу проксимального градієнта, оскільки оновлення також можна записати як:

$$w^{\text{new}} := \arg\min_w \left\{ Q_i(w) + \frac{1}{2\eta} \left\| w - w^{\text{old}} \right\|^2 \right\} \tag{2.25}$$

Momentum Стохастичний градієнтний спуск із імпульсом запам'ятовує оновлення Δw на кожній ітерації та визначає наступне оновлення як лінійну комбінацію градієнта та попереднього оновлення.

$$\begin{aligned} \Delta w &:= \alpha \Delta w - \eta \nabla Q_i(w) \\ w &:= w + \Delta w \end{aligned} \tag{2.26}$$

Усереднений стохастичний градієнтний спуск – це звичайний стохастичний градієнтний спуск, який записує середнє значення свого вектора параметрів за час. Тобто оновлення відбувається так само, як і для звичайного стохастичного градієнтного спуску, але алгоритм також відстежує:

$$\bar{w} = \frac{1}{t} \sum_{i=0}^{t-1} w_i. \tag{2.27}$$

Коли оптимізація виконана, цей усереднений вектор параметра займає місце w. AdaGrad (для адаптивного градієнтного алгоритму) – це модифікований стохастичний градієнтний алгоритм зі швидкістю навчання за параметром,

Неформально це збільшує швидкість навчання для менш розріджених параметрів і зменшує швидкість навчання для менш розріджених. Ця стратегія часто покращує продуктивність конвергенції порівняно зі стандартним стохастичним градієнтним спуском у налаштуваннях, де дані розріджені, а розріджені параметри більш інформативні. Приклади таких програм включають обробку природної мови та розпізнавання зображень.

Вона все ще має базову швидкість навчання η, але вона множиться на елементи вектора {Gj,j}, який є діагоналлю зовнішньої матриці добутку:

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^{\mathsf{T}} \tag{2.28}$$

де $g_r = \triangle Q_i(w)$, градієнт, на ітерації τ. Діагональ визначається як $g_r = \triangle Q_i(w)$.

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2. \tag{2.29}$$

Цей вектор, по суті, зберігає історичну суму градієнтних квадратів за розмірами та оновлюється після кожної ітерації. Формула для оновлення:

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j. \tag{2.30}$$

Важливою властивістю правила оновлення Адама є ретельний вибір розмірів кроків. Припускаючи, що = 0, ефективний крок, зроблений у просторі параметрів на кроці часу t is Δt = α · mb t/√vbt. Ефективний розмір кроку має дві верхні межі: $|\Delta t| \leq \alpha \cdot (1 - \beta 1)/\sqrt{1 - \beta 2}$ в випадку$(1 - \beta 1) > \sqrt{1 - \beta 2}$, і $|\Delta t| \leq \alpha$ інакше. Перший випадок трапляється лише у найважчому випадку розрідженості: коли градієнт дорівнює нулю на всіх часових кроках, крім поточного. Для менш розріджених випадків ефективний розмір кроку буде меншим.

ВИСНОВКИ ДО РОЗДІЛУ 2

Реалізація автоматизованої системи парсингу вимагає застосування технологій для побудови нейронної мережі та рлзробки системи парсингу, до яких віднесено: програмування на мові Python з бібліотеками numpy, keras, scikit-learn, selenium, webdriver, pillow; середовище розробки програмного коду PyCharm та Jupyter Notebook; середовища сервінгу моделі tensorflow та docker.

Вхідний потік даних відтворюється набором даних (датасетом), формування якого проходиться наступні етапи: конкретизація мети використання датасету (зокрема, класифікація зображень); вибір джерел та типу вхідних даних (з урахуванням доступності визначено: Unsplash, Pexels, Instagram, Pixabay, Google); категорізація об’єктів за ознакою наявності маски; збір зображень для забезпечення репрезентативності вибіркової оцінки; перевірка якості зображень; анотування; формування підвибірок для машинного навчання та оцінювання.

Сутність моделі розпізнавання людини в медичній масці представляє собою процес імітований нейронною мережею, побудова якої обгрунтовується математичною теорією

Практична реалізація автоматизованого парсингу використовує аналіз архітеркурних рішень та компонентної структури Selenium WebDriver, що дозволяє виділити основні складові механізму парсингу та відтворити їх програмну реалізацію.

РОЗДІЛ 3
АВТОМАТИЗОВАНА СИСТЕМА ПАРСИНГУ ФОТОГРАФІЙ

3.1 Дизайн та структура проекту системи

Перед тим, як почати працювати над практичною частиною - потрібно чітко сформулювати послідовність дій. Для цього був проведений брейн-штурм, метою якого була складена блок схема, наведена нижче.

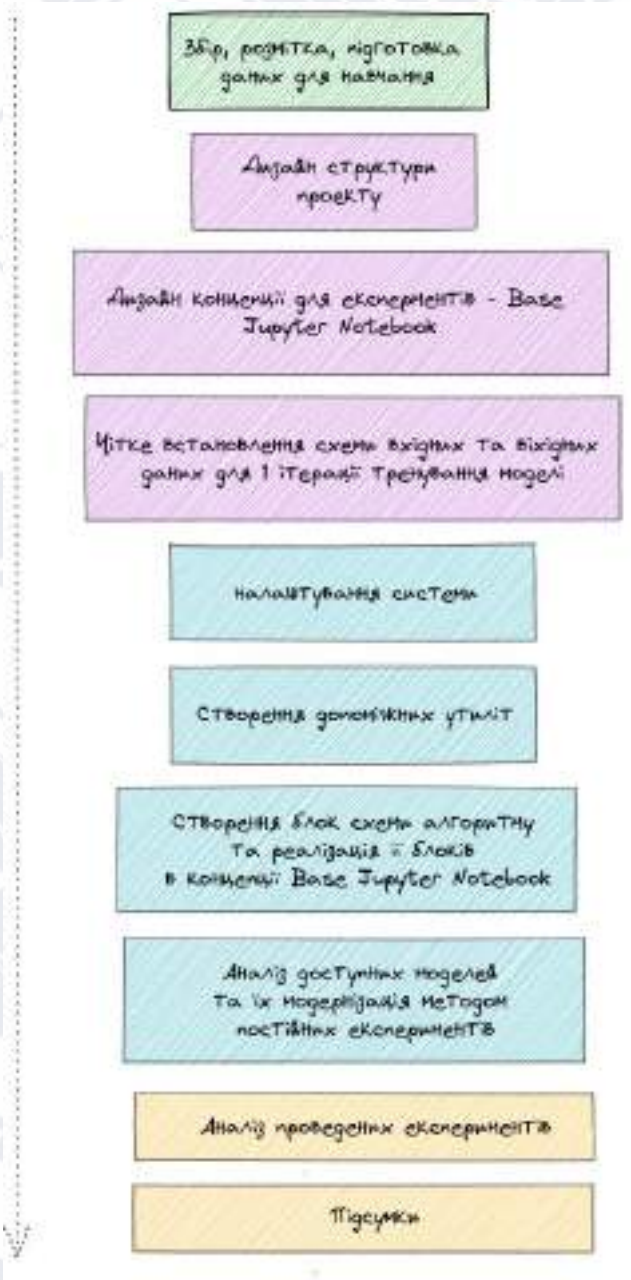


Рисунок 3.1 - Блок-схема реалізації практичної частини

На схемі ми бачимо декілька кольорових частин, які відповідають за умовні окремі фази практичного завдання. Значення кольорів розшифровано нижче.

1. Зелений колір - охоплює все, що пов’язано з роботою вхідних даних.

2. Рожевий колір - дизайн фаза практичної реалізації. Вона не включає в себе жодних взаємодій з кодом.

3. Блакитний колір - охоплює всі процедури пов’язані з написанням програмного коду та налаштування робочої системи для дослідження. В цій фазі відбувається взаємодія з нейронною мережею, її дослідження та модернізація. Відбувається реалізація допоміжних утиліт.

4. Жовтий колір - це кінцева фаза виконання практичного завдання. Вона відбувається тоді, коли вже весь основний код реалізації написаний, проведено усі потрібні експерименти та починається перехід до створення підсумків виконання роботи.

Проект, розроблений на мові Python включає в себе наступні складові: code, configurations, outputs. Структура проекта зображена на Рисунку 3.2.

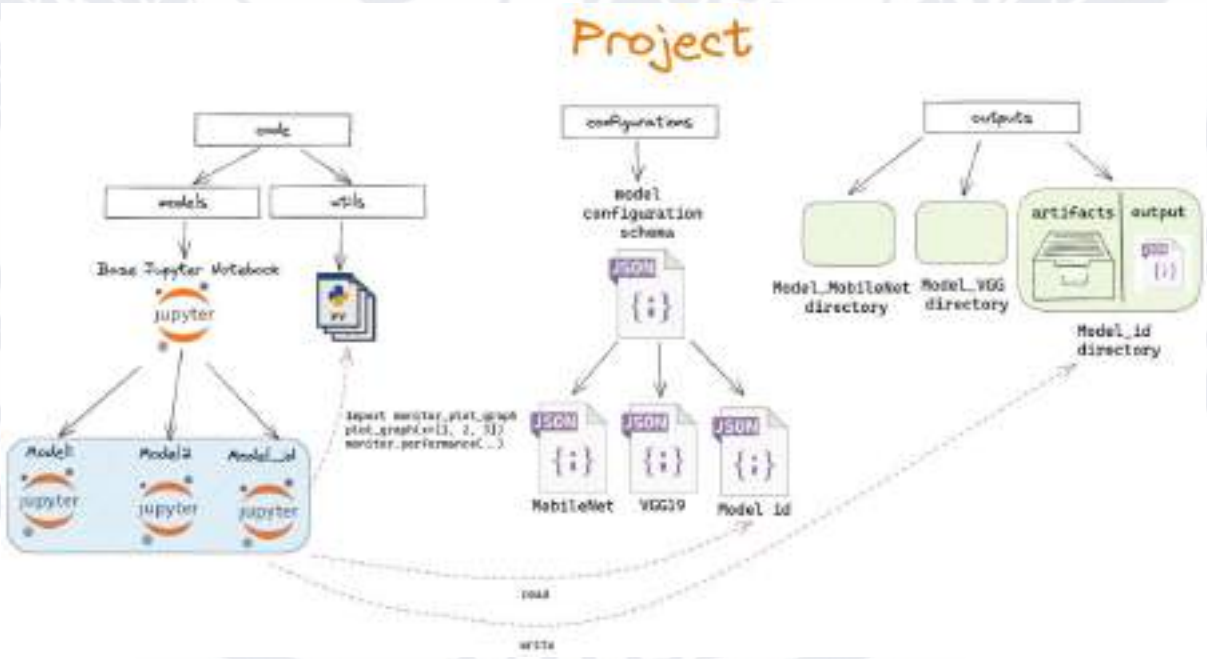


Рисунок 3.2 - Структура проекту моделі

Директорія `code` - містить в собі весь написаний програмний код на мові програмування Python, тобто дана директорія містить фактично всю реалізацію практичної частини нейронної мережі у вигляді програмного коду. Ця директорія містить також в собі підмножину папок :

- `models` - місце, де зберігаються усі виконані Jupyter Notebooks проекту та власне Base Jupyter Notebook, який слугує базовим для всіх Jupyter Notebooks дослідження.

- `utils` - місце, де зберігаються Python модулі з кодом, які будуть використовуватись Jupyter Notebooks під час їх запуску використовуючи їх за допомогою звичайних вбудований операцій “import”. Модулі в Python - це файли з розширенням .py, які містять функції, класи та інші об’єкти, які можуть бути використані програмі. За допомогою інструкції import, я можу включити ці модулі у свою програму і використовувати їх функціонал.

Директорія `utils` має структуру зображену на рисунку 3.3.

- model.py - містить класи та функції необхідні для тренування, модифікації нейронних мереж.

- plot.py - містить класи та функції необхідні для візуалізації даних та результатів аналізу.

- monitor.py - містить класи та функції, які відповідають за логування операцій, збереження результатів дослідження, виведення інформацій в Jupyter Notebook, створення вихідного файлу після запуску Jupyter Notebook.

- test.py - вміщує в собі тести модулів зазначених вище.

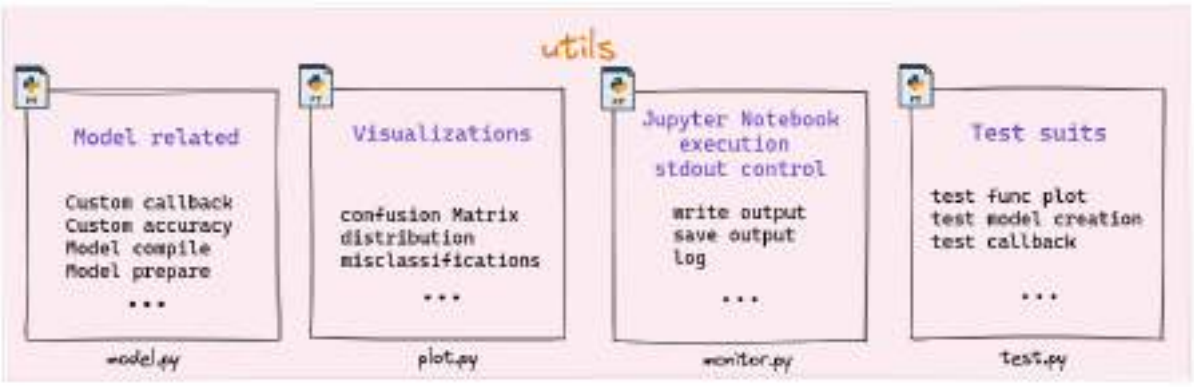


Рисунок 3.3 - Структура директорії utils.

Розберемо в деталях одну з написаних функцій файлу plot.py зображену на рисунку 3.3. Функція `plot_distribution_hist`, отримує список значень `values` і виконує наступні кроки:

1. Імпортує необхідні модулі: `numpy`, `matplotlib.pyplot`, `scipy.stats`. Це забезпечує доступ до функцій і класів, необхідних для обчислень та побудови графіків.

2. Створює фігуру та вісі графіка: Це виконується за допомогою `plt.subplots(figsize=(8, 6))`. Фігура та вісі графіка створюються для подальшого побудови гістограми розподілу.

3. Побудова гістограми: Використовуючи `ax.hist(values, edgecolor='black', color='#4C72B0', alpha=0.7)`, побудована гістограма з кольором, означеним як блакитний (`#4C72B0`). Результати гістограми зберігаються у змінній `counts`, `bins`.

4. Налаштування заголовка та міток вісей: Використовуються `ax.set_title()`, `ax.set_xlabel()`, `ax.set_ylabel()` для встановлення заголовка графіка та підписів вісей x та y.

5. Додавання сітки: За допомогою `ax.grid(True, linestyle='--', alpha=0.5)` додається сітка на графік для покращення його зовнішнього вигляду.

6. Налаштування розмірів шрифтів та видимості рамок: `ax.tick_params()` встановлює розмір шрифту для міток на вісях, а `ax.spines['top'].set_visible(False)`, `ax.spines['right'].set_visible(False)` приховує верхню та праву рамки графіка.

7. Відображення графіка: `plt.tight_layout()` забезпечує оптимальне розташування елементів графіка на фігурі, і `plt.show()` відображає графік.

8. Обчислення характеристик розподілу: Використовуються функції з модулю `numpy` для обчислення мінімального значення (`np.min`), максимального значення (`np.max`), медіани (`np.median`), середнього значення (`np.mean`), стандартного відхилення (`np.std`), а також коефіцієнтів асиметрії (`skew`) та ексцесу (`kurtosis`) за допомогою модулю `scipy.stats`.

9. Повернення результатів: Характеристики розподілу упаковуються у

словник і повертаються з функції. Отже, ця функція обчислює та відображає гістограму розподілу значень `values` та повертає основні статистичні характеристики цього розподілу (додаток Б, рис. Б.1).

Файл JSON - це файл, що містить дані в форматі JSON. Він зазвичай має розширення .json. Файли JSON використовуються для зберігання, обміну та передачі даних між різними програмами та системами.

Директорія `configurations` містить в собі конфігураційні файли для різних типів моделей.

Директорія `outputs` - директорія, яка містить в собі вихідні, які були отримані в ході тренування та оцінки моделі. Директорія в собі містить інші директорії, де назва кожної директорії відповідає ідентифікаційному номеру експерименту. Папка експерименту вміщує в собі артефакти, які згенерувались внаслідок прогону ноутбуку та вихідний json файл. Структура `output.json` файла наведена у додатку Б (рис. Б.2).

3.2 Концепція FlowModeler при створенні нейронної мережі

Концепція FlowModeler - це назва власної концепції підходу до вирішення поставленої задачі. Її сутність полягає в тому, щоб зробити дослідження моделей нейронних мереж та дослідження їх модифікацій зручним та ефективним.

Основний зміст концепції - використовувати єдиний Jupyter Notebook - для великої кількості поставлений експериментів. Концепція надає наступні переваги :

- Можливість автоматизувати ітерації експериментів
- Збирати результатів експериментів в централізованому місті
- Інкапсулювати блоки роботи з нейронною мережею
- Одноразове написання коду. Після того, як базовий ноутбук написаний, в ньому потрібно робити буде лише необхідні правки для конкретних кутових випадків.
- Експеримент оприділяється конфігураційним файлом, що є надзвичайно зручно та еластично. Конфігураційний файл дозволяє змінювати поведінку

програми або системи без необхідності внесення змін безпосередньо у вихідний код. Він дозволяють налаштовувати різні параметри, такі як розмір вікон тренувань моделі, шляхи до файлів, рівні логування і багато іншого.

- Контент FlowModeler Jupyter Notebook представлений набором зв’язаних блоків. Кожен блок представлений коміркою Jupyter Notebook, що дозволяє легко відредагувати, додати або видалити потрібний блок в разі необхідності. [28]

FlowModel концепт зображено на рисунку 3.4.

Рисунок 3.4 - FlowModeler концепт

FlowModel концепт має наступні блоки:

1. Налаштування Jupyter Schema.

2. Імпорти модулів.

3. Зчитування та ініціалізація конфігурації.

встановлення параметрів виконання ноутбука.

4. Ініціалізація базових об’єктів.

5. Налаштування фізичних девайсів для тренування моделі.

6. Зчитування даних, створення генераторів даних.

7. Препроцесинг даних.

8. Ініціалізація та налаштування базової моделі.

9. Встановлення коллбеків навчання, ініціалізація кастомних метрик.

10. Модернізація моделі та компілювання моделі.

11. Тренування моделі та збереження результатів тренування.

12. Збереження та візуалізація результатів тренування.

13. Повторення (10, 11, 12) в разі необхідності.

14. Оцінка моделі на тестовій виборці.

15. Аналіз поведінки моделі на конкретних даних.

16. Візуалізація виконання слоїв (feature extraction of cnn layers).

17. Фіксування часу виконання блоків, збереження метрик виконання ноутбуку.

Отже, ми ознайомились з логічними блоками - які власне собою являють блок-схему алгоритму навчання моделі. Тепер перейдемо до розгляду структури конфігураційного файлу - `configuration.json`, яка відображена у додатку Б на рисунку Б.3.

Цей JSON файл містить наступну структуру:

1. Ключ "CATEGORIES_NUMBER" вказує на загальну кількість категорій.

2. Ключ "MODEL" містить модель із наступними підключами:

- "name": назва моделі.

- "fine_tune_layers": масив, що містить послідовність шарів, які будуть налаштовуватися. Кожен шар представлений об’єктом з ключами "type" (тип шару) і "parameters" (параметри шару).

- "class_weights": об’єкт, який містить ваги класів. Ключі представляють індекси класів, а значення - ваги.

- "loss": функція втрат, наприклад, "categorical_crossentropy".

- "optimizer_base": базовий оптимізатор з ключами "type" (тип оптимізатора) і "parameters" (параметри оптимізатора).

- "optimizer_fine_tune": оптимізатор для налаштування з ключами "type" (тип оптимізатора) і "parameters" (параметри оптимізатора).

- "callbacks": масив об’єктів, що містять зворотні виклики моделі. Кожен об’єкт має ключ "type" (тип зворотного виклику) і "parameters" (параметри зворотного виклику).

- "layers_to_unfreeze_fine_tune": кількість шарів, які розморозяться для налаштування.

- "epoch_number_appended_layers": кількість епох для навчання доданих шарів.

- "epoch_number_fine_tune": кількість епох для налаштування моделі.

3. Ключ "DATA_AUGMENTATION_TRAIN" містить параметри аугментації даних для тренування моделі. Включає такі параметри:

- "samplewise_std_normalization": прапорець для виконання нормалізації по стандартному відхиленню.

- "horizontal_flip": прапорець для горизонтального перекидання зображень.

- "brightness_range": діапазон яскравості.

- "rotation_range": діапазон обертання зображень.

- "zoom_range": діапазон зуму зображень.

- "fill_mode": режим зап

4. Ключ "ID" містить ідентифікатор моделі, у цьому випадку, наприклад, "MobileNet_3cl".

5. Ключ "OUTPUT_FOLDER_PATH" містить шлях до вихідної теки, наприклад,"/home/vladislav/PycharmProjects/shared-backup/DonnuFaceMask/ outputs/ MobileNet_3cl".

Тепер перейдемо до розгляду структури вихідного файлу - `output.json`, який відображений у додатку Б на рисунку Б.4. Цей JSON файл містить наступну структуру:

1. Ключ "id" містить ідентифікатор моделі, у цьому випадку "MobileNet_4cl".

2. Ключ "model_scores" містить оцінки моделі і має наступні підключі:

- "train_model_classification_layers_stage_best_scores" містить найкращі

оцінки моделі на етапі тренування класифікаційних шарів. Включає оцінки для набору даних "train" та "validation", які в свою чергу містять оцінки "min_loss" (мінімальна втрата), "max_accuracy" (максимальна точність) та "max_custom_accuracy" (максимальна користувацька точність). Також містить "optimal_epoch_number" (оптимальна кількість епох) та "model_train_time" (час тренування моделі).

- "train_model_fine_tune_stage_best_scores" містить найкращі оцінки моделі на етапі налаштування. Включає оцінки для набору даних "train" та "validation" з аналогічною структурою до попереднього ключа.

3. Ключ "test_score" містить оцінки моделі на тестовому наборі даних. Включає "loss" (втрата), "accuracy" (точність) та "custom_acc" (користувацька точність).

4. Ключ "model_prediction_measurement" містить виміри передбачень моделі. Містить наступні підключі:

- "experiments_prediction_time_results" - масив результатів часу передбачень для експериментів.

- "input_image_shape" - масив розмірів вхідних зображень.

- "unit_of_measurement" - одиниця виміру часу передбачень (у цьому випадку "ms").

- "time_results_hist_description" - опис гістограми результатів часу передбачень, включає "min" (мінімальне значення), "max" (максимальне значення), "median" (медіану), "mean" (середнє значення), "std" (стандартне відхилення), "skewness" (асиметрію) та "kurtosis" (ексцес).

3.3 Контент FlowModeler Base Jupyter Notebook та реалізація нейронної мережі

Перейдемо до розгляду реалізації коду в Base Jupyter Notebook. Цей код використовує бібліотеку jupyterthemes для налаштування зовнішнього вигляду графічних візуалізацій у середовищі Jupyter Notebook. Таким чином, весь цей код встановлює стилі оформлення для графічних елементів у Jupyter

Notebook, включаючи тему оформлення, розмір фігури, шрифти, границі та сітку. Ви можете змінювати ці параметри, щоб відповідати вашим потребам і створювати більш привабливі візуалізації. [31]

```
from jupyterthemes import jtplot
jtplot.style(theme='monokai',context="notebook",ticks=True, grid=False)
jtplot.style(context='talk', fscale=1.2, spines=False, gridlines='--')
jtplot.style(figsize=(6, 4.5))
```

Рисунок 3.5 - Налаштування зовнішнього вигляду графічних візуалізацій у середовищі Jupyter Notebook

Далі ми імпортуємо різні модулі та функції, які використовуються для роботи з нейромережами та аналізу даних зображень.

```
import json, pprint, os, time
from plot_utils import plot_training_and_validation_custom_accuracy, \
    plot_training_and_validation_accuracy_and_loss, plot_confusion_matrix, \
    plot_distribution_hist, misclassification_visualizations
from model_utils import custom_acc, prepare_model, \
    unfreeze_model_layers, compile_model, materialize_callbacks
from monitor import NotebookOutputs
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Рисунок 3.6 - Імпорт різних модулів та функцій

Оголошуємо змінні, які будуть використовуватись як параметри ноутбука.

```
PROJECT_PATH = '/home/vladislav/PycharmProjects/tftest'
# PATH FOLDERS
TRAIN_IMAGES_FOLDER_PATH = f'{PROJECT_PATH}/Images/Train'
VALIDATION_IMAGES_FOLDER_PATH = f'{PROJECT_PATH}/Images/Validation'
TEST_IMAGES_FOLDER_PATH = f'{PROJECT_PATH}/Images/Test'
MAX_TRAIN_DATASET_LEN=len(os.listdir
                    ('/home/vladislav/PycharmProjects/tftest/Images/Train/0_without_mask'))
MAX_VALIDATION_DATASET_LEN=len(os.listdir
                    ('/home/vladislav/PycharmProjects/tftest/ImagesValidation/0_without_mask'))
.....
BATCH_SIZE = 32
SEED = 5
IMAGE_SIZE = 224
IMAGE_DIMENSION = 3
IMAGE_MODEL_SIZE = (IMAGE_SIZE, IMAGE_SIZE)
IMAGE_MODEL_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, IMAGE_DIMENSION)
DATA_CLASS_MODE = "categorical"
DEFAULT_MODEL_WEIGHTS = "imagenet"
MODEL_METRIC_ACCURACY = "accuracy"
```

Рисунок 3.7 - Оголошення параметрів ноутбука

Зчитуємо та ініціалізуємо конфігурацію ноутбука

```
CONFIGURATION = None
with open(CONFIGURATION_PATH) as config_file:
    CONFIGURATION = json.load(config_file)
if CONFIGURATION is None:
    raise ValueError("No configuration were defined!")
else:
    pprint.pprint(CONFIGURATION)
CATEGORIES_NUMBER = CONFIGURATION.get("CATEGORIES_NUMBER")
MODEL_NAME = CONFIGURATION.get("MODEL").get("name")
MODEL_FINE_TUNE_LAYERS = CONFIGURATION.get("MODEL").get("fine_tune_layers")
MODEL_CLASS_WEIGHTS = {int(key) : value for key, value in CONFIGURATION.get("MODEL").get("class_weights"
    ).items()}
MODEL_LOSS = CONFIGURATION.get("MODEL").get("loss")
...
```

Рисунок 3.8 - Зчитування конфігурації ноутбука

Далі використовуєм для забезпечення більш ефективного використання пам’яті GPU при роботі з TensorFlow. Спочатку використовуєм TensorFlow для отримання списку всіх фізичних пристроїв, які визначаються як ‘GPU’. Це може включати одну або декілька графічних карт, які встановлені у вашій системі. Наступний крок - це прохід по всіх виявлених GPU і встановлення політики росту пам’яті для кожного з них. Коли росту пам’яті встановлено в True, TensorFlow алокує пам’ять по мірі потреби, використовуючи лише те, що потрібно для проведення операцій, а не займаючи всю доступну пам’ять GPU відразу. Завершується цей код записом в журнал у форматі ‘notebook_outputs.log’ статусу доступності GPU для поточного середовища. Використовується умовний вираз True if tf.test.gpu_device_name() else False - якщо TensorFlow може отримати назву GPU-пристрою, то GPU доступний (повертає True), інакше - недоступний (False). [32]

```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)

notebook_outputs.log(f"GPU device availability status
            for environment -> {True if tf.test.gpu_device_name() else False}")
```

Рисунок 3.9 - GPU налаштування

Наступний код є частиною процесу підготовки даних для моделі глибокого навчання в бібліотеці Keras (TensorFlow). Даний підхід є важливий для ефективного тренування нейронних мереж на великих наборах даних, особливо коли вони не поміщаються в пам’ять. Початок коду визначає кількість кроків на епоху для тренувального та валідаційного наборів даних. Це робиться шляхом ділення загальної кількості елементів в тренувальному та валідаційному наборах даних (MAX_TRAIN_DATASET_LEN таMAX_VALIDATION_DATASET_LEN) на розмір пакету (BATCH_SIZE).+1 додається, щоб врахувати остачу від ділення. Інформація про кількість кроків на епоху виводиться в журнал. [33]

```
STEPS_PER_ONE_EPOCH = MAX_TRAIN_DATASET_LEN//BATCH_SIZE+1
STEPS_PER_ONE_EPOCH_VALIDATION = MAX_VALIDATION_DATASET_LEN//BATCH_SIZE+1
notebook_outputs.log(f"Steps number per training epoch : "
                     f"{STEPS_PER_ONE_EPOCH}")
notebook_outputs.log(f"Steps number per validation epoch : "
                     f"{STEPS_PER_ONE_EPOCH_VALIDATION}")
```

Рисунок 3.10 - Кількість кроків на епоху для тренувального та валідаційного наборів даних

Далі створюються генератори даних для тренування, валідації та тестування за допомогою класу ImageDataGenerator. Генератори даних використовуються для пакетного завантаження та обробки зображень з диску. Для тренувального генератора використовуються параметри аугментації даних (DATA_AUGMENTATION_TRAIN), що дозволяє збільшити набір даних і зробити модель більш стійкою до різних видів зображень. Валідаційний та тестовий генератори нормалізують дані, ділячи кожен піксель на стандартне відхилення всього зображення (samplewise_std_normalization=True).

```
train_gen = ImageDataGenerator(**DATA_AUGMENTATION_TRAIN)
...
valid_gen = ImageDataGenerator(samplewise_std_normalization=True,)
...
test_gen = ImageDataGenerator(samplewise_std_normalization=True,)
```

Рисунок 3.11 - Генератори даних для тренування, валідації та тестування

Потім з цих генераторів створюються ітератори для тренувального, валідаційного та тестового наборів даних, які використовують метод flow_from_directory. Цей метод завантажує зображення з вказаного каталогу, масштабує їх до вказаного розміру (IMAGE_MODEL_SIZE), використовує вказаний режим класу (DATA_CLASS_MODE) та визначає класи зображень на основі вказаних папок (DIRECTORY_CLASSES_FOLDERS). Для валідаційного та тестового наборів даних вказано параметр shuffle=False, щоб забезпечити однаковий порядок зображень для кожного етапу тестування. Для валідаційного набору створено ще один генератор (valid_generator_cm), можливо, для використання в конфузійній матриці або інших метриках, що вимагають постійного порядку зображень.

```
train_generator = train_gen.flow_from_directory(...)
valid_generator = valid_gen.flow_from_directory(...)
valid_generator_cm = valid_gen.flow_from_directory(...)
test_generator = test_gen.flow_from_directory(...)
```

Рисунок 3.12 - Ітератори для тренувального, валідаційного та тестового наборів даних

На завершення коду записується в журнал інформація про те, що генератори даних були успішно створені [34].

Наступний код зосереджений на створенні та підготовці моделі машинного навчання, специфічної для наших потреб. Демонструє, як можна створювати та налаштовувати моделі нейронних мереж з TensorFlow та Keras. Спочатку ініціалізується змінна ml_model як None. Далі визначається словник base_mode_params з параметрами, які будуть використані при створенні базової моделі.

```
ml_model = None
base_mode_params = {
    "input_shape": IMAGE_MODEL_SHAPE,
    "weights": DEFAULT_MODEL_WEIGHTS,
    "include_top": False
}
```

Рисунок 3.13 - Задання параметрів базової моделі

Далі в залежності від значення MODEL_NAME, вибирається одна з двох доступних базових моделей, MobileNet або VGG16. Якщо вказана інша модель, то з'явиться помилка. Після створення базової моделі вона передається в функцію prepare_model разом з додатковими шарами, що будуть додані до неї.

```
if MODEL_NAME == "MobileNet":
    from tensorflow.keras.applications import MobileNet
    base_model = MobileNet(**base_mode_params)
elif MODEL_NAME == "VGG16":
    from tensorflow.keras.applications import VGG16
    base_model = VGG16(**base_mode_params)
else:
    error_msg = f"No model=`{MODEL_NAME}` exists"
    notebook_outputs.log(error_msg)
    raise ValueError(error_msg)

ml_model=prepare_model(base_model=base_model,
                       append_layers=MODEL_FINE_TUNE_LAYERS)
```

Рисунок 3.14 - Вибір базової моделі

Потім створюються зворотні виклики (callbacks), які будуть використовуватися під час тренування моделі. Зворотні виклики це спеціальні функції, що виконуються в певні моменти тренування, наприклад після кожної епохи.

```
callbacks, custom_callbacks = materialize_callbacks(
    metric="val_accuracy",
    callbacks_configuration=MODEL_CALLBACKS,
    use_custom_metric=USE_CUSTOM_METRIC
)
```

Рисунок 3.15 - Створення коллбеків

Метрика, яка буде використовуватися для оцінки моделі, визначається в залежності від того, чи використовується користувацька метрика.

```
metric_custom_callback = custom_callbacks[0]
callbacks = custom_callbacks
metrics = [MODEL_METRIC_ACCURACY, custom_acc] if USE

_CUSTOM_METRIC else [MODEL_METRIC_ACCURACY]

notebook_outputs.log(f"Callbacks = {[callback for callback in callbacks]}.")
notebook_outputs.log(f"Metrics = {[metric for metric in metrics]}.")
```

Рисунок 3.16 - Визначення метрики

Наступний код демонструє процес тренування нейронної мережі з TensorFlow/Keras, включаючи контроль за якістю тренування та виконанням. Спочатку модель навчається за допомогою методу fit, використовуючи тренувальний генератор (train_generator), кількість кроків на епоху (STEPS_PER_ONE_EPOCH), кількість епох (EPOCH_NUMBER_ APPENDED_LAYERS), використовуючи валідаційний генератор (valid_generator), кількість кроків на валідацію (STEPS_PER_ ONE_EPOCH_VALIDATION), ваги класів (MODEL_CLASS_WEIGHTS) та зворотні виклики (callbacks). [35]

```
callback_train_model_classification_layers_stage=ml_model.fit(
    train_generator,
    steps_per_epoch=STEPS_PER_ONE_EPOCH,
    epochs=EPOCH_NUMBER_APPENDED_LAYERS, # 200
    verbose=1,
    validation_data=valid_generator,
    validation_steps=STEPS_PER_ONE_EPOCH_VALIDATION,
    class_weight=MODEL_CLASS_WEIGHTS,
    callbacks=callbacks
)
```

Рисунок 3.17 - Навчання моделі методом fit

Після тренування моделі, визначається словник train_model_classification_layers_stage_best_scores, що містить мінімальну втрату (min_loss), максимальну точність (max_accuracy) та максимальну користувацьку точність (max_custom_accuracy) для тренувального та валідаційного наборів даних, а також номер оптимальної епохи (optimal_epoch_number) та час тренування моделі (model_train_time). Та потім результати записуються в

вихідний файл.

```
train_model_classification_layers_stage_best_scores = { ... }
notebook_outputs.write_output(
    key="model_scores",
    value=train_model_classification_layers_stage_best_scores
)
```

Рисунок 3.18 - Визначення словника результатів тренування

Далі йде код, який розморожує (дозволяє тренування) певну кількість останніх шарів моделі, що дає можливість точніше налаштувати ці шари. Він демонструє підхід до точного налаштування нейронних мереж, коли певна кількість останніх шарів моделі розморожується для подальшого тренування.

```
if MODEL_NAME == "MobileNet":
    unfreeze_model_layers(model=ml_model, n_last_layers=MODEL_LAYERS_TO_UNFREEZE_FINE_TUNE)
elif MODEL_NAME == "VGG16":
    pass
else:
    error_msg = f"No model= {MODEL_NAME} exists"
    notebook_outputs.log(error_msg)
    raise ValueError(error_msg)

notebook_outputs.log(f"Model trainable variables : {len(ml_model.trainable_variables)}")
for i, layer in enumerate(ml_model.layers):
    print(f"{i}-{layer.name}-{layer.trainable}")
```

Рисунок 3.19 - Розморожування останніх слоїв моделі

Наступний код виконує оцінку моделі на тестовому наборі даних та створює матрицю невизначеності для результатів.

1. Метод evaluate моделі ml_model використовується для оцінки моделі на тестовому наборі даних, який генерується за допомогою test_generator. Результати зберігаються в змінній ml_model_evaluate_results

2. Результати оцінки моделі зберігаються за допомогою методу write_output класу notebook_outputs. Для кожної метрики з ml_model.metrics_names зберігається відповідне значення з ml_model_evaluate_results.

3. За допомогою функції plot_confusion_matrix створюється матриця невизначеності для результатів моделі на тестовому наборі даних. Це допомагає

візуалізувати, як модель працює на різних класах.

```
ml_model_evaluate_results = ml_model.evaluate(x=test_generator)
notebook_outputs.write_output(
    key="test_score",
    value={
        ml_model.metrics_names[i] : float(ml_model_evaluate_results[i])
            for i in range(len(ml_model.metrics_names))
    })

plot_confusion_matrix(model=ml_model,
                      data_generator=test_generator,
                      classes_list=range(CATEGORIES_NUMBER))
```

Рисунок 3.20 - Оцінка моделі на тестовому наборі даних та створює матрицю невизначеності для результатів.

Далі ми проходимо через набір даних із зображеннями людей із масками (`with_mask`), без масок (`without_mask`) та без людей (`without_person`), робить передбачення за допомогою моделі глибокого навчання (`ml_model`) та аналізує помилки передбачення. Він також вимірює час, який займає кожне передбачення.

1. `reshape_shape` визначає форму, в яку потрібно перетворити кожне зображення перед тим, як ввести його в модель.

2. Цикл `for` проходить через всі зображення в наборі даних.

3. Для кожного зображення він перетворює його на потрібну форму та нормалізує його, ділячи на стандартне відхилення.

4. Модель використовується для передбачення класу зображення. Час передбачення зберігається.

5. Якщо передбачення моделі не відповідає очікуваному класу, індекс зображення додається до відповідного списку помилок передбачення (`no_mask_false_detection`, `in_mask_false_detection`, `false_person_detection`).

6. Якщо виникає помилка (наприклад, якщо даних недостатньо), блок `except` пропускає поточне зображення та переходить до наступного.

В кінці ноутбуку знаходиться код, який використовується для візуалізації активаційних карт моделі глибокого навчання для конкретного зображення. Він

візуалізує вихід кожного шару моделі після застосування зображення.

```
no_mask_false_detection = []
in_mask_false_detection = []
false_person_detection = []

reshape_shape = (1, IMAGE_SIZE, IMAGE_SIZE, IMAGE_DIMENSION)
arr=np.asarray(with_mask[1]).reshape(*reshape_shape)
prediction_time_array = []

for i in range(size):
    try :
        with_mask_reshaped=np.asarray(with_mask[i]).reshape(*reshape_shape)
        ml_model_input = with_mask_reshaped/np.std(with_mask_reshaped)
        _time_ = time.time()
        ml_model_predict_result = ml_model.predict(ml_model_input)
        prediction_time_array.append(time.time() - _time_)
        if not np.where(ml_model_predict_result[0] == np.amax(ml_model_predict_result[0]))[0][0] == 1:
            no_mask_false_detection.append(i)
            continue
    except:
        continue

    ....
```

Рисунок 3.21 - Аналіз помилок передбачень та виміри швидкості моделі

Для покращення результатів моделі були створені допоміжні функції.

Власний коллбек клас CustomCallback, який є нащадком класу tf.keras.callbacks.Callback з TensorFlow. Цей клас використовується для кастомізації поведінки моделі під час тренування за допомогою кількох вбудованих методів, які викликаються на різних етапах тренування. [36]

- Метод __init__ - це конструктор класу. Він приймає різні параметри, такі як patience_stop_train, patience_decrease_lr, decrease_factor, monitor, use_custom_metric. Ці параметри використовуються для налаштування поведінки відбілювання ранньої зупинки та зменшення швидкості навчання.

- Метод on_train_begin викликається на початку тренування моделі. Він ініціалізує різні атрибути, які будуть використовуватися під час тренування.

- Метод on_epoch_end викликається після кожної епохи тренування. Він перевіряє, чи поточна точність перевищує найкращу зареєстровану точність. Якщо точність не покращується протягом певної кількості епох (patience_decrease_lr або patience_stop_train), він зменшує швидкість навчання

або зупиняє тренування.

- Метод lr_schedule використовується для зменшення швидкості навчання.

- Метод on_train_end викликається після закінчення тренування. Якщо тренування було зупинено рано, він виводить повідомлення.

Цей клас є дуже корисним для кастомізації процесу тренування моделі, оскільки дозволяє контролювати, як модель повинна реагувати на різні умови під час тренування.

```
class CustomCallback(tf.keras.callbacks.Callback):

    def __init__(self,
                 patience_stop_train=None,
                 patience_decrease_lr=None,
                 decrease_factor=None,
                 monitor=None,
                 use_custom_metric=None):

    def on_train_begin(self, logs=None):
    def on_epoch_end(self, epoch, logs=None):
    def lr_schedule(self, epoch, lr):
    def on_train_end(self, logs=None):
```

Рисунок 3.22 - Власний колбек

Функція `custom_acc`, яка вираховує власний варіант точності для прогнозів моделі (рис. 3.23). Ця точність враховує специфічні вимоги до прогнозування, а саме: найважливішими є три перші елементи вектора one-hot, а для останнього елемента має значення тільки те, чи він є одиницею; якщо так, то він додається до першого елемента.

1. Функція приймає два аргументи: `y_pred` та `y_true`, які відповідають прогнозованим і реальним значенням.

2. Далі код перетворює `y_pred` та `y_true` в one-hot представлення за допомогою функції `tf.one_hot`.

3. Функція `mask_prediction` визначена в середині `custom_acc` виконує спеціалізоване перетворення вектора one-hot, додаючи останній елемент до

першого, якщо він дорівнює одиниці.

4. Далі використовується `tf.equal` для порівняння перетворених прогнозованих і реальних значень.

5. Використовуючи `tf.reduce_all`, ми перевіряємо, чи усі елементи в кожному векторі дорівнюють один одному, і повертаємо булевий тензор.

6. Цей булевий тензор перетворюється в числовий за допомогою `tf.cast` із типом `tf.float32`.

7. На останньому кроці використовується `tf.reduce_mean` для обчислення середнього значення всіх елементів тензора, яке використовується як кінцеве значення точності.

```
def custom_acc(y_pred, y_true):

    y_pred_one_hot = tf.one_hot(tf.argmax(y_pred, axis=1), depth=4)
    y_true_one_hot = tf.one_hot(tf.argmax(y_true, axis=1), depth=4)

    def mask_prediction(y):
        bs = tf.shape(y)[0]

        tensor_left_3_el = tf.cast(tf.slice(y, [0, 0], [bs, 3]), tf.int32)
        tensor_right_2_el = tf.slice(y, [0, 3], [bs, 1])
        to_add = tf.concat([tf.cast(tensor_right_2_el, dtype=tf.int32), tf.zeros((bs, 2), dtype=tf.int32)],
            1)
        res = tf.add(tensor_left_3_el, to_add)
        return res

    Y_pred = mask_prediction(y_pred_one_hot)
    Y_true = mask_prediction(y_true_one_hot)
    equal_t = tf.equal(Y_pred, Y_true)
    reduce_t = tf.cast(tf.reduce_all(equal_t, axis=1), tf.float32)
    return tf.reduce_mean(reduce_t)
```

Рисунок 3.23 - Власна метрика точності

Ця функція використовується для обчислення спеціалізованої метрики точності, яка враховує конкретні вимоги до прогнозування. [33]

Функція `unfreeze_model_layers` розморожує останні `n_last_layers` шарів моделі, роблячи їх доступними для тренування (рис. 3.24).

1. Функція приймає два параметри: модель `model` і кількість останніх шарів `n_last_layers`, які потрібно "розморозити".

2. Якщо параметр `n_last_layers` не вказано, виникає помилка `ValueError`.

3. Далі виконується прохід по останніх `n_last_layers` шарах моделі

(починаючи з кінця) і властивість `trainable` цих шарів встановлюється як `True`. Це означає, що ці шари будуть тренуватися (їх ваги можуть оновлюватися) під час наступного циклу тренування моделі. Ця функція корисна, коли є ціль "розморозити" лише деякі шари в моделі, замість всієї моделі, особливо в ситуаціях, коли використовується передтреновану модель.

```
def unfreeze_model_layers(model, n_last_layers):

    if not n_last_layers:
        raise ValueError(f"unfreeze_model_layers : No 'n_last_layers' parameter defined")
    for layer in model.layers[len(model.layers)-n_last_layers:]:
        layer.trainable = True
```

Рисунок 3.24 - Розмороження останніх слоїв

Функція нижче відображає матрицю невідповідностей, яка є корисним інструментом для виявлення паттернів помилок класифікації моделі (рис. 3.25).

```
def plot_confusion_matrix(model,
                          data_generator,
                          classes_list,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    y_pred = np.asarray(model.predict(data_generator))

    y_pred = [np.where(num == np.max(num))[0] for num in y_pred]
    y_true = data_generator.classes
    count = 0
    for i in range(len(y_pred)):
        if y_pred[i] == y_true[i]:
            count += 1
    ...
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j,
                 i,
                 cm[i, j],
                 horizontalalignment="center",
                 color="black" if cm[i, j] > thresh else "black")

    plt.grid(False)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return plt
```

Рисунок 3.25 - Матриця невідповідностей

Ця функція корисна для аналізу помилок класифікації і допомогає покращити нашу модель, виявивши, в яких випадках вона найчастіше помиляється.

3.4 Оцінка та покращення розробленої моделі

Розпочнемо дослідження моделі з вхідною конфігурацією (додаток Ж, рисунок Ж.1) та проаналізуємо отримані результати. Бачимо, що середовище працює коректно та налаштування GPU активні. Логування ноутбука нам говорить про наступне:

1. "Steps number per training epoch: 95" означає, що під час кожної епохи тренування (проходу через весь набір даних) використовується 95 кроків. Кроки визначають, скільки разів модель навчатиметься на різних частках навчального набору даних.

2. "Steps number per validation epoch: 21" показує, що під час кожної епохи перевірки (проходу через весь набір даних для оцінки моделі) використовується 21 крок. Це дозволяє оцінити точність моделі на окремому наборі даних, який не був використаний під час тренування.

3. "Found 9152 images belonging to 3 classes" означає, що у загальному наборі даних для тренування знайдено 9152 зображення, які належать до 3 класів. Це означає, що модель буде навчатися розпізнавати ці 3 класи на основі цих зображень.

4. "Found 2020 images belonging to 3 classes" та "Found 750 images belonging to 3 classes" вказують на кількість зображень, знайдених у наборах даних для перевірки та тестування відповідно. Ці набори даних також містять 3 класи зображень.

```
Steps number per training epoch : 95
Steps number per validation epoch : 21
Found 9152 images belonging to 3 classes.
Found 2020 images belonging to 3 classes.
Found 2020 images belonging to 3 classes.
Found 750 images belonging to 3 classes.
Image data generators were created for training, validation, test set.
```

Рисунок 3.26 - Логування використанням генераторів даних зображень

Наступний блок виконання виводить нам інформацію про структуру моделі, яка дозволяє нам побачити назву слоїв, кількість нейронів на кожному слої.

1. "Total params: 229,878,983" означає, що загальна кількість параметрів у моделі становить 229 878 983. Параметри моделі включають ваги (коефіцієнти), зміщення та інші налаштування, які визначають її поведінку та здатність до розпізнавання.

2. "Trainable params: 226,672,007" показує, що кількість навчуваних параметрів у моделі становить 226 672 007. Це означає, що ці параметри будуть змінюватися під час тренування моделі для покращення її результатів.

3. "Non-trainable params: 3,206,976" вказує на кількість ненавчуваних параметрів у моделі, яка становить 3 206 976. Ці параметри, як правило, включаються в модель як фіксовані значення, наприклад, попередньо навчені ваги з іншої моделі або фіксовані фільтри зображень.

Загалом, кількість параметрів у моделі відображає складність та розмірність моделі, а також може вказувати на обсяг необхідних обчислень для тренування та використання моделі (додаток В, рис. В.1).

В наступному блоці ми бачимо, що модель закінчила своє навчання своєчасно, тобто не протренувавшись 150 повних епох, які були їй сконфігуровані у вигляді вхідних даних. Тобто, модель завершила своє тренування завчасно та зберегла найкращі свої ваги. Це спрацював наш кастомізований колбек, який припиняє тренування, якщо модель перестає покращуватись через n сконфігурованих епох.

Рисунок 3.27 - Візуалізація епох тренування моделі

Як зростала точність моделі під час тренування на тренувальній та валідаційний виборці ми можемо побачити на рисунку В.2, додаток В. Зміни значення функції втрати моделі під час тренування на тренувальній та

валідаційний виборці наведено у додатку В (рис. В.3). В наступному блоці нам повідомляється, що ми не використовуємо кастомну метрику, так як її не було сконфігуровано в вхідному файлі.

```
USE_CUSTOM_METRIC=False, graphics for custom accuracy will not be built.
```

Рисунок 3.28 - Відсутність кастомної метрики.

Далі відображаєм матрицю невідповідності, яка допомогає нам більш точно зрозуміти, як поводить себе модель та як саме вона помиляється.

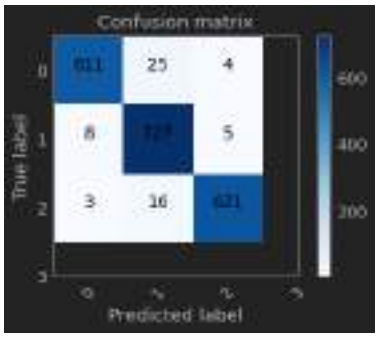


Рисунок 3.29 - Матриця невідповідності на валідаційний виборці.

Корегуємо слої моделі на встановлюємо, які слої потрібно перетреновувати, а які ні. Перетренеровуємо модель з скоректованими слоями - застосовуючи метод файн-тюнінгу (додаток В, рис. В.4). Файн-тюнінг моделі в машинному навчанні відноситься до процесу доналаштування попередньо навченої моделі, щоб підлаштувати її для конкретної задачі або даних. Візуалізація епох тренування моделі файн-тюнінгу наведна у додатку В на рисунку В.5. Іноді потрібно застосувати ці моделі до задач або даних, які вони не були навчені розпізнавати. Процес файн-тюнінгу включає наступні кроки:

1. Завантаження заздалегідь навченої моделі: Починаючи з попередньо навченої моделі, як правило, з великим набором параметрів, які захоплюють широкі знання з загальної задачі.

2. Замороження певних шарів: Зазвичай, ви заморожуєте попередньо

навчені шари моделі, щоб залишити їх в незмінному стані. Це дозволяє зберегти загальні репрезентативні можливості, які модель вивчила на попередній задачі.

3. Додавання нових шарів: додаються нові шари до моделі, які будуть відповідальні за вирішення конкретної задачі. Ці нові шари будуть навчатися під час процесу файн-тюнінгу.

4. Тренування моделі: Ви тренуєте модель на своїх даних, використовуючи комбінацію заморожених шарів та нових доданих шарів. Процес тренування може вимагати підлаштування гіперпараметрів та регуляризації для досягнення найкращих результатів.

5. Оцінка та налаштування: Після тренування оцінюється продуктивність моделі. Отримуємо нову матрицю невідповідності.

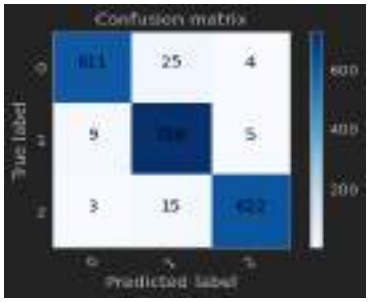


Рисунок 3.30 - Візуалізація матриці невідповідності після процесу файн-тюнінгу на валідаційний виборці

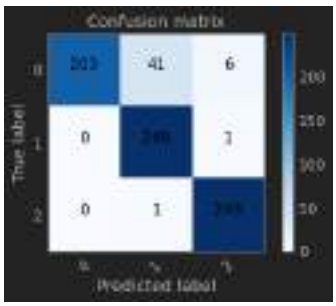


Рисунок 3.31 - Візуалізація матриці невідповідності після процесу файн-тюнінгу на тестовій виборці

Досліджуємо час евалюації моделі. Для цього ми берем нашу вже натреновану модель та заміряєм її швидкість на тестових даних.

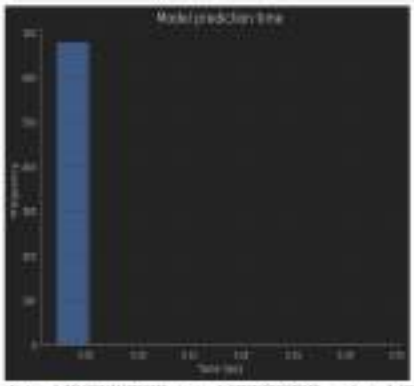


Рисунок 3.32 - Отримані результати швидкості моделі

Проаналізуємо дані, на яких помилилась модель.

Рисунок 3.33 - Результат виявлення моделі = `без маски`. Очікуваний результат = `з маскою`.

Рисунок 3.34 - Результат виявлення моделі = `з маскою`. Очікуваний результат = `без маски`.

Рисунок 3.35 - Результат визначення моделі – «людина з/без маски». Очікуваний результат = `особу не знайдено`.

Візуалізація виходу кожного шару моделі наведена на рисунку 3.38.

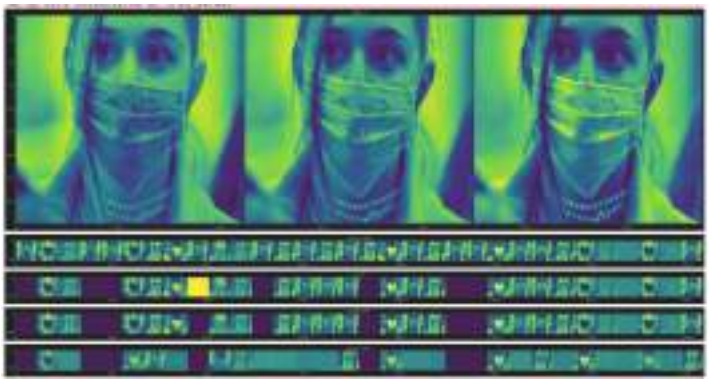


Рисунок 3.36 - Візуалізація виходу кожного шару моделі

Підсумки тренування моделі отримано за допомогою файлу outputs.json. Результати візуалізації скорів моделі наведено у додатку В (рис. В.6 та рис. В.7).

Побудуємо таблицю 3.1 з отриманих результатів тренування моделі.

Таблиця 3.1 - Візуалізація результатів моделі

Оцінки тренування	Етап 1	Етап 2
Час тренування. Етап 1	2706 с.	1616 с.
Оптимальна кількість епох	63	31
Точність на тренувальній виборці	98.2%	97.7%
Точність на валідаційний виборці	97.4%	97.4%
Точність на тестовій виборці	93%	
Швидкість роботи моделі	"min": 0.02142с. "max": 0.3373 с. "median": 0.0219 с. "mean": 0.0229 с.	

Таким чином**,** модель набрала досить хороші результати на тестовій виборці та показала чудову швидкість передбачення. Але, аналізуючи результати, ми бачимо, що модель робила велику кількість помилок на фотографіях подібних до рисунку 3.37.

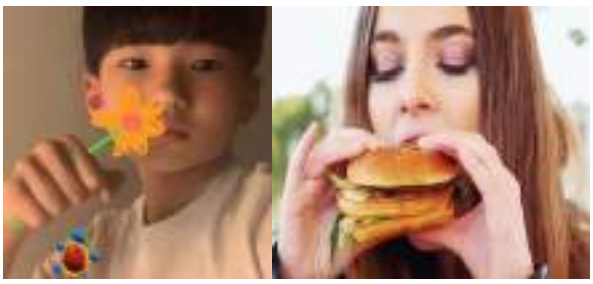


Рисунок 3.37 - Тип фотографій з частково закритим лицем іншим предметом

Розглянувши дані проблемні фотографії ми бачимо спільні риси:

1. Місце людини, на якому мала бути одягнута медична маска частково або повністю закрите іншим предметом, їжею або іншою частиною тіла.

2. Маска одягнена, але вона на закриває рівень губ. В даному випадку маска дійсно присутня на фото, але не одягнена.

Рисунок 3.38 - Тип фотографій з маскою, але не одягненою на рівень губ людини

Отже, спробуємо навчити модель адаптуватись до таких типів фотографій.

Для цього видозмінами вхідні дані датасету, а саме: навчимо нашу модель розпізнавати фотографії з підвищеною складністю, як окремий клас. Видозміна даних буде виглядати наступним чином, як зображено у додатку В, рисунку В.8.

Після цього розподіл анотованих даних буде вигляд діаграми у додатку В (рис. В.9). Тренувальна вибірка буде мати вигляд рисунку В.10 у додатку В.

Валідаційна вибірка буде мати вигляд датасету на рисунку В.11 (додаток В). В даному випадку буде використовуватись конфігурація додаток Д. Для даного розподілу даних, була введена нова кастомна метрика точності та новий

кастомний колбек, які були повністю описані в попередніх підрозділах.

Перейдемо до розгляду графіків тренування. Ми можемо побачити (рис. 3.39), як зростала точність моделі під час тренування на тренувальній та валідаційний вибірці.

Рисунок 3.39 - Графік зростання точності відносно епох на тренувальній вибірці

Далі ми можемо побачити, як спадало значення функції втрати моделі під час тренування на тренувальній та валідаційний вибірці (рис. 3.40).

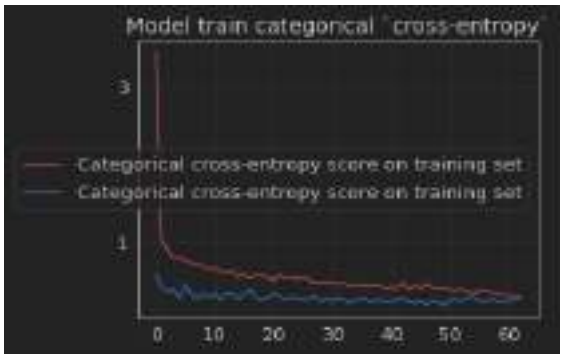


Рисунок 3.40 - Графік зростання точності відносно епох на тренувальній вибірці

Але важливо розуміти те, що тепер вирішальною в прийнятті рішень я кастомна метрика точності, а не звичайна. Тому в нас з’являється новий графік з кастомною метрикою точності (рис. 3.41).

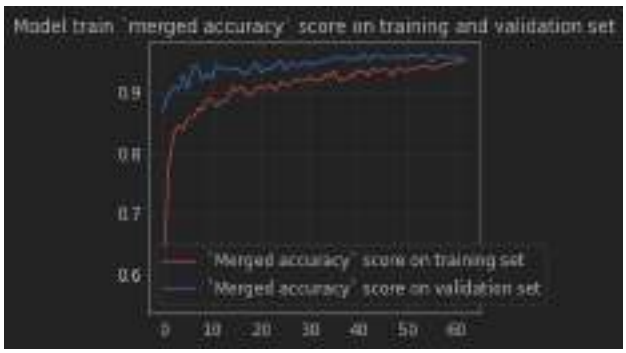


Рисунок 3.41 - Графік зростання точності відносно епох на тренувальній виборці з використанням кастомної метрики

Отримуємо нову матрицю невідповідності. Результати візуалізації матриці невідповідності після процесу файн-тюнінгу на валідаційний вибірці показано на рисунку 3.42. Результати візуалізації матриці невідповідності після процесу файн-тюнінгу на тестовій вибірці наведено на рисунку 3.43. Результати оцінки швидкості моделі наведено на рисунку 3.44.

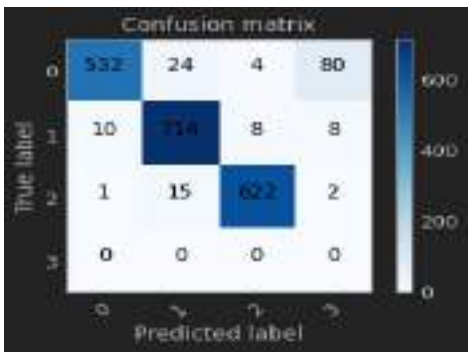


Рисунок 3.42 - Візуалізація матриці невідповідності після процесу файн-тюнінгу на валідаційний вибірці

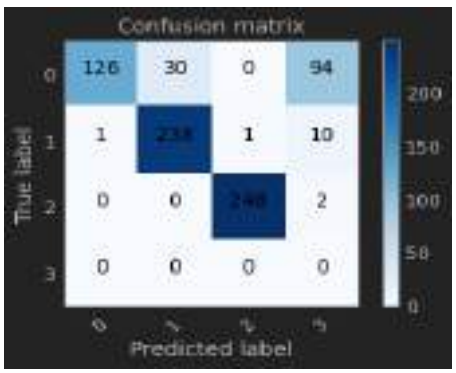


Рисунок 3.43 - Візуалізація матриці невідповідності після процесу файн-тюнінгу на тестовій вибірці

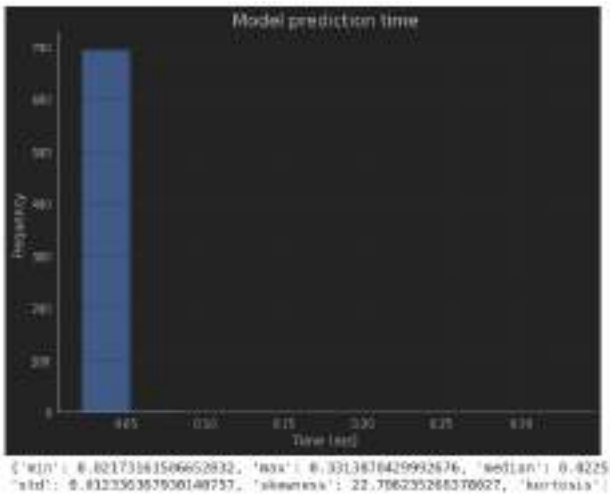


Рисунок 3.44 - Отримані результати швидкості моделі

Тепер модель не дуже добре сприймає типа фотографій, де людина закриває собі рукою лице поверх маски. Модель починає думати, що маски на лиці не існує. Дані, на яких помилилась модель наведено у додатку Д на рисунку Д.1. Результати візуалізації моделі наведено у додатку Д (рис. Д.2, рис. Д.3).

Підведемо підсумки тренування моделі використовуючи вихідний файл outputs.json. Побудуємо таблицю з отриманих результатів (табл. 3.2).

Таблиця 3.2 - Результати моделі

Характеристики оцінки	Етап 1	Етап 2
Час тренування. Етап 1	1866 с.	1673 с.
Оптимальна кількість епох	21	34
Точність на тренувальній виборці	95.2%	94.3%
Точність на валідаційний виборці	94.3%	96.6%
Точність на тестовій виборці	94.3%	
Швидкість роботи моделі	"min": 0.0217 с. "max": 0.3313 с. "median": 0.0225 с. "mean": 0.02402 с.	

Отже, як ми бачимо - модель з розщепленими класами показала кращий результат на 1.3%.

3.5. Проект парсеру сайтів

На сучасному етапі розвитку інтернет-технологій збір та обробка даних з веб-сайтів стає все більш актуальною задачею. Особливо це стосується зображень, які можуть бути використані для навчання моделей машинного навчання, візуалізації інформації або аналітики. Представлена схема ілюструє процес автоматичного витягнення, обробки та перевірки зображень з веб-сайтів. Цей процес включає в себе кілька ключових етапів, починаючи з надання користувачем конфігурації для парсеру і завершуючи перевіркою зображень за допомогою специфічної моделі. Давайте розглянемо цю схему детальніше.

Рисунок 3.45 - Схема роботи парсеру сайтів

1. Клієнт:

- Надає конфігураційний JSON файл для парсеру.

2. Встановлення конфігурації:

- Парсер отримує конфігураційний JSON файл від клієнта.

- На основі конфігурації відбувається налаштування параметрів для парсеру.

3. Витягнення URL зображень:

- За допомогою інструменту Selenium парсер переходить на веб-сайти (наприклад, Google) і витягує з них URL зображень на задану тему (у цьому випадку – "селфі в медичній масці").

4. Збереження URL в буфер:

- Усі зібрані URL зображень тимчасово зберігаються в буфері.

5. Формування унікального набору URL:

- Для уникнення дублікатів зображень, парсер створює унікальний набір URL з буфера.

6. Завантаження зображень:

- За допомогою зібраних URL парсер завантажує зображення.

7. Трансформація зображення:

- Після завантаження зображення може бути трансформовано (наприклад, змінено розмір, виправлено колір тощо).

8. Перевірка за допомогою моделі:

- Кожне завантажене та трансформоване зображення перевіряється за допомогою моделі машинного навчання.

9. Відсів зображень:

- Якщо модель визначає, що зображення не відповідає заданим критеріям (наприклад, на зображенні немає людини в масці), то таке зображення відхиляється.

- Якщо зображення відповідає критеріям, воно зберігається.

Ця схема описує послідовний процес витягування, обробки та перевірки зображень з веб-сайтів за допомогою парсера та моделі машинного навчання.

Проект, розроблений на мові Python включає в себе наступні складові: директорія parsing, яка розбита на наступні файли: main.py, photo_utils.py, scrapping.py, parser_config.py. Структура проекту зображена в додатку Ж.

Директорія `parsing` - містить в собі весь написаний програмний код на мові програмування Python, тобто дана директорія містить фактично всю реалізацію практичної частини парсингу у вигляді програмного коду. Ця директорія містить також в собі підмножину папок:

Директорія `parsing` має структуру зображену у додатку Ж (рис. Ж.2).

- main.py - є вхідним файлом для запуску парсера. Налаштування парсера та вхідні параметри зчитуються з конфігураційного файлу. Під час запуску файлу створюються задані конфігурацією скрапери, які в свою чергу виконують збір фотографій з заданого джерела сайту.

- photo_utils.py - містить утиліти для роботи з фотографіями, наприклад, зміною розміру фотографій.

- scrapping.py - містить класи та функції, які відповідають власне за самий

скрапінг файлів з публічних джерел. Кожен клас відповідає відповідному скраперу джерела. Цей файл містить в собі повністю всі методи та команди автоматизації процесу витягу фотографій з сайтів.

- parser_config.py - містить конфігурацію парсеру, а саме : цільова тематика скрапінгу, шляхи збереження даних, дані моделі, цільові сайти для витягу інформації, кількість екземплярів автоматичного скрапінгу .

Розберемо в деталях базовий клас скраперів написаний в файлі scraping.py наведену у додатку Ж.

Клас `BaseScraper` - це базовий клас для скрапінгу (збору даних) зображень з інтернету.

- Властивості класу:

- `driver`: Веб-драйвер, що використовується для автоматизованого керування браузером.

- `folder_path`: Шлях до папки на диску, де будуть зберігатися завантажені зображення.

- `what_to_scrap`: Теги для пошуку зображень (наприклад, `cat`, `human`, `car`).

- `max_fetches`: Максимальна кількість зображень, які можна завантажити.

- `URLSset`: Множина URL-адрес зображень для завантаження.

- `model`: Модель машинного навчання, яка може використовуватися для фільтрації зображень під час завантаження.

- Методи класу:

1. `__init__(self, what_to_scrap, max_fetches, folder_path, model=None)`: Конструктор класу, який ініціалізує об’єкт з заданими параметрами.

2. `get_URLS_len(self)`: Повертає кількість URL-адрес зображень у множині `URLSset`, які готові до завантаження.

3. `run(self)`: Запускає процес скрапінгу. Цей метод викликає інший метод `extract_images_url()`, який повинен бути перевизначений у підкласах.

4. `extract_images_url(self)`: Цей метод має бути перевизначений у

підкласах `BaseScraper`. Його мета - витягнути URL-адреси зображень для подальшого завантаження.

5. `download_batch_images(self, urls_tmp)`: Завантажує пакет зображень, що не були завантажені раніше, і додає їх до множини `URLSset`.

6. `download_image(self, URL)`: Завантажує та зберігає зображення за вказаною URL-адресою. Якщо задана модель машинного навчання (`model`), цей метод також може використовувати її для фільтрації зображень перед їх зберіганням.

Клас `BaseScraper` розроблений для створення базової структури для скрапінгу зображень з інтернету. Підкласи цього класу мають перевизначити метод `extract_images_url()` для витягування URL-адрес зображень з різних джерел. Також розглянемо один з класів, який відповідає за скрапинг ресурсів з заданого ресурсу. Клас `ScraperGoogle` є підкласом `BaseScraper` та призначений для збору зображень з Google Images. Ось детальний опис цього класу:

- Властивості класу:

- `URL`: URL-адреса пошукового запиту на Google Images, створена на основі тегу `what_to_scrap`.

- Методи класу:

1. `__init__(self, what_to_scrap, max_fetches, folder_path, model=None)`: Конструктор класу, який ініціалізує об’єкт за допомогою параметрів і визначає URL для пошуку на Google Images.

2. `run(self)`: Запускає процес скрапінгу зображень з Google Images. Викликає метод `extract_images_url()` для отримання URL-адрес зображень.

3. `extract_images_url(self)`: Отримує URL-адреси зображень з Google Images. Метод виконує прокрутку сторінки і викликає інші методи для збору URL-адрес зображень.

4. `scroll_and_collect_urls(self, start)`: Прокручує сторінку вниз і збирає URL-адреси зображень на поточній сторінці. Цей метод також намагається натиснути на зображення та збирати URL-адреси відкритих зображень.

5. `try_click_image(self, image)`: Спроба клацнути по переданому

зображенню. Якщо клацання не вдається (через помилки елементу або інші проблеми), то нічого не відбувається.

6. `collect_current_image_urls(self)`: Збирає URL-адреси зображень, які відображаються на поточній сторінці браузера.

7. `try_load_more(self)`: Спроба натиснути кнопку "Завантажити більше" на Google Images для отримання більшої кількості зображень. Якщо кнопка не знайдена або виникає будь-яка інша помилка, то цей крок просто ігнорується.

Загалом, клас `ScraperGoogle` є специфічним для Google Images реалізацією базового класу `BaseScraper`. Він використовує веб-драйвер для взаємодії з Google Images, прокручує сторінку, збирає URL-адреси зображень, а потім завантажує ці зображення на локальний диск.

Тепер перейдемо до розгляду структури конфігураційного файлу - `configuration.json` (додаток Ж, рис. Ж.3).

- Ключ "SUBJECT_TO_SCRAP" - вказує на тему для скрапінгу, наприклад, "selfie in medicine mask" (селфі в медичній масці).

- Ключ "SAVE_PHOTOS_BASE_DIR_PATH" - містить основний шлях до директорії, де будуть зберігатися фотографії, які були отримані в процесі скрапінгу.

- Ключ "SAVE_PHOTOS_DIR_NAME" - вказує назву папки, де конкретно будуть зберігатися зіскраплені фотографії з теми "selfie in medicine mask".

- Ключ "MODEL_LOCALE" - містить інформацію про модель машинного навчання:

- "MODEL_WEIGHTS" - містить шлях до файлу з вагами моделі.

- "MODEL_PATH" - містить шлях до файлу моделі.

- Ключ "ENABLE_MODEL" визначає, чи буде використовуватися модель машинного навчання під час процесу скрапінгу.

- Ключ "TARGET_SITES" - містить список веб-сайтів, з яких буде проводитися скрапінг. У цьому випадку це "Google", "Pixabay" та "Pexels".

- Ключ "INSTANCES_NUMBER" - вказує на кількість екземплярів або потоків, які будуть використовуватися для скрапінгу. У цьому випадку

використовується 1 екземпляр.

Під час розробки парсеру було розібрано та розроблено підходи до парсингу фотографій з наступних ресурсів : Google, Pixabay, Pexels. Розглянемо механізм роботи Selenium скрипта з Google сайтом. Він поділяється на такі основні етапи (рисунок 3.46) :

1. Перехід на сторінку перегляду Images сайту Google - з відповідним запитом картинок.

2. Клікаємо на першу картину. Реакцією на клік є збільшене відображення картинки. За допомогою цього, ми можемо отримати url на зображення заданої картинки в форматі “.jpg” або “.png”.

3. Дані посилання з 2 кроку зберігаються в буфері даних. І так ми проклікуєм фотографія за фотографією, допоки ми не дійдем до кінця сторінки - про що нам засигналізує кнопка “Show more results”. Особливістю, скрапінгу з Google картинок є в тому, що ми одразу не можемо отримати всі URLs посилання для завантаження картинок. Це завантажувальне посилання отримується шляхом проклікування картинки.

4. Дійшовши до кінця сторінки, ми активуємо кнопку “Show more results”, яка в свою чергу покаже нам подальший список картинок.

5. І так процес проходить по кругу поки вміст картинок не вичерпається.

Рисунок 3.46 - Скрапінг фотографій з Google

Розглянемо механізм роботи Selenium скрипта з Pixabay сайтом (додаток Ж, рис. Ж.4). Він поділяється на такі основні етапи:

1. Перехід по адресі : https://pixabay.com/en/images/search/%7Bwhat

_to_scrap%7D

2. Нам відкривається сторінка з параметром page=1, тобто перша сторінка для перегляду. Зручність сайту Pixabay полягає в тому, що можливості сайта дають нам можливість відразу зберігати декілька URL посилань на фотографію в форматі “.png” або “.jpeg”, не проклікуючи кожну картинку окремо, як в Google.

3. Поступово ми гортаєм вниз, що відкриває нам нові елементи фотографій доступні нам для завантаження. Фотографії завантажуються одрузу батчем. В поле завантаження попадають ті фотографії, які зарендирелись на даній сторінці.

4. Дійшовши до кінця однієї з сторінок, ми виконуємо запит на нову сторінку індексуючи параметр page на одиницю вгору.

5. Ітеративно здійснюєм такі операції, до тих пір, поки всі сторінки не будуть пройдені та всі фотографії не будуть завантажені.

Початок скрапінгу фотографій з джерела Pixabay, в цей момент скрапер збирає фотографії в свій буфер обміну, зображено на рисунку Ж.5 (додаток Ж). На лівій частині екрану ми бачимо, що парсер пройшов вже крізь деяку частину фотографій. На правій частині відображено зміст цільової папки збережених фотографій - вона поки що пуста, тому що фотографії, які зберезені в буфері обміну, ще не потрапили на файлову систему та не пройшли передбачення моделі. Даємо парсеру працювати декілька секунд та одразу ж бачимо, як в цільовій папці з’явились нові фотографії (додаток Ж, рис. Ж.6).

В сучасній добі високих технологій, прискорення обробки даних є однією з ключових проблем, з якою стикаються розробники. Особливо це актуально для завдань, що вимагають обробки великих масивів даних, таких як парсинг веб-сайтів. Використання мультипроцессингу може стати ефективним рішенням цієї проблеми, дозволяючи оптимізувати процес збору даних за рахунок паралельної роботи декількох процесів одночасно. У цьому розділі ми поглиблено розглянемо принципи реалізації мультипроцессингу при парсингу, його переваги та можливі труднощі, що можуть виникнути під час впровадження цього

підходу.

Мультипроцесинг - це техніка, яка дозволяє виконувати декілька процесів одночасно в основному завдяки використанню декількох процесорів або ядер процесора (рис. Ж.7, додаток Ж). Код main.py файлу парсеру починається з визначення декількох функцій, які допомагають у реалізації мультипроцесингу:

1. create_model(model_path, model_weights): Ця функція використовується для завантаження моделі машинного навчання з вказаного шляху та вагами. Вона повертає об’єкт моделі.

2. initialize_scrapers(...): Функція ініціалізує список скраперів. За допомогою циклу вона створює кілька об’єктів скраперів на основі заданих конфігурацій. Ці скрапери відповідають різним сайтам, з яких будуть збиратися дані.

3. run(scraper_obj): Проста обгортка для методу `run()` конкретного скрапера. Виконує задачу скрапінгу.

4. start_scraping(scraper_objs): Основна функція для реалізації мультипроцесингу. Вона створює та запускає декілька процесів на основі переданих їй об’єктів скраперів.

У функції `start_scraping(scraper_objs)` для кожного об’єкта скрапера створюється окремий процес за допомогою класу `Process`. Кожен процес запускається з функцією `run(scraper)`, яка виконує метод `run()` конкретного скрапера.

Після створення всіх процесів, вони запускаються одночасно. Функція `process.start()` запускає кожен процес, а `process.join()` забезпечує, що основний потік програми чекатиме, поки всі процеси завершаться.

Таким чином, завдяки мультипроцесингу, цей код може одночасно запускати декілька скраперів, що значно прискорює процес збору даних.

ВИСНОВКИ ДО РОЗДІЛУ 3

Модель нейронної мережі реалізована в проекті, складовими елементами якого є: операційний блок функціонування нейронної мережі; блок файлів конфігурації для різних типів моделей; блок вихідних результативних файлів, які отримані в ході тренування та оцінки моделі. Проекти нейронної мережі та парсеру мають власний дизайн та структуру.

На основі проекту нейронної мережі розроблено алгоритм та програмний код, який імітує її функціонування. З метою дослідження моделей нейронних мереж було розроблено концепцію FlowModeler, яка реалізує підхід до багаторазового експериментального оцінювання з різними характеристиками.

Процес навчання та дослідження моделі відбувається на побудованих класах зображень розподілених на тренувальний, валідаційний та тестовий набори даних, в межах яких працюють ітератори, які отримуються внаслідок роботи генераторів даних шляхом пакетного завантаження та обробки зображень з диску. Для покращення моделі розроблено ряд допоміжних функцій, які призначені для деталізації процесів оцінки та аналізу результатів моделі.

З метою отримання даних розроблено та реалізовано програмно автоматизований парсер для взаємодії з різними веб-сайтами, що враховує здатність адаптуватись та виконувати завдання з визначення наявності медичної маски на обличчі людини. Під час розробки парсеру було сформовано підходи до парсингу фотографій з наступних ресурсів: Google, Pixabay, Pexels.

ВИСНОВКИ

В результаті дослідження було розроблено та практично реалізовано автоматизовану систему парсингу фотографій отриманих з публічно доступних джерел із застосуванням нейронних мереж, основною функціональною можливістю якої є реалізація формування датасету за категоризацією ознаки («маска на обличчі») з покращеними елементами усунення похибок розпізнавання. На основі отриманих результатів можна зробити наступні висновки:

1. На основі проведеного аналізу та систематизації теоретичних положень щодо створення та функціонування нейронних мереж, розроблено концептуальну модель та її програмну реалізацію, основним функціональним призначенням якої є розпізнавання наявності медичної маски на обличчі людини;
2. Визначено основні класифікаційні ознаки відбору даних при формуванні датасету, що мають отримуються в результаті парсингу і можуть містити помилки візуалізації;
3. Вдосконалено алгоритмічні та програмні підходи щодо реалізації процесу парсингу фотографій покращенням вхідного датасету через деталізацію початкових класифікаційних ознак за результатами глибокого машинного навчання.
4. Проведено алгоритмізацію процесу парсингу, основною метою якого є формування датасету вхідних даних отриманих з публічно доступних джерел, що дозволило сформувати початкову вибірку даних для подальшого формування підвибірок даних: тренувальної, валідаційної та тестової.
5. Розроблено модель нейронної мережі, яка має здатність до покращення результатів оцінки за допомогою тренування шляхом машинного навчання з учителем;
6. Розроблено та програмно реалізовано додаток парсингу фотографій, який отримує дані у вигляді зображень людей з медичними масками з публічних

джерел: Google, Pixabay, Pexels. Прискорення обробки даних досягнуто шляхом запровадження механізму мультипроцессингу, направленого на одночасне паралельне виконання декількох процесів при використанні декількох процесорів з метою оптимізації процесу збору даних.

7. Проведено тренування моделі та отримано результати оцінювання на основі метрик: мінімальної втрати, максимальної користувацької точності для тренувального та валідаційного наборів даних, часу тренування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Ефект зсуву в нейронній мережі. URL: https://www.geeksforgeeks.org/effect-of-bias-in-neural-network/

2. Поняття нейромережі. URL:: https://termin.in.ua/neyromerezha/

3. Одношарові та багатошарові штучні нейронні мережі. URL: https://studfiles.net/preview/3170522/page:3/

4. Нейронні мережі від теорії до практики. URL: https://www.mql5.com/ru/articles/497

5. Нейронні мережі та глибоке навчання. URL: https://codeguida.com/post/739

6. Нейромережі лекція. URL: https://learn.ztu.edu.ua/pluginfile.php/176771/mod_resource/content/1/ШІ_КБ_Л-3_Нейромережи.pdf

7. Завдання, які вирішують за допомогою нейромоделювання URL: https://studfile.net/preview/5740125/page:6/

8. The mostly complete chart of Neural Networks, explained. URL: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

9. Мережа радіально базисних функцій. URL: https://uk.wikipedia.org/wiki/Мережа_радіальних_базисних_функцій

10. Нейронна мережа LSTM. URL: https://neurohive.io/ru/osnovy-data-science/lstm-nejronnaja-set/

11. Мова програмування Python. URL: https://uk.wikipedia.org/wiki/Python

12. About Python. URL: https://www.python.orgabout//

13. Top 15 Python Libraries for Data Science and Machine Learning. URL: https://www.projectpro.io/article/top-5-libraries-for-data-science-in-python/196

14. Top 30 Python Libraries for Data Science in 2023 URL: https://www.knowledgehut.com/blog/data-science/python-libraries-for-data-science

15. Top 20 Python Libraries for Data Science for 2023. URL:

https://www.simplilearn.com/top-python-libraries-for-data-science-article

16. Advantages and Disadvantages of TensorFlow. URL: https://techvidvan.com/tutorials/pros-and-cons-of-tensorflow/

17. Перші кроки в NLP: розглядаємо Python-бібліотеку TensorFlow та нейронні мережі в реальному завданні. URL: https://dou.ua/lenta/articles/first-steps-in-nlp-tensorflow/

18. Practical statistics for Data Scientists: П. Брюс, Э. Брюс. O'Really Media Inc., 1005 Gravenstein Highway North, 2018. 304 с.

19. Джеймс Г., Уиттон Д., Хасти Т., Тибширани Р. An Introduction to Statistical Learning: with Applications in R. Springer New York Heidelberg Dordrecht London, 2017.

20. What is web-scraping and how to use it. URL: https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/

21. Парсинг та обробка сайтів з використанням технології Web-scraping. URL: https://conf.ztu.edu.ua/wp-content/uploads/2019/06/29.pdf

22. Web scraping protection. URL: https://www.imperva.com/learn/application-security/web-scraping-attack/

23. What Are The Different Types Of Web Scraping Approach. URL: https://www.linkedin.com/pulse/what-different-types-web-scraping-approaches-rajat-thakur/

24. Парсинг сайтів. URL: https://allrival.com/post/articles/parsing-saytov-konkurentov-20-samyh-luchshih-servisov

25. Selenium WebDriver та для чого він потрібен. URL: https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/

26. Selenium webdriver як інструмент для автоматизованого тестування. URL: https://www.browserstack.com/guide/selenium-webdriver-tutorial

27. Selenium components. URL: https://www.selenium.dev/documentation/overview/components/

28. The Elements of Statistical Learning: Data Mining and Prediction: Роберт Тібшірані, Єрон Фрідман, Springer New York Heidelberg Dordrecht London, 2009.

745 с.

29. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython: Вес МакКінні, O'Really Media Inc., 1005 Gravenstein Highway North, 2017. 544 с.

30. Фостер Превост, Том Фавателла, Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. Springer New York Heidelberg Dordrecht London, 2013. 414 с.

31. Practical statistics for Data Scientists: П. Брюс, Є. Брюс., O'Really Media Inc., 1005 Gravenstein Highway North, 2018. 304 с.

32. Сайт програмного забезпечення Python. URL: https://www.python.org/

33. Сайт програмного забезпечення Selenium. URL: https://www.selenium.dev/

34. Сайт програмного забезпечення PyCharm. URL: https://www.jetbrains.com/pycharm/

35. Сайт програмного забезпечення Docker. URL: https://www.docker.com/

36. Сайт Google. URL: https://www.google.com/

37. Сайт Pexels. URL: https://www.pexels.com/

38. Сайт Pixabay. URL: https://pixabay.com/

39. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. URL: https://www.sciencedirect.com/science/article/pii/S1877050918308019#:~:text=Convolutional%20Neural%20Network%20,about%20various%20aspects%20of%20CNN

40. Convolutional neural networks (CNNs): concepts and applications in pharmacogenomics. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8342355/#:~:text=Convolutional%20neural%20networks%20,faced%20by%20other%20computational%20methods

41. An Introduction to Convolutional Neural Networks. URL: https://arxiv.org/abs/1511.08458#:~:text=One%20of%20the%20most%20impressive,

of%20getting%20started%20with%20ANNs

42. Deep Learning: Basics and Convolutional Neural Networks (CNNs). URL: https://www.ncbi.nlm.nih.gov/books/NBK597497/#:~:text=Chapter%203%20Deep%20Learning%3A%20Basics,Published%20online%3A%20July%2023%2C%202023

43. A primer on deep learning and convolutional neural networks for clinicians. URL: https://insightsimaging.springeropen.com/articles/10.1186/s13244-021-01052-z

44. Deep Learning: Basics and Convolutional Neural Networks (CNNs). URL: https://www.ncbi.nlm.nih.gov/books/NBK597497/#:~:text=Published%20online%3A%20July%2023%2C%202023,groups%20depending%20on%20their%20components

45. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. URL: https://www.mdpi.com/2079-3197/11/3/52#:~:text=Introduction%20There%20has%20been%20a,4%2C%205

46. Recent advances and applications of deep learning methods in materials science. URL: https://www.nature.com/articles/s41524-022-00734-6#:~:text=Convolutional%20neural%20networks%20,invariant%20image%20representations.%20There

47. A Comprehensive Overview and Comparative Analysis on Deep. URL: https://www.nature.com/articles/s41524-022-07346#:~:text=Convolutional%20neural%20networks%20,invariant%20image%20representations.%20There

48. A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU. URL: https://arxiv.org/pdf/2305.17473.pdf#:~:text=2,11

49. Frontiers | Dual-Channel Deep Graph Convolutional Neural Networks. URL: https://www.frontiersin.org/articles/10.3389/fncom.2023.1290491#:~:text=URL%3A%20https%3A%2F%2Fwww

50. Tensorflow. URL: https://www.tensorflow.org/

51. Keras. URL: https://keras.io/

ДОДАТКИ

Додаток А

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Рисунок А.1 - Архітектура MobileNet

Продовження додатку А

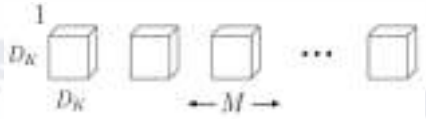


Рисунок А.2 - Регулярні конволюційні фільтри

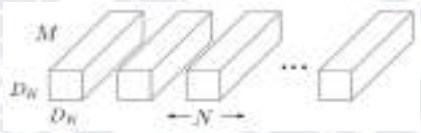


Рисунок А.3 - Глибокі фільтри

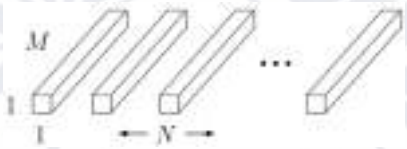


Рисунок А.4 - 1 × 1 Згорткові фільтри, які є точковою згорткою в контексті роздільної згортки по глибині

Продовження додатку А

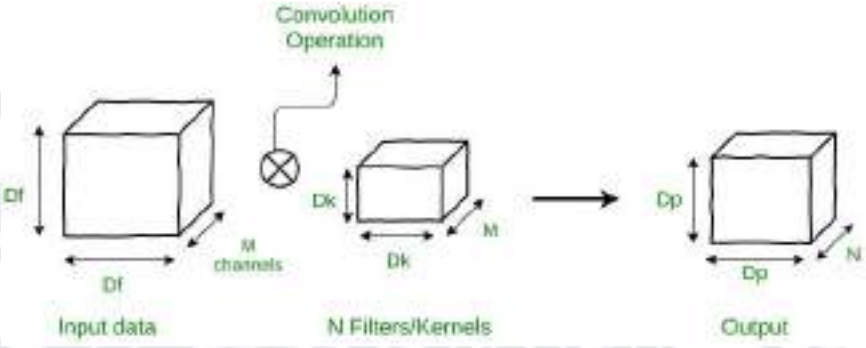


Рисунок А.5 - Схема конволюційної операції

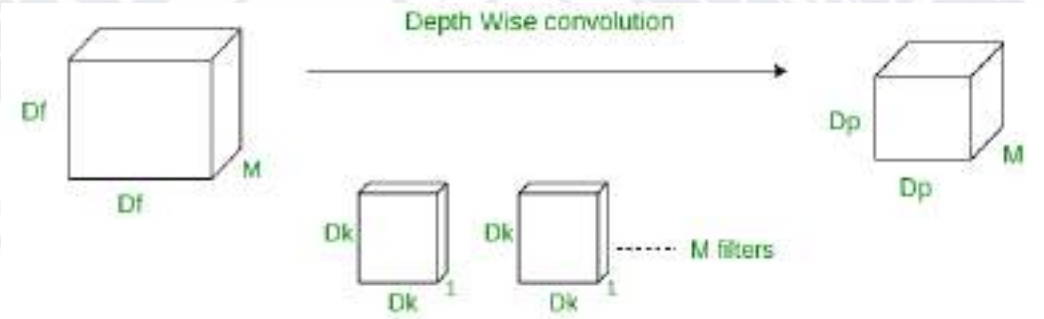


Рисунок А.6 - Глибока згортка

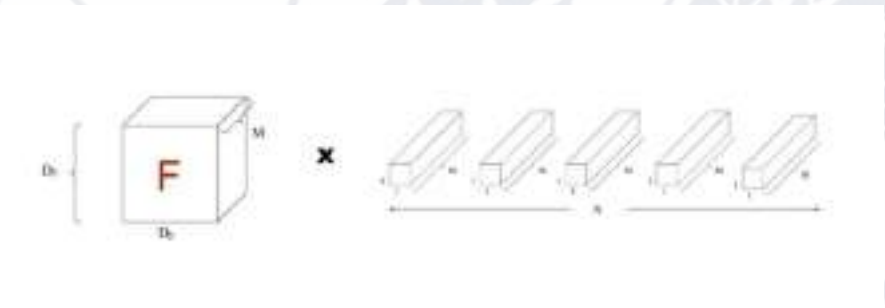


Рисунок А.7 - Поточкова згортка

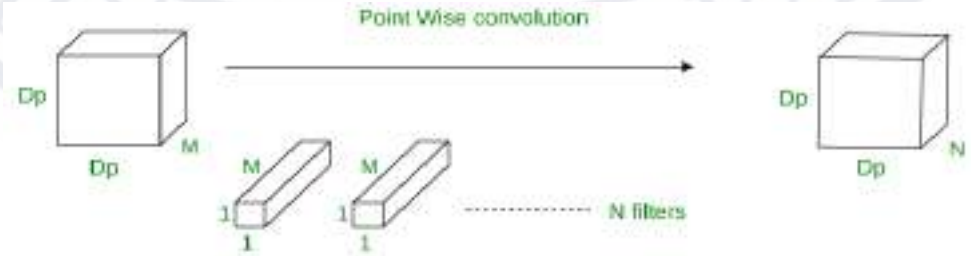
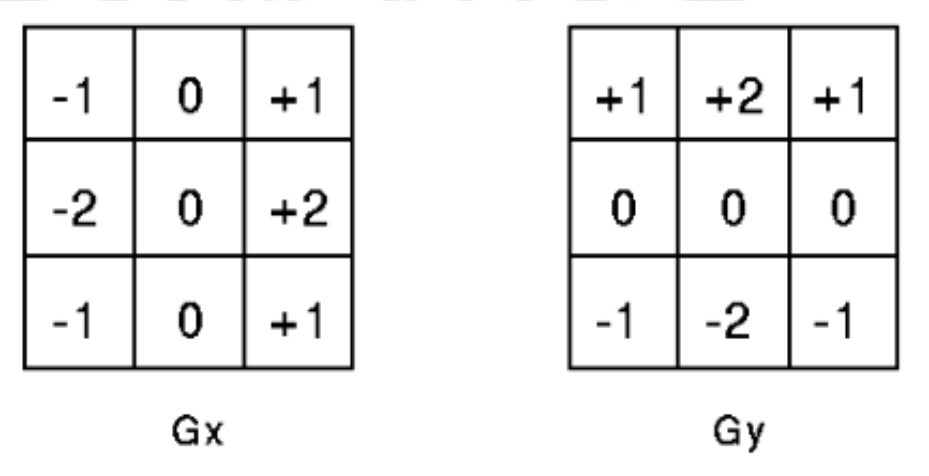


Рисунок А.8 - Вартість обчислення поточкової згортки

Продовження додатку А



-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Рисунок А.9 - Фільтр Sobel. Gx для вертикального краю, Gy для визначення горизонтального краю

Додаток Б

```python
def plot_distribution_hist(values):
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import skew, kurtosis
    fig, ax = plt.subplots(figsize=(8, 6))
    counts, bins, _ = ax.hist(values, edgecolor='black', color='#4C72B0', alpha=0.7)
    ax.set_title("Model prediction time", fontsize=16)
    ax.set_xlabel("Time (ms)", fontsize=12)
    ax.set_ylabel("Frequency", fontsize=12)
    ax.grid(True, linestyle='--', alpha=0.5)
    ax.tick_params(axis='both', which='major', labelsize=10)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    plt.tight_layout()
    plt.show()
    value_min = np.min(values)
    value_max = np.max(values)
    value_median = np.median(values)
    value_mean = np.mean(values)
    value_std = np.std(values)
    value_skewness = skew(values)
    value_kurtosis = kurtosis(values)
    return {
        "min" : value_min,
        "max" : value_max,
        "median" : value_median,
        "mean" : value_mean,
        "std" : value_std,
        "skewness" : value_skewness,
        "kurtosis" : value_kurtosis
    }
```

Рисунок Б.1 - Plot_distribution_hist

Продовження додатку Б

output.json


```
{
    "id": "MobileNet_4cl",
    "model_scores": [
        "train_model_classification_layers_stage_best_scores": {
            "train": {
                "min_loss": null,
                "max_accuracy": null,
                "max_custom_accuracy": null
            },
            "validation": {..},
            "optimal_epoch_number": null,
            "model_train_time": null
        },
        "train_model_fine_tune_stage_best_scores": {
            "train": {
                "min_loss": null,
                "max_accuracy": null,
                "max_custom_accuracy": null
            },
            "validation": {..},
            "optimal_epoch_number": null,
            "model_train_time": null
        }
    },
    "test_score": {
        "loss": null,
        "accuracy": null,
        "custom_acc": null
    },
    "model_prediction_measurement": {
        "experiments_prediction_time_results": [],
        "input_image_shape": [],
        "unit_of_measurement": "ms",
        "time_results_hist_description": {
            "min": null,
            "max": null,
            "median": null,
            "mean": null,
            "std": null,
            "skewness": null,
            "kurtosis": null
        }
    }
}
```


Рисунок Б.2 - output.json

Продовження додатку Б

configuration.json

```
{
  "CATEGORIES_NUMBER": null,
  "MODEL": {
    "name": null,
    "fine_tune_layers": null,
    "class_weights": null,
    "loss": null,
    "optimizer_base": null,
    "optimizer_fine_tune": null,
    "callbacks": null,
    "layers_to_unfreeze_fine_tune": null,
    "epoch_number_appended_layers": null,
    "epoch_number_fine_tune": null
                ...
  },
  "DATA_AUGMENTATION_TRAIN": {
    "samplewise_std_normalization": null,
    "horizontal_flip": null,
    "brightness_range": null,
    "rotation_range": null,
    "zoom_range": null,
    "fill_mode": null,
    "width_shift_range": null,
    "height_shift_range": null,
    "shear_range": null,
    "channel_shift_range": null
                ...
  },
  "ID": null,
  "OUTPUT_FOLDER_PATH": null
}
```

Рисунок Б.3 - Configuration.json

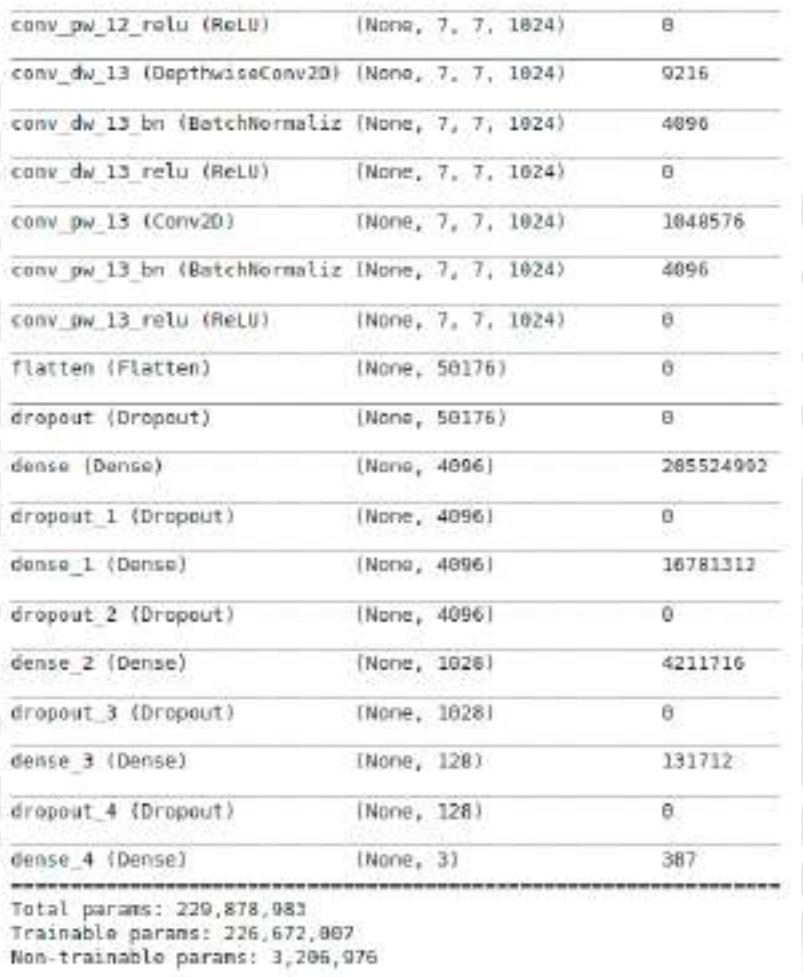
Продовження додатку Б

output.json


```
{
    "id": "MobileNet_4cl",
    "model_scores": [
        "train_model_classification_layers_stage_best_scores": {
            "train": {
                "min_loss": null,
                "max_accuracy": null,
                "max_custom_accuracy": null
            },
            "validation": {..},
            "optimal_epoch_number": null,
            "model_train_time": null
        },
        "train_model_fine_tune_stage_best_scores": {
            "train": {
                "min_loss": null,
                "max_accuracy": null,
                "max_custom_accuracy": null
            },
            "validation": {..},
            "optimal_epoch_number": null,
            "model_train_time": null
        }
    },
    "test_score": {
        "loss": null,
        "accuracy": null,
        "custom_acc": null
    },
    "model_prediction_measurement": {
        "experiments_prediction_time_results": [],
        "input_image_shape": [],
        "unit_of_measurement": "ms",
        "time_results_hist_description": {
            "min": null,
            "max": null,
            "median": null,
            "mean": null,
            "std": null,
            "skewness": null,
            "kurtosis": null
        }
    }
}
```


Рисунок Б.4 - Output.json

Додаток В



conv_pw_12_relu (ReLU)	(None, 7, 7, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	9216
conv_dw_13_bn (BatchNormaliz	(None, 7, 7, 1024)	4096
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)	0
conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	1048576
conv_pw_13_bn (BatchNormaliz	(None, 7, 7, 1024)	4096
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
flatten (Flatten)	(None, 50176)	0
dropout (Dropout)	(None, 50176)	0
dense (Dense)	(None, 4096)	205524992
dropout_1 (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 1028)	4211716
dropout_3 (Dropout)	(None, 1028)	0
dense_3 (Dense)	(None, 128)	131712
dropout_4 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 3)	387

Total params: 229,878,983
Trainable params: 226,672,807
Non-trainable params: 3,206,976

Рисунок В.1 - Опис моделі

Продовження додатку В

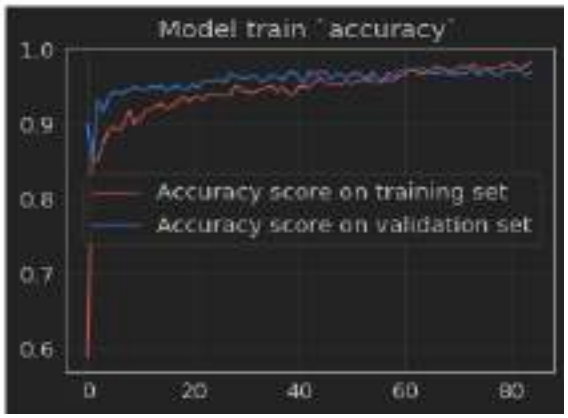


Рисунок В.2 - Графік зростання точності відносно епох на тренувальній виборці

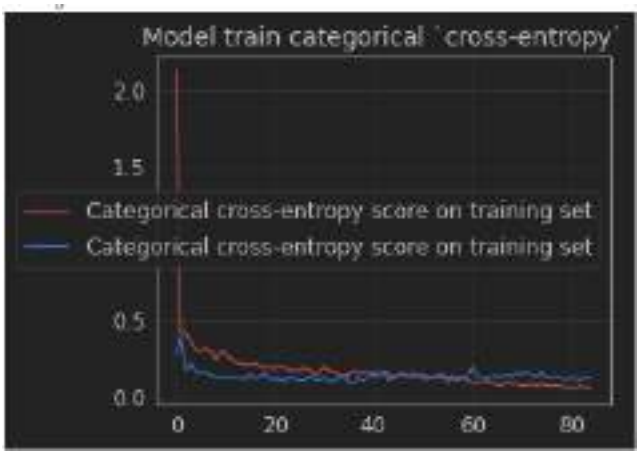


Рисунок В.3 - Графік зростання точності відносно епох на тренувальній вибірці

Продовження додатку В

```
63-conv_dw_10_bn-True
64-conv_dw_10_relu-False
65-conv_pw_10-False
66-conv_pw_10_bn-True
67-conv_pw_10_relu-False
68-conv_dw_11-False
69-conv_dw_11_bn-True
70-conv_dw_11_relu-False
71-conv_pw_11-False
72-conv_pw_11_bn-True
73-conv_pw_11_relu-False
74-conv_pad_12-True
75-conv_dw_12-True
76-conv_dw_12_bn-True
77-conv_dw_12_relu-True
78-conv_pw_12-True
79-conv_pw_12_bn-True
80-conv_pw_12_relu-True
81-conv_dw_13-True
82-conv_dw_13_bn-True
83-conv_dw_13_relu-True
84-conv_pw_13-True
85-conv_pw_13_bn-True
86-conv_pw_13_relu-True
87-flatten-True
88-dropout-True
89-dense-True
```

Рисунок В.4 - Відображення статусу слоїв моделі

```
accuracy: 0.9717
Epoch 47/100
95/95 [==============================] - 30s 314ms/step - loss: 0.0787 - accuracy: 0.9753 - val_loss: 0.1243 - val_
accuracy: 0.9732
Epoch 48/100
94/95 [============================>.] - ETA: 0s - loss: 0.0935 - accuracy: 0.9671  Reduced lr --> 0.00000031833760
80.00
95/95 [==============================] - 30s 315ms/step - loss: 0.0934 - accuracy: 0.9671 - val_loss: 0.1255 - val_
accuracy: 0.9717
Epoch 49/100
95/95 [==============================] - 30s 313ms/step - loss: 0.0624 - accuracy: 0.9780 - val_loss: 0.1224 - val_
accuracy: 0.9702
Epoch 50/100
95/95 [==============================] - 30s 314ms/step - loss: 0.0849 - accuracy: 0.9714 - val_loss: 0.1178 - val_
accuracy: 0.9732
Epoch 51/100
95/95 [==============================] - 30s 315ms/step - loss: 0.0892 - accuracy: 0.9691 - val_loss: 0.1302 - val_
accuracy: 0.9688
Epoch 52/100
95/95 [==============================] - 30s 313ms/step - loss: 0.0771 - accuracy: 0.9740 - val_loss: 0.1318 - val_
accuracy: 0.9688
Epoch 53/100
95/95 [==============================] - 30s 319ms/step - loss: 0.0888 - accuracy: 0.9711 - val_loss: 0.1286 - val_
accuracy: 0.9702
EarlyStopping ! *Saved best weights*
```

Рисунок В.5 - Візуалізація епох тренування моделі файн-тюнінгу

Продовження додатку В

Рисунок В.6 - Візуалізація скорів моделі

Продовження додатку В


```
"model_prediction_measurement": {
    "experiments_prediction_time_results": [],
    "input_image_shape": [
        1,
        224,
        224,
        3
    ],
    "unit_of_measurement": "s",
    "time_results_hist_description": {
        "min": 0.021428346633911133,
        "max": 0.3373737335205078,
        "median": 0.021989345550537111,
        "mean": 0.022931873183888949,
        "std": 0.012677127615202193,
        "skewness": 23.118903193622845,
        "kurtosis": 559.3429854877547
    }
}
```


Рисунок В.7 - Візуалізація часу роботи моделі

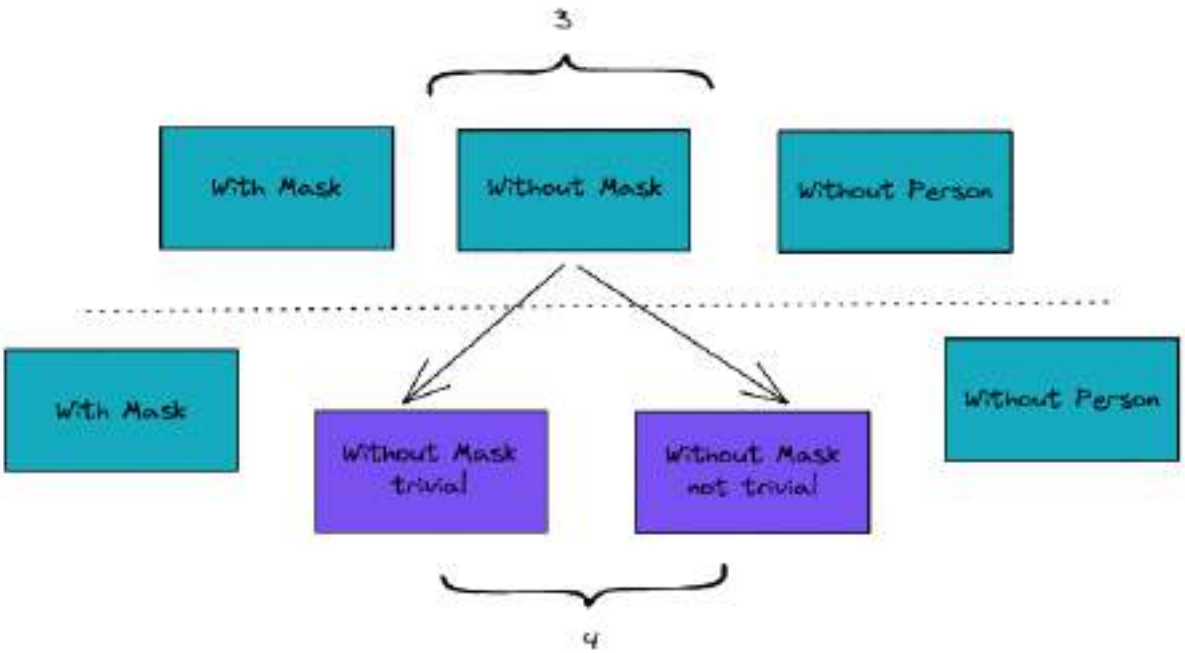


Рисунок В.8 - Розщеплення класу “Without Mask” на два повноцінних підкласи “Without Mask trivial” та “Without Mask not trivial”

Продовження додатку В

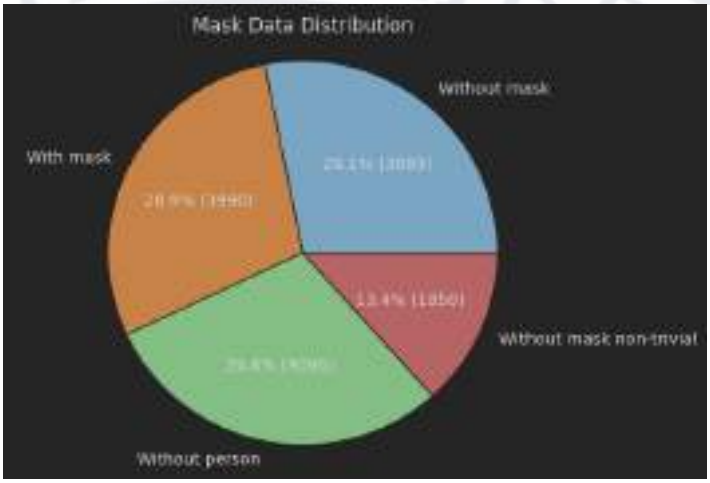


Рисунок В.9 - Новий розподіл даних після операції розщеплення

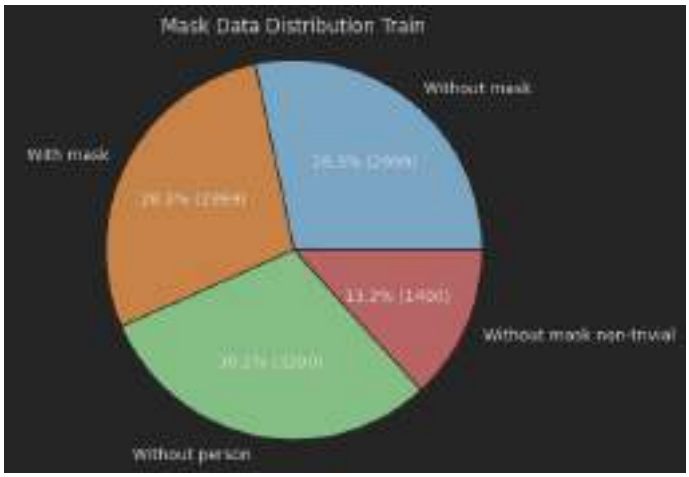


Рисунок В.10 - Новий розподіл даних після операції розщеплення

Продовження додатку В

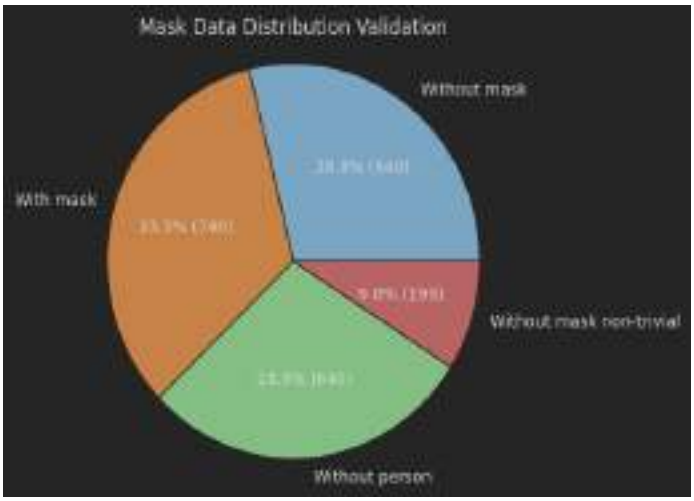


Рисунок В.11 - Новий розподіл даних валідаційної вибірки після операції розщеплення

Додаток Д

Конфігураційний файл для моделі з 3 класами


```
{
 "CATEGORIES_NUMBER" : 4,
 "MODEL" : {
  "name" : "MobileNet",
  "fine_tune_layers" : [
    {"type": "Dropout", "parameters": {"rate": 0.5}},
    {"type": "Dense", "parameters": {"units": 4096, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.35}},
    {"type": "Dense", "parameters": {"units": 4096, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.35}},
    {"type": "Dense", "parameters": {"units": 1028, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.25}},
    {"type": "Dense", "parameters": {"units": 128, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.2}},
    {"type": "Dense", "parameters": {"units": 4, "activation": "softmax"}}
  ],
  "class_weights" : {
    "0": 1,
    "1": 1,
    "2": 1,
    "3": 3
  },
  "loss" : "categorical_crossentropy",
  "optimizer_base" : {
    "type":  "Adam",
    "parameters": {
      "learning_rate": 0.0005
    }
  },
  "optimizer_fine_tune" : {
```

Продовження додатку Д


```
"type":  "SGD",
 "parameters": {
  "lr": 0.000001,
  "momentum": 0.9
 }
},
"callbacks" : [
 {
  "type":  "ReduceLROnPlateau",
  "parameters": {
   "factor": 0.3,
   "patience": 10,
   "min_delta": 0.001
  }
 },

 {
  "type":  "EarlyStopping",
  "parameters": {
   "min_delta": 0.001,
   "patience": 10,
   "restore_best_weights": true
  }
 },

 {
  "type":  "CustomCallback",
  "parameters": {
   "patience_stop_train": 20,
   "patience_decrease_lr": 15,
   "decrease_factor": 0.3
  }
 }
```

Продовження додатку Д


```
],
  "layers_to_unfreeze_fine_tune" : 24,
  "epoch_number_appended_layers" : 150,
  "epoch_number_fine_tune" : 100
},

"DATA_AUGMENTATION_TRAIN" : {
  "samplewise_std_normalization": true,
  "horizontal_flip": true,
  "brightness_range": [0.8,1.2],
  "rotation_range": 30,
  "zoom_range": 0.1,
  "fill_mode": "nearest",
  "width_shift_range": 0.15,
  "height_shift_range": 0.15
},
"ID" : "MobileNet_4cl",
"OUTPUT_FOLDER_PATH" :
"/home/vladislav/PycharmProjects/Nestlogic/Aiola/shared-
backup/Scratches/DonnuFaceMask/outputs/MobileNet_4cl"
}
```

Конфігураційний файл для моделі з 4 класами


```
{
 "CATEGORIES_NUMBER" : 3,
 "MODEL" : {
  "name" : "MobileNet",
  "fine_tune_layers" : [
    {"type": "Dropout", "parameters": {"rate": 0.5}},
    {"type": "Dense", "parameters": {"units": 4096, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.35}},
    {"type": "Dense", "parameters": {"units": 4096, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.35}},
    {"type": "Dense", "parameters": {"units": 1028, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.25}},
    {"type": "Dense", "parameters": {"units": 128, "activation": "relu"}},
    {"type": "Dropout", "parameters": {"rate": 0.2}},
    {"type": "Dense", "parameters": {"units": 3, "activation": "softmax"}}
  ],
  "class_weights" : {
    "0": 1,
    "1": 1,
    "2": 1
  },
  "loss" : "categorical_crossentropy",
  "optimizer_base" : {
    "type":  "Adam",
    "parameters": {
      "learning_rate": 0.0005
    }
  },
  "optimizer_fine_tune" : {
    "type":  "SGD",
```

Продовження додатку Д


```
  "parameters": {
    "lr": 0.000001,
    "momentum": 0.9
  }
 },
 "callbacks" : [
  {
    "type":  "ReduceLROnPlateau",
    "parameters": {
      "factor": 0.3,
      "patience": 10,
      "min_delta": 0.001
    }
  },

  {
    "type":  "EarlyStopping",
    "parameters": {
      "min_delta": 0.001,
      "patience": 10,
      "restore_best_weights": true
    }
  },

  {
    "type":  "CustomCallback",
    "parameters": {
      "patience_stop_train": 20,
      "patience_decrease_lr": 15,
      "decrease_factor": 0.3
    }
  }
 ],
```

Продовження додатку Д


```
"layers_to_unfreeze_fine_tune" : 24,
  "epoch_number_appended_layers" : 150,
  "epoch_number_fine_tune" : 100
},

"DATA_AUGMENTATION_TRAIN" : {
  "samplewise_std_normalization": true,
  "horizontal_flip": true,
  "brightness_range": [0.8,1.2],
  "rotation_range": 30,
  "zoom_range": 0.1,
  "fill_mode": "nearest",
  "width_shift_range": 0.15,
  "height_shift_range": 0.15
},
"ID" : "MobileNet_3cl",
"OUTPUT_FOLDER_PATH" :
"/home/vladislav/PycharmProjects/Nestlogic/Aiola/shared-
backup/Scratches/DonnuFaceMask/outputs/MobileNet_3cl"
}
```

Продовження додатку Д

Рисунок Д.1 - Дані, на яких помилилась модель


```
"model_prediction_measurement": {
    "experiments_prediction_time_results": []
    "input_image_shape": [
        1,
        224,
        224,
        3
    ],
    "unit_of_measurement": "ms",
    "time_results_hist_description": {
        "min": 0.02173161506652832,
        "max": 0.3313870429992676,
        "median": 0.02251267433166504,
        "mean": 0.024024073328848013,
        "std": 0.012336367930140757,
        "skewness": 22.706235266378027,
        "kurtosis": 553.1807013002021
    }
}
```


Рисунок Д.2 - Візуалізація часу роботи моделі

Продовження додатку Д

Рисунок Д.3 - Візуалізація скорів моделі

Додаток Ж

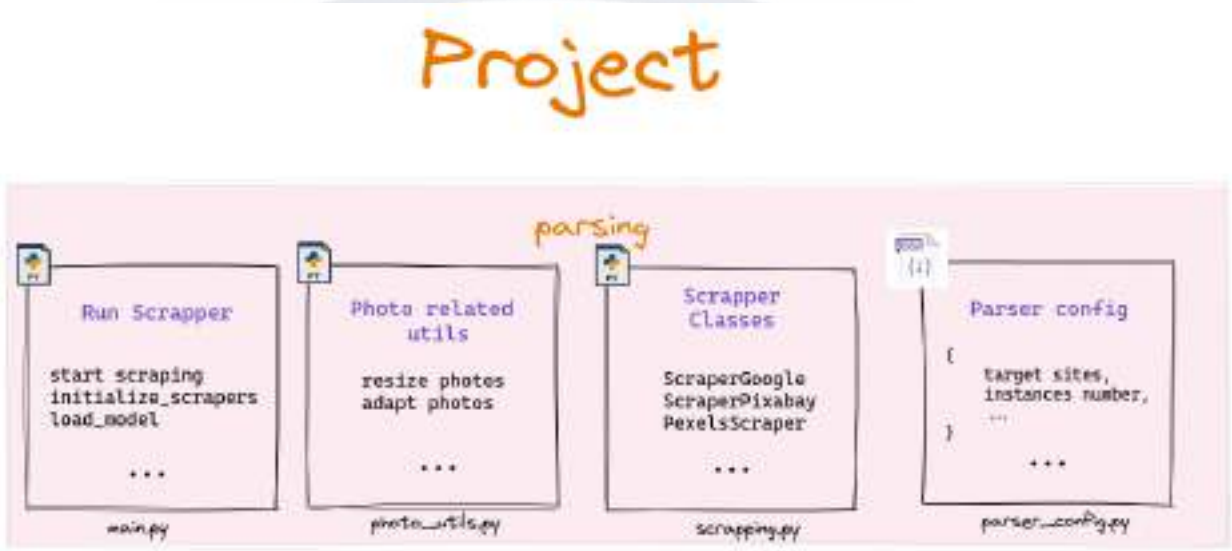


Рисунок Ж.1 - Структура проекту парсингу

Продовження додатку Ж

```
class BaseScraper():
    def __init__(self, what_to_scrap, max_fetches, folder_path, model=None):
        self.folder_path = folder_path
        self.what_to_scrap = what_to_scrap
        self.max_fetches = max_fetches
        self.URLSset = set()  # множина URL-адрес зображень для завантаження
        self.model = model

    def get_URLS_len(self):
        return len(self.URLSset)

    def run(self):
        self.extract_images_url()

    def extract_images_url(self):
        raise NotImplementedError("Цей метод має бути перевизначений у підкласі")

    def download_batch_images(self, urls_tmp):
        difference = urls_tmp - self.URLSset
        for src in difference:
            self.download_image(src)
        self.URLSset = self.URLSset.union(difference)

    def download_image(self, URL):

        try:
            image_content = requests.get(URL).content  # отримуємо байти зображення
        except Exception as e:
            print(f"Помилка під час завантаження зображення - {e}")
        try:
            image_file = io.BytesIO(image_content)
            image = Image.open(image_file).convert("RGB")
            save_image = True
            if self.model:
                without_mask = int(
                    self.model.predict(np.asarray(image.resize((224, 224))).reshape(1, 224, 224, 3))[0][0])
                if without_mask:
                    save_image = False
            if save_image:
                file_path = os.path.join(self.folder_path,
                                         hashlib.sha1(image_content).hexdigest()[:12] + '.jpg')
                with open(f"{file_path}", "wb") as f:
                    image.save(f, "JPEG", quality=85)
        except Exception as e:
            print(f"Помилка - не вдалося зберегти - {e}")
```

Рисунок Ж.2 - Base Scraper клас

Продовження додатку Ж

configuration.json

```
{
  "SUBJECT_TO_SCRAP": null,
  "SAVE_PHOTOS_BASE_DIR_PATH": null,
  "SAVE_PHOTOS_DIR_NAME": null,
  "MODEL_LOCALE": {
    "MODEL_WEIGHTS": null,
    "MODEL_PATH": null
  },
  "ENABLE_MODEL": null,
  "TARGET_SITES": null,
  "INSTANCES_NUMBER": null
}
```

Рисунок Ж.3 - Файл конфігурації парсеру

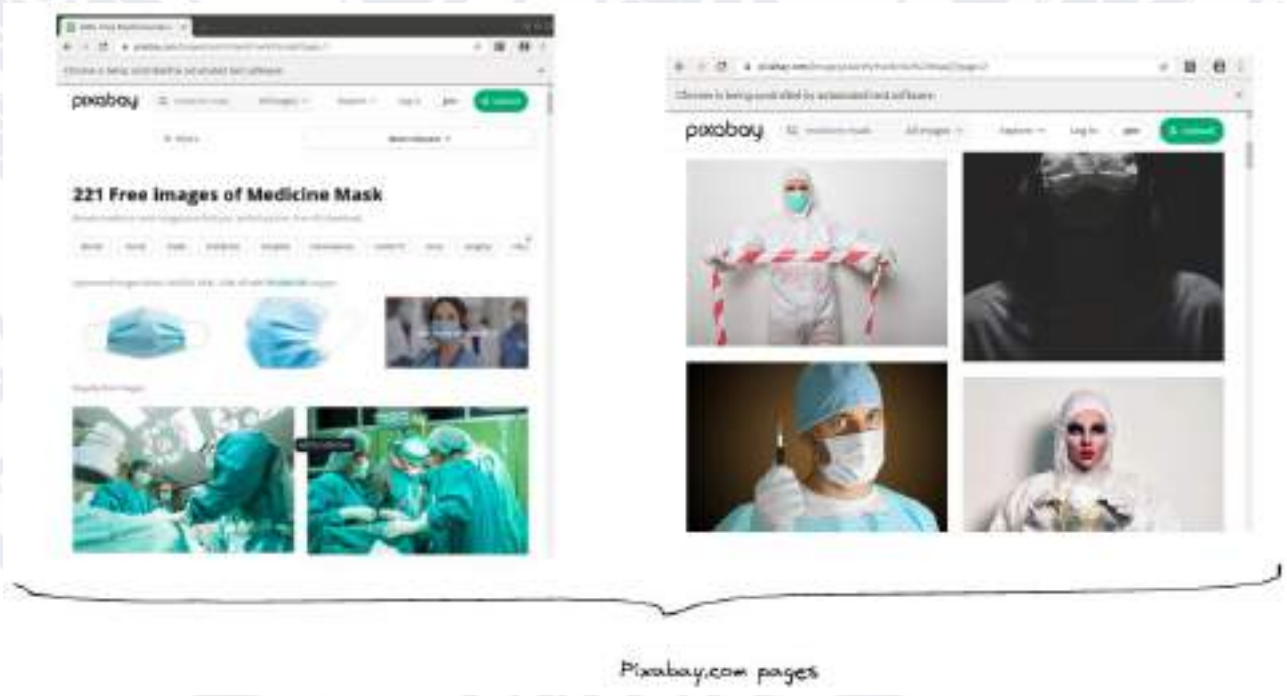


Рисунок Ж.4 - Скрапінг фотографій з Pixabay

Продовження додатку Ж

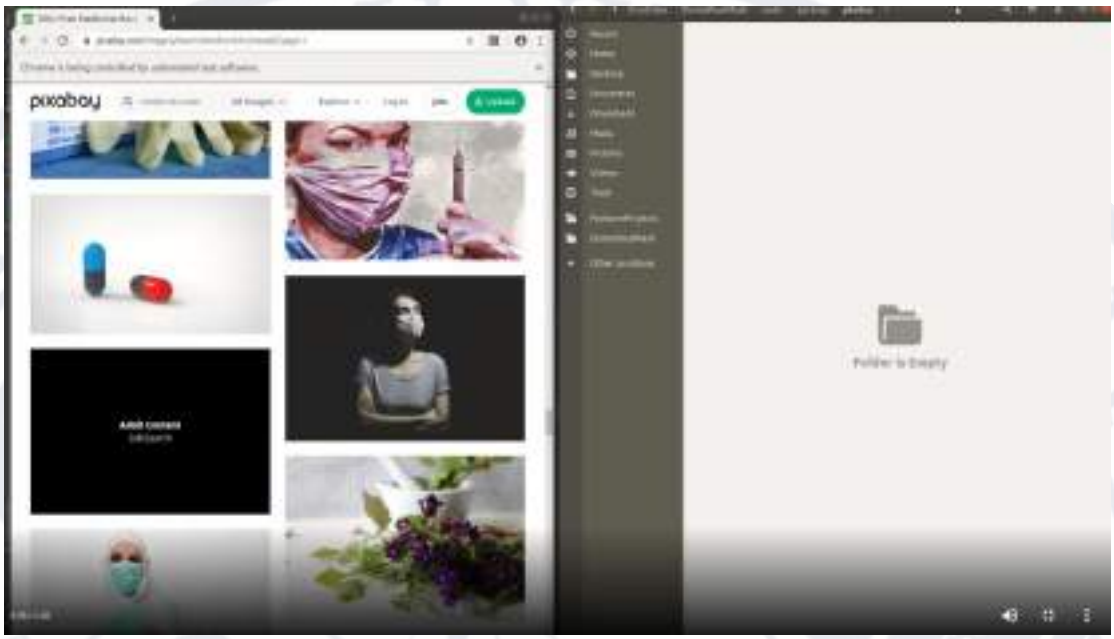


Рисунок Ж.5 - Початок скрапінгу фотографій з джерела Pixabay

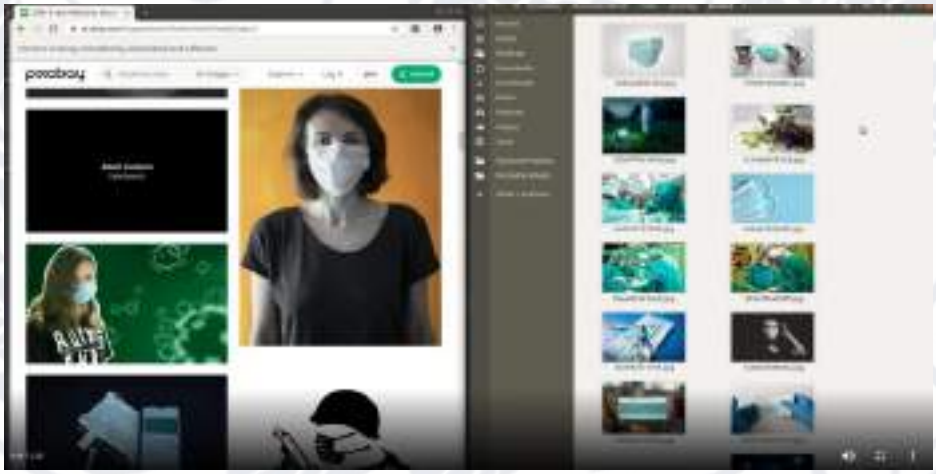


Рисунок Ж.6 - Початок скрапінгу фотографій з джерела Pixabay.

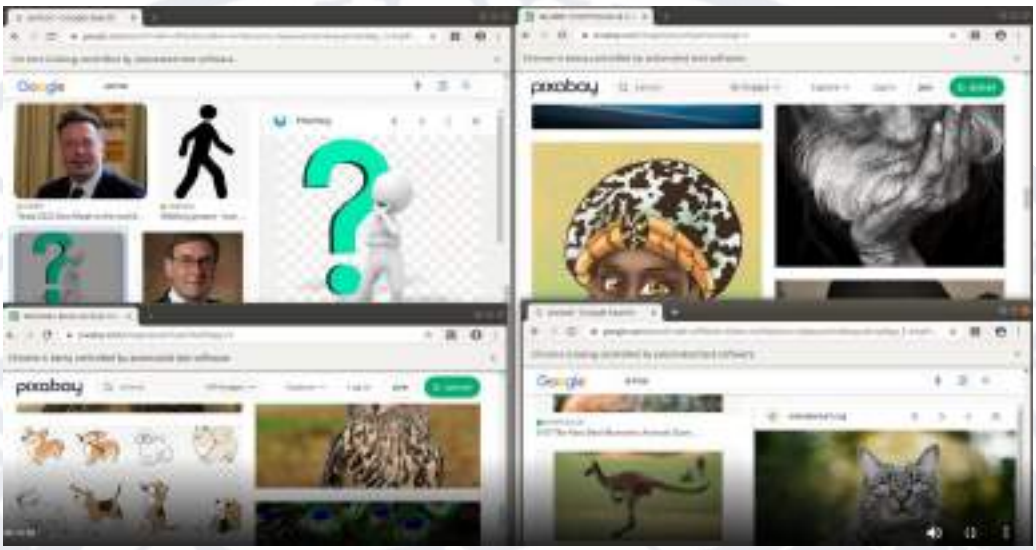


Рисунок Ж.7 - Мультипроцессинг декількох джерел одночасно

```
def initialize_scrapers(subject_to_scrap, folder_name_to_save_images, instances_number, model=None):
    scrapers = []
    for i in range(instances_number):
        if "Google" in TARGET_SITES:
            scrapers.append(
                ScraperGoogle(
                    what_to_scrap=subject_to_scrap,
                    max_fetches=400,
                    folder_path=os.path.join(SAVE_PHOTOS_BASE_DIR_PATH, folder_name_to_save_images),
                    model=model
                )
            )

        if "Pixabay" in TARGET_SITES:
            scrapers.append(ScraperPixabay(subject_to_scrap, 800, os.path.join(SAVE_PHOTOS_BASE_DIR_PATH, folder_name_to_save_images), model))

        if "Pexels" in TARGET_SITES:
            scrapers.append(ScraperPexels(subject_to_scrap, 800, os.path.join(SAVE_PHOTOS_BASE_DIR_PATH, folder_name_to_save_images), model))

    return scrapers
```

Рисунок Ж.8 - Ініціалізація скраперів

```
def start_scraping(scraper_objs):
    processes = []
    for scraper in scraper_objs:
        process = Process(target=run, args=(scraper,))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()
```

Рисунок Ж.9 - Функція, яка задає початок скрапінгу

Додаток К

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)