МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КЛАСИФІКАЦІЇ ЗМІЙ НА БАЗІ ІНСТРУМЕНТАРІЮ ШІ

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 401.22010318

Виконав студент 4-го курсу, групи 401
_____ Д. К. Степанчук
«17» червня 2024 р.

Керівник: PhD, ст. викладач
_____І. О. Кандиба
«17» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Степанчуку Дмитру Костянтиновичу.

1. Тема кваліфікаційної роботи «Програмне забезпечення класифікації змій на базі інструментарію ШІ».

Керівник роботи Кандиба Ігор Олександрович, PhD, ст. викладач.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «17» червня 2024 р.

3. Вхідні (початкові) дані до роботи: набір даних у фотографій змій; бази даних ареалів проживання певних видів змій; моделі машинного навчання та алгоритми штучного інтелекту; інструменти для моніторингу та аналізу продуктивності системи; критерії оцінки точності.

Очікуваний результат: система, здатна за фотографією змії спрогнозувати ймовірність того, що дана змія є отруйною.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз сучасного стану задачі класифікації зображень змій;
- огляд існуючих методів і підходів для класифікації об'єктів, зображень;

– дослідження застосування алгоритмів штучного інтелекту для класифікації об’єктів;

– розробка інформаційної системи для класифікації змій за їх зображенням та супутньою інформацією;

– тестування та оцінка ефективності розробленої системи.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Розробка заходів з поліпшення умов праці».

7. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А.О., канд. техн. наук	

Керівник роботи __PhD, ст. викладач Кандиба І. О._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Степанчук Д. К._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_14_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Програмне забезпечення класифікації змій на базі інструментарію ШІ

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз існуючих систем класифікації змій з використанням технологій ШІ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
12	Захист БКР перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	Виконано

Розробив студент __Степанчук Д. К._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __PhD, старший викладач Кандиба І. О. _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили
Степанчука Дмитра Костянтиновича
Тема: «Програмне забезпечення класифікації змій на базі інструментарію ШІ»

Кваліфікаційну роботу бакалавра присвячено розробці програмного забезпечення для класифікації змій за допомогою інструментарію штучного інтелекту. Актуальність полягає у потребі негайно визначити тип змії для встановлення інших даних, таких як її поширеність чи отруйність.

Об'єктом кваліфікаційної роботи є процес класифікації об'єктів на зображенні, зокрема змій.

Предметом кваліфікаційної роботи є інструментарій розробки програмного забезпечення для класифікації змій на базі інструментарію ШІ.

Метою кваліфікаційної роботи є вдосконалення процесу класифікації змій, шляхом розробки програмного забезпечення на базі інструментарію ШІ.

Для досягнення поставленої мети поставлено наступні завдання:

– дослідити галузі застосування ШІ;

– провести аналіз засобів класифікації зображень з використанням ШІ;

– провести дослідження сучасних засобів реалізації ШІ в контексті розпізнавання змій;

– провести навчання нейромереж та оцінити результат цього навчання;

– реалізувати тестування розробленого програмного забезпечення.

Кваліфікаційна робота бакалавра складається з фахової частини і спеціальної частини з охорони праці. Фахова частина в свою чергу поділяється на вступ, чотири розділи, висновки та переліку джерел посилання.

У вступі описується актуальність теми роботи, зазначено об'єкт, предмет, а також мета й завдання, які необхідно виконати для досягнення цієї мети.

У першому розділі проводиться аналіз предметної області. Також оглянуто наявні аналоги, здійснено їх порівняння на основі функціональних можливостей,

оцінено переваги та недоліки. У результаті було складено вимоги до програмного забезпечення, що розробляється, наведено схему роботи моделі.

У другому розділі описано технології, які будуть використані на етапі навчання моделей, розробки серверної частини та клієнта.

У третьому розділі описано процес та навчання нейронних мереж, проаналізовано результати навчання, розглянуто деталі реалізації блоку BCN моделі.

У четвертому розділі демонструється процес розробки серверної та клієнтської частини, розгортки моделі та здійснюється огляд результатів.

У висновках проводиться аналіз виконаної роботи та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 58 сторінок, містить 4 розділи, 44 ілюстрацій, 0 таблиць, 26 джерел в переліку посилань.

Ключові слова: *python, комп'ютерний зір, машинне навчання, класифікація, згорткові нейронні мережі, telegram, flask, pytorch, глибоке навчання.*

ABSTRACT

qualification work by a student of group 401 of Petro Mohyla BSNU

Dmytro Kostiantynovych Stepanchuk

Topic: "Snake classification software based on AI tools"

The bachelor's thesis is devoted to the development of software for snake classification using artificial intelligence tools. The relevance lies in the need to immediately determine the type of snake in order to establish other data, such as its prevalence or poisonousness.

The object of the qualification work is the process of classifying objects in an image, in particular snakes.

The subject of the qualification work is the tools for developing software for classifying snakes based on AI tools.

The purpose of the qualification work is to improve the process of classifying snakes by developing software based on AI tools.

To achieve this goal, the following tasks have been set:

- to investigate the areas of application of AI;
- to analyze image classification tools using AI;
- to study modern means of implementing AI in the context of snake recognition;
- to train neural networks and evaluate the result of this training;
- to implement testing of the developed software.

The bachelor's thesis consists of a professional part and a special part on labor protection. The professional part, in turn, is divided into an introduction, four chapters, conclusions and a list of references.

The introduction describes the relevance of the topic of the work, specifies the object, subject, as well as the goal and tasks to be performed to achieve this goal.

The first chapter analyzes the subject area. It also reviews existing analogs, compares them based on functionality, and evaluates their advantages and disadvantages. As a result, the requirements for the software to be developed were drawn up, and the model's workflow was presented.

The second section describes the technologies that will be used at the stage of model training, server and client development.

The third section describes the process and training of neural networks, analyzes the training results, and discusses the details of the implementation of the BCN block of the model.

The fourth section demonstrates the process of developing the server and client parts, model deployment, and reviews the results.

The conclusion analyzes the work done and the results obtained.

The bachelor's thesis is 58 pages long, contains 4 chapters, 44 illustrations, 0 tables, 26 references.

Keywords: *python, computer vision, machine learning, classification, convolutional neural networks, telegram, flask, pytorch, deep learning.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CNN – convolutional neural network

API – application programming interface

BN – batch norm

LN – layer norm

BCN – batch-channel norm

KNN – k nearest neighbors

RNN – recurrent neural network

ML – machine learning

DL – deep learning

PCA – principal component analysis

SVM – support vector machine

ВСТУП

Змії є надзвичайно захоплюючими та загадковими плазунами, які населяють майже всі куточки світу, за винятком Антарктиди та деяких віддалених островів. Вони відрізняються різноманіттям розмірів, форм і кольорів: від крихітних гадюк, довжина яких лише кілька сантиметрів, до величних пітонів, які можуть сягати 10 метрів. Ці дивовижні створіння пристосувалися до життя у найрізноманітніших середовищах, включаючи густі джунглі, спекотні пустелі, а також їх можна зустріти як на землі, так і на деревах, у воді та навіть під землею.

Наразі в світі відомо понад 3000 видів змій. Деякі з них є отруйними, і їх укуси можуть бути надзвичайно небезпечними, а іноді й смертельними для людини. Отрута змій є складним коктейлем білків, які можуть викликати різноманітні ефекти, такі як параліч, зупинка дихання, руйнування тканин і навіть смерть. Цікаво, що отруйні змії не утворюють єдиної або спеціальної таксономічної групи. Отрута виникла в кількох родинах, що свідчить про те, що отрута у змій з’являлася більше одного разу в результаті конвергентної еволюції. Приблизно чверть усіх видів змій вважаються отруйними.

Зустрічі зі зміями можуть траплятися як у дикій природі, так і в урбанізованих місцевостях. Це може бути під час прогулянки або наукового дослідження. Іноді виникає необхідність визначити вид змії, яку зустріли, особливо якщо є потреба в встановленні виду, але немає можливості взяти змію з собою. Це особливо важливо при укусах змій, оскільки правильна ідентифікація може бути критичною для надання медичної допомоги потерпілому. У таких випадках підручні інструменти фотофіксації можуть бути дуже корисними. Важливо не лише встановити вид змії, але й з'ясувати, чи є вона отруйною.

Класифікація змії за фотографією може бути складною задачею через обмежену інформацію, яку надає зображення. Проблеми можуть виникати через низьку роздільну здатність фото, низьку чіткість або погану контрастність. Усі ці фактори є викликами, які потрібно подолати, щоб таке програмне забезпечення мало практичне застосування.

Об’єктом кваліфікаційної роботи є процес класифікації об’єктів на зображенні, зокрема змій.

Предметом кваліфікаційної роботи є інструментарій розробки програмного забезпечення для класифікації змій на базі інструментарію ШІ.

Метою кваліфікаційної роботи є вдосконалення процесу класифікації змій, шляхом розробки програмного забезпечення на базі інструментарію ШІ.

Для досягнення поставленої мети поставлено наступні завдання:

– дослідити галузі застосування ШІ;

– провести аналіз засобів класифікації зображень з використанням ШІ;

– провести дослідження сучасних засобів реалізації ШІ в контексті розпізнавання змій;

– провести навчання нейромереж та оцінити результат цього навчання;

– реалізувати тестування розробленого програмного забезпечення.

1. АНАЛІЗ ГАЛУЗЕЙ ЗАСТОСУВАННЯ ШІ

1.1 Опис сучасного стану у засобах ШІ для задач класифікації

Перш за все треба визначити що таке задача класифікації, та які засоби штучного інтелекту зараз використовуються для вирішення задач класифікації.

Класифікація – це процес віднесення об'єкта або явища до однієї з декількох заздалегідь визначених категорій або класів на основі певних характеристик або ознак. У контексті машинного навчання та штучного інтелекту класифікація є однією з основних задач, яка передбачає навчання алгоритму на основі навчальних даних з метою прогнозування категорії нових, невідомих даних. Отже, вхідними даними виступає набір ознак (пікселі картинки, частоти звуку, слова у реченні, інші числові, порядкові або категоріальні величини), а вихідними даними є клас (категоріальна характеристика) або ймовірності чи міри приналежності (числові характеристики) до кожного з класів, що представлені у задачі.

Також варто сказати, що у методах штучного інтелекту є декілька принципових підходів, які можуть бути використані чи скомбіновані для вирішення певної задачі, мають різні характеристики, такі як обчислювальна складність та точність моделі.

Класичне машинне навчання включає методи, які використовуються для побудови моделей на основі даних. Ці методи включають алгоритми, які не використовують нейронні мережі з великою кількістю шарів. Класичне машинне навчання включає регресію, класифікацію, кластеризацію, і зменшення розмірності. У розрізі задач класифікації виділяють деякі з них:

– логістична регресія, багатокласова логістична регресія це статистичний метод для прогнозування ймовірностей віднесення спостережень до різних класів на основі досліджуваних змінних.

$$p(X) = \frac{1}{1 + \exp\left(-\left(\sum_{i=0}^{n} b_i\, x_i\right)\right)} \qquad (1.1)$$

Цей метод використовується у задачах класифікації. Розраховується за формулою (1).

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_{k \in K} \exp(z_k)} \quad (1.2)$$

У випадку багатокласової класифікації після знаходження мір приналежності до кожного з класів (вони називаються логітами), за допомогою функції softmax (2) знаходяться ймовірності приналежності до кожного класу;

– **KNN** – це простий і досить ефективний алгоритм для класифікації та регресії. Основна ідея полягає в тому, щоб класифікувати нові точки даних на основі класів їхніх найближчих сусідів у просторі ознак. Для задачі класифікації серед найближчих k сусідів до даної точки розглядаються їх класи. Клас, який буде найчастіше зустрічатись серед сусідів і буде виходом моделі. Для роботи алгоритму треба визначити такі гіперпараметри як кількість сусідів, метрику відстані та масштабувати ознаки, аби усі ознаки однаково враховувались;

– **naive bayes** – це проста, але потужна ймовірнісна класифікаційна модель, заснована на застосуванні теореми Байєса з припущенням про незалежність ознак. Незважаючи на це припущення, яке рідко виконується в реальних даних, модель часто демонструє високу ефективність у багатьох задачах класифікації.

Основна ідея Naive Bayes полягає в тому, що для кожного класу обчислюється ймовірність спільного розподілу ознак за допомогою теореми Байєса. Теорема Байєса дозволяє обчислити апостеріорну ймовірність класу на основі наявних даних Розраховується за формулою (3).

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \quad (1.3)$$

Навчання Naive Bayes полягає в обчисленні апріорних ймовірностей класів та умовних ймовірностей ознак для кожного класу на основі навчальної вибірки. Під час класифікації нових даних модель обчислює апостеріорні ймовірності для

всіх класів і вибирає клас з найвищою ймовірністю. Naive Bayes часто використовується у задачах текстової класифікації, таких як спам-фільтрація, аналіз настроїв та інші;

– **дерева рішень** – це метод машинного навчання, який використовується для класифікаційних та регресійних задач. Дерево рішень представляє собою ієрархічну структуру, де кожен вузол приймає рішення на основі певних правил, а гілки з'єднують ці вузли, ведучи до кінцевих рішень або прогнозів. Кореневий вузол є першим вузлом дерева і містить весь набір даних, починаючи процес поділу (рис. 1.1).

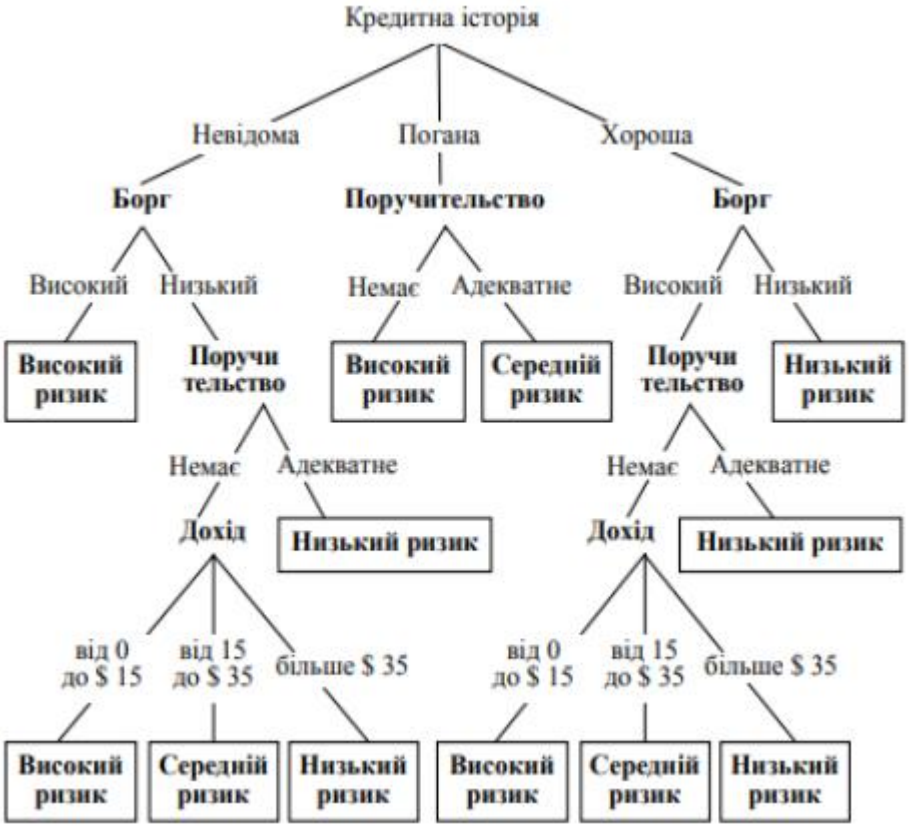


Рисунок 1.1 – Приклад візуалізації дерева рішень [7]

У кореневому вузлі вибирається ознака, яка найбільше зменшує невизначеність, наприклад, ентропію або індекс Джині, у наборі даних. Внутрішні вузли відповідають за подальший поділ даних на основі певних ознак. Кожен внутрішній вузол містить правило поділу, яке визначає, як дані розбиваються на підмножини. Листові вузли (або листки) є кінцевими вузлами дерева і містять

остаточне рішення або прогноз. Під час навчання дерева рішень алгоритм обирає найкращі ознаки для поділу на кожному кроці, щоб максимізувати точність прогнозів. Дерева рішень мають переваги, такі як простота візуалізації та інтерпретації, але можуть бути схильні до перенавчання, особливо якщо дерево стає занадто глибоким.

Головною відмінністю класичних алгоритмів машинного навчання є швидкодія, бо зазвичай такі алгоритми швидко навчаються, але через це мають недолік – вони більше схильні до перенавчання, для чого застосовуються методи регуляризації моделей машинного навчання. Також, через свою простоту, іноді вони є не дуже ефективними, якщо закономірності в даних складні.

Іншим підходом до машинного навчання є **глибинне навчання**, тобто навчання нейромереж. Предстаниками таких є MLP, CNN, RNN.

MLP – це один з найпоширеніших типів штучних нейронних мереж, що використовується в задачах класифікації та регресії. МЛП складається з кількох шарів нейронів: вхідного, одного або більше прихованих шарів і вихідного шару (рис. 1.2).

Вхідний шар отримує вхідні дані і передає їх до першого прихованого шару. Кожен нейрон у вхідному шарі відповідає за один елемент вхідного вектора. Приховані шари складаються з нейронів, які отримують сигнали від нейронів попереднього шару, обробляють їх і передають результати на наступний шар. Кількість прихованих шарів і нейронів у кожному шарі може бути різною в залежності від конкретної задачі. Вихідний шар обробляє результати з останнього прихованого шару і видає кінцевий результат мережі. Кількість нейронів у вихідному шарі залежить від характеру задачі, наприклад, для класифікації це може бути кількість класів. МЛП використовує функції активації, такі як сигмоїдна або ReLU, для нелінійного перетворення сигналів.

Навчання мережі здійснюється методом зворотного поширення помилки, який мінімізує різницю між передбаченням мережі і фактичними результатами, коригуючи ваги зв’язків між нейронами.

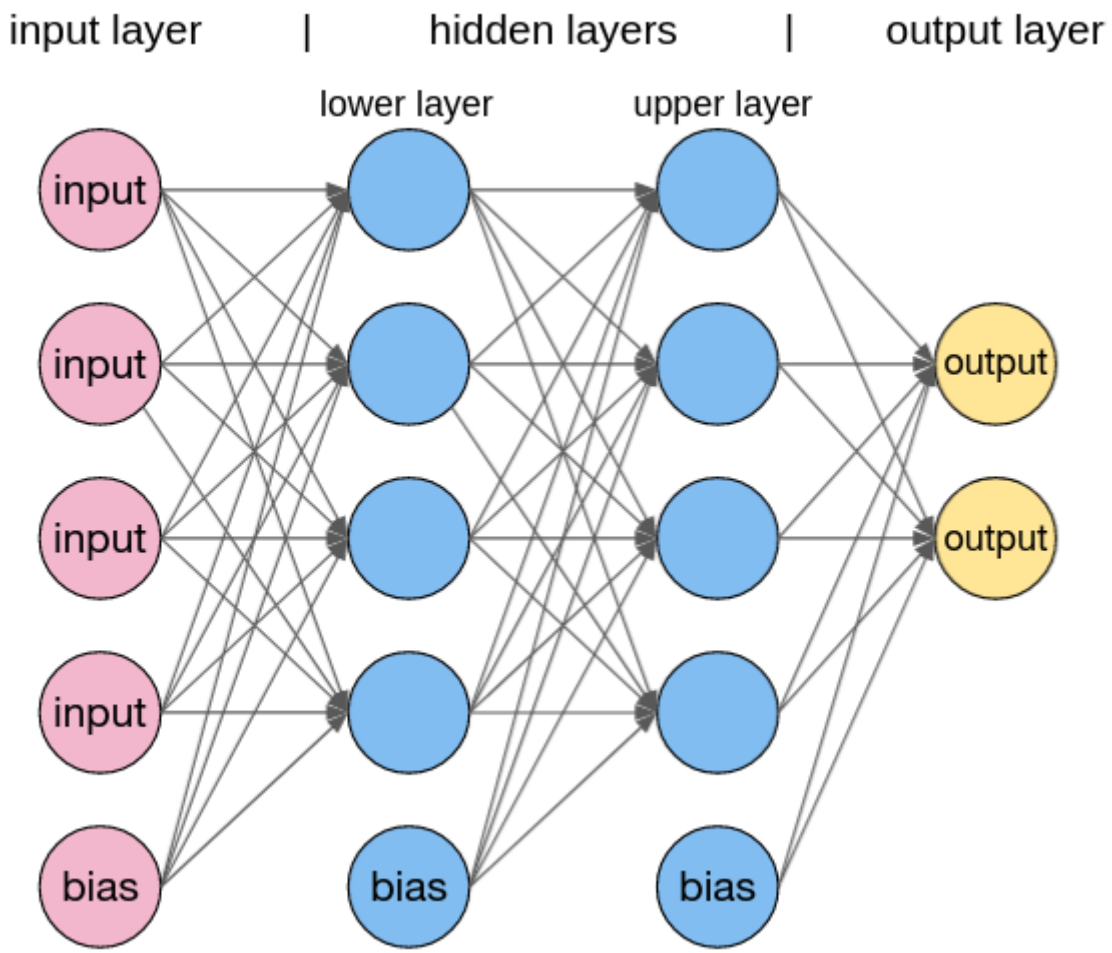


Рисунок 1.2 – Схематичне зображення багатошарового перцептрона [8]

CNN – це спеціальний тип штучної нейронної мережі, розроблений для ефективної обробки та аналізу даних з регулярною структурою, таких як зображення. CNN широко використовуються у задачах комп'ютерного зору, включаючи розпізнавання образів, класифікацію зображень, сегментацію та обробку відео.

CNN складаються з різних шарів, основними з яких є згорткові шари, шари пулінгу, і повнозв'язні шари (рис. 1.3). Згорткові шари використовують фільтри (ядра згортки), які проходять через вхідні дані і витягують локальні ознаки, такі як краї, текстури або інші деталі. Шари пулінгу зменшують розмірність просторових даних, зберігаючи важливу інформацію та знижуючи обчислювальну складність мережі. Повнозв'язні шари аналогічні тим, що використовуються в класичних

нейронних мережах, і служать для остаточного класифікаційного рішення або іншої кінцевої задачі.

CNN використовують функції активації, такі як ReLU (Rectified Linear Unit), які вводять нелінійність у модель. Навчання мережі здійснюється шляхом зворотного поширення помилки, як і в традиційних нейронних мережах, і оптимізації параметрів (ваг) за допомогою алгоритмів градієнтного спуску. CNN особливо ефективні у задачах, де важлива просторово-локальна структура даних, і здатні автоматично виявляти ієрархічні ознаки вхідних даних.

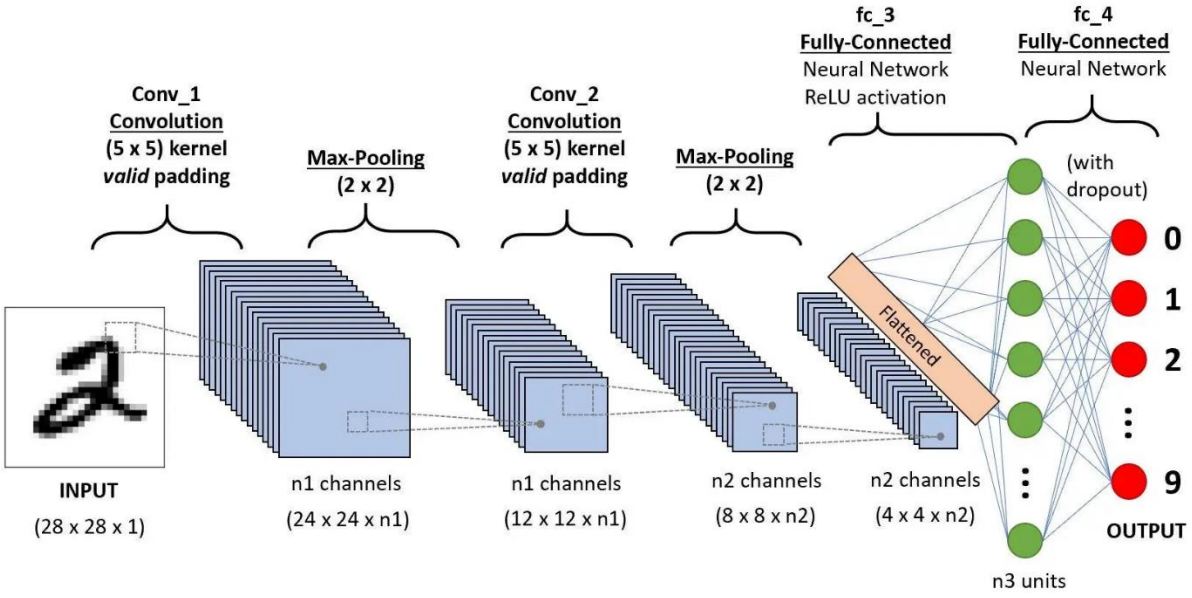


Рисунок 1.3 – Схематичне зображення згорткової нейромережі [9]

RNN – це тип штучної нейронної мережі, спеціально розроблений для обробки послідовних даних, таких як текст, мова або часові ряди. Основна особливість RNN полягає в її здатності зберігати інформацію про попередні етапи у послідовності через рекурентні зв’язки, що дозволяє їй враховувати контекст при прийнятті рішень (рис. 1.4).

В традиційній RNN вихід кожного нейрона залежить не лише від поточного входу, але й від виходу попереднього кроку (стану). Це дає можливість RNN моделювати часові залежності і контекстну інформацію, що є критично важливим

для задач обробки природної мови, машинного перекладу, розпізнавання мови, генерації тексту та аналізу часових рядів.

Однак традиційні RNN мають певні обмеження, зокрема, проблема зникнення або вибуху градієнтів при обробці довгих послідовностей. Для подолання цих проблем були розроблені покращені архітектури, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit), які мають механізми для збереження довготривалої інформації і більш стабільного навчання.

Навчання RNN здійснюється шляхом зворотного поширення помилки через час (BPTT, Backpropagation Through Time), що є модифікацією алгоритму зворотного поширення для обробки послідовних даних. RNN здатні моделювати складні часові залежності і широко використовуються у багатьох додатках, де важлива обробка послідовної інформації.

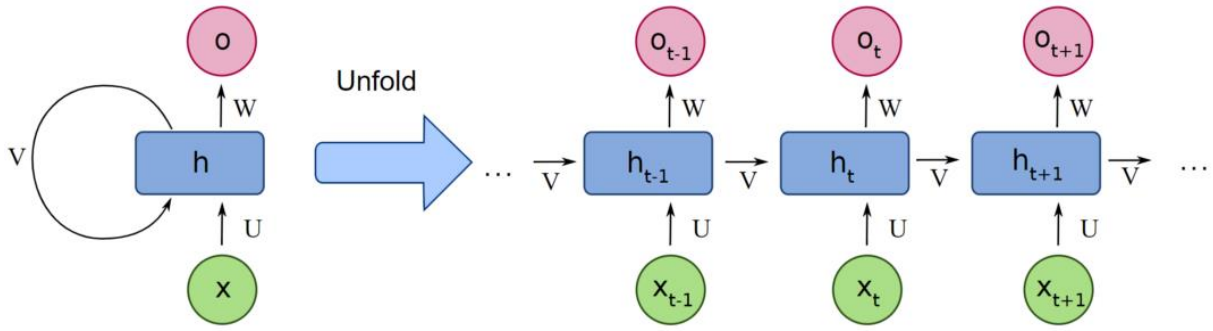


Рисунок 1.4 – Схематичне зображення рекурентної нейромережі [10]

Нейромережевий підхід є ресурсовитратним, але показує на порядок вищу точність у деяких сферах, як natural language processing чи computer vision. Також, хоч нейромережі теж можуть перенавчатись, але менш схильні до цього.

Третім підходом є ансамблеві алгоритми, які використовують велику кількість прогнозів різних «легких» моделей для формування кінцевого прогнозу.

Бегінг – техніка ансамблевого навчання, яка допомагає покращити стабільність і точність моделей, зменшуючи варіативність і запобігаючи перенавчанню (рис. 1.5). Основна ідея полягає в створенні кількох навчальних наборів даних шляхом випадкового вибору з початкового набору даних з поверненням, що називається бутстрепінгом. На кожному з цих наборів

тренуються окремі моделі, а їх прогнози об'єднуються, наприклад, середнім або голосуванням. Ця техніка часто використовується з деревами рішень, створюючи випадкові ліси. Бегінг зменшує варіативність, оскільки використовуються багато моделей, тренованих на різних підмножинах даних, що знижує ризик перенавчання однієї моделі. Також це покращує точність, оскільки об'єднання результатів декількох моделей часто приводить до більш точних прогнозів, ніж використання однієї моделі.

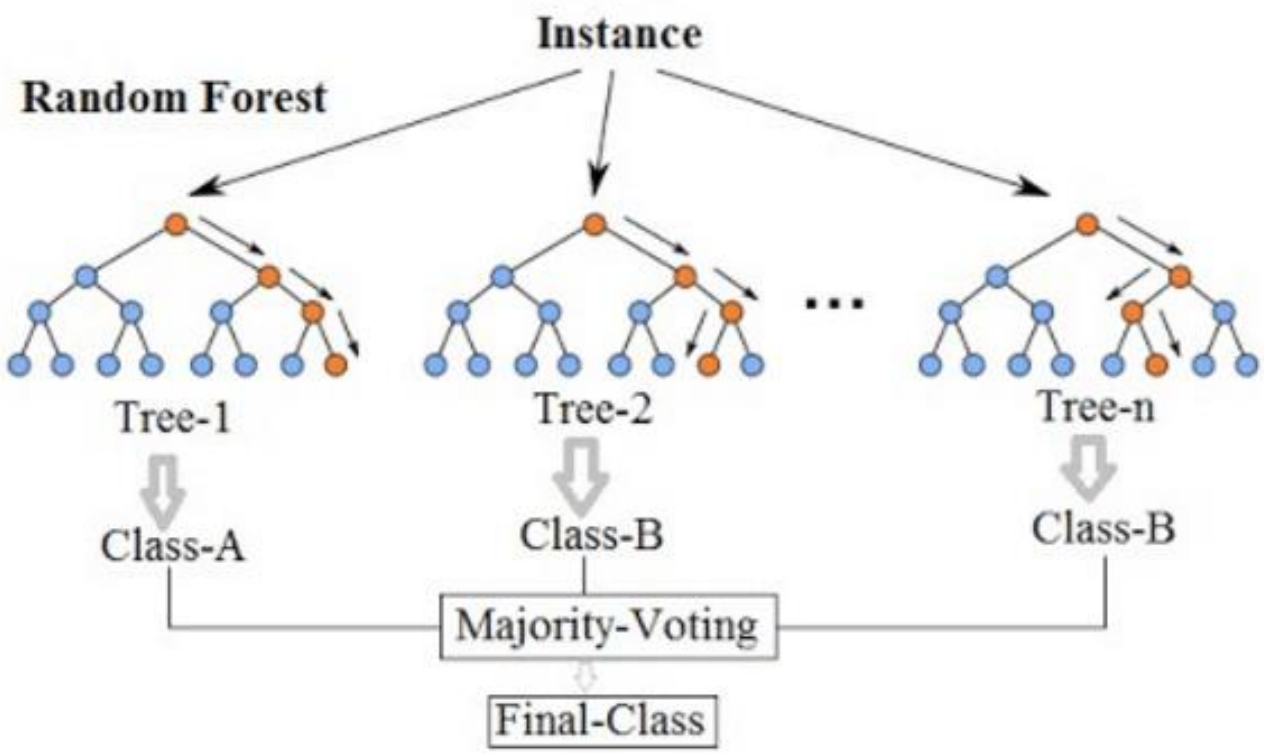


Рисунок 1.5 – Візуалізація методу випадкового лісу, який є одним з прикладів використання бегінгу [11]

Бустінг – це техніка ансамблевого навчання, яка створює модель шляхом послідовного тренування слабких моделей і коригування їх помилок. Основна ідея полягає в побудові моделі шляхом послідовного навчання нових моделей, які фокусуються на виправленні помилок, зроблених попередніми моделями. Це досягається за допомогою методів градієнтного підйому, таких як алгоритм градієнтного бустінгу. Кожна нова модель в бустінгу зосереджена на усуненні

помилок, що роблять попередні моделі, що призводить до покращення загальної точності моделі (рис. 1.6).

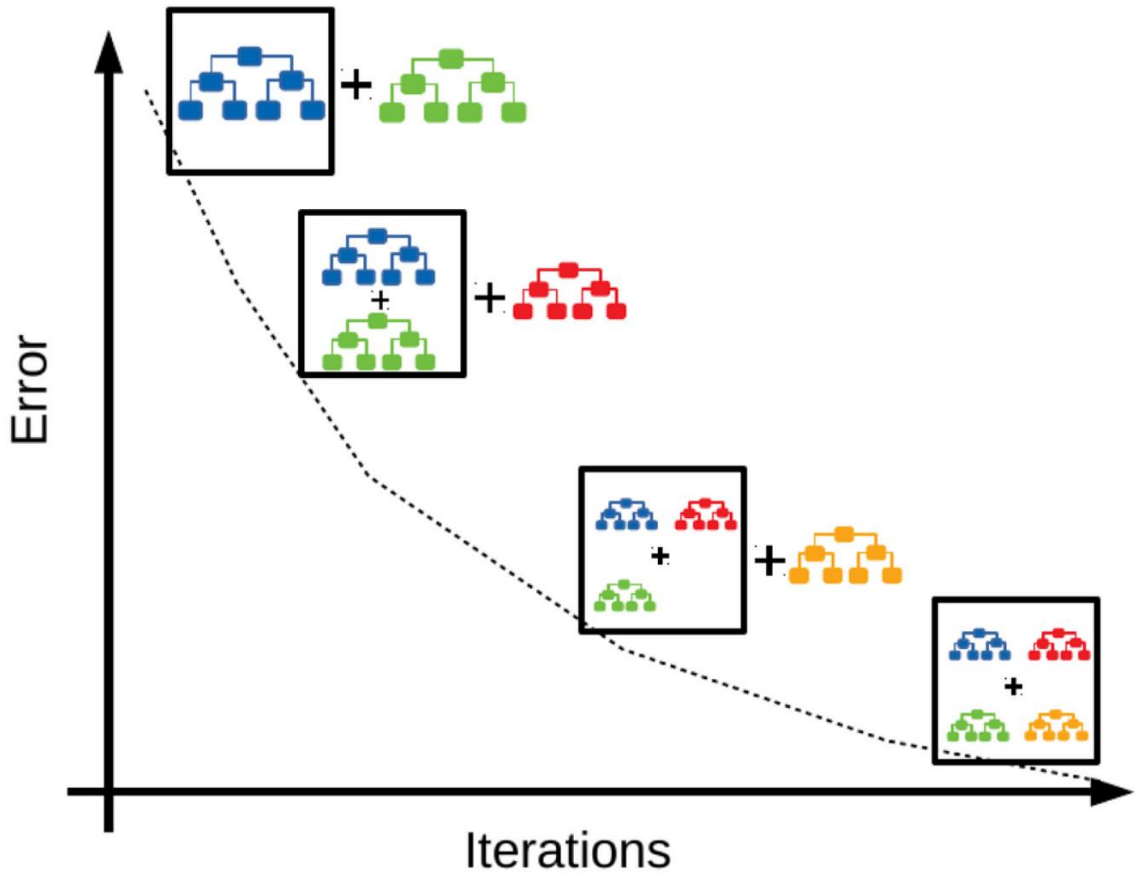


Рисунок 1.6 – Візуалізація бустінгу випадкових дерев [12]

Стекінг – метод ансамблевого навчання, де кілька базових моделей використовуються для генерації прогнозів, які потім комбінуються за допомогою метамоделі. Основна ідея полягає в тому, щоб базові моделі навчалися на вихідних даних і робили прогнози для тестових даних, які потім використовуються як вхідні дані для метамоделі. Метамодель об'єднує ці прогнози для отримання кінцевого прогнозу.

Плюси і мінуси є ідентичними неромережевому підходу – ресурсоємність, але вища точність та менша схильність до перенавчання.

Отже, існує декілька принципово різних підходів для класифікації методами машинного навчання.

1.2 Огляд та аналіз публікацій та розробленого програмного забезпечення, присвячених засобам класифікації зображень, зокрема змій

Аналіз наявних досліджень та публікацій дає змогу побачити різні підходи до вирішення задачі, побачити їх переваги і недоліки, які можна врахувати при розробці власного застосунку.

Існує деяка кількість наукових статей, які присвячені темі класифікації змій, вони базуються як на класичних алгоритмах ML, так і на нейронних мережах, а саме CNN.

Однією з таких є стаття [1] в якій представлені рішення з використанням алгоритмів KNN, SVM, з попереднім застосуванням PCA та спроби використати різні архітектури, такі як VGG16 та MobileNetV2, тобто такі, як можна використовувати на мобільних пристроях, але через порівняно невелику кількість параметрів, вони можуть давати меншу якість.

Інша стаття [2] розглядає лише нейромережевий підхід, фокусуючись на підборі архітектури та обробці вхідних даних, а саме очистці та аугментації, аби збільшити вибірку даних

Також, у вимірі цієї проблеми, окрім задач класифікації, розглядається і задача детекції на зображені і це має сенс у спільному використанні, бо локалізація шуканого об’єкту на зображені допоможе правильно класифікувати його, бо будуть усунуті ті дані, які не мають цінності у даній задачі.

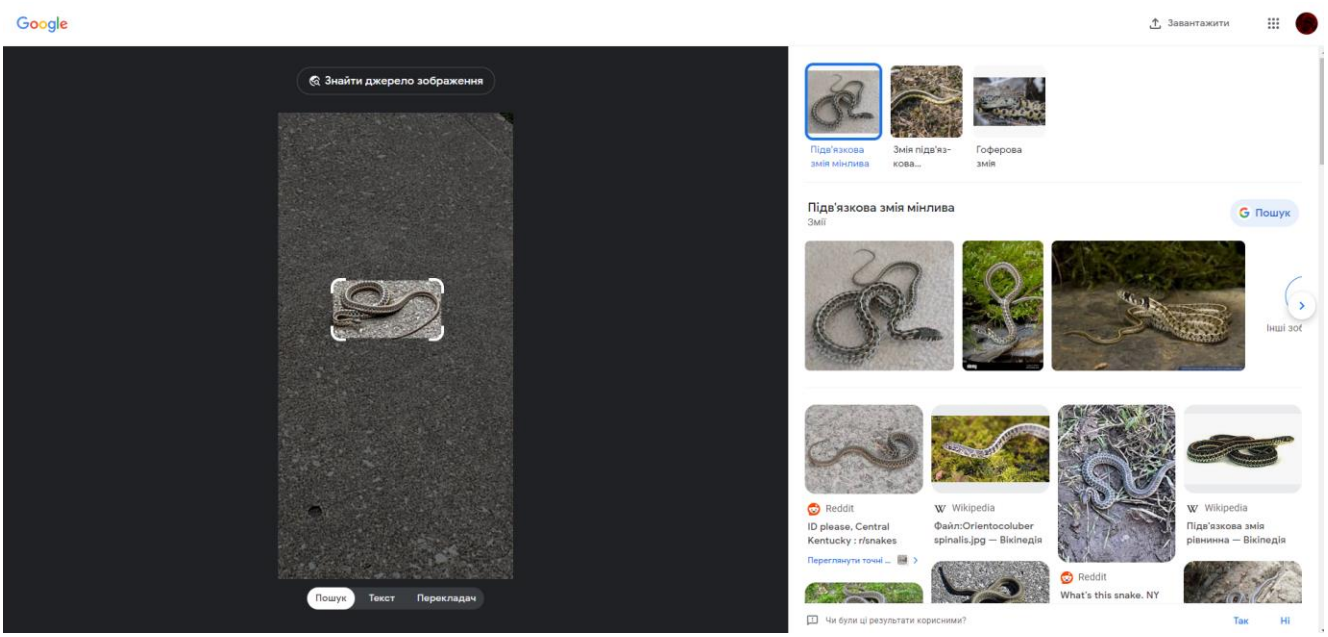


Рисунок 1.7 – Використання Google об’єктиву для класифікації змії по фото

Щодо комерційних розробок, то у відкритому доступі не було знайдено подібних сервісів, окрім Google об’єктиву (рис. 1.7), який вирішує більш широку задачу і не обмежується лише зміями і в цьому є головний його недолік, бо у разі розробки специфічної моделі виключно для цієї задачі, вона має показувати кращі результати.

Типовими рішеннями для класифікації зображень є архітектури VGGNet, ResNet, XceptionNet, MobileNet. Розглянемо кожну з них та підходи, які у них використовуються.

VGGNet (Visual Geometry Group Network) - це глибока згорткова нейронна мережа, яка була розроблена дослідниками з Visual Geometry Group в Університеті Оксфорду. Вона відома своєю простотою та ефективністю. Архітектура VGGNet включає кілька шарів згортки та пулінгу, а також кілька повнозв'язних шарів (рис. 1.8).

Основна ідея полягає в тому, щоб використовувати більше шарів з невеликою кількістю фільтрів у кожному з них. Наприклад, в оригінальній архітектурі VGGNet має 16-19 шарів, де кожен шар має 3x3 фільтри зі статичними пулінговими шарами

розміром 2x2. Це призводить до того, що мережа має дуже глибоку структуру зі зменшеною кількістю параметрів.

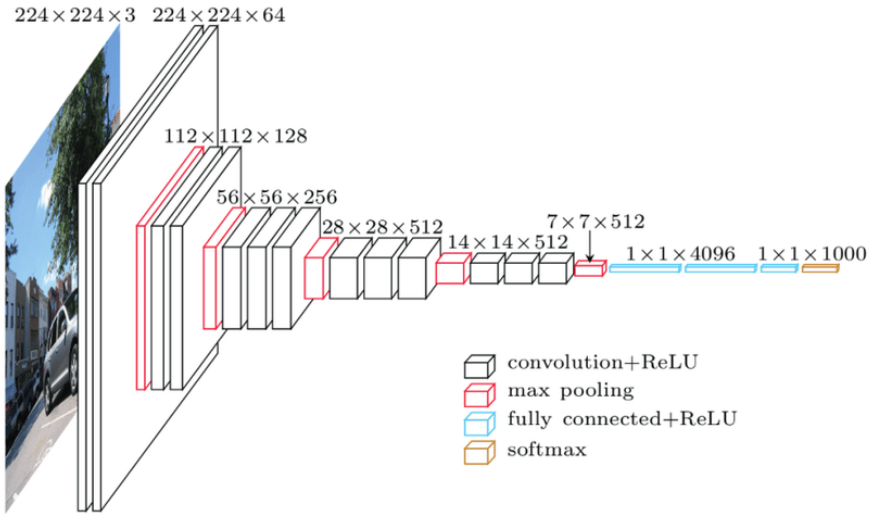


Рисунок 1.8 – VGGNet [13]

ResNet (Residual Neural Network) - це тип глибокої нейронної мережі, яка змінює стандартний підхід до глибокого навчання, дозволяючи тренувати нейронні мережі з більшою кількістю шарів без проблеми зникнення градієнту.

Основна ідея ResNet полягає у використанні "блоків залишкового навчання" (residual blocks), які додаються до вже існуючих функцій (функцій ідентифікації) входу до блоку. Це дозволяє нейронній мережі вчитися та пристосовуватися до нових функцій, замість того, щоб намагатися навчитися заново.

Коли вхідне зображення пройде через блок залишкового навчання, мережа вивчає трансформацію, яка буде залишати вхідні дані без змін (тобто, трансформація, що додається до входу, близька до тотожності). Це дозволяє зберігати інформацію про вхід та його градієнти, спрощуючи процес навчання глибоких мереж.

ResNet відома своєю здатністю тренувати нейронні мережі з сотнями, навіть тисячами шарів, що раніше було важко здійснити через проблему втрати градієнту. Це робить ResNet дуже популярним для завдань комп'ютерного зору, де глибокі мережі здатні досягати високих результатів у виявленні об'єктів та класифікації зображень.

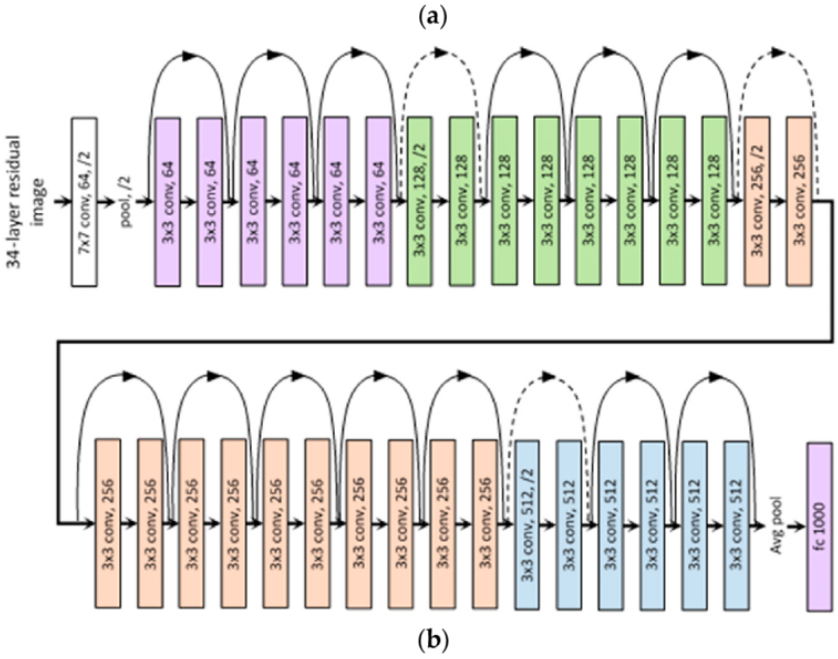


Рисунок 1.9 – ResNet [14]

MobileNet - це легка та ефективна архітектура глибокої нейронної мережі, спеціально розроблена для ресурсоємних пристроїв, таких як мобільні телефони та вбудовані системи. Вона була представлена компанією Google у 2017 році. Головною метою MobileNet є забезпечення високої точності класифікації зображень при мінімальному споживанні обчислювальних ресурсів та пам'яті.

Однією з ключових особливостей MobileNet є використання групових згорток (depthwise separable convolutions), які дозволяють розділити процес згортки на два окремих етапи: спочатку застосовується згортка на кожний канал вхідного зображення окремо, а потім застосовується 1x1 згортка для комбінування результатів. Цей підхід дозволяє значно зменшити кількість параметрів та обчислювальний витрати, що робить MobileNet ідеальним для використання на пристроях з обмеженими ресурсами.

MobileNet також використовує техніку глобальної середньої пулінгу (global average pooling), яка дозволяє зменшити розмір вихідних даних перед фінальним повнозв'язаним шаром. Це допомагає уникнути перенавчання та зменшити кількість параметрів у мережі.

Завдяки своїй легкості та ефективності, MobileNet стала популярним вибором для різних завдань комп'ютерного зору на мобільних пристроях, таких як класифікація зображень, виявлення об'єктів, а також використання у вбудованих системах та робототехніці.

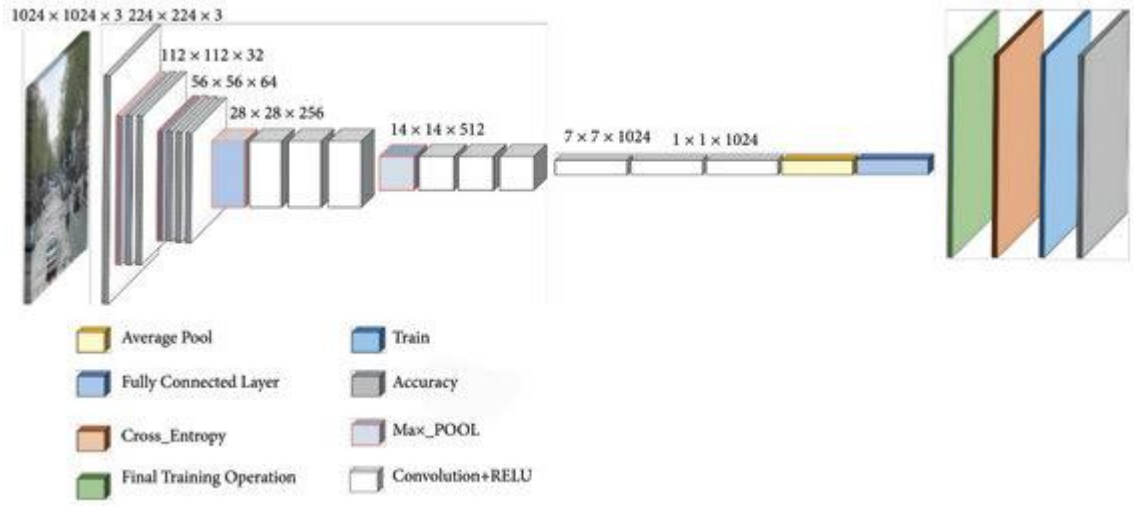


Рисунок 1.10 – MobileNet [15]

Xception (Extreme Inception) - це архітектура глибокої нейронної мережі, яка є еволюційним вдосконаленням архітектури Inception. Вона була представлена у 2016 році компанією Google Research. Основна ідея Xception полягає у використанні модуля "глибинної роздільної згортки" (depthwise separable convolution), який дозволяє здійснювати згортку та об'єднання каналів в окремих кроках.

У модулі глибинної роздільної згортки спершу застосовується згортка на кожний канал вхідного зображення окремо (глибинна згортка), а потім використовується 1x1 згортка для комбінування результатів (просторова згортка). Цей підхід дозволяє зменшити кількість параметрів та обчислювальні витрати, подібно до MobileNet, але в той же час зберігає ефективність та точність Inception.

Xception надає додаткову гнучкість у виборі глибини мережі та розміру фільтрів, що робить її придатною для різноманітних задач комп'ютерного зору.

Вона демонструє хорошу ефективність та точність на різних наборах даних та завданнях, включаючи класифікацію зображень та виявлення об'єктів.

Загалом, Xception є важливим кроком у розвитку архітектур глибоких нейронних мереж, який демонструє, що використання глибинної роздільної згортки може покращити ефективність та точність мережі, зберігаючи при цьому низькі обчислювальні витрати.

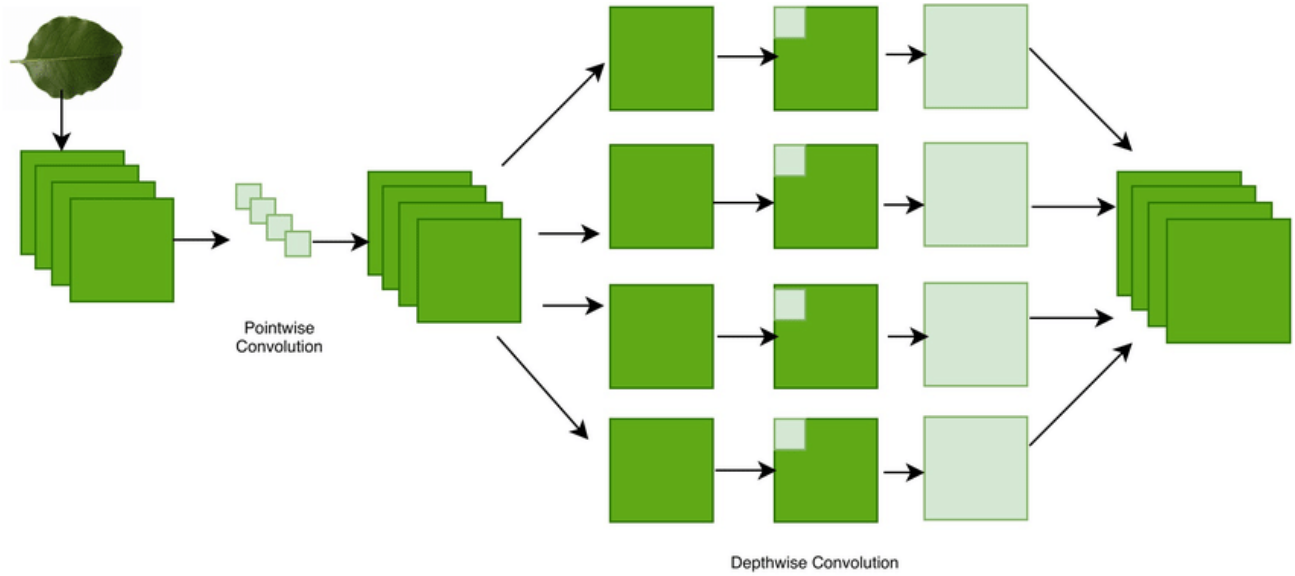


Рисунок 1.11 – XceptionNet [16]

Також хочеться приділити увагу способам регуляризації моделей. Регуляризація нейромереж – це набір методів, які використовуються для запобігання перенавчанню (overfitting) моделей машинного навчання, зокрема нейронних мереж. Перенавчання виникає, коли модель вивчає тренувальні дані занадто добре, включаючи шум і випадкові відхилення, що призводить до поганої узагальнюваності на нових, невідомих даних. Зазвичай у навчанні нейромереж використовують batch normalization. Це техніка, яка використовується для прискорення та стабілізації процесу навчання глибоких нейронних мереж. Вона була представлена в 2015 році Сергієм Іоффе та Крістіаном Шиллєром у їхній роботі "Batch Normalization: Accelerating Deep Network Training by Reducing Internal

Covariate Shift". Основна ідея полягає в нормалізації вхідних даних для кожного шару нейронної мережі.

Обчислення середнього та стандартного відхилення: Для кожного міні-батча вхідних даних обчислюється середнє значення та стандартне відхилення, а потім вхідні дані нормалізуються, використовуючи обчислені середнє та стандартне відхилення. Після нормалізації вводяться параметри масштабу і зміщення , які дозволяють моделі вивчати оптимальне масштабування та зсув нормалізованих даних. Ці параметри навчаються разом з іншими параметрами моделі.

Розвитком цієї технології стали інші техніки нормалізації, однією з яких є BCN (batch channel normalization), яка використовує нормування як за екземплярами батча, так і за каналами.

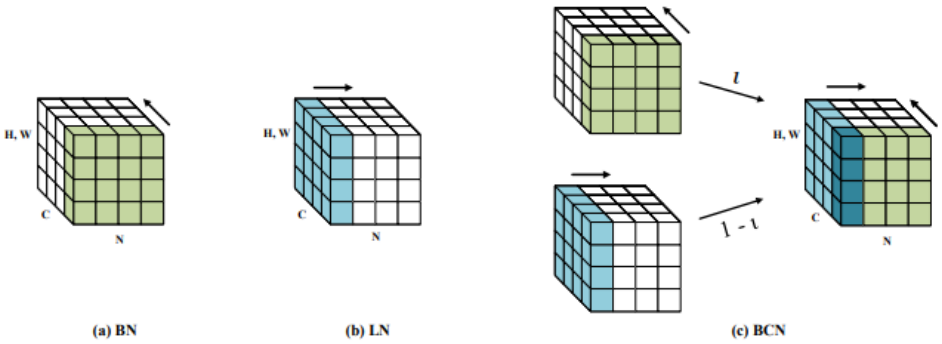


Рисунок 1.12 – Зображення роботи BCN [26]

Таким чином, існують параметри масштабування і зсуву вздовж осі екземплярів батча, так і за каналами. Експериментально було доведено, що така техніка покращує результати роботи моделі.

1.3 Постановка задачі

Отже, задачею є класифікація змій за наявним зображенням, ймовірно у поганій якості.

Саме зображення можна розділити на, безпосередньо, зображення та його метадані. Кожен з цих двох інформаційних об'єктів є носієм цінної інформації.

Лідером у класифікації зображень в останнє десятеліття є нейромережевий підхід, а саме згорткові нейромережі, бо вони забезпечують якість, що перевершує людину у класифікації зображень. Тож нема сенсу вибирати якийсь інший підхід для обробки цієї частини даних.

Тим не менш, вагома інформація може міститись і в метаданих, бо кожен вид змій має свій ареал проживання, тому, знаючи ймовірність зустріти певний вид в якийсь місцевості, можна врахувати за теоремою Баєса ймовірність того, що змія отруйна, по класифікації за фото та ймовірністю зустріти на певній території. Для цього потрібна окрема інформаційна система, вхідними даними якої будуть координати, у яких було зроблено фото, а результатом – ймовірності зустріти певний вид з переліку.

Далі, враховуючи результати роботи двох систем, вивести кінцеве рішення – ймовірність того, що певна змія є отруйною чи ні.

Враховуючи усе вищеперелічене, пропонується використати таку систему для вирішення задачі:

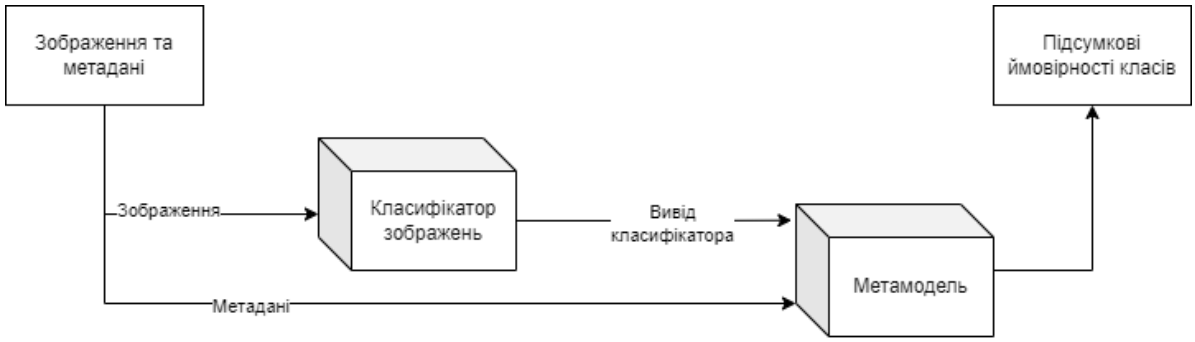


Рисунок 1.13 – Схема роботи системи

Також треба продумати про доступ до системи. Серед можливих варіантів є телеграм-бот, який забезпечить швидкий доступ до інструменту. Плюсами такого підходу є знайомість і зрозумілість інтерфейса користувачу. Мінусами в свою чергу є необхідність доступу до інтернету та залежність від роботи стороннього сервісу, яким є телеграм.

Іншим можливим варіантом є розробка власного мобільного застосунку. З переваг є можливість використовувати його локально, без доступу до мережі.

Однак серед недоліків є низька працездатність, бо використання нейромереж на мобільних пристроях може призвести до великих часових витрат або гіршої якості класифікації. Кожен з недоліків є неприпустимим для використання у вищеописаних ситуаціях, бо основними пріоритетами є точність і свобода доступу до технології.

Ще одним, найбільш привабливим, є варіант написання власного API, бо з одного боку, це дає універсальність, модульність, бо дає сервіс, для якого можуть бути написані різні клієнти, зокрема веб-, мобільна версії, телеграм бот. Єдиним недоліком можна вважати лише вже згадану необхідність доступу до мережі, але вирішення цієї проблеми призводить до створення ще більших проблем.

Саме ж API може складатись лише з одного-двох ендпоінта, які прийматимуть зображення, а повертатиме ймовірності приналежності до кожного класу.

Тож система має відповідати таким вимогам:

– мати API-інтерфейс для ефективної взаємодії з користувачами, використовуючи веб- чи мобільні клієнти, ботів у соціальних мережах тощо;

– бути системою, що приймає як зображення з метаданими, так і без них, аби обробляти випадки, коли метаданих нема чи вони є некоректними;

– вміти обробляти фотографії у поганій якості, тобто контрастності, роздільної здатності, чіткості, де змія займає меншу частину зображення;

– повертати ймовірність того, чи дана змія була отруйною.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було розглянуто сучасні методи і підходи у задача класифікації табличних даних, зображень, способи організації роботи декількох моделей.

Також сформовано вимоги до ПЗ, що містить загальний опис проєкту та його головних функцій, вимог до інформаційного та технічного забезпечення, інтерфейсів, а також властивостей програми.

Розроблено схему роботи системи.

2. ТЕХНОЛОГІЇ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Мова програмування Python

Python проста у використанні, та водночас повноцінна мова програмування, що надає набагато більше засобів для структурування і підтримки великих програм, ніж команди оболонки. З іншого боку, вона краще за C обробляє помилки, і, будучи мовою дуже високого рівня, має вбудовані типи даних високого рівня, такі як гнучкі масиви і словники, ефективна реалізація яких на C потребує значних витрат часу.

Python дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.

Python - інтерпретована мова, що дозволяє заощадити значну кількість часу, що зазвичай витрачається на компіляцію. Інтерпретатор можна використовувати інтерактивно, що дозволяє експериментувати з можливостями мови, писати шаблони програм або тестувати функції при розробці “знизу-вверх”. Він також зручний як настільний калькулятор. Python дозволяє писати дуже компактні й зручні для читання програми.

Програми, написані мовою Python, звичайно значно коротші еквівалента на C або C++ з декількох причин:

– типи даних високого рівня дозволять Вам виразити складні операції однією інструкцією;

– групування інструкцій виконується за допомогою відступів замість фігурних дужок.

Python розширювана мова: знання C дозволяє додавати нові функції, що вбудовуються, або модулі для виконання критичних операцій з максимальною

швидкістю або написання інтерфейсу до комерційних бібліотек, доступним тільки у двійковій формі.

Python має велику спільноту розробників, що активно підтримують мову і розробляють нові бібліотеки і інструменти. Ця спільнота дозволяє швидко реагувати на зміни в індустрії та впроваджувати нові ідеї в мову. Багато бібліотек і модулів для Python є відкритими і доступними для вільного використання, що сприяє розвитку екосистеми Python і забезпечує розробників потужними інструментами для різних завдань.

Наприклад, для наукових обчислень і обробки даних Python пропонує такі бібліотеки як NumPy, Pandas і SciPy. NumPy надає ефективні механізми для роботи з масивами даних і математичними функціями, що робить його ідеальним для наукових обчислень. Pandas забезпечує потужні інструменти для маніпулювання даними, у той час як SciPy пропонує алгоритми для наукових і технічних обчислень, включаючи чисельне інтегрування, оптимізацію, статистичний аналіз і багато іншого.

Однією з ключових особливостей Python є його підтримка різних парадигм програмування. Він підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє створювати структуровані і модульні програми за допомогою класів і об'єктів. Функціональне програмування в Python дозволяє використовувати функції як об'єкти першого класу, що дає можливість створювати високорівневі функції і здійснювати обробку даних безпосередньо в функціональному стилі.

Крім того, Python підтримує метапрограмування, що дає розробникам можливість змінювати свою програму під час виконання. Це забезпечує гнучкість в структуруванні програм і дозволяє створювати високорівневі абстракції для обробки складних завдань. Завдяки цим особливостям, Python є потужним і універсальним інструментом для розробки різноманітних програмних рішень.

2.2 Бібліотека Pytorch

PyTorch – це відкрите програмне забезпечення для машинного навчання і нейронних мереж, що розробляється командою Facebook. Основні переваги PyTorch включають зручний інтерфейс, заснований на Python, який спрощує розробку моделей і дослідження; динамічні обчислення, які дозволяють використовувати звичайний Python код при побудові нейронних мереж; та вбудовану підтримку автоматичного обчислення градієнтів, що спрощує процес навчання моделей.

PyTorch надає багато високорівневих модулів для побудови нейронних мереж, таких як torch.nn для визначення архітектури моделі, torch.optim для оптимізації параметрів моделі, і torch.utils для завантаження даних і навчання моделі. Він також підтримує структури даних, такі як тензори, що робить його ідеальним для роботи зі складними даними і великими обсягами інформації. Однією з ключових особливостей PyTorch є його підтримка GPU-пришвидшеного обчислення, що дозволяє виконувати обчислення на графічних процесорах для значного прискорення обчислень. Це робить PyTorch популярним інструментом для навчання нейронних мереж на великих обсягах даних.

Крім того, PyTorch відрізняється від інших фреймворків своєю "Pythonic" філософією, що дозволяє використовувати високорівневі структури даних Python, такі як списки і словники, без конвертації у специфічні для машинного навчання структури даних. Фреймворк підтримує широкий спектр функцій, включаючи обчислення градієнтів, обчислення втрат, автоматичне масштабування даних і обчислення, а також вбудовані оптимізатори для вибору швидкості навчання і моментуму. PyTorch забезпечує підтримку середовища інтеграції з облачними платформами і бібліотеками для обробки масивів, такими як NumPy. Фреймворк має велике співтовариство розробників і дослідників, що активно внесли свій внесок у підтримку та розвиток PyTorch.

Остання версія PyTorch, на час написання, підтримує розподілений тренування моделей і вбудовану підтримку для сервісних архітектур для

машинного навчання. PyTorch надає можливості для розгортання навчених моделей на продуктових серверах, що робить його ідеальним для реалізації та впровадження вирішень з машинного навчання. За допомогою PyTorch можна створювати інноваційні алгоритми машинного навчання і нейронних мереж, що застосовуються в різних сферах, включаючи комп'ютерне зору, обробку природної мови, автономне навчання, розпізнавання мови, рекомендаційні системи та інші.

PyTorch заснований на технологіях, які забезпечують підтримку великого обсягу обчислень та алгоритмів, що дозволяють вам працювати з великими наборами даних і забезпечувати високий рівень точності для ваших проектів машинного навчання. Цей фреймворк є одним з найпопулярніших в світі машинного навчання, завдяки своїм простоті, ефективності і масштабованості.

2.3 Бібліотека Scikit-learn

Scikit-learn бібліотека машинного навчання для Python, яка дозволяє використовувати різноманітні алгоритми і методи для розв'язання задач аналізу даних. Ця бібліотека надає зручний і простий у використанні API для виконання багатьох видів завдань машинного навчання, включаючи класифікацію, регресію, кластеризацію, відбір ознак і попередню обробку даних. Основні переваги Scikit-learn включають в себе високу ефективність, надійність і широкий вибір алгоритмів. Вона має у своєму арсеналі реалізації таких популярних методів, як лінійна і логістична регресія, метод опорних векторів (SVM), дерева рішень, випадкові ліси і багато інших.

Scikit-learn також пропонує інструменти для підготовки даних, таких як масштабування, нормалізація і обробка пропущених значень, що дозволяє легко підготувати дані для подальшого використання алгоритмів. Scikit-learn активно розвивається і підтримується великим співтовариством розробників. Вона часто оновлюється з новими функціями і покращеннями, що робить її відмінним вибором для науковців і професіоналів у сфері даних.

Ця бібліотека також інтегрована з іншими популярними інструментами аналізу даних в Python, такими як NumPy, SciPy і Pandas, що сприяє зручності

обробки та аналізу даних. Документація Scikit-learn є добре структурованою і містить велику кількість прикладів і викликів API, що полегшує навчання та впровадження алгоритмів машинного навчання.

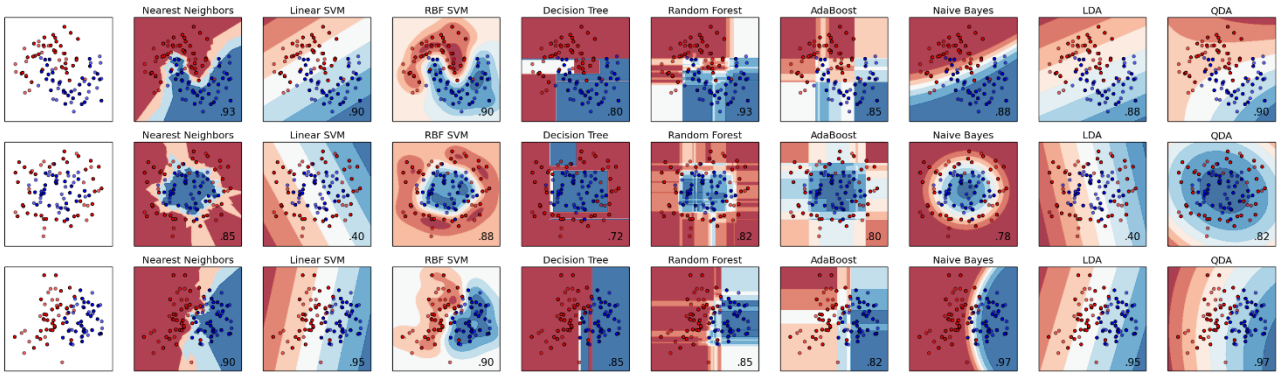


Рисунок 2.1– Візуалізація різних класифікаторів з scikit-learn на різних наборах даних [17]

Scikit-learn підтримує різні формати вхідних даних, що дозволяє легко і ефективно працювати з різними типами даних, включаючи структуровані та неструктуровані дані. Завдяки відкритому джерелу, Scikit-learn забезпечує високу ступінь гнучкості і налаштовуваності для потреб кожного конкретного проекту. Scikit-learn активно використовується у наукових дослідженнях, промислових проектах і освітніх програмах, що підтверджує її популярність та довіру в спільноті

Загалом, Scikit-learn є незамінним інструментом для всіх, хто працює з машинним навчанням і аналізом даних в Python, завдяки своїй потужності, простоті використання і широкому спектру функціональностей.

2.4 Бібліотека Flask

Flask - це легкий веб-фреймворк для Python, який дозволяє розробникам швидко створювати веб-додатки. Він став дуже популярним завдяки своїй простоті використання і гнучкості. Ось детальніше про ключові аспекти Flask:

Flask має просту структуру та легкий для розуміння API, що дозволяє швидко почати розробку. Немає складних конфігурацій або вимог до певної структури проекту.

Фреймворк підтримує модульне розподілення додатків. Розробники можуть розділяти свої додатки на невеликі модулі, що спрощує управління кодом та сприяє підтримці проектів різної складності. Flask має широкий вибір розширень (extensions), які додають додаткові функціональні можливості, такі як обробка форм, аутентифікація користувачів, інтеграція з базами даних тощо. Це дозволяє розробникам легко розширювати функціонал своїх додатків, не підвищуючи їхню складність. Flask інтегрується з багатьма стандартними бібліотеками і інструментами Python, такими як SQLAlchemy для роботи з базами даних, WTForms для обробки форм, Jinja2 для шаблонізації HTML і багатьма іншими.

Flask має велику і активну спільноту розробників, яка підтримує фреймворк, створює нові розширення і надає допомогу на форумах і у соціальних мережах. Це забезпечує швидке реагування на проблеми і підтримку нових функціональних можливостей. Завдяки своїй простоті та гнучкості, Flask є ідеальним вибором для початкових проектів, а також для більш складних додатків, які вимагають налаштовуваності і розширення функціональності.

Узагальнюючи, Flask - це потужний і простий у використанні фреймворк для розробки веб-додатків з Python, який забезпечує широкі можливості для створення сучасних веб-застосунків.

2.5 Сервіс Wandb

WandB (Weights and Biases) - це платформа для візуалізації та відстеження експериментів у глибокому навчанні та машинному навчанні. Основна мета WandB - надати розробникам і дослідникам потужні інструменти для аналізу, управління та збереження результатів їхніх моделей.

Платформа дозволяє легко візуалізувати метрики, архітектури моделей, графіки, зображення та багато іншого. WandB дозволяє створювати, відстежувати та порівнювати різні експерименти у вигляді таблиць та графіків. Інтеграція з такими бібліотеками, як TensorFlow, PyTorch, Keras, Scikit-learn та іншими, спрощує використання платформи. Результати експериментів можна легко зберігати та ділитися з колегами. WandB зберігає моделі, параметри та результати

експериментів, що дозволяє легко відновлювати їх. Результати експериментів можна аналізувати з різних кутів, використовуючи фільтри та групування. Можливість порівнювати результати різних моделей один з одним у вигляді порівняльних графіків спрощує аналіз. Система коментарів WandB дозволяє обмінюватися відгуками та враженнями з командою чи колегами. Підтримка багатьох типів моделей, від простих лінійних до складних нейронних мереж, робить платформу універсальною.

WandB має інтуїтивно зрозумілий і зручний інтерфейс, що спрощує навігацію та використання. Інтеграція з Jupyter та Google Colab дозволяє зручно використовувати платформу. Робота з числовими, категоріальними, текстовими та зображеннями даними є легкою завдяки підтримці WandB. Автоматичне відновлення результатів попередніх експериментів спрощує повторне використання даних. Збереження та відновлення всіх артефактів експерименту, включаючи дані, код та параметри моделі, забезпечує зручність управління проектами. Автоматизація процесу оновлення дозволяє швидко змінювати варіанти експериментів.

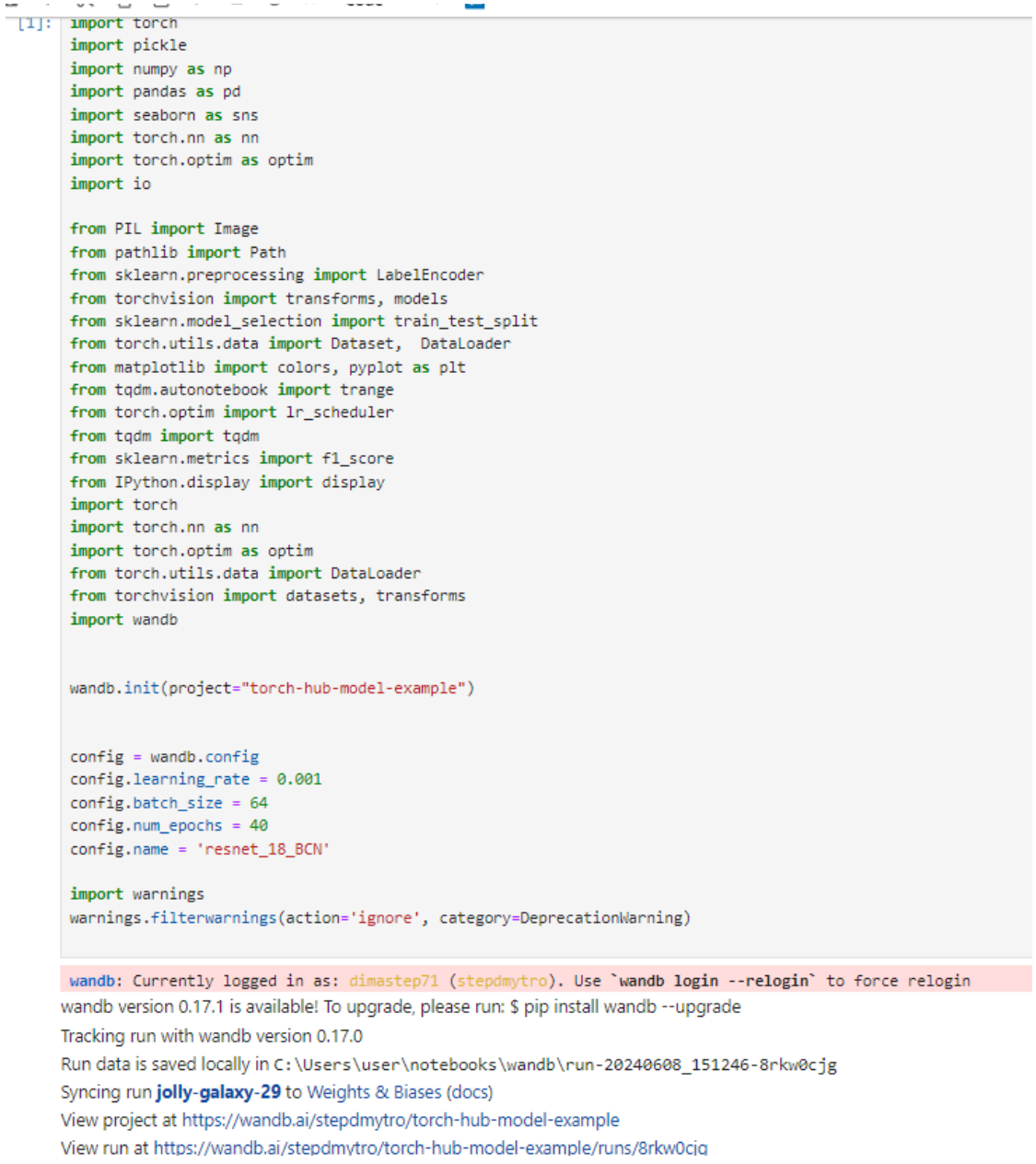
Висновки до розділу 2

Другий розділ кваліфікаційної роботи бакалавра було ретельно розглянуто і обґрунтовано вибір технологій для реалізації поставленої задачі, які технологія має перед своїми аналогами, описом функціоналу вибраних інструментів.

3. НАВЧАННЯ НЕЙРОМЕРЕЖ, ОЦІНКА РЕЗУЛЬТАТІВ НАВЧАННЯ

3.1 Програмна реалізація циклу навчання нейромереж

Для реалізації циклу навчання звичайні *.py файли не є зручними, оскільки кожна помилка перериває тривалий процес навчання моделі. Оптимальним інструментом для цього є *.ipynb ноутбуки, які дозволяють запускати окремі блоки коду та зберігати змінні, створені під час виконання. Такі ноутбуки задовольняють усі зазначені потреби та часто використовуються у комерційних і наукових ML-проєктах для дослідницького аналізу даних.



```
[1]: import torch
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
import torch.nn as nn
import torch.optim as optim
import io

from PIL import Image
from pathlib import Path
from sklearn.preprocessing import LabelEncoder
from torchvision import transforms, models
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset,  DataLoader
from matplotlib import colors, pyplot as plt
from tqdm.autonotebook import trange
from torch.optim import lr_scheduler
from tqdm import tqdm
from sklearn.metrics import f1_score
from IPython.display import display
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import wandb


wandb.init(project="torch-hub-model-example")


config = wandb.config
config.learning_rate = 0.001
config.batch_size = 64
config.num_epochs = 40
config.name = 'resnet_18_BCN'

import warnings
warnings.filterwarnings(action='ignore', category=DeprecationWarning)
```

```
wandb: Currently logged in as: dimastep71 (stepdmytro). Use `wandb login --relogin` to force relogin
wandb version 0.17.1 is available! To upgrade, please run: $ pip install wandb --upgrade
Tracking run with wandb version 0.17.0
Run data is saved locally in C:\Users\user\notebooks\wandb\run-20240608_151246-8rkw0cjg
Syncing run jolly-galaxy-29 to Weights & Biases (docs)
View project at https://wandb.ai/stepdmytro/torch-hub-model-example
View run at https://wandb.ai/stepdmytro/torch-hub-model-example/runs/8rkw0cjg
```

Рисунок 3.1 – Секція коду і результат його виконання у *.ipynb ноутбуці

У першій секції коду імпортуються всі необхідні бібліотеки, встановлюється контакт з сервісом wandb та задаються константи проєкту, які налаштовуються через wandb.config, аби їх значення були доступні під час перегляду результатів навчання моделі.

[2]:
```
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[3]:
```
class SnakeImageDataset(Dataset):
    def __init__(self, dir, img_df, transform=None):
        self.img_info = img_df
        self.img_dir = Path(dir)
        self.transform = transform

    def __len__(self):
        return len(self.img_info)

    def __getitem__(self, idx):
        row = self.img_info.iloc[idx]
        img_path = self.img_dir/str(row['class_id'])/(str(row['UUID'])+".jpg")
        image = Image.open(img_path)
        label = row['poisonous'], row['binomial']
        if self.transform:
            image = self.transform(image)
        return image, label
```

[4]:
```
df= pd.read_csv("Csv/train.csv", index_col="Unnamed: 0")
print(df.columns)
ds = SnakeImageDataset('train', df)
image, label = ds[123]
display(image)
display(label)
```

```
Index(['binomial', 'country', 'continent', 'genus', 'family', 'UUID',
       'class_id', 'snake_sub_family', 'poisonous', 'X', 'Y', 'height',
       'width'],
      dtype='object')
```

Рисунок 3.2 – Клас датасету

Далі було написано та протестовано клас датасету, який, як і очікувалось, надав як табличну інформацію, так і саме зображення змії. Це дозволяє зручно працювати з даними та візуалізувати їх на різних етапах.

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])


train_info = pd.read_csv("Csv/train.csv", index_col="Unnamed: 0")#.iloc[165:205]
test_info = pd.read_csv("Csv/test.csv", index_col="Unnamed: 0")#.iloc[0:50]

unique_classes = train_info['binomial'].unique()
class_to_idx = {cls: idx for idx, cls in enumerate(unique_classes)}
idx_to_class = {idx: cls for cls, idx in class_to_idx.items()}

train_info['binomial'] = train_info['binomial'].map(class_to_idx)
test_info['binomial'] = test_info['binomial'].map(class_to_idx)

train_dataset = SnakeImageDataset('train', train_info, transform)
test_dataset = SnakeImageDataset('test', test_info, transform)

train_loader = DataLoader(dataset=train_dataset, batch_size=config.batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=config.batch_size, shuffle=False)
```

```
run = 0
run_path = Path('runs')

from resnet import resnet18

model = resnet18(pretrained=False, num_classes=len(unique_classes))
run_path
try:
    with open('run.txt', 'r') as file:
        data = file.read()
        run = int(data)
        state_dict = torch.load(run_path/str(run)/f"best_model_test_{config.name}.pth")
        model.load_state_dict(state_dict)
except:
    ...
run+=1
with open('run.txt', 'w') as file:
    (run_path/str(run)).mkdir(parents=True, exist_ok=True)
    file.write(str(run))
run_path.mkdir(parents=True, exist_ok=True)
model = model.to(DEVICE)
```

Рисунок 3.3 – Ініціалізація змінних

Наступним важливим етапом є ініціалізація всіх змінних, необхідних для процесу навчання моделі, зокрема: завантажувачі даних, сама модель, змінні для зберігання ваг моделі. Також реалізовано логіку, яка завантажує ваги, якщо такі

збережено у файловій системі, щоб продовжити навчання з останнього збереженого стану.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=config.learning_rate)


best_train_accuracy = 0.0
best_test_accuracy = 0.0
for epoch in range(config.num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for images, labels in tqdm(train_loader, desc='Training model...'):
        images = images.to(DEVICE)
        labels = labels[1].to(DEVICE)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = 100 * correct / total

    wandb.log({"Epoch": epoch + 1, "Loss": epoch_loss, "Accuracy": epoch_accuracy})

    print(f"Epoch [{epoch+1}/{config.num_epochs}], Loss: {epoch_loss:.4f}, Accuracy: {epoch_accuracy:.2f}%")


    torch.save(model.state_dict(), run_path/str(run)/f'last_model_{config.name}.pth')

    if epoch_accuracy > best_train_accuracy:
        best_train_accuracy = epoch_accuracy
        torch.save(model.state_dict(), run_path/str(run)/f'best_model_train_{config.name}.pth')
        print(f'Best model saved with accuracy: {best_train_accuracy:.2f}%')

    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in tqdm(test_loader, desc='Testing model...'):
            images = images.to(DEVICE)
            labels = labels[1].to(DEVICE)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        test_accuracy = 100 * correct / total
        wandb.log({"Test Accuracy": test_accuracy})
        if test_accuracy > best_test_accuracy:
            best_test_accuracy = test_accuracy
            torch.save(model.state_dict(), run_path/str(run)/f'best_model_test_{config.name}.pth')
            print(f'Best model saved with test accuracy: {best_test_accuracy:.2f}%')

    print(f'Test Accuracy: {test_accuracy:.2f}%')
```

Рисунок 3.4 – Основний цикл навчання

Центральною частиною коду є основний цикл навчання. Фреймворк PyTorch надає велику гнучкість у розробці рішень, але через це багато операцій треба

прописувати вручну. Основна частина циклу повторюється задану кількість епох, якщо не задано критерій завчасної зупинки процесу навчання моделі. Кожна ітерація по епохах складається з двох частин – навчальної та тестувальної.

У навчальній частині алгоритм отримує батч даних (набір фіксованого розміру) та передає їх на вхід моделі. Вихід моделі порівнюється з дійсним значенням цільової змінної. Алгоритмом зворотного поширення помилки розраховуються градієнти, які використовує оптимізатор для коригування значень ваг моделі. Тут існує деяка неакуратність, оскільки останній батч менший за всі попередні, а значить значимість об’єктів останнього батчу при коригуванні ваг більша за інші, хоча це можна вважати статистичною похибкою. Проте існують способи уникнення цієї ситуації. Також особливістю є те, що модель повертає ймовірності, тож треба вибрати значення, яке модель вважає найімовірнішим. Для цього просто вибирається значення з найбільшою ймовірністю серед усіх класів. Такий підхід не буде правильним, якщо необхідно визначати декілька (верхні н) варіантів. Також розраховується метрика на тестових даних. Вона не оптимізується у процесі навчання. Оптимізується функція втрат, яка має корелювати з метрикою.

У тестувальній частині так само використовуються батчі даних, які передаються у модель, але вже без розрахунку градієнтів. Враховується метрика якості, у даному випадку багатокласової класифікації була використана метрика Accuracy, яка іноді не є репрезентативною, але у рамках даної роботи є достатньою.

Через вибагливість деяких моделей до апаратного забезпечення (відеокарти), на якому проводиться навчання, було створено модифікацію ноутбуку з використанням Google Colaboratory. Це дозволяє використовувати потужніші обчислювальні ресурси для ефективного навчання моделі.

```
!pip install wandb

!pip install -q kaggle

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d goelyash/165-different-snakes-species

!unzip 165-different-snakes-species.zip -d snakes_data
```

Рисунок 3.5 – Додаткові команди для запуску середовища

Перш за все, треба завантажити і правильно розмістити дані на віртуальній обчислювальній машині. Для цього використано наведені на рисунку команди у клітинці *.ipynb ноутбука.

Іншою різницею між локальним запуском ноутбука та запуску за допомогою google colaboratory є необхідність під’єднувати хмарний диск для збереження даних після відключення сеансу роботи ноутбука.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    root = Path("/content/snakes_data")

    Mounted at /content/drive
```

Рисунок 3.6 – Підв’язка хмарного сервісу гугл

Наведеним вище способом під’єднується віртуальна машина colaboratory до хмарного сервісу.

Окремо варто зазначити реалізацію мереж ResNet18 та ResNet34 з блоками batch channel normalization. Оригінальну реалізацію, що використана в дослідженнях, знайти не вдалось, але, завдяки опису роботи, цей результат є відтворювальним.

```
class BatchChannelNorm(nn.Module):
    def __init__(self, num_channels, epsilon=1e-5, momentum=0.9):
        super(BatchChannelNorm, self).__init__()
        self.num_channels = num_channels
        self.epsilon = epsilon
        self.momentum = momentum
        self.Batchh = BatchNormm2D(self.num_channels, epsilon=self.epsilon)
        self.layeer = LayerNormm2D(self.num_channels, epsilon=self.epsilon)
        # The BCN variable to be learnt
        self.BCN_var = nn.Parameter(torch.ones(self.num_channels))
        # Gamma and Beta for rescaling
        self.gamma = nn.Parameter(torch.ones(num_channels))
        self.beta = nn.Parameter(torch.zeros(num_channels))

    def forward(self, x):
        X = self.Batchh(x)
        Y = self.layeer(x)
        out = self.BCN_var.view([1, self.num_channels, 1, 1]) * X + (
                1 - self.BCN_var.view([1, self.num_channels, 1, 1])) * Y
        out = self.gamma.view([1, self.num_channels, 1, 1]) * out + self.beta.view([1, self.num_channels, 1, 1])
        return out
```

Рисунок 3.7 – Підв’язка хмарного сервісу гугл

Основним недоліком даної реалізації є мова програмування, на якій написаний цей компонент. Використання Python для реалізації цього блоку має надзвичайно сильний негативний вплив на швидкість навчання нейронних мереж. Це пов'язано з тим, що Python, хоч і є зручним для написання коду та має велику кількість бібліотек для машинного навчання, все ж таки є інтерпретованою мовою програмування. Через це виникають проблеми з продуктивністю, особливо при роботі з великими об'ємами даних або складними обчисленнями, які часто зустрічаються у процесі навчання нейронних мереж.

3.2 Аналіз результатів навчання моделей

Результати роботи коду можна переглянути на сайті wandb.

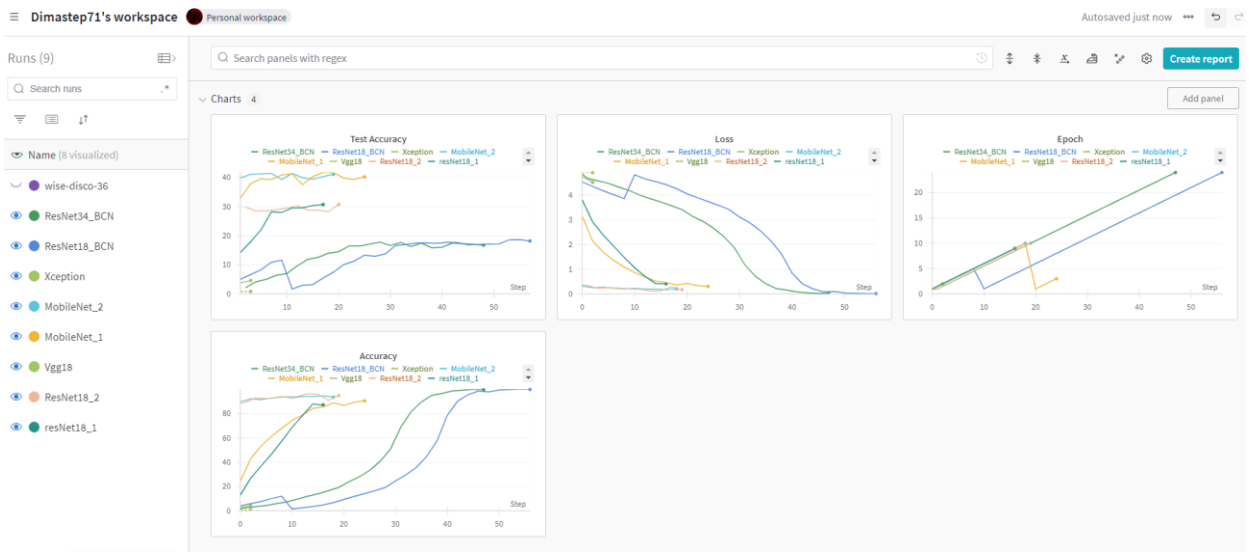


Рисунок 3.8 – Результати навчання моделей

У ході навчання моделей було використано моделі VggNet11, ResNet18, XCeptionNet, MobileNet18, ResNet18 з BCN шаром, ResNet34 з BCN шаром.

Через особливості коду, деякі запуски навчання моделі були розділені на два графіки, у інших же випадках в межах одного графік зберігаються дані про декілька послідовних запусків процесу навчання моделі.

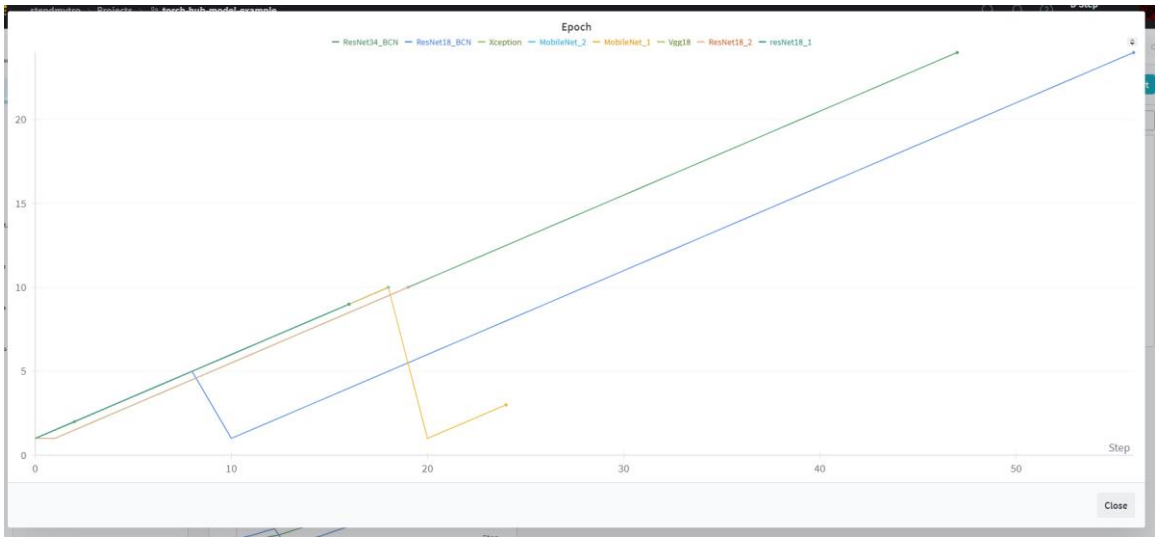


Рисунок 3.9 – Графіки номери епохи до номеру відправленого пакету в рамках однієї сесії

Можна помітити, що у двох випадках графік опускається і номер епохи починає рахуватись з нуля, а потім рости. Ця особливість не впливає на кінцевий результат, але може заплутати пересічного читача.

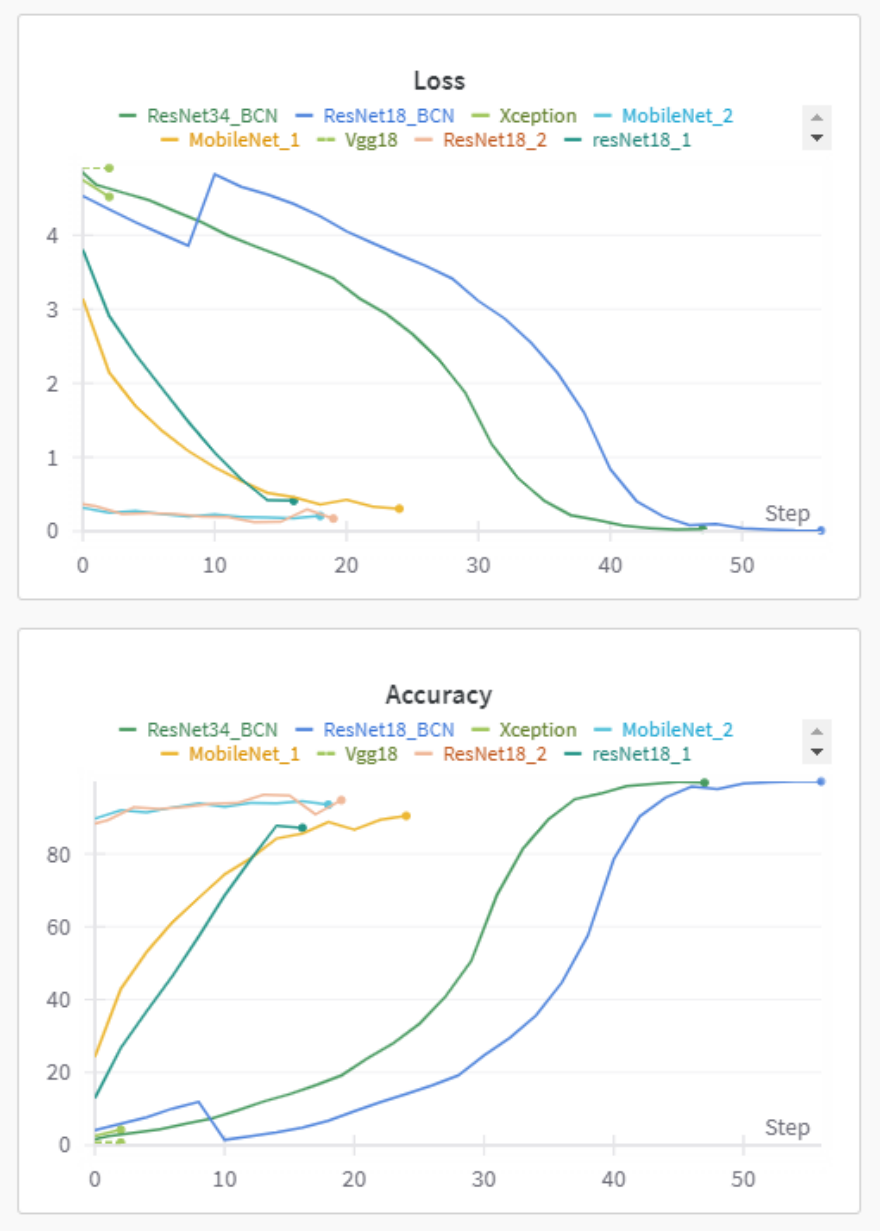


Рисунок 3.10 – Графіки значень функції втрат та точності до номеру відправленого пакету в рамках однієї сесії

Є сенс розглянути метрику якості моделі та значення функції втрат на кожній ітерації. Спостерігається сильний негативний зв’язок між цими величинами: зі зниженням функції втрат, точність на тренувальному наборі зростає. Також можна побачити різке падіння значень функції втрат на 30-40 ітераціях. Це явище

пов’язане з динамічною зміною швидкості навчання, що дозволяє більш точно знайти точку мінімуму функції, яка оптимізується.

За результатами навчання моделі XCeptionNet та VggNet11 не показали гарних результатів. Більше того, час навчання цих моделей перевищив увесь доступний час використання апаратного забезпечення. Таким чином, основними моделями, серед яких обирається найкраща, є MobileNet, ResNet18, ResNet18 BCN та ResNet34 BCN.

Точність на тренувальному наборі даних найбільша у ResNet18 BCN і дорівнює 99.8%. Проте, якщо розглянути тестувальний набір даних і функцію втрат, результати дещо змінюються. Це важливо для оцінки справжньої ефективності моделей, оскільки високі показники на тренувальних даних не завжди гарантують хорошу продуктивність на невидимих тестових даних.

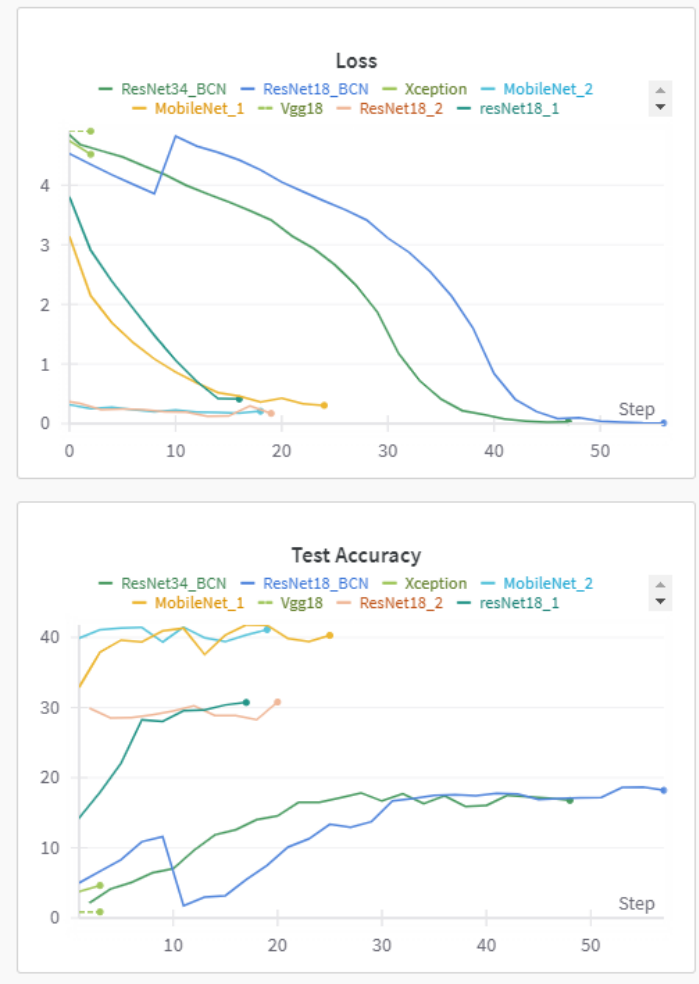


Рисунок 3.11 – Графіки значень функції втрат та точності на тестовому до номеру відправленого пакету в рамках однієї сесії

Результативність на тестовому наборі даних для всіх моделей є нижчою, ніж на тренувальному, але ступінь цього зниження значно варіюється для кожної моделі. Очевидно, що складність різних моделей напряму впливає на їх результативність: модель, яка має складні нелінійні зв'язки та більшу кількість параметрів (ваг), здатна відтворювати складніші взаємозв'язки у даних. Таким чином, можна бачити, що кожна модель має своє певне місце у певному діапазоні якості та рухається всередині цього діапазону.

Найвищі показники якості класифікації демонструє модель MobileNet, яка досягла точності 41.4% для багатокласової класифікації. Цей результат свідчить про те, що модель MobileNet краще справляється з виявленням складних взаємозв'язків у даних у порівнянні з іншими моделями. Водночас, більш прості моделі, які мають менше параметрів і менш складні нелінійні зв'язки, не здатні так ефективно і точно класифікувати дані, що й призводить до нижчих показників результативності.

Таким чином, видно, що модель MobileNet, завдяки своїй складності і більшій кількості параметрів, змогла досягти кращих результатів на тестовому наборі даних, порівняно з іншими моделями, і займає найвищу позицію у рейтингу за показниками якості класифікації.

Висновки до розділу 3

У третьому розділі КРБ було розроблено код для навчання нейромережевих моделей для вирішення задачі класифікації з логуванням результатів роботи програми.

За результатами навчання моделей та порівняння метрик якості, було проаналізовано та визначено модель, яка буде використовуватись у підсумковій версії програмного забезпечення.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ, РОЗГОРТКА МОДЕЛІ

4.1 Реалізація API для доступу до моделі

Для розгортки моделі машинного навчання було створено веб-додаток, розроблений за допомогою Flask, який дозволяє завантажувати зображення, обробляти їх за допомогою моделі глибокого навчання для класифікації і повертати результати у форматі JSON.

model_server.py × telegram_client.py

```
1  import pickle
2  import uuid
3  from datetime import datetime
4
5  from flask import Flask, request, jsonify
6  from PIL import Image
7  import os
8  import torch
9  from torchvision import transforms, models
10 import torch.nn as nn
```

Рисунок 4.1 – Імпорт необхідних бібліотек

Розглянемо бібліотеки, використані у проєкті та спосіб їх використання у проєкті.

Імпорт необхідних модулів:

– uuid для генерації унікальних ідентифікаторів файлів;

– datetime для роботи з датою і часом;

– torch і torch.nn для роботи з машинним навчанням і нейронними мережами;

– Flask, request, jsonify для створення API;

– PIL.Image для роботи зображеннями;

– os для роботи з операційною системою.

```
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'}
```

Рисунок 4.2 – Налаштування застосунку

Ініціалізація Flask застосунку з назвою модуля, налаштування папки для завантаження файлів (uploads) і дозволених розширень файлів (png, jpg, jpeg, gif).

```
if not os.path.exists(app.config['UPLOAD_FOLDER']):
    os.makedirs(app.config['UPLOAD_FOLDER'])
```

Рисунок 4.3 – Налаштування директорій

Перевірка і створення папки uploads, якщо вона ще не існує.

```
get_area_probs = get_probs_gen('points.csv')
```

Рисунок 4.4 – Створення екземпляру функції для визначення ймовірностей за геолокацією

Отримання ймовірностей для областей з файлу 'points.csv'.

```
1 usage
def allowed_file(filename):
    return ('.' in filename and
            filename.rsplit('.', 1)[1].lower()
            in app.config['ALLOWED_EXTENSIONS'])
```

Рисунок 4.5 – Функція перевірки вхідних даних.

Функція allowed_file, яка перевіряє, чи є тип файлу дозволеним за його розширенням.

```
def log_prediction(filename, top_classes, top_probs):
  with open("model_log.txt", "a") as f:
    f.write(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}, "
            f"{filename}, {top_classes}, {top_probs}\n")
```

Рисунок 4.6 – Функція логування

Функція log_prediction, яка записує результати передбачення в файл model_log.txt, включаючи ім'я файлу, топ-класи та їх ймовірності.

```
@app.route( rule: '/upload/', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    if file and allowed_file(file.filename):
        ext = file.filename.rsplit( sep: '.', maxsplit: 1)[1].lower()
        new_filename = f"{uuid.uuid4().hex}_{datetime.now().strftime('%Y%m%d%H%M%S')}.{ext}"
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], new_filename)
        file.save(filepath)

        try:
            image = Image.open(filepath)

            coords = get_coords(image)

            image = transform(image)
            image = image.unsqueeze(0)
```

Рисунок 4.7 – Обробка ендпоінту

Маршрут /upload/, який обробляє POST-запит для завантаження файлу. Він перевіряє наявність файлу у запиті, дозволяє лише дозволені типи файлів, генерує унікальне ім'я для файлу, зберігає файл у папку uploads, обробляє зображення, визначає ймовірності класів за допомогою моделі, коригує їх враховуючи

координати, сортує за ймовірністю, логує результати і повертає топ-3 результати разом з ім'ям файлу.

```
            with torch.no_grad():
                output = model(image)
                output = nn.functional.softmax(output, dim=1)
                probs = output.squeeze().tolist()
                classes = [idx_to_class[idx] for idx in range(len(probs))]
                results = [{"name": name, "prob": prob * 100} for name, prob in zip(classes, probs)]
                if coords is not None:
                    lat, lng = coords
                    area_probs = get_area_probs(lat, lng)
                    sum_ = 0
                    for result in results:
                        result['prob'] = result['prob'] * area_probs[result['name']]
                        sum_ += result['prob']
                    for result in results:
                        result['prob'] /= sum_/100
                results.sort(key=lambda x: x['prob'], reverse=True)
                log_prediction(new_filename, classes[:3], probs[:3])
                response = jsonify({"top_results": results[:3], "filename": new_filename})
            return response
        finally:
            if os.path.exists(filepath):
                os.remove(filepath)
    else:
        return jsonify({"error": "File type not allowed"}), 400


> if __name__ == '__main__':
    app.run(debug=True)
```

Рисунок 4.8 – Розрахунок ймовірностей та запуск застосунку

Запуск додатку Flask у режимі debug при запуску основного модуля.

4.2 Розгортка моделей машинного навчання та розробка допоміжних інструментів

У цьому файлі описані усі основні інтелектуальні системи, що використовуються у проєкті. Розглянемо детальніше:

```
import pickle
import torch
from torchvision import transforms, models
import torch.nn as nn
from PIL import ExifTags
import pandas as pd
from sklearn.metrics.pairwise import haversine_distances
from math import radians
```

Рисунок 4.9 – Імпорт необхідних бібліотек

Необхідні бібліотеки:

– pickle для серіалізації та десеріалізації об'єктів;
– torch для роботи з PyTorch;
– torchvision.transforms для трансформацій зображень;
– torchvision.models для попередньо натренованих моделей;
– torch.nn для роботи з нейронними мережами;
– PIL.ExifTags для роботи з EXIF-даними зображень;
– pandas для роботи з даними у вигляді таблиць;
– sklearn.metrics.pairwise.haversine_distances для обчислення Гаверсинової відстані;
– math.radians для перетворення градусів у радіани.

```
with open('idx_to_class.pkl', 'rb') as f:
    idx_to_class = pickle.load(f)

transform = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
        transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])
    ])
```

Рисунок 4.10 – Створення об’єктів-обробників даних

Завантаження мапи індексів у класи з файлу idx_to_class.pkl за допомогою pickle. Визначення трансформацій для зображень:

– зміна розміру до 256x256;

– перетворення зображення в тензор;

– нормалізація тензора.

```
def create_model(classes):
    model = models.mobilenet_v3_small(pretrained=True)
    model.classifier[3] = nn.Linear(model.classifier[3].in_features, classes)
    model.load_state_dict(torch.load("best_model_test_mobilenet_v3_small.pth"))
    model.eval()
    return model


model = create_model(len(idx_to_class))
```

Рисунок 4.11 – Створення моделі, завантаження вагів

Функція create_model, яка створює і завантажує модель MobileNetV3:

– завантаження попередньо натренованої моделі mobilenet_v3_small;

– заміна останнього шару на новий, відповідний кількості класів;

– завантаження попередньо збережених ваг моделі з файлу best_model_test_mobilenet_v3_small.pth;

– перемикання моделі в режим оцінки (eval).

Створення моделі за допомогою функції create_model і встановлення кількості класів рівною довжині мапи idx_to_class.

```python
def get_coords(image):
    def decimal_coords(coords, ref):
        decimal_degrees = float(coords[0]) + float(coords[1]) / 60 + float(coords[2]) / 3600
        if ref == "S" or ref == 'W':
            decimal_degrees = -1 * decimal_degrees
        return decimal_degrees

    try:
        GPSINFO_TAG = next(
            tag for tag, name in ExifTags.TAGS.items() if name == "GPSInfo"
        )

        info = image.getexif()

        gpsinfo = info.get_ifd(GPSINFO_TAG)

        image_lat = decimal_coords(gpsinfo[2], gpsinfo[1])
        image_lng = decimal_coords(gpsinfo[4], gpsinfo[3])
        return image_lat, image_lng
    except:
        return None
```

Рисунок 4.12 – Функція визначення координат за метаданими

Функція get_coords, яка отримує координати зображення з його EXIF-даних:

– перетворення градусів, хвилин і секунд у десяткові градуси;
– витягування даних GPS з EXIF;
– обчислення широти і довготи в десяткових градусах;
– повернення широти і довготи або None, якщо GPS-дані недоступні.

```python
def haversine(lat1, lng1, lat2, lng2):
    dot_1_radians = [radians(_) for _ in (lat1, lng1)]
    dot_2_radians = [radians(_) for _ in (lat2, lng2)]
    result = haversine_distances([dot_1_radians, dot_2_radians])
    return result[0, 1] * 6371000/1000
```

Рисунок 4.13 – Функція розрахунку відстаней

Функція haversine для обчислення Гаверсинової відстані між двома наборами координат (в кілометрах).

```
def get_probs_gen(source):
    data = pd.read_csv(source)

    data['lat'] = pd.to_numeric(data['lat'], errors='coerce')
    data['lng'] = pd.to_numeric(data['lng'], errors='coerce')

    def inner(image_lat, image_lng):
        distances = data.apply(lambda row: haversine(image_lat, image_lng, row['lat'], row['lng']), axis=1)
        print(type(distances))
        return data.iloc[distances.idxmin()]
    return inner
```

Рисунок 4.14 – Функція пошуку найближчої точки у наборі даних

Функція get_probs_gen, яка генерує функцію для обчислення ймовірностей областей:

– завантаження даних з файлу CSV (source);

– перетворення колонок lat і lng на числові значення;

– внутрішня функція, яка обчислює Гаверсинову відстань між координатами зображення і кожним рядком даних, повертає рядок з мінімальною відстанню.

Отже, реалізовано усі необхідні алгоритми, усі моделі розгорнуто та готове до обробки запитів.

4.3 Розробка клієнта

Клієнтом буде telegram-bot, який буде відповідати на повідомлення, яке містить фотографію,повідомленням, яке буде використовувати дані від сервера з розгорнутою на ньому моделлю. Розглянемо код, який реалізовує дану логіку:

```python
import asyncio
import logging
import sys
import os
import uuid
from datetime import datetime

import requests

from aiogram import Bot, Dispatcher, F
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.types import Message

from dotenv import load_dotenv

load_dotenv()

TOKEN = os.getenv("TOKEN")
endpoint = 'http://127.0.0.1:5000/upload/'
```

Рисунок 4.15 – Імпорт необхідних бібліотек, визначення констант

Імпорт бібліотек та визначення констант застосунку:

Завантаження змінних середовища з файлу .env за допомогою load_dotenv, отримання токена бота з середовища (з файлу .env).

Визначення кінцевої точки (endpoint) для завантаження файлів на локальний сервер.

```python
bot = Bot(token=TOKEN,
          default=DefaultBotProperties(
              parse_mode=ParseMode.HTML
          )
)
dp = Dispatcher()
```

Рисунок 4.16 – Створення диспетчера та ініціалізація бота

Ініціалізація бота Bot з параметром parse_mode=ParseMode.HTML для підтримки HTML-розмітки в повідомленнях та диспетчера Dispatcher для обробки повідомлень.

```python
@dp.message(F.photo)
async def file_handler(message: Message):
    file_id = message.document.file_id
    file = await bot.get_file(file_id)
    downloaded_file = await bot.download_file(file.file_path)

    file_name = f"{uuid.uuid4().hex}_{datetime.now().strftime('%Y%m%d%H%M%S')}_{file.file_path.split('/')[-1]}"
    try:
        with open(file_name, 'wb') as new_file:
            new_file.write(downloaded_file.read())

        with open(file_name, 'rb') as img_file:
            files = {'file': img_file}
            response = requests.post(endpoint, files=files)
            data = response.json()
            try:
                top_results = data.get('top_results', [])
                answer = "\n".join(
                    f"This is {item['name']} with probability {item['prob']:.2f}%"
                    for item in top_results
                )
                await message.reply(answer)
            except Exception as ex:
                print(ex)
                await message.reply(str(ex))
    finally:
        if os.path.exists(file_name):
            os.remove(file_name)
```

Рисунок 4.17 – Обробка сценарію надсилання фото

Декоратор @dp.message(F.photo) визначає функцію-обробник для повідомлень, що містять фотографії:

– отримання file_id з повідомлення;
– отримання інформації про файл за допомогою bot.get_file(file_id);
– завантаження файлу за допомогою bot.download_file(file.file_path);
– генерація унікального імені для файлу з допомогою uuid та datetime;
– збереження файлу на диск;

– відправка файлу на сервер за допомогою HTTP-запиту requests.post на кінцеву точку endpoint;

– обробка відповіді сервера та відправка користувачу результатів передбачення у відповідь на повідомлення;

– видалення файлу з диску після обробки.

```
async def main() -> None:
    await dp.start_polling(bot)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    asyncio.run(main())
```

Рисунок 4.18 – Запуск роботи бота

Асинхронна функція main, яка запускає диспетчер для обробки повідомлень.

Основний блок програми: налаштування базового рівня ведення журналу, запуск асинхронного основного циклу програми.

Отже, була розроблена остання частина системи для визначення виду змії за фотографією.

4.4 Тестування роботи систем

Як вже було описано, вхідними даними системи є фотографія змії, а вихідним значенням є ймовірності, з якими змія належить до певних класів змій.

Для тестування системи і перевірки якості моделі було використано дані з тестового датасету, тож можна перевірити достовірність передбачень моделі. При режимі роботи, в якому вона повертає декілька (у даному випадку 3) найймовірніших видів, результативність моделі (ймовірність, що вона вкаже одним з пунктів правильний вид) є суттєво вищою, ніж вимірювана метрика точності (повного співпадіння) , що використовувалась під час навчання моделі.

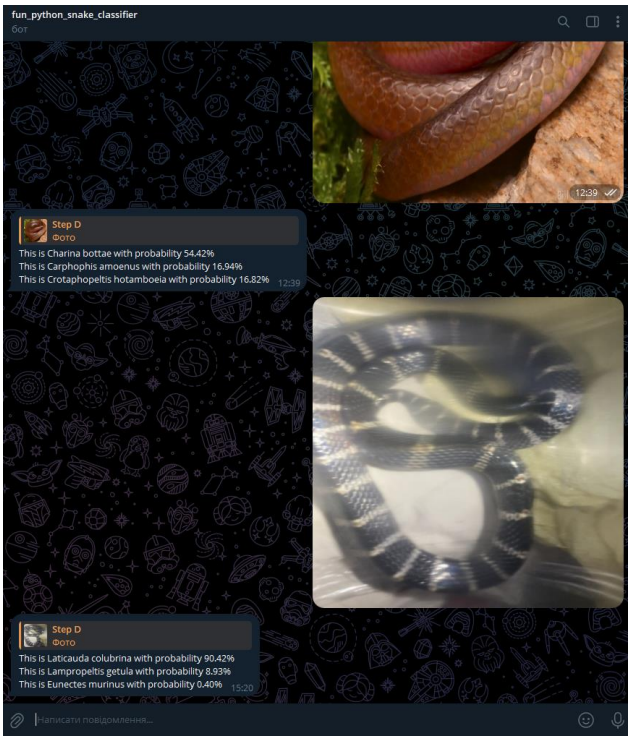


Рисунок 4.19 – Скриншот роботи системи

Отже, функційно система є повністю розробленою та завершеною. Є механізм обробки виключень для випадків, коли зображення не містить географічної прив’язки чи зберігається у специфічному форматі, який не містить в собі метаданих.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи бакалавра розроблено програмну реалізацію основних компонентів системи, включаючи серверну та клієнтську частини. Серверна частина було реалізовано через створення класу, який обробляє запити від клієнтів передаючи дані до моделі і повертаючи результат роботи моделі.

Створено клієнтську частину у вигляді телеграм-бота, який обробляє надсилання зображень та відповідає, виводячи ймовірності приналежності змії до певного класу.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було розроблено програмне забезпечення класифікації змій на базі інструментарію штучного інтелекту для оперативного визначення виду змій за зображенням та метаданими зображення.

Мета роботи досягнута. Поставлені завдання виконані, а саме:

– досліджено предметну область, сучасні підходи до вирішення задачі, проведено аналіз наявних аналогів, їх переваг і недоліків;

– сформовано основні вимоги до програмного забезпечення;

– розроблено схему взаємодії різних моделей машинного навчання;

– організовано навчання нейронних мереж та проаналізовано результати їхнього навчання;

– розроблено інші системи для роботи з метаданими зображень;

– розроблено код серверної частини системи;

– розроблено код клієнтської частини застосунку.

У ході виконання кваліфікаційної роботи було досліджено предметну галузь, що підтвердило актуальність створення класифікатора зображень змій.

Також було проведено аналіз наявних рішень, їх функціональних можливостей, оцінка доступності сервісів.

Було сформовано вимоги до програмного забезпечення та здійснено проєктування застосунку з урахуванням обраного стеку технологій, що, завдяки модульній структурі застосунку, забезпечує масштабованість продукту.

Визначено стек технологій необхідних для реалізації ПЗ, а саме вебфреймворк Flask, фреймворки машинного навчання scikit-learn та pytorch, фреймворк для роботи з telegram API aiogram.

Також було визначено архітекртури нейронних мереж, які будуть використовуватись для класифікації зображення, а саме ResNet, VggNet, MobileNet, XCeptionNet. Було проведено процедуру навчання моделей машинного навчання та

проаналізовано результати навчання, за результатом якого було обрано фінальну модель для використання.

Було розроблено систему, яка визначає географічні координати, у яких було зроблено знімок за допомогою екстракції метаданих, порівнює з наявними даними у пошуках найближчої точки та повертає ймовірність зустріти конкретний вид у даній місцевості щоб скорегувати отримані від класифікатора зображень дані. Також було розроблено вебсервер, який використовує вищеописані моделі та клієнт у вигляді бота, який приймає на вхід зображення і повертає ймовірності приналежності вмісту зображення до певного виду змій. Розроблену систему було протестовано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Snake species classification using deep learning techniques. URL: https://link.springer.com/article/10.1007/s11042-023-16773-0 (дата звернення: 10.05.2024).

2. A comparative study on image-based snake identification using machine learning. URL: https://www.nature.com/articles/s41598-021-96031-1 (дата звернення: 09.05.2024).

3. Deep Residual Learning for Image Recognition. URL: https://arxiv.org/abs/1512.03385 (дата звернення: 11.05.2024).

4. Very Deep Convolutional Networks for Large-Scale Image Recognition. URL: https://arxiv.org/abs/1409.1556 (дата звернення: 05.05.2024).

5. Xception: Deep Learning with Depthwise Separable Convolutions. URL: https://arxiv.org/abs/1610.02357 (дата звернення: 07.05.2024).

6. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. URL: https://arxiv.org/abs/1704.04861 (дата звернення: 08.05.2024).

7. Аналіз і оцінка ризиків у бізнесі URL: https://financial.lnu.edu.ua/wp-content/uploads/2018/09/NMK_L_5_IT.pdf (дата звернення: 07.05.2024).

8. Штучний інтелект: основні поняття та методології. URL: https://clicknoob.fun/technologies/shtuchnyj-intelekt-osnovni-ponyattya-ta-metodologiyi/ (дата звернення: 20.05.2024).

9. Convolutional Neural Networks as a modern approach in AI. URL: https://svitla.com/blog/convolutional-neural-networks-as-a-modern-approach-in-ai (дата звернення: 22.05.2024).

10. NARX neural prediction of oscillationalinstability at the IBR-2M reactor. URL: https://www.researchgate.net/figure/The-equivalent-of-a-NARX-neuromorphic-cell-The-classifiers-output-is-re-applied_fig3_365395813 (дата звернення: 21.05.2024).

11. Prediction of Degraded Infrastructure Conditions for Railway Operation. URL:

https://www.mdpi.com/1424-8220/24/8/2456 (дата звернення: 25.05.2024).

12. 10 Must-Know Models for ML Beginners: Gradient Boosting. URL: https://medium.com/@weidagang/gradient-boosting-from-scratch-84921eb64c3c (дата звернення: 16.05.2024).

13. Video Summarization using Keyframe Extraction and Video Skimming. URL: https://www.researchgate.net/figure/Sample-CNN-Architecture-for-VSUMM-and-RESNET16_fig2_336510662 (дата звернення: 5.06.2024).

14. Basic Intuitive Explanation of Resnet Architecture. URL: https://medium.com/@Saad_Alam/basic-intuitive-explanation-of-resnet-architecture-1d3f17c19289 (дата звернення: 09.06.2024).

15. Efficient Approach towards Detection and Identification of Copy Move and Image Splicing Forgeries Using Mask R-CNN with MobileNet V1. URL: https://www.researchgate.net/figure/The-architecture-of-MobileNet-V1-30_fig5_357623745 (дата звернення: 4.06.2024).

16. DeepHerb: A Vision Based System for Medicinal Plants Using Xception Features. URL: https://www.researchgate.net/figure/Concept-of-Xception-architecture_fig2_354937942 (дата звернення: 10.06.2024).

17. Scikit-learn Classification Algorithms. URL: https://www.datasciencecentral.com/scikit-learn-classification-algorithms/ (дата звернення: 22.05.2024).

18. Learn PyTorch for Deep Learning: Zero to Mastery book. URL: https://www.learnpytorch.io/ (дата звернення: 27.05.2024).

19. The Flask Mega-Tutorial. URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (дата звернення: 25.05.2024).

20. Aiogram documentation. URL: https://docs.aiogram.dev/en/latest/ (дата звернення: 29.05.2024).

21. Python documentation. URL: https://docs.python.org (дата звернення: 29.05.2024).

22. Alex Krizhevskym, Ilya Sutskever, Geoffrey E. Hinton. ImageNet

Classification with Depp Convolutional Neural Networks: стаття. Toronto: University of Toronto, 2012. 9 с. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (дата звернення: 14.05.2024).

23. Richard Szeliski. Computer Vision: Algorithms and Applications: електронна книга. Springer, 2010. 979 с. URL: https://www.cs.ccu.edu.tw/~damon/tmp/SzeliskiBook_20100903_draft.pdf (дата звернення: 19.05.2024).

24. Colaboratory. Google: вебсайт. URL: https://research.google.com/colaboratory/faq.html (дата звернення: 26.05.2024).

25. Machine Learning in Python. Scikit-learn: вебсайт. URL: https://scikitlearn.org/stable (дата звернення: 20.05.2024).

26. BCN: Batch Channel Normalization for Image Classification. URL: https://arxiv.org/pdf/2312.00596 (дата звернення: 26.01.2024).