Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Тесленка Святослава Ігоровича*
	(ПІБ)
академічної групи	*122М-20-2*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Розробка інформаційно-аналітичної системи для визначення географічних координат об'єктів на зображенні за допомогою багатозадачних згорткових нейронних мереж*

_____*С.І. Тесленко*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Мороз Б.І.*			
економічний	*Доц. Касьяненко Л.В.*			

Рецензент				

Нормоконтролер	*Доц. Реута О.В.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 ____ року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-20-2* *Тесленку Святославу Ігоровичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка інформаційно-аналітичної системи для визначення географічних координат об'єктів на зображенні за допомогою багатозадачних згорткових нейронних мереж*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. № 1036-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес визначення координат об'єктів, які присутні на зображенні.

Предмет досліджень – математичні моделі та методи аналізу зображень за допомогою нейронних мереж для визначення географічних координат об'єктів.

Мета роботи – дослідження методу для визначення географічних координат об'єктів на зображенні.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження, вебзастосунок для візуального відображення результату.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в розробці та застосуванні сучасних методів комп'ютерного зору і програмного забезпечення для моделювання та навчання нейронних мереж в задачах класифікації регіонів земної кулі, що дозволяє знаходити географічні координати об'єктів.

Практична цінність полягає в обґрунтуванні та розв'язанні проблем медіакриміналістики, що дає можливість більш точно, автоматизовано і зручно знаходити певні об'єкти, які були зображені на різноманітних вхідних зображеннях. Також дуже важливим є використання нового способу дроблення географічної кулі на клітини.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє візуалізувати та оцінити безпосереднє використання представленої системи. В результаті роботи повинна бути розроблена інформаційно-аналітична система, у вигляді веб застосунку, яка дозволяє завантажити зображення і отримати на карті найбільш ймовірні локації об'єктів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз теми та постановка задачі	12.09.2021-30.09.2021
Розробка моделі, методу та програмного забезпечення визначення координат об'єктів на фото	01.10.2021-30.11.2021
Використання програми та аналіз отриманих результатів	01.11.2021-18.12.2021

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки розробці алгоритму та програмного забезпечення для визначення координат об'єктів на фото.

Соціальний ефект від реалізації результатів роботи очікується позитивним, так як завдяки цьому буде полегшена робота медіакриміналістів та журналістів.

Завдання видав _____ *Мороз Б.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Тесленко С.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *10.09.2021 р.*

Термін подання кваліфікаційної роботи до ЕК _____

РЕФЕРАТ

Пояснювальна записка: 89 стор., 35 рис., 3 додатки, 23 джерела.

Об’єкт досліджень: процес визначення координат об’єктів, які присутні на зображенні.

Предмет дослідження: математичні моделі та методи аналізу зображень за допомогою нейронних мереж для визначення географічних координат об’єктів.

Мета магістерської роботи: дослідження методу для визначення географічних координат об'єктів на зображенні.

Методи дослідження. Для вирішення поставлених задач використані методи: чисельні методи в інформатиці, метод градієнтного спуску, згорткові нейронні мережі.

Новизна отриманих результатів дипломної роботи визначається тим, що полягає в розробці та застосуванні сучасних методів комп’ютерного зору і програмного забезпечення для моделювання та навчання нейронних мереж в задачах класифікації регіонів земної кулі, що дозволяє знаходити географічні координати об’єктів.

Практична цінність результатів полягає в обґрунтуванні та розв’язанні проблем медіакриміналістики, що дає можливість більш точно, автоматизовано і зручно знаходити певні об’єкти, які були зображені на різноманітних вхідних зображеннях. Також дуже важливим є використання нового способу дроблення географічної земної кулі на клітини.

Область застосування. Розроблене програмне забезпечення може застосовуватися для вирішення проблем медіакриміналістики, таких як маніпулювання змістом та метаданими.

Значення роботи та висновки. Розроблена система дозволяє ефективно знаходити цільові об’єкти на зображенні зі значним скороченням матеріальних та часових витрат, підвищити точність визначення їх координат.

Прогнози щодо розвитку досліджень. Покращити архітектуру мережі та зібрати набір даних який охопить більшу кількість країн. Можливого покращення метрик можна також досягти провівши підбір гіперпараметрів.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: глибоке навчання, класифікація, згорткові нейронні мережі, pytorch, багатозадачне навчання, крива заповнення простору, оцінка геолокації.

ABSTRACT

Explanatory note: 89 pages, 35 figures, 3 applications, 23 sources.

Object of research: the process of estimating the coordinates of the objects that are present in the image.

Subject of research: mathematical models and methods of image analysis using neural networks to determine the geographical coordinates of objects.

The purpose of the master's work: research of the method for estimating the geographical coordinates of objects in the image.

Research methods. To solve the problems used methods: numerical methods in computer science, the method of gradient descent, convolutional neural networks.

The novelty of the results of the thesis is determined by the fact that it is the development and application of modern methods of computer vision and software for modelling and training of neural networks in the classification of regions of the globe, which allows finding geographical coordinates.

The practical value of the results lies in the substantiation and solution of problems of media forensics, which allows more accurate, automated and convenient to find certain objects that have been depicted in a variety of input images. It is also very important how we use a new method of fragmenting the globe into cells.

Scope. The developed software can be used to solve problems of media forensics, such as content and metadata manipulation.

The value of the work and conclusions. The developed system allows you to effectively find target objects in the image with a significant reduction in resources costs, increase the accuracy of estimating their coordinates.

Further research and development. Improve the architecture of the network and gather a data set that covers more countries. It is possible to achieve improvement of the metrics by hyperparameters tuning process.

In the section "Economics" calculations of the complexity of software development, the cost of creating software and the duration of its development, as well as marketing research of the market for the software product.

Keyword list: deep learning, classification, convolutional neural networks, pytorch, multitask learning, space filling curve, geo-location estimation.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CNN – Convolutional Neural Network;

ЗНМ – Згорткові нейронні мережі;

ADAM – метод Adaptive Moment Estimation;

ГС – метод градієнтного спуску;

Pr(y|x) – умовна ймовірність випадкової події y при умові події x;

ЧМІ – чисельні методи у інформатиці;

$E_x[f(x)]$ – математичне сподівання f відносно x;

MTNN – multi-task neural network;

.py – формат файлу мови програмування Python.

ЗМІСТ

ВСТУП

Оцінка географічного розташування об’єкта на зображенні без попереднього знання – дуже складне завдання, оскільки існує величезна різноманітність можливих знімків, зроблених по всій земній кулі. Різний час доби, розташування об’єктів або параметри камери ще більше ускладнюють цю задачу. Крім того, зображення часто неоднозначні і тому дають дуже мало візуальних підказок щодо відповідного місця розташування об’єкта. З цих причин більшість існуючих підходів спрощують гео-локалізацію об’єктів, обмежуючи проблему вузькою підмножиною зображень, наприклад, відомих пам’яток і міст або природних зон, таких як ліси чи гори. Лише деякі системи розглядають загальне завдання, не покладаючись на конкретні зображення чи будь-які інші попередні припущення. Такі підходи особливо виграють від сучасного прогресу в глибокому навчанні та зростаючої кількості загальнодоступних наборів даних та зображень із географічними мітками в таких сервісах, як Instagram, Facebook, Imgur, Pinterest.

Через складність проблем і незбалансований розподіл місць, з яких було зроблено фотографію, на земній кулі, методи на основі згорткових нейронних мереж (CNN) добре працюють при розгляданні геолокалізації об’єктів на зображенні як завдання класифікації, розділяючи Землю на географічні клітинки з однаковою мінімальною кількістю зображень. Однак навіть сучасні CNN не в змозі запам’ятати візуальний вигляд всіх частин Землі і в той же час вивчити завдання визначення типу сцени зображення. Підходи до географічного поділу землі на клітини породжують проблему компромісу. У той час як детальне розподілення призводить до більш високої точності в конкретному міському масштабі (похибка менше 1 км), ширший розподіл комірок підвищує продуктивність у міжнародному масштабі (прибл. 1000 км). З іншого боку, природні типи сцен, такі як гори та поляни або зображення в приміщенні, швидше за все, визначаються ознаками, що кодуються рослинами та тваринами, або типом інтер’єру відповідно. Таким чином, можна стверджувати, що гео-

локалізація об’єктів на зображенні може значно виграти від додаткового знання опису сцени зображення, оскільки розбіжність у вибірках даних може бути значно зменшена разом із присутнім там непотрібним шумом.

Дану систему можна використовувати для вирішення проблем криміналістичної експертизи соціальних медіа, таких як маніпулювання змістом і метаданими. Оскільки зараз, завдяки поширенню соціальних медіаплатформ, медіа дані, які поширюються в Інтернеті, можуть охопити мільйони людей за дуже короткий час. В той же час, люди можуть легко поділитися підробленими фотографіями для зловмисних цілей, таких як створення паніки або маніпуляція громадською думкою, без особливих зусиль.

Тому, метою роботи є створення системи, яка підвищить ефективність визначення географічних координат об'єктів на зображенні.

Для досягнення поставленої мети необхідно вирішити наступні основні завдання:

– дослідити методи, які дозволяють визначати географічні координати об’єктів на фото;

– розробити програмне забезпечення, яке знаходить ймовірності локацій об’єктів на фото;

– зробити аналіз залежності отриманого рішення та розрахувати основні метрики навчання.

В ПЗ необхідно реалізувати:

1) простий та зрозумілій інтерфейс;
2) зручний спосіб завантаження зображення та введення даних;
3) візуалізацію отриманого рішення;
4) перевірку вхідних даних на їх коректність;
5) обробку виняткових ситуацій.

Якість навчання нейронної мережі характеризується наступними показниками:

- значення метрики F1, точність, повнота на тестовому наборі даних;
- присутністю перенавчання;

- збіжності функції втрат;
- щільністю дискретизації області.

Вихідною інформацією програми є вектор розрахованих ймовірностей для представлених географічних клітин.

Вся вхідна інформація повинна перевірятися на коректність та відповідність очікуваному типу.

Кваліфікаційна робота складається з трьох розділів. У першому розділі був проведений аналіз предметної області, описані основні ідеї моделювання архітектури нейронних мереж та використані набори даних. У другому розділі розглянуті методи навчання та оцінки. Третій розділ демонструє роботу програми і опис її роботи.

РОЗДІЛ 1
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

Наскільки важко визначити GPS-координати об'єктів на зображенні? Хоча сама проблема здається надзвичайно складною, зображення часто містять достатньо інформації, як-от орієнтири, текстура хмари, тип трави, дорожні знаки або архітектурні особливості, які дозволяють припустити місце, де було зроблено фотографію. Раніше така проблема вирішувалась методами пошуку зображень або зчитуванням доступних метаданих. Проблему поставлена як завдання класифікації, поділяючи поверхню Землі на географічні клітини за допомогою спеціальної кривої заповнення простору. Для навчання моделі були використані тисячі географічних регіонів різного масштабу. У цій роботі представлено декілька методів глибокого навчання, які використовують передові підходи роботи з географічними даними та переваги багатозадачного навчання. Пропонується брати до уваги зміст сцени зображення, тобто де було зняте зображення зовні, всередині, в місцях дикої природи чи міської обстановки тощо. В результаті додаткова інформація з різними просторовими роздільними можливостями, а також більш специфічні особливості для різних середовищ включаються в процес навчання згорткової нейронної мережі. Отримані значення цільових метрик демонструють ефективність запропонованого готового підходу, використовуючи допоміжну мережу для об’єднання двох наборів даних, виконується визначення міток типу сцени на наборі даних де доступні лише GPS координати. Розроблена модель не спирається на методи пошуку [1][2][3], які вимагають величезної обчислювальної потужності, і реалізує імовірнісний підхід.

До розповсюдження великої кількості ресурсів в мережі інтернет, один з найпростіших способів, який можна було використовувати для визначення місця розташування об'єкта, - це пошук схожих зображень в Google або TinEye. Якщо

знайти назву об'єкта, буде нескладно дізнатися назву місця, де він знаходиться. Але цей спосіб спрацьовує не завжди. Тому можна спробувати інші методи. Наприклад, перевірити те, чого не видно на фотографії, але що може дати важливе розуміння про самому зображенні.

У метаданих, а також EXIF-даних можна, серед іншого, знайти:

- дату і час створення зображення;
- дані про гео-локацію;
- модель камери та параметри знімку(діафрагма, витримка і т.д.);
- інформацію про власника знімка.

Це може бути корисним при перевірці двох аспектів: місця і часу створення фотографії, а також того, чи було зображення змінено і яким чином.

Якраз інформація про геолокацію (якщо вона буде в метаданих) може допомогти з граничною точністю встановити місце зйомки. Але в той же час наявність даних про гео-локацію залежить від декількох факторів. По-перше, від пристрою, яким була зроблена фотографія. У деяких камерах або мобільних пристроях може не бути GPS-датчика, який фіксує [4][5] координати. По-друге, від бажання користувачів мобільних пристроїв - вони можуть відключити гео-локацію через міркування приватності або зменшення навантажень на акумулятор. По-третє, наявність таких даних залежить від ресурсу, на якому фотографія була опублікована. Соціальні мережі Facebook, Twitter або Instagram видаляють метадані з самих фотографій під час їх завантаження на сервери цих ресурсів.

1.2. Глибинне навчання

Машинне навчання (англ. Machine learning) – це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово покращувати продуктивність у певній задачі) з даних без явного програмування. Еволюціювавши з досліджень розпізнавання образів та теорії обчислювального

навчання в галузі штучного інтелекту, машинне навчання досліджує вивчення та побудову алгоритмів, які можуть навчатися й робити прогнозування, опираючись на наявні дані – такі алгоритми долають слідування строго статичним програмним інструкціям, здійснюючи керовані даними прогнози або ухвалювання рішень шляхом побудови моделі з вибіркових входів. Машинне навчання застосовують у ряді обчислювальних задач, у яких розробка та програмування явних алгоритмів з доброю продуктивністю є складною або нездійсненною; до прикладів таких додатків належать фільтрування електронної пошти, виявляння мережних хакерів або зловмисних інсайдерів, що добиваються витоку даних, оптичне розпізнавання символів (ОРС), навчання ранжуванню та комп'ютерний зір. [6][7]

Згорткові нейронні мережі – це клас глибинних штучних нейронних мереж прямого поширення, який застосовується до аналізу зображень. ЗНМ використовують різновид багатошарових перцептронів, розроблений так, щоб вимагати використання мінімального обробки. Виходячи з їхньої архітектури спільних ваг та характеристик інваріантності відносно паралельного перенесення. ЗНМ використовують порівняно мало попередньої обробки, в порівнянні з іншими алгоритмами класифікації зображень. Це означає, що мережа навчається за допомогою фільтрів, що в традиційних алгоритмах приходиться розробляти вручну. Ця незалежність у конструюванні ознак від апріорних знань та людських зусиль є великою перевагою. ЗНМ складається з шарів входу та виходу, а також із декількох прихованих шарів. Приховані шари ЗНМ зазвичай складаються зі згорткових шарів, агрегуючих шарів, повнозв'язних шарів та шарів нормалізації. [8]

Згорткові шари застосовують до входу операцію згортки, передаючи результат до наступного шару. Згортка імітує реакцію окремого нейрону на зоровий стимул. Хоч повнозв'язні нейронні мережі прямого поширення можливо застосовувати, як для навчання ознак, так і для класифікації даних, застосування цієї архітектури до зображень є непрактичним.[9] Було б необхідним дуже велике число нейронів, навіть у поверхневій (протилежній до глибинної)

архітектурі, через дуже великі розміри входу, пов'язані з зображеннями, де кожен піксель є відповідною змінною. Наприклад, повнозв'язний шар для(маленького) зображення розміром 100 × 100 має 10 000 параметрів. Операція згортки дає змогу розв'язати цю проблему, оскільки вона зменшує кількість вільних параметрів, дозволяючи мережі бути глибшою за меншої кількості параметрів. Наприклад, незалежно від розміру зображення, області замощування розміру 5 × 5, кожна з одними й тими ж спільними вагами, вимагають лише 25 вільних параметрів. Таким чином, це розв'язує проблему зникання або вибуху градієнтів у тренуванні традиційних багатошарових нейронних мереж з багатьма шарами за допомогою зворотного поширення. На рис. 1.1. зображено такий шар нейронної мережі.

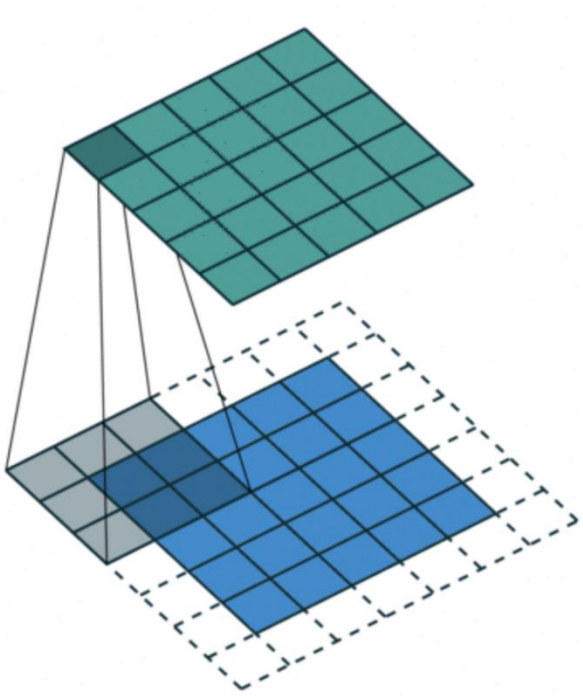

Рис. 1.1. Візуалізація операції згортки

Згорткові мережі можуть включати шари локального або глобального агрегування, які об'єднують виходи кластерів нейронів одного шару до наступного шару. Наприклад, агрегування за максимумом використовує максимальне значення з кожного з кластерів нейронів попереднього шару.

Іншим прикладом є усереднене агрегування, що використовує усереднене значення з кожного кластеру нейронів попереднього шару. [10].

Повноз'єднані шари з'єднують кожен нейрон одного шару з кожним нейроном наступного шару. Це, в принципі, є тим же, що й традиційна нейронна мережа багатошарового перцептрону. ЗНМ використовують спільні ваги в згорткових шарах, що означає, що для кожного рецептивного поля шару використовується один і той же фільтр (значення ваг); це зменшує обсяг необхідної пам'яті та поліпшує продуктивність.

1.3. Опис математичних методів навчання нейронної мережі.

Для тренування нейронних мереж використаних у даній кваліфікаційній роботі застосовувався метод градієнтного спуску.

Градіє́нтний спуск (англ. gradient descent) — це ітераційний алгоритм оптимізації першого порядку, в якому для знаходження локального мінімуму функції здійснюються кроки, пропорційні протилежному значенню градієнту (або наближеного градієнту) функції в поточній точці. Якщо натомість здійснюються кроки пропорційно самому значенню градієнту, то відбувається наближення до локального максимуму цієї функції.

Стандартна реалізація даного методу не є оптимальною для тренування глибоких мереж, тому у даній роботі використовується модифікація методу градієнтного спуску під назвою Adam.

Adam - це алгоритм оптимізації, який можна використовувати замість класичної процедури стохастичного градієнтного спуску для ітеративного оновлення ваги мережі на основі навчальних даних. [11].

Переваги використання Adam у невипуклих задачах оптимізації таким чином:

- простота реалізації;
- невеликі вимоги до пам’яті;
- інтуїтивні гіперпараметри;

- інваріативність зміни градієнту за напрямом.

Метод обчислює індивідуальні адаптивні швидкості навчання для різних параметрів з оцінок першого та другого моментів градієнтів. На рис. 1.2. зображено псевдокод алгоритму. [12].

Algorithm: Generalized Adam

S0. Initialize $m_0 = 0$ and x_1

For $t = 1, \dots, T$, do

S1. $m_t = \beta_{1,t} m_{t-1} + (1 - \beta_{1,t}) g_t$

S2. $\hat{v}_t = h_t(g_1, g_2, \dots, g_t)$

S3. $x_{t+1} = x_t - \frac{\alpha_t m_t}{\sqrt{\hat{v}_t}}$

End

Рис. 1.2. Псевдокод алгоритму

Adam є популярним алгоритмом у галузі глибинного навчання, тому що він швидко досягає гарних результатів.

Емпіричні результати показують, що Adam добре працює на практиці. В оригінальній статті якість алгоритму була продемонстрована емпірично, щоб показати, що збіжність функції витрат відповідає очікуванням теоретичного аналізу. Адам був застосований до навчання логістичної регресії в наборах даних розпізнавання символів MNIST і IMDB, та багатошарових нейронних мережах в наборі даних MNIST і нейронних згорткових мережах в наборі даних розпізнавання зображень CIFAR-10.

1.4. Обрані набори даних

Найбільш доцільними для вирішення задачі наборами даних були обрані набори Places2 та Im2GPS, які дозволяють отримати достатню кількість точок для навчання багатозадачної мережі. [13].

Набір даних Places2 розроблено відповідно до принципів візуального пізнання людини. Цей набір створює ядро візуальних знань, яке можна використовувати для навчання штучних систем для виконання завдань візуального розуміння високого рівня, таких як контекст сцени, розпізнавання об’єктів, передбачення дій і подій, а також теорія розумового висновку. Місця в цьому наборі даних визначаються їх функцією: мітки представляють базовий рівень середовища. Для ілюстрації набір даних має різні категорії спалень, вулиць тощо. На рис 1.3. представлена візуалізація декількох класів.

Рис. 1.3. Вибірка зображень з набору даних

Загалом Places2 містить понад 10 мільйонів зображень, які містять понад 400 унікальних категорій сцен. Набір даних містить від 5000 до 30 000 навчальних зображень на клас, подібно реальному світу. Використовуючи згорткові нейронні мережі (CNN), набір даних Places2 дозволяє вивчати глибинні функції обстановки на зображенні для різних задач класифікації.

Дуже корисною характеристикою набору даних є розподіл класів за частотою, оскільки збалансованість навчальної вибірки може вплинути на якість отриманої моделі. На рис. 1.4. зображено відсортований розподіл кількості зображень за категоріями в базі даних. Places2 містить 10624928 зображення з 434 категорій. Назви категорій відображаються кожні 6 інтервалів. [14].

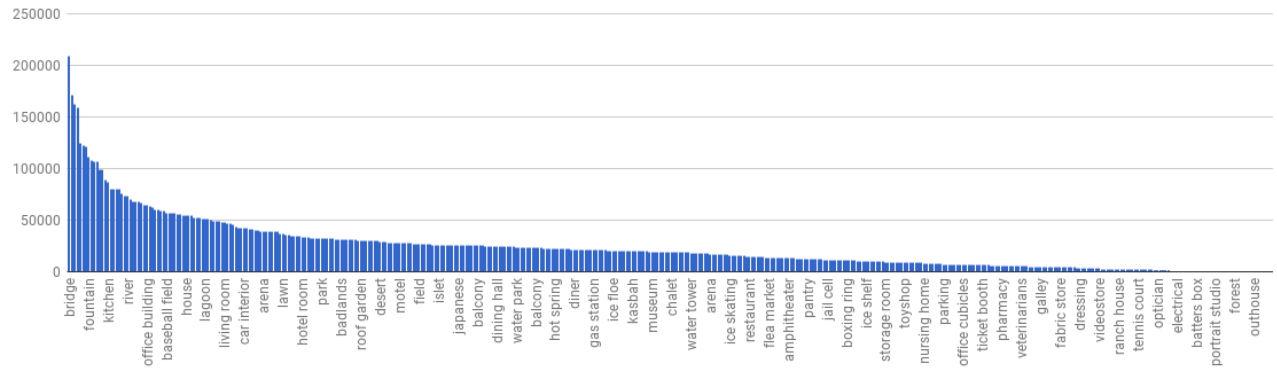

Рис. 1.4. Кількості міток для кожного класу

Наступний набір даних містить набір зображень та відповідні GPS координати. Оскільки оцінка географічної інформації із зображення — цікава, складна проблема комп’ютерного зору високого рівня, поява величезної кількості географічно відкаліброваних даних зображень є чудовою причиною для того, щоб комп’ютерний зір можна було використати у масштабі всієї планети! Автори цього набору даних пропонують простий алгоритм для оцінки розподілу за географічними місцями з одного зображення за допомогою підходу зіставлення сцени на основі даних. Для цього завдання було використано набір даних із понад шести мільйонів зображень із тегами GPS, які були зібрані з мережі інтернет. Im2GPS є найбільшим набором даних такої якості. На рис. 1.5. зображено розподіл даних присутніх в наборі.

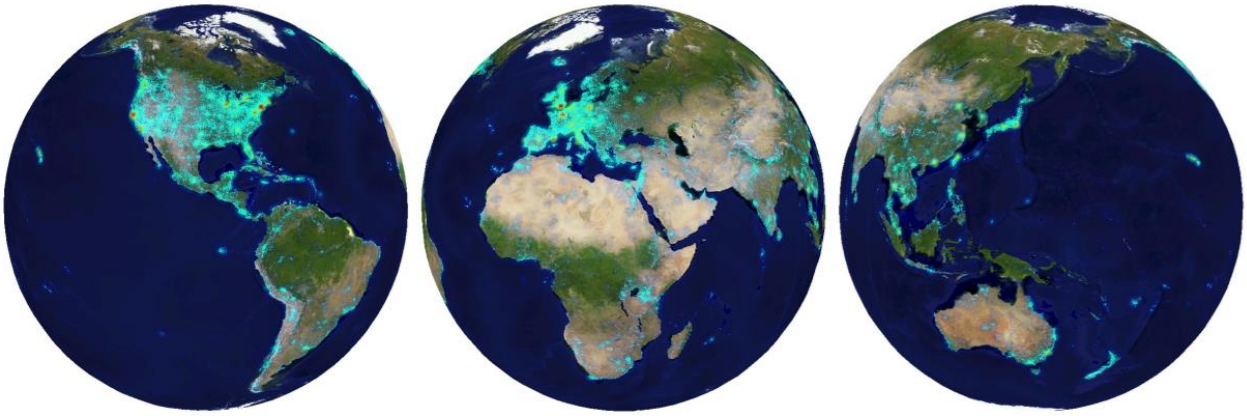

Рис. 1.5. Розподіл даних Im2GPS на земній кулі

Знання розподілу ймовірних гео-локацій для зображення надає величезну кількість додаткових метаданих для клімату, середньої температури за будь-який день, індексу рослинності, в сукупності ці дані дають можливість дуже чітко виявити шукану локацію.

1.4. Постановка задачі

Метою роботи є створення системи для визначення географічних координат об’єктів зображених на фотографії за допомогою багатозадачних нейронних мереж.

Для досягнення поставленої мети необхідно дослідити методи розрахунку розподілу температурних полів у ізотропних тілах обертання, чисельні методи в інформатиці, методи моделювання фізичних процесів теплопровідності, методи дискретизації областей, які використовуються для розрахунку розподілу температури.

Після проведених досліджень необхідно проаналізувати інформацію і відомості, які були зібрані, та на їх підставі розробити програмне забезпечення для визначення гео-локації об’єкта за зображенням.

1.5. Висновки

У даному розділі було проведено аналіз предметної області, були визначені поняття глибинного навчання і методів оптимізації. Нейронні мережі дають можливість проаналізувати розподіл геолокацій, які виділяються за певними наборами візуальних характеристик.

Отримана інформація дає розуміння розподілу даних у використаних наборах і доступності локацій для визначення, надає характеристику згорткових нейронних мереж. Виникає проблема у створенні програмного забезпечення, яке дозволить отримувати результат визначення геолокації об’єкта на зображенні.

РОЗДІЛ 2
ОПИС МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1. Моделювання архітектури ResNet

Велика кількість сучасних мереж починаються з вибору архітектури під конкретну задачу. Для вирішення задачі класифікацію доцільним вибором є архітектура ResNet, оскільки вона не затратна в реалізації та дозволяє досягти високих значень метрик класифікації та вирішити проблему затухаючого градієнту.

ResNet — це скорочене назва для Residual Network. Коли більш глибока нейронна мережа починає процес навчання, виникає проблема: зі збільшенням глибини мережі точність спочатку збільшується, а потім швидко усувається. Зниження точності навчання показує, що не всі глибинні мережі легко оптимізувати. Щоб подолати цю проблему, компанія Microsoft запровадила глибоку «залишкову» структуру навчання. Замість того, щоб сподіватися, що кожні кілька наступних шарів безпосередньо відповідають бажаному основному уявленню, вони явно дозволяють цим шарам відповідати «залишку». Ця ідея може бути реалізовано за допомогою нейронних мереж зі з'єднаннями для швидкого доступу. На рис. 2.1. зображена схема структури такого з’єднання.

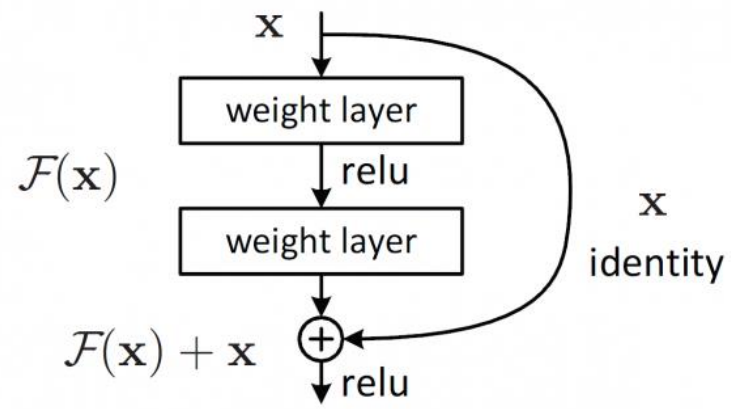

Рис. 2.1. З'єднання для швидкого доступу

Як можна побачити з представленої схеми, проміжкові вихід раннього шару використовує таке з'єднання для передачі інформації на подальші шари. Для визначення географічних координат об'єктів в даній роботі була обрана конкретна версія архітектури під назвою ResNet50. [14].

На рис. 2.2. Детально зображені шари архітектури ResNet50, вона містить такі основні елементи:

- згортка з розміром ядра 7 * 7 і 64 ядра різниці, всі з кроком розміру 2, що утворює 1 шар;
- наступним шаром є MaxPooling з розміром кроку 2;
- у наступній згортці є ядро 1 * 1,64, наступне за цим ядро 3 * 3,64 і, нарешті, ядро 1 * 1256. Ці три шари повторюються 3 рази, тож на цьому кроці отримано 9 однакових шарів;
- далі використано ядро 1 * 1,128, потім ядро 3 * 3,128 і, нарешті, ядро 1 * 1,512, цей крок повторювався 4 рази, таким чином створено ще 12 шарів на цьому етапі;
- після цього є ядро 1 * 1,256 і ще два ядра з 3 * 3,256 і 1 * 1,1024, і це повторюється 6 разів, що створює ще 18 шарів;
- а потім знову ядро 1 * 1,512 з ще двома 3 * 3,512 і 1 * 1,2048, і це повторювалося 3 рази, що створює загалом ще 9 шарів;
- після цього виконується операцію AveragePooling і додаємо фінальний шар мережу повнозв’язним шаром, що містить 1000 нейронів, і в кінці функцією softmax, тож отримуємо ще 1 шар.

Всього мережа містить 50 шарів не включаючи функції активації як окремі шари.

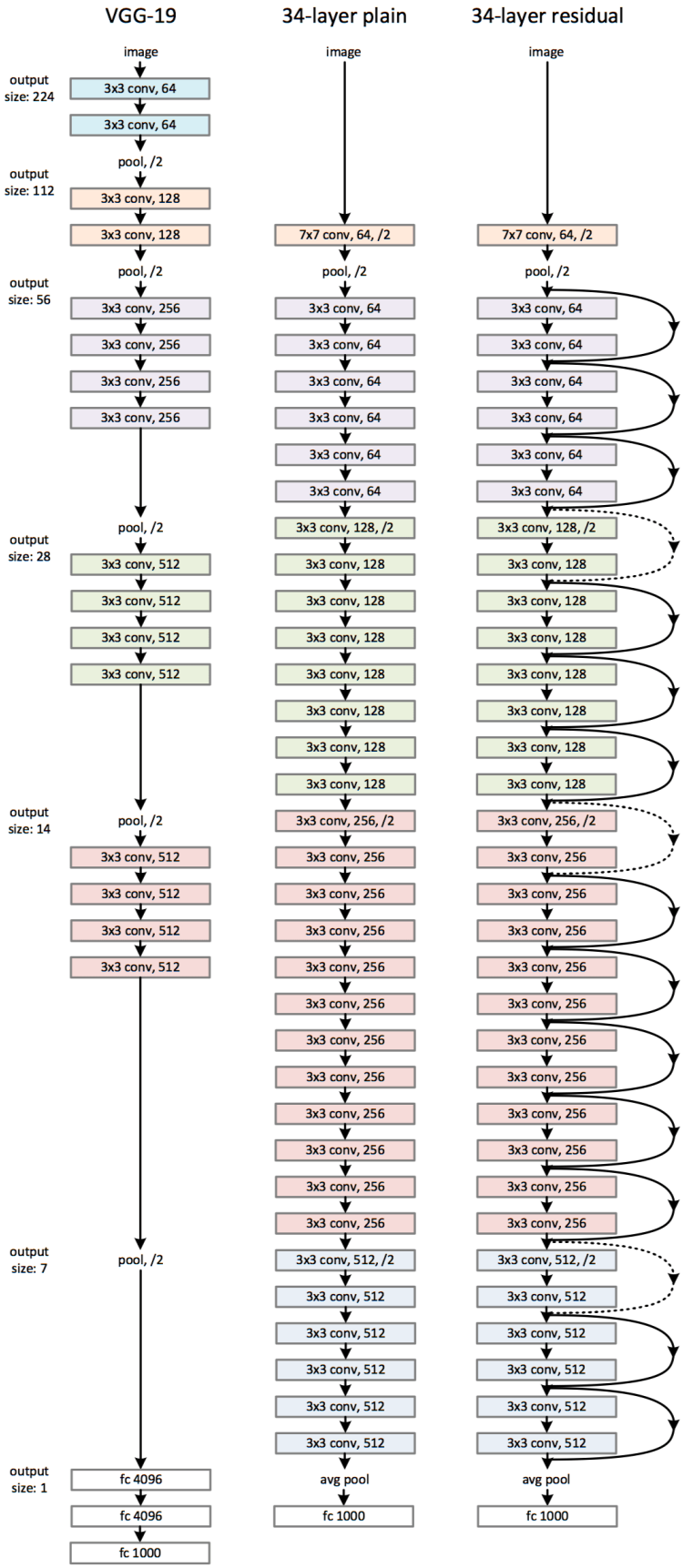

Рис. 2.2. Детальна архітектура нейронної мережі ResNet50

2.2. Автоматичне диференціювання

Для визначення параметрів мережі, що використовуються для визначення географічних координат, потрібно вираховувати часткові похідні, що є дуже затратною операцією, тому використовується підхід під назвою автоматичне диференціювання. [15].

Цей метод використовує той факт, що довільна функція в комп'ютерній програмі все одно буде обчислюватись за допомогою арифметичних дій (+, -, *, /) та елементарних функцій стандартних бібліотек (exp, log, sin, cos, і т.д.). Застосовуючи ланцюгове правило, похідна довільного порядку може бути обчислена з заданою точністю, за кількість операцій, що пропорційна кількості операцій для обчислення самої функції. При цьому, автоматичне диференціювання не є ні символьним диференціюванням, ні чисельним диференціюванням.

Символьне диференціювання не завжди ефективне, оскільки деякі функції важко представити єдиним виразом, а чисельне диференціювання призводить до внесення похибок округлення та дискретизації. Обидва ці методи не є зручними для обчислення похідних високих порядків, оскільки похибка і складність значно зростає. Також обидва ці методи є повільними при обчисленні часткових похідних для функції багатьох аргументів. [16]. Автоматичне диференціювання вирішує всі ці проблеми, але вводить додаткову програмну залежність. Для цього вводиться спеціальний граф «обчислень» на рис. 2. 3.

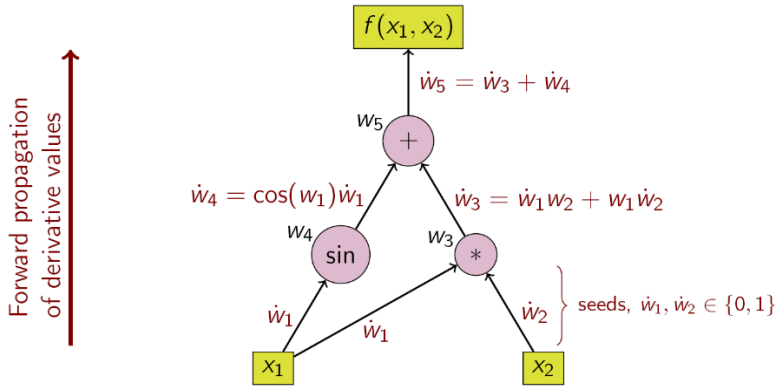

Рис. 2.3. Граф обчислень

Графи обчислень — це графи зі складовими рівняння. Вони є формою орієнтованих графів, які представляють математичний вираз. Дуже поширеним прикладом є обчислення постфікса, інфікса або префікса. Кожен вузол на графу може містити операцію, змінну або саме рівняння. Ці графи з'являються в більшості обчислень, які виконуються в комп'ютерах, структура такого графу представлена нижче на рис. 2.4.

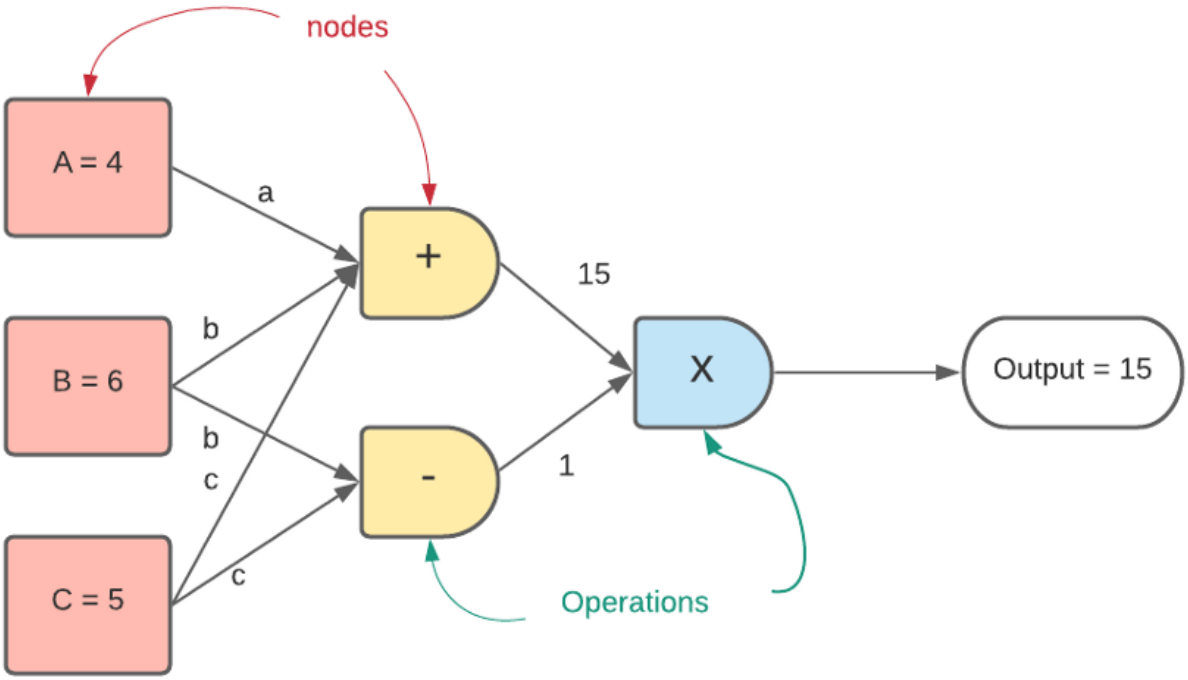

Рис. 2.4. Граф обчислень для простих операція

Усі нейронні мережі є графами обчислень. Не тільки вони, але й навіть такі алгоритми, як лінійна регресія, також можуть бути представлені у вигляді графів. Основною відмінністю традиційних графів від нейронних мереж є реалізація. Нейронні мережі, як правило, імітують обчислювальні графи для навчання, але не можуть обробляти дані, подібні до графів. Для роботи їм потрібні структуровані дані. [17].

2.3. Метод максимальної правдоподібності

Цей метод ґрунтується на припущенні про те, що вся інформація про статистичну вибірку міститься у цій функції. Метод максимальної правдоподібності був проаналізований, рекомендований і значно популяризований Р. Фішером між 1912 і 1922 роками (хоча раніше він використовувався Гаусом, Лапласом і іншими). Оцінка максимальної правдоподібності є популярним статистичним методом, який використовується для створення статистичної моделі на основі даних, і забезпечення оцінки параметрів моделі. [18].

Метод максимальної правдоподібності відповідає багатьом відомим методам оцінки в області статистики. Наприклад, припустимо, що метою є аналіз зросту мешканців України. Припустимо, що доступні дані стосовно зросту деякої кількості людей, а не всього населення. Крім того передбачається, що зріст є нормально розподіленою величиною з невідомою дисперсією і середнім значенням. Вибіркові середнє значення і дисперсія зросту є максимально правдоподібними до середнього значення і дисперсії всього населення. [19].

Для фіксованого набору даних і базової імовірнісної моделі, використовуючи метод максимальної правдоподібності, знаходяться значення параметрів моделі, які роблять дані «ближчими» до реальних. Оцінка максимальної правдоподібності дає унікальний і простий спосіб визначити рішення у разі нормального розподілу.

2.4. Квантування для нейронних мереж

Квантування відноситься до методів виконання обчислень і зберігання тензорів в типах даних меншої точності, ніж точність з плаваючою комою. Квантована модель виконує деякі або всі операції над тензорами з цілими числами, а не використовуючи тип даних з плаваючою комою. Це дозволяє створити більш компактне представлення моделі та використовувати

високопродуктивні векторизовані операції на багатьох апаратних платформах. Цей метод особливо корисний під час навчання на запуску моделі, оскільки він заощаджує багато витрат на обчислення, не жертвуючи занадто великою точністю обчислень. На даний момент основні фреймворки глибокого навчання, такі як TensorFlow і PyTorch, підтримують квантування. [20].

На рис. 2.5. наведені значення, які реєструються для параметрів під час навчання мережі .

Original Values	Power of 2 Bins																8 Bit Binary Rep	Quantized Value
	Sign Bit	2^{8}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}		
0.03125																		
-0.250																		
0.250																		
0.500																		
1.000																		
2.100																		
-2.125																		
8.250																		
16.250																		

Рис. 2.5. Перший крок квантизації

Потрібно знайти ідеальне двійкове представлення кожного зареєстрованого значення параметра. Старший біт (MSB) — це крайній лівий біт двійкового слова. Цей біт найбільше впливає на значення числа. MSB для кожного значення виділено жовтим кольором на рис. 2.6.

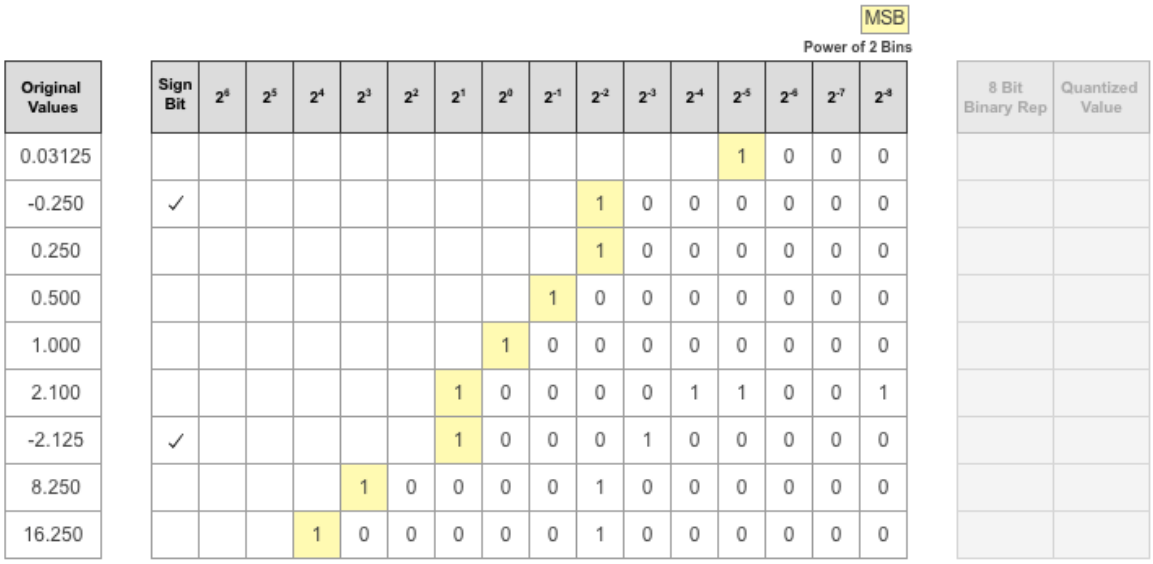

Рис. 2.6. Другий крок квантизації

Вирівнюючи двійкові слова, дуже легко помітити розподіл бітів, які використовуються зареєстрованими значеннями параметра. На рис. 2.7. MSB у кожному стовпці, виділена зеленим кольором, дає уявлення про зареєстровані значення. [22].

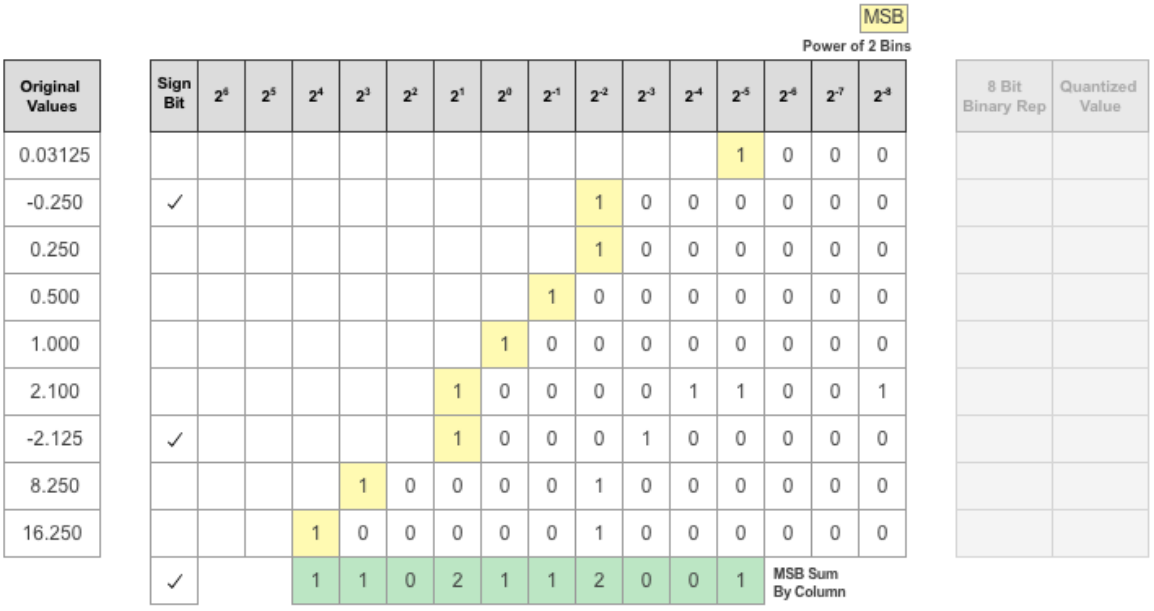

Рис. 2.7. Третій крок квантизації

Кількість MSB кожного розташування біта відображається як теплова карта. На цій тепловій карті темніші сині області відповідають більшій кількості MSB в розташуванні бітів на рис. 2.8.

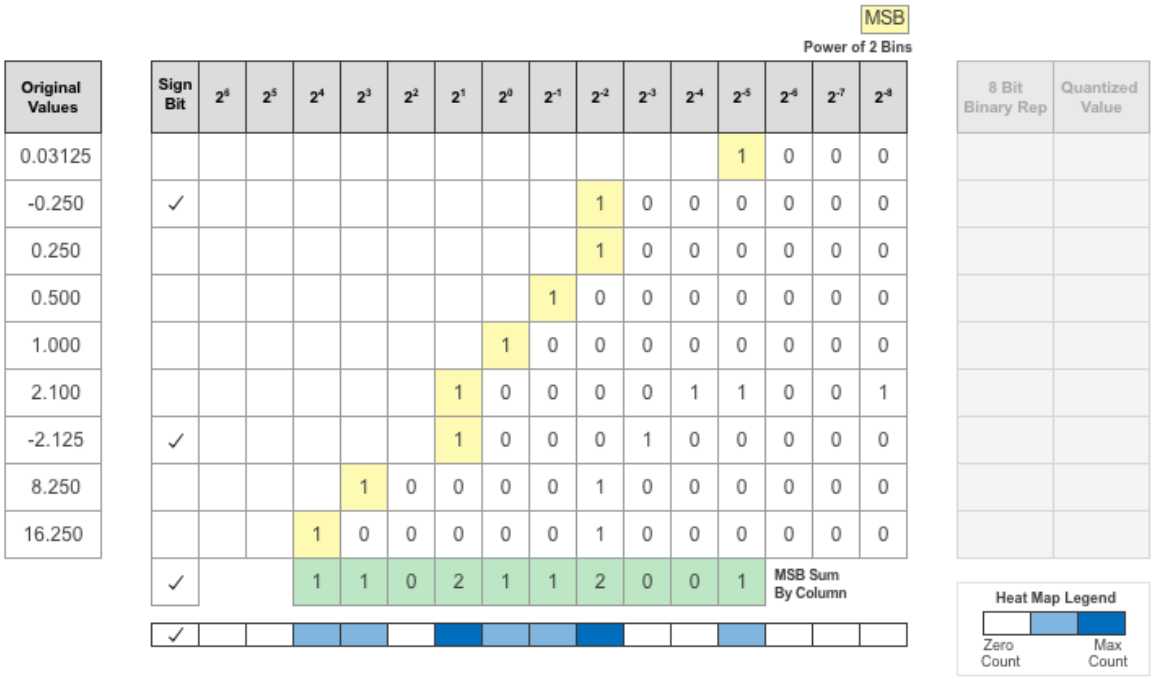

Рис. 2.8. Четвертий крок квантизації

Наступним кроком за алгоритмом призначається тип даних, який допоможе уникнути переповнення, охопити діапазон. Для представлення знакового значення потрібен додатковий знаковий біт. [23].

Після призначення типу даних будь-які біти за межами цього типу даних видаляються. Через призначення меншого типу даних фіксованої довжини для значень, які не можна представити типом даних, може виникнути втрата точності або переповнення. У цьому прикладі значення 0,03125 страждає від недостатньої тоності, тому квантованою величиною є 0. Значення 2.1 є квантованою величиною є 2,125. Значення 16,250 більше, ніж найбільше репрезентативное значення типу даних, тому це значення переповнюється, а квантовану величину встановлює до 15,874.

На рис. 2.9. зображено результат квантизації де кожне число тепер займає не більше 8 біт інформації.

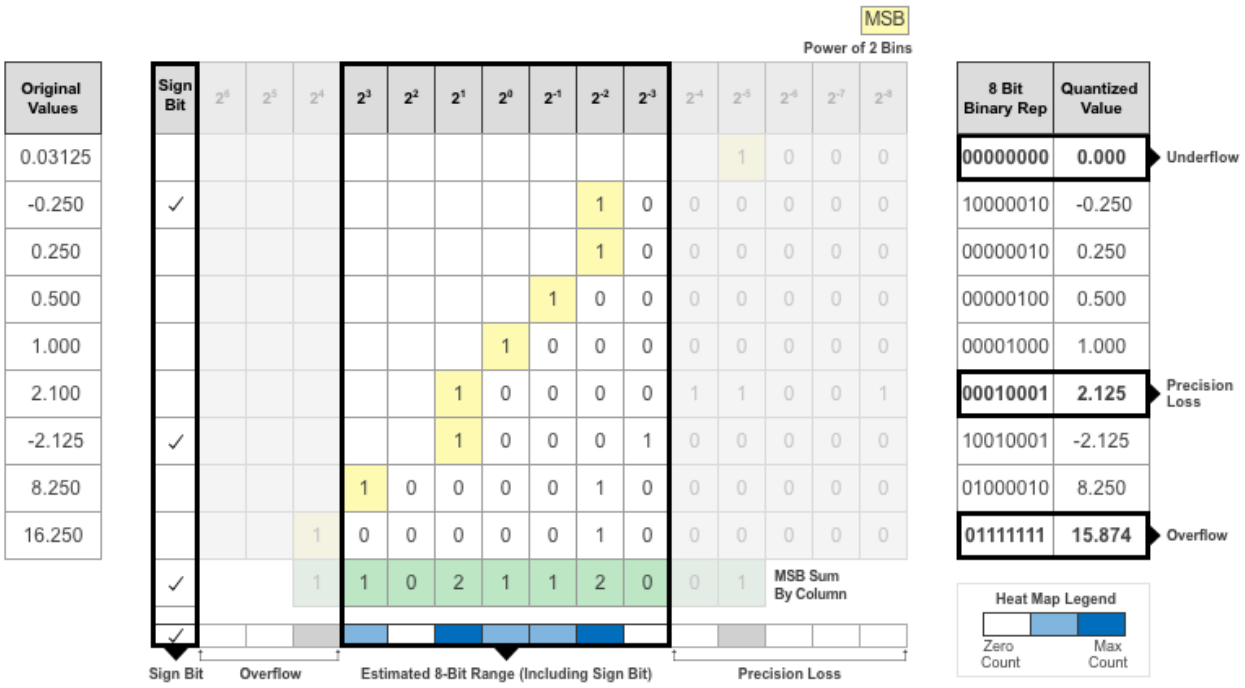

Рис. 2.9. Фінальний крок квантизації

2.5. Висновки

В даному розділі були проаналізовані методи для побудови нейронної мережі та оптимізації часу виконання після навчання. Серед наведених методів найбільшу цінність та практичність має метод автоматичного диференціювання, який дозволяє знаходити велику кількість часткових похідних першого порядку для прискорення навчання. Розрахунок, який використовує механізм квантизації, дає можливість знайти виконувати код нейронної мережі навіть на смартфонах після навчання, але результат може втрати незначну частину точності. Також було наведено архітектуру мережі, яка і задає параметри для навчання та буде використана при розробці програмного забезпечення для визначення географічних координат об’єктів на фото.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЗНАЧЕННЯ ГЕОГРАФІЧНИХ КООРДИНАТ ОБ'ЄКТІВ

3.1. Розробка архітектури системи

Для початку розглядається високорівнева архітектура системи, включаючи етапи розбиття землі на клітини та обробки даних з наборів (рис. 3.1).

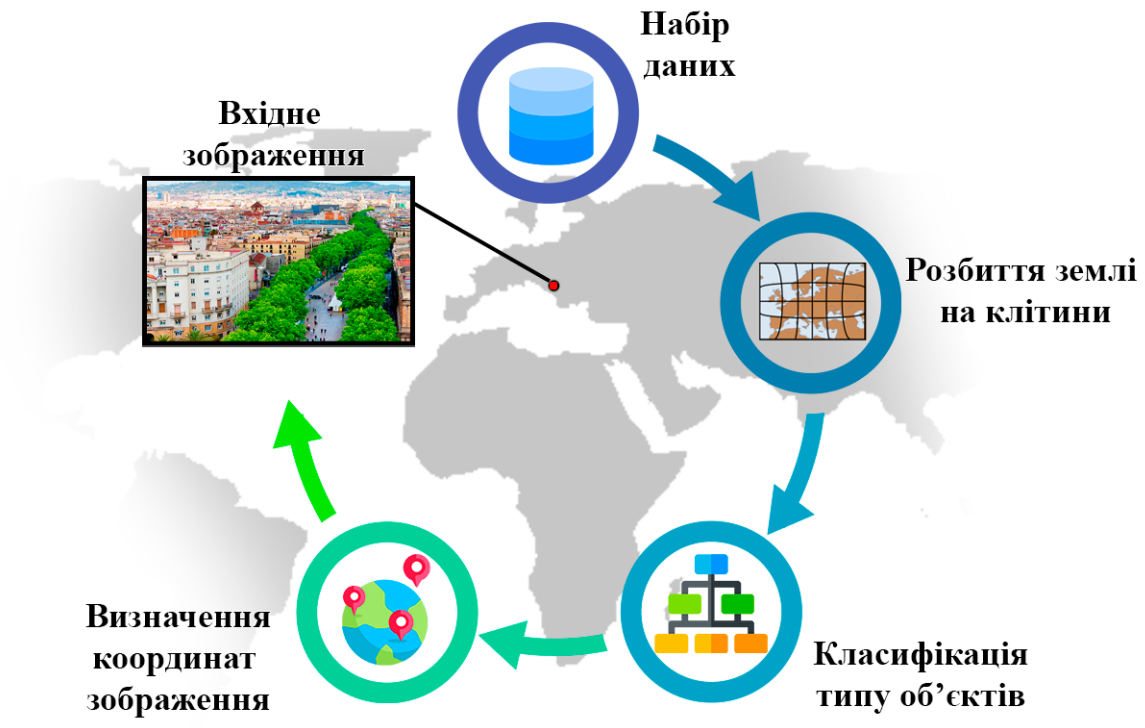

Рис. 3.1. Архітектура системи

Використовуючи завантажені набори даних, нейронна мережа навчається для задачі класифікації, і важливим питанням тут являється вибір класів, які буде використовувати мережа. Для цього потрібно вибрати правильний спосіб, як розбити земну кулю на клітини.

Для того, щоб правильно опрацювати різну кількість зображень, використовується різне розширення для розбиття на клітини. Процес тренування, розбиття та агрегації ймовірностей зображено на малюнку нижче (рис. 3.2).

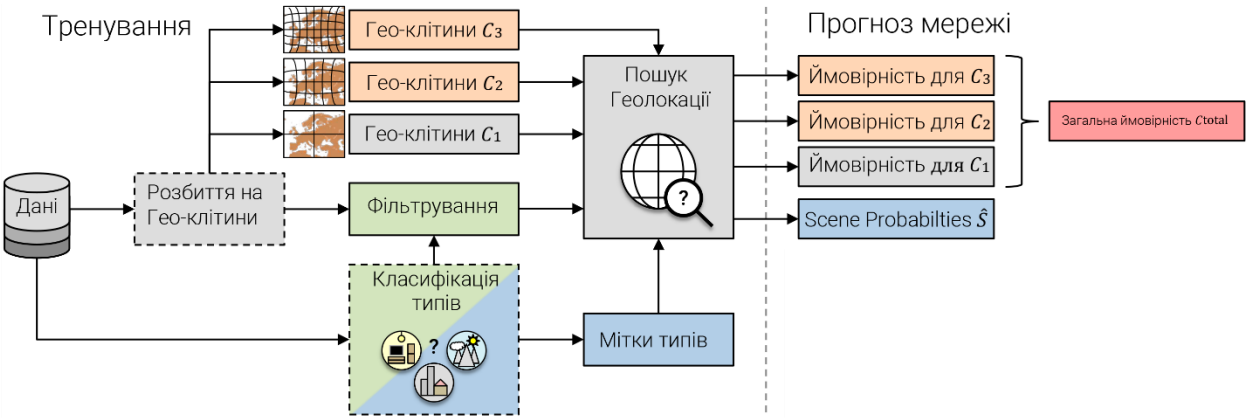

Рис. 3.2. Архітектура використання системи

Багатозадачна нейронна мережа**:** Оскільки вищезгаданий метод визначення геолокації може навчатися не ефективно через велику різноманітність можливих екологічних зображень територій, то для вирішення задачі пропонується архітектура для навчання на двох комплементарних задачах одночасно. Більш конкретно, додатковий повнозв'язний шар розташовується після шару «global pooling» архітектури згорткової мережі ResNet50. Кількість нейронів доданого шару є кількість категорій для типів зображень $|S|$. Параметри класифікаторів для обох задач збігаються. Як функція витрат для класифікації типів була обрана перехресна ентропія. Загальна функція витрат L_{total} мережі є сумою L_{scene} та L_{geo} для обох задач.

$$L_{total} = L_{scene} + L_{geo} \tag{3.1}$$

Ці дві функції втрати мають однакову вагу в представленій роботі. Кожна частина розраховується окремо як категорійна перехресна ентропія, з відповідною кількістю класів.

$$L_{CCE} = -\sum_{i=1}^{n} y_i * \log \hat{y}_i \tag{3.2}$$

Таким чином нейронна мережа отримує задачі у форматі, який допоможе узагальненню цільової функції.

3.2. Спосіб кодування географічних координат

Для того, щоб ефективно розв’язати задачу класифікації, поставлену для визначення географічних координат об’єктів, потрібно вибрати оптимальний спосіб розбиття земної кулі на клітини.

Для цього використовується бібліотека геометрії S2. Вона допомагає у створенні множини клітин *C*, що не перекриваються. Більш детально, земна поверхня проектується на описаний куб з шістьма гранями, що представляють початкові клітинки (рис. 3.3.).

Після цього виконується адаптивний ієрархічний розділ граней на основі координат GPS в наборі даних Im2GPS, таким чином, що кожна клітинка є вузлом чотирикутного дерева. Починаючи з кореневих вузлів, відповідне квадродерево поділяється рекурсивно, поки всі клітинки не містять максимум τ_{max} зображень.

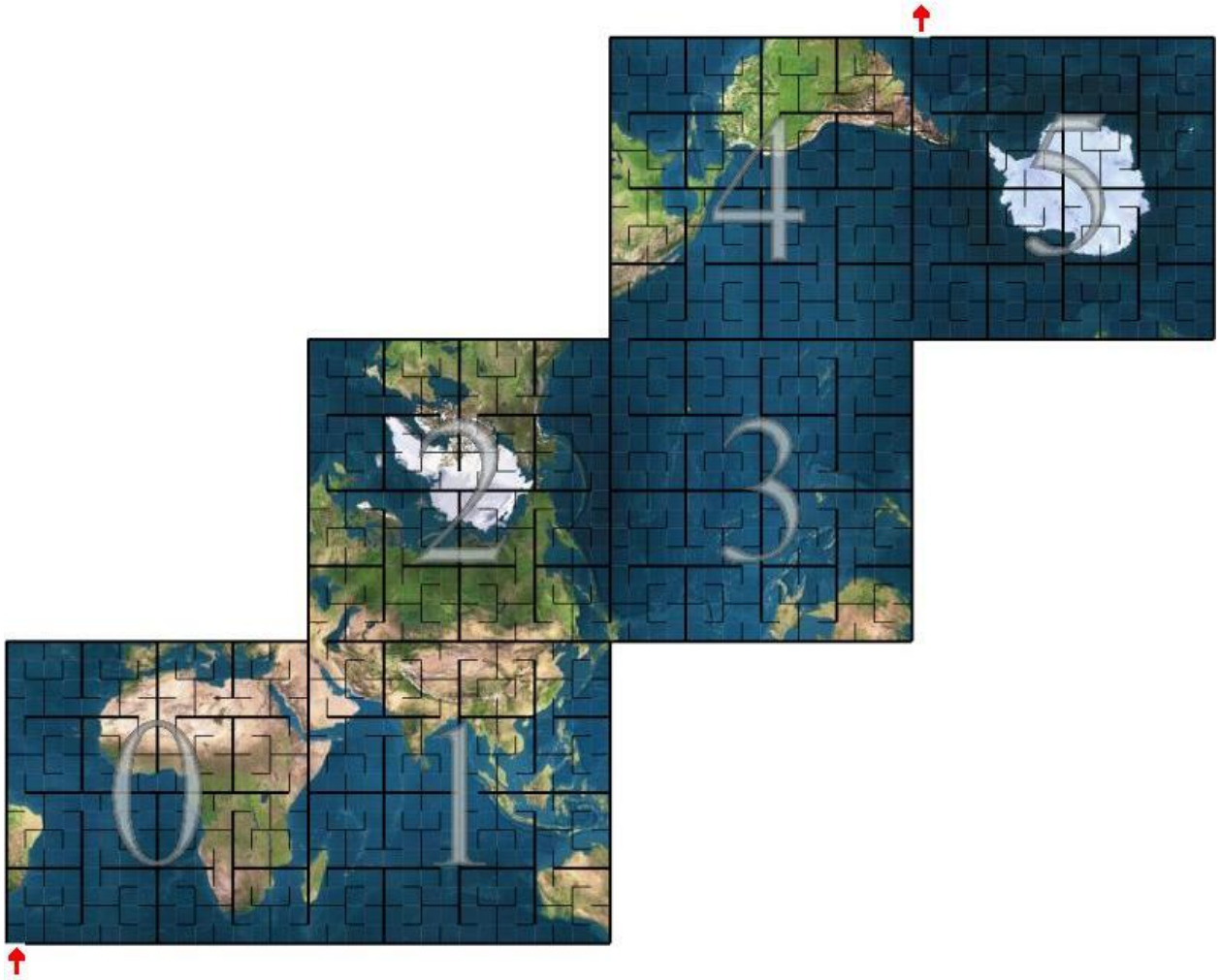

Рис. 3.3. Архітектура використання системи

Традиційна картографія заснована на картографічних проекціях, які є просто функціями, які відображають точки на поверхні Землі на точки на планарній карті. Картографічні проекції створюють спотворення через те, що форма Землі не дуже близька до форми площини. Наприклад, добре відома проекція Меркатора є розривною вздовж 180-градусного меридіана, має великомасштабні викривлення у високих широтах і взагалі не може представляти північний і південний полюси. Інші проекції роблять різні компроміси, але жодна плоска проекція не може добре відобразити всю поверхню Землі.

S2 підходить до цієї проблеми, працюючи виключно зі сферичними проекціями. Як випливає з назви, сферичні проекції відображають точки на поверхні Землі до ідеальної математичної сфери. Звичайно, такі відображення все ще створюють деяке спотворення, тому що Земля не зовсім сферична, але, як виявилося, Земля набагато ближче до сфери, ніж до площини. За допомогою сферичної проекції можна апроксимувати всю поверхню Землі з максимальним спотворенням 0,56%. Можливо, важливіше те, що сферичні проекції зберігають правильну топологію Землі – немає жодних сингулярностей чи розривів, з якими потрібно мати справу.

Крім того, клітинки S2 впорядковані послідовно вздовж кривої заповнення простору (фрактального типу). Конкретна крива, яка використовується для S2 розбиття, називається кривою S2 і складається з шести кривих Гільберта, пов’язаних між собою, щоб утворити єдину безперервну петлю по всій сфері. На рис. 3.4. проілюстровано криву S2 після 5 рівнів поділу.

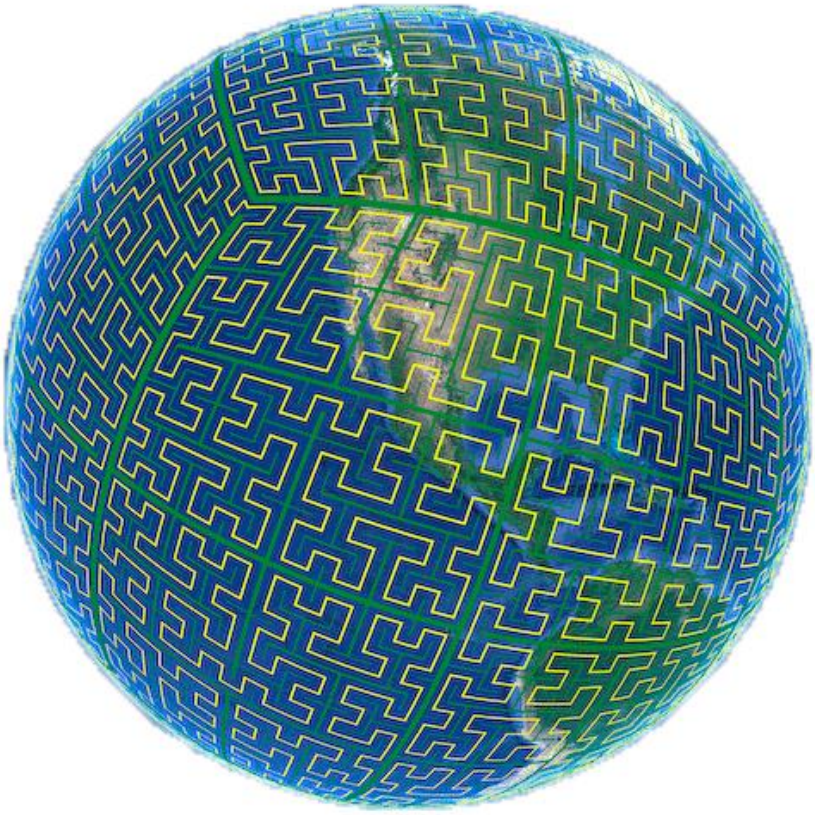

Рис. 3.4. Крива заповнення простору

3.3. Опис використаних технологій та мов програмування

Для розробки використовувалися: мова програмування Python, Javascript, менеджер пакетів pip та інтегроване середовище розробки PyCharm. Також застосовувався Docker як інструментарій для управління ізольованими Linux-контейнерами.

Перше, що варто зазначити – це вибір мови програмування. Python дуже добре підходить для цього проекту, бо він є близький по можливостям та швидкості до C++ у даній ситуації, проте більш простий при розробці серверної логіки та керуванні БД.

Взагалі, мова саме сфокусована на швидку і якісну розробку застосунків і забезпеченні високого рівню підтримання кодової бази (можливість підключення різноманітних модулів під конкретну задачу).

Додаток з цієї роботи може використовувати GPU для пришвидшення роботи нейронних мереж.

Також, через те, що Python – гарно структурована інтерпретована мова, коду вийде у декілька разів менше, ніж якби додаток писався на такій компільованій мові як C++.

Серед недоліків можна звернути увагу на складність вивчення різноманітних модулів цієї мови програмування, розуміння принципів роботи інтерпретатора та їх взаємодії недосвідченою людиною. Python підтримує суміш процедурних і об'єктно-орієнтованих методів, а також узагальнене програмування і метапрограмування, в статичних і динамічних стилях.

Pip — система керування пакунками, яка використовується для встановлення та управління програмними пакетами, які написані на Python. Багато пакетів можна знайти в Python Package Index.

PyCharm — інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веброзробку на Django. PyCharm розроблена російською компанією JetBrains на основі IntelliJ IDEA.

Django (Джанго) — високорівневий відкритий Python-фреймворк (програмний каркас) для розробки вебсистем.

PyTorch — відкрита бібліотека машинного навчання на основі бібліотеки Torch, що використовують для таких застосувань, як комп'ютерне бачення та обробка природної мови. Вона є вільним та відкритим програмним забезпеченням, що випускають під ліцензією Modified BSD. І хоча інтерфейс Python є більш відшліфованим, і головним зосередженням розробки, PyTorch також має зовнішній інтерфейс і для C++. За допомогою цієї бібліотеки можливо швидко реалізувати архітектуру обробки природної мови для оцінки коментарів користувачів (рис. 3.5.).

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

CSS — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних. CSS дозволяє прикріпляти стилі (рис. 3.6.) до документу. Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

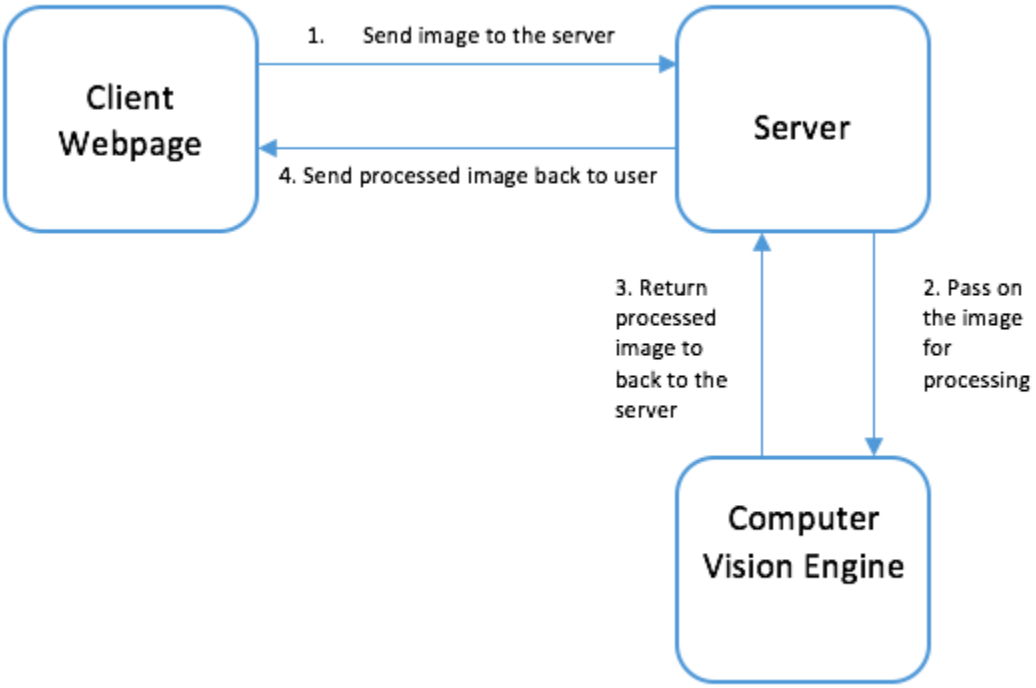

Рис. 3.5. Архітектура системи обробки зображень

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

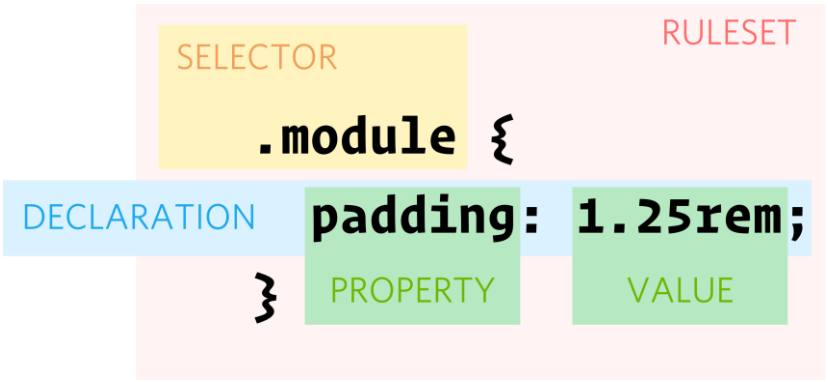

Рис. 3.6. CSS-стиль

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність

правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів.

HTML (англ. HyperText Markup Language — мова розмітки гіпертексту) — це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет

Веббраузери отримують HTML-документи з вебсервера або з локальної пам'яті і передають документи в мультимедійні вебсторінки. HTML описує структуру (рис. 3.7.) вебсторінки семантично і спочатку включені сигнали для зовнішнього вигляду документа.

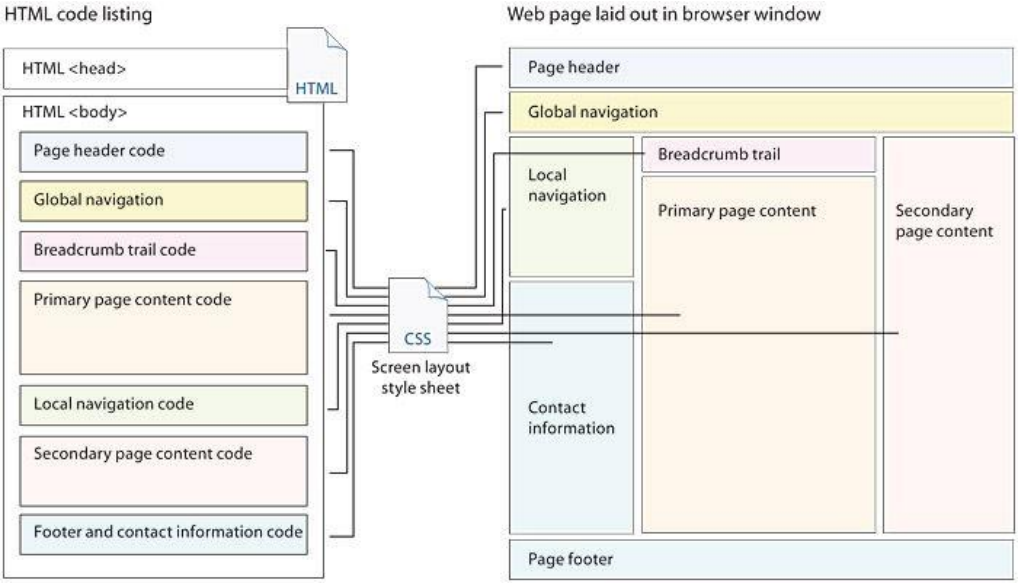

Рис. 3.7. Порівняння порядку HTML-коду на складній веб-сторінці з отриманим макетом сторінки за кожним блоком коду.

Іншими словами HTML — це мова розмітки, або ще один спосіб зберігання інформації. За допомогою HTML позначається текст, який вказує вебпереглядачу, як він має розуміти позначений текст, так само як і на жорсткому диску інформація зберігається в блоках, кластерах, секторах, доріжках і тільки за допомогою, такої визначеної структури комп'ютер розуміє, що треба, а що не треба зчитувати.

У HTML текст позначається за допомогою тегів. Кожен HTML документ буде складатися з деякої групи елементів, де кожен елемент буде визначатися

(починатися та закінчуватися) певним теґом (Для деяких елементів кінцевий теґ не є обов'язковим). Теґ — це назва елементу, записана у кутових дужках (< >).

Кожен HTML теґ має свою унікальну назву з визначеним синтаксисом, яка записується латинськими літерами і не чутлива до регістру.

World Wide Web Consortium (W3C), які супроводжують як HTML і CSS стандартів, заохочує використання CSS над явним презентаційним HTML з 1997 року.

HTML впроваджує засоби для:

– створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
– створення інтерактивних форм;
– включення зображень, звуку, відео, та інших об'єктів до тексту.

Docker — інструментарій для управління ізольованими Linux-контейнерами. З його допомогою можливо автоматизувати розгортання веб-сайту. Також можливе використання різних інтерфейсів для доступу до віртуалізації ядра операційної системи (рис. 3.10.) та можливість роботи з різними дистрибутивами.

Початковий код Docker написаний мовою Go і поширюється під ліцензією Apache 2.0. Інструментарій базується на застосуванні вбудованих в ядро Linux штатних механізмів ізоляції на основі просторів імен (namespaces) і груп управління (cgroups). Для створення контейнерів використовуються скрипти lxc. Для формування контейнера досить завантажити базовий образ оточення (команда docker pull base), після чого можна запускати в ізольованих оточеннях довільні програми (наприклад, для запуску bash можна виконати docker run -i -t base/bin/bash).

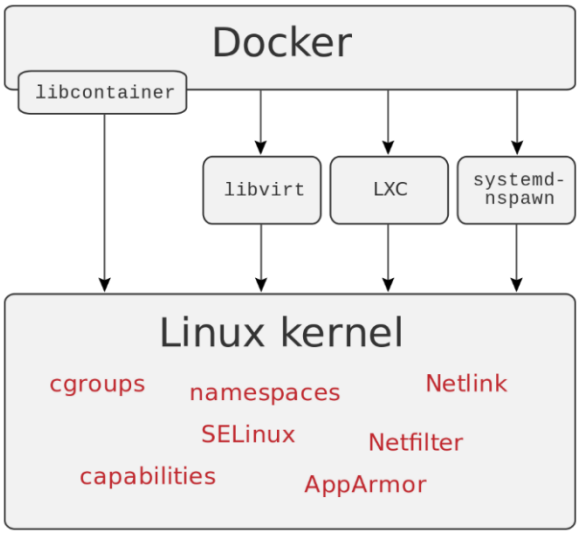

Рис. 3.8. Використання декількох інтерфейсів для доступу засобів віртуалізації ядра Linux

3.4. Опис роботи розробленої системи

3.4.1 Використані технічні засоби

Для серверних технічних засобів рекомендована конфігурація:

– процесор класу Intel ® Xeon з тактовою частотою 2.4GHz;
– шина даних - 1066MHz;
– кеш другого рівня - 4096 КБ;
– оперативна пам’ять 2 х DIMM DDR2-800 2048 Мб;
– жорсткі диски 3х 500 Гб SATA 2 16 Мб буфер, 7200 RPM;
– доступ до мережі Internet.

Для клієнтських технічних засобів рекомендована конфігурація:

– процесор класу Intel ® Core з тактовою частотою 2.1GHz;
– шина даних - 1066MHz;
– кеш другого рівня - 2048 КБ;
– оперативна пам’ять 2 х DIMM DDR2-800 1024 Мб;
– жорсткі диски 2х 250 Гб SATA 2 16 Мб буфер, 7200 RPM;
– рідкокристалічний монітор з діагоналлю не менше 17 ";
– доступ до мережі Internet;

– клавіатура;
– маніпулятор "миша".

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

3.4.2 Використані програмні засоби

Проект реалізований на мові програмування Python та Javascript. Також для коректної роботи розробленого застосунку необхідний довільний веб-браузер з підтримкою технології JavaScript.

Для серверного комп'ютера:

– ОС сімейства Windows (7,8, 10), Linux;
– HTTP-сервер nginx для розгортання

Для клієнтського комп'ютера:

– ОС сімейства Windows (7,8, 10), Linux;
– веб-браузер Firefox / Google Chrome / Opera.

3.4.3 Виклик та завантаження програми

Після розгортання веб-додатку на сервері, все, що буде потрібно – це перейти на домен, де буде знаходитись веб додаток. Завантаження і робота сайту буде виконуватись за допомогою веб-браузера з підтримкою JavaScript.

3.4.4 Опис інтерфейсу користувача

Інтерфейс користувача являє собою динамічно оновлювану сторінку.

Для того, щоб відкрити вебдодаток, необхідно зайти на офіційно зарезервоване доменне ім’я http://127.0.0.1:5000/ (у випадку, якщо веб-додаток запущений локально).

Основна сторінка додатку це — головна. (рис. 3.9.) На цій сторінці представлені така частина функціоналу, як: кнопка завантаження зображення на обробку, мапа відображення результату, приклади малюнків.

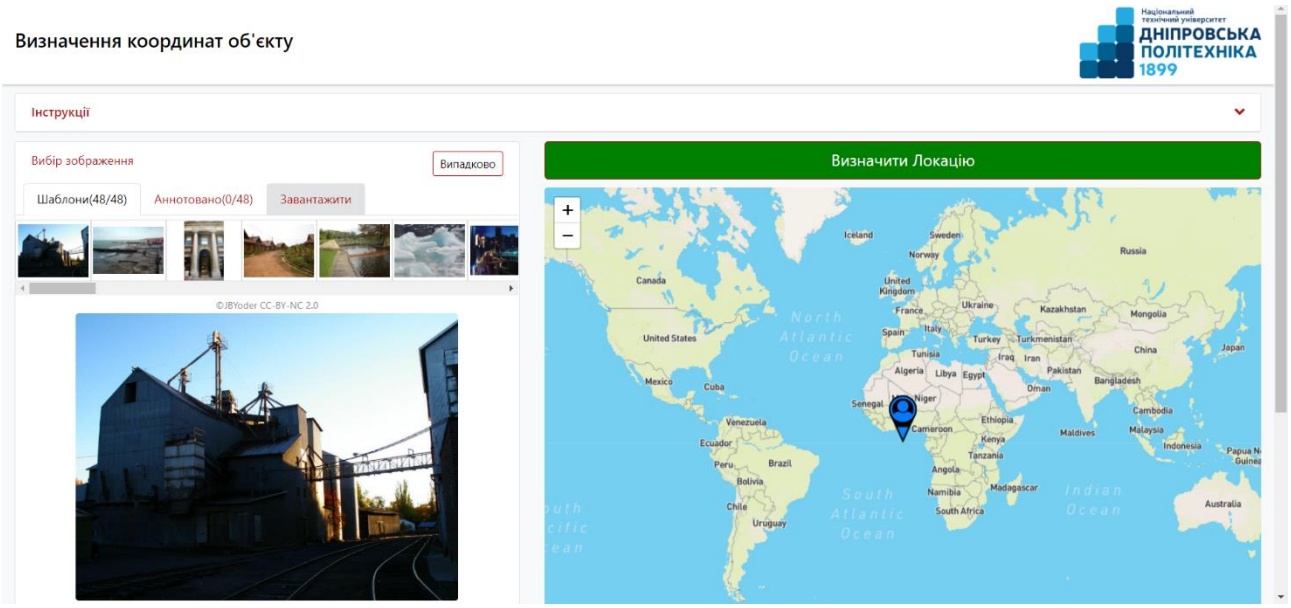

Рис. 3.9. Інтерфейс системи

Протестовано декілька зображень і візуалізовано визначення географічних координат об’єктів за зображенням. Для тестування були обрані особисті фотографії з подорожей. Для даних вхідних зображень проаналізовано вихід мережі. На рис. 3.10. зображено колесо обзору в Лондоні, мережа максимально точно, лише по зображенню знаходить можливі координати об’єкту (жовтий маркер це істинна локація, білий це найбільш ймовірна локація з точки зору системи) на рис. 3.11. На рис. 3.12 та 3.13 знятий наступний об’єкт в Барселоні з пляжу, що не заважає навіть за таким фото дізнатися локацію з точністю до 100 метрів.

Рис. 3.10. Вхідне зображення 1

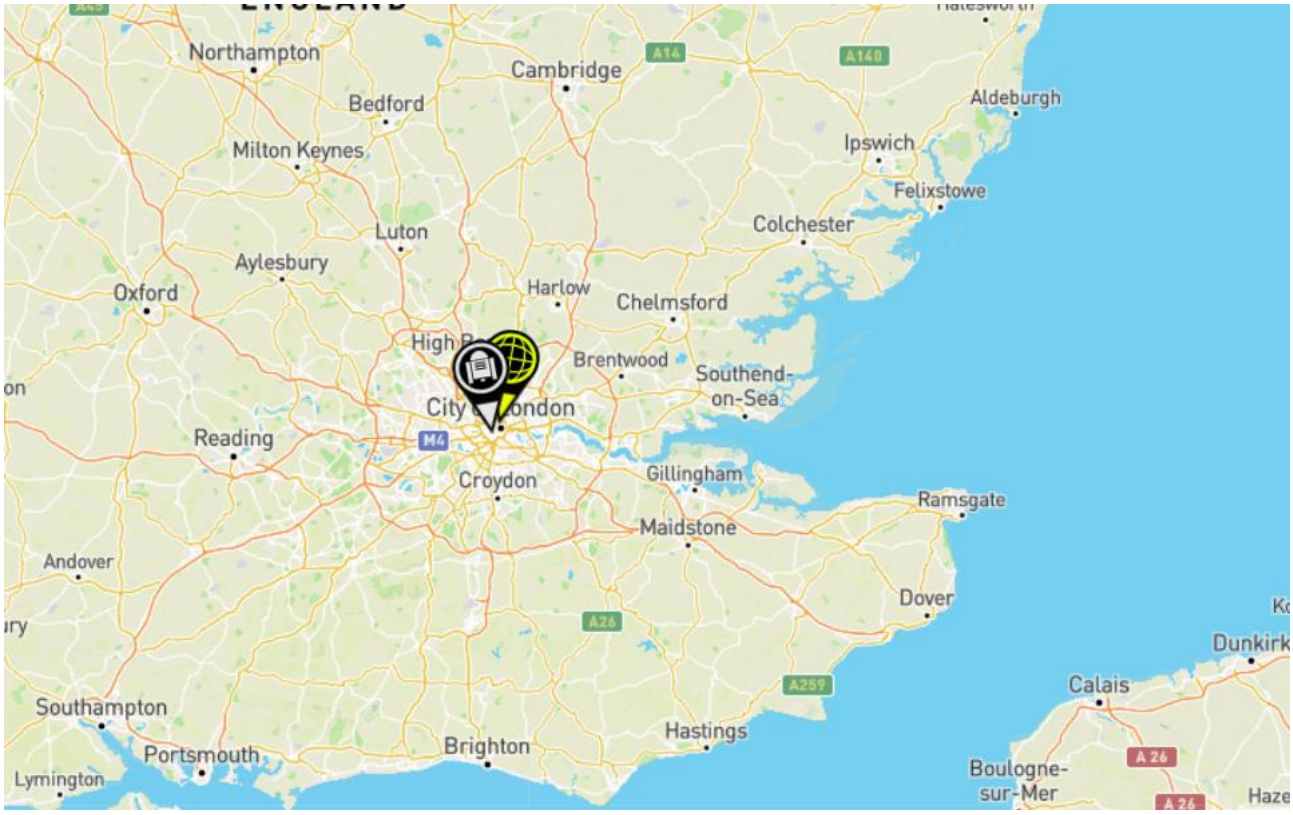

Рис. 3.11. Вихід мережі 2

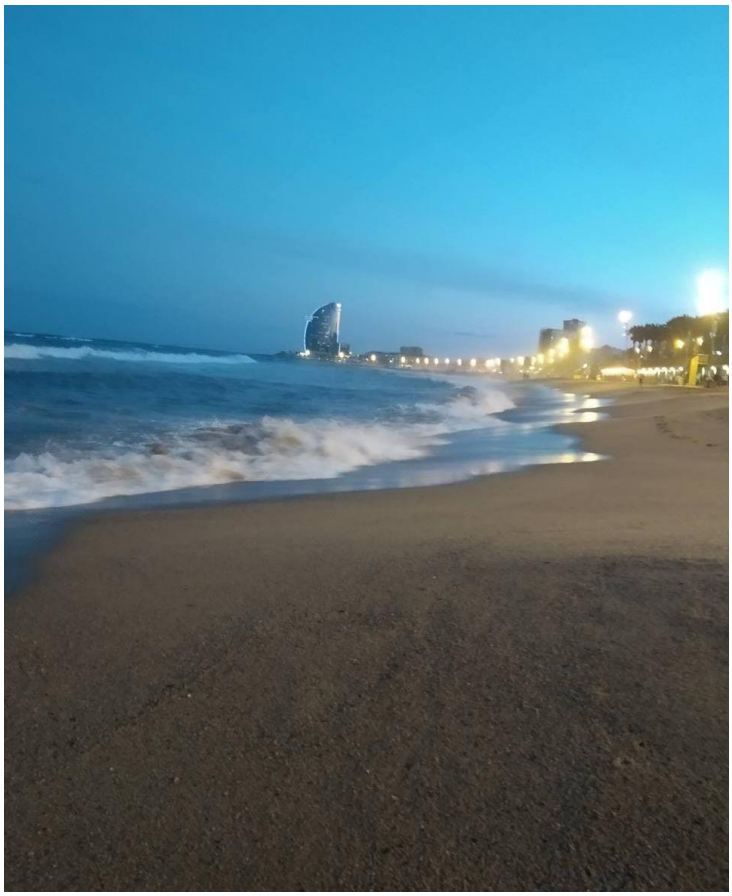

Рис. 3.12. Вхідне зображення 2

Рис. 3.13. Вихід мережі 2

3.3. Висновки

У даному розділі була проведена проектування та розробка запропонованої системи. Результати були отримані, як через інтерфейс системи, так і через термінал команд Windows. Програма надає візуальне відображення розподілу ймовірностей або найбільш ймовірну локацію об’єкту зображеного на фото. Вихідні дані декодуються з виходу мережі та відображаються в зручному інтерфейсі користувача.

РОЗДІЛ 4
ЕКОНОМІКА

Важливими етапами розробки програмного забезпечення є визначення трудомісткості розробки та розрахунок витрат на створення програмного продукту.

4.1. Визначення трудомісткості розробки програмного забезпечення

Задані дані:

1. передбачуване число операторів – 600;
2. коефіцієнт складності програми – 1,5;
3. коефіцієнт корекції програми в ході її розробки – 0,1;
4. годинна заробітна плата програміста, грн/год - 320;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0.8;
7. вартість машино-години ЕОМ, грн/год - 10.

Нормування праці в процесі розробки програмного забезпечення суттєво ускладнено через творчий характер роботи програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки за формулою:

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_{\partial}, \text{ людино-годин,} \qquad (4.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ та розраховується за формулою:

$$Q = q \cdot C \cdot (1 + p), \tag{4.2}$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт корекції програми в ході її розробки.

$$Q = 600 \cdot 1{,}5 \cdot (1 + 0{,}1) = 1089$$, людино-годин.

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \tag{4.3}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи спеціаліста до 2 років коефіцієнт складає 0,8.

$$t_u = \frac{1089 \cdot 1{,}3}{80 \cdot 0{,}8} = 22{,}2$$, людино-годин.

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}, \tag{4.4}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставимо у формулу відповідні значення, та отримаємо:

$$t_a = \frac{1089}{20 \cdot 0{,}8} = 68{,}1, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}, \text{ людино-годин.}$$

$$t_n = \frac{1089}{25 \cdot 0{,}8} = 54{,}5, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)\cdot k}, \text{ людино-годин.}$$

$$t_{\text{отл}} = \frac{1089}{5 \cdot 0{,}8} = 272{,}3, \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 \cdot t_{\text{отл}}, \text{ людино-годин.}$$

$$t_{\text{отл}}^{k} = 1{,}5 \cdot 272{,}3 = 408{,}4, \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до}\ , \text{ людино-годин,}$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)\cdot k}\ , \text{ людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}\ , \text{ людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{\text{др}} = \frac{1089}{15 \cdot 0{,}8} = 90{,}8, \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 55 = 68{,}1, \text{ людино-годин.}$$

$$t_{д} = 90{,}8 + 68{,}1 = 158{,}9, \text{ людино-годин.}$$

Використовуючи формулу (4.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 22{,}2 + 68{,}1 + 54{,}5 + 272{,}3 + 158{,}9 \approx 626 \text{ людино-годин.}$$

4.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ $К_{ПО}$ включають витрати заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ

$$К_{по} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (4.5)$$

Заробітна плата виконавця визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (4.6)$$

де t – загальна трудомісткість, людино-годин. $С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 626 \cdot 320 = 200320 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot С_{МЧ}, \text{ грн,} \quad (4.7)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год. $С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 272{,}3 \cdot 10 = 2723 \text{ грн.}$$

Отже витрати на створення ПЗ $К_{ПО}$ дорівнюють:

$$К_{ПО} = 200320 + 2723 = 203043 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$\text{Т} = \frac{t}{B_k \cdot F_p}, \text{ міс,} \quad (4.8)$$

де B_k – число виконавців, F_p- місячний фонд робочого класу (при 40 годинному робочому тижні F_p= 176 годин).

$$\text{Т} = \frac{626}{1 \cdot 176} = 3{,}6 \text{ , міс.}$$

Таким чином було розраховано наступні показники:

- Трудомісткість розробки ПЗ – 626 людино-годин;
- Витрати на створення – 203043 грн;
- Очікуваний період створення ПЗ – 3,6 міс.

4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту

Доцільність розробки програмного продукту для вирішення задачі визначення GPS координат об’єктів на фото, обґрунтовується її важливістю для журналістів та медіакриміналістів при отриманні та пошуку найбільш ймовірних локацій об’єкта на фотографії. Застосунок може бути розповсюджений серед молоді та часто подорожуючих людей, які можуть використовувати його для знаходження нових цікавих місць які були знайдені в інтернеті без геомітки.

Новизна отриманих результатів дипломної роботи визначається тим, що полягає в розробці та застосуванні сучасних методів комп’ютерного зору і

програмного забезпечення для моделювання та навчання нейронних мер еж в задачах класифікації регіонів земної кулі, що дозволяє знаходити географічні координати об’єктів.

Практична цінність результатів полягає в обґрунтуванні та розв’язанні проблем медіакриміналістики, що дає можливість більш точно, автоматизовано і зручно знаходити певні об’єкти, які були зображені на різноманітних вхідних зображеннях. Також дуже важливим є використання нового способу дроблення географічної земної кулі на клітини.

Область застосування. Розроблене програмне забезпечення може застосовуватися для вирішення проблем медіакриміналістики, таких як маніпулювання змістом та метаданими.

Значення роботи та висновки. Розроблена система дозволяє ефективно знаходити цільові об’єкти на зображенні зі значним скороченням матеріальних та часових витрат, підвищити точність визначення їх координат.

Практична цінність ПЗ полягає в тому, що методи і моделі, які використовуються в дипломній роботі дозволяють більш ефективно знаходити географічні координати об’єктів на фотографії. Це матиме значний вплив на реальні журналістські розслідування або роботу медіакриміналістів, наприклад збільшить їх точність та дозволить зменшити витрати на перевірку гіпотез. Також, використання розробленого програмного продукту дозволить зменшити витрати часу на вирішення задач пошуку ймовірних локацій.

Найближчим конкуртом розробленого ПЗ є пошукова система компанії Google, але вона має меншу точність та не дозволяє отримувати координати зображеного місця, а фактично тільки шукає схоже зображеняя. Розроблене ПЗ є універсальним, використовується для отримання координат об’єктів зображених на фотографії та не потребує високої кваліфікації користувача у сферах нейронних мереж або математики, що дозволяє залучати до роботи менш кваліфіковані ресурси і відповідно зменшити витрати.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

Оскільки кваліфікаційна робота має дослідницький характер, а розроблене програмне забезпечення створене з метою визначення об’єктів зображених на фотографії, чисельний розрахунок економічного ефекту не може бути виконаний. Однак можливо виділити соціальний ефект від реалізації результатів роботи, а саме:

- прискорення часу потрібного на розслідування медіа інцидентів;
- збільшення точності результатів визначення геолокації у порівнянні з кількістю локацій яку може визначити людина.

Висновок: у ході виконання роботи над даним економічним розділом, було визначено трудомісткість розробки програмного забезпечення (626 людино-годин), був проведений підрахунок витрат на створення програми (203043 гривень) та визначено очікуваний період створення ПЗ (3,6 місяці). Було проведено маркетинговий аналіз ринку збуту програмного продукту та визначено соціальний ефект від впровадження програмного продукту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблене програмне забезпечення для визначення географічних координат об’єктів на зображенні з використанням методів, які дозволяють отримати найбільш точний результат для зображення. За допомогою значень цільових метрик програмного забезпечення було проведене порівняння його зі схожими системами.

У цій роботі створено надійну архітектуру для великомасштабного прогнозування геолокації об’єкта на конкретному знімку з використанням лише піксельної інформації. Методика навчання, архітектура показали точний результат у цьому завданні. Крім того, було запроваджено метод розгляду задачі як задачу класифікації за допомогою кривої заповнення простору S2 для представлення кожної частини земної кулі у вигляді клітинки. Така система підходить для навчання в різних середовищах і оточеннях, використовуючи оптимальний класифікатор для оцінки гео-локації. У майбутньому дослідженні планується визначити, яка додаткова контекстна інформація може допомогти підвищити точність моделі на зібраних даних.

Для досягнення поставленої мети були виконані задачі:

– дослідити методи, які дозволяють визначати географічні координати об’єктів на фото;

– розробити програмне забезпечення, яке знаходить ймовірності локацій об’єктів на фото;

– зробити аналіз залежності отриманого рішення та розрахувати основні метрики навчання.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Меламед И.И., Сергеев С.И., Сигал И.Х. Задача коммивояжера. Вопросы теории // Автоматика и телемеханика. – № 9. – 1989. – С. 3-33.

2. Сесекин А.Н., Ченцов А.А., Ченцов А.Г. Обобщенная задача курьера с функцией затрат, зависящей от списка заданий // Известия российской академии наук. теория и системы управления. – Российская академия наук. – № 2. – 2010. – С. 68-77.

3. Chisman J.A. The clustered traveling salesman problem // Computers & Operations Research. – Volume 2. – Issue 2. – September 1975. – P. 115-119.

4. Bektas T. The multiple traveling salesman problem: an overview of formulations and solution procedures // Omega. – Elsevier. – № 34. – 2006. – P. 209-219.

5. Gutin G., Punnen A. P. The traveling salesman problem and its variations // Springer. – USA. – 2006. – 830 p.

6. Current J.R., Schilling D.A. The covering salesman problem // Transportation science. – Volume 23. – № 3. – 1989. – P. 151-229.

7. Nygard K.E., Yang C.H. Genetic algorithm for the traveling salesman problem with time windows // Computer Science and Operations Research: New Developments in their Interfaces. – Elsevier. – 2014. – P. 411-423

8. Mauricio Resende G.C., Ribeiro Celso C. A short tour of combinatorial optimization and computational complexity // Optimization by GRASP. – 2016. – P. 13-39.

9. Сергиенко И. В. Математические модели и методы решения задач дискретной оптимизации // Киев. – Наукова думка. – 1988. – 472 с.

10. Korte B., Vygen J. Combinatorial optimations: Theory and algorithms // Algorithms and Combinatorics. – Springer. – 2011. – Volume 21. – 659 p.

11. Ebert T., Fischer T., Belz J., Heinz T. O., Kampmann G., Nelles O. Extended Deterministic Local Search Algorithm for Maximin Latin Hypercube

Designs // 2015 IEEE Symposium Series on Computational Intelligence. – 2015. – P. 375-382.

12. Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by Simulated Annealing // Science. – Volume 220. - № 4598. – 1983. – P.671-680.

13. Kheiri A., Özcan E., Parkes A. J. A stochastic local search algorithm with adaptive acceptance for high-school timetabling // Annals of Operations Research. – Springer. – Volume 239(1). – 2016. – P.135-151.

14. Geng X., Chen Zh., Yang W., Shi D., Zhao K. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search // Applied Soft Computing. – Elsevier. – Volume 11. – 2011. – P.3680.

15. Holland J. H. Genetic algorithms and the optimal allocation of trials // SIAM Journal on Computing. – Volume 2. - №2. – 1973. – P.88-105.

16. Yu W., Li B., Jia H., Zhang M., Wang D. Application of multi- objective genetic algorithm to optimize energy efficiency and thermal comfort in building design // Energy and Buildings. – Volume 88. – 2015. – P.135-143.

17. Merz P., Freisleben B. Memetic Algorithms for the Traveling Salesman Problem // Complex Systems. – 2001. - № 13. – P. 297-345.

18. Cattaruzza D., Absi N., Feillet D., Vidal T. A memetic algorithm for the multi trip vehicle routing problem // European Journal of Operational Research. Volume 236. – Issue 3. – 2014. – P.833-848.

19. Colorni A., Dorigo M., Maniezzo V. Distributed optimization by ant colonies // Proceeding of ECAL91. – Elsevier Publishing. – P.134-142.

20. Liao T., Stützle T., Oca M. A. M. de, Dorigo M. A unified ant colony optimization algorithm for continuous optimization // European Journal of Operational Research. – Volume 234. – 2014. – P.597-609.

21. Wang Z., Xing H., Li T., Yang Y., Qu R., Pan Y. A modified ant colony optimization algorithm for network coding resource minimization // IEEE Transactions on Evolutionary Computation. – Volume 20. – Issue 3. – 2016. – P.325-342.

22. Гуляницький Л.Ф., Мулеса О.Ю. До класифікації метаевристик // XXI Всеукраїнська наукова конференція «Сучасні проблеми прикладної математики та інформатики». – Львів. – 2015. – С.139-142.

23. Методичні рекомендації до виконання кваліфікаційних робіт здобувачами другого (магістерського) рівня вищої освіти спеціальностей 121 «Інженерія програмного забезпечення» та 122 «Комп'ютерні науки» / Б.І. Мороз, О.В. Іванченко, О.В. Реута, О.С. Шевцова; М-во освіти і науки України, Нац. техн. ун-т "Дніпровська політехніка". – Дніпро : НТУ «ДП», 2021. – 87 с.

Додаток А

КОД ПРОГРАМИ

```
import os
import sys
import re
from math import ceil
from typing import Dict, List, Tuple, Union
from io import BytesIO
import random
from pathlib import Path
from multiprocessing import Pool

import pandas as pd
from PIL import Image
import torchvision
import torch
import msgpack


class
MsgPackIterableDatasetMultiTargetWithDynLabels(torch.utils.data.IterableDataset)
:
    """
    Data source: bunch of msgpack files
    Target values are generated on the fly given a mapping (id->[target1,
target, ...])
    """

    def __init__(
        self,
        path: str,
        target_mapping: Dict[str, int],
        key_img_id: str = "id",
        key_img_encoded: str = "image",
        transformation=None,
        shuffle=True,
        meta_path=None,
        cache_size=6 * 4096,
        lat_key="LAT",
        lon_key="LON",
    ):

        super(MsgPackIterableDatasetMultiTargetWithDynLabels, self).__init__()
        self.path = path
        self.cache_size = cache_size
        self.transformation = transformation
        self.shuffle = shuffle
        self.seed = random.randint(1, 100)
        self.key_img_id = key_img_id.encode("utf-8")
        self.key_img_encoded = key_img_encoded.encode("utf-8")
        self.target_mapping = target_mapping

        for k, v in self.target_mapping.items():
            if not isinstance(v, list):
                self.target_mapping[k] = [v]
        if len(self.target_mapping) == 0:
            raise ValueError("No samples found.")

        if not isinstance(self.path, (list, set)):
```

```
            self.path = [self.path]

        self.meta_path = meta_path
        if meta_path is not None:
            self.meta = pd.read_csv(meta_path, index_col=0)
            self.meta = self.meta.astype({lat_key: "float32", lon_key:
"float32"})
            self.lat_key = lat_key
            self.lon_key = lon_key

        self.shards = self.__init_shards(self.path)
        self.length = len(self.target_mapping)

    @staticmethod
    def __init_shards(path: Union[str, Path]) -> list:
        shards = []
        for i, p in enumerate(path):
            shards_re = r"shard_(\d+).msg"
            shards_index = [
                int(re.match(shards_re, x).group(1))
                for x in os.listdir(p)
                if re.match(shards_re, x)
            ]
            shards.extend(
                [
                    {
                        "path_index": i,
                        "path": p,
                        "shard_index": s,
                        "shard_path": os.path.join(p, f"shard_{s}.msg"),
                    }
                    for s in shards_index
                ]
            )
        if len(shards) == 0:
            raise ValueError("No shards found")
        return shards

    def _process_sample(self, x):
        # prepare image and target value

        # decode and initial resize if necessary
        img = Image.open(BytesIO(x[self.key_img_encoded]))
        if img.mode != "RGB":
            img = img.convert("RGB")

        if img.width > 320 and img.height > 320:
            img = torchvision.transforms.Resize(320)(img)

        # apply all user specified image transformations
        img = self.transformation(img)
        if self.meta_path is None:
            return img, x["target"]
        else:
            _id = x[self.key_img_id].decode("utf-8")
            meta = self.meta.loc[_id]
            return img, x["target"], meta[self.lat_key], meta[self.lon_key]

    def __iter__(self):

        shard_indices = list(range(len(self.shards)))

        if self.shuffle:
```

```
            random.seed(self.seed)
            random.shuffle(shard_indices)

        worker_info = torch.utils.data.get_worker_info()

        if worker_info is not None:

            def split_list(alist, splits=1):
                length = len(alist)
                return [
                    alist[i * length // splits : (i + 1) * length // splits]
                    for i in range(splits)
                ]

            shard_indices_split = split_list(shard_indices,
worker_info.num_workers)[
                worker_info.id
            ]

        else:
            shard_indices_split = shard_indices

        cache = []

        for shard_index in shard_indices_split:
            shard = self.shards[shard_index]

            with open(
                os.path.join(shard["path"],
f"shard_{shard['shard_index']}.msg"), "rb"
            ) as f:
                unpacker = msgpack.Unpacker(
                    f, max_buffer_size=1024 * 1024 * 1024, raw=True
                )
                for x in unpacker:
                    if x is None:
                        continue

                    # valid dataset sample?
                    _id = x[self.key_img_id].decode("utf-8")
                    try:
                        # set target value dynamically
                        if len(self.target_mapping[_id]) == 1:
                            x["target"] = self.target_mapping[_id][0]
                        else:
                            x["target"] = self.target_mapping[_id]
                    except KeyError:
                        # reject sample
                        # print(f'reject {_id} {type(_id)}')
                        continue

                    if len(cache) < self.cache_size:
                        cache.append(x)

                    if len(cache) == self.cache_size:

                        if self.shuffle:
                            random.shuffle(cache)
                        while cache:
                            yield self._process_sample(cache.pop())
        if self.shuffle:
            random.shuffle(cache)
```

```
        while cache:
            yield self._process_sample(cache.pop())

    def __len__(self):
        return self.length


class FiveCropImageDataset(torch.utils.data.Dataset):
    def __init__(
        self,
        meta_csv: Union[str, Path, None],
        image_dir: Union[str, Path],
        img_id_col: Union[str, int] = "img_id",
        allowed_extensions: List[str] = ["jpg", "jpeg", "png"]
    ):
        if isinstance(image_dir, str):
            image_dir = Path(image_dir)
        self.image_dir = image_dir
        self.img_id_col = img_id_col
        self.meta_info = None
        if meta_csv is not None:
            print(f"Read {meta_csv}")
            self.meta_info = pd.read_csv(meta_csv)
            self.meta_info.columns = map(str.lower, self.meta_info.columns)
            # rename column names if necessary to use existing data
            if "lat" in self.meta_info.columns:
                self.meta_info.rename(columns={"lat": "latitude"}, inplace=True)
            if "lon" in self.meta_info.columns:
                self.meta_info.rename(columns={"lon": "longitude"},
inplace=True)
            self.meta_info["img_path"] = self.meta_info[img_id_col].apply(
                lambda img_id: str(self.image_dir / img_id)
            )
        else:
            image_files = []
            for ext in allowed_extensions:
                image_files.extend([str(p) for p in
self.image_dir.glob(f"**/*.{ext}")])
            self.meta_info = pd.DataFrame(image_files, columns=["img_path"])
            self.meta_info[self.img_id_col] = self.meta_info["img_path"].apply(
                lambda x: Path(x).stem
            )
        self.tfm = torchvision.transforms.Compose(
            [
                torchvision.transforms.ToTensor(),
                torchvision.transforms.Normalize(
                    (0.485, 0.456, 0.406), (0.229, 0.224, 0.225)
                ),
            ]
        )

    def __len__(self):
        return len(self.meta_info.index)

    def __getitem__(self, idx) -> Tuple[torch.Tensor, dict]:
        meta = self.meta_info.iloc[idx]
        meta = meta.to_dict()
        meta["img_id"] = meta[self.img_id_col]

        image = Image.open(meta["img_path"]).convert("RGB")
        image = torchvision.transforms.Resize(256)(image)
        crops = torchvision.transforms.FiveCrop(224)(image)
        crops_transformed = []
```

```
from argparse import ArgumentParser

from pathlib import Path
from math import ceil
import pandas as pd
import torch
from tqdm.auto import tqdm

from classification.train_base import MultiPartitioningClassifier
from classification.dataset import FiveCropImageDataset


def parse_args():
    args = ArgumentParser()
    args.add_argument(
        "--checkpoint",
        type=Path,
        default=Path("models/base_M/epoch=014-val_loss=18.4833.ckpt"),
        help="Checkpoint to already trained model (*.ckpt)",
    )
    args.add_argument(
        "--hparams",
        type=Path,
        default=Path("models/base_M/hparams.yaml"),
        help="Path to hparams file (*.yaml) generated during training",
    )
    args.add_argument(
        "--image_dir",
        type=Path,
        default=Path("resources/images/im2gps"),
        help="Folder containing images. Supported file extensions: (*.jpg,
*.jpeg, *.png)",
    )
    # environment
    args.add_argument(
        "--gpu",
        action="store_true",
        help="Use GPU for inference if CUDA is available",
    )
    args.add_argument("--batch_size", type=int, default=64)
    args.add_argument(
        "--num_workers",
        type=int,
        default=4,
        help="Number of workers for image loading and pre-processing",
    )
    return args.parse_args()


args = parse_args()

print("Load model from ", args.checkpoint)
model = MultiPartitioningClassifier.load_from_checkpoint(
    checkpoint_path=str(args.checkpoint),
    hparams_file=str(args.hparams),
    map_location=None,
)
model.eval()
if args.gpu and torch.cuda.is_available():
    model.cuda()

print("Init dataloader")
dataloader = torch.utils.data.DataLoader(
```

```
    FiveCropImageDataset(meta_csv=None, image_dir=args.image_dir),
    batch_size=ceil(args.batch_size / 5),
    shuffle=False,
    num_workers=args.num_workers,
)
print("Number of images: ", len(dataloader.dataset))
if len(dataloader.dataset) == 0:
    raise RuntimeError(f"No images found in {args.image_dir}")

rows = []
for X in tqdm(dataloader):
    if args.gpu:
        X[0] = X[0].cuda()
    img_paths, pred_classes, pred_latitudes, pred_longitudes =
model.inference(X)
    for p_key in pred_classes.keys():
        for img_path, pred_class, pred_lat, pred_lng in zip(
            img_paths,
            pred_classes[p_key].cpu().numpy(),
            pred_latitudes[p_key].cpu().numpy(),
            pred_longitudes[p_key].cpu().numpy(),
        ):
            rows.append(
                {
                    "img_id": Path(img_path).stem,
                    "p_key": p_key,
                    "pred_class": pred_class,
                    "pred_lat": pred_lat,
                    "pred_lng": pred_lng,
                }
            )
df = pd.DataFrame.from_records(rows)
df.set_index(keys=["img_id", "p_key"], inplace=True)
print(df)
fout = Path(args.checkpoint).parent / f"inference_{args.image_dir.stem}.csv"
print("Write output to", fout)
df.to_csv(fout)
```

```
from pathlib import Path

from typing import List
import logging

import numpy as np
import pandas as pd
import s2sphere as s2


def print_partitioning_stats(partitionings):

    unique_classes = set()
    for p in partitionings:
        logging.info(f"{p.shortname} - Number of classes: {len(p)}")
        classes = p._df[p._col_class_label].tolist()
        unique_classes = unique_classes.union(classes)
    logging.info(f"Unique classes: {len(unique_classes)}")


class Partitioning:
    def __init__(
        self,
        csv_file: Path,
        shortname=None,
        skiprows=None,
        index_col="class_label",
        col_class_label="hex_id",
        col_latitute="latitude_mean",
        col_longitude="longitude_mean",
    ):

        """
        Required information in CSV:
            - class_indexes from 0 to n
            - respective class labels i.e. hexid
            - latitude and longitude
        """

        logging.info(f"Loading partitioning from file: {csv_file}")
        self._df = pd.read_csv(csv_file, index_col=index_col, skiprows=skiprows)
        self._df = self._df.sort_index()

        self._nclasses = len(self._df.index)
        self._col_class_label = col_class_label
        self._col_latitude = col_latitute
        self._col_longitude = col_longitude

        # map class label (hexid) to index
        self._label2index = dict(
            zip(self._df[self._col_class_label].tolist(), list(self._df.index))
        )

        self.name = csv_file.stem  # filename without extension
        if shortname:
            self.shortname = shortname
        else:
            self.shortname = self.name

    def __len__(self):
        return self._nclasses

    def __repr__(self):
```

```
        return f"{self.name} short: {self.shortname} n: {self._nclasses}"

    def get_class_label(self, idx):
        return self._df.iloc[idx][self._col_class_label]

    def get_lat_lng(self, idx):
        x = self._df.iloc[idx]
        return float(x[self._col_latitude]), float(x[self._col_longitude])

    def contains(self, class_label):
        if class_label in self._label2index:
            return True
        return False

    def label2index(self, class_label):
        try:
            return self._label2index[class_label]
        except KeyError as e:
            raise KeyError(f"unkown label {class_label} in {self}")


class Hierarchy:
    def __init__(self, partitionings: List[Partitioning]):

        """
        Provide a matrix of class indices where each class of the finest
partitioning will be assigned
        to the next coarser scales.

        Resulting index matrix M has shape: max(classes) * |partitionings| and
is ordered from coarse to fine
        """
        self.partitionings = partitionings

        print_partitioning_stats(self.partitionings)

        self.M = self.__build_hierarchy()

    def __build_hierarchy(self):
        def _hextobin(hexval):
            thelen = len(hexval) * 4
            binval = bin(int(hexval, 16))[2:]
            while (len(binval)) < thelen:
                binval = "0" + binval

            binval = binval.rstrip("0")
            return binval

        def _create_cell(lat, lng, level):
            p1 = s2.LatLng.from_degrees(lat, lng)
            cell = s2.Cell.from_lat_lng(p1)
            cell_parent = cell.id().parent(level)
            hexid = cell_parent.to_token()
            return hexid

        cell_hierarchy = []

        finest_partitioning = self.partitionings[-1]
        logging.info("Create hierarchy from partitionings...")
        if len(self.partitionings) > 1:
            # loop through finest partitioning
            for c in range(len(finest_partitioning)):
                cell_bin = _hextobin(self.partitionings[-1].get_class_label(c))
```

```
            level = int(len(cell_bin[3:-1]) / 2)
            parents = []

            # get parent cells
            for l in reversed(range(2, level + 1)):
                lat, lng = finest_partitioning.get_lat_lng(c)
                hexid_parent = _create_cell(lat, lng, l)
                # to coarsest partitioning
                for p in reversed(range(len(self.partitionings))):
                    if self.partitionings[p].contains(hexid_parent):
                        parents.append(
                            self.partitionings[p].label2index(hexid_parent)
                        )

                if len(parents) == len(self.partitionings):
                    break

            cell_hierarchy.append(parents[::-1])
    logging.info("Finished.")
    M = np.array(cell_hierarchy, dtype=np.int32)
    assert max([len(p) for p in self.partitionings]) == M.shape[0]
    assert len(self.partitionings) == M.shape[1]
    logging.debug(M)
    logging.info(f"M={M.shape}")
    return M
```

```
from argparse import ArgumentParser

from math import ceil
from pathlib import Path

import torch
import pytorch_lightning as pl
import pandas as pd

from classification.train_base import MultiPartitioningClassifier
from classification.dataset import FiveCropImageDataset


def parse_args():
    args = ArgumentParser()
    # model
    args.add_argument(
        "--checkpoint",
        type=Path,
        default=Path("models/base_M/epoch=014-val_loss=18.4833.ckpt"),
        help="Checkpoint to already trained model (*.ckpt)",
    )
    args.add_argument(
        "--hparams",
        type=Path,
        default=Path("models/base_M/hparams.yaml"),
        help="Path to hparams file (*.yaml) generated during training",
    )
    # testsets
    args.add_argument(
        "--image_dirs",
        nargs="+",
        default=["resources/images/im2gps", "resources/images/im2gps3k"],
        help="Whitespace separated list of image folders to evaluate",
    )
    args.add_argument(
        "--meta_files",
        nargs="+",
        default=[
            "resources/images/im2gps_places365.csv",
            "resources/images/im2gps3k_places365.csv",
        ],
        help="Whitespace separated list of respective meta data (ground-truth
GPS positions). Required columns: 'IMG_ID,LAT,LON' or 'img_id, latitude,
longitude'",
    )
    # environment
    args.add_argument(
        "--gpu",
        action="store_true",
        help="Use GPU for inference if CUDA is available",
    )
    args.add_argument(
        "--precision",
        type=int,
        default=32,
        help="Full precision (32), half precision (16)",
    )
    args.add_argument("--batch_size", type=int, default=64)
    args.add_argument(
        "--num_workers",
        type=int,
        default=4,
```

```
        help="Number of workers for image loading and pre-processing",
    )
    return args.parse_args()


args = parse_args()
print("Load model from checkpoint", args.checkpoint)
model = MultiPartitioningClassifier.load_from_checkpoint(
    checkpoint_path=str(args.checkpoint),
    hparams_file=str(args.hparams),
    map_location=None,
)

if args.gpu and torch.cuda.is_available():
    args.gpu = 1
else:
    args.gpu = None
trainer = pl.Trainer(gpus=args.gpu, precision=args.precision)

print("Init Testsets")
dataloader = []
for image_dir, meta_csv in zip(args.image_dirs, args.meta_files):
    dataset = FiveCropImageDataset(meta_csv, image_dir)
    dataloader.append(
        torch.utils.data.DataLoader(
            dataset,
            batch_size=ceil(args.batch_size / 5),
            shuffle=False,
            num_workers=args.num_workers,
        )
    )
print("Testing")
r = trainer.test(model, test_dataloaders=dataloader, verbose=False)
# formatting results
dfs = []
if len(args.image_dirs) > 1:
    r = r[0].values()
for results, name in zip(r, [Path(x).stem for x in args.image_dirs]):
    df = pd.DataFrame(results).T
    df["dataset"] = name
    df["partitioning"] = df.index
    df["partitioning"] = df["partitioning"].apply(lambda x: x.split("/")[-1])
    df.set_index(keys=["dataset", "partitioning"], inplace=True)
    dfs.append(df)

df = pd.concat(dfs)

print(df)
fout = Path(args.checkpoint).parent / ("test-" + "_".join(
    [str(Path(x).stem) for x in args.image_dirs]) + ".csv")
print("Write to", fout)
df.to_csv(fout)
from argparse import Namespace, ArgumentParser
from datetime import datetime
import json
import logging
from pathlib import Path

import yaml
import torch
import torchvision
import pytorch_lightning as pl
import pandas as pd
```

```
from classification import utils_global
from classification.s2_utils import Partitioning, Hierarchy
from classification.dataset import
MsgPackIterableDatasetMultiTargetWithDynLabels


class MultiPartitioningClassifier(pl.LightningModule):
    def __init__(self, hparams: Namespace):
        super().__init__()
        self.hparams = hparams

        self.partitionings, self.hierarchy = self.__init_partitionings()
        self.model, self.classifier = self.__build_model()

    def __init_partitionings(self):

        partitionings = []
        for shortname, path in zip(
            self.hparams.partitionings["shortnames"],
            self.hparams.partitionings["files"],
        ):
            partitionings.append(Partitioning(Path(path), shortname,
skiprows=2))

        if len(self.hparams.partitionings["files"]) == 1:
            return partitionings, None

        return partitionings, Hierarchy(partitionings)

    def __build_model(self):
        logging.info("Build model")
        model, nfeatures = utils_global.build_base_model(self.hparams.arch)

        classifier = torch.nn.ModuleList(
            [
                torch.nn.Linear(nfeatures, len(self.partitionings[i]))
                for i in range(len(self.partitionings))
            ]
        )

        if self.hparams.weights:
            logging.info("Load weights from pre-trained model")
            model, classifier = utils_global.load_weights_if_available(
                model, classifier, self.hparams.weights
            )

        return model, classifier

    def forward(self, x):
        fv = self.model(x)
        yhats = [self.classifier[i](fv) for i in range(len(self.partitionings))]
        return yhats

    def training_step(self, batch, batch_idx, optimizer_idx=None):
        images, target = batch

        if not isinstance(target, list) and len(target.shape) == 1:
            target = [target]

        # forward pass
        output = self(images)
```

```
        # individual losses per partitioning
        losses = [
            torch.nn.functional.cross_entropy(output[i], target[i])
            for i in range(len(output))
        ]

        loss = sum(losses)

        # stats
        losses_stats = {
            f"loss_train/{p}": l
            for (p, l) in zip([p.shortname for p in self.partitionings], losses)
        }
        for metric_name, metric_value in losses_stats.items():
            self.log(metric_name, metric_value, prog_bar=True, logger=True)
        self.log("train_loss", loss, prog_bar=True, logger=True)
        return {"loss": loss, **losses_stats}

    def validation_step(self, batch, batch_idx):
        images, target, true_lats, true_lngs = batch

        if not isinstance(target, list) and len(target.shape) == 1:
            target = [target]

        # forward
        output = self(images)

        # loss calculation
        losses = [
            torch.nn.functional.cross_entropy(output[i], target[i])
            for i in range(len(output))
        ]

        loss = sum(losses)

        # log top-k accuracy for each partitioning
        individual_accuracy_dict = utils_global.accuracy(
            output, target, [p.shortname for p in self.partitionings]
        )
        # log loss for each partitioning
        individual_loss_dict = {
            f"loss_val/{p}": l
            for (p, l) in zip([p.shortname for p in self.partitionings], losses)
        }

        # log GCD error@km threshold
        distances_dict = {}

        if self.hierarchy is not None:
            hierarchy_logits = [
                yhat[:, self.hierarchy.M[:, i]] for i, yhat in enumerate(output)
            ]
            hierarchy_logits = torch.stack(hierarchy_logits, dim=-1,)
            hierarchy_preds = torch.prod(hierarchy_logits, dim=-1)

        pnames = [p.shortname for p in self.partitionings]
        if self.hierarchy is not None:
            pnames.append("hierarchy")
        for i, pname in enumerate(pnames):
            # get predicted coordinates
            if i == len(self.partitionings):
                i = i - 1
                pred_class_indexes = torch.argmax(hierarchy_preds, dim=1)
```

```
            else:
                pred_class_indexes = torch.argmax(output[i], dim=1)
            pred_latlngs = [
                self.partitionings[i].get_lat_lng(idx)
                for idx in pred_class_indexes.tolist()
            ]
            pred_lats, pred_lngs = map(list, zip(*pred_latlngs))
            pred_lats = torch.tensor(pred_lats, dtype=torch.float)
            pred_lngs = torch.tensor(pred_lngs, dtype=torch.float)
            # calculate error
            distances = utils_global.vectorized_gc_distance(
                pred_lats,
                pred_lngs,
                true_lats.type_as(pred_lats),
                true_lngs.type_as(pred_lats),
            )
            distances_dict[f"gcd_{pname}_val"] = distances

        output = {
            "loss_val/total": loss,
            **individual_accuracy_dict,
            **individual_loss_dict,
            **distances_dict,
        }
        return output

    def validation_epoch_end(self, outputs):
        pnames = [p.shortname for p in self.partitionings]

        # top-k accuracy and loss per partitioning
        loss_acc_dict = utils_global.summarize_loss_acc_stats(pnames, outputs)

        # GCD stats per partitioning
        gcd_dict = utils_global.summarize_gcd_stats(pnames, outputs,
self.hierarchy)

        metrics = {
            "val_loss": loss_acc_dict["loss_val/total"],
            **loss_acc_dict,
            **gcd_dict,
        }
        for metric_name, metric_value in metrics.items():
            self.log(metric_name, metric_value, logger=True)

    def _multi_crop_inference(self, batch):
        images, meta_batch = batch
        cur_batch_size = images.shape[0]
        ncrops = images.shape[1]

        # reshape crop dimension to batch
        images = torch.reshape(images, (cur_batch_size * ncrops,
*images.shape[2:]))

        # forward pass
        yhats = self(images)
        yhats = [torch.nn.functional.softmax(yhat, dim=1) for yhat in yhats]

        # respape back to access individual crops
        yhats = [
            torch.reshape(yhat, (cur_batch_size, ncrops, *list(yhat.shape[1:])))
            for yhat in yhats
        ]
```

```
        # calculate max over crops
        yhats = [torch.max(yhat, dim=1)[0] for yhat in yhats]

        hierarchy_preds = None
        if self.hierarchy is not None:
            hierarchy_logits = torch.stack(
                [yhat[:, self.hierarchy.M[:, i]] for i, yhat in
enumerate(yhats)],
                dim=-1,
            )
            hierarchy_preds = torch.prod(hierarchy_logits, dim=-1)

        return yhats, meta_batch, hierarchy_preds

    def inference(self, batch):

        yhats, meta_batch, hierarchy_preds = self._multi_crop_inference(batch)

        if self.hierarchy is not None:
            nparts = len(self.partitionings) + 1
        else:
            nparts = len(self.partitionings)

        pred_class_dict = {}
        pred_lat_dict = {}
        pred_lng_dict = {}
        for i in range(nparts):
            # get pred class indices
            if self.hierarchy is not None and i == len(self.partitionings):
                pname = "hierarchy"
                pred_classes = torch.argmax(hierarchy_preds, dim=1)
                i = i - 1
            else:
                pname = self.partitionings[i].shortname
                pred_classes = torch.argmax(yhats[i], dim=1)

            # calculate GCD
            pred_lats, pred_lngs = map(
                list,
                zip(
                    *[
                        self.partitionings[i].get_lat_lng(c)
                        for c in pred_classes.tolist()
                    ]
                ),
            )
            pred_lats = torch.tensor(pred_lats, dtype=torch.float)
            pred_lngs = torch.tensor(pred_lngs, dtype=torch.float)
            pred_lat_dict[pname] = pred_lats
            pred_lng_dict[pname] = pred_lngs
            pred_class_dict[pname] = pred_classes

        return meta_batch["img_path"], pred_class_dict, pred_lat_dict,
pred_lng_dict

    def test_step(self, batch, batch_idx, dataloader_idx=None):

        yhats, meta_batch, hierarchy_preds = self._multi_crop_inference(batch)

        distances_dict = {}
        if self.hierarchy is not None:
            nparts = len(self.partitionings) + 1
        else:
```

```
            nparts = len(self.partitionings)

        for i in range(nparts):
            # get pred class indices
            if self.hierarchy is not None and i == len(self.partitionings):
                pname = "hierarchy"
                pred_classes = torch.argmax(hierarchy_preds, dim=1)
                i = i - 1
            else:
                pname = self.partitionings[i].shortname
                pred_classes = torch.argmax(yhats[i], dim=1)

            # calculate GCD
            pred_lats, pred_lngs = map(
                list,
                zip(
                    *[
                        self.partitionings[i].get_lat_lng(c)
                        for c in pred_classes.tolist()
                    ]
                ),
            )
            pred_lats = torch.tensor(pred_lats, dtype=torch.float)
            pred_lngs = torch.tensor(pred_lngs, dtype=torch.float)

            distances = utils_global.vectorized_gc_distance(
                pred_lats,
                pred_lngs,
                meta_batch["latitude"].type_as(pred_lats),
                meta_batch["longitude"].type_as(pred_lngs),
            )
            distances_dict[pname] = distances

        return distances_dict

    def test_epoch_end(self, outputs):
        result = utils_global.summarize_test_gcd(
            [p.shortname for p in self.partitionings], outputs, self.hierarchy
        )
        return {**result}

    def configure_optimizers(self):

        optim_feature_extrator = torch.optim.SGD(
            self.parameters(), **self.hparams.optim["params"]
        )

        return {
            "optimizer": optim_feature_extrator,
            "lr_scheduler": {
                "scheduler": torch.optim.lr_scheduler.MultiStepLR(
                    optim_feature_extrator, **self.hparams.scheduler["params"]
                ),
                "interval": "epoch",
                "name": "lr",
            },
        }

    def train_dataloader(self):

        with open(self.hparams.train_label_mapping, "r") as f:
            target_mapping = json.load(f)
```

```
        tfm = torchvision.transforms.Compose(
            [
                torchvision.transforms.RandomHorizontalFlip(),
                torchvision.transforms.RandomResizedCrop(224, scale=(0.66,
1.0)),
                torchvision.transforms.ToTensor(),
                torchvision.transforms.Normalize(
                    (0.485, 0.456, 0.406), (0.229, 0.224, 0.225)
                ),
            ]
        )

        dataset = MsgPackIterableDatasetMultiTargetWithDynLabels(
            path=self.hparams.msgpack_train_dir,
            target_mapping=target_mapping,
            key_img_id=self.hparams.key_img_id,
            key_img_encoded=self.hparams.key_img_encoded,
            shuffle=True,
            transformation=tfm,
        )

        dataloader = torch.utils.data.DataLoader(
            dataset,
            batch_size=self.hparams.batch_size,
            num_workers=self.hparams.num_workers_per_loader,
            pin_memory=True,
        )
        return dataloader

    def val_dataloader(self):

        with open(self.hparams.val_label_mapping, "r") as f:
            target_mapping = json.load(f)

        tfm = torchvision.transforms.Compose(
            [
                torchvision.transforms.Resize(256),
                torchvision.transforms.CenterCrop(224),
                torchvision.transforms.ToTensor(),
                torchvision.transforms.Normalize(
                    (0.485, 0.456, 0.406), (0.229, 0.224, 0.225)
                ),
            ]
        )
        dataset = MsgPackIterableDatasetMultiTargetWithDynLabels(
            path=self.hparams.msgpack_val_dir,
            target_mapping=target_mapping,
            key_img_id=self.hparams.key_img_id,
            key_img_encoded=self.hparams.key_img_encoded,
            shuffle=False,
            transformation=tfm,
            meta_path=self.hparams.val_meta_path,
            cache_size=1024,
        )

        dataloader = torch.utils.data.DataLoader(
            dataset,
            batch_size=self.hparams.batch_size,
            num_workers=self.hparams.num_workers_per_loader,
            pin_memory=True,
        )

        return dataloader
```

```
def parse_args():
    args = ArgumentParser()
    args.add_argument("-c", "--config", type=Path,
default=Path("config/baseM.yml"))
    args.add_argument("--progbar", action="store_true")
    return args.parse_args()


def main():
    args = parse_args()
    logging.basicConfig(level=logging.INFO)

    with open(args.config) as f:
        config = yaml.load(f, Loader=yaml.FullLoader)

    model_params = config["model_params"]
    trainer_params = config["trainer_params"]

    utils_global.check_is_valid_torchvision_architecture(model_params["arch"])

    out_dir = Path(config["out_dir"]) / datetime.now().strftime("%y%m%d-%H%M")
    out_dir.mkdir(exist_ok=True, parents=True)
    logging.info(f"Output directory: {out_dir}")

    # init classifier
    model = MultiPartitioningClassifier(hparams=Namespace(**model_params))

    logger = pl.loggers.TensorBoardLogger(save_dir=str(out_dir), name="tb_logs")
    checkpoint_dir = out_dir / "ckpts" / "{epoch:03d}-{val_loss:.4f}"
    checkpointer = pl.callbacks.model_checkpoint.ModelCheckpoint(checkpoint_dir)

    progress_bar_refresh_rate = 0
    if args.progbar:
        progress_bar_refresh_rate = 1

    trainer = pl.Trainer(
        **trainer_params,
        logger=logger,
        val_check_interval=model_params["val_check_interval"],
        checkpoint_callback=checkpointer,
        progress_bar_refresh_rate=progress_bar_refresh_rate,
    )

    trainer.fit(model)


if __name__ == "__main__":
    main()
```

```
import logging
from collections import OrderedDict
from pathlib import Path
from typing import Union, List

import torch
import torchvision


def check_is_valid_torchvision_architecture(architecture: str):
    """Raises an ValueError if architecture is not part of available torchvision
models
    """
    available = sorted(
        name
        for name in torchvision.models.__dict__
        if name.islower()
        and not name.startswith("__")
        and callable(torchvision.models.__dict__[name])
    )
    if architecture not in available:
        raise ValueError(f"{architecture} not in {available}")


def build_base_model(arch: str):

    model = torchvision.models.__dict__[arch](pretrained=True)

    # get input dimension before classification layer
    if arch in ["mobilenet_v2"]:
        nfeatures = model.classifier[-1].in_features
        model = torch.nn.Sequential(*list(model.children())[:-1])
    elif arch in ["densenet121", "densenet161", "densenet169"]:
        nfeatures = model.classifier.in_features
        model = torch.nn.Sequential(*list(model.children())[:-1])
    elif "resne" in arch:
        # usually all ResNet variants
        nfeatures = model.fc.in_features
        model = torch.nn.Sequential(*list(model.children())[:-2])
    else:
        raise NotImplementedError

    model.avgpool = torch.nn.AdaptiveAvgPool2d(1)
    model.flatten = torch.nn.Flatten(start_dim=1)
    return model, nfeatures


def load_weights_if_available(
    model: torch.nn.Module, classifier: torch.nn.Module, weights_path:
Union[str, Path]
):

    checkpoint = torch.load(weights_path, map_location=lambda storage, loc:
storage)

    state_dict_features = OrderedDict()
    state_dict_classifier = OrderedDict()
    for k, w in checkpoint["state_dict"].items():
        if k.startswith("model"):
            state_dict_features[k.replace("model.", "")] = w
        elif k.startswith("classifier"):
            state_dict_classifier[k.replace("classifier.", "")] = w
        else:
```

```
            logging.warning(f"Unexpected prefix in state_dict: {k}")
    model.load_state_dict(state_dict_features, strict=True)
    return model, classifier


def vectorized_gc_distance(latitudes, longitudes, latitudes_gt, longitudes_gt):
    R = 6371
    factor_rad = 0.01745329252
    longitudes = factor_rad * longitudes
    longitudes_gt = factor_rad * longitudes_gt
    latitudes = factor_rad * latitudes
    latitudes_gt = factor_rad * latitudes_gt
    delta_long = longitudes_gt - longitudes
    delta_lat = latitudes_gt - latitudes
    subterm0 = torch.sin(delta_lat / 2) ** 2
    subterm1 = torch.cos(latitudes) * torch.cos(latitudes_gt)
    subterm2 = torch.sin(delta_long / 2) ** 2
    subterm1 = subterm1 * subterm2
    a = subterm0 + subterm1
    c = 2 * torch.asin(torch.sqrt(a))
    gcd = R * c
    return gcd


def gcd_threshold_eval(gc_dists, thresholds=[1, 25, 200, 750, 2500]):
    # calculate accuracy for given gcd thresolds
    results = {}
    for thres in thresholds:
        results[thres] = torch.true_divide(
            torch.sum(gc_dists <= thres), len(gc_dists)
        ).item()
    return results


def accuracy(output, target, partitioning_shortnames: list, topk=(1, 5, 10)):
    def _accuracy(output, target, topk=(1,)):
        """Computes the accuracy over the k top predictions for the specified
values of k"""
        with torch.no_grad():
            maxk = max(topk)
            batch_size = target.size(0)

            _, pred = output.topk(maxk, 1, True, True)
            pred = pred.t()
            correct = pred.eq(target.view(1, -1).expand_as(pred))

            res = {}
            for k in topk:
                correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)
                res[k] = correct_k / batch_size
            return res

    with torch.no_grad():
        out_dict = {}
        for i, pname in enumerate(partitioning_shortnames):
            res_dict = _accuracy(output[i], target[i], topk=topk)
            for k, v in res_dict.items():
                out_dict[f"acc{k}_val/{pname}"] = v

        return out_dict


def summarize_gcd_stats(pnames: List[str], outputs, hierarchy=None):
```

```
    gcd_dict = {}
    metric_names = [f"gcd_{p}_val" for p in pnames]
    if hierarchy is not None:
        metric_names.append("gcd_hierarchy_val")
    for metric_name in metric_names:
        distances_flat = [output[metric_name] for output in outputs]
        distances_flat = torch.cat(distances_flat, dim=0)
        gcd_results = gcd_threshold_eval(distances_flat)
        for gcd_thres, acc in gcd_results.items():
            gcd_dict[f"{metric_name}/{gcd_thres}"] = acc
    return gcd_dict


def summarize_test_gcd(pnames, outputs, hierarchy=None):
    def _eval(output):
        # calculate acc@km for a list of given thresholds
        accuracy_outputs = {}
        if hierarchy is not None:
            pnames.append("hierarchy")
        for pname in pnames:
            # concat batches of distances
            distances_flat = torch.cat([x[pname] for x in output], dim=0)
            # acc for all distances
            acc_dict = gcd_threshold_eval(distances_flat)
            accuracy_outputs[f"acc_test/{pname}"] = acc_dict
        return accuracy_outputs

    result = {}

    if isinstance(outputs[0], dict):  # only one testset
        result = _eval(outputs)
    elif isinstance(outputs[0], list):  # multiple testsets
        for testset_index, output in enumerate(outputs):
            result[testset_index] = _eval(output)
    else:
        raise TypeError

    return result


def summarize_loss_acc_stats(pnames: List[str], outputs, topk=[1, 5, 10]):

    loss_acc_dict = {}
    metric_names = []
    for k in topk:
        accuracy_names = [f"acc{k}_val/{p}" for p in pnames]
        metric_names.extend(accuracy_names)
    metric_names.extend([f"loss_val/{p}" for p in pnames])
    for metric_name in ["loss_val/total", *metric_names]:
        metric_total = 0
        for output in outputs:
            metric_value = output[metric_name]
            metric_total += metric_value
        loss_acc_dict[metric_name] = metric_total / len(outputs)
    return loss_acc_dict
```

```
from argparse import ArgumentParser
import logging
import os
import re
import json
from io import BytesIO
from pathlib import Path
from typing import Union

import yaml
import msgpack
import pandas as pd

import torch
import torchvision
from PIL import Image


class MsgPackIterableMetaDataset(torch.utils.data.IterableDataset):
    def __init__(
        self,
        msgpack_path: Union[str, Path],
        image_ids_path: Union[str, Path],
        meta_path: Union[str, Path],
        image_ids_index_col: Union[str, int] = 0,
        meta_index_col: Union[str, int] = 0,
        key_img_id: str = "id",
        key_img_encoded: str = "image",
        transformation=None,
        cache_size=4096,
        ignore_image=False,
    ):

        super(MsgPackIterableMetaDataset, self).__init__()
        self.path = msgpack_path
        self.cache_size = cache_size
        self.transformation = transformation

        self.key_img_id = key_img_id.encode("utf-8")
        self.key_img_encoded = key_img_encoded.encode("utf-8")
        self.ignore_image = ignore_image
        self.image_ids = self.__init_image_ids(image_ids_path,
image_ids_index_col)

        if not isinstance(self.path, (list, set)):
            self.path = [self.path]

        self.meta = pd.read_csv(meta_path, index_col=meta_index_col)

        if "LAT" in self.meta.columns:
            self.meta.rename(
                columns={"LAT": "latitude", "LON": "longitude"}, inplace=True
            )

        self.meta = self.meta.astype({"latitude": "float32", "longitude":
"float32"})
        logging.debug(self.meta)
        self.shards = self.__init_shards(self.path)

    @staticmethod
    def __init_image_ids(image_ids_path: Union[Path, str], index_col=0) -> set:
        """
        Args:
```

```
        image_ids_path: path to CSV
        index_col: column name or index with image ids

    Returns: set of image ids to filter
    """

    df = pd.read_csv(image_ids_path, index_col=index_col)
    logging.debug(df)
    image_ids = set(df.index.tolist())
    return image_ids

@staticmethod
def __init_shards(path: Union[str, Path]) -> list:
    shards = []
    for i, p in enumerate(path):
        shards_re = r"shard_(\d+).msg"
        shards_index = [
            int(re.match(shards_re, x).group(1))
            for x in os.listdir(p)
            if re.match(shards_re, x)
        ]
        shards.extend(
            [
                {
                    "path_index": i,
                    "path": p,
                    "shard_index": s,
                    "shard_path": os.path.join(p, f"shard_{s}.msg"),
                }
                for s in shards_index
            ]
        )
    if len(shards) == 0:
        raise ValueError("No shards found")
    return shards

def _process_sample(self, x):
    img = None
    if not self.ignore_image:
        # prepare image and meta_data
        # decode and initial resize if necessary
        img = Image.open(BytesIO(x[self.key_img_encoded]))
        if img.mode != "RGB":
            img = img.convert("RGB")

        if img.width > 320 and img.height > 320:
            img = torchvision.transforms.Resize(320)(img)

        # apply all user specified image transformations
        img = self.transformation(img)

    _id = x[self.key_img_id].decode("utf-8")
    meta = self.meta.loc[_id].to_dict()
    meta["img_id"] = _id
    return img, meta

def __iter__(self):

    shard_indices = list(range(len(self.shards)))

    worker_info = torch.utils.data.get_worker_info()

    if worker_info is not None:
```

```
            def split_list(alist, splits=1):
                length = len(alist)
                return [
                    alist[i * length // splits : (i + 1) * length // splits]
                    for i in range(splits)
                ]

            shard_indices_split = split_list(shard_indices,
worker_info.num_workers)[
                worker_info.id
            ]

        else:
            shard_indices_split = shard_indices

        cache = []

        for shard_index in shard_indices_split:
            shard = self.shards[shard_index]

            with open(
                os.path.join(shard["path"],
f"shard_{shard['shard_index']}.msg"), "rb"
            ) as f:
                unpacker = msgpack.Unpacker(
                    f, max_buffer_size=1024 * 1024 * 1024, raw=True
                )
                for x in unpacker:
                    if x is None:
                        continue

                    # valid dataset sample?
                    _id = x[self.key_img_id].decode("utf-8")
                    if _id not in self.image_ids:
                        continue

                    if len(cache) < self.cache_size:
                        cache.append(x)

                    if len(cache) == self.cache_size:
                        while cache:
                            yield self._process_sample(cache.pop())
        while cache:
            yield self._process_sample(cache.pop())


def main():

    for dataset_type in ["train", "val"]:
        with open(config[f"{dataset_type}_label_mapping"]) as f:
            mapping = json.load(f)

        logging.info(f"Expected dataset size: {len(mapping)}")
        msgpack_path = config[f"msgpack_{dataset_type}_dir"]
        image_ids_path = config[f"{dataset_type}_meta_path"]
        dataset = MsgPackIterableMetaDataset(
            msgpack_path,
            image_ids_path,
            image_ids_path,
            key_img_id=config["key_img_id"],
            key_img_encoded=config["key_img_encoded"],
            ignore_image=True,
```

```
        )

        filtered_mapping = {}
        for _, meta in dataset:
            if meta["img_id"] in mapping:
                filtered_mapping[meta["img_id"]] = mapping[meta["img_id"]]
        logging.info(f"True dataset size: {len(filtered_mapping)}")

        with open(config[f"{dataset_type}_label_mapping"], "w") as fw:
            json.dump(filtered_mapping, fw)
    return


def parse_args():
    parser = ArgumentParser()
    parser.add_argument("-c", "--config", type=Path, default="config/baseM.yml")
    args = parser.parse_args()
    return args


if __name__ == "__main__":

    logging.basicConfig(
        format="%(asctime)s %(levelname)s: %(message)s",
        datefmt="%d-%m-%Y %H:%M:%S",
        level=logging.INFO,
    )

    args = parse_args()

    with open(args.config) as f:
        config = yaml.load(f, Loader=yaml.FullLoader)

    config = config["model_params"]

    main()


import os
import argparse
import re
from typing import Union
from io import BytesIO
import random
from pathlib import Path

from PIL import Image
import torchvision
import torch
import msgpack


class MsgPackIterableDataset(torch.utils.data.IterableDataset):

    def __init__(
        self,
        path: str,
        key_img_id: str = "id",
        key_img_encoded: str = "image",
        transformation=None,
        shuffle=False,
        cache_size=6 * 4096,
    ):
```

```
        super(MsgPackIterableDataset, self).__init__()
        self.path = path
        self.cache_size = cache_size
        self.transformation = transformation
        self.shuffle = shuffle
        self.seed = random.randint(1, 100)
        self.key_img_id = key_img_id.encode("utf-8")
        self.key_img_encoded = key_img_encoded.encode("utf-8")

        if not isinstance(self.path, (list, set)):
            self.path = [self.path]

        self.shards = self.__init_shards(self.path)

    @staticmethod
    def __init_shards(path: Union[str, Path]) -> list:
        shards = []
        for i, p in enumerate(path):
            shards_re = r"shard_(\d+).msg"
            shards_index = [
                int(re.match(shards_re, x).group(1))
                for x in os.listdir(p)
                if re.match(shards_re, x)
            ]
            shards.extend(
                [
                    {
                        "path_index": i,
                        "path": p,
                        "shard_index": s,
                        "shard_path": os.path.join(p, f"shard_{s}.msg"),
                    }
                    for s in shards_index
                ]
            )
        if len(shards) == 0:
            raise ValueError("No shards found")

        return shards

    def _process_sample(self, x):
        # decode and initial resize if necessary
        img = Image.open(BytesIO(x[self.key_img_encoded]))
        if img.mode != "RGB":
            img = img.convert("RGB")

        if img.width > 320 and img.height > 320:
            img = torchvision.transforms.Resize(320)(img)

        # apply all user specified image transformations
        img = self.transformation(img)

        _id = x[self.key_img_id].decode("utf-8")
        return img, _id

    def __iter__(self):

        shard_indices = list(range(len(self.shards)))

        if self.shuffle:
            random.seed(self.seed)
            random.shuffle(shard_indices)
```

```
        worker_info = torch.utils.data.get_worker_info()

        if worker_info is not None:

            def split_list(alist, splits=1):
                length = len(alist)
                return [
                    alist[i * length // splits : (i + 1) * length // splits]
                    for i in range(splits)
                ]

            shard_indices_split = split_list(shard_indices,
worker_info.num_workers)[
                worker_info.id
            ]

        else:
            shard_indices_split = shard_indices

        cache = []

        for shard_index in shard_indices_split:
            shard = self.shards[shard_index]

            with open(
                os.path.join(shard["path"],
f"shard_{shard['shard_index']}.msg"), "rb"
            ) as f:
                unpacker = msgpack.Unpacker(
                    f, max_buffer_size=1024 * 1024 * 1024, raw=True
                )
                for x in unpacker:
                    if x is None:
                        continue

                    if len(cache) < self.cache_size:
                        cache.append(x)

                    if len(cache) == self.cache_size:

                        if self.shuffle:
                            random.shuffle(cache)
                        while cache:
                            yield self._process_sample(cache.pop())
        if self.shuffle:
            random.shuffle(cache)

        while cache:
            yield self._process_sample(cache.pop())


if __name__ == "__main__":

    args = argparse.ArgumentParser()
    args.add_argument("--data", type=str, default="resources/images/mp16")
    args = args.parse_args()

    tfm = torchvision.transforms.Compose(
        [
            torchvision.transforms.ToTensor(),
        ]
    )
```

```
dataset = MsgPackIterableDataset(path=args.data, transformation=tfm)
dataloader = torch.utils.data.DataLoader(
        dataset,
        batch_size=1,
        num_workers=6,
        pin_memory=False,
    )

num_images = 0
for x, image_id in dataloader:
    if num_images == 0:
        print(x.shape, image_id)
    num_images +=1

print(f"{num_images=}")
```

Додаток Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Керівника економічної частини Касьяненко Л.В., к.е.н., доцента каф. ПЕП та ПУ
(прізвище, ім'я, по батькові, вчене звання, посада, місце роботи)

На кваліфікаційну роботу

студента Тесленка Святослав Ігоровича
(прізвище, ім'я, по батькові)

курсу II групи 122м-20-2
спеціальності 122 «Комп'ютерні науки»
на тему Розробка інформаційно-аналітичної системи для визначення географічних координат об'єктів на зображенні за допомогою багатозадачних згорткових нейронних мереж.

«___»_____2022 р. _____
(підпис)

Додаток В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Teslenko.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Teslenko.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Teslenko.zip	Архів. Містить коди програми.
Презентація	
Teslenko.ppt	Презентація кваліфікаційної роботи.