НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Радіотехнічний факультет

Кафедра прикладної радіоелектроніки

«На правах рукопису»
УДК 004.932

До захисту допущено:
В.о. зав. кафедри

Андрій МОВЧАНЮК
«12» грудня 2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інтелектуальні технології радіоелектронної техніки»

зі спеціальності 172 Електронні комунікації та радіотехніка

на тему: «Розпізнавання місцезнаходження ворожих цілей на знімках з використанням засобів штучного інтелекту»

Виконав:
студент 2 курсу, групи РЕ-з31мп
Клюско Володимир Юрійович
(Прізвище ім'я по батькові)

Керівник
К.т.н., доцент Приходько Ірина Олександрівна
(Посада, науковий ступінь ,вчене звання)

Рецензент

Доцент, к.т.н. каф. РТС Головін Володимир Андрійович
(Посада, науковий ступінь ,вчене звання)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Радіотехнічний факультет
Кафедра прикладної радіоелектроніки

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 Електронні комунікації та радіотехніка

Освітньо-професійна програма «Інтелектуальні технології радіоелектронної техніки»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри

Андрій МОВЧАНЮК
«12» грудня 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студента
Клюска Володимира Юрійовича
(Прізвище ім'я по батькові)

1.Тема дисертації «Розпізнавання місцезнаходження ворожих цілей на знімках з використанням засобів штучного інтелекту»
науковий керівник дисертації к.т.н., доцент каф. ПРЕ Приходько І.О.
затверджені наказом по університету від «08» листопада 2024 р. № 5024-с

2. Термін подання студентом дисертації 12 грудня 2024 року

3. Об'єкт дослідження процеси аналізу та обробки геопросторових даних для автоматизованого розпізнавання та ідентифікації місцерозташування об'єктів

4. Вихідні дані: географічно марковані набори даних, набори даних для навчання нейронних мереж

5. Перелік завдань, які потрібно розробити сегментація об'єктів на зображеннях, класифікація зображень, геолокація (визначення координат), побудова та інтеграція багаторівневої архітектури, підготовка даних

6. Орієнтовний перелік графічного (ілюстративного) матеріалу презентація, додатки

7. Орієнтовний перелік публікацій немає.

9. Дата видачі завдання 05 вересня 2024 року

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання теми магістерської дисертації	05.09.2024	
2	Розробка плану магістерської дисертації	10.09. 2024	
3	Початок збору інформації та аналіз літератури	20.09.2024-04.10.2024	
4	Розробка технічного завдання	05.10.2024-09.10.2024	
5	Підбір та підготовка вихідних даних (пошук наборів даних)	10.10.2024-14.10.2024	
6	Пошук та тестування моделі сегментації	15.10.2024-24.10.2024	
7	Розробка моделі класифікації	25.10.2024-10.11.2024	
8	Розробка та тестування моделі прогнозування місцезнаходження об’єкту	11.11.2024-20.11.2024	
9	Тестування моделі на контрольних даних та аналіз результатів	21.11.2024-27.11.2024	
10	Оформлення магістерської дисертації	28.11.2024р.-01.12.2024	

Студент Володимир КЛЮСКО

Науковий керівник Ірина ПРИХОДЬКО

АНОТАЦІЯ

Текстова частина дисертації роботи містить: 96 сторінок, 51 рисунок, 26 посилань.

Мета і завдання дисертації: розробка та впровадження методів автоматизованого розпізнавання місцезнаходження ворожих цілей на знімках із використанням комбінованих засобів штучного інтелекту для підвищення точності, швидкості та ефективності виявлення об'єктів на зображеннях у реальному часі.

Для досягнення вказаної мети потрібно вирішити такі завдання:

- Дослідити сучасні методи розпізнавання об’єктів на зображеннях без використання штучного інтелекту;
- Проаналізувати ефективність існуючих алгоритмів штучного інтелекту для завдань виявлення ворожих цілей на знімках;
- Розробити метод автоматизованого збору, анотації та попередньої обробки зображень для навчання нейронної мережі з урахуванням специфіки військових цілей;
- Обрати та оптимізувати набір даних для навчання, валідації та тестування моделі;
- Реалізувати та протестувати моделі сегментації, класифікації та розпізнавання місцезнаходження з використанням комбінованих архітектур;
- Провести оптимізацію моделей для роботи в реальному часі;
- Провести оцінку точності та надійності роботи запропонованої моделі в різних умовах, включаючи зміни освітлення, зашумленість знімків та інші фактори.

Об’єкт дослідження - процеси розпізнавання та геолокації ворожих цілей на зображеннях з використанням комбінованих архітектур згорткових нейронних мереж.

Предмет дослідження - методи побудови, оптимізації та інтеграції комбінованого способу розпізнавання ворожих цілей на зображеннях з високою точністю та швидкістю.

Ключові слова: розпізнавання, геолокація, геопросторова розвідка, класифікація, згорткова нейронна мережа, набір даних, Vision Transformer.

Методи дослідження роботи: методичною основою дослідження є системне опрацювання та аналіз теоретичних і практичних підходів до розпізнавання об'єктів на зображеннях з використанням згорткових нейронних мереж та інших засобів штучного інтелекту. В процесі навчання моделей було проведено аналіз їх продуктивності та точності для оптимізації параметрів нейронної мережі з метою підвищення точності розпізнавання та інтеграції в єдину програмну структуру. Застосовано наступні методи: метод переднавчання для прискорення навчання моделі на нових наборах даних шляхом використання попередньо навчених моделей; метрики оцінки продуктивності моделі такі як точність (accuracy), повнота (recall), та точність передбачення (precision) для виявлення ефективності розпізнавання цілей; валідація та тестування даних з розбиттям їх на навчальний, валідаційний та тренувальний набори даних для забезпечення достовірності результатів; аугментація даних для збільшення кількості тренувальних зображень шляхом створення варіацій з існуючих даних.

ABSTRACT

The dissertation text part contains 86 pages, 51 figures, 26 references.

The aim and objectives of the study are to develop and implement methods for automated recognition of the location of enemy targets in images using combined artificial intelligence tools to improve the accuracy, speed, and efficiency of detecting objects in images in real time.

To achieve this goal, we need to solve the following tasks:

- Investigate modern methods of object recognition in images without the use of artificial intelligence;

- Analyze the effectiveness of existing artificial intelligence algorithms for the task of detecting enemy targets in images;

- Develop a method for automated image collection, annotation, and pre-processing for training a neural network with the specifics of military targets;

- Select and optimize a dataset for training, validation, and testing the model;

- Implement and test segmentation, classification, and location recognition models using combined architectures;

- Optimize models for real-time operation;

- Evaluate the accuracy and reliability of the proposed model in various conditions, including changes in lighting, image noise, and other factors.

The research object is the processes of recognizing and classifying enemy targets in images using combined architectures of convolutional neural networks.

The research subject is building methods, optimizing, and integrating a combined method for recognizing enemy targets in images with high accuracy and speed.

Keywords: recognition, geolocation, geospatial intelligence, classification, convolutional neural network, dataset, Vision Transformer.

Methods: The methodological basis of the research is a systematic study and analysis of theoretical and practical approaches to object recognition in images using convolutional neural networks and other artificial intelligence tools. In the process

of training the models, their performance and accuracy were analyzed to optimize the neural network parameters in order to improve the recognition accuracy and integrate them into a single software structure. The following methods were applied: pre-training method to accelerate model training on new data sets by using pre-trained models; metrics for evaluating model performance such as accuracy, recall, and prediction accuracy to determine the effectiveness of target recognition; validation and testing of data by splitting it into training, validation, and training data sets to ensure the reliability of the results; data augmentation to increase the number of training images by creating variations from

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ШНМ — штучна нейронна мережа

ЗНМ — згорткова нейронна мережа

РЦ – розвідувальний цикл

OSINT – open source intelligence

mAP — mean average precision

GPU — graphics processing unit

ViT— Vision Transofmer

ViT-B — Vision Transformer Base

ViT-L — Vision Transformer Large

FCNs — Fully Convolutional Networks

CNN — convolutional neural network

ВСТУП

Геопросторова розвідка (геоінт) відіграє ключову роль у різних галузях, таких як екологія, сільське господарство, містобудування, картографія та військова справа. Збір, аналіз та інтерпретація геопросторових даних вимагають високої точності та ефективності, що робить цей процес складним і ресурсомістким. Сучасні технології машинного навчання, зокрема згорткові нейронні мережі (ЗНМ), відкривають нові можливості для автоматизації та покращення якості геоінту.

Згорткові нейронні мережі є потужним інструментом для аналізу зображень та обробки просторових даних завдяки своїй здатності автоматично виділяти особливості на різних рівнях абстракції. Це робить їх ідеальним рішенням для завдань, пов’язаних з обробкою супутникових знімків, аерофотозйомок та інших типів геопросторових даних. В даному проекті розглядається застосування згорткових нейронних мереж для проведення геопросторової розвідки, включаючи основні принципи їх роботи, переваги та приклади успішного використання в різних галузях, особлива увага присвячена воєнній сфері.

Особливо актуальним завданням у військовій справі є швидке та точне визначення місцезнаходження ворожих цілей для оперативного прийняття рішень. Традиційні методи аналізу зображень часто потребують значних людських ресурсів і часу, що є критичним фактором у військових умовах. Завдяки можливостям автоматизації та підвищенню точності за допомогою ЗНМ, з'являються інноваційні рішення, які дозволяють мінімізувати участь людини в процесі аналізу та швидше отримувати критично важливі дані.

Тому виникає потреба в розробці ефективних моделей розпізнавання та обробки геопросторових даних, які б не лише покращували точність, але й забезпечували швидку обробку великих масивів інформації в реальному часі.

1 ГЕОПРОСТОРОВА РОЗВІДКА: ЗНАЧЕННЯ, ЗАДАЧІ ТА АЛГОРИТМИ РОЗПІЗНАВАННЯ В ОБОРОННІЙ ПРОМИСЛОВОСТІ

1.1 Геопросторова розвідка, її значення у сучасному світі

Геопросторова розвідка – дисципліна, що включає в себе використання і аналіз зображень і геопросторової інформації для опису, оцінки і візуального зображення фізичних об’єктів. Геопросторова розвідка поєднує в собі декілька дисциплін, таких як: картографування, складання карт, аналіз зображень і розвідка за допомогою зображень. Ця дисципліна є досить молодою, вперше термін геопросторова розвідка (геоінт) було використоано в жовтні 2005 року генералом-лейтенантом ВПС США. Визначення було наступним: “Геопросторова розвідка охоплює всі аспекти космічних знімків і геопросторової інформації та послуг. Вона включає, але не обмежується аналізом буквальних зображень, геопросторових даних і інформації, технічно отриманої в результаті обробки, використання, буквального і небуквального аналізу спектральних, просторових і часових об'єднаних продуктів. Ці типи даних можна збирати на стаціонарних і рухомих цілях за допомогою електрооптичних радарів, радарів із синтезованою апертурою (SAR), відповідних сенсорних програм і нетехнічних засобів (включно з геопросторовою інформацією, отриманою персоналом у польових умовах)” [1].

Великий відсоток даних, які використовуються пов’язані і з фізичним розташуванням. Тому геопросторову розвідку як дисципліну можна розглядати як практичні знання, що вимагають не лише відображення про місцезнаходження, але й глибокого розуміння де та чому саме там знаходяться речі. Раніше зустрічався не чіткий поділ означень: в “промисловому” світі цей термін звучав частіше як “розвідка місцезнаходження”, а в оборонній промисловості – геоінт (з анг. – geospatial intelligence). З часом навіть і цей

умовний поділ зник - геопросторова розвідка слугує для використання просторових даних за допомогою просторової аналітики та міркувань.

Останні технологічні досягнення в сфері геопросторової розвідки сформувались саме завдяки багаторівневій інтеграції. Під багаторівневою інтеграцією мається на увазі безперешкодне злиття даних в різних оперативних сферах: на суші та на морі, в повітрі, в космосі та в інформаційному просторі. Ця інтеграція дозволила отримати цілісну картину ситуації, що в подальшому використовується для фундаментальнішого прийняття рішень. Найкращим прикладом багаторівневої інтеграції є сучасні військові операції, де геопросторові дані формують чітку картину того, що відбувається на полі бою. Повномасштабна агресія рф проти України вивела її на інший рівень: комерційні супутники, розвіддані з відкритих джерел і дані в режимі реального часу з соціальних мереж забезпечує інформацію про місцезнаходження потенційних цілей. Стратегічне планування військових операцій неможливо уявити зараз без залучення постійного спостереження та розвідки з космосу або з бепілотних літальних апаратів (БПЛА). Україною використовується зараз цілий ряд платформ та сенсорів для здійснення розвідки: БПЛА багатьох європейських країн та краї НАТО, американські комунікаційні мережі (Starlink) та комерційні американські супутникові знімки (Maxar і Planet), РЛС фінського провайдера ICEYE, американського провайдера Capella та інших комерційних провайдерів, включаючи RF GEOINT від американського провайдера HawkEye 360. Широкого використання отримали й інші платформи: дешеві розвідувальні дрони, смартфони.

Наприклад, компанія HawkEye 360 утримує угруповання супутників, які відстежують радіочастотні (РЧ) сигнали. Датчики HawkEye 360 фіксують значну активність перешкод Глобальної системи позиціонування (GPS) над територією України. Інші компанії, що займаються космічними даними, такі

як Spire, використовують наносупутники для відстеження авіаційного трафіку. Фінський стартап ICEYE надає Україні знімки зі своїх супутників SAR. Малі безпілотники мають значний вплив, демонструючи свою ефективність у розвідці, захопленні цілей і забезпеченні ситуаційної обізнаності в реальному часі (рис. 1.1) [3].

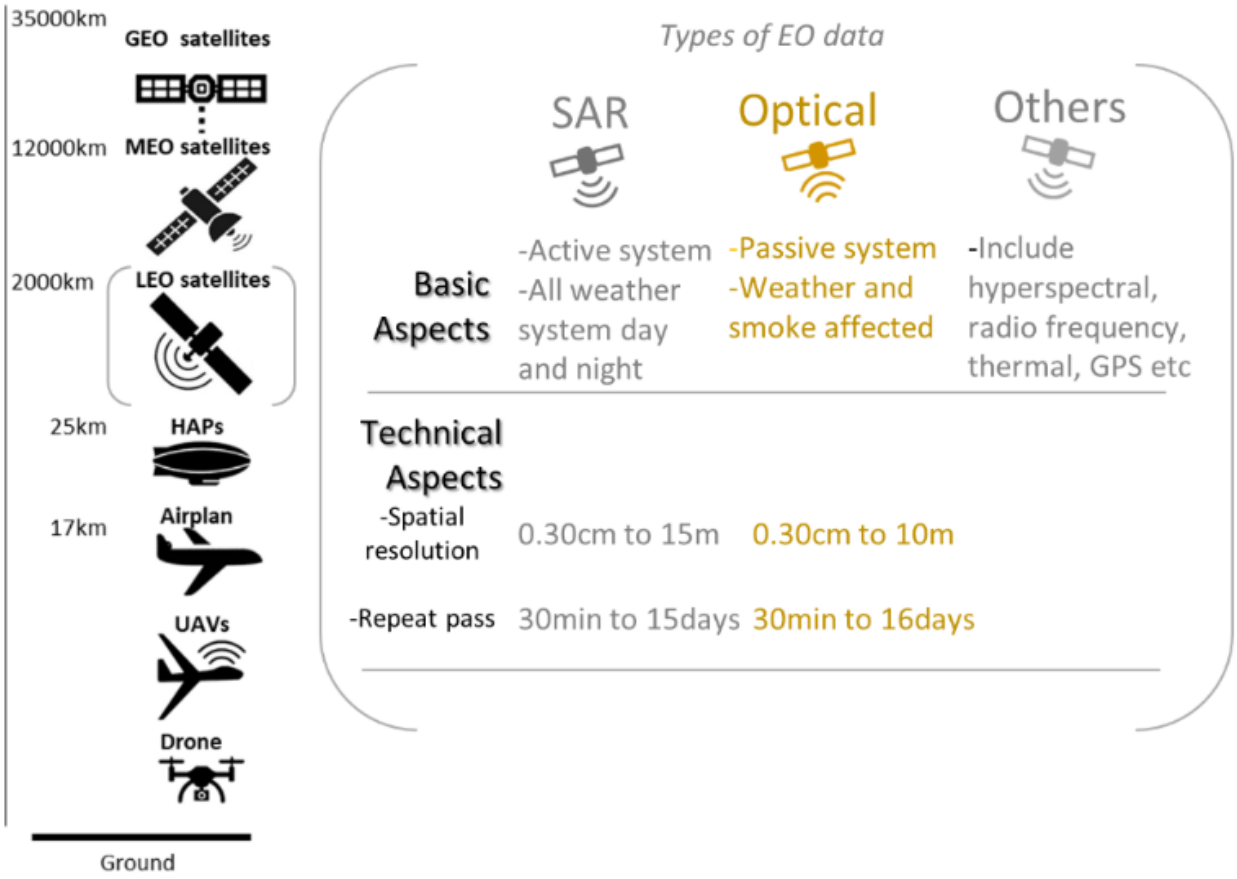

Рисунок 1.1 — Платформи спостереження та типи даних для геопросторової розвідки

У сучасних умовах для аналізу геопросторових даних та розпізнавання місцезнаходження об’єктів на фото важливим є використання різноманітних ресурсів, які надають доступ до супутникових знімків, картографічної інформації та геолокаційних сервісів. Поєднання таких джерел дозволяє отримати комплексний набір даних для підвищення точності алгоритмів

розпізнавання. Нижче представлено основні ресурси, які можуть бути інтегровані для виконання поставлених завдань:

- SatellitesPro. Цей сервіс забезпечує доступ до детальних супутникових зображень із можливістю аналізу місцевості в реальному часі. Він дозволяє отримувати високоякісні знімки для оцінки об'єктів та їхнього оточення;
- Google Earth Pro. Потужна платформа для роботи з супутниковими знімками та тривимірними моделями місцевості. Вона дозволяє інтегрувати різні шари даних, створювати карти та аналізувати територію в режимі реального часу.
- Ukraine Control Map. Карта, яка відображає актуальну ситуацію щодо контролю територій в Україні. Вона дозволяє співставляти місцевість із геополітичними реаліями, що важливо для військових і аналітичних завдань;
- Geoconfirmed. Платформа, яка використовується для підтвердженних геолокацій об'єктів і подій. Вона дозволяє верифікувати дані з інших джерел, що підвищує надійність отриманих результатів. Аналіз вже розпізнаних об’єктів може допомогти і з пошуком та підтвердженням власної геолокації;
- World Imagery Wayback. Ресурс, що надає історичні супутникові знімки, що є ключовим для аналізу змін місцевості з часом. Використання архівів дозволяє оцінювати динаміку території та виявляти зміни, спричинені діяльністю людини чи природними явищами;
- Sentinel Hub. Платформа для роботи з супутниковими даними Sentinel. Вона надає доступ до знімків високої роздільної здатності з різними спектральними візуалізаціями, що дозволяє аналізувати об'єкти на основі їхніх фізичних характеристик;

- Wikimapia. Краудсорсинговий картографічний ресурс, який надає детальну інформацію про об’єкти на місцевості. Він корисний для ідентифікації об’єктів, що не завжди представлені на офіційних картах;
- Шукач. Українська платформа, орієнтована на позначення цікавих місць і об’єктів. Вона дозволяє уточнювати інформацію про локальні особливості та використовувати її для більш детального аналізу;
- SunCalc. Інструмент для аналізу положення Сонця в конкретний момент часу. Його можна використовувати для визначення часу й напряму зйомки зображення, що може бути корисним у задачах верифікації даних.

Інформаційне та організаційне поєднання дозволяє більш ефективно виконувати військові задачі. Системи та групи людей об’єднуються для надання інформації з більшою швидкістю, масштабом та релевантністю. Структурно це означає, що всі розвідувальні дисципліни діють як єдине ціле на всіх рівнях. Яскравим прикладом такого успішне використання українськими військовими реактивної системи залпового вогню HIMARS – вражаючий успіх можна пояснити якісною розвідкою та локалізацією необхідних цілей, неготовність ворога. Іншим хорошим прикладом чіткої комунікації можна навести удари високоточною зброєю (де спочатку розпізнається та підтверджується присутність необхідних цілей, після чого йде ураження ворожого об’єкту, що показано на рисунку нижче)

Рисунок 1.1 — Ураження ракетами Storm Shadow бункеру та ворожого командного пункту, н.п. Мар’їно, Курська область

Розвідка цілей може бути ефективною не тільки використовуючи супутники, а й аналізуючи великі дані фронту. Це значною мірою може бути і аналіз ворожих ресурсів: фото та відеоматеріалів, активності в соцмережах. Саме використання відкритих даних дозволяє уточнити дані по необхідній цілі; знайти нову ціль або ж використатись для подальшого узгодження.

Розвідка на основі відкритих джерел (з англ. - Open source intelligence OSINT) – це метод збору інформації з публічно доступних джерел для аналітичного використання у різних сферах таких як безпека, дослідження та розвідка. OSINT включає в себе аналіз даних з медіа, офіційних документів, публічних записів, Інтернету та інших відкритих платформ. Саме поглиблення, включення геопросторової розвідки в робочі процеси OSINT може забезпечити кращі сигнали про нові ризики та загрози, або ж аналіз ситуації. Це посилює часовий аналіз, покращує розуміння людської географії, допомагає оцінити ступінь небезпеки тощо. Зрештою, ці кращі знання дозволяють керівним особам та командам приймати більш обґрунтовані рішення.

Аналізуючи опубліковані зображення, відео та дані з Google Maps OSINT-дослідники отримують більш чітку картину, ніж могли б це зробити маючи та аналізуючи лише супутникові знімки.

Збір OSINT-даних (з англ. – Open source intelligence) можна розділити на дві групи: пасивний та активний збір. У той час як активний збір використовує різноманітні методи для збору даних, пасивний збір передбачає використання платформ розвідки загроз. Найпоширеніші OSINT-ресурси зображені на рисунку 1.2.

Рисунок 1.2 — Медіа-призма для використання в OSINT-аналізі

1.2 Задачі геопросторової розвідки

1.2.1 Стратегічні задачі геопросторової розвідки в оборонній промисловості

В оборонній промисловості геопросторова розвідка використовується для наступних задач:

- Розвідувальні дані про можливості, наміри або діяльності осіб чи організацій за межами держави;
- Розвідувальні дані з метою задоволення оперативних цілеспрямованих, тренувальних та навчальних потреб збройних сил країни;
- Розвідка з метою підтримки органів влади країни у виконанні функцій національної безпеки;
- Технічна допомога та підтримка у виконанні функцій реагування на надзвичайні ситуації;
- Допомога Силам оборони в підтримці військових операцій;
- Співпраця з Силами оборони з питань розвідки. [2]

Перший пункт можна розглянути і на прикладі війни з рф, а саме відтворити хронологію фіксації російських військ до початку самого вторгнення. 31 березня Європейське командування Збройних сил США підвищило рівень тривоги до «потенційної неминучої кризи» у відповідь на оцінки, згідно з якими понад 100 000 російських військовослужбовців були розміщені вздовж кордону з Україною і в Криму, на додаток до військово-морських сил в Азовському морі. Це розгортання було охарактеризоване як найбільша мобілізація сил з моменту 2014 року.

Рисунок 1.2 — Виявлення розгортання військовослужбовців на території полігону Погоново, Воронезька область

Супутникові знімки полігону «Погоново», зроблені 10 квітня, проливають світло на масштаб розгортання і характер задіяної військової техніки. На знімках видно дві мотострілецькі бригади і додаткові елементи, ймовірно, дивізійного рівня, в тому числі польовий госпіталь, казарми і кілька неідентифікованих місць дислокації військ. Перебування військових на полігонах загалом зайняло більше часу, аніж це традиційно відбувається. Саме в той час почали з’являтись перші побоювання щодо можливого прихованого перекидання військ ближче до території України під виглядом “навчання”.

З плином часу виведення російських військ з території тимчасово окупованого Криму або з середніх по відстані до України (близько 150-200 км) полігонів все ще не відбувалось. Навіть навпаки, нові супутникові знімки, зроблені приватною американською компанією в грудні 2021 року, показують,

що рф продовжувала нарощувати свої сили в анексованому Криму та поблизу України протягом останніх тижнів. На знімках, оприлюднених Maxar 13 грудня 2021, чітко видно базу в Криму заповнену сотнями бронемашин та танків. Знімок цієї ж бази опублікований з супутника Maxar в жовтні 2021 показав, що база була наполовину порожня (рис. 1.3). Знімки деяких інших полігонів теж свідчили про наростання загрози збройного наступу.

Рисунок 1.3 — Знімок Maxar російської бронетехніки в Солоті 7 вересня (ліворуч) та 5 грудня (праворуч)

Січень та лютий 2022 року охарактеризувались піком наближення невідворотності вторгнення. Нові комерційні супутникові знімки підтверджували бурхливу військову активність рф в кількох місцях поблизу України. Це і полігони в Білорусі, деякі військові полігони за десятки кілометрів від кордону з Україною. До того ж це була присутність не просто військової техніки типу бронемашин та танків – фіксувалась і присутність артилерії, мостоукладальників (форсування водних перешкод), реактивної артилерії, розгортання великих контингентів військ, нарощування ударних вертольотів та винищувачів-бомбардувальників. Та врешті за добу-дві до

самого вторгнення фіксувались значні переміщення військової техніки рф ближче до кордону України (рис. 1.4)

Рисунок 1.4 — Польове розгортання військових колон в Головчино, Бєлгородська область за 15 км від кордону з Україною

Геопросторова розвідка відіграла важливу роль у моніторингу ситуації перед вторгненням рф в Україну, дозволяючи оцінити масштаб загрози. Завдяки аналізу супутникових знімків вдалося виявити масове зосередження військової техніки, розгортання особового складу на полігонах. Фіксація таких кроків та переміщення техніки дозволила краще підготуватись до вторгнення, посилюючи оборону, попередження союзників та стратегічного планування, що зіграло важливу роль у збереженні обороноздатності України в перші миттєвості війни та і в подальшому.

1.2.2 Цілі систем розпізнавання

У сучасній військовій сфері технології розпізнавання об'єктів відіграють ключову роль у підвищенні точності, швидкості та ефективності виконання

операцій. Вони забезпечують автоматичну або напівавтоматичну ідентифікацію та класифікацію цілей на основі даних, отриманих із супутників, дронів, наземних камер, радарів та інших сенсорів. Це дозволяє швидко реагувати на зміни обстановки, уникати втрат і мінімізувати ризики для особового складу. Технології розпізнавання охоплюють різні аспекти, включаючи виявлення наземних, повітряних і морських об'єктів, аналіз їхніх характеристик, визначення приналежності (свій-чужий), а також інтеграцію цих даних у системи управління бойовими діями.

У військових операціях використовуються як традиційні методи (радіолокація, тепловізійні сенсори), так і сучасні технології, такі як комп'ютерний зір і штучний інтелект. Останні дають змогу обробляти величезні обсяги даних і розпізнавати складні патерни, які важко ідентифікувати вручну. Особлива увага приділяється автоматизації процесів, зокрема в системах навігації, автономних дронах та роботизованій техніці. Це значно пришвидшує прийняття рішень і підвищує ефективність операцій.

Далі будуть розглянуті основні задачі розпізнавання у військовій галузі, які охоплюють виявлення, класифікацію та підтримку військових дій, а також технології, які сприяють реалізації цих задач.

- Виявлення та ідентифікація цілей. До цього пункту слід віднести розпізнавання військової техніки, інфраструктури та інших ворожих військових об’єктів;
- Системи “свій-чужий”. Використання автоматичних систем для визначення приналежності об'єктів, що дозволяє уникнути "дружнього вогню" та підвищити координацію між підрозділами;
- Розвідка та спостереження. Цей напрям особливо досягнув великого прориву під час війни та внаслідок неймовірно стрімкого прогресу БПЛА. Він включає в себе збір та аналіз даних з різних джерел,

включаючи супутникові знімки, аерофотозйомку та дані з дронів, для отримання інформації про розташування та рух ворожих сил;

- Автоматизація бойових систем. Інтеграція технологій розпізнавання в автономні бойові системи, такі як безпілотні танки чи дрони, для підвищення їхньої ефективності та зменшення ризику для особового складу.

1.3 Алгоритми розпізнавання об’єктів у геопросторовій розвідці

Зважаючи на те, що розвідувальна інформація надходить з дедалі більшої кількості джерел, консолідація релевантної інформації в кероване рішення є головним викликом для військових лідерів у всьому світі. Популярним способом вирішення цієї проблеми є комплексне розвідувальне рішення, яке включає підходи, функції і можливості для проведення розвідувальних операцій на всіх рівнях, від стратегічного до тактичного, для кінцевих користувачів, які приймають рішення в сфері безпеки і оборони, є мультирозвідка. Цей підхід полягає в обробці і зборі всієї наявної розвідувальної інформації, що передбачає збір відповідних розвідувальних даних з різних джерел, їх обробку і об'єднання в унікальне інтелектуальне ядро обробки. Розвідувальна інформація - це продукт, який є результатом збору, зіставлення, оцінки, аналізу, інтеграції та інтерпретації зібраної інформації з різних типів джерел. [4]

Розвідувальний цикл (РЦ) - це процес збору даних і перетворення їх на аналітику для використання результатів розвідки. Напрям, збір, обробка, використання та розповсюдження - це етапи цього процесу. Продукти РЦ можуть бути створені одним елементом РЦ або у співпраці з іншими елементами РЦ і надані з іншими елементами розвідки і надаватися клієнтам розвідки у різних форматах, наприклад, у вигляді документів, цифрові медіа, брифінги, карти, графіки, відео та іншими способами доставки.

В розвідці за допомогою зображень (IMINT) зображення об'єктів відтворюються оптичним або електронним способом на плівці, а розвідувальна інформація на плівці, електронних пристроях відображення або інших носіях, а розвідувальна інформація отримується шляхом збору, використання та аналізу зображень отримується в результаті збору, використання та аналізу зображень за допомогою візуальної фотографії, інфрачервоних датчиків, лазерів, електрооптики і радіолокаційних датчиків, таких як радари з синтезованою апертурою (SAR). Натомість геопросторова розвідка стає можливою завдяки поєднанню зображень, IMINT і геопросторових даних.

Існує досить велика кількість досліджень, проєктів та розробок, де було реалізовано поєднання геопросторової розвідки, IMINT, OSINT та аналізу соціальних мереж.

Клеантіс Карамвасіс та ін. (2021) [5] об'єднали супутники НЗ, соціальні мережі і краудсорсингову інформацію для боротьби з тероризмом і організованою злочинністю. Частина роботи, виконаної в цьому документі, була профінансована Європейським космічним агентством (ЄКА) в рамках проекту EO Science for Society (EOEP5 Block 4) - Expanding Demand for EO Derived Information EOLAW Project. Виконувалось дослідження послідовно в чотири етапи:

- OSINT аналіз великої кількості даних з соцмереж, медіа та інших інформаційних ресурсів;
- Аналіз оптичних даних для визначення змін;
- Формування просторової мережі з використанням інформації про дороги, будівлі (OSM) та населення;
- Формувати взаємозв'язки між локалізаціями з великих даних та змінами спостереження Землі (Earth Observation - EO) через побудовану просторову мережу (рис. 1.5).

Результат цього дослідження показує, що взаємозв'язок між інформацією з даних НЗ та інформацією з аналізу OSINT може суттєво допомогти у важливій роботі правоохоронних організацій. Розробка нових методів, заснованих на великих даних з онлайн джерел і спостереження Землі, може забезпечити свіжий погляд на цей напрямок. Такий інтегрований алгоритмічний підхід, де необхідно використовувати злиття інформації з різних джерел, може також покращити додатки, пов'язані з врегулюванням кризових ситуацій і забезпеченням безпеки.

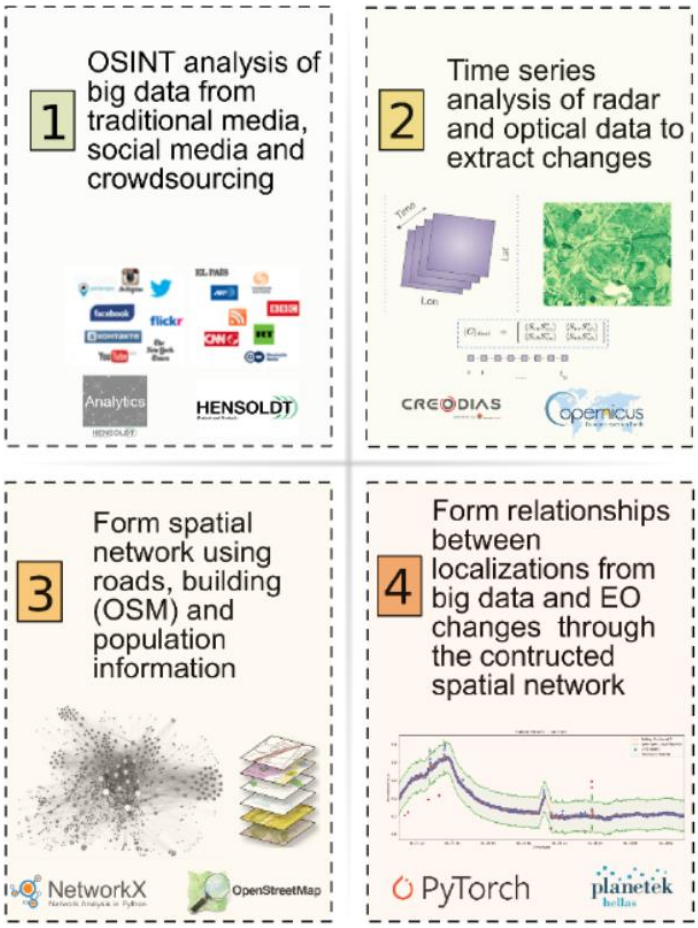

Рисунок 1.5 — Робочий цикл методологічного підходу Клеантіса Карамвасіса

Про використання переваг IMINT, OSINT та геопросторової розвідки у військовій справі є і робота Агати Ціолковської (2018 р.) [6]. В цій роботі було розглянуто функціонування OSINT на багатьох рівнях розвідувальної інформації. У своїй роботі вона робить висновок, що для воєнної розвідки

важлива соціальна підтримка збройних сил і їхніх дій. Цю інформацію можна отримати від OSINT технологій.

Мунір та ін. (2022) [7] запропонували модель ситуаційної обізнаності в армії з динамічним прийняттям рішень, яка включає штучний інтелект і динамічні прикладні системи на основі даних для адаптації вимірювань і ресурсів відповідно до мінливих ситуацій. У загальній архітектурі моделі одним з модулів, що підвищує точність ситуаційної обізнаності, є субмодуль, який збирає інтелектуальну інформацію з різних джерел (рис 1.6).

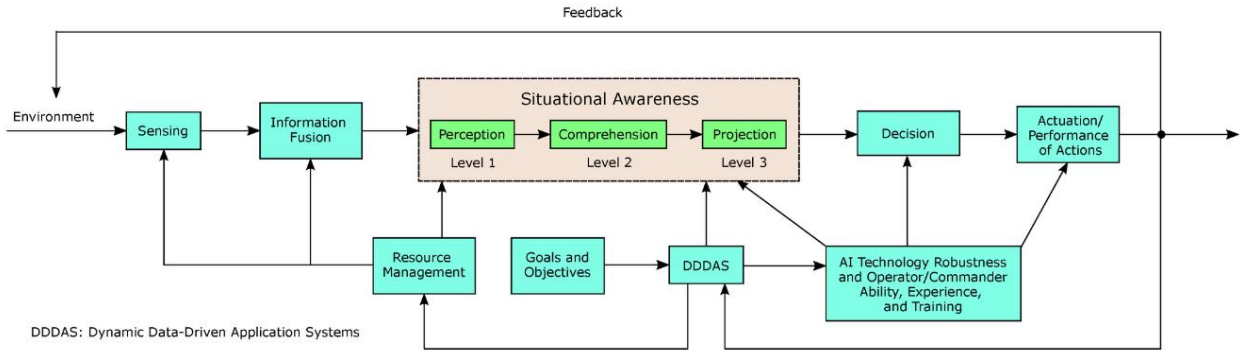

Рисунок 1.6 — Модуль “Ситуаційної обізнаності”

На першому рівні модуля “Ситуаційної обізнаності” є сприйняття статусу, атрибутів і динаміки об'єктів у навколишньому середовищі. Наприклад, пілот повинен розрізняти важливі об'єкти в навколишньому середовищі, такі як інші літаки, рельєф місцевості, попереджувальні сигнали, а також їхні відповідні характеристики.

Другим етапом аналізу є розуміння ситуації, яке ґрунтується на інтеграції розрізнених елементів аналізу рівня 1. Рівень 2 - це крок далі, ніж просто усвідомлення елементів середовища, оскільки він стосується розвитку розуміння важливості цих елементів по відношенню до цілей оператора – це розуміння об'єктів у навколишньому середовищі, зокрема, коли вони інтегровані разом.

Третій рівень “Ситуаційної обізнаності” пов'язаний зі здатністю прогнозувати майбутні дії суб'єктів у навколишньому середовищі, принаймні у найближчій перспективі. Таке прогнозування досягається на основі пізнання стану і динаміки елементів середовища та розуміння ситуації.

Загалом, запропонована модель “Ситуаційної обізнаності” представлена на прикладі використання вхідної інформації з точки зору військ та Повітряних сил. Запропонована модель включає штучний інтелект і динамічні прикладні системи на основі даних, які допомагають керувати вимірюванням і ресурсами відповідно до мінливих ситуацій, які сприймаються і прогнозуються ядром алгоритму.

Автори Котарідіс та Бенекос запропонували свою модель для ефективного викорситання засобів геопросторової розвідки, OSINT та IMINT [6]. Інформація з відкритих джерел збирається за допомогою OSINT та аналізу соціальних мереж, де потім групується в корисні та дієві розвідувальні дані (рис 1.7).

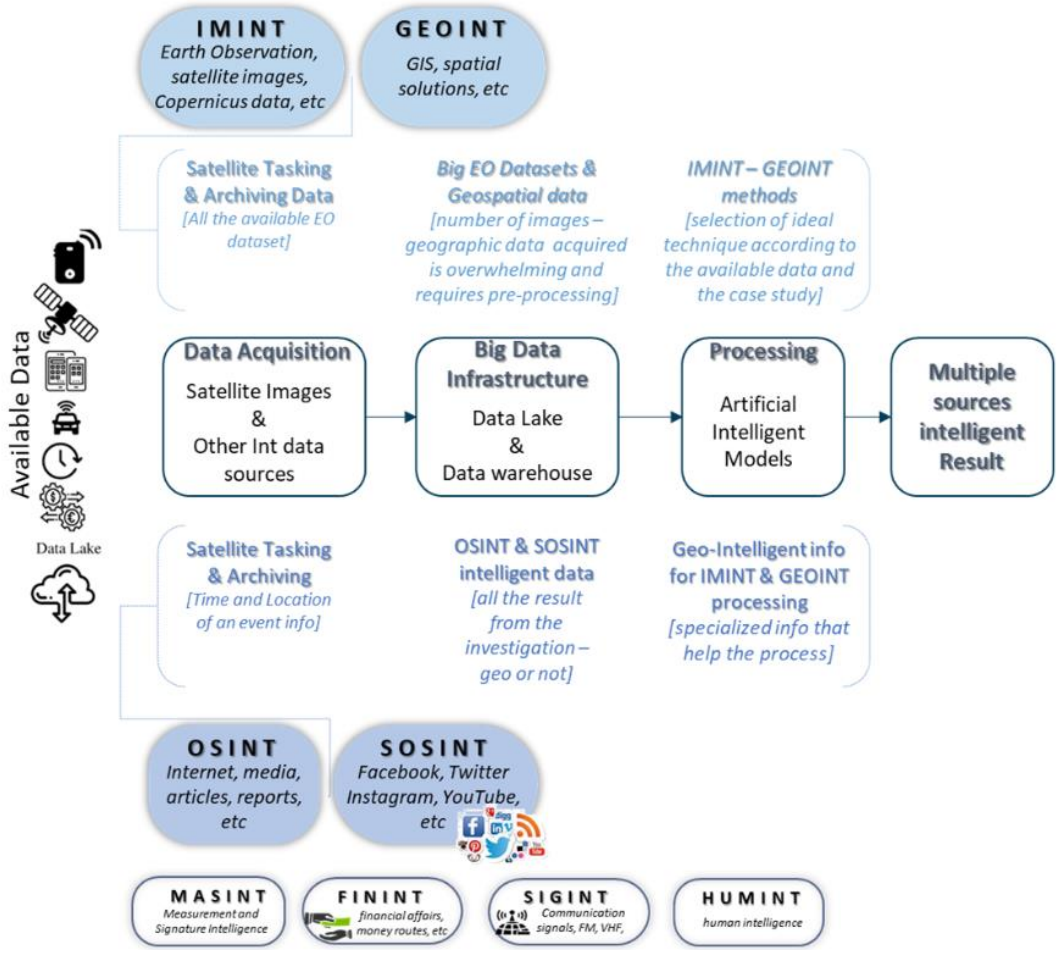

Рисунок 1.7 — Високорівнева архітектура на основі IMINT та геопросторової розвідки, яка інтегрує іншу додаткову розвідувальну інформацію

Фахівці з аерофотозйомки та геоінформаційних систем (ГІС) здійснюють пошук наявних даних відповідно до тематичного дослідження та його часових рамок. Моніторинг явища може мати три фази: до, під час і після події. Тематичне дослідження може охоплювати принаймні одну або всі ці фази відповідно до типу явища або характеристик спостереження. Більшість додатків у сфері безпеки і оборони (і найбільш важливих з точки зору кінцевого користувача) працюють у режимі, близькому до реального часу, і тривають до моменту події, що є дуже конкурентним середовищем. Оскільки перед подією повністю покривається всіма існуючими і архівними даними з

наявних супутникових знімків, можливості супутника з постановки завдань і реагування на конкретну область інтересу є дуже важливими для тематичних досліджень в режимі, близькому до реального часу, і після події.

Інформація OSINT може відігравати вирішальну роль у загальному конвеєрі IMINT-геопросторової розвідки.

Послуги та продукти аерофотозйомки, а також їхні технологічні розробки чітко пов'язані з наборами супутникових даних аерофотозйомки та їхнім архівом у часовій базовій лінії явища (рис. 1.8). Чим більше явище активується або з'являється в режимі, близькому до реального часу, тим більше набір даних повинен мати можливостей для його спостереження.

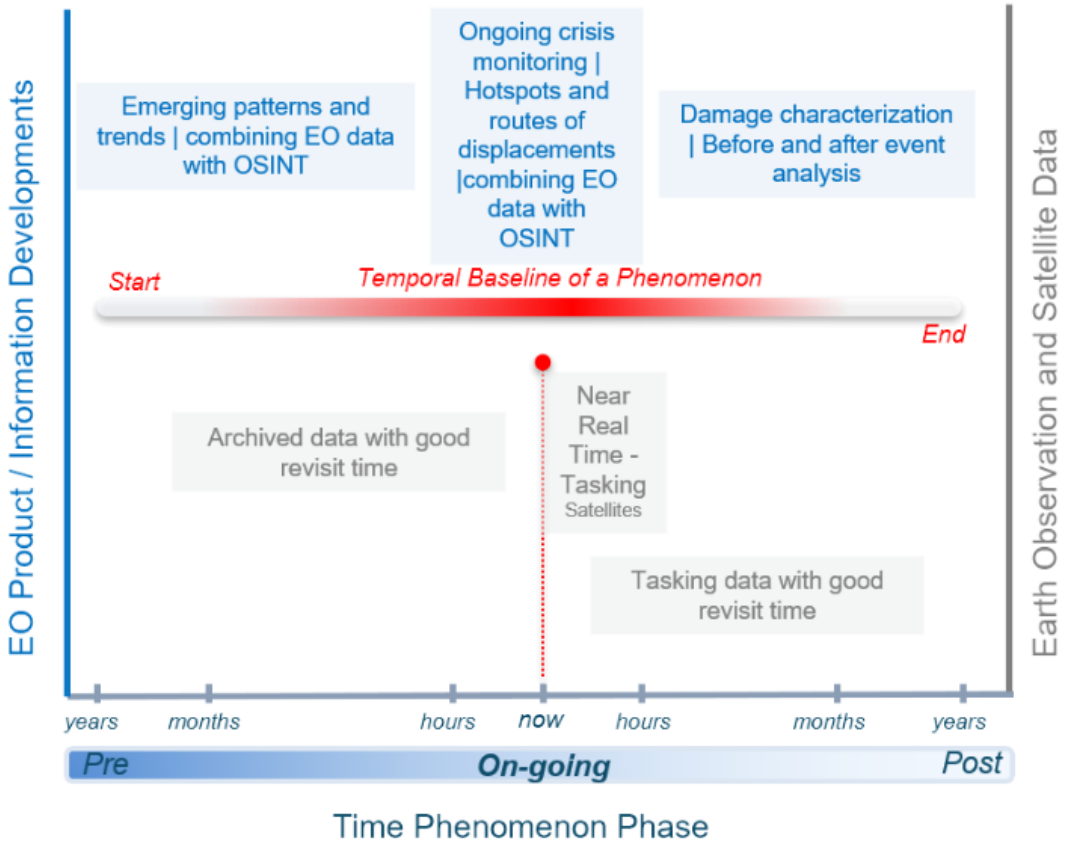

Рисунок 1.8 — Взаємозв’язок між продуктами аерофотозйомки, інформаційними розробками та супутниковими даними на основі часової бази спостереження за явищем

Наступним компонентом є інфраструктура даних, яка складається з усіх наборів даних. Найбільш поширеною інфраструктурою даних для геопросторової розвідки є озеро даних (з англ – Data Lake). Озеро даних можна визначити як «велике сховище даних, що зберігається у власному форматі». Структурування та обробка даних відбувається лише під час їх отримання в озерах даних. Інший варіант - сховище даних, «система управління даними, призначена для зберігання великих обсягів попередньо структурованих даних з численних джерел». Їхнє завдання - збирати та організовувати дані за допомогою точного методу класифікації, щоб швидко надавати інформацію та покращувати процес прийняття рішень. Після чого зібрані дані можна надалі використовувати для вирішення задачі геолокації об’єкту.

Повний цикл роботи OSINT-аналітика можна розглянути на прикладі війни з рф. Найвідоміші українські OSINT-аналітики, такі як: Deepstate, Кіберборошно, EjShahid, є прикладами регулярної та надзвичайно якісної роботи з геопросторовими даними. Робота таких фахівців виконується як публічно так і секретно для виконування задач надавання допоміжної розвідувальної інформації для Збройних Сил України. Публічно це зазвичай геолокація нанесення ураження по ворожій техніці / особовому складу / інфраструктурі, фіксація переміщення / наявності піхоти на певній локації (для фіксації змін карти фронту); в той же час як непублічна робота виконується зазвичай над виявленням місць дислокації ворожого особового складу, знаходження ремонтних баз техніки, визначення локації роботи ворожої техніки (реактивна артилерія, ствольна артилерія, вогневі позиції танків), знаходження спостережних пунктів (рис 1.9).

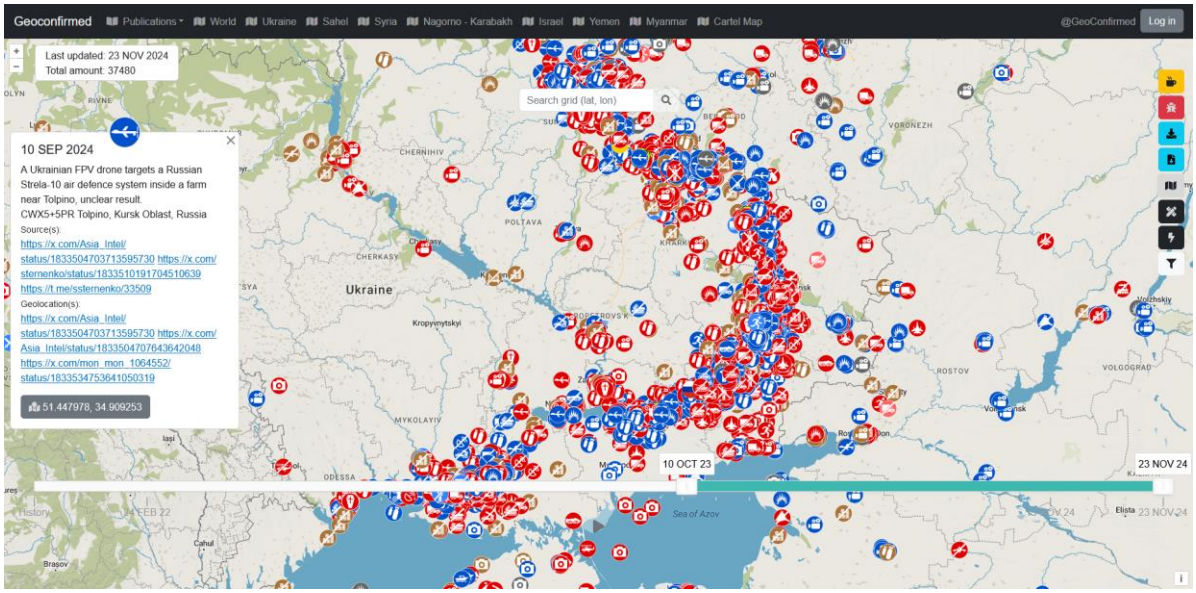

Рисунок 1.9 — Карта геолокованих даних фахівцями волонтерської організації GeoConfirmed за період 13.10.2023 по 23.11.2024

Величезна частина інформації, яка іде на аналіз для виявлення геолокації, береться з соцмереж, наприклад з ворожих telegram-ресурсів, відео з TikTok чи X (раніше Twitter). Саме такі ресурси є найближчими для роботи з даними майже в режимі реального часу. Після чого йде аналіз отриманого відео чи зображення. Виконується задача виявлення найбільших характерних ознак: інфраструктурна особливість (особливості рельєфу, будівля з характерними ознаками, наявність залізничних колій і т.д) для локалізації об’єкту та прив’язки його до можливої області знаходження. Ефективним варіантом визначення області знаходження об’єкту може бути і аналіз опублікованих раніше фото або відео-матеріалів автора. Наприклад, якщо це аналіз відео ворожого підрозділу, то це аналіз на якому напрямку ведення бойових дій знаходиться він. Такі дії робляться для фільтрації сіл / міст пошуку об’єкту для звуження можливих місць його знаходження. Після чого відбувається аналіз фото / відеоматеріалу з проектуванням його на вид зверху та подальшим порівнянням знімку з платформи аерефотозйомки (наприклад Google Earth) та подальшим визначенням місцезнаходження.

Висновки до першого розділу

В даному розділі було розглянуто значення геопросторової розвідки та основні тендеції цієї дисципліни в сучасному світі. Представлено основні ресурси, які можуть слугувати якісним набором даних для виконання задачі розпізнавання місцезнаходження об'єкту на зображенні. Окреслено основні задачі геопросторової розвідки та наведено приклад використання цієї дисципліни на основі подій, що розгортались біля кордонів України до початку повномасштабного вторгнення в 2022 році. Геопросторова як дисципліна показала себе надзвичайно важливою для якісної підготовки оборони держави та як інструмент розвідки. Обгрунтування вибору нейронної мережі.

Виконано огляд основних алгоритмів розпізнавання об'єктів у геопросторовій розвідці. Ключовою ознакою успішності алгоритму являється комплексний підхід до вирішення задачі розпізнавання місцезнаходження цілі. Наведено приклади процесу розпізнавання цілей українськими OSINT-фахівцями.

Враховуючи перспективність та актуальність дисципліни в умовах війни з рф, доцільним є розробка комплексного підходу до розпізнавання місцезнаходження ворожих цілей.

2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

Поява штучного інтелекту дозволила безпрецедентні масштаби генерації аналізу та зберігання даних. Однак швидкі темпи створення даних часто перевищують людські та технічні можливості ефективного управління ними. Сектори розвідки, особливо розвідки з відкритим вихідним кодом (OSINT), стикаються з унікальним тиском, пов'язаним з необхідністю інтерпретації величезних обсягів даних, що генеруються штучним інтелектом.

Можливості штучного інтелекту в обробці даних здатні революціонізувати OSINT, зробивши величезні масиви даних більш керованими. Ця трансформація передбачає автоматизований аналіз величезних обсягів даних, що дає змогу фахівцям розвідки визначати тенденції, виявляти аномалії та приймати обґрунтовані, ініціативні рішення. Інструменти на основі штучного інтелекту, такі як алгоритми NLP, допомагають розшифровувати великі текстові масиви, виокремлювати важливу інформацію та класифікувати її за пріоритетністю. Крім того, машинне навчання дозволяє OSINT динамічно розвиватися, підлаштовуючи свої параметри під нові типи загроз у міру їх появи [8].

Однак ці переваги несуть із собою і певні виклики, зокрема, перевантаження даними. Здатність штучного інтелекту постійно генерувати і оновлювати дані створює додатковий тиск на фахівців з розвідки, які повинні просіювати цю інформацію для отримання критично важливих висновків. Хоча штучний інтелект може пом'якшити деяке перевантаження за допомогою алгоритмів розстановки пріоритетів, ці інструменти повинні бути вдосконалені, щоб управляти все більш складними наборами даних без шкоди для якості та своєчасності прийняття рішень.

2.1 Постановка задачі

Ручний процес розпізнавання об’єкту людино наразі неможливо повністю замінити штучним інтелектом, проте пошук людиною має як свої переваги так

і недоліки. Однією з найбільших переваг – це підвищена контрольованість всього процесу. Штучний інтелект поки не може повністю автоматизуватись під процес пошуку даних, виділення характерних ознак на знімку об’єкта та подальшому пошуку його геолокації. Надзивчайно важливою є задача розбивати процес пошуку на багато етапів. Такий поетапний підхід дозволяє уникати помилок і забезпечує глибокий аналітичний підхід до розпізнавання, який штучний інтелект поки не здатний виконувати на належному рівні. Крім того, участь людини в процесі дає змогу враховувати зміни в підході, в кореляції вхідних даних та в більш гнучкому погляду на весь етап розпізнавання.

Проте ручне виконання таких задач має і недоліки, що тісно пов’язані з людським фактором. Такий процес пошуку є значно повільнішим, порівняно з автоматизованими системами, що може бути критично у випадках, коли швидкість прийняття рішення є вирішальною. Також людський фактор, зокрема втома або неуважність, може призводити до помилок у виявленні чи класифікації об’єктів. На цьому фоні важливо зазначити стрімкий розвиток штучного інтелекту, який відкриває нові можливості у сфері геопросторової розвідки. Сучасні алгоритми машинного навчання дозволяють автоматизувати задачі, зменшуючи навантаження на людину, підвищуючи швидкість аналізу та забезпечуючи обробку величезних обсягів даних у режимі реального часу. Комбінація людського присутності в процесі та штучного інтелекту стає найбільш ефективним підходом у вирішенні складних задач розпізнавання.

Нейронні мережі здатні вирішувати задачі класифікації, розпізнавання та передбачення. Всі ці три задачі можна віднести і до процесу розпізнавання цілей за допомогою штучного інтелекту.

Класифікація здійснюється за допомогою аналітичних моделей, відомих як класифікатори. Перевагою цього типу задач є відносна простота алгоритмів і методів їх реалізації порівняно з іншими технологіями аналізу даних. Існує

багато видів класифікаторів, для яких застосовуються як статичні підходи (наприклад, логістична регресія або дискримінантний аналіз), так і методи машинного навчання, зокрема нейронні мережі, дерева рішень, метод k-найближчих сусідів та машинні опорні вектори.

Штучний інтелект ефективно вирішує задачу розпізнавання завдяки своїй здатності аналізувати складні патерни в даних. Нейронні мережі навчаються на великій кількості прикладів, що дозволяє їм розпізнавати об’єкти, класифікувати їх за категоріями та навіть виявляти приховані ознаки, які важко ідентифікувати традиційними методами. Завдяки багатошаровій структурі, нейронні мережі можуть обробляти як зображення, так і текстові чи числові дані, забезпечуючи високу точність у задачах сегментації, класифікації та геолокації.

Нейронні мережі можуть виконувати і задачу прогнозування, тобто передбачати майбутні події та прогнозувати подальший розвиток ситуації. Основною метою передбачення є оцінка можливих сценаріїв розвитку подій та мінімізація ризиків під час прийняття рішень. Хоча прогнози часто мають похибки, їх точність залежить від якості та ефективності використовуваної прогнозувальної системи.

В цій роботі запропоновано поетапний підхід до розв’язання задачі геолокації:

1. Сегментація зображень з виділенням характерних ознак об’єктів;

2. Класифікація об’єктів (в даній роботі це буде класифікація за країнами);

3. Розпізнавання місцезнаходження для визначення координат.

Першим етапом йде задача сегментації зображення для виділення найбільш характерних ознак на фото. Це необхідно щоб надалі зменшити та локалізувати область можливих місць класифікації цілі. Цю задачу можна було виконувати і поставивши як ціль визначення положень найбільш

характерних ознак об’єкту (за допомогою обмежувальних рамок). На відміну від класифікації чи розпізнавання зображень, завдання виявлення об'єктів, успадковані від завдань класифікації, вимагають від алгоритму не лише визначення категорії, до якої належить об'єкт, що цікавить, але й точного визначення положення об'єкта за допомогою обмежувальної рамки, що робить завдання більш складним.

Методи виявлення об'єктів на основі глибокого навчання можна розділити на одноетапні та двоетапні алгоритми виявлення об'єктів (рис. 2.1), залежно від того, чи включається в процедуру крок для запропонованої області. Двохетапні алгоритми мають переваги з точки зору точності, оскільки вони додають етап до загального процесу, який схожий на додатковий етап скринінгу над першим кроком.

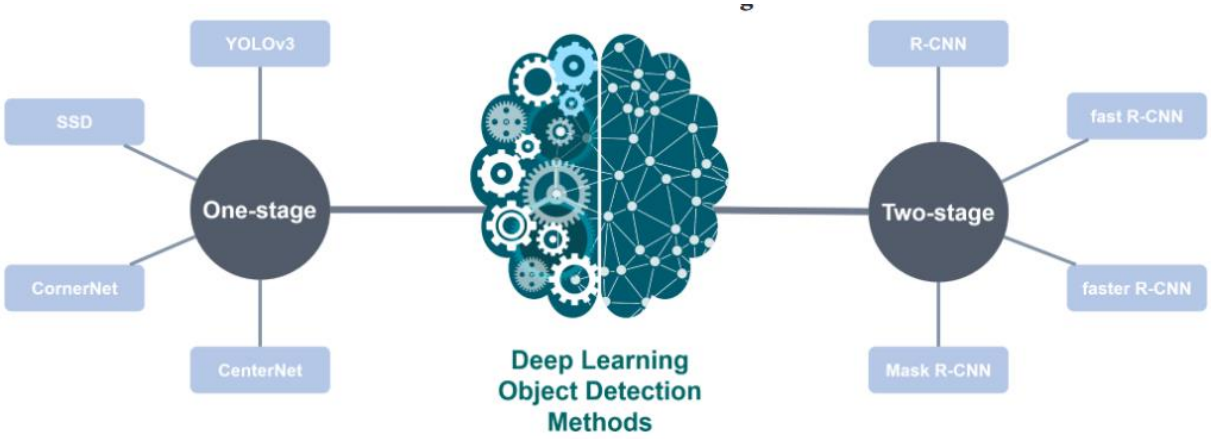

Рисунок 2.1 — Найсучасніші алгоритми виявлення об’єктів на основі глибоко навчання

Одноетапні алгоритми виконують лише одне виявлення; вони швидші, оскільки відсутня фаза генерації області пропозицій. До двоетапних алгоритмів відноситься сімейство R-CNN - набір алгоритмів, які є показовими для цього підходу. Зокрема, R-CNN, швидкий R-CNN та швидший R-CNN належать до цієї категорії. З іншого боку, одноетапні алгоритми включають сім'ї YOLO, SSD та без’якорні алгоритми.

Задача класифікації повинна вирішуватись для ще одного зменшення можливої області знаходження координат. Надзвичайно важливо ретельну підійти у виборі нейронної мережі для вирішення задачі класифікації та здійснити якісне навчання, адже в разі помилки на цьому етапі задача розпізнавання буде виконана неправильно. Тому для цього етапу буде порівняно та досліджено велику кількість нейронних мереж для вибору найефективнішої з них.

Задача місцезнаходження об’єкта по знімку виконується враховуючи попередні етапи. Кожне зображення тренувального набору для навчання містить: оригінальне зображення з координатами (в окремому csv файлі), зображення з сегментаційною маскою та визначення класу об’єкту (за країнами).

2.2 Інструменти та технології для навчання моделі

Для навчання моделі використовується інтерактивне середовище Google Colab, яке надає доступ до потужних обчислювальних ресурсів, зокрема GPU (графічного процесора) та великої оперативної пам’яті. В Google Colab можна застосовувати різні бібліотеки на Python, завантажувати та виконувати файли, аналізувати дані та взаємодіяти з результатами в браузері. Цей хмарний сервіс заснований на Jupyter Notebook, який працює через веб-браузер і надає доступ до GPU та TPU, без необхідності встановлення додаткового програмного забезпечення. Google Colab Pro дозволяє ефективно працювати з великими датасетами та забезпечує високу швидкість виконання алгоритмів завдяки виділенню обчислювальних одиниць. Colab підтримує формат .ipynb, спільний із Jupyter Notebook, що дозволяє легко переносити блокноти між локальним середовищем і хмарним. Однією з ключових переваг Google Colab є інтеграція з Google Drive, яка спрощує зберігання та спільний доступ до ваших проєктів. На відміну від Jupyter, Colab автоматично налаштовує необхідні бібліотеки та середовище для роботи з Python. У той час як Jupyter Notebook більше

підходить для локальної роботи з невеликими проєктами, Google Colab забезпечує масштабованість для обчислювально інтенсивних задач, таких як навчання моделей машинного навчання. Однак, Colab залежить від інтернет-з'єднання, а Jupyter може працювати автономно на локальному сервері, що дає йому певну перевагу в умовах обмеженого доступу до мережі. Для роботи буде використовуватись середовище Google Colab Pro (зі щомісячною підпискою). Виконуватись скрипти будуть на 53 ГБ оперативної пам’яті та 22.5 ГБ оперативної пам’яті графічного процесора (GPU). Виконана й інтеграція з Google Drive для збереження моделей та результатів.

Фреймворком буде Tensorflow із високорівневою бібліотекою Keras. Ця платформа дозволяє легко завантажувати натреновані моделі, такі як DenseNet121, ResNet або EfficientNet, що значно скорочує час навчання та покращує якість моделі. Крім того, TensorFlow/Keras забезпечує гнучкість у додаванні власних шарів до моделі для тонкого налаштування під конкретні завдання. Фреймворк ефективно використовує апаратне прискорення для значного підвищення швидкості обчислень під час навчання великих нейронних мереж. Завдяки повній інтеграції з Google Colab, TensorFlow/Keras дозволяє легко налаштовувати середовище для роботи, використовуючи хмарні ресурси без необхідності складної локальної конфігурації.

Через сховище даних Google Drive буде передаватись основний контент даного навчання. Саме через Google Drive навчальний набір буде завантажуватись від ПК до середовища Google Colab. Результати навчання у вигляді графіків, натренованих моделей буде зберігатись в даному сховищі.

З підпискою Pro Google Colab дає на вибір одразу декілька варіантів апаратних прискорювачів (рис. 2.2)

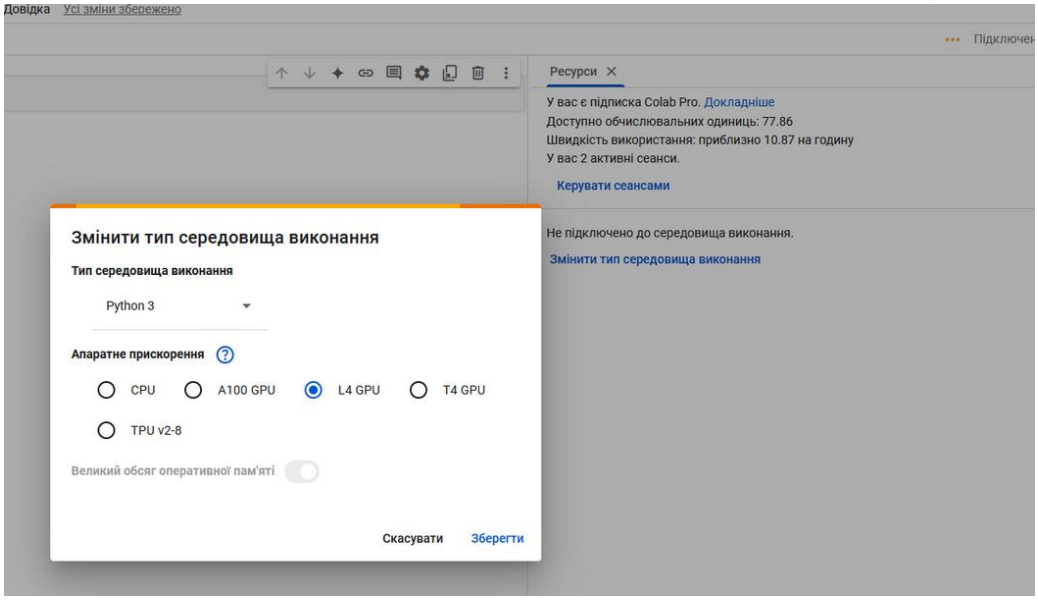

Рисунок 2.2 — Типи апаратного прискорення в середовищі Google Colab

Серед варіантів є:

- A100 GPU. Найпотужніший з запропонованих варіантів, підходить для великих моделей та об’ємних даних. Проте через високу продуктивність відбувається швидша втрата обчислювальної одиниці;
- L4 GPU. Оптимальний варіант для середніх та великих моделей. Забезпечує продуктивність, достатню для навчання на досить великих наборах даних, з помірним споживанням обчислювальних одиниць;
- T4 GPU. Менш швидка, проте ідеально підходить для невеликих моделей. Проте буде занадто повільною для навчання глибоких нейронних мереж на значному наборі даних.

Обрано L4 GPU для забезпечення ефективного балансу між продуктивністю та економією обчислювальних одиниць.

Мова програмування Python 3 найкраще підходить для навчання таких моделей. Ефективна вона і через велику кількість бібліотек, що використовується в роботі: matplotlib, seaborn, numpy, scikit-learn, shutil, cv2, pandas, plotly.

2.3 Нейронні мережі для навчання

В залежності від етапу та необхідної задачі для виконання буде розглянуто по декілька варіантів.

2.3.1 Нейронні мереж та Vision Transformer для сегментації

Семантична сегментація - це завдання комп'ютерного зору, яке передбачає класифікацію та присвоєння мітки кожному пікселю зображення. На відміну від виявлення об'єктів, яке ідентифікує і розміщує обмежувальні рамки навколо об'єктів, семантична сегментація забезпечує більш детальне розуміння зображення, окреслюючи межі об'єктів на рівні пікселів.

Один з перших типів архітектури, розроблений спеціально для задачі сегментації зображень, є Fully Convolutional Networks (**FCNs**). Основна ідея полягає в побудові «повністю згорткових» мереж, які приймають вхідні дані довільного розміру і видають вихідні дані відповідного розміру з ефективним висновком і навчанням. Для цієї нейронної мережі адаптовано сучасні класифікації мережі (AlexNet, мережу VGG та GoogLeNet) до повністю згорткових мереж і перенесено їхні вивчені репрезентації шляхом точного налаштування на задачу сегментації [9]. Також визначено нову архітектуру, яка поєднує семантичну інформацію з глибокого, грубого шару з інформацією про зовнішній вигляд з неглибокого, тонкого шару для створення точної та детальної сегментації (рис. 2.3).

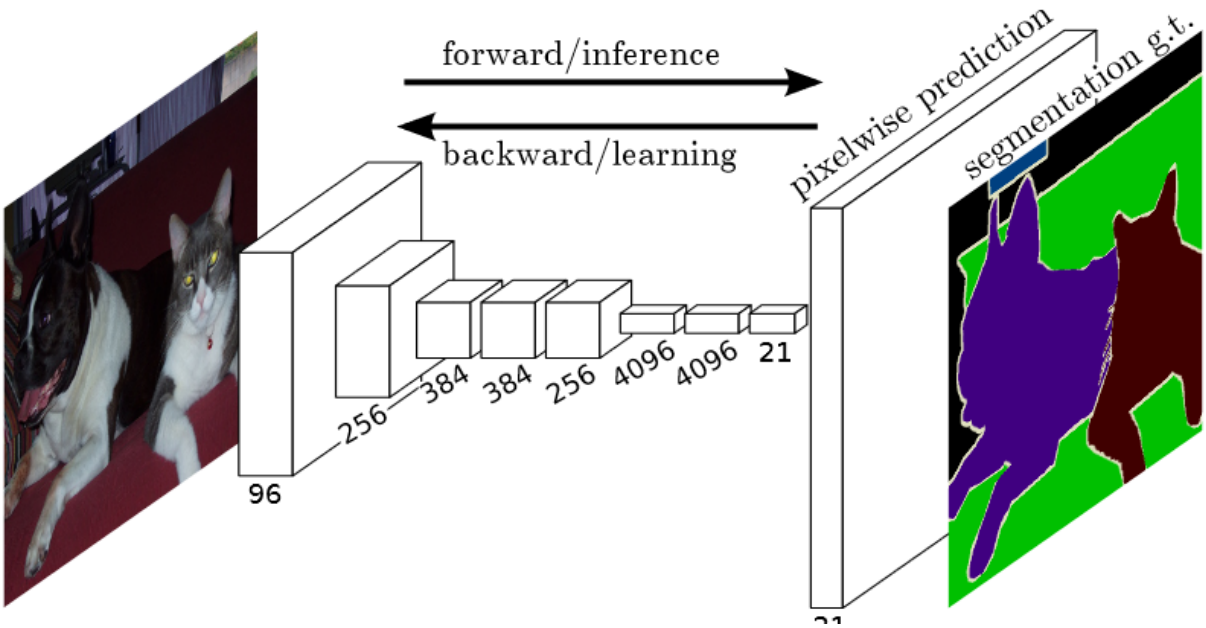

Рисунок 2.3 — Приклад реалізації FCNs

FCN генерує карту ознак (feature map), яка відповідає кожному пікселю вхідного зображення. Також використовує "deconvolution" або upsampling для відновлення розміру зображення до його оригінальної роздільної здатності. Це доволі легка архітектура, проте з плином часом все менше використовується.

U-Net одна з найпопулярніших архітектур для сегментації. Її назва пов’язана з U-подібною структурою, що включає шляхи скорочення (downsampling) і розширення (upsampling). Серед переваг: має "skip connections", які дозволяють передавати інформацію з початкових шарів до пізніших етапів; являється дуже універсальною моделлю і підходить для вирішення багатьох задач пов’язаних із сегментацією. Архітектура U-Net унікальна тим, що складається зі стискаючого та розгалужуючого каналів [10]. Стискаючий канал містить шари шифратора, які фіксують контекстну інформацію і зменшують просторову роздільну здатність вхідного сигналу, тоді як розширюючий канал містить шари дешифратора, які декодують закодовані дані і використовують інформацію зі стискаючого каналу за допомогою пропускних з'єднань для створення карти сегментації.

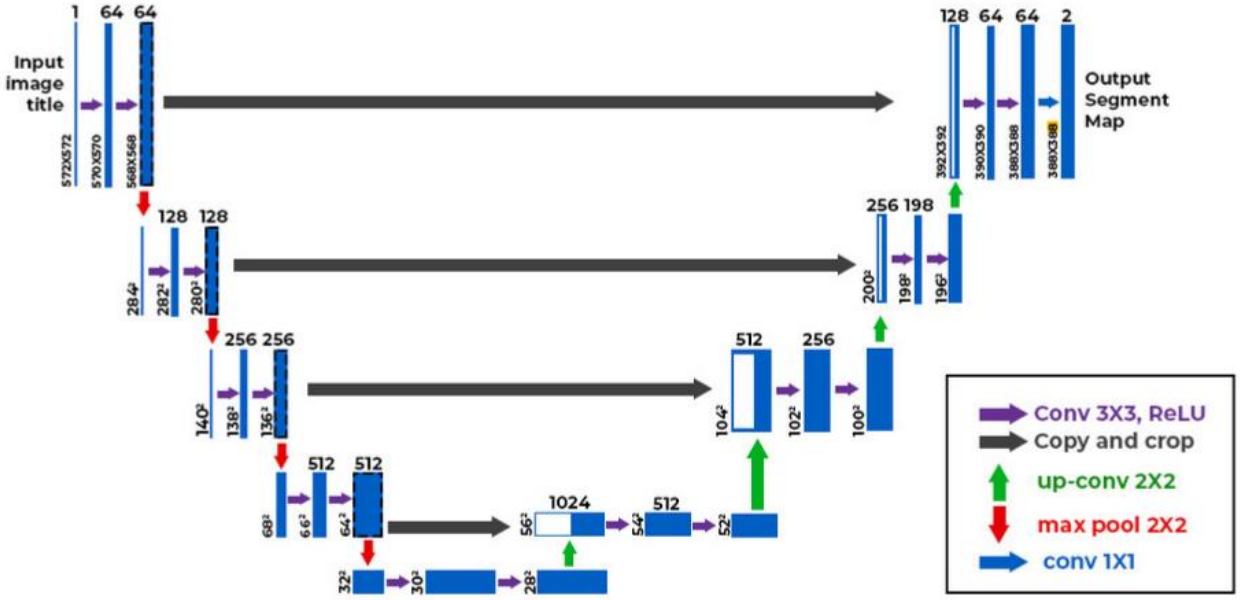

Рисунок 2.4 — Архітектура U-Net

Канал згортання відповідає за вилучення ознак із вхідного зображення, зменшуючи його роздільну здатність, але збільшуючи глибину ознак для отримання більш абстрактного представлення. Розширювальний канал використовує ці ознаки для відновлення просторової роздільної здатності, локалізації об'єктів і забезпечення точнішої сегментації, використовуючи пропускні зв'язки, які передають деталі від шляху згортання до декодера.

SegNet - це архітектура глибокого навчання, призначена для семантичної сегментації, метою якої є класифікація кожного пікселя зображення за наперед визначеною категорією. Це нейронна мережа кодер-декодер, спеціально розроблена для попіксельної сегментації зображень, що робить її високоефективною для задач, які вимагають детальної та точної сегментації [11]. Основна мета SegNet - виконувати семантичну сегментацію, навчаючись позначати кожен піксель зображення відповідно до його категорії.

DeepLabV3 - це вдосконалена архітектура нейронної мережі, призначена для семантичної сегментації зображень. Ця методика передбачає позначення кожного пікселя зображення класом, який відповідає тому, що цей піксель представляє. DeepLabV3+ є суттєвим кроком вперед у порівнянні зі своїми

попередниками в серії DeepLab, пропонуючи підвищену точність та ефективність сегментації складних зображень (рис. 2.5).

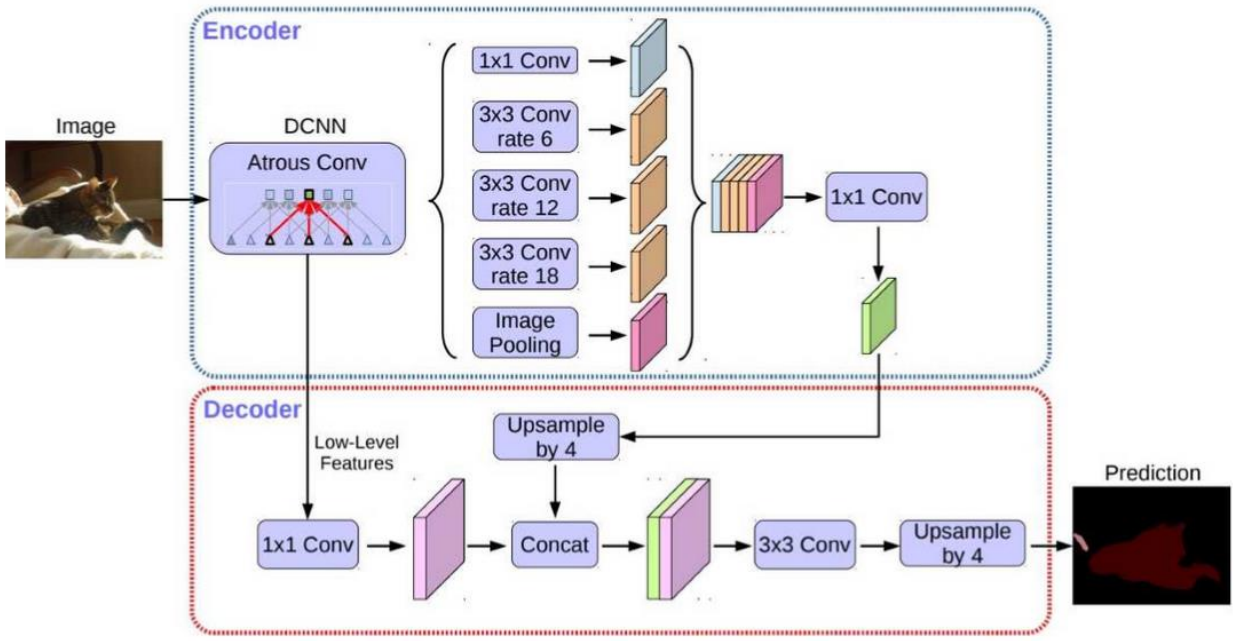

Рисунок 2.5 — Архітектура та реалізація DeepLabV3

DeepLabV3+ має архітектуру кодер-декодер, яка забезпечує ефективну сегментацію зображень. Кодер базується на модифікованій моделі Xception, яка використовує атрозійну згортку для захоплення ширшого контексту без втрати просторової роздільної здатності. Декодер уточнює семантичні ознаки, отримані кодером, шляхом поєднання з низькорівневими ознаками, що дозволяє точніше розпізнавати краї об'єктів і дрібні деталі. Завдяки такій структурі DeepLabV3+ досягає високої точності та чіткості сегментації, особливо на складних зображеннях [12].

Кожна з цих архітектур виконує глобальну задачу сегментації, проте підходи в кожної з них різні [13]. Наприклад семантичну сегментацію виконують U-Net, DeepLabv3+, PSPNet; інстансну сегментацію (окремих об’єктів) Mask R-CNN; доволі швидку сегментацію та як варіант при обмежених ресурсах - SegNet, MobileNet-based U-Net; детальна сегментація – поле роботи для HRNet, DeepLabv3+ (рис. 2.6).

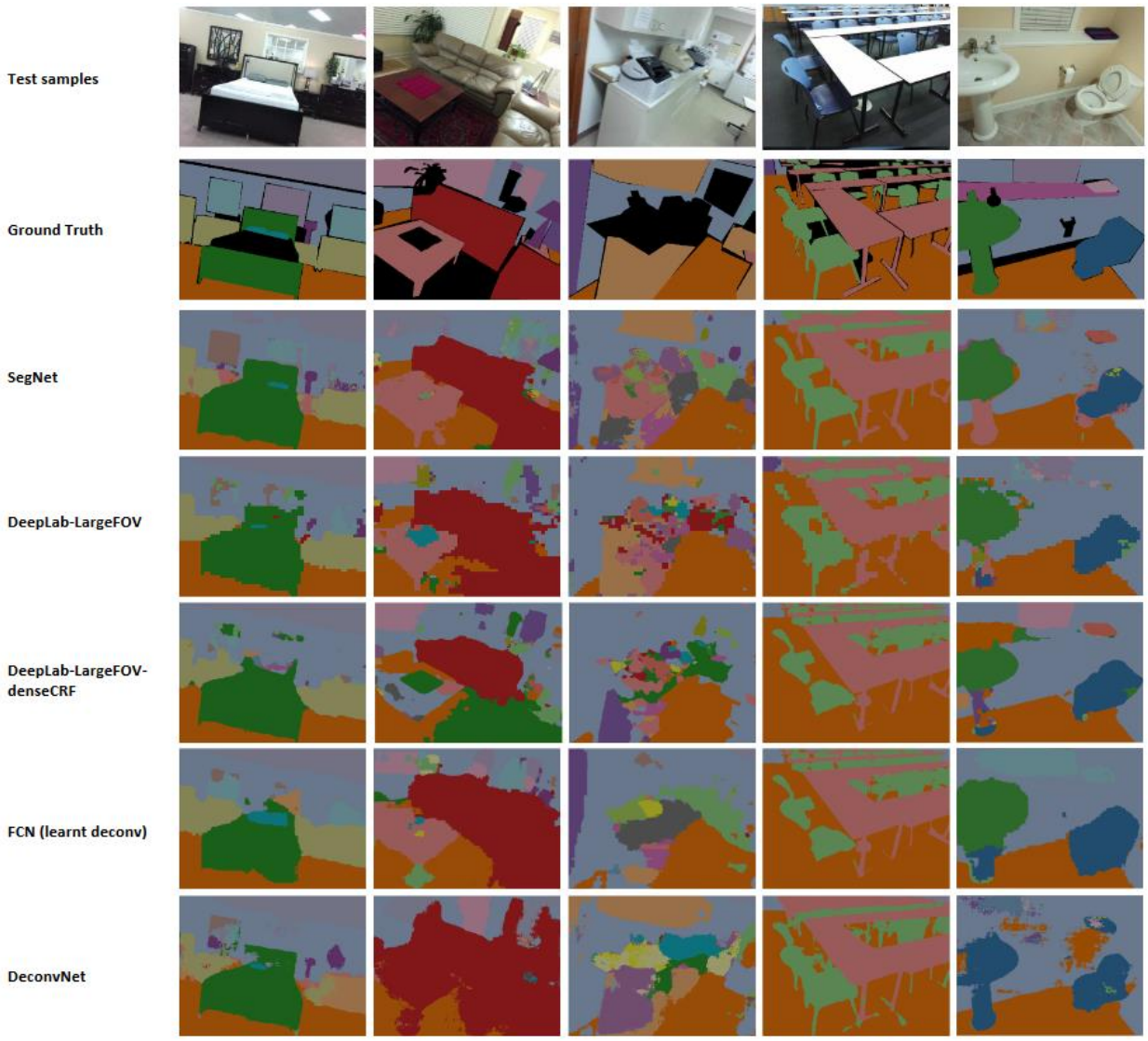

Рисунок 2.6 — Порівняння сегментації найпопулярніших архітектур

Якщо обирати нейронну мережу для виконання задачі семантичної сегментації (весь об’єкт піксель за пікселем) то найкращим варіантом був би вибір архітектури U-Net (це і пояснюється найбільшою популярністю мережі серед усіх конкурентів рис.2.7).

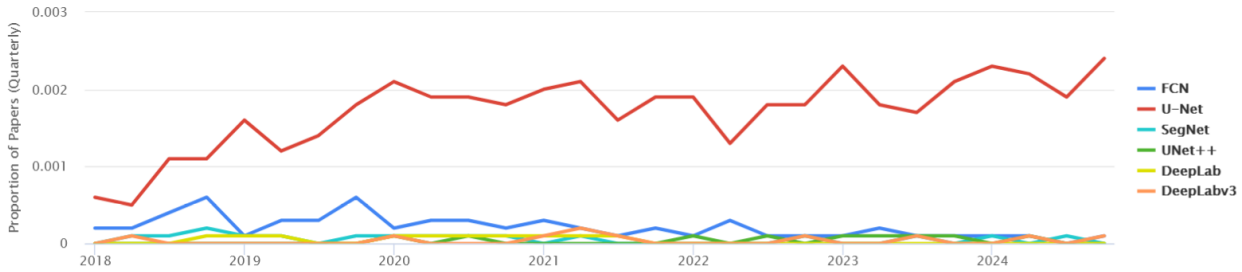

Рисунок 2.7 — Популярність використання архітектур для задач сегментації

Проте з часом на противагу згортковим нейронним мережам для вирішення задач сегментації з’явились Vision Transofmer (ViT). У ViT зображення розбивається на патчі (маленькі блоки), які обробляються як послідовність токенів, схожа на підхід у моделях обробки тексту, таких як Transformers. Одна з таких моделей, це Segment Anything Model (SAM) створена на базовій моделі VIT. Вона використовує трансформери для аналізу глобального контексту зображення через механізм self-attention. SAM – це модель, що вже навчена на дуже великих наборах даних (мільярди масок).

Якщо говорити про переваги U-Net над ViT, то в першу чергу слід виділити, що ця мережа вимагає менші обчислювальні витрати; вона краще підходить для задач, де представлено невеликий набір даних. Ця нейронна мережа більш гнучка у налаштуванні – може адаптуватись для тривимірних даних (наприклад, 3D U-Net).

Серед переваг SAM над U-Net можна назвати глобальне бачення. SAM враховує взаємозв'язки між усіма частинами зображення, що робить її більш точною для обробки складних наборах. Модель може сегментувати будь-який об’єкт у зображенні, навіть якщо він не був присутній у навчальному наборі. Масштабованість – завдяки навчанню на величезних наборах даних SAM здатна працювати універсально на нових доменах без додаткового навчання. В порівнянні з U-Net модель SAM буде більш ефективна на задачах, де дані відрізняються від тренувального набору. U-Net може не враховувати

глобальний контекст так добре, як SAM, що може призвести до менш точних результатів на великих і складних сценах.

Враховуючи те, що для роботи обрано доволі великий набір даних, що вимагає універсальності та роботи на різноманітних даних, SAM є кращим вибором.

2.3.2 Нейронні мережі для задач класифікації

Задача класифікації полягає в тому, щоб передбачити, чи належить щось до одного класу чи ні. Нейронні мережі стали важливим інструментом для класифікації. Нещодавні дослідження в галузі нейронної класифікації показали, що нейронні мережі є перпективною альтернативою різним традиційним методам класифікації. Перевага нейронних мереж полягає в наступних теоретичних аспектах. По-перше, нейронні мережі є самоадаптивними методами, керованими даними, оскільки вони можуть підлаштовуватися під дані без будь-якого явного визначення функціональної форми розподілу для базової моделі. По-друге, вони є універсальними функціональними апроксиматорами, оскільки нейронні мережі можуть апроксимувати будь-яку функцію з довільною точністю [14]. Оскільки будь-яка процедура класифікації шукає функціональний зв'язок між приналежністю до групи та атрибутами об'єкта, то точна ідентифікація цієї функції, що лежить в її основі, безсумнівно, важлива. По-третє, нейронні мережі є нелінійними моделями, що робить їх гнучкими у моделюванні складних взаємозв'язків реального світу. Нарешті, нейронні мережі здатні оцінювати апостеріорні ймовірності, що є основою для встановлення правила класифікації та проведення статистичного аналізу.

Для того, щоб вирішити проблему зникаючого/вибухаючого градієнта, архітектура **ResNET** запровадила концепцію під назвою «Залишкові блоки» (Residual Blocks). У цій мережі використано техніку, яка називається «пропуск з'єднань» [15]. Пропускне з'єднання з'єднує активації шару з наступними

шарами, пропускаючи деякі шари між ними. Це формує залишковий блок. Обнулення відбувається шляхом складання цих залишкових блоків разом (рис. 2.8).

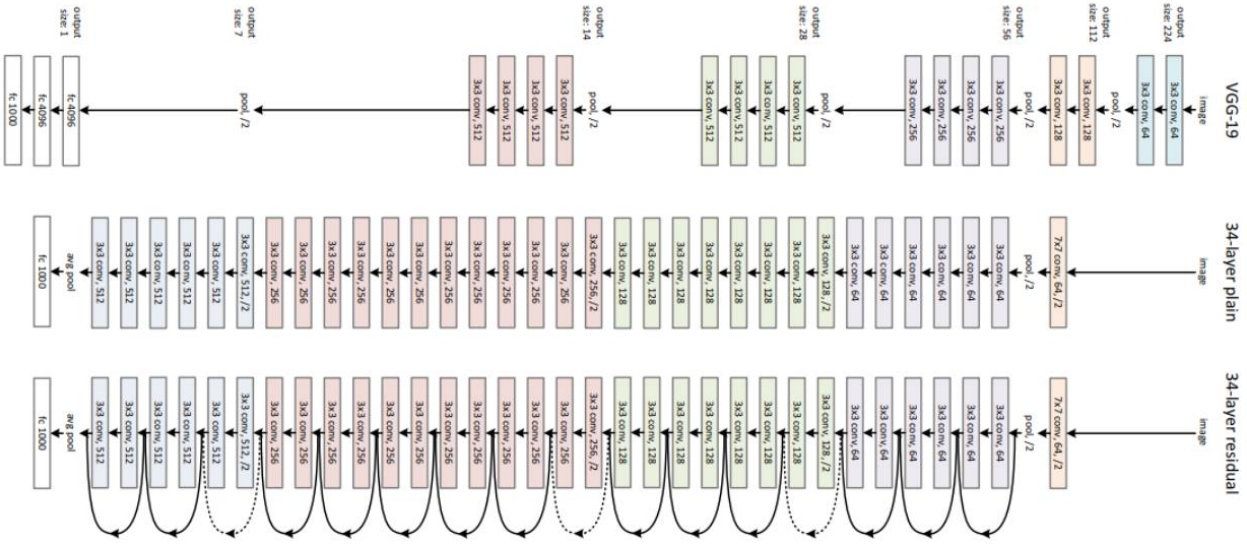

Рисунок 2.8 — Архітектура ResNET

ResNet це дуже точна модель для великих наборів даних, що ефективно працює навіть на складних зображеннях. Проте вимагає дуже високі обчислювальні витрати для моделей із великою кількістю шарів (наприклад, ResNet152).

Модель **VGG-16** - популярна модель класифікації зображень, яка виграла конкурс ImageNet у 2014 році. Вона має 16 шарів, включаючи 13 згорткових шарів і 3 повністю з'єднаних шари (рис. 2.9).

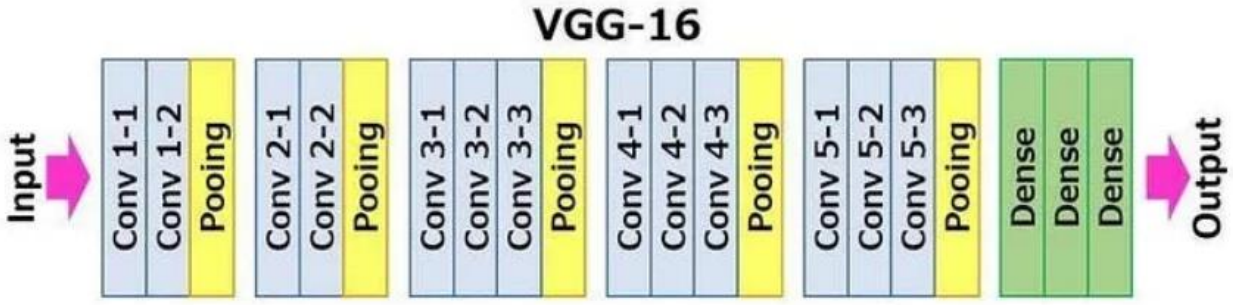

Рисунок 2.9 — Архітектура VGG-16

VGG-16 відома своєю простотою та ефективністю, а також здатністю досягати високої продуктивності в різних завданнях комп'ютерного зору, включаючи класифікацію зображень і розпізнавання об'єктів. Архітектура моделі включає стек згорткових шарів, за якими слідують шари максимального об'єднання з поступово зростаючою глибиною. Такий дизайн дозволяє моделі вивчати складні ієрархічні представлення візуальних особливостей, що призводить до надійних і точних прогнозів. Незважаючи на свою простоту в порівнянні з більш сучасними архітектурами, VGG-16 залишається популярним вибором для багатьох додатків глибокого навчання завдяки своїй універсальності та відмінній продуктивності.

EfficientNet, вперше представлена в роботі Tan and Le, 2019, є однією з найефективніших моделей (тобто такою, що вимагає найменшої кількості операцій FLOPS для виведення), яка досягає найсучаснішої точності як для imagenet, так і для звичайних задач навчання передачі класифікації зображень (рис 2.10). Найменша базова модель схожа на MnasNet, яка досягла близької до SOTA точності зі значно меншою за розміром моделлю. Впроваджуючи евристичний спосіб масштабування моделі, EfficientNet надає сімейство моделей (від B0 до B7), які представляють хороше поєднання ефективності та точності на різних масштабах.

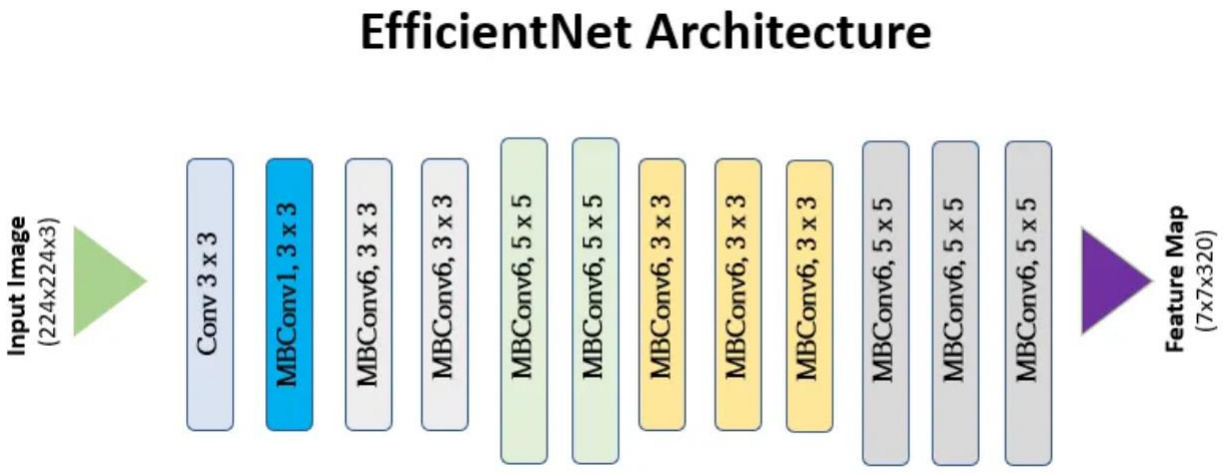

Рисунок 2.10 — Архітектура EfficientNet

У роботі Енріко Ранделлі (2023) було порівняно найновіші архітектури глибоких нейронних мереж для вирішення задачі класифікації зображень. Було розглянут моделі ResNet34 і ResNet50 сімейства ResNet, моделі EfficientNet-B4 і EfficientNet-B5 сімейства EfficientNet, розмір s і m сімейства EfficientNet_v2. Для кожної моделі завантажено її попередньо навчену версію з набору даних ImageNet і точно налаштовано ваги на користувацькому наборі даних, розміщеному на сайті Kaggle Brain Tumor Classification (MRI) [18], який добре підходить для виконання завдання класифікації зображень. Найкраще значення точності на тестовій вибірці має модель EfficientNet-B4 після 20 епох навчання (рис. 2.11).

Model	Test Accuracy	Train Accuracy	Val Accuracy
ResNet34	76.4	99.8	97.4
ResNet50	73.4	1.00	94.6
EfficientNet-B4	**79.9**	**1.00**	**98.4**
EfficientNet-B5	78.2	1.00	99.0
EfficientNet-V2-s	73.9	99.4	97.9
EfficientNet-V2-m	73.6	95.6	91.4
CCT-14_7x2_224	76.6	1.00	98.3
CCT-14_7x2_384	75.6	1.00	98.1

Рисунок 2.11 — Порівняння ефективності навчань різних архітектур на тренувальному наборі даних Kaggle Brain Tumor Classification (MRI)

Однак, для вирішення задачі класифікації для набору даних, який буде представлений в роботі (більше 50 класів) найкраще справлятись з задачею буде **DenseNet121**. Серед переваг DenseNet121:

- Ефективність параметрів. DenseNet121 забезпечує високий рівень повторного використання ознак через щільні зв’язки. Це дозволяє досягти високої точності при меншій кількості параметрів, ніж у ResNet або VGG;
- Висока точність. DenseNet121 добре працює на задачах із багатьма класами (наприклад, 50+ класів у поставленій задачі), оскільки вона зберігає деталі на всіх рівнях мережі;
- Збалансованість між швидкістю і точністю. У порівнянні з іншими моделями (наприклад, ResNet152 або Inception-ResNetV2), DenseNet121 забезпечує високу продуктивність при меншому використанні ресурсів;
- Оптимальна архітектура. DenseNet121 використовує атрозійні згортки (dilated convolutions), що дозволяє захоплювати ширший контекст зображення без втрати роздільної здатності;
- Попереднє навчання. Модель була попередньо навчена на ImageNet, що дає змогу використовувати її для задач із великим доменним розривом (transfer learning);
- Гнучкість для додаткового навчання. DenseNet121 легко адаптується до нових задач і може бути налаштована для будь-якої кількості класів шляхом додавання власних класифікаційних шарів.

Детальніше структуру цієї нейронної мережі буде розглянуто в Розділі 3 під час навчання нейронної мережі на задачі класифікації.

2.3.3 Нейронна мережа для геолокації

Поставлену задачу можна віднести до категорії геопросторового аналізу з використанням комп’ютерного зору, або більш повно задача може звучати як – знаходження географічних координат (широти та довготи) для певного зображення, що містить орієнтири, ландшафти, архітектуру та природні об’єкти. Складність задачі полягає в тому, що деякі регіони на планеті мають

дуже велику схожість між собою (ліси, деякі міські райони). Інша складність – знайти набір даних, що буде мати якомога більше унікальних візуальних ознак. Об’єкти на зображенні можуть бути зняті на різний висоті чи відстані, що додає шуму до даних, тому для точного розпізнавання необхідні великі й різноманітні набори геоприв'язаних зображень, це також може бути причиною низької точності моделі.

Регресія в задачі визначення координат — це процес передбачення точних числових значень широти та довготи об’єкта на зображенні. На відміну від класифікації, яка розподіляє зображення на обмежену кількість категорій (наприклад, географічних клітин), регресія передбачає два конкретних значення, які відповідають координатам на карті. Цей підхід дозволяє моделі працювати з безперервним простором, що дає змогу досягти більшої точності. Регресія є другою фазою після класифікації в комбінованих моделях. Спочатку модель обирає клітину, до якої належить зображення, а потім регресор уточнює координати в межах цієї клітини. Цей підхід зменшує обчислювальну складність і підвищує точність результату.

Основна мета даної моделі – поєднати можливості класифікації та регресії, щоб забезпечити як грубе визначення клітини (регіону), так і точні координати в її межах. Модель для геолокації буде побудована за наступними принципами:

- Побудова класів (географічних клітин). Спочатку буде виконуватись задача класифікації (класів тут буде значно більше, аніж в попередньому етапі), де кожна клітина відповідає певному класу. Мережа спочатку передбачає ймовірність, що зображення належить до конкретної клітини.
- Векторизація зображень. Використовуючи ЗНМ або Vision Transformers для створення векторних представлень зображення.

- Використовуючи спеціальні втрати, наприклад, Haversine Loss, щоб врахувати сферичну форму Землі.

Для вирішення кінцевого етапу задачі знаходження місцерозташування, а саме класифікації (більш детальної) та регресії, обиратись модель буде між нейронними мережами ResNet-50, EfficientNet-B4 та Vision Transformer. Структури та основні характерні ознаки моделей ResNet та EfficientNet було розглянуто в попередньому розділі, тому більш детально буде описано саме Vision Transformer, який буде обиратись для порівняння.

Vision Transformer для задач класифікації – архітектура глибокої нейронної мережі, створена для обробки зображень, яка адаптує ідеї їх використання для задач комп’ютерного зору. Замість використання згорткових операцій, Vision Transformer розбиває зображення на невеликі фрагменти (патчі) і обробляє їх як послідовності токенів, подібно до того, як трансформери працюють з текстовими даними. Кожен патч лінійно перетворюється у вектор фіксованої довжини (ембеддинг). Оскільки трансформери не враховують просторові зв'язки між патчами (на відміну від ЗНМ), додається позиційне кодування, щоб врахувати відносне розташування патчів у зображенні. Ембеддинги патчів обробляються стандартною архітектурою Vision Transformer (з шарами Self-Attention й повнозв’язними шарами), яка визначає взаємозв'язки між патчами. Потім додається спеціальний токен [CLS], який агрегує інформацію з усіх патчів і використовується для класифікації або іншого виходу (наприклад, регресії координат). І зрештою, останній шар Vision Transformer’а передає інформацію на вихідний шар (наприклад, лінійний шар), який виконує конкретне завдання класифікації чи регресії.

Один з таких Vision Transformer’ів було досліджено в роботі Лукаса Бейєра (2021) [19]. Досліджено пряме застосування цих моделей для задач розпізнавання зображення. Інтерпретація зображення відбувалась як рівність

патчів і обробка велась за допомогою стандартного трансформерного кодера (рис. 2.12). Ця доволі проста, але масштабована стратегія спрацювала напрочуд добре в поєднанні з попереднім навчанням на великих наборах даних.

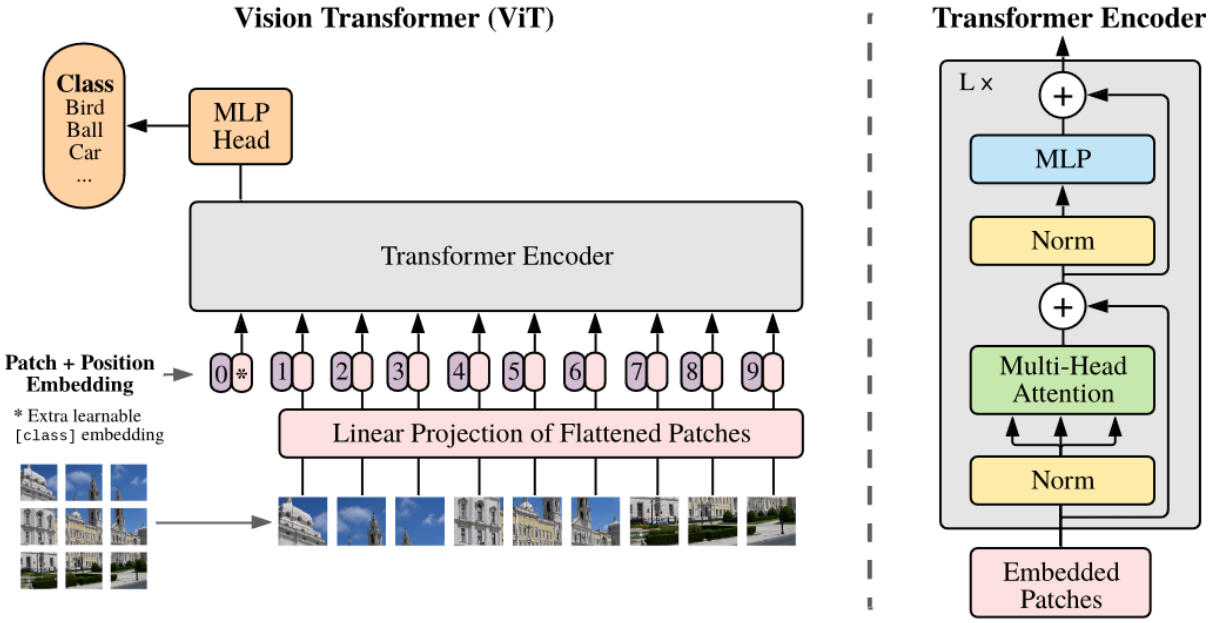

Рисунок 2.12 — Огляд моделі Vision Transformer

Проте виконуючи завдання класифікації і потім одразу регресії ці два підходи до побудови моделі (ЗНМ або Vision Transformer) існують ключові характерні особливості, які впливають на якість та швидкість навчання:

- Здатність до генералізації. Vision Transformer свої найкращі результати показує навчаючись на дуже великих наборах даних, маючи недостатню кількість зображень для навчання Vision Transformer може мати труднощі через слабке вивчення локальних ознак. ЗНМ завдяки локальній згортці, ефективно навчається навіть на менших наборах даних, механізм локальних фільтрів дозволяє витягувати суттєві локальні ознаки. В більших наборах даних ЗНМ також можуть показувати дуже хороші результати, але часто для

цього потрібно додавати більшої глибини (кількості шарів) для обробки складних даних;

- Обчислювальна ефективність. Vision Transformer може швидко масштабуватись, але вимагає високопродуктивних GPU з великою кількістю пам’яті. Звичайні фільтри ЗНМ оптимізовані для паралельних обчислень, тому вони працюють швидше на традиційних GPU навіть для великих наборів даних;
- Масштабування. Vision Transformer може легко масштабуватись до великих розмірів моделей, використовуючи більше шарів і шарів self-attention. ЗНМ теж легко масштабується, проте глибокі нейронні мережі можуть страждати від проблеми зникання градієнта;
- Практичне використання для задач з великою кількістю класів. Vision Transformer може мати перевагу завдяки глобальному сприйняттю, що полегшує класифікацію великої кількості класів, однак він потребує більших ресурсів для навчання і може бути чутливим до шуму в даних. ЗНМ добре працює з великою кількістю класів, але для значної кількості класів може вимагати великої кількості фільтрів і складнішої архітектури.

Якщо є доступ до великих обчислювальних ресурсів та наборів даних Vision Transformer може бути навіть кращим варіантом, проте в умовах обмеженості ресурсів, нерівномірно розподіленого набору даних з великою кількістю класів та враховуючи, що тренувальні та тестові зображення можуть мати різні дефекти, то буде обрано як модель ЗНМ.

При порівнянні двох моделей EfficientNet-B4 та ResNet50 (рис 2.11) основні метрики не показали критично великої різниці. Модель EfficientNet-B4 показує кращі результати, проте вона вимагає великих ресурсів через збільшену роздільну здатність і більшу глибину моделі. Перші спроби навчання на наборі даних 120 тис. зображень були саме за допомогою цієї

моделі, проте навіть використовуючи більш потужну GPU Tesla A100 навчання моделі не розпочалось протягом тривалого часу (рис. 2.13).

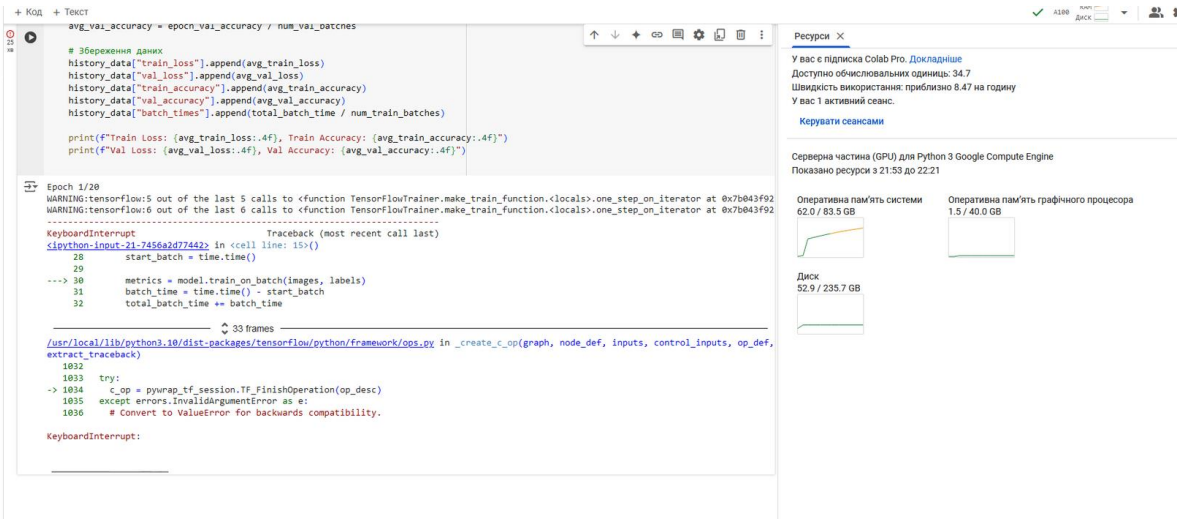

Рисунок 2.13 — Невдала спроба навчання моделі EfficientNet-B4 на обраному наборі даних

Тому, як хорошою альтернативою, хоч і з меншою точністю навчання, було обрано модель – ResNet50.

2.4 Вибір набору даних

Від правильного вибору набору даних сильно залежить ефективність та точність навчання, особливо коли це стосується задачі визначення місцезнаходження. Для вирішення поставленої задачі можна використати наступні ресурси:

- Платформи з відкритими даними. Одна з таких Kaggle містить велику кількість відкритих наборів даних, які можна використовувати для задач геолокації;
- Географічні платформи. Google Earth Engine, OpenStreetMap (OSM) або наприклад Mapillary дуже добре підходять як варіант для знаходження даних з супутникових зображень із широким географічним охопленням;

- Супутникові знімки. Landsat, Sentinel Hub, Planet Labs дають доступ до високоточних супутникових зображень. Проте такі тренувальні набори більше підходять для задач, де більшість операцій відбуваються з космічними знімками, а не з фото зробленими “вживу” через телефон чи камеру.

Існує кілька стандартних форматів. Вони зберігають геометричні дані разом з іншими описовими атрибутами географічних об'єктів. Наприклад, вони можуть зберігати координати маршруту для доріг разом з дорожнім покриттям, шириною, обмеженням швидкості, типом (міська вулиця, шосе тощо). Деякі з найпоширеніших форматів:Shapefile (найстаріший і найпоширеніший стандарт. Один «шейпфайл» фактично складається з набору файлів - один файл зберігає геометричні дані, інший файл зберігає користувацькі атрибути даних і т.д.). GeoPackage (новіша специфікація, яка набуває все більшої популярності збирає дані в один легкий файл бази даних SQLLite з декількома шарами)GeoJSON (використовує стандартний текстовий формат JSON).

Набором даних для тренування нейронної мережі було обрано датасет GeoLocation - Geoguessr Images (50K) [20]. Це набір даних з ~50 000 зображень geoguessr з конкурсу GeoWorld. Geoguessr - це онлайн-гра, в якій гравці переміщуються різними картами вуличного перегляду і намагаються вгадати місце розташування зображень, які вони бачать. Зображення були зібрані за допомогою селенових та географічних бібліотек Python. Кінцева мета цього набору даних - розглянути задачі машинного навчання та геолокації з багатокласовою класифікацією країн для зображень. Щоб даний набір даних був актуальним не тільки для задач класифікації, а й для розпізнавання місцезнаходження його надалі було оптимізовано та додано файл з координатами в csv форматі.

Набір даних складається з приблизно 150 різних підпапок, кожна з яких містить зображення з цієї країни. Як виявилось, розподіл навіть в таких географічних іграх не зовсім випадковий, і певні країни з'являються частіше, ніж інші. Розподіл країн нерівномірний, приблизно 1/5 частина набору даних містить зображення зі Сполучених Штатів. Через це кожна папка містить від 1 до 12 тисяч зображень.

Структура файлу показана на рис. 2.12.

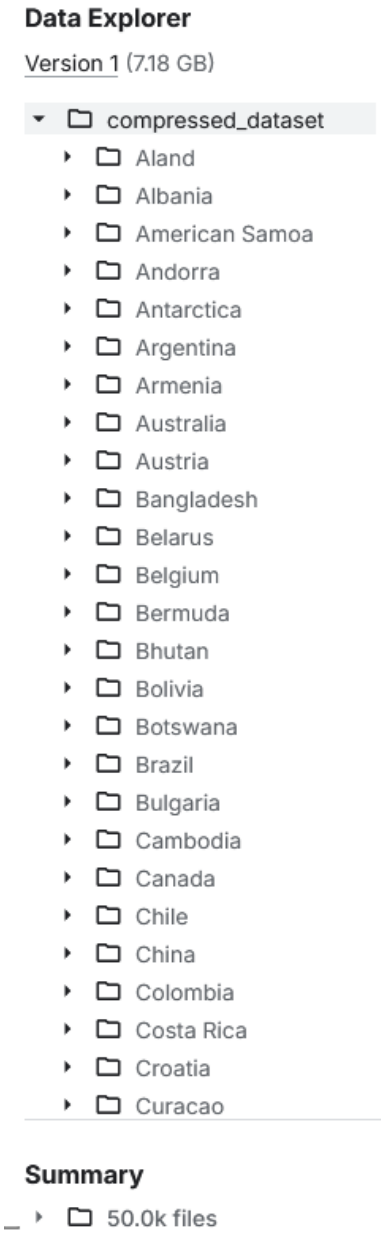

Рисунок 2.12 — Структура набору даних

Для ефективного та більш точного навчання цей набір даних необхідно оптимізувати. Зображення по країнам розбиті вкрай нерівномірно. Цю нерівномірність можна пояснити технічною складовою та розвитком країни – більш розвинуті в економічному плані країни частіше всього мають більше якісних оцифрованих даних. Також це може пояснюватись внутрішньою політикою держав – деякі країни можуть менш допускати публічного картографування території через чутливість даних чи традиції.

Далі представлено розподіл зображень набору даних:

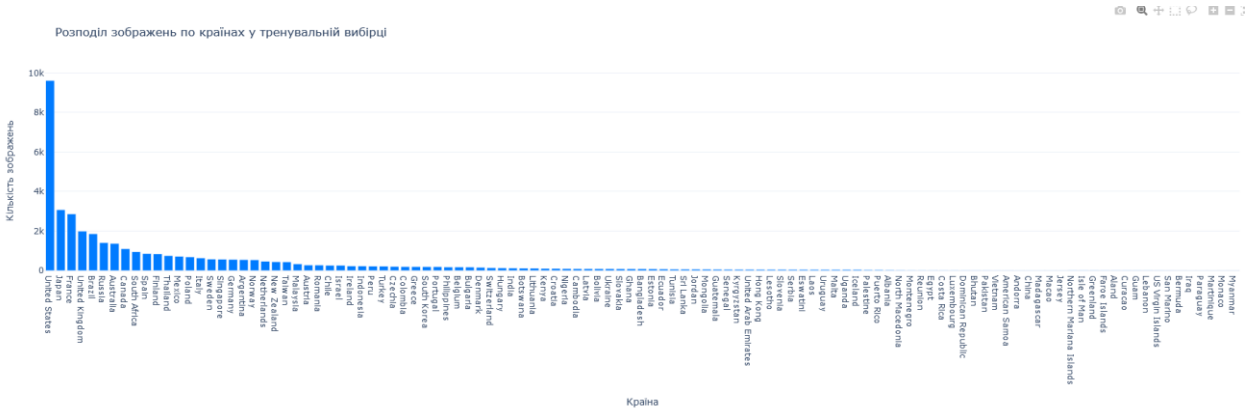

Рисунок 2.13 — Розподіл зображень по країнам у тренувальній вибірці

Як приклад наведено випадкове зображення, кожен знімок робився з гри GeoGuessr:

Випадкове зображення з тренувальної вибірки (Mexico)

Рисунок 2.13 — Випадкове зображення з набору даних

За допомогою теплової карти можна візуалізувати кількість зображень, доступних у наборі даних. Колір країни визначається кількістю доступних зображень. На колірному спектрі світло-зеленим кольором позначені країни з найменшою кількістю зображень, а темно-зеленим - країни з найбільшою кількістю зображень [21] (рис. 2.14).

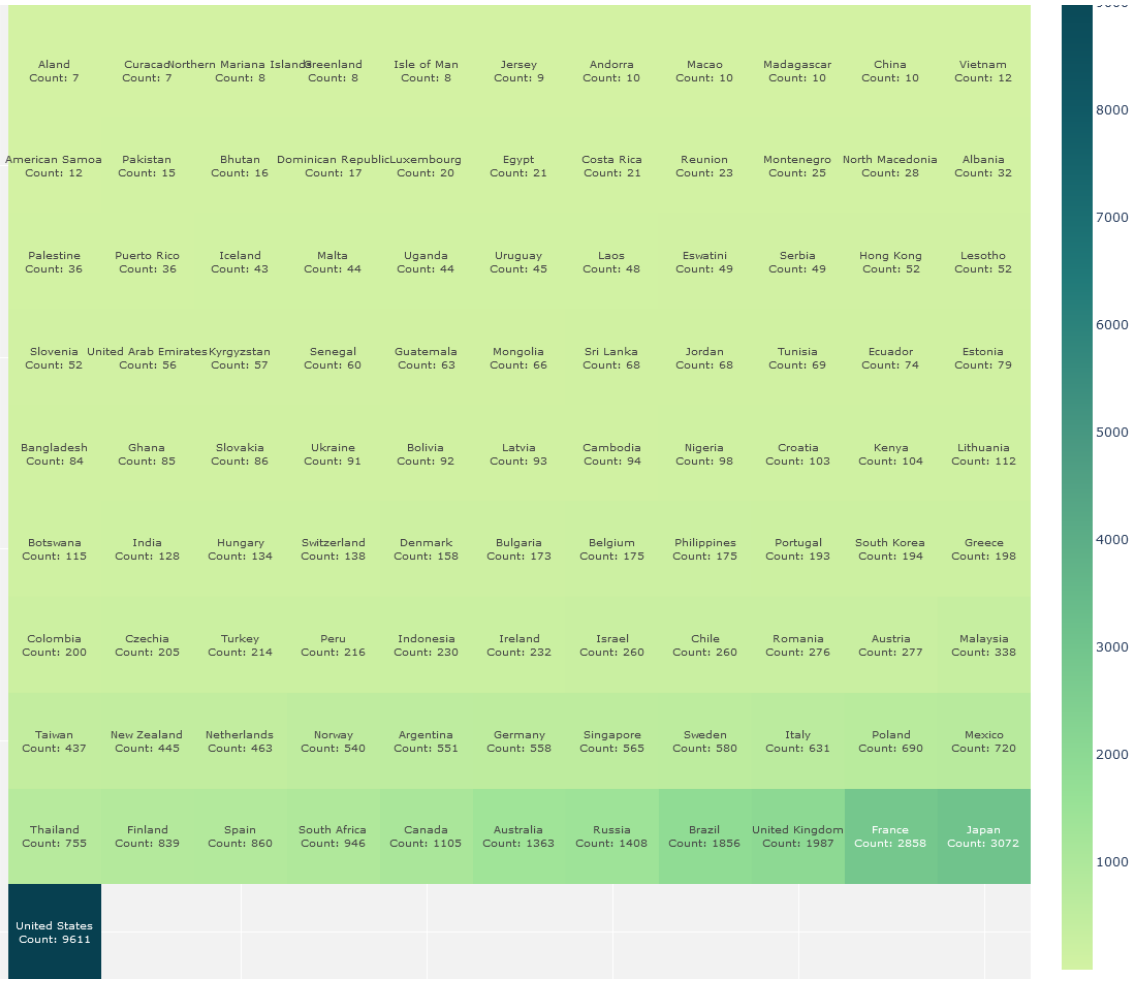

Рисунок 2.14 — Теплова карта вмісту набору даних

Для оптимізації набору даних необхідно видалити класи, де представлено мала кількість зображення для навчання. Орієнтуючись на графік розподілу можна припустити, що оптимальне число зображень для навчання має бути не

більше 80 зображень. Тому видалено всі класи, що мають кількість меншу за представлену:

```
# Розрахунок кількості зображень у кожному класі
filtered_distribution = df_data_distribution[df_data_distribution['frequency'] >= 80]

# Отримуємо список країн, які залишаються після фільтрації
filtered_countries = filtered_distribution['country'].tolist()

# Фільтруємо основний DataFrame, залишаючи лише записи з відібраними країнами
df_filtered_geo_data = df_geo_data[df_geo_data['country'].isin(filtered_countries)]

# Перевіряємо результати
print(f"Кількість країн після фільтрації: {len(filtered_countries)}")
print(f"Загальна кількість зображень після фільтрації: {len(df_filtered_geo_data)}")
```

```
Кількість країн після фільтрації: 62
Загальна кількість зображень після фільтрації: 48523
```

Рисунок 2.15 — Оптимізація набору даних

Підготовка “чистого набору даних” важлива для поставленої задачі, адже оригінальний набір даних містить нерелевантні зображення. Окрім того фільтрація допомагає прибрати зайві дані, тобто такі, де країни мають дуже малу кількість даних для навчання.

Надалі очищений набір даних розділяється на частини. Навчальний набір використовується для навчання моделі, щоб вона могла "запам'ятати" закономірності у даних. Містить більшу частину даних, в цьому випадку 80%.

Валідаційний набір даних використовується під час навчання для перевірки, як добре модель узагальнює, але без безпосереднього впливу на її параметри. Допомагає оцінити, чи не перенавчається модель (тобто, чи не запам'ятовує вона тільки навчальні дані). Становить 10% даних.

Тестовий набір даних використовується після завершення навчання, щоб оцінити реальну продуктивність моделі на нових, невідомих даних. Ці дані нейронна мережа бачить вперше. Складає 10% від загальної кількості даних.

Додатково для третього етапу визначення місцезнаходження буде використовуватись ще один набір даних на платформі Kaggle – Large Dataset

of Geotagged Images [22]. Цей набір даних складається з 4,2 мільйона (точніше 4,233,900) географічно прив'язаних зображень з набору даних YFCC100M [23]. Для кожного зображення його ідентифікатор, широта і довгота, де його було зроблено, а також саме зображення зберігаються у вигляді запису у форматі MessagePack. Кожен шард-файл (файл *.msg) містить 30 тисяч зображень (рис. 2.16).

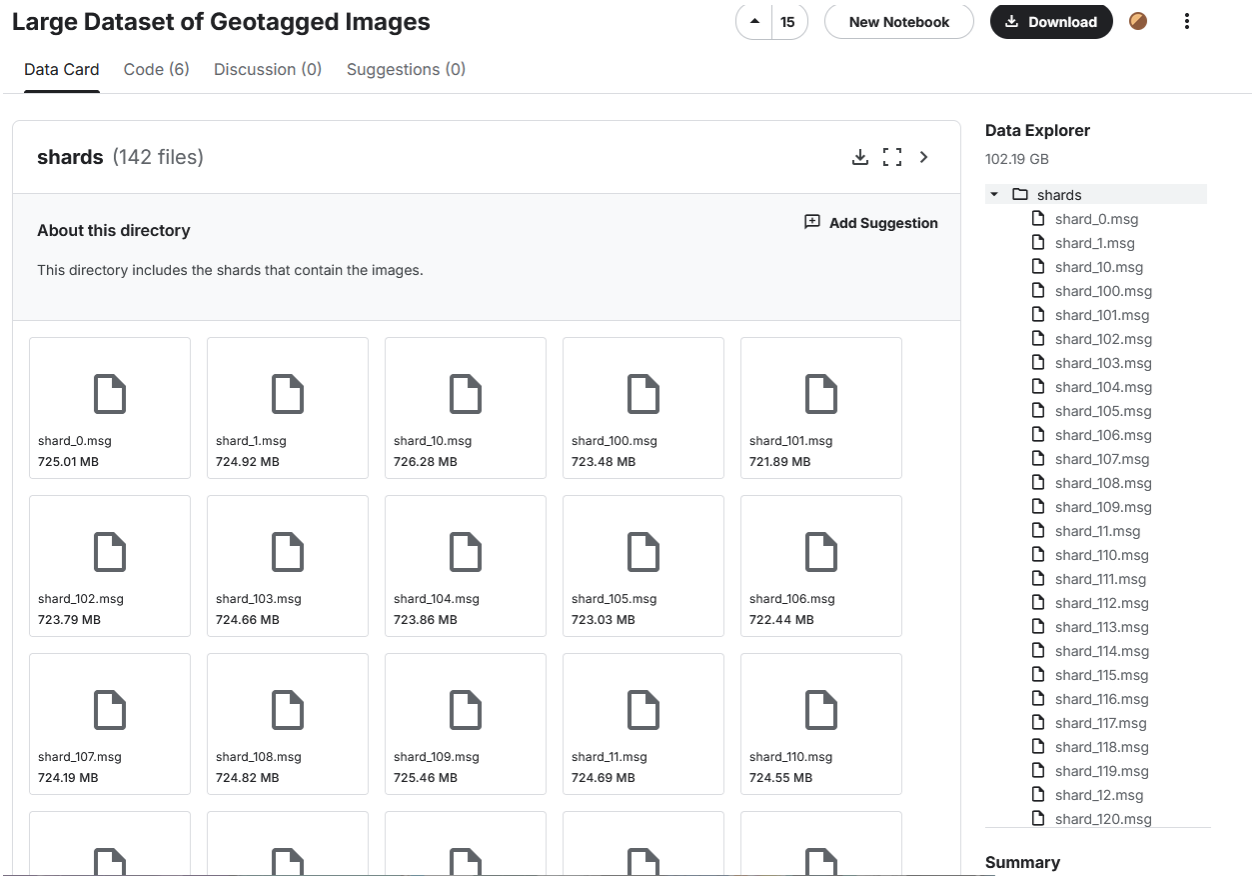

Рисунок 2.16 — Структура набору даних Large Dataset of Geotagged Images

Використовуватись в даному проекті будуть не всі дані з цього набору (через надзвичайно високі вимоги до навчання на такому набору даних), а додатково буде обрано 3 шард-файли (90 тис. зображень).

Висновки до другого розділу

В даному розділі представлено детальну методологію та інструментарій для вирішення задачі автоматизованого розпізнавання та геолокації ворожих цілей на основі використання нейронних мереж. Викладено поетапний підхід

до вирішення задачі, який включає сегментацію зображень, класифікацію об'єктів за країнами та визначення координат. Поєднання етапів забезпечує глибокий аналіз даних та підвищує точність і ефективність розпізнавання.

Для навчання моделей використовується інтерактивне хмарне середовище Google Colab Pro, яке забезпечує доступ до потужних обчислювальних ресурсів (GPU). Використання фреймворків TensorFlow/Keras спрощує розробку, тестування та навчання нейронних мереж, а також дозволяє використовувати попередньо натреновані моделі (ResNet, EfficientNet, DenseNet). Завдяки інтеграції з Google Drive зберігання моделей, даних і результатів обробки оптимізовано.

Нейронні мережі продемонстровані як ключовий інструмент для виконання задач сегментації, класифікації та прогнозування. Їх застосування дозволяє автоматизувати складні процеси аналізу геопросторових даних, забезпечуючи високу точність та адаптивність до нових умов.

Розглянуто та порівняно одразу декілька варіантів засобів штучного інтелекту для вирішення задач кожного етапу. Для вирішення задачі сегментації було відібрано Vision Transformer SAM, для класифікації - модель DenseNet121, а для розпізнавання координат на зображенні - ResNet50.

Обрано два набори даних, кожен з яких містить велику кількість зображень (перший - 50 тисяч, а другий - 90 тисяч), оптимізовано деякі з них та підготовлено до використання для навчання моделі.

3 РЕАЛІЗАЦІЯ ДОСЛІДЖЕННЯ

Виконання задачі буде проводитись в середовищі Google Colab в три етапи: сегментація зображень, класифікація, геолокація. Під кожну з задач буде своя модель / нейронна мережа, але всі вони будуть вчитись на одному і тому ж наборі даних. Використання результатів пройденого етапу максимально позитивно вплине на навчання нової моделі та буде використано для збільшення інформації для набору даних.

3.1 Сегментація об’єктів на зображеннях

Сегментація об’єктів є важливим етапом у задачі комп’ютерного зору, оскільки вона дозволяє вирішити кінцеву задачу прогнозування координат, а до цього – виділення окремі або об’єкти з фону на зображенні. Це особливо важливо для класифікації, оскільки сегментовані об'єкти забезпечують модель більш точними і релевантними даними, усуваючи перешкоди, такі як зайва інформація з фону. У контексті геолокації сегментація дозволяє фокусуватися на ключових об'єктах (наприклад, будівлях, ландшафтах чи дорогах), які містять геопросторову інформацію. Завдяки цьому підвищується точність моделей, які використовують ці об'єкти для визначення їхнього географічного розташування. Крім того, сегментовані об'єкти можуть бути зручніше інтегровані з іншими джерелами даних, такими як карти чи геоінформаційні системи (GIS).

Для обраного набору даних використовуватись буде модель SAM. Модель SAM – попередньо навчена на великому наборі даних SA-1B, який містить понад 1 мільярд масок, розподілених на 11 мільйонах ретельно відібраних зображень [24]. Основне завдання SAM є виконування дій сегментації швиидко на основі будь-якої підказки. Наприклад просторові, або текстові підказки, що ідентифікують об’єкт. У випадку з набором даних, що використовується там наведено тільки координати зображення, тому сама

модель повинна виділяти основі особливості на фото: характерні ознаки ландшафту (доріг, лісу, будівель і тд).

На даний момент компанією Ultralytics представлено дві моделі SAM: SAM base та SAM model. Відмінності двох моделей можна побачити на рисунку3.1

Model Type	Pre-trained Weights	Tasks Supported	Inference	Validation	Training	Export
SAM base	sam_b.pt	Instance Segmentation	✅	❌	❌	❌
SAM large	sam_l.pt	Instance Segmentation	✅	❌	❌	❌

Рисунок 3.1 — Порівняння моделей SAM base і SAM large

Модель base складається з кодера зображення, кодера підказки та декодера маски. У новій версії також додано деякі компоненти кодування пам'яті та уваги. Окрім того SAM base використовує ViT-B (Vision Transformer Base), модель із меншою кількістю параметрів і меншими обчислювальними витратами, в той час як SAM large: використовує ViT-L (Vision Transformer Large), яка є більшим варіантом SAM з покращеною точністю, але вищими вимогами до ресурсів. Враховуючи достатню тривалість всього циклу навчання та навчання нейронної мережі в декілька етапів, обрано модель SAM base (рис. 3.2)

```
[ ]  sam_checkpoint = "/content/drive/MyDrive/sam_vit_b_01ec64.pth"
     model_type = "vit_b"

     device = "cuda"

     # Завантаження моделі
     from segment_anything import sam_model_registry, SamAutomaticMaskGenerator

     sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
     sam.to(device=device)
```

Рисунок 3.2 — Підключення моделі SAM base

Після підключення моделі, розпакування набору даних в середовищі розробки та їх структуризації проведено перевірку на відповідність зображенням їх окремому файлу з координатами. Звірено кількість зображень та кількість наданих значень довготи та широти (рис. 3.3)

```python
import matplotlib.pyplot as plt

# Візуалізація декількох зображень із координатами
plt.figure(figsize=(10, 10))
for i in range(9):  # Вивести 9 зображень
    plt.subplot(3, 3, i + 1)
    plt.imshow(train_images[i])
    plt.title(f"Lat: {y_train.iloc[i, 0]:.2f}, Lon: {y_train.iloc[i, 1]:.2f}")
    plt.axis('off')
plt.show()
```

Рисунок 3.3 — Фрагмент коду та демонстрація декількох зображень із координатами

Після перевірки відповідності зображенням до наданих координат розпочато роботу над ініціалізацією генератора масок моделі SAM та реалізацією над усіма наданими зображеннями (рис 3.4).

```python
import cv2
from matplotlib import pyplot as plt
from segment_anything import SamAutomaticMaskGenerator

# Ініціалізація генератора масок
mask_generator = SamAutomaticMaskGenerator(sam)

# Завантаження зображення
img_path = "/content/dataset/dataset/106.png"  # Вкажіть шлях до зображення
image = cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB)

# Генерація масок
masks = mask_generator.generate(image)

# Функція для відображення масок
def show_anns(anns, axes=None):
    """
    Відображення згенерованих масок
    """
    if len(anns) == 0:
        return
    if axes:
        ax = axes
    else:
        ax = plt.gca()
        ax.set_autoscale_on(False)
    sorted_anns = sorted(anns, key=(lambda x: x['area']), reverse=True)
    polygons = []
    color = []

    for ann in sorted_anns:
        m = ann['segmentation']
        img = np.ones((m.shape[0], m.shape[1], 3))
        color_mask = np.random.random((1, 3)).tolist()[0]
        for i in range(3):
            img[:,:,i] = color_mask[i]
        ax.imshow(np.dstack((img, m*0.5)))
```

Рисунок 3.4 — Фрагмент коду, що виконує задачу ініціалізації та реалізації відображення масок

SAM аналізує зображення на рівні пікселів, щоб виявити області з подібними візуальними характеристиками, такими як текстури, кольори та форми. За допомогою попередньо навчених моделей SAM точно визначає межі кожного об'єкта, навіть якщо об'єкти перекриваються або є частиною складних сцен (рис. 3.5).

Рисунок 3.5 — Фрагмент коду з демонстрацією додавання масок до зображення

Після доволі тривалого навчання, навіть для менш вимогливої моделі base знадобилось багато часу для накладання масок на велику кількість зображень з набору даних (рис. 3.6 та 3.7).

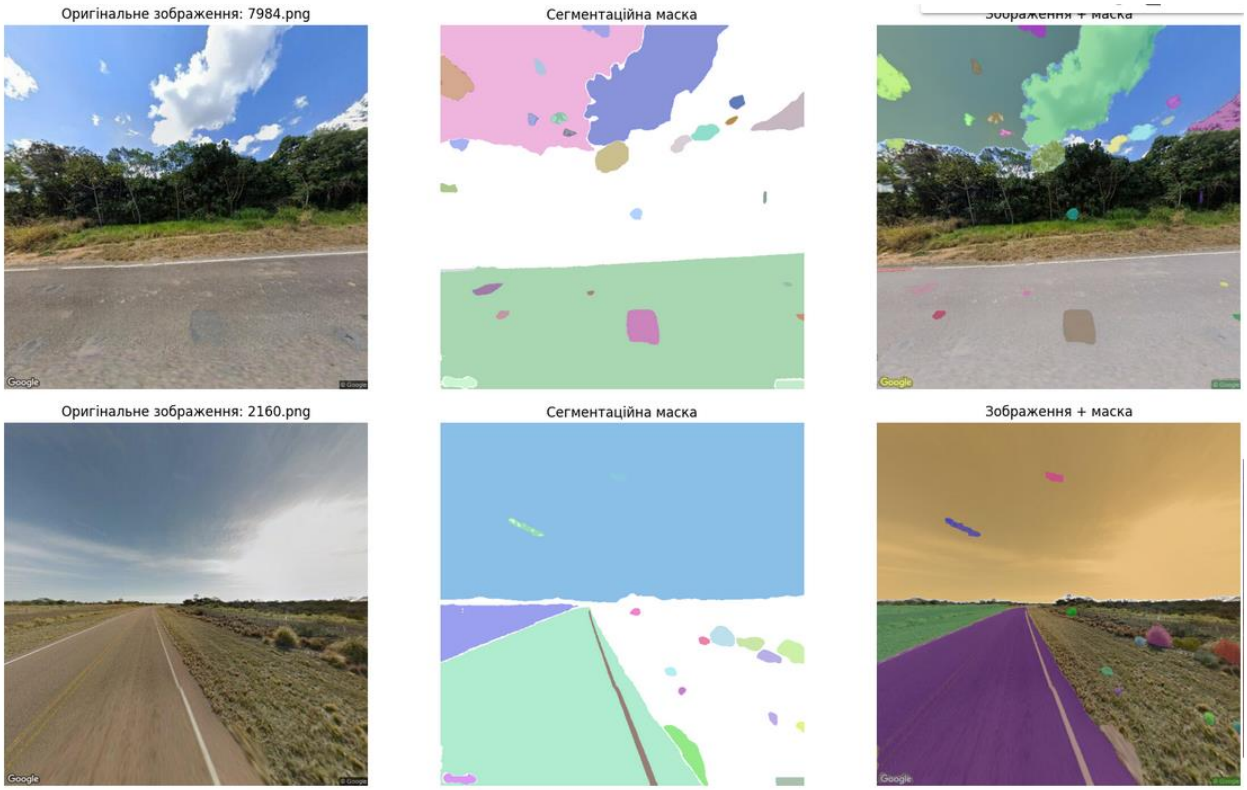

Рисунок 3.6 — Демонстрація накладання масок моделлю SAM на зображення з тренувального набору даних

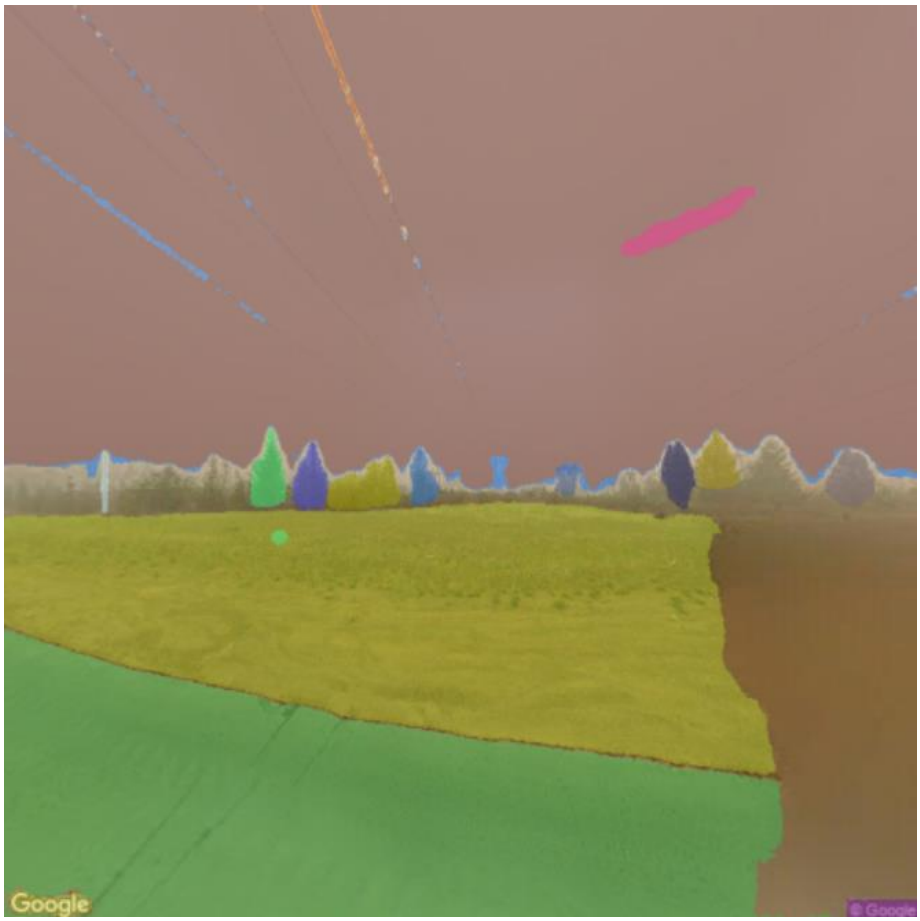

Рисунок 3.7 — Демонстрація поєднання накладеної маски та оригінального зображення

Далі кожне з отриманих зображень (у розподілі зображення + маска) буде використовуватись для наступного етапу – класифікації.

3.2 Класифікація зображень

Задачу класифікації можна вважати найважливішою в усій роботі, адже саме від цього етапу залежить на скільки надалі нейронна мережа буде правильно прогнозувати місцезнаходження за координатами. Класифікація дозволяє моделі навчитися розпізнавати специфічні візуальні особливості, характерні для різних географічних локацій, таких як архітектура, природний ландшафт, кліматичні умови або навіть стиль дорожньої інфраструктури. Це

створює базу для подальшого звуження області пошуку координат, обмежуючи їх до меж конкретної країни чи області.

Для кращої та точнішої роботи нейронної мережі було використано результати попереднього етапу сегментації. У майбутньому результати класифікації можуть бути використані як вхідні дані для моделей регресії, які прогнозуватимуть точні координати (широта та довгота) зображення. Крім того, такий підхід дозволяє створювати більш ефективні багаторівневі системи геолокації, де спочатку визначається країна, а потім уточнюються координати в межах її території. Це значно підвищує точність та продуктивність роботи моделі, а також знижує обчислювальні витрати при аналізі великих наборів зображень.

Першим етапом для задачі класифікації була підготовка даних. Саме на цьому моменті було оптимізовано набір даних з видаленням класів, де кількість зображень становила менше, аніж 80. Це було зроблено, адже до оптимізації навіть після проведення перших 7 епох навчання точність на тренувальному наборі даних складала біля 43%, а точність для тренувального набору – в районі 22%. Було вирішено оптимізувати набір даних для більш ефективного навчання моделі.

Після чого проведено збір метаданих із зображень: створено функцію extract_metadata_from_folder, яка обробляє кожну папку: витягує ширину, висоту, розмір файлу та повний шлях до кожного зображення. Для кожної категорії (країни) зібрано метадані та об'єднано їх у pandas.DataFrame. Після чого було проведено і аналіз даним по країнам – залишились лише ті країни (класи), що мають релевантну кількість даних для навчання, та які не вплинуть негативно на загальний результат навчання.

Використовуючи train_test_split, розділено дані на тренувальний (95%) та тестовий (5%) набори (рис. 3.8). Для кожного набору створено відповідні директорії, перевірено успішність поділу та переміщення даних.

```python
import os
from sklearn.model_selection import train_test_split
import shutil

# Шлях до основного датасету
dataset_path = "/content/full_dataset"
train_path = "/content/train_dataset"
test_path = "/content/test_dataset"

# Створення директорій для тренувального і тестового наборів
if not os.path.exists(train_path):
    os.makedirs(train_path)

if not os.path.exists(test_path):
    os.makedirs(test_path)

# Розділення даних для кожного класу
for class_folder in os.listdir(dataset_path):
    full_class_path = os.path.join(dataset_path, class_folder)
    images = os.listdir(full_class_path)

    train_images, test_images = train_test_split(images, test_size=0.1, random_state=42)

    # Копіювання тренувальних зображень
    class_train_path = os.path.join(train_path, class_folder)
    if not os.path.exists(class_train_path):
        os.makedirs(class_train_path)

    for img in train_images:
        shutil.copy(os.path.join(full_class_path, img), os.path.join(class_train_path, img))

    # Копіювання тестових зображень
    class_test_path = os.path.join(test_path, class_folder)
    if not os.path.exists(class_test_path):
        os.makedirs(class_test_path)

    for img in test_images:
        shutil.copy(os.path.join(full_class_path, img), os.path.join(class_test_path, img))

print("Датасет успішно розділено на тренувальний і тестовий набори.")
```

Рисунок 3.8 — Фрагмент коду з демонстрацією розподілу набору даних на тренувальний та тестовий

Далі проведено підготовку наборів даних для TensorFlow. Використано tf.keras.utils.image_dataset_from_directory для створення тренувального (з розділенням на тренувальну та валідаційну вибірки: 80% / 20% відповідно) та тестового набору. Налаштовано розмір кожного зображення (224x224) та розмір батчу 32 (рис. 3.9). Важливо встановити коректний розмір батчу для поставленої задачі, адже це параметр, який впливає на швидкість, стабільність

та пам'ять моделі під час навчання. Його оптимальне значення залежить від доступних ресурсів та розміру набору даних.

```
[ ]  # Параметри моделі
     img_width = 224  # Ширина зображення
     img_height = 224  # Висота зображення
     batch_size = 32  # Розмір батчу
     epochs = 10  # Кількість епох
     dropout_rate = 0.2  # Dropout rate
```

Рисунок 3.9 — Фрагмент коду з налаштуванням параметрів моделі

Після чого проведено побудову та компіляцію моделі. Завантажено попередньо натреновану модель DenseNet121 з вагами ImageNet (використано модель без верхніх шарів (include_top=False)). Для ефективнішого вирішення поставленої задачі додано два власні шари (рис. 3.10):

- Глобальний шар середнього пулінгу (GlobalAveragePooling2D). Це шар, який зменшує розміри просторового представлення (ширина × висота) до одного числа шляхом обчислення середнього значення по всій просторі для кожного каналу. Додавання такого шару необхідне для зменшення розмірності (скорочення кількості параметрів у моделі, зменшуючи ризик перенавчання, що робить навчання в значно ефективнішим) та збільшення глобальної інформації (врахування всієї карти ознак (feature map), створену згортковими шарами, для кожного каналу, що дає змогу моделі узагальнювати інформацію, зібрану згортковими шарами);
- Повнозв’язний шар із кількістю виходів, що відповідає кількості класів, та активацією softmax. Повнозв'язний шар (Dense) — це шар, у якому кожен нейрон входу пов'язаний з кожним нейроном виходу. Для задач класифікації softmax є стандартною функцією активації для останнього шару. Шар Dense дає можливість моделі передбачати ймовірність для кожного класу, що робить його ідеальним для

багатокласових задач, в той час як виходи softmax легко інтерпретуються як ймовірності для кожного класу.

```
[ ] from tensorflow.keras.applications import DenseNet121
    from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
    from tensorflow.keras.models import Sequential

    # Завантаження DenseNet121 з вагами ImageNet
    densenet = DenseNet121(
        input_shape=(img_height, img_width, 3),
        include_top=False,
        weights="imagenet"
    )

    # Побудова моделі
    model = Sequential([
        densenet,
        GlobalAveragePooling2D(),
        Dense(num_classes, activation="softmax")
    ])

    model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "sequential"
```

Layer (type)	Output Shape	Param #
densenet121 (Functional)	(None, 7, 7, 1024)	7,037,504
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dense (Dense)	(None, 124)	127,100

```
Total params: 7,164,604 (27.33 MB)
Trainable params: 7,080,956 (27.01 MB)
Non-trainable params: 83,648 (326.75 KB)
```

Рисунок 3.10 — Фрагмент коду з демонстрацією завантаження та налаштування моделі DenseNet121

Оптимізатором моделі обрано Adam, адже він являється одним із найпопулярніших алгоритмів оптимізації в машинному навчанні, який поєднує в собі найкращі властивості двох інших оптимізаторів: AdaGrad та RMSProp. Adam автоматично коригує швидкість навчання для кожного параметра на основі градієнтів, що зменшує необхідність ручного налаштування та завдяки своїй корекції моментів даний інтерпретатор відмінно працює з даними, що мають шум, або нерівномірні градієнти. Додано деякі колбеки перед навчанням: ModelCheckpoint (для збереження найкращої моделі на основі валідаційної точності), CSVLogger (для логування результатів навчання у CSV-файл) та LambdaCallback (виведення інформації про початок та завершення кожної епохи). Після чого було запущено навчання моделі (рис. 3.11)

```
[ ]  from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, LambdaCallback
     import time

     # Шлях для збереження найкращої моделі
     checkpoint_path = "/content/best_model.keras"

     # Шлях для збереження історії навчання
     log_path = "/content/training_log.csv"

     # Колбек для збереження найкращої моделі
     checkpoint_cb = ModelCheckpoint(
         filepath=checkpoint_path,
         monitor="val_accuracy",  # Збереження найкращої моделі за валідуючою точністю
         save_best_only=True,
         verbose=1
     )

     # Колбек для запису історії навчання в CSV
     csv_logger = CSVLogger(log_path, append=True)

     # Колбек для відображення часу після кожної епохи
     time_callback = LambdaCallback(
         on_epoch_begin=lambda epoch, logs: print(f"\n--- Епоха {epoch + 1} почалась ---"),
         on_epoch_end=lambda epoch, logs: print(f"--- Епоха {epoch + 1} завершилась ---"),
     )

     # Навчання моделі
     start_time = time.time()
     history = model.fit(
         train_ds,
         validation_data=val_ds,
         epochs=epochs,
         callbacks=[checkpoint_cb, csv_logger, time_callback]
     )
     end_time = time.time()

     print(f"Загальний час навчання: {end_time - start_time:.2f} секунд")

     # Оцінка на тестовому наборі
     loss, accuracy = model.evaluate(test_ds)
     print(f"Тестова точність: {accuracy:.2f}")
```

Рисунок 3.11 — Фрагмент коду з демонстрацією налаштування колбеків та початком навчання моделі

Навчання відбувалось 10 епох, точність та втрати навчання на тренувальних даних становили 84.36% та 0.50 відповідно (рис. 3.12).

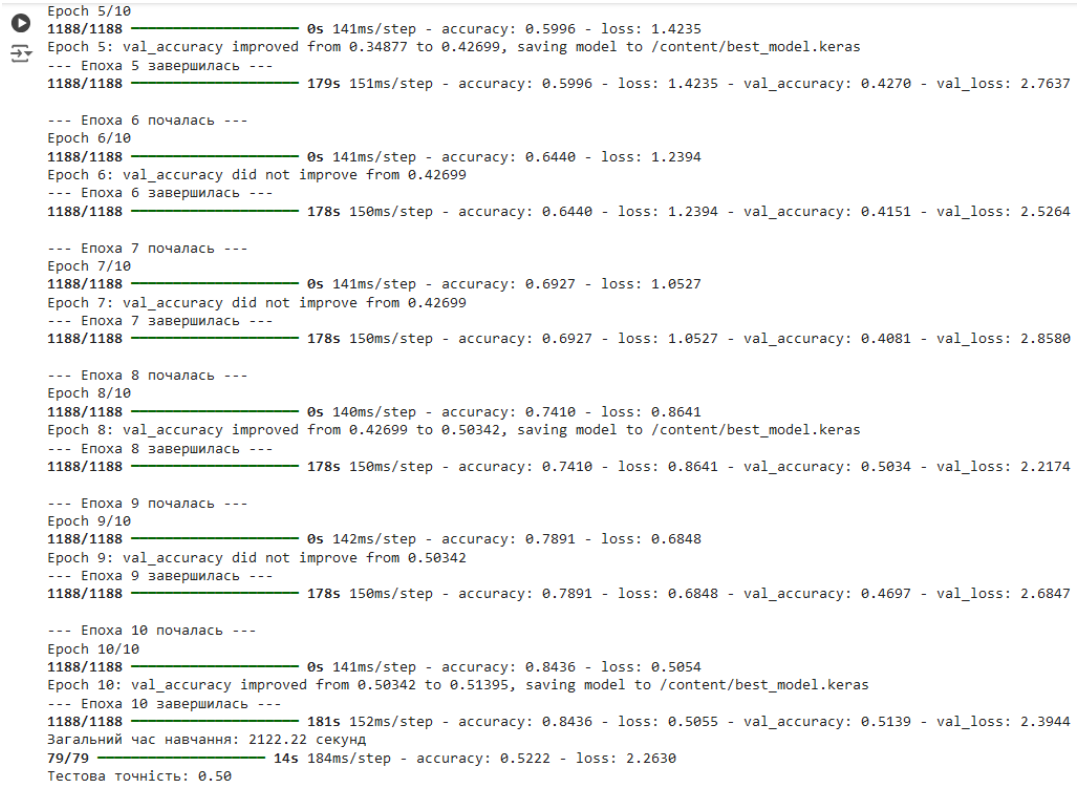

Рисунок 3.12 — Результати навчання моделі

Оптимізація набору даних та корегування параметрів моделі навчання допомогли збільшити точність навчання в 2 рази.

3.3 Геолокація (визначення координат)

На даному етапі задача знаходження координат по зображенню вирішується шляхом прив’язки зображень до географічних полігонів. Географічні полігони – це багатокутники на карті, які представляють певну територію або область. У цьому проекті вони використовуються для визначення географічного контексту зображень. Полігони містять просторову інформацію (координати вершин), яка дозволяє ідентифікувати, до якої області належить точка (координати зображення). Тому для вирішення такого завдання було дуже важливо знайти якісний набір даних, що містить не просто набір зображень з різних міст світу, а також щоб і містились інформація про координати місця на зображенні. Даний спосіб розпізнавання

місцезнаходження потребує також якісного файлу з географічними полігонами (у форматі .feather) та координати центроїдів (у форматі .npy).

Результати навчання моделей на попередніх етапах (сегментації та класифікації) є важливими для локалізації та зменшення області пошуку можливих координат місцерозташування, але основний акцент на останньому етапі навчання приділений саме на навчанні нової моделі з набору даних Large Dataset of Geotagged Images [20]. Навчання будл проведено подібно до дослідження виконаного користувачем Nemish_Extra [25]. Початкові етапи: робота з файлами shards, налаштування географічних полігонів були зроблені подібно до даної роботи, але обрана інша модель для навчання, та по-іншому інтерпретовано результати.

До 48523 відфільтрованих даних з попереднього набору даних додано ще 90 тисяч (що лежать в 3 файлах shards). Тому спочатку оброблено вхідні файли shards (у форматі .msg) із зображеннями та їх метаданими та розділено їх тренувальний, валідаційний та тестовий набори (рис. 3.13).

```
[ ]  import os

     shard_files = sorted([os.path.join(shards_dir, f) for f in os.listdir(shards_dir) if f.endswith('.msg')])

     # Розподіл: 80% train, 10% val, 10% test
     train_shards = shard_files[:int(len(shard_files) * 0.8)]
     val_shards = shard_files[int(len(shard_files) * 0.8):int(len(shard_files) * 0.9)]
     test_shards = shard_files[int(len(shard_files) * 0.9):]
```

```
[ ]  train_dir = "/content/train_images"
     val_dir = "/content/val_images"
     test_dir = "/content/test_images"

     os.makedirs(train_dir, exist_ok=True)
     os.makedirs(val_dir, exist_ok=True)
     os.makedirs(test_dir, exist_ok=True)
```

Рисунок 3.13 — Фрагмент коду з розподілом вхідних даних до 3 наборів даних

Інформація про координати, на відміну від попереднього набору даних, містилась не в окремому csv файлі, а в самій назві файлу. Витягнуто

координати з імен файлів та створено Dataframe для кожного набору (train_df, val_df, test_df), який збережено у форматі .pkl (рис 3.14).

```
[ ]     return pd.DataFrame(data)

    train_df = create_df(train_dir)
    val_df = create_df(val_dir)
    test_df = create_df(test_dir)

    train_df.to_pickle("train.pkl")
    val_df.to_pickle("val.pkl")
    test_df.to_pickle("test.pkl")
```

```
[ ] print(train_df.head())
    print(val_df.head())
    print(test_df.head())
```

```
                                          id        lat         lng
0  19.530666_103.768922_e029416205732.jpg.jpg  19.530666  103.768922
1   9.638661_97.965087_10992211884891.jpg.jpg   9.638661   97.965087
2   50.115363_8.689815_25d62681901751.jpg.jpg  50.115363    8.689815
3   43.953266_4.804522_b6554228620715.jpg.jpg  43.953266    4.804522
4  51.465985_-2.554528_8a4b6754899341.jpg.jpg  51.465985   -2.554528
                                          id        lat         lng
0  40.758415_-73.971065_425210978084784.jpg.jpg  40.758415  -73.971065
1    51.503373_-0.119733_27d26539832669.jpg.jpg  51.503373   -0.119733
2  -37.853523_144.894774_1be36197046216.jpg.jpg -37.853523  144.894774
3   41.066224_-73.840245_881b4447729306.jpg.jpg  41.066224  -73.840245
4    52.509508_13.376712_2e5c2323878593.jpg.jpg  52.509508   13.376712
                                          id        lat         lng
0  19.444822_-100.324544_ccdb3280632824.jpg.jpg  19.444822 -100.324544
1    53.352341_-6.261434_df292532061273.jpg.jpg  53.352341   -6.261434
2    63.674167_22.702833_3ac85950148580.jpg.jpg  63.674167   22.702833
3  38.482849_-122.748291_e5ad2517751436.jpg.jpg  38.482849 -122.748291
4     30.188037_-97.794241_005312827578.jpg.jpg  30.188037  -97.794241
```

Рисунок 3.14 — Фрагмент коду із демонстрацією створення Dataframe

Наступним кроком було зроблено прив’язку зображень до географічних полігонів (рис. 3.15). Полігон у цій роботі представлений багатокутною фігурою, заданою набором координат (широта та довгота). Полігон описує не тільки розподіл по країнам, а і дає межі по деяким містам / регіонам. Географічні полігони для цієї задачі завантажуються з файлу cluster_split_cells.feather [26]. Цей файл містить геопросторові дані, які описують множину полігонів: кожен полігон має унікальний індекс (наприклад, index або label), полігони зберігаються у форматі, сумісному з

бібліотекою GeoPandas. Полігони використовуються для визначення, чи входить точка (координати зображення) в межі певного полігона. Це реалізується за допомогою просторового приєднання (spatial join), яке виконується через бібліотеку GeoPandas. Під час цієї операції для кожного зображення, представленого точкою (широта і довгота), перевіряється, до якого полігона (або кількох полігонів) воно належить. Якщо точка входить в полігон, їй призначається мітка (label) цього полігона. Кожне зображення отримує числову мітку (наприклад, label = 9), яка відповідає полігону. Це дозволяє перевести координати в дискретні класи, що спрощує задачу класифікації для моделі. У результаті, знаючи зображення, модель може передбачити, до якого полігона воно належить, а отже, визначити географічне положення зображення.

```
[ ] import geopandas as gpd
    import pandas as pd

    # Шлях до файлу, який уже в робочій директорії
    merged_cells_path = "/content/cluster_split_cells.feather"

    # Функція для обробки DataFrame через `image_cell_index`
    def process_image_cell_index(points_csv, merged_cells_path):
        # Завантажуємо полігони
        merged_cells_df = gpd.read_feather(merged_cells_path)

        # Завантажуємо DataFrame із координатами
        points_df = pd.read_pickle(points_csv)

        # Створюємо GeoDataFrame для координат
        geometry = gpd.points_from_xy(points_df['lng'], points_df['lat'])
        geo_df = gpd.GeoDataFrame(points_df, crs="EPSG:4326", geometry=geometry)

        # Виконуємо просторовий join для визначення полігонів
        joined_df = gpd.sjoin(geo_df, merged_cells_df, how="left", predicate="within")

        # Перейменовуємо колонку з індексом полігону
        joined_df = joined_df.rename(columns={'index_right': 'label'})

        # Видаляємо записи, що не потрапили в жоден полігон
        before_drop = len(joined_df)
        joined_df = joined_df.dropna(subset=['label'])
        after_drop = len(joined_df)
        print(f"Видалено {before_drop - after_drop} записів, які не потрапили до полігонів.")

        # Перетворюємо label у ціле число
        joined_df['label'] = joined_df['label'].astype(int)

        # Видаляємо зайві колонки
        final_df = joined_df[['id', 'lat', 'lng', 'label']]
        return final_df

    # Обробка даних для train, val та test
    train_processed = process_image_cell_index("train.pkl", merged_cells_path)
    val_processed = process_image_cell_index("val.pkl", merged_cells_path)
    test_processed = process_image_cell_index("test.pkl", merged_cells_path)

    # Збереження оброблених DataFrame у pickle-файли
    train_processed.to_pickle("train_with_labels.pkl")
```

Рисунок 3.15 — Фрагмент коду із демонстрацією прив’язки зображень до полігонів

В результаті отримано набір даних, що мають інформацію про місцерозташування об’єкту на зображенні, їх id та їх мітку (рис. 3.16).

```
Validation DataFrame:
                                        id        lat        lng  label
1   18.501505_73.934812_3f6a8168494063.jpg  18.501505  73.934812   1025
2   52.223889_-1.860313_a4e65107075318.jpg  52.223889  -1.860313    806
3  38.916952_-77.024052_4cbc3007016516.jpg  38.916952 -77.024052   1914
4   41.52482_-72.074861_1ea84768191717.jpg  41.524820 -72.074861   2234
5  44.242616_-68.309211_885d2740665852.jpg  44.242616 -68.309211   2010
<class 'pandas.core.frame.DataFrame'>
Index: 28899 entries, 1 to 29999
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      28899 non-null  object
 1   lat     28899 non-null  float64
 2   lng     28899 non-null  float64
 3   label   28899 non-null  int64
dtypes: float64(2), int64(1), object(1)
memory usage: 1.1+ MB
None
```

Рисунок 3.16 — Результат прив’язки зображень до полігонів

Наступним кроком було підготовлено клас Dataset, який читає зображення, застосовує трансформації (Resize, ToTensor) і передає їх разом із мітками. Дані були завантажені у DataLoader для тренування, валідації та тестування (рис. 3.17).

```
[ ] import torch
    from torch.utils.data import Dataset, DataLoader
    from torchvision import transforms
    from PIL import Image

    # Створення класу для PyTorch Dataset
    class ImageDataset(Dataset):
        def __init__(self, df, image_dir, transform=None):
            self.df = df
            self.image_dir = image_dir
            self.transform = transform

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

        def __getitem__(self, idx):
            row = self.df.iloc[idx]
            image_path = os.path.join(self.image_dir, row['id'])
            image = Image.open(image_path).convert("RGB")
            label = row['label']
            if self.transform:
                image = self.transform(image)
            return image, label

    # Трансформації для зображень
    transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Розмір для ResNet або іншої моделі
        transforms.ToTensor()
    ])

    # Створення Dataset
    train_dataset = ImageDataset(train_processed, "train_images", transform=transform)
    val_dataset = ImageDataset(val_processed, "val_images", transform=transform)
    test_dataset = ImageDataset(test_processed, "test_images", transform=transform)

    # DataLoaders
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

    print("Датасети та DataLoader створені.")
```

Рисунок 3.17 — Підготовка набору даних до навчання

Наступним етапом було налаштування моделі нейронної мережі на основі попередньо натренованої моделі ResNet50 із додаванням нових шарів (рис. 3.18). ResNet50 використовує residual connections (залишкові зв’язки), щоб розв’язати проблему зникнення градієнтів і сприяти навчанню глибоких мереж. Завантажена модель натренована на наборі даних ImageNet, що охоплює тисячі класів і мільйони зображень. Нові шари, що додано:

- GlobalAveragePooling2D. Застосовується замість традиційного Flatten для зменшення розмірності вихідного тензора від згорткових шарів;

- Dense (512, activation='relu'). Повнозв'язний шар із 512 нейронами та функцією активації ReLU. Реалізує нелінійність, дозволяючи моделі краще навчатись складним закономірностям;
- Dropout. Регуляризація для уникнення перенавчання. Випадковим чином обнуляє певний відсоток нейронів під час тренування.

```
[ ] from tensorflow.keras.applications import ResNet50
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
    from tensorflow.keras.optimizers import Adam

    # Завантаження базової моделі ResNet50
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

    # Додавання користувацьких шарів
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(dropout_rate)(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    # Побудова моделі
    model = Model(inputs=base_model.input, outputs=predictions)

    # Замороження базових шарів
    for layer in base_model.layers:
        layer.trainable = False

    # Компіляція моделі
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    print("Модель ResNet50 налаштована та готова до навчання.")

⇒ Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
  94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
  Модель ResNet50 налаштована та готова до навчання.
```

Рисунок 3.18 — Налаштування моделі ResNet50 до навчання

Проведено навчання із задачею класифікації: на скільки нейронна мережа вгадала до якого мітки відноситься задане зображення. Навчання відбувалось 20 епох (рис. 3.19).

```
Epoch 10/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.5120 - loss: 1.7744 - val_accuracy: 0.3696 - val_loss: 2.6132
Epoch 11/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.5383 - loss: 1.6732 - val_accuracy: 0.3668 - val_loss: 2.6248
Epoch 12/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.5620 - loss: 1.5867 - val_accuracy: 0.3719 - val_loss: 2.6132
Epoch 13/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.5785 - loss: 1.5093 - val_accuracy: 0.3619 - val_loss: 2.6779
Epoch 14/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.5983 - loss: 1.4205 - val_accuracy: 0.3679 - val_loss: 2.6403
Epoch 15/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.6225 - loss: 1.3427 - val_accuracy: 0.3684 - val_loss: 2.6847
Epoch 16/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.6439 - loss: 1.2585 - val_accuracy: 0.3698 - val_loss: 2.7058
Epoch 17/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.6648 - loss: 1.1867 - val_accuracy: 0.3647 - val_loss: 2.7334
Epoch 18/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.6847 - loss: 1.1233 - val_accuracy: 0.3657 - val_loss: 2.7867
Epoch 19/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.7023 - loss: 1.0551 - val_accuracy: 0.3641 - val_loss: 2.8214
Epoch 20/20
2375/2375 ━━━━━━━━━━━━━━━━━━━━ 26s 11ms/step - accuracy: 0.7209 - loss: 0.9883 - val_accuracy: 0.3655 - val_loss: 2.8260
Навчання завершено!
```

Рисунок 3.19 — Результати навчання моделі

В результаті модель показала точність навчання на тренувальних та тестових наборах даних 72% та 36.66% відповідно, а величина втрат для тренувальних та тестових становила 0.98 та 2.82 відповідно.

Висновки до третього розділу

У цьому розділі представлено реалізацію трьох ключових етапів вирішення задачі автоматизованого визначення геолокації об’єктів на зображеннях за допомогою нейронних мереж: сегментації, класифікації та геолокації.

Виконано сегментацію об’єктів за допомогою моделі SAM, що забезпечує виділення ключових характеристик зображення (доріг, будівель, ландшафтів тощо). Модель була натренована на тренувальному наборі даних із додаванням масок до зображень, які згодом використовуються для класифікації. Сегментовані дані створили основу для подальшого етапу класифікації, що підвищує точність роботи моделі завдяки усуненню зайвих елементів із фону.

Розроблено модель на основі DenseNet121 із врахуванням попередньо підготовлених даних із сегментаційними масками. Оптимізовано набір даних, видалено класи з недостатньою кількістю прикладів для покращення навчання. Проведено підготовку, налаштування та навчання моделі з використанням TensorFlow/Keras, що дозволило досягти точності на тренувальних даних 84.36% і суттєвого покращення в порівнянні з початковим станом.

Використано метод прив’язки зображень до географічних полігонів для визначення їхнього розташування. Розроблено модель на основі ResNet50 із додатковими шарами для задачі класифікації за географічними мітками. Проведено навчання моделі на розширеному наборі даних із географічними

координатами та полігонами, що забезпечує визначення геолокації об’єктів на зображеннях.

Розроблений багатоступеневий підхід із використанням сучасних моделей і технологій (SAM, DenseNet121, ResNet50) забезпечує ефективне вирішення задачі автоматизованої геолокації об’єктів. Досягнуті результати демонструють перспективність запропонованого методу для практичного використання в задачах геопросторової розвідки.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ ВИКОРИСТАННЯ

4.1 Аналіз результатів навчання моделей

4.1.1 Результати сегментації об’єктів

Модель SAM є відмінним інструментом для сегментації об’єктів, що забезпечує автоматичне виділення характерних ознак на зображеннях. Використання SAM base дозволило вирішити задачу виділення ключових елементів на зображеннях, таких як дороги, будівлі, лісові масиви та інші ландшафтні особливості, які містять геопросторову інформацію.

Модель продемонструвала високу якість сегментації, особливо для великих і чітко виражених об’єктів. Похибки спостерігалися у випадках, коли об’єкти були частково приховані або накладалися один на одного. Однак, навіть у складних сценах SAM base вдало визначав межі об’єктів завдяки аналізу піксельних характеристик.

Навчання моделі та створення масок для великого набору даних було тривалим процесом. Навіть для "легкої" версії SAM base знадобився значний час для обробки всіх зображень. Це свідчить про високу ресурсомісткість моделі, яка, однак, компенсується якістю отриманих результатів.

Для оцінки якості застосування SAM для задачі сегментації можна навести графік відсоткового покриття площі зображення кожною сегментованою маскою (рис. 4.1)

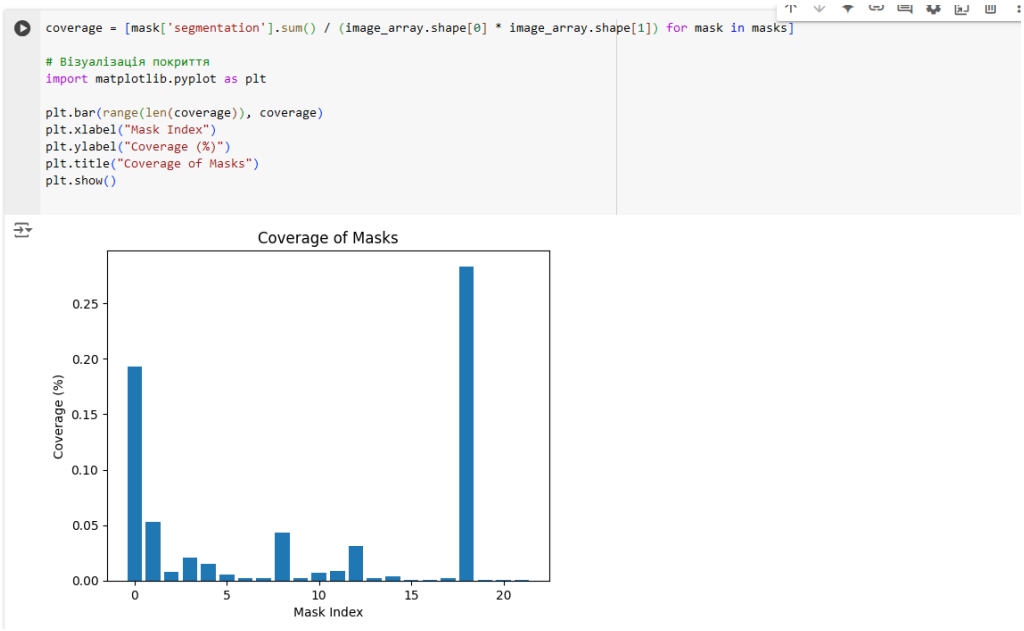

Рисунок 4.1 — Фрагмент коду та графік візуалізації покриття

Кожна мітка на осі X представляє індекс маски, яка була згенерована для даного зображення. Наприклад, 0, 1, 2, ... – це номери масок, що відповідають сегментам на зображенні. Вісь Y (coverage) показує відсоток площі зображення, покритий кожною маскою. Coverage обчислюється за формулою: кількість пікселів, що належить масці поділена на загальну кількість пікселів у зображенні.

Використовуючи цей графік можна обирати більш релевантні маски – є змога відкидати маски з дуже низьким покриттям, якщо вони не є важливими для задачі.

Маски з великим покриттям можуть представляти основні об'єкти на зображенні (наприклад, будівлі, дороги, великі предмети), в той час як маски з малим покриттям можуть відповідати менш значущим деталям, шуму або невеликим об'єктам (наприклад, дерева, тіні, дрібні предмети).

Сегментація звузила область пошуку, зменшивши кількість варіантів для географічної прив’язки зображень. Маски дозволили виділити ключові

елементи, які є маркерами для визначення геолокації (наприклад, дороги як орієнтири для місцевості).

4.1.2 Результат класифікації об’єктів

Другим етапом загальної задачі розпізнавання об’єктів була класифікація. Для вирішення цієї задачі було обрано модель DenseNet121 показала хороші результати класифікації на тренувальних і тестових наборах даних. Оптимізація набору даних та параметрів моделі суттєво вплинула на точність, проте залишилися виклики, пов’язані з недостатньою кількістю даних для окремих класів та перехресною плутаниною через схожість об'єктів. Подальше вдосконалення можливе шляхом балансування даних, інтеграції нових моделей та вдосконалення процесу навчання.

Для класів із великою кількістю даних (наприклад, країни з більш ніж 500 прикладів) точність перевищувала 85% (рис. 4.2). Класи з меншою кількістю даних (80–150 прикладів) мали значно нижчу точність, у деяких випадках навіть менше 50%. Деякі класи демонстрували високий рівень перехресної плутанини, особливо якщо їхні особливості (ландшафт, архітектура) були схожими між собою.

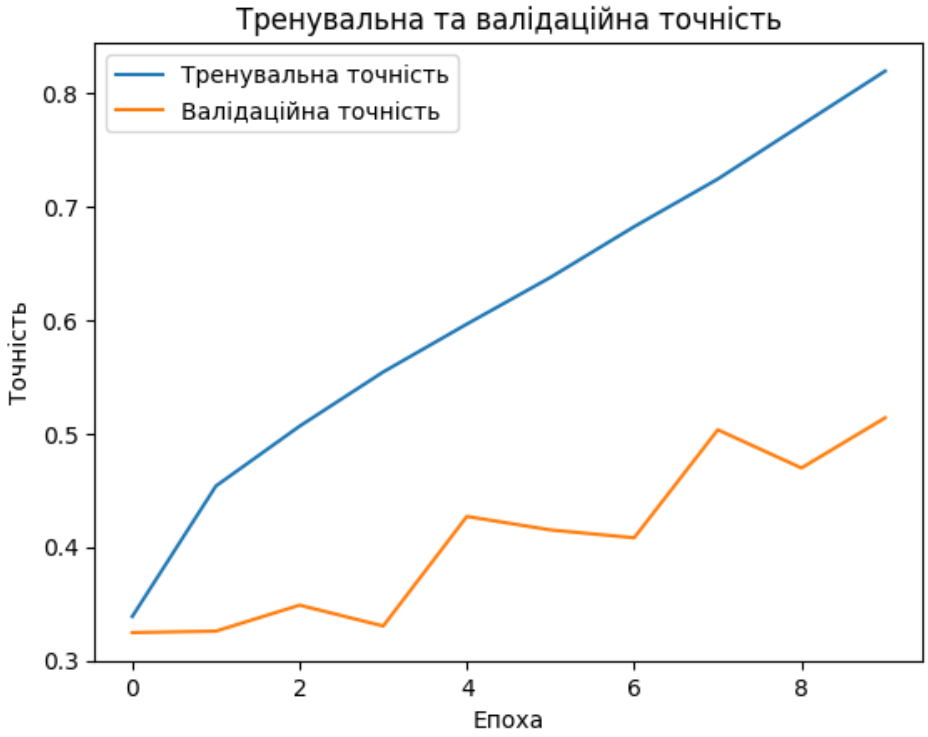

Рисунок 4.2 — Тренувальна та валідаційна точність навчання моделі класифікації

Після оптимізації даних точність на тренувальному наборі досягла 84.36%, а втрати становили 0.50 (рис. 4.3). Це свідчить про хороший рівень навчання без перенавчання моделі. Тестування моделі: точність на тестовому наборі склала 72%, що свідчить про хорошу генералізацію моделі. Втрати на тестових даних становили 0.98, що вказує на можливості для подальшого вдосконалення, особливо для класів із недостатньою кількістю прикладів. Висока різниця в точності між класами із великою кількістю даних і класами із малою кількістю прикладів свідчить про необхідність подальшого балансування набору даних.

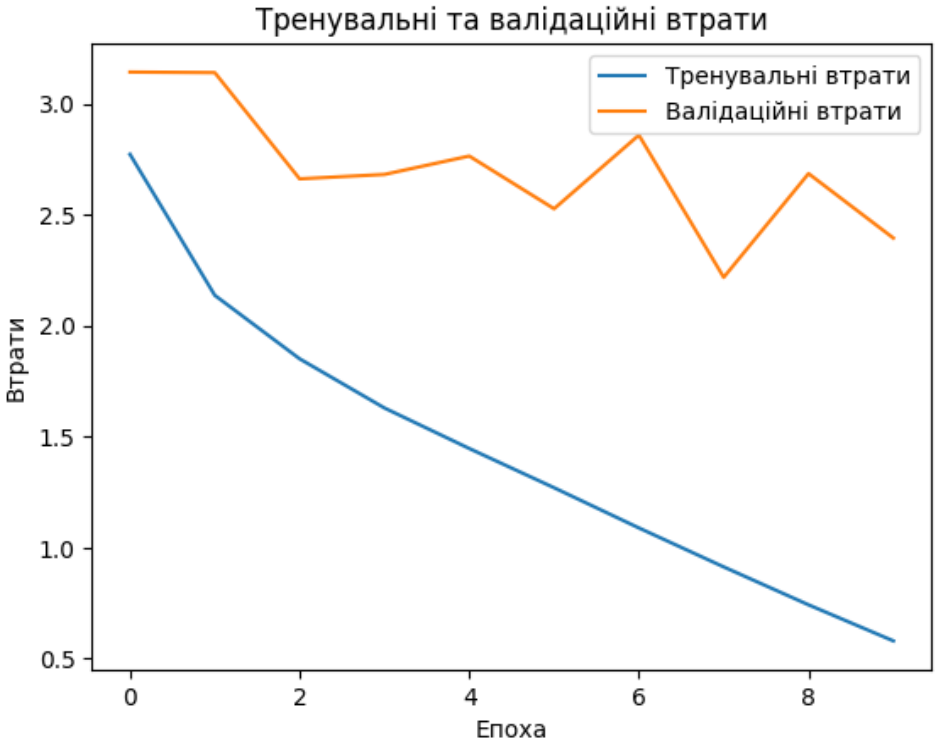

Рисунок 4.3 — Тренувальні та валідаційні втрати моделі класифікації

Модель дійсно в більшості випадків при завантаженні зображення з тестового набору вгадувала клас (рис. 4.4). Проте при завантаженні зображення зробленого особисто (одна з будівель Києва) модель не показувала точну класифікацію. Це можна пояснити тим, що кількість зображень все ж недостатня для максимально якісного використання моделі в режимі реального часу. Проте при знаходженні більш якісного та детального набору даних, та постійному навчанні моделі, ці результати можна в рази покращити, адже свою перспективність модель довела продемостованим великим відсотком точності на тестових та навчальних зображеннях.

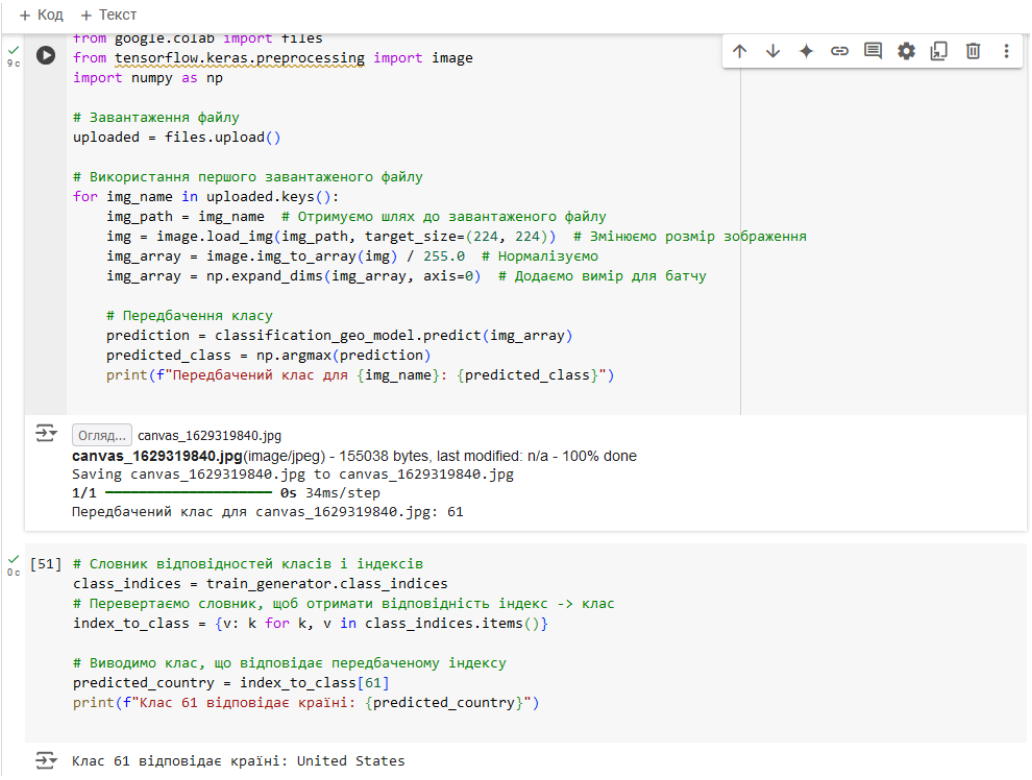

Рисунок 4.4 — Приклад роботи моделі класифікації

Модель DenseNet121 показала хороші результати класифікації на тренувальних і тестових наборах даних. Оптимізація набору даних та параметрів моделі суттєво вплинула на точність, проте залишилися виклики, пов’язані з недостатньою кількістю даних для окремих класів та перехресною плутаниною через схожість об'єктів. Подальше вдосконалення можливе шляхом балансування даних, інтеграції нових моделей та вдосконалення процесу навчання.

4.1.3 Результати геолокації об’єктів

Фінальним етапом задачі розпізнавання місцезнаходження ворожих об’єктів на зображення була геолокація. Модель ResNet50 із додатковими шарами успішно виконала задачу визначення геолокації на основі прив’язки зображень до географічних полігонів. Точність моделі на тренувальному наборі даних склала 72%, що свідчить про її здатність ідентифікувати ключові ознаки, пов'язані з географічним розташуванням. Однак точність на тестових

даних виявилася значно нижчою — 36.66%, що свідчить про проблеми із генералізацією (рис. 4.5). Прив’язка зображень до географічних полігонів допомогла зменшити область пошуку координат, але її ефективність залежала від якості полігонів та точності координат. Для вдосконалення моделі необхідно збільшити кількість даних, покращити їх різноманітність та розглянути більш гнучкі архітектури. Оптимізація параметрів навчання та розширення набору даних можуть суттєво підвищити результати.

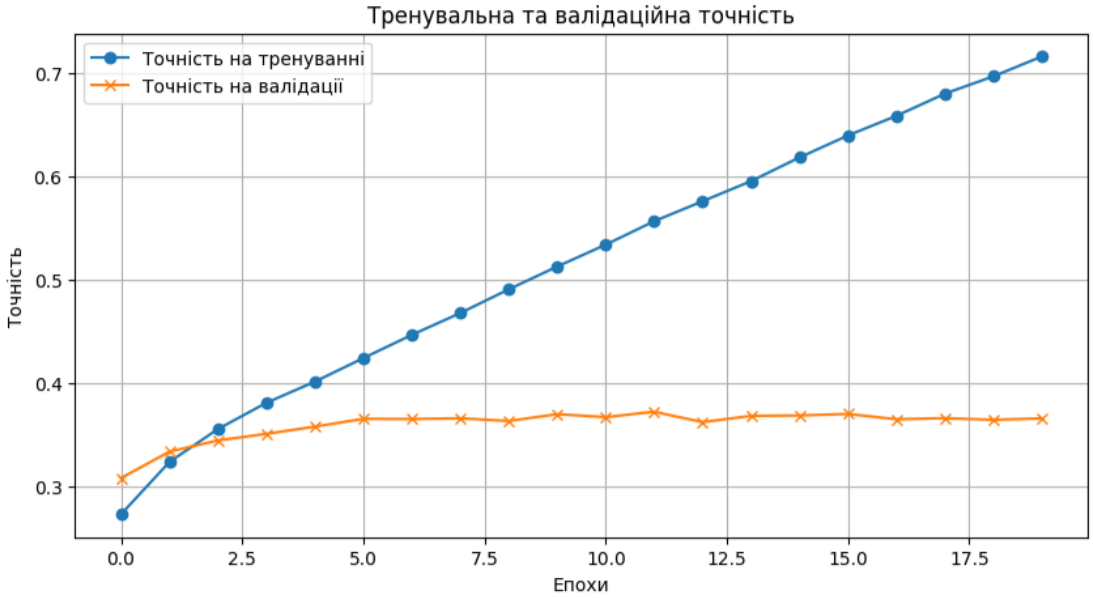

Рисунок 4.5 — Тренувальна та валідаційна точність навчання моделі

Модель добре навчилася ідентифікувати закономірності на основі великої кількості прикладів із тренувального набору. Низькі втрати (0.98) свідчать про хорошу якість навчання. Точність на тестовому наборі значно нижча, що може бути пов’язано з недостатньою різноманітністю даних або переважанням певних класів у тренувальному наборі. Втрати на тестових даних склали 2.82, що вказує на необхідність кращого узгодження моделі з невідомими даними (рис. 4.6).

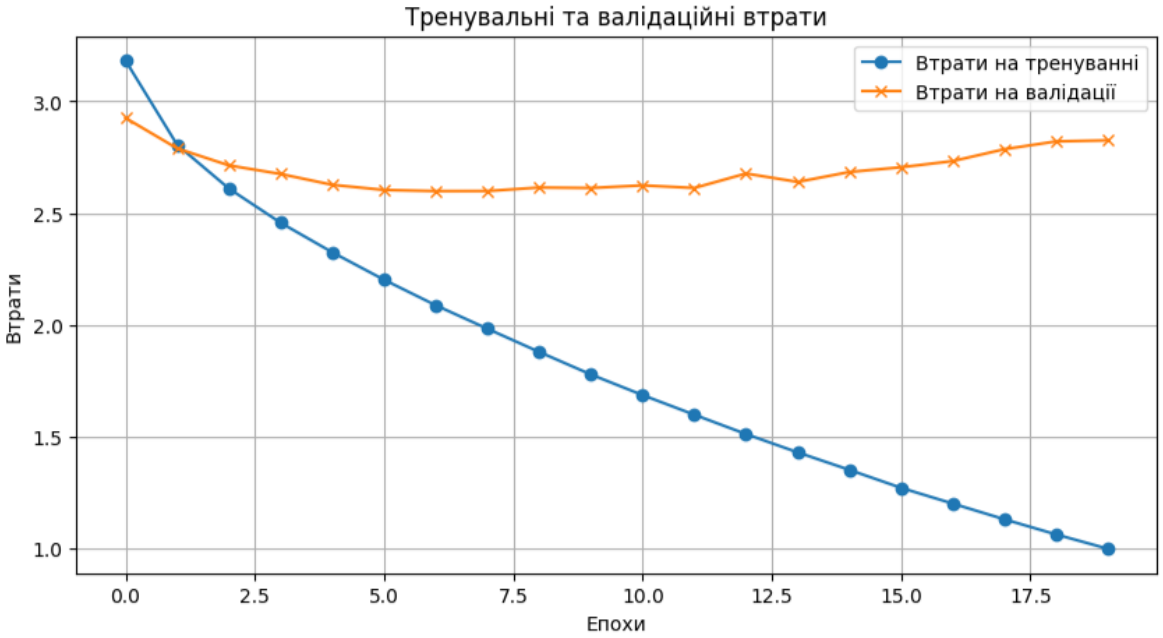

Рисунок 4.6 — Тренувальні та валідаційні втрати навчання моделі

Прив’язка зображень до географічних полігонів допомогла зменшити кількість можливих варіантів геолокації для кожного зображення. Процес прив’язки: полігони були створені на основі географічних координат і містили точну інформацію про регіональні межі. Просторове об’єднання дозволило ефективно визначити, до якого полігону належить кожне зображення. Прив’язка значно знизила обчислювальну складність задачі. Основні помилки виникали в районах, де полігони були занадто великими або неправильно адаптованими до місцевих особливостей.

Модель дійсно в 2 з 5 випадків на реальних зображеннях вгадала місцезнаходження об’єктів (в одному випадку з точністю до міста, в іншому з точністю до країни). Для підвищення точності моделі слід працювати над зменшенням розмірів полігонів для підвищення точності геоприв’язки. Використання додаткових геопросторових даних (наприклад, інформації про дороги, річки, ліси) для деталізації меж полігонів (рис. 4.7).

```
[ ]  # Оцінка моделі
     model.eval()  # Переводимо модель у режим оцінки
     predicted_coords = []
     true_coords = []

     with torch.no_grad():
         for images, labels in test_loader:
             images = images.to(device)
             outputs = model(images)  # Передбачення
             predicted_coords.append(outputs.cpu().numpy())  # Перетворення на numpy
             true_coords.append(labels.numpy())

     # Перетворюємо списки на масиви
     predicted_coords = np.vstack(predicted_coords)
     true_coords = np.vstack(true_coords)

     print(f"Передбачені координати: {predicted_coords[:5]}")
     print(f"Справжні координати: {true_coords[:5]}")
```

```
Передбачені координати: [[ -24.253335  -48.92751 ]
 [  47.60432  -122.34188 ]
 [  -28.10874   -79.90498 ]
 [  -20.39924, 123.66045 ]
 [  57.040974    8.495188]]
Справжні координати: [[ -36.574165  -54.45392 ]
 [  47.60857  -122.34025 ]
 [  43.26605   -79.90498 ]
 [  42.267384  -71.80759 ]
 [  56.49154, 8.85681]]
```

Рисунок 4.7 — Прогнозування моделі місцезнаходження об’єкту на зображенні

Модель ResNet50 із додатковими шарами продемонструвала хороші результати на тренувальному наборі, але потребує вдосконалення для покращення генералізації на тестових даних. Прив’язка зображень до географічних полігонів стала ефективним інструментом, але потребує деталізації меж і збільшення різноманітності даних. Вдосконалення моделі та оптимізація процесу прив’язки можуть значно покращити точність геолокації.

4.2 Перспективи використання розробленої системи

Геолокація в умовах війни є критично важливим процесом для розвідки, планування військових операцій та моніторингу дій противника. Розроблена система дозволяє автоматизувати значну частину цього процесу, зменшуючи вплив людського фактора та підвищуючи точність. Система здатна швидко

обробляти великий обсяг зображень (супутникових, аерофотознімків, кадрів із дронів), сегментуючи об'єкти та визначаючи їхні географічні координати. Це знижує навантаження на аналітиків та підвищує швидкість отримання інформації. Використання сегментації забезпечує виділення ключових об'єктів (наприклад, техніки, укріплень, шляхів постачання), а класифікація дозволяє визначати регіон або район, в якому вони знаходяться. Геолокація уточнює координати з високою точністю.

Однак, для більш ефективної роботи моделей необхідно отримати набір даних із зображеннями, що включає територію України та рф, який має містити:

- Супутникові знімки з високою роздільною здатністю, які охоплюють різні види ландшафтів (міста, села, поля, ліси, річки);
- Кадри з дронів із розміченими об'єктами, наприклад, технікою, укріпленнями чи інженерними спорудами;
- Фото з соціальних мереж або камер спостереження, які дозволять моделі вчитися працювати з різними кутами огляду та якістю зображення;
- Дані географічних полігонів із деталізованими межами районів і регіонів, зокрема, у форматі, сумісному з GeoPandas або подібними бібліотеками.

Висновки до четвертого розділу

У четвертому розділі було проведено детальний аналіз роботи та ефективності розробленої системи геолокації на основі згорткових нейронних мереж, зосереджений на трьох ключових етапах: сегментація, класифікація та геолокація.

Використання моделі SAM base продемонструвало високу якість сегментаційних масок, особливо для великих і чітко визначених об'єктів. Сегментація дозволила підготувати структуровані дані для наступних етапів,

значно зменшуючи шум та виділяючи ключові об'єкти (дороги, будівлі, ландшафтні особливості). Модель виявила свої обмеження у випадках низької якості зображень або неоднорідних текстур, що вимагає подальшого вдосконалення.

Модель DenseNet121 досягла точності 84.36% на тренувальних даних, але точність на тестових даних була нижчою (72%), що свідчить про необхідність покращення генералізації. Оптимізація набору даних через видалення нерелевантних класів та аугментацію дозволила значно підвищити точність класифікації. Основними проблемами стали перехресна плутанина між подібними класами (наприклад, ландшафти різних країн) і недостатня кількість даних для деяких класів.

Модель ResNet50 із додатковими шарами показала точність 72% на тренувальних даних, але точність на тестових знизилась до 36.66%, що потребує оптимізації. Прив’язка зображень до географічних полігонів допомогла звузити область пошуку координат, але точність залежала від якості полігонів і координат. Основні проблеми виникли через недостатню деталізацію полігонів та низьку різноманітність набору даних.

Загалом, у розділі було виконано повний цикл аналізу результатів роботи розробленої системи, від оцінки кожного етапу до виявлення проблем і формулювання рекомендацій для покращення. Продемонстровано ефективність поетапного підходу та підкреслено перспективи використання системи для військових і цивільних задач геолокації.

ВИСНОВКИ

В магістерській роботі вирішене актуальне науково-практичне завдання розробки та впровадження методів розпізнавання місцезнаходження ворожих об’єктів на знімках із використанням засобів штучного інтелекту. У результаті виконаного дослідження отримано наступні наукові та практичні результати:

1. Проведено аналіз сучасних методів розпізнавання об’єктів на зображеннях з використанням згорткових нейронних мереж. Вивчено переваги й недоліки існуючих моделей для задач сегментації, класифікації та геолокації, що дозволило обґрунтовано обрати архітектури SAM, DenseNet121 та ResNet50 для реалізації системи.
2. Розроблено методологію поетапного підходу до розв’язання задачі геолокації, яка включає три ключові етапи:

- Сегментація (виділення ключових об’єктів на зображеннях);
- Класифікація (ідентифікація регіону або країни);
- Геолокація (визначення точних координат на основі класифікації та сегментації).

3. Реалізовано сегментацію об’єктів за допомогою моделі SAM (Segment Anything Model), яка дозволила створювати точні сегментаційні маски для виділення ключових елементів зображень (доріг, будівель, ландшафтів). Результати показали високу точність моделі навіть за умов значної варіативності в даних.
4. Виконано класифікацію об’єктів за допомогою DenseNet121, що дало змогу ідентифікувати регіон або країну на основі візуальних ознак зображення. Оптимізація набору даних та налаштувань моделі дозволила досягти точності 84.36% на тренувальному наборі.
5. Розроблено модель для задачі геолокації на основі ResNet50 із додатковими шарами, яка прив’язує зображення до географічних полігонів. Це дозволило значно звузити область пошуку координат для

кожного об’єкта. Результати навчання моделі показали точність 72% на тренувальному наборі даних.

6. Оцінено ефективність поетапного підходу: продемонстровано, як результати сегментації покращують точність класифікації. Встановлено, що комбінування даних із сегментації та класифікації знижує похибки геолокації.
7. Визначено перспективи використання розробленої системи в оборонній сфері. Розробка може застосовуватись для оптимізації процесів геолокації в умовах бойових дій, автоматизації аналізу супутникових і аерофотознімків, підтримки роботи OSINT-аналітиків. Підкреслено необхідність подальшого збору якісних зображень для навчання системи та вдосконалення моделей.

Результати роботи довели, що поетапний підхід із використанням сучасних архітектур нейронних мереж є ефективним для вирішення задачі геолокації. Розроблена система може бути використана як основа для подальшого вдосконалення алгоритмів розпізнавання та інтеграції у військові й цивільні геоінформаційні системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. GEOINT - Geospatial Intelligence [Електронний ресурс] — Режим доступу до ресурсу: https://www.heavy.ai/technical-glossary/geoint.

2. Geospatial intelligence services [Електронний ресурс] — Режим доступу до ресурсу: https://www.defence.gov.au/defence-activities/products-services/geospatial-intelligence-services.

3. GEOINT Lessons Being Learned from the Russian-Ukrainian War [Електронний ресурс] — Режим доступу до ресурсу: https://usgif.org/geoint-lessons-being-learned-from-the-russian-ukrainian-war/.

4. Integrating Earth observation IMINT with OSINT data to create added-value multisource intelligence information: A case study of the Ukraine–Russia war [Електронний ресурс] – Режим доступу до ресурсу: https://securityanddefence.pl/pdf-170901-96088?filename=Integrating%20Earth.pdf.

5. Exploring links between EO Satellites, social media and crowdsourcing information against terrorism and organized crime.: https://www.researchgate.net/publication/353604960_Exploring_links_between_EO_Satellites_social_media_and_crowdsourcing_information_against_terrorism_and_organized_crime.

6. OPEN SOURCE INTELLIGENCE (OSINT) AS AN ELEMENT OF MILITARY RECON [Електронний ресурс] — Режим доступу до ресурсу: https://securityanddefence.pl/pdf-103337-36164?filename=Open%20source%20intelligence.pdf.

7. Situational Awareness: Techniques, Challenges, and Prospects [Електронний ресурс] — Режим доступу до ресурсу: https://www.mdpi.com/2673-2688/3/1/5.

8. Is AI Getting Out of Hand With the Amounts of Data It Produces and Are We Keeping Up in America? Integrating AI into Open-Source Intelligence

(OSINT): Managing Data Overload through Convergence with SIGINT, HUMINT, and Beyond [Електронний ресурс] — Режим доступу до ресурсу: https://www.researchgate.net/publication/385778676_Is_AI_Getting_Out_of_Hand_With_the_Amounts_of_Data_It_Produces_and_Are_We_Keeping_Up_in_America_Integrating_AI_into_Open-Source_Intelligence_OSINT_Managing_Data_Overload_through_Convergence_with_SIGIN.

9. Fully Convolutional Networks for Semantic Segmentation [Електронний ресурс] — Режим доступу до ресурсу: https://arxiv.org/pdf/1411.4038.

10. U-Net Architecture Explained [Електронний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/u-net-architecture-explained/.

11. Segnet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. [Електронний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/segnet-a-deep-convolutional-encoder-decoder-architecture-for-image-segmentation/.

12. Understanding DeepLabV3 in Image Segmentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.ikomia.ai/blog/understanding-deeplabv3-image-segmentation.

13. Segnet [Електронний ресурс] — Режим доступу до ресурсу: https://arxiv.org/pdf/1511.00561.

14. Neural Networks for Classification: A Survey [Електронний ресурс] — Режим доступу до ресурсу: https://www.researchgate.net/publication/3421357_Neural_Networks_for_Classification_A_Survey.

15. Residual Networks (ResNet) – Deep Learning [Електронний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/.

16. Boost Your Image Classification Model with pretrained VGG-16 [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/geekculture/boost-your-image-classification-model-with-pretrained-vgg-16-ec185f763104.

17. Image classification: ResNet vs EfficientNet vs EfficientNet_v2 vs Compact Convolutional Transformers [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/@enrico.randellini/image-classification-resnet-vs-efficientnet-vs-efficientnet-v2-vs-compact-convolutional-c205838bbf49.

18. Brain Tumor Classification (MRI) [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri.

19. AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE [Електронний ресурс] — Режим доступу до ресурсу: https://arxiv.org/pdf/2010.11929.

20. GeoLocation - Geoguessr Images (50K) [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/datasets/ubitquitin/geolocation-geoguessr-images-50k/data.

21. Geo Guesser [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/code/crypticsy/geo-guesser.

22. Large Dataset of Geotagged Images [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/datasets/habedi/large-dataset-of-geotagged-images.

23. YFCC100M [Електронний ресурс] — Режим доступу до ресурсу: https://paperswithcode.com/dataset/yfcc100m.

24. Key Features of the Segment Anything Model (SAM) [Електронний ресурс] — Режим доступу до ресурсу: https://docs.ultralytics.com/models/sam/#key-features-of-the-segment-anything-model-sam.

25. Medival_clip_cem_smooth [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/code/nemishextra/medival-clip-cem-smooth.

26. ClusteredCellsFlickr [Електронний ресурс] — Режим доступу до ресурсу: https://www.kaggle.com/datasets/simonedigregorio/clusteredcellsflickr/data.

ДОДАТОК А

Лістинг А.1 — Модуль ViT для сегментації

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

!unzip "/content/drive/MyDrive/archive(2).zip" -d /content/dataset
import os

# Вивести всі файли в папці
for root, dirs, files in os.walk('/content/dataset'):
    for name in files:
        print(os.path.join(root, name))

# Підрахунок кількості файлів у папці
image_count = sum(len(files) for _, _, files in os.walk('/content/dataset'))
print(f"Кількість зображень у датасеті: {image_count}")

import matplotlib.pyplot as plt
from PIL import Image

# Вивести перші 5 зображень
image_paths = [os.path.join(root, name) for root, _, files in
os.walk('/content/dataset') for name in files]

# Показати зображення
for i, img_path in enumerate(image_paths[:5]):
    img = Image.open(img_path)
    plt.figure()
```

```
    plt.title(f"Зображення {i+1}")
    plt.imshow(img)
    plt.axis('off')
    plt.show()

import pandas as pd

# Створення DataFrame
df = pd.DataFrame(image_paths, columns=["file_path"])
print(df.head())

# Завантаження CSV
coords_path = '/content/dataset/dataset/coords.csv'  # Оновлено шлях
coords_df = pd.read_csv(coords_path)
print(coords_df.head())  # Перевірка перших рядків

# Перевірка кількості записів
print(f"Кількість зображень: {len(df)}")
print(f"Кількість записів у coords.csv: {len(coords_df)}")

# Перевірка, чи всі зображення є у файлі coords.csv
missing_files = [img for img in df["file_path"] if img.split('/')[-1] not in coords_df['file_name'].values]
print(f"Відсутні файли: {len(missing_files)}")

import os

# Шлях до директорії із зображеннями та файлом coords.csv
```

```
dataset_path = '/content/dataset/dataset'

# Фільтруємо лише зображення .png
image_files = [f for f in os.listdir(dataset_path) if f.endswith('.png')]
print(f"Кількість зображень: {len(image_files)}")  # Має показати 10000

import pandas as pd

# Шлях до файлу coords.csv
coords_path = '/content/dataset/dataset/coords.csv'

# Завантажуємо CSV
coords_df = pd.read_csv(coords_path)

# Порівнюємо
csv_files = set(coords_df['file_name'])  # Імена файлів із CSV
image_files_set = set(image_files)  # Імена файлів із директорії

missing_in_csv = image_files_set - csv_files  # Зображення, яких немає в CSV
missing_in_images = csv_files - image_files_set  # Записи в CSV, яких немає серед зображень

print(f"Відсутні у coords.csv: {missing_in_csv}")
print(f"Відсутні серед зображень: {missing_in_images}")

import pandas as pd
```

```
# Завантаження файлу coords.csv
coords_path = '/content/dataset/dataset/coords.csv'
coords_df = pd.read_csv(coords_path, header=None, names=['latitude', 'longitude'])

# Генерація стовпця file_name (імена файлів повинні відповідати координатам)
coords_df['file_name'] = [f"{i}.png" for i in range(len(coords_df))]

# Перегляд нового DataFrame
print(coords_df.head())

# Збереження оновленого файлу coords.csv
coords_df.to_csv(coords_path, index=False)
print("Файл coords.csv оновлено!")

import os

# Шлях до директорії з зображеннями
dataset_path = '/content/dataset/dataset'

# Фільтрація лише зображень
image_files = [f for f in os.listdir(dataset_path) if f.endswith('.png')]

print(f"Кількість зображень: {len(image_files)}")
print(f"Кількість записів у coords.csv: {len(coords_df)}")

# Перевірка відповідності
```

```
if len(image_files) == len(coords_df):
    print("Кількість зображень і записів у coords.csv збігаються.")
else:
    print("Виявлено невідповідність!")

for i in range(5):  # перевірка перших 5 записів
    file_name = coords_df.loc[i, 'file_name']
    image_path = os.path.join(dataset_path, file_name)

    img = Image.open(image_path)
    lat, lon = coords_df.loc[i, 'latitude'], coords_df.loc[i, 'longitude']

    print(f"Файл: {file_name}, Координати: {lat}, {lon}")
    plt.imshow(img)
    plt.title(f"Координати: {lat}, {lon}")
    plt.axis('off')
    plt.show()

from sklearn.model_selection import train_test_split

# Розділення координат і файлів
X = coords_df['file_name']
y = coords_df[['latitude', 'longitude']]

# Розділення на тренувальний, валідаційний і тестовий набори
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Тренувальний набір: {len(X_train)} записів")
print(f"Валідаційний набір: {len(X_val)} записів")
print(f"Тестовий набір: {len(X_test)} записів")

import pandas as pd
import os
from sklearn.model_selection import train_test_split

import os
import shutil

# Шляхи
current_path = '/content/dataset/dataset/coords.csv'  # Поточне місце
new_path = '/content/dataset/coords.csv'  # Місце, куди хочемо перенести

# Перевіряємо, чи файл існує
if os.path.exists(current_path):
    print("Переміщуємо файл coords.csv...")
    shutil.move(current_path, new_path)  # Переміщуємо файл
    print(f"Файл переміщено до {new_path}")
else:
    print("Файл coords.csv не знайдено в папці.")

import pandas as pd
```

```
# Завантаження CSV
coords_path = '/content/dataset/coords.csv'
coords_df = pd.read_csv(coords_path, header=None, names=['latitude', 'longitude'])

# Перевірка перших і останніх записів
print("Перші 10 рядків:")
print(coords_df.head(10))
print("\nОстанні 10 рядків:")
print(coords_df.tail(10))
# Виведення всього списку (не рекомендується для великих файлів)
# print(coords_df)
import pandas as pd
coords_path = '/content/dataset/coords.csv'
# Завантаження CSV з ігноруванням першого рядка як даних
coords_df = pd.read_csv(coords_path, header=0, names=['latitude', 'longitude', 'file_name'])
print(coords_df.head())

import os
# Отримати список усіх файлів у папці
image_files = [f for f in os.listdir('/content/dataset/dataset') if f.endswith('.png')]
# Перевірка на відповідність
print(f"Кількість зображень: {len(image_files)}")
print(f"Кількість записів у coords.csv: {len(coords_df)}")
assert len(image_files) == len(coords_df), "Кількість зображень і записів у coords.csv має збігатися!"
```

```
from sklearn.model_selection import train_test_split
# Розділення на train, validation і test
X_train, X_temp, y_train, y_temp = train_test_split(
    coords_df['file_name'], coords_df[['latitude', 'longitude']], test_size=0.2,
random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
   X_temp, y_temp, test_size=0.5, random_state=42
)
print(f"Тренувальний набір: {len(X_train)} зображень")
print(f"Валідаційний набір: {len(X_val)} зображень")
print(f"Тестовий набір: {len(X_test)} зображень")
from sklearn.model_selection import train_test_split
import pandas as pd

# Завантаження coords.csv
coords_path = '/content/dataset/coords.csv'
coords_df = pd.read_csv(coords_path)
# Розділення на train, validation і test
X_train, X_temp, y_train, y_temp = train_test_split(
    coords_df['file_name'], coords_df[['latitude', 'longitude']], test_size=0.2,
random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
   X_temp, y_temp, test_size=0.5, random_state=42
)
print(f"Тренувальний набір: {len(X_train)} зображень")
```

```
print(f"Валідаційний набір: {len(X_val)} зображень")
print(f"Тестовий набір: {len(X_test)} зображень")
from PIL import Image
import numpy as np
import os

# Функція для завантаження та нормалізації зображень
def load_images(file_names, dataset_path, img_size=(224, 224)):
    images = []
    for file_name in file_names:
        img_path = os.path.join(dataset_path, file_name)
        img = Image.open(img_path).resize(img_size)
        images.append(np.array(img) / 255.0)  # Нормалізація до [0, 1]
    return np.array(images)
# Вкажіть шлях до датасету
dataset_path = '/content/dataset/dataset'
# Завантаження зображень
train_images = load_images(X_train, dataset_path)
val_images = load_images(X_val, dataset_path)
test_images = load_images(X_test, dataset_path)
print(f"Розмір train_images: {train_images.shape}")
print(f"Розмір val_images: {val_images.shape}")
print(f"Розмір test_images: {test_images.shape}")
import matplotlib.pyplot as plt
# Візуалізація декількох зображень із координатами
plt.figure(figsize=(10, 10))
for i in range(9):  # Вивести 9 зображень
    plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(train_images[i])
        plt.title(f"Lat: {y_train.iloc[i, 0]:.2f}, Lon: {y_train.iloc[i, 1]:.2f}")
        plt.axis('off')
    plt.show()
    import cv2
    from matplotlib import pyplot as plt
    from segment_anything import SamAutomaticMaskGenerator

    # Ініціалізація генератора масок
    mask_generator = SamAutomaticMaskGenerator(sam)

    # Завантаження зображення
    img_path = "/content/dataset/dataset/106.png" # Вкажіть шлях до зображення
    image = cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB)

    # Генерація масок
    masks = mask_generator.generate(image)

    # Функція для відображення масок
    def show_anns(anns, axes=None):
        """
        Відображення згенерованих масок
        """
        if len(anns) == 0:
            return
        if axes:
            ax = axes
```

```
    else:
        ax = plt.gca()
        ax.set_autoscale_on(False)
    sorted_anns = sorted(anns, key=(lambda x: x['area']), reverse=True)
    polygons = []
    color = []

    for ann in sorted_anns:
        m = ann['segmentation']
        img = np.ones((m.shape[0], m.shape[1], 3))
        color_mask = np.random.random((1, 3)).tolist()[0]
        for i in range(3):
            img[:,:,i] = color_mask[i]
        ax.imshow(np.dstack((img, m*0.5)))

# Відображення оригінального зображення та масок
_, axes = plt.subplots(1, 3, figsize=(20, 10))
axes[0].imshow(image)
axes[0].set_title("Оригінальне зображення")
axes[0].axis("off")

show_anns(masks, axes[1])
axes[1].set_title("Сегментаційна маска")
axes[1].axis("off")

axes[2].imshow(image)
show_anns(masks, axes[2])
axes[2].set_title("Зображення + маска")
```

```
axes[2].axis("off")

plt.show()
```

ДОДАТОК Б

Лістинг Б.1 — Модуль для класифікації

```
import os
import pathlib
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications import DenseNet121, ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

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

import os
for root, dirs, files in os.walk('/content/drive/My Drive'):
    for file in files:
        if file.endswith(".zip"):
            print(os.path.join(root, file))

!unzip -q "/content/drive/My Drive/archive(1).zip" -d "/content/geo_dataset"
```

```
import os
import shutil

# Шлях до папки compressed_dataset
compressed_dataset_path = os.path.join(base_dir, 'compressed_dataset')
# Отримання списку підпапок у compressed_dataset
subfolders = os.listdir(compressed_dataset_path)
# Переміщення кожної підпапки з compressed_dataset до кореневого каталогу
for folder in subfolders:
    folder_path = os.path.join(compressed_dataset_path, folder)
    shutil.move(folder_path, base_dir)
# Після переміщення видаляємо порожню папку compressed_dataset
os.rmdir(compressed_dataset_path)
print("Країни переміщено. Оновлений список папок у кореневій папці:")
print(os.listdir(base_dir))

import os
import pathlib
import pandas as pd
from PIL import Image

# Шлях до папки з даними
geo_data_dir = pathlib.Path('/content/geo_dataset')

# Функція для збору метаданих
def extract_metadata_from_folder(path):
```

```
        metadata = []
        for file_path in path.glob('*'):
            # Отримання ширини, висоти та розміру файлу
            width, height = Image.open(file_path).size
            metadata.append({
                'country': path.name,  # Назва країни (назва папки)
                'image_name': file_path.name,  # Назва файлу
                'width': width,  # Ширина зображення
                'height': height,  # Висота зображення
                'size': file_path.stat().st_size,  # Розмір файлу у байтах
                'path': str(file_path)  # Повний шлях до файлу
            })
        return metadata

    # Список всіх підпапок (країн)
    country_folders = [f for f in os.listdir(geo_data_dir) if
os.path.isdir(os.path.join(geo_data_dir, f))]
    # Ініціалізуємо порожній список для метаданих
    all_metadata = []
    # Ітеруємо по кожній країні та збираємо метадані
    for country_name in country_folders:
        folder_path = pathlib.Path(os.path.join(geo_data_dir, country_name))
        metadata = extract_metadata_from_folder(folder_path)
        all_metadata.extend(metadata)
    # Створюємо DataFrame зі зібраних метаданих
    df_geo_data = pd.DataFrame(all_metadata)
```

```
    # Групування за країною для підрахунку кількості зображень у кожній країні
    df_data_distribution = (
        df_geo_data.groupby('country')['image_name']
        .count()
        .reset_index()
        .rename(columns={'image_name': 'frequency'})
    )
    # Переглядаємо перші рядки метаданих
    print("Перші 5 записів метаданих:")
    print(df_geo_data.head())
    print("\nРозподіл кількості зображень за країнами:")
    print(df_data_distribution.head())

    from sklearn.model_selection import train_test_split
    import shutil

    # Розділення датасету на тренувальний і тестовий набори
    train, test = train_test_split(df_geo_data, test_size=0.05, random_state=123)

    # Шляхи для збереження підготовлених наборів
    geo_train_dir = pathlib.Path('/content/geo_train_dataset')
    geo_test_dir = pathlib.Path('/content/geo_test_dataset')

    # Створення папок і копіювання зображень
    for pathname, dataset in [
        [geo_train_dir, train],
        [geo_test_dir, test]
```

```
    ]:
        # Створення директорії
        pathname.mkdir(parents=True, exist_ok=True)

        for country in df_data_distribution['country'].unique():
            # Створення папки для кожної країни
            geo_country_dir = pathname / country
            geo_country_dir.mkdir(parents=True, exist_ok=True)
            # Копіювання зображень у відповідну папку
            for picture_path in dataset[dataset['country'] == country]['path']:
                                    target_picture_path = geo_country_dir /
pathlib.Path(picture_path).name
                shutil.copy(picture_path, target_picture_path)
    print("Датасет розділено та переміщено.")

    # Параметри моделі
    img_width = 224  # Ширина зображення
    img_height = 224  # Висота зображення
    batch_size = 32  # Розмір батчу
    epochs = 10  # Кількість епох
    dropout_rate = 0.2  # Dropout rate

    import tensorflow as tf

    # Створення навчального та валідаційного наборів
    train_ds = tf.keras.utils.image_dataset_from_directory(
        geo_train_dir,
        validation_split=0.2,
```

```
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    geo_train_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
# Завантаження тестового набору
test_ds = tf.keras.utils.image_dataset_from_directory(
    geo_test_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
# Кількість класів
num_classes = len(train_ds.class_names)
print(f"Кількість класів: {num_classes}")

# Налаштування продуктивності
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Sequential

# Завантаження DenseNet121 з вагами ImageNet
densenet = DenseNet121(
    input_shape=(img_height, img_width, 3),
    include_top=False,
    weights="imagenet"
)
# Побудова моделі
model = Sequential([
    densenet,
    GlobalAveragePooling2D(),
    Dense(num_classes, activation="softmax")
])
model.summary()
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, LambdaCallback
import time
```

```
# Шлях для збереження найкращої моделі
checkpoint_path = "/content/best_model.keras"

# Шлях для збереження історії навчання
log_path = "/content/training_log.csv"

# Колбек для збереження найкращої моделі
checkpoint_cb = ModelCheckpoint(
    filepath=checkpoint_path,
    monitor="val_accuracy",  # Збереження найкращої моделі за валідуючою точністю
    save_best_only=True,
    verbose=1
)

# Колбек для запису історії навчання в CSV
csv_logger = CSVLogger(log_path, append=True)

# Колбек для відображення часу після кожної епохи
time_callback = LambdaCallback(
    on_epoch_begin=lambda epoch, logs: print(f"\n--- Епоха {epoch + 1} почалась ---"),
    on_epoch_end=lambda epoch, logs: print(f"--- Епоха {epoch + 1} завершилась ---"),
)

# Навчання моделі
start_time = time.time()
```

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[checkpoint_cb, csv_logger, time_callback]
)
end_time = time.time()

print(f"Загальний час навчання: {end_time - start_time:.2f} секунд")

# Оцінка на тестовому наборі
loss, accuracy = model.evaluate(test_ds)
print(f"Тестова точність: {accuracy:.2f}")

import matplotlib.pyplot as plt

# Завантаження історії з файлу (якщо потрібно)
# history_df = pd.read_csv(log_path)

# Побудова графіку точності
plt.plot(history.history['accuracy'], label='Тренувальна точність')
plt.plot(history.history['val_accuracy'], label='Валідаційна точність')
plt.xlabel('Епоха')
plt.ylabel('Точність')
plt.legend()
plt.title('Тренувальна та валідаційна точність')
plt.show()
```

```
# Побудова графіку втрат
plt.plot(history.history['loss'], label='Тренувальні втрати')
plt.plot(history.history['val_loss'], label='Валідаційні втрати')
plt.xlabel('Епоха')
plt.ylabel('Втрати')
plt.legend()
plt.title('Тренувальні та валідаційні втрати')
plt.show()
```

ДОДАТОК В

Лістинг В.1 — Модуль розпізнавання місцезнаходження

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

import os
import msgpack
import pandas as pd
import geopandas as gpd
from tqdm import tqdm

def read_shards(dataset_dir, n_shards=10):
    """
    Зчитує дані із зазначеної кількості шардів.

    Параметри:
    - dataset_dir: шлях до директорії з файлами-шардами.
    - n_shards: кількість шардів для обробки.

    Повертає:
    - GeoDataFrame із широтою, довготою, id, та shard.
    """
    data = []
    for i in range(n_shards):
        shard_path = os.path.join(dataset_dir, f"shard_{i}.msg")
        print(f"Processing {shard_path}...")
        with open(shard_path, "rb") as infile:
```

```
                    for record in tqdm(msgpack.Unpacker(infile, raw=False), total=30_000):
                        # Видаляємо зображення з запису
                        record.pop('image')
                        # Додаємо індекс шару
                        record['shard'] = i
                        data.append(record)

            # Створюємо DataFrame
            df = pd.DataFrame(data)
            df.columns = ['id', 'lat', 'lon', 'shard']

            # Конвертуємо у GeoDataFrame
            gdf = gpd.GeoDataFrame(df,
                                   geometry=gpd.points_from_xy(df.lon, df.lat),
                                   crs="EPSG:4326")
            return gdf

        def read_shards(dataset_dir, n_shards=10):
            # Отримуємо список доступних файлів і сортуємо їх
            shard_files = sorted([f for f in os.listdir(dataset_dir) if f.endswith('.msg')])

            data = []
            for i, shard_file in enumerate(shard_files[:n_shards]):  # Обробляємо тільки n_shards
                shard_path = os.path.join(dataset_dir, shard_file)
                print(f"Processing {shard_path}...")
                with open(shard_path, "rb") as infile:
```

```
                    for record in tqdm(msgpack.Unpacker(infile, raw=False), total=30_000):
                        # Видаляємо зображення з запису
                        record.pop('image')
                        # Додаємо індекс шару
                        record['shard'] = shard_file
                        data.append(record)

    # Створюємо DataFrame
    df = pd.DataFrame(data)
    df.columns = ['id', 'lat', 'lon', 'shard']

    # Конвертуємо у GeoDataFrame
    gdf = gpd.GeoDataFrame(df,
                           geometry=gpd.points_from_xy(df.lon, df.lat),
                           crs="EPSG:4326")
    return gdf
dataset_dir = "/content/drive/MyDrive/dataset/shards"

# Зчитуємо перші 10 файлів із шардами
geo_df = read_shards(dataset_dir, n_shards=10)

# Виведемо перші записи для перевірки
print(geo_df.head())

# Зчитуємо всі доступні файли
geo_df = read_shards(dataset_dir, n_shards=4)
```

```
# Перевіряємо результати
print(geo_df.head())
print(f"Зчитано {len(geo_df)} записів.")

# Просторове приєднання точок до геосітки
geo_df = gpd.sjoin(geo_df, gdf, how='left')

# Видаляємо зайві колонки, якщо вони не потрібні
geo_df.drop(['country_id', 'country_name', 'images', 'idx_images'], inplace=True, axis=1, errors='ignore')
# Видаляємо точки, які не знайшли відповідність у геосітці
geo_df.dropna(inplace=True)
# Перетворюємо 'index_right' у цілий тип, якщо це необхідно
geo_df['index_right'] = geo_df['index_right'].astype(int)
# Перевіряємо результат
print(geo_df.head())
import numpy as np
from tqdm import tqdm

def generate_haversine_smooth_label(df, gdf, tau=60):
    # Завантажуємо координати точок і центроїдів
      points = np.array(df.geometry.apply(lambda x: [x.x, x.y]).to_list()) # Координати точок
        centroids = np.array(gdf.geometry.apply(lambda x: [x.centroid.x, x.centroid.y]).to_list())  # Центроїди осередків

    # Ініціалізуємо масив для збереження міток
    label_haversine = np.zeros((len(points), len(centroids)), dtype=np.float64)
```

```
        # Обчислюємо відстані та гладкі мітки
        for i in tqdm(range(len(centroids)), desc="Обчислення гладких міток"):
            # Відстані Гаверсина між точками та поточним центроїдом
            dlng = np.radians(points[:, 0] - centroids[i, 0])
            dlat = np.radians(points[:, 1] - centroids[i, 1])
                a = np.sin(dlat / 2) ** 2 + np.cos(np.radians(points[:, 1])) *
np.cos(np.radians(centroids[i, 1])) * np.sin(dlng / 2) ** 2
            c = 2 * np.arcsin(np.sqrt(a))
            distance = 6367 * c  # Відстань у кілометрах

            # Обчислюємо гладкі мітки (зважені за допомогою tau)
            label_haversine[:, i] = np.exp(-distance / tau)

        # Нормалізуємо мітки так, щоб сума для кожної точки дорівнювала 1
        label_haversine /= label_haversine.sum(axis=1, keepdims=True)

        return label_haversine.astype(np.float32)
    # Генеруємо "гладкі мітки"
    smooth_labels = generate_haversine_smooth_label(geo_df, gdf)

    # Перевіряємо форму масиву
    print(f"Гладкі мітки згенеровано: {smooth_labels.shape}")

    # Приклад мітки для першої точки
    print("Мітка для першої точки:", smooth_labels[0])

    from PIL import Image
```

```
import numpy as np
from io import BytesIO

def extract_images_from_shards(dataset_dir, shard_files, new_size=(256, 256)):
    """
    Зчитує зображення з шардів і перетворює їх у NumPy-масив.

    Параметри:
    - dataset_dir: шлях до директорії з файлами шардів.
    - shard_files: список файлів шардів.
    - new_size: новий розмір зображень (за замовчуванням 256x256).

    Повертає:
    - Масив зображень.
    """
    images = []
    for shard_file in shard_files:
        shard_path = os.path.join(dataset_dir, shard_file)
        print(f"Processing images from {shard_path}...")
        with open(shard_path, "rb") as infile:
                for record in tqdm(msgpack.Unpacker(infile, raw=False), total=30_000):
                    # Зображення перетворюється у формат PIL Image
                    image = Image.open(BytesIO(record['image']))
                    image = image.resize(new_size)  # Зміна розміру
                    image = np.array(image)  # Конвертація у NumPy
                    images.append(image)
```

```
    return np.array(images)

# Отримуємо список файлів із шардами
shard_files = sorted([f for f in os.listdir(dataset_dir) if f.endswith('.msg')])

# Зчитуємо зображення
images = extract_images_from_shards(dataset_dir, shard_files[:4]) # Використовуємо всі 4 шарди
print(f"Зображення зчитано: {images.shape}")

# Перетворення гладких міток у формат PyTorch
import torch
from torch.utils.data import TensorDataset

# Перетворюємо зображення у тензор
images_tensor = torch.from_numpy(images).permute(0, 3, 1, 2).float() # (Batch, Channels, Height, Width)

# Перетворюємо мітки у тензор
labels_tensor = torch.from_numpy(smooth_labels)

# Створюємо датасет
dataset = TensorDataset(images_tensor, labels_tensor)

print(f"Датасет створено: {len(dataset)} зразків")

min_size = min(images_tensor.size(0), labels_tensor.size(0))
```

```
    # Обрізаємо обидва масиви
    images_tensor = images_tensor[:min_size]
    labels_tensor = labels_tensor[:min_size]

    print(f"Після обрізання: images_tensor={images_tensor.size()}, labels_tensor={labels_tensor.size()}")

    from torch.utils.data import TensorDataset

    # Створюємо датасет
    dataset = TensorDataset(images_tensor, labels_tensor)

    print(f"Датасет створено: {len(dataset)} зразків")

    from torch.utils.data import DataLoader

    batch_size = 64  # Розмір батчу

    # Створюємо DataLoader
    data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

    # Перевірка DataLoader
    for images, labels in data_loader:
        print(images.shape, labels.shape)  # Виводимо розміри одного батчу
        break

    from torch.utils.data import random_split
```

```
train_size = int(0.8 * len(dataset))  # 80% для навчання
val_size = len(dataset) - train_size  # 20% для валідації

# Розбиття
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print(f"Навчальна вибірка: {len(train_dataset)}, Валідаційна вибірка: {len(val_dataset)}")

import torchvision.models as models
import torch.nn as nn

class ResNetGeoLocalization(nn.Module):
    def __init__(self, num_outputs):
        super(ResNetGeoLocalization, self).__init__()

        self.base_model = models.resnet50(pretrained=True)

        # Змінюємо останній шар (Fully Connected)
        num_features = self.base_model.fc.in_features
        self.base_model.fc = nn.Linear(num_features, num_outputs)

    def forward(self, x):
        return self.base_model(x)
import torch.optim as optim

criterion = nn.MSELoss()  # Функція втрат для регресії
```

```
optimizer = optim.Adam(model.parameters(), lr=0.0001)  # Менший learning rate для стабільного навчання

# Імпорт бібліотеки DataLoader
from torch.utils.data import DataLoader

# Створення DataLoader
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Перевірка, чи DataLoader створені
print(f"Train Loader: {len(train_loader)}, Validation Loader: {len(val_loader)}")

# Навчання
epochs = 25
device = "cuda" if torch.cuda.is_available() else "cpu"

for epoch in range(epochs):
    model.train()  # Режим навчання
    running_loss = 0.0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Обнулення градієнтів
        optimizer.zero_grad()
```

```
            # Прямий прохід
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Зворотний прохід
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        scheduler.step()  # Оновлюємо learning rate

            print(f"Епоха {epoch+1}, Втрата на тренуванні: {running_loss / len(train_loader):.4f}")

        # Оцінка на валідаційних даних
        model.eval()  # Режим оцінки
        val_loss = 0.0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

              print(f"Епоха {epoch+1}, Втрата на валідації: {val_loss / len(val_loader):.4f}")
```

```
import torch
import numpy as np
import matplotlib.pyplot as plt

# Метрики
def mean_absolute_error(y_true, y_pred):
    return torch.mean(torch.abs(y_true - y_pred)).item()

def mean_squared_error(y_true, y_pred):
    return torch.mean((y_true - y_pred) ** 2).item()

def root_mean_squared_error(y_true, y_pred):
    return torch.sqrt(torch.mean((y_true - y_pred) ** 2)).item()

def r2_score(y_true, y_pred):
    ss_total = torch.sum((y_true - torch.mean(y_true)) ** 2)
    ss_residual = torch.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total).item()

# Додатково для геолокації
def haversine_error(y_true, y_pred):
    R = 6371  # Радіус Землі в км
    lat1, lon1 = y_true[:, 0], y_true[:, 1]
    lat2, lon2 = y_pred[:, 0], y_pred[:, 1]
    dlat = torch.radians(lat2 - lat1)
    dlon = torch.radians(lon2 - lon1)
```

```
            a = torch.sin(dlat / 2)**2 + torch.cos(torch.radians(lat1)) *
torch.cos(torch.radians(lat2)) * torch.sin(dlon / 2)**2
        c = 2 * torch.atan2(torch.sqrt(a), torch.sqrt(1 - a))
        return torch.mean(R * c).item()

    # Оцінка моделі
    model.eval()
    all_preds = []
    all_targets = []

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            all_preds.append(outputs.cpu())
            all_targets.append(labels.cpu())

    # Об'єднуємо всі передбачення та мітки
    all_preds = torch.cat(all_preds)
    all_targets = torch.cat(all_targets)

    # Обчислення метрик
    mae = mean_absolute_error(all_targets, all_preds)
    mse = mean_squared_error(all_targets, all_preds)
    rmse = root_mean_squared_error(all_targets, all_preds)
    r2 = r2_score(all_targets, all_preds)

    print(f"MAE: {mae:.4f}, MSE: {mse:.4f}, RMSE: {rmse:.4f}, R²: {r2:.4f}")
```