МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень»
здобувача групи ІН.м - 26 Солоніни Станіслава Сергійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Станіслав СОЛОНІНА
(підпис)

Керівник,
старший викладач, Альона МОСКАЛЕНКО

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-26 Солоніни Станіслава Сергійовича

1. Тема роботи: «Інформаційна технологія швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень»
затверджую наказом по СумДУ від *«06» грудня 2023 року № 1412-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Аналіз об'єкту дослідження. 3) Розробка інформаційного та програмного забезпечення для тестування детектора об'єктів. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__ __» _____ 2023 р.
Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.10-10.11	Виконано
2	*Аналіз об'єкту дослідження*	11.11-14.11	Виконано
3	*Розробка інформаційного та програмного забезпечення для тестування детектора об'єктів*	15.11-20.11	Виконано
4	*Аналіз отриманих результатів*	21.11-23.11	Виконано
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	24.11-09.12	Виконано

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: стор. 77, рис. 8, табл. 5, додаток 1, 22 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи, присвячена швидкій адаптації детектора об'єктів до нових завдань у рамках обмежених ресурсів, є надзвичайно актуальною в контексті стрімкого розвитку області машинного зору та штучного інтелекту. Ця тема зосереджується на важливості розробки ефективних та ресурсозберігаючих методів для адаптації детекторів об'єктів, що значно покращує точність і надійність детекції в різноманітних умовах.

Об’єкт дослідження – інформаційна технологія адаптації детектора об'єктів.

Мета роботи – створення інформаційної технології швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень.

Методи дослідження — методи побудови інформаційних систем.

Результати — було спроектовано та реалізовано інформаційну систему адаптації детектора на основі програмної системи Detectron2 та алгоритму DINO V2. Під час виконання завдання було проаналізовано моделі глибинного навчання різного розміру та методи використання технік Open Vocabulary Learning та Few-Shot Learning при вирішенні даної задачі. Були розглянуті критерії ефективності детектора об'єктів. Результатом виконанням досліджень є створення технології швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень та сформована порівняльна характеристика ефективності роботи технік машинного навчання.

DETECTRON2, DINO V2, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ,
МАШИНЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ

ЗМІСТ

ВСТУП

Актуальність. В епоху стрімкого розвитку технологій, інформаційні системи та штучний інтелект набувають все більшого значення в різних галузях, включаючи адаптацію детекторів об'єктів до нових задач. Це особливо важливо в умовах, коли ресурси обмежені, що вимагає інноваційних підходів до обробки та аналізу даних. Швидка адаптація детекторів об'єктів до нових вимог з мінімальними ресурсними витратами є ключовим завданням, яке може сприяти підвищенню ефективності та гнучкості в багатьох технологічних процесах.

Об’єкт дослідження. Процес адаптації детекторів об'єктів до нових задач в умовах обмежених ресурсів.

Предмет дослідження. Методи та алгоритми, що дозволяють швидко адаптувати детектори об'єктів до змінюваних умов та задач, мінімізуючи при цьому використання ресурсів.

Гіпотеза. Швидкої адаптації детектора об'єктів до нових задач в умовах обмежених ресурсів можна досягнути шляхом застосування інформаційної системи, що реалізує модель навчання за малою кількістю зразків.

Новизна. Розроблення такої технології забезпечить нові можливості в галузі машинного навчання та штучного інтелекту, підвищуючи гнучкість та ефективність систем розпізнавання об'єктів. Це відкриє шлях для масштабного впровадження адаптивних технологій у різноманітних областях, включаючи як промисловість, так і сферу надання послуг.

Структура. Дана кваліфікаційна робота включає в себе вступ, формулювання проблеми та вибір підходів для її рішення, детальний розгляд програмного забезпечення інформаційної системи, висновки, список використаних джерел, а також додатки.

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Сучасний стан та тенденції розвитку алгоритмів детектування об'єктів на зображенні

Алгоритми детектування об'єктів на зображеннях мають широкий спектр застосувань в багатьох галузях. У медичній діагностиці алгоритми комп'ютерного зору і глибокого навчання знаходять все більше застосувань [1]. Ці технології використовуються для аналізу медичних знімків, таких як МРТ, КТ, або рентгенівські зображення, для виявлення та класифікації різних патологій. Це не лише поліпшує точність діагнозів, але й забезпечує швидше їх встановлення. В автономних транспортних засобах алгоритми детектування об'єктів відіграють вирішальну роль у забезпеченні безпеки та ефективності руху [2]. Вони дозволяють розпізнавати дорожні знаки, перешкоди, інші транспортні засоби, пішоходів та навіть прогнозувати їхні потенційні дії. У сфері безпеки алгоритми детектування об'єктів та комп'ютерного зору відіграють ключову роль [3], особливо у системах відеонагляду. Вони використовуються для виявлення підозрілих дій, розпізнавання осіб, автоматичного моніторингу та аналізу великих територій.

Сучасний стан алгоритмів детектування об'єктів на зображеннях характеризується використанням декількох передових технологій: Згорткові нейронні мережі (CNN) [4], Рекурентні нейронні мережі (RNN) [5], Трансформери [6]. Згорткові нейронні мережі (CNN): Ці мережі є найпопулярнішими для аналізу візуальних даних. CNN використовують згорткові шари, які ефективно аналізують локальні області зображення, виявляючи характеристики на різних рівнях складності [7]. Останні розробки включають глибші та більш ефективні архітектури, такі як ResNet та DenseNet. Рекурентні нейронні мережі (RNN): Хоча RNN традиційно використовуються для обробки послідовних даних [8, 9],

наприклад тексту або часових рядів, вони також знайшли застосування в аналізі відео та в розпізнаванні образів для виявлення об'єктів у часових послідовностях. Трансформери: Ця новітня технологія, яка спочатку була розроблена для задач обробки природної мови, тепер застосовується в області комп'ютерного зору. Трансформери, такі як Vision Transformer (ViT) [10], використовують механізми уваги для аналізу зображень, дозволяючи моделям краще взаємодіяти з різними частинами зображення та ефективно інтегрувати інформацію з різних областей.

В сучасному світі обробка великих даних є одним із ключових викликів у розвитку алгоритмів машинного навчання[11]. Із зростанням обсягів даних, особливо у сфері комп'ютерного зору, виникає необхідність у їх ефективній обробці та аналізі. Алгоритми повинні бути спроможні швидко обробляти великі набори даних, при цьому не втрачаючи точності та ефективності. Одним із рішень є використання розподілених систем та обчислень у хмарі, які дозволяють масштабувати процеси обробки даних. Також застосування алгоритмів зменшення розмірності та ефективних методів зберігання даних сприяє оптимізації цих процесів.

Зменшення витрат на обчислення в області нейронних мереж є важливим напрямком досліджень, оскільки більшість потужних моделей вимагають значних обчислювальних ресурсів [12]. Оптимізація архітектур нейронних мереж може включати в себе використання більш ефективних шарів та алгоритмів, а також підходи до зменшення розміру моделей без втрати точності.

Покращення точності алгоритмів детектування об'єктів є ключовою задачею в області комп'ютерного зору. Це включає не лише розробку більш ефективних алгоритмів, але й використання різноманітних наборів даних для тренування та вдосконалення технік навчання. Інновації у цій сфері часто зосереджуються на збільшенні розмірності та глибини мереж, що дозволяє моделям краще вловлювати складні взаємозв'язки у даних.

Автоматизація процесу навчання алгоритмів є одним із найбільш обіцяючих напрямків у розвитку штучного інтелекту [13]. Це включає використання алгоритмів для автоматичного налаштування гіперпараметрів та оптимізації архітектур нейронних мереж. Такий підхід може значно підвищити ефективність та точність моделей, оскільки він дозволяє знаходити оптимальні конфігурації, які можуть бути неочевидними для людських дослідників. Підходи автоматизації навчання включають в себе техніки, як-от нейронна архітектурна пошук (NAS) та автоматичне машинне навчання (AutoML). NAS зосереджується на автоматичному створенні оптимальних архітектур мереж, тоді як AutoML охоплює ширший спектр аспектів, включаючи підготовку даних, вибір моделей та налаштування гіперпараметрів.

Інтеграція комп'ютерного зору з розширеною (AR) та віртуальною реальністю (VR) відкриває нові можливості для багатьох галузей. У медицині, наприклад, це може включати використання AR для допомоги у хірургічних процедурах шляхом накладання важливої інформації безпосередньо на поле зору хірурга. У освіті, AR та VR можуть забезпечити інтерактивне та занурювальне навчання, дозволяючи студентам віртуально взаємодіяти з об'єктами або концепціями.

Розвиток глибокого навчання дозволяє алгоритмам не лише розпізнавати об'єкти, але й розуміти контекст та взаємодії між ними в зображенні [14]. Це включає здатність алгоритмів аналізувати складні сцени та розуміти відносини та дії, які відбуваються між об'єктами. Наприклад, в автономних транспортних засобах це може означати розуміння поведінки пішоходів та інших учасників дорожнього руху для безпечного маневрування. Однією з ключових технологій, яка сприяє цьому розвитку, є глибокі нейронні мережі, зокрема ті, що використовують механізми уваги, як у трансформерах.

Сучасний стан та тенденції розвитку алгоритмів детектування об'єктів на зображенні свідчать про значний прогрес у цій області. Розвиток глибинного навчання та штучного інтелекту відкриває нові горизонти для покращення точності та ефективності цих систем. Майбутні дослідження, ймовірно, зосередяться на створенні більш ефективних, швидких та надійних рішень, водночас звертаючи увагу на етичні та приватні питання, пов'язані з цими технологіями.

1.2. Моделі і методи навчання моделей штучного інтелекту за малою кількістю зразків

У сфері штучного інтелекту існують кілька моделей та методів навчання, які дозволяють ефективно працювати з малими даними. Ці методи важливі для вирішення задач, де збір великої кількості зразків є важким або неможливим.

Zero-Shot Learning (ZSL) - це техніка машинного навчання, яка дозволяє моделі класифікувати об'єкти з раніше небачених класів, не отримуючи спеціального навчання для цих класів [15]. Ця техніка корисна для автономних систем, які повинні мати можливість самостійно ідентифікувати та класифікувати нові об'єкти. У ZSL модель попередньо навчається на наборі класів (бачених класів), а потім її просять узагальнити на інший набір класів (небачених класів) без будь-якого додаткового навчання. Метою ZSL є перенесення знань, що вже містяться в моделі, на завдання класифікації невидимих класів. ZSL є підгалуззю трансферного навчання, яке передбачає адаптацію моделі до нової задачі або набору класів. Існує два типи трансферного навчання: гомогенне Transfer Learning, в якому простір ознак і простір міток однакові, і гетерогенне Transfer Learning, в якому простір ознак і простір міток різняться. ZSL належить до останньої категорії.

Zero-Shot Learning корисне тим, що може допомогти подолати труднощі та витрати, пов'язані з маркуванням даних - трудомістким і часто дорогим процессом [16]. Особливо важко отримати анотації від спеціалізованих експертів у певних галузях, таких як біомедичні дані, які потребують досвіду кваліфікованих медичних працівників. Крім того, може бути недостатньо навчальних даних для класу (наприклад, у випадку спроби маркування рідкісних дефектів у продуктах), або ж дані можуть бути незбалансованими. Ці два сценарії ускладнюють для моделі точне відображення реальних сценаріїв. Крім того, методи неконтрольованого навчання також можуть бути недостатніми в деяких завданнях, таких як класифікація різних підкатегорій одного і того ж об'єкта (наприклад, породи собак). ZSL може допомогти вирішити ці проблеми, дозволяючи моделі виконувати класифікацію нових класів (невидимих класів), використовуючи знання, які вона вже засвоїла під час навчання. ZSL має широкий спектр застосувань, включаючи класифікацію зображень, виявлення об'єктів, відстеження об'єктів, семантичну сегментацію, передачу стилю та обробку природної мови. Він особливо корисний у сценаріях, де мічені дані для нових класів є дефіцитними або дорогими для отримання.

У Zero-Shot Learning дані поділяються на три категорії:

- побачені класи - класи, які були використані для навчання моделі;
- невидимі класи - класи, які модель повинна вміти класифікувати без спеціального навчання. Дані з цих класів не використовувалися під час навчання;
- допоміжна інформація - описи, семантична інформація або вставки слів про всі невидимі класи. Це необхідно для розв'язання

проблеми навчання з нуля, оскільки не існує маркованих прикладів невидимих класів.

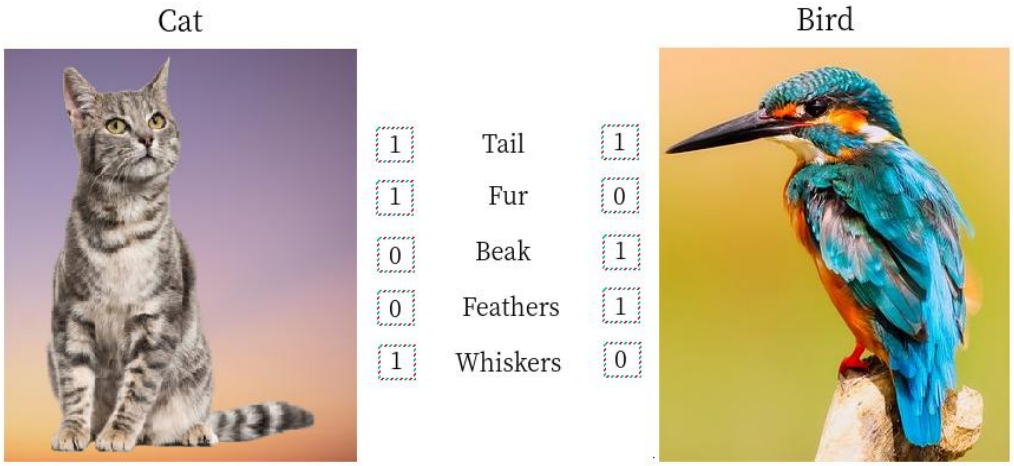

Рисунок 1.1 – Приклад семантичного вбудовування

Zero-Shot Learning - це двоетапний процес, який включає в себе навчання та висновок. Під час навчання модель дізнається про маркований набір зразків даних. Під час висновку модель використовує ці знання та допоміжну інформацію для класифікації нового набору класів [17].

У випадку комп'ютерного зору "знання" - це маркований навчальний набір видимих і невидимих класів. Надані дані повинні бути пов'язані у високорозмірному векторному просторі, який називається семантичним простором. Знання з видимих класів можуть бути перенесені на невидимі класи в цьому просторі.

Існують два підходи до вирішення проблеми розпізнавання нульового кадру: методи на основі класифікатора та методи на основі екземплярів. Методи на основі класифікаторів для навчання багатокласового класифікатора з Zero-Shot зазвичай передбачають використання підходу "один проти решти", коли для кожного невидимого класу навчається окремий бінарний класифікатор.

Залежно від конкретного підходу, який використовується для побудови класифікаторів, ці методи можна розділити на три підкатегорії:

- методи відповідності спрямовані на побудову класифікатора для невидимих класів шляхом встановлення зв'язку між бінарним класифікатором "один проти решти" для кожного класу і відповідним йому прототипом класу в семантичному просторі;
- методи відношень для побудови класифікатора для невидимих класів зосереджуються на відношеннях між цими класами та всередині них. У просторі ознак бінарні класифікатори "один проти решти" для видимих класів можуть бути навчені на основі наявних даних;
- комбіновані методи передбачають побудову класифікатора для невидимих класів шляхом комбінування класифікаторів для базових елементів, що входять до складу цих класів;
- методи на основі екземплярів для zero-shot learning передбачають спочатку отримання маркованих прикладів невидимих класів, а потім використання цих прикладів для навчання класифікатора.

Для оцінки ефективності класифікатора в zero-shot розпізнаванні , необхідно обчислити середню для категорії top-k точність. Ця метрика вимірює частку тестових вибірок, чий фактичний клас (мітка) знаходиться серед k класів з найвищою ймовірністю, передбаченою класифікатором.

Середня точність top-k для кожної категорії обчислюється по формулі (1.1) як середнє значення точності top-k для кожного классу.

$$Accuracy = \frac{1}{C}\sum_{i=0}^{c-1}\frac{\#\text{ correct predictions in i}}{\#\text{ of samples in i}}, \qquad (1.1)$$

де C – позначає кількість невидимих класів.

До обмежень Zero-Shot Learning включають:

- коли модель навчається на даних лише з видимих класів, вона з більшою ймовірністю передбачатиме вибірки з невидимих класів як такі, що належать до одного з видимих класів під час фази тестування;

- проблема "зсуву домену", яка виникає, коли розподіл даних у навчальній множині (видимі класи) і тестовій множині (яка може включати зразки як видимих, так і невидимих класів) значно відрізняється;
- zero-Shot Learning часто стикається з проблемою "хабності", яка виникає через прокляття розмірності при пошуку найближчого сусіда для даних високої розмірності. Точки, які називаються "хабами", мають тенденцію часто з'являтися в k-найближчих сусідів інших точок;
- втрата прихованої інформації, присутньої у видимих класах, яка може бути не важливою для розрізнення видимих класів, але може бути важливою для класифікації невидимих класів на етапі тестування.

Zero-Shot Learning (ZSL) є важливим напрямком у машинному навчанні, який дозволяє моделям класифікувати об'єкти з класів, які не були представлені під час тренування. Завдяки цьому підходу можливе розширення можливостей автономних систем, які потребують ідентифікувати та класифікувати нові об'єкти. Особливо корисним ZSL є в ситуаціях, де маркування даних є складним чи дорогим, наприклад, у біомедичній сфері або при роботі з рідкісними даними. Методика ZSL дозволяє моделям використовувати знання, отримані під час тренування на видимих класах, для класифікації об'єктів невидимих класів, використовуючи допоміжну інформацію. Це робить ZSL особливо важливим для задач, де збалансовані та повні набори даних недоступні. Таким чином, ZSL відкриває шлях для розв'язання багатьох складних задач, пов'язаних з обробкою та класифікацією даних у різних галузях.

Few-Shot Learning (FSL) - це техніка машинного навчання, яка дозволяє моделям вчитися та працювати ефективно, навіть якщо доступно лише кілька зразків для кожного классу [18]. Few-Shot Learning контрастує з традиційними методами навчання моделей машинного навчання, де зазвичай використовується велика кількість навчальних даних. Few-shot learning використовується в основному в комп'ютерному зорі. На практиці

few-shot learning корисне, коли навчальні приклади важко знайти (наприклад, випадки рідкісних захворювань) або коли вартість анотації даних висока.

Few-Shot Learning відіграє важливу роль у сучасних методах машинного навчання, і його вплив можна розглянути з різних аспектів.

Однією з ключових переваг Few-Shot Learning є його здатність навчати машини розпізнавати рідкісні випадки. Наприклад, у класифікації зображень рідкісних захворювань, як COVID-19, модель комп'ютерного зору, навчена за методами Few-Shot Learning, може точно класифікувати зображення рентгенівського знімка грудної клітки, використовуючи лише невелику кількість рентгенівських знімків.

Другим важливим аспектом є зменшення вартості даних. Few-Shot Learning вимагає меншої кількості даних для навчання моделі, що знижує витрати, пов'язані зі збором та аннотуванням даних. Невелика кількість навчальних даних означає низьку розмірність навчального набору даних і, відповідно, зниження обчислювальних витрат.

Нарешті, Few-Shot Learning дозволяє машинам навчатися, подібно до людини. Люди можуть швидко вловлювати різницю між об'єктами після перегляду кількох прикладів, наприклад, розпізнавати символи, написані від руки. Few-Shot Learning дозволяє комп'ютерам також ефективно навчатися на обмеженому наборі прикладів, тим самим підвищуючи їх здатність до гнучкого розпізнавання і аналізу даних.

Few-Shot Learning використовує підхід N-way-K-shot класифікації для розрізнення N класів на K прикладах. Використання звичайних методів не спрацює, оскільки сучасні алгоритми класифікації залежать від набагато більшої кількості параметрів, ніж навчальні приклади, і будуть погано узагальнювати. Якщо даних недостатньо, щоб обмежити проблему, то одним з можливих рішень є вивчення досвіду інших подібних проблем.

З цією метою більшість підходів характеризують навчання з кількох спроб як проблему meta-learning.

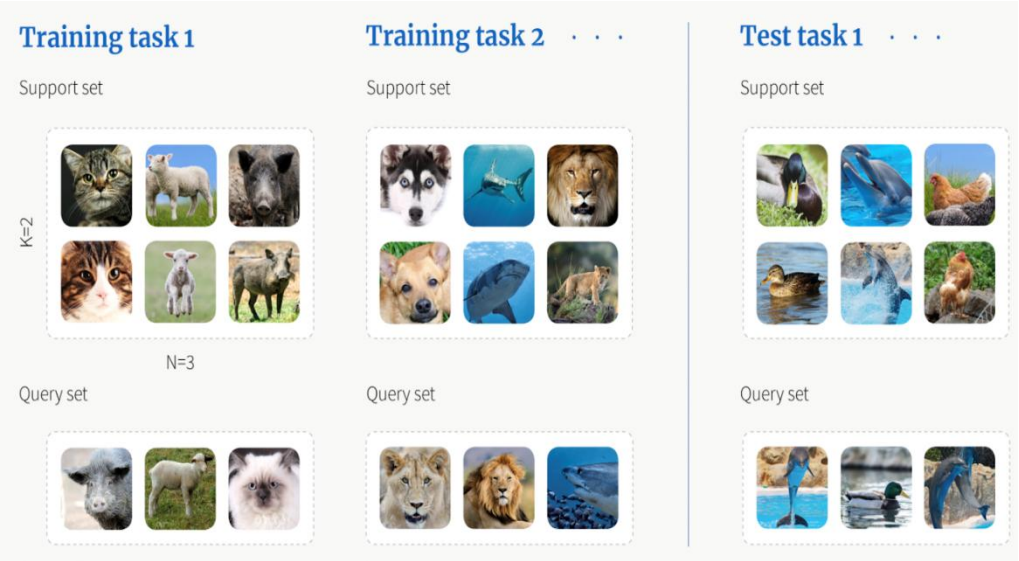

Рисунок 1.2 – Мета-структура навчання для FSL

Основною метою традиційних фреймворків Few-Shot Learning є вивчення функції подібності, яка може відображати схожість між класами в наборах підтримки та запитів. Функції подібності зазвичай виводять значення ймовірності подібності.

Функція досконалої подібності повинна вивести значення 1.0 при порівнянні двох зображень котів (I1 та I2). У двох інших випадках, коли зображення котів порівнюються із зображенням оцелота, значення схожості має бути 0.0. (рис. 1.3).

Few-Shot Learning може використовувати великомасштабний маркований набір даних для навчання параметрів такої функції подібності. Для цього можна використати навчальний набір, який застосовувався для попереднього навчання глибинної моделі в контрольованому режимі. Після того, як параметри функції подібності будуть навчені, її можна використовувати на етапі Few-Shot Learning для визначення ймовірностей подібності на множині запитів, використовуючи інформацію з опорної множини.

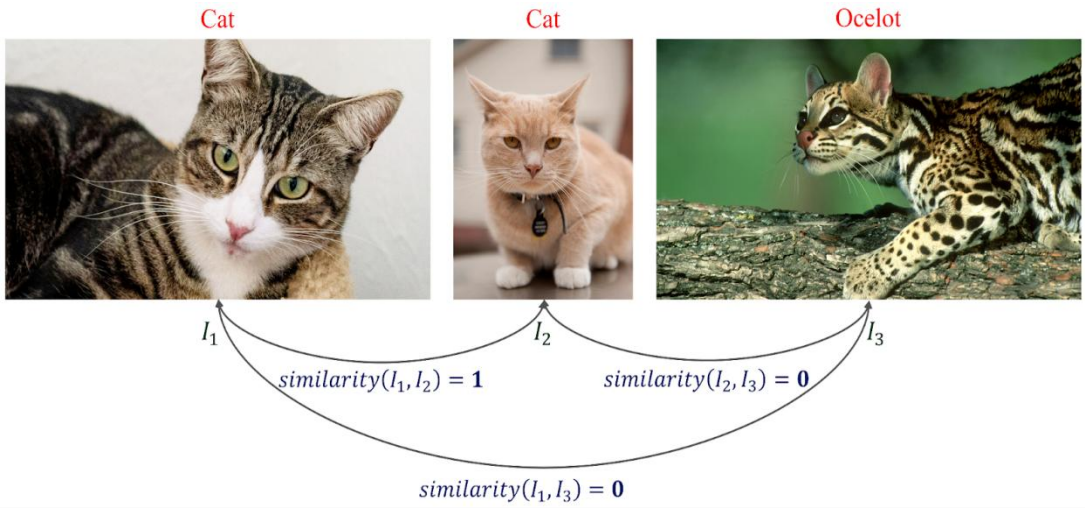

Рисунок 1.3 – Сценарій для міри схожості в Few-Shot Learning

Потім для кожної вибірки з набору запитів клас з найбільшою схожістю з опорного набору буде визначено як передбачення мітки класу за допомогою моделі Few-Shot. (рис. 1.4)

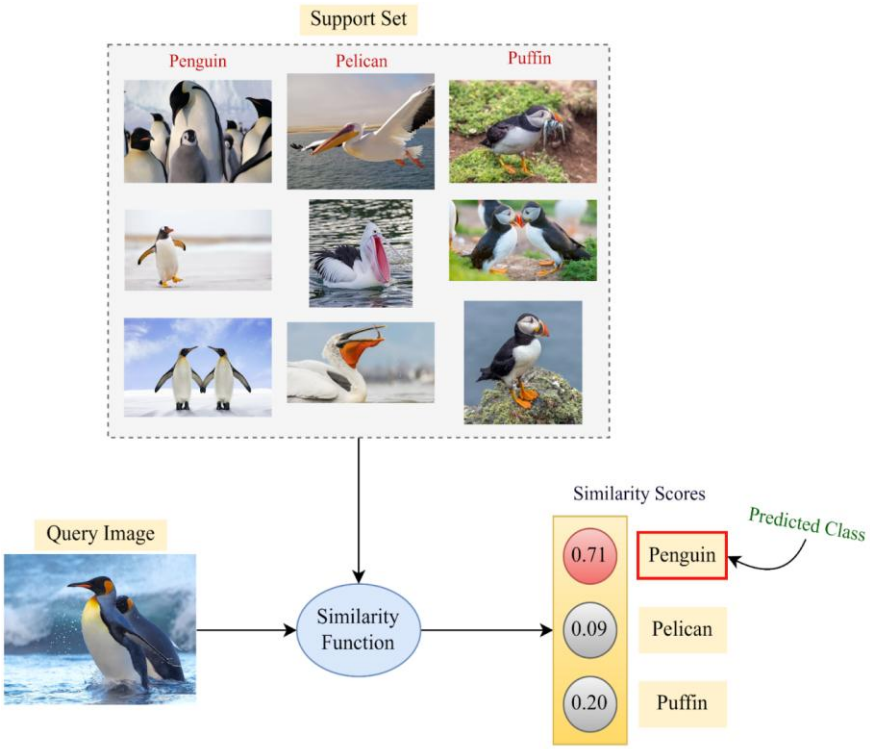

Рисунок 1.4 – Огляд того, як модель Few-Shot Learning робить прогноз.

Few-Shot Learning є важливою концепцією в машинному навчанні, яка дозволяє моделям виконувати завдання з мінімальною кількістю даних

для тренування [19]. Цей підхід можна умовно поділити на кілька категорій.

Першою категорією є Data-Level Few-Shot Learning, який має просту концепцію. Якщо навчання Few-Shot Learning-моделі сповільнюється через брак навчальних даних, можна додати більше даних, які можуть бути структурованими або ні. Це дозволяє моделі краще адаптуватися і зменшити ризик надмірного або недостатнього пристосування.

Друга категорія - це Few-Shot Learning на рівні параметрів, який передбачає використання Meta-Learning. У цьому контексті Meta-Learning контролює використання параметрів моделей для інтелектуального визначення важливих характеристик для поставленої задачі. Оскільки кількість вибірок у Few-Shot Learning обмежена, часто зустрічається проблема надмірної підгонки, оскільки вибірки мають обширні та високорозмірні простори.

Третя категорія - Metric-Level Few-Shot Learning. Вона спрямована на вивчення функції відстані між точками даних. Ознаки витягуються з зображень, а відстань між ними обчислюється в просторі вбудовування. Функція відстані може бути евклідовою відстанню, відстанню земного переміщення, відстанню на основі косинусної подібності тощо.

Четверта категорія включає підходи до Meta-Learning на основі градієнта. Вони використовують двох учнів - модель вчителя (базовий учень) і модель учня (мета-учень) з використанням дистиляції знань. Модель вчителя спрямовує модель учня через багатовимірний простір параметрів.

Few-Shot Learning застосовується в різних галузях. Наприклад, виявлення об'єктів є проблемою комп'ютерного зору, де ідентифікують і визначають місцезнаходження об'єктів на зображенні або відеопослідовності. Семантична сегментація, де кожному пікселю зображення присвоюється клас, також використовує метод Few-Shot

Learning для виконання бінарної та багатоміткальної сегментації. У галузі робототехніки підходи Few-Shot Learning дозволяють роботам імітувати здатність людини узагальнювати завдання, використовуючи лише кілька демонстрацій.

Останнім часом Few-Shot Learning стало популярним і в задачах обробки природної мови, де мітки для обробки мови за своєю суттю важко знайти.Few-Shot Learning (FSL) [20] є важливим напрямком у машинному навчанні, орієнтованим на ефективне навчання моделей за дуже обмеженої кількості даних. Цей підхід відрізняється від традиційного машинного навчання, яке залежить від великих обсягів даних, і є особливо корисним у ситуаціях, де збір даних є складним або дорогим, як, наприклад, у медичній діагностиці рідкісних захворювань. FSL імітує людське навчання, де здатність узагальнювати інформацію відбувається з дуже малої кількості прикладів. Методи FSL включають Data-Level, Metric-Level та Meta-Learning підходи і мають широке застосування в комп'ютерному зорі, робототехніці та обробці природної мови.

Transfer Learning (TL) - це техніка машинного навчання (ML), в якому знання, отримані при виконанні певного завдання, повторно використовуються для підвищення продуктивності при виконанні пов'язаного завдання [21]. Загальна ідея Transfer Learning полягає в тому, щоб використовувати знання, отримані моделлю при виконанні завдання з великою кількістю доступних маркованих навчальних даних, у новому завданні, в якому не так багато даних. Замість того, щоб починати процес навчання з нуля, процес починається з шаблонів, отриманих при вирішенні спорідненої задачі.

Transfer Learning здебільшого використовується в задачах комп'ютерного зору та обробки природної мови, таких як аналіз настроїв, через величезну кількість необхідних обчислювальних потужностей [22].

Transfer Learning не є технікою машинного навчання, але його можна розглядати як "методологію проектування" в межах, наприклад, активного навчання. Воно також не є ексклюзивною частиною або сферою дослідження машинного навчання. Тим не менш, воно стало досить популярним у поєднанні з нейронними мережами, які вимагають величезних обсягів даних і обчислювальних потужностей.

У комп'ютерному зорі нейронні мережі зазвичай намагаються розпізнати краї в ранніх шарах, форми в середньому шарі і деякі специфічні особливості завдання в пізніх шарах. При Transfer Learning використовуються ранні та середні шари, і перенавчаються лише останні (рис. 1.5). Це допомагає використовувати марковані дані завдання, на якому він спочатку навчався.

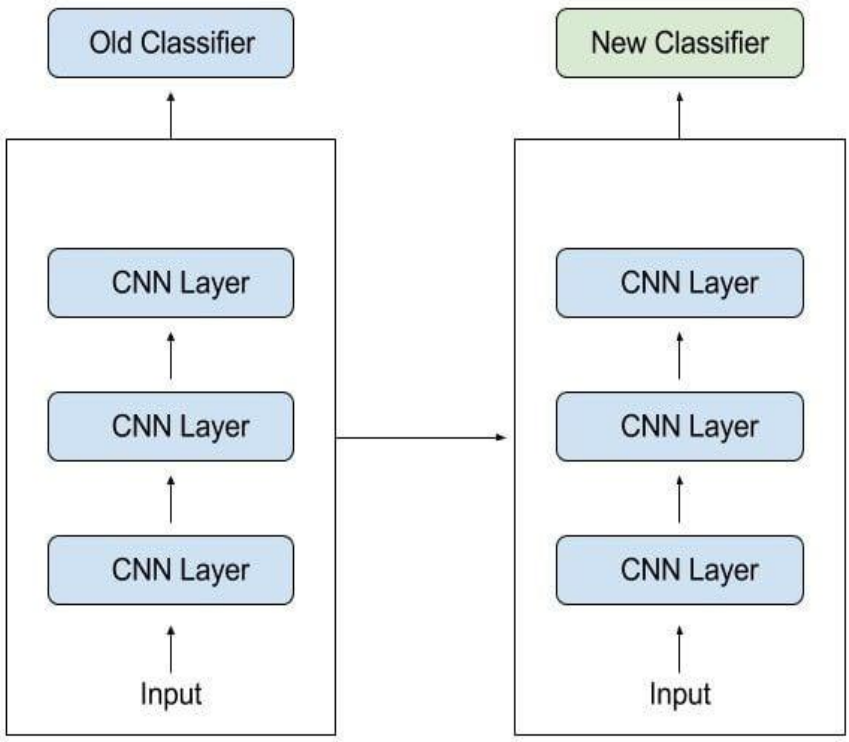

Рисунок 1.5 – Схема використання шарів Transfer Learning

При навчанні з перенесенням переноситься якомога більше знань з попередньої задачі, на якій навчалася модель, на нову задачу, що стоїть

перед нею. Ці знання можуть бути в різних формах, залежно від задачі та даних.

Transfer learning має кілька переваг, але основними з них є економія часу на навчання, краща продуктивність нейронних мереж (у більшості випадків) і відсутність потреби у великій кількості даних. Зазвичай для навчання нейронної мережі з нуля потрібно багато даних, але доступ до них не завжди доступний - саме тут у нагоді стає навчання з переносним кодом.

За допомогою Transfer learning можна побудувати надійну модель машинного навчання з порівняно невеликою кількістю навчальних даних, оскільки модель вже є попередньо навченою. Це особливо цінно при обробці природної мови, оскільки для створення великих маркованих наборів даних потрібні здебільшого експертні знання. Крім того, скорочується час навчання, оскільки іноді навчання глибокої нейронної мережі з нуля для вирішення складних завдань може зайняти кілька днів або навіть тижнів.

Transfer Learning можна використовувати для навчання з перенесенням:

- для навчання мережі з нуля недостатньо маркованих навчальних даних;
- вже існує мережа, яка попередньо навчена на подібному завданні, і зазвичай вона навчається на великих обсягах даних;
- коли завдання 1 і завдання 2 мають однакові вхідні дані.

Якщо оригінальна модель була навчена за допомогою бібліотеки з відкритим вихідним кодом, наприклад, TensorFlow, необхідно відновити її і перенавчити деякі шари для вашої задачі. Навчання з перенесенням працює лише в тому випадку, якщо особливості, вивчені в першій задачі, є загальними, тобто вони можуть бути корисними для іншої пов'язаної

задачі. Вхідні дані моделі повинні мати той самий розмір, на якому вона навчалася спочатку.

Transfer Learning (TL) є ефективною технікою в машинному навчанні, що дозволяє застосувати знання, отримані під час виконання одного завдання, до іншого, пов'язаного завдання. Ця методика особливо корисна у комп'ютерному зорі та обробці природної мови, де вона допомагає економити час та ресурси, які потрібні для навчання моделей з нуля. Основна перевага TL полягає в можливості використання попередньо навчених моделей для нових завдань, що значно зменшує потребу у великій кількості даних та обчислювальних ресурсів. Вона виявляється особливо цінною у сценаріях, де доступ до великих даних обмежений або відсутній.

Meta-Learning - це методологія в галузі машинного навчання, яка зосереджується на розробці моделей, здатних швидко адаптуватися до нових завдань, використовуючи обмежену кількість даних.

У Machine Learning створення найефективнішої оптимальної моделі передбачає пошук набору параметрів, набору навчальних даних і хорошого алгоритму навчання. Meta Learning допомагають знайти ці елементи за допомогою декількох навчальних епізодів; іншими словами, навчаючись вирішувати комплекс завдань замість того, щоб розв'язувати лише одне завдання за раз. Таким чином, модель Meta Learning стає кращою у вирішенні нового і непередбачуваного завдання. Цей метод також називають підходом "вчитися, щоб вчитися".

За допомогою Meta Learning ми можна навчити швидко адаптуватися до нових об'єктів, використовуючи попередній досвід роботи з іншими об'єктами. Можна навчити розпізнавати спільні риси в об'єктах, такі як їхня форма або текстура, і використовувати цю інформацію для швидкої адаптації до нових об'єктів.

Алгоритм Meta Learning:

- зібрати набір даних кожного об'єкта з різними формами і текстурами, і для кожного об'єкта вказати правильне місце для його розміщення;
- поділити набір даних на навчальний та тестовий;
- розділити навчальний набір даних на K пострілів, де кожне завдання складається з K об'єктів та їхніх відповідних місць розташування;

Metric-based Meta Learning широко використовується для різних завдань, таких як виявлення схожості зображень, розпізнавання підписів, розпізнавання облич тощо. Цей підхід фокусується на вивченні метрики відстані - функції, яка вимірює схожість або відмінність між парами точок даних. Metric Learning можна використовувати, щоб навчитися добре представляти простір для даних. Це робиться шляхом тренування алгоритму Metric Learning за допомогою різних наборів завдань, кожне з яких містить невелику кількість навчальних даних.

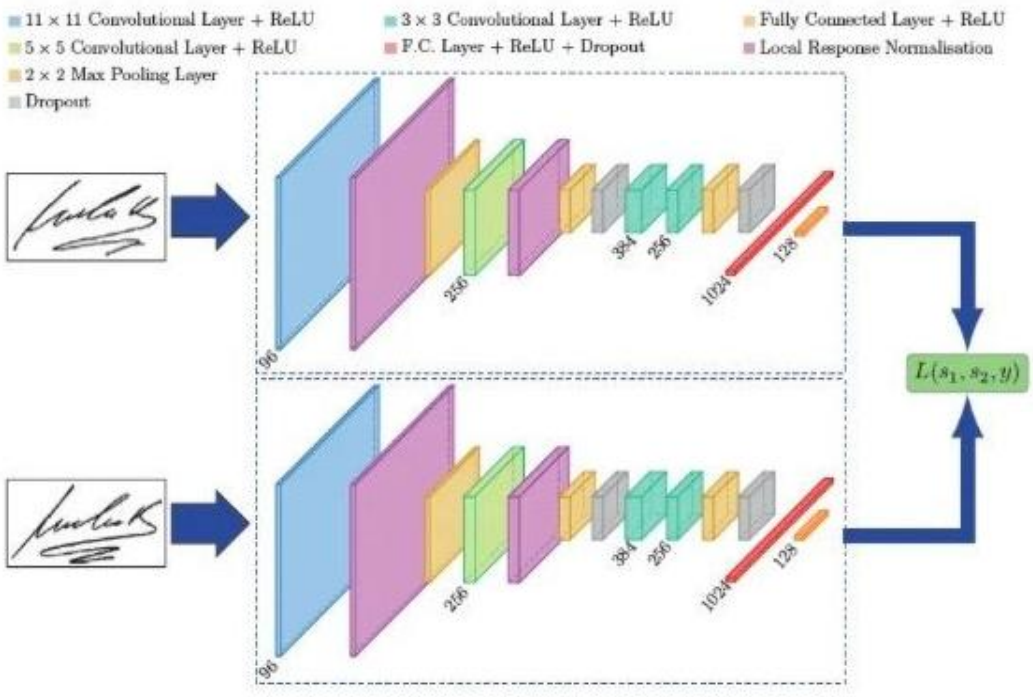

Рисунок 1.6 – Приклад Metric Learning на основі метрики

Meta Learning на основі моделей навчається знаходити параметри/архітектуру моделі таким чином, щоб швидко адаптувати її до нових завдань з невеликою кількістю навчальних даних. Meta Learning на основі оптимізації вивчає набір початкових параметрів моделі, які можна швидко адаптувати до нових завдань.

Meta Learning відкриває нові перспективи у сфері машинного навчання, пропонуючи ряд значущих переваг. Однією з основних переваг є швидка адаптація до нових завдань. Завдяки Meta Learning, модель може ефективно навчатися пристосовуватися до нових завдань з мінімальною кількістю навчальних даних, що є особливо важливим у сценаріях, де завдання змінюються непередбачувано.

Ще одна перевага полягає у ефективності використання даних. Meta Learning забезпечує високу якість навчання, навіть коли доступна тільки невелика кількість прикладів. Це досягається за рахунок того, що алгоритми Meta Learning вчаться вловлювати ключову структуру набору завдань, що дозволяє моделям узагальнювати досвід для вирішення нових завдань на основі меншої кількості даних.

Meta Learning також сприяє покращенню узагальнення. Моделі, які навчені за допомогою Meta Learning, ефективно виділяють корисні ознаки або репрезентації з різноманітних завдань. Це дозволяє моделям більш точно узагальнювати та адаптуватися до нових завдань.

Автоматичний вибір моделі та гіперпараметрів також є важливою перевагою Meta Learning. Цей підхід дозволяє автоматично вибирати оптимальні архітектури моделей та параметри для конкретних завдань, що значно спрощує процес розробки та впровадження моделей.

Окрім цього, Meta Learning ефективно використовується для адаптації до різних предметних областей, навчаючись на наборі доменно-специфічних репрезентацій. Це робить його ідеальним інструментом для вивчення різноманітних доменів.

Meta Learning знаходить широке застосування у різних галузях. Наприклад, використання цього підходу в навчанні з кількома спробами дозволяє моделям швидко адаптуватися до нових завдань. У галузі комп'ютерного зору Meta Learning використовується для вирішення завдань виявлення об'єктів, класифікації зображень та семантичної сегментації. В області обробки природної мови Meta Learning застосовується для міждоменного та міжмовного узагальнення. Також Meta Learning є ключовим елементом у розробці персоналізованих рекомендаційних систем та в робототехніці, де він використовується для навчання моделей, здатних адаптуватися до нових умов.

Meta-Learning є інноваційним підходом у машинному навчанні, який спрямований на розвиток моделей, здатних швидко адаптуватися до нових завдань за допомогою мінімальної кількості даних. Цей метод дозволяє ефективно використовувати попередні знання та досвід для вирішення нових завдань, значно скорочуючи час та ресурси, необхідні для навчання.

1.3. Формалізована постановка задачі

Основною метою цього дослідження є розробка детектора об'єктів, який можна швидко налаштувати на виявлення певних об'єктів з використанням обмеженої кількості зразків. Задача передбачає побудову та оптимізацію моделі для максимізації критеріїв валідації на тестовій вибірці.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- аналіз сучасних ембедінгів та моделей машинного навчання: дослідити, як різні ембедінги, такі як DinoV2 (small, medium, large), можуть впливати на ефективність детектування об'єктів;

- формування ознакового опису: використати попередньо навчені ембедінги для створення векторів ознак, які будуть використовуватися детектором для розпізнавання об'єктів;
- розробка та налаштування моделі детектора об'єктів: побудувати модель, яка здатна адаптуватися до різних задач, таких як детектування людей, машин та літаків, використовуючи мінімальну кількість зразків для тренування;
- тестування та аналіз моделі: провести тестування моделі на різних задачах детектування, визначити її ефективність та швидкість адаптації до нових об'єктів.

Етапи виконання:

- вибір та аналіз набору даних: зібрати та проаналізувати відповідні набори даних для детектування людей, машин та літаків;
- побудова та оптимізація вхідних даних: використати техніки обробки та нормалізації даних для підготовки до тренування моделі;
- визначення критеріїв валідації: встановити метрики, за якими буде оцінюватися ефективність моделі, такі як точність, повнота, швидкість обробки;
- тестування моделі: перевірити модель на тестовій вибірці, оцінити її ефективність та можливість швидкої адаптації до нових задач.

Запропонована модель буде досліджена на важливих практичних задачах, таких як детектування людей, машин та літаків, із можливістю швидкої зміни фокусу на нові об'єкти. Це має важливе значення в багатьох галузях, від систем безпеки до аналізу зображень з дронів.

Очікується, що розроблена модель забезпечить високу точність та швидкість в детектуванні об'єктів, а також демонструватиме гнучкість в адаптації до нових видів об'єктів. Важливим аспектом є також можливість використання моделі в реальному часі з обмеженими обчислювальними ресурсами.

2. ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ІНТЕЛЕКТУАЛЬНОГО ДЕТЕКТОРА ОБ'ЄКТІВ

2.1. Модель детектора об'єктів

Сьогоднішній швидкий розвиток технологій та зростаюча потреба в автоматизації процесів зумовлюють необхідність створення моделей, що можуть швидко адаптуватися до нових умов та завдань без значних витрат ресурсів. Це особливо важливо в контексті обмежених обчислювальних можливостей та необхідності оптимізації енергоспоживання.

Такий підхід включає застосування моделі детектора об'єктів, яка використовує принципи глибокого навчання та аналітики даних для виявлення та класифікації об'єктів у різних середовищах. Особливість цієї моделі полягає в її здатності до швидкої адаптації під змінювані умови та нові типи даних, що робить її ідеальною для використання в умовах, де є обмеження ресурсів або потрібна висока швидкість реагування.

Модель використовує глибоку нейронну мережу (CNN) для аналізу зображень. CNN ефективно виявляють та класифікують об'єкти на зображеннях за допомогою шарів, що включають фільтри для екстракції особливостей. Конволюційні шари перетворюють вхідні дані (зображення) на вищі рівні абстракції, виділяючи характеристики, такі як краї, форми тощо.

Модуль Виявлення Об'єктів (Object Detection Module) забезпечує локалізацію об'єктів у зображенні та класифікацію їх категорій та використовується для ідентифікації областей зображення (ROI), які містять об'єкти. ROI визначаються на основі виділених конволюційними шарами характеристик та піддаються подальшому аналізу та класифікації, використовуючи повністю з'єднані шари.

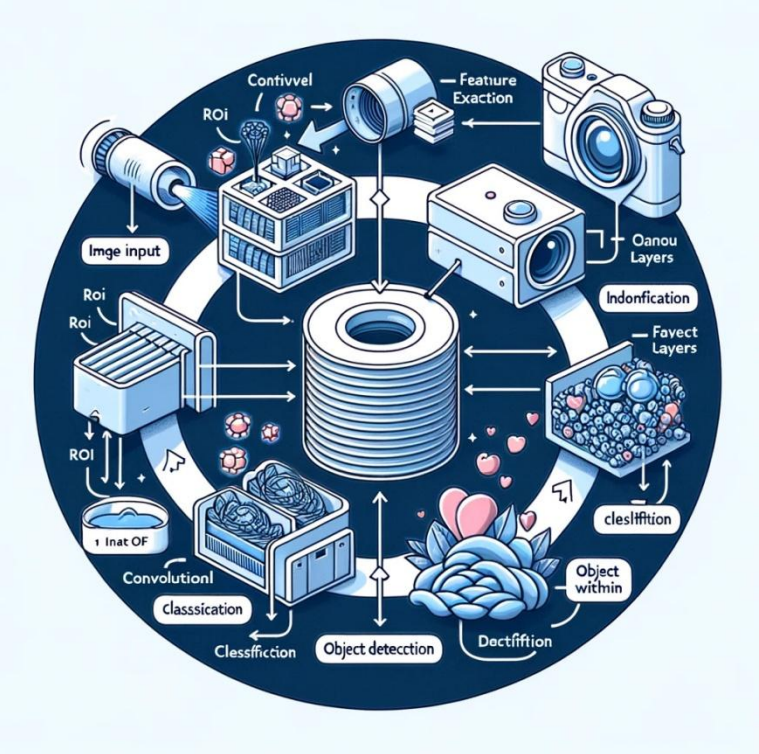

Рисунок 2.1 – Діаграма, яка ілюструє обробку областей інтересу (ROI) у виявленні об'єктів.

Конфігураційні Файли використовуються для налаштування гіперпараметрів моделі, включаючи кількість шарів, розмір фільтрів тощо. Ключові параметри, такі як CLASS_PROTOTYPES та BG_PROTOTYPES, дозволяють моделі адаптуватися до різноманітних сценаріїв детекції. Ці параметри визначають, які класи об'єктів будуть визначатися та як вони будуть оброблятися моделлю. CLASS_PROTOTYPES: Визначає набір прототипів для класів об'єктів, що виявляються. Це дозволяє моделі точно розпізнавати специфічні об'єкти залежно від задачі.

BG_PROTOTYPES: Визначає фонові прототипи, які використовуються для вдосконалення точності виявлення об'єктів на складних фонових умовах.

Функція втрат вимірює розбіжності між прогнозами моделі та фактичними цільовими значеннями. Вона надає числове представлення того, наскільки "добре" чи "погано" модель виконує свою задачу.

Модель використовує глибоке навчання для вдосконалення процесів виявлення та класифікації об'єктів. Алгоритми глибокого навчання забезпечують високу точність, здатність до самонавчання та адаптації. DINO V2 (Knowledge Distillation with No Labels) є методом самонавчання, який використовується в глибокому навчанні, зокрема у контексті візуального розпізнавання.

DINO сприяє поліпшенню детекції об'єктів, навіть за умов обмежених ресурсів, дозволяючи моделі ефективно аналізувати великі набори немаркованих зображень. Це підвищує здатність моделі адаптуватися до нових умов або об'єктів, які вона раніше не зустрічала.

Модель може швидко адаптуватися до нових типів даних, що важливо для швидкої адаптації детектора об'єктів. Економія ресурсів: Зменшення потреби в маркованих даних знижує час та витрати на підготовку даних.

У поєднанні з DINO, Vision Transformer (ViT) забезпечує інноваційний підхід до обробки зображень, який базується на трансформерах. ViT ефективно обробляє зображення, розбиваючи їх на набір патчів і обробляючи ці патчі аналогічно до слів у тексті. Інтеграція DINO та ViT сприяє автономному навчанню моделі, оптимізуючі аналіз немаркованих даних, що зменшує залежність від великих наборів маркованих даних. Це забезпечує моделі гнучкість у виявленні нових або незнайомих об'єктів, які можуть з'явитися в різних умовах. ViT перетворює зображення на серію патчів для більш детального аналізу, подібно до того, як трансформери обробляють слова в тексті.

Це покращує здатність моделі розпізнавати об'єкти в складних зображеннях, навіть при змінених умовах освітлення чи перспективи. Інтеграція DINO і ViT дає моделі можливість швидко адаптуватися до нових сценаріїв і об'єктів без необхідності додаткового маркування даних.

Поєднання цих двох технологій підвищує точність виявлення об'єктів, забезпечуючи при цьому швидку обробку даних.

2.2. Алгоритм навчання і налаштування під задачу

У сфері розробки інформаційних технологій, особливо в контексті швидкої адаптації до нових задач, ключовим аспектом є гнучкість та ефективність алгоритмів навчання. Це стає ще більш актуальним, коли розглядати область детекції об'єктів у різноманітних умовах, зокрема за умов ресурсних обмежень. Виклик полягає в тому, щоб розробити модель, яка не тільки ефективно виконує свої основні функції, але й спроможна швидко адаптуватися до нових умов або задач без значного збільшення обчислювальних витрат.

Адаптація детектора об'єктів до нових задач включає ряд етапів, починаючи від первинного тренування моделі з використанням стандартних датасетів та конфігурацій, і закінчуючи фінтюнінгом (тонким налаштуванням) під специфічні умови. Особливу увагу при цьому слід приділити методам few-shot learning, які дозволяють моделі ефективно навчатися на обмежених даних.

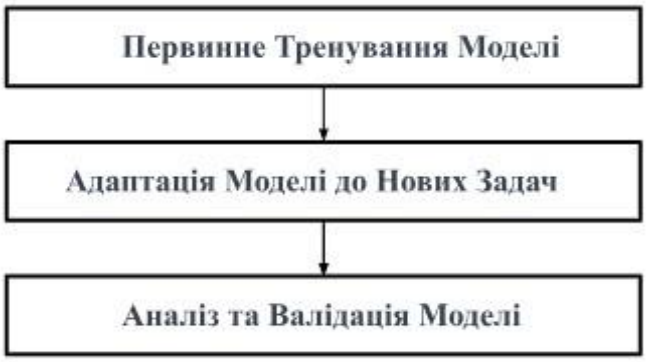

Рисунок 2.2 – Схема етапів алгоритму навчання та налаштування моделі детектора об'єктів

В першу чергу виконується первинне тренування моделі.

На цьому етапі ключовим завданням є налаштування конфігурації параметрів моделі. У конфігураційному файлі необхідно вказати основні параметри моделі, такі як тип архітектури мережі (build_dino_v2_vit), що вказує на використання архітектури DINO V2 з Vision Transformer, і розміри моделі для навчання (large, base, small).

Також важливо визначити шляхи до файлів прототипів для категорій об'єктів, що допоможуть моделі ефективно тренуватися на різних типах об'єктів, та до файлу прототипів для фонових об'єктів, що покращує відділення об'єктів від складних фонів.

Не менш важливим є вибір та підготовка спеціалізованих наборів даних, які містять зображення базових категорій для тренування та тестування. Це забезпечує первинну основу для навчання моделі. Анотація даних грає ключову роль, оскільки вона безпосередньо впливає на якість та точність моделі.

Далі виконується запуск процесу навчання, де модель використовує надані дані для вивчення характеристик об'єктів та їх класифікації. Встановлення гіперпараметрів, таких як кількість ітерацій, швидкість навчання та розмір пакету, є важливою частиною цього процесу.

Процес первинного тренування є фундаментальним для подальшої адаптації моделі до нових задач та умов.

Переваги первинного тренування моделі:

- ефективність навчання;
- гнучкість та адаптивність;
- зменшення перенавчання;
- покращення точності та надійності.

Недоліки:

- залежність від якості даних;
- високі вимоги до анотації;

- витрати на перевірку та валідацію.

Далі іде адаптація моделі до нових задач. На цьому етапі

важливо включити в датасет зображення, що відповідають новим категоріям об'єктів. При цьому важливо, щоб нові дані були правильно анотовані та відповідали формату, який може обробляти модель. Це гарантує, що модель зможе ефективно "вчитися" на цих даних.

Необхідно внести зміни в конфігураційні файли, щоб включити шляхи до нових файлів прототипів. Це дозволить моделі використовувати методи few-shot learning для ефективного навчання на обмеженому наборі даних, особливо важливо при адаптації до нових категорій об'єктів.

Після чого необхідно запустити процес навчання моделі з оновленим набором даних, який тепер включає нові категорії об'єктів. Цей крок необхідний для того, щоб модель могла адаптуватися та навчитися розпізнавати нові об'єкти. В процесі навчання важливо проводити моніторинг, щоб переконатися, що модель правильно адаптується до нових даних. Це включає в себе відстеження точності моделі, її здатності правильно класифікувати нові об'єкти, та визначення необхідності подальших корекцій у процесі навчання.

Переваги адаптації моделі до нових задач:

- гнучкість;
- ефективність;
- економія ресурсів;
- швидкість реагування.

Недоліки:

- ризик перенавчання;
- потреба у спеціалізованих даних;
- технічна складність;
- ризик зменшення загальної точності.

Після навчання необхідно виконати аналіз результатів навчання. Аналіз результатів навчання на тестових даних є фундаментальним для оцінки точності та надійності моделі. Це дозволяє зрозуміти, наскільки ефективно модель справляється зі своїми завданнями у реальних умовах.

Застосування різних метрик, таких як точність виявлення, середня точність по класу (mAP), час реагування, є ключовим для всебічного аналізу ефективності моделі. Це дозволяє оцінити різні аспекти продуктивності.

Оцінювання демонструє, наскільки добре модель виконує свої завдання в різних сценаріях, що є важливим для її практичного застосування. Процес оцінювання допомагає виявити слабкі місця в моделі, що можуть вимагати подальшого вдосконалення або корекції. Оцінювання є критично важливим для переконання в тому, що модель готова до використання у реальних умовах, гарантуючи її високу якість та надійність.

2.3. Критерії ефективності детектора об'єктів

Здатність точно і швидко ідентифікувати об'єкти в різноманітних умовах є критично важливою. Для цього розроблені спеціальні критерії та метрики, які допомагають оцінити різні аспекти продуктивності детектора об'єктів.

Точність Виявлення (Detection Accuracy): Цей показник оцінює, наскільки добре модель може ідентифікувати та правильно локалізувати об'єкти на зображенні. Вона вимірює відповідність між прогнозованими областями та фактичними областями об'єктів. Точність визначається як співвідношення кількості правильно виявлених об'єктів до загальної кількості об'єктів (як виявлених, так і пропущених).

Формула точності виявлення (2.1):

$$P = \frac{TP}{TP+FP+NP}, \quad (2.1)$$

де TP (True Positives) – кількість правильно виявлених об'єктів, FP (False Positives) – кількість об'єктів, помилково визнаних моделлю, FN (False Negatives) – кількість об'єктів, які модель не виявила.

Точність є важливим показником у розробці та оцінці детекторів об'єктів, особливо у застосуваннях, де важливо не лише виявити об'єкт, але й правильно визначити його місцеположення. Висока точність забезпечує надійність виявлення, що критично для застосувань, таких як автоматичне водіння, системи нагляду та інші вимогливі середовища. Точність може варіюватися залежно від складності зображень та різноманітності об'єктів. У складних умовах, таких як перекриття об'єктів або змінні умови освітлення, точність може знижуватися. Потрібно балансувати між зменшенням хибно позитивних і хибно негативних результатів для досягнення оптимальної точності.

Відношення Співвідношення Істинно Позитивних та Хибно Негативних Результатів (True Positive / False Negative Ratio): Цей показник фокусується на здатності моделі правильно ідентифікувати об'єкти (істинно позитивні результати) у порівнянні з випадками, коли вона неправильно пропускає об'єкти (хибно негативні результати).

Формула Відношення Співвідношення Істинно Позитивних та Хибно Негативних Результатів (2.2):

$$P = \frac{TP}{FN}, \quad (2.2)$$

де TP (True Positives) – кількість правильно ідентифікованих об'єктів, FN (False Negatives) – кількість об'єктів, які модель хибно пропустила.

Оптимальний баланс між істинно позитивними та хибно негативними результатами забезпечує високу надійність виявлення. Для критичних застосувань, таких як медична діагностика або безпека,

важливо мінімізувати хибно негативні результати, щоб не пропустити важливі об'єкти. Високе відношення TP/FN вказує на те, що модель ефективно виявляє об'єкти і рідко пропускає важливі випадки. Низьке відношення може вказувати на проблеми з чутливістю моделі, особливо у складних умовах або для рідкісних категорій об'єктів. Збільшення обсягу даних для тренування з високою якістю анотацій може допомогти зменшити кількість хибно негативних результатів. Необхідно переконатися, що набір даних для тренування та валідації моделі відображає реальні умови використання, щоб забезпечити адекватне тестування її чутливості.

Показник Якості Об'єднаної Площі (Intersection over Union, IoU): IoU оцінює перекриття між прогнозованою областю детектора об'єктів і фактичною областю об'єкта в зображенні. Цей показник важливий для оцінки точності локалізації об'єктів моделлю.

IoU розраховується як відношення площі перетину між прогнозованою і фактичною областями до об'єднаної площі цих областей.

Формула розрахунку IoU (2.3):

$$\text{IoU} = \frac{\text{Sпер}}{\text{Sоб}}, \qquad (2.3)$$

де Sпер – це частина простору, яка спільна для обох областей, а Sоб – це сума площ обох областей, віднявши площу перетину.

IoU дозволяє оцінити, наскільки добре модель здатна не тільки виявити об'єкт, але й точно визначити його місцеположення. Високе значення IoU свідчить про те, що модель точно локалізує об'єкти на зображенні. Аналізуючи IoU для різних об'єктів, можна визначити, які типи об'єктів або сценарії викликають проблеми в локалізації. Це допомагає у вдосконаленні моделі, зокрема в покращенні її здатності до точної локалізації.

IoU не враховує інші важливі аспекти виявлення об'єктів, такі як розпізнавання класу об'єкта. Також IoU може бути високим для великих об'єктів, але не так ефективним для малих або дуже складних об'єктів.

Середня Точність по Класу (Mean Average Precision, mAP): mAP вимірює середню точність виявлення об'єктів по всіх класах, які модель може виявляти. Ця метрика є стандартом у оцінці детекторів об'єктів і часто використовується в наукових дослідженнях та порівняльних аналізах. Для кожного класу обчислюється середня точність (Average Precision, AP), яка враховує істинно позитивні та хибно позитивні результати.

Формула AP для одного класу (2.4):

$$AP = \frac{\sum(TP/(TP+FP))}{K}, \quad (2.4)$$

де TP – істинно позитивні результати, FP – хибно позитивні результати, K – кількість об’єктів в класі.

mAP розраховується як середнє значення AP по всім класам. mAP враховує здатність моделі точно виявляти та класифікувати об'єкти у всіх класах, забезпечуючи загальну оцінку її продуктивності. Ця метрика важлива для оцінки загальної ефективності моделі в реальних умовах, де об'єкти можуть належати до різних класів. За допомогою mAP можна ідентифікувати слабкі місця моделі, наприклад, класи, в яких модель показує низьку точність. Вдосконалення моделі з фокусом на класах з низьким AP може підвищити загальну mAP. Хоча mAP є корисною метрикою, вона не враховує деякі аспекти, такі як локалізація об'єктів та їх взаємодія з середовищем. В реальних умовах важливо враховувати також інші фактори, як час реагування моделі, її адаптивність до нових умов та здатність працювати з різноманітними даними.

Час реагування детектора об'єктів відіграє ключову роль в багатьох застосуваннях, особливо де потрібна швидка відповідь, наприклад, в

системах автономного водіння, відеоспостереженні в реальному часі, робототехніці та ін. Це вказує на те, наскільки швидко модель може аналізувати зображення та виявляти об'єкти на них. Час реагування зазвичай вимірюється в мілісекундах або секундах і включає час, необхідний для обробки зображення та ідентифікації об'єктів. Важливо враховувати всі етапи обробки, від моменту отримання зображення до видачі результатів. Чим швидше модель здатна реагувати, тим ефективніше вона може бути використана в динамічних або критичних ситуаціях. Наприклад, в системах нагляду за безпекою, швидке виявлення може допомогти вчасно реагувати на потенційні загрози.

Оптимізація часу реагування може включати покращення алгоритмів обробки, зменшення складності моделі або використання більш потужного обладнання для обчислень. Важливо знайти баланс між швидкістю та точністю: дуже швидка модель може втратити в точності, а дуже точна може бути надто повільною для деяких застосувань.

Зменшення часу реагування може бути складним завданням, особливо при роботі з великими або складними зображеннями. Також важливо забезпечити, що швидкість не впливає негативно на інші аспекти моделі, такі як здатність до адаптації або виявлення різних типів об'єктів.

Ці критерії допомагають оцінити ефективність детектора об'єктів не лише за точністю, але й за швидкістю, надійністю та здатністю до адаптації до різноманітних умов. Вони необхідні для оцінки загальної продуктивності моделі та її придатності для конкретних застосувань.

3.РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЕТЕКТОРА ОБ'ЄКТІВ

3.1.Опис навчальних та тестових даних

Навчання та тестування буде проводитися на варіантах набору даних COCO-2014 (Common Objects in Context). Цей набір даних широко використовується у сфері комп'ютерного зору для завдань детекції, сегментації та категоризації об'єктів.

Набір даних COCO-2014 включає в себе 80 різних категорій об'єктів: person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, couch, potted plant, bed, dining table, toilet, tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush.

Для базового навчання та тестування моделі були обрані класи "car" та "person". Ці категорії були вибрані через їхнє широке поширення та важливість у реальному світі, що робить їх ідеальними для первинного оцінювання ефективності моделі.

Для адаптивного тестування, до вищезазначених категорій буде додано клас "airplane", щоб оцінити здатність моделі адаптуватися до нових видів об'єктів, які не були представлені під час первинного навчання.

Датасет складається з файлів зображень та файлу анотацій. Файл анотацій COCO-2014 являє собою JSON-файл, який містить детальні

анотації до зображень, що входять у набір даних COCO та складається з наступних ключових розділів:

- Info - цей розділ містить метадані про набір даних (Таблиця 3.1).
- Images - список зображень, які входять у набір даних (Таблиця 3.2).
- Annotations - анотації до об'єктів на зображеннях (Таблиця 3.3).
- Categories - список категорій об'єктів у наборі даних (Таблиця 3.4).
- Licenses - інформація про ліцензії для зображень у наборі даних (Таблиця 3.5).

Таблиця 3.1 – Елементи розділу Info

Параметр	**Визначення**
year	рік створення набору даних
version	версія набору даних
description	короткий опис набору даних
contributor	організації або особи, які внесли вклад у створення набору даних
url	url-адреса, де можна знайти додаткову інформацію про набір даних
date_created	дата створення набору даних

Таблиця 3.2 – Елементи розділу Images

Параметр	**Визначення**
id	унікальний ідентифікатор зображення
width та height	ширина та висота зображення
file_name	назва файлу зображення
license	ідентифікатор ліцензії

Параметр	**Визначення**
coco_url	url для доступу до зображення в наборі даних COCO
date_captured	дата, коли було захоплено зображення

Таблиця 3.3 – Елементи розділу Annotations

Параметр	**Визначення**
id	унікальний ідентифікатор анотації
image_id	ідентифікатор зображення, до якого належить анотація
category_id	ідентифікатор категорії об'єкта
segmentation	контур об'єкта для сегментації
area	площа об'єкта
bbox	обмежувальний прямокутник навколо об'єкта
iscrowd	прапорець, що вказує, чи є об'єкт частиною групи об'єктів

Таблиця 3.4 – Елементи розділу Categories

Параметр	**Визначення**
id	унікальний ідентифікатор категорії
name	назва категорії
supercategory	назва "суперкатегорії", до якої належить дана категорія

Таблиця 3.5 – Елементи розділу Licenses

Параметр	**Визначення**
id	ідентифікатор ліцензії
url	посилання на текст ліцензії
name	назва ліцензії

3.2. Короткий опис програмного забезпечення

Для виконання завдання, яке передбачає розробку інформаційної технології швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень, була вибрана об’єктно-орієнтована мова програмування Python.

Python - це високорівнева, інтерпретована мова програмування з динамічною типізацією та автоматичним керуванням пам'яттю. Вона відома своєю читабельністю та чистим синтаксисом, що робить її зручною для навчання програмуванню та розвитку складних проектів. Python підтримує різні парадигми програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування. Широко використовується в наукових дослідженнях, аналізі даних, штучному інтелекті, веб-розробці та автоматизації. Python має велику та активну спільноту, яка регулярно вносить вклад у велику кількість бібліотек та інструментів, що робить його однією з найбільш універсальних мов програмування. Завдяки своїй гнучкості та широкому спектру застосувань, Python є однією з найпопулярніших мов програмування у світі.

NumPy — це відкрита бібліотека для мови програмування Python, яка дозволяє здійснювати високопродуктивні операції з багатовимірними масивами та матрицями. Вона надає великий набір математичних функцій,

які допомагають виконувати різні види наукових та інженерних обчислень. Однією з ключових особливостей NumPy є ефективність, завдяки використанню оптимізованих C API, що робить її незамінною для обробки великих обсягів даних. NumPy широко використовується у наукових дослідженнях, особливо в областях, що вимагають складних обчислень, таких як лінійна алгебра, статистика, та інші математичні операції.

PyTorch - це бібліотека машинного навчання з відкритим вихідним кодом, відома своєю гнучкістю, простотою використання та потужною підтримкою глибокого навчання. Вона пропонує динамічні графіки обчислень, які дозволяють більш інтуїтивно зрозуміле моделювання та налагодження. PyTorch широко використовується як в академічних колах, так і в промисловості для вирішення різноманітних завдань машинного навчання.

Torchvision - це пакет в екосистемі PyTorch, який надає популярні набори даних, архітектури моделей та загальні перетворення зображень для комп'ютерного зору. Він спрощує процес впровадження та навчання моделей комп'ютерного зору, надаючи попередньо реалізовані компоненти та утиліти.

Tqdm - це бібліотека Python, яка забезпечує швидкий, розширюваний індикатор виконання для циклів та інших ітеративних процесів. Її можна використовувати в широкому діапазоні додатків, від простих скриптів до складних моделей машинного навчання, для візуального відображення прогресу.

DE-ViT - це програмна система для розпізнавання об'єктів, яка класифікує об'єкти, використовуючи вихідні зображення. Ця система забезпечує гнучкість у навчанні та оцінюванні, підтримуючи різноманітні конфігурації та набори даних, що робить її багатофункціональним інструментом у сфері комп'ютерного бачення.

FiftyOne - це інструмент з відкритим вихідним кодом для створення високоякісних наборів даних і моделей комп'ютерного зору. Він надає потужні інструменти візуалізації та аналізу наборів зображень і відео, що полегшує фахівцям з машинного навчання покращувати якість своїх наборів даних і продуктивність моделей.

Detectron2 - це програмна система, розроблена Facebook AI Research для виявлення та сегментації об'єктів. Вона реалізує найсучасніші алгоритми для цих завдань і базується на фреймворку PyTorch.Detectron2 відомий своєю високою продуктивністю та гнучкістю, що робить його популярним вибором для дослідників та розробників у галузі комп'ютерного зору.

Використання Python та допоміжних бібліотек є оптимальним вибором для розробки інформаційної технології швидкої адаптації детектора об'єктів. Це забезпечує необхідну гнучкість, потужність та простоту в реалізації складних алгоритмів у сфері штучного інтелекту та комп'ютерного зору. Обрані інструменти дозволяють ефективно обробляти великі обсяги даних та швидко адаптувати системи до нових задач.

3.3. Аналіз результатів експериментів

Спочатку необхідно виконати первинне тренування моделі з категоріями car та person.

Відношення Хибно Негативних Результатів (False Negative Ratio) зазнало коливань, починаючи з 0.09 і зростаючи до 0.15, потім знову знижуючись і стабілізуючись на рівні близько 0.11. Це свідчить про те, що модель має певні труднощі у виявленні усіх реальних об'єктів, що призводить до змінних показників хибно негативних результатів. Точність Виявлення (Detection Accuracy) показує загалом позитивну динаміку, зростаючи з 0.12 до 0.84. Це вказує на здатність моделі покращувати свою точність у виявленні та класифікації об'єктів з часом. Середня Точність

(AP) для категорії "car" зросла від 6.27 до 15.55, хоча й мала деякі коливання. Загалом, це свідчить про поліпшення здатності моделі ідентифікувати та класифікувати автомобілі.

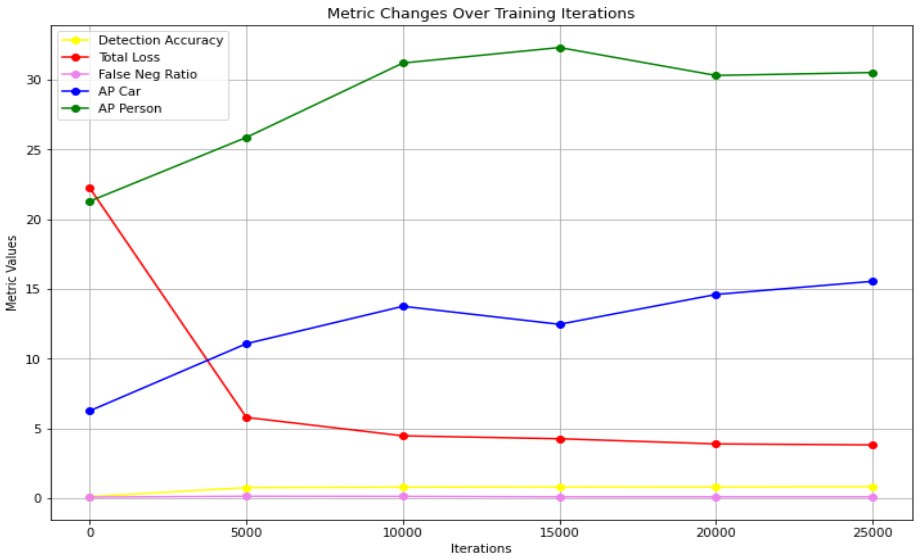

Рисунок 3.1 – Графік ефективності первинного тренування "small" моделі

Середня Точність (AP) для категорії "person" зросла з 21.29 до 32.30, після чого знизилася до 30.30 і злегка зросла до 30.51. Це може свідчити про те, що модель краще виявляє людей, але може мати проблеми з підтримкою постійної високої точності. Загальні Втрати (Total Loss) зменшилися з 22.23 до 3.83, що свідчить про покращення загальної ефективності та стабільності моделі.

При виконанні оцінювання моделі показники точності (AP) по категорії "person" склали 39.15, що свідчить про високий рівень точності виявлення та класифікації людських фігур у різних позах та середовищах, а для категорії "car" AP склали 15.63, що нижче, ніж у категорії "person". Це може бути пов'язано з різними факторами, такими як перекриття, різноманітність дизайнів автомобілів та умови навколишнього середовища.

Модель показує сильну продуктивність, особливо у виявленні категорій "person", що є важливим для застосувань, таких як відеоспостереження, автономне водіння та аналіз натовпу.

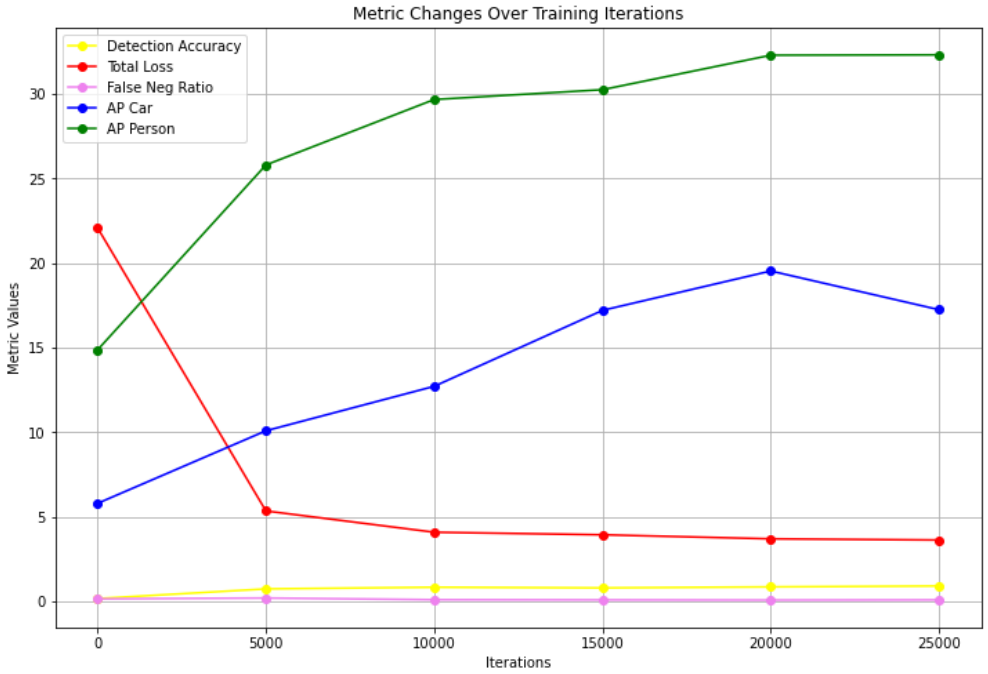

Рисунок 3.2 – Графік ефективності первинного тренування "medium" моделі

Відношення Хибно Негативних Результатів (False Negative Ratio) були відносно високими (0.14 та 0.19), але згодом стабілізувалися на більш низькому рівні (від 0.07 до 0.09). Це свідчить про те, що модель з часом стала краще виявляти об'єкти, зменшуючи кількість хибно негативних результатів. Точність Виявлення (Detection Accuracy)з часом значно покращилася, починаючи з низького рівня 0.16 і досягаючи 0.90 на останній ітерації. Це показує значний прогрес у точності моделі при ідентифікації та локалізації об'єктів. Середня Точність (AP) для категорії "car" зросла з 5.79 до 19.53, після чого злегка знизилась до 17.24. Загальна тенденція до покращення AP для цієї категорії свідчить про покращення здатності моделі точно класифікувати та локалізувати автомобілі. Середня Точність (AP) для категорії "person" зросла стабільно з 14.87 до 32.31, показуючи високу ефективність моделі у виявленні та класифікації людей.

Загальні Втрати (Total Loss) постійно знижувались протягом усіх ітерацій, з 22.07 на початку до 3.62 на кінці, що свідчить про покращення загальної продуктивності та надійності моделі.

При виконанні оцінювання моделі показники точності (AP) по категорії "person" становили 44.79, що є високим показником і вказує на ефективність моделі у виявленні людей, а для категорії "car" склало 25.13, що є нижчим порівняно з категорією "person".

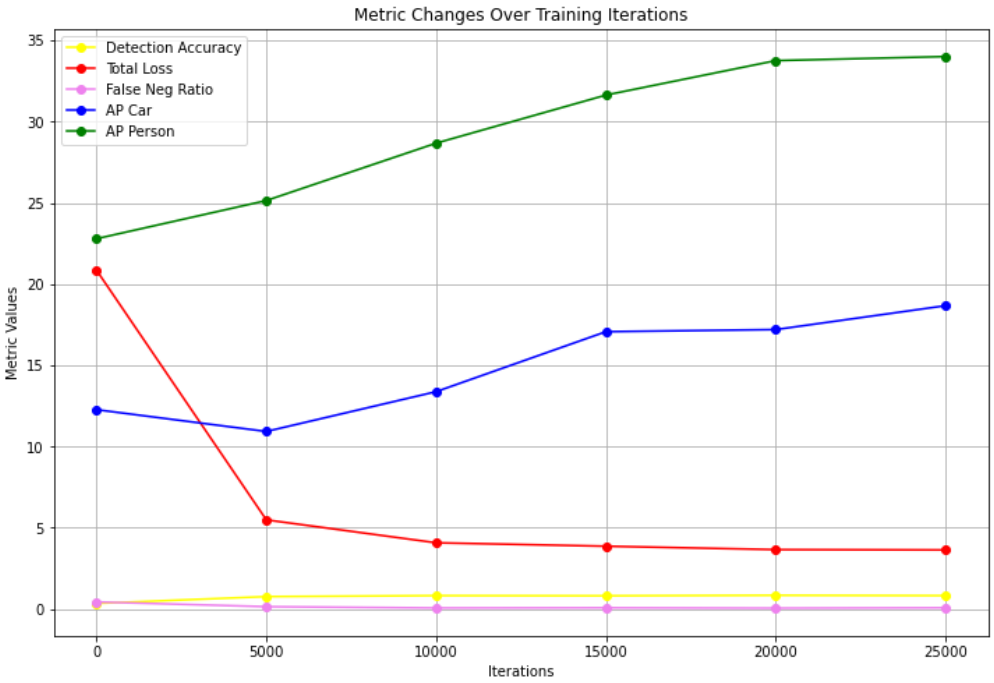

Рисунок 3.3 – Графік ефективності первинного тренування "large" моделі

Початковий показник Відношення Хибно Негативних Результатів (False Negative Ratio) був досить високий (0.44), що свідчило про значну кількість хибно негативних результатів. Проте, з часом він знизився та стабілізувався на рівні близько 0.08, що свідчить про покращення здатності моделі виявляти об'єкти. Точність Виявлення (Detection Accuracy) також покращилася, починаючи з 0.35 і зростаючи до 0.83. Це вказує на збільшення здатності моделі правильно класифікувати об'єкти. Середня Точність (AP) для категорії "car" відзначається зростанням із 12.27 до 18.67. Це демонструє, що модель стає більш точною у виявленні та класифікації автомобілів з плином часу.

Середня Точність (AP) для категорії "person" показує постійне зростання з 22.79 до 34.00. Це свідчить про значне покращення моделі у виявленні та класифікації людей. Загальні Втрати (Total Loss) зменшились від 20.85 до 3.64, що свідчить про значне покращення ефективності та стабільності моделі.

При виконанні оцінювання моделі показники точності (AP) по категорії "person" становили 46.08, що є високим показником і вказує на ефективність моделі у виявленні людей, а для категорії "car" склало 27.40. Модель добре справляється з виявленням людей, маючи вищий AP у порівнянні з виявленням автомобілів.

Після виконання первинного тренування моделі необхідно виконати тренування моделі з використанням few-shot learning.

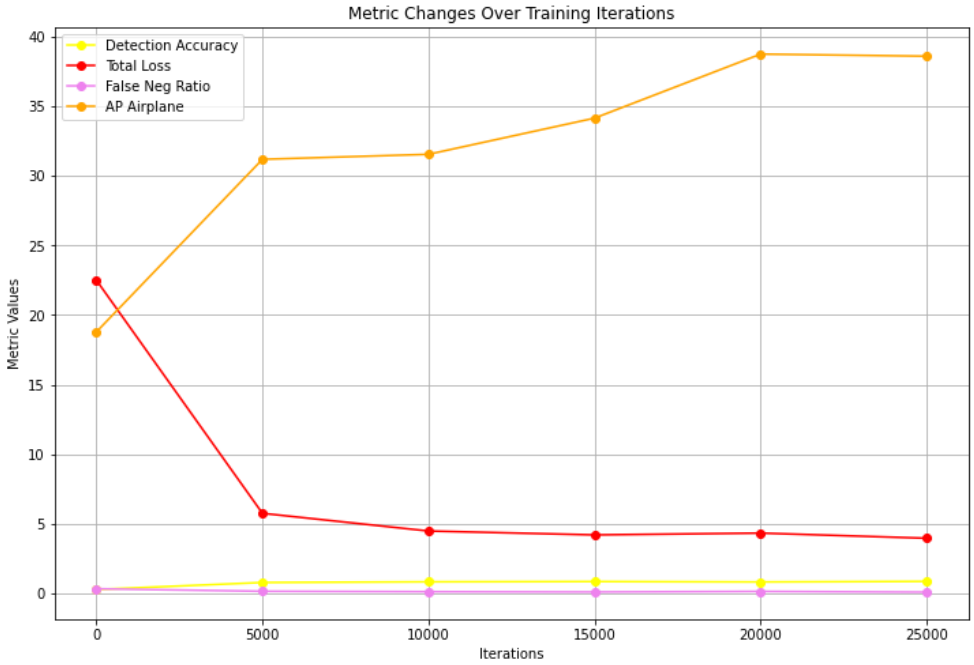

Рисунок 3.4 – Графік ефективності few-shot learning "small" моделі

Відношення Хибно Негативних Результатів (False Negative Ratio) знизилось з 0.33 до 0.09, вказуючи на зменшення кількості випадків, коли модель помилково пропускала об'єкти. Однак, спостерігається невелике збільшення хибно негативних результатів у пізніх ітераціях. Точність Виявлення (Detection Accuracy) покращився, починаючи з 0.28 у перших

ітераціях та досягаючи 0.86 у останніх. Це свідчить про здатність моделі до ефективного ідентифікування та класифікації об'єктів з плином часу. Середня Точність (AP) для категорії для категорії "airplane" збільшився з 18.78 до 38.59, демонструючи значне поліпшення здатності моделі точно виявляти та класифікувати об'єкти цього типу. У Загальних Втрат (Total Loss) спостерігається зниження загальних втрат з 22.50 до 3.96, що є ознакою поліпшення загальної ефективності моделі.

При виконанні оцінювання моделі few-shot learning 5 shots small показник точності (AP) по категорії "airplane" становить 32.42, що є помірним показником. Модель має кращу продуктивність на великих (APl - 33.13) та середніх (APm - 24.16) об'єктах порівняно з малими (APs - 8.97).

При виконанні оцінювання моделі few-shot learning 30 shots small показник точності (AP) по категорії "airplane" становить 39.13, що є високим показником і вказує на ефективність моделі у виявленні літаків. Модель має кращу продуктивність на великих (APl - 36.47) та середніх (APm - 26.45) об'єктах порівняно з малими (APs - 9.60).

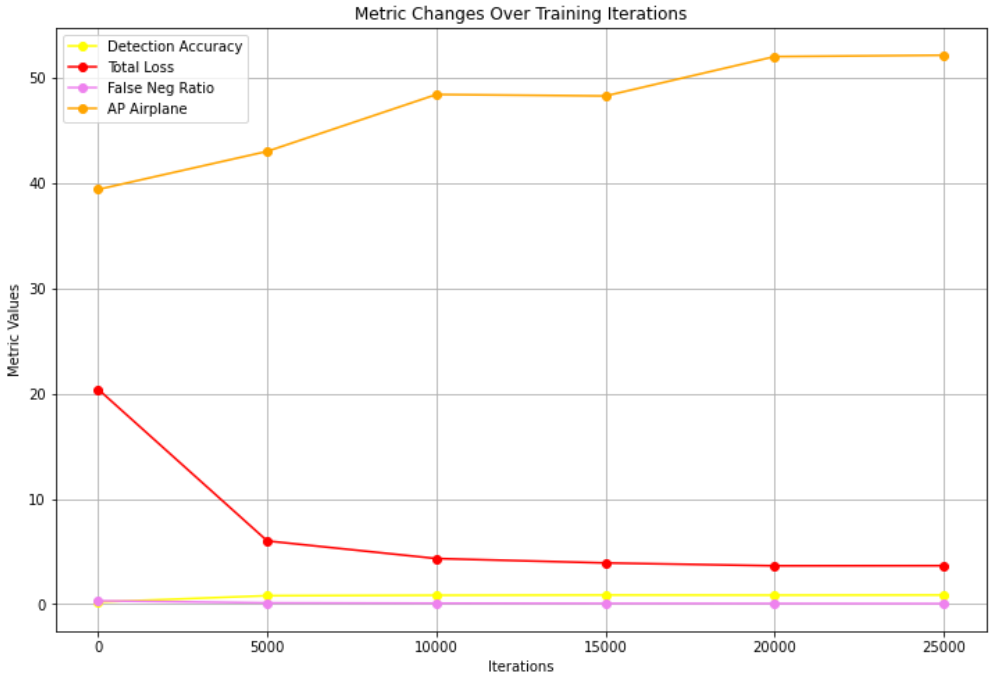

Рисунок 3.5 – Графік ефективності few-shot learning "medium" моделі

Відношення Хибно Негативних Результатів (False Negative Ratio) почалося з високого значення 0.33, але значно знизилось до 0.06 протягом тестування. Це свідчить про значне зменшення кількості випадків, коли модель пропускає наявні об'єкти, покращуючи її здатність до виявлення об'єктів. Точність Виявлення (Detection Accuracy) була досить низькою (0.21), але вона значно покращилася, досягнувши рівня 0.87. Це підтверджує, що модель стала набагато більш ефективною у правильній класифікації та ідентифікації об'єктів. Середня Точність (AP) для категорії "airplane" постійно зростала протягом усіх ітерацій, починаючи з 39.40 і досягають 52.15. Це свідчить про значне покращення здатності моделі точно виявляти та класифікувати літаки. Загальні Втрати (Total Loss) знизилися з 20.42 до 3.66, що свідчить про значне покращення загальної ефективності моделі. Зменшення втрат є показником того, що модель стає більш точною та стабільною.

При виконанні оцінювання моделі few-shot learning 5 shots medium показник точності (AP) по категорії "airplane" становить 42.25, що є високим показником і вказує на ефективність моделі у виявленні літаків. Модель має кращу продуктивність на великих (APl - 36.80) та середніх (APm - 28.51) об'єктах порівняно з малими (APs - 11.68).

При виконанні оцінювання моделі few-shot learning 30 shots medium показник точності (AP) по категорії "airplane" становить 53.42, що є дуже високим показником і вказує на ефективність моделі у виявленні літаків. Модель має кращу продуктивність на великих (APl - 42.40) та середніх (APm - 31.86) об'єктах порівняно з малими (APs - 13.36).

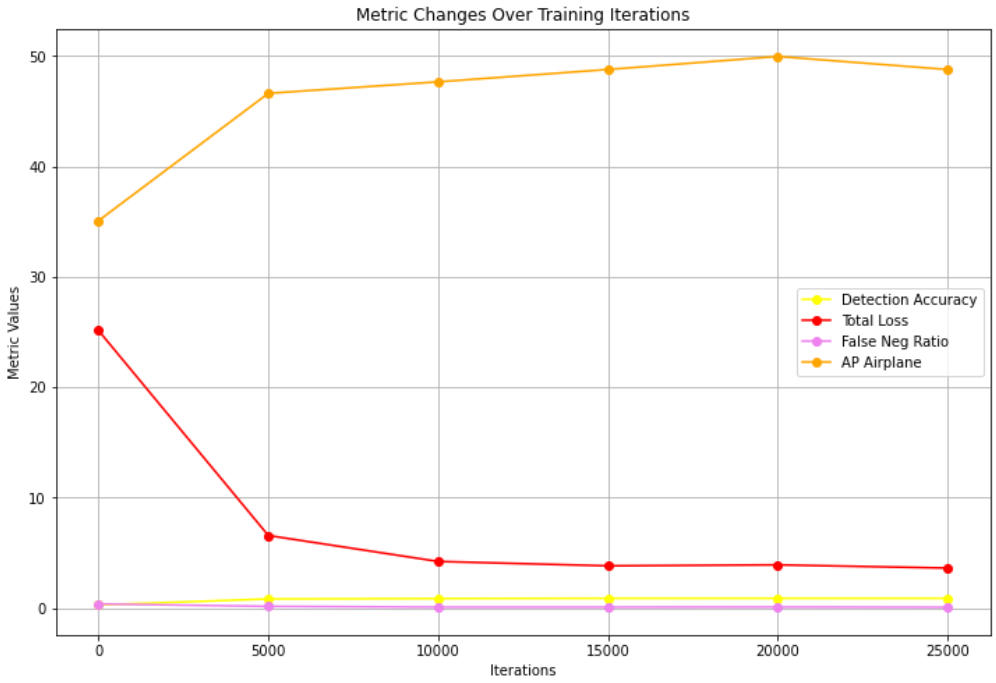

Рисунок 3.6 – Графік ефективності few-shot learning "large" моделі

Відношення Хибно Негативних Результатів (False Negative Ratio) покращилося, знижуючись з 0.33 до 0.06, що свідчить про зменшення випадків, коли модель неправильно пропускала об'єкти. Точність Виявлення (Detection Accuracy) зазнала значного покращення, зростаючи з 0.21 до 0.87. Це свідчить про ефективність моделі у правильному ідентифікуванні та локалізації об'єктів. Середня Точність (AP) для категорії "airplane" зросла з 39.40 до 52.15. Це показує високу ефективність моделі у виявленні та класифікації об'єктів цієї категорії. Загальні Втрати (Total Loss) помітно знизилися з 20.42 до 3.66, що є показником покращення загальної продуктивності моделі.

При виконанні оцінювання моделі few-shot learning 5 shots large показник точності (AP) по категорії "airplane" становить 45.49, що є високим показником і вказує на ефективність моделі у виявленні літаків. Модель має кращу продуктивність на великих (APl - 40.50) та середніх (APm - 31.20) об'єктах порівняно з малими (APs - 13.90).

При виконанні оцінювання моделі few-shot learning 30 shots large показник точності (AP) по категорії "airplane" становить 50.32, що є дуже

високим показником і вказує на ефективність моделі у виявленні літаків. Модель має кращу продуктивність на великих (APl - 43.36) та середніх (APm - 33.17) об'єктах порівняно з малими (APs - 14.48).

Моделі різного розміру: "small", "medium", та "large" моделі показали різні рівні ефективності. Моделі більшого розміру мали тенденцію до кращих результатів виявлення та класифікації об'єктів. Покращення у виявленні категорії "airplane" під час few-shot learning свідчить про здатність моделі ефективно навчатися на обмеженій кількості даних. Моделі показали кращі результати при використанні більшої кількості прикладів (30 shots порівняно з 5 shots).

ВИСНОВКИ

В магістерській кваліфікаційній роботі було проведено аналіз проблеми швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень. Виконано огляд відомих рішень в галузі машинного навчання та обрано напрямок перспективних досліджень.

Розроблено модель та методику навчання для детекторів об'єктів, що базується на принципах Few-Shot Learning та DINO V2 (DIstillation with NO labels). Використання цих методів дозволяє досягти адаптації систем детекції до нових завдань, не зважаючи обмеженний обсяг розмічених даних. Результати моделювання підтверджують працездатність розроблених алгоритмів, проте коливання у середній точності для певних категорій та в хибно негативних результатах свідчать про необхідність подальшої оптимізації та налаштування моделі для забезпечення більш стабільної продуктивності.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Litjens, G., et al., 2017. A Survey on Deep Learning in Medical Image Analysis. [online] Available at: <https://www.researchgate.net/publication/313857891_A_Survey_on_Deep_Learning_in_Medical_Image_Analysis> [Accessed 10 November 2023].

2. Janai, J., Güney, F., Behl, A. & Geiger, A., 2020. Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art. [online] Available at: <https://www.researchgate.net/publication/316270100_Computer_Vision_for_Autonomous_Vehicles_Problems_Datasets_and_State-of-the-Art> [Accessed 9 November 2023].

3. Zhang, Z., et al., 2019. Recent Trends in Intensive Computing. [online] Available at: <https://www.researchgate.net/publication/356752115_Parallel_Deep_Learning_Framework_for_Video_Surveillance_System> P. 153-162. [Accessed 10 November 2023].

4. He, K., Zhang, X., Ren, S. & Sun, J., 2016. Deep Residual Learning for Image Recognition. [online] Available at: <https://www.researchgate.net/publication/311609041_Deep_Residual_Learning_for_Image_Recognition> [Accessed 11 November 2023].

5. Hochreiter, S. & Schmidhuber, J., 1997. Long Short-Term Memory. [online] Available at: <https://www.researchgate.net/publication/13853244_Long_Short-term_Memory> [Accessed 6 November 2023].

6. Dosovitskiy, A., et al., 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. [online] Available at: <https://openreview.net/pdf?id=YicbFdNTTy> [Accessed 10 November 2023].

7. Habibi Aghdam, H. & Heravi, E. J., 2017. Guide to Convolutional Neural Networks. [online] Available at: <https://link.springer.com/book/10.1007/978-3-319-57550-6> [Accessed 24 November 2023].

8. Salem, F. M., 2022. Recurrent Neural Networks. [online] Available at: <https://link.springer.com/book/10.1007/978-3-030-89929-5> [Accessed 15 November 2023].

9. Du, K.-L. & Swamy, M.N.S., 2013. Neural Networks and Statistical Learning. [online] Available at: <https://www.researchgate.net/publication/264912450_Neural_Networks_and_Statistical_Learning> P. 337-353. [Accessed 24 November 2023].

10. Al-hammuri, K., Gebali, F., Kanan, A. & Chelvan, I. T., 2023. Vision transformer architecture and applications in digital health: a tutorial and survey. [online] Available at: <https://vciba.springeropen.com/articles/10.1186/s42492-023-00140-9> [Accessed 27 November 2023].

11. Dean, J., et al., 2012. Large Scale Distributed Deep Networks. [online] Available at: <https://www.researchgate.net/publication/266225209_Large_Scale_Distributed_Deep_Networks> [Accessed 10 November 2023].

12. Howard, A. G., et al., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. [online] Available at: <https://www.researchgate.net/publication/316184205_MobileNets_Efficient_Convolutional_Neural_Networks_for_Mobile_Vision_Applications> [Accessed 10 November 2023].

13. Elsken, T., Metzen, J. H. & Hutter, F., 2019. Neural Architecture Search: A Survey. [online] Available at: <https://www.mdpi.com/1424-8220/23/3/1713> [Accessed 10 November 2023].

14. Vinyals, O., et al., 2015. Show and Tell: A Neural Image Caption Generator. [online] Available at: <https://www.researchgate.net/publication/307747289_Show_and_tell_A_neural_image_caption_generator> [Accessed 11 November 2023].

15. Roy, A., Ghosal, D., Cambria, E., Majumder, N., Mihalcea, R. & Poria, S., 2022. Improving Zero-Shot Learning Baselines with Commonsense Knowledge. [online] Available at: <https://link.springer.com/article/10.1007/s12559-022-10044-0> [Accessed 21 November 2023].

16. Xian, Y., Lampert, C. H., Schiele, B. & Akata, Z., 2017. Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly. [online] Available at: <https://www.researchgate.net/publication/318121287_Zero-Shot_Learning__A_Comprehensive_Evaluation_of_the_Good_the_Bad_and_the_Ugly> [Accessed 21 November 2023].

17. Saad, E., Paprzycki, M. & Ganzha, M., 2022. Practical Aspects of Zero-Shot Learning. [online] Available at: <https://link.springer.com/chapter/10.1007/978-3-031-08754-7_12> [Accessed 18 November 2023].

18. Kai He, Nan Pu, Mingrui Lao & Michael S. Lew, 2023. Few-shot and meta-learning methods for image understanding: a survey. [online] Available at: <https://www.researchgate.net/publication/371953426_Few-shot_and_meta-learning_methods_for_image_understanding_a_survey> [Accessed 27 November 2023].

19. Wang, Y., Yao, Q., Kwok, J. T. & Ni, L. M., 2020. Generalizing from a Few Examples: A Survey on Few-shot Learning. [online] Available at: <https://www.researchgate.net/publication/342141918_Generalizing_from_a_Few_Examples_A_Survey_on_Few-shot_Learning> [Accessed 31 October 2023].

20. Ye, H.-J., Hu, H. & Zhan, D.-C., 2021. Learning Adaptive Classifiers Synthesis for Generalized Few-Shot Learning. [online] Available at: <https://www.researchgate.net/publication/350976596_Learning_Adaptive_Classifiers_Synthesis_for_Generalized_Few-Shot_Learning> [Accessed 10 November 2023].

21. Netzahualcoyotl Hernandez, Jens Lundström, Jesus Favela, Ian McChesney & Bert Arnrich, 2020. Literature Review on Transfer Learning for Human Activity Recognition Using Mobile and Wearable Devices with Environmental Technology. [online] Available at: <https://link.springer.com/article/10.1007/s42979-020-0070-4> [Accessed 10 November 2023].

22. Hosna, A., Merry, E., Gyalmo, J. & Alom, Z., 2022. Transfer learning: a friendly introduction. [online] Available at: <https://www.researchgate.net/publication/364641146_Transfer_learning_a_friendly_introduction> [Accessed 14 November 2023].

Увага: Посилання були оформлені станом на вказані дати доступу. Можливі зміни в доступності або URL-адресах посилань можуть мати місце після цих дат.

ДОДАТОК

Програмна реалізація швидкої адаптації детектора об'єктів до нових задач за умов ресурсних обмежень

```
task_choice="${task_choice:-ovd}"
vision_transformer="${vision_transformer:-l}"
data_source="${data_source:-coco}"
training_shot="${training_shot:-10}"
data_split="${data_split:-1}"
gpu_count="${gpu_count:-$(nvidia-smi -L | wc -l)}"
mode="${mode:-train}"

echo "Task: $task_choice, Vision Transformer: $vision_transformer, Data Source: $data_source, Training Shot: $training_shot, Data Split: $data_split, GPU Count: $gpu_count, Mode: $mode"

case $task_choice in

    ovd)
        python3 tools/train_net.py --num-gpus $gpu_count \
            $(if [ "$mode" = "eval" ]; then echo "--eval-only"; fi) \
            --config-file configs/open-vocabulary/coco/vit${vision_transformer}.yaml \
            MODEL.WEIGHTS weights/initial/open-vocabulary/vit${vision_transformer}+rpn.pth \
            DE.OFFLINE_RPN_CONFIG configs/RPN/mask_rcnn_R_50_C4_1x_ovd_FSD.yaml \
            OUTPUT_DIR output/train/open-vocabulary/coco/vit${vision_transformer}/ $@
```

```
    ;;

    fsod)
        python3 tools/train_net.py --num-gpus $gpu_count \
            $(if [ "$mode" = "eval" ]; then echo "--eval-only"; fi) \
            --config-file configs/few-
shot/vit${vision_transformer}_shot${training_shot}.yaml \
            MODEL.WEIGHTS weights/initial/few-
shot/vit${vision_transformer}+rpn.pth \
            DE.OFFLINE_RPN_CONFIG
configs/RPN/mask_rcnn_R_50_C4_1x_ovd_FSD.yaml \
            OUTPUT_DIR output/train/few-shot/shot-
${training_shot}/vit${vision_transformer}/ $@
    ;;

    *)
        echo "No action required"
        ;;
esac            MODEL.WEIGHTS weights/initial/few-
shot/vit${vision_transformer}+rpn.pth \
            DE.OFFLINE_RPN_CONFIG
configs/RPN/mask_rcnn_R_50_C4_1x_ovd_FSD.yaml \
            OUTPUT_DIR output/train/few-shot/shot-
${training_shot}/vit${vision_transformer}/ $@
    ;;
    *)
        echo "No action required"
        ;;
esac
```

```
import torch
from torch import nn
from torch.nn import functional
from torch.cuda.amp import autocast
from torchvision.ops.boxes import box_iou

import random
import numpy as np
from typing import Dict, List
from PIL import Image

from detectron2.layers.roi_align import ROIAlign
from detectron2.modeling.roi_heads.fast_rcnn import fast_rcnn_inference
from detectron2.modeling.box_regression import Box2BoxTransform
from detectron2.structures import Boxes, Instances
from detectron2.structures.masks import PolygonMasks
from detectron2.utils.events import get_event_storage

from fvcore.nn import giou_loss, smooth_l1_loss
from lib.dinov2.layers.block import Block
from lib.regionprop import augment_rois, region_coord_2_abs_coord,
abs_coord_2_region_coord
from .legasy import OpenSetDetectorWithExamples, generalized_box_iou,
interpolate, dice_loss, \
    sigmoid_ce_loss, _log_classification_stats, focal_loss, distance_embed,
box_cxcywh_to_xyxy

from .build import META_ARCH_REGISTRY
from ..backbone import Backbone, build_backbone
```

```
from ..matcher import Matcher


@META_ARCH_REGISTRY.register()
class CustomOpenSetDetector(OpenSetDetectorWithExamples):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def process_images(self, input_batches):
        with torch.no_grad():
            if any([self.offline_backbone.training,
self.offline_proposal_generator.training]):
                for model in [self.offline_backbone, self.offline_proposal_generator]:
                    model.eval()

            offline_img = self.offline_preprocess_image(input_batches)
            offline_features = self.offline_backbone(offline_img.tensor)
            offline_proposals, _ = self.offline_proposal_generator(offline_img,
offline_features, None)
            processed_img = self.preprocess_image(input_batches)

        return processed_img, offline_proposals

    def extract_patch_tokens(self, processed_img):
        with torch.no_grad():
            if self.backbone.training:
                self.backbone.eval()

            with autocast(enabled=True):
```

```
            complete_patch_tokens = self.backbone(processed_img.tensor)
            selected_patch_tokens = complete_patch_tokens[self.vit_feat_name]
            complete_patch_tokens.pop(self.vit_feat_name)

        return selected_patch_tokens, complete_patch_tokens

    def extract_tensors(self, instances, attribute_name):
        try:
            return [getattr(x, attribute_name).tensor for x in instances]
        except AttributeError:
            raise ValueError(f"Attribute {attribute_name} not found in instances")

    def process_detection_proposals(self, batched_inputs, class_count,
offline_proposals, processed_img):
            with (torch.no_grad()):
                gt_instances = [x["instances"].to(self.device) for x in batched_inputs]
                gt_boxes = self.extract_tensors(gt_instances, "gt_boxes")
                rpn_boxes = self.extract_tensors(offline_proposals, "proposal_boxes")

                if self.training:
                    noisy_boxes = self.prepare_noisy_boxes(gt_boxes,
processed_img.tensor.shape)
                    boxes = [torch.cat([gt_boxes[i].to(self.device),
noisy_boxes[i].to(self.device), rpn_boxes[i].to(self.device)]) for i in
range(len(batched_inputs))]
                else:
                    boxes = [box.to(self.device) for box in rpn_boxes]
```

```
        class_labels, matched_gt_boxes, resampled_proposals,
num_bg_samples, num_fg_samples, gt_masks = ([] for _ in range(6))

        for proposals_per_image, targets_per_image in zip(boxes,
gt_instances):
            match_quality_matrix = box_iou(
                targets_per_image.gt_boxes.tensor, proposals_per_image
            )
            matched_idxs, matched_labels =
self.proposal_matcher(match_quality_matrix)
            if len(targets_per_image.gt_classes) > 0:
                class_labels_i = targets_per_image.gt_classes[matched_idxs]
            else:
                assert torch.all(matched_labels == 0)
                class_labels_i = torch.zeros_like(matched_idxs)
            class_labels_i[matched_labels == 0] = class_count
            class_labels_i[matched_labels == -1] = -1

            if self.training or self.evaluation_shortcut:
                positive = ((class_labels_i != -1) & (class_labels_i !=
class_count)).nonzero().flatten()
                negative = (class_labels_i == class_count).nonzero().flatten()

                batch_size_per_image = self.batch_size_per_image
                num_pos = int(batch_size_per_image * self.pos_ratio)
                num_pos = min(positive.numel(), num_pos)
                num_neg = batch_size_per_image - num_pos
                num_neg = min(negative.numel(), num_neg)
```

```
                perm1 = torch.randperm(positive.numel(),
device=self.device)[:num_pos]
                perm2 =
torch.randperm(negative.numel())[:num_neg].to(self.device)
                pos_idx = positive[perm1]
                neg_idx = negative[perm2]
                sampled_idxs = torch.cat([pos_idx, neg_idx], dim=0)
            else:
                sampled_idxs = torch.arange(len(proposals_per_image),
device=self.device).long()

            proposals_per_image = proposals_per_image[sampled_idxs]
            class_labels_i = class_labels_i[sampled_idxs]

            if len(targets_per_image.gt_boxes.tensor) > 0:
                gt_boxes_i =
targets_per_image.gt_boxes.tensor[matched_idxs[sampled_idxs]]
                if self.use_mask:
                    gt_masks_i =
targets_per_image.gt_masks[matched_idxs[sampled_idxs]]
            else:
                gt_boxes_i = torch.zeros(len(sampled_idxs), 4,
device=self.device)
                if self.use_mask:
                    gt_masks_i = PolygonMasks([[np.zeros(6)], ] *
len(sampled_idxs)).to(self.device)

            resampled_proposals.append(proposals_per_image)
            class_labels.append(class_labels_i)
```

```
                matched_gt_boxes.append(gt_boxes_i)
                if self.use_mask:
                    gt_masks.append(gt_masks_i)

                num_bg_samples.append((class_labels_i ==
class_count).sum().item())
                num_fg_samples.append(class_labels_i.numel() -
num_bg_samples[-1])

            if self.training:
                storage = get_event_storage()
                avg_num_fg_samples = np.mean(num_fg_samples)
                avg_num_bg_samples = np.mean(num_bg_samples)
                storage.put_scalar("avg_fg_count", avg_num_fg_samples)
                storage.put_scalar("avg_bg_count", avg_num_bg_samples)

            class_labels = torch.cat(class_labels)
            matched_gt_boxes = torch.cat(matched_gt_boxes)
            if self.use_mask:
                gt_masks = PolygonMasks.cat(gt_masks)

            rois = []
            for bid, box in enumerate(resampled_proposals):
                box = box.to(self.device)
                batch_index = torch.full((len(box), 1),
fill_value=float(bid)).to(self.device)
                rois.append(torch.cat([batch_index, box], dim=1))
            rois = torch.cat(rois)
            return class_labels, gt_instances, gt_masks, matched_gt_boxes, rois
```

```
    def compute_class_embeddings_and_logits(self, class_labels, cl_weights,
class_count, roi_feature_count, aligned_roi_features):
        aligned_roi_features = aligned_roi_features.flatten(2)
        bs, spatial_size = aligned_roi_features.shape[0],
aligned_roi_features.shape[-1]
        feats = aligned_roi_features.transpose(-2, -1) @ cl_weights.T

        class_topk = self.num_sample_class
        class_indices = None
        if class_topk < 0:
            class_topk = class_count
            sample_class_enabled = False
        else:
            if class_topk == 0:
                class_topk = class_count
            sample_class_enabled = True

        if sample_class_enabled:
            num_active_classes = class_topk
            init_scores =
functional.normalize(aligned_roi_features.flatten(2).mean(2), dim=1) @
cl_weights.T
            topk_class_indices = torch.topk(init_scores, class_topk, dim=1).indices

            if self.training:
                class_indices = []
                for i in range(roi_feature_count):
                    curr_label = class_labels[i].item()
```

```
                topk_class_indices_i = topk_class_indices[i].cpu()
                if curr_label in topk_class_indices_i or curr_label == class_count:
                    curr_indices = topk_class_indices_i
                else:
                    curr_indices = torch.cat([torch.as_tensor([curr_label]),
topk_class_indices_i[:-1]])
                class_indices.append(curr_indices)
            class_indices = torch.stack(class_indices).to(self.device)
        else:
            class_indices = topk_class_indices

        class_indices = torch.sort(class_indices, dim=1).values
    else:
        num_active_classes = class_count

    other_classes = []
    if sample_class_enabled:
        indexes = torch.arange(0, class_count, device=self.device)[None, None,
:].repeat(bs, spatial_size, 1)
        for i in range(class_topk):
            cmask = indexes != class_indices[:, i].view(-1, 1, 1)
            _ = torch.gather(feats, 2, indexes[cmask].view(bs, spatial_size,
class_count - 1))
            other_classes.append(_[:, :, None, :])
    else:
        for c in range(class_count):
            cmask = torch.ones(class_count, device=self.device,
dtype=torch.bool)
            cmask[c] = False
```

```
            _ = feats[:, :, cmask]
            other_classes.append(_[:, :, None, :])

        other_classes = torch.cat(other_classes, dim=2)
        other_classes = other_classes.permute(0, 2, 1, 3)
        other_classes = other_classes.flatten(0, 1)
        other_classes, _ = torch.sort(other_classes, dim=-1)
        other_classes = interpolate(other_classes, self.T, mode='linear')
        other_classes = self.fc_other_class(other_classes)
        other_classes = other_classes.permute(0, 2, 1)

        inter_dist_emb = other_classes.reshape(bs * num_active_classes, -1,
self.roialign_size, self.roialign_size)
        intra_feats = torch.gather(feats, 2, class_indices[:, None, :].repeat(1,
spatial_size, 1)) if sample_class_enabled else feats
        intra_dist_emb = distance_embed(intra_feats.flatten(0, 1).to('cpu'),
num_pos_feats=self.Tpos_emb)
        intra_dist_emb = intra_dist_emb.to('cuda')
        intra_dist_emb = self.fc_intra_class(intra_dist_emb)
        intra_dist_emb = intra_dist_emb.reshape(bs, spatial_size,
num_active_classes, -1)
        intra_dist_emb = intra_dist_emb.permute(0, 2, 3, 1).flatten(0,
1).reshape(bs * num_active_classes, -1, self.roialign_size, self.roialign_size)

        bg_feats = aligned_roi_features.transpose(-2, -1) @ self.bg_tokens.T
        bg_dist_emb = self.fc_back_class(bg_feats)
        bg_dist_emb = bg_dist_emb.permute(0, 2, 1).reshape(bs, -1,
self.roialign_size, self.roialign_size)
```

```
        bg_dist_emb_c = bg_dist_emb[:, None, :, :, :].expand(-1,
num_active_classes, -1, -1, -1).flatten(0, 1)
        per_cls_input = torch.cat([intra_dist_emb, inter_dist_emb,
bg_dist_emb_c], dim=1)

        cls_logits = self.per_cls_cnn(per_cls_input)

        if isinstance(cls_logits, list):
            cls_logits = [v.reshape(bs, num_active_classes) for v in cls_logits]
        else:
            cls_logits = cls_logits.reshape(bs, num_active_classes)

        cls_dist_feats = interpolate(torch.sort(feats, dim=2).values, self.T,
mode='linear')
        bg_cls_dist_emb = self.fc_bg_class(cls_dist_feats)
        bg_cls_dist_emb = bg_cls_dist_emb.permute(0, 2, 1).reshape(bs, -1,
self.roialign_size, self.roialign_size)
        bg_logits = self.bg_cnn(torch.cat([bg_cls_dist_emb, bg_dist_emb],
dim=1))

        if isinstance(bg_logits, list):
            logits = []
            for c, b in zip(cls_logits, bg_logits):
                logits.append(torch.cat([c, b], dim=1) / self.cls_temp)
        else:
            logits = torch.cat([cls_logits, bg_logits], dim=1)
            logits = logits / self.cls_temp
        return bs, class_indices, class_topk, logits, num_active_classes,
sample_class_enabled
```

```
    def forward(self, batched_inputs: List[Dict[str, torch.Tensor]]):
        bs = len(batched_inputs)
        loss_dict = {}
        if not self.training: assert bs == 1

        cl_weights = self.train_class_weight if self.training else
self.test_class_weight

        class_count = len(cl_weights)
        processed_img, offline_proposals = self.process_images(batched_inputs)
        selected_patch_tokens, complete_patch_tokens =
self.extract_patch_tokens(processed_img)

        if self.training:
            class_labels, gt_instances, gt_masks, matched_gt_boxes, rois =
self.process_detection_proposals(batched_inputs, class_count,
offline_proposals, processed_img)
        else:
            proposal_tensor = offline_proposals[0].proposal_boxes.tensor
            boxes_proposal_tensor = proposal_tensor.to(self.device)
            rois = torch.cat([torch.full((len(boxes_proposal_tensor), 1),
fill_value=0).to(self.device), boxes_proposal_tensor], dim=1)

        aligned_roi_features = self.roi_align(selected_patch_tokens, rois)
        roi_feature_count = len(aligned_roi_features)
        class_labels = []

        if (self.training and (not self.only_train_mask)) or (not self.training):
```

```
            bs, class_indices, class_topk, logits, num_active_classes,
sample_class_enabled =
self.compute_class_embeddings_and_logits(class_labels, cl_weights,
class_count, roi_feature_count, aligned_roi_features)
        else:
            if self.training:
                self.turn_off_cls_training()

        if (self.training and (not self.only_train_mask)) or (not self.training):
            H, W = processed_img.tensor.shape[2:]
            if self.training:
                fg_indices = class_labels != class_count
                matched_gt_boxes = matched_gt_boxes[fg_indices]
                fg_proposals = rois[fg_indices, 1:]
                fg_batch_inds = rois[fg_indices, :1]
                fg_class_labels = class_labels[fg_indices]

                reg_bs = len(fg_proposals)
                aug_rois, pred_roi_mask, gt_roi_mask, covered_flag =
augment_rois(fg_proposals, matched_gt_boxes,img_h=H, img_w=W,
pooler_size=self.reg_roialign_size, min_expansion=0.4, expand_shortest=True)
                aug_rois = torch.cat([fg_batch_inds, aug_rois], dim=1)
                gt_region_coords = abs_coord_2_region_coord(aug_rois[:, 1:],
matched_gt_boxes, self.reg_roialign_size)

                storage = get_event_storage()
                storage.put_scalar("roi_cover_ratio", covered_flag.sum().item() /
covered_flag.numel())
            else:
```

```
            reg_bs = len(rois)
            aug_rois, pred_roi_mask, _, _ = augment_rois(rois[:, 1:], None,
img_h=H, img_w=W, pooler_size=self.reg_roialign_size, min_expansion=0.4,
expand_shortest=True)
            aug_rois = torch.cat([rois[:, :1], aug_rois], dim=1)

        aroi_feats = self.reg_roi_align(selected_patch_tokens, aug_rois)
        aroi_feats = aroi_feats.flatten(2)

        bg_aroi_feats = aroi_feats.transpose(-2, -1) @ self.bg_tokens.T
        bg_aroi_emb = self.reg_bg_dist_emb(bg_aroi_feats)

        fg_aroi_feats = aroi_feats.transpose(-2, -1) @ cl_weights.T
        K2 = self.reg_roialign_size ** 2

        if self.training:
            bg_aroi_emb = bg_aroi_emb.permute(0, 2, 1).reshape(reg_bs,
self.Temb, self.reg_roialign_size,
                                                self.reg_roialign_size)
            fg_aroi_feats = torch.gather(fg_aroi_feats, 2, fg_class_labels[...,
None, None].repeat(1, K2, 1))[:, :,0]
            fg_aroi_emb = distance_embed(fg_aroi_feats.to('cpu'),
num_pos_feats=self.Tpos_emb)
            self.reg_intra_dist_emb = self.reg_intra_dist_emb.to('cpu')
            fg_aroi_emb = self.reg_intra_dist_emb(fg_aroi_emb)
            fg_aroi_emb = fg_aroi_emb.to('cpu')
            bg_aroi_emb = bg_aroi_emb.to('cpu')
            fg_aroi_emb = fg_aroi_emb.permute(0, 2, 1).reshape(reg_bs,
self.Temb, self.reg_roialign_size,
```

```
                                        self.reg_roialign_size)
            aroi_emb = torch.cat([fg_aroi_emb, bg_aroi_emb], dim=1)
        else:
            fg_aroi_dist_feats = torch.gather(fg_aroi_feats, 2, class_indices[:,
None, :].repeat(1, K2, 1)) if sample_class_enabled else fg_aroi_feats
            fg_aroi_emb = distance_embed(fg_aroi_dist_feats.flatten(0,
1).to('cpu'), num_pos_feats=self.Tpos_emb)
            self.reg_intra_dist_emb = self.reg_intra_dist_emb.to('cpu')
            fg_aroi_emb = self.reg_intra_dist_emb(fg_aroi_emb)
            fg_aroi_emb = fg_aroi_emb.reshape(reg_bs, K2, num_active_classes,
-1)
            fg_aroi_emb = fg_aroi_emb.to('cpu')
            bg_aroi_emb = bg_aroi_emb.to('cpu')
            fg_aroi_emb = fg_aroi_emb.permute(0, 2, 3, 1).flatten(0,
1).reshape(reg_bs * num_active_classes, -1, self.reg_roialign_size,
self.reg_roialign_size)
            bg_aroi_emb = bg_aroi_emb.permute(0, 2, 1).reshape(reg_bs,
self.Temb, self.reg_roialign_size, self.reg_roialign_size)[:, None, :, :,
:].repeat(1, num_active_classes, 1, 1, 1).flatten( 0, 1)
            aroi_emb = torch.cat([fg_aroi_emb, bg_aroi_emb], dim=1)
            pred_roi_mask = pred_roi_mask[:, None, :, :].repeat(1,
num_active_classes, 1, 1).flatten(0, 1)

        masks = [pred_roi_mask[:, None, :, :].float(), ]
        num_masks = len(pred_roi_mask)
        embedding = aroi_emb

        if not self.training:
```

```
            aug_rois = aug_rois[:, None, :].repeat(1, num_active_classes,
1).flatten(0, 1)

        for i, (rp, rp_out) in enumerate([
            (self.rp1.to('cpu'), self.rp1_out.to('cpu')),
            (self.rp2.to('cpu'), self.rp2_out.to('cpu')),
            (self.rp3.to('cpu'), self.rp3_out.to('cpu')),
            (self.rp4.to('cpu'), self.rp4_out.to('cpu')),
            (self.rp5.to('cpu'), self.rp5_out.to('cpu'))]):

            masks = [m.to('cpu') for m in masks]
            all_mask_tensor = torch.cat(masks, dim=1).to('cpu')
            embedding = torch.cat([embedding.to('cpu'), all_mask_tensor],
dim=1)
            embedding = rp(embedding)
            pred_mask_logits = rp_out(embedding) / 0.1
            masks.insert(0, pred_mask_logits.sigmoid())

            pred_region_coords = self.r2c(pred_mask_logits).to('cpu')
            if self.training:
                gt_roi_mask = gt_roi_mask.to('cpu').float()
                loss_dict[f"aux_bce_loss_{i}"] =
sigmoid_ce_loss(pred_mask_logits.flatten(1), gt_roi_mask.flatten(1),
num_masks)
                loss_dict[f"aux_dice_loss_{i}"] =
dice_loss(pred_mask_logits.flatten(1), gt_roi_mask.flatten(1), num_masks)

                try:
                    gt_region_coords = gt_region_coords.to('cpu')
```

```
                except NameError:
                    pass

                loss_dict[f'rg_l1_loss_{i}'] =
functional.l1_loss(pred_region_coords, gt_region_coords)
                try:
                    loss_dict[f'rg_giou_loss_{i}'] = (1 -
torch.diag(generalized_box_iou(
                        box_cxcywh_to_xyxy(pred_region_coords),
                        box_cxcywh_to_xyxy(gt_region_coords)))).mean()
                except:
                    pass

            pred_abs_boxes = region_coord_2_abs_coord(aug_rois[:, 1:],
pred_region_coords, self.reg_roialign_size)
            fg_pred_deltas = pred_deltas = self.box2box_transform.get_deltas(
                fg_proposals if self.training else rois[:, None, 1:].repeat(1,
num_active_classes, 1).flatten(0, 1),
                pred_abs_boxes)

            if not self.training:
                pred_deltas = pred_deltas.reshape(reg_bs, num_active_classes, 4)
                pred_deltas = pred_deltas.flatten(1)
        else:
            if self.training:
                self.turn_off_box_training()

        if self.training:
            class_labels = class_labels.long()
```

```
        if not self.only_train_mask:
            if sample_class_enabled:
                bg_indices = class_labels == class_count
                fg_indices = class_labels != class_count
                class_labels[fg_indices] = (class_indices == class_labels.view(-1,
1)).nonzero()[:, 1]
                class_labels[bg_indices] = num_active_classes

            if isinstance(logits, list):
                _log_classification_stats(logits[-1].detach(), class_labels)

                for i, l in enumerate(logits):
                    loss = focal_loss(l, class_labels,
num_classes=num_active_classes, bg_weight=self.bg_cls_weight)
            else:
                _log_classification_stats(logits.detach(), class_labels)
                loss = focal_loss(logits, class_labels,
num_classes=num_active_classes, bg_weight=self.bg_cls_weight)
                loss_dict['focal_loss'] = loss

        if not self.only_train_mask:
            gt_pred_deltas = self.box2box_transform.get_deltas(fg_proposals,
matched_gt_boxes,)
            loss_box_reg = smooth_l1_loss(fg_pred_deltas, gt_pred_deltas,
self.smooth_l1_beta, reduction="sum")
            box_loss = loss_box_reg / max(class_labels.numel(), 1.0)
            if not torch.isinf(box_loss).any():
                loss_dict['bbox_loss'] = box_loss
            else:
```

```
                loss_dict['bbox_loss'] = torch.zeros(1, device=self.device)

        if self.use_mask:
            loss_dict.update(
                self.mask_forward(patch_tokens, rois[fg_indices],
class_labels[fg_indices], cl_weights, gt_masks=gt_masks[fg_indices],
feature_dict=all_patch_tokens))
        return loss_dict
    else:
        assert len(offline_proposals) == 1
        image_size = offline_proposals[0].image_size

        scores = functional.softmax(logits, dim=-1)
        output = {'scores': scores[:, :-1]}

        predict_boxes = self.box2box_transform.apply_deltas(pred_deltas,
rois[:, 1:],)

        if sample_class_enabled:
            full_scores = torch.zeros(len(scores), class_count + 1,
device=self.device)
            full_scores.scatter_(1, class_indices, scores)
            full_scores[:, -1] = scores[:, -1]

            class_indices = class_indices.to(self.device)
            predict_boxes = predict_boxes.to(self.device)

            full_boxes = torch.zeros(len(scores), class_count, 4,
device=self.device)
```

```
            predict_boxes = predict_boxes.view(len(scores), num_active_classes,
4)
            full_boxes.scatter_(1, class_indices[:, :, None].repeat(1, 1, 4),
predict_boxes)
            full_boxes = full_boxes.flatten(1)

            scores = full_scores
            output['scores'] = full_scores[:, :-1]
            predict_boxes = full_boxes

        if self.mult_rpn_score:
            rpn_scores = [x.objectness_logits for x in offline_proposals][0]
            rpn_scores[rpn_scores < 0] = 0
            scores = (scores * rpn_scores[:, None]) ** 0.5

        instances, _ = fast_rcnn_inference(
            [predict_boxes],
            [scores],
            [image_size],
            self.test_score_thresh,
            self.test_nms_thresh,
            False,
            "gaussian",
            0.5,
            0.01,
            self.test_topk_per_image,
            scores_bf_multiply=[scores],
            vis=False
        )
```

```
        if self.use_mask:
            instances[0].pred_masks = self.mask_forward(patch_tokens,
instances[0].pred_boxes.tensor, instances[0].pred_classes, cl_weights,
feature_dict=all_patch_tokens)

        results = self._postprocess(instances, batched_inputs)
        output['instances'] = results[0]['instances']
        return [output, ]
```