НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичного моделювання та аналізу даних

«До захисту допущено»
В.о. завідувача кафедри
_____ Іван ТЕРЕЩЕНКО
«__» _____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

зі спеціальності: 113 Прикладна математика
на тему: **«Порівняльний аналіз нейромережевих методів і моделей розпізнавання та сегментації об'єктів у візуальних процесах будівельного середовища»**

Виконав:
студент 4 курсу, групи ФІ-11
Данілов Костянтин Валентинович _____

Керівник:
к.т.н., доцент
Хайдуров Владислав Володимирович _____

Рецензент:
канд. фіз.-мат. наук, доцент
Цюпій Тамара Іванівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти — перший (бакалаврський)
Спеціальність (освітня програма) — 113 Прикладна математика,
ОПП «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Іван ТЕРЕЩЕНКО
«__» _____ 2025 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Данілов Костянтин Валентинович

1. Тема роботи: «*Порівняльний аналіз нейромережевих методів і моделей розпізнавання та сегментації об'єктів у візуальних процесах будівельного середовища*»,

керівник: к.т.н., доцент Хайдуров Владислав Володимирович,

затверджені наказом по університету № 1761-с від «26» 05 2025 р.

2. Термін подання студентом роботи: «__» _____ 2025 р.

3. Вихідні дані до роботи: *(впишіть вихідні дані до роботи)*

4. Зміст роботи: *Порівняльний аналіз сучасних нейронних архітектур для сегментації екземплярів об'єктів у будівельному середовищі, зокрема моделей Mask R-CNN, Mask2Former та YOLOv8l-seg, із використанням датасету реальних зображень ACID та оцінюванням їх ефективності за метриками точності (mAP) і швидкодії (FPS).*

5. Перелік ілюстративного матеріалу: *«Презентація доповіді»*

6. Дата видачі завдання: 10 вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2024 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2024 р.	Виконано

Студент _____ Данілов К.В.

Керівник _____ Хайдуров В.В.

РЕФЕРАТ

Кваліфікаційна робота містить: 65 сторінок, 8 рисунків, 3 таблиці, 23 джерела.

Розвиток комп'ютерного зору відкриває широкі можливості для автоматизованого аналізу візуальної інформації, зокрема у будівельній галузі. Це має особливе значення для контролю за технічним станом об'єктів та забезпечення безпеки праці. Одним з основних напрямів такого аналізу є сегментація екземплярів, яка дозволяє точно відокремлювати кожен об'єкт на зображенні. Складність сцен на будівельних майданчиках вимагає застосування сучасних глибоких моделей, здатних до високоточної сегментації.

У цій роботі проведено порівняльний аналіз трьох сучасних нейромережевих архітектур для сегментації екземплярів: двоетапної Mask R-CNN, одноетапної YOLOv8l-seg та трансформерної Mask2Former. Дослідження базувалося на реальному наборі даних Alberta Construction Image Dataset (ACID), який містить зображення будівельної техніки. У межах роботи було виконано попередню обробку даних, навчання моделей та їх оцінювання за метриками середньої точності (mAP) та швидкодії (FPS).

Результати експериментів показали, що Mask2Former досягає найвищої якості сегментації, YOLOv8l-seg забезпечує максимальну швидкодію при збереженні високої точності, тоді як Mask R-CNN поступається сучаснішим підходам за загальною ефективністю. Отримані висновки можуть слугувати основою для обґрунтованого вибору оптимальної моделі залежно від конкретних вимог до точності або швидкодії у практичних задачах.

КОМП'ЮТЕРНИЙ ЗІР, СЕГМЕНТАЦІЯ ЕКЗЕМПЛЯРІВ, БУДІВЕЛЬНЕ СЕРЕДОВИЩЕ, MASK R-CNN, YOLOV8-SEG, MASK2FORMER, ACID

ABSTRACT

The qualifying paper contains: 65 pages, 8 figures, 3 tables, 23 references.

The development of computer vision opens up wide opportunities for automated analysis of visual information, particularly in the construction industry. This is especially important for monitoring the technical condition of objects and ensuring workplace safety. One of the main directions of such analysis is instance segmentation, which allows for accurate separation of each object in an image. The complexity of construction site scenes necessitates the use of advanced deep learning models capable of high-accuracy segmentation.

This study presents a comparative analysis of three modern neural network architectures for instance segmentation: the two-stage Mask R-CNN, the one-stage YOLOv8l-seg, and the transformer-based Mask2Former. The research was based on the real-world Alberta Construction Image Dataset (ACID), which contains images of construction machinery. Within the scope of the work, data preprocessing, model training, and their evaluation by mean Average Precision (mAP) and inference speed (FPS) metrics were performed.

Experimental results demonstrate that Mask2Former achieves the highest segmentation accuracy, YOLOv8l-seg provides the fastest inference while maintaining high precision, whereas Mask R-CNN lags behind more modern approaches in overall performance. These findings provide a foundation for selecting an optimal model based on the specific trade-offs between accuracy and speed in practical applications.

COMPUTER VISION, INSTANCE SEGMENTATION, CONSTRUCTION ENVIRONMENT, MASK R-CNN, YOLOV8-SEG, MASK2FORMER, ACID

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN	— Convolutional Neural Network
MLP	— Multi-Layer Perceptron
FPN	— Feature Pyramid Network
PAN	— Path Aggregation Network
RPN	— Regional Proposal Network
RoI	— Region of Interest
MSDeformAttn	— Multi-Scale Deformable Attention
SGD	— Stochastic Gradient Descent
Adam	— Adaptive Moment Estimation
DFL	— Distribution Focal Loss
IoU	— Intersection over Union
AP	— Average Precision
mAP	— mean Average Precision
FPS	— Frames Per Second
GT	— Ground Truth
COCO	— Common Objects in Context
ACID	— Alberta Construction Image Dataset

ВСТУП

Актуальність дослідження. Розвиток комп'ютерного зору, зокрема в завданнях автоматизованої сегментації зображень, відкриває нові можливості для ефективного аналізу візуальних даних у прикладних галузях. Однією з них є будівельна індустрія, що характеризується складними умовами, високим рівнем ризику та потребою в постійному моніторингу технічного стану об'єктів і процесів. Сучасні будівельні майданчики дедалі частіше оснащуються системами відеоспостереження, що створює передумови для впровадження інтелектуальних методів виявлення та аналізу об'єктів у реальному часі.

У цьому контексті задачі розпізнавання та сегментації екземплярів об'єктів є критично важливими для автоматичного контролю виконання робіт, оцінювання технічного стану машин та безпеки праці. Особливий інтерес становлять моделі, здатні до високоточної піксельної сегментації із збереженням просторової структури сцени. Актуальність дослідження зумовлена необхідністю адаптації таких архітектур до специфіки будівельного середовища, де спостерігається висока щільність об'єктів, їхнє перекриття та значна варіативність форм.

Мета та завдання дослідження. Метою дипломної роботи є порівняльний аналіз ефективності сучасних нейромережевих методів сегментації екземплярів об'єктів у контексті візуального аналізу будівельного середовища. Для досягнення поставленої мети необхідно розв'язати такі завдання:

1) Провести огляд сучасних підходів і моделей сегментації екземплярів з фокусом на будівельні застосування;

2) Розглянути теоретичні засади та метрики оцінювання якості сегментації;

3) Реалізувати та налаштувати три обрані архітектури: Mask R-CNN, YOLOv8l-seg та Mask2Former;

4) Провести експериментальне дослідження на основі датасету Alberta Construction Image Dataset (ACID) та порівняти моделі за показниками середньої точності (mAP) та швидкодії (FPS).

Об'єктом дослідження є процеси автоматизованої обробки візуальних даних із будівельного середовища.

Предметом дослідження є нейромережеві моделі та алгоритми сегментації екземплярів, їхні архітектурні особливості, методи навчання та кількісна оцінка якості результатів.

Методи дослідження. У роботі застосовувалися методи математичного моделювання, оптимізації, статистичного аналізу, а також алгоритми глибокого навчання, зокрема згорткові нейронні мережі (CNN) та трансформерні архітектури (Transformer). Для навчання моделей використовувались сучасні фреймворки машинного навчання, такі як pytorch, detectron2 та ultralytics.

Наукова новизна отриманих результатів полягає у порівняльному аналізі ефективності трьох підходів до сегментації екземплярів – класичного двоетапного (Mask R-CNN), сучасного одноетапного (YOLOv8l-seg) та трансформерного (Mask2Former) – саме в умовах будівельного середовища, що характеризується складними візуальними характеристиками. Вперше для такого аналізу застосовано датасет ACID, що містить реальні зображення з будівельних майданчиків.

Практичне значення результатів полягає у можливості використання обґрунтованого вибору архітектури моделі в системах комп'ютерного зору для моніторингу будівельних процесів, контролю техніки та підвищення безпеки праці, з подальшою інтеграцією у виробничі IT-системи обробки візуальних даних.

Апробація результатів та публікації. Основні результати дослідження були апробовані під час виступу на XXIII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики», що проходила в онлайн-режимі 14-17 травня 2025 року.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Розділ містить огляд застосування комп'ютерного зору в будівництві, типів сегментації зображень, ключових метрик оцінки якості, а також обґрунтування вибору моделей для подальшого порівняльного аналізу.

1.1 Застосування комп'ютерного зору в будівельній галузі

Успішність реалізації будівельного проєкту значною мірою визначається дотриманням графіка виконання робіт і технічним станом будівельної техніки. Відставання або збої в роботі машин можуть призводити до зростання витрат, порушення строків, погіршення якості та підвищення аварійності. Контроль цих параметрів є критичним для забезпечення ефективності, якості та безпеки будівництва.

Традиційні методи моніторингу здебільшого базуються на ручному спостереженні та фіксації інформації, що супроводжується низкою обмежень, зокрема високою трудомісткістю, затримками в отриманні даних, ймовірністю помилок і неможливістю забезпечити безперервний контроль у реальному часі. У зв'язку з цим зростає попит на автоматизовані системи моніторингу, здатні своєчасно виявляти відхилення, оптимізувати ресурси та підвищувати точність управлінських рішень.

Широке впровадження відеоспостереження на будівельних майданчиках створює основу для використання методів комп'ютерного зору. Аналіз відеопотоків і зображень дозволяє автоматизувати контроль виконання робіт, оцінювати технічний стан обладнання, фіксувати зміни в середовищі та виявляти потенційні загрози. Отримані в процесі обробки зображень ознаки можуть бути використані як для поточної аналітики,

так і для покращення планування в майбутніх проєктах.

Техніки автоматичної сегментації та розпізнавання будівельних об'єктів стали потужним інструментом для аналізу цих зображень. Вони дозволяють сегментувати та розпізнавати області різних об'єктів на вхідному будівельному зображенні, відкриваючи нові підходи до візуального розуміння будівельного середовища. Наприклад, такі методи сегментації можуть дозволити роботам на майданчику виконувати комплексний аналіз навколишніх умов для безпечної навігації. Також вони забезпечують зручний спосіб для робітників швидко перевіряти стан інфраструктури, наприклад, оцінювати поверхневі дефекти та ідентифікувати пошкоджені елементи.

Останні роки у сфері комп'ютерного зору спостерігається значний прогрес, особливо у розвитку методів **сегментації екземплярів**. На відміну від традиційних методів аналізу зображень, сегментація екземплярів дозволяє деталізовано розпізнавати кожен екземпляр об'єкта на рівні пікселя, забезпечуючи високу роздільну здатність і точність. Інтенсивний розвиток цього напряму пов'язаний з появою ефективних нейромережевих архітектур, зокрема класичних двоетапних (Mask R-CNN, PANet) і сучасних одноетапних моделей на базі архітектури YOLO (наприклад, YOLACT, YOLOv8-seg). Їх застосування в будівельній сфері відкриває широкі можливості для моніторингу та управління [1].

Загалом, алгоритми сегментації екземплярів знайшли широке застосування в різних галузях, що додатково підкреслює їхню універсальність та практичну цінність. У медицині вони дозволяють локалізувати структури організму для ефективнішої діагностики, в аграрному секторі забезпечують точний аналіз посівів та боротьбу зі шкідниками, в автономних транспортних системах гарантують безпечну навігацію та взаємодію з навколишнім середовищем. Аналогічно, інтеграція таких методів у будівництві вже продемонструвала значний потенціал у підвищенні ефективності операцій і рівня безпеки праці.

На сьогодні запропоновано багато досліджень, присвячених задачам

сегментації екземплярів. Ефективність цих методів оцінювалася на кількох публічних датасетах, таких як COCO та ADE20K. Серед сучасних підходів особливу увагу привертають архітектури на базі трансформерів, які завдяки механізму уваги здатні моделювати глобальні залежності у зображенні. Трансформери вже продемонстрували високі результати у багатьох задачах, зокрема в медичній сегментації, автономному водінні, а також у перших дослідженнях для будівельних зображень [2].

Попри перспективність трансформерних архітектур для сегментації екземплярів, їхнє застосування до зображень будівельної тематики все ще залишається недостатньо дослідженим. Це мотивує необхідність комплексного аналізу доцільності використання як традиційних згорткових, так і новітніх трансформерних моделей для автоматизованої обробки візуальної інформації в умовах складного і динамічного будівельного середовища.

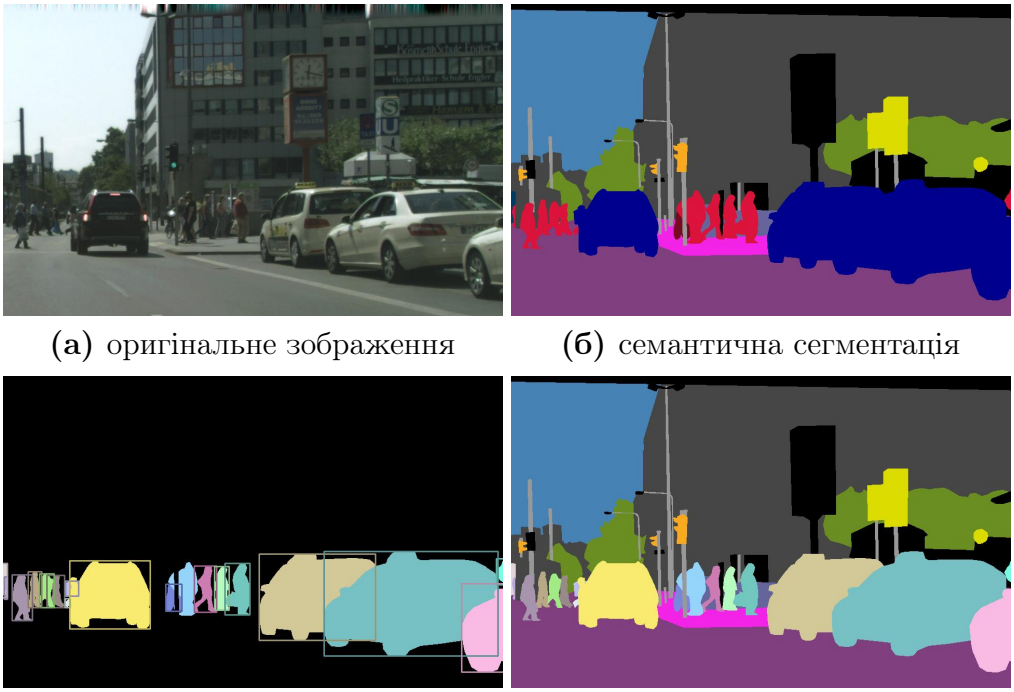
1.2 Основи задач сегментації зображень

Сегментація зображень – це одна з фундаментальних задач комп'ютерного зору, яка полягає у розділенні цифрового зображення на окремі області (сегменти), кожен з яких відповідає певному об'єкту чи його частині. Залежно від поставленої мети, виділяють три основні підходи до сегментації:

Семантична сегментація (semantic segmentation) присвоює кожному пікселю одну з наперед визначених класових міток, не розрізняючи окремі об'єкти одного класу. **Сегментація екземплярів** (instance segmentation) є детальнішим варіантом: окрім класу, метод генерує окрему бінарну маску для кожного екземпляра об'єкта, тим самим ідентифікуючи об'єкти одного класу незалежно. **Панoптична сегментація** (panoptic segmentation) поєднує переваги двох попередніх підходів, надаючи кожному пікселю і клас, і унікальний ідентифікатор екземпляра, що дозволяє отримати повну картину сцени [3].

Такий поділ типів сегментації відображає поступовий перехід від загального опису сцени до максимально деталізованого представлення кожного об'єкта, що визначає вибір методів і моделей у практичних застосуваннях.

Візуальну ілюстрацію відмінностей між основними типами сегментації наведено на рис. 1.1.



(а) оригінальне зображення **(б)** семантична сегментація

(в) сегментація екземплярів **(г)** паноптична сегментація

Рисунок 1.1 – Результати різних типів сегментації [3]

У цьому дослідженні основна увага приділяється задачі сегментації екземплярів, оскільки саме цей підхід забезпечує детальне окреслення меж кожного об'єкта на будівельних сценах і дозволяє аналізувати складні випадки перекриттів та різноманітних класів техніки.

1.3 Постановка задачі сегментації екземплярів

Задача сегментації екземплярів полягає у знаходженні відображення f_θ, параметризованого, як правило, нейронною мережею з

параметрами θ. Це відображення для кожного вхідного зображення $I \in \mathbb{R}^{H\times W\times C}$ (де H, W, C – висота, ширина та кількість каналів відповідно) генерує скінченну множину прогнозів. Кожен прогноз для k-го виявленого екземпляра об'єкта складається з маски $\hat{M}_k \in [0,1]^{H\times W}$ та відповідної мітки класу $\hat{c}_k \in K$, де K – фіксований набір можливих класів об'єктів. Отже, вихід моделі можна представити як:

$$f_\theta(I) = \left\{(\hat{M}_k, \hat{c}_k, \hat{s}_k)\right\}_{k=1}^{N},$$

де N – кількість виявлених екземплярів об'єктів на зображенні, а $s_k \in [0,1]$ – оцінка впевненості моделі у правильності k-го прогнозу (використовується для фільтрації результатів).

Навчання моделі, тобто знаходження оптимальних параметрів θ^*, полягає у мінімізації функції втрат $\mathcal{L}$ на навчальному наборі даних $\mathcal{D} = \{(I_i, \mathcal{Y}_i)\}_{i=1}^{N_{\text{train}}}$, де $\mathcal{Y}_i$ – це набір істинних масок та класів для зображення I_i:

$$\mathcal{Y}_i = \left\{(M_k^{\text{gt}}, c_k^{\text{gt}})\right\}_{k=1}^{N_i^{\text{gt}}}, \quad M_k^{\text{gt}} \in \{0,1\}^{H\times W},\, c_k^{\text{gt}} \in K,$$

Оптимальні параметри визначаються як:

$$\theta^* = \arg\min_\theta \sum_{(I,\mathcal{Y})\in\mathcal{D}} \mathcal{L}(f_\theta(I), \mathcal{Y}).$$

У процесі оптимізації модельні передбачення співставляються з істинними анотаціями за допомогою правил відповідності, таких як поріг Intersection over Union або метод зіставлення (наприклад, Hungarian Matching), що дозволяє коректно порівняти пари $(\hat{M}_k, \hat{c}_k)$ і $(M_k^{\text{gt}}, c_k^{\text{gt}})$.

Таким чином, задача сегментації екземплярів зводиться до побудови функції f_θ, здатної точно ідентифікувати всі екземпляри об'єктів на зображенні за класами та піксельними масками. Подальший вибір архітектури та функції втрат суттєво впливає на точність таких передбачень і буде розглянутий у наступних розділах.

1.4 Метрики оцінки якості сегментації екземплярів

Для об'єктивної оцінки якості моделей сегментації екземплярів використовуються метрики, що враховують як точність локалізації об'єктів, так і коректність класифікації. Стандартними метриками є Intersection over Union, Precision, Recall, Average Precision (AP) та mean Average Precision (mAP) [4, 5].

1.4.1 Intersection over Union (IoU)

Метрика IoU визначає ступінь перекриття між прогнозованою M_{pred} та еталонною M_{gt} сегментаційними масками одного об'єкта. Формально IoU записується так:

$$\text{IoU} = \frac{|M_{\text{pred}} \cap M_{\text{gt}}|}{|M_{\text{pred}} \cup M_{\text{gt}}|}.$$

IoU є метрикою, що дозволяє встановити поріг (наприклад, IoU $\geqslant$ 0.5), вище якого вважається, що модель успішно сегментувала об'єкт.

1.4.2 Точність (Precision) та повнота (Recall)

Точність (Precision) характеризує частку правильно виявлених об'єктів серед усіх знайдених моделлю:

$$\text{Precision} = \frac{TP}{TP + FP},$$

де TP (*True Positive*) – кількість правильно визначених об'єктів, а FP (*False Positive*) – кількість помилково визначених об'єктів.

Повнота (Recall) визначає, яку частку об'єктів із набору істинних

даних модель здатна знайти:

$$\text{Recall} = \frac{TP}{TP + FN},$$

де FN (*False Negative*) – кількість пропущених (не знайдених) об'єктів. Recall є однією з ключових характеристик для оцінювання ефективності алгоритмів машинного навчання, зокрема в галузі комп'ютерного зору. Вона відображає здатність моделі виявляти всі об'єкти, що належать до певного класу. У задачах сегментації та детекції об'єктів повнота використовується як індикатор здатності системи виявляти всі релевантні об'єкти або регіони інтересу на зображенні чи відеопотоці. Водночас, оскільки між Precision та Recall зазвичай існує компроміс, у практиці оцінювання якості моделі часто використовують інтегральні метрики, що узагальнюють цю залежність.

1.4.3 Average Precision (AP)

Average Precision (AP) – інтегральна метрика, що дозволяє узагальнити результати роботи моделі за різних порогів IoU. Вона визначається як площа під кривою Precision–Recall (PR-крива), яка обчислюється за різних порогів впевненості виявлення об'єктів:

$$\text{AP} = \int_0^1 p(r)\, dr,$$

де $p(r)$ – це залежність Precision від Recall для конкретного класу об'єктів при заданому порозі IoU.

У практичних реалізаціях ця інтегральна метрика часто апроксимується методом 11-точкової інтерполяції або шляхом точного підрахунку площі під PR-кривою, що дозволяє коректно порівнювати моделі.

1.4.4 Mean Average Precision (mAP)

Mean Average Precision (mAP) – це узагальнена метрика AP, усереднена по всіх класах та декількох порогах IoU. Вона дозволяє оцінювати загальну ефективність моделі на всьому наборі класів:

$$\begin{aligned}
\mathrm{mAP}_{50} &= \frac{1}{|K|} \sum_{c \in K} \mathrm{AP}_{c,\,0.5}, \\
\mathrm{mAP}_{75} &= \frac{1}{|K|} \sum_{c \in K} \mathrm{AP}_{c,\,0.75}, \\
\mathrm{mAP} &= \frac{1}{10} \sum_{t \in \{0.50, 0.55, \ldots, 0.95\}} \left(\frac{1}{|K|} \sum_{c \in K} \mathrm{AP}_{c,\,t} \right),
\end{aligned}$$

де K – множина усіх класів, $|K|$ – кількість класів у наборі, а $\mathrm{AP}_{c,t}$ – значення Average Precision для класу c при порозі IoU, що дорівнює t.

Метрика mAP є основним інструментом для порівняння моделей у задачах сегментації екземплярів, адже вона враховує як точність визначення меж об'єкта, так і якість класифікації, і використовується як стандарт у сучасних дослідженнях.

1.5 Обґрунтування вибору моделей для порівняння

Для детального аналізу ефективності сучасних методів сегментації екземплярів у будівельному середовищі в цьому дослідженні обрано три моделі – Mask R-CNN, YOLOv8l-seg та Mask2Former. Такий вибір зумовлений необхідністю порівняння різних підходів до реалізації архітектури нейронних мереж, що відображає еволюцію методів комп'ютерного зору за останнє десятиліття.

Mask R-CNN є класичним двоетапним згортковим підходом до сегментації екземплярів, який забезпечив прорив у точності задачі на

відомих відкритих датасетах [6]. Його універсальність і велика кількість доступних реалізацій обумовили широке застосування як еталонної моделі для порівняння нових методів.

YOLOv8l-seg представляє сучасний одноетапний підхід до сегментації екземплярів, орієнтований на застосування в реальному часі [7]. Модель поєднує високу швидкодію з конкурентною точністю, що особливо важливо для систем моніторингу на будівельних майданчиках.

Mask2Former є одним з сучасних представників трансформерних архітектур для сегментації екземплярів, який забезпечує високі показники якості навіть у складних візуальних умовах [8]. Включення цієї моделі дозволяє оцінити потенціал трансформерних рішень у порівнянні з традиційними CNN і сучасними одноетапними моделями.

Обраний набір моделей охоплює три основні підходи: двоетапний CNN (Mask R-CNN), одноетапний CNN (YOLOv8l-seg), а також трансформерну архітектуру (Mask2Former). Це дозволяє здійснити повноцінний порівняльний аналіз із урахуванням різних вимог до точності, швидкодії й адаптивності моделей у контексті задач комп'ютерного зору для будівельного сектора.

Висновки до розділу 1

У цьому розділі було розглянуто теоретичні засади сегментації екземплярів у контексті застосування методів комп'ютерного зору в будівельній галузі. Проаналізовано основні метрики, що застосовуються для оцінювання якості сегментації екземплярів, які будуть використовуватися для порівняння обраних моделей. Обґрунтовано вибір трьох підходів – Mask R-CNN як класичної двоетапної CNN-моделі, YOLOv8l-seg як сучасної одноетапної архітектури, та Mask2Former як трансформерного рішення.

2 МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ СЕГМЕНТАЦІЇ ЕКЗЕМПЛЯРІВ

У цьому розділі проведено детальний аналіз сучасних моделей сегментації екземплярів. Також розглянуто особливості оптимізації навчання глибоких нейронних мереж, зокрема використання алгоритмів SGD та AdamW.

2.1 Mask R-CNN: двоетапна архітектура для сегментації екземплярів

Mask R-CNN [6] є розширенням архітектури Faster R-CNN [9] для задачі сегментації екземплярів, в якій окрім класифікації та локалізації об'єктів додатково виконується піксельна сегментація кожного екземпляра. Основними складовими моделі є екстракція ознак за допомогою згорткової мережі (наприклад, ResNet), побудова ієрархії ознак через Feature Pyramid Network (FPN), генерація регіонів інтересу (RoI) за допомогою Region Proposal Network (RPN), операція точного вирівнювання ознак RoI Align, а також три гілки для вирішення підзадач класифікації, регресії рамки та сегментації маски (рис. 2.1).

На першому етапі вхідне зображення $I \in \mathbb{R}^{H \times W \times 3}$ подається на нейронну мережу основу (backbone), що формує набори багаторівневих тензорів ознак. Для підвищення якості виявлення об'єктів різного розміру використовується FPN [11], що поєднує ознаки з різних рівнів і формує ієрархію тензорів ознак $\{P_l\}$.

Далі Region Proposal Network (RPN) генерує набір регіонів інтересу $\mathcal{R} = \{\mathrm{RoI}_i\}_{i=1}^{N}$, де кожен регіон RoI_i визначається координатами прямокутника (x_1, y_1, x_2, y_2) у координатах зображення. Для кожної такої області виконується операція RoI Align, яка дозволяє отримати

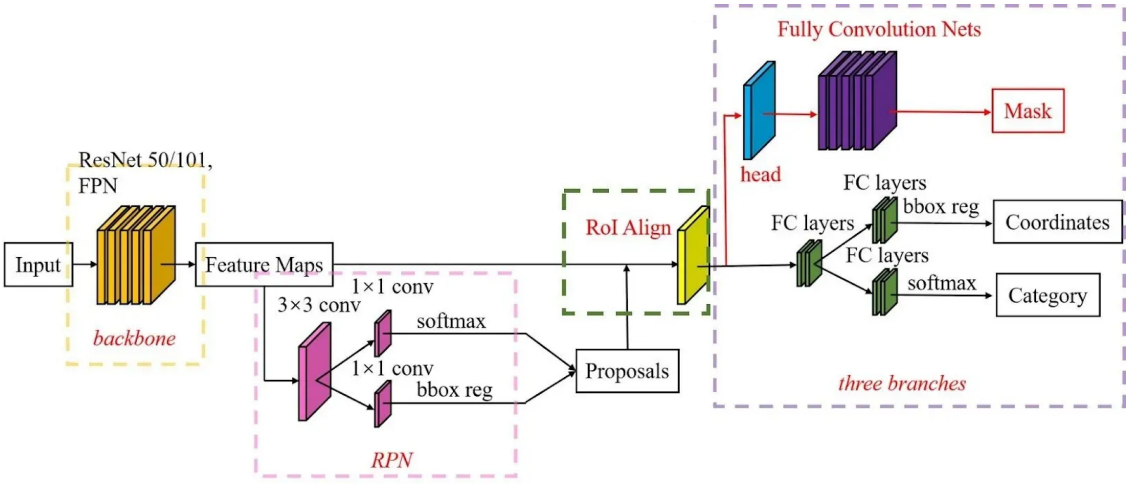


Рисунок 2.1 – Архітектура Mask R-CNN [10]

компактний тензор ознак фіксованого розміру, просторово відповідний цій області. На відміну від класичного RoI Pooling, який використовує грубе квантування координат, RoI Align забезпечує точне вирівнювання вилучених ознак з вхідним зображенням. Для цього межі області інтересу та підобластей не округлюються, а розраховуються у дійсних координатах. Значення ознак у кожному підрегіоні визначаються білінійною інтерполяцією у чотирьох регулярно вибраних точках, після чого агрегуються (максимумом або середнім). Такий підхід дозволяє зберегти високу просторову точність і підвищує якість подальшої сегментації.

Вирівняні ознаки подаються на три паралельні гілки: класифікаційну, регресійну та маскову. Класифікаційна гілка визначає клас об'єкта серед C можливих, регресійна уточнює параметри прямокутної рамки, а маскова гілка виконує піксельну сегментацію в межах RoI для кожного класу.

Оптимізація параметрів моделі здійснюється шляхом мінімізації сукупної функції втрат, що для кожного RoI має вигляд:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}},$$

де $\mathcal{L}_{\text{cls}}$ – функція втрат для багатокласової класифікації, $\mathcal{L}_{\text{box}}$ – функція

втрат для регресії параметрів рамки, $\mathcal{L}_{\text{mask}}$ – функція втрат для сегментації маски.

Для класифікації використовується функція крос-ентропії по класах:

$$\mathcal{L}_{\text{cls}} = -\sum_{c=1}^{C} y_c^* \log(\hat{p}_c),$$

де C – кількість класів, $y_c^* \in \{0, 1\}$ – індикатор істинного класу ($y_{c^*} = 1$, $y_{c \neq c^*} = 0$), $\hat{p}_c$ – ймовірність класу c, отримана шляхом застосування softmax:

$$\hat{p}_c = \frac{e^{s_c}}{\sum_{k=1}^{C} e^{s_k}},$$

де s_c – вихід моделі для класу c. Функція softmax гарантує, що всі значення $\hat{p}_c$ лежать у межах $[0, 1]$ і в сумі дорівнюють 1.

Регресійна втрата для параметрів прямокутної рамки моделюється за допомогою функції smooth L1:

$$\mathcal{L}_{\text{box}} = \sum_{u \in \{x,y,w,h\}} \text{smooth}_{L1}(\hat{t}_u - t_u^*),$$

де $\hat{t}_u$ – передбачене зміщення параметра u, t_u^* – істинне зміщення, яке розраховується для кожної координати як

$$t_x^* = \frac{x^* - x_a}{w_a}, \quad t_y^* = \frac{y^* - y_a}{h_a}, \quad t_w^* = \log\left(\frac{w^*}{w_a}\right), \quad t_h^* = \log\left(\frac{h^*}{h_a}\right),$$

де (x_a, y_a, w_a, h_a) – параметри anchor-box, а (x^*, y^*, w^*, h^*) – параметри істинної рамки. Функція smooth L1 має вигляд:

$$\text{smooth}_{L1}(d) = \begin{cases} 0.5d^2, & \text{якщо } |d| < 1, \\ |d| - 0.5, & \text{інакше.} \end{cases}$$

Маскова компонента втрат обчислюється як середнє значення бінарної

крос-ентропії по всіх пікселях передбаченої $m \times m$ маски:

$$\mathcal{L}_{\text{mask}} = -\frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} \left[M_{ij}^{*} \log(\hat{M}_{ij}) + (1 - M_{ij}^{*}) \log(1 - \hat{M}_{ij}) \right],$$

де $\hat{M}_{ij} \in [0,1]$ – передбачена ймовірність, що піксель (i, j) належить об'єкту, отримана застосуванням сигмоїдної функції $\sigma(x) = \frac{1}{1+e^{-x}}$ до відповідного виходу маскової гілки, а $M_{ij}^{*} \in \{0, 1\}$ – істинна бінарна мітка для пікселя (i, j).

Завдяки такій архітектурі Mask R-CNN дозволяє виконувати точну локалізацію, класифікацію та сегментацію об'єктів у єдиній моделі, що забезпечує високу якість результатів у задачах сегментації екземплярів.

2.2 Mask2Former: трансформерна уніфікована архітектура для сегментації зображень

Модель Mask2Former [8] являє собою уніфіковану архітектуру для розв'язання задач семантичної, екземплярної та паноптичної сегментації зображень. Вона є концептуальним і архітектурним продовженням моделі MaskFormer [12], з низкою суттєвих удосконалень, зокрема впровадженням маскованої крос-уваги, багаторівневої обробки ознак та покращеного механізму побудови масок. Її основою є трансформерні

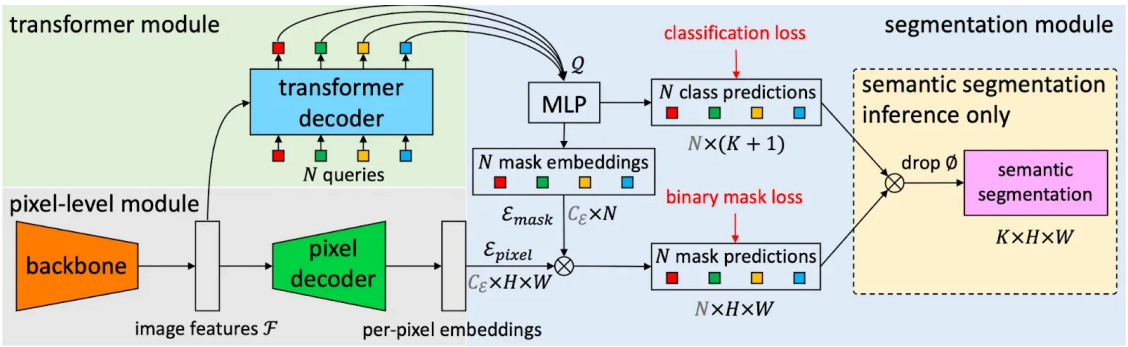


Рисунок 2.2 – Загальна архітектура MaskFormer [12] та Mask2Former [8]

механізми, що дозволяють ефективно враховувати як глобальний, так і локальний контекст сцени. Такий підхід є особливо актуальним для складних сценаріїв, таких як будівельні майданчики, де присутня велика кількість об'єктів, що можуть частково перекриватися.

Як показано на рис. 2.2, складається з наступних послідовно з'єднаних основних компонентів:

1) **Backbone**. Згорткова або трансформерна мережа (наприклад, Swin Transformer [13]), яка вилучає з вхідного зображення ієрархію (піраміду) тензорів ознак $\mathcal{F}_{\text{backbone}}$ на різних просторових роздільностях. Ці багатомасштабні ознаки слугують входом для піксельного декодера.

2) **Піксельний декодер**. Приймає на вхід багатомасштабні ознаки $\mathcal{F}_{\text{backbone}}$ з backbone. Використовуючи механізм багатомасштабної деформованої уваги (MSDeformAttn) [14], він агрегує та збагачує ці ознаки, генеруючи нову піраміду ознак для трансформерного декодера. Крім того, піксельний декодер генерує фінальні попіксельні ембедінги $\mathcal{E}_{\text{pixel}}$ для побудови масок.

3) **Трансформерний декодер**. Працює з фіксованим набором N навчуваних запитів. Він ітеративно оновлює ці запити через декілька шарів, кожен з яких використовує масковану крос-увагу до одного з рівнів ознак $(F^{(1/32)}, F^{(1/16)}, F^{(1/8)})$, отриманих з піксельного декодера. Кожен оновлений запит відповідає за потенційний сегмент зображення. З фінальних оновлених запитів генеруються параметри для передбачення класів та бінарних масок.

4) **Сегментаційний модуль**. На основі фінальних оновлених запитів з трансформерного декодера цей модуль генерує класи об'єктів та бінарні маски.

У наступних підрозділах детальніше розглядаються функціонування кожного з цих модулів.

2.2.1 Піксельний декодер

Піксельний декодер приймає на вхід ієрархію тензорів ознак $\mathcal{F}_{\text{backbone}} = \{F^{(s)}_{\text{backbone}}\}$, отриманих з backbone. Цей спеціалізований модуль будує нову багаторівневу піраміду ознак для подальшого використання у трансформерному декодері.

Pixel Decoder використовує механізм *Multi-Scale Deformable Attention* (MSDeformAttn) [14], який дозволяє ефективно агрегувати інформацію з різних масштабів. У результаті формуються три рівні ознак:

$$F^{(1/32)}, \quad F^{(1/16)}, \quad F^{(1/8)},$$

де кожен тензор $F^{(s)} \in \mathbb{R}^{H_s \times W_s \times C}$ відповідає відповідному рівню роздільності $s \in \left\{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}\right\}$ від вхідного зображення.

Для кожного рівня s ці ознаки додатково модифікуються шляхом підсумовування з двома типами позиційних ембедингів:

– синусоїдальним позиційним ембедингом $e^{(s)}_{\text{pos}} \in \mathbb{R}^{H_s W_s \times C}$, який кодує просторову інформацію про кожен піксель;

– навчуваними ембедингами рівня $e^{(s)}_{\text{lvl}} \in \mathbb{R}^{1 \times C}$, які дають змогу розрізняти ознаки з різних масштабів.

Отже, остаточне представлення для кожного рівня перед подачею в трансформерний декодер виглядає як:

$$\tilde{F}^{(s)} = F^{(s)} + e^{(s)}_{\text{pos}} + e^{(s)}_{\text{lvl}}.$$

Ці багаторівневі ознаки $\tilde{F}^{(s)}$ послідовно подаються в кожен шар трансформерного декодера як ключі K_l та значення V_l для механізму маскованої крос-уваги.

2.2.2 Трансформерний декодер

Трансформерний декодер у Mask2Former виконує ітеративне оновлення набору з N навчуваних запитів (object queries) $\mathcal{Q}_{\text{init}} = \{q_i\}_{i=1}^{N}$, де кожен $q_i \in \mathbb{R}^C$. Ці запити репрезентують гіпотетичні сегменти і є основою для формування кінцевих прогнозів масок та класів. На

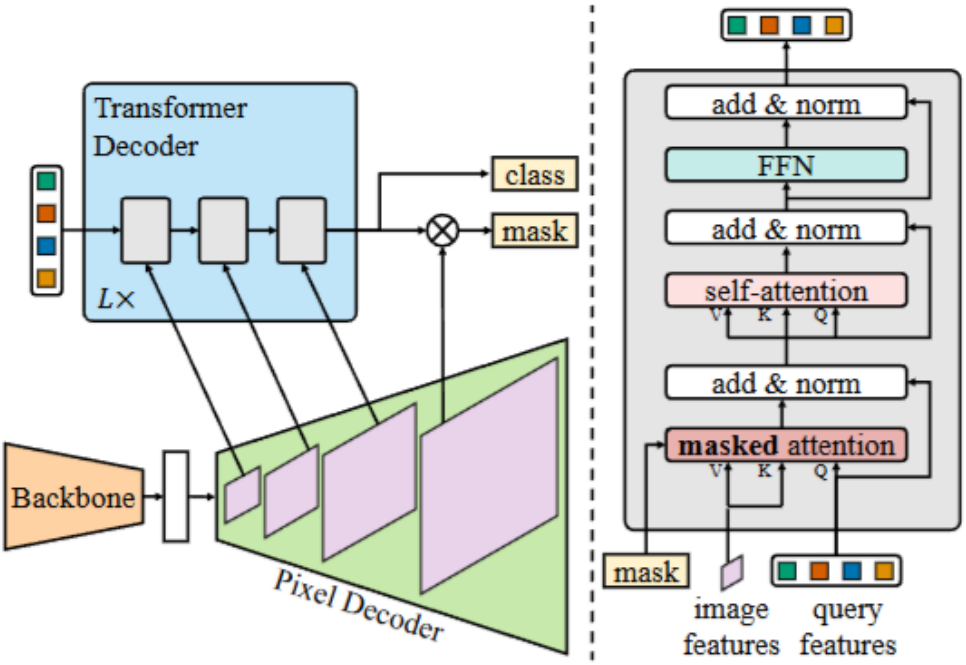


Рисунок 2.3 – Деталізована структура Mask2Former [8]

кожному шарі декодера запити взаємодіють із багаторівневими просторовими ознаками, згенерованими Pixel Decoder, на роздільностях 1/32, 1/16 та 1/8 від вхідного зображення (рис. 2.3). Для кожного рівня ознак $F^{(s)} \in \mathbb{R}^{H_s \times W_s \times C}$ додаються синусоїдальні позиційні ембеддинги $e^{(s)}_{\text{pos}}$ та навчувані ембедингни рівня масштабу $e^{(s)}_{\text{lvl}}$, що дозволяє моделі розрізняти просторове положення та масштабні рівні.

Оновлення запитів на l-му шарі трансформерного декодера здійснюється через механізм крос-уваги (cross-attention) із залишковим

з'єднанням:

$$X_l = \text{softmax}\left(Q_l K_l^\top\right) V_l + X_{l-1},$$

де $X_l \in \mathbb{R}^{N \times C}$ – матриця оновлених ембедингів запитів на l-му шарі. Тут $Q_l = f_Q(X_{l-1}) \in \mathbb{R}^{N \times C}$ – це матриця **запитів** (queries), отримана як лінійна проєкція попереднього стану запитів. $K_l = f_K(F^{(s)})$, $V_l = f_V(F^{(s)})$, $K_l, V_l \in \mathbb{R}^{H_l W_l \times C}$ – відповідно **ключі** (keys) та **значення** (values), які є перетвореннями просторових ознак зображення на цьому рівні. f_Q, f_K, f_V – лінійні перетворення.

На відміну від стандартної крос-уваги (cross-attention), Mask2Former застосовує *масковану увагу*. Цей підхід обмежує область уваги для кожного запиту лише тими пікселями, які належать до маски, передбаченої на попередньому шарі.

Маскована увага на l-му шарі декодера використовує матрицю уваги згідно з

$$X_l = \text{softmax}\left(\mathcal{M}_{l-1} + Q_l K_l^\top\right) V_l + X_{l-1}.$$

При цьому, маска уваги $\mathcal{M}_{l-1}$ для кожної просторової позиції (x, y) визначається як

$$\mathcal{M}_{l-1}(x,y) = \begin{cases} 0, & \text{якщо } \mathbf{M}_{l-1}(x,y) = 1, \\ -\infty, & \text{інакше.} \end{cases}$$

У цьому виразі $\mathbf{M}_{l-1} \in \{0,1\}^{N \times H_l W_l}$ – це бінаризоване (з порогом 0,5) передбачення маски з попереднього, $(l-1)$-го, шару трансформерного декодера, масштабоване до роздільної здатності K_l. Для початкового шару ($l = 0$), $\mathbf{M}_0$ є бінарною маскою, отриманою з X_0 (до того, як запити були подані на обробку трансформерним декодером).

Такий підхід дозволяє кожному запиту фокусуватися лише на релевантній області зображення, що сприяє кращій локалізації об'єктів і швидшій збіжності моделі під час навчання.

Після завершення L шарів трансформерного декодера отримується

оновлений набір запитів $\mathcal{Q}' = \{q'_i\}_{i=1}^{N}$, які використовуються в наступному модулі для прогнозування класів та масок.

2.2.3 Сегментаційний модуль

Сегментаційний модуль формує остаточні передбачення масок і класів на основі оновлених запитів трансформера та піксельних ознак. Кожен вихідний запит $q'_i \in \mathbb{R}^C$ з набору $\mathcal{Q}' = \{q'_i\}_{i=1}^{N}$ містить узагальнену інформацію про відповідний сегмент сцени. Для побудови піксельної маски кожен оновлений запит q'_i спочатку пропускається через MLP для отримання відповідного ембедингу маски $\mathcal{E}_{\text{mask},i} \in \mathbb{R}^C$. Потім модель виконує скалярний добуток між цим ембедингом маски $\mathcal{E}_{\text{mask},i}$ та ознаками піксельного простору $\mathcal{E}_{\text{pixel}} \in \mathbb{R}^{H/4\times W/4\times C}$ (отриманими з Pixel Decoder). Маска обчислюється як:

$$\hat{m}_i(x,y) = \sigma\left(\mathcal{E}^{\top}_{\text{mask},i} \cdot \mathcal{E}_{\text{pixel}}(x,y)\right),$$

де $\sigma(\cdot)$ – сигмоїдна функція. Отримане значення $\hat{m}_i(x,y) \in [0,1]$ відображає ймовірність того, що піксель (x,y) належить i-му сегменту, але не визначає його клас. Усі пікселі, асоційовані з одним і тим самим запитом q'_i, належать до однієї маски.

Клас сегменту визначається окремо, на рівні всієї маски. Для цього оновлений запит q'_i пропускається через лінійний класифікатор, після чого застосовується softmax-активація:

$$p_i = \text{softmax}(\text{Linear}(q'_i)).$$

Отже, кожна маска m_i отримує один клас $c_i = \arg\max p_i$, що відображає її семантичну категорію. Подібне розділення між локалізацією (маска) та класифікацією (клас маски) дозволяє моделі ефективно узагальнювати просторову і семантичну інформацію.

2.2.4 Функція втрат і процедура зіставлення

Оскільки трансформерний декодер генерує фіксований набір з N передбачень $(p_i, \hat{m}_i)$, а кількість істинних об'єктів (ground truth) на зображенні зазвичай менша, виникає необхідність зіставлення передбачень з GT-об'єктами. Для цього використовується угорський алгоритм [15], який забезпечує оптимальне зіставлення між передбаченнями $\hat{y} = \{(p_i, \hat{m}_i)\}_{i=1}^{N}$ і розширеним GT-набором $y = \{(c_i, g_i)\}_{i=1}^{N}$, що включає як істинні об'єкти, так і порожні (no-object, $\varnothing$).

Оптимальним зіставленням $\hat{\sigma}$ є перестановка, яка мінімізує загальну вартість:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i=1}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}),$$

де $\mathcal{L}_{\text{match}}$ – функція вартості зіставлення, що включає втрати класифікації та маски. Зокрема,

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_j) = -\mathbb{1}_{\{c_i \neq \varnothing\}} \log p_j(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{mask}}(g_i, \hat{m}_j),$$

де $p_j(c_i)$ – передбачена ймовірність класу c_i, m_i^{gt} – GT-маска, $\hat{m}_j$ – передбачена маска, і $\mathcal{L}_{\text{mask}}$ – комбінована маскова втрата, $\mathcal{L}_{\text{mask}}(m_i^{\text{gt}}, \hat{m}_j) = \lambda_{\text{ce}} \mathcal{L}_{\text{ce}}(m_i^{\text{gt}}, \hat{m}_j) + \lambda_{\text{dice}} \mathcal{L}_{\text{dice}}(m_i^{\text{gt}}, \hat{m}_j)$.

Загальна функція втрат для навчання задається у вигляді суми втрат по зіставлених парах:

$$\mathcal{L} = \sum_{i=1}^{N} \left[\lambda_{\text{cls}} \mathcal{L}_{\text{cls}}(c_i, p_{\hat{\sigma}(i)}) + \mathbb{1}_{\{c_i \neq \varnothing\}} \left(\begin{array}{c} \lambda_{\text{ce}} \mathcal{L}_{\text{ce}}(m_i^{\text{gt}}, \hat{m}_{\hat{\sigma}(i)}) + \\ \\ + \lambda_{\text{dice}} \mathcal{L}_{\text{dice}}(m_i^{\text{gt}}, \hat{m}_{\hat{\sigma}(i)}) \end{array} \right) \right],$$

де $\lambda_{\text{cls}}, \lambda_{\text{ce}}, \lambda_{\text{dice}}$ – вагові коефіцієнти відповідних компонент втрат.

Маскова втрата складається з кількох компонентів, які визначаються

нижче. Першою є бінарна крос-ентропійна втрата, що має вигляд:

$$\mathcal{L}_{\text{ce}}(m^{\text{gt}}, \hat{m}) = -\frac{1}{|\Omega|} \sum_{(x,y)\in\Omega} \begin{bmatrix} m^{\text{gt}}(x,y) \log \hat{m}(x,y) + \\ + (1 - m^{\text{gt}}(x,y)) \log(1 - \hat{m}(x,y)) \end{bmatrix}$$

а також dice-втрата:

$$\mathcal{L}_{\text{dice}}(m^{\text{gt}}, \hat{m}) = 1 - \frac{2 \sum_{(x,y)\in\Omega} \hat{m}(x,y) \cdot m^{\text{gt}}(x,y) + 1}{\sum_{(x,y)\in\Omega} \hat{m}(x,y) + \sum_{(x,y)\in\Omega} m^{\text{gt}}(x,y) + 1},$$

де Ω – множина всіх пікселів зображення.

Оскільки навчання уніфікованих моделей сегментації, таких як Mask2Former, потребує значних обчислювальних ресурсів через високий розмір передбачуваних масок, у цій архітектурі застосовується оптимізація, що зменшує навантаження на пам'ять. Замість обчислення функцій втрат по всіх пікселях, втрати масок обчислюються лише на підмножині з $K = 12544$ пікселів (тобто 112×112 точок). У процесі зіставлення використовується однакова множина точок для всіх передбачень і GT-масок, відібрана рівномірною випадковою вибіркою. Під час остаточного обчислення втрат для зіставлених пар застосовується вибірка за значимістю (importance sampling) для кожної пари окремо. Такий підхід дозволяє суттєво зменшити споживання пам'яті при збереженні точності сегментації [8].

2.3 YOLOv8-seg: одноетапна архітектура сегментації екземплярів

Архітектура YOLOv8-seg [1, 7] є сучасним одноетапним рішенням для задачі сегментації екземплярів, що характеризується поєднанням високої точності та обчислювальної ефективності. Розроблена на основі серії YOLO, дана модель інтегрує процеси детекції об'єктів та генерації бінарних масок для кожного екземпляра, застосовуючи підхід

прототипної сегментації, натхненний архітектурою YOLACT [16].

2.3.1 Загальна структура архітектури

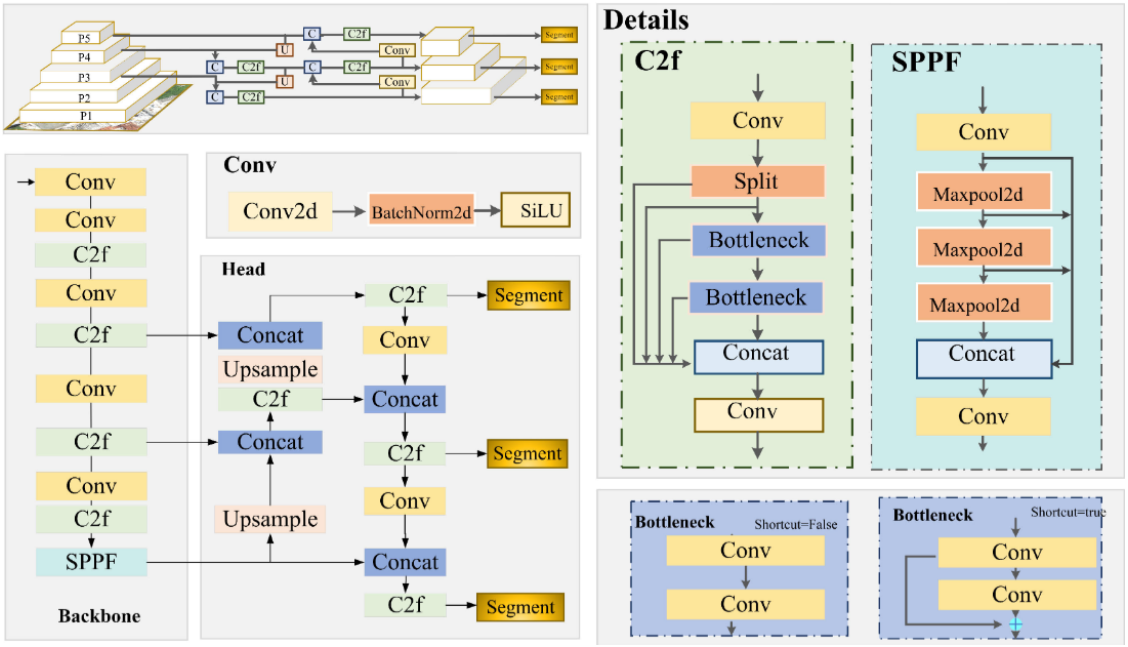


Рисунок 2.4 – Загальна архітектурна схема YOLOv8-seg [1].

Структурно модель YOLOv8-seg (рис. 2.4) складається з двох основних частин: основної мережі (backbone) та голови (head). Backbone, що використовує модулі C2f (Cross-Stage Partial Fusion) та SPPF (Spatial Pyramid Pooling-Fast), відповідає за вилучення ієрархічних тензорів ознак із вхідного зображення на різних просторових рівнях. Голова (head) отримує ці багатомасштабні тензори ознак і здійснює їх агрегацію, реалізуючи принципи, аналогічні **мережі пірамід ознак (Feature Pyramid Network, FPN)** [11] та **мережі агрегації шляхів (Path Aggregation Network, PAN)** [17]. Це дозволяє ефективно поєднувати інформацію з різних масштабів. Далі голова виконує прогнозування координат обмежувальних рамок об'єктів, їхню класифікацію та генерацію відповідних масок сегментації для кожного екземпляра.

2.3.2 Механізм формування масок

Ключовим елементом YOLOv8-seg є механізм генерації масок. Після екстракції ознак з backbone та їх подальшої багатомасштабної агрегації в компоненті head, один з підсумкових тензорів ознак високої роздільної здатності, $F \in \mathbb{R}^{\frac{H}{8} \times \frac{W}{8} \times C}$, передається до спеціалізованої згорткової підмережі, відомої як ProtoNet. Ця підмережа генерує набір з k глобальних прототипів масок:

$$P = \{P_j\}_{j=1}^{k}, \quad \text{де } P_j \in \mathbb{R}^{H_p \times W_p},$$

і $H_p \times W_p$ позначають просторові розміри кожного прототипу. Ці прототипи є незалежними від конкретних екземплярів об'єктів, а є радше базовими масками, з яких можна будувати фінальні маски. Паралельно, для кожного i-го виявленого екземпляра модель прогнозує вектор маскових коефіцієнтів $\mathbf{c}^{(i)} = (c_1^{(i)}, \ldots, c_k^{(i)}) \in \mathbb{R}^k$. Фінальна маска M_i для даного екземпляра формується як лінійна комбінація цих прототипів з відповідними коефіцієнтами, результат якої пропускається через сигмоїдну функцію активації $\sigma(\cdot)$:

$$M_i(x, y) = \sigma\left(\sum_{j=1}^{k} c_j^{(i)} \cdot P_j(x, y)\right),$$

Згенерована таким чином маска M_i потім обрізається відповідно до прогнозованої обмежувальної рамки box_i та інтерполюється до розмірів вихідного зображення.

2.3.3 Формалізація функції втрат

Навчання моделі YOLOv8-seg здійснюється шляхом мінімізації комбінованої функції втрат $\mathcal{L}$, яка є зваженою сумою чотирьох основних

компонент:

$$\mathcal{L} = \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \lambda_{\text{box}}\mathcal{L}_{\text{IoU}} + \lambda_{\text{dfl}}\mathcal{L}_{\text{DFL}} + \lambda_{\text{seg}}\mathcal{L}_{\text{mask}},$$

де $\lambda_{\{\cdot\}} \in \mathbb{R}_+$ – відповідні вагові коефіцієнти для кожної компоненти.

Втрата класифікації $\mathcal{L}_{\text{cls}}$ обчислюється за допомогою бінарної кросентропії (BCE) для кожного прогнозованого класу. Вона порівнює істинну бінарну мітку з прогнозованою ймовірністю. Формула виглядає так:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{N_{\text{obj}}} \sum_{i=1}^{N_{\text{obj}}} \left[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i) \right],$$

де $y_i \in \{0,1\}$ – істинна бінарна мітка класу для i-го об'єкта, а $\hat{p}_i \in (0,1)$ – відповідна прогнозована ймовірність належності до класу. N_{obj} – кількість об'єктів, що розглядаються.

Втрата обмежувальної рамки в YOLOv8-seg реалізується за допомогою двох основних компонент. Перша, втрата локалізації $\mathcal{L}_{\text{IoU}}$, оцінює точність прогнозованих рамок $\hat{b}_i$ відносно еталонних b_i^*. Для цього застосовуються метрики типу Intersection over Union (IoU) або її вдосконалені варіанти, такі як Complete IoU (CIoU), що враховують не лише перекриття, але й відстань між центрами та співвідношення сторін. Друга компонента, втрата $\mathcal{L}_{\text{DFL}}$ (Distribution Focal Loss) [18], є ключовою для досягнення високої точності регресії координат. Вона дозволяє моделювати координати обмежувальних рамок не як точкові значення, а як дискретні розподіли ймовірностей. Для цього весь діапазон можливих значень координати розбивається на N попередньо визначених дискретних інтервалів. Мережа прогнозує ймовірності для кожного інтервалу, а фінальне значення координати обчислюється як їхня зважена сума. Такий підхід забезпечує вищу точність та стабільність навчання регресії, дозволяючи моделі ефективно інтерполювати значення між дискретними інтервалами.

Нарешті, втрата сегментації $\mathcal{L}_{\text{mask}}$ також базується на бінарній

крос-ентропії та обчислюється попіксельно між прогнозованою маскою $M_i(x, y)$ та еталонною бінарною маскою $M_i^*(x, y)$. Обчислення відбувається лише в межах області Ω_i, визначеної відповідною істинною обмежувальною рамкою:

$$\mathcal{L}_{\text{mask}} = -\frac{1}{\sum_k |\Omega_k|} \sum_{i=1}^{N_{\text{obj}}} \sum_{(x,y)\in\Omega_i} \begin{bmatrix} M_i^*(x, y) \log M_i(x, y) + \\ + (1 - M_i^*(x, y)) \log(1 - M_i(x, y)) \end{bmatrix}$$

Вибір компонент $\lambda_{\{\cdot\}}$ дозволяє збалансувати внесок кожного завдання в загальний процес оптимізації.

2.4 Оптимізаційні алгоритми для навчання глибоких нейронних мереж

Навчання глибоких нейронних мереж є ітераційним процесом, спрямованим на мінімізацію функції втрат шляхом коригування параметрів моделі. Ефективність цього процесу значною мірою залежить від вибору та налаштування оптимізаційного алгоритму, який визначає, як саме відбувається оновлення параметрів. У цьому підрозділі розглянуто ключові оптимізатори, що використовуються в глибокому навчанні, зокрема стохастичний градієнтний спуск та його адаптивні варіанти.

2.4.1 Стохастичний градієнтний спуск (SGD)

Стохастичний градієнтний спуск (SGD) є фундаментальним оптимізаційним алгоритмом, що широко використовується для навчання нейронних мереж. Його основна ідея полягає в тому, що замість обчислення градієнта по всьому тренувальному набору даних, що є обчислювально дорогим для великих датасетів, градієнт обчислюється лише по одному випадково вибраному прикладу або, частіше, по

невеликому набору прикладів, який називається міні-пакетом (mini-batch).

Оновлення параметрів θ на ітерації t за допомогою SGD відбувається за формулою:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t),$$

де η – швидкість навчання (learning rate), а $\nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t)$ – градієнт функції втрат $\mathcal{L}$ по параметрах θ_t, обчислений для міні-пакету $\mathcal{B}_t$.

Варіант SGD з моментом дозволяє прискорити збіжність і подолати локальні мінімуми та плато у функції втрат. Він додає інерцію до процесу оновлення, враховуючи напрямок попередніх оновлень. Формули оновлення виглядають так:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t),$$
$$\theta_{t+1} = \theta_t - \eta v_{t+1},$$

де v_t – вектор моменту, а β – коефіцієнт моменту (зазвичай 0.9).

2.4.2 Адаптивні оптимізатори

На відміну від SGD, де швидкість навчання η є фіксованою або змінюється за певним графіком, адаптивні оптимізатори динамічно коригують швидкість навчання для кожного параметра моделі, враховуючи особливості градієнтів.

Adam є одним з найпопулярніших адаптивних оптимізаторів, який поєднує переваги моменту SGD з адаптивним масштабуванням градієнта, як у RMSprop. Він зберігає експоненційно зважені середні перших та

других моментів градієнтів.

$$
\begin{aligned}
g_t &= \nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t), \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\
\hat{m}_t &= m_t / (1-\beta_1^t) \quad \text{(корекція зміщення для першого моменту)}, \\
\hat{v}_t &= v_t / (1-\beta_2^t) \quad \text{(корекція зміщення для другого моменту)}, \\
\theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},
\end{aligned}
$$

де g_t – градієнт на поточній ітерації, m_t та v_t – оцінки першого та другого моменту градієнтів відповідно, β_1 та β_2 – коефіцієнти експоненційного згасання (зазвичай 0.9 та 0.999), а ϵ – мала константа для уникнення ділення на нуль.

AdamW [19] (Adam з декомпозицією регуляризації ваг) є модифікацією Adam, яка вирішує проблему неефективної регуляризації ваг (weight decay) в оригінальному Adam. В Adam такий механізм, як L2-регуляризація, змішується зі швидкістю навчання, що може зменшувати ефективність регуляризації, особливо для параметрів з великою адаптивною швидкістю навчання.

AdamW розділяє процес оновлення ваг та L2-регуляризації. На відміну від Adam, де L2-регуляризація додається до градієнта (тобто, $g_t + \lambda\theta_t$), в AdamW її застосовують окремо:

$$
\begin{aligned}
g_t &= \nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t), \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\
\hat{m}_t &= m_t / (1-\beta_1^t), \\
\hat{v}_t &= v_t / (1-\beta_2^t), \\
\theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda\eta\theta_t,
\end{aligned}
$$

де λ – коефіцієнт регуляризації ваг. Таке розділення дозволяє AdamW досягати кращих результатів, особливо в задачах з глибокими архітектурами, такими як трансформери, забезпечуючи ефективнішу регуляризацію та уникнення перенавчання. Завдяки цим перевагам, AdamW є рекомендованим оптимізатором для навчання багатьох сучасних моделей глибокого навчання.

2.4.3 Планувальники швидкості навчання

Швидкість навчання η є одним з найважливіших гіперпараметрів, що істотно впливає на збіжність та ефективність навчання глибоких нейронних мереж. Неправильно обрана постійна швидкість навчання може призвести до повільної збіжності, коливальних рухів навколо мінімуму, або навіть до розбіжності. **Планувальники швидкості навчання (Learning Rate Schedulers)** – це стратегії, які динамічно змінюють швидкість навчання протягом процесу оптимізації. Їхня мета полягає у забезпеченні швидкої збіжності на початкових етапах навчання та подальшого точного налаштування параметрів, щоб уникнути перескоку через мінімум і досягти кращої продуктивності моделі.

Серед поширених планувальників швидкості навчання виділяють кілька ключових підходів. Один з найпростіших та найпоширеніших – **ступінчасте зменшення (Step Decay)**. Цей метод передбачає зменшення швидкості навчання на певний коефіцієнт через фіксовану кількість епох. Це можна виразити як:

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/S \rfloor}, \quad 0 < \gamma < 1,$$

де η_0 – початкова швидкість навчання, γ – коефіцієнт зменшення (наприклад, 0.1), t – поточна епоха, а S – кількість епох, через які відбувається зменшення. Простота реалізації та ефективність роблять його популярним вибором, хоча він вимагає ручного налаштування

параметрів S та γ.

Іншим ефективним підходом є **косинусне відпалювання (cosine annealing)**, часто використовувана у поєднанні з розігрівом (warmup). Цей планувальник пропонує більш плавне зменшення швидкості навчання, слідуючи формі косинусної хвилі, що дозволяє швидко досліджувати простір параметрів на початку навчання, а потім поступово зменшувати швидкість навчання для більш точної збіжності. Формула для косинусного відпалювання виглядає так:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{t}{T_{\max}}\pi\right)\right),$$

де $\eta_{\min}$ – мінімальна швидкість навчання, $\eta_{\max}$ – максимальна швидкість навчання (зазвичай початкова), t – поточний номер ітерації або епохи, а $T_{\max}$ – загальна кількість ітерацій або епох. Цей планувальник часто демонструє кращі результати порівняно зі ступінчастим зменшенням, оскільки уникає різких стрибків швидкості навчання.

Вибір відповідного планувальника швидкості навчання, а також його параметрів, є важливим етапом налаштування процесу навчання. Правильне використання планувальників може суттєво прискорити збіжність, запобігти розбіжності та покращити кінцеву якість моделі.

2.4.4 Вибір оптимізатора

Вибір оптимізатора залежить від конкретної задачі, архітектури моделі та доступних обчислювальних ресурсів. Хоча SGD з моментом залишається потужним інструментом, особливо для згорткових мереж та великих міні-пакетів, адаптивні оптимізатори, такі як Adam та AdamW, часто демонструють швидшу збіжність на початкових етапах навчання і є більш простими у налаштуванні. Для архітектур на основі трансформерів, де ефективна регуляризація є критичною, AdamW зазвичай є кращим вибором.

Висновки до розділу 2

У цьому розділі проаналізовано ключові теоретичні відомості щодо сучасних методів сегментації екземплярів. Розглянуто архітектурні особливості провідних моделей, таких як Mask R-CNN, Mask2Former та YOLOv8-seg, включаючи механізми формування масок та відповідні функції втрат. Окрему увагу приділено оптимізаційним алгоритмам (SGD, Adam, AdamW) та планувальникам швидкості навчання. Представлені засади створюють необхідний фундамент для подальшого практичного дослідження архітектур у задачах сегментації об'єктів на будівельних майданчиках.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис набору даних Alberta Construction Image Dataset (ACID)

ACID відзначається значною різноманітністю візуальних умов, що забезпечує його релевантність для задач глибокого навчання у будівельній галузі. Датасет містить 10 000 зображень, зібраних із різних джерел, включаючи YouTube, Google Images, Naver, дрони, стаціонарні камери та фотографії з будівельних майданчиків, і охоплює широкий спектр освітлення (день, ніч, дощ, сніг), різноманітні ракурси, а також сцени з великою кількістю техніки, перекриттями та змінами маштабів. Для відбору фінальної підмножини було використано суворі критерії, що гарантують відсутність дублікатів, достатню роздільну здатність і адекватний розмір об'єктів [20, 21].

Усього у наборі вручну анотовано 15 767 екземплярів об'єктів 10 основних категорій будівельної техніки: екскаватор (excavator), коток (compactor), бульдозер (dozer), грейдер (grader), самоскид (dump truck), бетонозмішувач (cement truck), фронтальний навантажувач (wheel loader), екскаватор-навантажувач (backhoe loader), баштовий кран (tower crane), мобільний кран (mobile crane). Для кожного екземпляра доступні точні піксельні маски, що дозволяє детально аналізувати індивідуальні машини у складних сценах будівельних майданчиків. Висока якість ручних анотацій забезпечує придатність ACID для задач детекції, сегментації екземплярів і тестування сучасних моделей глибокого навчання [20].

На рис. 3.1 наведено зображення з набору даних ACID разом із відповідними масками сегментації екземплярів.

Дані приклади ілюструють як різноманітність візуальних умов, так і високу якість розмітки, що робить набір ACID релевантним для аналізу складних сцен будівельного середовища. Це створює необхідну основу для

Рисунок 3.1 – Приклади зображень та відповідних масок з набору даних ACID

подальшої підготовки даних до навчання та експериментів з різними архітектурами нейронних мереж.

3.2 Підготовка даних до навчання

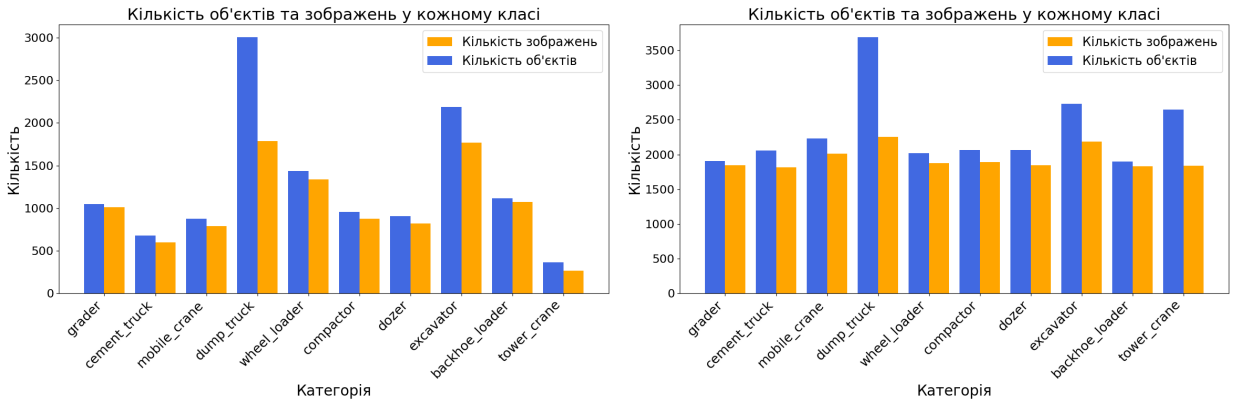
3.2.1 Розбиття на підмножини

Для забезпечення коректної оцінки якості моделей дані було розподілено на три підмножини: навчальну (80%), валідаційну (10%) та тестову (10%). Навчальна підмножина охоплює основний обсяг зображень і використовується для навчання моделей. Валідаційна вибірка призначена для підбору гіперпараметрів і моніторингу стабільності навчання, а тестова підмножина застосовується для фінального оцінювання точності та узагальнювальної здатності моделей. Розподіл здійснювався таким чином, щоб зберегти пропорційне представництво кожного класу в усіх підмножинах.

3.2.2 Візуалізація і статистичний аналіз анотацій

Для контролю якості анотацій і структури даних проведено візуалізацію та статистичний аналіз розподілу об'єктів за класами, а також аналіз кількості об'єктів на зображеннях. Побудовано гістограму розподілу кількості об'єктів за категоріями, що відображено на рис. 3.2а.

З метою розширення тренувальної вибірки й усунення дисбалансу класів у даних застосовувалися різні методи аугментацій. Використовувалися такі перетворення, як горизонтальне віддзеркалення, зміна яскравості та контрасту, додавання шуму, тощо. Під час кожної аугментації відповідно трансформувалися і полігони масок об'єктів. На рис. 3.2б представлено підсумковий розподіл зображень за класами після аугментацій.



(а) Розподіл до аугментацій **(б)** Розподіл після аугментацій

Рисунок 3.2 – Розподіл об'єктів за категоріями у навчальній вибірці до та після аугментації

Застосування аугментацій дало змогу вирівняти розподіл об'єктів між класами у тренувальній вибірці, що знижує ризик зміщення моделі на етапі навчання та підвищує її узагальнювальну здатність.

3.3 Експериментальна установка

3.3.1 Обчислювальне середовище

Експерименти виконувалися у хмарному середовищі Google Colab із використанням графічного процесора NVIDIA L4 (24 GB VRAM). Вибір цієї платформи зумовлений необхідністю високопродуктивного обчислення для тренування сучасних моделей сегментації екземплярів, оскільки великі архітектури (зокрема, Mask2Former) потребують значних обчислювальних ресурсів і обсягу відеопам'яті. Google Colab забезпечує оптимальний баланс між доступністю, простотою масштабування експериментів і інтеграцією з інструментами Python-екосистеми, що дозволяє швидко налаштовувати й розгортати середовище для тестування різних моделей без витрат на фізичну інфраструктуру.

Для реалізації навчання й оцінювання використовувалися такі бібліотеки: PyTorch [22] як основний фреймворк для глибинного навчання, Detectron2 [23] для експериментів із Mask R-CNN та Mask2Former, а також Ultralytics [7] для запуску моделей YOLOv8-seg. Усі бібліотеки були встановлені в актуальних на момент дослідження версіях.

3.3.2 Параметри навчання

Всі моделі навчалися на зображеннях високої роздільної здатності 1024×1024 пікселів. Розмір міні-батчу визначався доступними обчислювальними ресурсами, а саме обсягом відеопам'яті GPU. Для архітектур Mask R-CNN та YOLOv8l-seg застосовувався batch size 8, тоді як для Mask2Former – 4.

Параметри навчання для кожної з моделей систематизовано в таблиці 3.1. Вибір оптимізаторів, планувальників швидкості навчання та методів аугментації даних здійснювався з урахуванням архітектурних особливостей моделей й практичних рекомендацій літератури [6, 8, 7].

Параметр	**Mask R-CNN**	**YOLOv8l-seg**	**Mask2Former**
Оптимізатор	SGD	AdamW	AdamW
Batch size	8	8	4
Розмір зображення	1024×1024	1024×1024	1024×1024
Base learning rate	5×10^{-3}	10^{-4}	10^{-4}
End learning rate	5×10^{-5}	10^{-6}	10^{-6}
LR scheduler	StepLR	WarmupCosineLR	WarmupCosineLR
Аугментації під час навчання	Flip, ResizeScale, Crop, Brightness, Contrast	ResizeScale, Translate, Shear, HSV, Flip, Perspective	Flip, ResizeScale, Crop, Brightness, Contrast
Ітерацій / епох	30 000 ітерацій	20 епох	30 000 ітерацій
Час тренування	∼8 годин	∼8 годин	∼12 годин

Таблиця 3.1 – Основні параметри навчання та аугментації моделей

Представлені в таблиці параметри забезпечують відтворюваність проведених експериментів і створюють методологічне підґрунтя для об'єктивного порівняння архітектур в єдиних експериментальних умовах.

3.4 Аналіз результатів

У табл. 3.2 представлено результати тестування моделей за основними метриками точності та швидкодії. Найвищу mAP показала Mask2Former — 79.2% при високих значеннях mAP_{50} та mAP_{75}. Проте ця модель є суттєво повільнішою за конкурентів, її продуктивність складає лише 5.4 кадрів на секунду, що є критичним обмеженням для застосувань у реальному часі.

Модель	**Backbone**	**mAP, %**	**mAP_{50},%**	**mAP_{75},%**	**FPS,кадр/с**	**Params,млн**
Mask R-CNN	ResNet-101	71.6	92.2	80.3	19.3	63.28
YOLOv8l-seg	modified CSPDarknet53	77.1	94.4	84.8	**33.7**	45.94
Mask2Former	Swin-t	**79.2**	**94.9**	**85.4**	5.4	47.42

Таблиця 3.2 – Результати на тестовій вибірці

YOLOv8l-seg досягає найкращого балансу між точністю (77.1% mAP) і швидкодією (33.7 FPS), значно випереджаючи інші моделі за швидкістю

інференсу. Це робить YOLOv8l-seg оптимальним вибором для сценаріїв, де необхідна обробка відеопотоку в реальному часі.

Mask R-CNN показує гірші результати за точністю (mAP 71.6%), що свідчить про обмеженість класичних підходів у задачах високодеталізованої сегментації екземплярів на складних сценах.

З метою більш детального аналізу було виконано оцінювання якості сегментації окремо для кожного класу техніки, що дозволяє виявити сильні та слабкі сторони моделей залежно від типу об'єкта. Така деталізація є особливо важливою для сценаріїв, де критичною є якість саме для окремих класів, а не лише середній показник. Як видно з табл. 3.3, Mask2Former демонструє стабільно вищу точність для більшості класів, зокрема на складних для детекції об'єктах, як-от *мобільний кран* (75.2% проти 66.2% у YOLOv8l-seg і 64.0% у Mask R-CNN). Водночас для окремих класів, таких як *самоскид* чи *баштовий кран*, модель YOLOv8l-seg забезпечує співставні або навіть кращі результати.

Клас об'єкта	**Mask R-CNN, %**	**YOLOv8l-seg, %**	**Mask2Former, %**
Екскаватор-навантажувач	79.08	85.06	**87.22**
Бетонозмішувач	83.84	88.19	**90.34**
Коток	84.14	89.11	**90.40**
Бульдозер	75.08	80.73	**83.95**
Самоскид	71.32	**74.81**	73.41
Екскаватор	68.80	76.08	**77.47**
Грейдер	81.86	84.45	**86.54**
Мобільний кран	63.96	66.22	**75.19**
Баштовий кран	31.72	44.14	**44.31**
Фронтальний навантажувач	76.50	82.62	**83.58**

Таблиця 3.3 – Порівняння (mAP, %) за класами для різних моделей

Динаміку зміни функції втрат та метрики mAP у процесі навчання досліджуваних архітектур представлено на рис. 3.3. Усі три моделі демонструють стабільну динаміку навчання без очевидних ознак перенавчання. Значення функції втрат на валідації поступово зменшуються, а метрика mAP стабільно зростає або наближається до

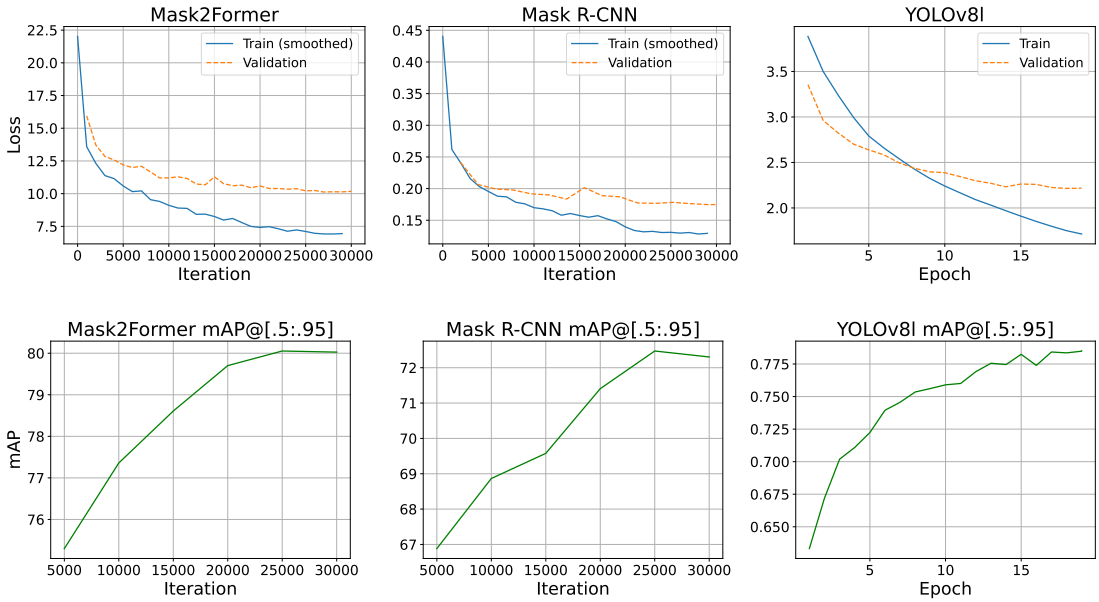
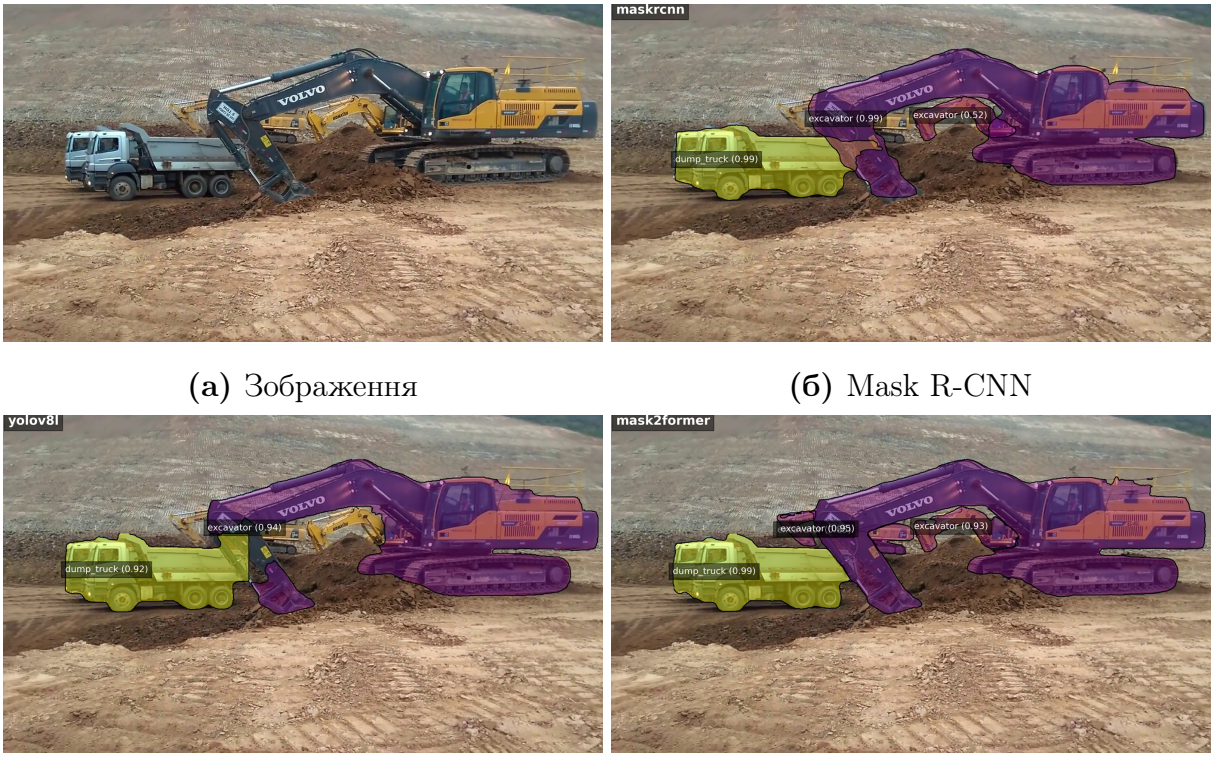


Рисунок 3.3 – Зміна функції втрат та mAP@[0.5:0.95] на валідації під час навчання

плато. Це свідчить про достатню узгодженість між тренувальними та валідаційними даними, а також про коректно налаштовані гіперпараметри оптимізації.

Для наочного аналізу розглянемо приклад сегментації на складному зображенні з тестової вибірки, що ілюструє складну ситуацію з перекриттями об'єктів, варіативним масштабом та наявністю дрібних екземплярів на задньому плані (рис. 3.4). Подібні сцени є типовими для будівельного середовища, де техніка часто розташована у щільному просторі, з перетинами контурів, змішаним освітленням і частковими оклюзіями. У такому контексті класичні моделі, що опираються переважно на локальні ознаки, зазвичай демонструють знижену точність. Насамперед, варто відзначити, що проблема перекриття об'єктів залишається суттєвим викликом для всіх досліджуваних архітектур. На передньому плані сцени присутні два самоскиди, що значно перекривають один одного. Жодна з моделей не змогла повністю та коректно розділити ці екземпляри. Це свідчить про те, що навіть найсучасніші архітектури можуть мати складнощі з розмежуванням об'єктів, які мають високий



(а) Зображення **(б)** Mask R-CNN

(в) YOLOv8l-seg **(г)** Mask2Former

Рисунок 3.4 – Порівняння результатів сегментації однієї сцени для трьох моделей

відсоток перекриття та візуально зливаються.

Mask2Former демонструє високу якість сегментації навіть для складних та відносно невеликих об'єктів на фоні. Модель коректно сегментувала обидва екскаватори на задньому плані. Маски, згенеровані цією моделлю, є найбільш точними та відповідають контурам об'єктів, навіть при їхньому невеликому розмірі. Це підтверджує переваги архітектур на основі трансформерів, зокрема їхню здатність ефективно використовувати глобальний контекст та взаємодію між запитами та ознаками, що дозволяє краще ідентифікувати та розмежовувати об'єкти у складних сценах.

Mask R-CNN показала помітно гірші результати на цій ділянці зображення. Модель виявила лише один з двох задніх екскаваторів, при цьому отримана маска була неточною, містила артефакти та мала значні дефекти накладання, що свідчить про обмеження архітектури у складних

випадках з малими та віддаленими об'єктами.

YOLOv8l-seg, незважаючи на те, що не розпізнала екскаваторів на задньому плані в цьому конкретному випадку, демонструє кращу якість масок для тих об'єктів, які вона все ж таки сегментувала порівняно з Mask R-CNN. Її маски більш згладжені та краще повторюють контури об'єктів, ніж маски Mask R-CNN.

Загалом, отримані результати вказують, що **Mask2Former** доцільно використовувати для задач, де критична точність сегментації і не обмежена швидкість інференсу. Натомість **YOLOv8l-seg** забезпечує оптимальний баланс між точністю та швидкодією, що робить цю модель кращим вибором для застосувань у реальному часі, наприклад для моніторингу й контролю безпеки. **Mask R-CNN** демонструє нижчу ефективність за сучасні підходи, особливо на складних сценах будівельного середовища, що обмежує її застосування в таких умовах.

Висновки до розділу 3

У цьому розділі було детально викладено практичну реалізацію дослідження, включаючи опис набору даних ACID, підготовку даних та параметри експериментальної установки.

Виконано порівняльний аналіз трьох моделей сегментації екземплярів Mask R-CNN, YOLOv8l-seg та Mask2Former. Загалом, **Mask2Former** показав найвищу точність (mAP 79.2%), демонструючи стабільно високі результати для більшості окремих класів. Однак його швидкість інференсу (5.4 кадрів на секунду) є найнижчою.

На противагу, **YOLOv8l-seg** досягла оптимального балансу між точністю (mAP 77.1%) та значно вищою швидкістю (33.7 кадрів на секунду), що робить її найкращим вибором для застосувань у реальному часі. **Mask R-CNN** продемонструвала нижчу загальну ефективність (mAP 71.6%) та гірші показники за окремими класами, що свідчить про обмеженість класичних підходів у складних будівельних сценах.

ВИСНОВКИ

У межах роботи проведено порівняльний аналіз сучасних моделей сегментації екземплярів у задачах комп'ютерного зору для моніторингу будівельного середовища. Розглянуто архітектурні особливості трьох підходів, реалізовано повний цикл їх навчання на датасеті ACID та проаналізовано результати.

Огляд літератури підтвердив актуальність проблеми сегментації екземплярів для автоматизованого контролю об'єктів на будівельних майданчиках. Показано, що складність візуальних умов зумовлює необхідність використання архітектур із високою контекстною чутливістю. Особливу увагу привертають трансформерні моделі, здатні ефективно обробляти такі сцени завдяки глобальним механізмам уваги.

У рамках експериментів порівняно три архітектури: Mask R-CNN, Mask2Former та YOLOv8l-seg. Підготовлено відповідне середовище навчання, включно з аугментацією та балансуванням класів. Проведено навчання на будівельному датасеті ACID, що містить анотовані зображення реальних будівельних майданчиків.

Результати тестування показали, що **Mask2Former** досяг найвищої точності (mAP 79.2%), але є найповільнішою моделлю (5.4 FPS). **YOLOv8l-seg** продемонструвала найкращий баланс між точністю (77.1% mAP) та швидкістю інференсу (33.7 FPS). **Mask R-CNN** показав нижчу точність (mAP 71.6%), що вказує на знижену придатність класичних CNN-архітектур у складних будівельних умовах. Аналіз за класами підтвердив перевагу Mask2Former у розпізнаванні складних об'єктів, тоді як YOLOv8l-seg продемонстрував стабільні результати при високій швидкодії, що робить її практичною для реального застосування.

Напрямки подальших досліджень включають розробку легковагових трансформерів та автоматизацію розмітки даних для розширення наявних датасетів.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Ruihan Bai та ін. «Automated construction site monitoring based on improved YOLOv8-seg instance segmentation algorithm». В: *IEEE Access* 11 (2023), с. 139082—139096.

[2] Xin Wang та ін. «Transformer-based automated segmentation of recycling materials for semantic understanding in construction». В: *Automation in Construction* 154 (2023), с. 104983.

[3] Alexander Kirillov та ін. «Panoptic segmentation». В: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, с. 9404—9413.

[4] Mark Everingham та ін. «The pascal visual object classes (voc) challenge». В: *International journal of computer vision* 88 (2010), с. 303—338.

[5] Rafael Padilla, Sergio L Netto та Eduardo AB Da Silva. «A survey on performance metrics for object-detection algorithms». В: *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE. 2020, с. 237—242.

[6] Kaiming He та ін. «Mask R-CNN». В: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, с. 2961—2969.

[7] Glenn Jocher, Jing Qiu та Ayush Chaurasia. *Ultralytics YOLO*. Вер. 8.0.0. https://ultralytics.com: Ultralytics, 10 січ. 2023. URL: https://github.com/ultralytics/ultralytics.

[8] Bowen Cheng та ін. «Masked-attention mask transformer for universal image segmentation». В: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, с. 1290—1299.

[9] Shaoqing Ren та ін. «Faster r-cnn: Towards real-time object detection with region proposal networks». В: *Advances in neural information processing systems* 28 (2015).

[10] Petru Potrimba. *What is Mask R-CNN? The Ultimate Guide.* 2023. URL: https://blog.roboflow.com/mask-rcnn/.

[11] Tsung-Yi Lin та ін. «Feature Pyramid Networks for Object Detection». В: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017.

[12] Bowen Cheng, Alex Schwing та Alexander Kirillov. «Per-pixel classification is not all you need for semantic segmentation». В: *Advances in neural information processing systems* 34 (2021), с. 17864—17875.

[13] Ze Liu та ін. «Swin transformer: Hierarchical vision transformer using shifted windows». В: *Proceedings of the IEEE/CVF international conference on computer vision.* 2021, с. 10012—10022.

[14] Xizhou Zhu та ін. «Deformable detr: Deformable transformers for end-to-end object detection». В: *arXiv preprint arXiv:2010.04159* (2020).

[15] Nicolas Carion та ін. «End-to-end object detection with transformers». В: *European conference on computer vision.* Springer. 2020, с. 213—229.

[16] Daniel Bolya та ін. «Yolact: Real-time instance segmentation». В: *Proceedings of the IEEE/CVF international conference on computer vision.* 2019, с. 9157—9166.

[17] Shu Liu та ін. «Path aggregation network for instance segmentation». В: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, с. 8759—8768.

[18] Xiang Li та ін. «Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection». В: *Advances in neural information processing systems* 33 (2020), с. 21002—21012.

[19] Ilya Loshchilov та Frank Hutter. «Decoupled weight decay regularization». В: *arXiv preprint arXiv:1711.05101* (2017).

[20] Bo Xiao та Shih-Chung Kang. «Development of an image data set of construction machines for deep learning object detection». В: *Journal of Computing in Civil Engineering* 35.2 (2021), с. 05020005.

[21] Bo Xiao, Yiheng Wang та Shih-Chung Kang. «Deep learning image captioning in construction management: a feasibility study». В: *Journal of Construction Engineering and Management* 148.7 (2022), с. 04022049.

[22] A Paszke. «Pytorch: An imperative style, high-performance deep learning library». В: *arXiv preprint arXiv:1912.01703* (2019).

[23] Yuxin Wu та ін. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

A.1 Mask R-CNN (detectron2)

```
# ===== Imports ========================================================================
from __future__ import annotations

import datetime
import json
import logging
import os
import random
import time
from pathlib import Path
from typing import List

import cv2
import matplotlib.pyplot as plt
import numpy as np
import torch
from detectron2 import model_zoo
from detectron2.config import get_cfg
from detectron2.data import (
    DatasetCatalog,
    MetadataCatalog,
    DatasetMapper,
    build_detection_test_loader,
    build_detection_train_loader,
    transforms as T,
)
from detectron2.data.datasets import register_coco_instances
from detectron2.engine import DefaultTrainer, HookBase
from detectron2.evaluation import COCOEvaluator
from detectron2.utils.logger import setup_logger, log_every_n_seconds
import detectron2.utils.comm as comm

# ===== Logger =========================================================================

setup_logger()

# ===== Dataset registration ===========================================================
DATA_ROOT = Path('/content/content/ACID')
IMG_DIR   = DATA_ROOT / 'images'
ANN_DIR   = DATA_ROOT / 'annotations_split'

```

```
register_coco_instances(
    'acid_train',
    {},
    str(ANN_DIR / 'train_annotations_augmented.json'),
    str(IMG_DIR),
)
register_coco_instances(
    'acid_val',
    {},
    str(ANN_DIR / 'val_annotations.json'),
    str(IMG_DIR),
)
register_coco_instances(
    'acid_test',
    {},
    str(ANN_DIR / 'test_annotations.json'),
    str(IMG_DIR),
)

# ===== Custom hook to log validation loss ==================================
class LossEvalHook(HookBase):
    """Run validation loss every <eval_period> iterations."""

    def __init__(self, eval_period: int, model, data_loader):
        self._period      = eval_period
        self._model       = model
        self._data_loader = data_loader

    def _compute_loss(self, inputs):
        metrics = self._model(inputs)
        metrics = {
            k: (v.detach().cpu().item() if isinstance(v, torch.Tensor) else float(v))
            for k, v in metrics.items()
        }
        return sum(metrics.values())

    def _do_loss_eval(self):
        total       = len(self._data_loader)
        num_warmup  = min(5, total - 1)
        start_time  = time.perf_counter()
        compute_t   = 0.0
        losses: List[float] = []

        for idx, batch in enumerate(self._data_loader):
            if idx == num_warmup:
```

```
                start_time, compute_t = time.perf_counter(), 0.0
            compute_start = time.perf_counter()
            if torch.cuda.is_available():
                torch.cuda.synchronize()
            compute_t += time.perf_counter() - compute_start

            loss = self._compute_loss(batch)
            losses.append(loss)

            iters = idx + 1 - num_warmup * int(idx >= num_warmup)
            sec_per_img = compute_t / iters
            if idx >= num_warmup * 2 or sec_per_img > 5:
                eta = datetime.timedelta(
                    seconds=int(((time.perf_counter()-start_time)/iters)*(total-idx-1))
                )
                log_every_n_seconds(
                    logging.INFO,
                    f'Val loss done {idx+1}/{total}. {sec_per_img:.4f} s/img. ETA={eta}',
                    n=5,
                )

        mean_loss = float(np.mean(losses))
        self.trainer.storage.put_scalar('validation_loss', mean_loss)
        comm.synchronize()

    def after_step(self):
        next_iter = self.trainer.iter + 1
        if next_iter % self._period == 0 or next_iter == self.trainer.max_iter:
            self._do_loss_eval()


# ===== Custom trainer ======================================================
class ACIDMaskRCNNTrainer(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        output_folder = output_folder or (Path(cfg.OUTPUT_DIR) / 'coco_eval')
        output_folder.mkdir(parents=True, exist_ok=True)
        return COCOEvaluator(dataset_name, cfg, False, str(output_folder))

    def build_hooks(self):
        hooks = super().build_hooks()
        iters_per_epoch = max(
            1,
            len(DatasetCatalog.get(self.cfg.DATASETS.TRAIN[0])) //
            self.cfg.SOLVER.IMS_PER_BATCH,
```

```
        )
        hooks.insert(
            -1,
            LossEvalHook(
                iters_per_epoch,
                self.model,
                build_detection_test_loader(
                    self.cfg,
                    self.cfg.DATASETS.TEST[0],
                    DatasetMapper(self.cfg, is_train=True),
                ),
            ),
        )
        return hooks

    @classmethod
    def build_train_loader(cls, cfg):
        augmentations = [
            T.ResizeScale(0.5, 1.5, 1024, 1024),
            T.RandomFlip(0.5, horizontal=True),
            T.RandomBrightness(0.8, 1.2),
            T.RandomContrast(0.8, 1.2),
            T.RandomSaturation(0.8, 1.2),
            T.RandomCrop('relative_range', (0.8, 0.8)),
        ]
        mapper = DatasetMapper(cfg, is_train=True, augmentations=augmentations)
        return build_detection_train_loader(cfg, mapper=mapper)


# ===== Configuration ======================================================
def build_cfg() -> 'detectron2.config.CfgNode':
    cfg = get_cfg()
    cfg.merge_from_file(
        model_zoo.get_config_file('COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml')
    )

    cfg.OUTPUT_DIR               = '/content/models/mask_rcnn'
    cfg.DATASETS.TRAIN           = ('acid_train',)
    cfg.DATASETS.TEST            = ('acid_val',)
    cfg.TEST.EVAL_PERIOD         = 5_000
    cfg.DATALOADER.NUM_WORKERS   = 8

    cfg.MODEL.WEIGHTS            = model_zoo.get_checkpoint_url(
        'COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml'
    )
```

```
    cfg.MODEL.ROI_HEADS.NUM_CLASSES          = 10
    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 1024

    # Solver
    cfg.SOLVER.IMS_PER_BATCH = 8
    cfg.SOLVER.BASE_LR       = 0.005
    cfg.SOLVER.MAX_ITER      = 30_000
    cfg.SOLVER.STEPS         = (20_000, 26_000)
    cfg.SOLVER.WARMUP_ITERS  = 1_000
    cfg.SOLVER.CLIP_GRADIENTS.ENABLED     = True
    cfg.SOLVER.CLIP_GRADIENTS.CLIP_VALUE = 1.0
    cfg.SOLVER.CHECKPOINT_PERIOD          = 2_000
    cfg.SOLVER.AMP.ENABLED                = True

    # Input
    cfg.INPUT.MIN_SIZE_TRAIN_SAMPLING = 'choice'
    cfg.INPUT.MIN_SIZE_TRAIN          = (800, 1024)

    Path(cfg.OUTPUT_DIR).mkdir(parents=True, exist_ok=True)
    return cfg


# ===== Main ========================================================================
if __name__ == '__main__':
    cfg = build_cfg()
    trainer = ACIDMaskRCNNTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()
```

A.2 Mask2Former (detectron2)

```
# ===== Imports =====================================================================

import os
import time
import datetime
import logging
import warnings
import numpy as np
import torch
from pathlib import Path

from detectron2.utils.logger import setup_logger, log_every_n_seconds
```

```
import detectron2.utils.comm as comm
from detectron2.config import get_cfg
from detectron2.engine import DefaultTrainer, HookBase
from detectron2.evaluation import COCOEvaluator
from detectron2.data import (
    MetadataCatalog,
    DatasetCatalog,
    build_detection_train_loader,
    build_detection_test_loader,
    transforms as T,
)
from detectron2.data.datasets import register_coco_instances

# Mask2Former-specific imports
from detectron2.projects.deeplab import add_deeplab_config
from mask2former import add_maskformer2_config
from mask2former.data.dataset_mappers.\
    mask_former_instance_dataset_mapper import MaskFormerInstanceDatasetMapper
from train_net import Trainer as Mask2FormerTrainer

# ===== Logger ========================================================

setup_logger()
setup_logger(name="mask2former")

# ===== Dataset Registration ==========================================

DATA_ROOT = Path("../ACID")
IMG_DIR   = DATA_ROOT / "images"
ANN_DIR   = DATA_ROOT / "annotations_split"

register_coco_instances(
    "my_dataset_train", {},
    str(ANN_DIR / "train_annotations_augmented.json"),
    str(IMG_DIR),
)
register_coco_instances(
    "my_dataset_val", {},
    str(ANN_DIR / "val_annotations.json"),
    str(IMG_DIR),
)
register_coco_instances(
    "my_dataset_test", {},
    str(ANN_DIR / "test_annotations.json"),
    str(IMG_DIR),
```

```
)

train_metadata = MetadataCatalog.get("my_dataset_train")
val_metadata   = MetadataCatalog.get("my_dataset_val")
test_metadata  = MetadataCatalog.get("my_dataset_test")

train_dataset_dicts = DatasetCatalog.get("my_dataset_train")
val_dataset_dicts   = DatasetCatalog.get("my_dataset_val")
test_dataset_dicts  = DatasetCatalog.get("my_dataset_test")

# ===== Custom hook to log validation loss ====================================

class LossEvalHook(HookBase):
    """Run validation loss every <eval_period> iterations."""
    def __init__(self, eval_period, model, data_loader):
        self._period      = eval_period
        self._model       = model
        self._data_loader = data_loader

    def _compute_loss(self, inputs):
        metrics = self._model(inputs)
        metrics = {
            k: (v.detach().cpu().item() if isinstance(v, torch.Tensor) else float(v))
            for k, v in metrics.items()
        }
        return sum(metrics.values())

    def _do_loss_eval(self):
        total = len(self._data_loader)
        num_warmup = min(5, total - 1)
        start_time = time.perf_counter()
        compute_t = 0.0
        losses = []

        for idx, batch in enumerate(self._data_loader):
            if idx == num_warmup:
                start_time, compute_t = time.perf_counter(), 0.0
            compute_start = time.perf_counter()
            if torch.cuda.is_available():
                torch.cuda.synchronize()
            compute_t += time.perf_counter() - compute_start

            loss = self._compute_loss(batch)
            losses.append(loss)

```

```
            iters = idx + 1 - num_warmup * int(idx >= num_warmup)
            sec_per_img = compute_t / iters
            if idx >= num_warmup * 2 or sec_per_img > 5:
                eta = datetime.timedelta(
                    seconds=int(((time.perf_counter()-start_time)/iters)*(total-idx-1))
                )
                log_every_n_seconds(
                    logging.INFO,
                    f"Val loss done {idx+1}/{total}. {sec_per_img:.4f} s/img. ETA={eta}",
                    n=5,
                )

        mean_loss = float(np.mean(losses))
        self.trainer.storage.put_scalar('validation_loss', mean_loss)
        comm.synchronize()

    def after_step(self):
        next_iter = self.trainer.iter + 1
        if next_iter % self._period == 0 or next_iter == self.trainer.max_iter:
            self._do_loss_eval()

# ===== Custom Trainer for Mask2Former ========================================

class MyMask2FormerTrainer(Mask2FormerTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        output_folder = output_folder or os.path.join(cfg.OUTPUT_DIR, "inference")
        os.makedirs(output_folder, exist_ok=True)
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

    def build_hooks(self):
        hooks = super().build_hooks()
        iters_per_epoch = max(
            1,
            len(DatasetCatalog.get(self.cfg.DATASETS.TRAIN[0])) //
            self.cfg.SOLVER.IMS_PER_BATCH,
        )
        hooks.insert(
            -1,
            LossEvalHook(
                iters_per_epoch,
                self.model,
                build_detection_test_loader(
                    self.cfg,
                    self.cfg.DATASETS.TEST[0],
```

```
                    MaskFormerInstanceDatasetMapper(self.cfg, is_train=True),
                ),
            ),
        )
        return hooks

    @classmethod
    def build_train_loader(cls, cfg):
        augmentations = [
            T.ResizeScale(
                min_scale=0.1, max_scale=2.0,
                target_height=1024, target_width=1024
            ),
            T.FixedSizeCrop(
                crop_size=(1024, 1024),
                pad=True, pad_value=128.0, seg_pad_value=255
            ),
            T.RandomBrightness(0.8, 1.2),
            T.RandomContrast(0.8, 1.2),
            T.RandomSaturation(0.8, 1.2),
        ]
        mapper=MaskFormerInstanceDatasetMapper(cfg,is_train=True,augmentations=augmentations)
        return build_detection_train_loader(cfg, mapper=mapper)

# ===== Configuration ========================================================

def build_cfg() -> 'detectron2.config.CfgNode':
    cfg = get_cfg()
    add_deeplab_config(cfg)
    add_maskformer2_config(cfg)
    cfg.merge_from_file(
        "configs/coco/instance-segmentation/swin/"
        "maskformer2_swin_tiny_bs16_50ep.yaml"
    )

    cfg.OUTPUT_DIR = "../models/Detectron2_Models"
    cfg.DATASETS.TRAIN = ("my_dataset_train",)
    cfg.DATASETS.TEST = ("my_dataset_val",)
    cfg.TEST.EVAL_PERIOD = 5000
    cfg.DATALOADER.NUM_WORKERS = 8

    # Model parameters
    cfg.MODEL.WEIGHTS = (
        "https://dl.fbaipublicfiles.com/maskformer/mask2former/"
        "coco/panoptic/maskformer2_swin_tiny_bs16_50ep/model_final_9fd0ae.pkl"
```

```
    )
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 10
    cfg.MODEL.SEM_SEG_HEAD.NUM_CLASSES = 10

    cfg.MODEL.MASK_FORMER.TEST.INSTANCE_ON = True
    cfg.MODEL.MASK_FORMER.TEST.SEMANTIC_ON = False
    cfg.MODEL.MASK_FORMER.TEST.PANOPTIC_ON = False
    cfg.INPUT.DATASET_MAPPER_NAME = "mask_former_instance"

    # Solver parameters
    cfg.SOLVER.IMS_PER_BATCH = 8
    cfg.SOLVER.OPTIMIZER = "ADAMW"
    cfg.SOLVER.BACKBONE_MULTIPLIER = 0.1
    cfg.SOLVER.WEIGHT_DECAY = 0.05
    cfg.SOLVER.BASE_LR = 0.0001
    cfg.SOLVER.BASE_LR_END = 0.0001 * 0.01
    cfg.SOLVER.LR_SCHEDULER_NAME = "WarmupCosineLR"
    cfg.SOLVER.MAX_ITER = 30000
    cfg.SOLVER.CHECKPOINT_PERIOD = 2000
    cfg.SOLVER.AMP.ENABLED = True

    cfg.DATALOADER.ASPECT_RATIO_GROUPING = True
    cfg.DATALOADER.SAMPLER_TRAIN = "RepeatFactorTrainingSampler"
    cfg.DATALOADER.REPEAT_THRESHOLD = 0.1
    cfg.DATALOADER.REPEAT_SQRT = False

    cfg.SOLVER.WARMUP_FACTOR = 0.001
    cfg.SOLVER.WARMUP_ITERS = 1000
    cfg.SOLVER.WARMUP_METHOD = "linear"

    cfg.SOLVER.CLIP_GRADIENTS.ENABLED = False

    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    return cfg

# ===== Main ======================================================================

if __name__ == "__main__":
    warnings.filterwarnings("ignore", category=FutureWarning)
    cfg = build_cfg()
    trainer = MyMask2FormerTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()
```

A.3 YOLOv8l-seg (ultralytics)

```python
# ===== Imports ==================================================================

import os
import json
import shutil
from tqdm import tqdm
import yaml

# ===== Directories for YOLO =====================================================

dataset_root = "yolo_dataset"
folders = [
    "images/train", "images/val", "images/test",
    "labels/train", "labels/val", "labels/test"
]
for folder in folders:
    os.makedirs(f"{dataset_root}/{folder}", exist_ok=True)

# ===== COCO to YOLO Segmentation Converter ======================================

def convert_coco_to_yolo_segmentation(json_file, output_folder):
    with open(json_file, 'r') as file:
        coco_data = json.load(file)

    os.makedirs(output_folder, exist_ok=True)

    annotations = coco_data['annotations']
    for annotation in annotations:
        image_id = annotation['image_id']
        category_id = annotation['category_id'] - 1
        segmentation = annotation['segmentation']

        for image in coco_data['images']:
            if image['id'] == image_id:
                image_filename = os.path.splitext(os.path.basename(image['file_name']))[0]
                image_width = image['width']
                image_height = image['height']
                break

        yolo_segmentation = [
            f"{(x) / image_width:.5f} {(y) / image_height:.5f}"
            for x, y in zip(segmentation[0][::2], segmentation[0][1::2])
```

```
        ]
        yolo_segmentation = ' '.join(yolo_segmentation)
        yolo_annotation = f"{category_id} {yolo_segmentation}"

        output_filename = os.path.join(output_folder, f"{image_filename}.txt")
        with open(output_filename, 'a+') as file:
            file.write(yolo_annotation + '\n')

# ===== COCO to YOLO annotation conversion ===================================

train_json = "/content/content/ACID/annotations_split/train_annotations_augmented.json"
val_json   = "/content/content/ACID/annotations_split/val_annotations.json"
test_json  = "/content/content/ACID/annotations_split/test_annotations.json"

convert_coco_to_yolo_segmentation(train_json, "yolo_dataset/labels/train")
convert_coco_to_yolo_segmentation(val_json,   "yolo_dataset/labels/val")
convert_coco_to_yolo_segmentation(test_json,  "yolo_dataset/labels/test")

# ===== Image copying ========================================================

images_dir = "/content/content/ACID/images"
def copy_images_from_json(json_path, dst_folder, images_dir):
    with open(json_path, 'r') as f:
        data = json.load(f)
    os.makedirs(dst_folder, exist_ok=True)
    image_filenames = [os.path.basename(img['file_name']) for img in data['images']]
    for filename in tqdm(image_filenames, desc=f"Copying to {dst_folder}"):
        src_path = os.path.join(images_dir, filename)
        dst_path = os.path.join(dst_folder, filename)
        if os.path.exists(src_path):
            shutil.copyfile(src_path, dst_path)

copy_images_from_json(train_json, "yolo_dataset/images/train", images_dir)
copy_images_from_json(val_json,   "yolo_dataset/images/val", images_dir)
copy_images_from_json(test_json,  "yolo_dataset/images/test", images_dir)

# ===== YAML file creation ===================================================

class_names = [
    "grader", "cement_truck", "mobile_crane", "dump_truck", "wheel_loader",
    "compactor", "dozer", "excavator", "backhoe_loader", "tower_crane"
]
data_config = {
    "path": "/content/yolo_dataset",
    "train": "images/train",
```

```
    "val": "images/val",
    "test": "images/test",
    "nc": len(class_names),
    "names": class_names
}
with open("yolo_dataset/data.yaml", "w") as f:
    yaml.dump(data_config, f, sort_keys=False)
# ===== YOLOv8l-seg training ================================================
from ultralytics import YOLO

model = YOLO("yolov8l-seg.pt")
model.info()

results = model.train(
    data="yolo_dataset/data.yaml",
    epochs=30,
    imgsz=1024,
    batch=8,
    optimizer="AdamW",
    lr0=1e-4,
    lrf=0.01,
    weight_decay=0.0005,
    warmup_epochs=3.0,
    warmup_momentum=0.8,
    momentum=0.9,
    box=7.5,
    cls=0.5,
    dfl=1.0,
    degrees=5.0,
    translate=0.1,
    scale=0.5,
    shear=2.0,
    perspective=0.0005,
    flipud=0.0,
    fliplr=0.5,
    mosaic=0,
    mixup=0,
    copy_paste=0,
    hsv_h=0.015,
    hsv_s=0.7,
    hsv_v=0.4,
    bgr=0.1,
)
```