

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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

**на тему: «Класифікація та сегментація медичних зображень за
допомогою моделі Detection Transformer (DETR)»**

Виконав:

студент (-ка) IV курсу, групи КА-13

Іванюта Олександр Олександрович

Керівник:

професор, д.т.н., доцент, Недашківська Надія Іванівна

Консультант з економічного розділу:

доцент, к.е.н., Рощина Надія Василівна

Консультант з нормконтролю:

Асистент каф. ММСА Канцедал Георгій Олегович

Рецензент:

Професор, д.т.н., Данилов Валерій Якович

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

Студент _____

Київ — 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти — перший (бакалаврський)

Спеціальність — 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Іванюті Олександрю Олександровичу

1. Тема роботи «Класифікація та сегментація медичних зображень за допомогою моделі Detection Transformer (DETR)», керівник роботи Недашківська Надія Іванівна, д.т.н., доцент, затверджені наказом по університету від «26» травня 2025 р. №1759-с
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи: медичний датасет BBBC041 (P. vivax infected human blood smears), що містить RGB-зображення клітин крові високої роздільної здатності з відповідними анотаціями, отриманий із відкритого ресурсу Broad Bioimage Benchmark Collection.
4. Зміст роботи: дослідження предметної області, теоретичні основи та архітектура моделі Detection Transformer, опис проведених досліджень, функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): зображення клітин крові, фрагменти анотованих зображень клітин крові, приклади передбачень моделі з bounding box-ами, графіки функції втрат та метрик якості, діаграми архітектури DETR.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент, к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики дослідження	25.03.2025	Виконано
2	Аналіз актуальності та дослідження тематичної літератури	30.03.2025	Виконано
3	Формування мети, завдань дослідження та огляд сучасних методів розв'язання завдання	10.04.2025	Виконано
4	Підготовка та попередня обробка даних	19.04.2025	Виконано
5	Розробка програмної реалізації	30.04.2025	Виконано
6	Проведення експериментів, побудова графіків, розрахунок метрик, візуалізація результатів	17.05.2025	Виконано
7	Аналіз результатів та інтерпретація ефективності моделі	21.05.2025	Виконано
8	Оформлення пояснювальної записки	01.06.2025	Виконано
9	Підготовка презентації доповіді	04.06.2025	Виконано
10	Попередній захист дипломної роботи	05.06.2025	Виконано
11	Захист дипломної роботи		Виконано

Студент

Олександр ІВАНЮТА

Керівник

Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота: 150 с., 25 рис., 10 табл., 2 додатки, 84 джерела.

ДІАГНОСТИКА МАЛЯРІЇ, DETECTION TRANSFORMER, СЕГМЕНТАЦІЯ, КЛАСИФІКАЦІЯ, DEEP LEARNING, COCO FORMAT, BVBC041

Метою роботи є розробка системи автоматизованого аналізу медичних зображень для виявлення клітин, уражених збудником малярії *Plasmodium vivax*, із використанням моделі Detection Transformer (DETR).

У роботі проведено огляд сучасних методів діагностики малярії, зокрема комп'ютерних підходів на основі глибокого навчання. Проаналізовано архітектуру моделі DETR, розглянуто її переваги у задачах об'єктного детектування та сегментації.

Особливу увагу приділено підготовці набору даних BVBC041: виконано анотування зображень, створення пайплайну для переведення в формат COCO, а також валідацію структурованих даних.

Здійснено донавчання попередньо натренованої моделі DETR на медичному датасеті BVBC041 із налаштуванням гіперпараметрів, також було реалізовано обробку результатів передбачення.

Розроблено інструментарій аналізу якості моделі: побудовано графіки функції втрат, точності, метрик AP та приклади передбачень. У фіналі роботи представлено інтерпретацію результатів та сформовано висновки щодо доцільності використання підходу DETR у медичній діагностиці.

ABSTRACT

Diploma Thesis (Bachelor's Thesis): 150 pages, 25 figures, 10 tables, 2 appendices, 84 references.

MALARIA DIAGNOSIS, DETECTION TRANSFORMER, SEGMENTATION, CLASSIFICATION, DEEP LEARNING, COCO FORMAT, BBBC041

The aim of this thesis is to develop an automated medical image analysis system for detecting cells infected with the *Plasmodium vivax* malaria parasite using the Detection Transformer (DETR) model.

The work includes an overview of current malaria diagnostic methods, with a focus on computer-based approaches powered by deep learning. The architecture of the DETR model is analyzed, emphasizing its advantages in object detection and segmentation tasks.

Special attention is given to the preparation of the BBBC041 dataset: image annotation was performed, a pipeline for conversion to the COCO format was implemented, and the structured data was validated.

Fine-tuning of a pretrained DETR model was carried out on the BBBC041 medical dataset with appropriate hyperparameter adjustment, and the prediction results were processed accordingly.

An evaluation toolkit was developed, including loss and accuracy plots, AP metrics, and visual examples of predictions. The final part of the thesis presents result interpretation and conclusions regarding the applicability of DETR-based approaches in medical diagnostics.

ЗМІСТ

ВСТУП	10
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Аналіз вимог до автоматизованих систем виявлення малярії на основі зображень	12
1.2 Біомедичні аспекти малярії та характеристика збудника.....	13
1.3 Сучасні методи діагностики малярії	16
1.3.1 Клінічна діагностика	17
1.3.2 Мікроскопічне дослідження мазків крові	17
1.3.3 Швидкі діагностичні тести (ШДТ)	18
1.3.4 Полімеразна ланцюгова реакція (ПЛР)	19
1.3.5 Петлеподібна ізотермічна ампліфікація (LAMP)	19
1.3.6 Інші діагностичні підходи	20
1.4 Існуючі методи автоматизованої діагностики малярії та класифікації клітин крові	20
1.4.1 Класичні алгоритми машинного навчання для автоматизованої діагностики малярії	23
1.4.2 Використання згорткових нейронних мереж (CNN) для діагностики малярії	24
1.4.3 Використання трансформерів для діагностики малярії	26
1.5 Висновки до першого розділу	29
РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ ТА АРХІТЕКТУРА МОДЕЛІ DETECTION TRANSFORMER	30
2.1 Теоретичні основи глибокого навчання для комп'ютерного зору	30
2.2 Архітектура Detection Transformer (DETR)	31
2.2.1 Загальна архітектура DETR	32

2.2.1.1 Архітектура CNN в DETR	33
2.2.1.2 Архітектура трансформера в DETR	34
2.2.1.1 Архітектура feed-forward networks в DETR	37
2.2.2 Функція втрат у DETR	38
2.2.2.1 Оптимальне зіставлення прогнозів	38
2.2.2.2 Роль Hungarian's алгоритму	39
2.2.2.3 Втрати локалізації об'єктів	40
2.3 Переваги використання DETR для біомедичних задач	40
2.4 Інтерпретація уваги в трансформерах	42
2.5 Метрики оцінювання якості моделі детекції	44
2.5.1 Метрики для задач об'єктної детекції	45
2.5.2 Особливості оцінювання для медичних зображень	47
2.5.2.1 Вплив точності локалізації на якість клінічних діагностичних рішень	47
2.5.2.2 Класовий дисбаланс і його вплив на оцінювання .	48
2.5.2.3 Обмеженість обсягів даних і вплив на метрики ...	48
2.5.2.4 Унікальні аспекти метрик у медичних задачах	49
2.6 Висновки до другого розділу	51
РОЗДІЛ 3. ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ	53
3.1 Отримання та корекція набору даних	53
3.2 Конвертація анотацій у формат COCO та підготовка вхідних зображень	55
3.3 Тренування моделі Detection Transformer (DETR)	58
3.4 Аналіз результатів тренування	61
3.4.1 Динаміка втрат	61
3.4.2 Точність детекції	65
3.4.3 Візуалізація передбачень	68
3.5 Порівняння з існуючими роботам	70
3.5.1 Методологія порівняння	70
3.5.2 Результати	71

3.6 Висновки до третього розділу	73
РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	74
4.1 Постановка задачі	74
4.2 Обґрунтування функцій програмного продукту	75
4.3 Обґрунтування системи параметрів програмного продукту	78
4.4 Аналіз експертного оцінювання параметрів	80
4.5 Аналіз рівня якості варіантів реалізації функцій	83
4.6 Економічний аналіз варіантів розробки програмного продукту ...	84
4.7 Висновки до четвертого розділу	89
ВИСНОВКИ	90
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	92
ДОДАТОК А ПРЕЗЕНТАЦІЯ	102
ДОДАТОК Б ЛІСТИНГ КОДУ	113

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ML (Machine Learning) — машинне навчання.

CV (Computer Vision) — комп'ютерний зір.

CNN (Convolutional Neural Network) — згорткова нейронна мережа.

U-Net — архітектура нейронної мережі для сегментації зображень.

DACNN (Dual Attention CNN) — згорткова нейронна мережа з подвійною увагою.

VGG-16 — глибока згорткова нейронна мережа з 16 шарами.

ResNet-50 — глибока нейронна мережа з 50 шарами, що використовує залишкові зв'язки для подолання проблеми зникнення градієнту.

FFN (Feed-Forward Network) — прямопередавальна нейронна мережа.

DETR (Detection Transformer) — трансформер для детекції об'єктів.

RT-DETR (Real-Time DETR) — вдосконалена та пришвидшена версія DETR, що орієнтована на реальний час.

Faster R-CNN (Faster Region-Based Convolutional Neural Network) — мережа для детекції об'єктів, яка використовує регіони інтересу.

YOLO (You Only Look Once) — модель детекції об'єктів, яка виконує розпізнавання за один прохід через мережу.

YOLOv3 — третя версія YOLO.

Swin-T (Swin Transformer) — віконна трансформерна архітектура.

GAN (Generative Adversarial Network) — генеративна змагальна мережа.

NMS (Non-Maximum Suppression) — алгоритм для відсікання зайвих передбачень при детекції об'єктів.

COCO (Common Objects in Context) — датасет для задач детекції, сегментації та класифікації об'єктів.

WHO, BOOЗ (World Health Organization) — Всесвітня організація охорони здоров'я.

ВСТУП

Упродовж останніх десятиліть автоматизація процесів у медицині стала невід'ємною частиною розвитку систем охорони здоров'я. Однією з найбільших, динамічних і перспективних галузей в цьому контексті є автоматизований аналіз медичних зображень. Зокрема, методи комп'ютерного зору та глибокого навчання, які демонструють високу ефективність при виявленні патологічних змін у різних типах медичних знімків: МРТ, КТ, рентгенограмах, гістологічних препаратах тощо. Одним із критично важливих напрямів застосування таких технологій є діагностика паразитарних інфекцій, зокрема малярії.

Малярія (детальніше у п. 1.2) — інфекційне захворювання, яке передається, здебільшого, через укуси заражених комарів роду *Anopheles*, спричинене паразитами роду *Plasmodium*. За даними Всесвітньої організації охорони здоров'я, щороку фіксується понад 200 мільйонів випадків малярії, велика частина з яких закінчується летально (400 000 смертей на рік), особливо в країнах Африки. Одним із поширених збудників хвороби є *Plasmodium vivax* — паразит, який викликає персистуючу форму малярії і здатен залишатися в організмі пацієнта тривалий час, спричиняючи рецидиви.

Традиційний метод діагностики малярії передбачає візуальний аналіз мазків крові, пофарбованих за методом Романовського-Гімзи або Wright-Giemsa, під мікроскопом. Медичний працівник вручну ідентифікує інфіковані клітини крові серед великої кількості здорових, що вимагає високої кваліфікації, значного досвіду та займає досить багато часу. У реальних умовах, особливо в регіонах з обмеженими ресурсами, якість такого аналізу може бути низькою, що, само собою, знижує точність діагностики. Отже, постає потреба у застосуванні автоматизованих підходів до виявлення та сегментації заражених клітин.

Завдяки стрімкому розвитку методів глибокого навчання, зокрема трансформерних архітектур, з'явилися нові підходи до побудови високоточних моделей для аналізу зображень. Однією з ключових інновацій у цій галузі є модель Detection Transformer (DETR), яка адаптує механізм самоуваги трансформерів до задач комп'ютерного зору. DETR здійснює детекцію об'єктів як задачу прямого прогнозування фіксованого набору об'єктів без використання традиційних компонентів, таких як anchor boxes або багатоступеневі механізми постобробки. Даний підхід дозволить зменшити складність архітектури, покращити узагальнювальну здатність моделі та забезпечить кращу інтерпретованість результатів.

Метою даної роботи є дослідити можливості використання архітектури DETR для задачі автоматичної класифікації та сегментації клітин крові на зображеннях.

Основними завданнями роботи є:

- 1) аналіз та підготовка датасету до навчання (включаючи анотації у форматі COCO для сумісності з DETR);
- 2) модифікація та адаптація моделі DETR до медичної задачі сегментації;
- 3) донавчання моделі на зображеннях з інфікованими клітинами;
- 4) оцінка точності класифікації та якості сегментації на тестовому наборі;
- 5) порівняння результатів із класичними підходами, що базуються на конволюційних нейронних мережах (наприклад, U-Net, Faster R-CNN тощо).

Таким чином, дана робота є спробою поєднати сучасні методи комп'ютерного зору з актуальною медичною задачею. Результати дослідження можуть мати практичне значення у контексті створення систем підтримки діагностики малярії та, ширше, паразитарних захворювань, що потребують точного аналізу мікроскопічних зображень.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог до автоматизованих систем виявлення малярії на основі зображень

Сучасна діагностика малярії, як правило, базується на мікроскопічному дослідженні мазків крові, що вимагає висококваліфікованих фахівців і є трудомістким процесом. З огляду на велику кількість зображень, які необхідно перевірити, а також можливу втому або людський фактор, зростає потреба у розробці ефективних автоматизованих систем підтримки діагностики, які могли б допомогти виявляти інфіковані клітини з високою точністю.

В межах даного дослідження основною вимогою до такої системи є класифікація клітин крові за двома категоріями: інфіковані (клітини, що містять паразитів *Plasmodium vivax*) та неінфіковані (звичайні еритроцити або лейкоцити). Цей підхід дозволяє суттєво спростити задачу порівняно з багатокласовими схемами, водночас зберігаючи її практичну цінність для попереднього автоматизованого скринінгу.

Для реалізації системи класифікації було використано відкритий набір даних BBBC041: *P. vivax*-infected human blood smears, доступний на сайті Broad Bioimage Benchmark Collection [1]. Цей датасет містить зображення клітин, промаркованих за типом.

З технічної точки зору система не передбачає наявності графічного інтерфейсу. Вся взаємодія здійснюється шляхом запуску відповідних Python-скриптів, які реалізують повний цикл роботи з моделлю: навчання на тренувальній вибірці, валідацію та тестування на нових зображеннях. Основна увага приділяється точності класифікації та здатності моделі узагальнювати знання на зображення, які не входили до навчальної вибірки.

Вибір моделі DETR (Detection Transformer) як основи обумовлений її здатністю ефективно працювати із задачами, пов'язаними з детекцією та

класифікацією на зображеннях без потреби у складній постобробці. Використання трансформерної архітектури дозволяє забезпечити високу точність при одночасному зменшенні залежності від гіперпараметрів, як-от розмірів anchor boxes, які характерні для класичних моделей комп'ютерного зору.

Таким чином, основні вимоги до системи можна сформулювати так:

- 1) використання зображень мазків крові високої якості з відповідною анотацією;
- 2) системи у вигляді скриптів, що дозволяють запускати тренування та тестування без інтерфейсу;
- 3) застосування сучасної глибокої моделі комп'ютерного зору (DETR), яка дозволяє здійснювати точну детекцію та класифікацію клітин;
- 4) можливість масштабування на більші набори даних або розширення до багатокласової класифікації в майбутньому.

1.2 Біомедичні аспекти малярії та характеристика збудника *Plasmodium vivax*

Малярія — захворювання, яке вражає людство з давніх часів. Незважаючи на розвиток медицини, вона залишається однією з провідних причин захворюваності та смертності у країнах з тропічним та субтропічним кліматом. За даними Всесвітньої організації охорони здоров'я (ВООЗ), у 2023 році було зареєстровано близько 263 мільйонів випадків малярії у 83 країнах світу, з яких понад 597 тисяч завершилися летально. Близько 95% смертей припадає на країни Африканського континенту, що свідчить про нерівномірний розподіл навантаження хвороби [2]. У відповідь на масштабність проблеми було започатковано низку міжнародних ініціатив, таких як програма malERA (Malaria Eradication Research Agenda), спрямована

на розробку науково обґрунтованих стратегій для ліквідації та повного викорінення малярії [3]. Дослідження у цій галузі охоплюють як епідеміологічні аспекти, так і розробку нових інструментів для діагностики, лікування та профілактики захворювання. Малярію у людей спричиняють паразити роду *Plasmodium*, серед яких клінічно найзначущими є *P. falciparum* та *P. vivax*, які разом відповідають за понад 95% усіх випадків захворювання. *P. falciparum* є найбільш смертоносним видом, водночас *P. vivax* має іншу специфіку перебігу: він здатен спричиняти рецидиви захворювання через наявність латентної печінкової форми (гіпнозоїтів), що ускладнює лікування та контроль інфекції. Крім того, існують інші види малярійних паразитів, такі як *P. malariae*, *P. knowlesi*, *P. ovale wallikeri* та *P. ovale curtisi* [4]. Проте на глобальному рівні вони становлять значно меншу частку інфекцій.

Найбільш поширеним збудником важкої форми малярії є *Plasmodium falciparum*, який домінує в регіоні Африки, де припадає понад 90% усіх зафіксованих випадків захворювання. Саме тому це найкраще охарактеризований вид [5] з точки зору патогенезу, клінічної картини та стратегії лікування.

Однак в останні роки інфекції, спричинені *P. vivax*, набувають дедалі більшого значення. Новітні дослідження продемонстрували, що *P. Vivax* [6] також здатен спричиняти тяжкі форми захворювання, включаючи ускладнення, які раніше асоціювалися виключно з *P. falciparum*. Ключова особливість *Plasmodium vivax* полягає в тому, що він здатен завершити свій розвиток у тілі комара при нижчих температурах і за коротший проміжок часу, ніж *P. falciparum*, що суттєво впливає на географію його поширення. Саме через це *P. vivax* вважається найбільш географічно поширеним видом: випадки зараження фіксуються в Латинській Америці, Південній та Південно-Східній Азії, на Близькому Сході, Африканському Розі та в регіоні Океанії [7].

Значна увага, яку в останні роки приділяють *P. vivax*, має як науковий, так і прикладний характер. Це підтверджується зростанням кількості наукових публікацій, присвячених цій тематиці (рис.1) та підвищеним інтересом з боку

систем охорони здоров'я щодо удосконалення діагностики та розробки специфічних підходів до лікування цього виду малярії. Однією з головних проблем, пов'язаних із *P. vivax*, є його здатність викликати численні рецидиви [8]. Це пояснюється наявністю гіпнозоїтів — латентної форми паразита, що зберігається у печінці та активується через тривалий період після первинної інфекції. Така особливість значно ускладнює елімінацію паразита з організму та вимагає застосування специфічних медикаментозних схем лікування.

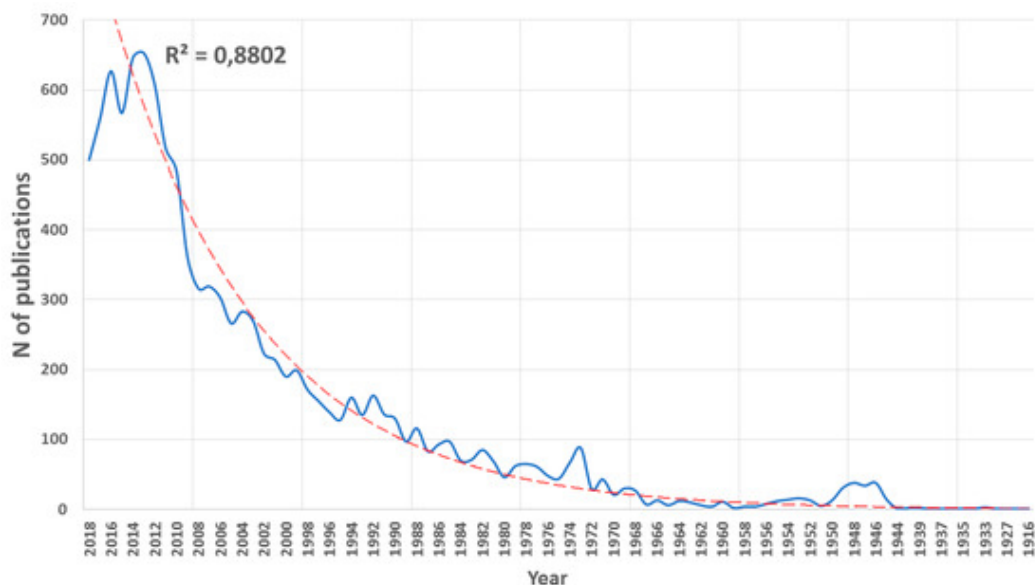


Рисунок 1.1 — Тенденція кількості публікацій з малярії та *Plasmodium vivax* за період з 1916 по 2018 рік

P. vivax, як і інші види *Plasmodium*, заражається від укусу інфікованої самки комара *Anopheles*, яка вводить у кровоносну систему спорову форму паразита — спорозоїти [9]. Якщо комар кусає заражену людину, гаметоцити потрапляють у його організм, де проходять цикл статевого розмноження в середовищі середньої кишки, утворюючи зиготи, а згодом — ооцити. Після дозрівання ооцити розриваються, вивільняючи нові спорозоїти, які мігрують до слинних залоз комара. Саме ці спорозоїти є інфекційною формою, яка передається наступній людині при укусі [10].

Життєвий цикл даного паразита, який показаний на рис. 1.2, є надзвичайно складним, що, у поєднанні з високим рівнем генетичної варіативності та здатністю ухилятися від імунної відповіді організму людини,

значно ускладнює розробку ефективної вакцини [11]. Також, *P. vivax* часто проявляється у вигляді інфекцій із низькою паразитемією, що значно ускладнює своєчасну діагностику та початок лікування [12]. Відсутність надійних експериментальних моделей для тестування вакцинних стратегій лише посилює ці труднощі. Утім, активні міжнародні зусилля у сфері досліджень та розробки, зокрема за ініціативи Всесвітньої організації охорони здоров'я (ВООС), дають підстави для обережного оптимізму щодо створення ефективної вакцини проти *P. vivax*, що стало одним із ключових напрямів сучасної паразитології та глобальної стратегії боротьби з малярією.

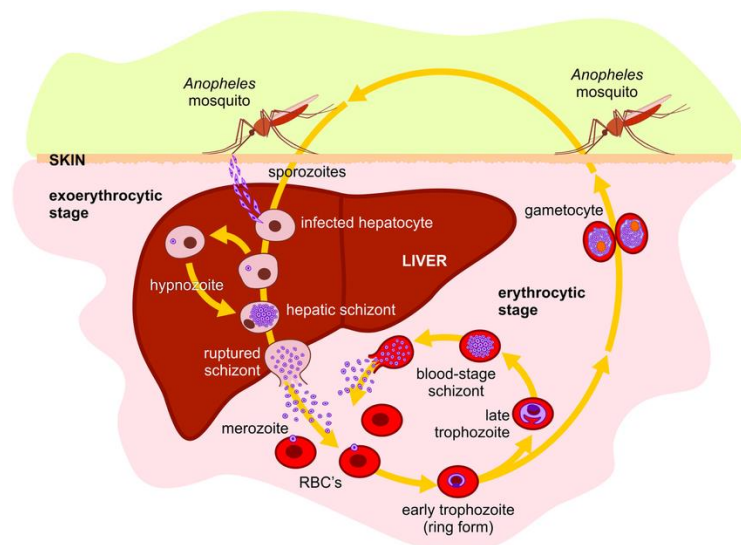


Рисунок 1.2 — Життєвий цикл *Plasmodium vivax* у людини-хазяїна [13]

1.3 Сучасні методи діагностики малярії

Діагностика малярії має вирішальне значення для лікування та ліквідації інфекцій. Лабораторна діагностика прийнята в усьому світі та рекомендована для виявлення малярії [14]. Методи діагностики інфекційних захворювань повинні бути швидкими, точними, простими та доступними [15]. Існує кілька основних підходів до діагностики малярії, які розрізняються за рівнем точності, швидкістю отримання результатів, вартістю та вимогами до обладнання й кваліфікації персоналу.

1.3.1 Клінічна діагностика

Клінічна діагностика є одним із найпростіших та найменш витратних методів виявлення малярії, особливо у регіонах з обмеженими ресурсами [16]. Вона ґрунтується на оцінці симптомів пацієнта, анамнезі, географічному походженні, епідеміологічній ситуації, сезонності та віковій групі. Такий підхід може мати певну цінність у первинному скринінгу, однак не може забезпечити достатню точність для остаточного діагнозу.

Зважаючи на велику ймовірність помилок у клінічному діагнозі через його суб'єктивність та неспецифічність, цей метод доцільно використовувати лише як попередній етап. Для підтвердження наявності інфекції необхідне лабораторне дослідження, зокрема аналіз мазків крові або інші методи візуалізації паразитів *Plasmodium*.

1.3.2 Мікроскопічне дослідження мазків крові

Як вважається, мікроскопія мазків крові залишається «золотим стандартом» у діагностиці малярії завдяки своїй високій специфічності та можливості прямої візуалізації паразитів *Plasmodium* у крові [17]. Метод передбачає виготовлення товстого та тонкого мазків крові з подальшим фарбуванням, найчастіше за методами Гімзи або Лейшмана, які дозволяють чітко ідентифікувати паразитарні структури завдяки характерному кольоровому забарвленню [18].

Тонкий мазок дозволяє виявити морфологічні ознаки різних видів *Plasmodium*, тоді як товстий мазок забезпечує вищу чутливість, дозволяючи виявляти навіть низькі рівні паразитемії. Комбіноване використання обох типів мазків дозволяє не лише підтвердити наявність малярійної інфекції, але

й провести її видову диференціацію та оцінити ступінь зараження [19].

Попри ефективність, метод має суттєві обмеження. Мікроскопічне дослідження є трудомістким процесом, що потребує висококваліфікованих спеціалістів, досвіду та якісного лабораторного обладнання. Помилки в інтерпретації можуть призвести як до хибнонегативних результатів (що затримує лікування), так і до хибнопозитивних (що спричиняє непотрібне призначення протималарійних препаратів і їхні побічні ефекти) [20].

1.3.3 Швидкі діагностичні тести (ШДТ)

Швидкі діагностичні тести (ШДТ) є важливим засобом для виявлення інфекцій, спричинених паразитами роду *Plasmodium*, особливо у випадках, коли ресурси обмежені або неможливо оперативно виконати мікроскопічний аналіз. В основі ШДТ лежить метод імунохроматографії, який дозволяє візуально зафіксувати взаємодію антигену з антитілом у вигляді кольорової смуги на нітроцелюлозному носії [21]. До ключових переваг таких тестів належить оперативність — результат зазвичай можна отримати протягом 15—30 хвилин, що робить їх надзвичайно корисними в польових умовах [22].

Водночас ШДТ мають певні обмеження. Одним з них є прозонний ефект — явище, при якому надмірна кількість антигену заважає утворенню імунного комплексу на тестовій смужці, що може призвести до хибнонегативного результату [23]. Хибнопозитивні результати трапляються рідше, проте можливі, зокрема через перехресну реактивність з іншими патогенами або при надмірному рівні паразитемії [24].

1.3.4 Полімеразна ланцюгова реакція (ПЛР)

Полімеразна ланцюгова реакція (ПЛР) є високочутливим та високоспецифічним методом молекулярної діагностики, що ґрунтується на ампліфікації ДНК паразита у зразку крові. Цей підхід дозволяє не лише виявляти інфекцію на дуже ранніх стадіях, але й точно ідентифікувати вид паразита, включаючи випадки змішаних інфекцій, що є особливо важливим у регіонах з високою епідеміологічною варіабельністю. У порівнянні з традиційною мікроскопією, ПЛР забезпечує значно вищу чутливість, особливо у випадках з низьким рівнем паразитемії або у безсимптомних носіїв [25].

Попри очевидні переваги, метод має обмеження. Застосування ПЛР потребує спеціалізованого лабораторного обладнання, підготовленого персоналу, а також значного часу на проведення аналізу (від 2 до 3 годин) і високих фінансових витрат. Тому його використання у віддалених або малорозвинених регіонах залишається обмеженим.

1.3.5 Петлеподібна ізотермічна ампліфікація (LAMP)

Петлеподібна ізотермічна ампліфікація (LAMP) є альтернативним молекулярним методом ампліфікації ДНК, який виконується при постійній температурі. Цей підхід базується на активності термостабільної ДНК-полімерази з *Bacillus stearothermophilus* і дозволяє швидко ампліфікувати нуклеїнові кислоти паразитів [26].

LAMP-тести мають високу діагностичну ефективність, зі звітами про чутливість до 99% та специфічність 93% при порівнянні з мікроскопією. Попри це, однією з основних технічних вимог залишається необхідність

використання флуоресцентного детектора для зчитування результатів. Це обмежує широке впровадження методу у сільських та віддалених регіонах.

1.3.6 Інші діагностичні підходи

Окрім вище розглянутих методів, для діагностики малярії також застосовуються інші способи виявлення малярії, які мають певні переваги залежно від мети обстеження та умов використання. До таких підходів належать серологічні методи, проточна цитометрія, біомаркери та кількісний метод «баффі-шерсті». Ці методи, хоча й не використовуються рутинно для первинної діагностики, демонструють потенціал у сфері епідеміологічного моніторингу, досліджень серопревалентності, диференційованої діагностики та новітніх технологій швидкого виявлення інфекції. Їх застосування доповнює традиційні та молекулярні методи, сприяючи комплексному підходу до виявлення та контролю малярії.

1.4 Існуючі методи автоматизованої діагностики малярії та класифікації клітин крові

Розвиток технологій комп'ютерного зору та машинного навчання відкрив нові можливості для автоматизованої діагностики інфекційних захворювань. Різні типи зображень, такі як рентген, КТ, МРТ активно використовуються для виявлення патологій. У випадку діагностики малярії все частіше застосовуються автоматизовані системи, що аналізують мікроскопічні зображення мазків крові. Такий підхід передбачає використання методів CV та алгоритмів ML для виявлення паразитів на цифрових

зображеннях тонких або товстих мазків.

Типовий процес включає декілька послідовних етапів: отримання зображень, їх попередню обробку, сегментацію клітин крові, виділення ознак та подальшу класифікацію клітин на заражені та не заражені.

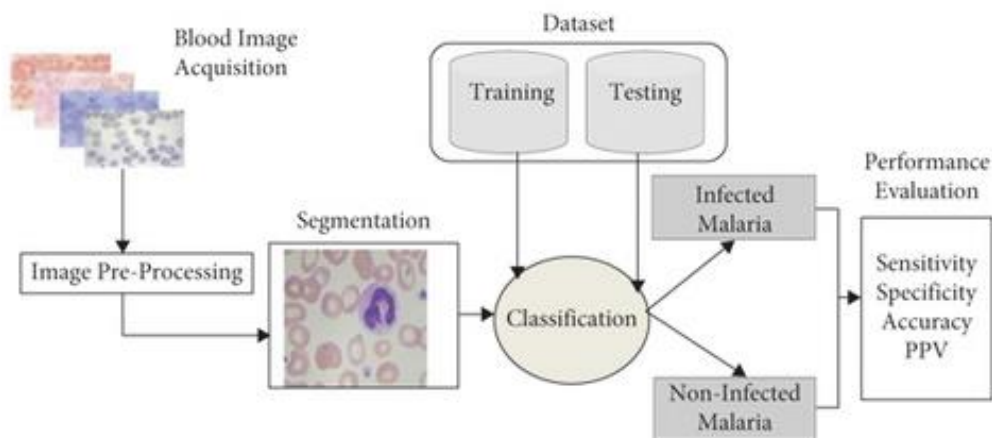


Рисунок 1.2 — Покроковий процес автоматизованих обчислювальних методів діагностики малярійних паразитів (взято з [27])

Отримання зображень є першим етапом у процесі автоматизованого виявлення малярії. Даний етап передбачає використання цифрових зображень мазків крові, які потім використовуються для подальшого аналізу та класифікації паразитів. Зразок крові кожного пацієнта розподіляється на дві частини: зображення товстого та тонкого мазків крові. На рис. 1.3 показано зображення товстих та тонких мазків крові. Товстий мазок крові — це крапля крові, яка допомагає виявити наявність паразитів, а тонкий мазок крові — це шар крові, який наноситься на предметне скло та допомагає ідентифікувати вид паразита, що викликає інфекцію. Більшість досліджень використовують цифрові зображення тонких мазків крові, і дуже мало дослідників працювали над зображеннями товстих цифрових мазків крові [28].

Для отримання зображень найчастіше використовують світлову мікроскопію, при цьому зображення фіксуються за допомогою вбудованих у мікроскопи цифрових камер. Інші методи, такі як флуоресцентна або поляризована мікроскопія, використовуються рідше, але також можуть бути

корисними в окремих випадках. Важливими факторами, які визначають подальші етапи обробки зображень та точність виявлення паразитів, є якість отриманих зображень, зокрема роздільна здатність та чіткість.

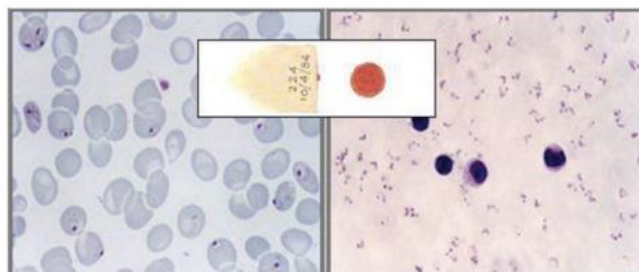


Рисунок 1.3 — Мікроскопічні зображення товстого (праворуч) та тонкого (ліворуч) мазка крові (взято з [29])

Попередня обробка зображень використовується в традиційних методах комп'ютерного зору для автоматичного виявлення паразитів. Вона дозволяє покращити якість вхідних зображень, усунути зайві артефакти та забезпечити узгодженість у даних, що надходять до системи. Серед основних процедур цього етапу — зменшення шуму [30], покращення контрастності, нормалізація кольору, корекція освітлення [31] та приведення зображень до єдиного розміру. Дані заходи дозволяють зробити зображення більш придатними для точного виявлення ознак паразитів, особливо у випадках варіативного забарвлення або недостатньої якості вихідних зразків.

Для вилучення ознак часто потрібна сегментація зображень, яка полягає у виділенні окремих об'єктів або регіонів інтересу на зображенні, таких як еритроцити або паразити. Основна мета цього етапу — локалізація та поділ структур, що підлягають подальшому аналізу. Для сегментації зображень мазків крові використовуються різноманітні алгоритми, зокрема морфологічні операції, методи порогового визначення, алгоритм вододілу [32], кластеризація методом k-середніх [33], перетворення Хафа [34] та інші класичні підходи [35]. Залежно від якості зображень та особливостей зразків можуть застосовуватись комбіновані або маркерні методи для покращення точності розділення клітинних структур.

Після сегментації зображень наступним важливим етапом є вилучення ознак, яке має на меті представити сегментовані об'єкти у вигляді інформативних та дискримінативних дескрипторів. Метою цього процесу є зниження розмірності вхідних даних, зменшення обчислювальної складності та забезпечення точності під час подальшої класифікації.

Ознаки можуть відображати різні характеристики зображення, наприклад, для виокремлення текстурних ознак, таких як ентропія, однорідність та середнє значення, часто використовується матриця співпадінь рівнів сірого (GLCM) [36]. Інтенсивнісні ознаки можуть бути отримані на основі кольорових каналів [37], а морфологічні параметри — за допомогою гранулометричного аналізу [38].

Якість та тип вилучених ознак значною мірою впливають на вибір і ефективність методів класифікації, що використовуються для остаточного розпізнавання паразитів. У наступному підрозділі буде розглянуто найбільш поширені підходи до автоматизованої діагностики малярії, зокрема методи на основі класичних алгоритмів машинного навчання (таких як дерева рішень або SVM), а також сучасні глибокі моделі — CNN-архітектури (AlexNet, ResNet, DenseNet) та трансформерні підходи, які демонструють високу точність у задачах класифікації медичних зображень.

1.4.1 Класичні алгоритми машинного навчання для автоматизованої діагностики малярії

Один із ранніх підходів до автоматизованого виявлення малярії на мікроскопічних зображеннях передбачає використання класичних методів машинного навчання, таких як дерева рішень, метод опорних векторів (SVM), наївний баєсівський класифікатор, а також кластеризаційні алгоритми.

Одним із найбільш показових прикладів є дослідження Fatima і Farid

(2020) [39], де автори запропонували метод, який використовує попередню обробку зображень для покращення якості та зменшення шуму. Зокрема, для зменшення шуму застосовували двосторонню фільтрацію, що дозволяє досягти кращої якості зображень мазка крові. Окрім цього, для виявлення паразитів застосовувався адаптивний метод порогового визначення, що допомагає точно виділити паразитів у клітинах крові. Для оцінки ефективності методу дослідники протестували його на наборі даних NIH Malaria, де результат виявлення паразитів перевищив 91%, що свідчить про високий рівень точності в порівнянні з іншими існуючими підходами. Такий підхід є корисним для медичних працівників, таких як патологоанатоми та гематологи, оскільки дозволяє точніше виявляти малярійних паразитів, що є важливим для правильного діагностування та лікування малярії.

1.4.2 Використання згорткових нейронних мереж (CNN) для діагностики малярії

В останні роки використання глибоких навчальних моделей, зокрема згорткових нейронних мереж (CNN), стало поширеним підходом для автоматизованої діагностики медичних зображень, в тому числі і малярії. Ці моделі здатні автоматично вивчати важливі ознаки зображень, знижуючи необхідність ручного визначення ознак і значно покращуючи точність та ефективність виявлення малярійних паразитів.

ЗНН складаються з кількох типів шарів: згорткових, об'єднуючих (pooling), повнозв'язних та нормалізаційних. На ранніх етапах модель ідентифікує базові патерни, як-от краї, кольори чи текстури; на пізніших — формує вищі абстракції, які дозволяють класифікувати об'єкти [40]. Активаційні функції керують передачею сигналу між нейронами, а регуляризаційні методи (наприклад, L2 або Dropout) зменшують ризик

перенавчання, яке часто виникає в умовах обмеженого обсягу анотованих даних.

З метою ефективного навчання CNN моделі потребують якісно анотованих даних. Типовий процес включає розділення даних на тренувальний та валідаційний. Ця ітеративна процедура дозволяє оптимізувати гіперпараметри та контролювати узагальнюючу здатність моделі [41].

Одним із класичних прикладів ефективного використання CNN у контексті малярії є дослідження Rajaraman et al. (2020) [42]. У ньому були протестовані декілька популярних архітектур — AlexNet, VGG-16, ResNet-50 — для класифікації клітин крові на заражені та незаражені. Архітектура ResNet-50 показала найвищу точність (до 96%), значно перевищивши класичні методи, що базуються на пороговій фільтрації або морфологічній обробці. Модель навчалась на зображеннях тонкого мазка крові та демонструвала здатність до генералізації на нових даних.

В іншій роботі Oyewola et al. (2022) [43] було представлено модифіковану CNN-модель DACNN, яка поєднує традиційне згорткове ядро з розширеними шарами детекції для більш точного виділення паразитів. Цей підхід також включав методи аугментації даних, що компенсували обмежений обсяг наявних анотованих зображень.

Окрім простої класифікації зображень, сучасні CNN застосовуються для виявлення об'єктів (object detection), що дозволяє локалізувати паразитів безпосередньо на зображенні. У цьому контексті виділяють дві групи архітектур.

1. Двоступеневі детектори, такі як Faster R-CNN [44] та SPPnet [45], які спочатку генерують регіони інтересу, а потім класифікують їх. Вони демонструють високу точність і використовуються в клінічних задачах, де важлива кожна помилка.
2. Одноступеневі детектори, як-от YOLOv3/v4/v5 [46], що дозволяють здійснювати швидку і точну локалізацію паразитів у реальному часі.

Їхнє застосування особливо перспективне для створення мобільних рішень для віддалених регіонів з високим рівнем захворюваності на малярію.

Попри всі переваги, CNN вимагають великих обсягів анотованих даних, яких бракує в медичній сфері. Наявні публічні набори даних, як-от NIH Malaria Dataset або набір зображень мазків товстої крові з IEEE DataPort (2021), є критично важливими для розвитку та тестування даних моделей. Для подолання обмежень у кількості даних застосовуються методи трансферного навчання, які дозволяють адаптувати заздалегідь навчені моделі на нові, вузькоспеціалізовані набори [47].

Крім того, сучасні роботи, зокрема Ali et al. (2024) [48], комбінують CNN з трансформерами для покращення інтерпретованості та стабільності моделей, що дозволяє зменшити ймовірність помилкової класифікації.

1.4.3 Використання трансформерів для діагностики малярії

Упродовж останніх років трансформери (Transformer-based architectures) [49] набули великого поширення в комп'ютерному баченні, зокрема завдяки успіху в обробці природної мови. Однією з найперспективніших трансформерних моделей для задач зору є Swin Transformer [50] — покращення оригінальної моделі Vision Transformer (ViT) [51]. Swin Transformer (Shifted Window Transformer) запропоновано як ієрархічну трансформерну архітектуру, що дозволяє виконувати обчислення в локальних областях із поступовим злиттям інформації у вищих шарах мережі. Це забезпечує як збереження локального контексту, так і глобальне агрегування ознак при лінійній складності по розміру зображення. Порівняно з Vision Transformer (ViT) [52], який йому попередник, Swin-трансформер є високоефективним і точнішим. Завдяки цим властивостям Swin-трансформер

служить основою в багатьох моделях на основі зору сьогодні.

На відміну від класичних трансформерів, які працюють з фіксованими патчами по всьому зображенню, Swin Transformer використовує механізм зсуву вікон (shifted windows) для міжвіконної взаємодії, щоб перемістити ізольовані ділянки у вікна з некомпетентними ділянками. Таким чином, вікна можуть складатися з ділянок, які не суміжні на вихідній карті ознак, і використовується механізм маскування, щоб зменшити обчислення самоаналізу в межах кожного підвікна. Це циклічне зсувування вводить перехресні зв'язки між вікнами та покращує продуктивність мережі, зберігаючи при цьому ефективність таку ж, як і при звичайному розділенні вікон.

На рисунку 1.4 показано архітектуру Swin-transformer (Swin-T).

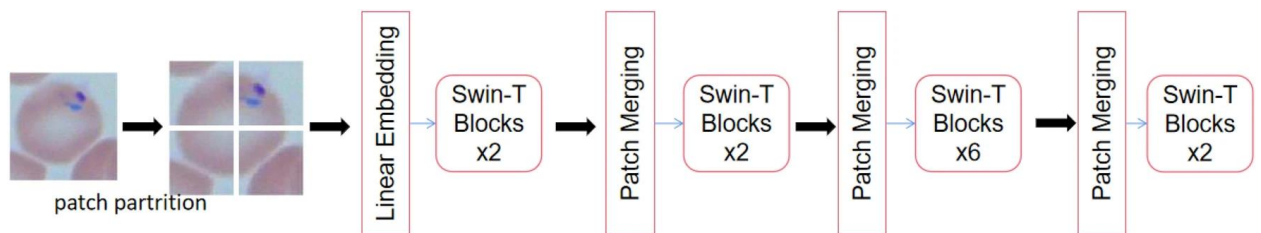


Рисунок 1.4 — Архітектура Swin-transformer

У роботі Tan та Liang (2023) [53] було продемонстровано, що Swin Transformer може бути ефективно використаний для класифікації паразитів малярії. Автори поєднали трансформерну архітектуру з генеративно-змагальною мережею WGAN-GP для створення змішаного набору даних: до розширених (аугментованих) зображень було додано синтетичні зображення, згенеровані GAN. В результаті модель на основі Swin Transformer досягла точності 99,867% у задачі багатокласової класифікації клітин, інфікованих *Plasmodium vivax*.

В експериментальних результатах ця модель перевершила інші методи за точністю, повнотою, специфічністю, F1-метрикою та показала найнижчий рівень хибнопозитивних результатів. Проте, вона поступалася MobileViT та MobileNet за часом виведення та використанням пам'яті, що має значення для

пристроїв із обмеженими ресурсами, як-от IoT чи смартфони.

Іншою важливою трансформерною архітектурою є YOLO-mp, яку Койрала та ін. [54] адаптували до задач класифікації видів *Plasmodium* та стадій прогресування інфекції. Модель досягла точності 78,5% у класифікації видів малярії, але лише mAP 0,399 у класифікації стадій (кільце, трофозоїт, шизонт, гаметоцит). Водночас дослідники відзначили суттєві проблеми з класифікацією при перекритті клітин, а також незбалансованість класів у наборі даних.

Ще один приклад використання трансформерів — робота Guemas et al. (2023) [55], в якій використовувалась модель на основі DETection TRansformer (DETR). Автори досягли загального mAP 63,8%, але відзначили суттєве зниження точності для видів *P. ovale* (19,9%) та *P. malariae* (15%), що вказує на проблему дисбалансу та складність класифікації рідкісних форм.

Також у дослідженні Adigun et al. (2022) [56] запропоновано використання Compact Convolutional Transformer (CCT), який поєднує згорткові шари для початкового вилучення ознак із трансформерним блоком. CCT досяг точності 96,41% на оригінальному датасеті та 99,25% на обробленому наборі, продемонструвавши конкурентоспроможність у задачах діагностики малярії.

Таким чином, трансформерні архітектури демонструють значний потенціал у задачах автоматизованої діагностики малярії. Їхні переваги — здатність до ефективного вилучення просторово-контекстних ознак, адаптивність до різних масштабів зображення та сумісність із сучасними методами аугментації й генерації даних — роблять їх особливо перспективними для аналізу мікроскопічних зображень крові. Попри певні виклики, зокрема складнощі з класифікацією рідкісних видів *Plasmodium* та обробкою перекритих клітин, трансформерні моделі залишаються на передовій сучасних підходів до медичної діагностики.

1.5 Висновки до першого розділу

У першому розділі було сформульовано постановку задачі автоматизованої діагностики паразитарної інфекції *Plasmodium vivax* на основі аналізу мікроскопічних зображень мазків крові. Було розглянуто клінічну природу малярії, зокрема особливості збудника *P. vivax*, шляхи зараження, стадії розвитку паразита в еритроцитах та важливість своєчасної та точної діагностики.

Розділ також містив огляд сучасних підходів до обробки медичних зображень, включаючи класичні методи машинного навчання, згорткові нейронні мережі (CNN) та сучасні трансформерні архітектури. CNN-моделі продемонстрували високу ефективність при класифікації інфікованих клітин, однак обмеженість у врахуванні глобального контексту та необхідність великої кількості анованих даних мотивували до пошуку альтернатив.

Особливу увагу було приділено трансформерам, зокрема архітектурам Vision Transformer (ViT), Swin Transformer та іншим модифікаціям, які показали високі результати у задачах сегментації та класифікації малярійних паразитів, включаючи мультикласову класифікацію за видами та стадіями розвитку. Розглянуто приклади застосування генеративно-змагальних мереж (GAN) для балансування датасетів і покращення навчання моделей.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ ТА АРХІТЕКТУРА МОДЕЛІ DETECTION TRANSFORMER

2.1 Теоретичні основи глибокого навчання для комп'ютерного зору

Глибоке навчання (англ. Deep Learning) є підгалуззю машинного навчання, що базується на використанні штучних нейронних мереж зі значною кількістю шарів. Його поява та стрімкий розвиток істотно вплинули на сферу комп'ютерного зору, дозволивши досягати раніше недосяжного рівня точності у таких задачах, як класифікація зображень, сегментація, детекція об'єктів та інші.

На відміну від класичних алгоритмів, які ґрунтуються на ручному конструюванні ознак, глибокі нейронні мережі здатні автоматично навчатися ієрархічних представлень вхідних даних.

Однією з найбільш впливових архітектур у галузі комп'ютерного зору є згорткові нейронні мережі (CNN). Вони були основним інструментом упродовж майже десятиліття завдяки здатності ефективно вилучати локальні просторові ознаки. Однак CNN мають обмеження у моделюванні глобального контексту зображення, що може бути критичним у випадках, коли об'єкти мають складну структуру або є частиною щільно розміщених кластерів, як, наприклад, клітини крові на мікроскопічних знімках.

З цієї причини в останні роки значну популярність здобули архітектури на основі трансформерів, які раніше зарекомендували себе у сфері обробки природної мови. Основною перевагою трансформерів є використання механізму самоуваги (self-attention), що дозволяє моделі звертати увагу на всі частини вхідного сигналу одночасно, формуючи глобальне уявлення про зображення. Даний підхід є корисним у медичних задачах, де локальні ознаки можуть бути неінформативними без контексту всього зображення.

Модель Detection Transformer (DETR), яка лежить в основі даної

дипломної роботи, поєднує у собі переваги CNN (для попереднього вилучення ознак) та трансформерів (для детекції об'єктів). На відміну від класичних підходів до детекції, що використовують жорстко закодовані якірні області (anchor boxes) і складні постобробки, DETR формулює задачу об'єктної детекції як задачу прямого зіставлення передбачень з реальними об'єктами за допомогою Hungarian algorithm. Це значно спрощує архітектуру та забезпечує високу гнучкість, особливо на невеликих медичних наборах даних.

Таким чином, глибоке навчання є як і потужним технічним інструментом, так і концептуальною базою, що дозволяє створювати адаптивні та точні системи аналізу зображень у медицині.

2.2 Архітектура Detection Transformer (DETR)

Зважаючи на особливості задачі виявлення клітин малярії на мікроскопічних зображеннях, а також сучасні тенденції розвитку комп'ютерного зору, у цій роботі було обрано архітектуру DETR як базову модель для побудови системи детекції. Як зазначено у підрозділі 2.1, цей підхід поєднує у собі здатність трансформерів до глобального моделювання просторових залежностей із ефективністю прямої регресії об'єктів без використання традиційних компонентів, таких як anchor boxes або non-maximum suppression. Нижче розглянуто детальні аспекти архітектури DETR, зокрема модулі підготовки ознак, механізм відповідності передбачень реальним об'єктам, а також функцію втрат, що забезпечує навчання моделі.

2.2.1 Загальна архітектура DETR

Архітектура Detection Transformer (DETR), як показано на рисунку 2.1, побудована як послідовність трьох основних компонентів: згорткової нейронної мережі (backbone) для вилучення ознак, трансформера, який обробляє ці ознаки, враховуючи контекст усіх об'єктів на зображенні, та блоку передбачень, що формує остаточні координати вікон детекції й класові мітки. Такий підхід дозволяє моделі здійснювати глобальне узгодження між усіма об'єктами на зображенні без використання традиційних евристичних механізмів.

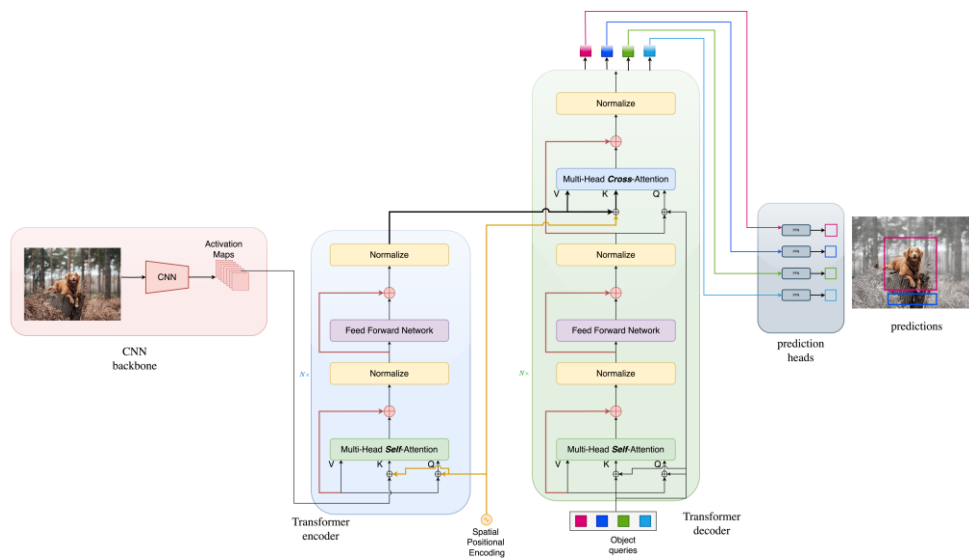


Рисунок 2.1 — Архітектура Detection Transformer(взято з [57])

У цьому підрозділі наведено загальну схему взаємодії основних частин DETR, а також коротко описано їх функціональні ролі. Детальний розгляд кожного компонента представлено у наступних підпунктах.

2.2.1.1 Архітектура CNN в DETR

Перший етап у процесі обробки зображення в моделі Detection Transformer (DETR) полягає у виділенні релевантних ознак за допомогою згорткової нейронної мережі (CNN) [58], як показано на рисунку 2.2. Ця складова архітектури відповідає за перетворення сирого вхідного зображення у більш компактне, але при цьому насичене та інформативне представлення, яке буде використано для подальшого аналізу та розпізнавання об'єктів.

Застосування CNN зумовлено її здатністю виявляти локальні просторові патерни завдяки процесу згортки, а також підтримувати структурні взаємозв'язки між пікселями, що є важливим для розпізнавання форм та контурів на зображенні. Ця властивість робить CNN оптимальним інструментом для початкової обробки візуальних даних перед передачею їх у більш складні модулі трансформера.

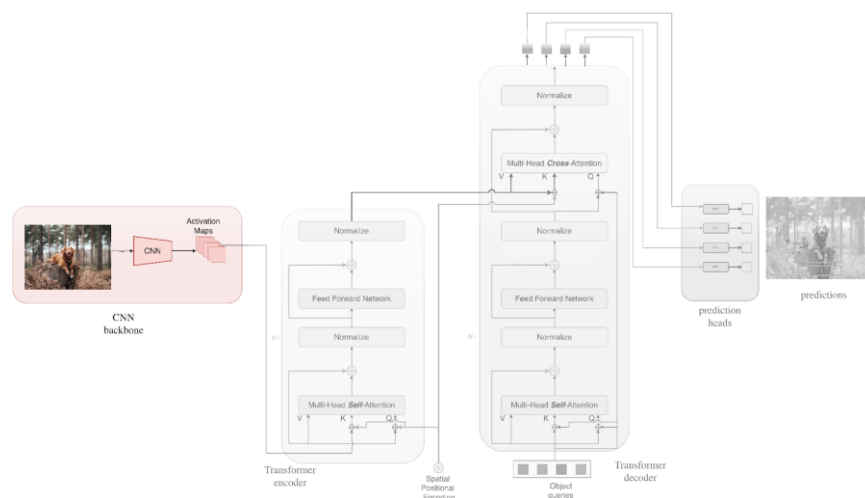


Рисунок 2.2 — CNN backbone виділений у всій архітектурі DETR (взято з [57])

Вхідне зображення, позначене має розмірність $H_0 \times W_0$ пікселів з трьома кольоровими каналами. Після проходження через шари CNN формується карта активацій f , що має розмірність $C \times H \times W$, де кількість каналів C дорівнює 2048, а просторові розміри H та W є зменшеними у 32 рази порівняно

з початковими розмірами зображення.

$$H = \frac{H_0}{32}, W = \frac{W_0}{32}.$$

Ця карта активацій виступає узагальненим описом ключових ознак вхідного зображення, водночас значно скорочуючи обсяг оброблюваних даних. Завдяки цьому, модель може зосередитися на виявленні суттєвих об'єктів та їх характеристик без втрати важливої інформації, що оптимізує роботу трансформера та підвищує точність детекції.

2.2.1.2 Архітектура трансформера в DETR

Після попереднього виділення ознак зображення за допомогою згорткової нейронної мережі, отримана карта ознак потребує додаткової обробки, аби бути сумісною з архітектурою трансформера. Оскільки трансформерна модель розроблена для роботи з послідовностями вхідних даних (токенів), передача 2D карти ознак безпосередньо у трансформерний енкодер є неможливою.

Для адаптації просторової карти ознак до формату послідовності виконується операція зменшення розмірності каналного виміру за допомогою згортки 1×1 . Цей крок зменшує кількість каналів з початкового великого числа до меншої розмірності d , що дозволяє знизити обчислювальні витрати та підвищити ефективність подальшої обробки. Результатом стає нова карта ознак $z_0 \in R^{d \times H \times W}$, яка має меншу глибину, зберігаючи при цьому найбільш релевантну інформацію. Однак навіть після зменшення розмірності каналів ця карта має двовимірну просторову структуру. Для подальшої обробки в трансформері потрібно «перетворити» цю двовимірну карту у послідовність векторів, які виступатимуть токенами. Для цього просторові

розміри карти $H \times W$ «згортаються» в один вимір, утворюючи послідовність довжиною N , де $N = H \times W$ кожен елемент послідовності є вектором розмірності d . Таким чином, трансформер отримує вхідні дані у вигляді послідовності токенів, придатних для послідовної обробки.

Далі, як показано на рисунку 2.3 послідовність передається до трансформерного енкодера [58]. Архітектура енкодера відповідає класичній побудові трансформера і складається з послідовності шарів, кожен з яких включає багатоголовий механізм самоуваги (multi-head self-attention), який детальніше буде розглянутий в розділі 2.4 та багат шаровий повнозв'язний нейронний блок (feed-forward network), який буде розглянутий в цьому ж підрозділі. Завдяки механізму самоуваги модель має здатність одночасно враховувати інформацію з усіх токенів послідовності, що дозволяє їй розуміти контекст та зв'язки між різними частинами зображення. Трансформер сам по собі не розпізнає порядок токенів і щоб компенсувати цей недолік, до вхідних ознак додаються спеціальні позиційні кодування [59], які передають інформацію про розташування кожного токена у просторі, що дозволяє моделі враховувати структуру зображення та більш точно ідентифікувати об'єкти.

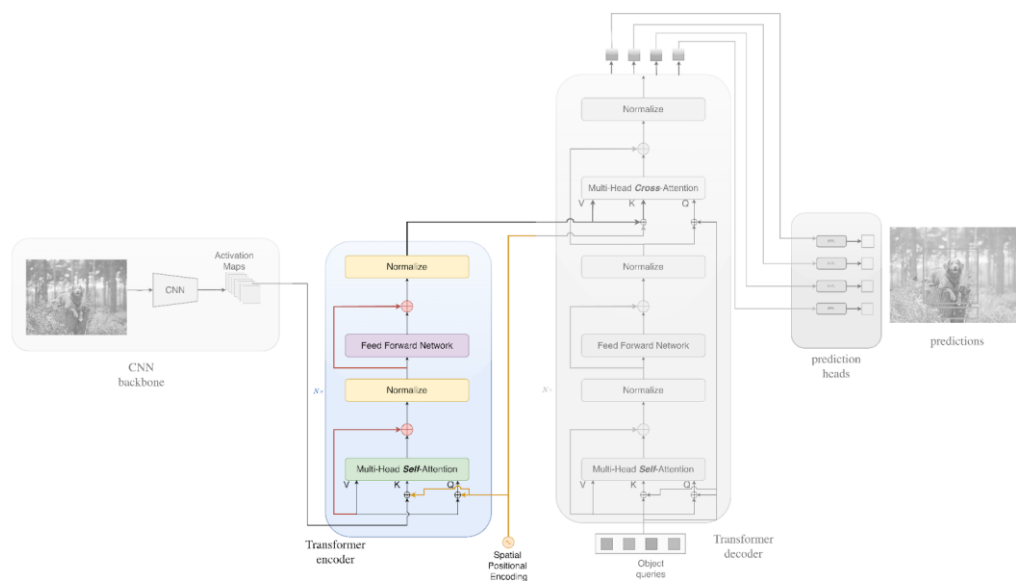


Рисунок 2.3 — Transformer Encoder виділений у всій архітектурі DETR

(взято з [57])

Декодер [58] у архітектурі DETR, як показано на рисунку 2.4, відповідає стандартній структурі трансформера, обробляючи N векторних представлень розмірності d за допомогою механізму самоуваги та механізму уваги між енкодером і декодером. Відмінність DETR від оригінальної трансформерної архітектури полягає в тому, що модель декодує всі N об'єктів паралельно на кожному шарі декодера, тоді як класичний трансформер за Васвані та ін. [60] формує вихідну послідовність послідовно, по одному елементу за раз (авторегресивно).

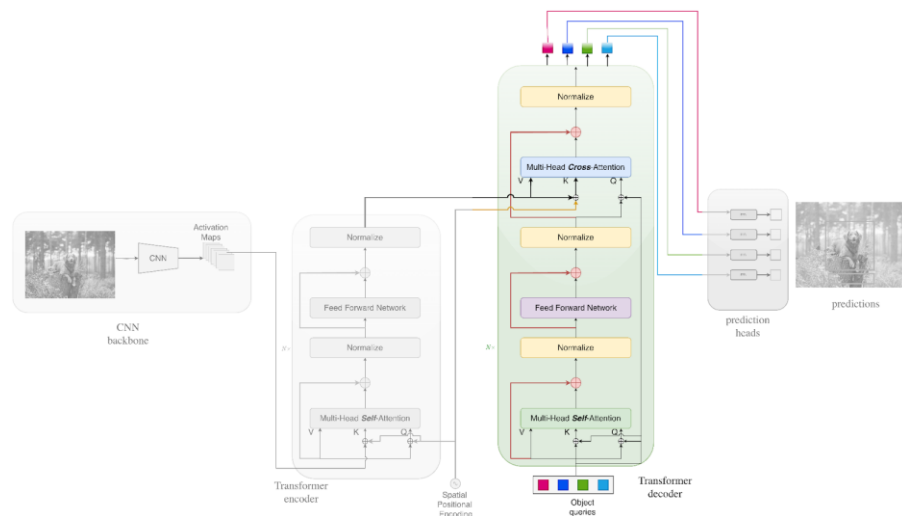


Рисунок 2.4 — Transformer Decoder виділений у всій архітектурі DETR
(взято з [57])

Оскільки декодер, так як і енкодер, не залежить від порядку вхідних векторів, для отримання різноманітних результатів кожен із N вхідних ембедингів повинен бути унікальним. Ці ембединг-вектори отримали назву «запити об'єктів» (object queries) [58]— це навчальні позиційні кодування, які додаються до кожного шару уваги декодера. Такий підхід дає змогу моделі ефективно ідентифікувати та визначати положення об'єктів на зображенні.

Пройшовши послідовно через усі шари декодера, N object queries перетворюються на вихідні вектори ознак, які потім окремо обробляються багатошаровим перцептроном (FFN) для передбачення координат рамок обмеження та відповідних класів. В результаті формується фіксований набір із N передбачень, які є кінцевими результатами моделі.

Застосування механізмів самоуваги та уваги між енкодером і декодером над вхідними вбудуваннями дозволяє моделі комплексно аналізувати взаємозв'язки між усіма об'єктами одночасно. Завдяки цьому модель враховує контекст усієї сцени, що значно підвищує якість і точність розпізнавання об'єктів.

2.2.1.3 Архітектура feed-forward networks в DETR

Для формування остаточних прогнозів модель, як показано на рисунку 2.5 використовує багатошарову нейронну мережу (FFN) [58], що складається з трьох рівнів із функцією активації ReLU та прихованим шаром розмірності d , а також завершується лінійним шаром. Ця мережа відповідає за визначення нормалізованих параметрів обмежувальних рамок — координат центру, висоти та ширини — у масштабі вхідного зображення.

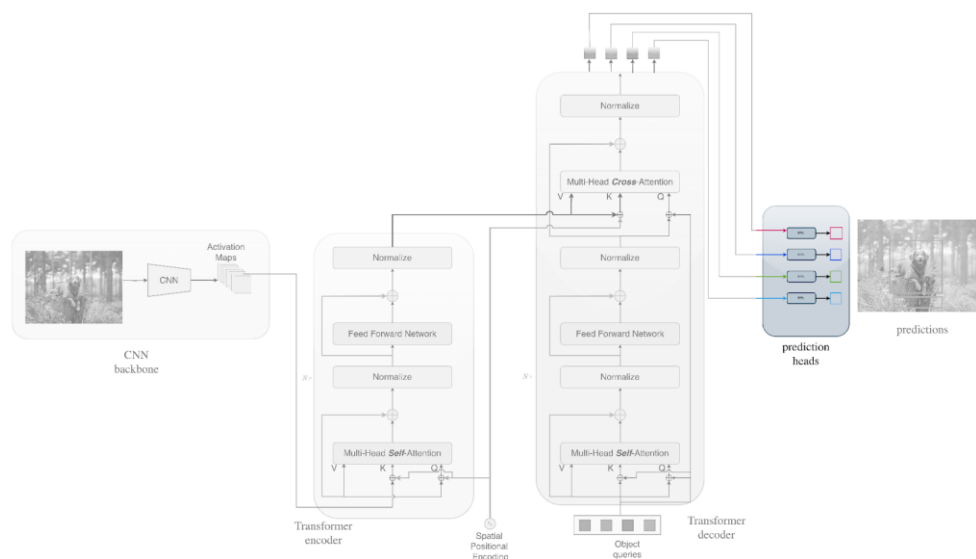


Рисунок 2.5 — FFN виділений у всій архітектурі DETR

(взято з [57])

Оскільки кількість прогнозованих рамок N заздалегідь фіксована та значно перевищує реальну кількість об'єктів на зображенні, вводиться

спеціальний клас-позначка « $\emptyset$ », який символізує відсутність об'єкта у відповідному слоті. Цей клас є фоновим сегментом, подібно до традиційних систем виявлення. Для підвищення ефективності навчання, а особливо для коректного визначення кількості об'єктів кожного класу, впроваджуються додаткові втрати, які застосовуються після кожного шару декодера. Важливо, що всі мережі прогнозування поділяють спільні параметри, а для стабілізації процесу навчання використовується спільний механізм нормалізації входів.

Зіставлення отриманих прогнозів із реальними об'єктами здійснюється за допомогою спеціального алгоритму, детальний опис якого наведено в розділі 2.5 цього дослідження.

2.2.2 Функція втрат у DETR

У процесі навчання модель DETR має зіставити фіксований набір з N передбачених об'єктів з реальною розміткою на зображенні. Зважаючи на те, що N зазвичай набагато більше за фактичну кількість об'єктів, необхідно впровадити ефективний механізм призначення, який би уникав дублювання та некоректних відповідностей. Для цього використовується точний метод оптимального зіставлення — Hungarian's алгоритм [58].

2.2.2.1 Оптимальне зіставлення прогнозів

Нехай реальні об'єкти описуються множиною $u = \{u_1, \dots, u_P\}$, а модель генерує набір з N передбачень $\hat{u} = \{\hat{u}_1, \dots, \hat{u}_P\}$, де $P \leq N$. Щоб здійснити зіставлення між ними, до ground truth додаються фіктивні об'єкти, що репрезентують «порожні» передбачення. Тоді задача зводиться до

знаходження перестановки $\sigma \in S_N$, яка мінімізує сумарну вартість зіставлення:

$$\hat{\sigma} = \underset{\sigma \in S_N}{\operatorname{argmin}} \sum_{i=1}^N \mathcal{L}_{match}(y_i, \widehat{y_{\sigma(i)}}),$$

де $\widehat{y_{\sigma(i)}}$ — відповідний прогноз для об'єкта y_i , а $\sigma(i)$ — індекс зіставленого передбачення. Функція $\mathcal{L}_{match}$ — це комбінована міра, що враховує невідповідність класу та об'єктом $y_i = (c_i, b_i)$, що включає клас c_i та координати рамки b_i , і відповідним передбаченням $\widehat{y_q}(i) = (\widehat{p_{\sigma(i)}}(c_i), \widehat{b_{\sigma(i)}})$.

Для розрахунку $\mathcal{L}_{match}$ застосовується наступна формула:

$$\mathcal{L}_{match}(y_i, \widehat{y_{\sigma(i)}}) = -1_{[c_i \neq \emptyset]} \cdot \widehat{p_{\sigma(i)}}(c_i) + 1_{[c_i \neq \emptyset]} \cdot \mathcal{L}_{box}(b_i, \widehat{b_{\sigma(i)}}),$$

де перший доданок оцінює ймовірність правильного класу, а другий — якість локалізації. Цей підхід дозволяє встановити однозначну відповідність між передбаченнями і реальними об'єктами без дублікатів.

2.2.2.2 Роль Hungarian's алгоритму

Для обчислення оптимальної відповідності використовується Hungarian's алгоритм. Він ефективно знаходить перестановку, яка мінімізує сумарні витрати відповідності. Це усуває потребу в таких евристичних процедурах, як визначення анкорів, ручна прив'язка або non-maximum suppression (NMS), які використовуються в традиційних об'єктних детекторах. Завдяки цьому, DETR забезпечує простішу архітектуру, що напряду передбачає цілісний набір об'єктів без складної постобробки. Тому після знаходження відповідностей між передбаченнями та ground truth, загальна функція втрат виглядає наступним чином:

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^N -\log \widehat{p_{\hat{\sigma}(i)}} + 1_{[c_i \neq \emptyset]} \cdot \mathcal{L}_{box}(b_i, \widehat{b_{\sigma(i)}}),$$

де $\hat{\sigma}$ — оптимальна перестановка зіставлення. У разі, якщо мітка $c_i = \emptyset$, логарифмічний доданок у втраті зменшується у 10 разів, щоб уникнути

дисбалансу між фоновими та об'єктними класами. Такий підхід подібний до субсемплінгу негативних прикладів у Faster R-CNN [61].

2.2.2.3 Втрати локалізації об'єктів

Функція локалізаційних втрат $\mathcal{L}_{box}$ [58] оцінює точність передбачення координат рамки. На відміну від підходів, що використовують дельти до анкорів, DETR безпосередньо прогнозує координати обмежувальних рамок. Щоб зменшити вплив масштабу, застосовується комбінація втрати і генералізованої IoU:

$$\mathcal{L}_{box}(b_i, \widehat{b}_{(i)}) = \lambda_{IoU} \cdot \mathcal{L}_{GIoU}(b_i, \widehat{b}_{(i)}) + \lambda_{L1} \cdot \|b_i - \widehat{b}_{\sigma(i)}\|_1,$$

де $\mathcal{L}_{GIoU}$ — генералізована метрика перекриття рамок, а λ_{IoU} та λ_{L1} — вагові коефіцієнти.

Обидві частини втрат нормалізуються на кількість реальних об'єктів у батчі. Це забезпечує стабільність навчання та дозволяє уникнути складних механізмів балансування позитивних/негативних прикладів або прив'язки передбачень до анкорів.

2.3 Переваги використання DETR для біомедичних задач

Архітектура DETR суттєво змінила підходи до обробки зображень у сфері комп'ютерного зору, зокрема завдяки поєднанню attention-механізмів і парадигми end-to-end навчання. Вона усуває потребу в ручному проектуванні анкорів та евристичних методах на кшталт non-maximum suppression, що є стандартом у традиційних детекторах, таких як Faster R-CNN чи YOLO. Замість цього DETR формулює задачу виявлення об'єктів як задачу прямого

парного зіставлення передбачень і ground truth за допомогою алгоритму Hungarian matching, що значно спрощує архітектуру та покращує її інтерпретованість.

Ці переваги є особливо цінними для біомедичних застосувань, де наявні специфічні виклики: висока роздільна здатність зображень, обмежена кількість анотованих прикладів, значний класовий дисбаланс та об'єкти, що вирізняються тонкими морфологічними ознаками. Особливості медичних зображень, що впливають на ефективність застосування DETR, можна окреслити такими аспектами. По-перше, клінічно значущі ознаки, такі як паразити, новоутворення чи мікрокальцифікати, часто займають незначну частину зображення. Це створює високі вимоги до точності локалізації та зумовлює необхідність здатності моделі виявляти дрібні об'єкти в масштабному контексті. По-друге, на відміну від природних сцен із великою варіативністю, медичні знімки створюються за строгими протоколами. Така одноманітність знижує шум і варіативність даних, проте водночас вимагає здатності моделі фокусуватися на вузькому спектрі релевантних ознак. По-третє, анотування медичних знімків є дорогим і трудомістким, тому обсяг доступних розмічених даних обмежений. Це також супроводжується суттєвим класовим дисбалансом, оскільки патологічні випадки трапляються значно рідше, ніж нормальні. Нарешті, у багатьох медичних задачах кожне зображення містить лише кілька об'єктів інтересу, часто однієї категорії. Це вимагає від моделі високої чутливості до незначних візуальних відмінностей між класами, що особливо актуально для задач виявлення конкретних патологій за обмеженою кількістю прикладів.

Попри вищезгадані виклики, останні роки позначені стрімким зростанням кількості адаптацій DETR до потреб медичної візуалізації. Зокрема, запропоновано низку модифікацій, спрямованих на поліпшення роботи з малими об'єктами, пришвидшення навчання та зменшення кількості хибнопозитивних виявлень:

- 1) Mathai et al. (2022) запропонували використання стратегії об'єднання

bounding box'ів у межах DETR для зменшення частоти хибнопозитивних результатів при виявленні лімфатичних вузлів;

- 2) Li et al. (2023) у моделі MyoriaDETR поєднали DETR з ієрархічною структурою Feature Pyramid Network, що дозволило покращити виявлення дрібних уражень при патологічній міопії;
- 3) Prangemeier et al. (2020) запропонували Cell-DETR — спрощену версію базової архітектури, яка зменшує кількість параметрів у десять разів без втрати точності в задачах виявлення клітин, забезпечуючи при цьому суттєве зростання швидкості інференсу;
- 4) Garrucho et al. (2023) застосували Deformable DETR до задачі виявлення мас у мамографії, зосередивши увагу на ефективності різних підходів до аугментації даних для підвищення точності.

Таким чином, навіть у межах складного та специфічного домену, такого як медична візуалізація, архітектура DETR демонструє гнучкість і потенціал до ефективної адаптації. Її здатність до енд-ту-енд навчання з мінімальною залежністю від інженерної тонкої настройки робить її перспективною платформою для розробки систем автоматичного виявлення патологій у клінічній практиці.

2.4 Інтерпретація уваги в трансформерах

Механізм уваги (self-attention) [58] є ключовим компонентом архітектур на основі трансформерів, зокрема в DETR. Його інтерпретація надає унікальну можливість зрозуміти, як модель приймає рішення — які частини зображення «важливіші» для прогнозу.

У трансформерах кожен піксель (або патч, у випадку візуальних моделей) представлений як вектор ознак — embedding [58]. Під час обчислення уваги кожен вектор запиту (Query) порівнюється з усіма

векторами ключів (Key) за допомогою скалярного добутку. Цей процес дозволяє визначити, наскільки «схожі» між собою різні частини зображення. На практиці це означає: якщо дві області мають схожі ознаки (наприклад, подібну текстуру, форму, колір), їх embeddings будуть близькими, і відповідна вага уваги буде високою.

У моделі DETR для кожного запиту — який відповідає об'єкту, що потенційно присутній у зображенні — обчислюється масив attention weights, що вказує, на які пікселі (patch'i) у фіч-мапі «звертає увагу» цей запит. Результатом є матриця розміром ($N_{\text{queries}} \times H \cdot W$), яка може бути візуалізована у вигляді теплової карти. Такі візуалізації допомагають зрозуміти, які частини зображення були найбільш інформативними для локалізації об'єкта.

Ще глибше розуміння можна отримати, аналізуючи власне матриці уваги. Вони формуються шляхом множення матриці запитів Q розміром $(H \cdot W, d)$ на транспоновану матрицю ключів K^T розміром $(d, H \cdot W)$ [58], що дає матрицю розміром $(H \cdot W \times H \cdot W)$. Кожен рядок у цій матриці — це розподіл уваги однієї позиції (пікселя) до всіх інших. Це дозволяє оцінити як локальні, так і глобальні залежності.

Наприклад, в офіційній статті від Facebook [58] з DETR розглядається перетворення цієї 2D матриці у 4D-масив розміром (H, W, H, W) , що полегшує візуалізацію — кожна позиція (X, Y) може бути зіставлена з картою уваги, яку вона генерує або яку до неї проявляють інші пікселі. Такий підхід дозволяє інтерпретувати увагу не лише як абстрактну величину, а як взаємозв'язок просторових координат. На рисунку зі статті [58] (Рис. 2.6), видно, що окремі об'єктні запити (object queries) фокусуються на різних об'єктах (наприклад, окремих коровах). Увага виражається у вигляді теплової карти, яка показує просторовий розподіл важливості. Це дозволяє інтерпретувати, яка частина зображення має значення для відповідного запиту трансформера.

Особливо це важливо у сфері медичної візуалізації, де прозорість моделей має критичне значення для довіри лікарів. Карти уваги можуть бути

використані як візуальні пояснення — наприклад, підтвердити, що модель справді «дивиться» на паразита або підозрілу ділянку тканини, а не на фон чи артефакти.

Таким чином, інтерпретація уваги в DETR відкриває шлях до пояснюваного ШІ в медицині, де не менш важлива не тільки точність, але й обґрунтованість рішення.

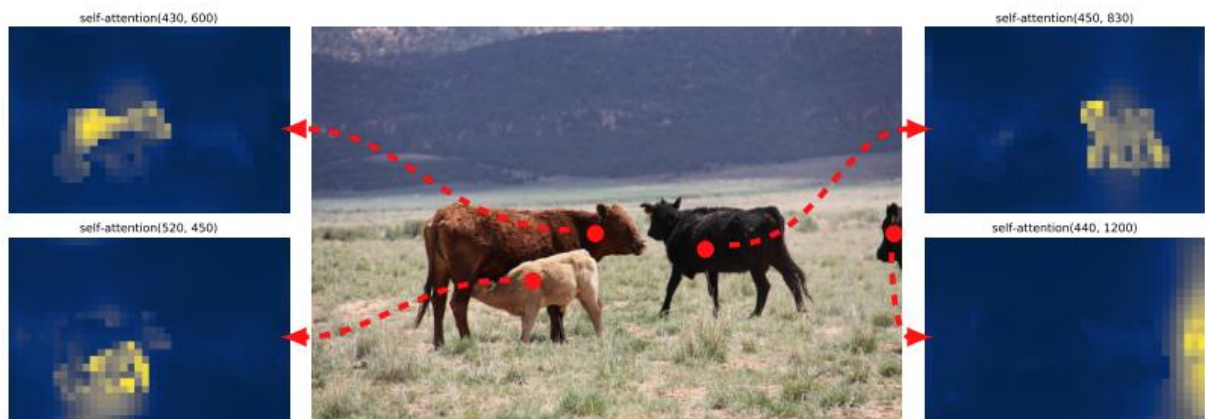


Рисунок 2.6 — Візуалізація self-attention в енкодері DETR: кожна карта показує, на які ділянки зображення спрямована увага конкретного запиту (Взято з [58])

2.5 Метрики оцінювання якості моделі детекції

Оцінка якості моделі детекції об'єктів також є важливою складовою процесу розробки та тестування алгоритмів комп'ютерного зору. На відміну від задач класифікації, де об'єкт уже визначено та лише потрібно обрати клас, у задачах детекції потрібно одночасно знайти положення об'єкта на зображенні (bounding box) та класифікувати його. Тому метрики для таких задач поєднують у собі як просторову точність (перетин областей), так і якість класифікації.

2.5.1 Метрики для задач об'єктної детекції

Однією з базових метрик для порівняння детектованого та реального (ground truth) об'єкта є Intersection over Union. Вона обчислюється як відношення площі перетину передбаченого прямокутника з істинним до площі їх об'єднання:

$$IoU = \frac{Area(Prediction_i \cap GroundTruth_i)}{Area(Prediction_i \cup GroundTruth_i)}.$$

Окрім порогових значень IoU, інформативним є також середнє IoU по всіх передбачених об'єктах. Цей показник відображає загальну якість локалізації незалежно від конкретного порогу. У задачах медичної детекції, зокрема при аналізі мазків крові, середнє IoU дає уявлення про точність позиціонування детектованих клітин на зображенні. Навіть якщо об'єкт виявлено вірно за класом, але межі бокса зміщені, це може негативно вплинути на подальший аналіз — наприклад, підрахунок паразитів або морфологічну оцінку клітин. Нижче наведемо формулу даної метрики:

$$AverageIoU = \frac{1}{TP} \sum_{i=1}^{TP} \frac{Area(Prediction_i \cap GroundTruth_i)}{Area(Prediction_i \cup GroundTruth_i)}.$$

Precision (точність) — відношення правильно класифікованих клітин до загальної кількості передбачених клітин даного класу. У задачі діагностики це показує, наскільки надійно модель відрізняє інфіковані клітини від здорових, не даючи зайвих хибних тривог.

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

де TP — правильні передбачення, FP — хибні спрацювання.

Recall (повнота) — відношення правильно виявлених клітин до загальної кількості таких клітин у наборі (GT). Високий Recall важливий для мінімізації пропусків інфікованих клітин, що критично для ранньої діагностики.

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

де FN — пропущені об'єкти.

Для повноцінного оцінювання ефективності моделі об'єктної детекції

недостатньо лише вказати точність або повноту при фіксованому порозі. У реальних сценаріях, особливо в медичних задачах, критично важливо враховувати поведінку моделі при зміні порогових значень — адже вибір того чи іншого порогу може значно змінити результати (наприклад, виявлення рідкісних клітин або паразитів при низькому recall).

Average Precision (AP) та Mean Average Precision (mAP) — це стандартизовані метрики, які дозволяють агрегувати якість детекції на різних рівнях чутливості та забезпечують узагальнену картину ефективності. Ці показники особливо важливі для порівняння моделей у змаганнях (наприклад, COCO), а також у медичних дослідженнях, де відсутність або надлишок передбачень має критичні наслідки.

Average Precision (AP) — це інтеграл (або площа під кривою) залежності Precision від Recall при зміні порогу впевненості:

$$AP = \int_0^1 p(r) dr.$$

Вона дозволяє оцінити якість моделі у широкому діапазоні recall, згладжуючи локальні коливання метрик.

Mean Average Precision (mAP) — середнє значення AP по всіх класах і/або по кількох порогах IoU (наприклад, від 0.5 до 0.95 з кроком 0.05, як у COCO challenge):

$$mAP@[0.5:0.95] = \frac{1}{10} \sum_0^1 AP@0.5 + 0.05 \cdot i.$$

Символ @ у метриках на кшталт $AP@0.5$ або $mAP@[0.5:0.95]$ означає «за умови» певного порогу IoU (Intersection over Union).

Ця метрика вважається золотим стандартом в об'єктивному порівнянні моделей, оскільки вона враховує як локалізацію (через IoU), так і точність/повноту передбачень.

2.5.2 Особливості оцінювання для медичних зображень

Оцінювання якості моделей детекції в задачах медичної візуалізації має низку специфічних особливостей, що відрізняють його від класичних завдань комп'ютерного зору на природних зображеннях. Це зумовлено специфікою медичних даних, а також високими вимогами до точності та надійності діагностики.

2.5.2.1 Вплив точності локалізації на якість клінічних діагностичних рішень

У медичних зображеннях об'єкти інтересу (наприклад, пухлини, мікроорганізми, патологічні утворення) часто мають дуже маленькі розміри та нечіткі межі. Точність локалізації цих об'єктів є критичною, адже від цього залежить правильність постановки діагнозу та подальше лікування пацієнта. Навіть незначна помилка у визначенні положення патології може призвести до пропуску важливої інформації.

Тому традиційні метрики, які вважають правильним детекцію при $\text{IoU} \geq 0.5$, можуть бути недостатньо чутливими у медичних задачах. Для таких випадків застосовують пороги IoU, що варіюються від 0.3 до 0.7 або навіть вище, а також аналізують якість локалізації за допомогою більш детальних метрик (наприклад, відстані між центрами передбаченого та істинного боксу).

2.5.2.2 Класовий дисбаланс і його вплив на оцінювання

В медичних задачах позитивні приклади (патології) зустрічаються значно рідше, ніж негативні (здорові ділянки). Це викликає сильний класовий дисбаланс у датасетах, що ускладнює тренування моделей і впливає на якість оцінювання.

Зокрема, метрики Precision і Recall мають різне значення у таких умовах:

- 1) Precision характеризує, яку частку виявлених патологій модель класифікувала правильно, і є важливою для уникнення хибнопозитивних результатів, що можуть призвести до зайвих додаткових досліджень і непотрібних занепокоєнь пацієнтів;
- 2) Recall демонструє, яку частку всіх наявних патологій модель виявила, що є критичним для забезпечення максимальної чутливості системи і мінімізації пропущених випадків.

У багатьох медичних застосуваннях перевага надається високому Recall, адже пропуск патології може мати серйозні наслідки для здоров'я пацієнта. Відповідно, при оцінці моделей часто розглядають Precision-Recall криві і метрики, що узагальнюють їх.

2.5.2.3 Обмеженість обсягів даних і вплив на метрики

Медичні дані часто мають обмежений обсяг через складність і вартість анотації знімків високої якості. Це призводить до високої варіативності оцінювань і збільшує ймовірність нестабільності метрик, особливо при поділі даних на тренувальні та тестові множини.

Для більш надійної оцінки якості моделей застосовують:

- 1) крос-валідацію;

- 2) детальний аналіз похибок — зокрема, розбір хибнопозитивних та хибнонегативних результатів, що допомагає зрозуміти, де саме модель працює найгірше;
- 3) використання метрик, які оцінюють як локалізацію, так і класифікацію, адже в медичних зображеннях часто важливо не лише знайти об'єкт, а й правильно його класифікувати.

2.5.2.4 Унікальні аспекти метрик у медичних задачах

У медичній діагностиці не всі помилки мають однакові клінічні наслідки. Помилки, пов'язані з виявленням певних патологій, можуть мати надзвичайно серйозний вплив на здоров'я пацієнта, а інші — бути менш критичними або мати тимчасовий характер.

Наприклад, помилка пропуску злоякісної пухлини (хибнонегативний результат) може призвести до відсутності своєчасного лікування, що загрожує життю пацієнта. Водночас хибнопозитивне виявлення доброякісного новоутворення може призвести до зайвих додаткових досліджень, підвищеної тривожності пацієнта і витрат на непотрібне лікування. Однак, у більшості випадків, пропуск важливих патологій вважається більш критичним, ніж зайві попередження.

З огляду на це, у медичних задачах часто застосовують метрики з врахуванням ваги помилок, які дозволяють зосередитись на найбільш важливих для клініциста випадках.

1. Деякі метрики адаптують шляхом введення ваг для різних класів патологій. Наприклад, клас злоякісних пухлин може мати більшу вагу в функції втрат, щоб модель більше «вважала» помилки у цьому класі.
2. Для навчання моделей використовують спеціальні функції втрат, які штрафують помилки залежно від їх клінічної важливості. Це дозволяє

підвищити чутливість до критичних класів та знизити ймовірність пропуску важливих патологій.

3. Зазвичай, у медичних системах прагнуть підвищити чутливість (Recall), навіть якщо це призводить до деякого зниження специфічності (Precision). Це пояснюється тим, що краще попередньо позначити потенційно небезпечну ділянку, ніж пропустити її.

Таким чином, вагове налаштування метрик і функцій втрат допомагає адаптувати алгоритми до специфіки медичної практики і мінімізувати ризики для пацієнтів.

Хоча автоматичні метрики дають об'єктивні чисельні показники якості моделей, у медичній галузі їх недостатньо для повного розуміння клінічної релевантності результатів. Тому важливим елементом оцінювання є залучення фахівців-медиків для експертної інтерпретації результатів.

Лікарі можуть врахувати клінічний контекст пацієнта, наявність супутніх захворювань, історію хвороби та інші фактори, що впливають на інтерпретацію зображення. Це особливо важливо, коли автоматична модель дає неоднозначні або суперечливі результати.

Медичні експерти аналізують хибнопозитивні та хибнонегативні випадки, щоб визначити, чи є ці помилки клінічно значущими. Наприклад, деякі помилки можуть бути прийнятними з медичної точки зору, якщо не впливають на лікування, тоді як інші — неприйнятними.

Відгуки експертів часто використовуються для валідації моделей перед їх застосуванням у клінічній практиці. Це дозволяє виявити слабкі місця алгоритму і доопрацювати його.

Спільна робота розробників і медиків дає змогу постійно покращувати моделі на основі реальних клінічних потреб та оцінок.

Таким чином, інтеграція експертної оцінки з автоматичними метриками є досить важливим фактором для створення ефективних і безпечних медичних систем.

2.6 Висновки до другого розділу

У другому розділі було детально розглянуто теоретичні основи та методичні підходи, які лежать в основі побудови сучасних моделей автоматизованої діагностики на основі глибокого навчання, зокрема в задачах детекції та сегментації об'єктів на медичних зображеннях.

Перш за все, було проаналізовано ключові принципи роботи трансформерів, зокрема механізму самоуваги, який забезпечує гнучку та ефективну обробку просторової інформації без жорсткої локальної індукції. Інтерпретація уваги в трансформерах показала, що цей механізм дозволяє моделі виявляти взаємозв'язки між різними частинами зображення, що є надзвичайно важливим для точного визначення меж та класифікації клітин або інших структур. Особливої уваги заслуговують підходи до візуалізації та аналізу карт уваги, які допомагають зрозуміти, як модель приймає рішення, що підвищує прозорість та інтерпретованість системи.

Окремий акцент у розділі зроблено на метриках оцінювання якості моделей об'єктної детекції, які є основою для кількісної оцінки результатів. Розглянуто такі метрики, як Intersection over Union (IoU), Average Precision (AP) та Mean Average Precision (mAP), їх математичні визначення та інтерпретацію у контексті задач детекції. Особливу увагу приділено тому, що в медичних задачах оцінювання моделей має свої унікальні аспекти, пов'язані із клінічною значущістю помилок. Зокрема, було підкреслено важливість врахування вагових коефіцієнтів для різних класів патологій, що відображає пріоритетність у виявленні небезпечних для життя пацієнта аномалій. Крім того, надано обґрунтування необхідності залучення експертної оцінки лікарів, які доповнюють автоматичні метрики, враховуючи клінічний контекст та складні ситуації, що не завжди корелюють із формальними чисельними показниками.

Таким чином, теоретична база, викладена у другому розділі, є

фундаментом для подальшої практичної роботи з розробки та впровадження моделей детекції на основі трансформерів. Застосування розглянутих методів дозволяє не лише підвищити точність і надійність автоматичної діагностики, але й зробити її більш прозорою, що є критично важливим у медичній сфері.

Підсумовуючи, другий розділ забезпечує комплексне розуміння сучасних підходів до обробки та аналізу медичних зображень із застосуванням глибоких нейронних мереж.

РОЗДІЛ 3. ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

3.1 Отримання та корекція набору даних

У цьому дослідженні було використано набір даних BBBC041v1 [1], який містить 1364 мікроскопічні зображення, зібрані трьома різними дослідницькими групами: у Бразилії (Stefanie Lopes), Південно-Східній Азії (Benoit Malleret) та в умовах лабораторного культивування в Оксфордському університеті (Gabriel Rangel). Всі зображення були вручну анотовані — для кожної клітини визначено обмежувальну рамку та клас.

Загалом у наборі присутні 86035 клітин, що належать до двох категорій неінфікованих клітин (еритроцити та лейкоцити) і чотирьох типів інфікованих клітин, які представляють різні стадії розвитку збудника малярії (кільця, трофозоїти, шизонти та гаметоцити). Також частину клітин було позначено як «важкі для класифікації» (difficult) — у випадках, коли не вдалося дати точну оцінку. У межах нашої роботи такі клітини вважаються інфікованими, що узгоджується з підходами, застосованими в низці попередніх досліджень [66].

Датасет був розділений на тренувальний та тестовий (табл. 3.1).

Таблиця 3.1 — Датасет

Набір	К-ть зображень	К-ть інфікованих	К-ть неінфікованих	Проблемність, %
Тренувальний	1208	2590	77523	3,2
Тестовий	120	308	5922	5,2
Загалом	1328	2898	83445	3,3

Через суттєву нерівномірність розподілу класів (понад 95% клітин — неінфіковані), було прийнято рішення зосередитися на задачі бінарної

класифікації: інфіковані проти неінфікованих клітин. Для цього відповідно змінено анотації — виконано попередню обробку, яка узгоджує їх із новою схемою класифікації (табл. 3.2).

Таблиця 3.2. — Відповідність між оригінальними класами клітин та об'єднаними категоріями для бінарної класифікації

Оригінальний клас	Клас після перетворення
Еритроцити	Неінфікованих
Лейкоцити	
Кільця (Ring)	Інфіковані
Трофозоїти (Trophozoite)	
Шизонти (Schizont)	
Гаметоцити (Gametocyte)	
Важкі для класифікації	

На рисунку 3.1 покажемо порівняння оригінальних зображень та зображень після зміни категорій класів. Перші три рисунка демонструють вихідні зображення з мікроскопічними мазками крові: перша — просто зображення, друга — з накладеними bounding box-ами, третя — з доданими підписами до кожного об'єкта (клітин). Інші три картинки демонструють зображення після перетворення категорій класів у формат «інфіковані» та «неінфіковані».

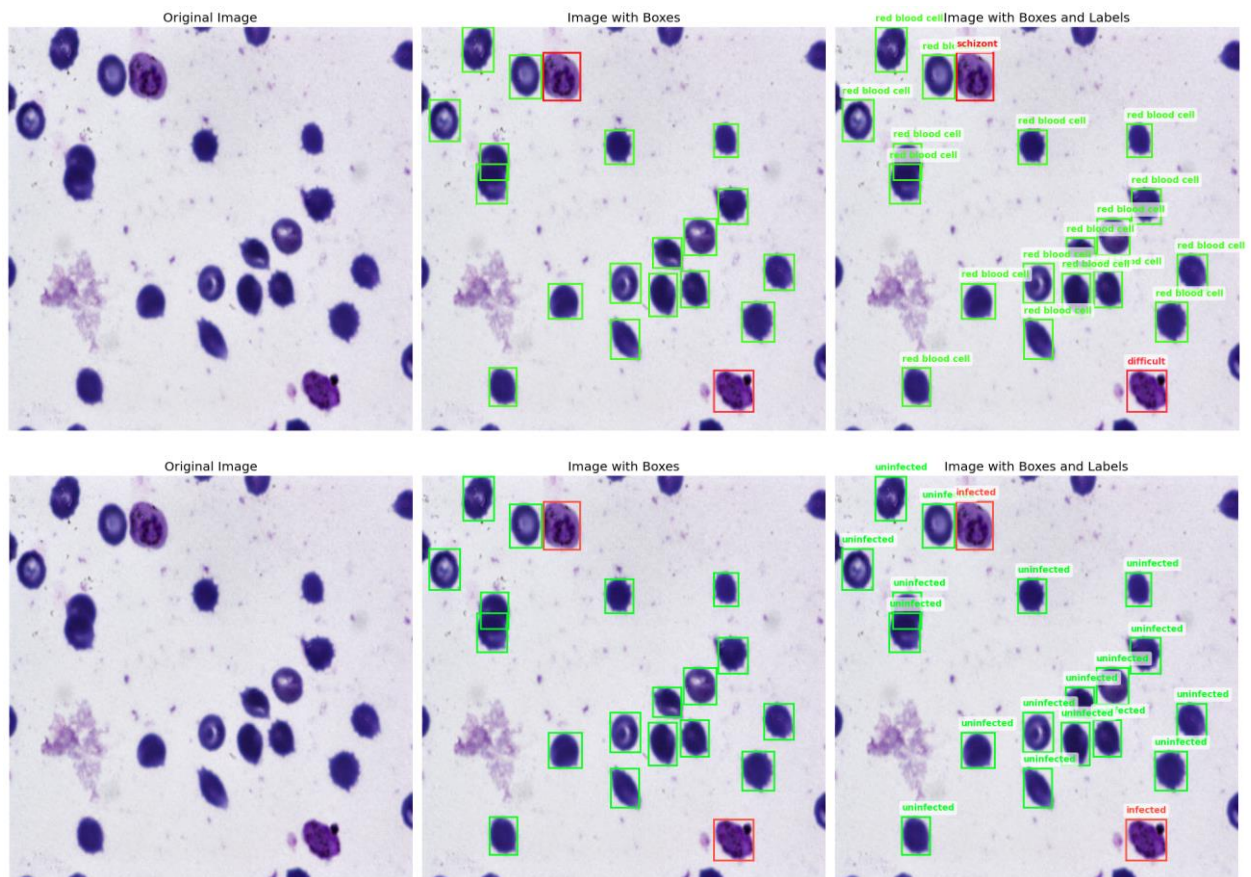


Рисунок 3.1 — Порівняння зображень до та після зміни категорій

3.2 Конвертація анотацій у формат COCO та підготовка вхідних зображень

Для забезпечення сумісності з обраною моделлю DETR, анотації датасету були перетворені у формат COCO (рисунок 3.2). Формат COCO є одним із найбільш поширених форматів представлення анотацій для задач детекції об'єктів, сегментації та кейпойнтів. Його структура передбачає наявність словників із описом зображень (images), категорій (categories) та об'єктів (annotations), де кожен об'єкт має координати обмежувального прямокутника (bounding box), клас, до якого він належить, та інші атрибути.

```

1  {
2    "images": [
3      {
4        "id": 1,
5        "file_name": "image_001.jpg",
6        "width": 640,
7        "height": 640
8      }
9    ],
10   "annotations": [
11     {
12       "id": 1,
13       "image_id": 1,
14       "category_id": 1,
15       "bbox": [x, y, width, height],
16       "area": width * height,
17       "iscrowd": 0
18     }
19   ],
20   "categories": [
21     {
22       "id": 1,
23       "name": "infected",
24       "supercategory": "cell"
25     },
26     {
27       "id": 2,
28       "name": "uninfected",
29       "supercategory": "cell"
30     }
31   ]
32 }

```

Рисунок 3.2 — Приклад структури COCO-формату

Перехід до COCO-формату є важливим етапом, оскільки він забезпечує повну сумісність із сучасними бібліотеками комп'ютерного зору, зокрема з PyTorch, Detectron2, MMDetection та багатьма моделями, зокрема DETR, яка була використана у цьому дослідженні. Вихідні анотації датасету BBBC041 (у форматі JSON) були автоматично перетворені за допомогою власного скрипта, що структурує дані відповідно до вимог COCO.

Окрім цього, всі вхідні зображення були попередньо масштабовані до розміру 640×640 пікселів, що відповідає налаштуванням попередньо-навченої моделі [63]. Ураховуючи, що зображення в оригінальному датасеті мали різну роздільну здатність (1600×1200, 1944×1383), було застосовано пропорційне масштабування з доповненням (padding), щоб уникнути спотворення об'єктів. Також були застосовані техніки збільшення даних (data augmentation) для зменшення дисбалансу класів і покращення здатності моделі до узагальнення. Зокрема:

Горизонтальне та вертикальне дзеркальне відображення з ймовірністю 0,5 дозволяють моделі навчатися на дзеркальних варіаціях клітин, тобто

$$I'(x, y) = I(W - 1 - x, y) \text{ або } I'(x, y) = I(x, H - 1 - y),$$

де W, H — ширина і висота зображення, а I та I' — оригінал та аугментований зразок.

Зміна кольору та корекція яскравості й контрасту здійснюються шляхом перетворення кожного пікселя за формулами

$$I' = \alpha I + \beta, I'' = \gamma(I' - \eta) + \eta,$$

де α і β — коефіцієнт контрасту та зміщення яскравості відповідно; γ — множник контрастності відносно середнього μ поточного каналу кольору. Це забезпечує стійкість до варіацій освітлення та забарвлення препаратів.

Для імітації нечіткості та варіації фокусу застосовується гаусове розмиття (RandomGaussianBlur):

$$I' = I * G_{\sigma}, G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-x^2+y^2}{2\sigma^2}\right) f n,$$

де $*$ — операція згортки, а σ генерується випадково в межах заданих значень.

Нарешті, для моделювання різних масштабів клітин поєднані кілька стратегій зміни розміру та обрізки. Спочатку зображення випадково масштабувалося до однієї зі сторін $\{400, 500, 600\}$ пікселів, потім із нього випадково вирізався прямокутник розміром від 384 до 600 пікселів, а далі знову виконувалося масштабування так, щоб коротка сторона становила одне зі значень $\{480, 512, 544\}$, але при цьому довга не перевищувала 1333 пікселя. Це можна подати як послідовність перетворень

$$I_1 = \text{resize}(I, s_1), I_2 = \text{crop}(I_1, c), I_3 = \text{resize}(I_2, s_2), \\ \text{des}_1 \in \{400, 500, 600\}, c \in [384, 600], \text{as}_2 \in \{480, 512, 544\}.$$

Загалом така комбінація аугментацій сприяє значному розширенню різноманітності тренувального набору, полегшує подолання дисбалансу між класами та допомагає моделі навчатися стійким до змін у положенні, розмірі й освітленості характеристикам паразитів.

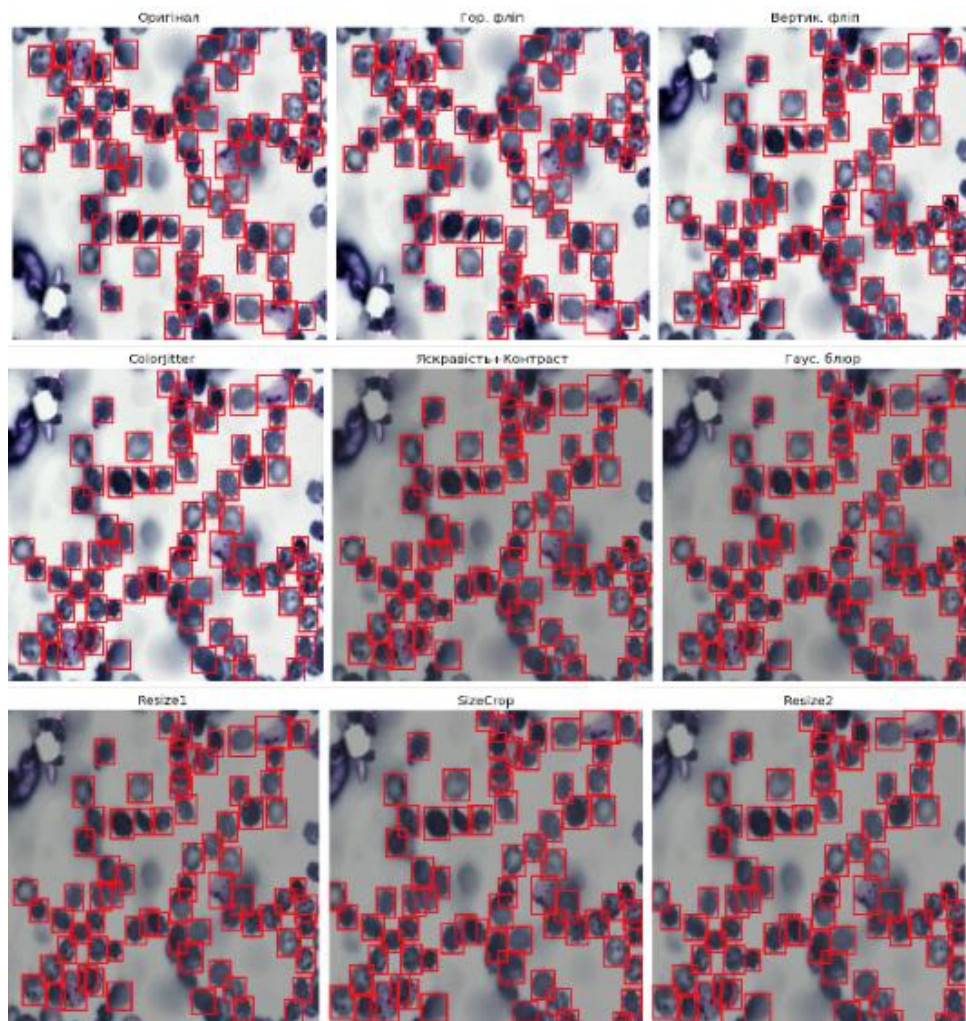


Рисунок 3.3 — Аугментація зображення

На рис.3.3 зліва зверху — оригінал, далі послідовно застосовані трансформації: горизонтальне фліп, вертикальне фліп, ColorJitter, зміна яскравості та контрасту, гаусове розмиття, ресайз, випадкове кадрування, фінальний ресайз.

3.3 Тренування моделі Detection Transformer (DETR)

Перед початком тренування моделі Detection Transformer (DETR) було визначено низку гіперпараметрів, які безпосередньо впливають на стабільність та ефективність навчання. Зокрема, швидкість навчання (learning

rate) була встановлена на рівні $1e-4$, а для параметрів бекбону (--lr_backbone) — $1e-5$. Таке зменшення швидкості навчання для попередньо натренованого бекбону (ResNet50) дозволяє зберегти вже набуті ознаки та запобігти їхньому «затиранню» в перші епохи тренування. Одночасно нові шари трансформера та класифікаційна голова навчаються активніше.

Для оптимізації використовувався алгоритм AdamW, який є модифікацією популярного адаптивного оптимізатора Adam із правильно реалізованою регуляризацією ваг (weight decay). У класичному варіанті регуляризація реалізується через додавання члена λ_{θ} до градієнта втрат, однак у випадку адаптивних методів, таких як Adam, це призводить до небажаної взаємодії з масштабуванням градієнтів. Щоб уникнути цього, Loshchilov і Hutter запропонували відокремити weight decay від обчислення градієнтів — підхід, що отримав назву decoupled weight decay regularization [64].

На відміну від класичного Adam, оновлення ваг у AdamW відбувається за формулою:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{m_t}{\sqrt{v_t + \epsilon}} - \alpha \cdot \lambda \cdot \theta_t,$$

де θ_t — ваги моделі на кроці t ;

α — швидкість навчання;

m_t і v_t — перший і другий моменти градієнтів відповідно;

ϵ — мале додатне число для числової стабільності;

λ — коефіцієнт регуляризації (weight decay).

Такий підхід дозволяє застосовувати регуляризацію незалежно від градієнтів функції втрат, що покращує контроль над узагальненням моделі. Оптимізатор AdamW знайшов широке застосування у сучасних трансформерних архітектурах. Зокрема, він використовувався під час навчання моделей BERT [65], DETR [66], Vision Transformer (ViT) [67], Swin Transformer [68] та багатьох інших. У цих роботах наголошується на тому, що застосування саме AdamW, а не класичного Adam, сприяє кращій стабільності навчання та загальній якості моделі.

Завдяки чіткому розділенню регуляризації ваг та оновлення параметрів, AdamW забезпечує більш стабільне та контрольоване навчання трансформерних моделей. У рамках даного експерименту значення коефіцієнта регуляризації (weight decay) було встановлено як $1e-4$ що є типовим значенням для задач детекції об'єктів, де необхідно запобігти перенавчанню моделі, зберігаючи при цьому гнучкість у налаштуванні параметрів.

Кількість епох тренування було встановлено рівною 20, що є компромісом між якістю та часом обчислень. Оскільки модель тренувалася в експериментальному режимі та мала обмеження по часу, кількість епох було суттєво зменшено порівняно з оригінальною реалізацією DETR (300 епох).

У структурі трансформера було визначено по 6 шарів для енкодера та декодера (`enc_layers = 6`, `dec_layers = 6`). Кількість голів самоуваги (`nheads`) дорівнює 8, що відповідає класичній конфігурації трансформерів із прихованим розміром `hidden_dim = 256`. Розмірність проміжного шару у feedforward-блоці (`dim_feedforward`) становить 2048, що забезпечує достатню виразну здатність моделі.

Як бекбон для вилучення ознак зображення використовується згортована нейронна мережа ResNet-50 [69], попередньо натренована на великому наборі зображень ImageNet [70]. Такий підхід дозволяє моделі використовувати вже наявні узагальнені ознаки для покращення якості виявлення об'єктів у новому завданні.

Для врахування просторового розташування ознак використано синусоїдальне позиційне кодування (sine positional encoding), що є класичним для архітектури DETR [71] і дозволяє моделі враховувати відносне положення об'єктів на зображенні без необхідності використання додаткових шарів або структур.

Для знаходження оптимальної відповідності між передбаченими об'єктами та істинною розміткою було використано Hungarian algorithm [60], який мінімізує загальну вартість призначення предиктів до міток. У функції

втрат застосовано ваги для різних компонентів: втрати класифікації (classification loss), втрати координат (L1 loss), та узагальненої інтерсекції на об'єднання (generalized IoU loss). Таке зважування дозволяє моделі досягати кращої точності локалізації та класифікації об'єктів.

У рамках тренування використовувалася модель DETR із бекбоном ResNet-50, попередньо навчена на датасеті COCO 2017. Під час навчання модель адаптується до специфіки медичних зображень, виконуючи пряме поширення, обчислення функції втрат, зворотне поширення градієнтів та оновлення вагів.

3.4 Аналіз результатів тренування

Після завершення процесу тренування моделі Detection Transformer (DETR) було проведено детальний аналіз її поведінки та якості прогнозів. Основну увагу зосереджено на оцінці динаміки функції втрат, продуктивності моделі за ключовими метриками точності, а також на візуальному аналізі передбачень на валідаційних прикладах. Такий підхід дозволяє комплексно оцінити здатність моделі до узагальнення та її ефективність у задачі виявлення клітин на медичних зображеннях.

3.4.1 Динаміка втрат

Одним із ключових елементів аналізу ефективності тренування нейронної мережі є відстеження зміни функції втрат (loss function) протягом епох. У випадку моделі Detection Transformer (DETR), яка використовується для виявлення та класифікації клітин у мікроскопічних зображеннях, функція

втрат є комплексною та включає кілька компонентів:

- 1) classification loss (Cross-Entropy Loss) — відповідає за правильну класифікацію об'єктів (інфікована / неінфікована клітина);
- 2) bounding box loss (L1 Loss) — оцінює точність передбачених координат прямокутників, що обмежують клітини;
- 3) generalized IoU loss (GIoU) — відповідає за просторову узгодженість передбачених об'єктів із реальними (GT) межами;
- 4) total loss — зважена сума всіх вище перелічених компонентів; саме ця метрика оптимізується під час навчання.

Для кращого розуміння динаміки навчання на рисунках 3.4-3.7 були побудовані графіки залежності значень функції втрат від кількості епох як для тренувальної, так і для валідаційної вибірок. На рисунку 3.4 бачимо фіксоване зменшення загальної функції втрат, що демонструє успішне тренування моделі. На тренувальній вибірці значення загальної функції втрат знизилось з 7.5 на початку до 5.32 після завершення навчання. На валідаційній вибірці функція втрат зменшилась з 5.7 до 4.61, що свідчить про покращення як класифікації, так і локалізації об'єктів.

Варто зазначити, що крива валідаційної втрати є досить нерівною, з вираженими коливаннями між епохами. Така «горбата» форма графіка може бути спричинена кількома факторами, які розписані нижче.

1. Малий обсяг валідаційної вибірки — у наборі даних використано лише близько 120 унікальних зображень, що обмежує варіативність прикладів, які потрапляють до кожного батчу. Це призводить до того, що навіть незначні зміни в параметрах моделі можуть суттєво впливати на валідаційні метрики, викликаючи коливання loss'у.
2. Нерівномірний розподіл класів — у даній вибірці спостерігатися дисбаланс між інфікованими та неінфікованими клітинами. Це ускладнює узагальнення моделі на всіх типах об'єктів і також може сприяти нестабільності валідаційного loss'у
3. Також, так як це медичні знімки, вони часто містять шум, перекриття

клітин, неоднорідність забарвлення, що робить задачу детекції складнішою

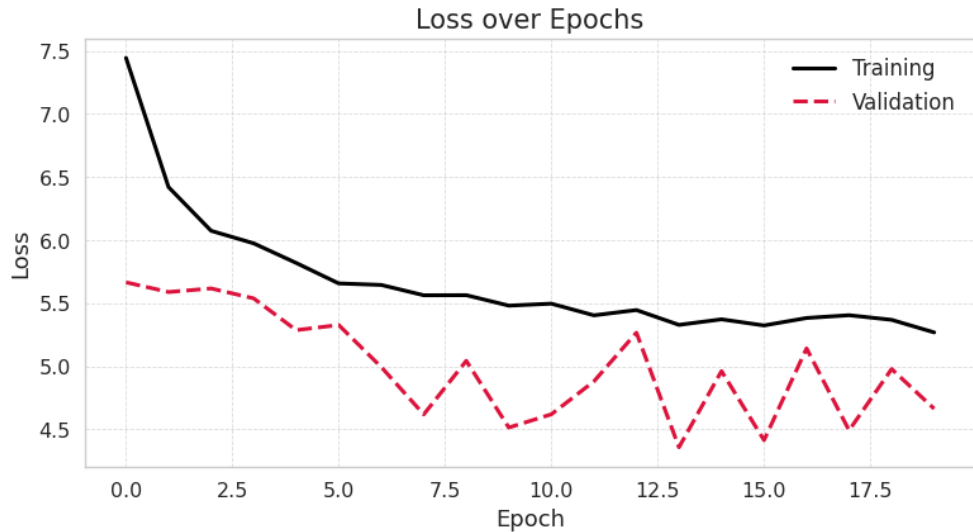


Рисунок 3.4 — Loss на тренувальній (чорним) та валідаційній (червоним) вибірках

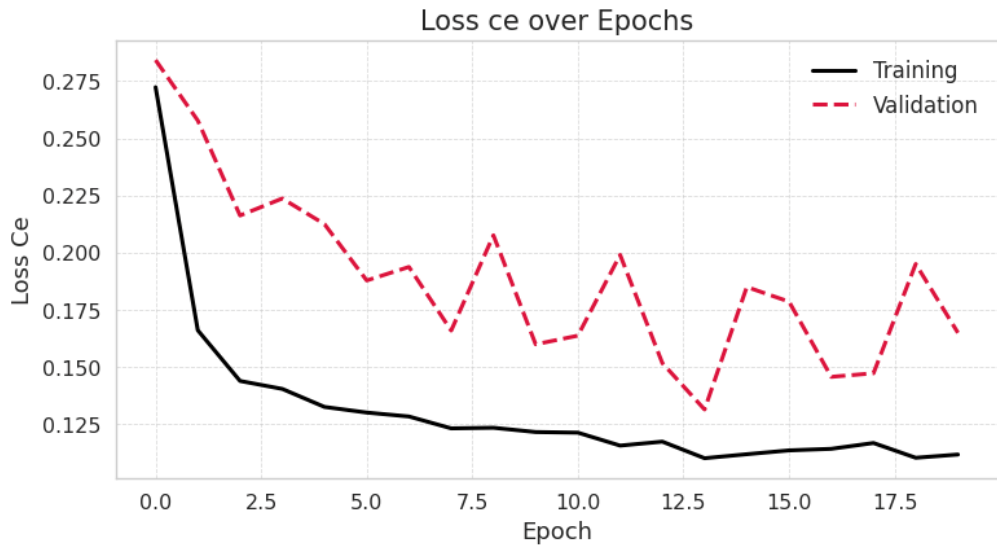


Рисунок 3.5 — Loss CE на тренувальній (чорним) та валідаційній (червоним) вибірках

На рисунку 3.5 чітко видно, що тренувальна крос-ентропійна втрата різко зменшується вже між 0 та 2 епохами, що свідчить про швидке первинне навчання моделі розпізнавати базові шаблони класів (інфіковані/неінфіковані клітини). Після цього темп зниження сповільнюється, і втрата поступово

вирівнюється, хоча загальна тенденція до зменшення зберігається до кінця навчання. Така динаміка є типовою для задач класифікації: спершу модель швидко опановує прості закономірності, а потім починає поступово уточнювати свої передбачення.

Валідаційна крива, як і у випадку з загальною функцією втрат, має нерівномірний «горбатий» характер. Вона демонструє значні коливання між епохами, що вже було детально пояснено в попередньому пункті — мала кількість валідаційних зображень, дисбаланс класів та складність візуального контенту можуть суттєво впливати на стабільність метрик на валідації.

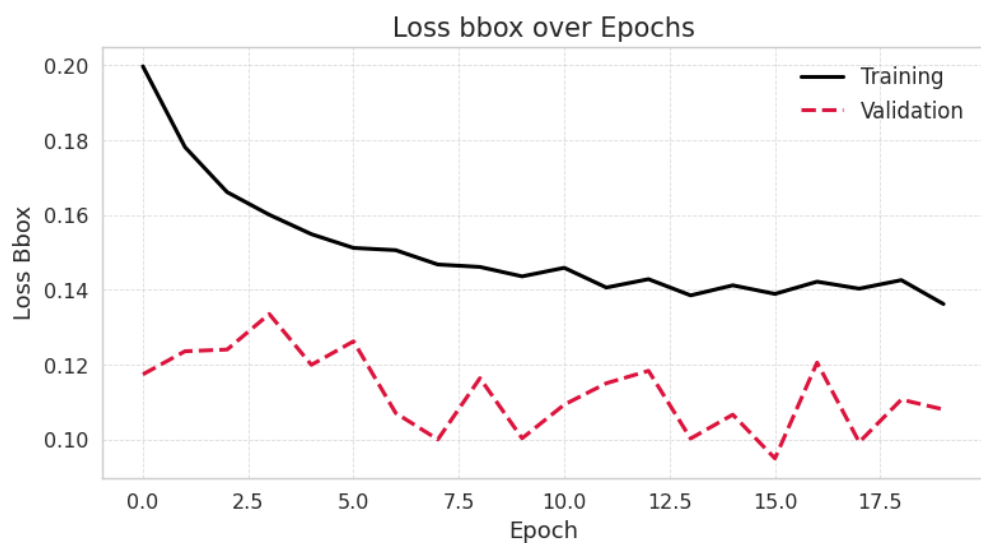


Рисунок 3.6 — Втрати для координат обмежувальних рамок (BBox Loss) протягом епох на тренувальній (чорним) та валідаційній (червоним) вибірках

На рисунку 3.6 відображено зміну втрат для передбачення координат обмежувальних рамок (bounding box regression loss) на тренувальній і валідаційній вибірках протягом усіх епох. Тренувальна крива демонструє поступове, стабільне зниження втрат, без різких спадів або стрибків, що свідчить про те, що модель послідовно вдосконалює точність локалізації об'єктів на зображеннях. Це може означати, що координати коробок уточнювались на основі вже досить стабільного уявлення про класи. Втрати на валідації, аналогічно до попередніх метрик, мають хвилеподібний характер, з нерівномірними коливаннями.

На рисунку 3.7 зображено зміну GIoU Loss — узагальненого показника перетину обмежувальних рамок — протягом усіх епох тренування. Тренувальна крива демонструє доволі чітке та стійке зменшення втрат GIoU, особливо протягом перших кількох епох, що свідчить про успішне навчання моделі формувати просторово точні рамки навколо об'єктів. Оскільки GIoU враховує як перетин, так і площу, яку охоплює спільна рамка, така стабільність означає, що модель навчилася уникати надмірних або зміщених передбачень. Крива втрат на валідаційній вибірці, як і у випадку з BBox Loss, має нерівномірний, коливальний характер.



Рисунок 3.7 — Втрати GIoU протягом епох на тренувальній (чорним) та валідаційній (червоним) вибірках

Загалом, аналіз динаміки функції втрат свідчить про ефективне навчання моделі DETR.

3.4.2 Точність детекції

Після детального аналізу окремих компонентів функції втрат — крос-ентропії, втрат за координатами та GIoU, — доцільно звернути увагу на інтегральну метрику, яка найбільш повно характеризує загальну якість роботи

моделі на завданні детекції. Йдеться про mean Average Precision (mAP) — ключовий показник точності детекції об'єктів, який враховує як правильність передбаченого класу, так і точність локалізації.

На графіку 3.8 зображено динаміку mAP на валідаційному наборі протягом епох. Показано як звичайне (сире) значення mAP, так і його згладжену версію, що усуває локальні коливання та дає чіткіше уявлення про загальну тенденцію. Незгладжена крива демонструє значні коливання, що вже спостерігалось у випадку з іншими метриками на валідації. Згладжена крива дозволяє краще простежити загальну динаміку. Вона чітко вказує на поступове покращення якості детекції: mAP зріс з приблизно 0.36 до 0.53, що є позитивним результатом і свідченням того, що модель вчиться точніше передбачати як класи, так і рамки об'єктів.

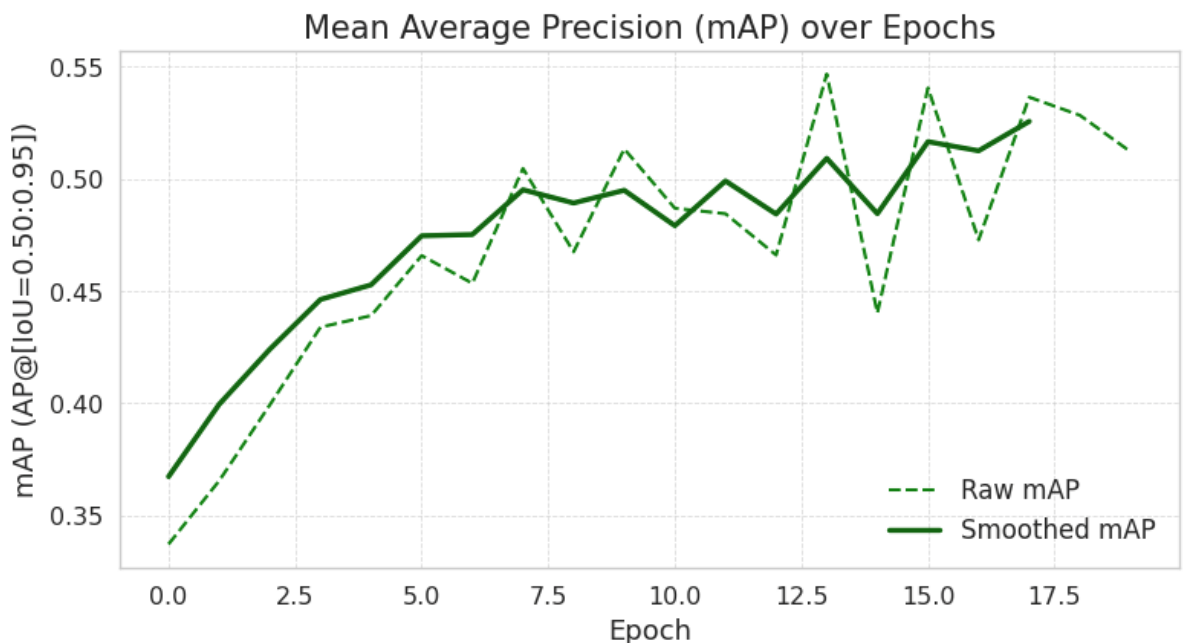


Рисунок 3.8. — Зміна mAP (mean Average Precision) на валідаційному наборі (суцільна лінія — згладжена крива mAP)

Після загальної оцінки метрик продуктивності моделі на валідаційному наборі (включаючи функції втрат та середнє значення mAP), доцільним було провести також локальну перевірку на окремих прикладах, щоб візуально та кількісно оцінити ефективність виявлення об'єктів у реальних умовах.

З цією метою було випадковим чином відібрано 5 зображень з

валідаційного набору. На кожному з них модель здійснила виявлення клітин, після чого результати були співставлені з реальними (GT) розмітками. Для кожної з категорій — infected (інфіковані клітини) та uninfected (неінфіковані клітини) — було обчислено основні метрики: precision, recall, accuracy, а також середнє значення IoU між передбаченими та реальними bounding box-ами (Таблиця 3.3).

Таблиця 3.3 — Результати локального тестування на 5 зображеннях

Метрика	Uninfected	Infected
Загальна кількість GT (ground truth)	237	9
Кількість передбачених об'єктів	283	12
True Positives (TP)	226	8
False Positives (FP)	26	4
False Negatives (FN)	11	1
Precision	0.8968	0.6667
Recall	0.9536	0.8889
Середнє IoU	0.8441	0.8667

Як видно з таблиці 3.3, модель демонструє високу точність і повноту при розпізнаванні неінфікованих клітин — понад 95% Recall і майже 90% Precision. Це підтверджує здатність моделі надійно ідентифікувати більшість таких клітин навіть на нових зображеннях. Водночас, результати для класу infected є менш стабільними: Precision становить лише ~66.7%, що свідчить про більшу кількість хибнопозитивних передбачень (FP). Проте Recall залишається високим (88.9%), тобто більшість дійсно інфікованих клітин були виявлені. Це важливо для клінічного застосування, оскільки в умовах діагностики критично важливо не пропустити справді інфіковані зразки. Ці

результати можна пояснити тим, що кількість інфікованих прикладів у вибірці значно менша, а отже модель має менше представлення цього класу. Крім того, висока візуальна схожість деяких здорових клітин з інфікованими могла призводити до плутанини в межах моделі.

3.4.3 Візуалізація передбачень

Після кількісного аналізу результатів важливо також звернути увагу на якісну сторону — як саме модель ідентифікує та локалізує клітини на реальних зображеннях. Для цього було виконано візуалізацію передбачень моделі на вибраних прикладах з валідаційного набору, що дозволяє наочно оцінити її роботу, виявити можливі помилки та особливості поведінки при класифікації інфікованих і неінфікованих клітин.

На кожному зображенні відображені як реальні bounding box'и (ground truth), так і передбачені моделю рамки з відповідними класами та ймовірностями. Це дозволяє наочно оцінити:

- 1) наскільки точно модель локалізує клітини;
- 2) чи правильно класифікуються інфіковані та неінфіковані клітини;
- 3) можливі випадки помилкових спрацьовувань (false positives) або пропущених клітин (false negatives).

На наведених ілюстраціях (рисунок 3.9) представлено порівняння оригінальної анотації зображень з результатами передбачень моделі. Зліва — вихідне зображення із нанесеними вручну боксами, які позначають клітини як інфіковані або неінфіковані. Справа — передбачення моделі DETR після тренування на нашому датасеті.

Перше зображення: Модель демонструє високу точність виявлення більшості клітин, правильно класифікуючи їх як інфіковані або неінфіковані. Проте спостерігається випадок, коли клітину, позначену в оригіналі як

неінфіковану, модель класифікувала як інфіковану. Така розбіжність може бути зумовлена наявністю візуальних артефактів або шумів, що ускладнюють точне визначення статусу клітини, або ж неточністю початкової анотації. Крім того, модель виявила кілька клітин на краях зображення, які не були позначені в оригінальній розмітці, але мають ознаки, характерні для клітин. Це свідчить про потенціал моделі ідентифікувати додаткові об'єкти, не враховані експертами, що можна розглядати як позитивний аспект.

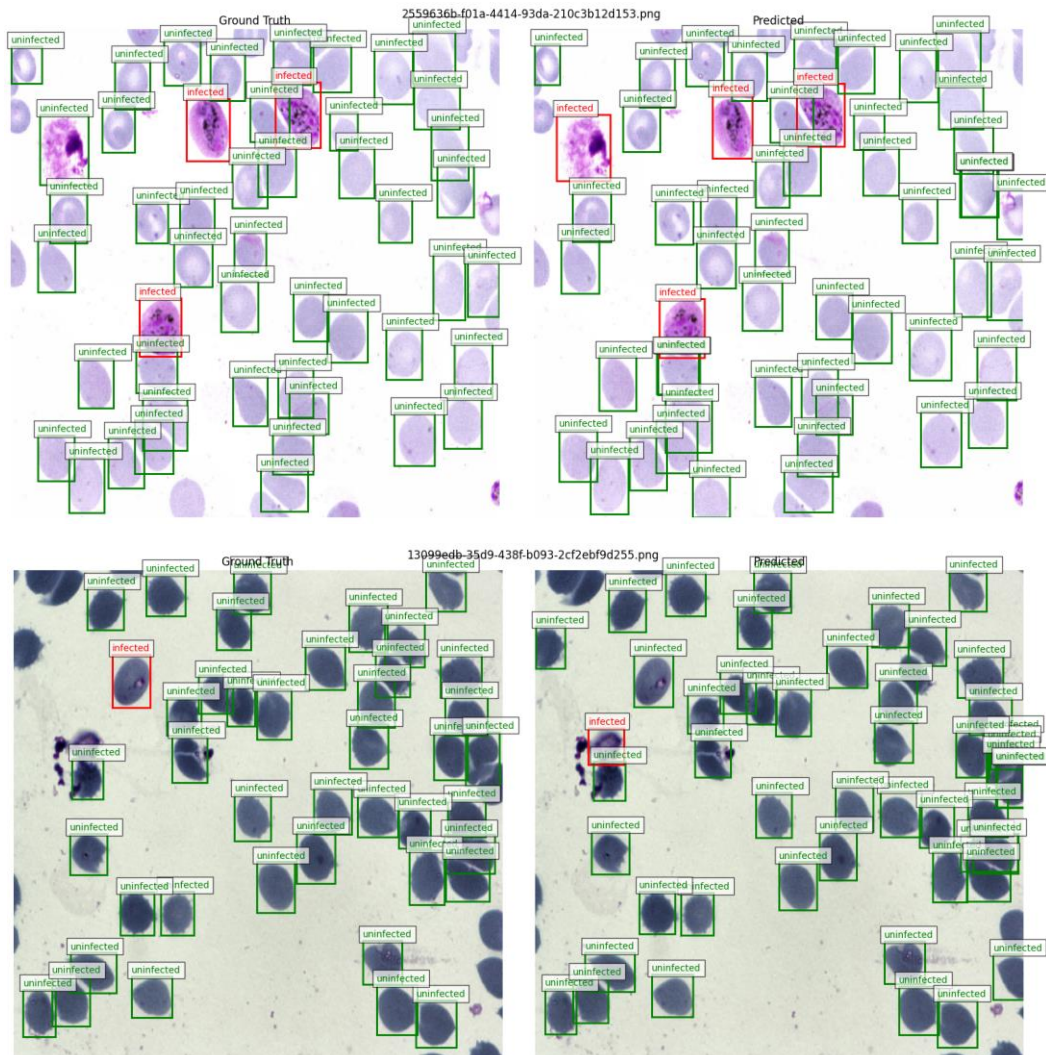


Рисунок 3.9 — Порівняння оригінальної анотації та передбачення моделі

Друге зображення: У цьому випадку модель пропустила одну клітину, яка була позначена як інфікована в оригіналі, проте натомість класифікувала як інфіковану інший об'єкт, який не був позначений вхідною розміткою. Візуально цей об'єкт також схожий на інфіковану клітину, що створює спірну

ситуацію щодо точності анотації. Аналогічно першому випадку, модель ідентифікувала додаткові клітини на периферії зображення, які спочатку не були розмічені, але, з огляду на їх характеристики, можуть представляти реальні клітини.

Загалом, наведені приклади демонструють узгодженість результатів моделі з експертною розміткою, а також потенціал моделі виявляти додаткові об'єкти. Деякі відмінності можуть бути пов'язані з неточностями початкової анотації або особливостями візуальних ознак, що ускладнює остаточне рішення.

3.5 Порівняння з існуючими роботами

У цьому підрозділі представлено порівняння результатів, отриманих у межах даної роботи з використанням моделі Detection Transformer (DETR), з результатами, наведеними у дослідженні FabioPC18 [72], де було проведено оцінку ефективності різних моделей глибокого навчання для задачі виявлення клітин, уражених малярією.

3.5.1 Методологія порівняння

У дослідженні FabioPC18 було використано кілька моделей глибокого навчання, зокрема Faster RCNN [73], YOLOv3 [74], U-net [75], та RT-DETR [76], для задачі виявлення клітин, уражених малярією. Для оцінки ефективності моделей використовувалися такі метрики: точність (Precision), повнота (Recall), F1, середній показник Intersection over Union (IoU) та середня точність (mAP).

У межах даної роботи було використано модель DETR, яка була донавчена на датасеті BBBC041, що містить зображення клітин крові з анотаціями у форматі COCO. Для оцінки ефективності моделі використовувалися ті ж метрики, що і в дослідженні FabioPC18, що забезпечує коректне порівняння результатів.

3.5.2 Результати

У таблиці 3.4 наведено порівняння результатів моделей, представлених у дослідженні FabioPC18, та моделі DETR, використаної у даній роботі.

Таблиця 3.4 — Порівняння результатів моделей детекції клітин малярії

Метрика	Клас	U-net	F-RCNN	RT-DETR	YOLOv8	DETR
Precision	Інфіковані	/	0.4887	0.7542	0.751	0.7399
	Неінфіковані	/	0.8015	0.7732	0.867	0.8752
Recall	Інфіковані	/	0.7045	0.5877	0.68	0.6506
	Неінфіковані	/	0.9672	0.7512	0.994	0.9431
F1/Dice	Інфіковані	0.5341	0.5770	0.6606	0.7137	0.66
	Неінфіковані	0.9097	0.8765	0.762	0.9261	0.9105
IoU	Інфіковані	0.3772	0.8553	0.6841	0.9143	0.7003
	Неінфіковані	0.8352	0.8534	0.7699	0.9141	0.8468
Accuracy	Інфіковані	0.4129	0.7045	0.5877	0.4059	0.6006
	Неінфіковані	0.9215	0.9672	0.7512	0.7257	0.7431

Модель DETR показала впевнені результати в задачі класифікації клітин крові. Зокрема, її точність склала 0.7399 для інфікованих клітин та 0.8752 для неінфікованих. Хоча за цим показником вона дещо поступається моделям YOLOv8 і RT-DETR, однак перевершує Faster R-CNN у виявленні саме інфікованих клітин, що має особливе значення для вчасної діагностики малярії.

Рівень повноти (recall) у DETR також залишається на досить високому

рівні — 0.6506 для інфікованих та 0.9431 для неінфікованих клітин. Незважаючи на те, що інші моделі, зокрема YOLOv8, демонструють вищі значення, DETR все ж забезпечує достатню здатність виявляти більшість об'єктів без критичних пропусків.

F1-метрика, яка враховує баланс між точністю та повнотою, становить 0.66 для інфікованих клітин і 0.9105 для неінфікованих. Ці результати вищі за показники U-Net та Faster R-CNN для інфікованих клітин, а також цілком співставні з RT-DETR, що підтверджує стабільну роботу моделі при класифікації обох класів.

За метрикою IoU, яка відображає якість просторового накладання передбачених меж на реальні об'єкти, модель DETR показала значення 0.7003 для інфікованих і 0.8468 для неінфікованих клітин. Хоча ці результати поступаються лише YOLOv8, вони свідчать про добру здатність моделі точно локалізувати об'єкти в зображеннях.

Загальна точність DETR склала 0.6006 для інфікованих клітин і 0.7431 для неінфікованих, що є середніми значеннями серед розглянутих моделей. Попри це, модель демонструє впевнене і стабільне функціонування без надмірної залежності від евристик чи специфічних налаштувань.

Таким чином, DETR може розглядатися як надійний інструмент для автоматизованої діагностики малярії. Вона показує збалансовану ефективність між точністю, повнотою та локалізацією, а її результати свідчать про потенціал застосування в клінічній практиці або системах підтримки прийняття рішень.

3.6 Висновки до третього розділу

У цьому розділі було детально описано процес підготовки та обробки даних для тренування моделі Detection Transformer (DETR) з метою автоматичного виявлення інфікованих клітин малярією на цифрових зображеннях мазків крові. Було здійснено конвертацію анотацій датасету BBBC041 у формат COCO для забезпечення сумісності з обраною моделлю, а також виконано стандартизацію розмірів вхідних зображень та застосовано методи попередньої обробки, зокрема нормалізацію інтенсивностей та покращення контрасту.

Тренування моделі відбувалося з контролем якості на валідаційному наборі даних та збереженням детальної статистики навчального процесу. Проведений аналіз результатів показав, що модель демонструє високу точність розпізнавання інфікованих клітин, а також здатність виявляти клітини, які могли бути пропущені в експертній розмітці.

Окрему увагу було приділено порівнянню результатів моделі DETR з іншими сучасними підходами, такими як YOLOv8, RT-DETR, Faster R-CNN та U-Net. Аналіз метрик (Precision, Recall, F1, IoU, Accuracy) показав, що модель DETR займає впевнену позицію серед конкурентів, демонструючи збалансовану якість розпізнавання як для інфікованих, так і неінфікованих клітин. Хоча за деякими показниками DETR поступається YOLOv8, вона перевершує інші моделі, зокрема у виявленні позитивного класу, що є ключовим для медичних застосувань.

Таким чином, поєднання детального підходу до обробки даних, використання сучасної архітектури DETR та проведення порівняльного аналізу з іншими методами дозволило обґрунтувати ефективність запропонованої моделі. Отримані результати підтверджують доцільність застосування DETR у задачах комп'ютерного зору в медицині, зокрема для автоматизованої діагностики малярії на основі мікроскопічних зображень.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі

У роботі застосовується метод функціонально-вартісного аналізу (ФВА) для проведення техніко-економічного обґрунтування розробки інтелектуальної системи для автоматизованої діагностики малярії на основі моделі Detection Transformer (DETR). Оскільки рішення щодо архітектури, алгоритмів та реалізації окремих модулів напряму впливають на ефективність функціонування всієї системи, кожна підсистема має відповідати загальній меті — забезпечити високу точність, надійність та швидкість автоматизованої обробки мікроскопічних зображень крові.

Проведений аналіз передбачає вивчення функцій програмного продукту, призначеного для завантаження, попередньої обробки, детекції паразитів *Plasmodium*, візуалізації результатів і підтримки прийняття рішень медичним персоналом. Система орієнтована на використання в умовах обмежених ресурсів, з можливістю розгортання в лабораторіях, лікарнях або польових умовах. Продукт повинен відповідати наступним технічним вимогам:

- 1) зручність та зрозумілість для користувача;
- 2) швидке завантаження та обробка зображень з наданням результатів аналізу в межах декількох секунд;
- 3) мінімальні витрати на розробку та впровадження програмного продукту;
- 4) функціонування на персональних комп'ютерах або ноутбуках із базовим апаратним забезпеченням без потреби у високопродуктивних GPU.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 — розробка інтелектуальної системи для автоматизованого виявлення паразитів малярії *Plasmodium* на мікроскопічних зображеннях, яка забезпечує ефективну діагностику за допомогою неймережевої моделі DETR. Виходячи з цієї мети, можна виділити такі ключові функції програмного продукту.

F_1 — вибір мови програмування.

F_2 — завантаження та попередня обробка зображень.

F_3 — детекція об'єктів за допомогою моделі DETR.

Кожна з цих функцій має декілька варіантів для реалізації. Для функції

F_1 :

- 1) а — Python;
- 2) б — R.

Для функції F_2 :

- 1) а — використання стандартних фреймворків;
- 2) б — власна реалізація препроцесингу.

Для функції F_3 :

- 1) а — тренування моделі з нуля;
- 2) б — пряме застосування попередньо натренованої моделі DETR.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

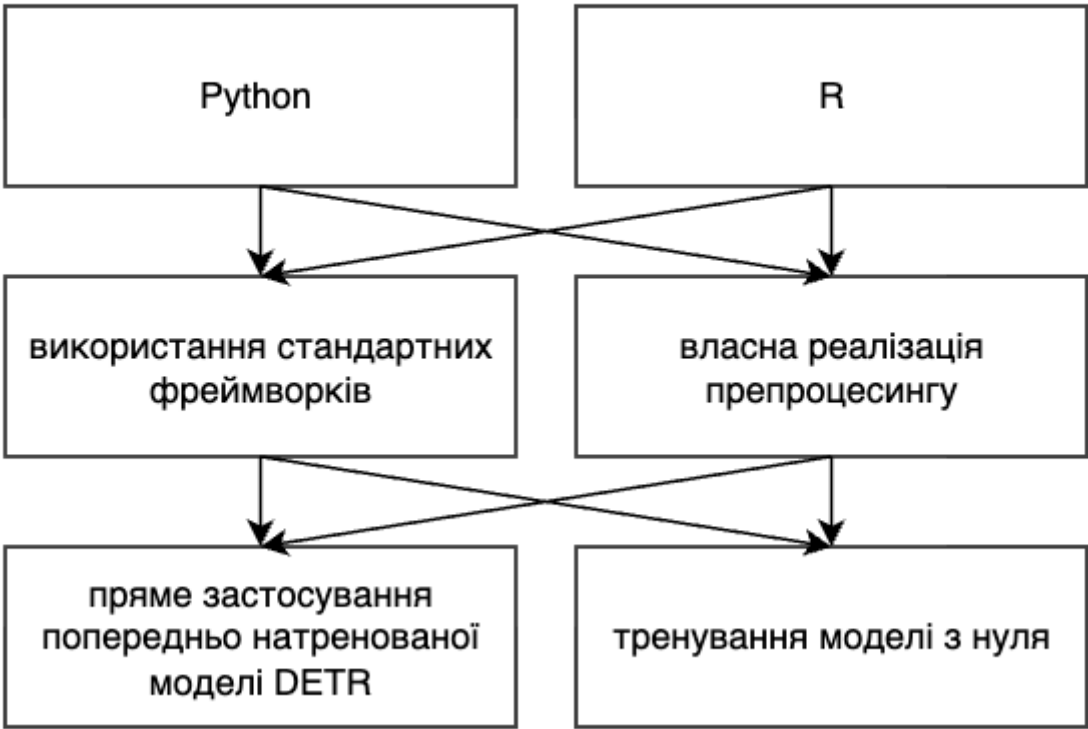


Рис. 4.1 — Морфологічна карта

Побудована морфологічна карта відображає всі можливі комбінації варіантів реалізації наведених вище функцій. На базі морфологічної карти побудуємо позитивно-негативну матрицю, наведену у таблиці 4.1.

Таблиця 4.1 — Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	а	Широкий спектр бібліотек; швидка розробка; активна спільнота розробників.	Відносно нижча продуктивність у порівнянні з компільованими мовами.
	б	Потужний інструментарій для статистичного аналізу та візуалізації даних.	Обмежена інтеграція з сучасними інструментами розгортання та машинного навчання.

Продовження таблиці 4.1

F_2	а	Прискорення розробки; багата функціональність; підтримка аугментацій.	Обмежена адаптація до нестандартних даних.
	б	Гнучкість; можливість врахування специфіки медичних зображень.	Збільшення часу розробки та потреба у валідації якості.
F_3	а	Повна адаптація до задачі; покращення точності.	Потреба у великому обсязі даних і часу на навчання.
	б	Швидке впровадження; економія обчислювальних ресурсів.	Модель може бути неадаптована до специфіки мікроскопічних зображень.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, оскільки їх використання не відповідають поставленим перед програмним продуктом задачам.

Для функції F_1 виключаємо варіант б, бо обмеженість інтеграції з сучасними інструментами розгортання є суттєвим недоліком. Для функції F_2 залишаємо варіант а — використання стандартних фреймворків, зокрема `torchvision.transforms`, які забезпечують базові трансформації, необхідні для коректної роботи моделі DETR. Цей підхід дозволяє дотримуватись офіційних практик реалізації, зменшує ризик помилок під час передачі даних у модель та скорочує час розробки. Крім того, використання перевірених трансформацій забезпечує узгодженість між етапами тренування і тестування моделі. Для функції F_3 залишаємо варіант б — донавчання попередньо натренованої моделі DETR. Це дозволяє адаптувати модель до специфіки медичного

датасету без необхідності навчання з нуля, що значно знижує обчислювальні витрати та скорочує час розробки. Таким чином, будемо розглядати такий варіант реалізації програмного продукту:

$$F_1 a - F_2 a - F_3 b.$$

4.3 Обґрунтування системи параметрів програмного продукту

На основі сформульованих технічних і функціональних вимог до системи автоматизованої діагностики малярії, визначено ключові параметри, що будуть використані для оцінки технічного рівня програмного продукту. Ці параметри дозволяють кількісно охарактеризувати ефективність, надійність та придатність системи до практичного використання у медичних установах. Для обґрунтування вибору програмного рішення враховуються наступні параметри:

- 1) x_1 — швидкодія детекції зображення;
- 2) x_2 — час виконання коду;
- 3) x_3 — точність детекції об'єктів;
- 4) x_4 — ресурсна складність моделі (кількість параметрів).

Гірші, середні і кращі значення параметрів визначаються на основі попереднього тестування системи, а також з урахуванням типових умов експлуатації у лабораторних або клінічних умовах (Табл. 4.2).

Таблиця 4.2 — Основні параметри програмного продукту

Назва	Позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Час розробки	x_1	годин	700	400	300
Час виконання коду	x_2	секунд	3	2	1
Точність детекції об'єктів	x_3	mAP	0.5	0.65	0.7

Продовження таблиці 4.2

Ресурсна складність моделі (кількість параметрів)	x_4	млн параметрів	60	50	40
---	-------	----------------	----	----	----

За наведеними значеннями в таблиці 4.2, побудуємо графічні характеристики параметрів (рис. 4.2 — 4.5):

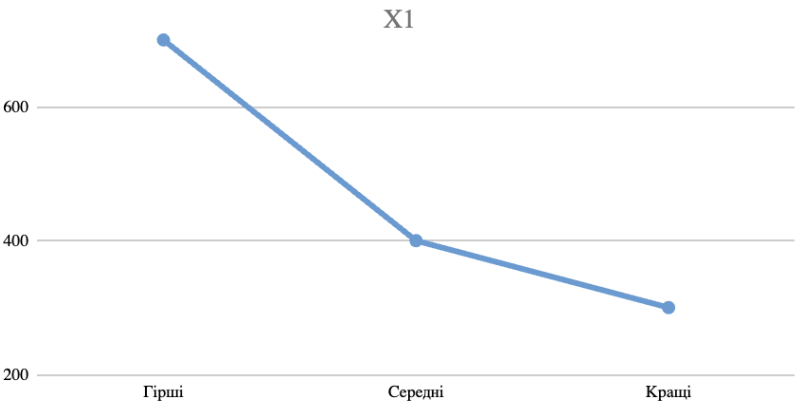


Рис. 4.2 — Швидкодія детекції зображення

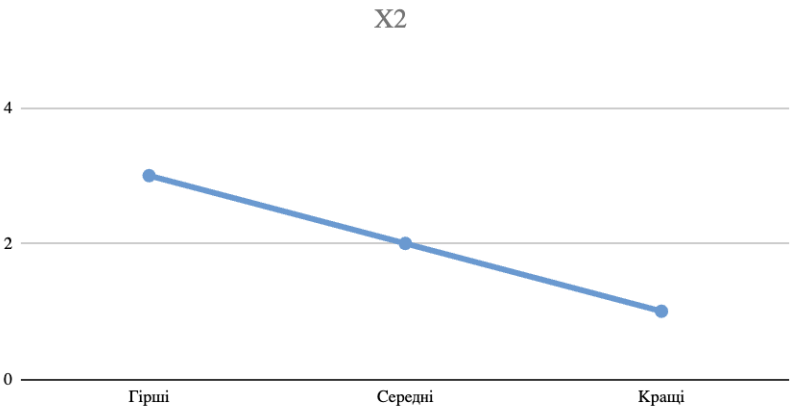


Рис. 4.3 — Точність детекції об'єктів

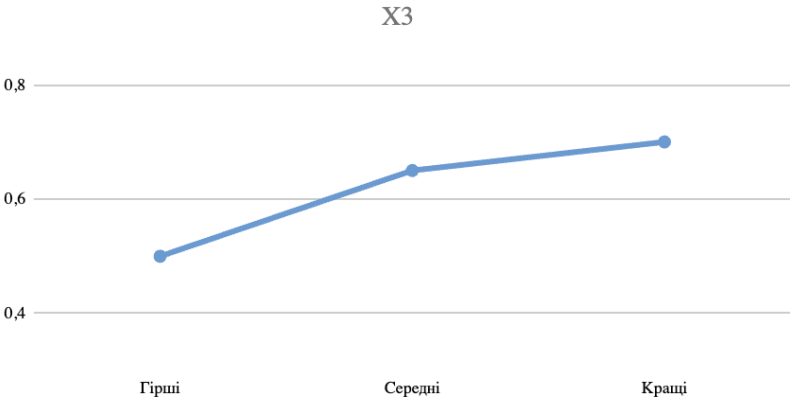


Рис. 4.4 — Ресурсна складність моделі (кількість параметрів)

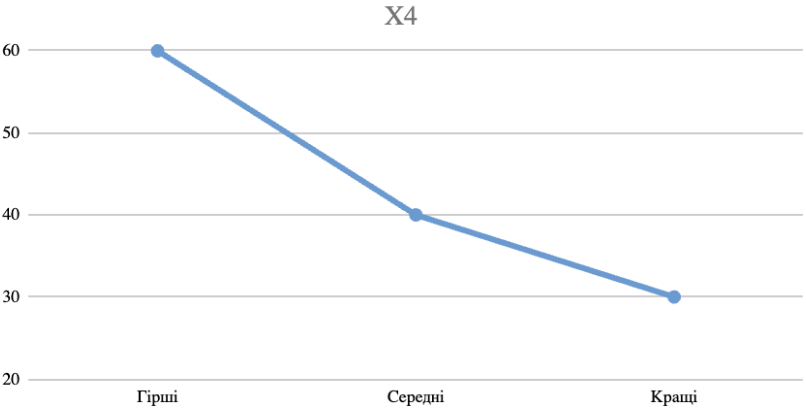


Рис. 4.5 — Ресурсна складність моделі (кількість параметрів)

4.4 Аналіз експертного оцінювання параметрів

Значимість кожного параметра визначається за допомогою методу попарного порівняння. Оцінку проводить експертна комісія, що складається з семи експертів. Кожний експерт оцінює ступінь важливості кожного параметру. Процедура визначення коефіцієнтів значимості передбачає виконання наступних кроків:

- 1) визначення рівня значимості кожного параметра;
- 2) перевірка придатності експертних оцінок;

- 3) визначення оцінки попарного пріоритету параметрів;
- 4) обробка результатів і визначення коефіцієнтів значимості.

Для перевірки придатності експертних оцінок визначимо наступні параметри. Сума рангів для кожного параметра і обчислюється за наступною формулою:

$$R_i = \sum_{j=1}^N r_{ij},$$

де r_{ij} — ранг i -го параметра експертом j ;

N — кількість експертів.

Загальна сума рангів обчислюється за наступною формулою:

$$R = \frac{Nn(n+1)}{2},$$

де N — кількість експертів;

бо n — кількість параметрів.

Позначимо середню суму рангів як T , вона обчислюється за наступною формулою:

$$T = \frac{R}{n}.$$

Відхилення суми рангів кожного параметра від середньої суми рангів обчислюється наступним чином:

$$\Delta_i = R_i - T.$$

Сума відхилень по всіх параметрах повинна дорівнювати 0.

Загальна сума квадратів відхилень:

$$S = \sum_{i=1}^N \Delta_i^2.$$

Наведемо результати експертного ранжування у таблиці 4.3.

Таблиця 4.3 — Результати експертного ранжування параметрів

Параметр	Ранг параметра за оцінкою експерта							R_i	Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
x_1	2	2	3	2	3	2	1	15	-2.5	6.25
x_2	2	1	2	2	1	1	1	10	-7.5	56.25
x_3	2	3	2	2	2	3	4	18	0.5	0.25

Продовження таблиці 4.3

x_4	4	4	3	4	4	4	4	27	9.5	90.25
Всього	10	10	10	10	10	10	10	70	0	153

Обчислимо коефіцієнт достовірності W :

$$W = \frac{12S}{N^2 n^2 (n-1)} = \frac{12 \cdot 153}{7^2 4^2 (4-1)} \approx 0,78.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 — Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
x_1 і x_2	=	>	>	=	>	>	=	>	1.5
x_1 і x_3	=	<	>	=	>	<	<	<	0.5
x_1 і x_4	<	<	=	<	<	<	<	<	0.5
x_2 і x_3	=	<	=	=	<	<	<	<	0.5
x_2 і x_4	<	<	<	<	<	<	<	<	0.5
x_3 і x_4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 & \text{при } X_i > X_j \\ 1 & \text{при } X_i = X_j \\ 0.5 & \text{при } X_i < X_j \end{cases}$$

З отриманих оцінок переваги складемо матрицю $A = \|a_{ij}\|$. Для кожного параметра зробимо розрахунок вагомості K_b за наступними формулами:

$$K_{b_i} = \frac{b_i}{\sum_{i=1}^n b_i},$$

$$b_i = \sum_{j=1}^N a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K'_{b_i} = \frac{b'_i}{\sum_{i=1}^n b'_i},$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j.$$

Як зазначено в таблиці 4.5, різниця значень коефіцієнтів вагомості на третій ітерації не перевищує 2%, тому більшої кількості ітерацій робити не потрібно.

Таблиця 4.5 — Обчислення вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	x_1	x_2	x_3	x_4	b_i	K_b	b_i^1	$K_{b_i}^1$	b_i^2	$K_{b_i}^2$
x_1	1	1.5	0.5	0.5	3.5	0.22	12.25	0.21	44.87	0.21
x_2	0.5	1	0.5	0.5	2.5	0.16	9.25	0.16	34.12	0.16
x_3	1.5	1.5	1	0.5	4.5	0.28	16.25	0.27	59.12	0.27
x_4	1.5	1.5	1.5	1	5.5	0.34	21.25	0.36	77.87	0.36

4.5 Аналіз рівня якості варіантів реалізації функції

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту розраховується за формулою:

$$K_K(j) = \sum_{i=1}^n K_{b_{i,j}} B_{i,j}.$$

Таблиця 4.6 — Розрахунок показників рівня якості варіантів реалізації основних функцій програмного продукту

Функція	Варіант реалізації	Параметр	Абсолютне значення	Бальна оцінка	Коефіцієнт вагомості	Коефіцієнт рівня якості
F_1	а	x_1	400	17	0.21	3.57
F_2	а	x_2	1.4	24	0.16	3.84
F_3	б	x_3	0.7	33	0.27	8.91
		x_4	42	30	0.36	10.8

Визначимо рівень якості кожного з варіантів за допомоги наступної формули:

$$K_K = K_{\text{ТУ}}[F_{1K}] + K_{\text{ТУ}}[F_{2K}] + \dots + K_{\text{ТУ}}[F_{zK}],$$

$$K_{K1} = 3.57 + 3.84 + 8.91 = 16.32,$$

$$K_{K2} = 3.57 + 3.84 + 10.8 = 18.21.$$

Спираючись на розрахунки, кращим є варіант 2, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

- 1) побудова конвеєру обробки та анотування медичних зображень охоплювала попередню обробку RGB-зображень високої роздільної здатності, побудову пайплайну для переведення анотацій у формат COCO та валідацію відповідності структури датасету вимогам моделі DETR;

- 2) навчання моделі DETR для задачі класифікації та сегментації клітин, включаючи налаштування гіперпараметрів, тренування, валідацію та збереження чекпойнтів;
- 3) розробка інструментарію аналізу якості моделі, що включає в себе побудову графіків, розрахунок основних метрик якості, генерація прикладів передбачень з візуалізацією об'єктів на зображеннях та інтерпретацію результатів.

Завдання 1 за ступенем новизни належить до групи Б, оскільки передбачає адаптацію існуючих рішень до специфіки конкретного медичного датасету. Завдання 2 та завдання 3 також належать до групи Б, оскільки використовуються відомі моделі та підходи, проте зі специфічним налаштуванням під поставлену задачу. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1, бо використовують типові інструменти для роботи з файлами, форматами та зображеннями; в завданні 2 — до групи 3, оскільки охоплює складні глибинні моделі з трансформерною архітектурою; а в завдання 3 — до групи 2, бо вимагає використання методів оцінки, обчислення метрик та побудови графіків, але не включає ускладнених алгоритмічних рішень. Для реалізації завдання 1 використовується переважно довідкова інформація (формати, бібліотеки, API), для завдання 2 — емпіричні дані (тренувальний датасет), а для завдання 3 — комбіновано: як довідкові матеріали, так і результати, отримані з моделі.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як:

$$T_k = T_p \cdot K_p \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_p — трудомісткість розробки програмного продукту;

K_p — поправочний коефіцієнт;

$K_{СК}$ — коефіцієнт на складність вхідної інформації;

K_M — коефіцієнт рівня мови програмування;

$K_{СТ}$ — коефіцієнт використання стандартних модулів і прикладних програм;

$K_{CT.M}$ — коефіцієнт стандартного математичного забезпечення.

Для першого завдання з застосуванням моделі ARX, виходячи із норм часу для завдань розрахункового характеру степеню новизни Б та групи складності алгоритму 1, трудомісткість дорівнює: $T_p = 35$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{II} = 1.6$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{CT} = 0.6$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 33.6.$$

Проведемо аналогічні розрахунки для другого завдання $T_p = 25$ людино-днів, $K_{II} = 0.7$, $K_{CK} = 1$, $K_{CT} = 0.7$:

$$T_2 = 12.25.$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість в годинах:

$$T_I = (33.6 + 12.25 + 16.32) \cdot 8 = 497.36,$$

$$T_{II} = (33.6 + 12.25 + 18.21) \cdot 8 = 512.48.$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь один програміст з окладом 25000 грн., один аналітик в області даних з окладом 30000. Визначимо середню зарплату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t}.$$

де M — місячний оклад працівників;

T_m — кількість робочих днів на місяць;

t — кількість робочих годин в день.

Обчислена середня зарплата дорівнює:

$$C_q = \frac{30000 + 25000}{2 \cdot 21 \cdot 8} \approx 163.7,$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{3II} = C_q \cdot T_i \cdot K_d.$$

де C_q — величина погодинної оплати праці програміста;

T — трудомісткість відповідного завдання;

K_D — норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$C_{ЗП,I} = 163.7 \cdot 497.36 \cdot 1.2 = 97701.39,$$

$$C_{ЗП,II} = 163.7 \cdot 512.48 \cdot 1.2 = 100671.57.$$

Відрахування на єдиний соціальний внесок становить 22%:

$$C_{ВІД,I} = C_{ЗП,I} \cdot 0.22 = 97701.39 \cdot 0.22 = 21494.3,$$

$$C_{ВІД,II} = C_{ЗП,II} \cdot 0.22 = 100671.57 \cdot 0.22 = 22147.74.$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M) Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0,22 то для однієї машини отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 25000 \cdot 0.22 = 66000.$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_{\Gamma}(1 + K_3) = 66000 \cdot (1 + 0.22) = 80520.$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0.22 = 80520 \cdot 0.22 = 17714.4.$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ — 25000 грн.

$$C_A = C_{ПР} \cdot K_A \cdot K_{ТМ} = 25000 \cdot 0.25 \cdot 1.4 = 8750,$$

де $K_{ТМ}$ — коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A — річна норма амортизації;

$C_{ПР}$ — договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = C_{ПР} \cdot K_P \cdot K_{ТМ} = 25000 \cdot 1.4 \cdot 0.06 = 2100,$$

де K_P — відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 102 - 12 - 15) \cdot 8 \cdot 0.9 = 1699.2,$$

де D_K — календарна кількість днів у році;

D_B, D_C — відповідно кількість вихідних та святкових днів;

D_P — кількість днів планових ремонтів устаткування;

t_3 — кількість робочих годин в день;

K_B — коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot C_{\text{ЕН}} = 1699.2 \cdot 0.3 \cdot 0.6 \cdot 9.43 = 2884.22,$$

де N_C — середньо-споживча потужність приладу;

K_3 — коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ — тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{\text{ПР}} \cdot 0.67 = 25000 \cdot 0.67 = 16750.$$

Тоді, річні експлуатаційні витрати будуть дорівнювати:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_A + C_P + C_{\text{ЕЛ}} + C_H = 80520 + 17714.4 + 8750 + 2100 + 2884.22 + 16750 = 128718.62.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = \frac{C_{\text{ЕКС}}}{T_{\text{ЕФ}}} = \frac{128718.62}{1699.2} \approx 75.75.$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М, I}} = C_{\text{М-Г}} \cdot T_I = 75.75 \cdot 497.36 = 36675.02,$$

$$C_{\text{М, II}} = C_{\text{М-Г}} \cdot T_{II} = 75.75 \cdot 512.48 = 38820.36.$$

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н, I}} = C_{\text{ЗП, I}} \cdot 0.67 = 97701.39 \cdot 0.67 = 65459.93,$$

$$C_{\text{Н, II}} = C_{\text{ЗП, II}} \cdot 0.67 = 100671.57 \cdot 0.67 = 67449.95.$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_M + C_H.$$

$$C_{\text{ПП, I}} = C_{\text{ЗП, I}} + C_{\text{ВІД, I}} + C_{\text{М, I}} + C_{\text{Н, I}} = 97701.39 + 21494.3 + 36675.02$$

$$+ 65459.93 = 221336.64,$$

$$C_{\text{ПП, II}} = C_{\text{ЗП, II}} + C_{\text{ВІД, II}} + C_{\text{М, II}} + C_{\text{Н, II}} = 100671.57 + 22147.74 + 38820.36$$

$$+ 67449.95 = 229089.62.$$

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = \frac{K_{kj}}{C_{\Phi j}},$$

$$K_{\text{ТЕР}1} = \frac{16.32}{221336.64} = 7.37 \cdot 10^{-5},$$

$$K_{\text{ТЕР}2} = \frac{18.21}{229089.62} = 7.95 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $7.37 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишилися після першого відбору варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості

Цей варіант реалізації програмного продукту має такі параметри:

- 1) Python в якості мови програмування;
- 2) використання стандартних фреймворків;
- 3) пряме застосування попередньо натренованої моделі DETR.

Цей варіант виконання програмного комплексу гарантує швидку роботу програмного продукту та реалізацію достовірної детекції об'єктів на зображенні.

4.7 Висновки до четвертого розділу

Проведено функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій. Проведено економічний аналіз варіантів розробки — трудомісткість, витрати на заробітну плату та інші витрати. На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У дипломній роботі проведено комплексне дослідження, присвячене задачі автоматизованої класифікації та сегментації клітин малярії на основі зображень мазків крові. Основна мета полягала у створенні ефективного підходу до діагностики малярії з використанням сучасних методів глибинного навчання, зокрема моделі Detection Transformer (DETR), адаптованої до медичної доменної області. На початковому етапі було здійснено детальний огляд традиційних та сучасних методів діагностики малярії, зокрема мікроскопічних, молекулярних, експрес-методів, а також підходів на основі штучного інтелекту, що дозволило обґрунтувати актуальність теми та вибір саме трансформерної архітектури.

У процесі дослідження було підготовлено та попередньо оброблено датасет BBBC041, що містить зображення клітин крові, інфікованих паразитом *Plasmodium vivax*. Було виконано анотування об'єктів, конвертацію даних у формат COCO, а також сформовано навчальну, валідаційну та тестову вибірки. Значну увагу приділено побудові та тренуванню моделі DETR для вирішення задачі одночасної локалізації та класифікації клітин. У рамках експериментів оцінено якість детекції та класифікації за метриками IoU, precision, recall та mAP. Також було проведено порівняння отриманих результатів із результатами інших моделей, зокрема Faster R-CNN, YOLOv8, RT-DETR та U-Net, що дозволило виявити переваги та недоліки трансформерного підходу в контексті медичної діагностики.

Практична значущість роботи полягає у розробці прототипу системи, здатної виявляти малярійні клітини без необхідності втручання спеціаліста, що відкриває перспективи для застосування такого підходу в автоматизованих скринінгових системах. Теоретичний внесок полягає в адаптації архітектури DETR до задачі медичної сегментації та її аналізі в порівнянні з класичними та сучасними рішеннями. Отримані результати підтверджують доцільність

використання трансформерних моделей у сфері медичних зображень, особливо у випадках, коли важливе поєднання точності виявлення та інтерпретованості моделі.

У подальших дослідженнях доцільно провести оптимізацію моделі за допомогою технік розширення даних, тонкого налаштування гіперпараметрів, а також розглянути варіанти гібридних архітектур (наприклад, DETR у поєднанні з CNN-базованими енкодерами). Крім того, перспективним напрямком є тестування моделі на зображеннях, отриманих із різних лабораторних джерел, що дозволить перевірити її здатність до узагальнення та адаптації до різних умов знімання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Broad Bioimage Benchmark Collection. URL: <https://bbbc.broadinstitute.org/BBBC041/> (Last accessed 01.06.2025).
2. Garrido-Cardenas J A., González-Cerón, L., Manzano-Agugliaro F., Mesa-Valle C. Plasmodium genomics: an approach for learning about and ending human malaria. Parasitology research, *Parasitol Res.* 2019. Vol. 118, № 1. P. 1 — 27. DOI: <https://doi.org/10.1007/s00436-018-6127-9>
3. malERA: An updated research agenda for malaria elimination and eradication / R. N. Rabinovich et. al. *PLoS Medicine*. 2017. Vol. 14, №. 11. DOI: <https://doi.org/10.1371/journal.pmed.1002456>.
4. Mueller I, Zimmerman PA, Reeder JC. Plasmodium malariae and Plasmodium ovale — the ‘bashful’ malaria parasites. *Trends in Parasitology*. 2007. Vol. 23 № 6. P. 278 — 283. DOI: <https://doi.org/10.1016/j.pt.2007.04.009>.
5. The age patterns of severe malaria syndromes in sub-Saharan Africa across a range of transmission intensities and seasonality settings / A. Roca-Feltrer et. al. *Malaria Journal*. 2010. Vol. 9, № 282. DOI: <https://doi.org/10.1186/1475-2875-9-282>.
6. Baird J.K. Neglect of Plasmodium vivax malaria. *Trends Parasitol.* 2007. Vol. 23, № 11. P. 533 — 539. DOI: <https://doi.org/10.1016/j.pt.2007.08.011>.
7. Geographical variation in Plasmodium vivax relapse / K.E. Battle et. al. *Malaria Journal*. 2014. Vol. 13, № 144. DOI: <https://doi.org/10.1186/1475-2875-13-144>.
8. White N.J. Determinants of relapse periodicity in Plasmodium vivax malaria. *Malaria Journal*. 2011. Vol. 10, № 297. DOI: <https://doi.org/10.1186/1475-2875-10-297>.
9. Smith R.C., Vega-Rodríguez J., Jacobs-Lorena M. The Plasmodium bottleneck: Malaria parasite losses in the mosquito vector. *Memorias do*

- Instituto Oswaldo Cruz* Oswaldo Cruz. 2014. Vol. 109, № 5. P. 640 — 661.
DOI: <https://doi.org/10.1590/0074-0276130597>.
10. Plasmodium Life Cycle. URL: http://www.tulane.edu/~wiser/malaria/mal_lc.PDF (Last accessed 01.06.2025).
 11. Tuju J., Kamuyu G., Murungi L.M., Osier F.H.A. Vaccine candidate discovery for the next generation of malaria vaccines. *Immunology* 2017. Vol. 152, № 2. P. 196 — 206. DOI: <https://doi.org/10.1111/imm.12780>.
 12. Advances and challenges in automated malaria diagnosis using digital microscopy imaging with artificial intelligence tools: A review / C.M. Rubio et. al. *Frontiers in Microbiology*. 2022. Vol. 13, № 1006659. DOI: <https://doi.org/10.3389/fmicb.2022.1006659>.
 13. Plasmodium vivax lifecycle in human host. URL: https://www.researchgate.net/figure/Plasmodium-vivax-lifecycle-in-human-host-P-vivax-is-transmitted-to-humans-by-the-bite_fig1_292189586 (Last accessed 01.06.2025).
 14. Tangpukdee N., Duangdee C., Wilairatana P., Krudsood S. Malaria diagnosis: a brief review. *The Korean Journal of Parasitology*. 2009. Vol. 47, № 2. P. 93 — 102. DOI: <https://doi.org/10.3347/kjp.2009.47.2.93>.
 15. World Health Organization. World Malaria Report 2021 URL: <https://www.who.int/publications/i/item/9789240040496>.
 16. Wongsrichanalai C., Barcus M.J., Muth S., Sutamihardja A. A review of malaria diagnostic tools: microscopy and rapid diagnostic test (RDT). *The American Journal of Tropical Medicine and Hygiene*. 2007. Vol. 77, № 6 P. 119 — 127.
 17. Vivax malaria: neglected and not benign / R.N. Price et. al. *The American Journal of Tropical Medicine and Hygiene*. 2007. Vol. 77, № 6. P. 79 — 87. DOI: <https://doi.org/10.4269/ajtmh.2007.77.79>.
 18. Bejon P., White N.J., Greenwood B. Malaria diagnosis and treatment recommendations. *Canada Communicable Disease Report*. 2014. Vol. 40, №

7. P. 133-143. DOI: <https://doi.org/10.14745/ccdr.v40i07a02>.
19. Use of a three-band HRP2/pLDH combination rapid diagnostic test increases diagnostic specificity for falciparum malaria in Ugandan children / M. Hawkes et. al. *Malaria Journal*. 2014. Vol. 13, № 43. DOI: <https://doi.org/10.1186/1475-2875-13-43>.
20. Poostchi M., Silamut K., Maude R.J., Jaeger S. Image analysis and machine learning for detecting malaria. *Malaria Journal*. 2018. Vol. 194. P. 36-55. DOI: <https://doi.org/10.1016/j.trsl.2017.12.004>.
21. World Malaria Report 2019. URL: <https://www.who.int/publications/i/item/9789241565721>.
22. Maltha J, Gillet P, Jacobs J. Malaria rapid diagnostic tests in endemic settings. *Malaria Journal*. 2013. Vol. 19, № 5. P. 399-407. DOI: <https://doi.org/10.1111/1469-0691.12151>.
23. Gillet P., Mori M., Van den Ende J., Jacobs J. Assessment of the prozone effect in malaria rapid diagnostic tests. *Malaria Journal*. 2009. Vol. 8, № 271. DOI: <https://doi.org/10.1186/1475-2875-8-271>.
24. Kotepui M., Kotepui K.U., Milanez G.D.J., Masangkay F.R. Prevalence and laboratory parameters of malaria and co-infection: A systematic review and meta-analysis. *BMC Infectious Diseases*. 2020. Vol. 20, № 869. DOI: <https://doi.org/10.1186/s12879-020-05689-5>.
25. PCR as a confirmatory technique for laboratory diagnosis of malaria / S. P. Johnston et. al. *Journal of clinical microbiology*. 2006. Vol. 44, № 3. DOI: <https://doi.org/10.1128/JCM.44.3.1087-1089.2006>.
26. Morris U., Aydin-Schmidt B. Performance and Application of Commercially Available Loop-Mediated Isothermal Amplification (LAMP) Kits in Malaria Endemic and Non-Endemic Settings. *Diagnostics*. 2021. Vol. 11, № 2. P. 336. DOI: <https://doi.org/10.3390/diagnostics11020336>.
27. Efficient deep learning-based approach for malaria detection using red blood cell smears. / M. Mujahid et. al. *Sci Rep*. 2024. Vol. 14, № 13249. DOI: <https://doi.org/10.1038/s41598-024-63831-0>.

28. A review on automated diagnosis of malaria parasite in microscopic blood smears images / Z. Jan et. al. *Multimedia Tools and Applications*. 2018. Vol. 77, № 8, P. 9801 — 9826. DOI: <https://doi.org/10.1007/s11042-017-4495-2>.
29. Microbe online, thick and thin blood smear for malaria diagnosis. URL: <https://microbeonline.com/microscopic-diagnosis-of-malaria/> (Last accessed 01.06.2025).
30. Fatima T., Farid M. S. Automatic detection of plasmodium parasites from microscopic blood images. *Journal of Parasitic Diseases*. 2020 Vol. 44. P. 69 — 78. DOI: <https://doi.org/10.1007/s12639-019-01163-x>.
31. Tek F. B., Dempster A. G., Kale İ. Adaptive gray world-based color normalization of thin blood film images. 2016. URL: <http://arxiv.org/abs/1607.04032> (Last accessed 01.06.2025).
32. Savkare S. S., Narote S. P. Automated system for malaria parasite identification, *Proceedings of the 2015 international conference on communication, information & computing technology (ICCICT)*. 2015. Mumbai, India. DOI: <https://doi.org/10.1109/iccict.2015.7045660>.
33. Bhatia S. New improved technique for initial cluster centers of K means clustering using Genetic Algorithm. *Proceedings of the International Conference for Convergence for Technology-2014*. 2014. Pune, India. DOI: <https://doi.org/10.1109/i2ct.2014.7092112>.
34. Ma C., Harrison P., Wang L., Coppel R. L. Automated estimation of parasitaemia of Plasmodium yoelii-infected mice by digital image analysis of Giemsa-stained thin blood smears. *Malaria Journal*. 2010. Vol. 9, №. 1. P. 348 — 349. DOI: <https://doi.org/10.1186/1475-2875-9-348>.
35. Image analysis and machine learning for detecting malaria / M. Poostchi et. al. *Transl. Res.* 2018. Vol. 194. P. 36 — 55. doi: DOI: <https://doi.org/10.1016/j.trsl.2017.12.004>.
36. Das D. K., Maiti A. K., Chakraborty C. Automated system for characterization and classification of malaria-infected stages using light

- microscopic images of thin blood smears. *Journal of Microscopy*. 2015. Vol. 257, №. 3. P. 238 — 252. DOI: <https://doi.org/10.1111/jmi.12206>.
37. Chayadevi M., Raju G. Usage of art for automatic malaria parasite identification based on fractal features. *International Journal of Video & Image Processing and Network Security*. 2014. Vol. 4. P. 7 — 15.
38. Malihi L., Ansari-Asl K., Behbahani A. Malaria parasite detection in giemsa-stained blood cell images. *Proceedings of the 2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*. 2013. Zanzan, Iran. DOI: <https://doi.org/10.1109/iranianmvip.2013.6780011>.
39. Fatima T, Farid MS. Automatic detection of Plasmodium parasites from microscopic blood images. *J Parasit Dis*. 2020. Vol. 44, № 1 P. 69-78. DOI: <https://doi.org/10.1007/s12639-019-01163-x>.
40. Goodfellow I., Bengio Y., Courville A. Deep Learning. The MIT Press Cambridge, Massachusetts London, England, 2017. <https://www.deeplearningbook.org/>.
41. Xu Y., Goodacre R. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of Analysis and Testing*. 2018. Vol. 2. P. 249 — 262. DOI: <https://doi.org/10.1007/s41664-018-0068-2>.
42. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images / S. Rajaraman et. al. *PeerJ*. 2020. Vol. 8. DOI: <https://doi.org/10.7717/peerj.4568>.
43. A Novel Data Augmentation Convolutional Neural Network for Detecting Malaria Parasite in Blood Smear Images / D. O. Oyewola et. al. *Applied Artificial Intelligence*. 2022. Vol. 36, № 1. DOI: <https://doi.org/10.1080/08839514.2022.2033473>.
44. Ren S., He K., Girshick R. Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern*

- Analysis and Machine Intelligence*. 2017 Vol. 39, № 6. P. 1137-1149. DOI: <https://doi.org/10.1109/TPAMI.2016.2577031>.
45. He K., Zhang X., Ren S., Sun J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: European Conference on Computer Vision (ECCV). 2014. DOI: https://doi.org/10.1007/978-3-319-10578-9_23.
 46. Rocha T., Ferreira C., Costa M. Evaluation of YOLO architectures for medical image detection. *Applied Sciences*. 2022. Vol. 12. № 4. DOI: <https://doi.org/10.3390/app12041813>.
 47. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning / H.C. Shin et. al. *IEEE Transactions on Medical Imaging*. 2016. Vol. 35, № 5. P. 1285 — 1298. DOI: <https://doi.org/10.1109/TMI.2016.2528162>.
 48. Ali S.A., Abdulqadir P.S., Abdullah S.A., Yunusa H. M2ANET: Mobile Malaria Attention Network for Efficient Classification of Plasmodium Parasites in Blood Cells. 2024. URL: <https://arxiv.org/abs/2405.14242> (Last accessed: 01.06.2025).
 49. Attention is all you need. Advances in Neural Information Processing Systems / A. Vaswani et. al. 2017. URL: <https://arxiv.org/abs/1706.03762> (Last accessed: 01.06.2025).
 50. Liu Z., Lin Y., Cao Y., Hu H. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 10 — 17 October 2021, Montreal, Canada. IEEE, 2021. DOI: <https://ieeexplore.ieee.org/document/9710580>.
 51. Tan D., Liang H. Multiclass malaria parasite recognition based on transformer models and GAN. *Scientific Reports*. 2023. Vol. 13, № 17136. DOI: <https://doi.org/10.1038/s41598-023-44297-y>.
 52. An image is worth 16x16 words: Transformers for image recognition at scale / A. Dosovitskiy et. al. 2021. URL: <https://arxiv.org/abs/2010.11929> (Last

- accessed: 01.06.2025).
53. Tan D., Liang X. Multiclass malaria parasite recognition based on transformer models and a generative adversarial network. *Sci Rep.* 2023. Vol. 13, № 17136. DOI: <https://doi.org/10.1038/s41598-023-44297-y>.
 54. Deep Learning for Real-Time Malaria Parasite Detection and Counting Using YOLO-mp / Doirala A. et. al. *IEEE Access*, 2022. Vol. 10, P. 102157 — 102172. DOI: <https://doi.org/10.1109/ACCESS.2022.3208270>.
 55. Automatic patient-level recognition of four Plasmodium species on thin blood smear by a real-time detection transformer (RT-DETR) object detection algorithm: a proof-of-concept and evaluation / E. Guemas et. al. *Microbiology Spectrum*. 2024. Vol. 12, № 2. DOI: <https://doi.org/10.1128/spectrum.01440-23>.
 56. Adigun M., Alake H., Balogun A. Malaria parasite detection using compact convolutional transformers. *Sensors*, 2022. Vol. 22, № 12. DOI: <https://doi.org/10.3390/s22124358>.
 57. DETR Breakdown Part 3: Architecture and Details. URL: <https://pyimagesearch.com/2023/06/26/detr-breakdown-part-3-architecture-and-details/> (Last accessed: 01.06.2025).
 58. Carion N., Massa F., Synnaeve G., Usunier N. End-to-End Object Detection with Transformers. Computer Vision — ECCV 2020: Proceedings of the 16th European Conference on Computer Vision, Glasgow, UK, 23 — 28 August 2020. Cham: Springer, 2020. DOI: https://doi.org/10.1007/978-3-030-58452-8_13.
 59. Parmar N., Vaswani A., Uszkoreit J., Kaiser L. Image Transformer. URL: <https://arxiv.org/abs/1802.05751>. (Last accessed: 01.06.2025).
 60. Kuhn H.W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955. Vol. 2, № 2. P. 83 — 97. DOI: <https://doi.org/10.1002/nav.3800020109>.
 61. Hybrid Matching for End-to-End Object Detection / S. Li et. al. 2024. URL: <https://arxiv.org/abs/2405.17677> (Last accessed: 01.06.2025).

62. A comparative analysis of deep learning architectures on high variation malaria parasite classification dataset / A. Rahman et. al. *Tissue & cell*. 2021 Vol. 69. P. 101473. DOI: <https://doi.org/10.1016/j.tice.2020.101473>.
63. DETRs beat YOLOs on real-time object detection / Y. Zhao et. al. 2023. URL: <https://arxiv.org/abs/2306.08668> (Last accessed: 01.06.2025).
64. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, Louisiana, USA, 6 — 9 May 2019. URL: <https://arxiv.org/abs/1711.05101> (Last accessed: 01.06.2025).
65. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin, M. W. Chang, K. Lee, K. Toutanova Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Minneapolis, USA, June 2–7, 2019. Association for Computational Linguistics, 2019. P. 4171–4186. — DOI: <https://doi.org/10.18653/v1/N19-1423>.
66. Carion N., Massa F., Synnaeve G. End-to-end object detection with transformers. Computer Vision — ECCV 2020: Proceedings of the 16th European Conference on Computer Vision, Glasgow, UK, 23 — 28 August 2020. Cham: Springer, 2020. P. 213 — 229. DOI: https://doi.org/10.1007/978-3-030-58452-8_13.
67. An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale / A. Dosovitskiy et. al. Proceedings of the International Conference on Learning Representations (ICLR 2021), Virtual Event, 3 — 7 May 2021. URL: <https://arxiv.org/abs/2010.11929> (Last accessed: 01.06.2025).
68. CSWin Transformer: A General Vision Transformer Backbone with Cross-Shaped Windows / X. Dong et. al. 2021. URL: <https://arxiv.org/abs/2107.00652> (Last accessed: 01.06.2025).
69. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and

- Pattern Recognition (CVPR 2016), Las Vegas, Nevada, USA, 26–30 June 2016. Los Alamitos, CA: IEEE, 2016. P. 770–778. DOI: <https://doi.org/10.1109/CVPR.2016.90>.
70. Deng J., Dong W., Socher R. ImageNet: A Large-Scale Hierarchical Image Database. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009), Miami, Florida, USA, 20–25 June 2009. Los Alamitos, CA: IEEE, 2009. P. 248–255. DOI: <https://doi.org/10.1109/CVPR.2009.5206848>.
71. End-to-End Object Detection with Transformers / N. Carion et. al. Proceedings of the 16th European Conference on Computer Vision (ECCV 2020), Glasgow, UK, 23–28 August 2020. URL: <https://arxiv.org/abs/2005.12872> (Last accessed: 01.06.2025).
72. FabioPC18. (2021). Malaria Cell Detection: Comparison of Methods. URL: https://github.com/FabioPC18/Malaria-Cell-Detection-Comparison-of-Methods/blob/master/Malaria_cell_detection_comparison_of_methods.pdf (Last accessed: 01.06.2025).
73. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems 28: Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NeurIPS 2015), Montreal, Canada, 7–12 December 2015. URL: <https://arxiv.org/abs/1506.01497> (Last accessed: 01.06.2025).
74. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. 2018. URL: <https://arxiv.org/abs/1804.02767> (Last accessed: 01.06.2025).
75. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention — MICCAI. 2015. P. 234 — 241.
76. Zhang Y., Wang Y., Li X., et al. RT-DETR: Real-Time Detection Transformer. 2023. URL: <https://arxiv.org/abs/2304.08069> (Last accessed: 01.06.2025).

- 77.Недашківська Н. І. Інтелектуальний аналіз даних: навч. посіб. для студ. спеціальності 124 «Системний аналіз», освітніх програм «Системний аналіз і управління», «Системний аналіз фінансового ринку». URL: <https://ela.kpi.ua/handle/123456789/53763> (Дата звернення: 01.06.2025).
- 78.Недашківська Н. І. Методи і моделі інтелектуального аналізу даних: навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки», освітньої програми «Системи і методи штучного інтелекту». URL: <https://ela.kpi.ua/handle/123456789/53764> (Дата звернення: 01.06.2025).
- 79.VanderPlas J. Python Data Science Handbook: Essential Tools for Working with Data. Sebastopol, CA. O'Reilly Media, 2017. 576 p.
- 80.Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA. The MIT Press, 2017. URL: <https://www.deeplearningbook.org/> (Last accessed: 01.06.2025).
- 81.Geron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol, CA. O'Reilly Media, 2017. 574 p.
- 82.McKinney W. Python for Data Analysis. Sebastopol, CA. O'Reilly Media, 2013. 482 p.
- 83.Brink H., Richards J. W., Fetherolf M. Real-World Machine Learning. Shelter Island, NY. Manning Publications, 2016. 336 p.
- 84.Mueller A. C., Guido S. An Introduction to Machine Learning with Python. Sebastopol, CA. O'Reilly Media, 2017. 392 p.

ДОДАТОК А ПРЕЗЕНТАЦІЯ

Класифікація та сегментація медичних зображень за допомогою моделі **Detection Transformer (DETR)**

Виконав: Іванюта О. О.
Дипломний керівник: Недашківська Н. І.

Актуальність дослідження

Малярія залишається одним із найнебезпечніших паразитарних захворювань у світі, з високими показниками захворюваності та смертності, особливо в країнах із низьким рівнем медичного забезпечення. Своєчасна та точна діагностика має вирішальне значення для ефективного лікування, проте традиційні методи виявлення (мікроскопія, швидкі тести) є трудомісткими, залежать від кваліфікації фахівців і часто характеризуються високим рівнем похибки.

У зв'язку з цим актуальною є розробка автоматизованих систем виявлення паразитованих клітин на основі сучасних методів комп'ютерного зору та глибокого навчання. Застосування моделей трансформерного типу, зокрема Detection Transformer (DETR), який відкриває нові перспективи для підвищення точності, швидкості та стабільності діагностичних рішень у галузі медичної візуалізації.

Мета роботи:

Дослідити можливості моделі Detection Transformer (DETR) для автоматизованої класифікації та сегментації клітин крові, уражених малярією, на основі медичних зображень.

Об'єкт дослідження:

Процес класифікації та сегментації крові, представленої у відкритому датасеті BBBC041, що включають як уражені, так і здорові клітини.

Предмет дослідження:

Моделі глибокого навчання на основі трансформерів, а саме архітектура Detection Transformer (DETR), її здатність до точної локалізації та класифікації клітин на зображеннях, а також методи попередньої обробки даних для задач біомедичної візуалізації.

3

Постановка задачі

- 1) Підготовка та анотація медичних зображень мазків крові з датасету BBBC041 для задачі детекції клітин, уражених малярією.
- 2) Вивчення та адаптація архітектури Detection Transformer (DETR) для задачі класифікації та сегментації клітин на медичних зображеннях.
- 3) Проведення навчання моделі DETR на підготовленому датасеті з урахуванням особливостей медичних зображень.
- 4) Оцінка якості роботи моделі за допомогою відповідних метрик об'єктної детекції та сегментації (AP, IoU, F1-score).
- 5) Порівняння результатів DETR з іншими сучасними підходами та аналіз їх переваг і недоліків у контексті задачі діагностики малярії.
- 6) Формування рекомендацій щодо подальшого використання та покращення моделі для медичних застосувань

4

Що таке Detection Transformer ?

Detection Transformer (DETR) — це сучасна модель глибокого навчання для задач детекції об'єктів, яка формулює проблему як пряме прогнозування множини об'єктів.

Модель використовує трансформер для глобального аналізу сцени, що дозволяє враховувати взаємозв'язки між об'єктами в зображенні. Завдяки використанню спеціальної функції втрат на основі парного відображення (Hungarian matching), DETR вчиться зіставляти кожну передбачену рамку з єдиною реальною, що зменшує дублювання і покращує якість локалізації.

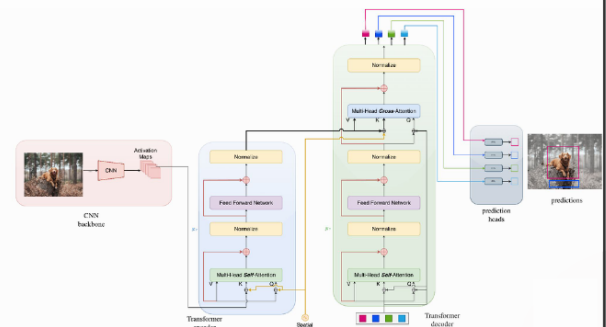
5

Загальна архітектура DETR

DETR — це end-to-end модель об'єктного виявлення, яка поєднує *CNN* для вилучення ознак із трансформерною архітектурою. Модель формулює задачу виявлення як задачу передбачення множини, що дозволяє уникнути традиційних евристик, як-от anchor boxes і *NMS*.

Компоненти:

- 1) *CNN-бекбон* (наприклад, ResNet-50).
- 2) *Трансформер* (енкодер + декодер).
- 3) *Feed-forward* голівки (класи + координати).

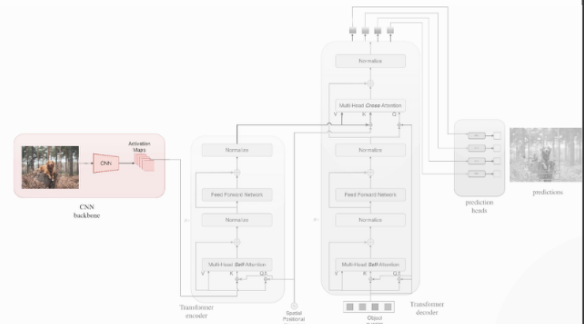


6

CNN Backbone

У моделі DETR згорткова нейронна мережа (CNN) використовується як бекбон для перетворення вхідного зображення у компактне, але інформативне представлення. Це представлення необхідне для подальшої обробки трансформером, який виконує виявлення та локалізацію об'єктів.

Вхідне зображення розміром $H_0 \times W_0 \times 3$ проходить через стандартну CNN (наприклад, ResNet-50), яка формує карту ознак зі зменшеними просторовими розмірами, але з великою кількістю каналів (2048). Зменшення масштабу зазвичай становить 32 рази по висоті та ширині.



7

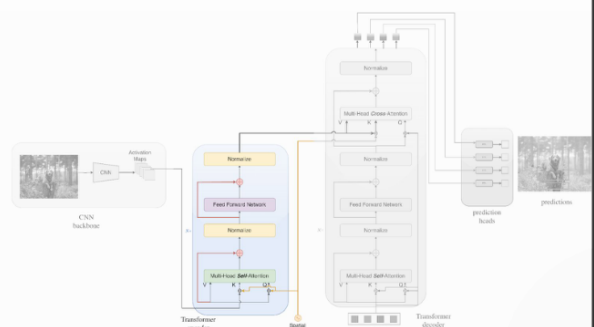
Encoder

Після того як CNN-бекбон сформував карту ознак, її перетворюють у послідовність токенів — векторів, які слугують входом для енкодера трансформера.

Енкодер складається з декількох ідентичних шарів, кожен з яких включає два основні блоки:

Multi-head Self-Attention, яка дозволяє кожному токenu враховувати інші токени у послідовності, що допомагає моделі враховувати контекст.

Повнозв'язна нейронна мережа, яка незалежно обробляє кожен токен, дозволяючи моделі трансформувати просторову інформацію у більш абстрактне представлення.

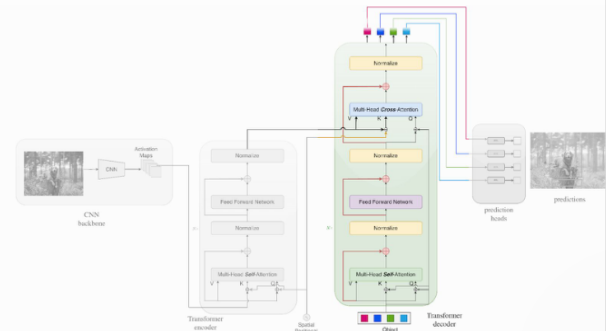


8

Decoder

У моделі Detection Transformer (DETR) декодер виконує ключову роль у виявленні об'єктів на зображенні. На відміну від класичних трансформерів, які формують вихід послідовно в авторегресивному режимі, у DETR обробка запитів відбувається паралельно. Це досягається шляхом подання на вхід декодера фіксованої кількості спеціальних векторів, що називаються запитами до об'єктів (*object queries*).

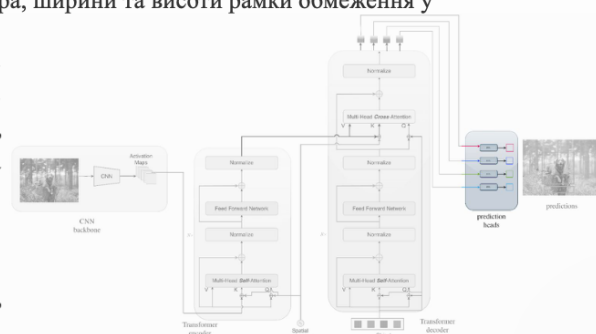
Ці запити не пов'язані з конкретними просторовими координатами або пікселями зображення. Натомість вони є навчуваними параметрами моделі, які мають здатність репрезентувати потенційні об'єкти, що можуть бути присутніми на зображенні. Декодер обробляє кожен з цих запитів, виконуючи два основні типи взаємодії: механізм самоуваги та механізм крос-уваги.



9

Feed-Forward Network (Prediction Heads)

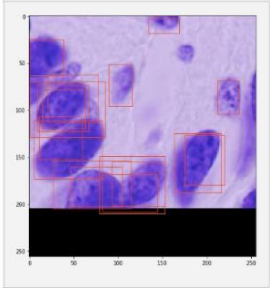
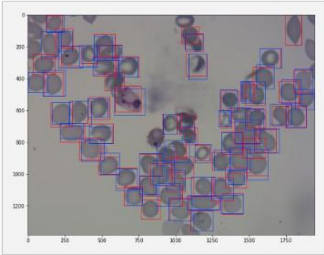
Заключний етап архітектури DETR реалізовано через трьохслойну повнозв'язну нейронну мережу з функцією активації *ReLU*, що дозволяє моделі прогнозувати параметри об'єктів на зображенні. Ця мережа відповідає за визначення класу кожного об'єкта, використовуючи лінійний шар із функцією *softmax*, яка розподіляє ймовірності по можливих категоріях, а також за точне оцінювання геометричних характеристик — центра, ширини та висоти рамки обмеження у нормалізованих координатах відносно розмірів вхідного зображення. Оскільки модель формує фіксований набір передбачень, їх кількість зазвичай перевищує реальну кількість об'єктів, тому для позначення відсутності об'єкта використовується спеціальний клас, подібний до класу "фон" у традиційних методах детекції. Такий підхід забезпечує коректне розмежування між реальними об'єктами та порожніми зонами, підвищуючи точність та надійність результатів.



10

Опис датасету та підготовки даних

Датасет BVBC041 містить медичні зображення мазків крові, уражених паразитом *Plasmodium vivax*, що викликає малярію. Загалом датасет включає понад 1300 високоякісних зображень з анотаціями клітин різних типів, включно з інфікованими та здоровими. Для підготовки даних до навчання було виконано попередню обробку, що включала нормалізацію зображень, конвертацію анотацій у формат, сумісний із моделлю DETR, а також застосування методів аугментації, таких як обертання, масштабування та горизонтальне відображення, для підвищення стійкості моделі до варіацій у зображеннях.



11

Опис датасету та підготовки даних

Для аналізу розподілу даних у нашому дослідженні було сформовано дві основні вибірки: тренувальну (Train) та тестову (Test). Детальні статистичні дані наведено у таблиці 1.

У початковому наборі даних клітини розподілені за кількома класами, що відображають різні типи та стадії інфекції. Для спрощення задачі класифікації було проведено об'єднання цих класів у дві категорії: «інфіковані» та «неінфіковані» клітини. Відповідність між оригінальними класами та бінарними категоріями подано у таблиці 2.

Набір	К-ть зображень	К-ть інфікованих	К-ть неінфікованих	Проблемність, %
Тренувальний	1208	2590	77523	3,2
Тестовий	120	308	5922	5,2
Загалом	1328	2898	83445	3,3

Таблиця 1. Train Vs Test

Оригінальний клас	Клас після перетворення
Еритроцити	Неінфікованих
Лейкоцити	
Кільця(Ring)	
Трофозоїти (Trophozoite)	Інфіковані
Шизонти (Schizont)	
Гаметоцити (Gametocyte)	
Важкі для класифікації	

Таблиця 2. оригінальними класами клітин та об'єднаними категоріями для бінарної класифікації.

12

Процес навчання моделі

Для підвищення якості класифікації та сегментації клітин крові модель DETR була донавчена на медичних зображеннях із датасету BBBC041. Початкові ваги моделі були отримані на загальному наборі даних.

Донавчання відбувалося з використанням оптимізатора *AdamW* із зниженою швидкістю навчання, що допомогло уникнути втрати вже набутих знань і одночасно адаптувати модель під специфіку медичних зображень. Розмір батчу та кількість епох були підібрані експериментально для балансу між якістю навчання та обчислювальними ресурсами.

Оцінка якості моделі здійснювалась за допомогою стандартних метрик для задач детекції — **mAP*, ***Precision*, ****Recall*, а також *****IoU* для оцінки точності локалізації. Цей підхід дозволив ефективно адаптувати модель DETR до специфічних особливостей зображень крові, покращуючи її здатність до автоматизованого виявлення та класифікації клітин, уражених малярією.

**mAP* — середнє значення точності класифікації об'єктів усіх класів.

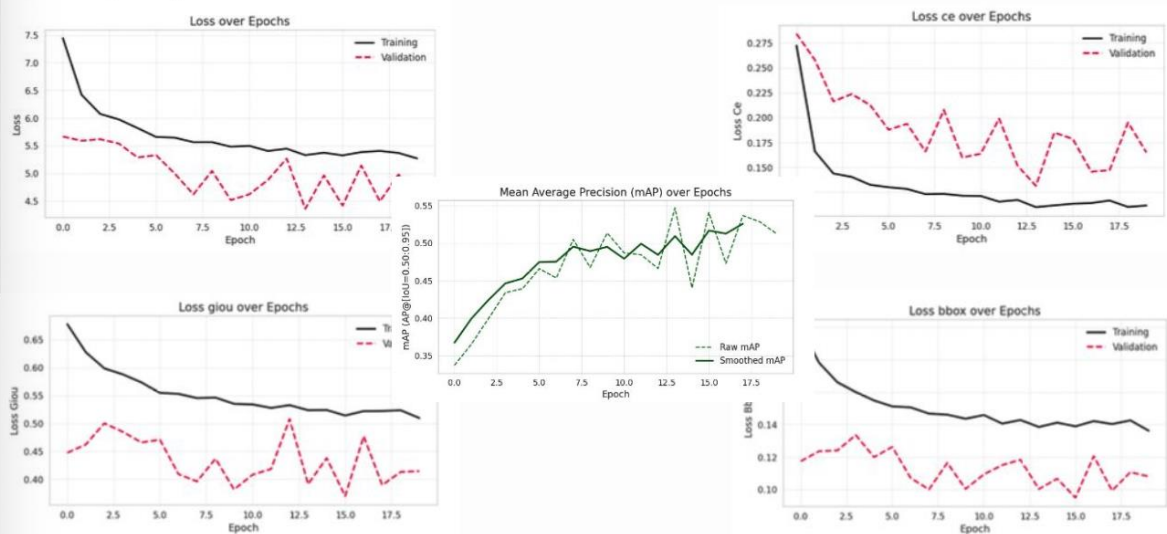
***Precision* — відсоток правильно знайдених об'єктів серед усіх знайдених.

****Recall* — відсоток правильно знайдених об'єктів серед усіх існуючих у зображенні.

*****IoU* — міра перекриття між передбаченою та реальною областю об'єкта.

13

Графіки навчання



14

Аналіз по класах

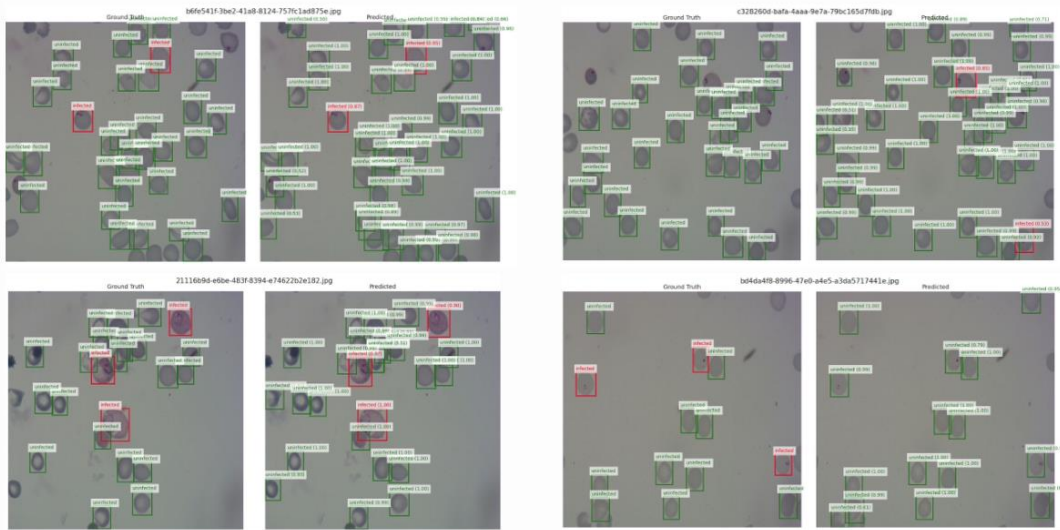
Після загальної оцінки метрик продуктивності моделі на валідаційному наборі (включаючи функції втрат та середнє значення mAP), доцільним було провести також локальну перевірку на окремих прикладах, щоб візуально та кількісно оцінити ефективність виявлення об'єктів у реальних умовах.

З цією метою було випадковим чином відібрано 5 зображень з валідаційного набору. На кожному з них модель здійснила виявлення клітин, після чого результати були співставлені з реальними (GT) розмітками. Для кожної з категорій — infected (інфіковані клітини) та uninfected (неінфіковані клітини) — було обчислено основні метрики: precision, recall, assuarcy, а також середнє значення IoU між передбаченими та реальними bounding box-ами.

Метрика	Uninfected	Infected
Загальна кількість GT (ground truth)	237	9
Кількість передбачених об'єктів	283	12
True Positives (TP)	226	8
False Positives (FP)	26	4
False Negatives (FN)	11	1
Precision	0.8968	0.6667
Recall	0.9536	0.8889
Середнє IoU	0.8441	0.8667

15

Візуалізація результатів



16

Порівняння моделі DETR з іншими методами

Метрика	Клас	U-net	F-RCNN	RT-DETR	YOLOv8	DETR
Precision	Інфіковані	/	0.4887	0.7542	0.751	0.7399
	Неінфіковані	/	0.8015	0.7732	0.867	0.8752
Recall	Інфіковані	/	0.7045	0.5877	0.68	0.6506
	Неінфіковані	/	0.9672	0.7512	0.994	0.9431
F1/Dice	Інфіковані	0.5341	0.5770	0.6606	0.7137	0.66
	Неінфіковані	0.9097	0.8765	0.762	0.9261	0.9105
IoU	Інфіковані	0.3772	0.8553	0.6841	0.9143	0.7003
	Неінфіковані	0.8352	0.8534	0.7699	0.9141	0.8468
Accuracy	Інфіковані	0.4129	0.7045	0.5877	0.4059	0.6006
	Неінфіковані	0.9215	0.9672	0.7512	0.7257	0.7431

17

Аналіз отриманих результатів

У таблиці представлено результати порівняння моделі DETR з іншими сучасними підходами (U-Net, Faster R-CNN, RT-DETR, YOLOv8), узятими зі звіту FabioPC18/Malaria-Cell-Detection-Comparison-of-Methods. Для коректності порівняння використано ті самі метрики (Precision, Recall, F1/Dice, IoU, Accuracy) для класів інфікованих та неінфікованих клітин.

DETR демонструє конкурентоспроможні результати, особливо в задачі виявлення інфікованих клітин: точність 0.7399, повнота 0.6506 та F1-метрика 0.66. Ці значення перевищують показники U-Net та Faster R-CNN. За IoU (0.7003 для інфікованих) модель також показує добру здатність до локалізації об'єктів.

Незважаючи на те, що YOLOv8 має дещо вищі результати за деякими метриками, DETR забезпечує стабільний баланс точності, повноти й узгодженості локалізації, що робить її надійним інструментом для задач автоматизованої діагностики малярії.

18

Висновки

У ході виконання дипломної роботи було проведено всебічне дослідження методів виявлення та класифікації клітин, інфікованих збудником малярії *Plasmodium vivax*, на основі аналізу мікроскопічних зображень крові. Ретельно проаналізовано традиційні підходи до діагностики захворювання, а також сучасні методи автоматизованої обробки медичних зображень.

Основну увагу було зосереджено на вивченні та застосуванні архітектури Detection Transformer (DETR), яка поєднує механізм самоуваги трансформерів з прямою регресією об'єктів. Після попередньої обробки та підготовки датасету BBBC041 було проведено донавчання моделі DETR на задачі двокласової класифікації клітин.

Отримані результати засвідчили ефективність підходу: модель DETR досягла точності 0.7399 та F1-метрики 0.66 для інфікованих клітин, що перевищує показники класичних моделей, таких як Faster R-CNN та U-Net, і є порівнюваними з більш сучасними YOLOv8 та RT-DETR. Водночас модель демонструє хорошу локалізацію об'єктів (IoU 0.7003), що є критично важливим для практичного використання в медичній діагностиці.

19

Перспективи подальших досліджень

Результатом дослідження стало створення ефективної моделі на основі DETR для задачі автоматизованого виявлення клітин, інфікованих *Plasmodium vivax*. Отримані результати свідчать про практичну доцільність використання трансформерних архітектур у медичних задачах комп'ютерного зору. Разом із цим відкриваються можливості для подальшого вдосконалення як моделі, так і її застосування.

Серед перспективних напрямів дослідження — використання більш потужних трансформерних архітектур, зокрема Deformable DETR чи DINO, які демонструють покращену локалізацію об'єктів і швидшу збіжність. Доцільним є також розширення датасету за рахунок інших паразитарних інфекцій або мікроскопічних зображень, отриманих з різних лабораторій, що дозволить підвищити генералізацію моделі.

Практичним кроком є розробка інтерактивного веб-інтерфейсу або мобільного застосунку, який забезпечить доступ лікарів до швидкої автоматичної діагностики на місці. Крім того, інтеграція пояснюваних AI-методів (наприклад, Grad-CAM) допоможе зробити процес прийняття рішень прозорішим і зручнішим для медичного персоналу.

20

ДЯКУЮ ЗА УВАГУ

ДОДАТОК Б ЛІСТИНГ КОДУ

blood_to_coco.py

```

import re
import os
import cv2
import json
import itertools
import numpy as np
from glob import glob
import scipy.io as sio
from pycocotools import mask as cocomask
from PIL import Image
categories = [
    {"supercategory": "none", "name": "difficult", "id": 0},
    {"supercategory": "none", "name": "gametocyte", "id": 1},
    {"supercategory": "none", "name": "leukocyte", "id": 2},
    {"supercategory": "none", "name": "red blood cell", "id": 3},
    {"supercategory": "none", "name": "ring", "id": 4},
    {"supercategory": "none", "name": "schizont", "id": 5},
    {"supercategory": "none", "name": "trophozoite", "id": 6}
]
phases = ["training", "test"]
for phase in phases:
    root_path = "/Users/alex/Downloads/malaria/images_new"
    gt_path = f"/Users/alex/Downloads/malaria/{phase}_new.json"
    json_file = f"{phase}.json"
    with open(gt_path, 'r') as f:
        data = json.load(f)
    res_file = {
        "categories": categories,
        "images": [],
        "annotations": []
    }
    annot_count = 0
    image_id = 0
    processed = 0
    for item in data:
        img_path = item["image"]["pathname"]
        img = Image.open(os.path.join(root_path, img_path))
        img_w, img_h = img.size
        img_elem = {
            "file_name": img_path,
            "height": img_h,
            "width": img_w,
            "id": image_id
        }
        res_file["images"].append(img_elem)
        for obj in item["objects"]:
            xmin = obj["bounding_box"]["minimum"]["c"]
            ymin = obj["bounding_box"]["minimum"]["r"]
            xmax = obj["bounding_box"]["maximum"]["c"]
            ymax = obj["bounding_box"]["maximum"]["r"]
            w = xmax - xmin
            h = ymax - ymin
            area = w * h
            category_name = obj["category"]

```

```

category_id = next(category['id'] for category in categories if category['name'] == category_name)
annot_elem = {
    "id": annot_count,
    "bbox": [float(xmin), float(ymin), float(w), float(h)],
    "segmentation": [],
    "image_id": image_id,
    "ignore": 0,
    "category_id": category_id,
    "iscrowd": 0,
    "area": float(area)
}
res_file["annotations"].append(annot_elem)
annot_count += 1
image_id += 1
processed += 1
with open(json_file, "w") as f:
    json.dump(res_file, f)
print(f'Processed {processed} {phase} images...')

```

blood.py

```

from pathlib import Path
import torch
import torch.utils.data
import torchvision
from pycocotools import mask as Blood_mask
import datasets.transforms as T

class BloodDetection(torchvision.datasets.CocoDetection):
    def __init__(self, img_folder, ann_file, transforms, return_masks):
        super(BloodDetection, self).__init__(img_folder, ann_file)
        self._transforms = transforms
        self.prepare = ConvertBloodPolysToMask(return_masks)

    def __getitem__(self, idx):
        img, target = super(BloodDetection, self).__getitem__(idx)
        image_id = self.ids[idx]
        target = {'image_id': image_id, 'annotations': target}
        img, target = self.prepare(img, target)
        if self._transforms is not None:
            img, target = self._transforms(img, target)
        return img, target

def convert_Blood_poly_to_mask(segmentations, height, width):
    masks = []
    for polygons in segmentations:
        rles = Blood_mask.frPyObjects(polygons, height, width)
        mask = Blood_mask.decode(rles)
        if len(mask.shape) < 3:
            mask = mask[..., None]
        mask = torch.as_tensor(mask, dtype=torch.uint8)
        mask = mask.any(dim=2)
        masks.append(mask)
    if masks:
        masks = torch.stack(masks, dim=0)
    else:
        masks = torch.zeros((0, height, width), dtype=torch.uint8)
    return masks

class ConvertBloodPolysToMask(object):
    def __init__(self, return_masks=False):

```

```

self.return_masks = return_masks

def __call__(self, image, target):
    w, h = image.size
    image_id = target["image_id"]
    image_id = torch.tensor([image_id])
    anno = target["annotations"]
    anno = [obj for obj in anno if 'iscrowd' not in obj or obj['iscrowd'] == 0]
    boxes = [obj["bbox"] for obj in anno]
    boxes = torch.as_tensor(boxes, dtype=torch.float32).reshape(-1, 4)
    boxes[:, 2:] += boxes[:, :2]
    boxes[:, 0::2].clamp_(min=0, max=w)
    boxes[:, 1::2].clamp_(min=0, max=h)
    classes = [obj["category_id"] for obj in anno]
    classes = torch.tensor(classes, dtype=torch.int64)
    if self.return_masks:
        segmentations = [obj["segmentation"] for obj in anno]
        masks = convert_Blood_poly_to_mask(segmentations, h, w)
    keypoints = None
    if anno and "keypoints" in anno[0]:
        keypoints = [obj["keypoints"] for obj in anno]
        keypoints = torch.as_tensor(keypoints, dtype=torch.float32)
        num_keypoints = keypoints.shape[0]
        if num_keypoints:
            keypoints = keypoints.view(num_keypoints, -1, 3)
    keep = (boxes[:, 3] > boxes[:, 1]) & (boxes[:, 2] > boxes[:, 0])
    boxes = boxes[keep]
    classes = classes[keep]
    if self.return_masks:
        masks = masks[keep]
    if keypoints is not None:
        keypoints = keypoints[keep]
    target = {}
    target["boxes"] = boxes
    target["labels"] = classes
    if self.return_masks:
        target["masks"] = masks
    target["image_id"] = image_id
    if keypoints is not None:
        target["keypoints"] = keypoints
    area = torch.tensor([obj["area"] for obj in anno])
    iscrowd = torch.tensor([obj["iscrowd"] if "iscrowd" in obj else 0 for obj in anno])
    target["area"] = area[keep]
    target["iscrowd"] = iscrowd[keep]
    target["orig_size"] = torch.as_tensor([int(h), int(w)])
    target["size"] = torch.as_tensor([int(h), int(w)])
    return image, target

def make_Blood_transforms(image_set):
    normalize = T.Compose([
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    scales = [480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800]
    if image_set == 'train':
        return T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(p=0.5),
            T.RandomColorJitter(),
            T.RandomBrightnessContrast(),
            T.RandomGaussianBlur(),
            T.RandomSelect(
                T.RandomResize(scales, max_size=1333),

```

```

        T.Compose([
            T.RandomResize([400, 500, 600]),
            T.RandomSizeCrop(384, 600),
            T.RandomResize(scales, max_size=1333),
        ])
    ),
    normalize,
])
if image_set == 'val':
    return T.Compose([
        T.RandomResize([800], max_size=1333),
        normalize,
    ])
raise ValueError(f'unknown {image_set}')

def build(image_set, args):
    root = Path(args.data_path)
    assert root.exists(), f'provided Blood path {root} does not exist'
    PATHS = {
        "train": (root / "images_new", root / f'train_coco_binary.json'),
        "val": (root / "images_new", root / f'test_coco_binary.json'),
    }
    img_folder, ann_file = PATHS[image_set]
    dataset = BloodDetection(img_folder, ann_file, transforms=make_Blood_transforms(image_set),
return_masks=args.masks)
    return dataset

```

backbone.py

```

from collections import OrderedDict
import torch
import torch.nn.functional as F
import torchvision
from torch import nn
from torchvision.models._utils import IntermediateLayerGetter
from typing import Dict, List
from util.misc import NestedTensor, is_main_process
from .position_encoding import build_position_encoding

class FrozenBatchNorm2d(torch.nn.Module):
    """
    BatchNorm2d where the batch statistics and the affine parameters are fixed.
    Copy-paste from torchvision.misc.ops with added eps before rqsrt,
    without which any other models than torchvision.models.resnet[18,34,50,101]
    produce nans.
    """

    def __init__(self, n):
        super(FrozenBatchNorm2d, self).__init__()
        self.register_buffer("weight", torch.ones(n))
        self.register_buffer("bias", torch.zeros(n))
        self.register_buffer("running_mean", torch.zeros(n))
        self.register_buffer("running_var", torch.ones(n))

    def _load_from_state_dict(self, state_dict, prefix, local_metadata, strict,
                            missing_keys, unexpected_keys, error_msgs):
        num_batches_tracked_key = prefix + 'num_batches_tracked'
        if num_batches_tracked_key in state_dict:
            del state_dict[num_batches_tracked_key]
        super(FrozenBatchNorm2d, self)._load_from_state_dict(
            state_dict, prefix, local_metadata, strict,

```

```

        missing_keys, unexpected_keys, error_msgs)

def forward(self, x):
    # move reshapes to the beginning
    # to make it fuser-friendly
    w = self.weight.reshape(1, -1, 1, 1)
    b = self.bias.reshape(1, -1, 1, 1)
    rv = self.running_var.reshape(1, -1, 1, 1)
    rm = self.running_mean.reshape(1, -1, 1, 1)
    eps = 1e-5
    scale = w * (rv + eps).rsqrt()
    bias = b - rm * scale
    return x * scale + bias

class BackboneBase(nn.Module):

    def __init__(self, backbone: nn.Module, train_backbone: bool, num_channels: int, return_intermediate_layers: bool):
        super().__init__()
        for name, parameter in backbone.named_parameters():
            if not train_backbone or 'layer2' not in name and 'layer3' not in name and 'layer4' not in name:
                parameter.requires_grad_(False)
        if return_intermediate_layers:
            return_layers = {"layer1": "0", "layer2": "1", "layer3": "2", "layer4": "3"}
        else:
            return_layers = {"layer4": "0"}
        self.body = IntermediateLayerGetter(backbone, return_layers=return_layers)
        self.num_channels = num_channels

    def forward(self, tensor_list: NestedTensor):
        xs = self.body(tensor_list.tensors)
        out: Dict[str, NestedTensor] = {}
        for name, x in xs.items():
            m = tensor_list.mask
            assert m is not None
            mask = F.interpolate(m[None].float(), size=x.shape[-2:]).to(torch.bool)[0]
            out[name] = NestedTensor(x, mask)
        return out

class Backbone(BackboneBase):
    """ResNet backbone with frozen BatchNorm."""
    def __init__(self, name: str,
                  train_backbone: bool,
                  return_intermediate_layers: bool,
                  dilation: bool):
        backbone = getattr(torchvision.models, name)(
            replace_stride_with_dilation=[False, False, dilation],
            pretrained=is_main_process(), norm_layer=FrozenBatchNorm2d)
        num_channels = 512 if name in ('resnet18', 'resnet34') else 2048
        super().__init__(backbone, train_backbone, num_channels, return_intermediate_layers)

class Joiner(nn.Sequential):
    def __init__(self, backbone, position_embedding):
        super().__init__(backbone, position_embedding)
    def forward(self, tensor_list: NestedTensor):
        xs = self[0](tensor_list)
        out: List[NestedTensor] = []
        pos = []
        for name, x in xs.items():
            out.append(x)
            # position encoding
            pos.append(self[1](x).to(x.tensors.dtype))

        return out, pos

```

```
def build_backbone(args):
    position_embedding = build_position_encoding(args)
    train_backbone = args.lr_backbone > 0
    return_interm_layers = args.masks
    backbone = Backbone(args.backbone, train_backbone, return_interm_layers, args.dilation)
    model = Joiner(backbone, position_embedding)
    model.num_channels = backbone.num_channels
    return model
```

detr.py

```
import torch
import torch.nn.functional as F
from torch import nn

from util import box_ops
from util.misc import (NestedTensor, nested_tensor_from_tensor_list,
                       accuracy, get_world_size, interpolate,
                       is_dist_avail_and_initialized)

from .backbone import build_backbone
from .matcher import build_matcher
from .segmentation import (DETRsegm, PostProcessPanoptic, PostProcessSegm,
                           dice_loss, sigmoid_focal_loss)
from .transformer import build_transformer

class DETR(nn.Module):
    """ This is the DETR module that performs object detection """
    def __init__(self, backbone, transformer, num_classes, num_queries, aux_loss=False):
        """ Initializes the model.
        Parameters:
            backbone: torch module of the backbone to be used. See backbone.py
            transformer: torch module of the transformer architecture. See transformer.py
            num_classes: number of object classes
            num_queries: number of object queries, ie detection slot. This is the maximal number of objects
                DETR can detect in a single image. For COCO, we recommend 100 queries.
            aux_loss: True if auxiliary decoding losses (loss at each decoder layer) are to be used.
        """
        super().__init__()
        self.num_queries = num_queries
        self.transformer = transformer
        hidden_dim = transformer.d_model
        self.class_embed = nn.Linear(hidden_dim, num_classes + 1)
        self.bbox_embed = MLP(hidden_dim, hidden_dim, 4, 3)
        self.query_embed = nn.Embedding(num_queries, hidden_dim)
        self.input_proj = nn.Conv2d(backbone.num_channels, hidden_dim, kernel_size=1)
        self.backbone = backbone
        self.aux_loss = aux_loss

    def forward(self, samples: NestedTensor):
        """ The forward expects a NestedTensor, which consists of:
            - samples.tensor: batched images, of shape [batch_size x 3 x H x W]
            - samples.mask: a binary mask of shape [batch_size x H x W], containing 1 on padded pixels

        It returns a dict with the following elements:
            - "pred_logits": the classification logits (including no-object) for all queries.
                Shape= [batch_size x num_queries x (num_classes + 1)]
            - "pred_boxes": The normalized boxes coordinates for all queries, represented as
                (center_x, center_y, height, width). These values are normalized in [0, 1],
        """
```

```

        relative to the size of each individual image (disregarding possible padding).
        See PostProcess for information on how to retrieve the unnormalized bounding box.
    - "aux_outputs": Optional, only returned when auxiliary losses are activated. It is a list of
        dictionaries containing the two above keys for each decoder layer.
    """
    if isinstance(samples, (list, torch.Tensor)):
        samples = nested_tensor_from_tensor_list(samples)
    features, pos = self.backbone(samples)

    src, mask = features[-1].decompose()
    assert mask is not None
    hs = self.transformer(self.input_proj(src), mask, self.query_embed.weight, pos[-1])[0]

    outputs_class = self.class_embed(hs)
    outputs_coord = self.bbox_embed(hs).sigmoid()
    out = {'pred_logits': outputs_class[-1], 'pred_boxes': outputs_coord[-1]}
    if self.aux_loss:
        out['aux_outputs'] = self._set_aux_loss(outputs_class, outputs_coord)
    return out

@torch.jit.unused
def _set_aux_loss(self, outputs_class, outputs_coord):
    # this is a workaround to make torchscript happy, as torchscript
    # doesn't support dictionary with non-homogeneous values, such
    # as a dict having both a Tensor and a list.
    return [{'pred_logits': a, 'pred_boxes': b}
            for a, b in zip(outputs_class[:-1], outputs_coord[:-1])]

class SetCriterion(nn.Module):
    """ This class computes the loss for DETR.
    The process happens in two steps:
    1) we compute hungarian assignment between ground truth boxes and the outputs of the model
    2) we supervise each pair of matched ground-truth / prediction (supervise class and box)
    """
    def __init__(self, num_classes, matcher, weight_dict, eos_coef, losses):
        """ Create the criterion.
        Parameters:
            num_classes: number of object categories, omitting the special no-object category
            matcher: module able to compute a matching between targets and proposals
            weight_dict: dict containing as key the names of the losses and as values their relative weight.
            eos_coef: relative classification weight applied to the no-object category
            losses: list of all the losses to be applied. See get_loss for list of available losses.
        """
        super().__init__()
        self.num_classes = num_classes
        self.matcher = matcher
        self.weight_dict = weight_dict
        self.eos_coef = eos_coef
        self.losses = losses
        empty_weight = torch.ones(self.num_classes + 1)
        empty_weight[-1] = self.eos_coef
        self.register_buffer('empty_weight', empty_weight)

    def loss_labels(self, outputs, targets, indices, num_boxes, log=True):
        """Classification loss (NLL)
        targets dicts must contain the key "labels" containing a tensor of dim [nb_target_boxes]
        """
        assert 'pred_logits' in outputs
        src_logits = outputs['pred_logits']

        idx = self._get_src_permutation_idx(indices)
        target_classes_o = torch.cat([t["labels"][J] for t, (_, J) in zip(targets, indices)])

```

```

target_classes = torch.full(src_logits.shape[:2], self.num_classes,
                             dtype=torch.int64, device=src_logits.device)
target_classes[idx] = target_classes_o

loss_ce = F.cross_entropy(src_logits.transpose(1, 2), target_classes, self.empty_weight)
losses = {'loss_ce': loss_ce}

if log:
    # TODO this should probably be a separate loss, not hacked in this one here
    losses['class_error'] = 100 - accuracy(src_logits[idx], target_classes_o)[0]
return losses

@torch.no_grad()
def loss_cardinality(self, outputs, targets, indices, num_boxes):
    """ Compute the cardinality error, ie the absolute error in the number of predicted non-empty boxes
    This is not really a loss, it is intended for logging purposes only. It doesn't propagate gradients
    """
    pred_logits = outputs['pred_logits']
    device = pred_logits.device
    tgt_lengths = torch.as_tensor([len(v["labels"]) for v in targets], device=device)
    # Count the number of predictions that are NOT "no-object" (which is the last class)
    card_pred = (pred_logits.argmax(-1) != pred_logits.shape[-1] - 1).sum(1)
    card_err = F.l1_loss(card_pred.float(), tgt_lengths.float())
    losses = {'cardinality_error': card_err}
    return losses

def loss_boxes(self, outputs, targets, indices, num_boxes):
    """Compute the losses related to the bounding boxes, the L1 regression loss and the GIoU loss
    targets dicts must contain the key "boxes" containing a tensor of dim [nb_target_boxes, 4]
    The target boxes are expected in format (center_x, center_y, w, h), normalized by the image size.
    """
    assert 'pred_boxes' in outputs
    idx = self._get_src_permutation_idx(indices)
    src_boxes = outputs['pred_boxes'][idx]
    target_boxes = torch.cat([t['boxes'][i] for t, (_, i) in zip(targets, indices)], dim=0)

    loss_bbox = F.l1_loss(src_boxes, target_boxes, reduction='none')

    losses = {}
    losses['loss_bbox'] = loss_bbox.sum() / num_boxes

    loss_giou = 1 - torch.diag(box_ops.generalized_box_iou(
        box_ops.box_cxcywh_to_xyxy(src_boxes),
        box_ops.box_cxcywh_to_xyxy(target_boxes)))
    losses['loss_giou'] = loss_giou.sum() / num_boxes
    return losses

def loss_masks(self, outputs, targets, indices, num_boxes):
    """Compute the losses related to the masks: the focal loss and the dice loss.
    targets dicts must contain the key "masks" containing a tensor of dim [nb_target_boxes, h, w]
    """
    assert "pred_masks" in outputs

    src_idx = self._get_src_permutation_idx(indices)
    tgt_idx = self._get_tgt_permutation_idx(indices)
    src_masks = outputs["pred_masks"]
    src_masks = src_masks[src_idx]
    masks = [t["masks"] for t in targets]
    # TODO use valid to mask invalid areas due to padding in loss
    target_masks, valid = nested_tensor_from_tensor_list(masks).decompose()
    target_masks = target_masks.to(src_masks)
    target_masks = target_masks[tgt_idx]

```



```

        if loss == 'labels':
            # Logging is enabled only for the last layer
            kwargs = {'log': False}
            l_dict = self.get_loss(loss, aux_outputs, targets, indices, num_boxes, **kwargs)
            l_dict = {k + f'_{i}': v for k, v in l_dict.items()}
            losses.update(l_dict)

    return losses

class PostProcess(nn.Module):
    """ This module converts the model's output into the format expected by the coco api"""
    @torch.no_grad()
    def forward(self, outputs, target_sizes):
        """ Perform the computation
        Parameters:
            outputs: raw outputs of the model
            target_sizes: tensor of dimension [batch_size x 2] containing the size of each images of the batch
                For evaluation, this must be the original image size (before any data augmentation)
                For visualization, this should be the image size after data augment, but before padding
        """
        out_logits, out_bbox = outputs['pred_logits'], outputs['pred_boxes']
        assert len(out_logits) == len(target_sizes)
        assert target_sizes.shape[1] == 2
        prob = F.softmax(out_logits, -1)
        scores, labels = prob[..., :-1].max(-1)
        # convert to [x0, y0, x1, y1] format
        boxes = box_ops.box_cxcywh_to_xyxy(out_bbox)
        # and from relative [0, 1] to absolute [0, height] coordinates
        img_h, img_w = target_sizes.unbind(1)
        scale_fct = torch.stack([img_w, img_h, img_w, img_h], dim=1)
        boxes = boxes * scale_fct[:, None, :]
        results = [{'scores': s, 'labels': l, 'boxes': b} for s, l, b in zip(scores, labels, boxes)]
        return results

class MLP(nn.Module):
    """ Very simple multi-layer perceptron (also called FFN)"""
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers):
        super().__init__()
        self.num_layers = num_layers
        h = [hidden_dim] * (num_layers - 1)
        self.layers = nn.ModuleList(nn.Linear(n, k) for n, k in zip([input_dim] + h, h + [output_dim]))

    def forward(self, x):
        for i, layer in enumerate(self.layers):
            x = F.relu(layer(x)) if i < self.num_layers - 1 else layer(x)
        return x

def build(args):
    # num_classes = 20 if args.dataset_file != 'coco' else 91
    # if args.dataset_file == "coco_panoptic":
    #     num_classes = 250
    if args.dataset_file == "blood":
        num_classes = 2
    device = torch.device(args.device)
    backbone = build_backbone(args)
    transformer = build_transformer(args)
    num_classes = args.num_classes ###
    model = DETR(
        backbone,
        transformer,
        num_classes=num_classes,
        num_queries=args.num_queries,
        aux_loss=args.aux_loss,

```

```

)
if args.masks:
    model = DETRsegm(model, freeze_detr=(args.frozen_weights is not None))
matcher = build_matcher(args)
weight_dict = {'loss_ce': 1, 'loss_bbox': args.bbox_loss_coef}
weight_dict['loss_giou'] = args.giou_loss_coef
if args.masks:
    weight_dict["loss_mask"] = args.mask_loss_coef
    weight_dict["loss_dice"] = args.dice_loss_coef
# TODO this is a hack
if args.aux_loss:
    aux_weight_dict = {}
    for i in range(args.dec_layers - 1):
        aux_weight_dict.update({k + f'_{i}': v for k, v in weight_dict.items()})
    weight_dict.update(aux_weight_dict)
losses = ['labels', 'boxes', 'cardinality']
if args.masks:
    losses += ["masks"]
criterion = SetCriterion(num_classes, matcher=matcher, weight_dict=weight_dict,
                        eos_coef=args.eos_coef, losses=losses)
criterion.to(device)
postprocessors = {'bbox': PostProcess()}
if args.masks:
    postprocessors['segm'] = PostProcessSegm()
    if args.dataset_file == "coco_panoptic":
        is_thing_map = {i: i <= 90 for i in range(201)}
        postprocessors["panoptic"] = PostProcessPanoptic(is_thing_map, threshold=0.85)
return model, criterion, postprocessors

```

matcher.py

```

import torch
from scipy.optimize import linear_sum_assignment
from torch import nn
from util.box_ops import box_cxcywh_to_xyxy, generalized_box_iou
class HungarianMatcher(nn.Module):
    """This class computes an assignment between the targets and the predictions of the network

    For efficiency reasons, the targets don't include the no_object. Because of this, in general,
    there are more predictions than targets. In this case, we do a 1-to-1 matching of the best predictions,
    while the others are un-matched (and thus treated as non-objects).
    """

    def __init__(self, cost_class: float = 1, cost_bbox: float = 1, cost_giou: float = 1):
        """Creates the matcher

        Params:
            cost_class: This is the relative weight of the classification error in the matching cost
            cost_bbox: This is the relative weight of the L1 error of the bounding box coordinates in the matching cost
            cost_giou: This is the relative weight of the giou loss of the bounding box in the matching cost
        """
        super().__init__()
        self.cost_class = cost_class
        self.cost_bbox = cost_bbox
        self.cost_giou = cost_giou
        assert cost_class != 0 or cost_bbox != 0 or cost_giou != 0, "all costs cant be 0"

    @torch.no_grad()
    def forward(self, outputs, targets):
        """ Performs the matching

        Params:
            outputs: This is a dict that contains at least these entries:

```

"pred_logits": Tensor of dim [batch_size, num_queries, num_classes] with the classification logits
 "pred_boxes": Tensor of dim [batch_size, num_queries, 4] with the predicted box coordinates
 targets: This is a list of targets (len(targets) = batch_size), where each target is a dict containing:
 "labels": Tensor of dim [num_target_boxes] (where num_target_boxes is the number of ground-truth objects in the target) containing the class labels
 "boxes": Tensor of dim [num_target_boxes, 4] containing the target box coordinates

Returns:

A list of size batch_size, containing tuples of (index_i, index_j) where:

- index_i is the indices of the selected predictions (in order)
- index_j is the indices of the corresponding selected targets (in order)

For each batch element, it holds:

len(index_i) = len(index_j) = min(num_queries, num_target_boxes)

"""

```
bs, num_queries = outputs["pred_logits"].shape[:2]
out_prob = outputs["pred_logits"].flatten(0, 1).softmax(-1) # [batch_size * num_queries, num_classes]
out_bbox = outputs["pred_boxes"].flatten(0, 1) # [batch_size * num_queries, 4]
tgt_ids = torch.cat([v["labels"] for v in targets])
tgt_bbox = torch.cat([v["boxes"] for v in targets])
cost_class = -out_prob[:, tgt_ids]
cost_bbox = torch.cdist(out_bbox, tgt_bbox, p=1)
cost_giou = -generalized_box_iou(box_cxcywh_to_xyxy(out_bbox), box_cxcywh_to_xyxy(tgt_bbox))
C = self.cost_bbox * cost_bbox + self.cost_class * cost_class + self.cost_giou * cost_giou
C = C.view(bs, num_queries, -1).cpu()
sizes = [len(v["boxes"]) for v in targets]
indices = [linear_sum_assignment(c[i]) for i, c in enumerate(C.split(sizes, -1))]
return [(torch.as_tensor(i, dtype=torch.int64), torch.as_tensor(j, dtype=torch.int64)) for i, j in indices]
```

```
def build_matcher(args):
```

```
    return HungarianMatcher(cost_class=args.set_cost_class, cost_bbox=args.set_cost_bbox,
cost_giou=args.set_cost_giou)
```

position_encoding.py

```
import math
import torch
from torch import nn
from util.misc import NestedTensor
class PositionEmbeddingSine(nn.Module):
    def __init__(self, num_pos_feats=64, temperature=10000, normalize=False, scale=None):
        super().__init__()
        self.num_pos_feats = num_pos_feats
        self.temperature = temperature
        self.normalize = normalize
        if scale is not None and normalize is False:
            raise ValueError("normalize should be True if scale is passed")
        if scale is None:
            scale = 2 * math.pi
        self.scale = scale

    def forward(self, tensor_list: NestedTensor):
        x = tensor_list.tensors
        mask = tensor_list.mask
        assert mask is not None
        not_mask = ~mask
        y_embed = not_mask.cumsum(1, dtype=torch.float32)
        x_embed = not_mask.cumsum(2, dtype=torch.float32)
        if self.normalize:
            eps = 1e-6
            y_embed = y_embed / (y_embed[:, -1:, :] + eps) * self.scale
            x_embed = x_embed / (x_embed[:, :, -1:] + eps) * self.scale
        dim_t = torch.arange(self.num_pos_feats, dtype=torch.float32, device=x.device)
```

```

dim_t = self.temperature ** (2 * (dim_t // 2) / self.num_pos_feats)
pos_x = x_embed[:, :, :, None] / dim_t
pos_y = y_embed[:, :, :, None] / dim_t
pos_x = torch.stack((pos_x[:, :, :, 0::2].sin(), pos_x[:, :, :, 1::2].cos()), dim=4).flatten(3)
pos_y = torch.stack((pos_y[:, :, :, 0::2].sin(), pos_y[:, :, :, 1::2].cos()), dim=4).flatten(3)
pos = torch.cat((pos_y, pos_x), dim=3).permute(0, 3, 1, 2)
return pos

class PositionEmbeddingLearned(nn.Module):
    def __init__(self, num_pos_feats=256):
        super().__init__()
        self.row_embed = nn.Embedding(50, num_pos_feats)
        self.col_embed = nn.Embedding(50, num_pos_feats)
        self.reset_parameters()

    def reset_parameters(self):
        nn.init.uniform_(self.row_embed.weight)
        nn.init.uniform_(self.col_embed.weight)

    def forward(self, tensor_list: NestedTensor):
        x = tensor_list.tensors
        h, w = x.shape[-2:]
        i = torch.arange(w, device=x.device)
        j = torch.arange(h, device=x.device)
        x_emb = self.col_embed(i)
        y_emb = self.row_embed(j)
        pos = torch.cat([
            x_emb.unsqueeze(0).repeat(h, 1, 1),
            y_emb.unsqueeze(1).repeat(1, w, 1),
        ], dim=-1).permute(2, 0, 1).unsqueeze(0).repeat(x.shape[0], 1, 1, 1)
        return pos

def build_position_encoding(args):
    N_steps = args.hidden_dim // 2
    if args.position_embedding in ('v2', 'sine'):
        # TODO find a better way of exposing other arguments
        position_embedding = PositionEmbeddingSine(N_steps, normalize=True)
    elif args.position_embedding in ('v3', 'learned'):
        position_embedding = PositionEmbeddingLearned(N_steps)
    else:
        raise ValueError(f"not supported {args.position_embedding}")
    return position_embedding

```

segmentation.py

```

import io
from collections import defaultdict
from typing import List, Optional
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import Tensor
from PIL import Image
import util.box_ops as box_ops
from util.misc import NestedTensor, interpolate, nested_tensor_from_tensor_list
try:
    from panopticapi.utils import id2rgb, rgb2id
except ImportError:
    pass

class DETRsegm(nn.Module):

```

```

def __init__(self, detr, freeze_detr=False):
    super().__init__()
    self.detr = detr
    if freeze_detr:
        for p in self.parameters():
            p.requires_grad_(False)
    hidden_dim, nheads = detr.transformer.d_model, detr.transformer.nhead
    self.bbox_attention = MHAttentionMap(hidden_dim, hidden_dim, nheads, dropout=0.0)
    self.mask_head = MaskHeadSmallConv(hidden_dim + nheads, [1024, 512, 256], hidden_dim)

def forward(self, samples: NestedTensor):
    if isinstance(samples, (list, torch.Tensor)):
        samples = nested_tensor_from_tensor_list(samples)
    features, pos = self.detr.backbone(samples)
    bs = features[-1].tensors.shape[0]
    src, mask = features[-1].decompose()
    assert mask is not None
    src_proj = self.detr.input_proj(src)
    hs, memory = self.detr.transformer(src_proj, mask, self.detr.query_embed.weight, pos[-1])
    outputs_class = self.detr.class_embed(hs)
    outputs_coord = self.detr.bbox_embed(hs).sigmoid()
    out = {"pred_logits": outputs_class[-1], "pred_boxes": outputs_coord[-1]}
    if self.detr.aux_loss:
        out['aux_outputs'] = self.detr._set_aux_loss(outputs_class, outputs_coord)

    # FIXME h_boxes takes the last one computed, keep this in mind
    bbox_mask = self.bbox_attention(hs[-1], memory, mask=mask)

    seg_masks = self.mask_head(src_proj, bbox_mask, [features[2].tensors, features[1].tensors, features[0].tensors])
    outputs_seg_masks = seg_masks.view(bs, self.detr.num_queries, seg_masks.shape[-2], seg_masks.shape[-1])

    out["pred_masks"] = outputs_seg_masks
    return out

def _expand(tensor, length: int):
    return tensor.unsqueeze(1).repeat(1, int(length), 1, 1, 1).flatten(0, 1)

class MaskHeadSmallConv(nn.Module):
    def __init__(self, dim, fpn_dims, context_dim):
        super().__init__()
        inter_dims = [dim, context_dim // 2, context_dim // 4, context_dim // 8, context_dim // 16, context_dim // 64]
        self.lay1 = torch.nn.Conv2d(dim, dim, 3, padding=1)
        self.gn1 = torch.nn.GroupNorm(8, dim)
        self.lay2 = torch.nn.Conv2d(dim, inter_dims[1], 3, padding=1)
        self.gn2 = torch.nn.GroupNorm(8, inter_dims[1])
        self.lay3 = torch.nn.Conv2d(inter_dims[1], inter_dims[2], 3, padding=1)
        self.gn3 = torch.nn.GroupNorm(8, inter_dims[2])
        self.lay4 = torch.nn.Conv2d(inter_dims[2], inter_dims[3], 3, padding=1)
        self.gn4 = torch.nn.GroupNorm(8, inter_dims[3])
        self.lay5 = torch.nn.Conv2d(inter_dims[3], inter_dims[4], 3, padding=1)
        self.gn5 = torch.nn.GroupNorm(8, inter_dims[4])
        self.out_lay = torch.nn.Conv2d(inter_dims[4], 1, 3, padding=1)
        self.dim = dim
        self.adapter1 = torch.nn.Conv2d(fpn_dims[0], inter_dims[1], 1)
        self.adapter2 = torch.nn.Conv2d(fpn_dims[1], inter_dims[2], 1)
        self.adapter3 = torch.nn.Conv2d(fpn_dims[2], inter_dims[3], 1)
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_uniform_(m.weight, a=1)
                nn.init.constant_(m.bias, 0)

def forward(self, x: Tensor, bbox_mask: Tensor, fpns: List[Tensor]):

```

```

x = torch.cat([_expand(x, bbox_mask.shape[1]), bbox_mask.flatten(0, 1)], 1)
x = self.lay1(x)
x = self.gn1(x)
x = F.relu(x)
x = self.lay2(x)
x = self.gn2(x)
x = F.relu(x)
cur_fpn = self.adapter1(fpns[0])
if cur_fpn.size(0) != x.size(0):
    cur_fpn = _expand(cur_fpn, x.size(0) // cur_fpn.size(0))
x = cur_fpn + F.interpolate(x, size=cur_fpn.shape[-2:], mode="nearest")
x = self.lay3(x)
x = self.gn3(x)
x = F.relu(x)
cur_fpn = self.adapter2(fpns[1])
if cur_fpn.size(0) != x.size(0):
    cur_fpn = _expand(cur_fpn, x.size(0) // cur_fpn.size(0))
x = cur_fpn + F.interpolate(x, size=cur_fpn.shape[-2:], mode="nearest")
x = self.lay4(x)
x = self.gn4(x)
x = F.relu(x)
cur_fpn = self.adapter3(fpns[2])
if cur_fpn.size(0) != x.size(0):
    cur_fpn = _expand(cur_fpn, x.size(0) // cur_fpn.size(0))
x = cur_fpn + F.interpolate(x, size=cur_fpn.shape[-2:], mode="nearest")
x = self.lay5(x)
x = self.gn5(x)
x = F.relu(x)

x = self.out_lay(x)
return x

```

```
class MHAttentionMap(nn.Module):
```

```

    def __init__(self, query_dim, hidden_dim, num_heads, dropout=0.0, bias=True):
        super().__init__()
        self.num_heads = num_heads
        self.hidden_dim = hidden_dim
        self.dropout = nn.Dropout(dropout)
        self.q_linear = nn.Linear(query_dim, hidden_dim, bias=bias)
        self.k_linear = nn.Linear(query_dim, hidden_dim, bias=bias)
        nn.init.zeros_(self.k_linear.bias)
        nn.init.zeros_(self.q_linear.bias)
        nn.init.xavier_uniform_(self.k_linear.weight)
        nn.init.xavier_uniform_(self.q_linear.weight)
        self.normalize_fact = float(hidden_dim / self.num_heads) ** -0.5
    def forward(self, q, k, mask: Optional[Tensor] = None):
        q = self.q_linear(q)
        k = F.conv2d(k, self.k_linear.weight.unsqueeze(-1).unsqueeze(-1), self.k_linear.bias)
        qh = q.view(q.shape[0], q.shape[1], self.num_heads, self.hidden_dim // self.num_heads)
        kh = k.view(k.shape[0], self.num_heads, self.hidden_dim // self.num_heads, k.shape[-2], k.shape[-1])
        weights = torch.einsum("bqnc,bnchw->bqnhw", qh * self.normalize_fact, kh)
        if mask is not None:
            weights.masked_fill_(mask.unsqueeze(1).unsqueeze(1), float("-inf"))
        weights = F.softmax(weights.flatten(2), dim=-1).view(weights.size())
        weights = self.dropout(weights)
        return weights

```

```

def dice_loss(inputs, targets, num_boxes):
    inputs = inputs.sigmoid()
    inputs = inputs.flatten(1)
    numerator = 2 * (inputs * targets).sum(1)

```

```

denominator = inputs.sum(-1) + targets.sum(-1)
loss = 1 - (numerator + 1) / (denominator + 1)
return loss.sum() / num_boxes

def sigmoid_focal_loss(inputs, targets, num_boxes, alpha: float = 0.25, gamma: float = 2):
    prob = inputs.sigmoid()
    ce_loss = F.binary_cross_entropy_with_logits(inputs, targets, reduction="none")
    p_t = prob * targets + (1 - prob) * (1 - targets)
    loss = ce_loss * ((1 - p_t) ** gamma)
    if alpha >= 0:
        alpha_t = alpha * targets + (1 - alpha) * (1 - targets)
        loss = alpha_t * loss
    return loss.mean(1).sum() / num_boxes

class PostProcessSegm(nn.Module):
    def __init__(self, threshold=0.5):
        super().__init__()
        self.threshold = threshold
    @torch.no_grad()
    def forward(self, results, outputs, orig_target_sizes, max_target_sizes):
        assert len(orig_target_sizes) == len(max_target_sizes)
        max_h, max_w = max_target_sizes.max(0)[0].tolist()
        outputs_masks = outputs["pred_masks"].squeeze(2)
        outputs_masks = F.interpolate(outputs_masks, size=(max_h, max_w), mode="bilinear", align_corners=False)
        outputs_masks = (outputs_masks.sigmoid() > self.threshold).cpu()
        for i, (cur_mask, t, tt) in enumerate(zip(outputs_masks, max_target_sizes, orig_target_sizes)):
            img_h, img_w = t[0], t[1]
            results[i]["masks"] = cur_mask[:, :img_h, :img_w].unsqueeze(1)
            results[i]["masks"] = F.interpolate(
                results[i]["masks"].float(), size=tuple(tt.tolist()), mode="nearest"
            ).byte()
        return results

class PostProcessPanoptic(nn.Module):
    def __init__(self, is_thing_map, threshold=0.85):
        super().__init__()
        self.threshold = threshold
        self.is_thing_map = is_thing_map

    def forward(self, outputs, processed_sizes, target_sizes=None):
        if target_sizes is None:
            target_sizes = processed_sizes
        assert len(processed_sizes) == len(target_sizes)
        out_logits, raw_masks, raw_boxes = outputs["pred_logits"], outputs["pred_masks"], outputs["pred_boxes"]
        assert len(out_logits) == len(raw_masks) == len(target_sizes)
        preds = []

        def to_tuple(tup):
            if isinstance(tup, tuple):
                return tup
            return tuple(tup.cpu().tolist())

        for cur_logits, cur_masks, cur_boxes, size, target_size in zip(
            out_logits, raw_masks, raw_boxes, processed_sizes, target_sizes
        ):
            scores, labels = cur_logits.softmax(-1).max(-1)
            keep = labels.ne(outputs["pred_logits"].shape[-1] - 1) & (scores > self.threshold)
            cur_scores, cur_classes = cur_logits.softmax(-1).max(-1)
            cur_scores = cur_scores[keep]
            cur_classes = cur_classes[keep]
            cur_masks = cur_masks[keep]
            cur_masks = interpolate(cur_masks[:, None], to_tuple(size), mode="bilinear").squeeze(1)
            cur_boxes = box_ops.box_cxcywh_to_xyxy(cur_boxes[keep])
            h, w = cur_masks.shape[-2:]

```

```

assert len(cur_boxes) == len(cur_classes)
cur_masks = cur_masks.flatten(1)
stuff_equiv_classes = defaultdict(lambda: [])
for k, label in enumerate(cur_classes):
    if not self.is_thing_map[label.item()]:
        stuff_equiv_classes[label.item()].append(k)
def get_ids_area(masks, scores, dedup=False):
    m_id = masks.transpose(0, 1).softmax(-1)
    if m_id.shape[-1] == 0:
        # We didn't detect any mask :(
        m_id = torch.zeros((h, w), dtype=torch.long, device=m_id.device)
    else:
        m_id = m_id.argmax(-1).view(h, w)
    if dedup:
        # Merge the masks corresponding to the same stuff class
        for equiv in stuff_equiv_classes.values():
            if len(equiv) > 1:
                for eq_id in equiv:
                    m_id.masked_fill_(m_id.eq(eq_id), equiv[0])
    final_h, final_w = to_tuple(target_size)
    seg_img = Image.fromarray(id2rgb(m_id.view(h, w).cpu().numpy()))
    seg_img = seg_img.resize(size=(final_w, final_h), resample=Image.NEAREST)
    np_seg_img = (
        torch.ByteTensor(torch.ByteStorage.from_buffer(seg_img.tobytes())).view(final_h, final_w, 3).numpy()
    )
    m_id = torch.from_numpy(rgb2id(np_seg_img))
    area = []
    for i in range(len(scores)):
        area.append(m_id.eq(i).sum().item())
    return area, seg_img

area, seg_img = get_ids_area(cur_masks, cur_scores, dedup=True)
if cur_classes.numel() > 0:
    # We know filter empty masks as long as we find some
    while True:
        filtered_small = torch.as_tensor(
            [area[i] <= 4 for i, c in enumerate(cur_classes)], dtype=torch.bool, device=keep.device
        )
        if filtered_small.any().item():
            cur_scores = cur_scores[~filtered_small]
            cur_classes = cur_classes[~filtered_small]
            cur_masks = cur_masks[~filtered_small]
            area, seg_img = get_ids_area(cur_masks, cur_scores)
        else:
            break
    else:
        cur_classes = torch.ones(1, dtype=torch.long, device=cur_classes.device)
segments_info = []
for i, a in enumerate(area):
    cat = cur_classes[i].item()
    segments_info.append({"id": i, "isthing": self.is_thing_map[cat], "category_id": cat, "area": a})
del cur_classes
with io.BytesIO() as out:
    seg_img.save(out, format="PNG")
    predictions = {"png_string": out.getvalue(), "segments_info": segments_info}
preds.append(predictions)
return preds

```

transformer.py

```
import copy
```

```
from typing import Optional, List
```

```
import torch
import torch.nn.functional as F
from torch import nn, Tensor
```

```
class Transformer(nn.Module):
```

```
    def __init__(self, d_model=512, nhead=8, num_encoder_layers=6,
                  num_decoder_layers=6, dim_feedforward=2048, dropout=0.1,
                  activation="relu", normalize_before=False,
                  return_intermediate_dec=False):
        super().__init__()
        encoder_layer = TransformerEncoderLayer(d_model, nhead, dim_feedforward,
                                                dropout, activation, normalize_before)
        encoder_norm = nn.LayerNorm(d_model) if normalize_before else None
        self.encoder = TransformerEncoder(encoder_layer, num_encoder_layers, encoder_norm)
        decoder_layer = TransformerDecoderLayer(d_model, nhead, dim_feedforward,
                                                dropout, activation, normalize_before)
        decoder_norm = nn.LayerNorm(d_model)
        self.decoder = TransformerDecoder(decoder_layer, num_decoder_layers, decoder_norm,
                                         return_intermediate=return_intermediate_dec)
        self._reset_parameters()
        self.d_model = d_model
        self.nhead = nhead
```

```
    def _reset_parameters(self):
        for p in self.parameters():
            if p.dim() > 1:
                nn.init.xavier_uniform_(p)
```

```
    def forward(self, src, mask, query_embed, pos_embed):
        # flatten NxCxHxW to HWxNxC
        bs, c, h, w = src.shape
        src = src.flatten(2).permute(2, 0, 1)
        pos_embed = pos_embed.flatten(2).permute(2, 0, 1)
        query_embed = query_embed.unsqueeze(1).repeat(1, bs, 1)
        mask = mask.flatten(1)
        tgt = torch.zeros_like(query_embed)
        memory = self.encoder(src, src_key_padding_mask=mask, pos=pos_embed)
        hs = self.decoder(tgt, memory, memory_key_padding_mask=mask,
                          pos=pos_embed, query_pos=query_embed)
        return hs.transpose(1, 2), memory.permute(1, 2, 0).view(bs, c, h, w)
```

```
class TransformerEncoder(nn.Module):
```

```
    def __init__(self, encoder_layer, num_layers, norm=None):
        super().__init__()
        self.layers = _get_clones(encoder_layer, num_layers)
        self.num_layers = num_layers
        self.norm = norm
```

```
    def forward(self, src,
                mask: Optional[Tensor] = None,
                src_key_padding_mask: Optional[Tensor] = None,
                pos: Optional[Tensor] = None):
        output = src
        for layer in self.layers:
            output = layer(output, src_mask=mask,
                           src_key_padding_mask=src_key_padding_mask, pos=pos)
        if self.norm is not None:
            output = self.norm(output)
```

```
    return output
```

```
class TransformerDecoder(nn.Module):
```

```
    def __init__(self, decoder_layer, num_layers, norm=None, return_intermediate=False):
        super().__init__()
        self.layers = _get_clones(decoder_layer, num_layers)
        self.num_layers = num_layers
        self.norm = norm
        self.return_intermediate = return_intermediate

    def forward(self, tgt, memory,
                tgt_mask: Optional[Tensor] = None,
                memory_mask: Optional[Tensor] = None,
                tgt_key_padding_mask: Optional[Tensor] = None,
                memory_key_padding_mask: Optional[Tensor] = None,
                pos: Optional[Tensor] = None,
                query_pos: Optional[Tensor] = None):
        output = tgt
        intermediate = []
        for layer in self.layers:
            output = layer(output, memory, tgt_mask=tgt_mask,
                           memory_mask=memory_mask,
                           tgt_key_padding_mask=tgt_key_padding_mask,
                           memory_key_padding_mask=memory_key_padding_mask,
                           pos=pos, query_pos=query_pos)
            if self.return_intermediate:
                intermediate.append(self.norm(output))
        if self.norm is not None:
            output = self.norm(output)
            if self.return_intermediate:
                intermediate.pop()
                intermediate.append(output)
        if self.return_intermediate:
            return torch.stack(intermediate)
        return output.unsqueeze(0)
```

```
class TransformerEncoderLayer(nn.Module):
```

```
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1,
                 activation="relu", normalize_before=False):
        super().__init__()
        self.self_attn = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        # Implementation of Feedforward model
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(dim_feedforward, d_model)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout1 = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout)
        self.activation = _get_activation_fn(activation)
        self.normalize_before = normalize_before

    def with_pos_embed(self, tensor, pos: Optional[Tensor]):
        return tensor if pos is None else tensor + pos

    def forward_post(self,
                    src,
                    src_mask: Optional[Tensor] = None,
                    src_key_padding_mask: Optional[Tensor] = None,
                    pos: Optional[Tensor] = None):
        q = k = self.with_pos_embed(src, pos)
```

```

src2 = self.self_attn(q, k, value=src, attn_mask=src_mask,
                      key_padding_mask=src_key_padding_mask)[0]
src = src + self.dropout1(src2)
src = self.norm1(src)
src2 = self.linear2(self.dropout(self.activation(self.linear1(src))))
src = src + self.dropout2(src2)
src = self.norm2(src)
return src

def forward_pre(self, src,
                src_mask: Optional[Tensor] = None,
                src_key_padding_mask: Optional[Tensor] = None,
                pos: Optional[Tensor] = None):
    src2 = self.norm1(src)
    q = k = self.with_pos_embed(src2, pos)
    src2 = self.self_attn(q, k, value=src2, attn_mask=src_mask,
                          key_padding_mask=src_key_padding_mask)[0]
    src = src + self.dropout1(src2)
    src2 = self.norm2(src)
    src2 = self.linear2(self.dropout(self.activation(self.linear1(src2))))
    src = src + self.dropout2(src2)
    return src

def forward(self, src,
            src_mask: Optional[Tensor] = None,
            src_key_padding_mask: Optional[Tensor] = None,
            pos: Optional[Tensor] = None):
    if self.normalize_before:
        return self.forward_pre(src, src_mask, src_key_padding_mask, pos)
    return self.forward_post(src, src_mask, src_key_padding_mask, pos)

class TransformerDecoderLayer(nn.Module):
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1,
                 activation="relu", normalize_before=False):
        super().__init__()
        self.self_attn = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        self.multihead_attn = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        # Implementation of Feedforward model
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(dim_feedforward, d_model)

        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.norm3 = nn.LayerNorm(d_model)
        self.dropout1 = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout)
        self.dropout3 = nn.Dropout(dropout)
        self.activation = _get_activation_fn(activation)
        self.normalize_before = normalize_before
    def with_pos_embed(self, tensor, pos: Optional[Tensor]):
        return tensor if pos is None else tensor + pos

    def forward_post(self, tgt, memory,
                    tgt_mask: Optional[Tensor] = None,
                    memory_mask: Optional[Tensor] = None,
                    tgt_key_padding_mask: Optional[Tensor] = None,
                    memory_key_padding_mask: Optional[Tensor] = None,
                    pos: Optional[Tensor] = None,
                    query_pos: Optional[Tensor] = None):
        q = k = self.with_pos_embed(tgt, query_pos)
        tgt2 = self.self_attn(q, k, value=tgt, attn_mask=tgt_mask,
                              key_padding_mask=tgt_key_padding_mask)[0]

```

```

tgt = tgt + self.dropout1(tgt2)
tgt = self.norm1(tgt)
tgt2 = self.multihead_attn(query=self.with_pos_embed(tgt, query_pos),
                           key=self.with_pos_embed(memory, pos),
                           value=memory, attn_mask=memory_mask,
                           key_padding_mask=memory_key_padding_mask)[0]
tgt = tgt + self.dropout2(tgt2)
tgt = self.norm2(tgt)
tgt2 = self.linear2(self.dropout(self.activation(self.linear1(tgt))))
tgt = tgt + self.dropout3(tgt2)
tgt = self.norm3(tgt)
return tgt

def forward_pre(self, tgt, memory,
                tgt_mask: Optional[Tensor] = None,
                memory_mask: Optional[Tensor] = None,
                tgt_key_padding_mask: Optional[Tensor] = None,
                memory_key_padding_mask: Optional[Tensor] = None,
                pos: Optional[Tensor] = None,
                query_pos: Optional[Tensor] = None):
    tgt2 = self.norm1(tgt)
    q = k = self.with_pos_embed(tgt2, query_pos)
    tgt2 = self.self_attn(q, k, value=tgt2, attn_mask=tgt_mask,
                        key_padding_mask=tgt_key_padding_mask)[0]
    tgt = tgt + self.dropout1(tgt2)
    tgt2 = self.norm2(tgt)
    tgt2 = self.multihead_attn(query=self.with_pos_embed(tgt2, query_pos),
                              key=self.with_pos_embed(memory, pos),
                              value=memory, attn_mask=memory_mask,
                              key_padding_mask=memory_key_padding_mask)[0]
    tgt = tgt + self.dropout2(tgt2)
    tgt2 = self.norm3(tgt)
    tgt2 = self.linear2(self.dropout(self.activation(self.linear1(tgt2))))
    tgt = tgt + self.dropout3(tgt2)
    return tgt

def forward(self, tgt, memory,
            tgt_mask: Optional[Tensor] = None,
            memory_mask: Optional[Tensor] = None,
            tgt_key_padding_mask: Optional[Tensor] = None,
            memory_key_padding_mask: Optional[Tensor] = None,
            pos: Optional[Tensor] = None,
            query_pos: Optional[Tensor] = None):
    if self.normalize_before:
        return self.forward_pre(tgt, memory, tgt_mask, memory_mask,
                                tgt_key_padding_mask, memory_key_padding_mask, pos, query_pos)
    return self.forward_post(tgt, memory, tgt_mask, memory_mask,
                             tgt_key_padding_mask, memory_key_padding_mask, pos, query_pos)

def _get_clones(module, N):
    return nn.ModuleList([copy.deepcopy(module) for i in range(N)])

def build_transformer(args):
    return Transformer(
        d_model=args.hidden_dim,
        dropout=args.dropout,
        nhead=args.nheads,
        dim_feedforward=args.dim_feedforward,
        num_encoder_layers=args.enc_layers,
        num_decoder_layers=args.dec_layers,
        normalize_before=args.pre_norm,
        return_intermediate_dec=True,
    )

```

```
def _get_activation_fn(activation):
    """Return an activation function given a string"""
    if activation == "relu":
        return F.relu
    if activation == "gelu":
        return F.gelu
    if activation == "glu":
        return F.glu
    raise RuntimeError(F"activation should be relu/gelu, not {activation}.")
```

misc.py

```
import os
import subprocess
import time
from collections import defaultdict, deque
import datetime
import pickle
from packaging import version
from typing import Optional, List
import torch
import torch.distributed as dist
from torch import Tensor
import torchvision
from torchvision.ops import _new_empty_tensor
from torchvision.ops.misc import _output_size

class SmoothedValue(object):
    """Track a series of values and provide access to smoothed values over a
    window or the global series average.
    """

    def __init__(self, window_size=20, fmt=None):
        if fmt is None:
            fmt = "{median:.4f} ( {global_avg:.4f})"
        self.deque = deque(maxlen=window_size)
        self.total = 0.0
        self.count = 0
        self.fmt = fmt

    def update(self, value, n=1):
        self.deque.append(value)
        self.count += n
        self.total += value * n

    def synchronize_between_processes(self):
        """
        Warning: does not synchronize the deque!
        """
        if not is_dist_avail_and_initialized():
            return
        t = torch.tensor([self.count, self.total], dtype=torch.float64, device='cuda')
        dist.barrier()
        dist.all_reduce(t)
        t = t.tolist()
        self.count = int(t[0])
        self.total = t[1]

    @property
    def median(self):
        def median(self):
```

```

    d = torch.tensor(list(self.deque))
    return d.median().item()

@property
def avg(self):
    d = torch.tensor(list(self.deque), dtype=torch.float32)
    return d.mean().item()

@property
def global_avg(self):
    return self.total / self.count

@property
def max(self):
    return max(self.deque)

@property
def value(self):
    return self.deque[-1]

def __str__(self):
    return self.fmt.format(
        median=self.median,
        avg=self.avg,
        global_avg=self.global_avg,
        max=self.max,
        value=self.value)

def all_gather(data):

    world_size = get_world_size()
    if world_size == 1:
        return [data]
    buffer = pickle.dumps(data)
    storage = torch.ByteStorage.from_buffer(buffer)
    tensor = torch.ByteTensor(storage).to("cuda")
    local_size = torch.tensor([tensor.numel()], device="cuda")
    size_list = [torch.tensor([0], device="cuda") for _ in range(world_size)]
    dist.all_gather(size_list, local_size)
    size_list = [int(size.item()) for size in size_list]
    max_size = max(size_list)
    tensor_list = []
    for _ in size_list:
        tensor_list.append(torch.empty((max_size,), dtype=torch.uint8, device="cuda"))
    if local_size != max_size:
        padding = torch.empty(size=(max_size - local_size,), dtype=torch.uint8, device="cuda")
        tensor = torch.cat((tensor, padding), dim=0)
    dist.all_gather(tensor_list, tensor)

    data_list = []
    for size, tensor in zip(size_list, tensor_list):
        buffer = tensor.cpu().numpy().tobytes()[:size]
        data_list.append(pickle.loads(buffer))

    return data_list

def reduce_dict(input_dict, average=True):
    """
    Args:
        input_dict (dict): all the values will be reduced
        average (bool): whether to do average or sum
    Reduce the values in the dictionary from all processes so that all processes

```

have the averaged results. Returns a dict with the same fields as input_dict, after reduction.

```
"""
world_size = get_world_size()
if world_size < 2:
    return input_dict
with torch.no_grad():
    names = []
    values = []
    # sort the keys so that they are consistent across processes
    for k in sorted(input_dict.keys()):
        names.append(k)
        values.append(input_dict[k])
    values = torch.stack(values, dim=0)
    dist.all_reduce(values)
    if average:
        values /= world_size
    reduced_dict = {k: v for k, v in zip(names, values)}
return reduced_dict
```

```
class MetricLogger(object):
    def __init__(self, delimiter="\t"):
        self.meters = defaultdict(SmoothedValue)
        self.delimiter = delimiter

    def update(self, **kwargs):
        for k, v in kwargs.items():
            if isinstance(v, torch.Tensor):
                v = v.item()
            assert isinstance(v, (float, int))
            self.meters[k].update(v)

    def __getattr__(self, attr):
        if attr in self.meters:
            return self.meters[attr]
        if attr in self.__dict__:
            return self.__dict__[attr]
        raise AttributeError("{} object has no attribute '{}'".format(
            type(self).__name__, attr))

    def __str__(self):
        loss_str = []
        for name, meter in self.meters.items():
            loss_str.append(
                "{}: {}".format(name, str(meter))
            )
        return self.delimiter.join(loss_str)

    def synchronize_between_processes(self):
        for meter in self.meters.values():
            meter.synchronize_between_processes()

    def add_meter(self, name, meter):
        self.meters[name] = meter

    def log_every(self, iterable, print_freq, header=None):
        i = 0
        if not header:
            header = ""
        start_time = time.time()
        end = time.time()
        iter_time = SmoothedValue(fmt='{avg:.4f}')
```

```

data_time = SmoothedValue(fmt='{avg:.4f}')
space_fmt = ':' + str(len(str(len(iterable)))) + 'd'
if torch.cuda.is_available():
    log_msg = self.delimiter.join([
        header,
        '{0' + space_fmt + '}/{1}',
        'eta: {eta}',
        '{meters}',
        'time: {time}',
        'data: {data}',
        'max mem: {memory:.0f}'
    ])
else:
    log_msg = self.delimiter.join([
        header,
        '{0' + space_fmt + '}/{1}',
        'eta: {eta}',
        '{meters}',
        'time: {time}',
        'data: {data}'
    ])
MB = 1024.0 * 1024.0
for obj in iterable:
    data_time.update(time.time() - end)
    yield obj
    iter_time.update(time.time() - end)
    if i % print_freq == 0 or i == len(iterable) - 1:
        eta_seconds = iter_time.global_avg * (len(iterable) - i)
        eta_string = str(datetime.timedelta(seconds=int(eta_seconds)))
        if torch.cuda.is_available():
            print(log_msg.format(
                i, len(iterable), eta=eta_string,
                meters=str(self),
                time=str(iter_time), data=str(data_time),
                memory=torch.cuda.max_memory_allocated() / MB))
        else:
            print(log_msg.format(
                i, len(iterable), eta=eta_string,
                meters=str(self),
                time=str(iter_time), data=str(data_time)))
        i += 1
    end = time.time()
total_time = time.time() - start_time
total_time_str = str(datetime.timedelta(seconds=int(total_time)))
print('{} Total time: {} ({:.4f} s / it).format(
    header, total_time_str, total_time / len(iterable)))

def get_sha():
    cwd = os.path.dirname(os.path.abspath(__file__))

    def _run(command):
        return subprocess.check_output(command, cwd=cwd).decode('ascii').strip()
    sha = 'N/A'
    diff = "clean"
    branch = 'N/A'
    try:
        sha = _run(['git', 'rev-parse', 'HEAD'])
        subprocess.check_output(['git', 'diff'], cwd=cwd)
        diff = _run(['git', 'diff-index', 'HEAD'])
        diff = "has uncommitted changes" if diff else "clean"
        branch = _run(['git', 'rev-parse', '--abbrev-ref', 'HEAD'])
    except Exception:
        pass

```

```

message = f'sha: {sha}, status: {diff}, branch: {branch}'
return message

def collate_fn(batch):
    batch = list(zip(*batch))
    batch[0] = nested_tensor_from_tensor_list(batch[0])
    return tuple(batch)

def _max_by_axis(the_list):
    # type: (List[List[int]]) -> List[int]
    maxes = the_list[0]
    for sublist in the_list[1:]:
        for index, item in enumerate(sublist):
            maxes[index] = max(maxes[index], item)
    return maxes

class NestedTensor(object):
    def __init__(self, tensors, mask: Optional[Tensor]):
        self.tensors = tensors
        self.mask = mask

    def to(self, device):
        # type: (Device) -> NestedTensor # noqa
        cast_tensor = self.tensors.to(device)
        mask = self.mask
        if mask is not None:
            assert mask is not None
            cast_mask = mask.to(device)
        else:
            cast_mask = None
        return NestedTensor(cast_tensor, cast_mask)

    def decompose(self):
        return self.tensors, self.mask

    def __repr__(self):
        return str(self.tensors)

def nested_tensor_from_tensor_list(tensor_list: List[Tensor]):
    # TODO make this more general
    if tensor_list[0].ndim == 3:
        if torchvision._is_tracing():
            # nested_tensor_from_tensor_list() does not export well to ONNX
            # call _onnx_nested_tensor_from_tensor_list() instead
            return _onnx_nested_tensor_from_tensor_list(tensor_list)

        max_size = _max_by_axis([list(img.shape) for img in tensor_list])
        batch_shape = [len(tensor_list)] + max_size
        b, c, h, w = batch_shape
        dtype = tensor_list[0].dtype
        device = tensor_list[0].device
        tensor = torch.zeros(batch_shape, dtype=dtype, device=device)
        mask = torch.ones((b, h, w), dtype=torch.bool, device=device)
        for img, pad_img, m in zip(tensor_list, tensor, mask):
            pad_img[:, :img.shape[0], :img.shape[1], :img.shape[2]].copy_(img)
            m[:, :img.shape[1], :img.shape[2]] = True
        else:
            raise ValueError('not supported')
        return NestedTensor(tensor, mask)

```

@torch.jit.unused

```

def _onnx_nested_tensor_from_tensor_list(tensor_list: List[Tensor]) -> NestedTensor:
    max_size = []
    for i in range(tensor_list[0].dim()):
        max_size_i = torch.max(torch.stack([img.shape[i] for img in tensor_list]).to(torch.float32)).to(torch.int64)
        max_size.append(max_size_i)
    max_size = tuple(max_size)
    padded_imgs = []
    padded_masks = []
    for img in tensor_list:
        padding = [(s1 - s2) for s1, s2 in zip(max_size, tuple(img.shape))]
        padded_img = torch.nn.functional.pad(img, (0, padding[2], 0, padding[1], 0, padding[0]))
        padded_imgs.append(padded_img)
        m = torch.zeros_like(img[0], dtype=torch.int, device=img.device)
        padded_mask = torch.nn.functional.pad(m, (0, padding[2], 0, padding[1]), "constant", 1)
        padded_masks.append(padded_mask.to(torch.bool))
    tensor = torch.stack(padded_imgs)
    mask = torch.stack(padded_masks)
    return NestedTensor(tensor, mask=mask)

def setup_for_distributed(is_master):
    """
    This function disables printing when not in master process
    """
    import builtins as __builtin__
    builtin_print = __builtin__.print
    def print(*args, **kwargs):
        force = kwargs.pop('force', False)
        if is_master or force:
            builtin_print(*args, **kwargs)
    __builtin__.print = print

def is_dist_avail_and_initialized():
    if not dist.is_available():
        return False
    if not dist.is_initialized():
        return False
    return True

def get_world_size():
    if not is_dist_avail_and_initialized():
        return 1
    return dist.get_world_size()

def get_rank():
    if not is_dist_avail_and_initialized():
        return 0
    return dist.get_rank()

def is_main_process():
    return get_rank() == 0

def save_on_master(*args, **kwargs):
    if is_main_process():
        torch.save(*args, **kwargs)

def init_distributed_mode(args):
    if 'RANK' in os.environ and 'WORLD_SIZE' in os.environ:
        args.rank = int(os.environ["RANK"])
        args.world_size = int(os.environ["WORLD_SIZE"])
        args.gpu = int(os.environ["LOCAL_RANK"])
    elif 'SLURM_PROCID' in os.environ:
        args.rank = int(os.environ["SLURM_PROCID"])
        args.gpu = args.rank % torch.cuda.device_count()

```

```

else:
    print('Not using distributed mode')
    args.distributed = False
    return
args.distributed = True
torch.cuda.set_device(args.gpu)
args.dist_backend = 'nccl'
print('| distributed init (rank {}): {}'.format(
    args.rank, args.dist_url), flush=True)
torch.distributed.init_process_group(backend=args.dist_backend, init_method=args.dist_url,
                                   world_size=args.world_size, rank=args.rank)
torch.distributed.barrier()
setup_for_distributed(args.rank == 0)

@torch.no_grad()
def accuracy(output, target, topk=(1,)):
    """Computes the precision@k for the specified values of k"""
    if target.numel() == 0:
        return [torch.zeros([], device=output.device)]
    maxk = max(topk)
    batch_size = target.size(0)
    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    res = []
    for k in topk:
        correct_k = correct[:,k].view(-1).float().sum(0)
        res.append(correct_k.mul_(100.0 / batch_size))
    return res

def interpolate(input, size=None, scale_factor=None, mode="nearest", align_corners=None):
    # type: (Tensor, Optional[List[int]], Optional[float], str, Optional[bool]) -> Tensor
    """
    Equivalent to nn.functional.interpolate, but with support for empty batch sizes.
    This will eventually be supported natively by PyTorch, and this
    class can go away.
    """
    if version.parse(torchvision.__version__) < version.parse('0.7'):
        if input.numel() > 0:
            return torch.nn.functional.interpolate(
                input, size, scale_factor, mode, align_corners
            )
        output_shape = _output_size(2, input, size, scale_factor)
        output_shape = list(input.shape[:-2]) + list(output_shape)
        return _new_empty_tensor(input, output_shape)
    else:
        return torchvision.ops.misc.interpolate(input, size, scale_factor, mode, align_corners)

```

engine.py

```

import math
import os
import sys
from typing import Iterable

import torch

import util.misc as utils
from datasets.coco_eval import CocoEvaluator
from datasets.panoptic_eval import PanopticEvaluator
def train_one_epoch(model: torch.nn.Module, criterion: torch.nn.Module,

```

```

        data_loader: Iterable, optimizer: torch.optim.Optimizer,
        device: torch.device, epoch: int, max_norm: float = 0):
    model.train()
    criterion.train()
    metric_logger = utils.MetricLogger(delimiter=" ")
    metric_logger.add_meter('lr', utils.SmoothedValue(window_size=1, fmt='{value:.6f}'))
    metric_logger.add_meter('class_error', utils.SmoothedValue(window_size=1, fmt='{value:.2f}'))
    header = 'Epoch: [{}]'.format(epoch)
    print_freq = 10

    for samples, targets in metric_logger.log_every(data_loader, print_freq, header):
        samples = samples.to(device)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        outputs = model(samples)
        loss_dict = criterion(outputs, targets)
        weight_dict = criterion.weight_dict
        losses = sum(loss_dict[k] * weight_dict[k] for k in loss_dict.keys() if k in weight_dict)

        loss_dict_reduced = utils.reduce_dict(loss_dict)
        loss_dict_reduced_unscaled = {f'{k}_unscaled': v
                                      for k, v in loss_dict_reduced.items()}
        loss_dict_reduced_scaled = {k: v * weight_dict[k]
                                    for k, v in loss_dict_reduced.items() if k in weight_dict}
        losses_reduced_scaled = sum(loss_dict_reduced_scaled.values())

        loss_value = losses_reduced_scaled.item()

        if not math.isfinite(loss_value):
            print("Loss is {}, stopping training".format(loss_value))
            print(loss_dict_reduced)
            sys.exit(1)

        optimizer.zero_grad()
        losses.backward()
        if max_norm > 0:
            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)
        optimizer.step()

        metric_logger.update(loss=loss_value, **loss_dict_reduced_scaled, **loss_dict_reduced_unscaled)
        metric_logger.update(class_error=loss_dict_reduced['class_error'])
        metric_logger.update(lr=optimizer.param_groups[0]["lr"])
    # gather the stats from all processes
    metric_logger.synchronize_between_processes()
    print("Averaged stats:", metric_logger)
    return {k: meter.global_avg for k, meter in metric_logger.meters.items()}

@torch.no_grad()
def evaluate(model, criterion, postprocessors, data_loader, base_ds, device, output_dir):
    model.eval()
    criterion.eval()
    metric_logger = utils.MetricLogger(delimiter=" ")
    metric_logger.add_meter('class_error', utils.SmoothedValue(window_size=1, fmt='{value:.2f}'))
    header = 'Test:'
    iou_types = tuple(k for k in ('segm', 'bbox') if k in postprocessors.keys())
    coco_evaluator = CocoEvaluator(base_ds, iou_types)
    # coco_evaluator.coco_eval[iou_types[0]].params.iouThrs = [0, 0.1, 0.5, 0.75]

    panoptic_evaluator = None
    if 'panoptic' in postprocessors.keys():
        panoptic_evaluator = PanopticEvaluator(
            data_loader.dataset.ann_file,
            data_loader.dataset.ann_folder,

```

```

        output_dir=os.path.join(output_dir, "panoptic_eval"),
    )
for samples, targets in metric_logger.log_every(data_loader, 10, header):
    samples = samples.to(device)
    targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
    outputs = model(samples)
    loss_dict = criterion(outputs, targets)
    weight_dict = criterion.weight_dict
    # reduce losses over all GPUs for logging purposes
    loss_dict_reduced = utils.reduce_dict(loss_dict)
    loss_dict_reduced_scaled = {k: v * weight_dict[k]
                                for k, v in loss_dict_reduced.items() if k in weight_dict}
    loss_dict_reduced_unscaled = {f'{k}_unscaled': v
                                   for k, v in loss_dict_reduced.items()}
    metric_logger.update(loss=sum(loss_dict_reduced_scaled.values()),
                         **loss_dict_reduced_scaled,
                         **loss_dict_reduced_unscaled)
    metric_logger.update(class_error=loss_dict_reduced['class_error'])
    orig_target_sizes = torch.stack([t["orig_size"] for t in targets], dim=0)
    results = postprocessors['bbox'](outputs, orig_target_sizes)
    if 'segm' in postprocessors.keys():
        target_sizes = torch.stack([t["size"] for t in targets], dim=0)
        results = postprocessors['segm'](results, outputs, orig_target_sizes, target_sizes)
    res = {target['image_id'].item(): output for target, output in zip(targets, results)}
    if coco_evaluator is not None:
        coco_evaluator.update(res)
    if panoptic_evaluator is not None:
        res_pano = postprocessors["panoptic"](outputs, target_sizes, orig_target_sizes)
        for i, target in enumerate(targets):
            image_id = target["image_id"].item()
            file_name = f"{image_id:012d}.png"
            res_pano[i]["image_id"] = image_id
            res_pano[i]["file_name"] = file_name
        panoptic_evaluator.update(res_pano)
metric_logger.synchronize_between_processes()
print("Averaged stats:", metric_logger)
if coco_evaluator is not None:
    coco_evaluator.synchronize_between_processes()
if panoptic_evaluator is not None:
    panoptic_evaluator.synchronize_between_processes()
if coco_evaluator is not None:
    coco_evaluator.accumulate()
    coco_evaluator.summarize()
panoptic_res = None
if panoptic_evaluator is not None:
    panoptic_res = panoptic_evaluator.summarize()
stats = {k: meter.global_avg for k, meter in metric_logger.meters.items()}
if coco_evaluator is not None:
    if 'bbox' in postprocessors.keys():
        stats['coco_eval_bbox'] = coco_evaluator.coco_eval['bbox'].stats.tolist()
    if 'segm' in postprocessors.keys():
        stats['coco_eval_masks'] = coco_evaluator.coco_eval['segm'].stats.tolist()
if panoptic_res is not None:
    stats['PQ_all'] = panoptic_res["All"]
    stats['PQ_th'] = panoptic_res["Things"]
    stats['PQ_st'] = panoptic_res["Stuff"]
return stats, coco_evaluator

```

main.py

```
import argparse
```

```

import datetime
import json
import random
import time
from pathlib import Path

import numpy as np
import torch
from torch.utils.data import DataLoader, DistributedSampler

import datasets
import util.misc as utils
from datasets import build_dataset, get_coco_api_from_dataset
from engine import evaluate, train_one_epoch
from models import build_model

def get_args_parser():
    parser = argparse.ArgumentParser('Set transformer detector', add_help=False)
    parser.add_argument('--lr', default=1e-4, type=float)
    parser.add_argument('--lr_backbone', default=1e-5, type=float)
    parser.add_argument('--batch_size', default=2, type=int)
    parser.add_argument('--weight_decay', default=1e-4, type=float)
    parser.add_argument('--epochs', default=20, type=int) # 300
    parser.add_argument('--lr_drop', default=200, type=int)
    parser.add_argument('--clip_max_norm', default=0.1, type=float,
                        help='gradient clipping max norm')

    # Model parameters
    parser.add_argument('--frozen_weights', type=str, default=None,
                        help="Path to the pretrained model. If set, only the mask head will be trained")
    # * Backbone
    parser.add_argument('--backbone', default='resnet50', type=str,
                        help="Name of the convolutional backbone to use")
    parser.add_argument('--dilation', action='store_true',
                        help="If true, we replace stride with dilation in the last convolutional block (DC5)")
    parser.add_argument('--position_embedding', default='sine', type=str, choices=('sine', 'learned'),
                        help="Type of positional embedding to use on top of the image features")

    # * Transformer
    parser.add_argument('--enc_layers', default=6, type=int,
                        help="Number of encoding layers in the transformer")
    parser.add_argument('--dec_layers', default=6, type=int,
                        help="Number of decoding layers in the transformer")
    parser.add_argument('--dim_feedforward', default=2048, type=int,
                        help="Intermediate size of the feedforward layers in the transformer blocks")
    parser.add_argument('--hidden_dim', default=256, type=int,
                        help="Size of the embeddings (dimension of the transformer)")
    parser.add_argument('--dropout', default=0.1, type=float,
                        help="Dropout applied in the transformer")
    parser.add_argument('--nheads', default=8, type=int,
                        help="Number of attention heads inside the transformer's attentions")
    parser.add_argument('--num_queries', default=100, type=int,
                        help="Number of query slots")
    parser.add_argument('--pre_norm', action='store_true')

    # * Segmentation
    parser.add_argument('--masks', action='store_true',
                        help="Train segmentation head if the flag is provided")

    # Loss
    parser.add_argument('--no_aux_loss', dest='aux_loss', action='store_false',
                        help="Disables auxiliary decoding losses (loss at each layer)")

```

```

# * Matcher
parser.add_argument('--set_cost_class', default=1, type=float,
                    help="Class coefficient in the matching cost")
parser.add_argument('--set_cost_bbox', default=5, type=float,
                    help="L1 box coefficient in the matching cost")
parser.add_argument('--set_cost_giou', default=2, type=float,
                    help="giou box coefficient in the matching cost")
# * Loss coefficients
parser.add_argument('--mask_loss_coef', default=1, type=float)
parser.add_argument('--dice_loss_coef', default=1, type=float)
parser.add_argument('--bbox_loss_coef', default=5, type=float)
parser.add_argument('--giou_loss_coef', default=2, type=float)
parser.add_argument('--eos_coef', default=0.1, type=float,
                    help="Relative classification weight of the no-object class")

# dataset parameters
parser.add_argument('--dataset_file', default='coco')
parser.add_argument('--data_path', type=str)
parser.add_argument('--coco_panoptic_path', type=str)
parser.add_argument('--remove_difficult', action='store_true')

parser.add_argument('--output_dir', default="",
                    help='path where to save, empty for no saving')
parser.add_argument('--device', default='cuda',
                    help='device to use for training / testing')
parser.add_argument('--seed', default=42, type=int)
parser.add_argument('--resume', default="", help='resume from checkpoint')
parser.add_argument('--start_epoch', default=0, type=int, metavar='N',
                    help='start epoch')
parser.add_argument('--eval', action='store_true')
parser.add_argument('--num_workers', default=2, type=int)

# distributed training parameters
parser.add_argument('--world_size', default=1, type=int,
                    help='number of distributed processes')
parser.add_argument('--dist_url', default='env://', help='url used to set up distributed training')
return parser

def main(args):
    utils.init_distributed_mode(args)
    print("git:\n {} \n".format(utils.get_sha()))

    if args.frozen_weights is not None:
        assert args.masks, "Frozen training is meant for segmentation only"
    print(args)

    device = torch.device(args.device)

    # fix the seed for reproducibility
    seed = args.seed + utils.get_rank()
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)

    model, criterion, postprocessors = build_model(args)
    model.to(device)

    model_without_ddp = model
    if args.distributed:
        model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
        model_without_ddp = model.module
    n_parameters = sum(p.numel() for p in model.parameters() if p.requires_grad)

```

```

print('number of params:', n_parameters)

param_dicts = [
    {"params": [p for n, p in model_without_ddp.named_parameters() if "backbone" not in n and p.requires_grad]},
    {
        "params": [p for n, p in model_without_ddp.named_parameters() if "backbone" in n and p.requires_grad],
        "lr": args.lr_backbone,
    },
]
optimizer = torch.optim.AdamW(param_dicts, lr=args.lr,
                               weight_decay=args.weight_decay)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, args.lr_drop)

dataset_train = build_dataset(image_set='train', args=args)
dataset_val = build_dataset(image_set='val', args=args)

if args.distributed:
    sampler_train = DistributedSampler(dataset_train)
    sampler_val = DistributedSampler(dataset_val, shuffle=False)
else:
    sampler_train = torch.utils.data.RandomSampler(dataset_train)
    sampler_val = torch.utils.data.SequentialSampler(dataset_val)

batch_sampler_train = torch.utils.data.BatchSampler(
    sampler_train, args.batch_size, drop_last=True)

data_loader_train = DataLoader(dataset_train, batch_sampler=batch_sampler_train,
                               collate_fn=utils.collate_fn, num_workers=args.num_workers)
data_loader_val = DataLoader(dataset_val, args.batch_size, sampler=sampler_val,
                             drop_last=False, collate_fn=utils.collate_fn, num_workers=args.num_workers)

if args.dataset_file == "coco_panoptic":
    # We also evaluate AP during panoptic training, on original coco DS
    coco_val = datasets.coco.build("val", args)
    base_ds = get_coco_api_from_dataset(coco_val)
else:
    base_ds = get_coco_api_from_dataset(dataset_val)

if args.frozen_weights is not None:
    checkpoint = torch.load(args.frozen_weights, map_location='cpu')
    model_without_ddp.detr.load_state_dict(checkpoint['model'])

output_dir = Path(args.output_dir)
if args.resume:
    if args.resume.startswith('https'):
        checkpoint = torch.hub.load_state_dict_from_url(
            args.resume, map_location='cpu', check_hash=True)
    else:
        checkpoint = torch.load(args.resume, map_location='cpu')

    del checkpoint["model"]["class_embed.weight"]
    del checkpoint["model"]["class_embed.bias"]
    #del checkpoint["model"]["query_embed.weight"]

    model_without_ddp.load_state_dict(checkpoint['model'], strict=False)
    if not args.eval and 'optimizer' in checkpoint and 'lr_scheduler' in checkpoint and 'epoch' in checkpoint:
        optimizer.load_state_dict(checkpoint['optimizer'])
        lr_scheduler.load_state_dict(checkpoint['lr_scheduler'])
        args.start_epoch = checkpoint['epoch'] + 1

if args.eval:
    test_stats, coco_evaluator = evaluate(model, criterion, postprocessors,
                                          data_loader_val, base_ds, device, args.output_dir)

```

```

if args.output_dir:
    utils.save_on_master(coco_evaluator.coco_eval["bbox"].eval, output_dir / "eval.pth")
return

print("Start training")
start_time = time.time()
for epoch in range(args.start_epoch, args.epochs):
    if args.distributed:
        sampler_train.set_epoch(epoch)
    train_stats = train_one_epoch(
        model, criterion, data_loader_train, optimizer, device, epoch,
        args.clip_max_norm)
    lr_scheduler.step()
    if args.output_dir:
        checkpoint_paths = [output_dir / 'checkpoint.pth']
        # extra checkpoint before LR drop and every 100 epochs
        if (epoch + 1) % args.lr_drop == 0 or (epoch + 1) % 100 == 0:
            checkpoint_paths.append(output_dir / f'checkpoint{epoch:04}.pth')
        for checkpoint_path in checkpoint_paths:
            utils.save_on_master({
                'model': model_without_ddp.state_dict(),
                'optimizer': optimizer.state_dict(),
                'lr_scheduler': lr_scheduler.state_dict(),
                'epoch': epoch,
                'args': args,
            }, checkpoint_path)

    test_stats, coco_evaluator = evaluate(
        model, criterion, postprocessors, data_loader_val, base_ds, device, args.output_dir
    )

    log_stats = {**{f'train_{k}': v for k, v in train_stats.items()},
                  **{f'test_{k}': v for k, v in test_stats.items()},
                  'epoch': epoch,
                  'n_parameters': n_parameters}

    if args.output_dir and utils.is_main_process():
        with (output_dir / "log.txt").open("a") as f:
            f.write(json.dumps(log_stats) + "\n")

    # for evaluation logs
    if coco_evaluator is not None:
        (output_dir / 'eval').mkdir(exist_ok=True)
        if "bbox" in coco_evaluator.coco_eval:
            filenames = ['latest.pth']
            if epoch % 50 == 0:
                filenames.append(f'{epoch:03}.pth')
            for name in filenames:
                torch.save(coco_evaluator.coco_eval["bbox"].eval,
                           output_dir / "eval" / name)

    total_time = time.time() - start_time
    total_time_str = str(datetime.timedelta(seconds=int(total_time)))
    print('Training time {}'.format(total_time_str))

if __name__ == '__main__':
    print("\n NEW VERSION DETR v.2025_05_03 \n")
    parser = argparse.ArgumentParser('DETR training and evaluation script', parents=[get_args_parser()])
    args = parser.parse_args()
    if args.output_dir:
        Path(args.output_dir).mkdir(parents=True, exist_ok=True)
    main(args)

```

build_plot_and_metrics.py

```

import os
import json
import torch
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
from torchvision.ops import box_iou

from models import build_model
from datasets.blood import make_Blood_transforms

def box_cxcywh_to_xyxy(x):
    x_c, y_c, w, h = x.unbind(1)
    b = [(x_c - 0.5 * w), (y_c - 0.5 * h), (x_c + 0.5 * w), (y_c + 0.5 * h)]
    return torch.stack(b, dim=1)

def rescale_bboxes(out_bbox, size):
    img_w, img_h = size
    b = box_cxcywh_to_xyxy(out_bbox)
    b = b * torch.tensor([img_w, img_h, img_w, img_h], dtype=torch.float32)
    return b

def calculate_image_metrics_per_class(targets, predictions, iou_threshold=0.4):
    # Метрики для класів 0 (non-infected) та 1 (infected)
    metrics_per_class = {
        0: {'tp': 0, 'fp': 0, 'fn': 0, 'total': 0, 'iou': 0},
        1: {'tp': 0, 'fp': 0, 'fn': 0, 'total': 0, 'iou': 0},
    }
    for cls in [0, 1]:
        gt_boxes = torch.stack([t['xyxy'] for t in targets if t['class_id'] == cls]) if any(t['class_id'] == cls for t in targets)
        else torch.empty((0, 4))
        pred_boxes = torch.stack([p['xyxy'] for p in predictions if p['class_id'] == cls]) if any(p['class_id'] == cls for p in predictions)
        else torch.empty((0, 4))

        metrics_per_class[cls]['total'] = gt_boxes.size(0)

        matched_gt = set()

        if pred_boxes.size(0) > 0 and gt_boxes.size(0) > 0:
            ious = box_iou(pred_boxes, gt_boxes)

            for j, iou_row in enumerate(ious):
                max_iou, gt_idx = iou_row.max(0)
                if max_iou > iou_threshold and gt_idx.item() not in matched_gt:
                    matched_gt.add(gt_idx.item())
                    metrics_per_class[cls]['tp'] += 1
                    metrics_per_class[cls]['iou'] += max_iou.item()

            for idx in range(len(pred_boxes)):
                if all(ious[idx, gt_i] <= iou_threshold for gt_i in range(len(gt_boxes))):
                    metrics_per_class[cls]['fp'] += 1

            # FN = GT boxes, які не були співставлені
            for idx in range(len(gt_boxes)):
                if idx not in matched_gt:
                    metrics_per_class[cls]['fn'] += 1
        else:
            metrics_per_class[cls]['fn'] += gt_boxes.size(0)

```

```

        metrics_per_class[cls]['fp'] += pred_boxes.size(0)
results = {}
for cls in [0, 1]:
    tp = metrics_per_class[cls]['tp']
    fp = metrics_per_class[cls]['fp']
    fn = metrics_per_class[cls]['fn']
    total = metrics_per_class[cls]['total']
    avg_iou = metrics_per_class[cls]['iou'] / tp if tp > 0 else 0

    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    accuracy = tp / total if total > 0 else 0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
    results[cls] = {
        'tp': tp, 'fp': fp, 'fn': fn,
        'precision': precision,
        'recall': recall,
        'accuracy': accuracy,
        'avg_iou': avg_iou,
        'f1': f1
    }
return results

@torch.no_grad()
def run_inference_and_show_gt(model_path, image_dir, coco_path, threshold=0.5, max_images=5,
show_scores=True):
    from argparse import Namespace
    torch.serialization.add_safe_globals([Namespace])

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    args = Namespace(
        masks=False, backbone='resnet50', dilation=False, position_embedding='sine',
        hidden_dim=256, dim_feedforward=2048, enc_layers=6, dec_layers=6,
        dropout=0.1, nheads=8, num_queries=100, pre_norm=False, aux_loss=True,
        dataset_file='blood', device=device.type, lr_backbone=1e-5,
        return_intern_layers=False, train_backbone=True, num_classes=2,
        set_cost_class=1, set_cost_bbox=5, set_cost_giou=2,
        bbox_loss_coef=5, giou_loss_coef=2, eos_coef=0.1,
        mask_loss_coef=1, dice_loss_coef=1
    )
    model, _, postprocessors = build_model(args)
    checkpoint = torch.load(model_path, map_location=device)
    model.load_state_dict(checkpoint['model'], strict=True)
    model.to(device).eval()

    with open(coco_path, 'r') as f:
        coco_data = json.load(f)
    ann_map = {}
    for ann in coco_data['annotations']:
        ann_map.setdefault(ann['image_id'], []).append(ann)

    cat_id_to_name = {cat['id']: cat['name'] for cat in coco_data['categories']}
    class_colors = {0: 'green', 1: 'red'}
    transform = make_Blood_transforms("val")
    metrics_list = []
    for img_info in coco_data['images'][:max_images]:
        img_id = img_info['id']
        file_name = img_info['file_name']
        image_path = os.path.join(image_dir, file_name)
        orig_image = Image.open(image_path).convert("RGB")
        w, h = orig_image.size
        dummy_target = {"size": torch.tensor([h, w]), "orig_size": torch.tensor([h, w])}
        image_tensor, _ = transform(orig_image, dummy_target)

```

```

image_tensor = image_tensor.unsqueeze(0).to(device)
outputs = model(image_tensor)
probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
keep = probas.max(-1).values > threshold
scores = probas.max(-1).values[keep]
classes = probas[keep].argmax(-1)
boxes = rescale_bboxes(outputs['pred_boxes'][0, keep].cpu(), orig_image.size)
preds = []
targs = []
for ann in ann_map.get(img_id, []):
    x, y, w_box, h_box = ann['bbox']
    cat_id = ann['category_id']
    targs.append({
        "xyxy": torch.tensor([x, y, x + w_box, y + h_box], dtype=torch.float32),
        "class_id": cat_id
    })
for i, box in enumerate(boxes):
    preds.append({
        "xyxy": box,
        "class_id": classes[i].item(),
        "score": scores[i].item()
    })
image_metrics = calculate_image_metrics_per_class(targs, preds)
metrics_list.append(image_metrics)
# === Plot ===
fig, axes = plt.subplots(1, 2, figsize=(16, 8))
axes[0].imshow(orig_image)
axes[1].imshow(orig_image)
axes[0].set_title("Ground Truth")
axes[1].set_title("Predicted")
for ann in targs:
    x0, y0, x1, y1 = ann['xyxy']
    cls = ann['class_id']
    color = class_colors.get(cls, 'blue')
    rect = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2, edgecolor=color, facecolor='none')
    axes[0].add_patch(rect)
    axes[0].text(x0, y0 - 5, cat_id_to_name[cls], color=color, fontsize=10,
        bbox=dict(facecolor='white', alpha=0.7))

for pred in preds:
    x0, y0, x1, y1 = pred['xyxy']
    cls = pred['class_id']
    score = pred['score']
    color = class_colors.get(cls, 'blue')
    rect = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2, edgecolor=color, facecolor='none')
    axes[1].add_patch(rect)
    label_text = cat_id_to_name[cls]
    if show_scores:
        label_text += f" ({score:.2f})"
    axes[1].text(x0, y0 - 5, label_text, color=color, fontsize=10,
        bbox=dict(facecolor='white', alpha=0.7))
for ax in axes:
    ax.axis("off")
plt.suptitle(file_name)
# Вивід метрик для кожного класу внизу
metrics_text = []
for cls in [0, 1]:
    cls_name = 'non-infected' if cls == 0 else 'infected'
    m = image_metrics[cls]
    metrics_text.append(
        f"{cls_name}: IoU={m['avg_iou']:.3f}, Prec={m['precision']:.3f}, Rec={m['recall']:.3f},
        Acc={m['accuracy']:.3f}, F1={m['f1']:.3f}"
    )

```

```

plt.figtext(0.5, 0.01, " | ".join(metrics_text), ha="center", fontsize=12)
plt.tight_layout()
plt.show()
if metrics_list:
    avg_metrics = {0: {}, 1: {}}
    for cls in [0, 1]:
        for key in metrics_list[0][cls]:
            avg_metrics[cls][key] = sum(m[cls][key] for m in metrics_list) / len(metrics_list)

    print("\n=== AVERAGE METRICS PER CLASS ===")
    for cls in [0, 1]:
        cls_name = 'non-infected' if cls == 0 else 'infected'
        print(f"\nClass '{cls_name}' ({cls}):")
        for k, v in avg_metrics[cls].items():
            print(f" {k.capitalize():<10}: {v:.4f}")

!python3 main.py --dataset_file blood --data_path /content/drive/MyDrive/DETR_v3/detr_tutorial/dataset/ --
output_dir output

run_inference_and_show_gt(
    model_path= model_path,
    image_dir= image_dir,
    coco_path=coco_path
    threshold=0.5,
    max_images=None,
    show_scores=True
)

```