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

Факультет електроніки

Кафедра мікроелектроніки

До захисту допущено:

В.о. завідувача кафедри

_____ Дмитро ТАТАРЧУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Мікро- та наноелектроніка»

спеціальності 153 «Мікро- та наносистемна техніка»

на тему: «Інтелектуальна система дистанційного розпізнавання об'єктів»

Виконав (-ла):
студент (-ка) IV курсу, групи ДП-91
Василенко Михайло Євгенович _____

Керівник: проф. каф. МЕ, д.ф.-м.н.,
Королюк Дмитро Володимирович _____

Консультант з нормоконтролю: ст.викл.каф.МЕ, к.т.н.,
Королевич Любомир Миколайович _____

Консультант з інформаційних питань: доц.каф.МЕ, к.т.н., доц.,
Діденко Юрій Вікторович _____

Рецензент: проф. каф. ЕПС, д.т.н.,
Мельник Ігор Віталійович _____

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

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет електроніки
Кафедра мікроелектроніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 153 «Мікро- та наносистемна техніка»

Освітньо-професійна програма «Мікро- та наноелектроніка»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Дмитро ТАТАРЧУК
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Василенку Михайлу Євгеновичу

1. Тема роботи «Інтелектуальна система дистанційного розпізнавання об’єктів», керівник роботи Королюк Дмитро Володимирович, проф. каф. МЕ, д.ф.-м.н.,проф., затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи 08.06.23

3. Вихідні дані до роботи

Система розпізнавання об’єктів з повітря

4. Зміст роботи

1. Вивчення теоретичних основ машинного навчання. 2. аналіз методів та алгоритмів розпізнавання об'єктів 3. пошук даних для навчання моделі. 4. навчання моделі ЗНМ для розпізнавання об’єктів з повітря. 5. тестування моделі та оптимізація гіперпараметрів 6. оцінка якості розробленої моделі.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Презентація, графіки активаційних функцій, ілюстрації досліджених архітектур ЗНМ, ілюстрації ключових принципів методів, ілюстрація способів анотації для ОВВ рамок, ілюстрації роботи в CVAT і google colab, графіки метрик моделі в процесі навчання, приклади виявлення об'єктів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 17.04.23

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Відмітка про виконання
1	Вивчення теоретичних основ машинного навчання	17.04.2023 –30.04.2023	
2	Аналіз методів та алгоритмів розпізнавання об'єктів	24.04.2023 –7.05.2023	
3	Пошук даних для навчання моделі	1.05.2023 – 7.05.2023	
4	Навчання моделі ЗНМ для розпізнавання об’єктів з повітря	8.05.2023 – 21.05.2023	
5	Тестування моделі та оптимізація гіперпараметрів	21.05.2023 – 28.05.2023	
6	Оцінка якості розробленої моделі	28.05.2023 – 01.06.2023	

Студент Михайло, ВАСИЛЕНКО

Керівник Дмитро, КОРОЛЮК

Реферат

Дипломна робота присвячена дослідженню інтелектуальних систем дистанційного розпізнавання об’єктів

Дипломна робота складається зі вступу, 5 розділів, висновків, списку використаних джерел, що складається з 16 джерел і 2 додатків. Роботу викладено на 73 аркушах друкованого тексту, містить 23 формули, 36 рисунків та 1 таблицю.

Метою даної дипломної роботи є дослідження модельних підходів візуального розпізнання об’єктів на основі згорткових нейромереж, для розробки швидкої моделі розпізнання об’єктів на аерознімках зі складними сценаріями

Під час розроблення моделі було використано такі методи машинного навчання: Трансферне навчання, аугментація даних, CSL[12], OneCycleLR [14], CГС, adam[13]. Так само було використано архітектуру YOLOv5 і набір даних VSAI[10].

За результатом роботи отримана модель, яка має посередні показники точності та повноти, і поки що не може використовуватися в практичних завданнях. Рекомендується використовувати отриману модель для продовження дослідження в цій галузі, використовувати як переднавчену модель для донавчання на новому наборі даних для багаторакурсного розпізнавання.

Ключові слова: ЗНМ, візуальне розпізнавання, аерозйомка, аеророзвідка.

Abstract.

This dissertation is devoted to the study of intelligent systems for remote object recognition.

The dissertation consists of an introduction, 5 chapters, conclusions, a list of references consisting of 16 sources and 2 appendices. The work is presented on 73 pages of printed text, contains 23 formulas, 36 figures and 1 table.

The purpose of this thesis is to study model-based approaches to visual object recognition based on convolutional neural networks to develop a fast model for recognizing objects in aerial images with complex scenarios.

The following machine learning methods were used to develop the model: Transfer learning, data augmentation, CSL [12], OneCycleLR [14], SGD, adam [13]. We also used the YOLOv5 architecture and the VSAI dataset [10].

As a result, we obtained a model that has mediocre accuracy and completeness, and cannot yet be used in practical tasks. It is recommended to use the obtained model to continue research in this area, to use it as a pre-trained model for retraining on a new dataset for multi-view recognition.

Keywords: CNN, visual recognition, aerial survey, aerial reconnaissance.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ЗНМ – згорткова нейронна мережа

VGG16 - глибока згорткова нейронна мережа, яка використовується для класифікації зображень.

Darknet – глибока ЗНМ, для класифікації зображень

Darknet-19, darknet-51, CSPDarknet53 – варіанти мережі Darknet

RPN – мережа, для генерації регіонів пропозицій

СГС (стохастичний градієнтний спуск) – метод оптимізації градієнтного спуску за допомогою стохастичного наближення

Adam (Adaptive moment estimation) – адаптивный метод оптимізації на основі оцінки першого і другого моментів градієнтів

Momentum, імпульс – метод, який допомагає прискорити СГС у потрібному напрямку та пом'якшує коливання

Weight decay, затухання ваг – регуляризаційна техніка, що використовується в навчанні нейронних мереж для запобігання перенавчання

OneCycleLR - стратегія управління швидкістю навчання

Софтмакс (від англ. softmax) – нормована експоненційна функція

ReLU – активаційна функція, яка виводить максимум від нуля та вхідного значення

Leaky ReLU - модифікація ReLU, допускає малі негативні значення

Кросс-ентропія – метрика різниці між двома ймовірнісними розподілами

Focal Loss – фокусна функція втрат

IoU (Intersection over Union) – перетин через об'єднання

mAP (Mean Average Precision) – середня точність за всіма класами

RoI – регіон інтересу на зображенні, який містить важливу інформацію

IoU (Intersection over Union) – перетин через об'єднання

Подавлення немаксимальних значень – метод для відбору найкращих обмежувальних рамок у процесі виявлення об'єктів

OBB (Oriented Bounding Boxes) – орієнтовані обмежувальні рамки

AABB (Axis-aligned bounding boxes) – осьові обмежувальні рамки

CSL (Circular smooth label) – метод знаходження кута повороту OBB

R-CNN – метод виявлення об'єктів, який використовує згорткові нейронні мережі для класифікації об'єктів у регіонах.

Fast R-CNN - удосконалена версія R-CNN, яка виконує виявлення об'єктів та класифікацію в одному проході.

Faster R-CNN - розвиток Fast R-CNN, з використанням RPN

R-FCN – варіант Faster R-CNN, який використовує повністю згорткові мережі.

YOLO – одностадійний метод виявлення об'єктів, який виконує класифікацію та локалізацію об'єктів одночасно

SSD – одностадійний детектор, який використовує кілька карт ознак різного масштабу для виявлення об'єктів різного розміру

RetinaNet – одностадійний детектор, який використовує Focal Loss для рішення проблеми дисбалансу між фоновими та передніми класами

PANet – структура мережі, що агрегує контекстуальну інформацію для виявлення об'єктів.

SAM – механізм просторової уваги, що підсилює витягування ознак в мережах виявлення об'єктів

VSAI – набір даних для виявлення об'єктів з повітря

DOTA – великий набір даних для виявлення об'єктів на аерознімках

CVAT – інструмент для анотації відео та зображень

Docker – платформа для розробки та запуску програм у контейнерах

WSL – сумісний шар, для запуску Linux-додатків на Windows.

Ubuntu – це відкрита операційна система, заснована на Linux.

Google Colab - хмарне середовище для запуску коду

Вступ

Безпілотні літальні апарати (БПЛА) дедалі більше впроваджуються в процеси людської життєдіяльності, що підвищує затребуваність в автоматизації виявлення об'єктів з безпілотників. Проте, більшість таких систем комп'ютерного зору розраховані тільки на надірну зйомку, а прогрес у сфері довільно-орієнтованої зйомки йде повільно.

Дослідження в цій галузі, у перспективі, можуть дати можливість створити більш універсальну систему розпізнавання об'єктів з повітря, з різних ракурсів. Така система буде корисною в будь-яких галузях, пов'язаних з аеророзвідкою, наприклад, у сфері безпеки, моніторингу довкілля, дослідженні місцевості, пошуку і порятунку тощо.

Така система в перспективі має бути здатна працювати в реальному часі, для виконання завдань швидкого реагування, тому дослідження було зосереджено на швидкодіючих методах виявлення об'єктів.

Метою цієї роботи є дослідження методів і алгоритмів розпізнавання об'єктів на аерофотознімках за допомогою згорткових нейронних мереж. Під час роботи планується навчити нейромережу виявляти об'єкти на довільно-орієнтованих аерознімках, а також, провести аналіз ефективності запропонованої системи.

1. Теоретичні основи алгоритмів розпізнавання об'єктів

1.1 Згорткові нейронні мережі

Згорткові нейронні мережі (ЗНМ) - це клас глибоких нейронних мереж, спеціально розроблений для аналізу візуальних даних, таких як зображення і відео. ЗНМ володіє унікальною здатністю автоматичного вилучення ієрархічних ознак з даних, що робить їх особливо придатними для завдань комп'ютерного зору, таких як класифікація зображень, сегментація і виявлення об'єктів. Згорткові нейронні мережі складаються з декількох шарів, які обробляють вхідні дані послідовно.

Узагальнено, роботу ЗНМ можна описати таким прикладом: спочатку дані входять у згортковий шар, де фільтри визначають локальні особливості на зображенні, створюючи карти ознак, після чого застосовується функція активації, така як ReLU [2], для внесення нелінійності; потім виконується пулінг, який зменшує розмірність карт ознак, зберігаючи при цьому найважливіші особливості; цей процес повторюється аж до останнього згорткового шару, після чого карти ознак подаються на повнозв'язний шар, який вибудовує інформацію про ознаки в глобальний контекст; на завершення, функція активації, така як софтмакс, застосовується до виходу повнозв'язного шару для передбачення ймовірностей приналежності до різних класів. У результаті, згорткова нейронна мережа послідовно обробляє й аналізує інформацію на різних рівнях абстракції, починаючи з локальних особливостей і закінчуючи глобальним контекстом, для виконання завдання класифікації або регресії.

1.2 Активаційні функції

Активаційні функції, це важливі складові нейронних мереж. Вони вводять нелінійність у модели, що дає змогу ЗНМ вивчати складні залежності в даних.

Розглянемо найпоширеніші активаційні функції.

ReLU [2] є найбільш популярною активаційною функцією через свою ефективність і простоту обчислення, і має такий аналітичний вигляд:

$$ReLU(x) = max(0, x). \tag{1.1}$$

Графічне зображення цієї функції наведено на рис. 1.1.

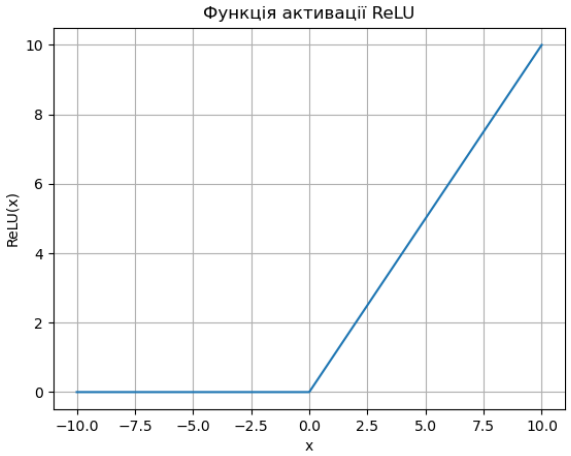

Рисунок 1.1 – Графік активаційної функції ReLU

Сигмоїдна функція [1] перетворює вхідні дані в діапазоні (0, 1), що робить її корисною для задач бінарної класифікації. Однак, через проблему затухання градієнта, вона використовується рідше, ніж ReLU. Ця функція має наступний аналітичний вигляд:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}. \tag{1.2}$$

Графічне зображення цієї функції наведено на рис.1.2.

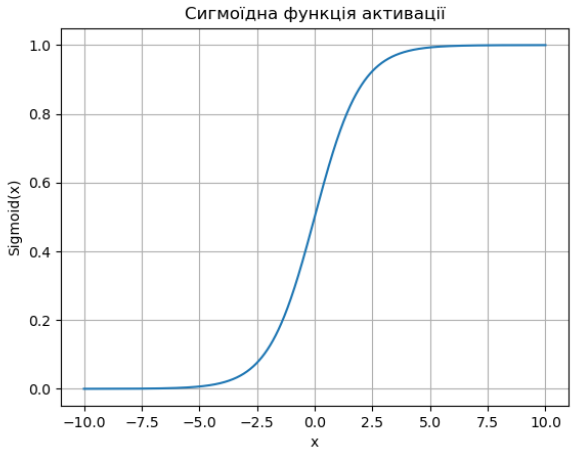

Рисунок 1.2 – Графік сигмоїдної функції активації

Функція гіперболічного тангенсу [1] перетворює вхідні дані в діапазоні (-1, 1), і є по суті масштабованою версією сигмоїди. Вона також схильна до проблеми затухання градієнта, хоча меншою мірою, ніж сигмоїда. Ця функція має наступний аналітичний вигляд:

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1. \tag{1.3}$$

Графічне зображення цієї функції наведено на рис.1.3.

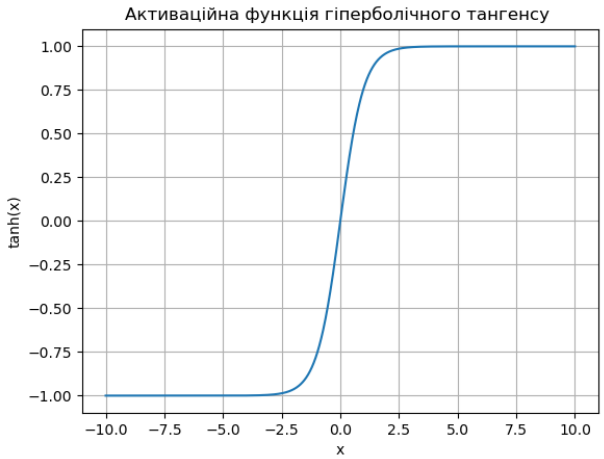

Рисунок 1.3 – Графік гіперболічної функції активації

Leaky ReLU [2] - це модифікація ReLU, яка дає змогу пропускати невеликі негативні значення для вирішення проблеми "мертвих нейронів".

$$Leaky\ ReLU\left(x\right)\ =\ max(\alpha x,\ x), \tag{1.4}$$

де α - невеликий коефіцієнт.

Графічне зображення цієї функції, з коефіцієнтом $\alpha = 0.2$, наведено на рис.1.4.

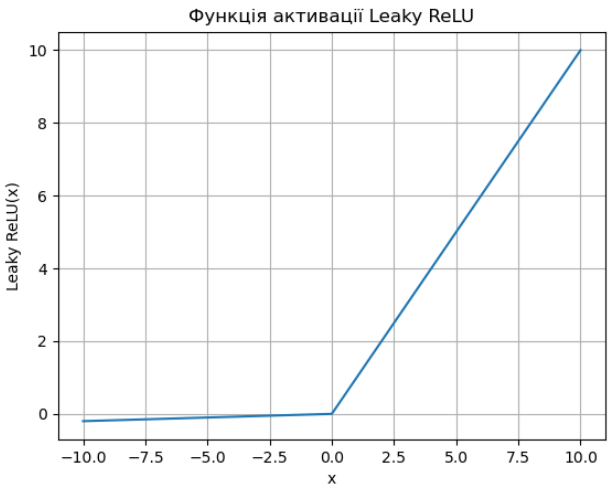

Рисунок 1.4 – Графік активаційної функції Leaky ReLU

1.3 Навчання моделі

Перед початком навчання нейронної мережі, всі ваги і зміщення мають бути ініціалізовані. Ініціалізація ваг важлива для забезпечення гарного старту в процесі навчання. Найпростішим способом є ініціалізація всіх ваг за допомогою невеликих випадкових чисел (наприклад, з нормального або рівномірного розподілу). Існують і більш просунуті методи ініціалізації, які враховують

розміри вхідного і вихідного шару для визначення оптимального масштабу випадкових ваг.

Під час прямого проходу, вхідні дані передаються через нейронну мережу шар за шаром. На кожному шарі виконуються певні операції, такі як згортка, агрегування або активація, і результат передається на наступний шар. Зрештою, на вихідному шарі генерується передбачення нейронної мережі. У разі навчання з учителем, передбачення порівнюють зі справжніми мітками, і на основі цього порівняння обчислюють функцію втрат.

Безпосередньо для навчання використовується функція зворотного поширення помилки. Цей метод використовує градієнтний спуск для оптимізації ваг і зміщень. Процес починається з обчислення градієнта функції втрат по відношенню до кожної з ваг і зміщень. Градієнт показує напрямок найбільшого зростання функції втрат, і оновлення ваг відбувається в протилежному напрямку, щоб мінімізувати втрати.

Під час зворотного поширення, градієнти обчислюються послідовно для кожного шару, починаючи з вихідного шару і рухаючись углиб мережі. Градієнти обчислюються з використанням правила ланцюгового диференціювання, яке дає змогу виразити градієнт складної функції через добуток градієнтів проміжних функцій. Таким чином, градієнти для ваг і зсувів кожного шару обчислюються на основі градієнтів попереднього шару (або, у разі вихідного шару, на основі градієнта функції втрат).

Після обчислення градієнтів для всіх ваг і зсувів у нейронній мережі, відбувається оновлення ваг із використанням алгоритму оптимізації, такого як стохастичний градієнтний спуск або adam. Оновлення ваг полягає в коригуванні їхніх значень з урахуванням обчислених градієнтів і швидкості навчання. Швидкість навчання є гіперпараметром, який визначає, наскільки сильно ваги будуть змінюватися під час кожного оновлення. Невелика швидкість навчання може призвести до більш точних результатів, але потребуватиме більше часу на

навчання, в той час як велика швидкість навчання може призвести до більш швидкого навчання, але з ризиком "пропуску" оптимальних значень ваг.

1.4 Вилучення ознак

Витяг ознак є ключовим етапом в алгоритмах виявлення об'єктів, заснованих на згорткових нейронних мережах (ЗНМ). Ідея полягає в тому, щоб перетворити вихідне зображення на набір значущих ознак, які можна використовувати для визначення об'єктів та їхнього місця розташування.

Згорткові шари є основними блоками згорткових нейронних мереж і відповідають за вилучення ознак із зображень. Кожен згортковий слой складається з декількох фільтрів (або ядер), які рухаються по вхідному зображенню з певним кроком. Результат згортки фільтра із зображенням - це карта ознак, яка представляє активацію фільтра на різних ділянках зображення. Це дає змогу згортковим шарам витягувати різні ознаки.

Згорткові нейронні мережі зазвичай мають кілька згорткових шарів, які послідовно застосовуються до зображення. Нижчі шари витягують прості та локальні ознаки, такі як межі та кути, тоді як глибші шари витягують складніші та абстрактніші ознаки, що представляють різні частини та об'єкти на зображенні. Ієрархічна структура ознак дає змогу ЗНМ краще розрізняти об'єкти різних класів і масштабів.

Агрегувальні шари використовуються для зменшення розмірності карт ознак і усунення надлишкової інформації. Вони допомагають знизити обчислювальне навантаження і зменшити кількість параметрів мережі, запобігаючи перенавчанню. Найпоширенішими видами агрегування є максимізаційне агрегування і усереднювальне агрегування.

Витягнуті ознаки використовуються в алгоритмах виявлення об'єктів для визначення місця розташування і класів об'єктів на зображенні. Наприклад, в алгоритмах, заснованих на регіонах, таких як R-CNN [3], Fast R-CNN [4], Faster

R-CNN [5], карти ознак використовуються для генерації пропозицій областей, які можуть містити об'єкти. Потім ці пропозиції областей класифікуються і коригуються для визначення точного місця розташування об'єктів.

В алгоритмах однопрохідного виявлення, таких як YOLO [7] і SSD [6], витягнуті ознаки безпосередньо використовуються для передбачення координат обмежувальних рамок і класів об'єктів. Ці алгоритми розбивають карту ознак на комірки і передбачають ймовірність наявності об'єкта, його клас і відносні координати для кожної комірки. У результаті, алгоритми однопрохідного виявлення зазвичай швидші, ніж алгоритми, засновані на регіонах, але менш точні.

1.6 Обмежувальні рамки, регіональні пропозиції та якірні рамки

В алгоритмах виявлення об'єктів главною метою є передбачення обмежувальних рамок і класів об'єктів, тому розберемо пов'язані з ними алгоритми докладніше.

Регіональні пропозиції - це кандидати на обмежувальні рамки, які алгоритми виявлення об'єктів генерують на основі ознак зображення. Ці пропозиції являють собою області зображення, які можуть містити об'єкти. Алгоритми генерують регіональні пропозиції, аналізуючи картини ознак на різних масштабах і просторових ієрархіях. Цей метод використовується в алгоритмах виявлення об'єктів, заснованих на регіонах

Якірні рамки являють собою набір попередньо визначені прямокутні форми, які мають різні співвідношення сторін і масштаби. Їх використовують в однопрохідних алгоритмах виявлення об'єктів для прискорення процесу виявлення об'єктів. У таких алгоритмах, зображення ділиться на рівномірну сітку, і кожна комірка сітки аналізується на наявність об'єктів. Для кожного осередку сітки передбачається певна кількість якірних рамок. Кількість якірних рамок на комірку залежить від алгоритму і конфігурації.

Замість того щоб безпосередньо передбачати координати обмежувальних рамок, однопрохідні алгоритми навчаються передбачати зсуви щодо якірних рамок. Це дає змогу алгоритму легше впоратися з масштабами і співвідношеннями сторін, оскільки базові форми вже задані якірними рамками.

Якірні рамки слугують відправною точкою для передбачення справжніх обмежувальних рамок і класів об'єктів.

Обмежувальні рамки зазвичай бувають осьовими, паралельними до осей зображення, що є поширеним варіантом для горизонтальних просторів. Однак, існують і орієнтовані рамки, які можуть повертатися відносно осей, використовуються для об'єктів з довільною орієнтацією.

1.5 Класифікація та регресія

Класифікація включає присвоєння міток класів об'єктам на основі витягнутих ознак. В архітектурах виявлення об'єктів зазвичай використовують згорткові нейронні мережі для вилучення ознак із зображень, які потім обробляють для передбачення ймовірностей належності до різних класів. Ймовірності класів зазвичай нормалізують за допомогою функції софтмакс для отримання ймовірнісного розподілу класів. Потім визначається клас із максимальною ймовірністю, який вважається передбаченим класом об'єкта.

Регресія обмежувальних рамок полягає у визначенні координат обмежувальних рамок об'єктів на основі витягнутих ознак. Мета регресії полягає в тому, щоб отримати обмежувальні рамки, які найточніше описують місце розташування і форму об'єкта на зображенні. В архітектурах виявлення об'єктів регресія зазвичай виконується паралельно з класифікацією, використовуючи ті самі ознаки для передбачення координат обмежувальних рамок.

1.6 Функції втрат і оптимізація

Функції втрат відіграють ключову роль у навчанні моделей виявлення об'єктів, оцінюючи різницю між передбаченнями моделі та істинними значеннями. Вони дають змогу визначити, наскільки добре модель передбачає результати.

Наведемо найпоширеніші з них

Крос-ентропійна втрата: використовується в задачах класифікації для оцінювання різниці між істинним імовірнісним розподілом класів і передбаченим імовірнісним розподілом; це стандартна функція втрат для багатокласової класифікації.

Focal Loss [9]: модифікація крос-ентропійної втрати, яка зменшує внесок простих прикладів у функцію втрат і збільшує внесок важких прикладів. Це дає змогу моделі сфокусуватися на складних об'єктах і поліпшити продуктивність.

Методи оптимізації використовуються для оновлення ваг моделі в процесі навчання на основі обчислених функцій втрат. У цій роботі будуть використовуватися два методи оптимізації.

Стохастичний градієнтний спуск: це ітеративний метод оптимізації, який використовує випадкову підмножину даних для оцінки градієнта функції втрат; СГС оновлює ваги моделі на кожному кроці, мінімізуючи функцію втрат і рухаючись у напрямку антиградієнта; СГС часто використовується з методами, такими як імпульс і затухання ваг для прискорення збіжності та запобігання застряганню в локальних мінімумах.

Adam (Adaptive Moment Estimation): це популярний метод оптимізації, що ґрунтується на адаптивній оцінці імпульсів першого та другого порядку. Він поєднує ідеї методів імпульсу та адаптивного коефіцієнта навчання для досягнення більш швидкої та стабільної збіжності.

1.7 Трансферне навчання і переднавчені моделі

Трансферне навчання - це метод машинного навчання, який дає змогу використовувати знання, отримані під час розв'язання одного завдання, для розв'язання іншого, часто пов'язаного завдання. У контексті виявлення об'єктів трансферне навчання часто застосовується для використання попередньо навчених моделей для прискорення процесу навчання і поліпшення продуктивності на нових даних.

Трансферне навчання пропонує ряд переваг, особливо для роботи з обмеженими даними, або коли навчання з нуля займає забагато часу. Застосування попередньо навчених, на великій кількості даних, моделей, дозволяє значно прискорити процес навчання, тому що вони вже можуть витягувати корисні ознаки із зображень.

Трансферне навчання включає в себе кілька етапів. Для початку, вибір переднавченої моделі, який залежить від поставленого завдання. Зазвичай це згорткова нейронна мережа, навчена на великому і різноманітному наборі даних.

Далі навчену модель потрібно адаптувати до нового завдання виявлення об'єктів. Часто доводиться модифікувати архітектуру моделі. Це може включати в себе заміну останніх шарів для адаптації до кількості класів об'єктів або додавання нових шарів для складніших завдань.

1.8 Оцінювання продуктивності

Оцінювання продуктивності алгоритмів виявлення об'єктів є критично важливим етапом, що дає змогу порівняти різні моделі та визначити, яка з них найбільше підходить для конкретного завдання. Розглянемо основні метрики.

Точність (precision): точність визначає, яка частка виявлених об'єктів є дійсно позитивними (тобто вірно виявленими); точність обчислюється за формулою:

$$Precision = \frac{TP}{(TP + FP)}, \quad (1.5)$$

де *TP* - кількість істинно позитивних об'єктів, а *FP* - кількість хибно позитивних об'єктів.

Повнота (recall): повнота показує, яку частку об'єктів з усіх позитивних прикладів у наборі даних модель змогла виявити; повнота обчислюється за формулою:

$$Recall = \frac{TP}{TP + FN}, \quad (1.6)$$

де *FN* - кількість хибно негативних об'єктів.

F1-міра: F1-міра є середнім гармонійним між точністю і повнотою; вона дає змогу врахувати і точність, і повноту в одній метриці. F1-міра обчислюється за формулою:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}. \quad (1.7)$$

IoU (Intersection over Union): метрика, яка визначає ступінь збігу передбаченої та істинної обмежувальної рамки, обчислюючи відношення площі їх перетину до об'єднання. Вище значення IoU вказує на кращий збіг, і передбачення з IoU вище встановленого порога (напр., 0.5) вважаються вірними.

mAP (mean Average Precision): це усереднена точність за всіма класами об'єктів, що використовується для оцінки ефективності алгоритмів виявлення об'єктів. mAP відображає загальну продуктивність алгоритму.

2 Технологічні рішення розпізнавання об'єктів

Перед тим, як розглядати методи виявлення об'єктів з повітря, слід зрозуміти основні принципи виявлення об'єктів. Відмінність виявлення від класифікації полягає в потребі не лише визначити клас об'єкта, а й його місцезнаходження на зображенні. Простий підхід до цього - розбити зображення на регіони і класифікувати кожен окремо, але це неефективно через різноманітність розмірів і місцезнаходження об'єктів.

Метод пропозиції регіонів, який використовує селективний пошук, дозволяє зменшити кількість регіонів до 2000. Ці регіони потім обробляються згортковою нейронною мережею (ЗНМ), яка витягує ознаки для подальшої класифікації. Цей підхід був впроваджений в R-CNN [3], одній з перших моделей глибокого навчання для виявлення об'єктів.

Однак, R-CNN має недоліки, такі як великий час навчання і неможливість використання в реальному часі. Fast R-CNN [4], нова архітектура, вирішує ці проблеми, використовуючи вхідне зображення для створення згорточної карти ознак, замість обробки кожного регіону окремо.

Наступний крок - Faster R-CNN [5], який використовує окрему нейромережу, RPN [5] (Region Proposal Network), для передбачення пропозицій регіонів, замість селективного пошуку. Це дозволяє досягти швидкості тестування 200 мс, що робить можливим використання Faster R-CNN для виявлення об'єктів у реальному часі.

2.2 Згорткова нейронна мережа на основі регіонів (R-FCN)

R-FCN [8] (Region-based Fully Convolutional Networks) - це згорткова нейромережа, що розвиває концепцію Fast/Faster R-CNN. Однак, вона відрізняється тим, що використовує повністю згортковий підхід до виявлення об'єктів на основі регіонів, розподіляючи обчислення по всьому зображенню, а

не застосовуючи їх до кожного регіону окремо. Це робить R-FCN швидшою порівняно з попередниками.

Як і в архітектурах R-CNN, R-FCN використовує двоетапний процес виявлення об'єктів, що включає пропозицію регіонів і їх класифікацію. Регіони-кандидати визначаються за допомогою мережі пропозиції регіонів (RPN [5]), яка є повністю згортковою архітектурою. Відповідно до принципів Faster R-CNN [5], ознаки діляться між RPN і R-FCN. Загальний вигляд системи представлено на рис.2.2.1.

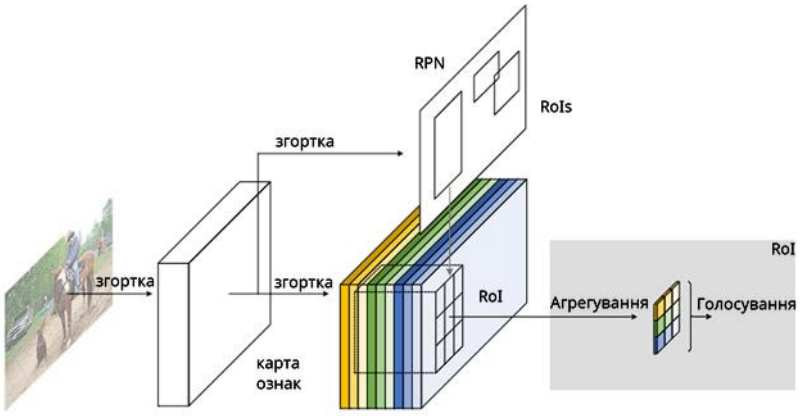

Рисунок 2.1 – Загальна архітектура R-FCN (адаптовано з [8])

Архітектура R-FCN призначена для класифікації RoI на категорії об'єктів і фону. У R-FCN всі вагові шари, що навчаються, є згортковими і обчислюються на всьому зображенні. Останній згортковий шар створює набір із k^2 позиційно-чутливих карт оцінок для кожної категорії і, таким чином, має $k^2(C + 1)$-канальний вихідний шар із C категоріями об'єктів (+1 для фону).

Набір із k^2 карт оцінок відповідає просторовій сітці $k \times k$, що описує відносні позиції. Наприклад, за $k \times k = 3 \times 3$, отже, 9 карт оцінки кодують випадки *{верхній лівий, верхній центральний, верхній правий, ..., нижній правий}* категорій об'єктів.

R-FCN завершується позиційно-чутливим шаром об'єднання регіонів інтересу. Цей шар об'єднує виходи останнього згорткового шару і генерує оцінки для кожного RoI. Позиційно-чутливий шар RoI проводить селективне об'єднання, і кожна з $k \times k$ клітинок об'єднує відповіді тільки від однієї карти оцінок з набору $k \times k$ карт оцінок. Завдяки наскрізному навчанню, цей шар RoI

допомагає останньому згортковому шару для навчання спеціалізованих карт оцінок, чутливих до положення. Рис.2.2 ілюструє цю ідею. На Рис.2.3 і 2.4 показано приклад.

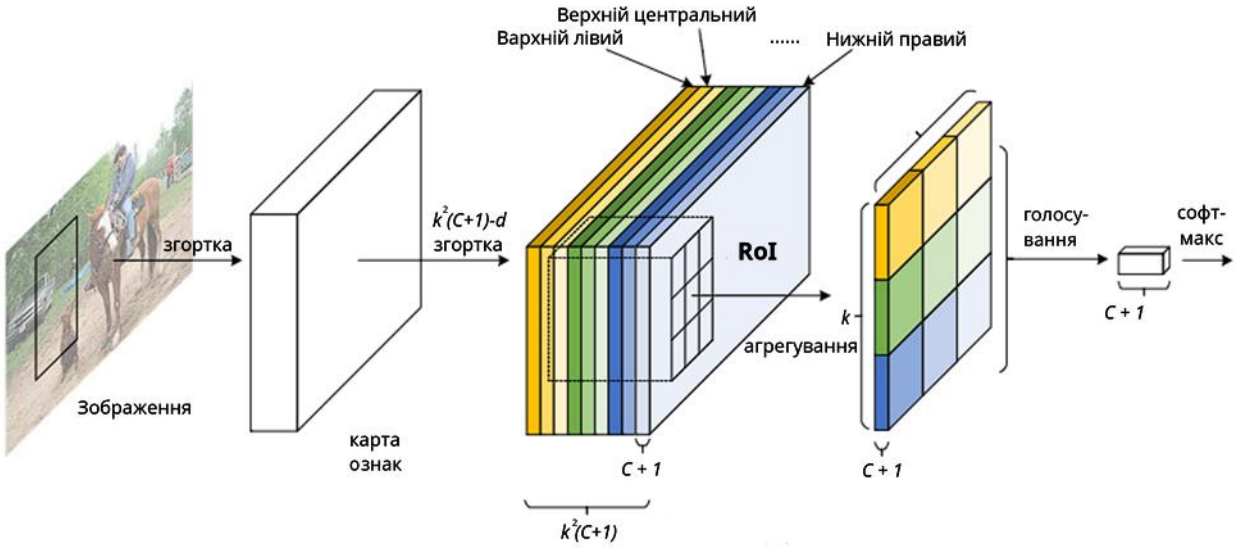

Рисунок 2.2 – Ключова ідея R-FCN для виявлення об'єктів (адаптовано з [8])

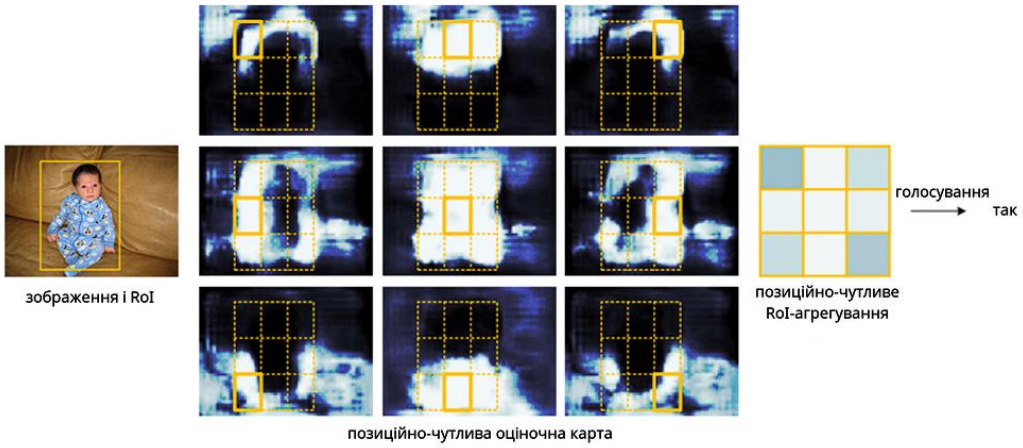

Рисунок 2.3 – Візуалізація R-FCN ($k \times k = 3 \times 3$) для категорії людей (адаптовано з [8])

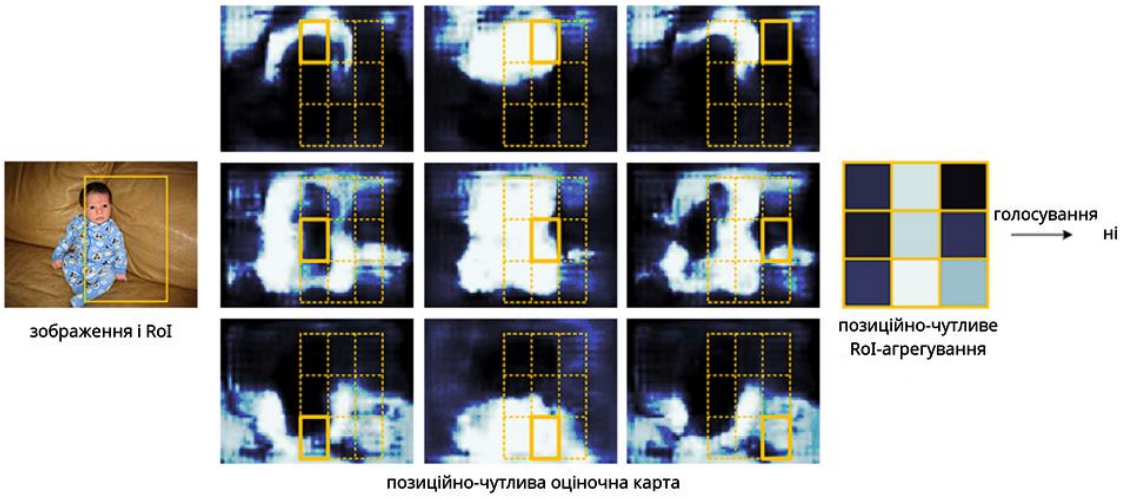

Рисунок 2.4 – Візуалізація, коли RoI неправильно накладається на об'єкт (адаптовано з [8])

функція втрат, визначена для кожного RoI, є сумою втрат крос-ентропії та втрат регресії рамок [8]:

$$L\left(s,\ t_{x,y,w,h}\right)\ =\ L_{cls}(s_{c^*})\ +\lambda[c^*>\ 0]L_{reg}(t,\ t^*)\,. \tag{2.1}$$

Тут c^* - справжня мітка RoI ($c^*=0$ означає фон). $L_{cls}(s_{c^*})=-log(s_{c^*})$ - втрата перехресної ентропії для класифікації, L_{reg} - втрата регресії обмежувального поля, а t^* являє собою поле істини. $[c^*>\ 0]$ - це індикатор, який дорівнює 1, якщо аргумент істинний, і 0 в іншому випадку. Вага балансу встановлюється λ = 1. Позитивні приклади визначаються як RO, що мають перетин по об'єднанню (IoU) з полем істини не менше ніж 0.5, і негативні в іншому випадку.

Висока продуктивність обчислень на кожному Roi забезпечує майже безвитратну обробку прикладів. За умови N пропозицій на зображенні, при прямому проході оцінюються втрати всіх N пропозицій. Потім сортуються всі RoI (позитивні та негативні) за втратами та вибираються ті, які мають найбільші втрати. Зворотне поширення виконується на основі обраних прикладів. Оскільки обчислення для кожного RoI нехтувано малі, час зворотного поширення майже не залежить від N.

Карти ознак, які поділяють RPN і R-FCN, обчислюються. Потім частина RPN пропонує регіони інтересу, за якими частина R-FCN оцінює бали за категоріями і регресує обмежувальні поля. Під час виведення оцінюються 300 RoI. Результати обробляються методом подавлення немаксимальних значень з використанням порога 0.3 IoU [8].

Підбиваючи підсумки, R-FCN зберігає точність виявлення об'єктів на рівні Faster R-CNN, але значно збільшує швидкість обробки. У R-FCN вся мережа є повністю згортковою, що дає змогу суттєво прискорити процес виявлення, особливо під час роботи з великими зображеннями.

Незважаючи на те, що R-FCN і швидший за багато інших систем виявлення об'єктів, він все ще відстає за швидкістю від деяких одностадійних детекторів, таких як YOLO [7] або SSD [6]. Проте R-FCN є важливим кроком у розвитку технології виявлення об'єктів.

2.3 Одностадійний детектор з уніфікованим підходом (YOLO)

Після серії архітектур R-CNN, YOLO [7] надав новий підхід до виявлення об'єктів. У попередніх роботах з виявлення об'єктів класифікатори використовуються повторно для виявлення. Замість цього YOLO розглядає виявлення об'єктів як задачу регресії на просторово розділені обмежувальні рамки і пов'язані з ними ймовірності класів. Одна нейронна мережа передбачає обмежувальні рамки та ймовірності класів безпосередньо за цілими зображеннями за одну обробку. Оскільки весь процес виявлення є єдиною мережею, весь процес, від вхідних даних до кінцевого результату, може оптимізуватися спільно для досягнення найкращої продуктивності.

Загальна ідея YOLO досить проста: одна згорткова мережа одночасно передбачає кілька обмежувальних рамок і ймовірності класів для цих рамок; YOLO навчається на повних зображеннях і безпосередньо оптимізує ефективність виявлення.

Ця одностадійна модель має кілька переваг перед двостадійними методами виявлення об'єктів. Ілюстрацію ідеї можна побачити на рис.2.5.

Рисунок 2.5 – Система виявлення YOLO. Система YOLO (1) змінює розмір вхідного зображення до 448 × 448, (2) запускає одну згорткову мережу на зображенні і (3) встановлює поріг виявлення відповідно до довіри моделі (адаптовано з [7])

YOLO використовує ознаки всього зображення для передбачення кожної обмежувальної рамки. також YOLO передбачає всі обмежувальні рамки для всіх класів зображення одночасно. Це означає, що мережа роздумує глобально про повне зображення і всі об'єкти на ньому. Конструкція YOLO забезпечує наскрізне навчання і швидкість роботи в реальному часі при збереженні високої середньої точності. Ця система ділить вхідне зображення на сітку $S \times S$. Якщо центр об'єкта потрапляє в клітинку сітки, ця клітинка відповідає за виявлення об'єкта. Кожна комірка сітки передбачає B обмежувальних комірок і дає оцінку впевненості для цих комірок. Ці показники впевненості відображають, наскільки модель упевнена в тому, що в комірці знаходиться об'єкт, а також наскільки точно, на її думку, вона передбачає комірку.

Упевненість визначається як $Pr(Object) * IOU_{pred}^{truth}$. Якщо в цей комірці немає об'єкта, то оцінка впевненості має дорівнювати нулю. Інакше, показник упевненості дорівнює перетинанню за об'єднанням (IOU) між передбачуваною (*pred*) коміркою і істиною (*truth*).

Кожна обмежувальна рамка складається з 5 передбачень: *x, y, w, h* і *впевненість*. Координати *(x, y)* являють собою центр поля відносно меж комірки сітки. Ширина *w* і висота *h* передбачаються щодо всього зображення. Нарешті, прогноз упевненості являє собою IOU між передбаченою коміркою і будь-якою базово істинною коміркою.

Кожна комірка сітки також передбачає *C* умовних імовірностей класу *C*, $Pr(Class_i \mid Object)$. Ці ймовірності зумовлені тим, що комірка сітки містить об'єкт. Передбачається тільки один набір імовірностей класів для кожної комірки сітки, незалежно від кількості комірок *B*. Під час тестування перемножуються умовні ймовірності класів і прогнози впевненості для кожної комірки [7],

$$Pr(Class_i \mid Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}, \quad (2.2)$$

що дає специфічні для класу оцінки впевненості для кожної комірки. Ці оцінки відображають як імовірність появи даного класу в комірці, так і те, наскільки добре передбачена комірка відповідає об'єкту.

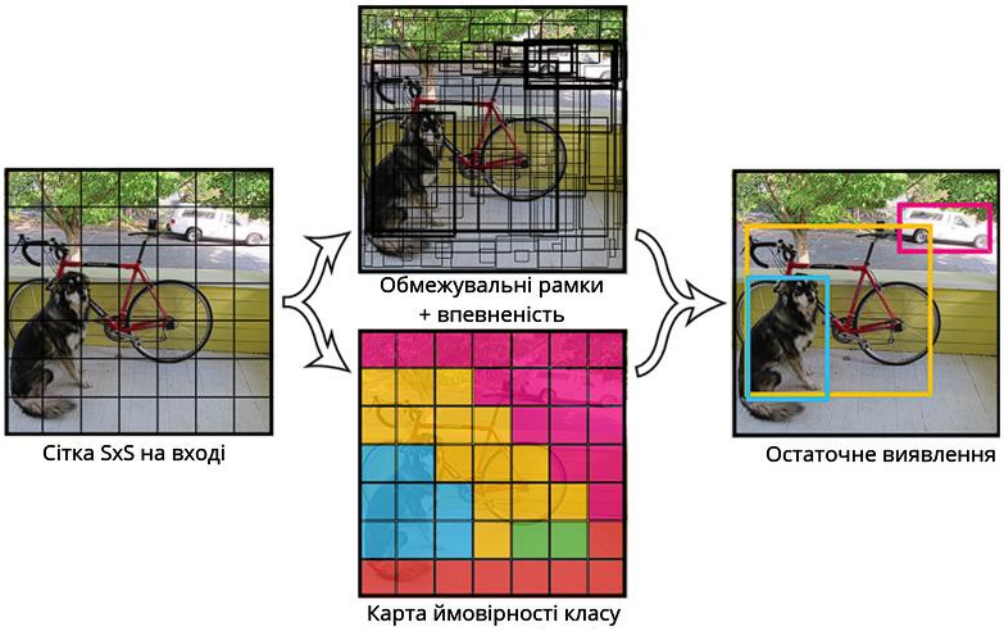

Рисунок 2.6 – модель (адаптовано з [7])

Архітектура YOLO має такий вигляд. Початкові згорткові шари мережі витягують ознаки із зображення, а повнозв'язні шари передбачають вихідні ймовірності та координати. Мережа складається з 24 згорткових шарів і 2 повнозв'язних шарів. Далі використовуються 1 × 1 шари зменшення розмірності, за якими йдуть 3 × 3 згорткових шари. Повна архітектура показана на рис.2.7.

Fast YOLO [7] використовує нейронну мережу з меншою кількістю згорткових шарів (9 замість 24) і меншою кількістю фільтрів у цих шарах. Крім розміру мережі, всі параметри навчання і тестування однакові для YOLO і Fast YOLO.

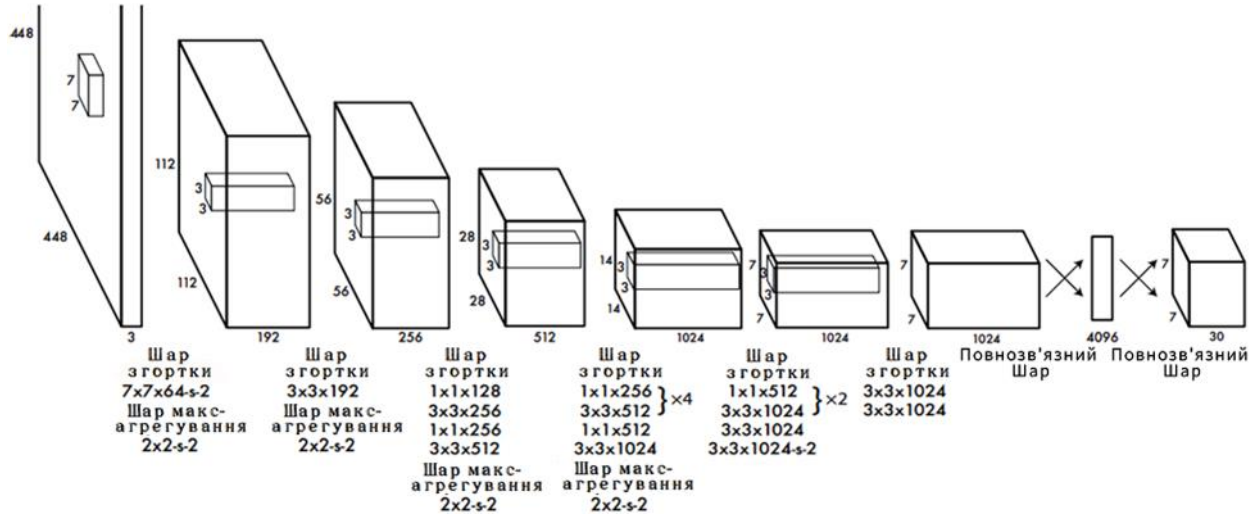

Рисунок 2.7 – Архітектура YOLO (адаптовано з [7])

Кінцевим виходом мережі є тензор передбачень 7 × 7 × 30.

Для останнього шару використовується лінійна функція активації, а всі інші шари використовують таку функцію активації leaky ReLU:

$$\varphi(x) = \begin{cases} x, & \text{якщо } x > 0 \\ 0.1x, & \textit{інакше} \end{cases} . \qquad (2.3)$$

YOLO використовує суму квадратів помилок як функцію втрат через те, що вона проста в оптимізації. Однак цей підхід не ідеально відповідає меті максимізації середньої точності. Вага помилки локалізації, то наскільки точно модель визначає положення об'єкта в кадрі, дорівнює вазі помилки класифікації, то, наскільки точно модель визначає клас об'єкта. Також, у кожному зображенні багато сіткових комірок не містять об'єктів. Це призводить до того, що оцінка впевненості цих комірок прагне до нуля, часто перебиваючи градієнт від

комірок, які дійсно містять об'єкти. Це може призводити до нестабільності моделі та передчасного припинення процесу навчання.

Цю проблему розв'язують збільшенням втрати від передбачень координат рамки, що обмежує, і зменшуємо втрати від передбачень впевненості для рамок, що не містять об'єктів. Для цього використовуються два параметри, $\lambda_{coord} = 5$ і $\lambda_{noobj} = 5$ [7].

Функція суми квадратів помилок також однаково зважує помилки у великих і малих комірках. І для того щоб невеликі відхилення у великих комірках мали менше значення, ніж у маленьких, передбачається квадратний корінь із ширини та висоти обмежувальної рамки замість ширини та висоти безпосередньо.

YOLO передбачає кілька обмежувальних рамок на клітинку сітки. Під час навчання, для того щоб за кожен об'єкт відповідав тільки один передбачувач, один із передбачувачів призначається "відповідальним" за передбачення об'єкта на основі того, яке передбачення має найбільший поточний IOU з базовою істиною. Це призводить до спеціалізації між передбачувачами обмежувальних рамок. Кожен передбачувач краще передбачає певні розміри, співвідношення сторін або класи об'єктів, що покращує загальний результат.

Під час навчання оптимізація відбувається за допомогою такої багатокомпонентної функції втрат [7]:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(x_i - \hat{x}_i \right)^2 + \left(y_i - \hat{y}_i \right)^2 \right] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
, \qquad (2.4)
$$

де 1_i^{obj} позначає появу об'єкта в комірці *i*, 1_{ij}^{obj} позначає, що *j*-й передбачувач обмежувальної рамки в комірці *i* "відповідає" за це передбачення.

Ця функція втрат штрафує за помилку класифікації, тільки якщо об'єкт присутній у цієї комірці сітки (звідси умовна ймовірність класу, розглянута раніше). Вона також штрафує за помилку координат граничного поля, тільки якщо цей передбачувач "відповідає" за поле істини (тобто має найвищий IOU серед усіх передбачувачей у цій комірці сітки).

Конструкція сітки забезпечує просторову різноманітність у передбаченнях обмежувальних рамок. Часто зрозуміло, в який осередок сітки потрапляє об'єкт, і мережа передбачає тільки одну комірку для кожного об'єкта. Однак деякі великі об'єкти або об'єкти, що розташовані поблизу кордону кількох комірок, можуть бути якісно локалізовані кількома комірками. Для усунення таких множинних виявлень можна використовувати подавлення немаксимальних значень. Хоча це не критично для продуктивності, як у випадку R-CNN, подавлення немаксимальних значень додає 2-3% у mAP [7].

Підбиваючи підсумки, YOLO - це надзвичайно швидка архітектура, в якій виявлення об'єктів розглядають як проблему регресії, що дає змогу відмовитися від комплексних алгоритмів. Щоб передбачити виявлення, достатньо запустити цю нейронну мережу на новому зображенні під час тестування. За рахунок цього і досягається висока швидкість роботи, що дає змогу YOLO бути системою виявлення об'єктів у реальному часі. Базова версія YOLO обробляє зображення зі швидкістю 45 кадрів на секунду на графічному процесорі Titan X. Зменшена версія мережі, Fast YOLO, обробляє 155 кадрів на секунду, при цьому на момент оригінальної розробки, досягала вдвічі більшого mAP, ніж інші детектори реального часу. Ця система дає змогу обробляти потокове відео в режимі реального часу із затримкою менше ніж 25 мілісекунд.

YOLO здатний глобально оцінювати зображення під час складання прогнозів. На відміну від методів, заснованих на ковзному вікні і пропозиції

регіонів, YOLO бачить все зображення під час навчання і тестування, тому він неявно кодує контекстну інформацію про класи, а також їхній зовнішній вигляд.

Оскільки YOLO володіє високим ступенем узагальненості, він з меншою ймовірністю зламається при застосуванні до нових областей або при несподіваних вхідних даних. За точністю YOLO відстає від більш комплексних сучасних систем виявлення. Навіть дивлячись на здатність швидко ідентифікувати об'єкти на зображеннях, у цієї системи є проблеми з точністю локалізації деяких об'єктів, особливо маленьких.

Так само варто зазначити, що навчання YOLO може бути складним через його складну функцію втрат, яка має враховувати різні аспекти, такі як координати рамки, розміри рамки, ймовірність присутності об'єкта та ймовірності класу.

Деякі з цих проблем поступово виправляли під час модернізації YOLO, аж до YOLOv5, модернізовану версію якої ми будемо використовувати при навчанні нашої моделі. Так само ідею YOLO продовжила і вдосконалила система під назвою SSD

2.4 Одностадійний детектор із фіксованими рамками (SSD)

Single Shot MultiBox Detector [6], або SSD, це ще один одностадійний детектор об'єктів, який генерує фіксований набір рамок і оцінює присутність об'єктів у них. Він базується на прямопрохідній згортковій мережі, заснованій на стандартній архітектурі для класифікації зображень, до якої додається допоміжна структура для виявлення.

SSD використовує багатомасштабні карти ознак, додаючи додаткові згорткові шари ознак до базової мережі. Це відрізняє його від YOLO, який працює з одномасштабною ознаковою картою.

Кожен доданий шар ознак створює фіксований набір прогнозів виявлення за допомогою згорткових фільтрів (див. рис.2.4.1). Це виконується за допомогою

ядра розміром 3 × 3 × p, яке передбачає параметри потенційного виявлення, виробляючи оцінку категорії або зміщення форми відносно координат вихідних рамок.

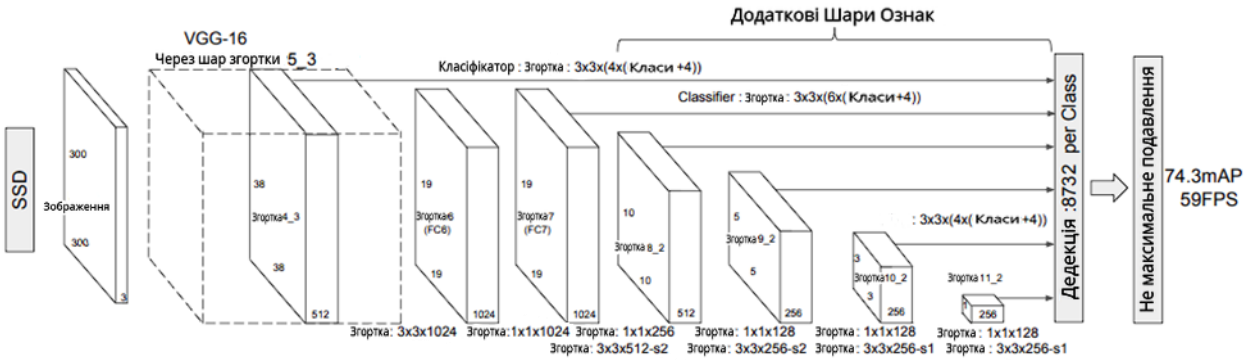

Рисунок 2.8 – Архітектура SSD (адаптовано з [6])

SSD додає кілька шарів ознак до кінця базової мережі, які передбачають зміщення до вихідних рамок різних масштабів і співвідношень сторін, а також пов'язані з ними впевненості в прогнозах.

SSD генерує фіксований набір рамок, пов'язаних з комірками на верхніх картах ознак мережі. Кожна комірка передбачає зміщення форми рамки та оцінки класів, вказуючи на наявність об'єкта класу в рамці. Для кожної з k рамок у комірці обчислюється с оцінок класів і 4 зміщення форми рамки, що призводить до $(c + 4)k$ фільтрів на комірку і $(c + 4)kmn$ виходів для карти ознак $m \times n$ (див. рис.2.9). Початкові рамки, подібні до якірних рамок у Faster R-CNN [5], застосовуються до кількох карт ознак різної роздільної здатності, що дозволяє ефективно дискретизувати простір можливих форм рамок.

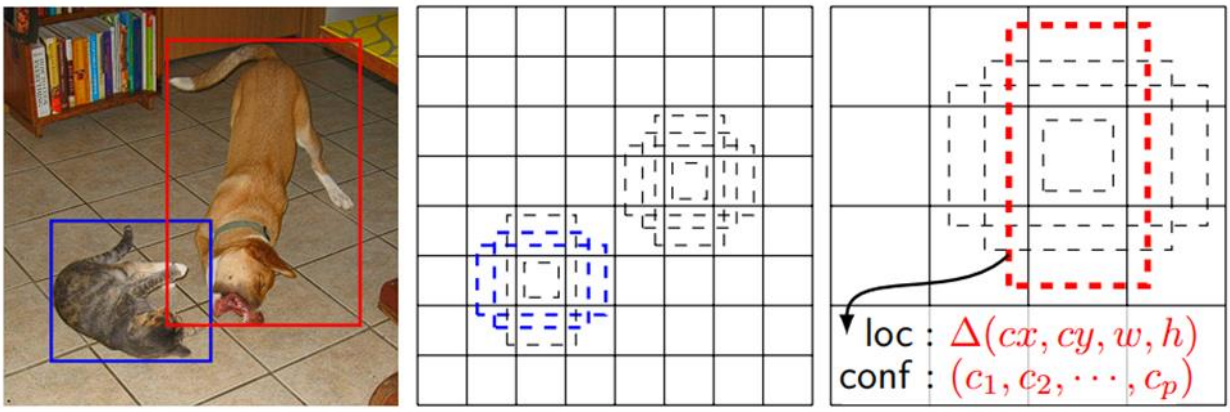

Рисунок 2.9 - Фреймворк SSD (адаптовано з [6])

Навчання SSD схоже на YOLO: потрібно призначити реальну позицію певним виходам детектора, визначити функцію втрат, застосувати зворотне поширення помилки. Воно також включає вибір вихідних рамок, масштабів для виявлення, стратегій відбору складних негативних прикладів та збільшення даних. Під час навчання SSD визначає, які вихідні рамки відповідають реальному виявленню. Для кожної реальної рамки вибір відбувається з вихідних рамок, що відрізняються за місцем, співвідношенням сторін та масштабом. Зіставлення реальної рамки з вихідною відбувається за перетином за Жаккаром [16]. Вихідні рамки зіставляються з реальними, які мають перетин за Жаккаром вище 0.5. Це спрощує навчання, дозволяючи мережі передбачати високі оцінки для кількох перекриваючихся вихідних рамок.

Для того щоб справлятися з об'єктами різного масштабу, в SSD використовуються як нижні, так і верхні карти ознак для виявлення. На рис.2.9 показано дві зразкові карти ознак (8x8 і 4x4), які використовуються в рамках моделі. На практиці можна використовувати набагато більше карт із невеликими обчислювальними витратами.

У SSD розташування стандартних рамок проєктується так, щоб певні карти ознак вчилися реагувати на конкретні масштаби об'єктів. Якщо ми маємо *m* карт ознак для прогнозування, то масштаб стандартних рамок для кожної карти ознак розраховують так [6]:

$$S_k = S_{\min} + \frac{S_{\max} - S_{\min}}{m-1}(k-1),\ k \in [1,\ m], \tag{2.5}$$

де $S_{\min} = 0.2$ а $S_{\max} = 0.9$. Це означає, що найнижчий шар має масштаб 0.2, а верхній шар - масштаб 0.9, а всі шари між ними рівномірно розподілені. Далі різні співвідношення сторін накладаються на стандартні рамки, і позначаються як $a_r \in \left\{1,\ 2,\ 3,\ \frac{1}{2}\ ,\frac{1}{3}\right\}$.

Ширина обчислюється як

$$w_k^a = s_k\sqrt{a_r}\ , \tag{2.6}$$

а висота як

$$h_k^a = s_k \Big/ \sqrt{a_r}\ , \tag{2.7}$$

для кожної стандартної рамки. Для співвідношення сторін 1 ми також додаємо стандартну рамку, масштаб якої дорівнює:

$$S_k' = \sqrt{S_k S_k + 1}, \tag{2.8}$$

що призводить до 6 стандартних рамок на кожне місце на карті ознак. Центр кожної стандартної рамки встановлюється на $\left(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|}\right)$, де $|f_k|$ — це розмір k-ої квадратної карти ознак, $i, j \in \left[0, |f_k|\right)$.

Під час навчання SSD, прогнози для стандартних рамок з різними масштабами і співвідношеннями сторін з усіх місць на картах ознак комбінуються, що дає різноманітний набір прогнозів. Наприклад, на рис.2.9,

собака зіставляється з рамкою на карті ознак 4 × 4, але не з будь-якими стандартними рамками на карті ознак 8 × 8. Після зіставлення, більшість рамок є негативними, що створює дисбаланс між позитивними і негативними прикладами. Тому, замість використання всіх негативних прикладів, вибираються найкращі з них з найвищими втратами впевненості, зі співвідношенням до позитивних не більше ніж 3:1. Це призводить до швидкої оптимізації та стабільного навчання.

Для стійкості до різних розмірів і форм вхідних об'єктів, використовують такі методи [7]: кожне навчальне зображення випадковим чином вибирається і обробляється згідно з визначеними критеріями, включаючи використання всього оригінального вхідного кадра; вибір ділянки з мінімальним перетином Жаккара [16] 0.1, 0.3, 0.5, 0.7 або 0.9; вибір розміру ділянки від 0.1 до 1 від розміру вихідного зображення зі співвідношенням сторін від 1/2 до 2.

У підсумку, SSD, хоча й подібний до YOLO за одностадійним підходом до виявлення об'єктів, має декілька ключових відмінностей.

SSD, на відміну від YOLO, використовує різні масштаби і співвідношення сторін для передбачувальних рамок, що покращує обробку об'єктів різних розмірів. Він створює багато "якірних" рамок на різних шарах згорткової мережі. SSD не обмежується центруванням об'єктів в комірках сітки, як YOLO, тому може передбачити об'єкти в будь-якому місці зображення. SSD зазвичай має вищу точність, особливо при виявленні малих об'єктів, але працює повільніше, що може бути критичним для роботи в реальному часі.

2.5 Функція фокусної втрати (Focal Loss)

Незважаючи на швидкість, одностадійні детектори зазнають програють в точності двоетапним однією з причин чого є дисбаланс класів переднього і заднього плану. Для вирішення цієї проблеми, була створена архітектура RetinaNet [9] з функцією втрат Focal Loss [9].

RetinaNet демонструє вражаючі результати - витримуючи швидкість одноступеневих детекторів, вона перевершує за точністю всі двостадійні детектори на час її створення.

Focal Loss являє собою функцію втрат крос-ентропії, що динамічно масштабується, де масштабувальний коефіцієнт зменшується до нуля в міру збільшення впевненості в правильному класі, що демонструється на рис.2.10. Цей масштабувальний коефіцієнт може автоматично зменшити внесок легких прикладів у процесі навчання і швидко сконцентрувати модель на складних прикладах.

Розберемо функцію Focal Loss. Для початку Focal Loss вводиться, виходячи з втрат крос-ентропії (CE) для бінарної класифікації [9]:

$$CE(p, y) = \begin{cases} -\log(p), \text{ якщо } y = 1 \\ -\log(1-p), \text{ інакше} \end{cases}. \quad (2.9)$$

У вищевказаній формулі $y \in \{\pm 1\}$, визначає клас істинного значення, а $p \in [0, 1]$, це оціночна ймовірність моделі для класу з міткою $y = 1$. Для зручності позначень вводиться параметр p_t:

$$p_t = \begin{cases} p, \text{ якщо } y = 1 \\ 1 - p, \text{ інакше} \end{cases}. \quad (2.10)$$

Переписана формула має такий вигляд:

$$CE(p, y) = CE(p_t) = -\log(p_t). \quad (2.11)$$

У цьому випадку бінарна класифікація використовується виключно для зручності у формулюваннях. У багатокласових випадках Focal Loss так само є простим і добре працює.

Найчастішим методом боротьби з дисбалансом класів є введення вагового коефіцієнта α, що належить до діапазону [0, 1], для класу 1 і 1 - α для класу -1. На практиці α може бути встановлено обернено до частоти класу або розглядатися як гіперпараметр, який встановлюється за допомогою крос-валідації. Для зручності позначень визначається аналогічно тому, як визначається p_t.

α-збалансована функція втрат крос-ентропії має такий вигляд [9]:

$$CE(p_t) = -\alpha_t log\left(p_t\right). \tag{2.12}$$

Ця функція втрат являє собою просте розширення функції крос-ентропії, яке ми розглядаємо як базу для запропонованої фокусної функції втрат.

Великий дисбаланс класів, з яким стикаються під час навчання одностадійні детектори, пригнічує функцію втрат на основі крос-ентропії. Від'ємні приклади, які легко класифікуються, становлять більшу частину втрат і домінують у градієнті. Хоча α балансує важливість позитивних і негативних прикладів, він не розрізняє легкі та важкі приклади. Натомість можна змінити функцію втрат таким чином, щоб зменшити вагу легких прикладів і, таким чином, зосередити навчання на важких негативних прикладах.

Для цього потрібно додати модулюючий фактор $(1 - pt)^{\gamma}$ до функції втрат на основі перехресної ентропії, з параметром фокусування, що налаштовується $\gamma \geq 0$. Фокусна функція втрат тоді визначається таким чином [9]:

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t). \tag{2.13}$$

Focal Loss має дві цікаві властивості. Коли приклад неправильно класифікований і малий, модулюючий фактор близький до 1, і втрати не впливають на нього. Коли оцінка ймовірності прикладу наближається до 1,

коефіцієнт стає рівним 0, і втрати для добре класифікованих прикладів зменшуються.

Параметр фокусування γ плавно регулює швидкість зниження ваги легких прикладів. Коли $\gamma = 0$, *FL* еквівалентний *CE*, а при збільшенні γ вплив модулюючого фактора також збільшується.

Модулювальний фактор зменшує внесок у втрати від легких прикладів і розширює діапазон, у якому приклад отримує низькі втрати. Наприклад, з γ = 2, приклад, класифікований із $p_t = 0.9$, матиме втрати в 100 разів нижчі, ніж при використанні *CE*, а с $p_t \approx 0.968$ - втрати будуть у 1000 разів нижчими. Це, своєю чергою, збільшує важливість коригування помилково класифікованих прикладів.

Ось такий вигляд матиме α-збалансований варіант Focal Loss [9]:

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) . \qquad (2.14)$$

Підбиваючи підсумки, Focal Loss, розроблена спеціально для боротьби з проблемою дисбалансу класів у одноетапних детекторах, ефективно збалансовує класи, зосереджуючи модель на "важких" випадках та зменшуючи вплив "легких" випадків фону. Завдяки використанню Focal Loss, одноетапні детектори можуть досягати високої точності виявлення об'єктів, Протиставляючись їх двостадійним аналогам. При цьому, не дивлячись на додаткову складність функції Focal Loss, вона майже не впливає на швидкість обробки

3 Технологічне рішення розпізнавання об'єктів з повітря

Наявні горизонтальні детектори мають фундаментальні обмеження для завдань аерозйомки, де об'єкти можуть з'являтися під різними кутами. Тому було запропоновано багато детекторів обертання, заснованих на загальній схемі виявлення. Найпопулярнішим методом для цього завдання є параметрична регресія, яка в основному включає п'ятипараметричні та восьмипараметричні регресійні методи.

Найпоширеніші п'ятипараметричні регресійні методи реалізують виявлення довільно орієнтованої граничної області шляхом додавання додаткового параметра кута *θ*. На рисунку 3.1(а) показано одне з прямокутних визначень *(x, y, w, h, θ)* з кутовим діапазоном 90°, де *θ* позначає гострий кут до осі x, а для іншої сторони ми позначаємо його як *w*. Є інше визначення *(x, y, h, w, θ)*, проілюстрованого на рисунку 3.1(б), з кутовим діапазоном 180°, де *θ* визначається довгою стороною *h* прямокутника і віссю *x*.

Восьмипараметричні детектори на основі регресії безпосередньо регресують чотири кути *(x1, y1, x2, y2, x3, y3, x4, y4)* об'єкта, тому прогноз є чотирикутником. Ключовим кроком до чотирикутної регресії є попереднє визначення чотирьох кутових точок, що дозволяє уникнути дуже великих втрат, навіть якщо прогноз правильний, як показано на рисунку 3.1(в).

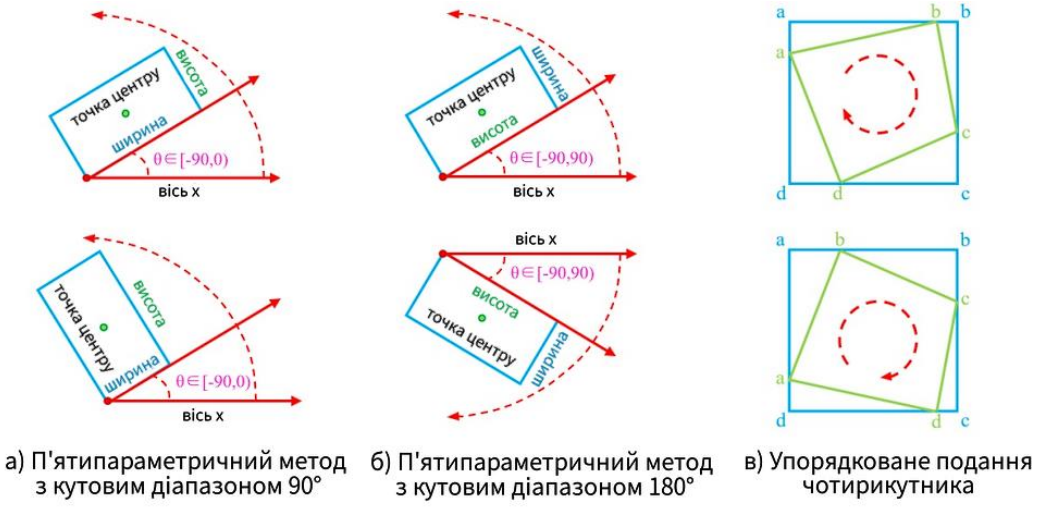

Рисунок 3.1 – Варіанти визначення обмежувальних рамок (адаптовано з [12])

Хоча метод виявлення обертання на основі параметричної регресії досяг конкурентної ефективності в різних задачах технічного зору і став основою для низки чудових методів виявлення, ці методи суттєво страждають від проблеми розривних границь [11]. Проблеми розриву границь часто спричинені кутовою періодичністю в п'ятипараметричному методі та кутовим впорядкуванням у восьмипараметричному методі.

Проблема розриву границі часто призводить до того, що значення втрат моделі на границі раптово зростає. Таким чином, методи повинні вдаватися до певних і часто складних трюків, щоб пом'якшити цю проблему. Тому ці методи виявлення часто є неточними в граничних умовах.

Для вирішення цієї проблеми було запропоновано метод CSL [12], який розглядає прогнозування кута не як проблему регресії, а як проблему класифікації

3.1 Циркулярна гладка мітка (CSL)

Csl (Circular Smooth Label), пропонує прямолінійне рішення - використовувати кут об'єкта як його мітку категорії, а кількість категорій пов'язати з діапазоном кута.

Перетворення з регресії на класифікацію може спричинити певні втрати в точності. Взявши метод із п'ятьма параметрами і діапазоном кута 180° як приклад, ω (за замовчуванням ω = 1°) градусів на інтервал належить до категорії. Можно обчислити максимальні втрати в точності $Max(loss)$ і очікувані втрати в точності $E(loss)$:

$$Max(loss) = \omega/2$$

$$E(loss) = \int_a^b x * \frac{1}{b-a} dx = \int_0^{\omega/2} x * \frac{1}{\omega/2 - 0} dx = \frac{\omega}{4}. \quad (3.1)$$

Виходячи з наведених вище рівнянь, можна побачити, що втрати для детектора обертання є незначними. Наприклад, коли два прямокутники зі співвідношенням сторін 1 : 9 відрізняються на 0.25° і 0.5° (очікувана за замовчуванням і максимальна втрата точності), то Intersection over Union (IoU) між ними зменшується лише на 0,02 і 0,05. Однак, one-hot мітки не підходять для таких передбачень кутів. Наприклад, якщо істинна мітка це 0°, а результати передбачення класифікатора дорівнюють 1° і -90° відповідно, їхні втрати передбачення будуть однаковими, але результати передбачення, які ближчі до істинної мітки, повинні бути більш допустимими з точки зору виявлення.

Тому CSL містить кругле кодування міток із періодичністю, і присвоєне значення мітки згладжується з певною похибкою (см. Рис. 3.2). Це дає змогу отримати більш стійкий кутовий прогноз через класифікацію. Вираз для CSL має такий вигляд [12]:

$$CSL(x) = \begin{cases} g(x), & \theta - r < x < \theta + r \\ 0, & \text{інакше} \end{cases}, \tag{3.2}$$

де *g(x)* - віконна функція, *r* - радіус віконної функції, θ представляє кут поточного обмежувального прямокутника. Ідеальна віконна функція *g(x)* повинна мати такі властивості:

- Періодичність: $g(x) = g(x + kT), k \in N$. $T = 180/\omega$ - кількість відрізків, на які розбивається кут, значення за замовчуванням - 180.

- Симетрія: $0 \le g(\theta + \varepsilon) = g(\theta - \varepsilon) \le 1$, $|\varepsilon| < r$. θ - центр симетрії.

- Максимум: $g(\theta) = 1$.

- Монотонність: $0 \le g(\theta \pm \varepsilon) = g(\theta \pm \varsigma) \le 1$, $|\varsigma| < |\varepsilon| < r$. Функція представляє монотонний незростаючий напрямок від центральної точки в обидві сторони.

На рисунку 3.2 показано функції, що відповідають вищеописаним умовам: імпульсні функції, прямокутні функції, трикутні функції та гаусові функції.

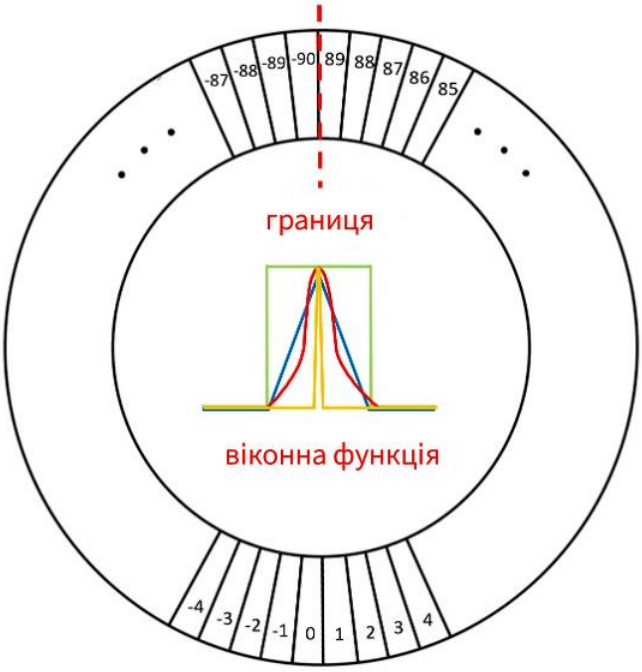

Рисунок 3.2 – CSL метод для кутової класифікації (адаптовано з [12])

Підбиваючи підсумки, метод CSL пропонує розв'язувати задачу повороту рамок як задачу класифікації, а не регресії, що добре поєднується з філософією одностадійних детекторів. Так само він пропонує для класифікації кутів використовувати кругле періодичне кодування міток, замість класичного, що дає змогу адаптивно оцінювати втрати кутів і призводить до коректнішої оцінки точності.

4 РОЗМІТКА НАБОРУ ДАНИХ

Розмітка даних, це необхідний етап у процесі навчання моделей машинного навчання, особливо в завданнях виявлення об'єктів. Розмічені дані відіграють ключову роль у визначенні якості та продуктивності моделі.

Для розмітки ми будемо використовувати інструмент анотування комп'ютерного зору CVAT(Computer Vision Annotation Tool). Це безкоштовний веб-інструмент з відкритим вихідним кодом для анотування зображень і відео, який використовується для маркування даних для алгоритмів комп'ютерного зору. CVAT автоматично і рівномірно добудовує покадрову розмітку між ручною розміткою, що сильно спрощує розмітку відефрагментів, з рівномірно рухомими об'єктами.

Для того щоб почати працювати в CVAT, його потрібно встановити на свій сервер або локальну машину. Для цього нам знадобиться ще 3 інструменти: Docker, WSL і Ubuntu.

WSL (Windows Subsystem for Linux) - це підмережа, що дає змогу використовувати Linux-специфічні інструменти та запускати Linux-додатки прямо на своєму комп'ютері на Windows.

Docker - програма, яка дає змогу упаковувати додатки в стандартизовані контейнери. Контейнери Docker можуть бути запущені на будь-якій машині, що підтримує Docker, і забезпечують ізольоване та відтворюване оточення для додатків. Це полегшує розгортання, масштабування та управління додатками.

Ubuntu - популярний дистрибутивів Linux, який використовується як на серверах, так і на настільних комп'ютерах.

Таким чином, працюючи на системі Windows, ми встановили WSL, вибрали Ubuntu як дистрибутив Linux і встановили Docker всередині цього середовища Ubuntu. Після скачали Docker-образ CVAT і запустили його всередині Docker. Це дозволить нам використовувати CVAT.

Після встановлення та запуску CVAT, можна почати створювати нові завдання анотації. Завдання - це набір зображень або відео, які потрібно розмітити.

В інтерфейсі CVAT ми завантажуємо свої зображення або відео, задаємо параметри анотації та вказуємо, які типи об'єктів потрібно анотувати. У нашому випадку, для розмітки були взяті 10 різних відео, зняті з дронів, отримані з відкритих джерел. Інтерфейс CVAT-а і приклад налаштованих анотацій можна побачити на рисунках 4.1 та 4.2 відповідно

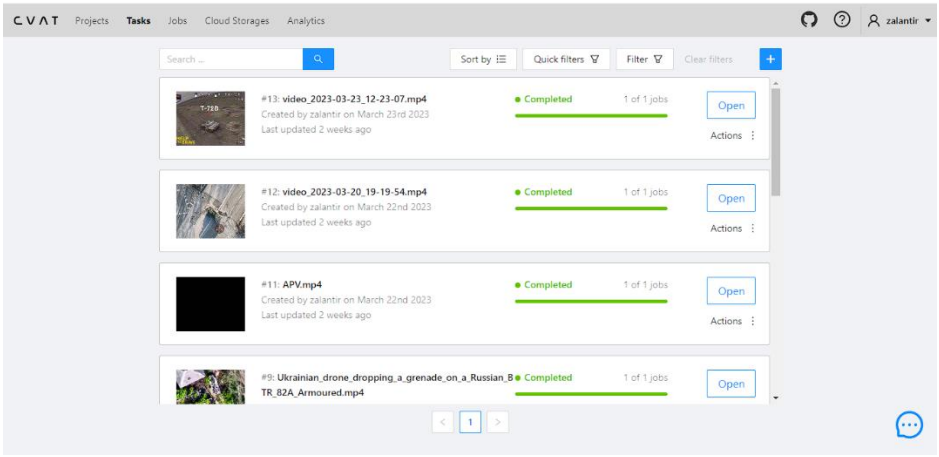

Рисунок 4.1 – Інтерфейс CVAT з поставленими завданнями

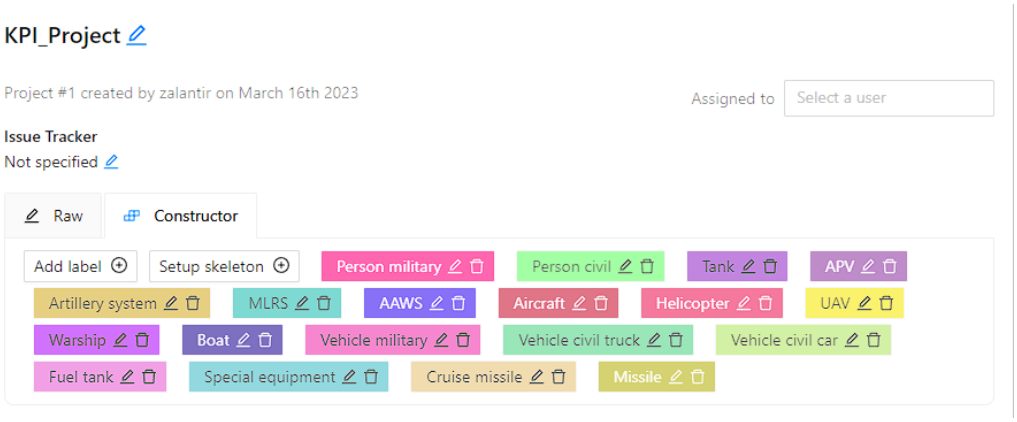

Рисунок 4.2 – Приклад налаштованих анотацій

Сам процес анотації включає в себе визначення та маркування об'єктів інтересу на зображеннях і відеофрагментах. Це включає класифікацію об'єктів, визначення їхніх меж і розташування, а також, у нашому випадку, визначення їхньої орієнтації.

Для анотації ми використовуємо OBB (Oriented Bounding Box), тому що такий вид рамок можна обертати, створюючи довільну орієнтацію, що критично важливо для нашого завдання.

Для підвищення точності подальшого навчання, у процесі розмітки так само важливо розділяти об'єкти на перекриті (рис. 4.3) і не перекриті (рис. 4.4). Чіткий поділ між перекритими і не перекритими об'єктами дає змогу моделі краще навчитися на таких складних випадках, коли два об'єкти перекривають один одного, або якщо частина об'єкта схована за кадром чи іншою перешкодою.

Рисунок 4.3 – Приклад розмітки не перекритого об'єкта (Джерело: відео "ЗАЛІЗНИЙ КАПУТ" у телеграм, канал БУАР110 ОМБр, модифіковано автором)

Рисунок 4.4 – Приклад розмітки перекритого об'єкта (Джерело: відео "Знищення окупантів причетних до Бучанської трагедії" на YouTube, канал Division GROUP, модифіковано автором)

Після того, як анотацію завершено, можно експортувати анотації в різних форматах, таких як XML, JSON, CSV та інших форматах, підтримуваних різними бібліотеками комп'ютерного зору.

У створеного нами набору даних є кілька проблем, які роблять його не оптимальним рішенням для поставленого завдання.

Для навчання моделі виявлення об'єктів з повітря, з довільно-орієнтованою камерою, потрібна більш різноманітна і збалансована вибірка кадрів, з різними кутами повороту камери, ніж та вибірка, яку було створено на основі знайдених в інтернеті записів.

Матеріал, взятий для розмітки, не був призначений для створення анотацій, і якість самих відео дуже низька, присутні спотворення, а також деякі з них позначені водним знаком, як на малюнку 4.3. При цьому найважливіше в набору даних - це якість самих даних. Неважливо, як багато даних зібрано, якщо вони поганої якості, модель не працюватиме ефективно. Якісні дані - це дані без шуму, спотворень і з мінімальною кількістю помилок.

Дані мають репрезентативно відображати проблему. У моєму випадку, це зйомка з повітря, з різних кутів і перспектив. Тому для свого завдання я вибрав готовий набір даних розмітки з дрона, знайдений на площадці kaggle.com.

Обраний набір даних, під назвою VSAI [10], являє собою багаторакурсний набір даних для виявлення транспортних засобів в різних перспективах за допомогою аерофотознімків.

Цей набір даних був розроблений командою фахівців з коледжу аерокосмічних наук та інженерії міста Чанша, якраз для дослідницьких робіт у цій галузі.

На вибір цього набору даних вплинуло кілька факторів.

По-перше, це комплексний і повний набір даних зі специфікою завдання, орієнтований на довільну зйомку з повітря. Більшість попередніх відкритих наборів даних подібної тематики містили або строго горизонтальну, або строго

вертикальну зйомку. Так само існують набори даних з розміткою під фіксованими кутами невеликої вибірки, які не підходять за специфікою завдання.

По-друге, це дуже складний набор даних, що містить близько 9000 зображень з анотаціями, з великою кількістю негативних або складних прикладів. Складність прикладів, крім багаторакурсного знімання, зумовлюється різною висотою знімання (від 50 до 500м), різними погодними умовами, різним рівнем освітлення, різною щільністю анотованих об'єктів і різним масштабом цих об'єктів (деякі анотації містять критично малі об'єкти розміром до 5 пікселів). Усе це дає змогу навчити та протестувати модель на складних природних сценах.

По-третє, вибірка класів у цьому наборі даних складається тільки з двох класів – маленький транспорт і великий транспорт. Таке узагальнення дає змогу сконцентруватися на самому завданні розпізнавання об'єктів з повітря, не заглиблюючись у специфіку самої класифікації.

5 Розробка моделі для виявлення об'єктів із повітря

5.1 Робочий простір

Як середовище розробки було обрано сервіс Google Colaboratory, що є безкоштовним інтерактивним хмарним середовищем для роботи з кодом на мові Python від Google у браузері. Це середовище дає змогу використовувати блокноти Colab (див. рис. 5.1), які ґрунтуються на Jupyter Notebook, що дає змогу модульно писати код усередині одного блокнота.

Це середовище добре підходить для машинного навчання, у ньому вже встановлено багато необхідних бібліотек для цієї задачі. Головна особливість цього сервісу в тому, що під час роботи в ньому можна використовувати серверне віддалене робоче середовище, в якому користувачеві виділяють місце, оперативну пам'ять, центральний процесор, а за необхідності можна під'єднати графічний процесор, для розв'язання складних завдань (див. рис. 5.2).

Так само цей сервіс по суті дає можливість працювати на системі Linux водночас працюючи на Windows, навіть не підключаючи віртуальну машину. Останньою важливою особливістю є можливість працювати на різних пристроях, незалежно від їхньої продуктивності

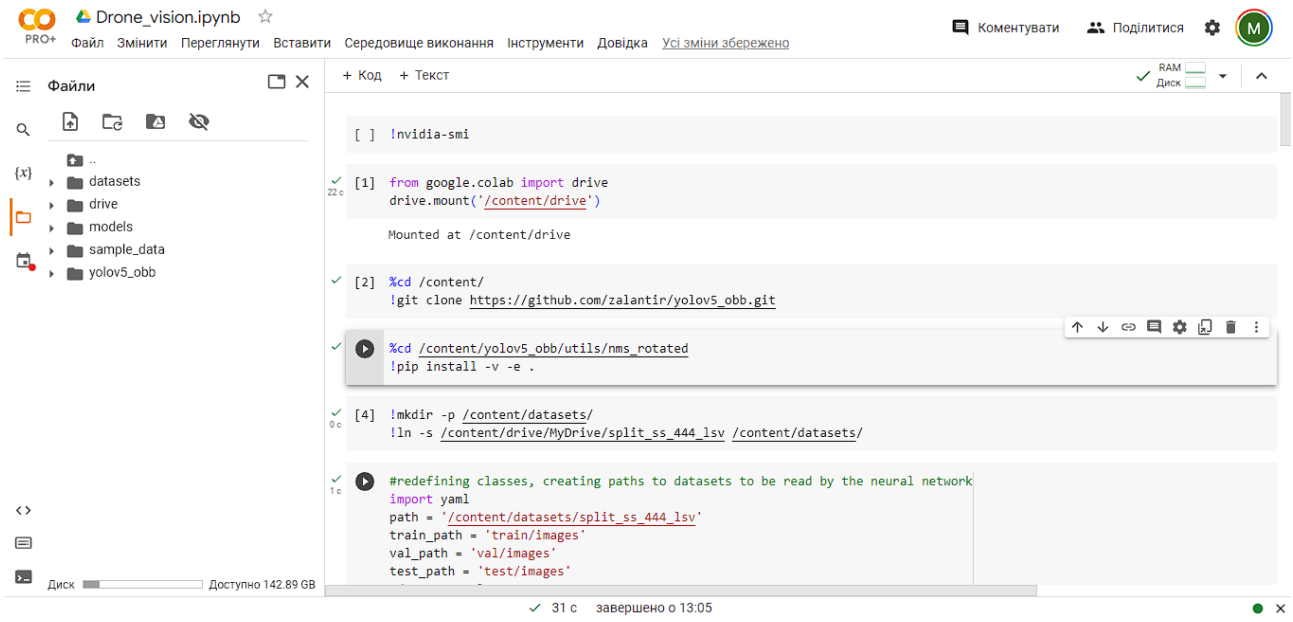

Рисунок 5.1 - Робочий блокнот Colab

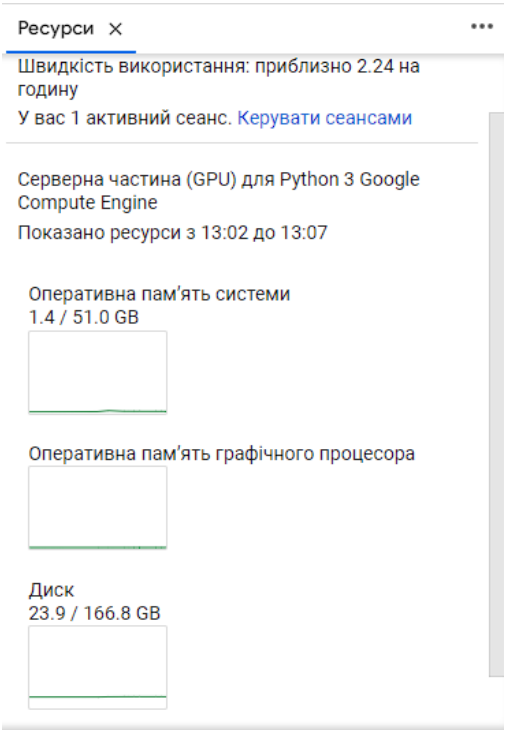

Рисунок 5.2 - Виділені ресурси

5.2 Вибір моделі

Для початку нам необхідно вибрати переднавчену мережу, щоб донавчити її для нашого завдання. Такий метод навчання називається трансферним, і він дає

змогу ефективніше навчати нейромережу виконувати поставлені завдання суміжної або вужчої специфіки. У переднавченій мережі ваги розподілені не випадково, і більш наближені до бажаних ваг, ніж якби ми проводили навчання з нуля. Це дає змогу підвищити якість навчання і може поліпшити кінцевий результат.

Як переднавчену модель було обрано YOLOv5, з модифікацією на роботу з орієнтованою розміткою. Такий вибір було зроблено з кількох причин.

Такий вибір було зроблено з кількох причин. Архітектура YOLO дуже легка, щоб розв'язувати задачі в реальному часі. До того ж ця архітектура розвивалася.

Оригінальна архітектура була єдиною нейронною мережею, а в наступних версіях з'явилася основна мережа (спочатку darknet-19, потім darknet-51).

У YOLOv2 адаптували якірні рамки архітектури SSD, що дало змогу краще працювати з різними масштабами і пропорціями. Так само в цій версії ввели концепцію "багатозадачного навчання", що дає змогу навчатися на даних із неповними мітками (коли в набору даних є мітки класів, але немає розмітки, як у наборі даних ImageNet).

До версії YOLOv3, ця архітектура стала повністю згортковою, за аналогією з R-FCN, відмовившись від двох повнозв'язних шарів. Замість того щоб використовувати якорі одного розміру на всьому зображенні, в YOLOv3 було введено різні якорі на різних "масштабах" зображення, що допомогло ще поліпшити виявлення об'єктів різних розмірів.

У YOLOv4 було реалізовано кілька нових технік, як-от CSPDarknet53 [15], для ефективного вилучення ознак, PANet [15] і SAM [15] блоки, для ефективного опрацювання ознак різних масштабів.

Версія YOLOv5 ще більше оптимізована: покращили навчання і швидкість виведення порівняно з YOLOv4. Було введено різні методи аугментації даних під час навчання, як-от масштабування, обрізка, обертання тощо. Так само було адаптовано функцію Focal Loss.

Усі ці зміни зробили YOLO дуже зручною, гнучкою та універсальною системою виявлення об'єктів, яка переймала найкращі методи для одностадійних детекторів протягом свого розвитку.

На сьогодні існують декілька версій архітектур YOLOv5, які відрізняються кількістю шарів і кількістю параметрів. Основною мережею у нас виступає YOLOv5 medium, тому що має найкраще співвідношення точності та швидкості. Більш важкі моделі можуть сильно сповільнити обробку на слабких пристроях, а приріст точності буде дуже малий. У більш слабких моделей, при більшій швидкості дуже сильно падає точність результатів.

Модифікація для роботи з OBB рамками заснована на методі CSL. Цей метод добре підходить для архітектури YOLO, доповнюючи класифікацію об'єктів і класифікацію рамок ще класифікацією кута повороту рамок.

5.3 Налаштування моделі

Для початку ми завантажуємо репозиторій із самою архітектурою, набор даних із розміткою в локальне сховище середовища Colab і структуруємо директорії для коректної роботи коду. Після налаштовуємо залежності та вимоги для роботи з цією архітектурою.

Далі ми створюємо новий файл формату *.yaml*, у якому прописуємо кількість і назву наявних класів об'єктів у нашому наборі даних, а також прописуємо шляхи до кореневої папки набору даних та відносно цього шляху, також прописуємо шляхи до зображень із кожної вибірки: тренувальної, тестової та валідаційної.

Після цього приступаємо до налаштування тренувальної моделі. Для початку вказуємо шлях до .yaml файлу, щоб наша тренувальна модель могла працювати з нашим набором даних.

Далі визначаємо стартові ваги моделі. Для цієї задачі були викачані ваги цієї моделі, попередньо навченої на наборі даних вертикальної зйомки з повітря

DOTA. Вертикальне знімання з повітря є ближчою задачею для довільно-орієнтованого знімання з повітря, ніж горизонтальне знімання з землі, тому такі ваги можуть прискорити процес навчання і зробити фінальну модель точнішою.

Для тонкого налаштування гіперпараметрів потрібно провести низку випробувань із різними налаштуваннями, тому для початку ми виставимо їх, дотримуючись рекомендацій з налаштування моделей YOLOv5. Однак, ми все-таки зменшимо значення двох параметрів.

Перший з них - це параметр затухання ваг для регуляризації (weight decay). цей параметр допомагає запобігти перенавчанню моделі шляхом накладення штрафу на величину ваг. Він робить це шляхом додавання додаткового члена до функції втрат, який штрафує більші ваги. Його зменшення знизить вплив цього штрафу, що може призвести до більш швидкого навчання, але також може збільшити ризик перенавчання, оскільки модель буде менш обмежена у своїх вагах.

Другий параметр - це значення імпульсу для оптимізатора (momentum). Це параметр СГС, який прискорює збіжність. Він робить це шляхом врахування попередніх кроків у навчанні та "штовхає" оновлення ваги в напрямку стійких градієнтів. Зменшення цього параметра може зробити оновлення ваги менш агресивними та потенційно більш стабільними, але може уповільнити швидкість збіжності, оскільки кожен крок навчання буде "менш впевненим" у своєму напрямку.

Таким чином ці два параметри компенсують один одного, а їхнє взаємне зменшення потенційно дасть змогу якісним вагам ефективніше закріплюватися за умови акуратнішого просування до цих ваг.

Наприкінці виставляємо розмір батча і кількість епох. У нашому випадку це 2 і 30 відповідно. Маленький розмір батча дасть змогу ретельніше налаштовувати ваги, менше їх узагальнювати, це зробить навчання більш довгим, але результат буде більш акуратним. Велика кількість епох дасть змогу

ретельно опрацювати ваги моделі. У сумі зі зменшенням затухання ваги і імпульсу, це дасть нам плавне і акуратне навчання моделі.

Нижче наведено опис усіх гіперпараметрів, що використовуються для навчання моделі, а також їхні значення для першої спроби навчання.

lr0 *= 0.0032*: початкова швидкість навчання для оптимізатора.

lrf *= 0.12*: коефіцієнт кінцевої швидкості навчання за стратегією OneCycleLR [14] (*lr0* * *lrf*).

momentum *= 0.843:* імпульс для оптимізатора СГС, який допомагає прискорити СГС у відповідному напрямку і зменшити коливання; так само цей параметр використовується як параметр *beta1* під час використання оптимізатора adam (beta2 = 0.999 зафіксовано в коді).

weight_decay *= 0.00036*: це параметр регуляризації, який додається до функції втрат для запобігання перенавчання шляхом надання невеликого штрафу за великі ваги.

warmup_epochs *= 2.0*: це кількість епох, протягом яких швидкість навчання лінійно збільшується до параметра *lr0*; Такий підхід може допомогти моделі навчитися більш стабільно на початку.

warmup_momentum *= 0.5*: початкове значення імпульсу під час розігріву.

warmup_bias_lr = 0.05: швидкість навчання для зміщення під час розігріву.

box *= 0.05*: вага втрати обмежувальних рамок.

cls *= 0.5*: вага втрати класифікації.

cls_pw *= 1.0*: вага для позитивного класу у втраті класифікації.

theta *= 0.5*: вага втрати кута повороту в OBB.

theta_pw *= 1.0*: вага для позитивного класу у втраті кута.

obj *= 1.0*: вага втрати об'єкта.

obj_pw *= 1.0*: вага для позитивного класу у втраті об'єкта.

iou_t *= 0.2*: поріг IoU для відбору позитивних і негативних якірних рамок; цей параметр визначає, які передбачення вважати позитивними, а які негативними під час навчання.

anchor_t *= 4.0*: поріг IoU для прив'язки якірних рамок; цей параметр визначає, які передбачення вважати пов'язаними з конкретними анкорами.

fl_gamma *= 0.0*: параметр Focal Loss.

hsv_h *= 0.015,* ***hsv_s*** *= 0.7,* ***hsv_v*** *= 0.4*: параметри для випадкової корекції кольору зображення.

degrees *= 180.0*: максимальний кут випадкового повороту зображення.

translate *= 0.1*: максимальний випадковий зсув зображення.

scale *= 0.25*: максимальний коефіцієнт випадкового масштабування зображення.

shear *= 0.0*: максимальний кут випадкового зсуву зображення.

perspective *= 0.0*: максимальний ступінь випадкового перспективного спотворення зображення.

flipud *= 0.5*: ймовірність виконання відображення зображення по вертикалі.

fliplr *= 0.5*: ймовірність виконання відображення зображення по горизонталі.

mosaic *= 0.75*: ймовірність об'єднання чотирьох різних зображень в одне, створюючи квадратний "мозаїчний" патерн.

mixup *= 0.1*: ймовірність створення нових навчальних прикладів шляхом виконання лінійного поєднання зображень та їхніх міток.

copy_paste *= 0.0*: ймовірність копіювання об'єктів з одного зображення і вставки їх в інше.

cls_theta *= 180*: діапазон випадкової корекції кута повороту зображення.

csl_radius *= 2.0*: радіус для CSL, визначає область навколо центру об'єкта, у межах якої передбачення вважаються позитивними.

5.4 Навчання моделі

У процесі навчання моделі виникла ще одна проблема. Деякі взаємопов'язані параметри не могли зв'язатися, бо частина з них перебували в сховищі блокнота Colab, а частина в самому графічному процесорі. Ця проблема виникає через специфіку самого середовища, в якому сховище даних і графічний процесор можуть перебувати в різних місцях. Для розв'язання цієї проблеми код було модернізовано так, щоб потрібні параметри переадресовувалися на пристрій, де вони необхідні.

Після завершення 30 епох, ми запускаємо процес валідації, для перевірки результатів на цьому етапі. Після закінчення валідації ми отримали метрики втрат, метрики точності та повноти.

Box Loss (втрати рамок): це втрати, пов'язані з передбаченням координат обмежувальних рамок об'єкта;

Theta Loss (втрати кута повороту): це втрати, пов'язані з передбаченням кута повороту обмежувальних рамок.

Cls Loss (втрати класифікації): це втрати, пов'язані з визначенням класу об'єкта.

Obj Loss (втрати об'єкта): це втрати, пов'язані з визначенням імовірності наявності об'єкта в певній рамці;

Precision (точність) моделі: це метрика, що показує, як багато з об'єктів, виявлених моделлю, були правильно виявлені.

Recall (повнота) моделі: це метрика, що показує, як багато з усіх реальних об'єктів були виявлені моделлю.

Ілюстрацію метрик втрат можна побачити на рис. 5.3, а ілюстрацію точності та повноти - на рис. 5.4.

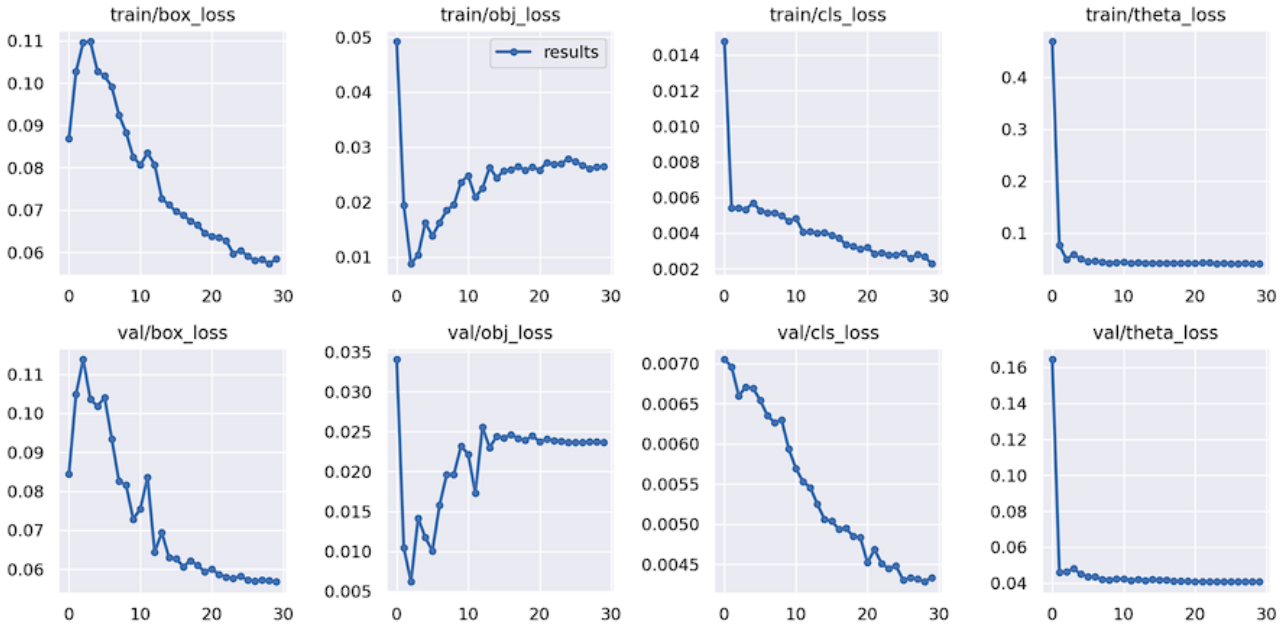

Рисунок 5.3 – Метрики втрат на тренувальному (зверху) і валідаційному (знизу) наборі даних.

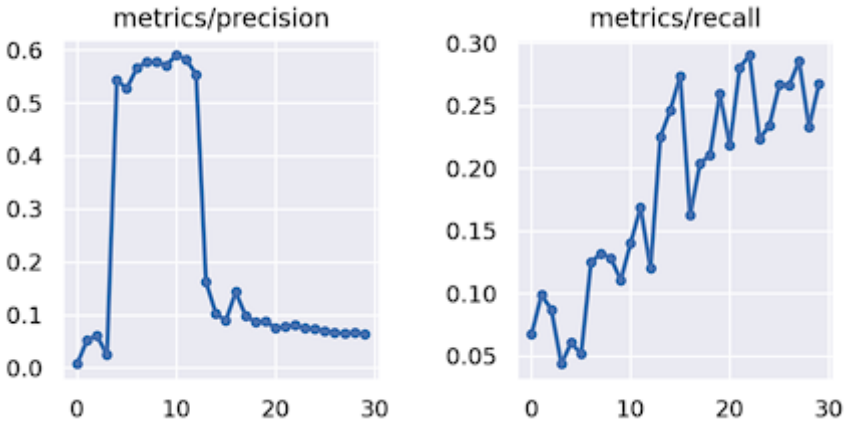

Рисунок 5.4 – Метрики точності (зліва) та повноти (справа)

З отриманих метрик можна зробити такі висновки.

Падіння втрат рамок означає, що модель стає дедалі кращою у визначенні точних координат обмежувальних рамок для кожного об'єкта.

Падіння втрат кута повороту вказує на поліпшення моделі у визначенні коректного кута орієнтації рамки. У нашому випадку вони різко впали на перших епохах, і майже не прогресувала далі, що може означати, що модель доволі

швидко адаптувалася до цієї задачі і досягла стелі продуктивності в цьому напрямку.

Падіння втрат класифікації означає, що модель покращує свою здатність правильно класифікувати об'єкти.

У нашому випадку різке падіння втрат за класифікацією на тренувальних даних на початку навчання зазвичай свідчить про те, що модель швидко поліпшується у своїй здатності класифікувати об'єкти. Поступове зниження після цього вказує на те, що модель продовжує поліпшуватися, але вже менш драматично.

При цьому на валідаційних даних ця метрика знижується рівномірно. Сам факт рівномірного зниження на валідаційному наборі свідчить про те, що модель не перенавчається і покращує свою продуктивність на даних, які вона не бачила під час навчання.

Падіння втрат на об'єктах вказує на те, що модель покращує свою здатність правильно визначати наявність або відсутність об'єктів у рамках. У нашому випадку цей параметр швидко впав на початку і почав зростати в процесі, стабілізуючись на високих значеннях. Швидке падіння свідчить про те, що модель швидко навчилася базових навичок виявлення об'єктів. А подальше зростання може вказувати на декілька проблем: перенавчання, недонавчання, або неоптимальне налаштування гіперпараметрів, наприклад, занадто висока швидкість навчання, через що модель "перестрибує" оптимальні значення ваг.

Точність різко збільшується і так само різко знижується після 10 епохи, тоді як повнота поступово зростає. Зі збільшенням кількості виявлених об'єктів падає точність їхньої класифікації.

У підсумку, у нас дуже низькі показники точності та щільності, і при цьому майже всі втрати знижуються, до того ж графіки зниження втрат на валідаційному наборі та на тренувальному дуже схожі, що свідчить про те, що модель успішно адаптується і навчається на нових даних, а не вгадує їх. Різкі зміни на графіках можуть свідчити про занадто агресивну зміну ваг. Ця система

поки що погано справляється з виявленням і класифікацією об'єктів. У районі 10 епохи модель виявляла всього 15% від усіх анотованих об'єктів і класифікувала їх з точністю 60%. А в районі 22 епохи вона досягла максимального значення в 30% від усіх об'єктів, але вірно класифікувала менше 10%. Далі проведемо ще пару досліджень для уточнення результатів.

5.5 Експерименти

Для початку простежимо як погане налаштування параметрів може негативно вплинути на хід навчання.

Виставимо розмір батча 16 для узагальнення даних. При використанні великого батча для оновлення градієнтів ваг ми фактично беремо середнє значення градієнта від великої кількості зразків. І хоча це може дати більш "стабільне" і "точне" оцінювання градієнта, модель може просто оптимізуватися для середнього випадку та ігнорувати окремі, аномальні випадки.

Виставимо кількість епох 200, щоб простежити за перенавчанням моделі.

Важливо розуміти, що значення втрат не коректно відображають прогресування моделі, іноді вони можуть вказувати на погано підібрані гіперпараметри

Для прикладу, параметри *iou_t* і *anchor_t* зробимо рівними 0.5, так само активуємо focal loss виставивши параметр *fl_gamma* = 2. Таким чином, збільшуючи *iou_t* і *fl_gamma* та зменшуючи *anchor_t*, ми встановлюємо вищі вимоги для призначення позитивних прикладів, а також збільшуємо внесок важкокласифікованих прикладів. Результуючі метрики зображені на рисунках 5.6 та 5.6.

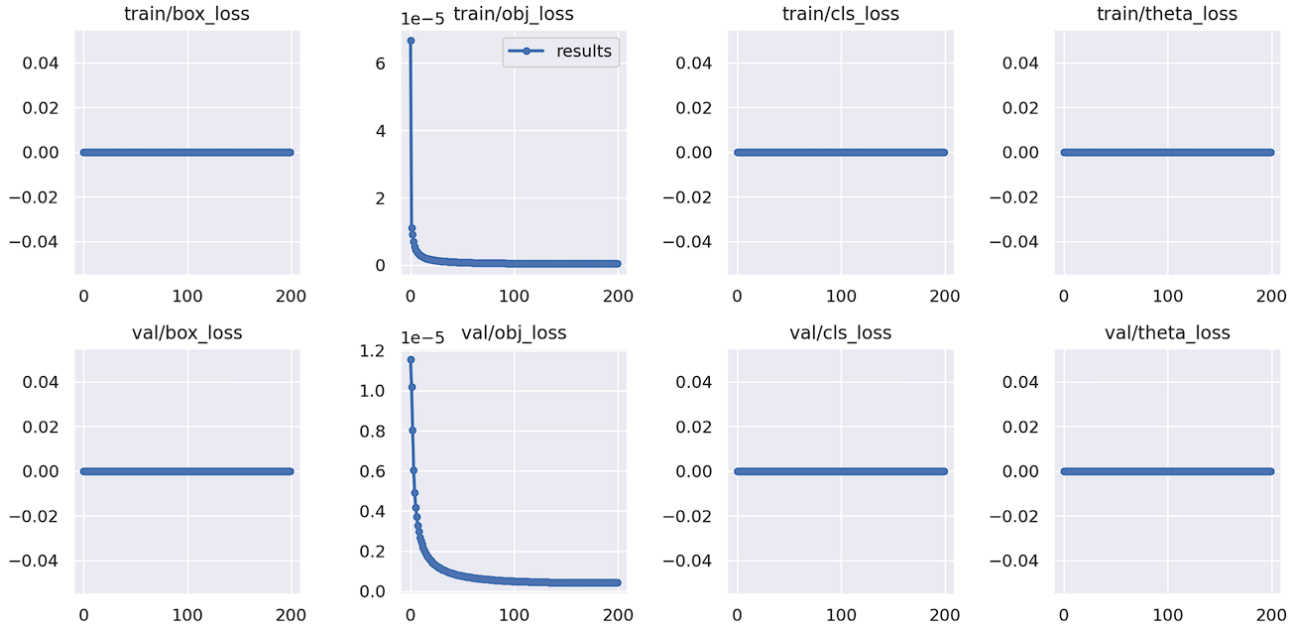

Рисунок 5.5 – втрати 2 спроби навчання

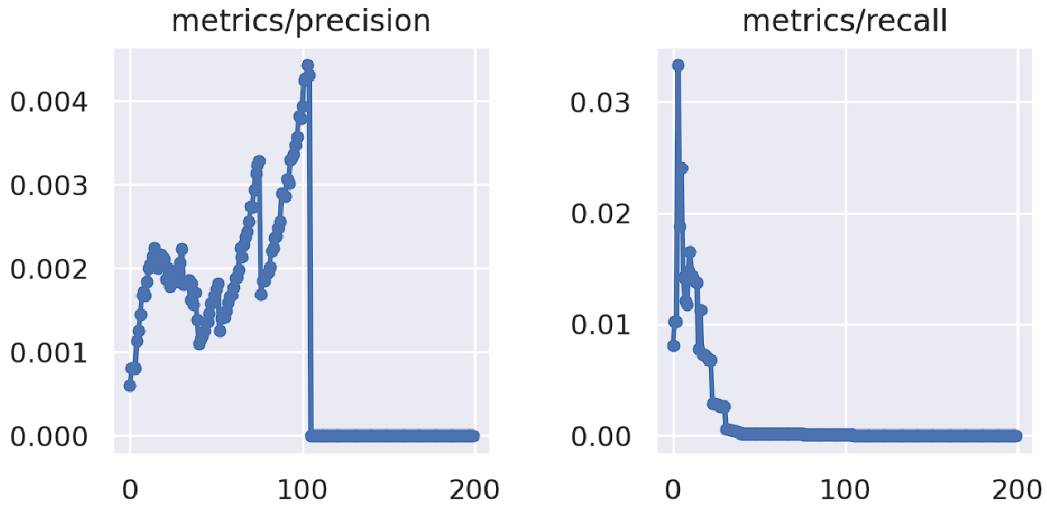

Рисунок 5.6 – точність і повнота 2 спроби навчання

Як ми бачимо, втрати рамок, класифікації та повороту стабільно залишаються на нулі протягом усього навчання, що є аномальним, а втрати об'єктів набагато вищі на початку, якщо порівнювати з 1 спробою навчання, але швидко знижуються і прагнуть до нуля.

Точність дуже низька, і хоча вона поступово зростає, після 100 епохи вона падає в нуль і залишається так до кінця навчання. Точність після 100 епохи впала в нуль, тому що повнота після 100 епохи теж стала дорівнювати нулю, і модель просто перестала виявляти хоч якісь об'єкти. До цього повнота так само перебуває на дуже низьких значеннях.

Таким чином, коригувати гіперпараметри потрібно акуратно, бажано не роблячи занадто різких змін.

У наступній спробі навчання спробуємо вибрати більш компромісну стратегію, навчаючи модель упродовж 30 епох, з розміром батча 8.

У попередніх спробах ми використовували СГС у ролі оптимізатора, тепер поставимо замість нього Adam. Він має більшу адаптивність, автоматично налаштовує швидкість навчання для кожного параметра на основі оцінок першого (beta1) і другого (beta2) імпульсу градієнтів. Так само Adam менш чутливий до вибору початкової швидкості навчання та інших гіперпараметрів, що спрощує налаштування моделі. Виставимо значення *beta1 (momentum)* = 0.9, а *beta2* = 0.999, виходячи з рекомендацій[13].

Спробуємо збільшити затухання градієнта до 0.005, а поріг IOU виставимо рівним 0.3, трохи збільшуючи його порівняно з першою спробою навчання. Параметр *anchor_t* зменшимо з 4 до 2 порівняно з першою спробою навчання, і активуємо функцію focal loss на значенні 1, щоб приділяти більше уваги прикладам, які важко класифікувати. Таке налаштування зробить критерії трохи суворішими, але не надто порівняно з 2 спробою.

У процесі навчання стався збій серверів, і модель не встигла навчитися повністю. Збереглися тільки ваги моделі в найкращому варіанті та в останньому варіанті. Ми взяли найкращий варіант ваг, які були на 15 епосі, і донавчили ще на 10 епохах, отримавши метрики для 10 останніх епох, зображених на рисунках 5.7 і 5.8.

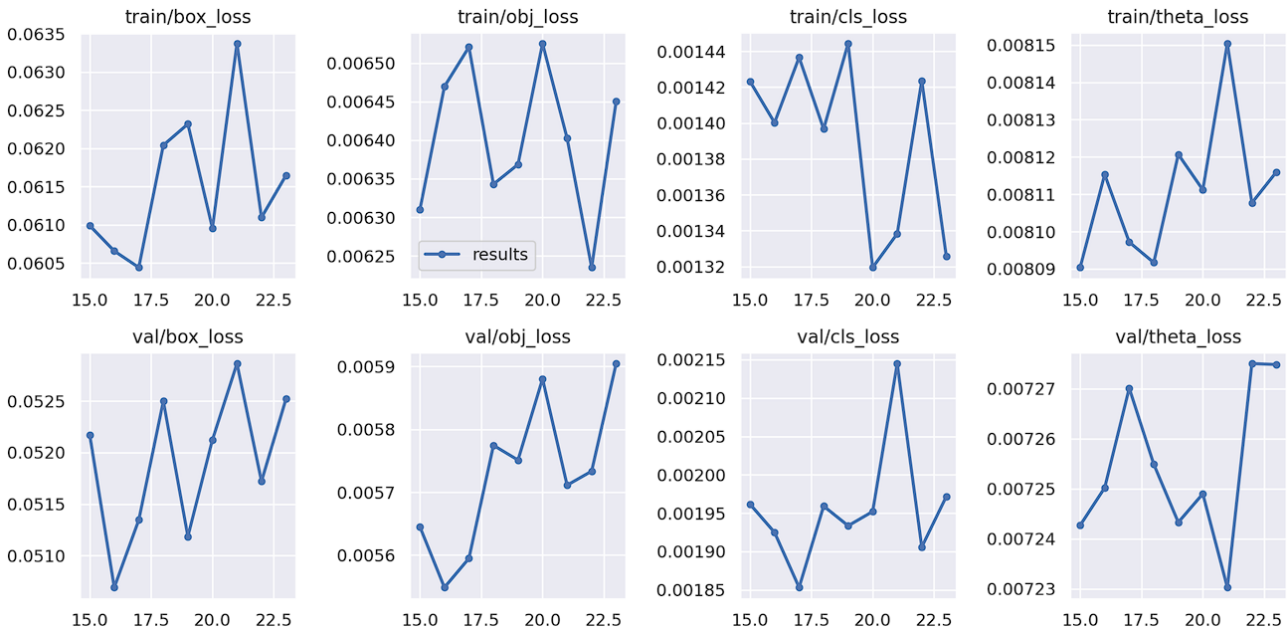

Рисунок 5.7 – втрати 3 спроби навчання

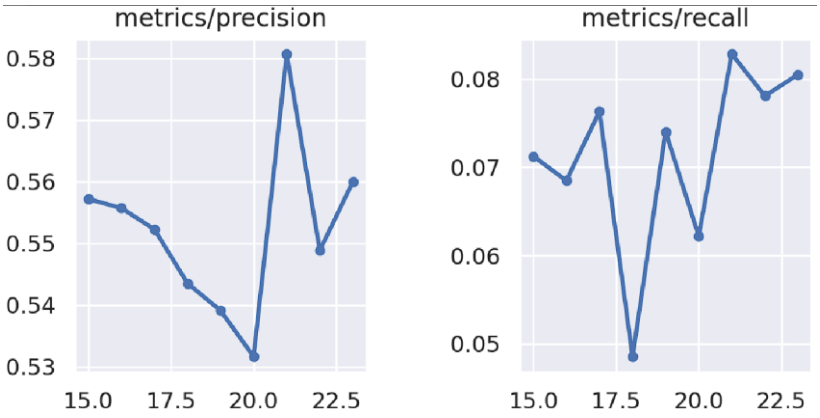

Рисунок 5.8 – точність і повнота 3 спроби навчання

Через збій, що стався, у нас не вистачає даних, щоб повноцінно оцінити, як розвивалася модель протягом усіх епох навчання. Однак, за останніми 10 епохами можна судити, що критичних поліпшень не відбулося, модель скоріше стала навіть гірше справлятися із завданням. Трохи впала точність порівняно з першою спробою, і дуже сильно впала повнота порівняно з першою спробою (з 0.3 на піку до 0.08 на піку). Втрати не сильно змінилися, якщо судити за 10 останніми епохами в першій спробі. Однак, зросли втрати об'єктів, а з ними і так були проблеми в першій спробі.

Попередні результати не перевищують певних значень точності та щільності, причиною чого може бути застрягання в локальних мінімумах і недостатня кількість епох для розвитку моделі. Перевіримо це виставивши такі параметри.

Розмір батча виставимо 2, для акуратної зміни ваг, кількість епох виставимо 100, щоб простежити за моделлю на більшій дистанції. Початкову швидкість навчання збільшимо до 0.0044, а коефіцієнт кінцевої виставимо 0.5. Вимкнемо параметри розігріву, виставивши *warmup_epochs* = 0. *iou_t* і *anchor_t* повернемо до початкових значень 0.2 і 4 відповідно, тому що їхня зміна сильно впливає на процес навчання, але до цього не показувала позитивних результатів. Так само зменшимо кут повороту зображення зі 180° до 45°, тому що наш набір даних сам по собі багаторакурсний і не дуже потребує додаткової аугментації. Так само я збільшив параметр scale з 0.25 до 0.75, це може дати змогу моделі краще справлятися з різними масштабами об'єктів у нашому наборі даних. Отримані метрики зображено на рисунках 5.9 та 5.10.

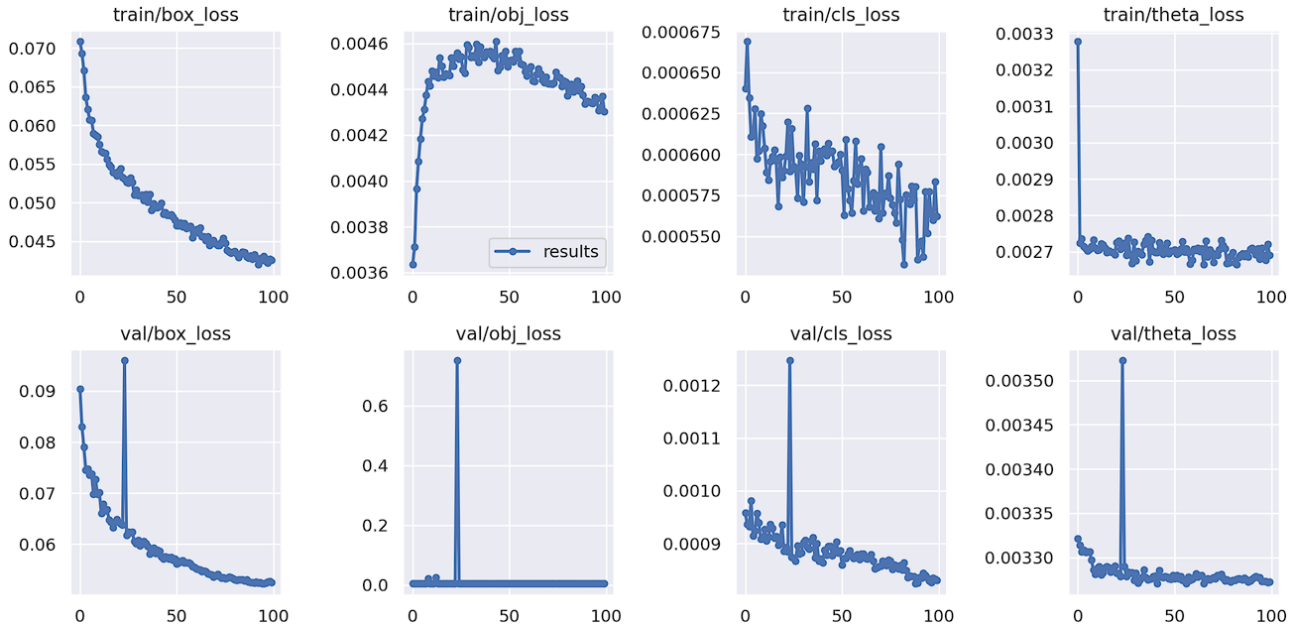

Рисунок 5.9 – втрати 4 спроби навчання

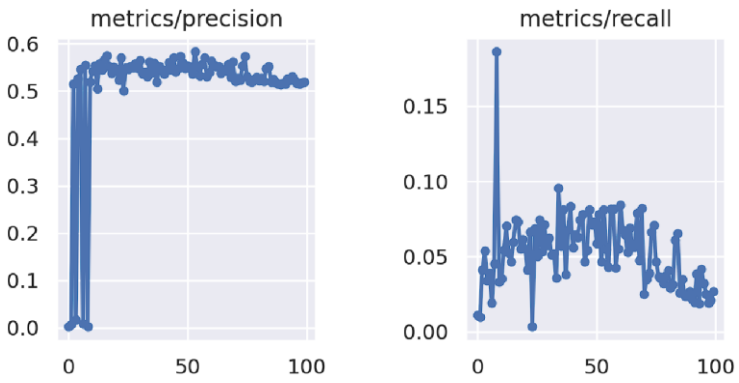

Рисунок 10 – точність і повнота 4 спроби навчання

З результатів видно, що поліпшити ефективність моделі таким чином не вийшло. Понад те, повнота впала ще більше, через те що ми взяли занадто великий коефіцієнт масштабування. Найімовірніше, з цієї ж причини втрати об'єктів на валідаційному наборі майже нульові, тому що в більшості випадків, модель не справляється з таким великим розкидом масштабів.

Проведемо фінальний експеримент для підбиття підсумків за нашою моделлю. Замінимо Adam назад на СГС. Це рішення зумовлене тим, що Adam не показав поліпшень результатів, а СГС є більш перевіреним і, як наслідок, більш надійним методом.

Сильна зміна масштабування негативно позначилася на попередніх результатах. Пов'язано це з тим, що використовуваний набір даних дуже складний і різноманітний у своїх масштабах, ракурсах і умовах, через що додаткова аугментація може сильно ускладнювати його навчання. Спробуємо вимкнути всю аугментацію, зменшивши такі параметри до нуля: *degrees, translate, scale, flipud, fliplr, mosaic, mixup*.

Для компенсації відключеної аугментації, збільшимо узагальнювальну здатність моделі, збільшивши розмір батча з 2 до 4. Кількість епох залишимо 100.

Знизимо початкову швидкість навчання в 10 разів, для максимально обережної роботи з вагами, коефіцієнт кінцевої швидкості навчання залишимо 0.5. Імпульс збільшимо до 0.937. За такої низької швидкості навчання, підвищене значення імпульсу дасть змогу навчатися трохи швидше, коли ваги змінюються

в одному напрямку впродовж безлічі ітерацій. Отримані метрики зображено на рисунках 5.11 та 5.12.

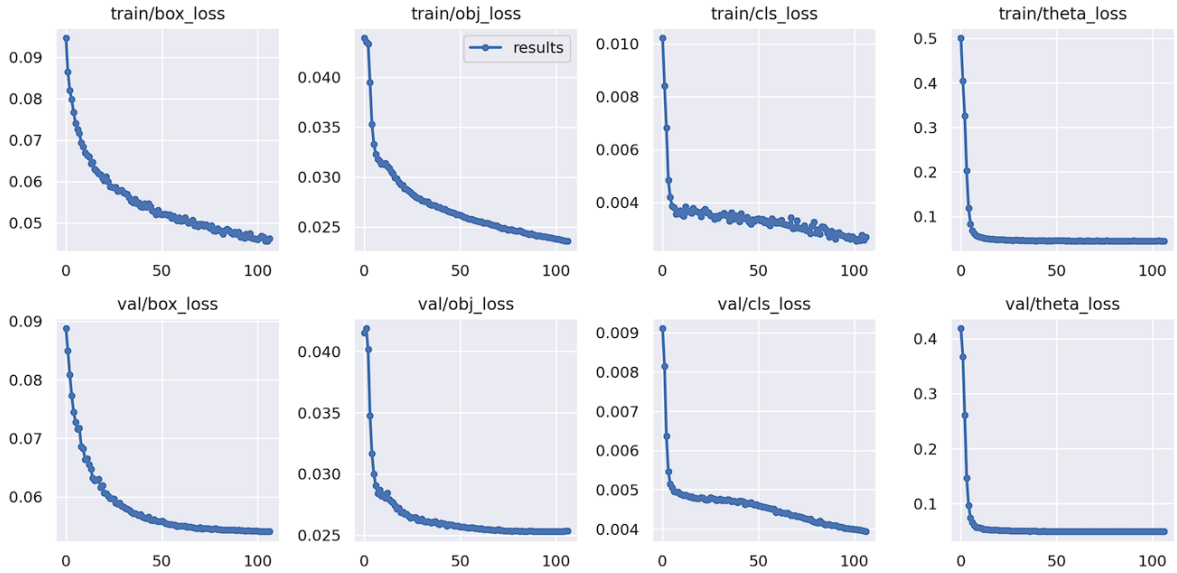

Рисунок 5.11 – втрати 5 спроби навчання

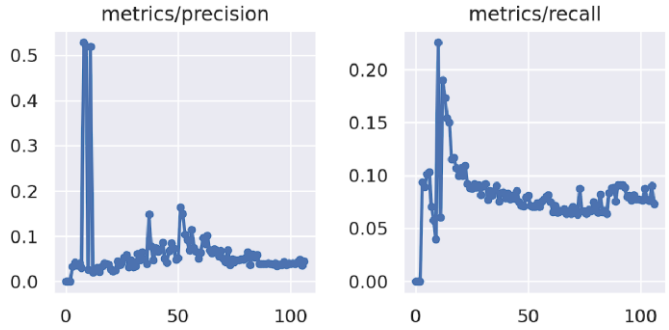

Рисунок 5.12 – точність і повнота 5 спроби навчання

За цими метриками видно, що модель досить швидко навчається до певного етапу, після чого не прогресує. Найкращі результати було отримано під час першої спроби навчання, тому їх буде взято за основу під час тестування.

5.6 Тестування

Під час тестування використовувався графічний процесор NVIDIA Tesla T4 з 16 гб об'ємом відеопам'яті. Результати тестування показано в таблиці 5.1.

Таблиця 5.1 – Результати тестування

Клас об'єкта	Кількість зображень	Кількість об'єктів	Точність	Повнота	mAP.5	mAP.95
1. Загалом	1520	7122	0.135	0.276	0.0691	0.0157
2. Малий транспорт	1520	6870	0.116	0.512	0.122	0.0261
3. Великий транспорт	1520	252	0.154	0.0397	0.0166	0.00522

Точність для обох класів дуже низька. Загалом, як буде видно на рисунку 5.13, точність класифікації виявлених об'єктів висока (червоними рамками позначається клас малого транспорту, зеленими - великого транспорту). Однак ми використовуємо архітектуру YOLO разом із методом CSL, які роблять виявлення рамок і кута їхнього повороту теж завданням класифікації. Параметр точності узагальнює точність класифікації об'єктів, їхніх рамок і повороту цих рамок, і модель найгірше справляється саме з класифікацією поворотів, що так само видно на рисунку 5.13.

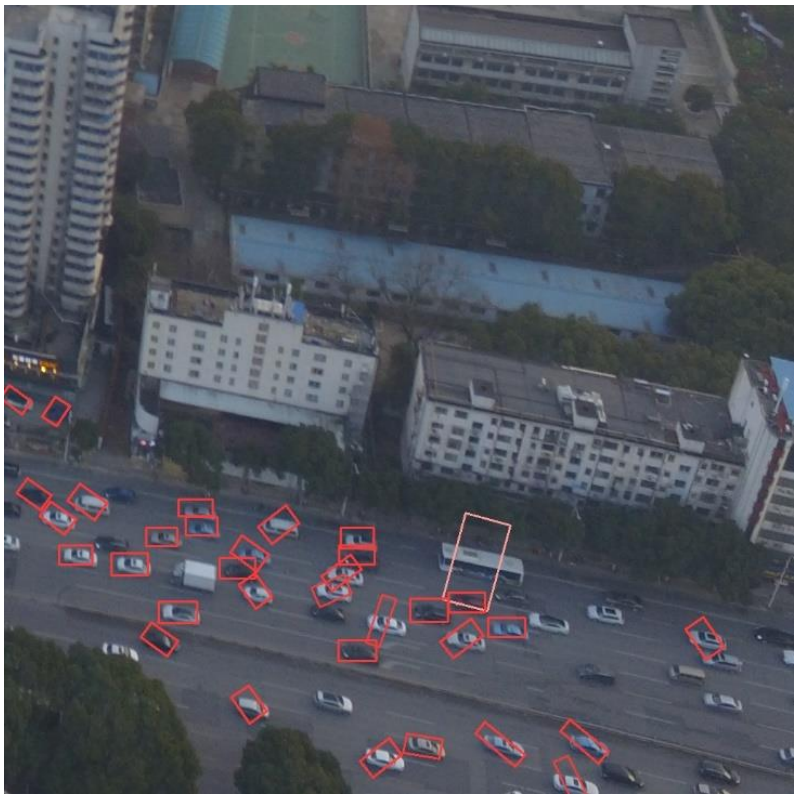

Рисунок 5.13 – Приклад точності виявлення (Адаптовано з: Kuggle. 2022. VSAI_Dataset)

Модель змогла виявити більше половини всього малого транспорту, і лише 4% великого транспорту. Такий розкид пов'язаний з дуже малою кількістю розміченого великого транспорту (2193 в усьому наборі даних), і незбалансованістю порівняно з кількістю малого транспорту (47,519 в усьому наборі даних). Загалом, модель здатна непогано впоратися із зображеннями, з великою кількістю об'єктів, приклад чого зображений на рисунку 5.14.

Рисунок 5.14 – Приклад повноти виявлення (Адаптовано з: Kuggle. 2022. VSAI_Dataset)

mAP з упевненістю моделі в 50% і з упевненістю моделі в 95% дуже низькі. За цими значеннями можна ще раз побачити, що модель найкраще справляється змалим транспортом, зі значенням mAP.5 = 12%.

Так само було помічено, що модель краще справляється з віддаленими від камери об'єктами, і чим ближче об'єкт до камери, тим гірше вона його розпізнає, приклад чого показано на малюнках 5.15 і 5.16. Це може бути пов'язано з тим, що обраний набір даних зосереджений на більш віддалених від камери сценах, і гірше налаштовується на наближення камери до об'єкта.

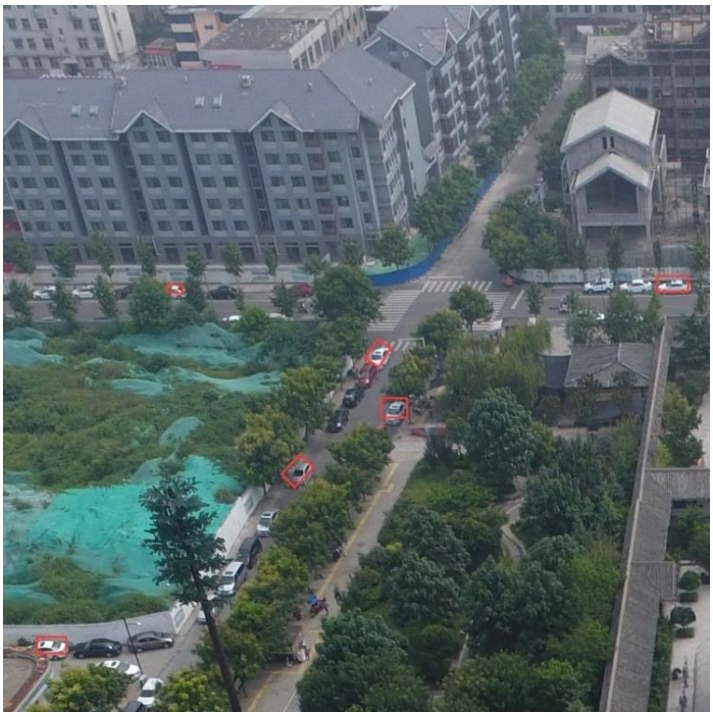

Рисунок 5.15 – Приклад виявлення на середній дистанції від об'єкта (Адаптовано з: Kuggle. 2022. VSAI_Dataset)

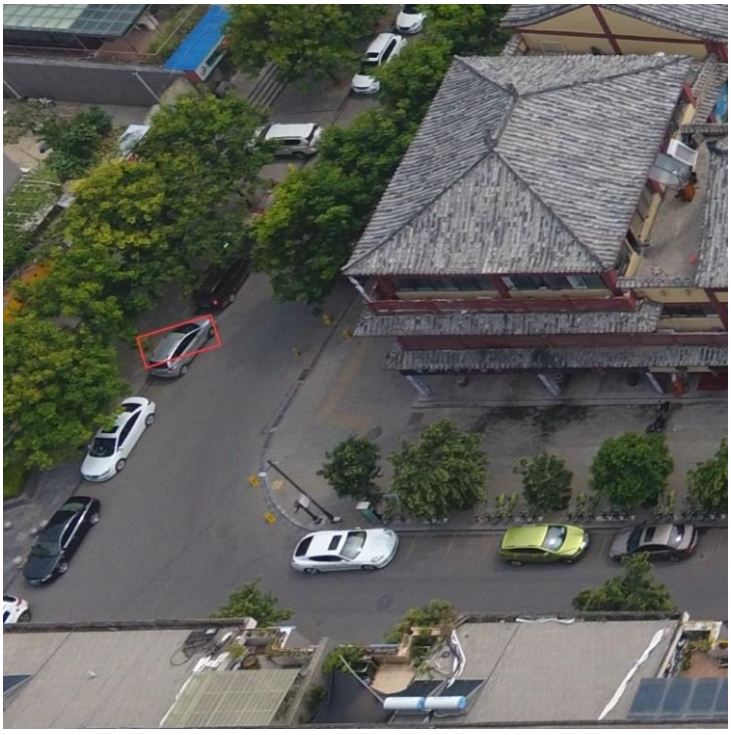

Рисунок 5.16 – Приклад виявлення на близькій дистанції від об'єкта (Адаптовано з: Kuggle. 2022. VSAI_Dataset)

Незважаючи на низькі показники ефективності, модель здатна розпізнавати перекриті об'єкти, приклад чого зображений на рисунку 5.17.

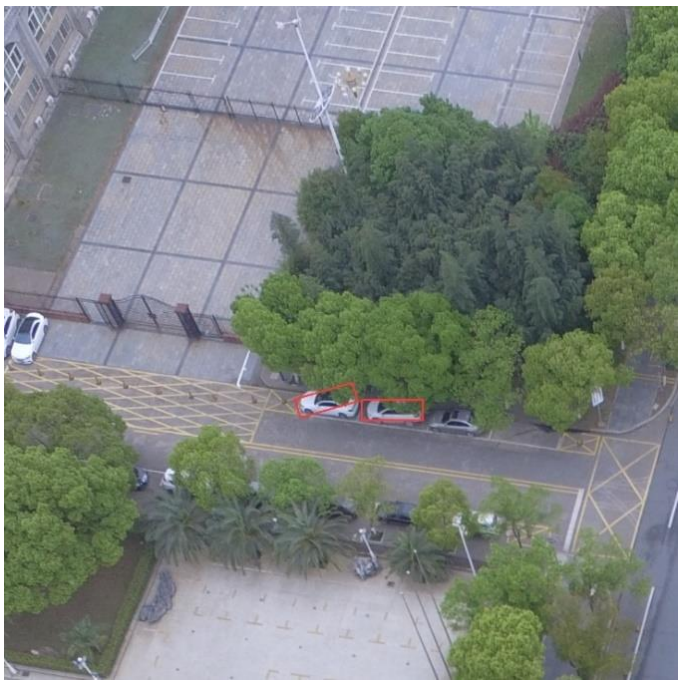

Рисунок 5.17 – Приклад виявлення перекритих об'єктів (Адаптовано з: Kuggle. 2022. VSAI_Dataset)

Отримана модель здатна працювати в режимі реального часу. Час опрацювання кожного зображення становить приблизно 44 мс, що майже дорівнює 23 кадрам на секунду

У підсумку, у моделі все ще є серйозні проблеми з точністю і повнотою, однак, вона змогла адаптуватися до багаторакурсної зйомки.

Висновки

Провівши дослідження методів і алгоритмів виявлення об'єктів, було розроблено експериментальну модель детектора для багаторакурсного виявлення об'єктів із повітря. Під час експериментів було виявлено, що для такої задачі, модель навчається ефективніше при зниженні імпульсу і затухання ваг.

Отримана модель має посередні показники точності та повноти, і поки що не може використовуватися в практичних завданнях, однак, як було показано під час тестування, концептуально вона здатна впоратися із завданнями багаторакурсного виявлення, що робить доречним подальші дослідження в цій галузі для швидких, одностадійних детекторів, а зокрема, для архітектури YOLO.

ПЕРЕЛІК ПОСИЛАНЬ

1. Avinash Sharma V. Understanding Activation Functions in Neural Networks. Medium. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
2. Empirical Evaluation of Activation Functions in Deep Convolution Neural Network for Facial Expression Recognition / Khalid M. et al. 2020 43rd International Conference on Telecommunications and Signal Processing (TSP). 2020. P. 204–207
3. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Girshick, R., Donahue, J., Darrell, T., & Malik, J. 2014 IEEE Conference on Computer Vision and Pattern Recognition. 2014. P. 580-587
4. Girshick R. Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV). 2015. P. 1440-1448
5. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. / Ren S., He K., Girshick R., Sun J. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. Vol. 39, No. 6, P. 1137-1149
6. SSD: Single Shot MultiBox Detector / W. Liu et al. Computer Vision - ECCV 2016. 2016. p. 21-37
7. You Only Look Once: Unified, Real-Time Object Detection / Redmon J., Divvala S., Girshick R., Farhadi A. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. p. 779-788
8. R-FCN: Object Detection via Region-based Fully Convolutional Networks / Dai, J., Li, Y., He, K., Sun, J. Neural Information Processing Systems (NIPS). 2016. p. 379-387
9. Focal Loss for Dense Object Detection / T.-Y. Lin et al. IEEE Transactions on Pattern Analysis and Machine Intelligence (ICCV). 2020. vol. 42. no. 2. p. 318-327

10. VSAI: A Multi-View Dataset for Vehicle Detection in Complex Scenarios Using Aerial Images / J. Wang et al. Drones. 2022. vol. 6, no. 7. p. 161.
11. Learning Modulated Loss for Rotated Object Detection / Qian W., et al. Proceedings of the AAAI Conference on Artificial Intelligence. 2021. Vol. 35, no. 3. P. 2458–2466.
12. Yang X., Yan J. Arbitrary-Oriented Object Detection with Circular Smooth Label. Computer Vision – ECCV 2020. Cham, 2020. P. 677–694.
13. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv. 2017. URL: https://arxiv.org/abs/1412.6980
14. Smith L. N., Topin N. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. arXiv preprint arXiv. 2018. URL: https://arxiv.org/abs/1708.07120
15. Gouider C., Seddik H. YOLOv4 enhancement with efficient channel recalibration approach in CSPdarknet53. 2022 IEEE Information Technologies & Smart Industrial Systems (ITSIS), Paris, France, 15–17 July 2022. P. 1-6
16. Optimization for Medical Image Segmentation: Theory and Practice When Evaluating With Dice Score or Jaccard Index / Eelbode T., et al. IEEE Transactions on Medical Imaging. 2020. Vol. 39, no. 11. P. 3679 – 3690.

Додатки

1. Приклади роботи моделі на тестовій вибірці: https://bit.ly/43MmLfd
2. Усі спроби навчання: https://bit.ly/45U1ySb