КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Кваліфікаційна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз»

за спеціальністю 124 Системний аналіз

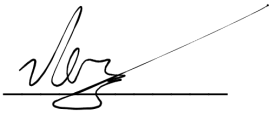
на тему:

Використання YOLO для задач розпізнавання і класифікації військової техніки

Виконав студент 4 курсу
Лазарєв Ігор Андрійович

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Шевчук Юлія Михайлівна

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



Роботу розглянуто й допущено до захисту на засіданні кафедри системного аналізу та теорії прийняття рішень

« 07 » 06 20 22 р., протокол № 10
Завідувач кафедри
Олександр НАКОНЕЧНИЙ

Київ – 2022

Зміст

Вступ

На сьогоднішній день технології досягли достатнього рівня, коли словосполучення «штучний інтелект» вже займає своє місце серед найбільш обговорюваних тем, а діяльність людини складно уявити без використання комп’ютерної техніки.

Виявлення об'єктів у зображеннях дистанційного зондування з високою роздільною здатністю є фундаментальною та складною проблемою в області дистанційного зондування. Аналіз зображень для цивільного та військового застосування через складне сусіднє середовище може викликати алгоритми розпізнавання, які помилково сприймають нерелевантні наземні об’єкти за цільові. Нейронна мережа глибокої згортки (DCNN) є відомою у темі виявлення завдяки його потужній здатності вилучення функцій і досягла найсучасніших результатів у Computer Vision. Загальний шлях виявлення об'єктів на основі DCNN складається з пропозиції регіону, виділення ознак CNN, класифікації регіонів та постобробки.

Модель YOLO розглядає виявлення об’єктів як проблему регресії, використовуючи одну CNN, що передбачає обмежувальні рамки та ймовірності класів у наскрізний спосіб і зробити прогноз швидшим. У цій статті для виявлення об’єктів у високій роздільній здатності використовується модель на основі YOLO (You Only Look Once).

Для практичної реалізації проекту було взято і розмічено декілька відкритих ресурсів даних, проведено навчання і тестування двох типів однієї з найбільш сучасних і успішних у сфері розпізнавання об’єктів моделі – YOLO.

Метою роботи є:

- Дослідження сучасних методів розпізнавання об’єктів
- Визначення процесу навчання і реалізації нейронної мережі у розпізнаванні об’єктів
- Визначення відносної ефективності моделі YOLO для задачі класфікації з великою кількістю класів і неякісними даними

Кваліфікаційна робота складається зі вступу, п’яти розділів, висновку, списку використаної літератури. Список використаної літератури включає в себе 23 джерела. Робота виконана на 39 сторінках друкованого тексту.

В ході виконання роботи, спочатку описується задача розпізнавання об'єктів і реалізація процесу за допомогою машинного навчання. Досліджено найбільш відомі і ефективні моделі для даного виду задач.
Далі надано більш обширний опис самої моделі YOLO та принцип роботи, історію модифікацій.
Наступними кроками є створення, навчання і огляд результатів роботи CNN на базі створених даних і залежності від вибраної моделі.

1. Розпізнавання об’єктів

1.1. Що таке розпізнавання об’єктів?

Розпізнавання об’єктів — це загальний термін для опису сукупності пов’язаних завдань комп’ютерного зору, які передбачають ідентифікацію об’єктів на цифрових фотографіях.

Класифікація зображень передбачає передбачення класу одного об’єкта на зображенні. Локалізація об’єкта – це визначення розташування одного або кількох об’єктів на зображенні та малювання великої кількості квадратів навколо їх протяжності. Виявлення об’єктів поєднує ці дві задачі та локалізує та класифікує один або кілька об’єктів на зображенні.

Коли користувач або фахівець звертається до «розпізнавання об’єктів», вони часто мають на увазі «виявлення об’єктів».

Таким чином, ми можемо розрізнити ці три завдання комп’ютерного зору:

- Класифікація зображень: передбачте тип або клас об’єкта на зображенні.
    - Вхідні дані: зображення з одним об’єктом, наприклад фотографією.
    - Вихідні дані: мітка класу (наприклад, одне або кілька цілих чисел, які зіставлені з мітками класу).
- Локалізація об’єктів: визначте наявність об’єктів на зображенні та вкажіть їх розташування за допомогою обмежуючого прямокутника.
    - Вхідні дані: зображення з одним або кількома об’єктами, наприклад фотографією.
    - Вихідні дані: одна або кілька обмежувальних прямокутників (наприклад, визначених точкою, шириною та висотою).
- Виявлення об’єктів: визначте наявність об’єктів із обмежувальною рамкою та типами чи класами розташованих об’єктів на зображенні.
    - Вхідні дані: зображення з одним або кількома об’єктами, наприклад фотографією.
    - Вихідні дані: одна або кілька обмежувальних прямокутників (наприклад, визначених точкою, шириною та висотою) і мітка класу для кожної обмежувальної рамки.

Ще одним розширенням цієї розбивки завдань комп’ютерного зору є сегментація об’єктів, також звану «сегментацією екземплярів об’єкта» або «семантичною сегментацією», де екземпляри розпізнаних об’єктів

позначаються виділенням конкретних пікселів об’єкта замість грубої рамки.

З цієї розбивки ми бачимо, що розпізнавання об’єктів відноситься до набору складних завдань комп’ютерного зору.

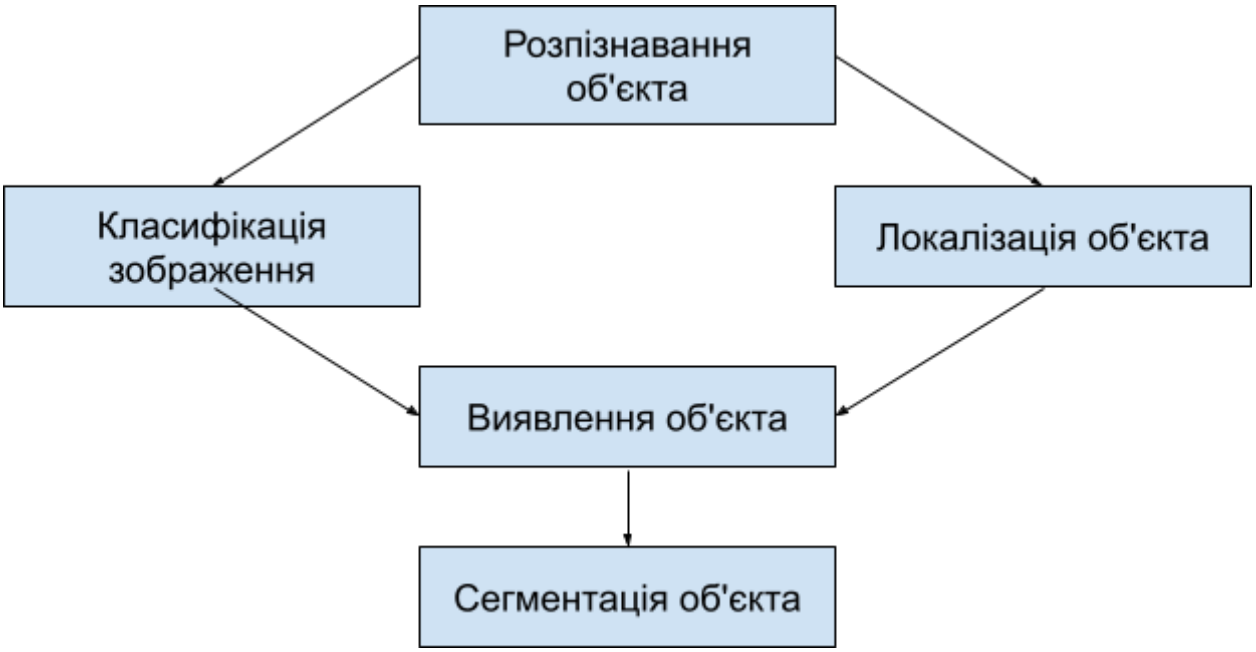


Рис.1.1 - Огляд завдань комп’ютерного зору розпізнавання об’єктів

Більшість останніх нововведень у проблемах розпізнавання зображень з'явилися в рамках участі в завданнях ILSVRC(ImageNet Large Scale Visual Recognition Challenge).

Це щорічний академічний конкурс з окремим завданням для кожного з цих трьох типів проблем, з метою сприяння незалежним і окремим покращенням на кожному рівні, які можна використовувати ширше. Наприклад, перегляньте список трьох відповідних типів завдань нижче, взятого з оглядового документа ILSVRC 2015 року:

- Класифікація зображень. Алгоритми створюють список категорій об'єктів, присутніх на зображенні.
- Локалізація окремого об’єкта: алгоритми створюють список категорій об’єктів, присутніх на зображенні, разом із вирівняною по осі обмежувальною рамкою, що вказує положення та масштаб одного екземпляра кожної категорії об’єктів.

- Виявлення об’єктів. Алгоритми створюють список категорій об’єктів, присутніх на зображенні, разом із вирівняною по осі обмежувальною рамкою, яка вказує положення та масштаб кожного екземпляра кожної категорії об’єктів.

Ми бачимо, що «Локалізація одного об’єкта» є простішою версією ширшого визначення «Локалізація об’єкта», яка обмежує завдання локалізації об’єктами одного типу в зображенні, що, як ми можемо вважати, є легшим завданням.

Нижче наведено приклад порівняння локалізації окремого об’єкта та виявлення об’єкта, взятого з статті ILSVRC. Зверніть увагу на різницю в основних очікуваннях істини в кожному випадку.

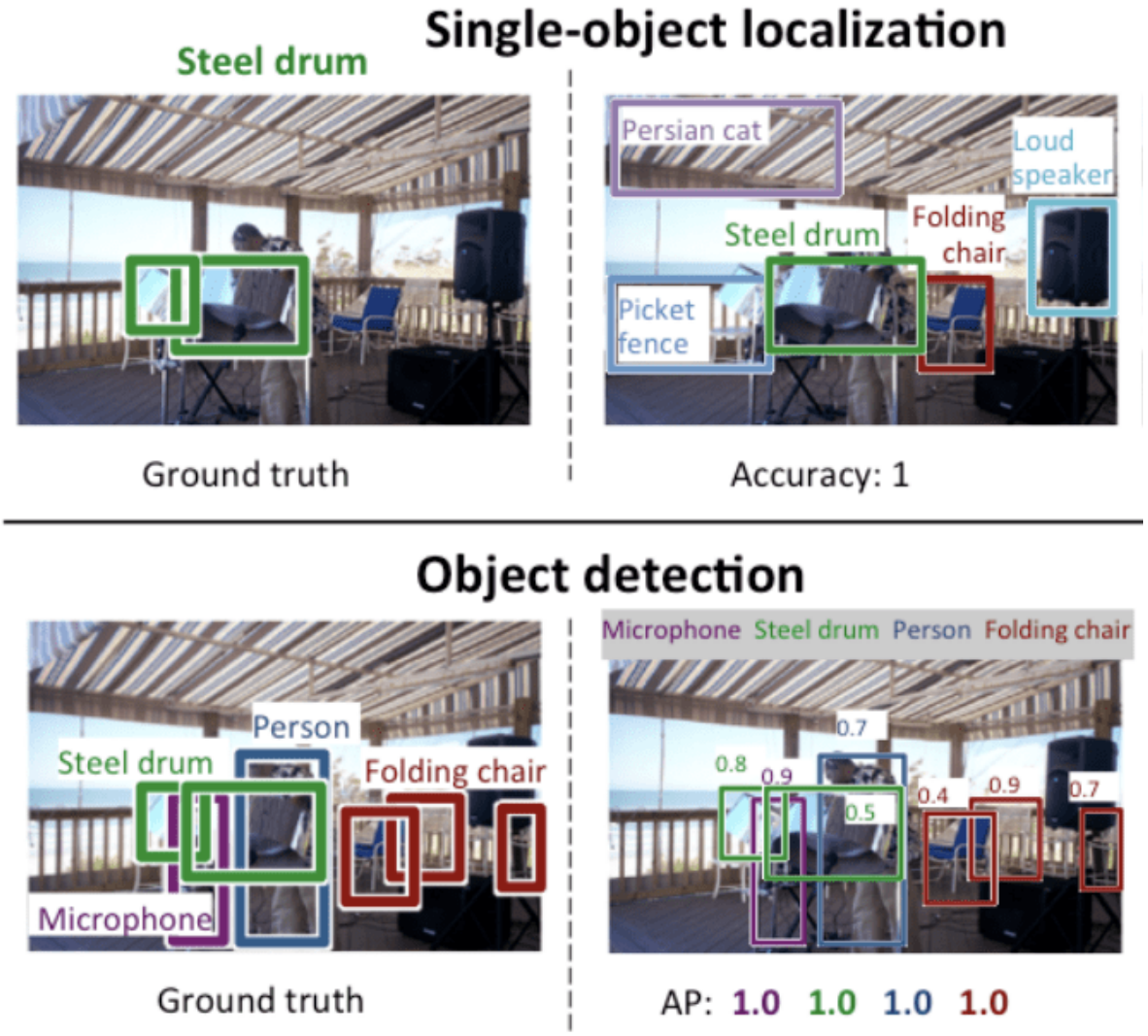


Рис 1.2 - Порівняння між локалізацією окремого об'єкта та виявленням об'єктів.

Ефективність моделі для класифікації зображень оцінюється за допомогою середньої помилки класифікації для передбачених міток класів. Ефективність моделі для локалізації окремого об’єкта оцінюється за допомогою відстані між очікуваним і прогнозованим обмежуючим квадратом для очікуваного класу. У той час як продуктивність моделі для розпізнавання об’єктів оцінюється за допомогою точності та запам’ятовування кожного з найкращих відповідних обмежувальних рамок для відомих об’єктів на зображенні.

Тепер, коли ми знайомі з проблемою локалізації та виявлення об’єктів, давайте подивимося на кілька останніх найефективніших моделей глибокого навчання.

1.2. Виявлення об'єктів на основі машинного навчання

З розвитком методів машинного навчання, особливо виділення ознак і методів класифікації, був зроблений прорив в області комп'ютерного зору, перевівши проблему виявлення цілі в проблему класифікації для машинного навчання. На схемі показано типову реалізацію методу виявлення цілей на основі машинного навчання. Навчальний процес вивчає класифікатор із навчального набору даних у контрольованому, напівкерованому або слабко контрольованому манері.

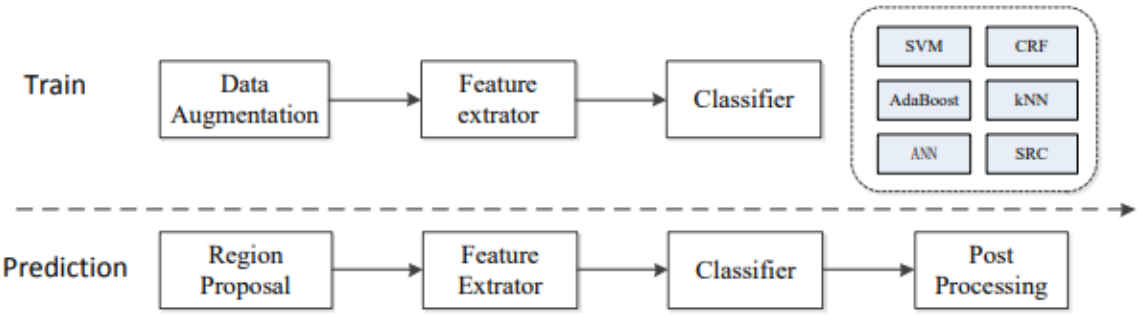


Рисунок 1.3 - Структура виявлення об'єктів на основі машинного навчання

У процесі виявлення серії регіональних зображень блоки, витягнуті за допомогою розсувного вікна або алгоритму вилучення області-кандидата витягуються за допомогою алгоритму вилучення ознак як вхідні дані класифікатора, і мітка класу виводиться як відповідне передбачення. Як правило, проводиться виявлення. Результати також необхідно виправити та оптимізувати за допомогою операцій постобробки. Алгоритми виділення області-кандидата включають розсувні вікна, суперпіксельну сегментацію,

EdgeBox, SelectSearch, Bing тощо. Алгоритми вилучення функцій включають HOG, BoW, особливості текстури, особливості розрідженого зображення, Haar, особливості та методи на основі CNN; класифікатори включають SVM, AdaBoost , k-найближчі сусіди (kNN), умовні випадкові поля (CRF), розріджені уявлення (SRC) і штучні нейронні мережі (ANN); Методи постобробки включають NMS, Boxfusion та Bounding-box Regression.

2. Виявлення об'єктів на основі CNN

Згорткова нейронна мережа (CNN) — це алгоритм глибокого навчання, який може приймати вхідне зображення, призначати важливість (засвоювані ваги та упередження) різним аспектам/об’єктам зображення та мати можливість відрізнити один від іншого. Попередня обробка, необхідна в CNN, набагато нижча в порівнянні з іншими алгоритмами класифікації. Хоча в примітивних методах фільтри розробляються вручну, з достатньою підготовкою, CNN мають можливість вивчати ці фільтри/характеристики.

Традиційні CNN зазвичай використовуються для завдання класифікації, і вихідною класифікацією є дискретна мітка класу. При виявленні об'єктів завдання для зображення дистанційного зондування, додаткова інформація, наприклад необхідно визначити місце розташування об’єкта. Кожен піксель на зображенні слід передбачити та позначити за допомогою алгоритму класифікації. В поле комп'ютерного зору, методи виявлення об'єктів для зображення на основі CNN можна розділити на два типи: локалізація об'єктів і семантична сегментація.

Ціль локалізації об'єкта полягає в отриманні обмежувальної рамки об'єкта у вхідних даних зображення, а метою сегментації сцени є отримання а передбачена маска для вхідного зображення.

RCNN (запропонований Г. Росс та ін. у 2014 р.) є репрезентативним методом розташування об’єкта, які створюють основу для вирішення проблеми виявлення об'єкта.

Місцезнаходження. RCNN складається з алгоритму пропозиції регіону (наприклад вибірковий пошук, BING і так далі) для вилучення великої кількості ROI (області інтересу) з вхідного зображення, модель CNN, щоб виділити особливості ROI, класифікатор для класифікації ROI, і алгоритм регресії для виправлення положення кордону – обмежувальної рамки.

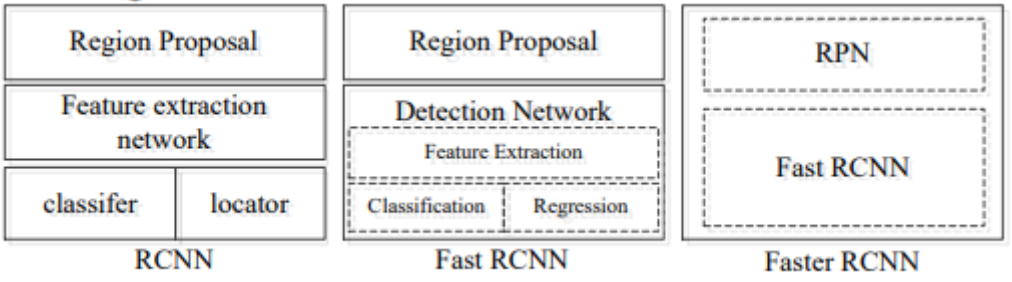


Рис 2.1 - Структура виявлення об'єктів моделей сімейства RCNN

Основними недоліками RCNN є обчислювальна складність і точність положення кордону. Просторова піраміда мережі об'єднання (SPPnet) реалізують спільне використання обчислень прискорення R-CNN. SPPnet обчислює карту згортки ціле зображення. Класифікатор витягує вектори ознак регіони-кандидати з карти ознак для класифікації. Вхідне зображення алгоритму R-CNN обмежено фіксованим розміром,
що обмежує гнучкість алгоритму. Регулювання
Розмір зображення в основному визначається обрізанням або деформацією. Ця операція призведе до певної втрати інформації.
SPPnet може об'єднати зображення будь-якого розміру в функцію фіксованої довжини представництво. SPPnet прискорює RCNN в 10-100 разів. Процес навчання SPPnet розділений на кілька етапів, як RCNN, включаючи вилучення функцій, налаштування мережі, навчання SVM і, нарешті, регресія обмежувальної рамки. Модифікація алгоритму не може раніше оновлювати вагу кожного згорткового шару СПП, що обмежує глибину згорткового шару.
Швидкість вилучення площі стає слабким місцем системи виявлення. Щоб подолати цю проблему, швидше RCNN замінює модуль вибіркового пошуку в RCNN/швидкій RCNN із мережею регіональних пропозицій (RPN), яка ще більше покращує точність і швидкість виявлення, і може бути досягнуто без урахування видобутку регіону. Виявлення майже в режимі реального часу. YOLO вирішує виявлення об’єктів як проблему регресії. Після того, як вхідне зображення передає висновок, воно може отримати посаду, категорію та відповідну впевненість мовірність всіх об'єктів на зображенні, подальше покращення швидкість виявлення та здійснення виявлення цілі в реальному часі Модель R-CNN (як показано на малюнку) використовує SelectSearch алгоритм вилучення регіонів-кандидатів, CNN як функція вилучення, SVM як класифікатор і обмежувальна рамка Алгоритм регресії для виконання регресії поля кандидата. Це фреймворк також став моделлю виявлення цілей на основі
глибоке навчання. Фундамент. В алгоритмі RCNN кожен зображення кандидата необхідно окремо класифікувати, а також час накладні витрати великі. За недоліки класичних моделей RCNN з низькою ефективністю, Росс Б. Гіршик з співавторами запропонував Fast-RCNN зменшити повторювані обчислення після перенесення вилучення регіону на карту об’єктів; Швидше-rcnn використовує мережу регіональних пропозицій

(RPN) для завершення регіону функція екстракції; YOLO безпосередньо. У вихідному шарі розташування обмежувальної рамки регресії та її категорію, визначення цільового відео в реальному часі.

2.1. Сімейство моделей R-CNN

Сімейство методів R-CNN відноситься до R-CNN, що може означати «Регіони з функціями CNN» або «Region-Based Convolutional Neural Network», розроблені Россом Гіршиком та ін.

Це включає в себе методи R-CNN, Fast R-CNN і Faster-RCNN, розроблені та продемонстровані для локалізації та розпізнавання об’єктів.

Давайте по черзі розглянемо основні моменти кожної з цих технік.

2.1.1. R-CNN

R-CNN був описаний у статті 2014 року Россом Гіршиком та ін. від UC Berkeley під назвою «Багата ієрархія функцій для точного виявлення об’єктів та семантичної сегментації».

Можливо, це було одне з перших великих і успішних застосувань згорткових нейронних мереж для вирішення проблеми локалізації, виявлення та сегментації об’єктів. Цей підхід був продемонстрований на контрольних наборах даних, досягнувши найсучасніших результатів у наборі даних VOC-2012 та наборі даних виявлення об’єктів ILSVRC-200 200 класів.

Запропонована ними модель R-CNN складається з трьох модулів; вони є:

- Модуль 1: Регіональна пропозиція. Створення та вилучення категорій незалежних пропозицій регіону, напр. обмежувальні рамки кандидата.
- Модуль 2: Екстрактор функцій. Витягніть функцію з кожного регіону-кандидата, напр. використання глибокої згорткової нейронної мережі.
- Модуль 3: Класифікатор. Класифікувати ознаки як один із відомих класів, напр. модель лінійного класифікатора SVM.

Архітектура моделі узагальнена на зображені нижче.

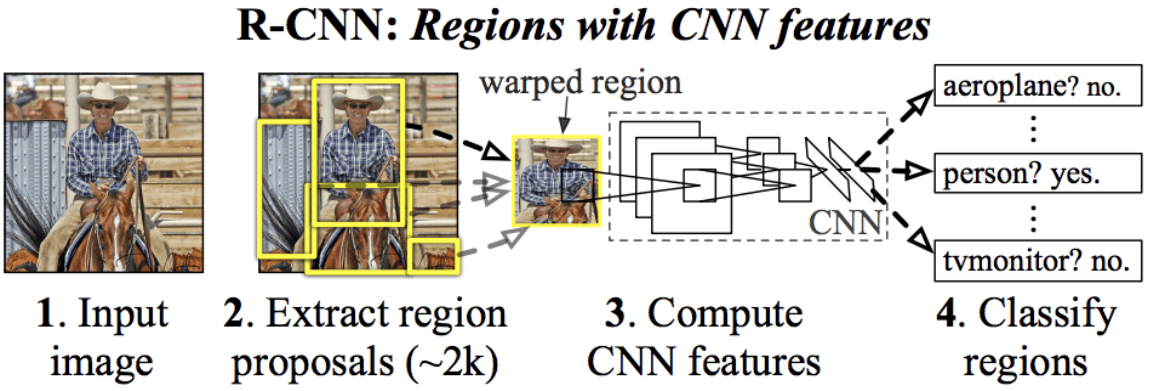


Рис 2.2 - Резюме архітектури моделі R-CNN, взятої з ієрархій функцій Rich для точного виявлення об'єктів і семантичної сегментації.

Техніка комп'ютерного зору використовується для пропонування регіонів-кандидатів або обмежувальних рамок потенційних об'єктів на зображенні, які називаються «селективним пошуком», хоча гнучкість дизайну дозволяє використовувати інші алгоритми пропозиції регіонів.
Екстрактор функцій, використаний у моделі, був AlexNet deep CNN, який переміг у конкурсі з класифікації зображень ILSVRC-2012. Результатом CNN був вектор із 4096 елементів, який описує вміст зображення, яке подається на лінійний SVM для класифікації, зокрема, один SVM навчається для кожного відомого класу.
Це відносно просте і зрозуміле застосування CNN до проблеми локалізації та розпізнавання об'єктів. Недоліком підходу є те, що він повільний, що вимагає проходження вилучення ознак на основі CNN для кожного з регіонів-кандидатів, згенерованих алгоритмом пропозиції регіону. Це проблема, оскільки в статті описується модель, яка працює приблизно з 2000 запропонованих регіонів на одне зображення під час тестування.
Вихідний код Python (Caffe) і MatLab для R-CNN, описаний у статті, був доступний у репозиторії R-CNN GitHub.

2.1.2. Fast R-CNN

Враховуючи великий успіх R-CNN, Росс Гіршик, який тоді працював у Microsoft Research, запропонував розширення для вирішення проблем швидкості R-CNN у статті 2015 року під назвою «Fast R-CNN».
Стаття починається з огляду обмежень R-CNN, який можна підсумувати таким чином:

- Навчання — це багатоетапний конвеєр. Передбачає підготовку та експлуатацію трьох окремих моделей.
- Навчання дороге в просторі та часі. Навчання CNN щодо такої кількості пропозицій регіону на одне зображення дуже повільне.
- Виявлення об’єктів відбувається повільно. Робити прогнози за допомогою глибокого CNN на таку кількість пропозицій регіону дуже повільно.

У статті 2014 року «Об’єднання просторових пірамід у глибоких згорткових мережах для візуального розпізнавання» була запропонована попередня робота для прискорення техніки, яка називається мережами об’єднання просторових пірамід або SPPnets. Це прискорило вилучення функцій, але по суті використовувало тип алгоритму кешування прямого проходу.

Швидкий R-CNN пропонується як єдина модель замість конвеєра для вивчення та виведення регіонів і класифікацій безпосередньо.

Архітектура моделі приймає фотографію набір пропозицій регіону як вхідні дані, які передаються через глибоку згорткову нейронну мережу. Попередньо навчений CNN, такий як VGG-16, використовується для виділення функцій. Кінець глибокого CNN — це користувацький шар, який називається шаром об’єднання інтересів або об’єднанням інтересів, який витягує особливості, характерні для даного регіону-кандидата.

Вихід CNN потім інтерпретується повністю підключеним шаром, після чого модель розбивається на два виходи, один для передбачення класу через softmax шар, а інший з лінійним виводом для обмежувального прямокутника. Потім цей процес повторюється кілька разів для кожної цікавить області на даному зображенні.

Архітектура моделі узагальнена на зображені нижче.

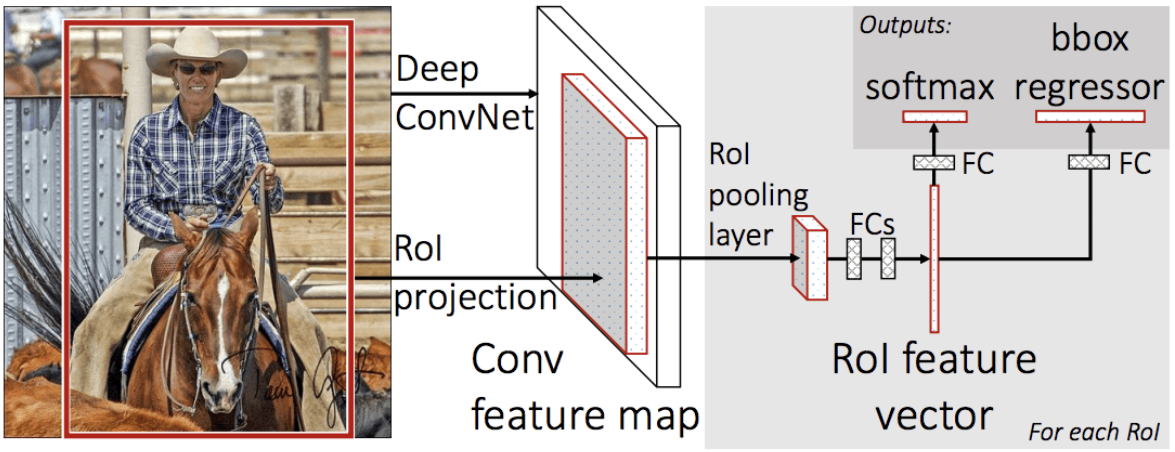


Рис 2.3 - Резюме архітектури моделі Fast R-CNN.

Модель значно швидше навчається та робить прогнози, але все одно вимагає набору регіонів-кандидатів, які необхідно запропонувати разом із кожним вхідним зображенням.
Вихідний код Python і C++ (Caffe) для Fast R-CNN, описаний у статті, був доступний у репозиторії GitHub.

2.1.3. Faster R-CNN

Архітектура моделі була додатково покращена як для швидкості навчання, так і для виявлення, Шаоцін Рен та ін. у Microsoft Research у статті 2016 року під назвою «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.».
Архітектура стала основою для досягнення перших результатів у конкурсних завданнях ILSVRC-2015 і MS COCO-2015.
Архітектура була розроблена, щоб запропонувати та уточнити пропозиції регіону як частину процесу навчання, іменованого як регіональна мережа пропозицій, або RPN. Потім ці регіони використовуються разом з моделлю Fast R-CNN в одній моделі. Ці покращення зменшують кількість пропозицій щодо регіонів і прискорюють роботу моделі під час тестування майже до реального часу з найсучаснішою продуктивністю.
Хоча це єдина уніфікована модель, архітектура складається з двох модулів:

- Модуль 1: Мережа регіональних пропозицій. Згортка нейронна мережа для пропонування регіонів і типу об'єкта, який потрібно розглянути в регіоні.

- Модуль 2: Швидкий R-CNN. Згортка нейронна мережа для вилучення ознак із запропонованих областей та виведення обмежувального прямокутника та міток класів.

Обидва модулі працюють на одному виході глибокої CNN. Мережа регіональних пропозицій діє як механізм уваги для мережі Fast R-CNN, інформуючи другу мережу про те, куди шукати чи звернути увагу.
Архітектура моделі узагальнена на зображенні нижче.

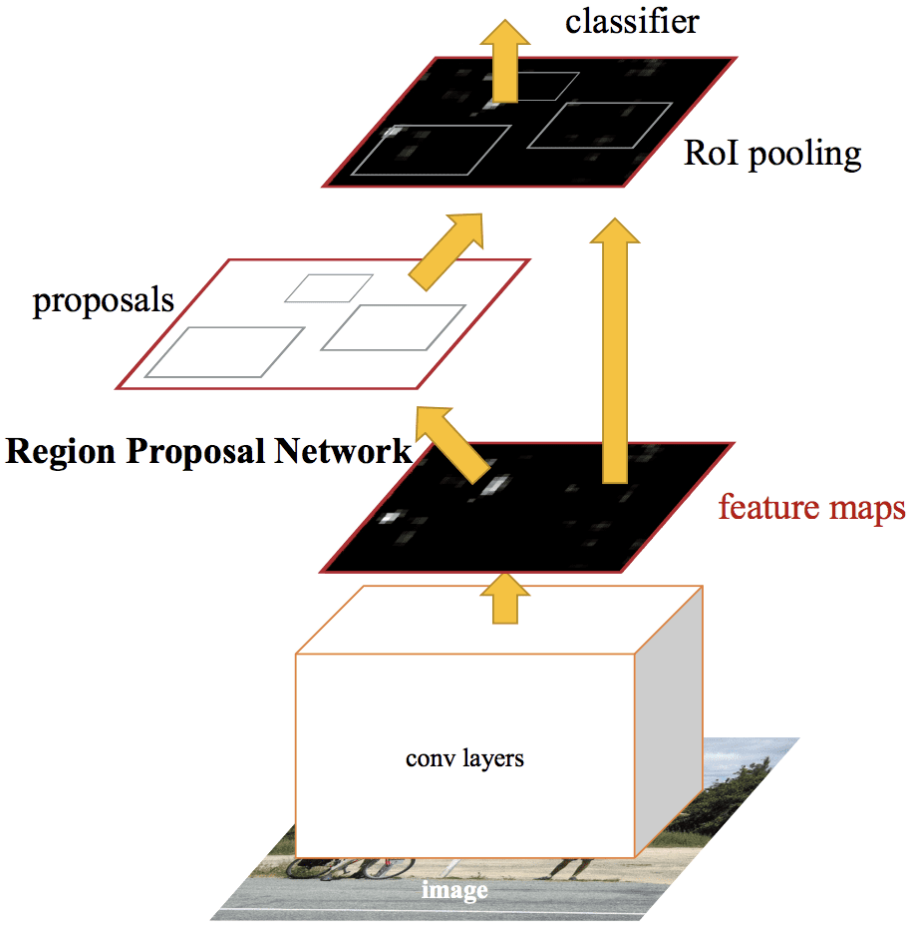


Рис. 2.4 – Резюме архітектури моделі Faster R-CNN.

RPN працює, беручи вихідні дані попередньо навченої глибокої CNN, наприклад VGG-16, і передаючи невелику мережу через карту об’єктів і виводячи кілька пропозицій регіону та прогноз класу для кожного. Пропозиції регіонів — це обмежувальні рамки, засновані на так званих прив’язних блоках або заздалегідь визначених фігурах, призначених для прискорення та покращення пропозиції регіонів. Прогноз класу є бінарним, що вказує на наявність об’єкта чи ні, так звану «об’єктність» пропонованого регіону.

Використовується процедура чергування навчання, коли обидві підмережі навчаються одночасно, хоча і чергуються. Це дозволяє налаштувати параметри в глибокому CNN детектора функцій для обох завдань одночасно.
На момент написання статті ця архітектура Faster R-CNN є вершиною сімейства моделей і продовжує досягати майже найсучасніших результатів у задачах розпізнавання об’єктів. Ще одне розширення додає підтримку сегментації зображень, описану в статті 2017 року «Mask R-CNN».
Вихідний код Python і C++ (Caffe) для Fast R-CNN, описаний у статті, був доступний у репозиторії GitHub.

2.2. Сімейство моделей YOLO

Інше популярне сімейство моделей розпізнавання об’єктів спільно згадується як YOLO або «Ти дивишся лише раз», розроблений Джозефом Редмоном та ін.
Моделі R-CNN можуть бути, як правило, більш точними, але сімейство моделей YOLO швидкі, набагато швидші, ніж R-CNN, досягаючи виявлення об'єктів в режимі реального часу.

2.2.1. YOLO

Модель YOLO вперше була описана Джозефом Редмоном та ін. у статті 2015 року під назвою «Ви дивитесь лише раз: уніфіковане виявлення об’єктів у реальному часі». Зауважте, що Росс Гіршик, розробник R-CNN, також був автором і учасником цієї роботи, тоді ще у Facebook AI Research.
Підхід включає в себе одну навчену нейронну мережу від кінця до кінця, яка бере фотографію як вхідні дані та прогнозує обмежувальні рамки та мітки класів для кожної обмежувальної рамки безпосередньо. Метод забезпечує меншу точність прогнозування (наприклад, більше помилок локалізації), хоча працює зі швидкістю 45 кадрів в секунду і до 155 кадрів в секунду для версії моделі, оптимізованої за швидкістю.
Модель працює, спочатку розбиваючи вхідне зображення на сітку комірок, де кожна клітинка відповідає за передбачення обмежувальної рамки, якщо центр обмежувальної рамки потрапляє всередину комірки. Кожна клітинка сітки передбачає обмежувальну рамку, що включає координати x, y, ширину, висоту та впевненість. Прогноз класу також базується на кожній клітинці.

Наприклад, зображення можна розділити на сітку 7×7, і кожна клітинка в сітці може передбачити 2 обмежувальні рамки, в результаті чого буде запропоновано 94 пропонованих обмежувальні рамки. Карта ймовірностей класів і обмежувальні рамки з впевненістю потім об'єднуються в остаточний набір обмежувальних прямокутників і міток класів. Зображення, взяте з паперу нижче, підсумовує два виходи моделі.

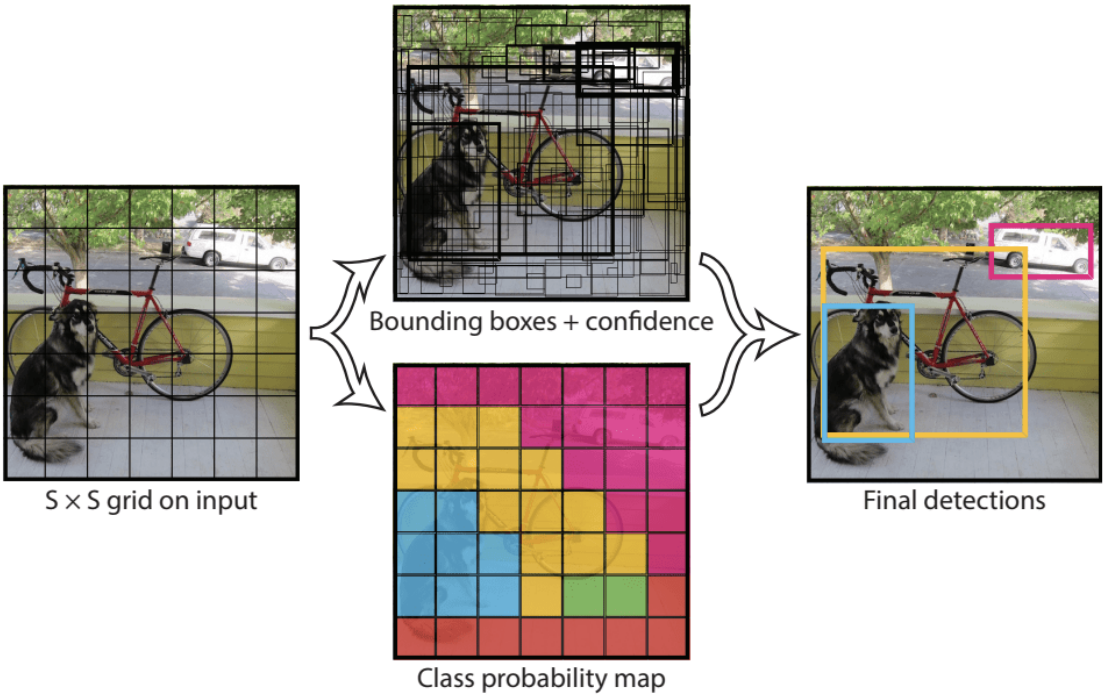


Рис 2.5 - Резюме прогнозів, зроблених моделлю YOLO.

2.2.2. YOLOv2 (YOLO9000) і YOLOv3

Модель була оновлена Джозефом Редмоном і Алі Фархаді з метою подальшого покращення продуктивності моделі в їхній статті 2016 року під назвою «YOLO9000: Better, Faster, Stronger».

Хоча ця варіація моделі називається YOLO v2, описано екземпляр моделі, який навчався на двох наборах даних розпізнавання об'єктів паралельно, здатних передбачити 9000 класів об'єктів, тому отримав назву «YOLO9000».

У модель було внесено ряд навчальних та архітектурних змін, таких як використання пакетної нормалізації та вхідних зображень з високою роздільною здатністю.

Як і Faster R-CNN, модель YOLOv2 використовує блоки прив’язки, попередньо визначені обмежувальні рамки з корисними формами та розмірами, які налаштовуються під час навчання. Вибір обмежувальних рамок для зображення попередньо обробляється за допомогою аналізу k-середніх на наборі навчальних даних.

Важливо, що передбачене представлення обмежувальних рамок змінено, щоб дозволити невеликим змінам мати менш різкий вплив на передбачення, що приведе до більш стабільної моделі. Замість того, щоб безпосередньо передбачати положення та розмір, зміщення прогнозуються для переміщення та зміни форми попередньо визначених прив’язних блоків щодо комірки сітки та демпфуються логістичною функцією.

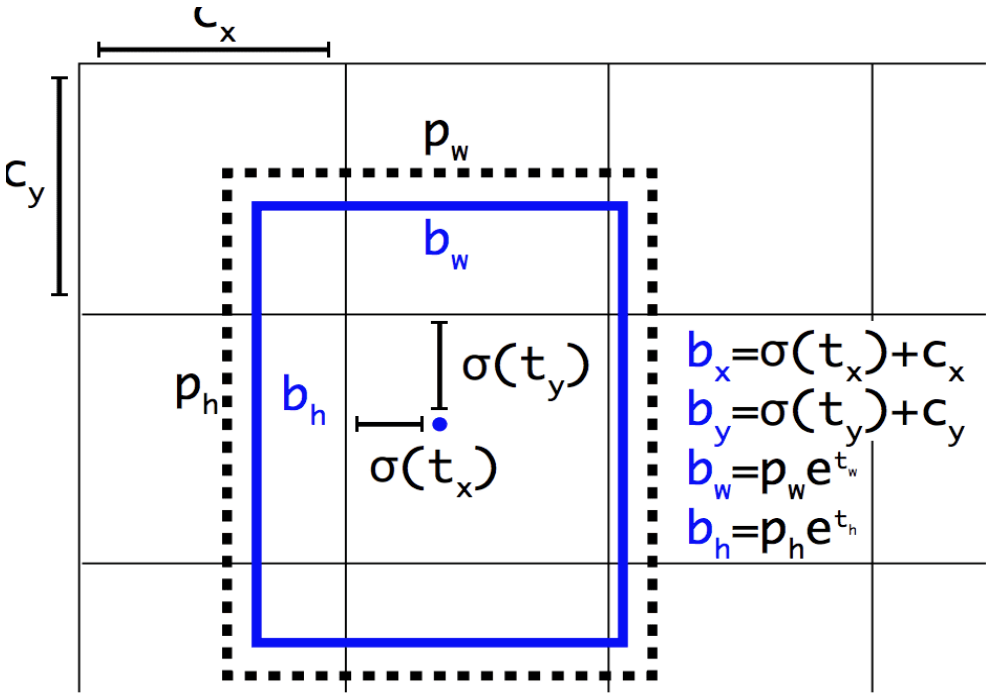


Рис 2.6 - Приклад представлення, вибраного під час передбачення положення та форми обмежувальної рамки.

Приклад представлення, вибраного під час передбачення положення та форми обмежувальної рамки

Приклад представлення, вибраного під час прогнозування положення та форми обмежувальної рамки. Взято з: YOLO9000: краще, швидше, міцніше

Подальші вдосконалення моделі були запропоновані Джозефом Редмоном та Алі Фархаді у їхній статті 2018 року під назвою «YOLOv3: Поступове покращення». Покращення були досить незначними, включаючи більш глибоку мережу детекторів функцій і незначні зміни у представленні.

2.2.3. YOLOv3 і подальші модифікації

YOLO швидко став відомим серед комп'ютерного зору завдяки своїй високій швидкості та гарній точності. Однак, у лютому 2020 року Джозеф Редмон, творець YOLO, оголосив, що припинив дослідження комп'ютерного зору. Це призвело до гарячих обговорень у спільноті та поставило важливе питання: чи будуть якісь оновлення YOLO в майбутньому?

Вихід Редмона не був кінцем YOLO. У квітні 2020 року було випущено 4-е покоління YOLO, яке знову переживає багато людей із комп'ютерного зору. Його було представлено в статті під назвою «YOLOv4: оптимальна швидкість і точність виявлення об'єктів» Олексія Бочковського та інших.

Крім того, роботу Редмона продовжив Олексій у форку основного репозитарію. YOLO v4 вважається найшвидшою та найточнішою моделлю в реальному часі для виявлення об'єктів.

Основні покращення в YOLO v4

YOLO v4 бере на себе вплив сучасного BoF (мішок халяви) і кілька BoS (мішок акцій). BoF покращує точність детектора, не збільшуючи час висновку. Вони лише збільшують вартість навчання. З іншого боку, BoS збільшують вартість висновку на невелику суму, однак вони значно покращують точність виявлення об'єктів.

Продуктивність YOLO v4

YOLO v4 також заснований на Darknet і отримав значення AP у 43,5 відсотка в наборі даних COCO разом зі швидкістю в реальному часі 65 кадрів в секунду на Tesla V100, перевершуючи найшвидші та найточніші детектори з точки зору швидкості та точності.

У порівнянні з YOLO v3, AP і FPS зросли на 10 і 12 відсотків відповідно.

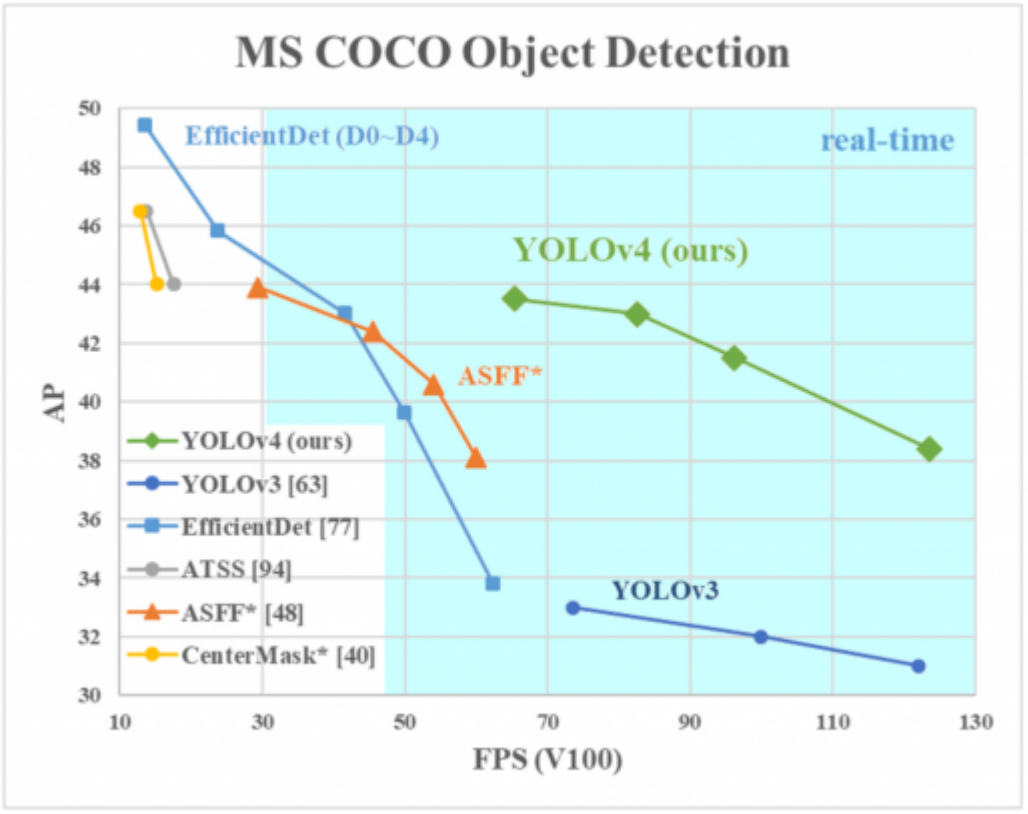


Рис 2.7 - Порівняння пропонованого YOLOv4 та ін найсучасніші детектори об'єктів. YOLOv4 працює вдвічі швидше ніж EfficientDet з порівнянною продуктивністю.

Після виходу YOLO v4, лише через два місяці, була випущена ще одна версія YOLO під назвою YOLO v5. Це Гленн Джохер, який уже відомий серед спільноти за створення популярної реалізації PyTorch YOLO v3.
9 червня 2020 року Джохер заявив, що його реалізація YOLO v5 є загальнодоступною і рекомендована для використання в нових проектах. Однак він не опублікував документ, що супроводжував би свій реліз, коли спочатку випускав цю нову версію.

Основні покращення в YOLO v5
YOLO v5 відрізняється від усіх інших попередніх випусків, оскільки це реалізація PyTorch, а не форк від оригінального Darknet. Як і YOLO v4, YOLO v5 має хребет CSP і шию PA-NET. Основні вдосконалення включають збільшення даних мозаїки та прив’язки обмежувальної рамки автоматичного навчання.

Підсумовуючи, YOLO v4 є останньою реалізацією цього найсучаснішого детектора об’єктів на основі Darknet. У ньому також є стаття, опублікована з контрольними показниками Олексія Бочковського. З іншого боку, YOLO v5 є новою реалізацією PyTorch від Ultralytics, і при тестуванні з більшим розміром партії він має вищу швидкість перешкод, ніж більшість детекторів. Однак на момент написання цієї статті не було опубліковано жодної рецензованої статті для YOLO v5. PP-YOLO — це ще одне оновлення YOLO, засноване на фреймворку глибокого навчання під назвою PaddlePaddle, і воно покращує модель YOLO v3 для досягнення кращого балансу між ефективністю та ефективністю.

3. Визначення об’єктів на основі YOLO

3.1. Архітектура

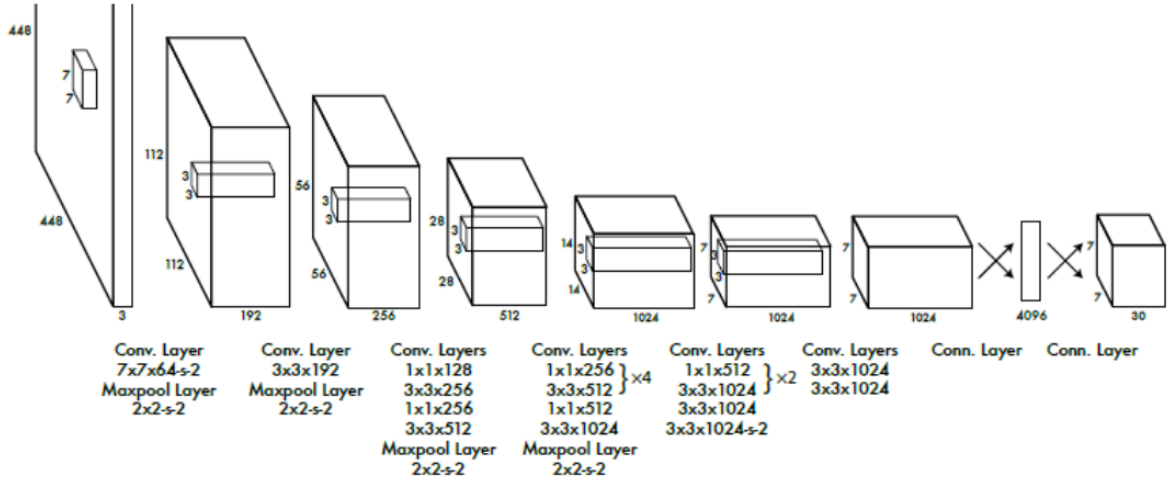


Рис 3.1 – Архітектура моделі YOLO

Модель YOLO — це структура на основі CNN. На початку, функція згорткового вилучення зображення передбачає вихід ймовірність через весь зв'язаний шар. Через подібне до GoogleNet також навчив швидку версію YOLO, створюючи FastYOLO на кінцевому виході тензора 7 * 7 * 30. На малюнку 1 показано мережеву структуру YOLO, яка має 24 згорткові шари і 2 повністю з'єднані шари. Шар згортки, що чергується 1 × 1 зменшує простір об’єктів попереднього шару, покращуючи дозвіл виявлення шляхом попереднього навчання згорткових шарів на класифікація ImageNet. Остаточний вихід нашої мережі тензор передбачень 7 × 7 × 30.

3.2. Навчання моделі

(1) Збільшення даних

Метою розширення даних є створення нового зразка екземпляри, а коли навчальних вибірок мало, дані збільшення є дуже корисним для підвищення надійності мережі. Для зображень дистанційного зондування, такі методи, як обертання на 45 градусів, масштабування 15-25%, різання, перемикання смуги частот, використовується вертикальне/горизонтальне перегортання та інші операції із зображенням ця стаття для збільшення узагальнюючої ємності мережі.

Наведені вище операції не використовуються для набору перевірки та тестовий набір.

(2) Оптимізація

У CNN найбільш поширеним оптимізатором є SGD (Stochastic Gradient Descen), але він має певні проблеми з пошуком правильної швидкості навчання та легко зближуються до локального оптимуму. У цій статті оптимізатор Адама (Adaptive Moment Estimation) використовується в якості навчальної стратегії для розрахунку адаптивної швидкості навчання кожного параметра. Оптимізатор Адама коригує швидкість навчання кожного параметра відповідно до оцінки першого моменту та оцінка другого моменту його градієнтної функції втрат. Оптимізатор Adam працює швидко і може виправити проблеми швидкості навчання, повільної швидкості зближення і великого коливання функції втрат. Оптимізатор Адама описано формулами, наведеними нижче.

$$\hat{m}_t = m_t / 1 - \beta_1^t \quad (1)$$

$$\hat{v}_t = v_t / 1 - \beta_2^t \quad (2)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \quad (3)$$

c = focal length
x, y = image coordinates
X_0, Y_0, Z_0 = coordinates of projection center
X, Y, Z = object coordinates

Формула 1 представляє відхилення першого моменту, формула 2 представляє відхилення другого моменту, η – крок, ε – константа (значення за замовчуванням є 10^{-8}).

(3) Функція втрати

У математичній оптимізації, статистиці, економетриці, теорії прийняття рішень, машинному навчанні та обчислювальній нейронауці, функція втрат або функція вартості — це функція, яка відображає подію або значення однієї або кількох змінних на дійсне число інтуїтивно, що представляє

деяку "вартість", пов'язану з подією. Задача оптимізації спрямована на мінімізацію функції втрат. Цільова функція є або функцією втрат, або її негативною, у цьому випадку його слід максимізувати. У YOLO середньоквадратична помилка використовується як функція втрат для оптимізації параметрів моделі та середньо-квадратична помилка вектора виведення з мережі і вектора відповідний реальному образу. Як показано нижче, де: $coordError$ – це помилка координат, $iouError$ – помилка IOU та $classError$ – це помилка класифікації.

$$loss = \sum_{i=0}^{S^2} coordError + iouError + classError \qquad (4)$$

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (h_i - \sqrt{\hat{h}_i})^2 \right] \\
&+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+\sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \qquad (5)$$

where x, y, w, C, p = network prediction

$\hat{x}, \hat{y}, \hat{w}, \hat{C}, \hat{p}$ = groud truth

1_i^{obj} means object fall into the lattice i

1_{ij}^{obj} means object fall into the the bounding box j in lattice i

1_{ij}^{noobj} means object not fall into the the bounding box j in lattice i

3.3. Постобробка

Останнім кроком YOLO є виконання немаксимального придушення (NMS) на векторах S*S*(B*5+C). Метою NMS є ліквідація зайвих комірок та знаходження найкращого місця виявлення об’єкта. Алгоритми NMS використовуються у добре відомих системах виявлення цілей, таких як RCNN і SPPnet.

3.4. Точне налаштування

Стратегії передачі навчання залежать від різних факторів, але двома найважливішими з них є розмір нового набору даних і його схожість з початковим набором даних. Детальна настройка дозволяє нам довести потужність найсучасніших моделей DCNN для нових доменів де недостатні дані та обмеження часу/вартості можуть запобігати їх використанню.

3.5. Показник оцінки

Щоб оцінити продуктивність алгоритму, результати роботи моделі слід порівнювати з істиною. У цій статті точність (*Precision*) і відкликання (*Recall*) використовуються для оцінки подібності і відмінності між результатами виявлення та істиною в тестовому наборі даних.

$$\mathrm{P} = TP/(TP + FP) \quad (6)$$

$$\mathrm{R} = TP/(TP + FN) \quad (7)$$

where TP = true positive

FP = false positive

FN = false negative.

3.6. Переваги та недоліки

Переваги:

1) Швидше. YOLO вирішує виявлення об'єктів як проблему регресії, які спрощують весь конвеєр мережі виявлення.
2) Нижчий фоновий рівень помилково-позитивних результатів. Під час навчання і процесу прогнозування, інформація щодо всього зображення може бути сприйнята. Метод на основі RCNN може сприймати тільки локальну інформацію з фрейма кандидата.
3) Сильна універсальність. YOLO можна застосувати для виявлення об’єктів неприродних зображень.

Недоліки:

1) Модель використовує декілька шарів зниження дискретизації. Характеристики об'єктів, які вивчаються в Інтернеті, не є гарними, що впливає на ефект виявлення.
2) Погане розпізнавання точного положення об'єкта.
3) Виявлення малих цілей і щільних цілей погане.
4) Нижча швидкість відкликання.

4. Реалізація

4.1. Апаратне та програмне середовище

Модель реалізована з Keras із бекендом Tensorflow.

Keras — це високорівневий API нейронних мереж, написаний на Python і здатний працювати поверх TensorFlow, CNTK або Theano. Tensorflow, розроблений командою Google Brain, є бібліотекою програмного забезпечення з відкритим кодом для програмування потоків даних у різних діапазонах завдань. Потрібні багато сторонніх бібліотек, наприклад Tiffle для читання зображень дистанційного зондування, OpenCV для базового зображення обробки, Shapely для обробки полігональних даних, Matplotlib як інструмент візуалізації, Roboflow для побудови наборів даних. Експерименти проводяться завдяки сервісу Google Colab, який надає можливість тимчасової оренди серверу, який може бути достатньо ефективно використаний при навчанні моделі.

4.2. Експериментальний процес

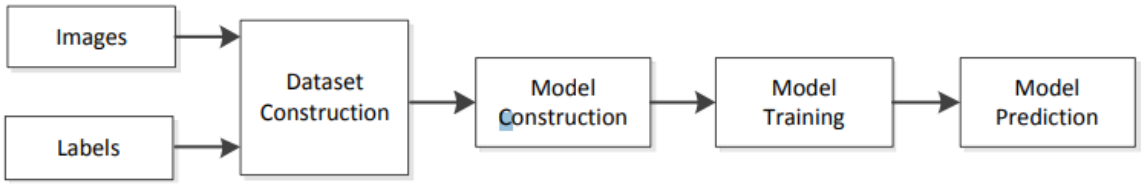


Рис. 4.1 – Експериментальний процес

Експериментальний процес можна розділити на чотири етапи:

1) Побудова набору даних. Зображення дистанційного зондування (підвищення від відкритого джерело даних, наприклад GoogleEarth, USGS, DigitalGlobe і так далі) анімуються за допомогою imgLabel для отримання стандарту. Розділення визначеного набору даних на навчальні, валідаційні та тестові набори.
2) Конструкція моделі. Побудова структури CNN та встановлення її гіперпараметрів.
3) Модельне навчання. Навчання з навчальними наборами та наборами для перевірки. Модельний прогноз. Тестування з тестовим набором, і результат використовується для оцінки моделі.

4.3. Набір даних

У роботі за основу взяті декілька наборів фотографій з відкритих ресурсів, таких як Kaggle та GoogleEarth. Після того, як зібрана достатньо велика кількість якісних фото для експерименту, використовується програма Roboflow в якості інструмента для анотації.

4.3.1. Збір і анотація

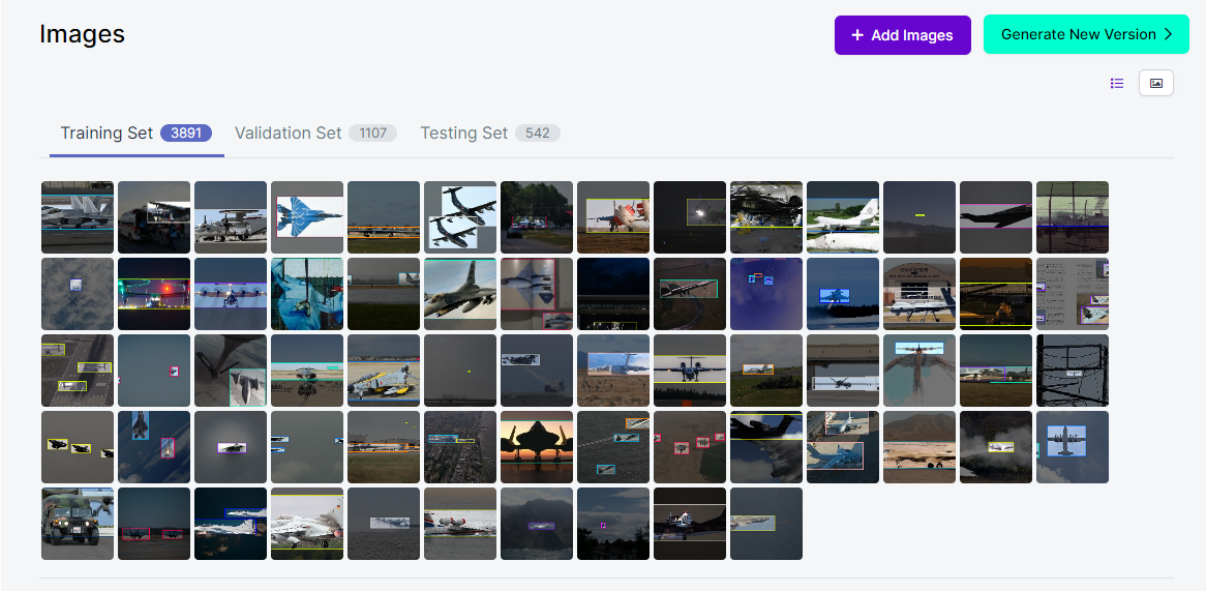


Рис. 4.2 – Використовуваний набір даних, завантажений до проекту RoboFlow.

Процес самої анотації є доволі простим:

1) Для обраного фото визначається рамка, в якій знаходиться об’єкт для розпізнавання.
2) Рамка мінімізується до країв об’єкта задля зменшення похибки при навчанн.
3) Вказується, до якого класу відноситься об’єкт (в нашому випадку класи – це назви техніки/авіації).
4) Створюється файл з назвою, аналогічній початковій фотографії, але з розширенням «.csv», в якому вказується клас і 4 точки (x, y) відповідно рамці визначеного об’єкта.

В даному випадку, за рахунок використання RoboFlow крок 4 виконується автоматично.

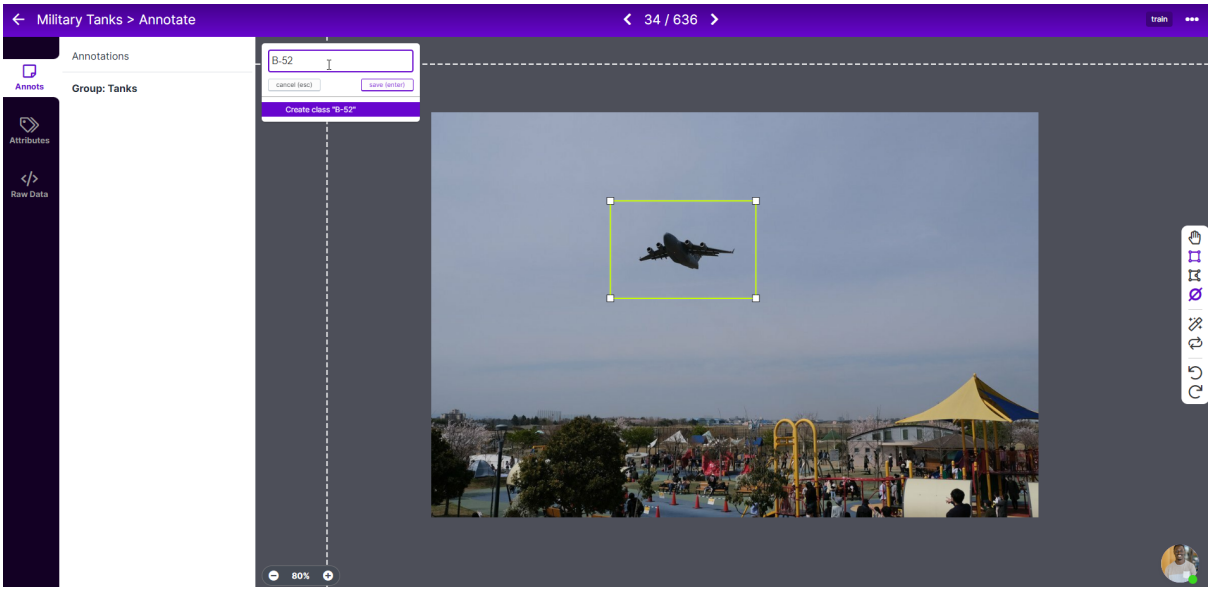


Рис. 4.3 – Процес анотації зображення за допомогою RoboFlow.

	A	B	C	D	E	F	G	H	I	J
1	filename	width	height	class	xmin	ymin	xmax	ymax		
2	000e7662	1200	856	B-52	522	240	126	497		
3										
4										

Рис. 4.4 – Згенерована анотація у форматі CSV-файла.

4.3.2. Генерація додаткових даних за рахунок обробки існуючих

Для збільшення кількості даних в датасеті використовуються різні методи обробки фото, до яких, наприклад, входять:

- Обрізка
- Зміна контрасту, кольорів (для нашого експерименту влучно буде використовувати чорно-білий ефект, так як фотографії з безпілотників зазвичай чорно-білі)
- Погіршення якості, шум
- Зміна розміру
- Розмиття
- Поворот

Все це робиться задля того, щоб покращити результати тестування моделі на неякісних фото, які зазвичай отримують в реальному світі.

4.3.3. Розподілення

В даному випадку, так як ми використовуємо модель YOLOv5, має бути сгенерований .yaml-файл, в котрому буде міститися дані щодо класів, файлів і їх відповідних анотацій.

Також дані поділяються на три набори:

- Тренувальний - приклади, які використовують під час процесу навчання, та використовують для допасовування параметрів класифікатора.
- Затверджувальний – той, який використовують для налаштування гіперпараметрів класифікатора. Його іноді також називають розробницьким набором.
- Тестувальний – це набір даних, що є незалежним від тренувального набору даних, але слідує тому же розподілові ймовірності, що й тренувальний. Якщо модель, допасована до тренувального набору даних, також добре допасовується й до випробувального набору даних, то було мінімальне перенавчання (див. рисунок нижче). Краща допасованість до тренувального набору даних, на противагу до випробувального набору даних, зазвичай вказує на перенавчання.

4.4. Вибір і навчання моделі

В якості експеримента були використані два варіанта моделі YOLOv5 - YOLOv5n6 і YOLOv5x6. Порівняння між доступними варіантами моделей описані наступним графіком відповідності швидкості обробки до якості класифікації тренованої моделі.

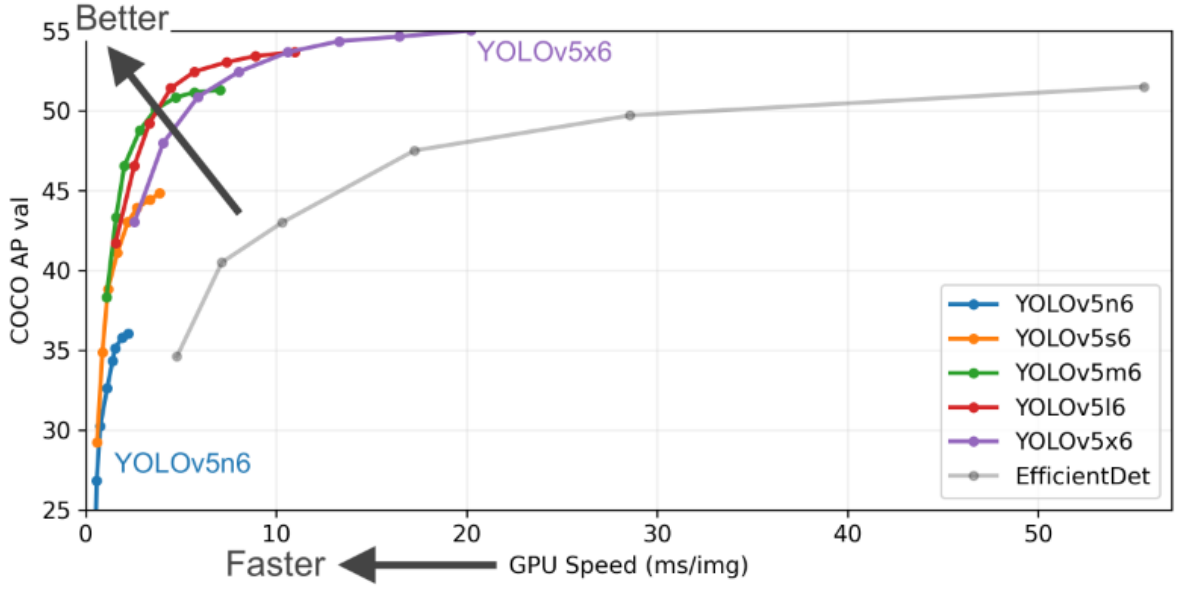


Рис. 4.5 – Порівняння залежності швидкості і точності разних типів моделей YOLOv5.

Час, витрачений на навчання моделі на обраному наборі даних сильно варіюється для різних моделей відповідно до графіка. Моделі YOLOv5n6 знадобилося 2.5 години для обробки тестового набору, тоді як YOLOv5x6 – 20.

5. Результати тренування

5.1. Характеристики

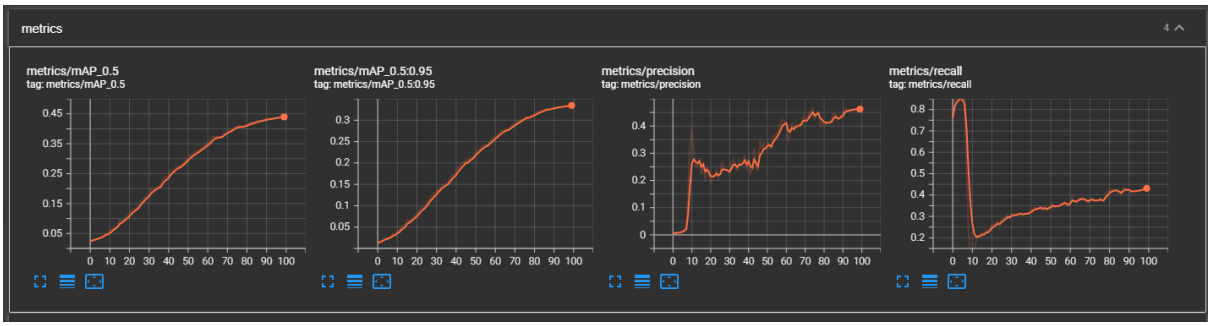


Рис. 5.1 – Метрики результатів роботи моделі YOLOv5n6

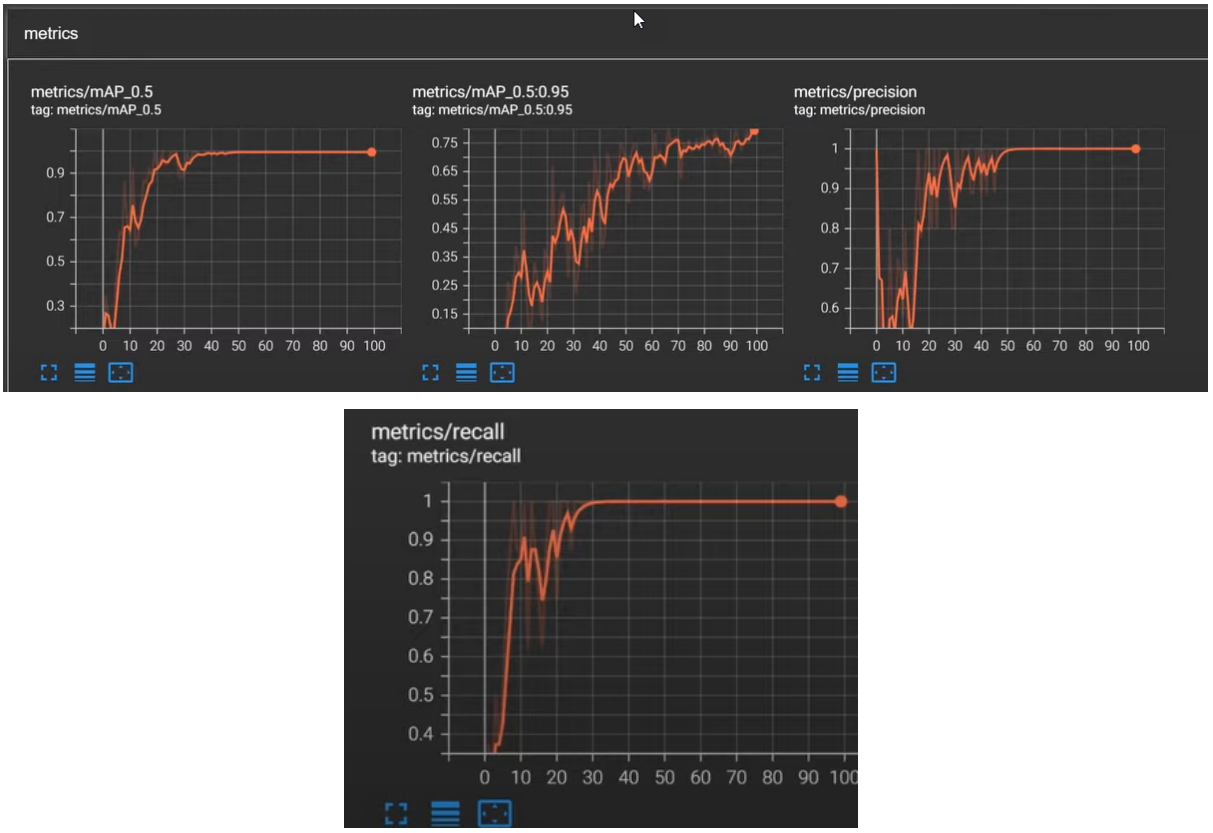


Рис. 5.2 – Метрики результатів роботи моделі YOLOv5x6

mAP(mean Average Precision) – забезпечує вимірювання якості на всіх рівнях відкликання для класифікації одного класу, його можна розглядати як область під кривою точного відкликання. Тоді mAP є середнім AP у багатокласовій класифікації.

$Precision$ – точність, розраховується як відношення між кількістю позитивних зразків, правильно класифікованих, до загальної кількості зразків, класифікованих як позитивні (правильно або неправильно). Точність вимірює точність моделі при класифікації зразка як позитивного.

$Recall$ – Відкликання, розраховується як відношення між кількістю позитивних зразків, правильно класифікованих як позитивні, до загальної кількості позитивних зразків. Відкликання вимірює здатність моделі виявляти позитивні зразки. Чим вище відкликання, тим більше позитивних зразків виявлено.

5.2. Зразки результатів обробки

Вихідними даними роботи моделі є також самі новозгенеровані файли, в яких на аналізовані фото накладається рамка з остаточним найбільш імовірним класом.

Приклади результатів роботи YOLOv5n6 на тестовому наборі:

Рис. 5.3 – 5.5 – Приклади вихідних даних моделі YOLOv5n6.

Приклади результатів роботи YOLOv5x6 на тестовому наборі:

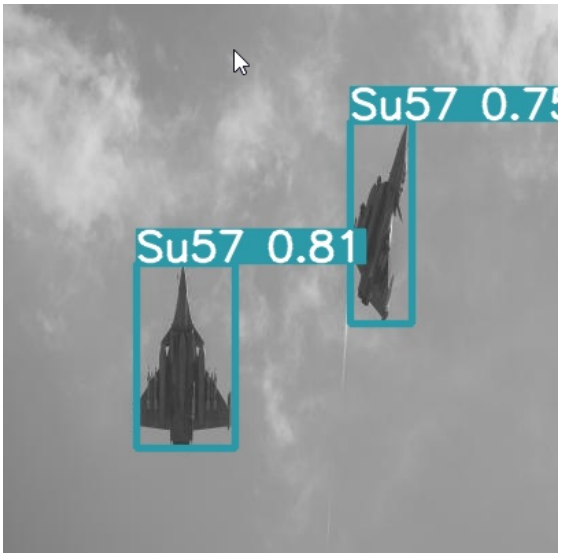


Рис. 5.6 – 5.7 – Приклади вихідних даних моделі YOLOv5n6.

Висновки

У кваліфікаційній роботі розглядається проблема швидкого виявлення об'єктів для зображення високої роздільної здатності дистанційного зондування з CNN. Модель YOLO використовується в цій роботі для виявлення об'єктів у високій роздільній здатності зображення дистанційного зондування. Експерименти з набором даних продемонстрували, що модель YOLO має сильну застосовність для зображення дистанційного зондування, особливо в швидкості розпізнавання. Однак, за малих наборів даних, неправильному розмежуванню або підборі використаної моделі – всі відомі недоліки моделі будуть видні як візуально (за результатами обробки тестового набору), так і з «сухої» статистики, особливо в порівнянні з більш потужними моделями.

Основними недоліками YOLO є погана точність позиціонування, погане навчання наближення та узагальнення для зображень незвичайне співвідношення сторін і предмети, які дуже близькі один до одного.

Для цього потрібна велика кількість високоякісних даних і анотацій, модель навчання, яка спирається на професійну інтерпретацію, досвід і багато ручної роботи. Тому вирішити ці проблеми – це орієнтація майбутніх досліджень.

Список використаних джерел

1. Penatti O A B, Nogueira K, Santos J A D, Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?, Computer Vision and Pattern Recognition Workshops. IEEE, 44-51, 2015.

2. Azayev T, Object detection in high resolution satellite images, Czech Technical University, 2016.

3. Zhong Y, Fei F, Liu Y, et al, SatCNN: satellite image dataset classification using agile convolutional neural networks, Remote Sensing Letters, Vol.8, No.2, 136-145, 2017.

4. Ball, John E., and C. S. Chan., Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community." Journal of Applied Remote Sensing 11.4(2017).

5. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. Computer Vision and Pattern Recognition (pp.779-788). IEEE.

6. Ren, Shaoqing, et al., Faster R-CNN: towards real-time object detection with region proposal networks, International Conference on Neural Information Processing Systems MIT Press, 2015:91-99.

7. Cheng G, Han J. A survey on object detection in optical remote sensing images, ISPRS Journal of Photogrammetry and Remote Sensing, Vol.117, 11-28, 2016.

8. Long, Yang, et al., Accurate Object Localization in Remote Sensing Images Based on Convolutional Neural Networks, IEEE Transactions on Geoscience & Remote Sensing 55.5(2017):2486-2498.

9. Cao, Yu She, X. Niu, and Y. Dou, Region-based convolutional neural networks for object detection in very high resolution remote sensing images, International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery IEEE, 2016:548-554.

10. Kingma D P, Ba J, Adam: A Method for Stochastic Optimization, Computer Science, 2014.

11. Wang, Jia Qi, et al, Aircraft Detection in Remote Sensing Images via CNN Multi-scale Feature Representation, smce(2017).

12. Han, Xiaobing, Y. Zhong, and L. Zhang, An Efficient and Robust Integrated Geospatial Object Detection Framework for High Spatial Resolution Remote Sensing Imagery, Remote Sensing 9.7(2017):666.

13. Shi, Shaohuai, et al, Benchmarking State-of-the-Art Deep Learning Software Tools, (2016).

14. Cheng, Gong, et al, Object detection in VHR optical remote sensing images via learning rotation-invariant HOG feature, International Workshop on Earth Observation and Remote Sensing Applications IEEE, 2016:433-436.

15. Cheng, Gong, et al, Multi-class geospatial object detection and geographic image classification based on collection of part detectors, Isprs Journal of Photogrammetry & Remote Sensing 98.1(2014):119-132.

16. Cheng, Gong, P. Zhou, and J. Han, Learning Rotation-Invariant Convolutional Neural Networks for Object Detection in VHR Optical Remote Sensing Images, IEEE Transactions on Geoscience & Remote Sensing 54.12(2016):7405-7415.

17. Бородинов А. А., Мясников В. В. Сравнение алгоритмов классификации радарных изображений при различных методах предобработки на примере базы MSTAR // Сборник трудов IV международной конференции и молодежной школы «Информационные технологии и нанотехнологии» (ИТНТ-2018) - Самара: Новая техника, 2018. С. 586-594.

18. Биленко С. В., Чередеев К. Ю., Зограбян М. К. Перспективы использования глубоких нейронных сетей в радиолокации // Вопросы радиоэлектроники. 2017. № 1. С. 57-63.

19. Yann LeCun, J. S. Denker, S. Solla, R. E. Howard and L. D. Jackel: Optimal Brain Damage, in Touretzky, David (Eds), Advances in Neural Information Processing Systems 2 (NIPS*89), Morgan Kaufman, Denver, CO, 1990

20. Y. LeCun and Y. Bengio: Convolutional Networks for Images, Speech, and Time-Series, in Arbib, M. A. (Eds), The Handbook of Brain Theory and Neural Networks, MIT Press, 1995

21. Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), Neural Networks: Tricks of the trade, Springer, 1998

22. Ranzato Marc'Aurelio, Christopher Poultney, Sumit Chopra and Yann LeCun: Efficient Learning of Sparse Representations with an Energy-Based Model, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems (NIPS 2006), MIT Press, 2006

23. YOLOv4: Optimal Speed and Accuracy of Object Detection, Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, 2020

Додаток

1. detect.py

```
import argparse
import os
import sys
from pathlib import Path

import torch
import torch.backends.cudnn as cudnn

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages,
LoadStreams
from utils.general import (LOGGER, check_file, check_img_size,
check_imshow, check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args,
scale_coords, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync


@torch.no_grad()
def run(
        weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
        source=ROOT / 'data/images',  # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
```

```
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
):
    source = str(source)
        save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
      webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
        save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
            (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
```

```
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    if webcam:
        view_img = check_imshow()
            cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1  # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt else bs, 3, *imgsz))  # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
    for path, im, im0s, vid_cap, s in dataset:
        t1 = time_sync()
        im = torch.from_numpy(im).to(device)
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
        pred = model(im, augment=augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2
```

```
        # NMS
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)
        dt[2] += time_sync() - t3

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
             txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image'
else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
                        annotator = Annotator(im0, line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
```

```
            if save_txt:  # Write to file
                                xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label
format
                    with open(f'{txt_path}.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

            if save_img or save_crop or view_img:  # Add bbox to image
                c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf else
f'{names[c]} {conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))
            if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] /
f'{p.stem}.jpg', BGR=True)

        # Stream results
        im0 = annotator.result()
        if view_img:
            cv2.imshow(str(p), im0)
            cv2.waitKey(1)  # 1 millisecond

        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
            else:  # 'video' or 'stream'
                if vid_path[i] != save_path:  # new video
                    vid_path[i] = save_path
                    if isinstance(vid_writer[i], cv2.VideoWriter):
                        vid_writer[i].release()  # release previous video writer
                    if vid_cap:  # video
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                    else:  # stream
```

```
                    fps, w, h = 30, im0.shape[1], im0.shape[0]
                    save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
                    vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer[i].write(im0)

        # Print time (inference-only)
        LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')

    # Print results
    t = tuple(x / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights)  # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
        parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path(s)')
        parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
        parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
                parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
```

```
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
```

```
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
ONNX inference')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt


def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

2. train.py

```
import argparse
import math
import os
import random
import sys
import time
from copy import deepcopy
from datetime import datetime
from pathlib import Path

import numpy as np
import torch
import torch.distributed as dist
import torch.nn as nn
import yaml
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.optim import SGD, Adam, AdamW, lr_scheduler
from tqdm import tqdm
```

```
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

import val  # for end-of-epoch mAP
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.autobatch import check_train_batch_size
from utils.callbacks import Callbacks
from utils.dataloaders import create_dataloader
from utils.downloads import attempt_download
from utils.general import (LOGGER, check_amp, check_dataset, check_file,
check_git_status, check_img_size,
                           check_requirements, check_suffix, check_version,
check_yaml, colorstr, get_latest_run,
                           increment_path, init_seeds, intersect_dicts,
labels_to_class_weights,
                           labels_to_image_weights, methods, one_cycle, print_args,
print_mutation, strip_optimizer)
from utils.loggers import Loggers
from utils.loggers.wandb.wandb_utils import check_wandb_resume
from utils.loss import ComputeLoss
from utils.metrics import fitness
from utils.plots import plot_evolve, plot_labels
from utils.torch_utils import EarlyStopping, ModelEMA, de_parallel,
select_device, torch_distributed_zero_first

LOCAL_RANK = int(os.getenv('LOCAL_RANK', -1))  #
https://pytorch.org/docs/stable/elastic/run.html
RANK = int(os.getenv('RANK', -1))
WORLD_SIZE = int(os.getenv('WORLD_SIZE', 1))
```

```
def train(hyp, opt, device, callbacks):  # hyp is path/to/hyp.yaml or hyp dictionary
    save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg, resume, noval, nosave, workers, freeze = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights, opt.single_cls, opt.evolve, opt.data, opt.cfg, \
        opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze
    callbacks.run('on_pretrain_routine_start')

    # Directories
    w = save_dir / 'weights'  # weights dir
    (w.parent if evolve else w).mkdir(parents=True, exist_ok=True)  # make dir
    last, best = w / 'last.pt', w / 'best.pt'

    # Hyperparameters
    if isinstance(hyp, str):
        with open(hyp, errors='ignore') as f:
            hyp = yaml.safe_load(f)  # load hyps dict
    LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))

    # Save run settings
    if not evolve:
        with open(save_dir / 'hyp.yaml', 'w') as f:
            yaml.safe_dump(hyp, f, sort_keys=False)
        with open(save_dir / 'opt.yaml', 'w') as f:
            yaml.safe_dump(vars(opt), f, sort_keys=False)

    # Loggers
    data_dict = None
    if RANK in {-1, 0}:
        loggers = Loggers(save_dir, weights, opt, hyp, LOGGER)  # loggers instance
        if loggers.wandb:
            data_dict = loggers.wandb.data_dict
            if resume:
```

```
            weights, epochs, hyp, batch_size = opt.weights, opt.epochs, opt.hyp,
opt.batch_size

        # Register actions
        for k in methods(loggers):
            callbacks.register_action(k, callback=getattr(loggers, k))

    # Config
    plots = not evolve and not opt.noplots  # create plots
    cuda = device.type != 'cpu'
    init_seeds(1 + RANK)
    with torch_distributed_zero_first(LOCAL_RANK):
        data_dict = data_dict or check_dataset(data)  # check if None
    train_path, val_path = data_dict['train'], data_dict['val']
    nc = 1 if single_cls else int(data_dict['nc'])  # number of classes
        names = ['item'] if single_cls and len(data_dict['names']) != 1 else
data_dict['names']  # class names
    assert len(names) == nc, f'{len(names)} names found for nc={nc} dataset in
{data}'  # check
    is_coco = isinstance(val_path, str) and val_path.endswith('coco/val2017.txt')
# COCO dataset

    # Model
    check_suffix(weights, '.pt')  # check weights
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(LOCAL_RANK):
            weights = attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location='cpu')  # load checkpoint to CPU
to avoid CUDA memory leak
                    model = Model(cfg or ckpt['model'].yaml, ch=3, nc=nc,
anchors=hyp.get('anchors')).to(device)  # create
        exclude = ['anchor'] if (cfg or hyp.get('anchors')) and not resume else []  #
exclude keys
        csd = ckpt['model'].float().state_dict()  # checkpoint state_dict as FP32
        csd = intersect_dicts(csd, model.state_dict(), exclude=exclude)  # intersect
        model.load_state_dict(csd, strict=False)  # load
```

```
        LOGGER.info(f'Transferred {len(csd)}/{len(model.state_dict())} items
from {weights}')  # report
    else:
        model = Model(cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  #
create
    amp = check_amp(model)  # check AMP

    # Freeze
        freeze = [f'model.{x}.' for x in (freeze if len(freeze) > 1 else
range(freeze[0]))]  # layers to freeze
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
            LOGGER.info(f'freezing {k}')
            v.requires_grad = False

    # Image size
    gs = max(int(model.stride.max()), 32)  # grid size (max stride)
        imgsz = check_img_size(opt.imgsz, gs, floor=gs * 2)  # verify imgsz is
gs-multiple

    # Batch size
    if RANK == -1 and batch_size == -1:  # single-GPU only, estimate best batch
size
        batch_size = check_train_batch_size(model, imgsz, amp)
        loggers.on_params_update({"batch_size": batch_size})

    # Optimizer
    nbs = 64  # nominal batch size
        accumulate = max(round(nbs / batch_size), 1)  # accumulate loss before
optimizing
    hyp['weight_decay'] *= batch_size * accumulate / nbs  # scale weight_decay
    LOGGER.info(f"Scaled weight_decay = {hyp['weight_decay']}")

    g = [], [], []  # optimizer parameter groups
     bn = tuple(v for k, v in nn.__dict__.items() if 'Norm' in k)  # normalization
layers, i.e. BatchNorm2d()
```

```
    for v in model.modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):  # bias
            g[2].append(v.bias)
        if isinstance(v, bn):  # weight (no decay)
            g[1].append(v.weight)
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):  # weight (with decay)
            g[0].append(v.weight)

    if opt.optimizer == 'Adam':
        optimizer = Adam(g[2], lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # adjust beta1 to momentum
    elif opt.optimizer == 'AdamW':
        optimizer = AdamW(g[2], lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # adjust beta1 to momentum
    else:
        optimizer = SGD(g[2], lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)

    optimizer.add_param_group({'params': g[0], 'weight_decay': hyp['weight_decay']})  # add g0 with weight_decay
    optimizer.add_param_group({'params': g[1]})  # add g1 (BatchNorm2d weights)
    LOGGER.info(f"{colorstr('optimizer:')} {type(optimizer).__name__} with parameter groups "
                f"{len(g[1])} weight (no decay), {len(g[0])} weight, {len(g[2])} bias")
    del g

    # Scheduler
    if opt.cos_lr:
        lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']
    else:
        lf = lambda x: (1 - x / epochs) * (1.0 - hyp['lrf']) + hyp['lrf']  # linear
        scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)  # plot_lr_scheduler(optimizer, scheduler, epochs)
```

```
    # EMA
    ema = ModelEMA(model) if RANK in {-1, 0} else None

    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
            best_fitness = ckpt['best_fitness']

        # EMA
        if ema and ckpt.get('ema'):
            ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
            ema.updates = ckpt['updates']

        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if resume:
                assert start_epoch > 0, f'{weights} training to {epochs} epochs is finished, nothing to resume.'
        if epochs < start_epoch:
            LOGGER.info(f"{weights} has been trained for {ckpt['epoch']} epochs. Fine-tuning for {epochs} more epochs.")
            epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, csd

    # DP mode
    if cuda and RANK == -1 and torch.cuda.device_count() > 1:
                    LOGGER.warning('WARNING: DP not recommended, use torch.distributed.run for best DDP Multi-GPU results.\n'
                                                    'See Multi-GPU Tutorial at https://github.com/ultralytics/yolov5/issues/475 to get started.')
        model = torch.nn.DataParallel(model)

    # SyncBatchNorm
```

```
    if opt.sync_bn and cuda and RANK != -1:
                                                                    model          =
torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        LOGGER.info('Using SyncBatchNorm()')

    # Trainloader
    train_loader, dataset = create_dataloader(train_path,
                                              imgsz,
                                              batch_size // WORLD_SIZE,
                                              gs,
                                              single_cls,
                                              hyp=hyp,
                                              augment=True,
                                              cache=None if opt.cache == 'val' else opt.cache,
                                              rect=opt.rect,
                                              rank=LOCAL_RANK,
                                              workers=workers,
                                              image_weights=opt.image_weights,
                                              quad=opt.quad,
                                              prefix=colorstr('train: '),
                                              shuffle=True)
    mlc = int(np.concatenate(dataset.labels, 0)[:, 0].max())  # max label class
    nb = len(train_loader)  # number of batches
    assert mlc < nc, f'Label class {mlc} exceeds nc={nc} in {data}. Possible class
labels are 0-{nc - 1}'

    # Process 0
    if RANK in {-1, 0}:
        val_loader = create_dataloader(val_path,
                                       imgsz,
                                       batch_size // WORLD_SIZE * 2,
                                       gs,
                                       single_cls,
                                       hyp=hyp,
                                       cache=None if noval else opt.cache,
                                       rect=True,
                                       rank=-1,
```

```
                                workers=workers * 2,
                                pad=0.5,
                                prefix=colorstr('val: '))[0]

        if not resume:
            labels = np.concatenate(dataset.labels, 0)
            # c = torch.tensor(labels[:, 0])  # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                plot_labels(labels, names, save_dir)

            # Anchors
            if not opt.noautoanchor:
                            check_anchors(dataset, model=model, thr=hyp['anchor_t'],
imgsz=imgsz)
            model.half().float()  # pre-reduce anchor precision

        callbacks.run('on_pretrain_routine_end')

    # DDP mode
    if cuda and RANK != -1:
        if check_version(torch.__version__, '1.11.0'):
                                model = DDP(model, device_ids=[LOCAL_RANK],
output_device=LOCAL_RANK, static_graph=True)
        else:
                                model = DDP(model, device_ids=[LOCAL_RANK],
output_device=LOCAL_RANK)

    # Model attributes
      nl = de_parallel(model).model[-1].nl  # number of detection layers (to scale
hyps)
    hyp['box'] *= 3 / nl  # scale to layers
    hyp['cls'] *= nc / 80 * 3 / nl  # scale to classes and layers
    hyp['obj'] *= (imgsz / 640) ** 2 * 3 / nl  # scale to image size and layers
    hyp['label_smoothing'] = opt.label_smoothing
    model.nc = nc  # attach number of classes to model
```

```
    model.hyp = hyp  # attach hyperparameters to model
     model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device)
* nc  # attach class weights
    model.names = names

    # Start training
    t0 = time.time()
      nw = max(round(hyp['warmup_epochs'] * nb), 100)  # number of warmup
iterations, max(3 epochs, 100 iterations)
     # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of
training
    last_opt_step = -1
    maps = np.zeros(nc)  # mAP per class
     results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box,
obj, cls)
    scheduler.last_epoch = start_epoch - 1  # do not move
    scaler = torch.cuda.amp.GradScaler(enabled=amp)
    stopper = EarlyStopping(patience=opt.patience)
    compute_loss = ComputeLoss(model)  # init loss class
    callbacks.run('on_train_start')
    LOGGER.info(f'Image sizes {imgsz} train, {imgsz} val\n'
                  f'Using {train_loader.num_workers * WORLD_SIZE} dataloader
workers\n'
             f"Logging results to {colorstr('bold', save_dir)}\n"
             f'Starting training for {epochs} epochs...')
                for epoch in range(start_epoch, epochs):        # epoch
------------------------------------------------------------------
        callbacks.run('on_train_epoch_start')
        model.train()

        # Update image weights (optional, single-GPU only)
        if opt.image_weights:
             cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # class
weights
            iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)
# image weights
```

```
                dataset.indices = random.choices(range(dataset.n), weights=iw,
k=dataset.n)  # rand weighted idx

    # Update mosaic border (optional)
    # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
    # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

    mloss = torch.zeros(3, device=device)  # mean losses
    if RANK != -1:
        train_loader.sampler.set_epoch(epoch)
    pbar = enumerate(train_loader)
     LOGGER.info(('\n' + '%10s' * 7) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls',
'labels', 'img_size'))
    if RANK in {-1, 0}:
                                        pbar = tqdm(pbar, total=nb,
bar_format='{l_bar}{bar:10}{r_bar}{bar:-10b}')  # progress bar
    optimizer.zero_grad()
                for i, (imgs, targets, paths, _) in pbar:    # batch
-------------------------------------------------------------
        callbacks.run('on_train_batch_start')
        ni = i + nb * epoch  # number integrated batches (since train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255  # uint8 to
float32, 0-255 to 0.0-1.0

        # Warmup
        if ni <= nw:
            xi = [0, nw]  # x interp
                # compute_loss.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio
(obj_loss = 1.0 or iou)
            accumulate = max(1, np.interp(ni, xi, [1, nbs / batch_size]).round())
            for j, x in enumerate(optimizer.param_groups):
                # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0,
x['initial_lr'] * lf(epoch)])
                if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'],
hyp['momentum']])
```

```
        # Multi-scale
        if opt.multi_scale:
            sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  # size
            sf = sz / max(imgs.shape[2:])  # scale factor
            if sf != 1:
                ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  # new
shape (stretched to gs-multiple)
                imgs = nn.functional.interpolate(imgs, size=ns, mode='bilinear',
align_corners=False)

        # Forward
        with torch.cuda.amp.autocast(amp):
            pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred, targets.to(device))  # loss
scaled by batch_size
            if RANK != -1:
                loss *= WORLD_SIZE  # gradient averaged between devices in
DDP mode
            if opt.quad:
                loss *= 4.

        # Backward
        scaler.scale(loss).backward()

        # Optimize
        if ni - last_opt_step >= accumulate:
            scaler.step(optimizer)  # optimizer.step
            scaler.update()
            optimizer.zero_grad()
            if ema:
                ema.update(model)
            last_opt_step = ni

        # Log
        if RANK in {-1, 0}:
            mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
```

```
                    mem = f'{torch.cuda.memory_reserved() / 1E9 if
torch.cuda.is_available() else 0:.3g}G'  # (GB)
                pbar.set_description(('%10s' * 2 + '%10.4g' * 5) %
                                     (f'{epoch}/{epochs - 1}', mem, *mloss, targets.shape[0],
imgs.shape[-1]))
                    callbacks.run('on_train_batch_end', ni, model, imgs, targets, paths,
plots)
                if callbacks.stop_training:
                    return
                                                                # end batch
-------------------------------------------------------------------------------------------------

            # Scheduler
            lr = [x['lr'] for x in optimizer.param_groups]  # for loggers
            scheduler.step()

            if RANK in {-1, 0}:
                # mAP
                callbacks.run('on_train_epoch_end', epoch=epoch)
                    ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'names', 'stride',
'class_weights'])
                final_epoch = (epoch + 1 == epochs) or stopper.possible_stop
                if not noval or final_epoch:  # Calculate mAP
                    results, maps, _ = val.run(data_dict,
                                               batch_size=batch_size // WORLD_SIZE * 2,
                                               imgsz=imgsz,
                                               model=ema.ema,
                                               single_cls=single_cls,
                                               dataloader=val_loader,
                                               save_dir=save_dir,
                                               plots=False,
                                               callbacks=callbacks,
                                               compute_loss=compute_loss)

                # Update best mAP
                  fi = fitness(np.array(results).reshape(1, -1))  # weighted combination of
[P, R, mAP@.5, mAP@.5-.95]
```

```
            if fi > best_fitness:
                best_fitness = fi
            log_vals = list(mloss) + list(results) + lr
            callbacks.run('on_fit_epoch_end', log_vals, epoch, best_fitness, fi)

            # Save model
            if (not nosave) or (final_epoch and not evolve):  # if save
                ckpt = {
                    'epoch': epoch,
                    'best_fitness': best_fitness,
                    'model': deepcopy(de_parallel(model)).half(),
                    'ema': deepcopy(ema.ema).half(),
                    'updates': ema.updates,
                    'optimizer': optimizer.state_dict(),
                         'wandb_id': loggers.wandb.wandb_run.id if loggers.wandb else
None,
                    'date': datetime.now().isoformat()}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                if opt.save_period > 0 and epoch % opt.save_period == 0:
                    torch.save(ckpt, w / f'epoch{epoch}.pt')
                del ckpt
                 callbacks.run('on_model_save', last, epoch, final_epoch, best_fitness,
fi)

            # Stop Single-GPU
            if RANK == -1 and stopper(epoch=epoch, fitness=fi):
                break

                                        # Stop DDP TODO: known issues
shttps://github.com/ultralytics/yolov5/pull/4576
            # stop = stopper(epoch=epoch, fitness=fi)
            # if RANK == 0:
            #    dist.broadcast_object_list([stop], 0)  # broadcast 'stop' to all ranks
```

```
        # Stop DPP
        # with torch_distributed_zero_first(RANK):
        # if stop:
        #    break  # must break all DDP ranks

                                                                # end epoch
----------------------------------------------------------------------------------------------------
----
                                                        # end training
----------------------------------------------------------------------------------------------------
-----
    if RANK in {-1, 0}:
            LOGGER.info(f'\n{epoch - start_epoch + 1} epochs completed in
{(time.time() - t0) / 3600:.3f} hours.')
        for f in last, best:
            if f.exists():
                strip_optimizer(f)  # strip optimizers
                if f is best:
                    LOGGER.info(f'\nValidating {f}...')
                    results, _, _ = val.run(
                        data_dict,
                        batch_size=batch_size // WORLD_SIZE * 2,
                        imgsz=imgsz,
                        model=attempt_load(f, device).half(),
                         iou_thres=0.65 if is_coco else 0.60,  # best pycocotools results at
0.65
                        single_cls=single_cls,
                        dataloader=val_loader,
                        save_dir=save_dir,
                        save_json=is_coco,
                        verbose=True,
                        plots=plots,
                        callbacks=callbacks,
                        compute_loss=compute_loss)  # val best model with plots
                    if is_coco:
```

```
                    callbacks.run('on_fit_epoch_end', list(mloss) + list(results) + lr,
epoch, best_fitness, fi)

        callbacks.run('on_train_end', last, best, plots, epoch, results)

    torch.cuda.empty_cache()
    return results


def parse_opt(known=False):
    parser = argparse.ArgumentParser()
        parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt',
help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
     parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml',
help='dataset.yaml path')
                    parser.add_argument('--hyp', type=str, default=ROOT /
'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
       parser.add_argument('--batch-size', type=int, default=16, help='total batch
size for all GPUs, -1 for autobatch')
       parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640,
help='train, val image size (pixels)')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
          parser.add_argument('--resume', nargs='?', const=True, default=False,
help='resume most recent training')
        parser.add_argument('--nosave', action='store_true', help='only save final
checkpoint')
      parser.add_argument('--noval', action='store_true', help='only validate final
epoch')
         parser.add_argument('--noautoanchor', action='store_true', help='disable
AutoAnchor')
    parser.add_argument('--noplots', action='store_true', help='save no plot files')
     parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve
hyperparameters for x generations')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
```

```
    parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
    parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
    parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')

    # Weights & Biases arguments
    parser.add_argument('--entity', default=None, help='W&B: Entity')
```

```
    parser.add_argument('--upload_dataset', nargs='?', const=True, default=False,
help='W&B: Upload data, "val" option')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set
bounding-box image logging interval')
    parser.add_argument('--artifact_alias', type=str, default='latest', help='W&B:
Version of dataset artifact to use')

   opt = parser.parse_known_args()[0] if known else parser.parse_args()
   return opt


def main(opt, callbacks=Callbacks()):
   # Checks
   if RANK in {-1, 0}:
      print_args(vars(opt))
      check_git_status()
      check_requirements(exclude=['thop'])

   # Resume
     if opt.resume and not check_wandb_resume(opt) and not opt.evolve:  #
resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  #
specified or most recent path
      assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
      with open(Path(ckpt).parent.parent / 'opt.yaml', errors='ignore') as f:
         opt = argparse.Namespace(**yaml.safe_load(f))  # replace
      opt.cfg, opt.weights, opt.resume = '', ckpt, True  # reinstate
      LOGGER.info(f'Resuming training from {ckpt}')
   else:
      opt.data, opt.cfg, opt.hyp, opt.weights, opt.project = \
               check_file(opt.data),  check_yaml(opt.cfg),  check_yaml(opt.hyp),
str(opt.weights), str(opt.project)  # checks
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be
specified'
      if opt.evolve:
            if opt.project == str(ROOT / 'runs/train'):  # if default project name,
rename to runs/evolve
```

```
            opt.project = str(ROOT / 'runs/evolve')
            opt.exist_ok, opt.resume = opt.resume, False  # pass resume to exist_ok and disable resume
        if opt.name == 'cfg':
            opt.name = Path(opt.cfg).stem  # use model.yaml as name
                opt.save_dir = str(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))

    # DDP mode
    device = select_device(opt.device, batch_size=opt.batch_size)
    if LOCAL_RANK != -1:
        msg = 'is not compatible with YOLOv5 Multi-GPU DDP training'
        assert not opt.image_weights, f'--image-weights {msg}'
        assert not opt.evolve, f'--evolve {msg}'
         assert opt.batch_size != -1, f'AutoBatch with --batch-size -1 {msg}, please pass a valid --batch-size'
                assert opt.batch_size % WORLD_SIZE == 0, f'--batch-size {opt.batch_size} must be multiple of WORLD_SIZE'
            assert torch.cuda.device_count() > LOCAL_RANK, 'insufficient CUDA devices for DDP command'
        torch.cuda.set_device(LOCAL_RANK)
        device = torch.device('cuda', LOCAL_RANK)
            dist.init_process_group(backend="nccl" if dist.is_nccl_available() else "gloo")

    # Train
    if not opt.evolve:
        train(opt.hyp, opt, device, callbacks)
        if WORLD_SIZE > 1 and RANK == 0:
            LOGGER.info('Destroying process group... ')
            dist.destroy_process_group()

    # Evolve hyperparameters (optional)
    else:
            # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
        meta = {
```

```
        'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
        'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
        'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
        'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
        'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
        'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
        'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
        'box': (1, 0.02, 0.2),  # box loss gain
        'cls': (1, 0.2, 4.0),  # cls loss gain
        'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
        'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
        'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
        'iou_t': (0, 0.1, 0.7),  # IoU training threshold
        'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
        'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
            'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default
gamma=1.5)
        'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
        'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation (fraction)
        'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
        'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
        'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
        'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
        'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
          'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction), range
0-0.001
        'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
        'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
        'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
        'mixup': (1, 0.0, 1.0),  # image mixup (probability)
        'copy_paste': (1, 0.0, 1.0)}  # segment copy-paste (probability)

    with open(opt.hyp, errors='ignore') as f:
        hyp = yaml.safe_load(f)  # load hyps dict
        if 'anchors' not in hyp:  # anchors commented in hyp.yaml
            hyp['anchors'] = 3
```

```
        opt.noval, opt.nosave, save_dir = True, True, Path(opt.save_dir)  # only
val/save final epoch
    # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable indices
        evolve_yaml, evolve_csv = save_dir / 'hyp_evolve.yaml', save_dir /
'evolve.csv'
    if opt.bucket:
            os.system(f'gsutil cp gs://{opt.bucket}/evolve.csv {evolve_csv}')  #
download evolve.csv if exists

    for _ in range(opt.evolve):  # generations to evolve
      if evolve_csv.exists():  # if evolve.csv exists: select best hyps and mutate
        # Select parent(s)
        parent = 'single'  # parent selection method: 'single' or 'weighted'
        x = np.loadtxt(evolve_csv, ndmin=2, delimiter=',', skiprows=1)
        n = min(5, len(x))  # number of previous results to consider
        x = x[np.argsort(-fitness(x))][:n]  # top n mutations
        w = fitness(x) - fitness(x).min() + 1E-6  # weights (sum > 0)
        if parent == 'single' or len(x) == 1:
          # x = x[random.randint(0, n - 1)]  # random selection
                x = x[random.choices(range(n), weights=w)[0]]  # weighted
selection
        elif parent == 'weighted':
          x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted combination

        # Mutate
        mp, s = 0.8, 0.2  # mutation probability, sigma
        npr = np.random
        npr.seed(int(time.time()))
        g = np.array([meta[k][0] for k in hyp.keys()])  # gains 0-1
        ng = len(meta)
        v = np.ones(ng)
        while all(v == 1):  # mutate until a change occurs (prevent duplicates)
            v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s
+ 1).clip(0.3, 3.0)
        for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
          hyp[k] = float(x[i + 7] * v[i])  # mutate
```

```
            # Constrain to limits
            for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1])  # lower limit
                hyp[k] = min(hyp[k], v[2])  # upper limit
                hyp[k] = round(hyp[k], 5)  # significant digits

            # Train mutation
            results = train(hyp.copy(), opt, device, callbacks)
            callbacks = Callbacks()
            # Write mutation results
            print_mutation(results, hyp.copy(), save_dir, opt.bucket)

        # Plot results
        plot_evolve(evolve_csv)
                LOGGER.info(f'Hyperparameter evolution finished {opt.evolve} generations\n'
                f"Results saved to {colorstr('bold', save_dir)}\n"
                f'Usage example: $ python train.py --hyp {evolve_yaml}')


def run(**kwargs):
            # Usage: import train; train.run(data='coco128.yaml', imgsz=320, weights='yolov5m.pt')
    opt = parse_opt(True)
    for k, v in kwargs.items():
        setattr(opt, k, v)
    main(opt)
    return opt


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

3. main.py

```
from roboflow import Roboflow
rf = Roboflow(api_key='udiai45yrT6976')
```

```
project = rf.workspace().project('military-tanks')
dataset = project.version(1).download('yolov5')
```

```
os.system('python train.py —img 416 —batch 16 —epochs 100 —data "dataset/data.yaml" —weights yolov5n6.pt —cache')
```

```
os.system('python detect.py —conf 0.5 —img 416 —source "dataset/valid/images" —weights "runs/train/exp/weights/best.pt"')
```