**detect.py**

import argparse

import time

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

from email.mime.application import MIMEApplication

from pathlib import Path

import cv2

import torch

import torch.backends.cudnn as cudnn

from numpy import random

from models.experimental import attempt\_load

from utils.datasets import LoadStreams, LoadImages

from utils.general import check\_img\_size, check\_requirements, check\_imshow, non\_max\_suppression, apply\_classifier, \

    scale\_coords, xyxy2xywh, strip\_optimizer, set\_logging, increment\_path

from utils.plots import plot\_one\_box

from utils.torch\_utils import select\_device, load\_classifier, time\_synchronized

# Str sss = ""

def detect(save\_img=False):

    source, weights, view\_img, save\_txt, imgsz = opt.source, opt.weights, opt.view\_img, opt.save\_txt, opt.img\_size

    save\_img = not opt.nosave and not source.endswith(

        '.txt')  # 保存預測影像

    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(

        ('rtsp://', 'rtmp://', 'http://'))

    # 目錄

    save\_dir = Path(increment\_path(Path(opt.project) / opt.name,

                                   exist\_ok=opt.exist\_ok))  # 批量執行

    (save\_dir / 'labels' if save\_txt else save\_dir).mkdir(parents=True,

                                                          exist\_ok=True)  # 建立目錄

    # 初始化

    set\_logging()

    device = select\_device(opt.device)

    half = device.type != 'cpu'  # 半精度僅在CUDA上支援

    # 讀取模型

    model = attempt\_load(weights, map\_location=device)  # 讀取 FP32 模型

    stride = int(model.stride.max())  # 模型發展

    imgsz = check\_img\_size(imgsz, s=stride)  # 檢查圖片大小

    if half:

        model.half()  # 至 FP16

    # 二階段分類器

    classify = False

    if classify:

        modelc = load\_classifier(name='resnet101', n=2)  # 初始化

        modelc.load\_state\_dict(torch.load(

            'weights/resnet101.pt', map\_location=device)['model']).to(device).eval()

    # 設置資料讀取器

    vid\_path, vid\_writer = None, None

    if webcam:

        view\_img = check\_imshow()

        cudnn.benchmark = True  # 設為 True 來加快恆定的圖像尺寸推斷

        dataset = LoadStreams(source, img\_size=imgsz, stride=stride)

    else:

        dataset = LoadImages(source, img\_size=imgsz, stride=stride)

    # 取得名字與顏色

    names = model.module.names if hasattr(model, 'module') else model.names

    colors = [[random.randint(0, 255) for \_ in range(3)] for \_ in names]

    # 執行預測

    if device.type != 'cpu':

        model(torch.zeros(1, 3, imgsz, imgsz).to(device).type\_as(

            next(model.parameters())))  # 執行一次

    t0 = time.time()

    for path, img, im0s, vid\_cap in dataset:

        img = torch.from\_numpy(img).to(device)

        img = img.half() if half else img.float()

        img /= 255.0  # 0 - 255 to 0.0 - 1.0

        if img.ndimension() == 3:

            img = img.unsqueeze(0)

        # 預測

        t1 = time\_synchronized()

        pred = model(img, augment=opt.augment)[0]

        # 套用 NMS

        pred = non\_max\_suppression(

            pred, opt.conf\_thres, opt.iou\_thres, classes=opt.classes, agnostic=opt.agnostic\_nms)

        t2 = time\_synchronized()

        # 套用分類器

        if classify:

            pred = apply\_classifier(pred, modelc, img, im0s)

        # 執行偵測

        for i, det in enumerate(pred):  # 依每張圖像進行偵測

            if webcam:  # batch\_size >= 1

                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(

                ), dataset.count

            else:

                p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

            p = Path(p)  # 至 Path

            save\_path = str(save\_dir / p.name)  # img.jpg

            txt\_path = str(save\_dir / 'labels' / p.stem) + \

                ('' if dataset.mode == 'image' else f'\_{frame}')  # img.txt

            s += '%gx%g ' % img.shape[2:]  # 列印字串

            # 正規化

            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]

            if len(det):

                # 將boxes從img\_size縮放到im0大小

                det[:, :4] = scale\_coords(

                    img.shape[2:], det[:, :4], im0.shape).round()

                # 列印結果

                for c in det[:, -1].unique():

                    n = (det[:, -1] == c).sum()  # 依每個class執行偵測

                    # 增加到字串

                    s += f"{n} {names[int(c)]}{'s' \* (n > 1)}, "

                # 輸出成果

                for \*xyxy, conf, cls in reversed(det):

                    if save\_txt:  # 輸出到檔案

                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)

                                          ) / gn).view(-1).tolist()  # 正規化 xywh

                        # 格式化字串

                        line = (

                            cls, \*xywh, conf) if opt.save\_conf else (cls, \*xywh)

                        with open(txt\_path + '.txt', 'a') as f:

                            f.write(('%g ' \* len(line)).rstrip() % line + '\n')

                    if save\_img or view\_img:  # 將 bbox 加到 image

                        label = f'{names[int(cls)]} {conf:.2f}'

                        plot\_one\_box(xyxy, im0, label=label,

                                     color=colors[int(cls)], line\_thickness=3)

            # 列印時間(預測 + NMS)

            print(f'{s}Done. ({t2 - t1:.3f}s)')

            # 播放成果

            if view\_img:

                cv2.imshow(str(p), im0)

                cv2.waitKey(1)  # 1 毫秒

            # 儲存成果 (推斷後的圖片)

            if save\_img:

                if dataset.mode == 'image':

                    cv2.imwrite(save\_path, im0)

                else:  # 'video' 或 'stream'

                    if vid\_path != save\_path:  # 新影片

                        vid\_path = save\_path

                        if isinstance(vid\_writer, cv2.VideoWriter):

                            vid\_writer.release()  # 釋出上一個影片寫入

                        if vid\_cap:  # 影片

                            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:  # 播放

                            fps, w, h = 30, im0.shape[1], im0.shape[0]

                            save\_path += '.mp4'

                        vid\_writer = cv2.VideoWriter(

                            save\_path, cv2.VideoWriter\_fourcc(\*'mp4v'), fps, (w, h))

                    vid\_writer.write(im0)

    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 ''

        print(f"Results saved to {save\_dir}{s}")

    print(f'Done. ({time.time() - t0:.3f}s)')

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('--weights', nargs='+', type=str,

                        default='yolov5s.pt', help='model.pt path(s)')

    # file/folder, 0 for webcam

    parser.add\_argument('--source', type=str,

                        default='data/images', help='source')

    parser.add\_argument('--img-size', type=int, default=640,

                        help='inference size (pixels)')

    parser.add\_argument('--conf-thres', type=float,

                        default=0.25, help='object confidence threshold')

    parser.add\_argument('--iou-thres', type=float,

                        default=0.45, help='IOU threshold for NMS')

    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='display 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('--nosave', action='store\_true',

                        help='do not save images/videos')

    parser.add\_argument('--classes', nargs='+', type=int,

                        help='filter by class: --class 0, or --class 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('--update', action='store\_true',

                        help='update all models')

    parser.add\_argument('--project', default='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')

    opt = parser.parse\_args()

    print(opt)

    check\_requirements(exclude=('pycocotools', 'thop'))

    with torch.no\_grad():

        if opt.update:  # 更新所有模型 (來解決來源變更警告)

            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:

                detect()

                strip\_optimizer(opt.weights)

        else:

            detect()

print("請輸入save\_dir/檔名")

emailpath = input()

slen = len(emailpath) - 3

receiver = "goodspeed36121214@gmail.com"

sender = "C107118151@nkust.edu.tw"

pwd = "goodspeed"

msg = MIMEMultipart()

msg["Subject"] = "偵測結果"  # 郵件的主題

msg["From"] = sender

msg["To"] = receiver

part = MIMEText("結果如下")  # 郵件的正文

msg.attach(part)

if(emailpath[slen:] == "jpg"):

    # jpg型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.jpg")

    msg.attach(part)

elif(emailpath[slen:] == "png"):

    # png型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.png")

    msg.attach(part)

elif(emailpath[slen:] == "bmp"):

    # bmp型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.bmp")

    msg.attach(part)

elif(emailpath[slen:] == "mp4"):

    # mp4型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.mp4")

    msg.attach(part)

elif(emailpath[(slen-1):] == "jpeg"):

    # jpeg型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.jpeg")

    msg.attach(part)

elif(emailpath[slen:] == "wmv"):

    # wmv型別附件

    part = MIMEApplication(open(emailpath, 'rb').read())

    part.add\_header('Content-Disposition', 'attachment', filename="result.wmv")

    msg.attach(part)

try:

    s = smtplib.SMTP("smtp.gmail.com", timeout=30)  # 連線smtp郵件伺服器,埠預設是25

    s.ehlo()

    s.starttls()

    s.login(sender, pwd)  # 登陸伺服器

    s.sendmail(sender, receiver, msg.as\_string())  # 傳送郵件

    s.close()

    print('郵件傳送成功！')

except smtplib.SMTPException:

    print('郵件傳送失敗！')

**test.py**

import argparse

import json

import os

from pathlib import Path

from threading import Thread

import numpy as np

import torch

import yaml

from tqdm import tqdm

from models.experimental import attempt\_load

from utils.datasets import create\_dataloader

from utils.general import coco80\_to\_coco91\_class, check\_dataset, check\_file, check\_img\_size, check\_requirements, \

    box\_iou, non\_max\_suppression, scale\_coords, xyxy2xywh, xywh2xyxy, set\_logging, increment\_path, colorstr

from utils.metrics import ap\_per\_class, ConfusionMatrix

from utils.plots import plot\_images, output\_to\_target, plot\_study\_txt

from utils.torch\_utils import select\_device, time\_synchronized

def test(data,

         weights=None,

         batch\_size=32,

         imgsz=640,

         conf\_thres=0.001,

         iou\_thres=0.6,  # for NMS

         save\_json=False,

         single\_cls=False,

         augment=False,

         verbose=False,

         model=None,

         dataloader=None,

         save\_dir=Path(''),  # for 儲存圖片

         save\_txt=False,  # for 自動標記

         save\_hybrid=False,  # for 混合自動標記

         save\_conf=False,  # 保存自動標記的可信度

         plots=True,

         log\_imgs=0,  # 記錄的圖像數

         compute\_loss=None):

    # 初始化/加載模型並設置設備

    training = model is not None

    if training:  # 由train.py呼叫

        device = next(model.parameters()).device  # 獲取模型設備

    else:  # 直接呼叫

        set\_logging()

        device = select\_device(opt.device, batch\_size=batch\_size)

        # 目錄

        save\_dir = Path(increment\_path(Path(opt.project) / opt.name, exist\_ok=opt.exist\_ok))  # 批量執行

        (save\_dir / 'labels' if save\_txt else save\_dir).mkdir(parents=True, exist\_ok=True)  # 生成目錄

        # 讀取模型

        model = attempt\_load(weights, map\_location=device)  # 讀取 FP32 模型

        gs = max(int(model.stride.max()), 32)  # 網格大小 (max stride)

        imgsz = check\_img\_size(imgsz, s=gs)  # 檢查 img\_size

        # 禁用多GPU，與.half()不兼容 https://github.com/ultralytics/yolov5/issues/99

        # if device.type != 'cpu' and torch.cuda.device\_count() > 1:

        #     model = nn.DataParallel(model)

    # 半精度

    half = device.type != 'cpu'  # 半精度僅在CUDA上支援

    if half:

        model.half()

    # 配置

    model.eval()

    is\_coco = data.endswith('coco.yaml')  # is COCO dataset

    with open(data) as f:

        data = yaml.load(f, Loader=yaml.SafeLoader)  # 模型字典

    check\_dataset(data)  # 檢查

    nc = 1 if single\_cls else int(data['nc'])  # classes數量

    iouv = torch.linspace(0.5, 0.95, 10).to(device)  # iou vector for mAP@0.5:0.95

    niou = iouv.numel()

    # 記錄

    log\_imgs, wandb = min(log\_imgs, 100), None  # ceil

    try:

        import wandb  # Weights & Biases

    except ImportError:

        log\_imgs = 0

    # 資料讀取器

    if not training:

        if device.type != 'cpu':

            model(torch.zeros(1, 3, imgsz, imgsz).to(device).type\_as(next(model.parameters())))  # 執行一次

        path = data['test'] if opt.task == 'test' else data['val']  # path to val/test images

        dataloader = create\_dataloader(path, imgsz, batch\_size, gs, opt, pad=0.5, rect=True,

                                       prefix=colorstr('test: ' if opt.task == 'test' else 'val: '))[0]

    seen = 0

    confusion\_matrix = ConfusionMatrix(nc=nc)

    names = {k: v for k, v in enumerate(model.names if hasattr(model, 'names') else model.module.names)}

    coco91class = coco80\_to\_coco91\_class()

    s = ('%20s' + '%12s' \* 6) % ('Class', 'Images', 'Targets', 'P', 'R', 'mAP@.5', 'mAP@.5:.95')

    p, r, f1, mp, mr, map50, map, t0, t1 = 0., 0., 0., 0., 0., 0., 0., 0., 0.

    loss = torch.zeros(3, device=device)

    jdict, stats, ap, ap\_class, wandb\_images = [], [], [], [], []

    for batch\_i, (img, targets, paths, shapes) in enumerate(tqdm(dataloader, desc=s)):

        img = img.to(device, non\_blocking=True)

        img = img.half() if half else img.float()  # uint8 to fp16/32

        img /= 255.0  # 0 - 255 to 0.0 - 1.0

        targets = targets.to(device)

        nb, \_, height, width = img.shape  # batch size, channels, height, width

        with torch.no\_grad():

            # 執行模型

            t = time\_synchronized()

            out, train\_out = model(img, augment=augment)  # 推斷與訓練的輸出

            t0 += time\_synchronized() - t

            # 計算loss

            if compute\_loss:

                loss += compute\_loss([x.float() for x in train\_out], targets)[1][:3]  # box, obj, cls

            # 執行 NMS

            targets[:, 2:] \*= torch.Tensor([width, height, width, height]).to(device)  # to pixels

            lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if save\_hybrid else []  # for 自動標記

            t = time\_synchronized()

            out = non\_max\_suppression(out, conf\_thres=conf\_thres, iou\_thres=iou\_thres, labels=lb, multi\_label=True)

            t1 += time\_synchronized() - t

        # 依每張圖片進行統計

        for si, pred in enumerate(out):

            labels = targets[targets[:, 0] == si, 1:]

            nl = len(labels)

            tcls = labels[:, 0].tolist() if nl else []  # target class

            path = Path(paths[si])

            seen += 1

            if len(pred) == 0:

                if nl:

                    stats.append((torch.zeros(0, niou, dtype=torch.bool), torch.Tensor(), torch.Tensor(), tcls))

                continue

            # 預測

            predn = pred.clone()

            scale\_coords(img[si].shape[1:], predn[:, :4], shapes[si][0], shapes[si][1])  # 本機空間預測

            # 附加到文本文件

            if save\_txt:

                gn = torch.tensor(shapes[si][0])[[1, 0, 1, 0]]  # 正規化 whwh

                for \*xyxy, conf, cls in predn.tolist():

                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # 正規化 xywh

                    line = (cls, \*xywh, conf) if save\_conf else (cls, \*xywh)  # 字串格式化

                    with open(save\_dir / 'labels' / (path.stem + '.txt'), 'a') as f:

                        f.write(('%g ' \* len(line)).rstrip() % line + '\n')

            # W&B logging

            if plots and len(wandb\_images) < log\_imgs:

                box\_data = [{"position": {"minX": xyxy[0], "minY": xyxy[1], "maxX": xyxy[2], "maxY": xyxy[3]},

                             "class\_id": int(cls),

                             "box\_caption": "%s %.3f" % (names[cls], conf),

                             "scores": {"class\_score": conf},

                             "domain": "pixel"} for \*xyxy, conf, cls in pred.tolist()]

                boxes = {"predictions": {"box\_data": box\_data, "class\_labels": names}}  # 推斷空間

                wandb\_images.append(wandb.Image(img[si], boxes=boxes, caption=path.name))

            # 附加到pycocotools JSON字典

            if save\_json:

                # [{"image\_id": 42, "category\_id": 18, "bbox": [258.15, 41.29, 348.26, 243.78], "score": 0.236}, ...

                image\_id = int(path.stem) if path.stem.isnumeric() else path.stem

                box = xyxy2xywh(predn[:, :4])  # xywh

                box[:, :2] -= box[:, 2:] / 2  # xy 中心到左上角

                for p, b in zip(pred.tolist(), box.tolist()):

                    jdict.append({'image\_id': image\_id,

                                  'category\_id': coco91class[int(p[5])] if is\_coco else int(p[5]),

                                  'bbox': [round(x, 3) for x in b],

                                  'score': round(p[4], 5)})

            # 將所有預測分指定為不正確

            correct = torch.zeros(pred.shape[0], niou, dtype=torch.bool, device=device)

            if nl:

                detected = []  # 指定指標

                tcls\_tensor = labels[:, 0]

                # 指定 boxes

                tbox = xywh2xyxy(labels[:, 1:5])

                scale\_coords(img[si].shape[1:], tbox, shapes[si][0], shapes[si][1])  # 本機標籤

                if plots:

                    confusion\_matrix.process\_batch(predn, torch.cat((labels[:, 0:1], tbox), 1))

                # Per target class

                for cls in torch.unique(tcls\_tensor):

                    ti = (cls == tcls\_tensor).nonzero(as\_tuple=False).view(-1)  # 預測指標

                    pi = (cls == pred[:, 5]).nonzero(as\_tuple=False).view(-1)  # 指定指標

                    # 搜索偵測

                    if pi.shape[0]:

                        # 指定 ious 的預測

                        ious, i = box\_iou(predn[pi, :4], tbox[ti]).max(1)  # best ious, indices

                        # 追加偵測

                        detected\_set = set()

                        for j in (ious > iouv[0]).nonzero(as\_tuple=False):

                            d = ti[i[j]]  # 目標偵測

                            if d.item() not in detected\_set:

                                detected\_set.add(d.item())

                                detected.append(d)

                                correct[pi[j]] = ious[j] > iouv  # iou\_thres is 1xn

                                if len(detected) == nl:  # 全目標已在圖片中定位

                                    break

            # 追加統計數據 (correct, conf, pcls, tcls)

            stats.append((correct.cpu(), pred[:, 4].cpu(), pred[:, 5].cpu(), tcls))

        # 繪製圖像

        if plots and batch\_i < 3:

            f = save\_dir / f'test\_batch{batch\_i}\_labels.jpg'  # labels

            Thread(target=plot\_images, args=(img, targets, paths, f, names), daemon=True).start()

            f = save\_dir / f'test\_batch{batch\_i}\_pred.jpg'  # 預測

            Thread(target=plot\_images, args=(img, output\_to\_target(out), paths, f, names), daemon=True).start()

    # 計算統計數據

    stats = [np.concatenate(x, 0) for x in zip(\*stats)]  # to numpy

    if len(stats) and stats[0].any():

        p, r, ap, f1, ap\_class = ap\_per\_class(\*stats, plot=plots, save\_dir=save\_dir, names=names)

        ap50, ap = ap[:, 0], ap.mean(1)  # AP@0.5, AP@0.5:0.95

        mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.mean()

        nt = np.bincount(stats[3].astype(np.int64), minlength=nc)  # 每個class的目標數量

    else:

        nt = torch.zeros(1)

    # 列印成果

    pf = '%20s' + '%12.3g' \* 6  # 格式化列印

    print(pf % ('all', seen, nt.sum(), mp, mr, map50, map))

    # 依每個class列印成果

    if (verbose or (nc < 50 and not training)) and nc > 1 and len(stats):

        for i, c in enumerate(ap\_class):

            print(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i], ap[i]))

    # 列印速度

    t = tuple(x / seen \* 1E3 for x in (t0, t1, t0 + t1)) + (imgsz, imgsz, batch\_size)  # tuple

    if not training:

        print('Speed: %.1f/%.1f/%.1f ms inference/NMS/total per %gx%g image at batch-size %g' % t)

    # 繪製

    if plots:

        confusion\_matrix.plot(save\_dir=save\_dir, names=list(names.values()))

        if wandb and wandb.run:

            val\_batches = [wandb.Image(str(f), caption=f.name) for f in sorted(save\_dir.glob('test\*.jpg'))]

            wandb.log({"Images": wandb\_images, "Validation": val\_batches}, commit=False)

    # 儲存JSON

    if save\_json and len(jdict):

        w = Path(weights[0] if isinstance(weights, list) else weights).stem if weights is not None else ''  # 權重

        anno\_json = '../coco/annotations/instances\_val2017.json'  # 註解json

        pred\_json = str(save\_dir / f"{w}\_predictions.json")  # 預測json

        print('\nEvaluating pycocotools mAP... saving %s...' % pred\_json)

        with open(pred\_json, 'w') as f:

            json.dump(jdict, f)

        try:  # https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb

            from pycocotools.coco import COCO

            from pycocotools.cocoeval import COCOeval

            anno = COCO(anno\_json)  # 初始化註釋api

            pred = anno.loadRes(pred\_json)  # 初始化預測 api

            eval = COCOeval(anno, pred, 'bbox')

            if is\_coco:

                eval.params.imgIds = [int(Path(x).stem) for x in dataloader.dataset.img\_files]  # 要評估的圖片ID

            eval.evaluate()

            eval.accumulate()

            eval.summarize()

            map, map50 = eval.stats[:2]  # 更新成果(mAP@0.5:0.95, mAP@0.5)

        except Exception as e:

            print(f'pycocotools unable to run: {e}')

    # 回傳成果

    model.float()  # 訓練用

    if not training:

        s = f"\n{len(list(save\_dir.glob('labels/\*.txt')))} labels saved to {save\_dir / 'labels'}" if save\_txt else ''

        print(f"Results saved to {save\_dir}{s}")

    maps = np.zeros(nc) + map

    for i, c in enumerate(ap\_class):

        maps[c] = ap[i]

    return (mp, mr, map50, map, \*(loss.cpu() / len(dataloader)).tolist()), maps, t

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser(prog='test.py')

    parser.add\_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')

    parser.add\_argument('--data', type=str, default='data/coco128.yaml', help='\*.data path')

    parser.add\_argument('--batch-size', type=int, default=32, help='size of each image batch')

    parser.add\_argument('--img-size', type=int, default=640, help='inference size (pixels)')

    parser.add\_argument('--conf-thres', type=float, default=0.001, help='object confidence threshold')

    parser.add\_argument('--iou-thres', type=float, default=0.6, help='IOU threshold for NMS')

    parser.add\_argument('--task', default='val', help="'val', 'test', 'study'")

    parser.add\_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')

    parser.add\_argument('--single-cls', action='store\_true', help='treat as single-class dataset')

    parser.add\_argument('--augment', action='store\_true', help='augmented inference')

    parser.add\_argument('--verbose', action='store\_true', help='report mAP by class')

    parser.add\_argument('--save-txt', action='store\_true', help='save results to \*.txt')

    parser.add\_argument('--save-hybrid', action='store\_true', help='save label+prediction hybrid results to \*.txt')

    parser.add\_argument('--save-conf', action='store\_true', help='save confidences in --save-txt labels')

    parser.add\_argument('--save-json', action='store\_true', help='save a cocoapi-compatible JSON results file')

    parser.add\_argument('--project', default='runs/test', 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')

    opt = parser.parse\_args()

    opt.save\_json |= opt.data.endswith('coco.yaml')

    opt.data = check\_file(opt.data)  # 檢查檔案

    print(opt)

    check\_requirements()

    if opt.task in ['val', 'test']:  # 正常運行

        test(opt.data,

             opt.weights,

             opt.batch\_size,

             opt.img\_size,

             opt.conf\_thres,

             opt.iou\_thres,

             opt.save\_json,

             opt.single\_cls,

             opt.augment,

             opt.verbose,

             save\_txt=opt.save\_txt | opt.save\_hybrid,

             save\_hybrid=opt.save\_hybrid,

             save\_conf=opt.save\_conf,

             )

    elif opt.task == 'speed':  # 速度基準

        for w in opt.weights:

            test(opt.data, w, opt.batch\_size, opt.img\_size, 0.25, 0.45, save\_json=False, plots=False)

    elif opt.task == 'study':  # 運行一系列設置並保存/繪製

        # python test.py --task study --data coco.yaml --iou 0.7 --weights yolov5s.pt yolov5m.pt yolov5l.pt yolov5x.pt

        x = list(range(256, 1536 + 128, 128))  # x 軸 (image sizes)

        for w in opt.weights:

            f = f'study\_{Path(opt.data).stem}\_{Path(w).stem}.txt'  # 欲儲存的檔案名稱

            y = []  # y 軸

            for i in x:  # img-size

                print(f'\nRunning {f} point {i}...')

                r, \_, t = test(opt.data, w, opt.batch\_size, i, opt.conf\_thres, opt.iou\_thres, opt.save\_json,

                               plots=False)

                y.append(r + t)  # 成果及運行時間

            np.savetxt(f, y, fmt='%10.4g')  # 儲存

        os.system('zip -r study.zip study\_\*.txt')

        plot\_study\_txt(x=x)  # 繪製

train.py

import argparse

import logging

import math

import os

import random

import time

from copy import deepcopy

from pathlib import Path

from threading import Thread

import numpy as np

import torch.distributed as dist

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import torch.optim.lr\_scheduler as lr\_scheduler

import torch.utils.data

import yaml

from torch.cuda import amp

from torch.nn.parallel import DistributedDataParallel as DDP

from torch.utils.tensorboard import SummaryWriter

from tqdm import tqdm

import test  # 導入 test.py 以在每個時程後獲得mAP

from models.experimental import attempt\_load

from models.yolo import Model

from utils.autoanchor import check\_anchors

from utils.datasets import create\_dataloader

from utils.general import labels\_to\_class\_weights, increment\_path, labels\_to\_image\_weights, init\_seeds, \

    fitness, strip\_optimizer, get\_latest\_run, check\_dataset, check\_file, check\_git\_status, check\_img\_size, \

    check\_requirements, print\_mutation, set\_logging, one\_cycle, colorstr

from utils.google\_utils import attempt\_download

from utils.loss import ComputeLoss

from utils.plots import plot\_images, plot\_labels, plot\_results, plot\_evolution

from utils.torch\_utils import ModelEMA, select\_device, intersect\_dicts, torch\_distributed\_zero\_first, is\_parallel

logger = logging.getLogger(\_\_name\_\_)

def train(hyp, opt, device, tb\_writer=None, wandb=None):

    logger.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))

    save\_dir, epochs, batch\_size, total\_batch\_size, weights, rank = \

        Path(opt.save\_dir), opt.epochs, opt.batch\_size, opt.total\_batch\_size, opt.weights, opt.global\_rank

    # 目錄

    wdir = save\_dir / 'weights'

    wdir.mkdir(parents=True, exist\_ok=True)  # 創造目錄

    last = wdir / 'last.pt'

    best = wdir / 'best.pt'

    results\_file = save\_dir / 'results.txt'

    # 儲存運行設定

    with open(save\_dir / 'hyp.yaml', 'w') as f:

        yaml.dump(hyp, f, sort\_keys=False)

    with open(save\_dir / 'opt.yaml', 'w') as f:

        yaml.dump(vars(opt), f, sort\_keys=False)

    # 配置

    plots = not opt.evolve  # 創建繪製

    cuda = device.type != 'cpu'

    init\_seeds(2 + rank)

    with open(opt.data) as f:

        data\_dict = yaml.load(f, Loader=yaml.SafeLoader)  # 數據dict

    with torch\_distributed\_zero\_first(rank):

        check\_dataset(data\_dict)  # 檢查

    train\_path = data\_dict['train']

    test\_path = data\_dict['val']

    nc = 1 if opt.single\_cls else int(data\_dict['nc'])  #  classes數量

    names = ['item'] if opt.single\_cls and len(data\_dict['names']) != 1 else data\_dict['names']  # class名稱

    assert len(names) == nc, '%g names found for nc=%g dataset in %s' % (len(names), nc, opt.data)  # 檢查

    # 模型

    pretrained = weights.endswith('.pt')

    if pretrained:

        with torch\_distributed\_zero\_first(rank):

            attempt\_download(weights)  # 在本機未尋獲時下載

        ckpt = torch.load(weights, map\_location=device)  # 讀取檢查點

        if hyp.get('anchors'):

            ckpt['model'].yaml['anchors'] = round(hyp['anchors'])  # 強制自動錨定

        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc).to(device)  # 創建

        exclude = ['anchor'] if opt.cfg or hyp.get('anchors') else []  # 排除鍵

        state\_dict = ckpt['model'].float().state\_dict()  # to FP32

        state\_dict = intersect\_dicts(state\_dict, model.state\_dict(), exclude=exclude)  # 合流

        model.load\_state\_dict(state\_dict, strict=False)  # 讀取

        logger.info('Transferred %g/%g items from %s' % (len(state\_dict), len(model.state\_dict()), weights))  # 回報

    else:

        model = Model(opt.cfg, ch=3, nc=nc).to(device)  # 創建

    # 固定

    freeze = []  # 要固定的參數名稱(完整或部分)

    for k, v in model.named\_parameters():

        v.requires\_grad = True  # 訓練所有層

        if any(x in k for x in freeze):

            print('freezing %s' % k)

            v.requires\_grad = False

    # 優化

    nbs = 64  # 虛設批量大小

    accumulate = max(round(nbs / total\_batch\_size), 1)  # 優化前累積漏失

    hyp['weight\_decay'] \*= total\_batch\_size \* accumulate / nbs  # 計算權重衰減

    logger.info(f"Scaled weight\_decay = {hyp['weight\_decay']}")

    pg0, pg1, pg2 = [], [], []  # 優化器參數組

    for k, v in model.named\_modules():

        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):

            pg2.append(v.bias)  # 指標

        if isinstance(v, nn.BatchNorm2d):

            pg0.append(v.weight)  # 無衰減

        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):

            pg1.append(v.weight)  # 套用衰減

    if opt.adam:

        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # 將beta1調整為動量

    else:

        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)

    optimizer.add\_param\_group({'params': pg1, 'weight\_decay': hyp['weight\_decay']})  # 添加具有權重衰減的pg1

    optimizer.add\_param\_group({'params': pg2})  # 添加pg2 (指標)

    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2), len(pg1), len(pg0)))

    del pg0, pg1, pg2

    # 排程器 https://arxiv.org/pdf/1812.01187.pdf

    # https://pytorch.org/docs/stable/\_modules/torch/optim/lr\_scheduler.html#OneCycleLR

    if opt.linear\_lr:

        lf = lambda x: (1 - x / (epochs - 1)) \* (1.0 - hyp['lrf']) + hyp['lrf']  # 線性

    else:

        lf = one\_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']

    scheduler = lr\_scheduler.LambdaLR(optimizer, lr\_lambda=lf)

    # 繪製lr排程器(優化器, 排程器,時程)

    # 記錄器

    if rank in [-1, 0] and wandb and wandb.run is None:

        opt.hyp = hyp  # 添加超參數

        wandb\_run = wandb.init(config=opt, resume="allow",

                               project='YOLOv5' if opt.project == 'runs/train' else Path(opt.project).stem,

                               name=save\_dir.stem,

                               entity=opt.entity,

                               id=ckpt.get('wandb\_id') if 'ckpt' in locals() else None)

    loggers = {'wandb': wandb}  # 記錄器字典

    # EMA

    ema = ModelEMA(model) if rank in [-1, 0] else None

    # Recovery

    start\_epoch, best\_fitness = 0, 0.0

    if pretrained:

        # 優化器

        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'][0].float().state\_dict())

            ema.updates = ckpt['ema'][1]

        # 成果

        if ckpt.get('training\_results') is not None:

            results\_file.write\_text(ckpt['training\_results'])  # 寫入results.txt

        # 時程

        start\_epoch = ckpt['epoch'] + 1

        if opt.resume:

            assert start\_epoch > 0, '%s training to %g epochs is finished, nothing to resume.' % (weights, epochs)

        if epochs < start\_epoch:

            logger.info('%s has been trained for %g epochs. Fine-tuning for %g additional epochs.' %

                        (weights, ckpt['epoch'], epochs))

            epochs += ckpt['epoch']  # 微調其他時程

        del ckpt, state\_dict

    # 圖片大小

    gs = max(int(model.stride.max()), 32)  # 網格大小(最大跨度)

    nl = model.model[-1].nl  # 檢測層數(used for scaling hyp['obj'])

    imgsz, imgsz\_test = [check\_img\_size(x, gs) for x in opt.img\_size]  # 驗證imgsz是gs-multiples

    # DP模式

    if cuda and rank == -1 and torch.cuda.device\_count() > 1:

        model = torch.nn.DataParallel(model)

    # 同步批量處理規範

    if opt.sync\_bn and cuda and rank != -1:

        model = torch.nn.SyncBatchNorm.convert\_sync\_batchnorm(model).to(device)

        logger.info('Using SyncBatchNorm()')

    # DDP模式

    if cuda and rank != -1:

        model = DDP(model, device\_ids=[opt.local\_rank], output\_device=opt.local\_rank)

    # 訓練讀取器

    dataloader, dataset = create\_dataloader(train\_path, imgsz, batch\_size, gs, opt,

                                            hyp=hyp, augment=True, cache=opt.cache\_images, rect=opt.rect, rank=rank,

                                            world\_size=opt.world\_size, workers=opt.workers,

                                            image\_weights=opt.image\_weights, quad=opt.quad, prefix=colorstr('train: '))

    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # 最大標籤class

    nb = len(dataloader)  # 批量數量

    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels are 0-%g' % (mlc, nc, opt.data, nc - 1)

    # Process 0

    if rank in [-1, 0]:

        testloader = create\_dataloader(test\_path, imgsz\_test, batch\_size \* 2, gs, opt,  # 測試加載器

                                       hyp=hyp, cache=opt.cache\_images and not opt.notest, rect=True, rank=-1,

                                       world\_size=opt.world\_size, workers=opt.workers,

                                       pad=0.5, prefix=colorstr('val: '))[0]

        if not opt.resume:

            labels = np.concatenate(dataset.labels, 0)

            c = torch.tensor(labels[:, 0])  # classes

            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # 頻率

            # model.\_initialize\_biases(cf.to(device))

            if plots:

                plot\_labels(labels, save\_dir, loggers)

                if tb\_writer:

                    tb\_writer.add\_histogram('classes', c, 0)

            # 錨點

            if not opt.noautoanchor:

                check\_anchors(dataset, model=model, thr=hyp['anchor\_t'], imgsz=imgsz)

    # 模組參數

    hyp['box'] \*= 3. / nl  # 縮放到圖層

    hyp['cls'] \*= nc / 80. \* 3. / nl  # 縮放到classes和圖層

    hyp['obj'] \*= (imgsz / 640) \*\* 2 \* 3. / nl  # 縮放到圖像大小和圖層

    model.nc = nc  # 將classes數量附加到模型

    model.hyp = hyp  # 將超參數附加到模型

    model.gr = 1.0  # iou損失率 (obj\_loss = 1.0 or iou)

    model.class\_weights = labels\_to\_class\_weights(dataset.labels, nc).to(device) \* nc  # 附加classes權重

    model.names = names

    # 開始訓練

    t0 = time.time()

    nw = max(round(hyp['warmup\_epochs'] \* nb), 1000)  # 預熱迭代次數，最大值(3 時程, 1k迭代)

    # nw = min(nw, (epochs - start\_epoch) / 2 \* nb)  #將預熱限制為小於訓練的1/2

    maps = np.zeros(nc)  # 每個class的mAP

    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  # 不要移動

    scaler = amp.GradScaler(enabled=cuda)

    compute\_loss = ComputeLoss(model)  # 初始損耗等級

    logger.info(f'Image sizes {imgsz} train, {imgsz\_test} test\n'

                f'Using {dataloader.num\_workers} dataloader workers\n'

                f'Logging results to {save\_dir}\n'

                f'Starting training for {epochs} epochs...')

    for epoch in range(start\_epoch, epochs):  # 時脈------------------------------------------------------------------

        model.train()

        # 更新圖像權重(optional)

        if opt.image\_weights:

            # 產生索引

            if rank in [-1, 0]:

                cw = model.class\_weights.cpu().numpy() \* (1 - maps) \*\* 2 / nc  # class權重

                iw = labels\_to\_image\_weights(dataset.labels, nc=nc, class\_weights=cw)  # 圖像權重

                dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)  # 排權重idx

            # 當DDP時廣播

            if rank != -1:

                indices = (torch.tensor(dataset.indices) if rank == 0 else torch.zeros(dataset.n)).int()

                dist.broadcast(indices, 0)

                if rank != 0:

                    dataset.indices = indices.cpu().numpy()

        # 更新馬賽克邊框

        # b = int(random.uniform(0.25 \* imgsz, 0.75 \* imgsz + gs) // gs \* gs)

        # dataset.mosaic\_border = [b - imgsz, -b]  # 高度,寬度邊框

        mloss = torch.zeros(4, device=device)  # 平均損失

        if rank != -1:

            dataloader.sampler.set\_epoch(epoch)

        pbar = enumerate(dataloader)

        logger.info(('\n' + '%10s' \* 8) % ('Epoch', 'gpu\_mem', 'box', 'obj', 'cls', 'total', 'targets', 'img\_size'))

        if rank in [-1, 0]:

            pbar = tqdm(pbar, total=nb)  # 進度條

        optimizer.zero\_grad()

        for i, (imgs, targets, paths, \_) in pbar:  # batch -------------------------------------------------------------

            ni = i + nb \* epoch  # 累計批次數(從訓練開始)

            imgs = imgs.to(device, non\_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.0

            # 預熱

            if ni <= nw:

                xi = [0, nw]  # x interp

                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou損失率(obj\_loss = 1.0 or iou)

                accumulate = max(1, np.interp(ni, xi, [1, nbs / total\_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']])

            # 多尺度

            if opt.multi\_scale:

                sz = random.randrange(imgsz \* 0.5, imgsz \* 1.5 + gs) // gs \* gs  # 尺寸

                sf = sz / max(imgs.shape[2:])  # 比例因子

                if sf != 1:

                    ns = [math.ceil(x \* sf / gs) \* gs for x in imgs.shape[2:]]  # 新形狀(拉伸至gs-multiple)

                    imgs = F.interpolate(imgs, size=ns, mode='bilinear', align\_corners=False)

            # Forward

            with amp.autocast(enabled=cuda):

                pred = model(imgs)  # forward

                loss, loss\_items = compute\_loss(pred, targets.to(device))  # 損失按批次大小縮放

                if rank != -1:

                    loss \*= opt.world\_size  # DDP模式下設備之間的平均梯度

                if opt.quad:

                    loss \*= 4.

            # Backward

            scaler.scale(loss).backward()

            # 優化

            if ni % accumulate == 0:

                scaler.step(optimizer)  # optimizer.step

                scaler.update()

                optimizer.zero\_grad()

                if ema:

                    ema.update(model)

            # 列印

            if rank in [-1, 0]:

                mloss = (mloss \* i + loss\_items) / (i + 1)  # 更新平均損失

                mem = '%.3gG' % (torch.cuda.memory\_reserved() / 1E9 if torch.cuda.is\_available() else 0)  # (GB)

                s = ('%10s' \* 2 + '%10.4g' \* 6) % (

                    '%g/%g' % (epoch, epochs - 1), mem, \*mloss, targets.shape[0], imgs.shape[-1])

                pbar.set\_description(s)

                # 繪製

                if plots and ni < 3:

                    f = save\_dir / f'train\_batch{ni}.jpg'  # 檔案名稱

                    Thread(target=plot\_images, args=(imgs, targets, paths, f), daemon=True).start()

                    # if tb\_writer:

                    #     tb\_writer.add\_image(f, result, dataformats='HWC', global\_step=epoch)

                    #     tb\_writer.add\_graph(model, imgs)  # 將模型添加到量板

                elif plots and ni == 10 and wandb:

                    wandb.log({"Mosaics": [wandb.Image(str(x), caption=x.name) for x in save\_dir.glob('train\*.jpg')

                                           if x.exists()]}, commit=False)

            # 批量結束 ------------------------------------------------------------------------------------------------

        # 時脈結束 ----------------------------------------------------------------------------------------------------

        # 排程器

        lr = [x['lr'] for x in optimizer.param\_groups]  # 給量版

        scheduler.step()

        # DDP process 0 or single-GPU

        if rank in [-1, 0]:

            # mAP

            ema.update\_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names', 'stride', 'class\_weights'])

            final\_epoch = epoch + 1 == epochs

            if not opt.notest or final\_epoch:  # 計算mAP

                results, maps, times = test.test(opt.data,

                                                 batch\_size=batch\_size \* 2,

                                                 imgsz=imgsz\_test,

                                                 model=ema.ema,

                                                 single\_cls=opt.single\_cls,

                                                 dataloader=testloader,

                                                 save\_dir=save\_dir,

                                                 verbose=nc < 50 and final\_epoch,

                                                 plots=plots and final\_epoch,

                                                 log\_imgs=opt.log\_imgs if wandb else 0,

                                                 compute\_loss=compute\_loss)

            # 寫入

            with open(results\_file, 'a') as f:

                f.write(s + '%10.4g' \* 7 % results + '\n')  # 附加指標, val\_loss

            if len(opt.name) and opt.bucket:

                os.system('gsutil cp %s gs://%s/results/results%s.txt' % (results\_file, opt.bucket, opt.name))

            # Log

            tags = ['train/box\_loss', 'train/obj\_loss', 'train/cls\_loss',  # 訓練損失

                    'metrics/precision', 'metrics/recall', 'metrics/mAP\_0.5', 'metrics/mAP\_0.5:0.95',

                    'val/box\_loss', 'val/obj\_loss', 'val/cls\_loss',  # val loss

                    'x/lr0', 'x/lr1', 'x/lr2']  # 參數

            for x, tag in zip(list(mloss[:-1]) + list(results) + lr, tags):

                if tb\_writer:

                    tb\_writer.add\_scalar(tag, x, epoch)  # 量板

                if wandb:

                    wandb.log({tag: x}, step=epoch, commit=tag == tags[-1])  # W&B

            # 更新最佳 mAP

            fi = fitness(np.array(results).reshape(1, -1))  # [P, R, mAP@.5, mAP@.5-.95] 的加權組合

            if fi > best\_fitness:

                best\_fitness = fi

            # 儲存模型

            if (not opt.nosave) or (final\_epoch and not opt.evolve):  # if save

                ckpt = {'epoch': epoch,

                        'best\_fitness': best\_fitness,

                        'training\_results': results\_file.read\_text(),

                        'model': ema.ema if final\_epoch else deepcopy(

                            model.module if is\_parallel(model) else model).half(),

                        'ema': (deepcopy(ema.ema).half(), ema.updates),

                        'optimizer': optimizer.state\_dict(),

                        'wandb\_id': wandb\_run.id if wandb else None}

                # 保存最後，最好的並刪除

                torch.save(ckpt, last)

                if best\_fitness == fi:

                    torch.save(ckpt, best)

                del ckpt

        # 時脈結束 ----------------------------------------------------------------------------------------------------

    # 訓練結束

    if rank in [-1, 0]:

        # 帶狀優化器

        final = best if best.exists() else last  # 最終模型

        for f in last, best:

            if f.exists():

                strip\_optimizer(f)

        if opt.bucket:

            os.system(f'gsutil cp {final} gs://{opt.bucket}/weights')  # 上傳

        # 繪製

        if plots:

            plot\_results(save\_dir=save\_dir)  # 儲存為results.png

            if wandb:

                files = ['results.png', 'confusion\_matrix.png', \*[f'{x}\_curve.png' for x in ('F1', 'PR', 'P', 'R')]]

                wandb.log({"Results": [wandb.Image(str(save\_dir / f), caption=f) for f in files

                                       if (save\_dir / f).exists()]})

                if opt.log\_artifacts:

                    wandb.log\_artifact(artifact\_or\_path=str(final), type='model', name=save\_dir.stem)

        # 測試best.pt

        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start\_epoch + 1, (time.time() - t0) / 3600))

        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO

            for m in (last, best) if best.exists() else (last):  # 速度, mAP 測試

                results, \_, \_ = test.test(opt.data,

                                          batch\_size=batch\_size \* 2,

                                          imgsz=imgsz\_test,

                                          conf\_thres=0.001,

                                          iou\_thres=0.7,

                                          model=attempt\_load(m, device).half(),

                                          single\_cls=opt.single\_cls,

                                          dataloader=testloader,

                                          save\_dir=save\_dir,

                                          save\_json=True,

                                          plots=False)

    else:

        dist.destroy\_process\_group()

    wandb.run.finish() if wandb and wandb.run else None

    torch.cuda.empty\_cache()

    return results

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('--weights', type=str, default='yolov5s.pt', help='initial weights path')

    parser.add\_argument('--cfg', type=str, default='', help='model.yaml path')

    parser.add\_argument('--data', type=str, default='data/coco128.yaml', help='data.yaml path')

    parser.add\_argument('--hyp', type=str, default='data/hyp.scratch.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')

    parser.add\_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train, test] image sizes')

    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('--notest', action='store\_true', help='only test final epoch')

    parser.add\_argument('--noautoanchor', action='store\_true', help='disable autoanchor check')

    parser.add\_argument('--evolve', action='store\_true', help='evolve hyperparameters')

    parser.add\_argument('--bucket', type=str, default='', help='gsutil bucket')

    parser.add\_argument('--cache-images', action='store\_true', help='cache images for faster training')

    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('--adam', action='store\_true', help='use torch.optim.Adam() optimizer')

    parser.add\_argument('--sync-bn', action='store\_true', help='use SyncBatchNorm, only available in DDP mode')

    parser.add\_argument('--local\_rank', type=int, default=-1, help='DDP parameter, do not modify')

    parser.add\_argument('--log-imgs', type=int, default=16, help='number of images for W&B logging, max 100')

    parser.add\_argument('--log-artifacts', action='store\_true', help='log artifacts, i.e. final trained model')

    parser.add\_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')

    parser.add\_argument('--project', default='runs/train', help='save to project/name')

    parser.add\_argument('--entity', default=None, help='W&B entity')

    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('--linear-lr', action='store\_true', help='linear LR')

    opt = parser.parse\_args()

    # 設置DDP變量

    opt.world\_size = int(os.environ['WORLD\_SIZE']) if 'WORLD\_SIZE' in os.environ else 1

    opt.global\_rank = int(os.environ['RANK']) if 'RANK' in os.environ else -1

    set\_logging(opt.global\_rank)

    if opt.global\_rank in [-1, 0]:

        check\_git\_status()

        check\_requirements()

    # 復原

    if opt.resume:  # 恢復中斷的運行

        ckpt = opt.resume if isinstance(opt.resume, str) else get\_latest\_run()  # 指定或最近的路徑

        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'

        apriori = opt.global\_rank, opt.local\_rank

        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:

            opt = argparse.Namespace(\*\*yaml.load(f, Loader=yaml.SafeLoader))  # 取代

        opt.cfg, opt.weights, opt.resume, opt.batch\_size, opt.global\_rank, opt.local\_rank = '', ckpt, True, opt.total\_batch\_size, \*apriori  # 恢復

        logger.info('Resuming training from %s' % ckpt)

    else:

        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else 'hyp.scratch.yaml')

        opt.data, opt.cfg, opt.hyp = check\_file(opt.data), check\_file(opt.cfg), check\_file(opt.hyp)  # 檢查檔案

        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be specified'

        opt.img\_size.extend([opt.img\_size[-1]] \* (2 - len(opt.img\_size)))  # 擴展到2種尺寸(訓練, 測試)

        opt.name = 'evolve' if opt.evolve else opt.name

        opt.save\_dir = increment\_path(Path(opt.project) / opt.name, exist\_ok=opt.exist\_ok | opt.evolve)  # 增量運行

    # DDP模式

    opt.total\_batch\_size = opt.batch\_size

    device = select\_device(opt.device, batch\_size=opt.batch\_size)

    if opt.local\_rank != -1:

        assert torch.cuda.device\_count() > opt.local\_rank

        torch.cuda.set\_device(opt.local\_rank)

        device = torch.device('cuda', opt.local\_rank)

        dist.init\_process\_group(backend='nccl', init\_method='env://')  # 分佈式後端

        assert opt.batch\_size % opt.world\_size == 0, '--batch-size must be multiple of CUDA device count'

        opt.batch\_size = opt.total\_batch\_size // opt.world\_size

    # 參數

    with open(opt.hyp) as f:

        hyp = yaml.load(f, Loader=yaml.SafeLoader)  # 讀取參數

    # 訓練

    logger.info(opt)

    try:

        import wandb

    except ImportError:

        wandb = None

        prefix = colorstr('wandb: ')

        logger.info(f"{prefix}Install Weights & Biases for YOLOv5 logging with 'pip install wandb' (recommended)")

    if not opt.evolve:

        tb\_writer = None  # 初始化紀錄器

        if opt.global\_rank in [-1, 0]:

            logger.info(f'Start Tensorboard with "tensorboard --logdir {opt.project}", view at http://localhost:6006/')

            tb\_writer = SummaryWriter(opt.save\_dir)  # 量板

        train(hyp, opt, device, tb\_writer, wandb)

    # 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)

        assert opt.local\_rank == -1, 'DDP mode not implemented for --evolve'

        opt.notest, opt.nosave = True, True  # 僅測試/保存最終時脈

        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # 演化指數

        yaml\_file = Path(opt.save\_dir) / 'hyp\_evolved.yaml'  # 在這裡保存最佳結果

        if opt.bucket:

            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # 如果存在，下載evolve.txt

        for \_ in range(300):  # generations to evolve

            if Path('evolve.txt').exists():  # 若evolve.txt存在: 選擇最佳的超參數和變異值

                # 選擇父級

                parent = 'single'  # 父級選擇方法：”單精度”或”加權”

                x = np.loadtxt('evolve.txt', ndmin=2)

                n = min(5, len(x))  # 要考慮的先前結果數

                x = x[np.argsort(-fitness(x))][:n]  # 前n個突變

                w = fitness(x) - fitness(x).min()  # 權重

                if parent == 'single' or len(x) == 1:

                    # x = x[random.randint(0, n - 1)]  # 隨機選擇

                    x = x[random.choices(range(n), weights=w)[0]]  # 權重選擇

                elif parent == 'weighted':

                    x = (x \* w.reshape(n, 1)).sum(0) / w.sum()  # 權重組合

                # 變異

                mp, s = 0.8, 0.2  # 變異率, sigma

                npr = np.random

                npr.seed(int(time.time()))

                g = np.array([x[0] for x in meta.values()])  # gains 0-1

                ng = len(meta)

                v = np.ones(ng)

                while all(v == 1):  # 變異直到發生變更(防止重複)

                    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])  # 變異

            # 限制極限

            for k, v in meta.items():

                hyp[k] = max(hyp[k], v[1])  # 下極限

                hyp[k] = min(hyp[k], v[2])  # 上極限

                hyp[k] = round(hyp[k], 5)  # 有效數字

            # 訓練變異

            results = train(hyp.copy(), opt, device, wandb=wandb)

            # 寫入變異結果

            print\_mutation(hyp.copy(), results, yaml\_file, opt.bucket)

        # 繪製結果

        plot\_evolution(yaml\_file)

        print(f'Hyperparameter evolution complete. Best results saved as: {yaml\_file}\n'

              f'Command to train a new model with these hyperparameters: $ python train.py --hyp {yaml\_file}')