Lora模块程序实现代码：

#define R0  510.f//采样电阻

#define VC  1240.f//采样信号幅值

#define K   10.43 //电导率常数

u32 ddl,ddl\_25;//电导率 us.cm-1

float yandu;

//将电导电极经过ad转换得到的数据，经过公式计算和温度补偿得到对应的电导率和盐度

void vddl\_get(float V0)

{

   double G=0; //所测电导

   G=V0/((VC-V0)\*R0);

   ddl=K\*1000000.f\*G;

   ddl\_25=(float)ddl/(1+0.02\*(DS18B20\_DATA.T.Temperature-25.f));

   yandu=(1.38888\*ddl-0.02478\*ddl\*DS18B20\_DATA.T.Temperature-6171.9)/1000.f;

   if(ddl<=0)yandu=0;

   if(yandu<=0)yandu=0

float test;

float ph,k\_ll/\*理论斜率\*/;

void vddl\_get\_ph(float V0)

{

   test=V0;

   k\_ll=-0.0001984250996144478\*(273.15+DS18B20\_DATA.T.Temperature);

   ph=V0/1000.0/k\_ll\*(1.0)+7.0;

   if(ph<=0)ph=0;

}

u32 TIM\_T =100000;

u32 value1,value2;

//通过对定时器1的配置来计算风速

struct a xduty\_fre;

void TIM1\_CC\_IRQHandler(void)

{

    if(TIM\_GetITStatus(TIM1,TIM\_IT\_CC1)!=RESET)

    {

        TIM\_ClearITPendingBit(TIM1,TIM\_IT\_CC1);

        value1=TIM1->CCR1;

        value2=TIM1->CCR2;

        if(value1!=0)

        {

            xduty\_fre.freq.freq\_f=TIM\_T/(float)(value1+1);

            xduty\_fre.duty=(float)(value2+1)\*100/(value1+1);

            xduty\_fre.fengsu=xduty\_fre.freq.freq\_f\*0.0875;

            xduty\_fre.fengsu2=xduty\_fre.fengsu;

            xduty\_fre.flag=1;

        }

        else

        {

            xduty\_fre.freq.freq\_f=0;

            xduty\_fre.duty=0;

        }

    }

}

//每隔一段时间检测数据是否接收成功，成功后重新使能定时器继续接收，如果没有风则返回0，然后发送数据。

        if((GlobalTime-test\_time)>=280)

       {

            time++;

          if(xduty\_fre.flag)

          {

             xduty\_fre.flag=0;

             time=0;

             TIM\_Cmd(TIM1,ENABLE);

          }

          else

          {

               if(time>=5)

               {

                     time=0;

                     xduty\_fre.fengsu2=0;

               }

          }

          vlora\_tx();

          test\_time=GlobalTime;

       }

/\*\*

连续测量（1mm）   ADDR 06 83” 3X 3X 3X 2E 3X 3X 3X”CS 正确返回

                  ADDR 06 83” ’E’ ’R’ ’R’ ’-’ ’-’ ’3X’ ’3X’”CS 错误返回

\*\*/

//通过串口接收并筛选出激光测距传感器回传的距离，然后合并后进行存储，等待发送

void USART3\_IRQHandler(void)

{

   u8 res,i,sum;

   static u8 state=0;

   if(USART\_GetITStatus(USART3, USART\_IT\_RXNE) != RESET)//接收到数据

   {

      res =USART\_ReceiveData(USART3);

      switch(state)

      {

         case 0:

            if(res==0x80)

            {

               state=1;

               USART3\_RX\_BUF[rx\_ptr++]=res;

            }

            else

            {

               state=0,rx\_ptr=0 ;

            }

         break;

         case 1:

            if(res==0x06)

            {

               state=2;

               USART3\_RX\_BUF[rx\_ptr++]=res;

            }

            else

            {

               state=0,rx\_ptr=0 ;

            }

         break;

         case 2:

            if(res==0x83)

            {

               state=3;

               USART3\_RX\_BUF[rx\_ptr++]=res;

            }

            else

            {

               state=0,rx\_ptr=0 ;

            }

         break;

         case 3:

            if(rx\_ptr<11)

            {

               USART3\_RX\_BUF[rx\_ptr++]=res;

            }

            else

            {

               USART3\_RX\_BUF[rx\_ptr]=res;

               for(sum=0,i=0;i<10;i++)

               {

                  sum+=USART3\_RX\_BUF[i];

               }

               sum=(~sum)+1;

               if(sum==USART3\_RX\_BUF[10])//cs

               {

                    if(strstr((const char\*)USART3\_RX\_BUF,"ERR")==NULL)

                    {

                       disatance\_f=(float)(USART3\_RX\_BUF[3]-0x30)\*100.0+(float)(USART3\_RX\_BUF[4]-0x30)\*10.0+(float)(USART3\_RX\_BUF[5]-0x30)+ (float)(USART3\_RX\_BUF[7]-0x30)\*0.1+(float)(USART3\_RX\_BUF[8]-0x30)\*0.01+(float)(USART3\_RX\_BUF[9]-0x30)\*0.001;

                    }

               }

               state=0,rx\_ptr=0 ;

            }

         break;

         default:

            state=0,rx\_ptr=0 ;

         break;

      }

   }

}

1.加速度输出：

0x55 0x51 AxL AxH AyL AyH AzL AzH TL TH SUM

计算方法：

ax =((AxH<<8)|AxL)/32768\*16g(g 为重力加速度，可取 9.8m/s 2 )

ay =((AyH<<8)|AyL)/32768\*16g(g 为重力加速度，可取 9.8m/s 2 )

az =((AzH<<8)|AzL)/32768\*16g(g 为重力加速度，可取 9.8m/s 2 )

温度计算公式：

T=((TH<<8)|TL) /100 ℃

校验和：

Sum=0x55+0x51+AxH+AxL+AyH+AyL+AzH+AzL+TH+TL

2. 角速度输出：

0x55 0x52 wxL wxH wyL wyH wzL wzH TL TH SUM

计算方法：

wx =((wxH<<8)|wxL)/32768\*2000(°/s)

wy =((wyH<<8)|wyL)/32768\*2000(°/s)

wz =((wzH<<8)|wzL)/32768\*2000(°/s)

温度计算公式：

T=((TH<<8)|TL) /100 ℃

校验和：

Sum=0x55+0x52+wxH+wxL+wyH+wyL+wzH+wzL+TH+TL

3. 角度输出：

0x55 0x53 RollL RollH PitchL PitchH YawL YawH TL TH SUM

计算方法：

滚转角（x 轴）Roll=((RollH<<8)|RollL)/32768\*180(°)

俯仰角（y 轴）Pitch=((PitchH<<8)|PitchL)/32768\*180(°)

偏航角（z 轴）Yaw=((YawH<<8)|YawL)/32768\*180(°)

温度计算公式：

T=((TH<<8)|TL) /100 ℃

校验和：

Sum=0x55+0x53+RollH+RollL+PitchH+PitchL+YawH+YawL+TH+TL

4. 磁场输出：

0x55 0x54 HxL HxH HyL HyH HzL HzH TL TH SUM

计算方法：

磁场（x 轴） Hx =(( HxH <<8)| HxL )

磁场（y 轴） Hy =(( HyH <<8)| HyL )

磁场（z 轴） Hz =(( HzH <<8)| HzL )

温度计算公式：

T=((TH<<8)|TL) /100 ℃

校验和：

Sum=0x55+0x54+HxH+HxL+HyH+HyL+HzH+HzL+TH+TL

1. 四元素输出：

0x55 0x59 Q0L Q0H Q1L Q1H Q2L Q2H Q3L Q3H SUM

计算方法：

Q0=((Q0H<<8)|Q0L)/32768

Q1=((Q1H<<8)|Q1L)/32768

Q2=((Q2H<<8)|Q2L)/32768

Q3=((Q3H<<8)|Q3L)/32768

校验和：

Sum=0x55+0x59+Q0L+Q0H+Q1L +Q1H +Q2L+Q2H+Q3L+Q3H

说明：

1、 数据是按照 16 进制方式发送的，不是 ASCII 码。

2、 每个数据分低字节和高字节依次传送，二者组合成一个有符号的short类型的数据。假设 Data 为实际的数据，DataH 为其高字节部分，DataL 为其低字节部分， 那么：

Data=(short)（DataH<<8|DataL）。这里一定要注意 DataH 需要先强制转换为一个有符号的 short 类型的数据以后再移位，并且 Data 的数据类型也是有符号的 short 类型，这样才能表示出负数。

1. 欧拉角输出：

void QuaternionToEuler(xGD\* ex)

{

    const double Epsilon = 0.0009765625f;

    const double Threshold = 0.5f - Epsilon;

    double TEST=0;

    struct

    {

        float w;

        float x;

        float y;

        float z;

    }q;

    q.w=ex->q0;

    q.x=ex->q1;

    q.y=ex->q2;

    q.z=ex->q3;

    TEST= q.w\*q.y - q.x\*q.z;

    if (TEST < -Threshold || TEST > Threshold) // 奇异姿态,俯仰角为±90°

    {

      int sign = Sign(TEST);

      xgd.euler\_z = -2 \* sign \* (double)atan2(q.x, q.w)\*180/PI; // yaw

      xgd.euler\_y  = sign \* 90; // pitch

      xgd.euler\_x = 0; // roll

    }

    else

    {

        ex->euler\_x = atan2(2 \* (q.y\*q.z + q.w\*q.x), q.w\*q.w - q.x\*q.x - q.y\*q.y + q.z\*q.z)\*180/PI;

        ex->euler\_y = asin(-2 \* (q.x\*q.z - q.w\*q.y))\*180/PI;

        ex->euler\_z = atan2(2 \* (q.x\*q.y + q.w\*q.z), q.w\*q.w + q.x\*q.x - q.y\*q.y - q.z\*q.z)\*180/PI;

    }

}

//分析GNGGA信息

//gpsx:nmea信息结构体

//buf:接收到的GPS数据缓冲区首地址

void NMEA\_GNGGA\_Analysis(nmea\_msg \*gpsx,u8 \*buf)

{

    u8 \*p1,dx;

    u8 posx;

    p1=(u8\*)strstr((const char \*)buf,"$GNGGA");

    posx=NMEA\_Comma\_Pos(p1,6);                          //得到GPS状态

    if(posx!=0XFF)gpsx->gpssta=NMEA\_Str2num(p1+posx,&dx);

    posx=NMEA\_Comma\_Pos(p1,7);                          //得到用于定位的卫星数

    if(posx!=0XFF)gpsx->posslnum=NMEA\_Str2num(p1+posx,&dx);

    posx=NMEA\_Comma\_Pos(p1,9);                          //得到海拔高度

    if(posx!=0XFF)gpsx->altitude=NMEA\_Str2num(p1+posx,&dx);

}

数据采集云端代码

#include <DFRobot\_PH.h>

#include <DFRobot\_TDS.h>

#include <Microbit\_Matrix.h>

#include <DFRobot\_Libraries.h>

#include <DFRobot\_WiFi\_IoT\_Module.h>

// 动态变量

volatile float mind\_n\_a;

// 函数声明

void obloqMqttEventT2(const char \*obloq\_message);

// 静态常量

const uint8\_t bbcBitmap[][5] = {

{B00110,B00110,B00110,B00000,B00110}

};

const String topics[5] = {"CL08StOSR","GJdUItdSR","","",""};

const MqttHandle msgHandles[5] = {NULL,NULL,obloqMqttEventT2,NULL,NULL};

// 创建对象

DFRobot\_WiFi\_IoT\_Module\_UART olq;

DFRobot\_TDS tds\_1(1);

DFRobot\_PH ph;

DFRobot\_Servo360 servo360\_8;

// 主程序开始

void setup() {

olq.registerMqttHandle(msgHandles);

servo360\_8.attach(8);

MMatrix.show(bbcBitmap[0]);

olq.startConnect(13, 14, "111", "22222222", "yZPT3ZOIR", "yZEoqZdIgz", topics, "iot.dfrobot.com.cn", 1883);

}

void loop() {

olq.publish(olq.topic\_0, (tds\_1.getTdsValue()));

delay(10000);

olq.publish(olq.topic\_1, ph.readPH(2));

delay(10000);

}

// 事件回调函数

void obloqMqttEventT2(const char \*obloq\_message) {

if (((String(obloq\_message).toInt())>=0)) {

MMatrix.print((String(obloq\_message).toInt()));

mind\_n\_a = (String(obloq\_message).toInt());

servo360\_8.speed(100);

delay(mind\_n\_a \* 1000);

servo360\_8.speed(0);

}

}

Detect.py

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

"""

Run inference on images, videos, directories, streams, etc.

Usage - sources:

$ python path/to/detect.py --weights yolov5s.pt --source 0 # webcam

img.jpg # image

vid.mp4 # video

path/ # directory

path/\*.jpg # glob

'https://youtu.be/Zgi9g1ksQHc' # YouTube

'rtsp://example.com/media.mp4' # RTSP, RTMP, HTTP stream

Usage - formats:

$ python path/to/detect.py --weights yolov5s.pt # PyTorch

yolov5s.torchscript # TorchScript

yolov5s.onnx # ONNX Runtime or OpenCV DNN with --dnn

yolov5s.xml # OpenVINO

yolov5s.engine # TensorRT

yolov5s.mlmodel #CoreML (macOS-only)

yolov5s\_saved\_model # TensorFlow SavedModel

yolov5s.pb # TensorFlow GraphDef

yolov5s.tflite # TensorFlow Lite

yolov5s\_edgetpu.tflite # TensorFlow Edge TPU

"""

import argparse

import os

import platform

import sys

import time

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, smart\_inference\_mode, time\_sync

@smart\_inference\_mode()

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

t0 = time.time()

count = 0

tplay = 0

model.warmup(imgsz=(1 if pt else bs, 3, \*imgsz)) # warmup

seen, windows, dt = 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 names[int(cls)] == 'dog\_wind\_test':

value = det[:, 4].max().item() # 进行多个目标检测，检测的所需目标才发出语音告警

if value > 0.70:

count += 1

if count > 8:

count = 0

if time.time() - tplay > 4.0:

import os

os.system(

'start /b C:/conda-yolov-mp3/yolov5-6.2-horse/yolov5-v5s-mp3/ffmpeg/bin/ffplay.exe -autoexit -nodisp D:/conda-yolov-mp3/yolov5-6.2-horse/floating.mp3') # 音乐播放

# 参数含义： start /b 后台启动 ffplay音乐播放软件的位置 -autoexit 播放完毕自动退出 -nodisp不显示窗口 mp3语音的位置

tplay = time.time()

if names[int(cls)] == 'horse':

value = det[:, 4].max().item() # 进行多个目标检测，检测的所需目标才发出语音告警

if value > 0.70:

count += 1

if count > 8:

count = 0

if time.time() - tplay > 4.0:

import os

os.system(

'start /b C:/conda-yolov-mp3/yolov5-6.2-horse/yolov5-v5s-mp3/ffmpeg/bin/ffplay.exe -autoexit -nodisp D:/conda-yolov-mp3/yolov5-6.2-horse/floating.mp3') # 音乐播放

# 参数含义： start /b 后台启动 ffplay音乐播放软件的位置 -autoexit 播放完毕自动退出 -nodisp不显示窗口 mp3语音的位置

tplay = time.time()

if names[int(cls)] == 'cattle':

value = det[:, 4].max().item() # 进行多个目标检测，检测的所需目标才发出语音告警

if value > 0.80:

count += 1

if count > 8:

count = 0

if time.time() - tplay > 4.0:

import os

os.system(

'start /b C:/conda-yolov-mp3/yolov5-6.2-horse/yolov5-v5s-mp3/ffmpeg/bin/ffplay.exe -autoexit -nodisp D:/conda-yolov-mp3/yolov5-6.2-horse/floating.mp3') # 音乐播放

# 参数含义： start /b 后台启动 ffplay音乐播放软件的位置 -autoexit 播放完毕自动退出 -nodisp不显示窗口 mp3语音的位置

tplay = time.time()

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:

if platform.system() == 'Linux' and p not in windows:

windows.append(p)

cv2.namedWindow(str(p), cv2.WINDOW\_NORMAL | cv2.WINDOW\_KEEPRATIO) # allow window resize (Linux)

cv2.resizeWindow(str(p), im0.shape[1], im0.shape[0])

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[0]) # update model (to fix SourceChangeWarning)

def parse\_opt():

parser = argparse.ArgumentParser()

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

parser.add\_argument('--source', type=str, default=0, 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)

export.py

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

"""

Export a YOLOv5 PyTorch model to other formats. TensorFlow exports authored by https://github.com/zldrobit

Format | `export.py --include` | Model

--- | --- | ---

PyTorch | - | yolov5s.pt

TorchScript | `torchscript` | yolov5s.torchscript

ONNX | `onnx` | yolov5s.onnx

OpenVINO | `openvino` | yolov5s\_openvino\_model/

TensorRT | `engine` | yolov5s.engine

CoreML | `coreml` | yolov5s.mlmodel

TensorFlow SavedModel | `saved\_model` | yolov5s\_saved\_model/

TensorFlow GraphDef | `pb` | yolov5s.pb

TensorFlow Lite | `tflite` | yolov5s.tflite

TensorFlow Edge TPU | `edgetpu` | yolov5s\_edgetpu.tflite

TensorFlow.js | `tfjs` | yolov5s\_web\_model/

Requirements:

$ pip install -r requirements.txt coremltools onnx onnx-simplifier onnxruntime openvino-dev tensorflow-cpu # CPU

$ pip install -r requirements.txt coremltools onnx onnx-simplifier onnxruntime-gpu openvino-dev tensorflow # GPU

Usage:

$ python path/to/export.py --weights yolov5s.pt --include torchscript onnx openvino engine coreml tflite ...

Inference:

$ python path/to/detect.py --weights yolov5s.pt # PyTorch

yolov5s.torchscript # TorchScript

yolov5s.onnx # ONNX Runtime or OpenCV DNN with --dnn

yolov5s.xml # OpenVINO

yolov5s.engine # TensorRT

yolov5s.mlmodel # CoreML (macOS-only)

yolov5s\_saved\_model # TensorFlow SavedModel

yolov5s.pb # TensorFlow GraphDef

yolov5s.tflite # TensorFlow Lite

yolov5s\_edgetpu.tflite # TensorFlow Edge TPU

TensorFlow.js:

$ cd .. && git clone https://github.com/zldrobit/tfjs-yolov5-example.git && cd tfjs-yolov5-example

$ npm install

$ ln -s ../../yolov5/yolov5s\_web\_model public/yolov5s\_web\_model

$ npm start

"""

import argparse

import json

import os

import platform

import subprocess

import sys

import time

import warnings

from pathlib import Path

import pandas as pd

import torch

import yaml

from torch.utils.mobile\_optimizer import optimize\_for\_mobile

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

if platform.system() != 'Windows':

ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.experimental import attempt\_load

from models.yolo import Detect

from utils.dataloaders import LoadImages

from utils.general import (LOGGER, check\_dataset, check\_img\_size, check\_requirements, check\_version, check\_yaml,

colorstr, file\_size, print\_args, url2file)

from utils.torch\_utils import select\_device, smart\_inference\_mode

def export\_formats():

# YOLOv5 export formats

x = [

['PyTorch', '-', '.pt', True, True],

['TorchScript', 'torchscript', '.torchscript', True, True],

['ONNX', 'onnx', '.onnx', True, True],

['OpenVINO', 'openvino', '\_openvino\_model', True, False],

['TensorRT', 'engine', '.engine', False, True],

['CoreML', 'coreml', '.mlmodel', True, False],

['TensorFlow SavedModel', 'saved\_model', '\_saved\_model', True, True],

['TensorFlow GraphDef', 'pb', '.pb', True, True],

['TensorFlow Lite', 'tflite', '.tflite', True, False],

['TensorFlow Edge TPU', 'edgetpu', '\_edgetpu.tflite', False, False],

['TensorFlow.js', 'tfjs', '\_web\_model', False, False],]

return pd.DataFrame(x, columns=['Format', 'Argument', 'Suffix', 'CPU', 'GPU'])

def export\_torchscript(model, im, file, optimize, prefix=colorstr('TorchScript:')):

# YOLOv5 TorchScript model export

try:

LOGGER.info(f'\n{prefix} starting export with torch {torch.\_\_version\_\_}...')

f = file.with\_suffix('.torchscript')

ts = torch.jit.trace(model, im, strict=False)

d = {"shape": im.shape, "stride": int(max(model.stride)), "names": model.names}

extra\_files = {'config.txt': json.dumps(d)} # torch.\_C.ExtraFilesMap()

if optimize: # https://pytorch.org/tutorials/recipes/mobile\_interpreter.html

optimize\_for\_mobile(ts).\_save\_for\_lite\_interpreter(str(f), \_extra\_files=extra\_files)

else:

ts.save(str(f), \_extra\_files=extra\_files)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'{prefix} export failure: {e}')

def export\_onnx(model, im, file, opset, train, dynamic, simplify, prefix=colorstr('ONNX:')):

# YOLOv5 ONNX export

try:

check\_requirements(('onnx',))

import onnx

LOGGER.info(f'\n{prefix} starting export with onnx {onnx.\_\_version\_\_}...')

f = file.with\_suffix('.onnx')

torch.onnx.export(

model.cpu() if dynamic else model, # --dynamic only compatible with cpu

im.cpu() if dynamic else im,

f,

verbose=False,

opset\_version=opset,

training=torch.onnx.TrainingMode.TRAINING if train else torch.onnx.TrainingMode.EVAL,

do\_constant\_folding=not train,

input\_names=['images'],

output\_names=['output'],

dynamic\_axes={

'images': {

0: 'batch',

2: 'height',

3: 'width'}, # shape(1,3,640,640)

'output': {

0: 'batch',

1: 'anchors'} # shape(1,25200,85)

} if dynamic else None)

# Checks

model\_onnx = onnx.load(f) # load onnx model

onnx.checker.check\_model(model\_onnx) # check onnx model

# Metadata

d = {'stride': int(max(model.stride)), 'names': model.names}

for k, v in d.items():

meta = model\_onnx.metadata\_props.add()

meta.key, meta.value = k, str(v)

onnx.save(model\_onnx, f)

# Simplify

if simplify:

try:

cuda = torch.cuda.is\_available()

check\_requirements(('onnxruntime-gpu' if cuda else 'onnxruntime', 'onnx-simplifier>=0.4.1'))

import onnxsim

LOGGER.info(f'{prefix} simplifying with onnx-simplifier {onnxsim.\_\_version\_\_}...')

model\_onnx, check = onnxsim.simplify(model\_onnx)

assert check, 'assert check failed'

onnx.save(model\_onnx, f)

except Exception as e:

LOGGER.info(f'{prefix} simplifier failure: {e}')

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'{prefix} export failure: {e}')

def export\_openvino(model, file, half, prefix=colorstr('OpenVINO:')):

# YOLOv5 OpenVINO export

try:

check\_requirements(('openvino-dev',)) # requires openvino-dev: https://pypi.org/project/openvino-dev/

import openvino.inference\_engine as ie

LOGGER.info(f'\n{prefix} starting export with openvino {ie.\_\_version\_\_}...')

f = str(file).replace('.pt', f'\_openvino\_model{os.sep}')

cmd = f"mo --input\_model {file.with\_suffix('.onnx')} --output\_dir {f} --data\_type {'FP16' if half else 'FP32'}"

subprocess.check\_output(cmd.split()) # export

with open(Path(f) / file.with\_suffix('.yaml').name, 'w') as g:

yaml.dump({'stride': int(max(model.stride)), 'names': model.names}, g) # add metadata.yaml

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

def export\_coreml(model, im, file, int8, half, prefix=colorstr('CoreML:')):

# YOLOv5 CoreML export

try:

check\_requirements(('coremltools',))

import coremltools as ct

LOGGER.info(f'\n{prefix} starting export with coremltools {ct.\_\_version\_\_}...')

f = file.with\_suffix('.mlmodel')

ts = torch.jit.trace(model, im, strict=False) # TorchScript model

ct\_model = ct.convert(ts, inputs=[ct.ImageType('image', shape=im.shape, scale=1 / 255, bias=[0, 0, 0])])

bits, mode = (8, 'kmeans\_lut') if int8 else (16, 'linear') if half else (32, None)

if bits < 32:

if platform.system() == 'Darwin': # quantization only supported on macOS

with warnings.catch\_warnings():

warnings.filterwarnings("ignore", category=DeprecationWarning) # suppress numpy==1.20 float warning

ct\_model = ct.models.neural\_network.quantization\_utils.quantize\_weights(ct\_model, bits, mode)

else:

print(f'{prefix} quantization only supported on macOS, skipping...')

ct\_model.save(f)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return ct\_model, f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

return None, None

def export\_engine(model, im, file, train, half, dynamic, simplify, workspace=4, verbose=False):

# YOLOv5 TensorRT export https://developer.nvidia.com/tensorrt

prefix = colorstr('TensorRT:')

try:

assert im.device.type != 'cpu', 'export running on CPU but must be on GPU, i.e. `python export.py --device 0`'

try:

import tensorrt as trt

except Exception:

if platform.system() == 'Linux':

check\_requirements(('nvidia-tensorrt',), cmds=('-U --index-url https://pypi.ngc.nvidia.com',))

import tensorrt as trt

if trt.\_\_version\_\_[0] == '7': # TensorRT 7 handling https://github.com/ultralytics/yolov5/issues/6012

grid = model.model[-1].anchor\_grid

model.model[-1].anchor\_grid = [a[..., :1, :1, :] for a in grid]

export\_onnx(model, im, file, 12, train, dynamic, simplify) # opset 12

model.model[-1].anchor\_grid = grid

else: # TensorRT >= 8

check\_version(trt.\_\_version\_\_, '8.0.0', hard=True) # require tensorrt>=8.0.0

export\_onnx(model, im, file, 13, train, dynamic, simplify) # opset 13

onnx = file.with\_suffix('.onnx')

LOGGER.info(f'\n{prefix} starting export with TensorRT {trt.\_\_version\_\_}...')

assert onnx.exists(), f'failed to export ONNX file: {onnx}'

f = file.with\_suffix('.engine') # TensorRT engine file

logger = trt.Logger(trt.Logger.INFO)

if verbose:

logger.min\_severity = trt.Logger.Severity.VERBOSE

builder = trt.Builder(logger)

config = builder.create\_builder\_config()

config.max\_workspace\_size = workspace \* 1 << 30

# config.set\_memory\_pool\_limit(trt.MemoryPoolType.WORKSPACE, workspace << 30) # fix TRT 8.4 deprecation notice

flag = (1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT\_BATCH))

network = builder.create\_network(flag)

parser = trt.OnnxParser(network, logger)

if not parser.parse\_from\_file(str(onnx)):

raise RuntimeError(f'failed to load ONNX file: {onnx}')

inputs = [network.get\_input(i) for i in range(network.num\_inputs)]

outputs = [network.get\_output(i) for i in range(network.num\_outputs)]

LOGGER.info(f'{prefix} Network Description:')

for inp in inputs:

LOGGER.info(f'{prefix}\tinput "{inp.name}" with shape {inp.shape} and dtype {inp.dtype}')

for out in outputs:

LOGGER.info(f'{prefix}\toutput "{out.name}" with shape {out.shape} and dtype {out.dtype}')

if dynamic:

if im.shape[0] <= 1:

LOGGER.warning(f"{prefix}WARNING: --dynamic model requires maximum --batch-size argument")

profile = builder.create\_optimization\_profile()

for inp in inputs:

profile.set\_shape(inp.name, (1, \*im.shape[1:]), (max(1, im.shape[0] // 2), \*im.shape[1:]), im.shape)

config.add\_optimization\_profile(profile)

LOGGER.info(f'{prefix} building FP{16 if builder.platform\_has\_fast\_fp16 and half else 32} engine in {f}')

if builder.platform\_has\_fast\_fp16 and half:

config.set\_flag(trt.BuilderFlag.FP16)

with builder.build\_engine(network, config) as engine, open(f, 'wb') as t:

t.write(engine.serialize())

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

def export\_saved\_model(model,

im,

file,

dynamic,

tf\_nms=False,

agnostic\_nms=False,

topk\_per\_class=100,

topk\_all=100,

iou\_thres=0.45,

conf\_thres=0.25,

keras=False,

prefix=colorstr('TensorFlow SavedModel:')):

# YOLOv5 TensorFlow SavedModel export

try:

import tensorflow as tf

from tensorflow.python.framework.convert\_to\_constants import convert\_variables\_to\_constants\_v2

from models.tf import TFDetect, TFModel

LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.\_\_version\_\_}...')

f = str(file).replace('.pt', '\_saved\_model')

batch\_size, ch, \*imgsz = list(im.shape) # BCHW

tf\_model = TFModel(cfg=model.yaml, model=model, nc=model.nc, imgsz=imgsz)

im = tf.zeros((batch\_size, \*imgsz, ch)) # BHWC order for TensorFlow

\_ = tf\_model.predict(im, tf\_nms, agnostic\_nms, topk\_per\_class, topk\_all, iou\_thres, conf\_thres)

inputs = tf.keras.Input(shape=(\*imgsz, ch), batch\_size=None if dynamic else batch\_size)

outputs = tf\_model.predict(inputs, tf\_nms, agnostic\_nms, topk\_per\_class, topk\_all, iou\_thres, conf\_thres)

keras\_model = tf.keras.Model(inputs=inputs, outputs=outputs)

keras\_model.trainable = False

keras\_model.summary()

if keras:

keras\_model.save(f, save\_format='tf')

else:

spec = tf.TensorSpec(keras\_model.inputs[0].shape, keras\_model.inputs[0].dtype)

m = tf.function(lambda x: keras\_model(x)) # full model

m = m.get\_concrete\_function(spec)

frozen\_func = convert\_variables\_to\_constants\_v2(m)

tfm = tf.Module()

tfm.\_\_call\_\_ = tf.function(lambda x: frozen\_func(x)[:4] if tf\_nms else frozen\_func(x)[0], [spec])

tfm.\_\_call\_\_(im)

tf.saved\_model.save(tfm,

f,

options=tf.saved\_model.SaveOptions(experimental\_custom\_gradients=False)

if check\_version(tf.\_\_version\_\_, '2.6') else tf.saved\_model.SaveOptions())

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return keras\_model, f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

return None, None

def export\_pb(keras\_model, file, prefix=colorstr('TensorFlow GraphDef:')):

# YOLOv5 TensorFlow GraphDef \*.pb export https://github.com/leimao/Frozen\_Graph\_TensorFlow

try:

import tensorflow as tf

from tensorflow.python.framework.convert\_to\_constants import convert\_variables\_to\_constants\_v2

LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.\_\_version\_\_}...')

f = file.with\_suffix('.pb')

m = tf.function(lambda x: keras\_model(x)) # full model

m = m.get\_concrete\_function(tf.TensorSpec(keras\_model.inputs[0].shape, keras\_model.inputs[0].dtype))

frozen\_func = convert\_variables\_to\_constants\_v2(m)

frozen\_func.graph.as\_graph\_def()

tf.io.write\_graph(graph\_or\_graph\_def=frozen\_func.graph, logdir=str(f.parent), name=f.name, as\_text=False)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

def export\_tflite(keras\_model, im, file, int8, data, nms, agnostic\_nms, prefix=colorstr('TensorFlow Lite:')):

# YOLOv5 TensorFlow Lite export

try:

import tensorflow as tf

LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.\_\_version\_\_}...')

batch\_size, ch, \*imgsz = list(im.shape) # BCHW

f = str(file).replace('.pt', '-fp16.tflite')

converter = tf.lite.TFLiteConverter.from\_keras\_model(keras\_model)

converter.target\_spec.supported\_ops = [tf.lite.OpsSet.TFLITE\_BUILTINS]

converter.target\_spec.supported\_types = [tf.float16]

converter.optimizations = [tf.lite.Optimize.DEFAULT]

if int8:

from models.tf import representative\_dataset\_gen

dataset = LoadImages(check\_dataset(check\_yaml(data))['train'], img\_size=imgsz, auto=False)

converter.representative\_dataset = lambda: representative\_dataset\_gen(dataset, ncalib=100)

converter.target\_spec.supported\_ops = [tf.lite.OpsSet.TFLITE\_BUILTINS\_INT8]

converter.target\_spec.supported\_types = []

converter.inference\_input\_type = tf.uint8 # or tf.int8

converter.inference\_output\_type = tf.uint8 # or tf.int8

converter.experimental\_new\_quantizer = True

f = str(file).replace('.pt', '-int8.tflite')

if nms or agnostic\_nms:

converter.target\_spec.supported\_ops.append(tf.lite.OpsSet.SELECT\_TF\_OPS)

tflite\_model = converter.convert()

open(f, "wb").write(tflite\_model)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

def export\_edgetpu(file, prefix=colorstr('Edge TPU:')):

# YOLOv5 Edge TPU export https://coral.ai/docs/edgetpu/models-intro/

try:

cmd = 'edgetpu\_compiler --version'

help\_url = 'https://coral.ai/docs/edgetpu/compiler/'

assert platform.system() == 'Linux', f'export only supported on Linux. See {help\_url}'

if subprocess.run(f'{cmd} >/dev/null', shell=True).returncode != 0:

LOGGER.info(f'\n{prefix} export requires Edge TPU compiler. Attempting install from {help\_url}')

sudo = subprocess.run('sudo --version >/dev/null', shell=True).returncode == 0 # sudo installed on system

for c in (

'curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -',

'echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list',

'sudo apt-get update', 'sudo apt-get install edgetpu-compiler'):

subprocess.run(c if sudo else c.replace('sudo ', ''), shell=True, check=True)

ver = subprocess.run(cmd, shell=True, capture\_output=True, check=True).stdout.decode().split()[-1]

LOGGER.info(f'\n{prefix} starting export with Edge TPU compiler {ver}...')

f = str(file).replace('.pt', '-int8\_edgetpu.tflite') # Edge TPU model

f\_tfl = str(file).replace('.pt', '-int8.tflite') # TFLite model

cmd = f"edgetpu\_compiler -s -d -k 10 --out\_dir {file.parent} {f\_tfl}"

subprocess.run(cmd.split(), check=True)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

def export\_tfjs(file, prefix=colorstr('TensorFlow.js:')):

# YOLOv5 TensorFlow.js export

try:

check\_requirements(('tensorflowjs',))

import re

import tensorflowjs as tfjs

LOGGER.info(f'\n{prefix} starting export with tensorflowjs {tfjs.\_\_version\_\_}...')

f = str(file).replace('.pt', '\_web\_model') # js dir

f\_pb = file.with\_suffix('.pb') # \*.pb path

f\_json = f'{f}/model.json' # \*.json path

cmd = f'tensorflowjs\_converter --input\_format=tf\_frozen\_model ' \

f'--output\_node\_names=Identity,Identity\_1,Identity\_2,Identity\_3 {f\_pb} {f}'

subprocess.run(cmd.split())

with open(f\_json) as j:

json = j.read()

with open(f\_json, 'w') as j: # sort JSON Identity\_\* in ascending order

subst = re.sub(

r'{"outputs": {"Identity.?.?": {"name": "Identity.?.?"}, '

r'"Identity.?.?": {"name": "Identity.?.?"}, '

r'"Identity.?.?": {"name": "Identity.?.?"}, '

r'"Identity.?.?": {"name": "Identity.?.?"}}}', r'{"outputs": {"Identity": {"name": "Identity"}, '

r'"Identity\_1": {"name": "Identity\_1"}, '

r'"Identity\_2": {"name": "Identity\_2"}, '

r'"Identity\_3": {"name": "Identity\_3"}}}', json)

j.write(subst)

LOGGER.info(f'{prefix} export success, saved as {f} ({file\_size(f):.1f} MB)')

return f

except Exception as e:

LOGGER.info(f'\n{prefix} export failure: {e}')

@smart\_inference\_mode()

def run(

data=ROOT / 'data/coco128.yaml', # 'dataset.yaml path'

weights=ROOT / 'yolov5s.pt', # weights path

imgsz=(640, 640), # image (height, width)

batch\_size=1, # batch size

device='cpu', # cuda device, i.e. 0 or 0,1,2,3 or cpu

include=('torchscript', 'onnx'), # include formats

half=False, # FP16 half-precision export

inplace=False, # set YOLOv5 Detect() inplace=True

train=False, # model.train() mode

keras=False, # use Keras

optimize=False, # TorchScript: optimize for mobile

int8=False, # CoreML/TF INT8 quantization

dynamic=False, # ONNX/TF/TensorRT: dynamic axes

simplify=False, # ONNX: simplify model

opset=12, # ONNX: opset version

verbose=False, # TensorRT: verbose log

workspace=4, # TensorRT: workspace size (GB)

nms=False, # TF: add NMS to model

agnostic\_nms=False, # TF: add agnostic NMS to model

topk\_per\_class=100, # TF.js NMS: topk per class to keep

topk\_all=100, # TF.js NMS: topk for all classes to keep

iou\_thres=0.45, # TF.js NMS: IoU threshold

conf\_thres=0.25, # TF.js NMS: confidence threshold

):

t = time.time()

include = [x.lower() for x in include] # to lowercase

fmts = tuple(export\_formats()['Argument'][1:]) # --include arguments

flags = [x in include for x in fmts]

assert sum(flags) == len(include), f'ERROR: Invalid --include {include}, valid --include arguments are {fmts}'

jit, onnx, xml, engine, coreml, saved\_model, pb, tflite, edgetpu, tfjs = flags # export booleans

file = Path(url2file(weights) if str(weights).startswith(('http:/', 'https:/')) else weights) # PyTorch weights

# Load PyTorch model

device = select\_device(device)

if half:

assert device.type != 'cpu' or coreml, '--half only compatible with GPU export, i.e. use --device 0'

assert not dynamic, '--half not compatible with --dynamic, i.e. use either --half or --dynamic but not both'

model = attempt\_load(weights, device=device, inplace=True, fuse=True) # load FP32 model

# Checks

imgsz \*= 2 if len(imgsz) == 1 else 1 # expand

if optimize:

assert device.type == 'cpu', '--optimize not compatible with cuda devices, i.e. use --device cpu'

# Input

gs = int(max(model.stride)) # grid size (max stride)

imgsz = [check\_img\_size(x, gs) for x in imgsz] # verify img\_size are gs-multiples

im = torch.zeros(batch\_size, 3, \*imgsz).to(device) # image size(1,3,320,192) BCHW iDetection

# Update model

model.train() if train else model.eval() # training mode = no Detect() layer grid construction

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

if isinstance(m, Detect):

m.inplace = inplace

m.onnx\_dynamic = dynamic

m.export = True

for \_ in range(2):

y = model(im) # dry runs

if half and not coreml:

im, model = im.half(), model.half() # to FP16

shape = tuple((y[0] if isinstance(y, tuple) else y).shape) # model output shape

LOGGER.info(f"\n{colorstr('PyTorch:')} starting from {file} with output shape {shape} ({file\_size(file):.1f} MB)")

# Exports

f = [''] \* 10 # exported filenames

warnings.filterwarnings(action='ignore', category=torch.jit.TracerWarning) # suppress TracerWarning

if jit:

f[0] = export\_torchscript(model, im, file, optimize)

if engine: # TensorRT required before ONNX

f[1] = export\_engine(model, im, file, train, half, dynamic, simplify, workspace, verbose)

if onnx or xml: # OpenVINO requires ONNX

f[2] = export\_onnx(model, im, file, opset, train, dynamic, simplify)

if xml: # OpenVINO

f[3] = export\_openvino(model, file, half)

if coreml:

\_, f[4] = export\_coreml(model, im, file, int8, half)

# TensorFlow Exports

if any((saved\_model, pb, tflite, edgetpu, tfjs)):

if int8 or edgetpu: # TFLite --int8 bug https://github.com/ultralytics/yolov5/issues/5707

check\_requirements(('flatbuffers==1.12',)) # required before `import tensorflow`

assert not tflite or not tfjs, 'TFLite and TF.js models must be exported separately, please pass only one type.'

model, f[5] = export\_saved\_model(model.cpu(),

im,

file,

dynamic,

tf\_nms=nms or agnostic\_nms or tfjs,

agnostic\_nms=agnostic\_nms or tfjs,

topk\_per\_class=topk\_per\_class,

topk\_all=topk\_all,

iou\_thres=iou\_thres,

conf\_thres=conf\_thres,

keras=keras)

if pb or tfjs: # pb prerequisite to tfjs

f[6] = export\_pb(model, file)

if tflite or edgetpu:

f[7] = export\_tflite(model, im, file, int8=int8 or edgetpu, data=data, nms=nms, agnostic\_nms=agnostic\_nms)

if edgetpu:

f[8] = export\_edgetpu(file)

if tfjs:

f[9] = export\_tfjs(file)

# Finish

f = [str(x) for x in f if x] # filter out '' and None

if any(f):

h = '--half' if half else '' # --half FP16 inference arg

LOGGER.info(f'\nExport complete ({time.time() - t:.2f}s)'

f"\nResults saved to {colorstr('bold', file.parent.resolve())}"

f"\nDetect: python detect.py --weights {f[-1]} {h}"

f"\nValidate: python val.py --weights {f[-1]} {h}"

f"\nPyTorch Hub: model = torch.hub.load('ultralytics/yolov5', 'custom', '{f[-1]}')"

f"\nVisualize: https://netron.app")

return f # return list of exported files/dirs

def parse\_opt():

parser = argparse.ArgumentParser()

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

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

parser.add\_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640, 640], help='image (h, w)')

parser.add\_argument('--batch-size', type=int, default=1, help='batch size')

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

parser.add\_argument('--half', action='store\_true', help='FP16 half-precision export')

parser.add\_argument('--inplace', action='store\_true', help='set YOLOv5 Detect() inplace=True')

parser.add\_argument('--train', action='store\_true', help='model.train() mode')

parser.add\_argument('--keras', action='store\_true', help='TF: use Keras')

parser.add\_argument('--optimize', action='store\_true', help='TorchScript: optimize for mobile')

parser.add\_argument('--int8', action='store\_true', help='CoreML/TF INT8 quantization')

parser.add\_argument('--dynamic', action='store\_true', help='ONNX/TF/TensorRT: dynamic axes')

parser.add\_argument('--simplify', action='store\_true', help='ONNX: simplify model')

parser.add\_argument('--opset', type=int, default=12, help='ONNX: opset version')

parser.add\_argument('--verbose', action='store\_true', help='TensorRT: verbose log')

parser.add\_argument('--workspace', type=int, default=4, help='TensorRT: workspace size (GB)')

parser.add\_argument('--nms', action='store\_true', help='TF: add NMS to model')

parser.add\_argument('--agnostic-nms', action='store\_true', help='TF: add agnostic NMS to model')

parser.add\_argument('--topk-per-class', type=int, default=100, help='TF.js NMS: topk per class to keep')

parser.add\_argument('--topk-all', type=int, default=100, help='TF.js NMS: topk for all classes to keep')

parser.add\_argument('--iou-thres', type=float, default=0.45, help='TF.js NMS: IoU threshold')

parser.add\_argument('--conf-thres', type=float, default=0.25, help='TF.js NMS: confidence threshold')

parser.add\_argument('--include',

nargs='+',

default=['torchscript', 'onnx'],

help='torchscript, onnx, openvino, engine, coreml, saved\_model, pb, tflite, edgetpu, tfjs')

opt = parser.parse\_args()

print\_args(vars(opt))

return opt

def main(opt):

for opt.weights in (opt.weights if isinstance(opt.weights, list) else [opt.weights]):

run(\*\*vars(opt))

if \_\_name\_\_ == "\_\_main\_\_":

opt = parse\_opt()

main(opt)

hubconf.py

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

"""

PyTorch Hub models https://pytorch.org/hub/ultralytics\_yolov5/

Usage:

import torch

model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

model = torch.hub.load('ultralytics/yolov5:master', 'custom', 'path/to/yolov5s.onnx') # file from branch

"""

import torch

def \_create(name, pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):

"""Creates or loads a YOLOv5 model

Arguments:

name (str): model name 'yolov5s' or path 'path/to/best.pt'

pretrained (bool): load pretrained weights into the model

channels (int): number of input channels

classes (int): number of model classes

autoshape (bool): apply YOLOv5 .autoshape() wrapper to model

verbose (bool): print all information to screen

device (str, torch.device, None): device to use for model parameters

Returns:

YOLOv5 model

"""

from pathlib import Path

from models.common import AutoShape, DetectMultiBackend

from models.experimental import attempt\_load

from models.yolo import Model

from utils.downloads import attempt\_download

from utils.general import LOGGER, check\_requirements, intersect\_dicts, logging

from utils.torch\_utils import select\_device

if not verbose:

LOGGER.setLevel(logging.WARNING)

check\_requirements(exclude=('tensorboard', 'thop', 'opencv-python'))

name = Path(name)

path = name.with\_suffix('.pt') if name.suffix == '' and not name.is\_dir() else name # checkpoint path

try:

device = select\_device(device)

if pretrained and channels == 3 and classes == 80:

try:

model = DetectMultiBackend(path, device=device, fuse=autoshape) # detection model

if autoshape:

model = AutoShape(model) # for file/URI/PIL/cv2/np inputs and NMS

except Exception:

model = attempt\_load(path, device=device, fuse=False) # arbitrary model

else:

cfg = list((Path(\_\_file\_\_).parent / 'models').rglob(f'{path.stem}.yaml'))[0] # model.yaml path

model = Model(cfg, channels, classes) # create model

if pretrained:

ckpt = torch.load(attempt\_download(path), map\_location=device) # load

csd = ckpt['model'].float().state\_dict() # checkpoint state\_dict as FP32

csd = intersect\_dicts(csd, model.state\_dict(), exclude=['anchors']) # intersect

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

if len(ckpt['model'].names) == classes:

model.names = ckpt['model'].names # set class names attribute

if not verbose:

LOGGER.setLevel(logging.INFO) # reset to default

return model.to(device)

except Exception as e:

help\_url = 'https://github.com/ultralytics/yolov5/issues/36'

s = f'{e}. Cache may be out of date, try `force\_reload=True` or see {help\_url} for help.'

raise Exception(s) from e

def custom(path='path/to/model.pt', autoshape=True, \_verbose=True, device=None):

# YOLOv5 custom or local model

return \_create(path, autoshape=autoshape, verbose=\_verbose, device=device)

def yolov5n(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-nano model https://github.com/ultralytics/yolov5

return \_create('yolov5n', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5s(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-small model https://github.com/ultralytics/yolov5

return \_create('yolov5s', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5m(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-medium model https://github.com/ultralytics/yolov5

return \_create('yolov5m', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5l(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-large model https://github.com/ultralytics/yolov5

return \_create('yolov5l', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5x(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-xlarge model https://github.com/ultralytics/yolov5

return \_create('yolov5x', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5n6(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-nano-P6 model https://github.com/ultralytics/yolov5

return \_create('yolov5n6', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5s6(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-small-P6 model https://github.com/ultralytics/yolov5

return \_create('yolov5s6', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5m6(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-medium-P6 model https://github.com/ultralytics/yolov5

return \_create('yolov5m6', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5l6(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-large-P6 model https://github.com/ultralytics/yolov5

return \_create('yolov5l6', pretrained, channels, classes, autoshape, \_verbose, device)

def yolov5x6(pretrained=True, channels=3, classes=80, autoshape=True, \_verbose=True, device=None):

# YOLOv5-xlarge-P6 model https://github.com/ultralytics/yolov5

return \_create('yolov5x6', pretrained, channels, classes, autoshape, \_verbose, device)

if \_\_name\_\_ == '\_\_main\_\_':

import argparse

from pathlib import Path

import numpy as np

from PIL import Image

from utils.general import cv2, print\_args

# Argparser

parser = argparse.ArgumentParser()

parser.add\_argument('--model', type=str, default='yolov5s', help='model name')

opt = parser.parse\_args()

print\_args(vars(opt))

# Model

model = \_create(name=opt.model, pretrained=True, channels=3, classes=80, autoshape=True, verbose=True)

# model = custom(path='path/to/model.pt') # custom

# Images

imgs = [

'data/images/zidane.jpg', # filename

Path('data/images/zidane.jpg'), # Path

'https://ultralytics.com/images/zidane.jpg', # URI

cv2.imread('data/images/bus.jpg')[:, :, ::-1], # OpenCV

Image.open('data/images/bus.jpg'), # PIL

np.zeros((320, 640, 3))] # numpy

# Inference

results = model(imgs, size=320) # batched inference

# Results

results.print()

results.save()

train.py

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

"""

Train a YOLOv5 model on a custom dataset.

Models and datasets download automatically from the latest YOLOv5 release.

Models: https://github.com/ultralytics/yolov5/tree/master/models

Datasets: https://github.com/ultralytics/yolov5/tree/master/data

Tutorial: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

Usage:

$ python path/to/train.py --data coco128.yaml --weights yolov5s.pt --img 640 # from pretrained (RECOMMENDED)

$ python path/to/train.py --data coco128.yaml --weights '' --cfg yolov5s.yaml --img 640 # from scratch

"""

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.optim import 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, is\_url

from utils.general import (LOGGER, check\_amp, check\_dataset, check\_file, check\_git\_status, check\_img\_size,

check\_requirements, check\_suffix, 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, yaml\_save)

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, smart\_DDP, smart\_optimizer,

smart\_resume, 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()))

opt.hyp = hyp.copy() # for saving hyps to checkpoints

# Save run settings

if not evolve:

yaml\_save(save\_dir / 'hyp.yaml', hyp)

yaml\_save(save\_dir / 'opt.yaml', vars(opt))

# Loggers

data\_dict = None

if RANK in {-1, 0}:

loggers = Loggers(save\_dir, weights, opt, hyp, LOGGER) # loggers instance

if loggers.clearml:

data\_dict = loggers.clearml.data\_dict # None if no ClearML dataset or filled in by ClearML

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(opt.seed + 1 + RANK, deterministic=True)

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

# v.register\_hook(lambda x: torch.nan\_to\_num(x)) # NaN to 0 (commented for erratic training results)

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

optimizer = smart\_optimizer(model, opt.optimizer, hyp['lr0'], hyp['momentum'], hyp['weight\_decay'])

# 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

best\_fitness, start\_epoch = 0.0, 0

if pretrained:

if resume:

best\_fitness, start\_epoch, epochs = smart\_resume(ckpt, optimizer, ema, weights, epochs, resume)

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)

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

mlc = int(labels[:, 0].max()) # max label class

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:

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:

model = smart\_DDP(model)

# 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()

nb = len(train\_loader) # number of batches

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, stop = EarlyStopping(patience=opt.patience), False

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 == 0 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 - https://pytorch.org/docs/master/notes/amp\_examples.html

if ni - last\_opt\_step >= accumulate:

scaler.unscale\_(optimizer) # unscale gradients

torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=10.0) # clip gradients

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,

half=amp,

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]

stop = stopper(epoch=epoch, fitness=fi) # early stop check

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,

'opt': vars(opt),

'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)

# EarlyStopping

if RANK != -1: # if DDP training

broadcast\_list = [stop if RANK == 0 else None]

dist.broadcast\_object\_list(broadcast\_list, 0) # broadcast 'stop' to all ranks

if RANK != 0:

stop = broadcast\_list[0]

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('--seed', type=int, default=0, help='Global training seed')

parser.add\_argument('--local\_rank', type=int, default=-1, help='Automatic DDP Multi-GPU argument, 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')

return parser.parse\_known\_args()[0] if known else parser.parse\_args()

def main(opt, callbacks=Callbacks()):

# Checks

if RANK in {-1, 0}:

print\_args(vars(opt))

check\_git\_status()

check\_requirements()

# Resume

if opt.resume and not (check\_wandb\_resume(opt) or opt.evolve): # resume from specified or most recent last.pt

last = Path(check\_file(opt.resume) if isinstance(opt.resume, str) else get\_latest\_run())

opt\_yaml = last.parent.parent / 'opt.yaml' # train options yaml

opt\_data = opt.data # original dataset

if opt\_yaml.is\_file():

with open(opt\_yaml, errors='ignore') as f:

d = yaml.safe\_load(f)

else:

d = torch.load(last, map\_location='cpu')['opt']

opt = argparse.Namespace(\*\*d) # replace

opt.cfg, opt.weights, opt.resume = '', str(last), True # reinstate

if is\_url(opt\_data):

opt.data = check\_file(opt\_data) # avoid HUB resume auth timeout

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)

# 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

if opt.noautoanchor:

del hyp['anchors'], meta['anchors']

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)