**Logbook**

From: 08/01/2022 To: 28/04/2022

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Month** | **List the main activities** (only few words per activity) | **Interaction with the supervisor** | | | **Any other form of supervisory interaction** (second supervisor, industry, fellows etc.) |
| Number of meetings | Mode of meeting (face- to-face, online e.g., Skype, WeChat etc.) | Number of emails exchanged |
| 2022.01 | 1. Gaining the value for the original YOLOv5 model.  2. Adjusting the structure of the dataset according to the results. | 2 | Email and face-to-face | 4 | Work with another student who did the similar project with me Yuhua Nie, and discussed about the task. |
| 2022.02. | 1. Adjusting the structure of the original YOLOv5 network model. | 2 | Email and WeChat | 3 | Discussed about the model effect with a postgraduate, and got some advice from her. |
| 2022.03. | 1. Training the improved model and get the weight value.  2. Testing the effect and obtaining the improved model. | 2 | Email and WeChat | 2 | Work with another student who did the similar project with me Yuhua Nie. |
| 2022.04. | 1. Final report writing.  2. Making some improvement for the final project.  3. Making a summary for the whole project. | 1 | Email | 2 | Work with another student who did the similar project with me Yuhua Nie. |

2022.01.10.

I got the first version results for the YOLOv5 training model.

The original YOLOv5 network code are shown below:

# parameters

nc: 1 # number of classes

depth\_multiple: 0.33 # model depth multiple

width\_multiple: 0.50 # layer channel multiple

# anchors

anchors:

- [10,13, 16,30, 33,23] # P3/8

- [30,61, 62,45, 59,119] # P4/16

- [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone

backbone:

# [from, number, module, args]

[[-1, 1, Focus, [64, 3]], # 0-P1/2

[-1, 1, Conv, [128, 3, 2]], # 1-P2/4

[-1, 3, C3, [128]],

[-1, 1, Conv, [256, 3, 2]], # 3-P3/8

[-1, 9, C3, [256]],

[-1, 1, Conv, [512, 3, 2]], # 5-P4/16

[-1, 9, C3, [512]],

[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32

[-1, 1, SPP, [1024, [5, 9, 13]]],

[-1, 3, C3, [1024, False]], # 9

]

# YOLOv5 head

head:

[[-1, 1, Conv, [512, 1, 1]],

[-1, 1, nn.Upsample, [None, 2, 'nearest']],

[[-1, 6], 1, Concat, [1]], # cat backbone P4

[-1, 3, C3, [512, False]], # 13

[-1, 1, Conv, [256, 1, 1]],

[-1, 1, nn.Upsample, [None, 2, 'nearest']],

[[-1, 4], 1, Concat, [1]], # cat backbone P3

[-1, 3, C3, [256, False]], # 17 (P3/8-small)

[-1, 1, Conv, [256, 3, 2]],

[[-1, 14], 1, Concat, [1]], # cat head P4

[-1, 3, C3, [512, False]], # 20 (P4/16-medium)

[-1, 1, Conv, [512, 3, 2]],

[[-1, 10], 1, Concat, [1]], # cat head P5

[-1, 3, C3, [1024, False]], # 23 (P5/32-large)

[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)

]

2022.02.15.

I adjust the structure of the dataset, to improve the generalization ability of the model. The code I divide the train set and test set according to the rate 8:2 is shown below:

import xml.etree.ElementTree as ET

import pickle

import os

from os import listdir, getcwd

from os.path import join

import random

from shutil import copyfile

classes = "face"

# classes=["ball"]

TRAIN\_RATIO = 80

def clear\_hidden\_files(path):

dir\_list = os.listdir(path)

for i in dir\_list:

abspath = os.path.join(os.path.abspath(path), i)

if os.path.isfile(abspath):

if i.startswith(".\_"):

os.remove(abspath)

else:

clear\_hidden\_files(abspath)

def convert(size, box):

dw = 1. / size[0]

dh = 1. / size[1]

x = (box[0] + box[1]) / 2.0

y = (box[2] + box[3]) / 2.0

w = box[1] - box[0]

h = box[3] - box[2]

x = x \* dw

w = w \* dw

y = y \* dh

h = h \* dh

return (x, y, w, h)

def convert\_annotation(image\_id):

in\_file = open('VOCdevkit/VOC2007/Annotations/%s.xml' % image\_id)

out\_file = open('VOCdevkit/VOC2007/YOLOLabels/%s.txt' % image\_id, 'w')

tree = ET.parse(in\_file)

root = tree.getroot()

size = root.find('size')

w = int(size.find('width').text)

h = int(size.find('height').text)

for obj in root.iter('object'):

difficult = obj.find('difficult').text

cls = obj.find('name').text

if cls not in classes or int(difficult) == 1:

continue

cls\_id = classes.index(cls)

xmlbox = obj.find('bndbox')

b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),

float(xmlbox.find('ymax').text))

bb = convert((w, h), b)

out\_file.write(str(cls\_id) + " " + " ".join([str(a) for a in bb]) + '\n')

in\_file.close()

out\_file.close()

wd = os.getcwd()

wd = os.getcwd()

data\_base\_dir = os.path.join(wd, "D:\Python project\Final Project\yolov5-5.0\VOCdevkit/")

if not os.path.isdir(data\_base\_dir):

os.mkdir(data\_base\_dir)

work\_sapce\_dir = os.path.join(wd, "D:\Python project\Final Project\yolov5-5.0\VOCdevkit\VOC2007/")

if not os.path.isdir(work\_sapce\_dir):

os.mkdir(work\_sapce\_dir)

annotation\_dir = os.path.join(work\_sapce\_dir, "Annotations/")

if not os.path.isdir(annotation\_dir):

os.mkdir(annotation\_dir)

clear\_hidden\_files(annotation\_dir)

image\_dir = os.path.join(work\_sapce\_dir, "JPEGImages/")

if not os.path.isdir(image\_dir):

os.mkdir(image\_dir)

clear\_hidden\_files(image\_dir)

yolo\_labels\_dir = os.path.join(work\_sapce\_dir, "YOLOLabels/")

if not os.path.isdir(yolo\_labels\_dir):

os.mkdir(yolo\_labels\_dir)

clear\_hidden\_files(yolo\_labels\_dir)

yolov5\_images\_dir = os.path.join(data\_base\_dir, "images/")

if not os.path.isdir(yolov5\_images\_dir):

os.mkdir(yolov5\_images\_dir)

clear\_hidden\_files(yolov5\_images\_dir)

yolov5\_labels\_dir = os.path.join(data\_base\_dir, "labels/")

if not os.path.isdir(yolov5\_labels\_dir):

os.mkdir(yolov5\_labels\_dir)

clear\_hidden\_files(yolov5\_labels\_dir)

yolov5\_images\_train\_dir = os.path.join(yolov5\_images\_dir, "train/")

if not os.path.isdir(yolov5\_images\_train\_dir):

os.mkdir(yolov5\_images\_train\_dir)

clear\_hidden\_files(yolov5\_images\_train\_dir)

yolov5\_images\_test\_dir = os.path.join(yolov5\_images\_dir, "val/")

if not os.path.isdir(yolov5\_images\_test\_dir):

os.mkdir(yolov5\_images\_test\_dir)

clear\_hidden\_files(yolov5\_images\_test\_dir)

yolov5\_labels\_train\_dir = os.path.join(yolov5\_labels\_dir, "train/")

if not os.path.isdir(yolov5\_labels\_train\_dir):

os.mkdir(yolov5\_labels\_train\_dir)

clear\_hidden\_files(yolov5\_labels\_train\_dir)

yolov5\_labels\_test\_dir = os.path.join(yolov5\_labels\_dir, "val/")

if not os.path.isdir(yolov5\_labels\_test\_dir):

os.mkdir(yolov5\_labels\_test\_dir)

clear\_hidden\_files(yolov5\_labels\_test\_dir)

train\_file = open(os.path.join(wd, "../yolov5\_train.txt"), 'w')

test\_file = open(os.path.join(wd, "../yolov5\_val.txt"), 'w')

train\_file.close()

test\_file.close()

train\_file = open(os.path.join(wd, "../yolov5\_train.txt"), 'a')

test\_file = open(os.path.join(wd, "../yolov5\_val.txt"), 'a')

list\_imgs = os.listdir(image\_dir) # list image files

prob = random.randint(1, 100)

print("Probability: %d" % prob)

for i in range(0, len(list\_imgs)):

path = os.path.join(image\_dir, list\_imgs[i])

if os.path.isfile(path):

image\_path = image\_dir + list\_imgs[i]

voc\_path = list\_imgs[i]

(nameWithoutExtention, extention) = os.path.splitext(os.path.basename(image\_path))

(voc\_nameWithoutExtention, voc\_extention) = os.path.splitext(os.path.basename(voc\_path))

annotation\_name = nameWithoutExtention + '.xml'

annotation\_path = os.path.join(annotation\_dir, annotation\_name)

label\_name = nameWithoutExtention + '.txt'

label\_path = os.path.join(yolo\_labels\_dir, label\_name)

prob = random.randint(1, 100)

print("Probability: %d" % prob)

if (prob < TRAIN\_RATIO): # train dataset

if os.path.exists(annotation\_path):

train\_file.write(image\_path + '\n')

convert\_annotation(nameWithoutExtention) # convert label

copyfile(image\_path, yolov5\_images\_train\_dir + voc\_path)

copyfile(label\_path, yolov5\_labels\_train\_dir + label\_name)

else: # test dataset

if os.path.exists(annotation\_path):

test\_file.write(image\_path + '\n')

convert\_annotation(nameWithoutExtention) # convert label

copyfile(image\_path, yolov5\_images\_test\_dir + voc\_path)

copyfile(label\_path, yolov5\_labels\_test\_dir + label\_name)

train\_file.close()

test\_file.close()

2022.03.20.

I made some improvement for the original YOLOv5 model, and got the final version improved model. The code is shown below:

# parameters

nc: 1 # number of classes

depth\_multiple: 0.33 # model depth multiple

width\_multiple: 0.50 # layer channel multiple

# anchors

anchors:

- [5,6, 8,14, 15,11] # P2/4

- [10,13, 16,30, 33,23] # P3/8

- [30,61, 62,45, 59,119] # P4/16

- [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone

backbone:

# [from, number, module, args]

[[-1, 1, Focus, [64, 3]], # 0-P1/2:320

[-1, 1, Conv, [128, 3, 2]], # 1-P2/4:160

[-1, 3, BottleneckCSP, [128]],

[-1, 1, Conv, [256, 3, 2]], # 3-P3/8:80

[-1, 9, BottleneckCSP, [256]],

[-1, 1, Conv, [512, 3, 2]], # 5-P4/16:40

[-1, 9, BottleneckCSP, [512]],

[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32:20

[-1, 1, SPP, [1024, [5, 9, 13]]],

[-1, 3, BottleneckCSP, [1024, False]], # 9:20

]

# YOLOv5 head

head:

[[-1, 1, Conv, [512, 1, 1]], #20\*20

[-1, 1, nn.Upsample, [None, 2, 'nearest']], #40\*40

[[-1, 6], 1, Concat, [1]], # cat backbone P4 40\*40

[-1, 3, BottleneckCSP, [512, False]], # 13 40\*40

[-1, 1, Conv, [512, 1, 1]], #40\*40

[-1, 1, nn.Upsample, [None, 2, 'nearest']],

[[-1, 4], 1, Concat, [1]], # cat backbone P3 80\*80

[-1, 3, BottleneckCSP, [512, False]], # 17 (P3/8-small) 80\*80

[-1, 1, Conv, [256, 1, 1]], #18 80\*80

[-1, 1, nn.Upsample, [None, 2, 'nearest']], #19 160\*160

[[-1, 2], 1, Concat, [1]], #20 cat backbone p2 160\*160

[-1, 3, BottleneckCSP, [256, False]], #21 (P2/4-tiny) 160\*160

[-1, 1, Conv, [256, 3, 2]], #22 80\*80

[[-1, 18], 1, Concat, [1]], #23 80\*80

[-1, 3, BottleneckCSP, [256, False]], #24 80\*80

[-1, 1, Conv, [256, 3, 2]], #25 40\*40

[[-1, 14], 1, Concat, [1]], # 26 cat head P4 40\*40

[-1, 3, BottleneckCSP, [512, False]], # 27 (P4/16-medium) 40\*40

[-1, 1, Conv, [512, 3, 2]], #28 20\*20

[[-1, 10], 1, Concat, [1]], #29 cat head P5 #20\*20

[-1, 3, BottleneckCSP, [1024, False]], # 30 (P5/32-large) 20\*20

[[21, 24, 27, 30], 1, Detect, [nc, anchors]], # Detect(p2, P3, P4, P5)

]

2022.04.05.

I use the improved model to detect the person in video, and the results is great. The training and detecting codes are shown below:

Training code:

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 # import test.py to get mAP after each epoch

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

from utils.wandb\_logging.wandb\_utils import WandbLogger, check\_wandb\_resume

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

def train(hyp, opt, device, tb\_writer=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

# Directories

wdir = save\_dir / 'weights'

wdir.mkdir(parents=True, exist\_ok=True) # make dir

last = wdir / 'last.pt'

best = wdir / 'best.pt'

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

# Save run settings

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)

# Configure

plots = not opt.evolve # create plots

cuda = device.type != 'cpu'

init\_seeds(2 + rank)

with open(opt.data) as f:

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

is\_coco = opt.data.endswith('coco.yaml')

# Logging- Doing this before checking the dataset. Might update data\_dict

loggers = {'wandb': None} # loggers dict

if rank in [-1, 0]:

opt.hyp = hyp # add hyperparameters

run\_id = torch.load(weights).get('wandb\_id') if weights.endswith('.pt') and os.path.isfile(weights) else None

wandb\_logger = WandbLogger(opt, Path(opt.save\_dir).stem, run\_id, data\_dict)

loggers['wandb'] = wandb\_logger.wandb

data\_dict = wandb\_logger.data\_dict

if wandb\_logger.wandb:

weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp # WandbLogger might update weights, epochs if resuming

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

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

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

# Model

pretrained = weights.endswith('.pt')

if pretrained:

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

attempt\_download(weights) # download if not found locally

ckpt = torch.load(weights, map\_location=device) # load checkpoint

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

exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and not opt.resume else [] # exclude keys

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

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

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

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

else:

model = Model(opt.cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device) # create

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

check\_dataset(data\_dict) # check

train\_path = data\_dict['train']

test\_path = data\_dict['val']

# Freeze

freeze = [] # parameter names to freeze (full or partial)

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

v.requires\_grad = True # train all layers

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

print('freezing %s' % k)

v.requires\_grad = False

# Optimizer

nbs = 64 # nominal batch size

accumulate = max(round(nbs / total\_batch\_size), 1) # accumulate loss before optimizing

hyp['weight\_decay'] \*= total\_batch\_size \* accumulate / nbs # scale weight\_decay

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

pg0, pg1, pg2 = [], [], [] # optimizer parameter groups

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

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

pg2.append(v.bias) # biases

if isinstance(v, nn.BatchNorm2d):

pg0.append(v.weight) # no decay

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

pg1.append(v.weight) # apply decay

if opt.adam:

optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) # adjust beta1 to momentum

else:

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

optimizer.add\_param\_group({'params': pg1, 'weight\_decay': hyp['weight\_decay']}) # add pg1 with weight\_decay

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

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

del pg0, pg1, pg2

# Scheduler 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'] # linear

else:

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

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

# plot\_lr\_scheduler(optimizer, scheduler, epochs)

# EMA

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

# Resume

start\_epoch, best\_fitness = 0, 0.0

if pretrained:

# Optimizer

if ckpt['optimizer'] is not None:

optimizer.load\_state\_dict(ckpt['optimizer'])

best\_fitness = ckpt['best\_fitness']

# EMA

if ema and ckpt.get('ema'):

ema.ema.load\_state\_dict(ckpt['ema'].float().state\_dict())

ema.updates = ckpt['updates']

# Results

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

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

# Epochs

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'] # finetune additional epochs

del ckpt, state\_dict

# Image sizes

gs = max(int(model.stride.max()), 32) # grid size (max stride)

nl = model.model[-1].nl # number of detection layers (used for scaling hyp['obj'])

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

# DP mode

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

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

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() # max label class

nb = len(dataloader) # number of batches

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, # testloader

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. # frequency

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

if plots:

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

if tb\_writer:

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

# Anchors

if not opt.noautoanchor:

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

model.half().float() # pre-reduce anchor precision

# DDP mode

if cuda and rank != -1:

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

# nn.MultiheadAttention incompatibility with DDP https://github.com/pytorch/pytorch/issues/26698

find\_unused\_parameters=any(isinstance(layer, nn.MultiheadAttention) for layer in model.modules()))

# Model parameters

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.gr = 1.0 # iou loss ratio (obj\_loss = 1.0 or iou)

model.class\_weights = labels\_to\_class\_weights(dataset.labels, nc).to(device) \* nc # attach class weights

model.names = names

# Start training

t0 = time.time()

nw = max(round(hyp['warmup\_epochs'] \* nb), 1000) # number of warmup iterations, max(3 epochs, 1k iterations)

# nw = min(nw, (epochs - start\_epoch) / 2 \* nb) # limit warmup to < 1/2 of training

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 = amp.GradScaler(enabled=cuda)

compute\_loss = ComputeLoss(model) # init loss class

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): # epoch ------------------------------------------------------------------

model.train()

# Update image weights (optional)

if opt.image\_weights:

# Generate indices

if rank in [-1, 0]:

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

# Broadcast if DDP

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()

# Update mosaic border

# 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(4, device=device) # mean losses

if rank != -1:

dataloader.sampler.set\_epoch(epoch)

pbar = enumerate(dataloader)

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

if rank in [-1, 0]:

pbar = tqdm(pbar, total=nb) # progress bar

optimizer.zero\_grad()

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

ni = i + nb \* epoch # number integrated batches (since train start)

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

# Warmup

if ni <= nw:

xi = [0, nw] # x interp

# model.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 / 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']])

# 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 = 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)) # loss scaled by batch\_size

if rank != -1:

loss \*= opt.world\_size # gradient averaged between devices in DDP mode

if opt.quad:

loss \*= 4.

# Backward

scaler.scale(loss).backward()

# Optimize

if ni % accumulate == 0:

scaler.step(optimizer) # optimizer.step

scaler.update()

optimizer.zero\_grad()

if ema:

ema.update(model)

# Print

if rank in [-1, 0]:

mloss = (mloss \* i + loss\_items) / (i + 1) # update mean losses

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)

# Plot

if plots and ni < 3:

f = save\_dir / f'train\_batch{ni}.jpg' # filename

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(torch.jit.trace(model, imgs, strict=False), []) # add model graph

elif plots and ni == 10 and wandb\_logger.wandb:

wandb\_logger.log({"Mosaics": [wandb\_logger.wandb.Image(str(x), caption=x.name) for x in

save\_dir.glob('train\*.jpg') if x.exists()]})

# end batch ------------------------------------------------------------------------------------------------

# end epoch ----------------------------------------------------------------------------------------------------

# Scheduler

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

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: # Calculate mAP

wandb\_logger.current\_epoch = epoch + 1

results, maps, times = test.test(data\_dict,

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,

wandb\_logger=wandb\_logger,

compute\_loss=compute\_loss,

is\_coco=is\_coco)

# Write

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

f.write(s + '%10.4g' \* 7 % results + '\n') # append metrics, 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', # train loss

'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'] # params

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

if tb\_writer:

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

if wandb\_logger.wandb:

wandb\_logger.log({tag: x}) # W&B

# Update best mAP

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

if fi > best\_fitness:

best\_fitness = fi

wandb\_logger.end\_epoch(best\_result=best\_fitness == fi)

# Save model

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

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

'updates': ema.updates,

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

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

# Save last, best and delete

torch.save(ckpt, last)

if best\_fitness == fi:

torch.save(ckpt, best)

if wandb\_logger.wandb:

if ((epoch + 1) % opt.save\_period == 0 and not final\_epoch) and opt.save\_period != -1:

wandb\_logger.log\_model(

last.parent, opt, epoch, fi, best\_model=best\_fitness == fi)

del ckpt

# end epoch ----------------------------------------------------------------------------------------------------

# end training

if rank in [-1, 0]:

# Plots

if plots:

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

if wandb\_logger.wandb:

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

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

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

# Test 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): # speed, mAP tests

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,

is\_coco=is\_coco)

# Strip optimizers

final = best if best.exists() else last # final model

for f in last, best:

if f.exists():

strip\_optimizer(f) # strip optimizers

if opt.bucket:

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

if wandb\_logger.wandb and not opt.evolve: # Log the stripped model

wandb\_logger.wandb.log\_artifact(str(final), type='model',

name='run\_' + wandb\_logger.wandb\_run.id + '\_model',

aliases=['last', 'best', 'stripped'])

wandb\_logger.finish\_run()

else:

dist.destroy\_process\_group()

torch.cuda.empty\_cache()

return results

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

parser = argparse.ArgumentParser()

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

parser.add\_argument('--cfg', type=str, default='models/face1.yaml', help='model.yaml path')

parser.add\_argument('--data', type=str, default='data/face.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=200)

parser.add\_argument('--batch-size', type=int, default=6, 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('--workers', type=int, default=16, 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')

parser.add\_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')

parser.add\_argument('--upload\_dataset', action='store\_true', help='Upload dataset as W&B artifact table')

parser.add\_argument('--bbox\_interval', type=int, default=-1, help='Set bounding-box image logging interval for W&B')

parser.add\_argument('--save\_period', type=int, default=-1, help='Log model after every "save\_period" epoch')

parser.add\_argument('--artifact\_alias', type=str, default="latest", help='version of dataset artifact to be used')

opt = parser.parse\_args()

# Set DDP variables

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()

# Resume

wandb\_run = check\_wandb\_resume(opt)

if opt.resume and not wandb\_run: # resume an interrupted run

ckpt = opt.resume if isinstance(opt.resume, str) else get\_latest\_run() # specified or most recent path

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

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)) # replace

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

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) # check files

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))) # extend to 2 sizes (train, test)

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) # increment run

# DDP mode

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://') # distributed backend

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

# Hyperparameters

with open(opt.hyp) as f:

hyp = yaml.load(f, Loader=yaml.SafeLoader) # load hyps

# Train

logger.info(opt)

if not opt.evolve:

tb\_writer = None # init loggers

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

prefix = colorstr('tensorboard: ')

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

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

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

# 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 # only test/save final epoch

# ei = [isinstance(x, (int, float)) for x in hyp.values()] # evolvable indices

yaml\_file = Path(opt.save\_dir) / 'hyp\_evolved.yaml' # save best result here

if opt.bucket:

os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket) # download evolve.txt if exists

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

if Path('evolve.txt').exists(): # if evolve.txt exists: select best hyps and mutate

# Select parent(s)

parent = 'single' # parent selection method: 'single' or 'weighted'

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

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() # weights

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([x[0] for x in meta.values()]) # 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)

# Write mutation results

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

# Plot results

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}')

The detecting code:

import argparse

import time

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

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') # save inference images

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

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

# Directories

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

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

# Initialize

set\_logging()

device = select\_device(opt.device)

half = device.type != 'cpu' # half precision only supported on CUDA

# Load model

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

stride = int(model.stride.max()) # model stride

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

if half:

model.half() # to FP16

# Second-stage classifier

classify = False

if classify:

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

modelc.load\_state\_dict(torch.load('weights/resnet101.pt', map\_location=device)['model']).to(device).eval()

# Set Dataloader

vid\_path, vid\_writer = None, None

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)

else:

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

# Get names and colors

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

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

# Run inference

if device.type != 'cpu':

model(torch.zeros(1, 3, imgsz, imgsz).to(device).type\_as(next(model.parameters()))) # run once

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() # uint8 to fp16/32

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

if img.ndimension() == 3:

img = img.unsqueeze(0)

# Inference

t1 = time\_synchronized()

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

# Apply NMS

pred = non\_max\_suppression(pred, opt.conf\_thres, opt.iou\_thres, classes=opt.classes, agnostic=opt.agnostic\_nms)

t2 = time\_synchronized()

# Apply Classifier

if classify:

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

# Process detections

for i, det in enumerate(pred): # detections per image

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) # to 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:] # print string

gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh

if len(det):

# Rescale boxes from img\_size to im0 size

det[:, :4] = scale\_coords(img.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 opt.save\_conf else (cls, \*xywh) # label format

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

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

if save\_img or view\_img: # Add bbox to image

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

plot\_one\_box(xyxy, im0, label=label, color=colors[int(cls)], line\_thickness=3)

# Print time (inference + NMS)

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

# Stream results

if view\_img:

cv2.imshow(str(p), im0)

cv2.waitKey(1) # wait for 1

# Save results (image with detections)

if save\_img:

if dataset.mode == 'image':

cv2.imwrite(save\_path, im0)

else: # 'video' or 'stream'

if vid\_path != save\_path: # new video

vid\_path = save\_path

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

vid\_writer.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 += '.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='weights/best.pt', help='model.pt path(s)')

parser.add\_argument('--source', type=str, default='detect\_dataset/face4.JPG', help='source') # file/folder, 0 for webcam

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', default=True)

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: # update all models (to fix SourceChangeWarning)

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

detect()

strip\_optimizer(opt.weights)

else:

detect()