电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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.

电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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)
]
```

电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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()
```

电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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)
 ]
```

电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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}')
```

电子科技大学
格拉斯哥学院
Glasgow College, UESTC

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()
```