SISTEM DETEKSI KERUSAKAN GEDUNG MENGGUNAKAN ALGORITMA *YOU ONLY LOOK ONCE* DENGAN *UNMANNED AERO VEHICLE*

SKRIPSI

Farros Hilmi Zain

4317020027

PROGRAM STUDI INSTRUMENTASI DAN KONTROL INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2021

SISTEM DETEKSI KERUSAKAN GEDUNG MENGGUNAKAN ALGORITMA *YOU ONLY LOOK ONCE* DENGAN *UNMANNED AERO VEHICLE*

SKRIPSI

Farros Hilmi Zain

4317020027

PROGRAM STUDI INSTRUMENTASI DAN KONTROL INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2021

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah Hasil Karya Saya Sendiri dan Semua Sumber baik yang dikutip dan dirujuk telah Saya Nyatakan dengan Benar

Nama : **Farros Hilmi Zain**

NIM : **4317020027**

Tanda Tangan :

Tanggal : **6 Juli 2021**

LEMBAR PENGESAHAN SKRIPSI

Skripsi diajukan oleh :

Nama : Farros Hilmi Zain

NIM : 4317020027

Program Studi : Instrumentasi dan Kontrol Industri

Judul Tugas Akhir : Sistem Deteksi Kerusakan Gedung Menggunakan Algoritma *You Only Look Once* Dengan *Unmanned Aero Vehicle*

Telah diuji oleh tim penguji dalam Sidang Tugas Akhir 1 pada Jumat, 16 Juli 2021 dan dinyatakan **LULUS**.

Pembimbing I : Dr.Handri Santoso, M.Eng, (tanda tangan)

Depok, 16 Juli 2021

Disahkan oleh Ketua Jurusan Teknik elektro

Ir. Sri Danaryani, M.T

196305031991032001

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan Skripsi ini. Penulisan Skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Terapan Politeknik, Politeknik Negeri Jakarta, Jurusan Teknik Elektro, Program Studi Instrumentasi dan Kontrol Industri. Skripsi ini berjudul "Sistem Deteksi Kerusakan Gedung Menggunakan Algoritma *You Only Look Once* Dengan *Unmanned Aero Vehicle*". Skripsi ini membahas tentang metode deteksikondisi gedung yang digunakan untuk melihat kondisi kerusakan bagian gedung yang tidak terjangkau oleh mata manusia. Monitoring *Image Processing* yang diimplementasikan berupa metode YOLO (*You Only Look Once*). Penulis menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan Skripsi ini, sangatlah sulit bagi penulis untuk menyelesaikan Skripsi ini. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Ir. Sri Danaryani, M.T, selaku Ketua Jurusan Teknik Elektro.
2. Rika Novita, S.T, M.T, selaku Kepala Program Studi Instrumentasi dan Kontrol Industri.
3. Dr.Handri Santoso, M.Eng selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan penulis dalam penyusunan Skripsi ini.
4. Orang tua dan anggota keluarga penulis yang telah memberikan bantuan dukungan, material dan moral.
5. Semua sahabat yang telah banyak membantu penulis dalam menyelesaikan Skripsi ini.

Akhir kata, penulis berharap Tuhan Yang Maha Esa berkenan membalas kebaikan dari semua pihak yang telah membantu. Semoga Skripsi ini membawa manfaat bagi pengembangan IPTEK.

Depok, 02 Juli 2021

Penulis

ABSTRAK

Memantau dan mendeteksi kondisi bangunan gedung pabrik bertingkat diperlukan ketelitian dan keselamatan kerja agar berjalan lancar. Namun, menurut International Labour Organization bekerja di ketinggian memiliki resiko yang lebih tinggi yang dapat menyebabkan dampak fisik jangka panjang dan penelitian tentang kecelakaan kerja di ketinggian menunjukkan 74% pekerja pernah mengalami kecelakaan kerja di ketinggian. Oleh karena itu perlunya suatu sistem yang dapat mendeteksi kerusakan gedung bertingkat secara tidak langsung dengan unmanned aero vehicle sehingga resiko bekerja dapat dikurangi. Pada penelitain ini dipaparkan mengenai deteksi kerusakan-kerusakan gedung menggunakan beberapa algoritma YOLO yaitu: YOLOv3, YOLOv5s , YOLOv5m , YOLOv5x yang diproeses oleh Raspberry Pi dengan akuisisi citra oleh unmanned aero vehicle. Kemampuan mendeteksi kerusakan-kerusakan gedung pada beberapa versi algoritma YOLO menghasilkan beberapa nilai akurasi yang berbeda. Berdasarkan hasil pengujian yang dilakukan didapatkan nilai akurasi sebesar 55,7% untuk YOLOv3 ,YOLOv5s 64,3%, YOLOv5m 59%, YOLOv5x 58,89%. Sehingga YOLOv5s potensial sebagai sistem deteksi kerusakan bangunan gedung

Kata Kunci : Kerusakan gedung, Wall Cladding, Object Detection , You Only Look Once V3, You Only Look Once V5, Raspberry Pi, Unmanned Aero Vehicle

ABSTRACT

To function effectively, monitoring and detecting the state of high-rise manufacturing buildings need precision and work safety. Working at heights, on the other hand, has a higher risk of causing long-term physical effects, according to the International Labor Organization, and research on work accidents at heights reveals that 74 percent of employees have been involved in work accidents at heights. As an outcome, there is a need for a system that can identify high-rise structure damage indirectly using unmanned aerial aircraft, reducing the danger of working. The identification of building defects is described in this study utilizing multiple YOLO algorithms, including YOLOv3, YOLOv5s, YOLOv5m, and YOLOv5x, which are processed by the Raspberry Pi and picture capture by unmanned aerial vehicles. The ability to detect building defect in various versions of the YOLO algorithm generates a variety of accuracy ratings. The accuracy score for YOLOv3 was 55.7 percent, YOLOv5s was 64.3 percent, YOLOv5m was 59 percent, and YOLOv5x was 58.89 percent, according to the results of the testing. As a result, YOLOv5s has the potential to be used as a building defects detection system.

Keyword : *Building Defect, Object Detection , You Only Look Once V3, You Only Look Once V5, Raspberry Pi, Unmanned Aero Vehicle*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Bangunan gedung pabrik merupakan prasarana yang penting untuk menunjang kegiatan produksi. Namun, kondisi bangunan gedung pabrik yang bertingkat tidak dapat dijangkau oleh mata telanjang seluruhnya. Hal ini dikarenakan jangkauan mata yang terbatas yang tidak dapat melihat kerusakan gedung secara langsung. Kerusakan kerusakan gedung yang terjadi dikarenakan faktor cuaca, bencana alam, dan perawatan yang tidak dilakukan karena membutuhkan upaya sebelumnya yaitu pemantauan (*monitoring*).

Pemantauan konvensional umumnya dilakukan langsung oleh pekerja dengan memanjat atap gedung bertingkat sehingga memiliki resiko kecelakaan kerja.Berdasarkan riset yang dirilis pada pekerjaan di ketinggian, 74% pekerja pernah mengalami kecelakaan kerja di ketinggian [1]. Menurut International Labour Organization, jatuh dari ketinggian merupakan resiko kategori A yang memiliki dampak fisik jangka panjang [2].

Salah satu upaya yang dilakukan untuk mengurangi resiko kecelakaan kerja pada pekerjaan ini adalah dilakukannya pemantauan secara tidak langsung, yaitu menggunakan *unmanned aero vehicle*. *Unmanned aero vehicle* atau pesawat nirawak saat ini sudah dilengkapi dengan kamera sehingga menjangkau hingga ketinggian lebih dari 50 meter. Keunggulan pesawat nirawak antara dapat mengurangi biaya, mengurangi resiko, dan andal. Pesawat nirawak yang ditunjang dengan algoritma *You Only Look Once* dapat berfungsi sbg alat klasifikasi kerusakan gedung. Prinsip dari algoritma *YOLO* yang digunakan adalah mendeteksi benda atau target dengan menilai blok piksel berdasarkan warna dan bentuk yang telah di-*training* sehingga kerusakan-kerusakan pada gedung dapat terdeteksi dan diklasifikasi [3].

Berdasarkan permasalahan tersebut, pada penelitian ini dibuat model untuk mengklasifikasikan kerusakan bangunan gedung pabrik menggunakan metode You

Only Look Once (YOLO) dengan basis Convolutional Neural Network (CNN) [4]. Proses ini diharapkan dapat membantu dalam proses penilaian kerusakan bangunan untuk memulai rehabilitasi dan mempermudah dalam perawatan maupun perbaikan yang dapat dilakukan pada bangunan tersebut. Terdapat beberapa penelitian terdahulu mengenai deteksi kerusakan bangunan dan beberapa metode yang digunakan. Samsul (2010) mendeteksi kecacatan pada bangunan melalui pendekatan *Deep Learning* dengan metode CNN [5]. Kemudian, referensi lain yang dapat mengklasifikasikan kerusakan bangunan sekolah menggunakan metode CNN [6]. CNN memililki kekurangan yaitu proses yang lama dan membutuhkan ukuran *Graphic Processing* Unit yang besar [7]. Sedangkan algoritma YOLO-V5 dapat mendeteksi target namun dengan ukuran *Graphic Processing* Unit yang dapat disesuaikan oleh komputer yang digunakan. Algoritma YOLO V5 salah satunya telah diterapkan untuk mendeteksi social distancing di Turki YOLO terbukti mampu mengklasifikasikan objek sehingga memiliki potensi yang baik untuk pemantauan kerusakan gedung karena performa nya dalam mendeteksi dan mengklasifikasi objek dengan jarak yang cukup jauh [8].

Pada penelitian ini akan dilakukan pengklasifikasian kondisi kerusakan eksterior bangunan gedung pabrik berbahan *wall cladding* menjadi 3 kelas, yaitu noda, karat, dan lubang menggunakan komputer Raspberry Pi yang memiliki spesifikasi kecil namun dapat melakukan pengklasifikasian kerusakan gedung ini. Noda-noda yang bersumber dari cerobong seperti jelaga buang membekas dan menempel pada bagian atap maupun dinding *cladding*. Noda dapat berbentuk cairan yang berpotensi menyebabkan karat jika tidak segera dibersihkan. Adapun karat yang terbentuk oleh reaksi oksidasi besi oleh air dapat merusak bagian dinding *cladding* yang biasa digunakan oleh gedung pabrik. Karat merupakan awal mula pembentukan lubang dan dapat memberikan celah air ataupun benda lain yang dapat memasuki bagian interior gedung sehingga dapat menghambat proses kerja hingga menghentikan proses kerja.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diutarakan di atas, maka rumusan masalah dari penelitian ini adalah:

- Bagaimana meningkatkan efisiensi proses *monitoring* gedung pabrik bertingkat?
- Bagaimana mengurangi kecelakaan kerja pada proses *monitoring* gedung pabrik bertingkat?
- Bagaimana dapat mendeteksi dan mengklasifikasi kerusakan pada atap maupun dinding bermaterial *cladding*?
- Bagaimana mengimplementasikan algortima *YOLO* pada proses *monitoring* gedung bertingkat?

1.3 Tujuan

Pesawat nirawak sebagai alat *monitoring* gedung pabrik bertingkat dapat digunakan untuk menganalisis kondisi fisik gedung tanpa mempertaruhkan keselamatan pekerja. Selain itu, pesawat nirawak dapat mengurangi beban pekerjaan manusia dalam menginspeksi bagian eksterior gedung sehingga pengecekan dapat dilakukan lebih rutin, cepat, dan aman. Prinsip *computer vision* diimplementasikan dengan menyematkan kamera pada pesawat nirawak yang ditunjang dengan pemrograman berbasis YOLO untuk menunjang sistem pemantauan pada bagian eksterior gedung.

1.4 Batasan Masalah

Penulisan ini hanya membahas bagaimana algoritma *YOLO* diterapkan pada sistem inspeksi gedung pabrik mennggunakan Raspberry Pi.

1.5 Luaran

Luaran penelitian ini adalah sistem *monitoring* gedung pabrik bertingkat menggunakan pesawat nirawak berbasis algoritma YOLO yang kompatibel dijalankan pada komputer Raspberry Pi.

BAB V
KESIMPULAN

5.1. Kesimpulan

Adapun kesimpulan yang dapat penulis ambil dari hasil pengujian dan analisa yang telah dilakukan yaitu:

- Berdasarkan dari pengujian yang telah dilakukan, arsitektur mampu mendeteksi kerusakan-kerusakan gedung yang telah diklasifikasikan. Namun, terdapat beberapa nilai akurasi, waktu yang digunakan untuk memproses data, dan ukuran model yang dimodifikasi agar cocok untuk diterapkan dengan spesifikasi Raspberry Pi 4 yang ada.
- Berdasarkan dari data yang telah diuji, arsitektur YOLOv5s memiliki nilai akurasi terbaik dalam mendeteksi kerusakan-kerusakan gedung. Akurasi yang diperoleh bernilai 64,3%. Selain itu YOLOv5s memiliki ukuran model yang kecil dan memakan waktu yang lebih pendek untuk mendeteksi kerusakan-kerusakan yang ada pada bangunan gedung.
- Kemampuan Raspberry Pi yang terbatas membutuhkan model yang ringan untuk memproses data, sehingga YOLOv5s cocok digunakan untuk mendeteksi kerusakan-kerusakan gedung menggunakan Raspberry Pi 4.
- Perbedaan hasil yang didapat dipengaruhi dari nilai *Floating Point Precision* (FLOPs) yang dirubah dari 32bit menjadi 16bit, kemudian modifikasi CSP Network yang berbasis DenseNet yang membuat nilai parameter beragam sehingga dapat dijalankan sesuai keinginan (YOLOv5s digunakan untuk mendeteksi lebih cepat dengan akurasi yang lebih kecil, YOLOv5x digunakan untuk mendeteksi dengan akurasi lebih baik namun lebih lambat).

5.2. Saran

Untuk meningkatkan akurasi menjadi lebih baik, dapat menambahkan *dataset* kerusakan-kerusakan gedung. Selain itu dapat menggunakan komputer yang memiliki spesifikasi yang lebih tinggi dan arsitektur dengan parameter yang lebih tinggi.

DAFTAR PUSTAKA

[1] M. S. Q. Siti Riptifah Tri Handari, "Faktor-Faktor Kejadian Kecelakaan Kerja pada Pekerja Ketinggian di PT. X Tahun 2019," vol. 556, pp. 90–98, 2019, [Online]. Available: https://jurnal.umj.ac.id/index.php/JKK.

[2] J., S. H., and E. W.I., "Analisis Risiko Kecelakaan Kerja Pada Proyek Bangunan Gedung Dengan Metode Fmea," *J. Muara Sains, Teknol. Kedokt. dan Ilmu Kesehat.*, vol. 1, no. 1, pp. 115–123, 2017, doi: 10.24912/jmstkik.v1i1.419.

[3] J. Redmon and A. Farhadi, "YOLO v.3," *Tech Rep.*, pp. 1–6, 2018, [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[4] A. Adarsh, P. Rathi, and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," *2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020*, pp. 687–694, 2020, doi: 10.1109/ICACCS48705.2020.9074315.

[5] A. Georgiou, "Verification of a building defect classification system for housing," *Struct. Surv.*, vol. 28, no. 5, pp. 370–383, 2010, doi: 10.1108/02630801011089164.

[6] A. M. Rizki and N. Marina, "Klasifikasi Kerusakan Bangunan Sekolah Menggunakan Metode Convolutional Neural Network Dengan Pre-Trained Model Vgg-16," *J. Ilm. Teknol. dan Rekayasa*, vol. 24, no. 3, pp. 197–206, 2019, doi: 10.35760/tr.2019.v24i3.2396.

[7] A. Patil and M. Rane, "Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition," *Smart Innov. Syst. Technol.*, vol. 195, pp. 21–30, 2021, doi: 10.1007/978-981-15-7078-0_3.

[8] R. SHUKLA, A. K. MAHAPATRA, and J. SELVIN PAUL PETER, "Social distancing tracker using yolo v5," *Turkish J. Physiother. Rehabil.*, vol. 32, no. 2, pp. 1785–1793, 2021.

[9] G. Ostojić, S. Stankovski, B. Tejić, N. Dukić, and S. Tegeltija, "Design, control and application of quadcopter," *Int. J. Ind. Eng. Manag.*, vol. 6, no. 1, pp. 43–48, 2015.

[10] R. Pi, "Raspberry Pi 4 Model B," no. June, 2019.

[11] S. Edition, *Computer Vision Programming*. 2020.

[12] A. Zein, "Pendeteksian Kantuk Secara Real Time Menggunakan Pustaka OPENCV dan DLIB PYTHON," *Sainstech J. Penelit. dan Pengkaj. Sains dan Teknol.*, vol. 28, no. 2, pp. 22–26, 2018, doi: 10.37277/stch.v28i2.238.

[13] I. B. Mustaffa and S. F. B. M. Khairul, "Identification of fruit size and maturity through fruit images using OpenCV-Python and Rasberry Pi," *Proceeding 2017 Int. Conf. Robot. Autom. Sci. ICORAS 2017*, vol. 2018-March, pp. 1–3, 2018, doi: 10.1109/ICORAS.2017.8308068.

[14] A. Rosebrock, "Convolutional Neural Networks CNNs and Layer Types," 2021. https://www.pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ (accessed Jun. 28, 2021).

[15] A. Yanuar, "Fully-Connected Layer CNN dan Implementasinya," 2018. https://machinelearning.mipa.ugm.ac.id/2018/06/25/fully-connected-layer-cnn-dan-implementasinya/ (accessed Jun. 28, 2021).

[16] A. Rosebrock, “Intersection over Union (IoU) for object detection,” 2016. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (accessed Jun. 28, 2021).

[17] S. Narkhede, “Understanding Confusion Matrix,” 2018, [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[18] D. Yang, Y. Cui, Z. Yu, and H. Yuan, “Deep Learning Based Steel Pipe Weld Defect Detection Deep Learning Based Steel Pipe Weld Defect Detection.”

[19] E. Bisong, *Training a Neural Network*. 2019.

[20] J. Brownlee, “A Gentle Introduction to Transfer Learning for Deep Learning,” 2017. https://machinelearningmastery.com/transfer-learning-for-deep-learning/#:~:text=Transfer learning is a machine,model on a second task.&text=Common examples of transfer learning,your own predictive modeling problems.

[21] K. Huang and J. Hinze, “Analysis of Construction Worker Fall Accidents,” *J. Constr. Eng. Manag.*, vol. 129, no. 3, pp. 262–271, 2003, doi: 10.1061/(asce)0733-9364(2003)129:3(262).

[22] B. McNoe, J. Langley, T. Driscoll, and A. Feyer, *Work-related slip , trip and fall injuries in New Zealand*. 2005.

[23] P. Menteri and P. Umum, “Pedoman pemeliharaan dan perawatan bangunan gedung,” 2008.

[24] A. Tzempelikos, M. Bessoudo, A. K. Athienitis, and R. Zmeureanu, “Indoor thermal environmental conditions near glazed facades with shading devices – Part II : Thermal comfort simulation and impact of glazing and shading properties Indoor thermal environmental conditions near glazed facades with shading devices e Part II,” *Build. Environ.*, vol. 45, no. 11, pp. 2517–2525, 2010, doi: 10.1016/j.buildenv.2010.05.014.

[25] E. Technology, “The Effectiveness of Envelope Design in High Rise Office Building using Exterior Wall Cladding as Green Technology Solutions in Malaysia ’ s Urban Context,” vol. 1, no. 1, pp. 1–9, 2019.

[26] M. Ozel, “Thermal performance and optimum insulation thickness of building walls with different structure materials,” *Appl. Therm. Eng.*, vol. 31, no. 17–18, pp. 3854–3863, 2011, doi: 10.1016/j.applthermaleng.2011.07.033.

LAMPIRAN

Lampiran 1 Daftar Riwayat Hidup Penulis

Farros Hilmi Zain anak ketiga dari tiga bersaudara. Lahir di Tangerang, 31 Desember 1999. Lulus dari SD Pelita Atsiri Permai tahun 2011, SMPN 1 Kota Depok tahun 2014, dan SMA Batik 1 Surakarta pada tahun 2017, kemudian melanjutkan kuliah Sarjana Terapan (S.Tr.) di Politeknik Negeri Jakarta, jurusan Teknik Elektro, program studi Instrumentasi dan Kontrol Industri (IKI) (2017-2021).

```
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.safe_dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.safe_dump(vars(opt), f, sort_keys=False)

    # Configure
    plots = not opt.evolve  # create plots
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.safe_load(f)  # 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, 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_graph(torch.jit.trace(model, imgs, strict=False), [])  #
add model graph
                    #         tb_writer.add_image(f, result, dataformats='HWC',
global_step=epoch)
            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

        # 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='yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
```

```
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear LR')
    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(exclude=('pycocotools', 'thop'))

    # 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.safe_load(f))  # 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 = str(increment_path(Path(opt.project) / opt.name,
exist_ok=opt.exist_ok | opt.evolve))

# 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.safe_load(f)  # 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}')
```

Lampiran 3 *Source Code Detection*

```
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn

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,
save_one_box
from utils.plots import colors, plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized


def detect(opt):
    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 = 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
    names = model.module.names if hasattr(model, 'module') else model.names  # get
class names
    if half:
        model.half()  # to FP16
```

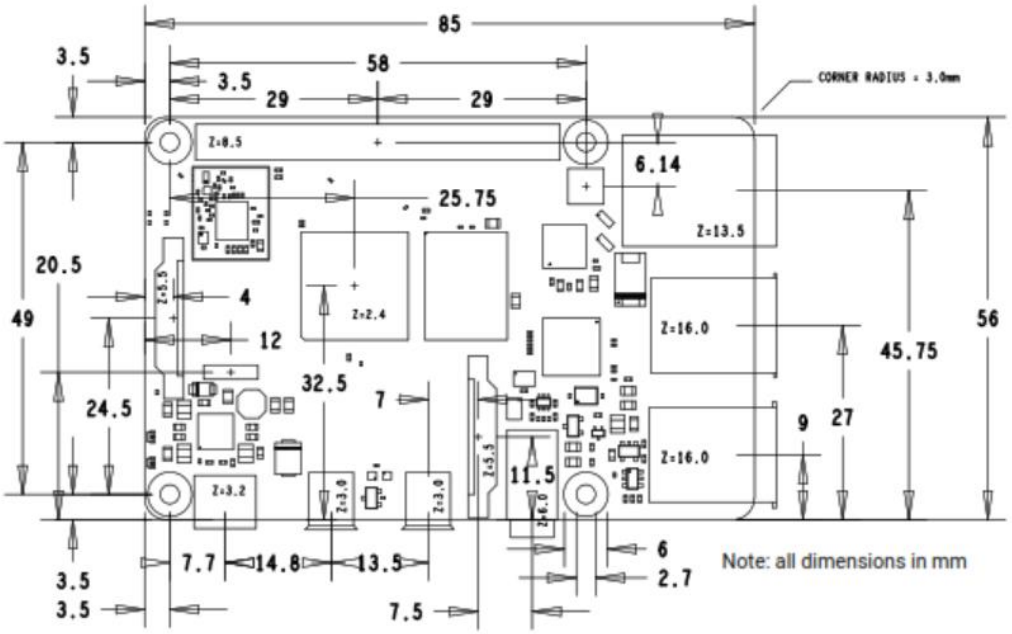
```
    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt',
map_location=device)['model']).to(device).eval()

    # 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)

    # 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, opt.classes,
opt.agnostic_nms,
                                   max_det=opt.max_det)
        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], f'{i}: ', im0s[i].copy(), dataset.count
```

```
        else:
            p, s, im0, frame = path, '', im0s.copy(), 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
        imc = im0.copy() if opt.save_crop else im0  # for opt.save_crop
        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 opt.save_crop or view_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    label = None if opt.hide_labels else (names[c] if opt.hide_conf else
f'{names[c]} {conf:.2f}')
                    plot_one_box(xyxy, im0, label=label, color=colors(c, True),
line_thickness=opt.line_thickness)
                    if opt.save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] /
f'{p.stem}.jpg', BGR=True)

            # 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)  # 1 millisecond
```

```
    # Save results (image with detections)
    if save_img:
        if dataset.mode == 'image':
            cv2.imwrite(save_path, im0)
        else:  # 'video' or 'stream'
            if vid_path != 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='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/images', 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('--max-det', type=int, default=1000, help='maximum number of detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
```

```
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --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')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    opt = parser.parse_args()
    print(opt)
    check_requirements(exclude=('tensorboard', '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(opt=opt)
                strip_optimizer(opt.weights)
        else:
            detect(opt=opt)
```

Lampiran 4 Spesifikasi Raspberrry Pi 4



- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient